Team Reference Document

TI1050: 薛鸿涛、潘律旨、罗天松

2020 年 10 月 16 日

目录

1 数学

1.1 整数划分

1.1.1 小常数版本

牛客 7803I

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
constexpr int N = 5e5 + 10;
constexpr int M = 1300;
constexpr int mod = 998244353;
int n, f[M], g[N];

// g[i] 表示 i 的划分数
void gao() {
    f[1] = 1, f[2] = 2, f[3] = 5, f[4] = 7;
    for (int i = 5; i < M; ++i) f[i] = 3 + 2 * f[i - 2] - f[i - 4];
    g[0] = 1;
    for (int i = 1; i <= n; ++i) {
        ll sum[4] = {0}, now = 0;
        int j = 1;
        for (; f[j + 3] <= i; j += 4) {
            sum[0] += g[i - f[j]];
            sum[1] += g[i - f[j + 1]];
            sum[2] -= g[i - f[j + 2]];
            sum[3] -= g[i - f[j + 3]];
        }
        now = (sum[0] + sum[1] + sum[2] + sum[3]) % mod;
        for (; f[j] <= i; ++j) {
            if ((j + 1) >> 1 & 1) {
                now += g[i - f[j]];
            } else {
                now -= g[i - f[j]];
            }
        }
        now = (now + mod) % mod;
        g[i] = now;
    }
}

int main() {
    cin >> n;
    gao();
    cout << g[n - 1] << endl;
    return 0;
}
```

1.1.2 牛逼版本

牛客 7803I
好像只能求模数为 998244353 的情况。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;

```

```
const int N=5e5+50, G=3, mod=998244353;
inline int add(int x,int y) {return (x+y>=mod) ? (x+y-mod) : (x+y);}
inline int dec(int x,int y) {return (x-y<0) ? (x-y+mod) : (x-y);}
inline int mul(int x,int y) {return (LL)x*y%mod;}
inline int power(int a,int b,int rs=1) {for(;b;b>>=1,a=mul(a,a)) if(b&1) rs=
    mul(rs,a); return rs;}
int n,k,g[N*8],f[N*8],tp[N*8],w[N*8],pos[N*8];
inline void init(int len) { k=len<<1; for(int i=1;i<k;i++) pos[i]=(i&1) ? ((
    pos[i>>1]>>1)^(k>>1)) : (pos[i>>1]>>1);}
inline void dft(int *a) {
    for(int i=1;i<k;i++) if(pos[i]>i) swap(a[pos[i]],a[i]);
    for(int bl=1;bl<k;bl<<=1) {
        int tl=bl<<1, wn=power(G,(mod-1)/tl);
        w[0]=1; for(int i=1;i<bl;i++) w[i]=mul(w[i-1],wn);
        for(int bg=0;bg<k;bg+=tl)
            for(int j=0;j<bl;j++) {
                int &t1=a[bg+j], &t2=a[bg+j+bl], t=mul(t2,w[j]);
                t2=dec(t1,t); t1=add(t1,t);
            }
    }
}

inline void fuck(int *a,int *b,int len) {
    if(len==1) {b[0]=power(a[0],mod-2); return;}
    if(len!=1) fuck(a,b,len>>1);
    init(len);
    for(int i=0;i<len;i++) tp[i]=a[i];
    for(int i=len;i<k;i++) tp[i]=0;
    dft(b); dft(tp);
    for(int i=0;i<k;i++) b[i]=dec(mul(2,b[i]),mul(tp[i],mul(b[i],b[i])));
    reverse(b+1,b+k); dft(b);
    const int inv=power(k,mod-2);
    for(int i=0;i<len;i++) b[i]=mul(b[i],inv);
    for(int i=len;i<k;i++) b[i]=0;
}

//题目要求 n - 1 的划分数
int main() {
    cin>>n;n--;
    for(int k=1;(3*k*k-k)/2<=n;++k) {
        int v=(k&1) ? mod-1 : 1;
        g[(3*k*k-k)/2]+=v;
        g[(3*k*k+k)/2]+=v;
    } g[0]=1;
    for(k=1;k<=n;k<<=1);
    fuck(g,f,k);
    cout<<f[n]<<endl;
}
```

1.1.3 例题

HDU 4651, 4658
题意：求 n 的划分数。4659 要求元素重复次数小于 k。
代码中 $ans[i]$ 表示的就是 i 的划分数，$solve(n,k)$ 表示重复次数小于 k 的划分数

```
#include <bits/stdc++.h>
using namespace std;
const int mod = 1e9 + 7;
```

```
4  const int N = 1e5 + 10;
5  int ans[N];
6  void init() {
7      memset(ans, 0, sizeof(ans));
8      ans[0] = 1;
9      for (int i = 1; i < N; ++i) {
10         ans[i] = 0;
11         for (int j = 1; ; j++) {
12             int tmp = (3 * j - 1) * j / 2;
13             if (tmp > i) break;
14             int _tmp = ans[i - tmp];
15             if (tmp + j <= i) {
16                 _tmp = (_tmp + ans[i - tmp - j]) % mod;
17             }
18             if (j & 1) {
19                 ans[i] = (ans[i] + _tmp) % mod;
20             } else {
21                 ans[i] = (ans[i] - _tmp + mod) % mod;
22             }
23         }
24     }
25 }
26
27 int solve(int n, int k) {
28     int res = ans[n];
29     for (int i = 1; ; i++) {
30         int tmp = k * i * (3 * i - 1) / 2;
31         if (tmp > n) {
32             break;
33         }
34         int _tmp = ans[n - tmp];
35         if (tmp + i * k <= n) {
36             _tmp = (_tmp + ans[n - tmp - i * k]) % mod;
37         }
38         if (i & 1) {
39             res = (res - _tmp + mod) % mod;
40         } else {
41             res = (res + _tmp) % mod;
42         }
43     }
44     return res;
45 }
46
47 int main() {
48     init();
49     int _T, n, k;
50     cin >> _T;
51     while (_T--) {
52         cin >> n >> k;
53         cout << solve(n, k) << '\n';
54     }
55     return 0;
56 }
```

1.2 线性基

1.2.1 应用

1. UVALive 8512

   题意：

   给出 n 个数 a_i，每次询问一个区间 $[l,r]$，询问从中挑选任意个数的异或和再和 k 按位或的最大值，即假设挑选的数为 $a_{p_1},a_{p_2},\cdots,a_{p_m}$，那么要使得下式最大：

   $$k\ or\ (a_{p_1} \oplus a_{p_2} \cdots a_{p_m})$$

   思路：先对所有 a_i 进行处理，将二进制位上 k 为 1 的位，在 a_i 中都置为 0。

   然后就是线段树维护线性基，求区间最大值。

   因为这样一来，相当于把 k 为 1 的那些位和 a_i 对应的那些位剥离开来。

1.3 对偶问题

$$max\{c^T x | Ax \le b\} = min\{b^T y | A^T y \ge c\}$$

1.4 单纯形法

线性规划的标准形式：

求 $Max\ \ z = \sum_{j=1}^{n} C_j X_j$

$$\begin{cases} \sum\limits_{j=1}^{n} A_{i,j} X_j \le B_j, i = 1,2,\cdots,m \\ X_j \quad \ge 0, j = 1,2,\cdots,n \end{cases}$$

通俗来说，即：

- 目标函数要求最大化
- 约束条件均为不等式
- 决策变量非负约束

普通线性规划化为标准形：

- 若目标函数为最小化, 可以通过取负或者对偶, 求最大化
- 约束不等式为等式，可以拆成两个不等式约束，即 $X = b$ 可以拆成

$$\begin{cases} X \le b \\ X \ge b \end{cases} \rightarrow \begin{cases} X \le b \\ -X \le -b \end{cases}$$

- 若存在取值无约束的变量，可转变为两个非负变量的差

$$-\infty \le_k \le \infty \rightarrow \begin{cases} x_k = x_m - x_n \\ x_m, x_n \ge 0 \end{cases} \rightarrow \begin{cases} x_k - x_m + x_n = 0 \\ x_m, x_n \ge 0 \end{cases}$$

再转化为两个不等式

1.4.1 UOJ 179

题意:

有 n 个实数变量 $x_1, x_2, \cdots, x_n$, 和 m 条约束, 其中第 i 条约束形如 $\sum_{j=1}^{n} a_{ij}x_j \le b_j$ 此外这 n 个变量需要满足非负性限制，即 $x_j \ge 0$

指定 x_j 每个变量的取值, 使得目标函数 $F = \sum_{j=1}^{n} c_j x_j$ 最大

无解输出"Infeasible"。

没有最大值输出"Unbounded"。

```
#include <bits/stdc++.h>
using namespace std;
typedef double db;
const db eps = 1e-5;

struct LP {
    int m, n;
    vector<int> B, N;
    vector<vector<db> > D;
    LP() {}
    LP(const vector<vector<db> > &A, const vector<db> &b, const vector<db> &c) {
        m = (int)b.size(); n = (int)c.size(); N = vector<int>(n + 1); B = vector<int>(m + 1);
        D = vector <vector<db> >(m + 2, vector<db>(n + 2));
        for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j];
        for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n + 1] = b[i]; }
        for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
        N[n] = -1; D[m + 1][n] = 1;
    }
    void Pivot(int r, int s) {
        for (int i = 0; i < m + 2; i++) if (i != r)
            for (int j = 0; j < n + 2; j++) if (j != s)
                D[i][j] -= D[r][j] * D[i][s] / D[r][s];
        for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] /= D[r][s];
        for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] /= -D[r][s];
        D[r][s] = 1.0 / D[r][s];
        swap(B[r], N[s]);
    }
    bool Simplex(int phase) {
        int x = phase == 1 ? m + 1 : m;
        while (true) {
            int s = -1;
            for (int j = 0; j <= n; j++) {
                if (phase == 2 && N[j] == -1) continue;
                if (s == -1 || D[x][j] < D[x][s] || (D[x][j] == D[x][s] && N[j] < N[s])) s = j;
            }
            if (D[x][s] > -eps) return true;
            int r = -1;
            for (int i = 0; i < m; i++) {
                if (D[i][s] < eps) continue;
                if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s]
                    ||
                    ((D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i] < B[r])) r = i;
```

```
            }
            if (r == -1) return false;
            Pivot(r, s);
        }
    }
    db Solve(vector<db> &x) {
        int r = 0;
        for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
        if (D[r][n + 1] < -eps) {
            Pivot(r, n);
            //无解
            if (!Simplex(1) || D[m + 1][n + 1] < -eps) return -
               numeric_limits<db>::infinity();
            for (int i = 0; i < m; i++) if (B[i] == -1) {
                int s = -1;
                for (int j = 0; j <= n; j++)
                    if (s == -1 || D[i][j] < D[i][s] || (D[i][j] == D[i][s]
                       && N[j] < N[s])) s = j;
                Pivot(i, s);
            }
        }
        //无穷解
        if (!Simplex(2)) return numeric_limits<db>::infinity();
        x = vector<db>(n);
        for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
        return D[m][n + 1];
    }
};

const int N = 1e4 + 10;
int n, m, t, c[N];

int main() {
    while (scanf("%d%d%d", &n, &m, &t) != EOF) {
        vector <vector<db> > A(m + 2, vector <db>(n + 2, 0));
        vector <db> B(m);
        vector <db> C(n);
        vector <db> X;
        for (int i = 0; i < n; ++i) scanf("%d", c + i), C[i] = c[i];
        for (int i = 0, b; i < m; ++i) {
            for (int j = 0, a; j < n; ++j) {
                scanf("%d", &a);
                A[i][j] = a;
            }
            scanf("%d", &b);
            B[i] = b;
        }
        LP lp(A, B, C);
        db res = lp.Solve(X);
        if (res == -numeric_limits<db>::infinity()) {
            puts("Infeasible");
        } else if (res == numeric_limits<db>::infinity()) {
            puts("Unbounded");
        } else {
            printf("%.10f\n", res);
            if (t) for (int i = 0; i < n; ++i)
                printf("%.10f%c", X[i], " \n"[i == n - 1]);
        }
```

```
    }
    return 0;
}
```

1.4.2 Luogu P3337

$A[i][0]i \in [1,n]$ 等价于 B 矩阵。
$A[0][i]i \in [1,m]$ 等价于 C 矩阵。

```
#include <bits/stdc++.h>
using namespace std;

template <class T>
struct LP {
    static const T INF = 0x3f3f3f3f;
    int n, m;
    vector <T> v;
    vector <vector<T> > a;
    LP(const vector<vector<T> > &_a, int _n, int _m) {
        a = _a; n = _n; m = _m;
        v.resize(m + 1);
    }
    void pivot(int x, int y) {
        int tp = 0;
        for (int i = 0; i <= m; i++) if (a[x][i]) v[++tp] = i;
        for (int i = 0;i <= n; i++) {
            if (i != x && a[i][y]) {
                int k = a[i][y]; a[i][y] = 0;
                for (int j = 1; j <= tp; j++)
                    a[i][v[j]] -= k * a[x][v[j]];
            }
        }
    }
    T solve() {
        a[n + 1][0] = INF;
        while(1) {
            int x = 0, y = n + 1;
            for (int i = 1; i <= m; i++)
                if (a[0][i] > 0) { x = i; break; }
            if (!x) return -a[0][0];
            for (int i = 1; i <= n; i++)
                if(a[i][x] > 0 && a[i][0] < a[y][0]) y = i;
            pivot(y, x);
        }
    }
};

const int N = 1e4 + 10;
int n, m, a[N];

int main() {
    while (scanf("%d%d", &n, &m) != EOF) {
        vector <vector<int> > A(n + 2, vector <int>(m + 2, 0));
        for (int i = 1; i <= n; ++i) scanf("%d", a + i), A[i][0] = a[i];
        for (int i = 1, l, r, d; i <= m; ++i) {
            scanf("%d%d%d", &l, &r, &d);
            A[0][i] = d;
            for (int j = l; j <= r; ++j)
                A[j][i] = 1;
```

```
        }
        LP <int> lp(A, n, m);
        printf("%d\n", lp.solve());
    }
    return 0;
}
```

1.5 梯度下降法

每次朝着函数的梯度的反方向前进 α 步, α 为学习率

$$\Theta^n = \Theta^{n+1} - \alpha \nabla J(\Theta)$$

选定合适的 α 和 n 迭代下去即可。

CometOJ 16J

题意:

有 n 个玩家玩一个游戏，游戏分两轮，对于每一轮游戏，标签相同的玩家需要跑到同一个点集合

假设一个玩家的初始位置为 S_0, 两轮的集结点为 S_1, S_2, 它需要付出的体力为 $dis(S_1 - S_0) + dis(S_2 - S_1)$, 其中 dis 为两点之间的欧几里得距离。

对于每一轮每一种标签定一个集结点，使得所有玩家的消耗的体力和最少。

思路:

易知，n 的大小不会影响体力消耗函数的梯度，不妨令它为 1

我们令 x_i 表示初始位置向量的第 i 维度值，y_i 表示第一轮集结点位置的第 i 维度的值, z_i 表示第二轮集结点位置的第 i 维度的值，那么有:

$$f(y_0, y_1, z_0, z_1) = (y_0 - x_0)^2 + (y_1 - x_1)^2 + (z_0 - y_0)^2 + (z_1 - y_1)^2$$

求其梯度分量有:

$$\begin{aligned}
\frac{\partial f}{\partial y_0} &= 2(2y_0 - x_0 - z_0) \\
\frac{\partial f}{\partial y_1} &= 2(2y_1 - x_1 - z_1) \\
\frac{\partial f}{\partial z_0} &= -y_0 + z_0 \\
\frac{\partial f}{\partial z_1} &= -y_1 + z_1
\end{aligned}$$

按公式迭代即可。

```
#include <bits/stdc++.h>
using namespace std;
typedef double db;
const int N = 5e2 + 10;
int n, f[N], g[N];
db x[N][2], y[N][2], z[N][2], nxY[N][2], nxZ[N][2];

db sqr(db x) { return x * x; }
db gao() {
    int loop = 50000;
    db alpha = 1e-3;
    while (loop--) {
        for (int i = 1; i <= n; ++i) {
            nxY[f[i]][0] -= alpha * (2.0 * y[f[i]][0] - x[i][0] - z[g[i
                ]][0]);
            nxY[f[i]][1] -= alpha * (2.0 * y[f[i]][1] - x[i][1] - z[g[i
                ]][1]);
```

```
            nxZ[g[i]][0] -= alpha * (-y[f[i]][0] + z[g[i]][0]);
            nxZ[g[i]][1] -= alpha * (-y[f[i]][1] + z[g[i]][1]);
        }
        for (int i = 1; i <= n; ++i) {
            for (int j = 0; j < 2; ++j) {
                y[i][j] = nxY[i][j];
                z[i][j] = nxZ[i][j];
            }
        }
    }
    db res = 0;
    for (int i = 1; i <= n; ++i) {
        for (int j = 0; j < 2; ++j) {
            res += sqr(x[i][j] - y[f[i]][j]);
            res += sqr(y[f[i]][j] - z[g[i]][j]);
        }
    }
    return res;
}

int main() {
    while (scanf("%d", &n) != EOF) {
        for (int i = 1; i <= n; ++i) scanf("%lf%lf", x[i], x[i] + 1);
        for (int i = 1; i <= n; ++i) scanf("%d", f + i);
        for (int i = 1; i <= n; ++i) scanf("%d", g + i);
        printf("%.6f\n", gao());
    }
    return 0;
}
```

1.6 斯特灵数

1.6.1 第一类

$\begin{bmatrix} n \\ m \end{bmatrix}$ 表示 n 个元素分成 m 个环的方案数。

递推式：

$$\begin{bmatrix} n \\ m \end{bmatrix} = \begin{bmatrix} n-1 \\ m-1 \end{bmatrix} + (n-1) \cdot \begin{bmatrix} n-1 \\ m \end{bmatrix}$$

考虑之前已经放好的 $n-1$ 个数，那么有一个新加入的数时有两种选择，第一种是自己成一个环，贡献是 $\begin{bmatrix} n-1 \\ m-1 \end{bmatrix}$，第二种是放到一个已经存在的环内，那么它可以放在任意一个数的前面，所以贡献是 $n-1 \cdot \begin{bmatrix} n-1 \\ m \end{bmatrix}$。

下面用 $S_1(n,m)$ 来表示 $\begin{bmatrix} n \\ m \end{bmatrix}$

性质：

- $$n! = \sum_{i=0}^{n} S_1(n,i)$$

  证明：

  第一类斯特林数的本质是环排列的个数，环的本质可以理解为置换，那么所有第一类斯特林数对应的环排列可以和置换一一对应，所以总数是 $n!$。

  也可以考虑有 n 个点，每个点的入度和出度都为 1，每次选择一个入度为 0 和出度为 0 的点，将它们连上一条边，那么显然第一次有 n 种选择，第二次有 $n-1$ 种选择 $\cdots$，所以是 $n!$。

1.6.2 第二类

$\left\{ {n \atop m} \right\}$ 表示将 n 个元素放入 m 个相同盒子里，每个盒子非空的方案数。

递推式：

$$\left\{ {n \atop m} \right\} = \left\{ {n-1 \atop m-1} \right\} + m \cdot \left\{ {n-1 \atop m} \right\}$$

考虑新加入一个元素是新建一个盒子还是放入原有的盒子即可。

下面用 $S_2(n,m)$ 来表示 $\left\{ {n \atop m} \right\}$

容斥的计算方法：

$$S_2(n,m) = \frac{1}{m!}\sum_{i=0}^{m}(-1)^i\binom{m}{i}(m-i)^n$$

考虑先将盒子进行编号，最后结果除去顺序。考虑有几个盒子为空，枚举出来，然后剩下的元素随便放在非空的盒子里，因为这里算完之后是至少有 i 个盒子为空，所以需要容斥。

和自然数幂的关系：

$$m^n = \sum_{i=0}^{m} S_2(n,i)\binom{m}{i} i!$$

m^n 可以理解为把 n 个球放到 m 有标号的盒子里，盒子可以为空，现在枚举哪些盒子非空，放进去的方案数就是第二类斯特林数乘阶乘。

1.7 威尔逊定理

p 是质数，有：

$$(p-1)! \equiv -1 \bmod p$$

1.8 贝尔数

贝尔数给出了集合划分的数目，B_n 表示将 n 个元素进行划分成 $1, 2, \cdots n$ 个集合的方案数。

$B_0 = B_1 = 1$，前几项的贝尔数为 $1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, \cdots$

递推公式：

- $$B_{n+1} = \sum_{k=0}^{n}\binom{n}{k}B_k$$
- Dobinski 公式：

$$B_n = \frac{1}{e}\sum_{k=0}^{\infty}\frac{k^n}{k!}$$

- Touchard 同余，若 p 是任意质数，有

$$B_{p+n} = B_n + B_{n+1} \bmod p$$

- 每个贝尔数都是第二类 Stirling 数的和:

$$B_n = \sum_{k=1}^{n} S(n,k)$$

其中第二类 Stirling 数 $S(n,k)$ 是把基数为 n 的集划分为正好 k 个非空集的方法的数目

- 贝尔函数的质数母函数是:

$$B(x) = \sum_{n=0}^{\infty} \frac{B_n}{n!} x^n = e^{e^x} - 1$$

1.8.1 应用

1. 2019ICPC 上海网络赛 F

   题意:

   定义一个长度为 n 的字符串，每个位置需要满足当前位置的大小小于等于它前面那个字母的最大值 +1，问字典序第 k 大的字符串是多少。

   思路:

   其实是个集合划分的字符串表示，方案总是数贝尔数。

   那么考虑预处理一个 dp，$f[n][i][j]$ 表示长度为 n，当前在第 i 位，最大值为 j 的所有后缀的方案数是多少。

   那么显然有 $f[n][n][j] = 1$，那么考虑倒着推上去。

   那么有:

$$f[n][i][j] = f[n][i+1][j] * j + f[n][i+1][j+1]$$

   表示下一个字符放 $1-j$ 的方案数为 $f[n][i+1][j]$，如果放 $j+1$ 那么其方案数为 $f[n][i+1][j+1]$

   然后从左往右逐位确定即可。

1.9 n 个球放入 m 个盒子问题

n 个球, m 个箱子。

1. 球相同、盒子不同、无空箱

$$f(n,m) = \begin{cases} C(n-1,m-1) & n \geq m \\ 0 & n < m \end{cases}$$

   使用插板法：n 个球中间有 $n-1$ 个间隙，现在要分成 m 个盒子，不能有空箱子，所以需要在 $n-1$ 个间隙中选出 $m-1$ 个间隙。

2. 球相同、盒子不同、允许空箱

$$f(n,m) = C(n+m-1,m-1)$$

   在第一种情况下继续讨论，先假设 m 个盒子里都放好了 1 个球，其实就是现在有 $m+n$ 个相同的球，要放入 m 个不同的箱子，没有空箱的问题。

3. 球不同，盒子相同，无空箱第二类斯特林数 $dp[n][m]$

4. 球相同，盒子相同，允许空箱

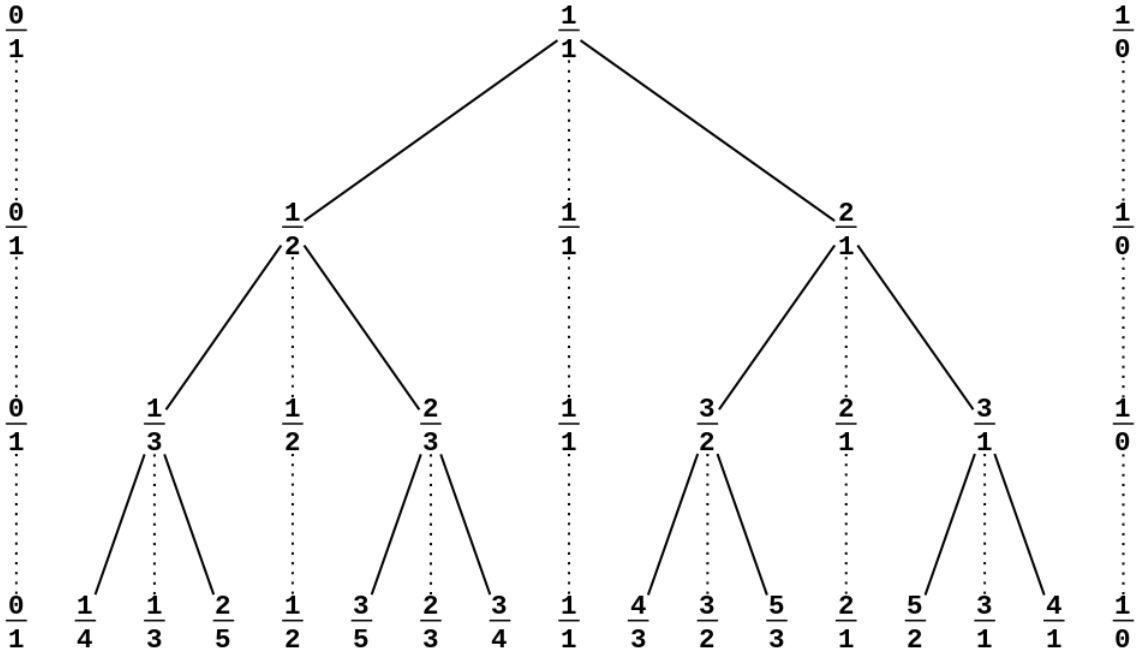
   考虑 $f[i][j]$ 表示有 i 个箱子，j 个小球的方案数。

- 当 $i > j$ 时，总会有 $i - j$ 个盒子空着，所以去掉它们对方案数没有影响，即 $f[i][j] = f[j][j]$
- 否则，分两种情况讨论：
  - 如果至少有一个盒子空着，那么去掉它不影响方案数，即 $f[i-1][j]$
  - 否则，从每个盒子中都去掉一个球，不影响方案数，即 $f[i][j-i]$

  即 $f[i][j] = f[i-1][j] + f[i][j-i]$

1.10 Stern-Brocot 树与 Farey 序列

1.10.1 Stern-Brocot 树



第 i 层的序列是深度为 $i-1$ 的 Stern-Brocot 树的中序遍历。
性质：

- 单调性

  在每一层的序列中，真分数是单调递增的。

  在满足 $\frac{a}{b} \le \frac{c}{d}$，即要证明 $\frac{a}{b} \le \frac{a+c}{b+d} \le \frac{c}{d}$。

  此处只证一边，有 $\frac{a}{b} \le \frac{c}{d} \to ad \le bc \to ad + ab \le bc + ab \to \frac{a}{b} \le \frac{a+c}{b+d}$。

- 最简性

  序列中的分数 (除了 $\frac{0}{1}, \frac{1}{0}$), 都是最简分数。

1.10.2 Farey 序列

第 i 个 Farey 序列记作 F_i, 表示把分母小于等于 i 的所有最简真分数按大小顺序形成的排列。
Farey 序列 F_i 是 Stern-Brocot 第 $i-1$ 次迭代后的序列的子序列。
Farey 序列同样满足最简性与单调性，对于序列中连续的三个数 $\frac{a}{b}, \frac{x}{y}, \frac{c}{d}$，有 $x = a + c, y = b + d$。

F_i 的长度 $L_i = 1 + \sum_{k=1}^{i} \varphi(k)$。

1.10.3 SPOJ-DIVCNT1

题意:

计算 $S(n) = \sum_{i=1}^{n} d(i)$，其中 $d(i)$ 表示 i 的正因子个数。

时间复杂度 $O(n^{\frac{1}{3}} \log n)$

```
#include<bits/stdc++.h>
typedef long long ll;
typedef __int128 u128;
using namespace std;
const int N = 1e7 + 5;

namespace DIVCNT {
    struct point{
        ll x, y;
        point(ll _x = 0,ll _y = 0){ x = _x; y = _y; }
        point operator + (const point &t) const {
            return point(x + t.x, y + t.y);
        }
    }st[N], L, R, M;
    ll n;
    inline bool inR(ll x,ll y) { return x * y <= n; }
    inline double slope(ll x) { return (double)n / x / x;}
    inline u128 gao(ll _n){
        n = _n;
        u128 ret = 0;
        int t = 0, rt = cbrt(n);
        st[++t] = point(1, 0);
        st[++t] = point(1, 1);
        ll m = sqrt(n), x = n / m, y = m + 1;
        while (1) {
            for (L = st[t--]; !inR(x + L.x, y - L.y); x += L.x, y -= L.y)
                ret += x * L.y + (L.y + 1) * (L.x - 1) / 2;
            if (y <= rt) break;
            for (R = st[t]; inR(x + R.x, y - R.y); R = st[--t]) L = R;
            while (1){
                M = L + R;
                if (!inR(x + M.x, y - M.y)) st[++t] = (R = M);
                else {
                    if (slope(x + M.x) <= (double)R.y / R.x) break;
                    L = M;
                }
            }
        }
        for (int i = 1; i < y; i++) ret += n / i;
        return ret * 2 - 1ll * m * m;
    }
};

inline void write(u128 x){
    if (x >= 10) write(x / 10);
    putchar(x % 10 + '0');
}

inline void writeln(const u128 &x){
    write(x);
    putchar('\n');
}
```

```
53
54 int main(){
55     int _T; scanf("%d", &_T);
56     while (_T--) {
57         ll n;
58         scanf("%lld", &n);
59         writeln(DIVCNT::gao(n));
60     }
61     return 0;
62 }
```

1.11 各种表

1.11.1 因子个数表

- 10^5 内, 拥有最大因子个数的数是 83160, 有 128 个因子
- 10^6 内, 拥有最大因子个数的数是 720720, 有 240 个因子
- 10^7 内, 拥有最大因子个数的数是 8648640, 有 448 个因子
- 10^8 内, 拥有最大因子个数的数是 73513440, 有 768 个因子
- 10^9 内, 拥有最大因子个数的数是 735134400, 有 1344 个因子
- 10^{10} 内, 拥有最大因子个数的数是 6983776800, 有 2304 个因子
- 10^{11} 内, 拥有最大因子个数的数是 97772875200, 有 4032 个因子
- 10^{12} 内, 拥有最大因子个数的数是 963761198400, 有 6720 个因子
- 10^{13} 内, 拥有最大因子个数的数是 9316358251200, 有 10752 个因子
- 10^{14} 内, 拥有最大因子个数的数是 97821761637600, 有 17280 个因子
- 10^{15} 内, 拥有最大因子个数的数是 866421317361600, 有 26880 个因子
- 10^{16} 内, 拥有最大因子个数的数是 8086598962041600, 有 41472 个因子
- 10^{17} 内, 拥有最大因子个数的数是 74801040398884800, 有 64512 个因子

2 图论

2.1 Dijkstra

2.1.1 无向图最小环

CF 1325E
枚举每个起点, 跑 *Dijkstra*，对于其中没有状态转移的边，$< u, v >$, 此时 $dis[u] + dis[v] + 1$ 是一个环
时间复杂度 $O(n^2 \log E)$

```
1  int pre[N], dis[N], vis[N];
2  void bfs(int u) {
3      queue <int> que;
4      for (int i = 1; i < N; ++i) {
5          dis[i] = INF;
6          vis[i] = 0;
7          pre[i] = 0;
8      }
9      pre[u] = u;
10     dis[u] = 0;
11     que.push(u);
```

```
    while (!que.empty()) {
        int u = que.front(); que.pop();
        for (auto &v : G[u]) {
            if (!vis[v] && dis[v] > dis[u] + 1) {
                vis[v] = 1;
                dis[v] = dis[u] + 1;
                pre[v] = u;
                que.push(v);
            } else {
                if (pre[v] != u && pre[u] != v && dis[v] + dis[u] >= 2)
                    chmin(res, dis[v] + dis[u] + 1);
            }
        }
    }
}
```

2.2 Johnson 全源最短路

添加一个 0 号点，向其它每个点连一条边权为 0 的边。

用 $SPFA$ 以 0 号点为源点跑最短路，记为 h_i, 加入存在一条从 u 到 v 的边, 边权为 w 的边, 则将该边的边权重新设置为 $w + h_u - h_v$

再以每个点为起点，跑 n 轮 $Dijkstra$ 即可，算法复杂度 $O(nm \log n)$

正确性证明：

重新标注的图上，一条从 s 到 t 的路径 $s \to p_1 \to p_2 \to \cdots \to p_k \to t$ 的长度为：

$$(w(s,p_1) + h_s - h_{p_1}) + (w(p_1,p_2) + h_{p_1} - h_{p_2}) + \cdots + (w(p_k,t) + h_{p_k} - h_t)$$

化简后为：

$$w(s,p_1) + w(p_1,p_2) + \cdots + w(p_k,t) + h_s - h_t$$

所以对于 $s \to t$ 的所有路径中，$h_s - h_t$ 的值不变，可以类比物理中的势能。

所以新图上的最短路等价于原图的最短路。

根据三角形不等式，新图上任意一条边满足 $h_v \le h_u + w(u,v)$，重新标记后为 $w'(u,v) = w(u,v) + h_u - h_v \ge 0$，满足非负。

洛谷 P5905

题意：给出 n 个点 m 条边的带权有向图, 求所有点对的最短路径长度, 边权可能非负，图中可能有重边和自环。

若图中有负环，输出 -1

$1 \le n \le 3 \cdot 10^3, 1 \le m \le 6 \cdot 10^3$

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 6e3 + 10, INF = 1e9;
int n, m;

struct Graph {
    struct E { int to, nx; ll w; }e[N << 1]; int h[N], cnt;
    void init(int n) { for (int i = 0; i <= n; ++i) h[i] = -1; cnt = -1; }
    void addedge(int u, int v, ll w = 0) { e[++cnt] = { v, h[u], w}; h[u] =
       cnt; }
}G;

struct SPFA {
    ll dis[N]; int cnt[N]; bool used[N];
```

```
    // true 没有负环
    // false 有负环
    bool gao(int s) {
        for (int i = 0; i <= n; ++i) {
            dis[i] = INF;
            cnt[i] = 0;
            used[i] = 0;
        }
        queue <int> que;
        dis[s] = 0; used[s] = 1;
        que.push(s);
        while (!que.empty()) {
            int u = que.front(); que.pop();
            used[u] = 0;
            for (int i = G.h[u]; ~i; i = G.e[i].nx) {
                int v = G.e[i].to; ll w = G.e[i].w;
                if (dis[v] > dis[u] + w) {
                    dis[v] = dis[u] + w;
                    if (!used[v]) {
                        used[v] = 1;
                        que.push(v);
                        ++cnt[v];
                        if (cnt[v] == n) return false;
                    }
                }
            }
        }
        return true;
    }
}spfa;

struct Dijkstra {
    struct node {
        int u; ll w;
        node(int u = 0, ll w = 0) : u(u), w(w) {}
        bool operator < (const node &other) const { return w > other.w; }
    };
    ll dis[N]; bool used[N];
    void gao(int s) {
        for (int i = 0; i <= n; ++i) {
            dis[i] = INF;
            used[i] = 0;
        }
        priority_queue <node> pq;
        dis[s] = 0;
        pq.push(node(s, dis[s]));
        while (!pq.empty()) {
            int u = pq.top().u; pq.pop();
            if (used[u]) continue;
            used[u] = 1;
            for (int i = G.h[u]; ~i; i = G.e[i].nx) {
                int v = G.e[i].to; ll w = G.e[i].w;
                if (!used[v] && dis[v] > dis[u] + w) {
                    dis[v] = dis[u] + w;
                    pq.push(node(v, dis[v]));
                }
            }
```

```
72          }
73      }
74  }dij;
75
76  int main() {
77      scanf("%d%d", &n, &m);
78      G.init(n);
79      for (int i = 1, u, v, w; i <= m; ++i) {
80          scanf("%d%d%d", &u, &v, &w);
81          G.addedge(u, v, w);
82      }
83      for (int i = 1; i <= n; ++i)
84          G.addedge(0, i, 0);
85      if (!spfa.gao(0)) {
86          puts("-1");
87          return 0;
88      }
89      ll *h = spfa.dis;
90      for (int u = 1; u <= n; ++u) {
91          for (int i = G.h[u]; ~i; i = G.e[i].nx) {
92              int v = G.e[i].to;
93              G.e[i].w += h[u] - h[v];
94          }
95      }
96      for (int i = 1; i <= n; ++i) {
97          dij.gao(i);
98          ll *f = dij.dis;
99          ll res = 0;
100         for (int j = 1; j <= n; ++j) {
101             if (f[j] == INF) res += 1ll * j * INF;
102             else res += 1ll * j * (f[j] + h[j] - h[i]);
103         }
104         printf("%lld\n", res);
105     }
106     return 0;
107 }
```

2.3 最小生成树

2.3.1 CDQ 分治动态维护

BZOJ 2001
题意:
n 个城市, m 条边，q 次修改边权操作，修改不独立，每次修改后给出当前图最小生成树权值和。
思路:

- Reduction:

  把修改 $L-R$ 所影响到的所有边的边权设为 ∞，然后跑 MST，则此时不在修改边集并且不在最小生成树的边集里的边直接舍去, 因为不管怎样这些边都不会被选中。

- Contraction:

  把修改 $L-R$ 所影响到的所有边的边权设为 $-\infty$，然后跑 MST，则此时不在修改边集并且在最小生成树的边集里的边直接选中, 因为不管怎样这些边都会被选中，在下一层分治操作中就不用管这些边了，我们假设所有锁定边为 *must*，往下传就好了。

考虑每一层的边数与当前图的点数同阶，每次缩边缩掉的是 $-\infty$ 生成树中的未修改边，相当于割掉了 $-\infty$ 生成树中的修改边, 显然最多割掉 $O(len)$ 条边, len 为当前层修改边条树, 那么最多分裂成 $O(len)$ 个联通块, 每层图的点数就是 $O(len)$ 级别的。

其实不用回滚并茶集，每次直接将当前边集涉及到的点清空以下即可，因为清空复杂度和当前边数同阶。总的复杂度为 $O(n\log^2 n)$

```cpp
#include <bits/stdc++.h>
#define SZ(x) (int(x.size()))
using namespace std;
typedef long long ll;
const int N = 5e4 + 10;
const int INF = 0x3f3f3f3f;
int n, m, q, eW[N];

struct Edge {
    int u, v, id; ll w;
    bool operator < (const Edge &other) const { return w < other.w; }
}e[N];

struct qNode { int id; ll val, ans; } qrr[N];

struct UFS {
    int fa[N];
    int find(int x) { return fa[x] == 0 ? x : fa[x] = find(fa[x]); }
    bool merge(int u, int v) {
        int fu = find(u), fv = find(v);
        if (fu != fv) {
            fa[fu] = fv;
            return true;
        }
        return false;
    }
    void reset(const vector <Edge> &e) { for (int i = 0; i < SZ(e); ++i) fa[e[i].u] = fa[e[i].v] = 0; }
}ufs;

namespace CDQ {
    int tag[N];
    vector <Edge> t[30], tmp, ttmp;
    //删除无用边
    void Reduction() {
        ufs.reset(tmp);
        sort(tmp.begin(), tmp.end());
        int cur = 0;
        for (int i = 0, u, v; i < SZ(tmp); ++i) {
            u = tmp[i].u, v = tmp[i].v;
            if (tmp[i].w == INF || ufs.merge(u, v)) {
                tmp[cur] = tmp[i];
                tag[tmp[cur].id] = cur;
                cur++;
            }
        }
        tmp.resize(cur);
    }
    //缩掉必须边
    void Contraction(ll &must) {
        ufs.reset(tmp);
        sort(tmp.begin(), tmp.end());
        ttmp.clear();
        for (int i = 0, u, v; i < SZ(tmp); ++i) {
            u = tmp[i].u, v = tmp[i].v;
            if (ufs.merge(u, v)) {
```

```
56                  ttmp.push_back(tmp[i]);
57              }
58          }
59          ufs.reset(ttmp);
60          for (int i = 0, u, v; i < SZ(ttmp); ++i) {
61              u = ttmp[i].u, v = ttmp[i].v;
62              if (ttmp[i].w != -INF && ufs.merge(u, v)) {
63                  must += ttmp[i].w;
64              }
65          }
66          int cur = 0;
67          for (int i = 0; i < SZ(tmp); ++i) {
68              int u = ufs.find(tmp[i].u), v = ufs.find(tmp[i].v);
69              if (u != v) {
70                  tmp[cur] = tmp[i];
71                  tag[tmp[cur].id] = cur;
72                  tmp[cur].u = u;
73                  tmp[cur].v = v;
74                  ++cur;
75              }
76          }
77          tmp.resize(cur);
78      }
79      void solve(int l, int r, int dep, ll must) {
80          if (l == r) eW[qrr[l].id] = qrr[l].val;
81          for (int i = 0; i < SZ(t[dep]); ++i) {
82              t[dep][i].w = eW[t[dep][i].id];
83              tag[t[dep][i].id] = i;
84          }
85          tmp = t[dep];
86          if (l == r) {
87              qrr[l].ans = must;
88              ufs.reset(tmp);
89              sort(tmp.begin(), tmp.end());
90              for (int i = 0, u, v; i < SZ(tmp); ++i) {
91                  u = tmp[i].u, v = tmp[i].v;
92                  if (ufs.merge(u, v)) {
93                      qrr[l].ans += tmp[i].w;
94                  }
95              }
96              return;
97          }
98          for (int i = l; i <= r; ++i) tmp[tag[qrr[i].id]].w = INF;
99          Reduction();
100         for (int i = l; i <= r; ++i) tmp[tag[qrr[i].id]].w = -INF;
101         Contraction(must);
102         t[dep + 1] = tmp;
103         int mid = (l + r) >> 1;
104         solve(l, mid, dep + 1, must);
105         solve(mid + 1, r, dep + 1, must);
106     }
107     void gao() {
108         memset(tag, 0, sizeof tag);
109         t[0].clear();
110         for (int i = 1; i <= m; ++i)
111             t[0].push_back(e[i]);
112         solve(1, q, 0, 0);
113     }
```

```
}

int main() {
    scanf("%d%d%d", &n, &m, &q);
    for (int i = 1; i <= m; ++i) {
        e[i].id = i;
        scanf("%d%d%lld", &e[i].u, &e[i].v, &e[i].w);
        eW[i] = e[i].w;
    }
    for (int i = 1; i <= q; ++i) {
        scanf("%d%lld", &qrr[i].id, &qrr[i].val);
    }
    CDQ::gao();
    for (int i = 1; i <= q; ++i) printf("%lld\n", qrr[i].ans);
    return 0;
}
```

2.4 二分图最大权匹配 (KM)

UOJ80

KM 本质是求最佳匹配的 (即先保证匹配边最多，再保证权值最大)，但是可以通过添加权值为 0 的边使得肯定存在完备匹配，那么就是求最大权匹配了。

几个概念：

- 顶标：给每个顶点赋一个数值，成为该顶点的定标。
- 可行顶标：如果对于所有的边，满足此边两端的两个顶点的定标和不小于此边权重，则称这一组顶标为可行顶标。
- 相等子图：在当前顶标下，满足边两端的两个顶点的定标和正好等于此边权重的所有边，所构成的子图。

相等子图的性质：

- 在任意时刻，相等子图上的最大权匹配一定小于等于相等子图的顶标和。
- 在任意时刻，相等子图的顶标和即为所有顶点的顶标和。
- 扩充相等子图后，相等子图的顶标和将会减小。
- 当相等子图的最大匹配为原图的完备匹配时，匹配边的权值和等于所有顶点的顶标和，此匹配即为最佳匹配

KM 算法过程中任意时刻都满足 $x_i + y_j \le w_{i,j}$，所以最终可以求得 $\sum x_i + \sum y_j$ 的最小值，但是相等子图上的最大权匹配一定小于等于相等子图的顶标和，所以取等号时，边权和最大，顶标和最小。

相反，如果要满足 $x_i + y_j \le w_{i,j}$，并且使得 $\sum x_i + \sum y_j$ 最大，只需要取负号满足 $-(x_i + y_j) \le -w_{i,j}$，最后将结果也取负即可。

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
//时间复杂度O(nx * nx * ny)
//点的编号从1开始
struct KM {
    static const int N = 4e2 + 10;
    static const int INF = 0x3f3f3f3f;
    int lx[N], ly[N], slack[N], w[N][N]; //以上为权值类型
    int vx[N], vy[N], pre[N], left[N], right[N], nl, nr, n;
    void match(int& u) {
        for (; u; swap(u, right[pre[u]]))
            left[u] = pre[u];
```

```
    }
    void bfs(int u) {
        static int q[N], front, rear;
        front = 0; vx[q[rear = 1] = u] = true;
        while (true) {
            while (front < rear) {
                int u = q[++front];
                for (int v = 1; v <= n; ++v) {
                    int tmp;
                    if (vy[v] || (tmp = lx[u] + ly[v] - w[u][v]) > slack[v])
                        continue;
                    pre[v] = u;
                    if (!tmp) {
                        if (!left[v]) return match(v);
                        vy[v] = vx[q[++rear] = left[v]] = true;
                    } else {
                        slack[v] = tmp;
                    }
                }
            }
            int a = INF;
            for (int i = 1; i <= n; ++i) {
                if (!vy[i] && a > slack[i]) a = slack[u = i];
            }
            for (int i = 1; i <= n; ++i) {
                if (vx[i]) lx[i] -= a;
                if (vy[i]) ly[i] += a;
                else slack[i] -= a;
            }
            if (!left[u]) return match(u);
            vy[u] = vx[q[++rear] = left[u]] = true;
        }
    }
    void exec() {
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= n; ++j) {
                slack[j] = INF;
                vx[j] = vy[j] = false;
            }
            bfs(i);
        }
    }
    void init(int _nl, int _nr) {
        nl = _nl, nr = _nr;
        n = max(nl, nr);
        for (int u = 1; u <= n; ++u) {
            lx[u] = ly[u] = 0;
            left[u] = right[u] = 0;
            for (int v = 1; v <= n; ++v) {
                w[u][v] = 0;
            }
        }
    }
    ll work() {
        for (int u = 1; u <= n; ++u) {
            for (int v = 1; v <= n; ++v) {
                lx[u] = max(lx[u], w[u][v]);
            }
```

```
        }
        exec();
        ll ans = 0;
        for (int i = 1; i <= n; ++i) {
            ans += lx[i] + ly[i];
        }
        return ans;
    }
    void output() {
        for (int i = 1; i <= nl; ++i) {
            printf("%d%c", (w[i][right[i]] ? right[i] : 0), " \n"[i == nl]);
        }
    }
}km;
int nl, nr, m;

int main() {
    while (scanf("%d%d%d", &nl, &nr, &m) != EOF) {
        km.init(nl, nr);
        for (int i = 1, u, v, w; i <= m; ++i) {
            scanf("%d%d%d", &u, &v, &w);
            km.w[u][v] = w;
        }
        printf("%lld\n", km.work());
        km.output();
    }
    return 0;
}
```

3 离散化

```
struct Hash {
    vector <int> a;
    int& operator[](int x) { return a[x - 1]; }
    int size() { return a.size(); }
    void init() { a.clear(); }
    void add(int x) { a.push_back(x); }
    void gao() { sort(a.begin(), a.end()); a.erase(unique(a.begin(), a.end()
        ), a.end()); }
    int get(int x) { return lower_bound(a.begin(), a.end(), x) - a.begin() +
        1; }
}hs;
```

4 PB-DS

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>//用tree
#include<ext/pb_ds/hash_policy.hpp>//用hash
#include<ext/pb_ds/trie_policy.hpp>//用trie
#include<ext/pb_ds/priority_queue.hpp>//用priority_queue
using namespace __gnu_pbds;
//或者:
#include<bits/extc++.h>
using namespace __gnu_pbds;
```

```
10 //bits/extc++.h与bits/stdc++.h类似，bits/extc++.h是所有拓展库，bits/stdc++.h
       是所有标准库
```

4.1 Hash-Table

```
1  #include <bits/stdc++.h>
2  #include <bits/extc++.h>
3  using namespace std;
4  using namespace __gnu_pbds;
5  const int N = 1e5 + 10;
6  //拉链法
7  //cc_hash_table<string, int> h;
8
9  //探测法
10 gp_hash_table<string, int> h;
11 int n;
12 char s[N];
13
14 int main() {
15     while (scanf("%d", &n) != EOF) {
16         for (int i = 1; i <= n; ++i) {
17             scanf("%s", s + 1);
18             //支持find和[]
19             if (h.find(s + 1) == h.end()) {
20                 h[s + 1] = 1;
21             }
22         }
23         printf("%d\n", (int)h.size());
24     }
25     return 0;
26 }
```

4.2 Hash-Map

```
1  #include<bits/stdc++.h>
2  #include<bits/extc++.h>
3  #define fi first
4  #define se second
5  using namespace std;
6
7  struct splitmix64_hash {
8      static uint64_t splitmix64(uint64_t x) {
9          x += 0x9e3779b97f4a7c15;
10         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
11         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
12         return x ^ (x >> 31);
13     }
14     size_t operator()(uint64_t x) const {
15         static const uint64_t FIXED_RANDOM = std::chrono::steady_clock::now
               ().time_since_epoch().count();
16         return splitmix64(x + FIXED_RANDOM);
17     }
18 };
19
20 template <typename K, typename V, typename Hash = splitmix64_hash>
21 using hash_map = __gnu_pbds::gp_hash_table<K, V, Hash>;
```

```
22
23 template <typename K, typename Hash = splitmix64_hash>
24 using hash_set = hash_map<K, __gnu_pbds::null_type, Hash>;
25
26 int main() {
27     hash_map<int64_t, int> mp;
28     int n = 1e7;
29     for (int i = 1; i <= n; ++i) {
30         mp[i] = 1;
31     }
32     long long res = 0;
33     for (auto &it : mp)
34         res += it.fi;
35     cout << res << endl;
36     return 0;
37 }
```

4.3 Priority-Queue

一切的开始:

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/priority_queue.hpp>
3 using namespace __gnu_pbds;
```

4.3.1 TAG 类型

priority_queue<int,greater<int>,TAG> Q;//小根堆，大根堆写 less<int>
其中的 TAG 为类型，有以下几种:

- pairing_heap_tag push 和 join 为 $O(1)$，其余为均摊 $\Theta(logn)$
- thin_heap_tag 只支持 push 和 pop，均为均摊 $\Theta(logn)$
- binomial_heap_tag push 为均摊 $O(1)$, 其余为 $\Theta(logn)$
- rc_binomial_heap_tag push 为 $O(1)$，其余为 $\Theta(logn)$
- binary_heap_tag push 为 $O(1)$, 不支持 join, 其余为 $\Theta(logn)$, 但是如果只有 increase_key，那么 modify 为均摊 $O(1)$
- 其中 pairing_heap_tag 最快，并且默认就为 pairing_heap_tag

选择:

- 在只有 push, pop 的时候, binary_heap_tag 速度比较快，在绝大多数情况下优于 std::priority_queue
- 在有 modify 操作的时候，可以考虑采用 pairing_heap_tag, thin_heap_tag 或手写数据结构
- 只有 push, pop, join 的时候，可以考虑用 pairing_heap_tag 或 binomial_heap_tag

4.3.2 基本操作

- it = Q.push(x); 插入时会返回迭代器
- Q.clear();
- Q.pop();
- Q.top();
- Q.join(b); // b 元素加入到 Q 中，b 清空

- Q.empty();
- Q.size();
- Q.modify(it,6);
- Q.erase(it);
- 迭代器是 point_iterator，可以用迭代器遍历整个优先队列

4.3.3 BZOJ 3040

堆优化 Dijkstra

```
#include <bits/stdc++.h>
#include <bits/extc++.h>
using namespace std;
using namespace __gnu_pbds;

#define ll long long
#define INFLL 0x3f3f3f3f3f3f3f3f
#define N 1000010
#define M 10000010
int n, m, T, rxa, rxc, rya, ryc, rp;
struct Graph {
    struct node {
        int to, nx, w;
        node() {}
        node (int to, int nx, int w) : to(to), nx(nx), w(w) {}
    }a[M];
    int head[N], pos;
    void init() {
        memset(head, -1, sizeof head);
        pos = 0;
    }
    inline void add(int u, int v, int w) {
        a[pos] = node(v, head[u], w); head[u] = pos++;
    }
}G;
#define erp(u) for (int it = G.head[u], v = G.a[it].to, w = G.a[it].w; ~it;
   it = G.a[it].nx, v = G.a[it].to, w = G.a[it].w)

struct node {
    int u; ll w;
    node() {}
    node (int u, ll w) : u(u), w(w) {}
    bool operator < (const node &other) const {
        return w > other.w;
    }
};
//less是大根堆，表示使用!<来当排序规则，priority_queue 默认用的也是less
#define heap __gnu_pbds::priority_queue <node, less<node>, pairing_heap_tag>

ll dis[N];
heap::point_iterator id[N];
heap pq;
void Dijkstra() {
    for (int i = 2; i <= n; ++i) {
        dis[i] = INFLL;
        id[i] = 0;
```

```
    }
    while (!pq.empty()) pq.pop();
    dis[1] = 0;
    id[1] = pq.push(node(1, 0));
    while (!pq.empty()) {
        int u = pq.top().u; pq.pop();
        erp(u) if (dis[v] > dis[u] + w) {
            dis[v] = dis[u] + w;
            if (id[v] == 0) {
                id[v] = pq.push(node(v, dis[v]));
            } else {
                pq.modify(id[v], node(v, dis[v]));
            }
        }
    }
}

int main() {
    while (scanf("%d%d", &n, &m) != EOF) {
        scanf("%d%d%d%d%d%d", &T, &rxa, &rxc, &rya, &ryc, &rp);
        G.init();
        ll x = 0, y = 0, a, b;
        for (int i = 1; i <= T; ++i) {
            x = (x * rxa + rxc) % rp;
            y = (y * rya + ryc) % rp;
            a = min(x % n + 1, y % n + 1);
            b = max(x % n + 1, y % n + 1);
            G.add(a, b, 100000000 - 100 * a);
        }
        for (int i = 1, u, v, w; i <= m - T; ++i) {
            scanf("%d%d%d", &u, &v, &w);
            G.add(u, v, w);
        }
        Dijkstra();
        printf("%lld\n", dis[n]);
    }
    return 0;
}
```

4.4 Tree

TAG 有 rb_tree, splay_tree, ov_tree, 后两种较慢，一般使用 rb_tree

4.4.1 基本操作

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
#define pII pair<int,int>
#define mp(x,y) make_pair(x,y)
tree<pII,null_type,less<pII>,rb_tree_tag,tree_order_statistics_node_update>
    tr;
pII //存储的类型
null_type //无映射(低版本g++为null_mapped_type)
如果是map，这里是Mapped，即map中的第二关键字
less<pII> //从小到大排序
rb_tree_tag //红黑树
```

```
tree_order_statistics_node_update //更新方式
tr.insert(mp(x,y)); //插入;
tr.erase(mp(x,y)); //删除;
tr.order_of_key(pII(x,y)); //求排名
tr.find_by_order(x); //找第x + 1小的元素的迭代器，如果x太大会返回end()
tr.join(b); //将b并入tr，前提是两棵树类型一样且没有重复元素
tr.split(v,b); //分裂，key小于等于v的元素属于tr，其余的属于b
tr.lower_bound(x); //返回第一个大于等于x的元素的迭代器
tr.upper_bound(x); //返回第一个大于x的元素的迭代器
//以上所有操作的时间复杂度均为O(logn)
```

4.4.2 Luogu P3369

题意:
维护一些操作:

1. 插入 x 数
2. 删除 x 数 (若有多个相同的数，应只删除一个)
3. 查询 x 数的排名 (若有多个相同的数，应输出最小的排名)
4. 查询排名为 x 的数
5. 求 x 的前驱 (前驱定义为小于 x，且最大的数)
6. 求 x 的后继 (后继定义为大于 x，且最小的数)

```
#include <bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
using pII = pair<int, int>;
#define fi first
#define se second
tree<pII,null_type,less<pII>,rb_tree_tag,tree_order_statistics_node_update>
   rbtree;

int main() {
    int n; scanf("%d", &n);
    rbtree.clear();
    int op, x;
    for (int i = 1; i <= n; ++i) {
        scanf("%d%d", &op, &x);
        if (op == 1) {
            rbtree.insert(pII(x, i));
        } else if (op == 2) {
            rbtree.erase(rbtree.lower_bound(pII(x, 0)));
        } else if (op == 3) {
            printf("%d\n", (int)rbtree.order_of_key(pII(x, 0)) + 1);
        } else if (op == 4) {
            int ans = rbtree.find_by_order(x - 1)->fi;
            printf("%d\n", ans);
        } else if (op == 5) {
            int ans = (--rbtree.lower_bound(pII(x, 0)))->fi;
            printf("%d\n", ans);
        } else if (op == 6) {
            int ans = rbtree.upper_bound(pII(x, n))->fi;
```

```
            printf("%d\n", ans);
        }
    }
    return 0;
}
```

4.4.3 CF 459D

以 pari<int, int> 作为键值，int 作为权值能够查询区间权值和的 map。
只能插入删除，不可使用 [] 操作符。

```
#include <bits/stdc++.h>
#include <bits/extc++.h>
#define fi first
#define se second
using namespace __gnu_pbds;
using namespace std;
using pII = pair <int, int>;
using ll = long long;
constexpr int N = 1e6 + 10;
int n, m, a[N], f[N];
unordered_map <int, int> cnt;

template<class Node_CItr, class Node_Itr, class Cmp_Fn, class _Alloc>
struct map_node_update {
    virtual Node_CItr node_begin() const = 0;
    virtual Node_CItr node_end() const = 0;
    typedef int metadata_type;
    const static int INF = 0x3f3f3f3f;
    void operator() (Node_Itr it, Node_CItr end_it) {
        Node_Itr l = it.get_l_child();
        Node_Itr r = it.get_r_child();
        metadata_type left = 0, right = 0;
        if(l != end_it) left = l.get_metadata();
        if(r != end_it) right = r.get_metadata();
        const_cast<metadata_type &>(it.get_metadata()) =
            left + right + (*it)->second;
    }
    metadata_type querysum(pII x) {
        metadata_type ans = 0;
        Node_CItr it = node_begin();
        while (it != node_end()) {
            Node_CItr l = it.get_l_child();
            Node_CItr r = it.get_r_child();
            //x比当前节点要小，走向左儿子
            if(Cmp_Fn()(x, (*it)->first)) {
                it = l;
            } else {
                //x大于等于当前节点
                ans += (*it)->second;
                if(l != node_end()) ans += l.get_metadata();
                it = r;
            }
        }
        return ans;
    }
    //[l, r]范围内的权值
    metadata_type querysum(int l, int r) {
```

```
48          if (l > r) return 0;
49          return querysum(pII(r, INF)) - querysum(pII(l, -INF));
50      }
51  };
52  //pII fi key se id Mapeed value
53  //不可使用非标准的[]操作, 为了防止重复,用pair当键值, se是id
54  tree<pII, int, less<pII>, rb_tree_tag, map_node_update> rbtree;
55
56  int main() {
57      scanf("%d", &n);
58      cnt.clear();
59      for (int i = 1; i <= n; ++i) scanf("%d", a + i);
60      for (int i = 1; i <= n; ++i) {
61          ++cnt[a[i]];
62          f[i] = cnt[a[i]];
63      }
64      cnt.clear();
65      rbtree.clear();
66      ll res = 0;
67      for (int i = n; i >= 1; --i) {
68          ++cnt[a[i]];
69          res += rbtree.querysum(1, f[i] - 1);
70          rbtree.insert(make_pair(pII(cnt[a[i]], i), 1));
71      }
72      printf("%lld\n", res);
73      return 0;
74  }
```

4.5 Trie

4.5.1 基本操作

```
1  typedef trie<string,null_type,trie_string_access_traits<>,pat_trie_tag,
       trie_prefix_search_node_update> tr;
2  //第一个参数必须为字符串类型, tag也有别的tag, 但pat最快, 与tree相同,
       node_update支持自定义
3  tr.insert(s); //插入s
4  tr.erase(s); //删除s
5  tr.join(b); //将b并入tr
6  //pair的使用如下:
7  pair<tr::iterator,tr::iterator> range=tr.prefix_range(s);
8  for(tr::iterator it = range.first; it != range.second; it++)
9      cout << *i t<< ' ' <<endl;
10 //pair中第一个是起始迭代器, 第二个是终止迭代器, 遍历过去就可以找到所有字符串
       了。
```

4.5.2 Times1414

```
1  #include<bits/stdc++.h>
2  #include<ext/pb_ds/assoc_container.hpp>
3  #include<ext/pb_ds/trie_policy.hpp>
4  using namespace std;
5  using namespace __gnu_pbds;
6  trie<string,null_type,trie_string_access_traits<>,pat_trie_tag,
       trie_prefix_search_node_update> tr;
7
```

```
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr); cout.tie(nullptr);
    tr.clear(); tr.insert("sun");
    string s;
    while (cin >> s) {
        if (s[0] == '?') {
            s = s.substr(1);
            cout << s << "\n";
            auto range = tr.prefix_range(s);
            int t = 0;
            for (auto it = range.first; t < 20 && it != range.second; ++t,
               ++it) {
                cout << "  " << *it << "\n";
            }
        } else {
            tr.insert(s.substr(1));
        }
    }
    return 0;
};
```

4.6 rope

一切的开始:

```
#include <ext/rope>
using namespace __gnu_cxx;
```

4.6.1 基本操作

- insert(int pos, T *s, int n)

  将字符串 *s* 的前 *n* 位插入 *rope* 的下标 *pos* 处, 如没有参数 *n* 则将字符串 *s* 的所有位都插入 *rope* 的下标 *pos* 处

- append(T *s,int pos,int n)

  把字符串 *s* 中从下标 *pos* 开始的 *n* 个字符连接到 *rope* 的结尾, 如没有参数 *n* 则把字符串 *s* 中下标 *pos* 后的所有字符连接到 *rope* 的结尾, 如没有参数 *pos* 则把整个字符串 *s* 连接到 *rope* 的结尾

- substr(int pos, int len)

  提取 *rope* 的从下标 *pos* 开始的 *len* 个字符

- at(int x)

  访问 *rope* 的下标为 *x* 的元素

- erase(int pos, int num)

  从 *rope* 的下标 *pos* 开始删除 *num* 个字符

- copy(int pos, int len, T *s)

  从 *rope* 的下标 *pos* 开始的 *len* 个字符用字符串 *s* 代替，如果 *pos* 后的位数不够就补足

- replace(int pos, T *s)

  从 *rope* 的下标 *pos* 开始替换成字符串 *s*，*s* 的长度为从 *pos* 开始替换的位数，如果 *pos* 后的位数不够就补足

- length()

  获取 rope 当前长度

原理是用块状链表实现，时间复杂度 $O(n\sqrt{n})$

4.6.2 进阶操作

- 区间翻转
  维护一正一反两个 *rope*, 翻转等价于交换两个子串
- 区间循环位移
  拆成多个子串，再连接起来
- 区间字符自增 1，即 $a->b, b->c, \cdots, z->a$
  维护 26 个 *rope*
- 可持久化
  维护根结点，可以直接复制之前版本的根结点是 $O(1)$ 的。
  可以参考 BZOJ 3673

4.6.3 BZOJ 1269

题意:

- MOVE(k)，将光标移动第 k 个字符后
- INSERT(n, s), 在光标后插入长度为 n 的字符串 s，光标位置不变
- DELETE(n), 删除光标后的 n 个字符
- ROTATE(n), 反转光标后的 n 个字符
- GET(), 输出光标后的第一个字符
- PREV(), 光标前移一个字符
- NEXT(), 光标后移一个字符

```
#include <bits/stdc++.h>
#include <bits/extc++.h>
using namespace std;
using namespace __gnu_cxx;
const int N = 2e6 + 10;
rope <char> A, B, T;
int n, pos, len;
char s[N], revs[N];

int main() {
    int _T;
    scanf("%d", &_T);
    char op[5];
    pos = 0;
    while (_T--) {
        scanf("%s", op);
        if (op[0] == 'M') {
            scanf("%d", &pos);
        } else if (op[0] == 'I') {
            scanf("%d", &n);
            for (int i = 0; i < n; ++i) {
                do { s[i] = getchar(); }
                while (s[i] == '\n');
                revs[n - i - 1] = s[i];
            }
            s[n] = revs[n] = 0;
            len = A.length();
```

```
            A.insert(pos, s);
            B.insert(len - pos, revs);
        } else if (op[0] == 'D') {
            scanf("%d", &n);
            len = A.length();
            A.erase(pos, n);
            B.erase(len - pos - n, n);
        } else if (op[0] == 'R') {
            scanf("%d", &n);
            len = A.length();
            T = A.substr(pos, n);
            A = A.substr(0, pos) + B.substr(len - pos - n, n) + A.substr(pos
                + n, len - pos - n);
            B = B.substr(0, len - pos - n) + T + B.substr(len - pos, pos);
        } else if (op[0] == 'G') {
            printf("%c\n", A.at(pos));
        } else if (op[0] == 'P') {
            --pos;
        } else if (op[0] == 'N') {
            ++pos;
        }
    }
    return 0;
}
```

4.6.4 BZOJ 3673

- 1 a b，合并 a, b 所在集合
- 2 k，回到第 k 次操后之后的状态 (查询算一次操作)
- 3 a b，询问 a 和 b 是否在同一集合

```
#include <bits/stdc++.h>
#include <bits/extc++.h>
using namespace std;
using namespace __gnu_cxx;
const int N = 2e5 + 10;
int n, q, a[N];

struct UFS {
    rope <int> *fa;
    int find(int x) {
        if (fa->at(x) == 0) return x;
        fa->replace(x, find(fa->at(x)));
        return fa->at(x);
    }
    void merge(int x, int y) {
        int fx = find(x), fy = find(y);
        if (fx != fy) {
            fa->replace(fx, fy);
        }
    }
    bool same(int x, int y) { return find(x) == find(y); }
}ufs[N];

int main() {
    scanf("%d%d", &n, &q);
    memset(a, 0, sizeof (a[0]) * (n + 5));
```

```
27      //用数组新建rope
28      ufs[0].fa = new rope<int> (a, a + 1 + n);
29      for (int i = 1, op, x, y, k; i <= q; ++i) {
30          scanf("%d", &op);
31          //复制之前版本的rope，不会影响之前的版本
32          ufs[i].fa = new rope<int> (*ufs[i - 1].fa);
33          if (op == 1) {
34              scanf("%d%d", &x, &y);
35              ufs[i].merge(x, y);
36          } else if (op == 2) {
37              scanf("%d", &k);
38              //直接继承，会影响之前的版本
39              ufs[i].fa = ufs[k].fa;
40          } else if (op == 3) {
41              scanf("%d%d", &x, &y);
42              printf("%d\n", ufs[i].same(x, y));
43          }
44      }
45  }
```

4.6.5 Luogu P3391

题意:
维护序列的区间翻转

```
1   #include <bits/stdc++.h>
2   #include <bits/extc++.h>
3   using namespace std;
4   using namespace __gnu_cxx;
5   const int N = 1e5 + 10;
6   rope <int> A, B, T;
7   int n, q, a[N];
8
9   int main() {
10      while (scanf("%d%d", &n, &q) != EOF) {
11          A.clear(); B.clear();
12          for (int i = 1; i <= n; ++i) a[i] = i;
13          A = rope<int>(a, a + 1 + n);
14          reverse(a + 1, a + 1 + n);
15          B = rope<int>(a, a + 1 + n);
16          for (int i = 1, l, r; i <= q; ++i) {
17              scanf("%d%d", &l, &r);
18              T = A.substr(l, r - l + 1);
19              A = A.substr(0, (l - 1) + 1) + B.substr(n - r + 1, r - l + 1) +
                    A.substr(r + 1, n - (r + 1) + 1);
20              B = B.substr(0, n - r + 1) + T + B.substr(n - l + 2, n - (n - l
                    + 2) + 1);
21          }
22          for (int i = 1; i <= n; ++i)
23              printf("%d%c", A[i], " \n"[i == n]);
24      }
25      return 0;
26  }
```

5 广义 RMQ

普通的 *RMQ* 仅仅支持能够重复贡献信息的区间查询功能，如区间最值

但是我们可以通过一个黑科技来实现 $O(n\log n)+O(1)+O(n\log n)$ 的复杂度实现支持结合律和快速合并的信息的区间查询

这里以区间乘法为例：

设 $A_{k,i}=\prod_{j=\lfloor \frac{i}{2^k}\rfloor 2^k}^{i} a_j, B_{k,i}=\prod_{j=i}^{\lceil \frac{i+1}{2^k}\rceil 2^k-1} a_j$

$A_{k,i}$ 表示的是（$\le i$ 的最大的 2^k 的倍数）到 i 这一段数的乘积，$B_{k,i}$ 表示的是 i 到（$>i$ 的最小的 2^k 的倍数 -1）这一段数的乘积，这两个都可以 $O(n\log n)$ 预处理

对于询问 $[l,r]$，如果 $l=r$ 那么答案就是 a_l，下面讨论 $l<r$ 的情况

如果我们可以找到一个 k，使得 $[l+1,r]$ 中只有一个 2^k 的倍数，那么 $B_{k,l}A_{k,r}$ 就是答案

容易验证一个满足要求的 k 就是 $\log_2(\text{highbit}(l\oplus r))$。

CodeChef-SEGPROD

```
template <typename RMQItem, RMQItem base>
struct RMQ {
    vector <RMQItem> v;
    vector <vector<RMQItem>> tl, tr;
    //确立两数运算规则
    RMQItem op(const RMQItem &a, const RMQItem &b) { return 1ll * a * b % p;
        }
    int log2Up(int n) {
        int res = 0;
        while ((1 << res) < n) res++;
        return res;
    }
    //下标从0开始
    RMQ(const vector<RMQItem> &a) {
        int n = a.size();
        v = a;
        tl = tr = vector <vector<RMQItem>>(log2Up(n) + 1, vector <RMQItem>(n
            ));
        for (int k = 0; (1 << k) <= n; ++k) {
            int ones = (1 << k) - 1;
            RMQItem tmp = base;
            for (int i = 0; i < n; ++i) {
                tmp = op(tmp, a[i]);
                tl[k][i] = tmp;
                if ((i & ones) == ones) tmp = base;
            }
            tmp = base;
            for (int i = n - 1; i >= 0; --i) {
                tmp = op(tmp, a[i]);
                tr[k][i] = tmp;
                if ((i & ones) == 0) tmp = base;
            }
        }
    }
    RMQItem query(int l, int r) {
        if (l == r) return v[l];
        int num = 31 - __builtin_clz(l ^ r);
        return op(tl[num][r], tr[num][l]);
    }
};
```

6 线段树

6.1 扫描线

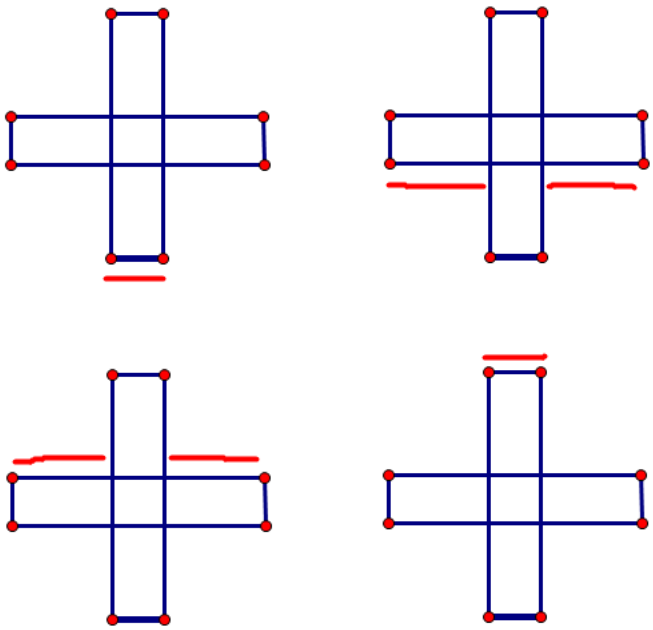
6.1.1 HDU 1828

题意:

给出 n 个矩形，求周长并。

求周长并只需要像面积并一样类似的维护线段，但是这样只能求宽，不能求高，但是我们把矩形转一下，让高变成宽再求一次即可。

每次得到的贡献需要和上一次做差。表示这次新增或删除的贡献。



```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 10;
int n;

namespace RectangularPerimeterUnion {
    int cor[2][N][2], m;
    struct E {
        int l, r, h, d;
        E() {}
        E(int l, int r, int h, int d) : l(l), r(r), h(h), d(d) {}
        bool operator < (const E &other) const { if (h != other.h) return h
            < other.h; return d > other.d; }
    }e[N << 1];
    struct Hash {
        vector <int> a;
        int operator[](int x) { return a[x - 1]; }
        int size() { return a.size(); }
        void init() { a.clear(); }
        void add(int x) { a.push_back(x); }
        void gao() { sort(a.begin(), a.end()); a.erase(unique(a.begin(), a.
            end()), a.end()); }
        int get(int x) { return lower_bound(a.begin(), a.end(), x) - a.begin
            () + 1; }
```

```
22      }hs;
23      struct SEG {
24          struct node {
25              int cnt, len;
26              node() { cnt = len = 0; }
27          }t[N << 2];
28          void build(int id, int l, int r) {
29              t[id] = node();
30              if (l == r) return;
31              int mid = (l + r) >> 1;
32              build(id << 1, l, mid);
33              build(id << 1 | 1, mid + 1, r);
34          }
35          void up(int id, int l, int r) {
36              if (t[id].cnt > 0) {
37                  t[id].len = hs[r] - hs[l - 1];
38              } else {
39                  if (l == r) {
40                      t[id].len = 0;
41                  } else {
42                      t[id].len = t[id << 1].len + t[id << 1 | 1].len;
43                  }
44              }
45          }
46          void update(int id, int l, int r, int ql, int qr, int v) {
47              if (ql > qr) return;
48              if (l >= ql && r <= qr) {
49                  t[id].cnt += v;
50                  up(id, l, r);
51                  return;
52              }
53              int mid = (l + r) >> 1;
54              if (ql <= mid) update(id << 1, l, mid, ql, qr, v);
55              if (qr > mid) update(id << 1 | 1, mid + 1, r, ql, qr, v);
56              up(id, l, r);
57          }
58      }seg;
59      int calc(int p) {
60          hs.init();
61          for (int i = 1; i <= n; ++i) {
62              hs.add(cor[p][i][0]);
63              hs.add(cor[p][i][1]);
64          }
65          hs.gao();
66          m = 0;
67          for (int i = 1; i <= n; ++i) {
68              int l = hs.get(cor[p][i][0]);
69              int r = hs.get(cor[p][i][1]);
70              e[++m] = E(l + 1, r, cor[p ^ 1][i][0], 1);
71              e[++m] = E(l + 1, r, cor[p ^ 1][i][1], -1);
72          }
73          sort(e + 1, e + 1 + m);
74          int _n = hs.size();
75          seg.build(1, 1, _n);
76          int last = 0;
77          int res = 0;
78          for (int i = 1; i <= m; ++i) {
79              int l = e[i].l, r = e[i].r;
```

```
            seg.update(1, 1, _n, l, r, e[i].d);
            int now = seg.t[1].len;
            res += abs(now - last);
            last = now;
        }
        return res;
    }
    int gao() {
        for (int i = 1; i <= n; ++i) {
            scanf("%d%d%d%d", &cor[0][i][0], &cor[1][i][0], &cor[0][i][1], &
               cor[1][i][1]);
        }
        return calc(0) + calc(1);
    }
}

int main() {
    while (scanf("%d", &n) != EOF) printf("%d\n", RectangularPerimeterUnion
       ::gao());
    return 0;
}
```

只做一遍，同时维护竖线。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 10;
int n;

namespace RectangularPerimeterUnion {
    int m;
    struct E {
        int l, r, h, d;
        E() {}
        E(int l, int r, int h, int d) : l(l), r(r), h(h), d(d) {}
        bool operator < (const E &other) const { if (h != other.h) return h
           < other.h; return d > other.d; }
    }e[N << 1];
    struct Hash {
        vector <int> a;
        int operator[](int x) { return a[x - 1]; }
        int size() { return a.size(); }
        void init() { a.clear(); }
        void add(int x) { a.push_back(x); }
        void gao() { sort(a.begin(), a.end()); a.erase(unique(a.begin(), a.
           end()), a.end()); }
        int get(int x) { return lower_bound(a.begin(), a.end(), x) - a.begin
           () + 1; }
    }hs;
    struct SEG {
        struct node {
            int cnt, len, vtl, lbd, rbd;
            node() { cnt = len = vtl = lbd = rbd = 0; }
        }t[N << 2];
        void build(int id, int l, int r) {
            t[id] = node();
            if (l == r) return;
            int mid = (l + r) >> 1;
            build(id << 1, l, mid);
            build(id << 1 | 1, mid + 1, r);
```

```
34          }
35          void up(int id, int l, int r) {
36              if (t[id].cnt > 0) {
37                  t[id].lbd = t[id].rbd = true;
38                  t[id].len = hs[r] - hs[l - 1];
39                  t[id].vtl = 2;
40              } else {
41                  if (l == r) {
42                      t[id] = node();
43                  } else {
44                      t[id].lbd = t[id << 1].lbd;
45                      t[id].rbd = t[id << 1 | 1].rbd;
46                      t[id].len = t[id << 1].len + t[id << 1 | 1].len;
47                      t[id].vtl = t[id << 1].vtl + t[id << 1 | 1].vtl;
48                      if (t[id << 1].rbd && t[id << 1 | 1].lbd) t[id].vtl -=
                            2;
49                  }
50              }
51          }
52          void update(int id, int l, int r, int ql, int qr, int v) {
53              if (ql > qr) return;
54              if (l >= ql && r <= qr) {
55                  t[id].cnt += v;
56                  up(id, l, r);
57                  return;
58              }
59              int mid = (l + r) >> 1;
60              if (ql <= mid) update(id << 1, l, mid, ql, qr, v);
61              if (qr > mid) update(id << 1 | 1, mid + 1, r, ql, qr, v);
62              up(id, l, r);
63          }
64      }seg;
65      int gao() {
66          hs.init();
67          m = 0;
68          for (int i = 1; i <= n; ++i) {
69              int x1, y1, x2, y2;
70              scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
71              e[++m] = E(x1, x2, y1, 1);
72              e[++m] = E(x1, x2, y2, -1);
73              hs.add(x1);
74              hs.add(x2);
75          }
76          hs.gao();
77          sort(e + 1, e + 1 + m);
78          int _n = hs.size();
79          seg.build(1, 1, _n);
80          int res = 0, last = 0;
81          e[m + 1].h = 0;
82          for (int i = 1; i <= m; ++i) {
83              int l = hs.get(e[i].l) + 1;
84              int r = hs.get(e[i].r);
85              seg.update(1, 1, _n, l, r, e[i].d);
86              res += seg.t[1].vtl * (e[i + 1].h - e[i].h);
87              int now = seg.t[1].len;
88              res += abs(now - last);
89              last = now;
90          }
```

```
91          return res;
92      }
93  }
94
95  int main() {
96      while (scanf("%d", &n) != EOF) printf("%d\n", RectangularPerimeterUnion
            ::gao());
97      return 0;
98  }
```

6.2 线段树合并与分裂

时间复杂度:

n 棵线段树初始有 $O(n\log n)$ 的节点, 总点数就是 $O(n\log n)$ 的

每次合并会将两个公共节点合并成一个，即删去一个节点，每个节点只会被删去一次, 所以时间复杂度为 $O(n\log n)$

每一个分裂会产生 $O(\log n)$ 个新节点

m 次分裂最多产生 $O(m\log n)$ 个新节点，所以即使降它们全部合并复杂度也只是 $O(m\log n)$

Luogu P2824

题意:

给出一个 $[1,n]$ 的排列, 进行 m 次局部排序。

- 0 l r, 将区间 $[l,r]$ 进行升序排序
- 1 l r, 将区间 $[l,r]$ 进行降序排序

考虑排序可以用合并权值线段树来做，并且保存升/降序的标记即可。

那么区间排序，如果用 *ODT* 的思想用 set 维护区间，并且每次分裂只会分裂出 $O(1)$ 个新区间，每个区间最多只会被合并一次，所以这部分复杂度为 $O(m\log n)$

然后用线段树合并与分裂来维护 set 中每段区间对应的权值线段树。

这里能分裂是因为每个小区间都是有序的，分裂相当于分割出权值线段树的一段前缀或者后缀

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   const int N = 1e5 + 10;
4   int n, q, a[N];
5
6   struct SEG {
7       struct node {
8           int ls, rs, v;
9           node() {}
10          void init() { ls = rs = v = 0; }
11      }t[N * 60];
12      int tot;
13      int newnode() { ++tot; t[tot].init(); return tot; }
14      void init() { tot = 0; }
15      void pushup(int id) { t[id].v = t[t[id].ls].v + t[t[id].rs].v; }
16      void update(int &id, int l, int r, int pos, int v) {
17          if (!id) id = newnode();
18          if (l == r) {
19              t[id].v += v;
20              return;
21          }
22          int mid = (l + r) >> 1;
23          if (pos <= mid) update(t[id].ls, l, mid, pos, v);
24          else update(t[id].rs, mid + 1, r, pos, v);
25          pushup(id);
26      }
```

```
27      void merge(int &x, int y, int l, int r) {
28          if (!x || !y) { x |= y; return; }
29          if (l == r) {
30              t[x].v += t[y].v;
31              return;
32          }
33          int mid = (l + r) >> 1;
34          merge(t[x].ls, t[y].ls, l, mid);
35          merge(t[x].rs, t[y].rs, mid + 1, r);
36          pushup(x);
37      }
38      //k表示保存在y中的区间长度
39      void split(int &x, int y, int k, int opt) {
40          if (t[y].v == k) return;
41          x = newnode();
42          t[x].v = t[y].v - k;
43          t[y].v = k;
44          if (opt) {
45              if (k <= t[t[y].rs].v) {
46                  split(t[x].rs, t[y].rs, k, opt);
47                  t[x].ls = t[y].ls;
48                  t[y].ls = 0;
49              } else {
50                  split(t[x].ls, t[y].ls, k - t[t[y].rs].v, opt);
51              }
52          } else {
53              if (k <= t[t[y].ls].v) {
54                  split(t[x].ls, t[y].ls, k, opt);
55                  t[x].rs = t[y].rs;
56                  t[y].rs = 0;
57              } else {
58                  split(t[x].rs, t[y].rs, k - t[t[y].ls].v, opt);
59              }
60          }
61      }
62      //查找区间第k大
63      int query(int id, int l, int r, int k, int opt) {
64          if (l == r) return l;
65          int mid = (l + r) >> 1;
66          if (opt) {
67              if (k <= t[t[id].rs].v) return query(t[id].rs, mid + 1, r, k,
                    opt);
68              else return query(t[id].ls, l, mid, k - t[t[id].rs].v, opt);
69          } else {
70              if (k <= t[t[id].ls].v) return query(t[id].ls, l, mid, k, opt);
71              else return query(t[id].rs, mid + 1, r, k - t[t[id].ls].v, opt);
72          }
73      }
74  }seg;
75
76  struct ODT {
77      //opt 0 升序 1 降序
78      struct node {
79          int l;
80          mutable int r, segrt, opt;
81          node() {}
82          node(int l, int r = -1, int segrt = 0, int opt = 0) : l(l), r(r),
                segrt(segrt), opt(opt) {}
```

```
83          bool operator < (const node &other) const { return l < other.l; }
84      };
85      using IT = set<node>::iterator;
86      set <node> se;
87      void init() { se.clear(); }
88      IT split(int pos) {
89          IT it = se.lower_bound(node(pos));
90          if (it != se.end() && it->l == pos)
91              return it;
92          --it;
93          int l = it->l, r = it->r, opt = it->opt, segrt = it->segrt, newsegrt
                ;
94          seg.split(newsegrt, it->segrt, pos - it->l, opt);
95          se.erase(it);
96          se.insert(node(l, pos - 1, segrt, opt));
97          return se.insert(node(pos, r, newsegrt, opt)).first;
98      }
99      void assign(int l, int r, int opt) {
100         IT itr = split(r + 1), itl = split(l), it = itl;
101         itl->opt = opt;
102         itl->r = r;
103         for (++it; it != itr; ++it) {
104             seg.merge(itl->segrt, it->segrt, 1, n);
105         }
106         it = itl; ++it;
107         se.erase(it, itr);
108     }
109     int query(int pos) {
110         IT it = se.upper_bound(node(pos)); --it;
111         return seg.query(it->segrt, 1, n, pos - it->l + 1, it->opt);
112     }
113 }odt;
114
115 int main() {
116     while (scanf("%d%d", &n, &q) != EOF) {
117         for (int i = 1; i <= n; ++i) {
118             scanf("%d", a + i);
119         }
120         seg.init();
121         odt.init();
122         for (int i = 1; i <= n; ++i) {
123             auto it = odt.se.insert(ODT::node(i, i, 0, 0)).first;
124             seg.update(it->segrt, 1, n, a[i], 1);
125         }
126         for (int i = 1, op, l, r; i <= q; ++i) {
127             scanf("%d%d%d", &op, &l, &r);
128             odt.assign(l, r, op);
129         }
130         int x; scanf("%d", &x);
131         printf("%d\n", odt.query(x));
132     }
133     return 0;
134 }
```

6.3 势能线段树

6.3.1 HDU 5306

HDU5306
题意:
三种操作:

- 将区间 $[l_i, r_i]$ 内的数 a_i 变成 $min(a_i, t)$
- 输出区间最大值
- 输出区间和

思路:
维护区间最大值和次大值, 以及最大值的个数:

- 如果更新值大于等于最大值那么不用往下更新
- 如果更新值小于最大值大于次大值，那么直接更改最大值即可
- 否则暴力往下更新

时间复杂度 $O(n \log n)$

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e6 + 10, INF = 0x3f3f3f3f;
int n, q, a[N];

struct SEG {
    //+ < 取Min, 询问Max  - > 取Max, 询问Min 修改build初始化叶子结点
    static const int INF = 0x3f3f3f3f;
    static bool cmp(int x, int y) { return x < y; }
    static int get(int x, int y) { return cmp(x, y) ? y : x; }
    struct TAG {
        int v[2], cnt;
        TAG() {}
        void init() { v[0] = v[1] = -INF, cnt = 0;  }
        TAG operator + (const TAG &other) const {
            TAG res; res.init();
            if (v[0] == other.v[0]) {
                res.v[0] = v[0];
                res.cnt = cnt + other.cnt;
                res.v[1] = get(v[1], other.v[1]);
            } else {
                if (cmp(v[0], other.v[0])) {
                    res = other;
                    res.v[1] = get(other.v[1], v[0]);
                } else {
                    res = *this;
                    res.v[1] = get(v[1], other.v[0]);
                }
            }
            return res;
        }
    };
    struct node {
        ll sum; int lazy;
        TAG tag;
        node() { tag.init(); sum = 0; lazy = INF; }
```

```
38          void up(int _lazy) {
39              if (cmp(_lazy, tag.v[0]) == 0) return;
40              sum += 1ll * tag.cnt * (_lazy - tag.v[0]);
41              tag.v[0] = _lazy;
42              lazy = _lazy;
43          }
44          node operator + (const node &other) const {
45              node res = node();
46              res.sum = sum + other.sum;
47              res.tag = tag + other.tag;
48              return res;
49          }
50      }t[N << 2], res;
51      void build(int id, int l, int r) {
52          t[id] = node();
53          if (l == r) {
54              t[id].tag.cnt = 1;
55              t[id].tag.v[0] = a[l];
56              t[id].sum = a[l];
57              return;
58          }
59          int mid = (l + r) >> 1;
60          build(id << 1, l, mid);
61          build(id << 1 | 1, mid + 1, r);
62          t[id] = t[id << 1] + t[id << 1 | 1];
63      }
64      void pushdown(int id) {
65          int &lazy = t[id].lazy;
66          if (lazy == INF) return;
67          t[id << 1].up(lazy);
68          t[id << 1 | 1].up(lazy);
69          lazy = INF;
70      }
71      void update(int id, int l, int r, int ql, int qr, int x) {
72          if (cmp(x, t[id].tag.v[0]) == 0) return;
73          if (l == r) {
74              x = cmp(x, t[id].tag.v[0]) ? x : t[id].tag.v[0];
75              t[id].sum = x;
76              t[id].tag.v[0] = x;
77              t[id].tag.v[1] = -INF;
78              return;
79          }
80          if (l >= ql && r <= qr && cmp(x, t[id].tag.v[0]) && cmp(t[id].tag.v
                [1], x)) {
81              t[id].up(x);
82              return;
83          }
84          int mid = (l + r) >> 1;
85          pushdown(id);
86          if (ql <= mid) update(id << 1, l, mid, ql, qr, x);
87          if (qr > mid) update(id << 1 | 1, mid + 1, r, ql, qr, x);
88          t[id] = t[id << 1] + t[id << 1 | 1];
89      }
90      void query(int id, int l, int r, int ql, int qr) {
91          if (l >= ql && r <= qr) {
92              res = res + t[id];
93              return;
94          }
```

```
95          int mid = (l + r) >> 1;
96          pushdown(id);
97          if (ql <= mid) query(id << 1, l, mid, ql, qr);
98          if (qr > mid) query(id << 1 | 1, mid + 1, r, ql, qr);
99      }
100 }seg;
101
102 int main() {
103     int T; scanf("%d", &T);
104     while (T--) {
105         scanf("%d%d", &n, &q);
106         for (int i = 1; i <= n; ++i) {
107             scanf("%d", a + i);
108         }
109         seg.build(1, 1, n);
110         int op, x, y, t;
111         while (q--) {
112             scanf("%d%d%d", &op, &x, &y);
113             if (!op) {
114                 scanf("%d", &t);
115                 seg.update(1, 1, n, x, y, t);
116             } else {
117                 seg.res = SEG::node();
118                 seg.query(1, 1, n, x, y);
119                 printf("%lld\n", (op == 1 ? seg.res.tag.v[0] : seg.res.sum))
                        ;
120             }
121         }
122     }
123     return 0;
124 }
```

6.4 李超线段树

定义：

支持动态维护一个平面直角坐标系，支持在中间插入一条线段或者直线，支持询问与 $x = k$ 这条直线相交的所有线段中，交点的 y 轴坐标的最值

维护什么?

维护这个区间内的所有直线中，从上往下能够看到的最长的那个线段，也就是没有被其他直线覆盖长度最长的段

怎么插入?

- 如果这个区间没有记录最长的线段，那么直接把这个区间记录的线段修改为这条线段，然后返回。
- 如果当前线段在这个区间内已经被这个区间内的最长线段为覆盖，那么直接返回。
- 反过来，如果完全覆盖了之前记录的线段，那么直接赋值、返回。
- 否则和已经记录的直线有交，判断哪根线段覆盖的区域较长，把这个区间记录的值给修改一下，然后把短的那一半丢下去递归。

复杂度?

每次递归下去直线的长度至少减少一半，复杂度为一个 log。

如果是插入线段，要先找到对应的区间插入，复杂度是两个 log。

6.4.1 插入直线

BZOJ 1568

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef double db;
const db eps = 1e-8;
int sgn(db x) { if (fabs(x) < eps) return 0; return x < 0 ? -1 : 1; }
const int N = 5e4 + 10;
#define N 50010
int n, q;

struct SEG {
    struct node {
        bool F; db k, b;
        node(bool F = 0, db k = 0, db b = 0) : F(F), k(k), b(b) {}
        db calc(db x) { return k * x + b; }
    }t[N << 2];
    void build(int id, int l, int r) {
        t[id] = node();
        if (l == r) return;
        int mid = (l + r) >> 1;
        build(id << 1, l, mid);
        build(id << 1 | 1, mid + 1, r);
    }
    db div(db a, db b) { return a / b; }
//  db div(db a, db b) { return a / b - ((a ^ b) < 0 && a % b); }
    void update(int id, int l, int r, node tmp) {
        if (t[id].F == 0) {
            t[id] = tmp;
            return;
        }
        int mid = (l + r) >> 1;
        db preL = t[id].calc(l), preR = t[id].calc(r);
        db newL = tmp.calc(l), newR = tmp.calc(r);
        //维护最大值优势线段
        if (sgn(preL - newL) >= 0 && sgn(preR - newR) >= 0) return;
        if (newL > preL && newR > preR) {
            t[id] = tmp;
            return;
        }
        //计算交点
        db x = div(tmp.b - t[id].b, t[id].k - tmp.k);
        if (newL > preL) {
            if (x > mid) {
                update(id << 1 | 1, mid + 1, r, t[id]);
                t[id] = tmp;
            } else {
                update(id << 1, l, mid, tmp);
            }
        } else {
            if (x > mid) {
                update(id << 1 | 1, mid + 1, r, tmp);
            } else {
                update(id << 1, l, mid, t[id]);
                t[id] = tmp;
            }
        }
    }
    void update(int id, int l, int r, db k, db b) { update(id, l, r, node(1,
        k, b)); }
```

```
    db query(int id, int l, int r, int x) {
        if (l == r) return t[id].calc(x);
        int mid = (l + r) >> 1;
        db res = t[id].calc(x);
        if (x <= mid) res = max(res, query(id << 1, l, mid, x));
        else res = max(res, query(id << 1 | 1, mid + 1, r, x));
        return res;
    }
}seg;

int main() {
    n = 50000;
    while (scanf("%d", &q) != EOF) {
        seg.build(1, 1, n);
        char op[20]; db k, b; int x;
        for (int _q = 1; _q <= q; ++_q) {
            scanf("%s", op + 1);
            switch(op[1]) {
                case 'P' :
                    scanf("%lf%lf", &b, &k);
                    seg.update(1, 1, n, k, b - k);
                    break;
                case 'Q' :
                    scanf("%d\n", &x);
                    printf("%lld\n", (long long)(seg.query(1, 1, n, x) /
                       100));
                    break;
            }
        }
    }
    return 0;
}
```

6.4.2 插入线段

BZOJ 3165 LOJ 6034
更新复杂度 $O(\log^2 n)$, 查询复杂度 $O(\log n)$

```
#include <bits/stdc++.h>
using namespace std;
typedef double db;
const int N = 1e5 + 10;
const db eps = 1e-8;
int sgn(db x) { if (fabs(x) < eps) return 0; return x < 0 ? -1 : 1; }
int q, ans, tot;
db K[N], B[N];
struct SEG {
    struct node {
        db k, b; int pos;
        bool F;
        node(db k = 0, db b = 0, int pos = 0, bool F = 0) : k(k), b(b), pos(
           pos), F(F) {}
        db calc(db x) { return k * x + b; }
    }t[N << 2];
    void build(int id, int l, int r) {
        t[id] = node();
        if (l == r) return;
        int mid = (l + r) >> 1;
        build(id << 1, l, mid);
```

```
21          build(id << 1 | 1, mid + 1, r);
22      }
23      db div(db a, db b) { return a / b; }
24  //  db div(db a, db b) { return a / b - ((a ^ b) < 0 && a % b); }
25      void update(int id, int l, int r, node tmp) {
26          if (t[id].F == 0) {
27              t[id] = tmp;
28              return;
29          }
30          int mid = (l + r) >> 1;
31          db preL = t[id].calc(l), preR = t[id].calc(r);
32          db newL = tmp.calc(l), newR = tmp.calc(r);
33          if (sgn(preL - newL) >= 0 && sgn(preR - newR) >= 0) return;
34          if (newL > preL && newR > preR) {
35              t[id] = tmp;
36              return;
37          }
38          db x = div(tmp.b - t[id].b, t[id].k - tmp.k);
39          if (newL > preL) {
40              if (x > mid) {
41                  update(id << 1 | 1, mid + 1, r, t[id]);
42                  t[id] = tmp;
43              } else {
44                  update(id << 1, l, mid, tmp);
45              }
46          } else {
47              if (x > mid) {
48                  update(id << 1 | 1, mid + 1, r, tmp);
49              } else {
50                  update(id << 1, l, mid, t[id]);
51                  t[id] = tmp;
52              }
53          }
54      }
55      void update(int id, int l, int r, int ql, int qr, node tmp) {
56          if (l >= ql && r <= qr) {
57              update(id, l, r, tmp);
58              return;
59          }
60          int mid = (l + r) >> 1;
61          if (ql <= mid) update(id << 1, l, mid, ql, qr, tmp);
62          if (qr > mid) update(id << 1 | 1, mid + 1, r, ql, qr, tmp);
63      }
64      void update(int id, int l, int r, int ql, int qr, db k, db b, int pos) {
            update(id, l, r, ql, qr, node(k, b, pos, 1)); }
65      void Cmax(int &a, int b, int x) {
66          db ya = K[a] * x + B[a];
67          db yb = K[b] * x + B[b];
68          if (ya < yb || (fabs(ya - yb) < eps && a > b)) a = b;
69      }
70      int query(int id, int l, int r, int x) {
71          if (l == r) return t[id].pos;
72          int mid = (l + r) >> 1;
73          int res = t[id].pos;
74          if (x <= mid) Cmax(res, query(id << 1, l, mid, x), x);
75          else Cmax(res, query(id << 1 | 1, mid + 1, r, x), x);
76          return res;
77      }
```

```
}seg;

int main() {
    while(scanf("%d", &q) != EOF) {
        int n = 100000;
        seg.build(1, 1, n);
        int op, x[2], y[2];
        int mod = 39989;
        ans = 0;
        for (int _q = 1; _q <= q; ++_q) {
            scanf("%d", &op);
            switch(op) {
                case 0 :
                    scanf("%d", x);
                    x[0] = (x[0] + ans - 1) % mod + 1;
                    printf("%d\n", ans = seg.query(1, 1, n, x[0]));
                    break;
                case 1 :
                    for (int i = 0; i < 2; ++i) {
                        scanf("%d%d", x + i, y + i);
                        x[i] = (x[i] + ans - 1) % mod + 1;
                        y[i] = (y[i] + ans - 1) % 1000000000 + 1;
                    }
                    if (x[0] > x[1]) {
                        swap(x[0], x[1]);
                        swap(y[0], y[1]);
                    }
                    ++tot;
                    K[tot] = 1.0 * (y[0] - y[1]) / (x[0] - x[1]);
                    B[tot] = y[0] - K[tot] * x[0];
                    seg.update(1, 1, n, x[0], x[1], K[tot], B[tot], tot);
                    break;
            }
        }
    }
    return 0;
}
```

6.5 历史最值线段树

BZOJ 3064
题意:

- *Q x y*, 查询 $[x,y]$ 之间的历史最大值
- *A x y*, 查询 $[x,y]$ 之间的最大值
- *P x y z*, 使得 $[x,y]$ 之间的 $a_i = a_i + z$
- *C x y z*, 使得 $[x,y]$ 之间的 $a_i = z$

思路:
定义标记 (a,b) 为 $A_i = max(A_i + a, b)$
区间加操作等价于标记 $(a,-\infty)$, 区间赋值操作等价于 $(-\infty,a)$
两个标记合并: $(a,b)+(c,d) = (a+c, max(b+c,d))$
两个标记取最大值: $(a,b)*(c,d) = (max(a,c), max(b,d))$
维护当前最大值，当前标记，历史最大值，历史标记最大值即可

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5 + 10;
int n, q, a[N];

struct SEG {
    const static ll INF = 0x3f3f3f3f3f3f3f3f;
    struct TAG {
        ll a, b;
        TAG() {}
        TAG(ll a, ll b) : a(a), b(b) {}
        void init() { a = 0, b = -INF; }
        TAG operator + (const TAG &other) { return {max(-INF, a + other.a),
           max(b + other.a, other.b)}; }
        TAG operator * (const TAG &other) { return {max(a, other.a), max(b,
           other.b)}; }
        ll calc(ll x) { return max(x + a, b); }
    };
    struct node {
        ll v, hv;
        TAG lazy, hlazy;
        node() {}
        void init() { v = hv = -INF; lazy.init(); hlazy.init(); }
        void up(TAG _lazy, TAG _hlazy) {
            hlazy = hlazy * (lazy + _hlazy);
            lazy = lazy + _lazy;
            hv = max(hv, _hlazy.calc(v));
            v = _lazy.calc(v);
        }
        node operator + (const node &other) const {
            node res; res.init();
            res.v = max(v, other.v);
            res.hv = max(hv, other.hv);
            return res;
        }
    }t[N << 2], res;
    void pushdown(int id) {
        TAG &lazy = t[id].lazy, &hlazy = t[id].hlazy;
        t[id << 1].up(lazy, hlazy);
        t[id << 1 | 1].up(lazy, hlazy);
        lazy.init(); hlazy.init();
    }
    void build(int id, int l, int r) {
        t[id].init();
        if (l == r) {
            t[id].v = t[id].hv = a[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(id << 1, l, mid);
        build(id << 1 | 1, mid + 1, r);
        t[id] = t[id << 1] + t[id << 1 | 1];
    }
    void update(int id,int l, int r, int ql, int qr, TAG v) {
        if (l >= ql && r <= qr) {
            t[id].up(v, v);
            return;
```

```
57          }
58          int mid = (l + r) >> 1;
59          pushdown(id);
60          if (ql <= mid) update(id << 1, l, mid, ql, qr, v);
61          if (qr > mid) update(id << 1 | 1, mid + 1, r, ql, qr, v);
62          t[id] = t[id << 1] + t[id << 1 | 1];
63      }
64      void query(int id, int l, int r, int ql, int qr) {
65          if (l >= ql && r <= qr) {
66              res = res + t[id];
67              return;
68          }
69          int mid = (l + r) >> 1;
70          pushdown(id);
71          if (ql <= mid) query(id << 1, l, mid, ql, qr);
72          if (qr > mid) query(id << 1 | 1, mid + 1, r, ql, qr);
73      }
74  }seg;
75
76  int main() {
77      while (scanf("%d", &n) != EOF) {
78          for (int i = 1; i <= n; ++i) scanf("%d", a + i);
79          seg.build(1, 1, n);
80          scanf("%d", &q);
81          char op[10];
82          for (int _q = 1, x, y, z; _q <= q; ++_q) {
83              scanf("%s%d%d", op, &x, &y);
84              if (op[0] == 'Q') {
85                  seg.res.init();
86                  seg.query(1, 1, n, x, y);
87                  printf("%lld\n", seg.res.v);
88              } else if (op[0] == 'A') {
89                  seg.res.init();
90                  seg.query(1, 1, n, x, y);
91                  printf("%lld\n", seg.res.hv);
92              } else if (op[0] == 'P') {
93                  scanf("%d", &z);
94                  seg.update(1, 1, n, x, y, SEG::TAG(z, -SEG::INF));
95              } else if (op[0] == 'C') {
96                  scanf("%d", &z);
97                  seg.update(1, 1, n, x, y, SEG::TAG(-SEG::INF, z));
98              }
99          }
100     }
101     return 0;
102 }
```

6.6 线段树维护立方和

HDU 4578
题意:

- 区间加
- 区间乘
- 区间赋值
- 给出 $l, r, p(1 \le p \le 3)$, 询问 $\sum_{i=l}^{r} a_i^p$

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 10, mod = 1e4 + 7;
int n, q;
struct SEG {
    struct node {
        int l, r;
        int sum[3], lazy[3];
        node() {}
        node(int _l, int _r) {
            l = _l, r = _r;
            memset(sum, 0, sizeof sum);
            lazy[0] = 0, lazy[1] = 1, lazy[2] = -1;
        }
    }t[N << 2];
    void pushup(int id) {
        for (int i = 0; i < 3; ++i) {
            t[id].sum[i] = (t[id << 1].sum[i] + t[id << 1 | 1].sum[i]) % mod
                ;
        }
    }
    //add
    void work1(node &r, int b) {
        int len = r.r - r.l + 1;
        r.sum[2] = (r.sum[2] + (len * b % mod * b % mod * b % mod) + (3 * b
            % mod * b % mod * r.sum[0] % mod) + (3 * b % mod * r.sum[1] % mod
            )) % mod;
        r.sum[1] = (r.sum[1] + (2 * b % mod * r.sum[0] % mod) + (len * b %
            mod * b % mod)) % mod;
        r.sum[0] = (r.sum[0] + len * b % mod) % mod;
        r.lazy[0] = (r.lazy[0] + b) % mod;
    }
    //mul
    void work2(node &r, int b) {
        r.sum[0] = (r.sum[0] * b) % mod;
        r.sum[1] = (r.sum[1] * b % mod * b) % mod;
        r.sum[2] = (r.sum[2] * b % mod * b % mod * b) % mod;
        r.lazy[0] = r.lazy[0] * b % mod;
        r.lazy[1] = r.lazy[1] * b % mod;
    }
    //change
    void work3(node &r, int b) {
        int len = r.r - r.l + 1;
        r.sum[0] = len * b % mod;
        r.sum[1] = r.sum[0] * b % mod;
        r.sum[2] = r.sum[1] * b % mod;
        r.lazy[0] = 0; r.lazy[1] = 1;
        r.lazy[2] = b;
    }
    void pushdown(int id) {
        if (t[id].l >= t[id].r) return;
        if (~t[id].lazy[2]) {
            work3(t[id << 1], t[id].lazy[2]);
            work3(t[id << 1 | 1], t[id].lazy[2]);
            t[id].lazy[2] = -1;
        }
        if (t[id].lazy[1] != 1) {
            work2(t[id << 1], t[id].lazy[1]);
```

```
55              work2(t[id << 1 | 1], t[id].lazy[1]);
56              t[id].lazy[1] = 1;
57          }
58          if (t[id].lazy[0]) {
59              work1(t[id << 1], t[id].lazy[0]);
60              work1(t[id << 1 | 1], t[id].lazy[0]);
61              t[id].lazy[0] = 0;
62          }
63      }
64      void build(int id, int l, int r) {
65          t[id] = node(l, r);
66          if (l == r) return;
67          int mid = (l + r) >> 1;
68          build(id << 1, l, mid);
69          build(id << 1 | 1, mid + 1, r);
70      }
71      void update(int id, int l, int r, int vis, int val) {
72          if (t[id].l >= l && t[id].r <= r) {
73              if (vis == 1) work1(t[id], val);
74              else if (vis == 2) work2(t[id], val);
75              else work3(t[id], val);
76              return;
77          }
78          pushdown(id);
79          int mid = (t[id].l + t[id].r) >> 1;
80          if (l <= mid) update(id << 1, l, r, vis, val);
81          if (r > mid) update(id << 1 | 1, l, r, vis, val);
82          pushup(id);
83      }
84      int query(int id, int l, int r, int p) {
85          if (t[id].l >= l && t[id].r <= r) return t[id].sum[p];
86          int res = 0;
87          pushdown(id);
88          int mid = (t[id].l + t[id].r) >> 1;
89          if (l <= mid) res = (res + query(id << 1, l, r, p)) % mod;
90          if (r > mid) res = (res + query(id << 1 | 1, l, r, p)) % mod;
91          return res;
92      }
93  }seg;
94
95  int main() {
96      while (scanf("%d%d", &n, &q), n || q) {
97          seg.build(1, 1, n);
98          for (int i = 1, op, x, y, c; i <= q; ++i) {
99              scanf("%d%d%d%d", &op, &x, &y, &c);
100             if (op <= 3) seg.update(1, x, y, op, c);
101             else printf("%d\n", seg.query(1, x, y, c - 1));
102         }
103     }
104     return 0;
105 }
```

6.7 可持久化区间线段树

SPOJ-TTM

给出一个序列 a_i，维护四种操作：

- $C\ l\ r\ d$, 将 $a_i, i \in [l,r]$ 加上 d，并且使当前时间戳 $+1$

- $Q\ l\ r$, 查询当前版本 $\sum_{i=l}^{r} a_i$
- $H\ l\ r\ t$, 查询时间戳为 t 的版本的 $\sum_{i=l}^{r} a_i$
- $B\ t$，回到时间戳 t

可持久化区间线段树不能用 *pushdown* 和 *pushup*，因为 *pushdown* 的时候那些儿子是旧版本中的儿子，*down* 下去后应该变成新版本的，但是各种版本的 *lazy* 标记打在一个点上的时候就没法处理了。

使用标记永久化，对于当前点以及其祖先的贡献直接在 *update* 过程中更新好，对于它孩子的贡献，直接在那个点处打个标记，查询的时候加上即可。

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 1e5 + 10;
int n, q, a[N], rt[N];

struct SEG {
    struct node {
        int ls, rs;
        ll lazy, sum;
        node() { ls = rs = lazy = sum = 0; }
    }t[N * 40];
    int tot;
    int newnode() { ++tot; t[tot] = node();  return tot; }
    void init() { tot = 0; }
    void build(int &id, int l, int r) {
        if (!id) id = newnode();
        if (l == r) {
            t[id].sum = a[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(t[id].ls, l, mid);
        build(t[id].rs, mid + 1, r);
        t[id].sum = t[t[id].ls].sum + t[t[id].rs].sum;
    }
    void update(int &now, int pre, int l, int r, int ql, int qr, ll v) {
        now = newnode();
        t[now] = t[pre];
        t[now].sum += v * (min(r, qr) - max(l, ql) + 1);
        if (l >= ql && r <= qr) {
            t[now].lazy += v;
            return;
        }
        int mid = (l + r) >> 1;
        if (ql <= mid) update(t[now].ls, t[pre].ls, l, mid, ql, qr, v);
        if (qr > mid) update(t[now].rs, t[pre].rs, mid + 1, r, ql, qr, v);
    }
    ll query(int id, int l, int r, int ql, int qr) {
        if (ql > qr) return 0;
        if (l >= ql && r <= qr) return t[id].sum;
        int mid = (l + r) >> 1;
        ll res = t[id].lazy * (min(r, qr) - max(l, ql) + 1);
        if (ql <= mid) res += query(t[id].ls, l, mid, ql, qr);
        if (qr > mid) res += query(t[id].rs, mid + 1, r, ql, qr);
        return res;
    }
```

```
}seg;

int main() {
    while (scanf("%d%d", &n, &q) != EOF) {
        for (int i = 1; i <= n; ++i) scanf("%d", a + i);
        seg.init();
        seg.build(rt[0], 1, n);
        int cntrt = 0;
        char op; int l, r, t; ll d;
        for (int i = 1; i <= q; ++i) {
            scanf(" %c", &op);
            if (op == 'C') {
                scanf("%d%d%lld", &l, &r, &d);
                ++cntrt;
                rt[cntrt] = rt[cntrt - 1];
                seg.update(rt[cntrt], rt[cntrt], 1, n, l, r, d);
            } else if (op == 'Q') {
                scanf("%d%d", &l, &r);
                printf("%lld\n", seg.query(rt[cntrt], 1, n, l, r));
            } else if (op == 'H') {
                scanf("%d%d%d", &l, &r, &t);
                printf("%lld\n", seg.query(rt[t], 1, n, l, r));
            } else {
                scanf("%d", &t);
                cntrt = t;
            }
        }
    }
    return 0;
}
```

6.8 区间线段树套区间线段树

Luogu P3437

题意:

查询矩形区间最大值，更新矩形区间最大值

思路:

区间线段树套区间线段树，建树复杂度 $O(n^2)$，直接暴力建树即可，如果有初始值就暴力遍历取 *max* 即可，更新和查询复杂度 $O(\log^2 n)$。

但是考虑到 *pushup* 的复杂度是不对的，所以不要 *pushup*，直接用标记永久化。

内存不够可以尝试开两倍或者三倍内存，不一定要开四倍，因为单棵线段树长度比较小。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e3 + 5;
int n, m, q;

struct ODTree {
    struct node {
        int Max, lazy;
        node() {};
        void init() { Max = lazy = 0; }
    }t[N * 3];
    void build(int id, int l, int r, int Tl, int Tr) {
        t[id].init();
        if (l == r) return;
        int mid = (l + r) >> 1;
        build(id << 1, l, mid, Tl, Tr);
```

```
17          build(id << 1 | 1, mid + 1, r, Tl, Tr);
18      }
19      void update(int id, int l, int r, int ql, int qr, int v) {
20          if (min(r, qr) - max(l, ql) + 1 > 0) {
21              t[id].Max = max(t[id].Max, v);
22          }
23          if (l >= ql && r <= qr) {
24              t[id].lazy = max(t[id].lazy, v);
25              return;
26          }
27          int mid = (l + r) >> 1;
28          if (ql <= mid) update(id << 1, l, mid, ql, qr, v);
29          if (qr > mid) update(id << 1 | 1, mid + 1, r, ql, qr, v);
30      }
31      int query(int id, int l, int r, int ql, int qr) {
32          if (l >= ql && r <= qr) return t[id].Max;
33          int mid = (l + r) >> 1;
34          int res = t[id].lazy;
35          if (ql <= mid) res = max(res, query(id << 1, l, mid, ql, qr));
36          if (qr > mid) res = max(res, query(id << 1 | 1, mid + 1, r, ql, qr))
                ;
37          return res;
38      }
39  };
40
41  struct TDTree {
42      struct node {
43          ODTree Max, lazy;
44          node() {}
45      }t[N * 3];
46      void build(int id, int l, int r) {
47          t[id].Max.build(1, 1, m, l, r);
48          t[id].lazy.build(1, 1, m, l, r);
49          if (l == r) return;
50          int mid = (l + r) >> 1;
51          build(id << 1, l, mid);
52          build(id << 1 | 1, mid + 1, r);
53      }
54      void update(int id, int l, int r, int ql, int qr, int Ol, int Or, int v)
            {
55          if (min(r, qr) - max(l, ql) + 1 > 0) {
56              t[id].Max.update(1, 1, m, Ol, Or, v);
57          }
58          if (l >= ql && r <= qr) {
59              t[id].lazy.update(1, 1, m, Ol, Or, v);
60              return;
61          }
62          int mid = (l + r) >> 1;
63          if (ql <= mid) update(id << 1, l, mid, ql, qr, Ol, Or, v);
64          if (qr > mid) update(id << 1 | 1, mid + 1, r, ql, qr, Ol, Or, v);
65      }
66      int query(int id, int l, int r, int ql, int qr, int Ol, int Or) {
67          if (l >= ql && r <= qr) return t[id].Max.query(1, 1, m, Ol, Or);
68          int mid = (l + r) >> 1;
69          int res = t[id].lazy.query(1, 1, m, Ol, Or);
70          if (ql <= mid) res = max(res, query(id << 1, l, mid, ql, qr, Ol, Or)
                );
71          if (qr > mid) res = max(res, query(id << 1 | 1, mid + 1, r, ql, qr,
```

```
            Ol, Or));
72          return res;
73      }
74  }td;
75
76  int main() {
77      while (scanf("%d%d%d", &n, &m, &q) != EOF) {
78          ++n, ++m;
79          td.build(1, 1, n);
80          for (int i = 1, d, s, w, x, y; i <= q; ++i) {
81              scanf("%d%d%d%d%d", &d, &s, &w, &x, &y);
82              td.update(1, 1, n, x + 1, x + d, y + 1, y + s, td.query(1, 1, n,
                    x + 1, x + d, y + 1, y + s) + w);
83          }
84          printf("%d\n", td.query(1, 1, n, 1, n, 1, m));
85      }
86      return 0;
87  }
```

6.9 区间线段树套权值线段树

牛客 3979I

题意:

一个序列，维护两种操作:

- 1 l r x, 对 $i \in [l,r]$，令 $A_i = min(A_i, x)$
- 2 l r k，询问 $[l,r]$ 中第 k 小的数字

思路:

考虑区间线段树上的每一个节点 $[l,r]$，可以用一个动态开点权值线段树记录 $[l,r]$ 中每种权值出现了多少次。

询问可以转化成在 $O(logn)$ 棵线段树上二分，询问复杂度 $o(log^2n)$。

修改的时候，定位到一些节点 $[l_i,r_i]$，将所有 $> x$ 的数都并到 x 处，主要要将 $[l_i,r_i]$ 及其祖先的内层线段树都要操作，时间复杂度 $O(tlog^2n)$，t 为合并的节点数。

然后打标记下传即可，注意 *pushdown* 的时候不需要更新祖先的内层线段树，因为已经更新过了。

每个节点只会被合并一次，一共有 $O(nlogn)$ 个节点，所以总的时间复杂度 $O(nlog^3n)$

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   const int N = 8e4 + 10;
4   int n, q, a[N];
5
6   struct VSEG {
7       struct node {
8           int sum, Max, ls, rs;
9           node() {}
10          void init() {
11              sum = Max = 0;
12              ls = rs = 0;
13          }
14      }t[N * 400];
15      int cnt = 0;
16      void init() { cnt = 0; }
17      int newnode() {
18          ++cnt;
19          t[cnt].init();
20          return cnt;
21      }
22      void up(int id) {
```

```
23          int ls = t[id].ls, rs = t[id].rs;
24          t[id].sum = t[ls].sum + t[rs].sum;
25          t[id].Max = max(t[ls].Max, t[rs].Max);
26      }
27      void update(int id, int l, int r, int pos, int v) {
28          if (l == r) {
29              t[id].sum += v;
30              if (t[id].sum) t[id].Max = pos;
31              else t[id].Max = 0;
32              return;
33          }
34          int mid = (l + r) >> 1;
35          if (pos <= mid) {
36              if (!t[id].ls) t[id].ls = newnode();
37              update(t[id].ls, l, mid, pos, v);
38          } else {
39              if (!t[id].rs) t[id].rs = newnode();
40              update(t[id].rs, mid + 1, r, pos, v);
41          }
42          up(id);
43      }
44      int query(int id, int l, int r, int pos) {
45          if (!id) return 0;
46          if (l == r) return t[id].sum;
47          int mid = (l + r) >> 1;
48          if (pos <= mid) return query(t[id].ls, l, mid, pos);
49          else return query(t[id].rs, mid + 1, r, pos);
50      }
51  }vseg;
52
53  struct SEG {
54      struct node {
55          int id, lazy, l, r;
56          node() {}
57          void init() {
58              id = l = r = 0;
59          }
60      }t[N << 2];
61      vector <int> vec;
62      void build(int id, int l, int r) {
63          t[id].l = l; t[id].r = r;
64          t[id].id = vseg.newnode();
65          for (int i = l; i <= r; ++i) {
66              vseg.update(t[id].id, 1, n, a[i], 1);
67          }
68          t[id].lazy = n;
69          if (l == r) {
70              t[id].lazy = a[l];
71              return;
72          }
73          int mid = (l + r) >> 1;
74          build(id << 1, l, mid);
75          build(id << 1 | 1, mid + 1, r);
76      }
77      void down(int id) {
78          int &lazy = t[id].lazy;
79          t[id << 1].lazy = min(lazy, t[id << 1].lazy);
80          t[id << 1 | 1].lazy = min(lazy, t[id << 1 | 1].lazy);
```

```
81          for (auto &it : {t[id << 1].id, t[id << 1 | 1].id}) {
82              while (vseg.t[it].Max > lazy) {
83                  int pos = vseg.t[it].Max;
84                  int del = vseg.query(it, 1, n, pos);
85                  vseg.update(it, 1, n, pos, -del);
86                  vseg.update(it, 1, n, lazy, del);
87              }
88          }
89      }
90      void update(int id, int l, int r, int ql, int qr, int x) {
91          if (l >= ql && r <= qr) {
92              vector <int> vec;
93              int it = id;
94              while (it) {
95                  vec.push_back(t[it].id);
96                  it >>= 1;
97              }
98              while (vseg.t[t[id].id].Max > x) {
99                  int pos = vseg.t[t[id].id].Max;
100                 int del = vseg.query(t[id].id, 1, n, pos);
101                 for (auto &it : vec) {
102                     vseg.update(it, 1, n, pos, -del);
103                     vseg.update(it, 1, n, x, del);
104                 }
105             }
106             t[id].lazy = min(t[id].lazy, x);
107             return;
108         }
109         int mid = (l + r) >> 1;
110         down(id);
111         if (ql <= mid) update(id << 1, l, mid, ql, qr, x);
112         if (qr > mid) update(id << 1 | 1, mid + 1, r, ql, qr, x);
113     }
114     void add(int id, int l, int r, int ql, int qr) {
115         if (l >= ql && r <= qr) {
116             vec.push_back(t[id].id);
117             return;
118         }
119         int mid = (l + r) >> 1;
120         down(id);
121         if (ql <= mid) add(id << 1, l, mid, ql, qr);
122         if (qr > mid) add(id << 1 | 1, mid + 1, r, ql, qr);
123     }
124     int querygao(int l, int r, int k) {
125         if (l == r) return l;
126         int mid = (l + r) >> 1;
127         int lsum = 0;
128         for (auto &it : vec) lsum += vseg.t[vseg.t[it].ls].sum;
129         if (lsum >= k) {
130             for (auto &it : vec) it = vseg.t[it].ls;
131             return querygao(l, mid, k);
132         } else {
133             for (auto &it : vec) it = vseg.t[it].rs;
134             return querygao(mid + 1, r, k - lsum);
135         }
136     }
137     int query(int l, int r, int k) {
138         vec.clear();
```

```
        add(1, 1, n, l, r);
        return querygao(1, n, k);
    }
}seg;

int main() {
    while (scanf("%d%d", &n, &q) != EOF) {
        for (int i = 1; i <= n; ++i) scanf("%d", a + i);
        vseg.init();
        seg.build(1, 1, n);
        for (int _q = 1, tp, l, r, x; _q <= q; ++_q) {
            scanf("%d%d%d%d", &tp, &l, &r, &x);
            if (tp == 1) {
                if (x >= n) continue;
                seg.update(1, 1, n, l, r, x);
            } else {
                printf("%d\n", seg.query(l, r, x));
            }
        }
    }
    return 0;
}
```

7 KD-Tree

7.1 Luogu P1429

平面最近点对。
建树时的两个优化：

- 选择的维度要满足其内部点的分布的差异度最大，即每次选择的切割维度是方差最大的维度。
- 每次在维度上选择切割点时选择该维度上的中位数，这样可以保证每次分成的左右子树大小尽量相等。

使用优化 2 能够保证树高最多为 $O(\log n)$。
查询的时候要用全局最优化剪枝

```
#include <bits/stdc++.h>
using namespace std;
typedef double db;
const int N = 2e5 + 10;
const int K = 2;
const db INF = 0x3f3f3f3f3f3f3f3f;
int n; db ans;
inline db sqr(db x) { return x * x; }

struct Point {
    db x[K]; int id;
    db dis(const Point &b) const {
        db res = 0;
        for (int i = 0; i < K; ++i)
            res += sqr(x[i] - b.x[i]);
        return res;
    }
}s[N];

struct KDTree {
    struct cmpx {
        int div;
```

```
23          cmpx(int div = 0) : div(div) {}
24          bool operator () (const Point &a, const Point &b) {
25              for (int i = 0; i < K; ++i)
26                  if (a.x[(i + div) % K] != b.x[(i + div) % K])
27                      return a.x[(i + div) % K] < b.x[(i + div) % K];
28              return true;
29          }
30      };
31      bool cmp(const Point &a, const Point &b, int div) { return cmpx(div)(a,
            b); }
32      struct Node {
33          Point e;
34          db Min[K], Max[K];
35          Node *ls, *rs;
36          int div;
37      }pool[N], *tail, *root;
38      void init() { tail = pool; }
39      void pushup(Node *a) {
40          for (int i = 0; i < K; ++i) {
41              a->Min[i] = a->Max[i] = a->e.x[i];
42          }
43          if (a->ls) {
44              for (int i = 0; i < K; ++i) {
45                  a->Min[i] = min(a->Min[i], a->ls->Min[i]);
46                  a->Max[i] = max(a->Max[i], a->ls->Max[i]);
47              }
48          }
49          if (a->rs) {
50              for (int i = 0; i < K; ++i) {
51                  a->Min[i] = min(a->Min[i], a->rs->Min[i]);
52                  a->Max[i] = max(a->Max[i], a->rs->Max[i]);
53              }
54          }
55      }
56      Node* build(int l, int r) {
57          if (l > r) return nullptr;
58          int mid = (l + r) >> 1;
59          Node *p = tail++;
60          p->div = 0;
61          if (l == r) {
62              p->e = s[mid];
63              pushup(p);
64              return p;
65          }
66          //average variance
67          double avx[K] = {0}, vax[K] = {0};
68          for (int i = l; i <= r; ++i)
69              for (int j = 0; j < K; ++j)
70                  avx[j] += s[i].x[j];
71          for (int i = 0; i < K; ++i) avx[i] /= (double)(r - l + 1);
72          for (int i = l; i <= r; ++i)
73              for (int j = 0; j < K; ++j)
74                  vax[j] += sqr(s[i].x[j] - avx[j]);
75          for (int i = 1; i < K; ++i) {
76              if (vax[i] > vax[p->div])
77                  p->div = i;
78          }
79          nth_element(s + l, s + mid, s + r + 1, cmpx(p->div));
```

```
        p->e = s[mid];
        p->ls = build(l, mid - 1);
        p->rs = build(mid + 1, r);
        pushup(p);
        return p;
    }
    db calc(Node *a, Point &e) {
        if (a == nullptr) return INF;
        db res = 0;
        for (int i = 0; i < K; ++i) {
            if (a->Min[i] > e.x[i])
                res += sqr(a->Min[i] - e.x[i]);
            if (a->Max[i] < e.x[i])
                res += sqr(a->Max[i] - e.x[i]);
        }
        return res;
    }
    void query(Node *t, int l, int r, Point &e) {
        if (l > r) return;
        int mid = (l + r) >> 1;
        if (t->e.id != e.id) ans = min(ans, e.dis(t->e));
        if (l == r) return;
        db distl = calc(t->ls, e), distr = calc(t->rs, e);
        if (distl < ans && distr < ans) {
            if (distl < distr) {
                query(t->ls, l, mid - 1, e);
                if (distr < ans) query(t->rs, mid + 1, r, e);
            } else {
                query(t->rs, mid + 1, r, e);
                if (distl < ans) query(t->ls, l, mid - 1, e);
            }
        } else {
            if (distl < ans) query(t->ls, l, mid - 1, e);
            if (distr < ans) query(t->rs, mid + 1, r, e);
        }
    }
}kdTree;

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%lf%lf", &s[i].x[0], &s[i].x[1]);
        s[i].id = i;
    }
    kdTree.init();
    kdTree.root = kdTree.build(1, n);
    ans = 2e18;
    for (int i = 1; i <= n; i++) kdTree.query(kdTree.root, 1, n, s[i]);
    printf("%.4f\n", sqrt(ans));
    return 0;
}
```

7.2 BZOJ 4520

给出 n 个点，找欧式距离下的第 k 远点对。
点对为无序点对。
复杂度跟 k 有关，k 不能太大。

```
#include <bits/stdc++.h>
#define SZ(x) (int(x.size()))
using namespace std;
typedef long long db;
const int N = 1e5 + 10;
const int DIM = 2;
const int K = 2;
const db INF = 0x3f3f3f3f3f3f3f3f;
int n, k;
inline db sqr(db x) { return x * x; }

struct Point {
    db x[DIM]; int id;
    void scan() { for (int i = 0; i < K; ++i) scanf("%lld", x + i); }
    void print() { for (int i = 0; i < K; ++i) printf("%lld%c", x[i], " \n"[
       i == K - 1]); }
    db dis(const Point &b) const {
        db res = 0;
        for (int i = 0; i < K; ++i)
            res += sqr(x[i] - b.x[i]);
        return res;
    }
}s[N];

struct KDTree {
    struct cmpx {
        int div;
        cmpx(int div = 0) : div(div) {}
        bool operator () (const Point &a, const Point &b) {
            for (int i = 0; i < K; ++i)
                if (a.x[(i + div) % K] != b.x[(i + div) % K])
                    return a.x[(i + div) % K] < b.x[(i + div) % K];
            return true;
        }
    };
    bool cmp(const Point &a, const Point &b, int div) { return cmpx(div)(a,
       b); }
    struct Node {
        Point e;
        db Min[DIM], Max[DIM];
        Node *ls, *rs;
        int div;
        void init() { ls = rs = NULL; div = 0;}
    }pool[N], *tail, *root;
    void init() { tail = pool; }
    Node* newnode() { tail->init(); return tail++; }
    void pushup(Node *a) {
        for (int i = 0; i < K; ++i) {
            a->Min[i] = a->Max[i] = a->e.x[i];
        }
        if (a->ls) {
            for (int i = 0; i < K; ++i) {
                a->Min[i] = min(a->Min[i], a->ls->Min[i]);
                a->Max[i] = max(a->Max[i], a->ls->Max[i]);
            }
        }
        if (a->rs) {
            for (int i = 0; i < K; ++i) {
```

```
57                  a->Min[i] = min(a->Min[i], a->rs->Min[i]);
58                  a->Max[i] = max(a->Max[i], a->rs->Max[i]);
59              }
60          }
61      }
62      Node* build(int l, int r, int div) {
63          if (l > r) return NULL;
64          int mid = (l + r) >> 1;
65          Node *p = newnode();
66          p->div = div;
67          nth_element(s + l, s + mid, s + r + 1, cmpx(div));
68          p->e = s[mid];
69          p->ls = build(l, mid - 1, (div + 1) % K);
70          p->rs = build(mid + 1, r, (div + 1) % K);
71          pushup(p);
72          return p;
73      }
74      struct qnode {
75          Point p;
76          db dis;
77          qnode() {}
78          qnode(Point p, db dis) : p(p), dis(dis) {}
79          bool operator < (const qnode &other) const { return dis > other.dis;
                }
80      };
81      priority_queue <qnode> pq;
82      db calc(Node *t, Point e) {
83          if (t == NULL) return 0;
84          db res = 0;
85          for (int i = 0; i < K; ++i) {
86              res += max(sqr(e.x[i] - t->Max[i]), sqr(e.x[i] - t->Min[i]));
87          }
88          return res;
89      }
90      void search(Node *t, int k, Point p) {
91          if (!t) return;
92          if (!cmp(p, t->e, t->div)) {
93              search(t->ls, k, p);
94              if (SZ(pq) < k) {
95                  if (p.id > t->e.id) pq.push(qnode(t->e, p.dis(t->e)));
96                  search(t->rs, k, p);
97              } else {
98                  if (p.id > t->e.id && p.dis(t->e) > pq.top().dis) {
99                      pq.pop();
100                     pq.push(qnode(t->e, p.dis(t->e)));
101                 }
102                 if (calc(t->rs, p) > pq.top().dis)
103                     search(t->rs, k, p);
104             }
105         } else {
106             search(t->rs, k, p);
107             if (SZ(pq) < k) {
108                 if (p.id > t->e.id) pq.push(qnode(t->e, p.dis(t->e)));
109                 search(t->ls, k, p);
110             } else {
111                 if (p.id > t->e.id && p.dis(t->e) > pq.top().dis) {
112                     pq.pop();
113                     pq.push(qnode(t->e, p.dis(t->e)));
```

```
                }
                if (calc(t->ls, p) > pq.top().dis)
                    search(t->ls, k, p);
            }
        }
    }
}kdTree;

int main() {
    while (scanf("%d%d", &n, &k) != EOF) {
        for (int i = 1; i <= n; ++i) {
            s[i].scan();
            s[i].id = i;
        }
        kdTree.init();
        kdTree.root = kdTree.build(1, n, 0);
        for (int i = 1; i <= n; ++i) kdTree.search(kdTree.root, k, s[i]);
        printf("%lld\n", kdTree.pq.top().dis);
    }
    return 0;
}
```

7.3 Luogu P4148

单点加，矩形求和，强制在线，卡空间, 坐标范围 $5 \cdot 10^5$, 操作次数 $2 \cdot 10^5$
查询流程:

- 在查询矩形区域内的所有点的权值和时，仍然需要记录子树内每一维度上的坐标的最大值和最小值。如果当前子树对应的矩形与所求矩形没有交点，则不继续搜索其子树;
- 如果当前子树对应的矩形完全包含在所求矩形内，返回当前子树内所有点的权值和;
- 否则，判断当前点是否在所求矩形内，更新答案并递归在左右子树中查找答案。

插入操作直接暴力插入，用替罪羊思想重构即可。
如果有删除操作，直接打标记，有必要的话可以进行删除重构。
在 $2D$ 树上进行矩阵查询, 已经被完全覆盖或者完全不相交的子树不会继续查询。
则单次查询时间复杂度最优是 $O(\log n)$, 最坏是 $O(\sqrt{n})$, 在 k 维的情况下, 最坏时间复杂度为 $O(n^{1-\frac{1}{k}})$ 的

```
#include <bits/stdc++.h>
using namespace std;
typedef int db;
const int N = 2e5 + 10;
const int DIM = 2;
const int K = 2;

struct Point {
    db x[DIM]; int v;
    bool In(db *Min, db *Max) {
        for (int i = 0; i < K; ++i) {
            if (x[i] < Min[i] || x[i] > Max[i])
                return 0;
        }
        return 1;
    }
}s[N]; int cntS;

struct KDTree {
    static constexpr double alpha = 0.75;
```

```
21      struct cmpx {
22          int div;
23          cmpx(int div = 0) : div(div) {}
24          bool operator () (const int &a, const int &b) {
25              return s[a].x[div] < s[b].x[div];
26          }
27      };
28      bool cmp(const int &a, const int &b, const int &div) { return s[a].x[div
            ] <= s[b].x[div]; }
29      struct Node {
30          int ls, rs, div;
31          db Min[DIM], Max[DIM];
32          int sze, sum;
33          bool In(db *_Min, db *_Max) {
34              for (int i = 0; i < K; ++i) {
35                  if (_Min[i] > Min[i] || _Max[i] < Max[i])
36                      return 0;
37              }
38              return 1;
39          }
40          bool Out(db *_Min, db *_Max) {
41              for (int i = 0; i < K; ++i) {
42                  if (_Max[i] < Min[i] || _Min[i] > Max[i])
43                      return 1;
44              }
45              return 0;
46          }
47      }t[N];
48      int root, g[N];
49      void init() { root = 0; }
50      void print(int o) {
51          if (!o) return;
52          print(t[o].ls);
53          g[++*g] = o;
54          print(t[o].rs);
55      }
56      void pushup(int o) {
57          t[o].sze = t[t[o].ls].sze + t[t[o].rs].sze + 1;
58          t[o].sum = t[t[o].ls].sum + t[t[o].rs].sum + s[o].v;
59          for (int i = 0; i < K; ++i) {
60              t[o].Min[i] = t[o].Max[i] = s[o].x[i];
61          }
62          if (t[o].ls) {
63              for (int i = 0; i < K; ++i) {
64                  t[o].Min[i] = min(t[o].Min[i], t[t[o].ls].Min[i]);
65                  t[o].Max[i] = max(t[o].Max[i], t[t[o].ls].Max[i]);
66              }
67          }
68          if (t[o].rs) {
69              for (int i = 0; i < K; ++i) {
70                  t[o].Min[i] = min(t[o].Min[i], t[t[o].rs].Min[i]);
71                  t[o].Max[i] = max(t[o].Max[i], t[t[o].rs].Max[i]);
72              }
73          }
74      }
75      int build(int l, int r) {
76          if (l > r) return 0;
77          int mid = (l + r) >> 1;
```

```
78          double ave[2] = {0}, vax[2] = {0};
79          for (int i = l; i <= r; i++)
80              for (int j = 0; j < K; ++j)
81                  ave[j] += s[g[i]].x[j];
82          for (int i = 0; i < K; ++i) ave[i] /= (r - l + 1);
83          for (int i = l; i <= r; i++)
84              for (int j = 0; j < K; ++j)
85                  vax[j] += (s[g[i]].x[j] - ave[j]) * (s[g[i]].x[j] - ave[j]);
86          int div = 0;
87          for (int i = 1; i < K; ++i) {
88              if (vax[i] > vax[div])
89                  div = i;
90          }
91          nth_element(g + l, g + mid, g + r + 1, cmpx(div));
92          int o = g[mid];
93          t[o].div = div;
94          t[o].ls = build(l, mid - 1);
95          t[o].rs = build(mid + 1, r);
96          pushup(o);
97          return o;
98      }
99      void rebuild(int &o) {
100         *g = 0;
101         print(o);
102         o = build(1, *g);
103     }
104     bool bad(int o) { return alpha * t[o].sze <= (double)max(t[t[o].ls].sze,
            t[t[o].rs].sze); }
105     void insert(int &o, int v) {
106         if (!o) {
107             o = v;
108             pushup(o);
109             return;
110         }
111         if (cmp(v, o, t[o].div))
112             insert(t[o].ls, v);
113         else
114             insert(t[o].rs, v);
115         pushup(o);
116         if (bad(o)) rebuild(o);
117     }
118     int query(int o, db *Min, db *Max) {
119         if (!o || t[o].Out(Min, Max)) return 0;
120         if (t[o].In(Min, Max)) return t[o].sum;
121         int res = 0;
122         if (s[o].In(Min, Max)) res += s[o].v;
123         return query(t[o].ls, Min, Max) + query(t[o].rs, Min, Max) + res;
124     }
125 }kdTree;
126
127 int main() {
128     int n, op, ans = 0;
129     db Min[DIM], Max[DIM];
130     scanf("%d", &n);
131     while (scanf("%d", &op) != EOF) {
132         if (op == 1) {
133             ++cntS;
134             scanf("%d%d%d", &s[cntS].x[0], &s[cntS].x[1], &s[cntS].v);
```

```
135             s[cntS].x[0] ^= ans;
136             s[cntS].x[1] ^= ans;
137             s[cntS].v ^= ans;
138             kdTree.insert(kdTree.root, cntS);
139         }
140         if (op == 2) {
141             for (int i = 0; i < K; ++i) scanf("%d", Min + i), Min[i] ^= ans;
142             for (int i = 0; i < K; ++i) scanf("%d", Max + i), Max[i] ^= ans;
143             printf("%d\n", ans = kdTree.query(kdTree.root, Min, Max));
144         }
145         if (op == 3) break;
146     }
147     return 0;
148 }
```

7.4 BZOJ 2716

每次插入一个点，或者给出一个点，询问已经被插入的点中和它曼哈顿距离最近的点的距离。
带插入平面最近点对加上信仰剪枝即可。

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef int db;
4  const int N = 1e6 + 10;
5  const int DIM = 2;
6  const int K = 2;
7  const db INF = 0x3f3f3f3f;
8
9  struct Point {
10     db x[DIM];
11     void scan() { for (int i = 0; i < K; ++i) scanf("%d", x + i); }
12     db dis(const Point &b) const {
13         db res = 0;
14         for (int i = 0; i < K; ++i)
15             res += abs(x[i] - b.x[i]);
16         return res;
17     }
18 }s[N]; int cntS;
19
20 struct KDTree {
21     #define alpha 0.725
22     struct cmpx {
23         int div;
24         cmpx(int div = 0) : div(div) {}
25         bool operator () (const int &a, const int &b) {
26             return s[a].x[div] < s[b].x[div];
27         }
28     };
29     bool cmp(const int &a, const int &b, const int &div) { return s[a].x[div
           ] <= s[b].x[div]; }
30     struct Node {
31         int ls, rs, div;
32         db Min[DIM], Max[DIM];
33         int sze;
34     }t[N];
35     int root, g[N];
36     db ans;
37     void init() { root = 0; }
```

```
38      void print(int o) {
39          if (!o) return;
40          print(t[o].ls);
41          g[++*g] = o;
42          print(t[o].rs);
43      }
44      void pushup(int o) {
45          t[o].sze = t[t[o].ls].sze + t[t[o].rs].sze + 1;
46          for (int i = 0; i < K; ++i) {
47              t[o].Min[i] = t[o].Max[i] = s[o].x[i];
48          }
49          if (t[o].ls) {
50              for (int i = 0; i < K; ++i) {
51                  t[o].Min[i] = min(t[o].Min[i], t[t[o].ls].Min[i]);
52                  t[o].Max[i] = max(t[o].Max[i], t[t[o].ls].Max[i]);
53              }
54          }
55          if (t[o].rs) {
56              for (int i = 0; i < K; ++i) {
57                  t[o].Min[i] = min(t[o].Min[i], t[t[o].rs].Min[i]);
58                  t[o].Max[i] = max(t[o].Max[i], t[t[o].rs].Max[i]);
59              }
60          }
61      }
62      int build(int l, int r) {
63          if (l > r) return 0;
64          int mid = (l + r) >> 1;
65          double ave[2] = {0}, vax[2] = {0};
66          for (int i = l; i <= r; i++)
67              for (int j = 0; j < K; ++j)
68                  ave[j] += s[g[i]].x[j];
69          for (int i = 0; i < K; ++i) ave[i] /= (r - l + 1);
70          for (int i = l; i <= r; i++)
71              for (int j = 0; j < K; ++j)
72                  vax[j] += (s[g[i]].x[j] - ave[j]) * (s[g[i]].x[j] - ave[j]);
73          int div = 0;
74          for (int i = 1; i < K; ++i) {
75              if (vax[i] > vax[div])
76                  div = i;
77          }
78          nth_element(g + l, g + mid, g + r + 1, cmpx(div));
79          int o = g[mid];
80          t[o].div = div;
81          t[o].ls = build(l, mid - 1);
82          t[o].rs = build(mid + 1, r);
83          pushup(o);
84          return o;
85      }
86      void rebuild(int &o) {
87          *g = 0;
88          print(o);
89          o = build(1, *g);
90      }
91      bool bad(int o) { return alpha * t[o].sze <= (double)max(t[t[o].ls].sze,
            t[t[o].rs].sze); }
92      void insert(int &o, int v) {
93          if (!o) {
94              o = v;
```

```
95              pushup(o);
96              return;
97          }
98          if (cmp(v, o, t[o].div))
99              insert(t[o].ls, v);
100         else
101             insert(t[o].rs, v);
102         pushup(o);
103         if (bad(o)) rebuild(o);
104     }
105     db calc(int o, Point &e) {
106         if (!o) return INF;
107         db res = 0;
108         for (int i = 0; i < K; ++i) {
109             if (t[o].Min[i] > e.x[i])
110                 res += abs(t[o].Min[i] - e.x[i]);
111             if (t[o].Max[i] < e.x[i])
112                 res += abs(t[o].Max[i] - e.x[i]);
113         }
114         return res;
115     }
116     void query(int o, Point &e) {
117         if (!o) return;
118         ans = min(ans, e.dis(s[o]));
119         db dl = calc(t[o].ls, e), dr = calc(t[o].rs, e);
120         if (dl < dr) {
121             if (dl < ans) query(t[o].ls, e);
122             if (dr < ans) query(t[o].rs, e);
123         } else {
124             if (dr < ans) query(t[o].rs, e);
125             if (dl < ans) query(t[o].ls, e);
126         }
127     }
128     db query(Point &e) {
129         ans = INF;
130         query(root, e);
131         return ans;
132     }
133 }kdTree;
134
135 int main() {
136     int n, q, op;
137     Point e;
138     cntS = 0;
139     scanf("%d%d", &n, &q);
140     for (int i = 1; i <= n; ++i) {
141         ++cntS;
142         s[cntS].scan();
143         kdTree.g[i] = i;
144     }
145     kdTree.root = kdTree.build(1, n);
146     while (q--) {
147         scanf("%d", &op);
148         if (op == 1) {
149             ++cntS; s[cntS].scan();
150             kdTree.insert(kdTree.root, cntS);
151         } else {
152             e.scan();
```

```
153             printf("%d\n", kdTree.query(e));
154         }
155     }
156     return 0;
157 }
```

8 CDQ 分治

解决和点对 (i, j) 有关的问题：

1. 找到序列中点 mid
2. 将点分成三类：
   - $i \in [l, mid], j \in [l, mid]$ 的点对
   - $i \in [l, mid], j \in [mid + 1, r]$ 的点对
   - $i \in [mid + 1, r], j \in [mid + 1, r]$ 的点对
3. 只需处理第二类点，第一类和第三类直接递归即可。

8.1 HDU 5126

单点加，立方体求和
相当于四维偏序问题，多套一层 CDQ 即可。

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  const int N = 1e5 + 10;
4  int ans[N];
5
6  struct Hash {
7      vector <int> a;
8      int& operator[](int x) { return a[x - 1]; }
9      int size() { return a.size(); }
10     void init() { a.clear(); }
11     void add(int x) { a.push_back(x); }
12     void gao() { sort(a.begin(), a.end()); a.erase(unique(a.begin(), a.end()
           ), a.end()); }
13     int get(int x) { return lower_bound(a.begin(), a.end(), x) - a.begin() +
           1; }
14 }hs;
15
16 struct BIT {
17     int a[N], n;
18     void init(int _n = N - 5) { n = _n; memset(a, 0, sizeof (a[0]) * (n + 5)
           ); }
19     void update(int x, int v) { for (; x <= n; x += x & -x) a[x] += v; }
20     int query(int x) {
21         int res = 0;
22         for (; x > 0; x -= x & -x)
23             res += a[x];
24         return res;
25     }
26 }bit;
27
28 namespace CDQ {
29     struct node {
30         // 0 加点 -1, 1询问
```

```
31          int op, x, y, z, id;
32          node() {}
33          node(int op, int x, int y, int z, int id = 0) : op(op), x(x), y(y),
                z(z), id(id) {}
34      }t[N * 8], t1[N * 8], t2[N * 8];
35      bool cmp1(const node &a, const node &b) {
36          if (a.x == b.x)
37              return a.id < b.id;
38          return a.x < b.x;
39      }
40      bool cmp2(const node &a, const node &b) {
41          if (a.y == b.y)
42              return a.id < b.id;
43          return a.y < b.y;
44      }
45      int tot, tot1, tot2;
46      void init() { hs.init(); tot = 0; }
47      void addModify(int x, int y, int z) {
48          t[++tot] = node(0, x, y, z);
49          hs.add(z);
50      }
51      void addQuery(int x1, int y1, int z1, int x2, int y2, int z2, int id) {
52          t[++tot] = node(1, x2, y2, z2, id);
53          t[++tot] = node(-1, x2, y2, z1 - 1, id);
54          t[++tot] = node(-1, x2, y1 - 1, z2, id);
55          t[++tot] = node(-1, x1 - 1, y2, z2, id);
56          t[++tot] = node(1, x2, y1 - 1, z1 - 1, id);
57          t[++tot] = node(1, x1 - 1, y2, z1 - 1, id);
58          t[++tot] = node(1, x1 - 1, y1 - 1, z2, id);
59          t[++tot] = node(-1, x1 - 1, y1 - 1, z1 - 1, id);
60          hs.add(z2); hs.add(z1 - 1);
61      }
62      void calc() {
63          for (int i = 1; i <= tot2; ++i) {
64              if (t2[i].op == 0) {
65                  bit.update(t2[i].z, 1);
66              } else {
67                  ans[t2[i].id] += t2[i].op * bit.query(t2[i].z);
68              }
69          }
70          for (int i = 1; i <= tot2; ++i)
71              if (t2[i].op == 0)
72                  bit.update(t2[i].z, -1);
73      }
74      void solve2(int l, int r) {
75          if (l >= r) return;
76          int mid = (l + r) >> 1;
77          solve2(l, mid); solve2(mid + 1, r);
78          tot2 = 0;
79          for (int i = l; i <= mid; ++i) {
80              if (t1[i].op == 0)
81                  t2[++tot2] = t1[i];
82          }
83          for (int i = mid + 1; i <= r; ++i) {
84              if (t1[i].op)
85                  t2[++tot2] = t1[i];
86          }
87          sort(t2 + 1, t2 + 1 + tot2, cmp2);
```

```
88          calc();
89      }
90      void solve1(int l, int r) {
91          if (l >= r) return;
92          int mid = (l + r) >> 1;
93          solve1(l, mid); solve1(mid + 1, r);
94          tot1 = 0;
95          for (int i = l; i <= mid; ++i) {
96              if (t[i].op == 0)
97                  t1[++tot1] = t[i];
98          }
99          for (int i = mid + 1; i <= r; ++i) {
100             if (t[i].op)
101                 t1[++tot1] = t[i];
102         }
103         sort(t1 + 1, t1 + 1 + tot1, cmp1);
104         solve2(1, tot1);
105     }
106     void gao() {
107         hs.gao();
108         bit.init(hs.size());
109         for (int i = 1; i <= tot; ++i)
110             t[i].z = hs.get(t[i].z);
111         solve1(1, tot);
112     }
113 }
114
115 int main() {
116     int _T; scanf("%d", &_T);
117     while (_T--) {
118         int q;
119         scanf("%d", &q);
120         CDQ::init();
121         int _q = 0;
122         for (int i = 1; i <= q; ++i) {
123             int op, x[2], y[2], z[2];
124             scanf("%d%d%d%d", &op, x, y, z);
125             if (op == 1) CDQ::addModify(x[0], y[0], z[0]);
126             else {
127                 scanf("%d%d%d", x + 1, y + 1, z + 1);
128                 CDQ::addQuery(x[0], y[0], z[0], x[1], y[1], z[1], ++_q);
129             }
130         }
131         memset(ans, 0, sizeof (ans[0]) * (_q + 1));
132         CDQ::gao();
133         for (int i = 1; i <= _q; ++i)
134             printf("%d\n", ans[i]);
135     }
136     return 0;
137 }
```

8.2 Luogu P2487

题意:
给出 n 个二元祖，对每个元祖求其在最长上升子序列中的方案数除总的最长上升子序列方案数。
转移方程类似:

$$dp_i = 1 + max_{j=1}^{i-1} dp_j[a_j < a_i][b_j < b_i]$$

令递归 (l, mid)，然后处理当前区间跨过 mid 的贡献，然后递归 $(mid+1, r)$。

dp 的时候多维护方案即可。

```
#include <bits/stdc++.h>
using namespace std;
#define dbg(x...) do { cout << "\033[32;1m" << #x << " -> "; err(x); } while (0)
void err() { cout << "\033[39;0m" << endl; }
template <class T, class... Ts> void err(const T& arg, const Ts&... args) { cout << arg << ' '; err(args...); }
typedef double db;
typedef pair<int, db> pID;
#define fi first
#define se second
const int N = 5e4 + 10;
int n;

pID op(pID a, pID b) {
    if (a.fi > b.fi) return a;
    if (a.fi < b.fi) return b;
    return pID(a.fi, a.se + b.se);
}

struct Hash {
    vector <int> a;
    int& operator[](int x) { return a[x - 1]; }
    int size() { return a.size(); }
    void init() { a.clear(); }
    void add(int x) { a.push_back(x); }
    void gao() { sort(a.begin(), a.end()); a.erase(unique(a.begin(), a.end()), a.end()); }
    int get(int x) { return lower_bound(a.begin(), a.end(), x) - a.begin() + 1; }
}hy, hz;

struct SEG {
    pID t[N << 2];
    void build(int id, int l, int r) {
        t[id] = pID(0, 1);
        if (l == r) return;
        int mid = (l + r) >> 1;
        build(id << 1, l, mid);
        build(id << 1 | 1, mid + 1, r);
    }
    void update(int id, int l, int r, int pos, pID v) {
        if (l == r) {
            if (v == pID(-1, -1)) t[id] = pID(0, 1);
            else t[id] = op(t[id], v);
            return;
        }
        int mid = (l + r) >> 1;
        if (pos <= mid) update(id << 1, l, mid, pos, v);
        else update(id << 1 | 1, mid + 1, r, pos, v);
        t[id] = op(t[id << 1], t[id << 1 | 1]);
    }
    pID query(int id, int l, int r, int ql, int qr) {
        if (l >= ql && r <= qr) return t[id];
        int mid = (l + r) >> 1;
        pID res = pID(0, 1);
```

```
53          if (ql <= mid) res = op(res, query(id << 1, l, mid, ql, qr));
54          if (qr > mid) res = op(res, query(id << 1 | 1, mid + 1, r, ql, qr));
55          return res;
56      }
57  }seg;
58
59  struct E { int x, y, z; }e[N];
60
61  struct CDQ {
62      struct node {
63          int x, y, z;
64          bool operator < (const node &other) const { return x < other.x; }
65      }t[N], t1[N];
66      static bool cmp(const node &a, const node &b) {
67          if (a.y == b.y)
68              return a.x < b.x;
69          return a.y < b.y;
70      }
71      int tot;
72      void init() { tot = 0; }
73      void addNode(int x, int y, int z) { t[++tot] = {x, y, z}; }
74      pID f[N];
75      void solve(int l, int r) {
76          if (l == r) {
77              if (f[l].fi <= 1)
78                  f[l] = pID(1, 1);
79              return;
80          }
81          int mid = (l + r) >> 1;
82          solve(l, mid);
83          for (int i = l; i <= r; ++i) t1[i] = t[i];
84          sort(t1 + l, t1 + r + 1, cmp);
85          for (int i = l; i <= r; ++i) {
86              if (t1[i].x <= mid) {
87                  seg.update(1, 1, hz.size(), t1[i].z, f[t1[i].x]);
88              } else {
89                  pID tmp = seg.query(1, 1, hz.size(), 1, t1[i].z);
90                  ++tmp.fi;
91                  f[t1[i].x] = op(f[t1[i].x], tmp);
92              }
93          }
94          for (int i = l; i <= r; ++i) {
95              if (t1[i].x <= mid) {
96                  seg.update(1, 1, hz.size(), t1[i].z, pID(-1, -1));
97              }
98          }
99          solve(mid + 1, r);
100     }
101     void gao() {
102         seg.build(1, 1, hz.size());
103         memset(f, 0, sizeof (f[0]) * (tot + 5));
104         sort(t + 1, t + 1 + tot);
105         solve(1, tot);
106     }
107 }cdqF, cdqG;
108
109 int main() {
110     while (scanf("%d", &n) != EOF) {
```

```
111         hy.init(); hz.init();
112         for (int i = 1; i <= n; ++i) {
113             e[i].x = i;
114             scanf("%d%d", &e[i].y, &e[i].z);
115             hy.add(e[i].y);
116             hz.add(e[i].z);
117         }
118         hy.gao(); hz.gao();
119         for (int i = 1; i <= n; ++i) {
120             e[i].y = hy.get(e[i].y);
121             e[i].z = hz.get(e[i].z);
122         }
123         cdqF.init();
124         for (int i = 1; i <= n; ++i) {
125             cdqF.addNode(e[i].x, hy.size() - e[i].y + 1, hz.size() - e[i].z
                + 1);
126         }
127         cdqF.gao();
128         cdqG.init();
129         for (int i = 1; i <= n; ++i) {
130             cdqG.addNode(n - e[i].x + 1, e[i].y, e[i].z);
131         }
132         cdqG.gao();
133         pID res = pID(0, 0);
134         for (int i = 1; i <= n; ++i) res = op(res, cdqF.f[i]);
135         printf("%d\n", res.fi);
136         for (int i = 1; i <= n; ++i) {
137             int j = n - i + 1;
138             pID _f = cdqF.f[i], _g = cdqG.f[j];
139             db ans = 0;
140             if (_f.fi + _g.fi - 1 == res.fi) {
141                 ans = _f.se * _g.se / res.se;
142             }
143             printf("%.5f%c", ans, " \n"[i == n]);
144         }
145     }
146     return 0;
147 }
```

8.3 LOJ 135

题意：维护矩形加、矩形求和。

考虑静态问题，就是给出一堆矩形，每次询问一个矩形内的和。

考虑线段树 + 扫描线的做法，对矩形加用一次前缀和思想拆成两次单点加，对矩形求和再用一次前缀和思想拆成两次单点求和, 另一维度直接用线段树维护前缀增量与前缀和即可

那么现在动态矩形加，使用 *CDQ* 分治时间维度，那么只需要考虑跨 *mid* 的修改和询问对，这个时候就变成了上述的静态问题。

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const int N = 4e5 + 10, M = 2e3 + 100;
5  int n, m; ll ans[N];
6
7  struct BIT_2D {
8      struct BIT {
9          ll a[M];
10         void init() { memset(a, 0, sizeof a); }
```

```
11          void add(int x, ll v) { for (; x < M; a[x] += v, x += x & -x); }
12          ll ask(int x) { ll res = 0; for (; x > 0; res += a[x], x -= x & -x);
                return res; }
13      }bit1, bit2;
14      void init() { bit1.init(); bit2.init(); }
15      ll ask(int x) { return bit1.ask(x) * (x + 1) - bit2.ask(x); }
16      void add(int x, ll v) { bit1.add(x, v); bit2.add(x, x * v); }
17      ll ask(int l, int r) { return ask(r) - ask(l - 1); }
18      void add(int l, int r, ll v) { add(l, v); add(r + 1, -v); }
19  }bit[2];
20
21  namespace CDQ {
22      struct node {
23          int tp, opt, x, y, z1, z2, id, v;
24          node() {}
25          node(int tp, int opt, int x, int y, int z1, int z2, int id, int v) :
                tp(tp), opt(opt), x(x), y(y), z1(z1), z2(z2), id(id), v(v) {}
26          bool operator < (const node &other) const { return x < other.x; }
27      }t[N], t1[N];
28      bool cmp1(const node &a, const node &b) {
29          if (a.y == b.y)
30              return a.x < b.x;
31          return a.y < b.y;
32      }
33      int tot, tot1;
34      void init() { tot = 0; }
35      void addQuery(int tp, int opt, int x, int y, int z1, int z2, int id) { t
           [++tot] = node(tp, opt, x, y, z1, z2, id, 0); }
36      void addModify(int tp, int opt, int x, int y, int z1, int z2, int v) { t
           [++tot] = node(tp, opt, x, y, z1, z2, 0, v); }
37      void calc() {
38          for (int i = 1; i <= tot1; ++i) {
39              if (t1[i].tp == 0) {
40                  bit[0].add(t1[i].z1, t1[i].z2, -1ll * (t1[i].y - 1) * t1[i].
                       v * t1[i].opt);
41                  bit[1].add(t1[i].z1, t1[i].z2, t1[i].v * t1[i].opt);
42              } else {
43                  ans[t1[i].id] += 1ll * t1[i].opt * (bit[0].ask(t1[i].z1, t1[
                       i].z2) + 1ll * t1[i].y * bit[1].ask(t1[i].z1, t1[i].z2));
44              }
45          }
46          for (int i = 1; i <= tot1; ++i) {
47              if (t1[i].tp == 0) {
48                  bit[0].add(t1[i].z1, t1[i].z2, -1ll * (t1[i].y - 1) * t1[i].
                       v * t1[i].opt * -1);
49                  bit[1].add(t1[i].z1, t1[i].z2, t1[i].v * t1[i].opt * -1);
50              }
51          }
52      }
53      void solve(int l, int r) {
54          if (l == r) return;
55          int mid = (l + r) >> 1;
56          solve(l, mid); solve(mid + 1, r);
57          tot1 = 0;
58          for (int i = l; i <= mid; ++i) {
59              if (t[i].tp == 0)
60                  t1[++tot1] = t[i];
61          }
```

```
62          for (int i = mid + 1; i <= r; ++i) {
63              if (t[i].tp == 1)
64                  t1[++tot1] = t[i];
65          }
66          sort(t1 + 1, t1 + 1 + tot1, cmp1);
67          calc();
68      }
69      void gao() {
70          memset(ans, 0, sizeof (ans[0]) * (tot + 5));
71          bit[0].init();
72          bit[1].init();
73          solve(1, tot);
74      }
75  }
76
77  int main() {
78      scanf("%d%d", &n, &m);
79      int op, x[2], y[2], v;
80      CDQ::init();
81      int q = 0, _q = 0;
82      while (scanf("%d", &op) != EOF) {
83          ++q;
84          scanf("%d%d%d%d", x, y, x + 1, y + 1);
85          if (op == 1) {
86              scanf("%d", &v);
87              CDQ::addModify(0, 1, q, x[0], y[0], y[1], v);
88              CDQ::addModify(0, -1, q, x[1] + 1, y[0], y[1], v);
89          } else {
90              ++_q;
91              CDQ::addQuery(1, -1, q, x[0] - 1, y[0], y[1], _q);
92              CDQ::addQuery(1, 1, q, x[1], y[0], y[1], _q);
93          }
94      }
95      CDQ::gao();
96      for (int i = 1; i <= _q; ++i)
97          printf("%lld\n", ans[i]);
98      return 0;
99  }
```

9 树上问题

9.1 动态维护树的直径

CF 1192B
题意：给出一棵树，每次修改一条边的权值，针对每次修改都需要输出当前树的直径。强制在线。

9.1.1 解法一

考虑树的全 *DFS* 序,(长度为 $2n-1$ 的 *DFS* 序，允许一个节点在 *DFS* 序中出现多次)。$p_1, p_2, \cdots, p_{2n-1}$
令 $index(x)$ 表示节点 x 在全 *DFS* 序中第一次出现的下标，即 $p_{index(x)} = x$
考虑两个节点 x 和 y, 其下标 $a = index(x), b = index(y)$, 不妨设 $a \le b$, 这两个节点之间的距离：

$$\begin{aligned} dis(x,y) &= dis(r,x) + dis(r,y) - 2 \cdot dis(r, LCA(x,y)) \\ &= dis(r,x) + dis(r,y) - 2 \cdot \min_{a \le c \le b}\{dis(r,p_c)\} \end{aligned}$$

那么树的直径为：

$$\max_{x,y}\{\mathrm{dis}(x,y)\} = \max_{1 \le a \le c \le b \le 2n-1}\{\mathrm{dis}(r,p_a) + \mathrm{dis}(r,p_b) - 2\mathrm{dis}(r,p_c)\}.$$

可以发现这个信息可以合并，那么用线段树维护全 DFS 序的信息，时间复杂度 $O(n+q\log n)$

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5 + 10;
int n, q; ll W;
struct Edge { int u, v; ll w; }e[N];

struct Graph {
    struct E { int to, nx; ll w; }e[N << 1]; int h[N], cnt;
    void init(int n) { for (int i = 0; i <= n; ++i) h[i] = -1; cnt = -1; }
    void addedge(int u, int v, ll w = 0) { e[++cnt] = { v, h[u], w}; h[u] =
       cnt; }
}G;

struct SEG {
    struct node {
        ll val, M, LM, MR, LMR, lazy;
        node() { val = M = LM = MR = LMR = lazy = 0; }
        void up(ll _lazy) {
            val += _lazy;
            M -= _lazy * 2;
            LM -= _lazy;
            MR -= _lazy;
            lazy += _lazy;
        }
        node operator + (const node &other) const {
            node res = node();
            res.val = max(val, other.val);
            res.M = max(M, other.M);
            res.LM = max(LM, max(other.LM, val + other.M));
            res.MR = max(MR, max(other.MR, M + other.val));
            res.LMR = max(LMR, max(other.LMR, max(LM + other.val, val +
               other.MR)));
            return res;
        }
    }t[(N * 2) << 2];
    void build(int id, int l, int r) {
        t[id] = node();
        if (l == r) return;
        int mid = (l + r) >> 1;
        build(id << 1, l, mid);
        build(id << 1 | 1, mid + 1, r);
    }
    void pushdown(int id) {
        ll &lazy = t[id].lazy;
        t[id << 1].up(lazy);
        t[id << 1 | 1].up(lazy);
        lazy = 0;
    }
    void update(int id, int l, int r, int ql, int qr, ll v) {
        if (l >= ql && r <= qr) {
            t[id].up(v);
            return;
        }
        int mid = (l + r) >> 1;
        pushdown(id);
        if (ql <= mid) update(id << 1, l, mid, ql, qr, v);
```

```
56          if (qr > mid) update(id << 1 | 1, mid + 1, r, ql, qr, v);
57          t[id] = t[id << 1] + t[id << 1 | 1];
58      }
59  }seg;
60
61  namespace Tree {
62      int fa[N], in[N], out[N], r[N << 1]; ll weight[N];
63      void dfs(int u) {
64          r[++*r] = u;
65          in[u] = *r;
66          for (int i = G.h[u]; ~i; i = G.e[i].nx) {
67              int v = G.e[i].to; ll w = G.e[i].w;
68              if (v == fa[u]) continue;
69              fa[v] = u;
70              weight[v] = w;
71              dfs(v);
72              r[++*r] = u;
73          }
74          out[u] = *r;
75      }
76      void gao() {
77          *r = 0;
78          dfs(1);
79          seg.build(1, 1, *r);
80          for (int i = 1; i < n; ++i) {
81              if (fa[e[i].u] == e[i].v) swap(e[i].u, e[i].v);
82              seg.update(1, 1, *r, in[e[i].v], out[e[i].v], weight[e[i].v]);
83          }
84          ll lst = 0;
85          for (int i = 1; i <= q; ++i) {
86              int _d; ll _w;
87              scanf("%d%lld", &_d, &_w);
88              _d = (_d + lst) % (n - 1) + 1;
89              _w = (_w + lst) % W;
90              seg.update(1, 1, *r, in[e[_d].v], out[e[_d].v], _w - weight[e[_d
                    ].v]);
91              weight[e[_d].v] = _w;
92              lst = seg.t[1].LMR;
93              printf("%lld\n", lst);
94          }
95      }
96  }
97
98  int main() {
99      while (scanf("%d%d%lld", &n, &q, &W) != EOF) {
100         G.init(n);
101         for (int i = 1; i < n; ++i) {
102             scanf("%d%d%lld", &e[i].u, &e[i].v, &e[i].w);
103             G.addedge(e[i].u, e[i].v, e[i].w);
104             G.addedge(e[i].v, e[i].u, e[i].w);
105         }
106         Tree::gao();
107     }
108     return 0;
109 }
```

9.1.2 解法二

注意到树的直径有以下性质：定理：令 $farthest(x)$ 表示与节点 x 距离最远的节点的集合。则对任意节点 x，都有任意 $a \in farthest(x)$ 与 $b \in farthest(a)$，a 与 b 的距离即为直径。

我们任选一个节点 r 作为树的根，任意边 (x,y)，若 x 为 y 的父节点，则把边权置于节点 y 上，视作点权，记作 $w[y]$，则修改边权都可以视作修改点权。

我们考虑询问树的直径，可以分成两个步骤：1. 找到离 r 距离最远的叶子节点 x。2. 找到离 x 距离最远的节点 y。

1. 找到离 r 距离最远的叶子节点 x。我们在树的 DFS 序 $p_1, p_2, \cdots, p_n$ 上查找 $dis(r, p_i)$ 的最大值及其下标 i，则 $x = p_i$。可在 $O(\log n)$ 时间复杂度内解决。
2. 找到离 x 距离最远的节点 y。设 $LCA(x,y) = u$，则 $dis(x,y) = dis(r,x) + dis(r,y) - 2dis(r,u)$。考虑重链剖分，在 x 到 r 的路径中的所有重链中，找到最大的节点 u，使得 $dis(r, light(u)) - 2dis(r,u)$ 最大，其中 $light(u)$ 表示节点 u 的轻链连接出去的后代节点集合（包括 u 本身）。可在 $O(\log n)$ 时间复杂度内解决。

接下来考虑修改操作，我们只需要维护

1. 步骤 1 的 $dis(r, p_i)$。受影响的只有其子树中的点，只需要用线段树维护区间整体加减，以及区间最值即可。时间复杂度 $O(\log n)$
2. 步骤 2 的 $dis(r, light(u)) - 2dis(r,u)$。对于其子树中的点，同样需要用线段树维护区间整体加减，以及区间最值。此外，对于子树外的点，考虑改变一条边权，在它的祖先方向，它只会影响到 $O(logn)$ 个点 (即每次跳到重链头节点的父节点) 的轻链贡献。既然第一步的贡献已经维护好了，对于这 $O(\log n)$ 个点的轻链贡献，可以暴力往上跳 $O(\log n)$ 步，每次在另一棵线段树上查询代价，时间复杂度 $O(\log^2 n)$

于是，我们可以在时间复杂度 $O(n + q\log^2 n)$ 内解决。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5 + 10;
const ll INF = 1e18;
int n, q; ll W;
struct Edge { int u, v; ll w; }e[N];

struct Graph {
    struct E { int to, nx; ll w; }e[N << 1]; int h[N], cnt;
    void init(int n) { for (int i = 0; i <= n; ++i) h[i] = -1; cnt = -1; }
    void addedge(int u, int v, ll w = 0) { e[++cnt] = { v, h[u], w}; h[u] =
       cnt; }
}G;

struct SEG {
    struct node {
        ll val, lazy;
        node() { val = lazy = 0; }
        void up(ll _lazy) {
            lazy += _lazy;
            val += _lazy;
        }
        node operator + (const node &other) const {
            node res = node();
            res.val = max(val, other.val);
            return res;
        }
    }t[N << 2];
    void build(int id, int l, int r) {
        t[id] = node();
        if (l == r) return;
        int mid = (l + r) >> 1;
```

```
33          build(id << 1, l, mid);
34          build(id << 1 | 1, mid + 1, r);
35      }
36      void pushdown(int id) {
37          ll &lazy = t[id].lazy;
38          t[id << 1].up(lazy);
39          t[id << 1 | 1].up(lazy);
40          lazy = 0;
41      }
42      void update(int id, int l, int r, int ql, int qr, ll v) {
43          if (ql > qr) return;
44          if (l >= ql && r <= qr) {
45              t[id].up(v);
46              return;
47          }
48          int mid = (l + r) >> 1;
49          pushdown(id);
50          if (ql <= mid) update(id << 1, l, mid, ql, qr, v);
51          if (qr > mid) update(id << 1 | 1, mid + 1, r, ql, qr, v);
52          t[id] = t[id << 1] + t[id << 1 | 1];
53      }
54      ll query(int id, int l, int r, int ql, int qr) {
55          if (ql > qr) return -INF;
56          if (l >= ql && r <= qr) return t[id].val;
57          int mid = (l + r) >> 1;
58          pushdown(id);
59          ll res = -INF;
60          if (ql <= mid) res = max(res, query(id << 1, l, mid, ql, qr));
61          if (qr > mid) res = max(res, query(id << 1 | 1, mid + 1, r, ql, qr))
                ;
62          return res;
63      }
64      int getPos(int id, int l, int r, ll v) {
65          if (l == r) return l;
66          int mid = (l + r) >> 1;
67          pushdown(id);
68          if (t[id << 1].val == v) return getPos(id << 1, l, mid, v);
69          else return getPos(id << 1 | 1, mid + 1, r, v);
70      }
71      void modify(int pos, ll v) {
72          ll pre = query(1, 1, n, pos, pos);
73          update(1, 1, n, pos, pos, v - pre);
74      }
75  }seg[2];
76
77  struct HLD {
78      int fa[N], deep[N], sze[N], son[N], top[N], in[N], fin[N], out[N];
79      ll dis[N], weight[N];
80      void dfs(int u) {
81          sze[u] = 1; son[u] = 0;
82          for (int i = G.h[u]; ~i; i = G.e[i].nx) {
83              int v = G.e[i].to; ll w = G.e[i].w;
84              if (v == fa[u]) continue;
85              fa[v] = u;
86              deep[v] = deep[u] + 1;
87              weight[v] = w;
88              dis[v] = dis[u] + w;
89              dfs(v);
```

```
90              sze[u] += sze[v];
91              if (!son[u] || sze[v] > sze[son[u]]) son[u] = v;
92          }
93      }
94      void gettop(int u, int tp) {
95          in[u] = ++*in;
96          fin[*in] = u;
97          top[u] = tp;
98          if (son[u]) gettop(son[u], tp);
99          for (int i = G.h[u]; ~i; i = G.e[i].nx) {
100             int v = G.e[i].to;
101             if (v != son[u] && v != fa[u]) {
102                 gettop(v, v);
103             }
104         }
105         out[u] = *in;
106     }
107     void modify(int u, ll w) {
108         seg[0].update(1, 1, n, in[u], out[u], w - weight[u]);
109         seg[1].update(1, 1, n, in[u], out[u], weight[u] - w);
110         weight[u] = w;
111         while (u) {
112             u = fa[top[u]];
113             if (u) {
114                 ll Max = max(seg[0].query(1, 1, n, in[u], in[u]), seg[0].
                        query(1, 1, n, out[son[u]] + 1, out[u]));
115                 seg[1].modify(in[u], Max - seg[0].query(1, 1, n, in[u], in[u
                        ]) * 2);
116             }
117         }
118     }
119     ll diameter() {
120         ll Mdis = seg[0].t[1].val;
121         int u = fin[seg[0].getPos(1, 1, n, Mdis)];
122         ll res = 0;
123         while (u) {
124             int _u = top[u];
125             if (in[_u] < in[u]) {
126                 res = max(res, seg[1].query(1, 1, n, in[_u], in[u] - 1));
127             }
128             u = fa[_u];
129             if (u) {
130                 ll Max = max(seg[0].query(1, 1, n, in[u], in[_u] - 1), seg
                        [0].query(1, 1, n, out[_u] + 1, out[u]));
131                 res = max(res, Max - seg[0].query(1, 1, n, in[u], in[u]) *
                        2);
132             }
133         }
134         return res + Mdis;
135     }
136     void gao() {
137         *in = 0;
138         fa[1] = 0;
139         deep[1] = 0;
140         dfs(1);
141         gettop(1, 1);
142         for (int i = 1; i < n; ++i) {
143             if (fa[e[i].u] == e[i].v)
```

```
144                 swap(e[i].u, e[i].v);
145         }
146         seg[0].build(1, 1, n);
147         seg[1].build(1, 1, n);
148         for (int i = 1; i <= n; ++i) {
149             seg[0].update(1, 1, n, i, i, dis[fin[i]]);
150         }
151         for (int i = 1; i <= n; ++i) {
152             ll Max = dis[fin[i]];
153             Max = max(Max, seg[0].query(1, 1, n, out[son[fin[i]]] + 1, out[
                    fin[i]]));
154             seg[1].update(1, 1, n, i, i, Max - dis[fin[i]] * 2);
155         }
156         ll lst = 0;
157         for (int i = 1; i <= q; ++i) {
158             int _d; ll _w;
159             scanf("%d%lld", &_d, &_w);
160             _d = (_d + lst) % (n - 1) + 1;
161             _w = (_w + lst) % W;
162             modify(e[_d].v, _w);
163             lst = diameter();
164             printf("%lld\n", lst);
165         }
166     }
167 }hld;
168
169 int main() {
170     while (scanf("%d%d%lld", &n, &q, &W) != EOF) {
171         G.init(n);
172         for (int i = 1; i < n; ++i) {
173             scanf("%d%d%lld", &e[i].u, &e[i].v, &e[i].w);
174             G.addedge(e[i].u, e[i].v, e[i].w);
175             G.addedge(e[i].v, e[i].u, e[i].w);
176         }
177         hld.gao();
178     }
179     return 0;
180 }
```

9.2 树的重心

性质:

- 一个点是树的重心，当且仅当去掉这个点后，剩下的连通块中最大的 *sze* 都不会超过原树大小的一半

CF 686D
题意:
给出一棵 n 个点, 根为 1 的有根树，有 q 次询问，每次询问以 u 为根的子树中的重心编号。
$1 \le n, q \le 3 \cdot 10^5$ 做法:
因为 q 与 n 同阶，不妨预处理出所有子树的答案:
显然叶子节点的重心就是其本身，然后再考虑某个非叶子节点 u 的重心:
u 的重心肯定它最大的儿子 v 的重心和 u 的路径上，所以每次往上走，然后判断就可以了，并且容易发现每个点只会被判断一次，所以总复杂度 $O(n)$
证明:

- 如果 u 的最大的儿子 v 的 *sze* 不超过 u 子树大小的一半，那么 u 就是重心，显然 u 在 u 到 v 重心的路径上

- 否则在其他子树中都不可能存在重心，并且我们发现在父亲方向处加了点，往下走肯定是不优的，肯定要往上走

```
#include <bits/stdc++.h>
using namespace std;
const int N = 3e5 + 10;
int n, q, fa[N], f[N], sze[N], ma_sze[N];
vector <vector<int>> G;

bool ok(int goal, int u) {
    if (ma_sze[goal] * 2 > sze[u] || (sze[u] - sze[goal]) * 2 > sze[u])
        return false;
    return true;
}

void dfs(int u) {
    sze[u] = 1;
    ma_sze[u] = 0;
    int Max = 0, pos = 0;
    for (auto &v : G[u]) {
        dfs(v);
        sze[u] += sze[v];
        ma_sze[u] = max(ma_sze[u], sze[v]);
        if (sze[v] > Max) {
            Max = sze[v];
            pos = f[v];
        }
    }
    if (!pos) f[u] = u;
    else {
        f[u] = pos;
        while (!ok(f[u], u)) f[u] = fa[f[u]];
    }
}

int main() {
    while (scanf("%d%d", &n, &q) != EOF) {
        G.clear(); G.resize(n + 1);
        fa[1] = 1;
        for (int i = 2; i <= n; ++i) {
            scanf("%d", fa + i);
            G[fa[i]].push_back(i);
        }
        dfs(1);
        for (int i = 1, x; i <= q; ++i) {
            scanf("%d", &x);
            printf("%d\n", f[x]);
        }
    }
    return 0;
}
```

9.3 树的直径

ZOJ 3820
题意:
在一棵树上放置两点，定义其他点的权值为到这两点的距离的小的那个
问选择哪两点，使得最大的其他点的权值最小

思路:
考虑只放一个点的情况，那么显然是直径上的中点
再考虑放两个点，那肯定以直径中点为界将树分成两半，一边放一个
那么各自取各自直径的中点即可

```
#include <bits/stdc++.h>
using namespace std;
const int N = 2e5 + 10;
int n;
vector <vector<int>> G;

namespace DiameterTree {
    //获得直径，先以任意点为根，找出离它最远的点
    //再以最远的点为根，找出离它最远的点，这两个点就是直径的两个端点
    int fa[N], dep[N], far;
    void get_far(int st) {
        queue <int> q;
        q.push(st);
        dep[st] = 1;
        fa[st] = -1;
        while (!q.empty()) {
            int u = q.front(); q.pop();
            far = u;
            for (auto v : G[u]) {
                if (v == fa[u]) continue;
                fa[v] = u;
                dep[v] = dep[u] + 1;
                q.push(v);
            }
        }
    }
    //找直径的中点，直接从下端点往上找父亲找len / 2次就好了
    int get_center(int st) {
        get_far(st);
        get_far(far);
        int shift = dep[far] / 2;
        if (dep[far] % 2 == 0) --shift;
        while (shift--)
            far = fa[far];
        return far;
    }
};

int main() {
    int _T; cin >> _T;
    while (_T--) {
        scanf("%d", &n);
        G.clear(); G.resize(n + 1);
        for (int i = 1, u, v; i < n; ++i) {
            scanf("%d%d", &u, &v);
            G[u].push_back(v);
            G[v].push_back(u);
        }
        int center = DiameterTree::get_center(1);
        for (vector <int>::iterator it = G[center].begin(); it != G[center].
           end(); ++it)
            if (*it == DiameterTree::fa[center]) {
                G[center].erase(it);
                break;
```

```
54              }
55          for (vector <int>::iterator it = G[DiameterTree::fa[center]].begin()
                ; it != G[DiameterTree::fa[center]].end(); ++it)
56              if (*it == center) {
57                  G[DiameterTree::fa[center]].erase(it);
58                  break;
59              }
60          int p1, p2, dis;
61          p1 = DiameterTree::get_center(DiameterTree::fa[center]);
62          dis = DiameterTree::dep[p1];
63          p2 = DiameterTree::get_center(center);
64          dis = max(dis, DiameterTree::dep[p2]) - 1;
65          printf("%d %d %d\n", dis, p1, p2);
66      }
67      return 0;
68  }
```

10 猫树

给出一个某种元素的序列 $a_1, a_2, \cdots, a_n$, 进行 m 次询问, 每次询问一段 $[l, r]$ 的某种支持结合律和快速合并的信息，要求在线。

```
1   template <typename CatTreeItem, CatTreeItem base>
2   struct CatTree {
3       int lev, sz;
4       vector <CatTreeItem> v;
5       vector <vector<CatTreeItem>> t;
6       //确立两数运算规则
7       CatTreeItem op(const CatTreeItem &a, const CatTreeItem &b) { return 1ll
            * a * b % p; }
8       int log2Up(int n) { return 31 - __builtin_clz(n); }
9       void build(int id, int l, int r) {
10          if (l == r) return;
11          int mid = (l + r) >> 1;
12          t[id][mid] = v[mid];
13          for (int i = mid - 1; i >= l; --i) t[id][i] = op(t[id][i + 1], v[i])
                ;
14          if (mid < r) {
15              t[id][mid + 1] = v[mid + 1];
16              for (int i = mid + 2; i <= r; ++i) t[id][i] = op(t[id][i - 1], v
                    [i]);
17          }
18          build(id + 1, l, mid);
19          build(id + 1, mid + 1, r);
20      }
21      //下标从0开始
22      CatTree(const vector<CatTreeItem> &a) {
23          int n = a.size();
24          lev = __builtin_clz(n);
25          sz = 1 << (31 - lev);
26          if (n != sz) { sz <<= 1; --lev; }
27          v = vector<CatTreeItem>(sz, base);
28          t = vector <vector<CatTreeItem>>(log2Up(n) + 1, vector <CatTreeItem
                >(sz, base));
29          for (int i = 0; i < n; ++i) v[i] = a[i];
30          build(0, 0, sz - 1);
31      }
```

```
32      CatTreeItem query(int l, int r) {
33          if (l == r) return v[l];
34          int id = __builtin_clz(l ^ r) - lev - 1;
35          return op(t[id][l], t[id][r]);
36      }
37  };
```

11 SqrtTree

给一个长度为 n 的序列 a_i, 再给出一个满足结合律的运算 $\circ$(譬如 gcd, min, max, + 均满足结合律), 每次询问区间 $[l,r]$, 要计算 $\circ_{i=l}^{r} a_i$。

SqrtTree 以在 $O(n \log\log n)$ 时间预处理，$O(1)$ 回答询问 (但是常数很大)，$O(\sqrt{n})$ 单点修改，空间复杂度 $O(n)$。

CodeChef-SEGPROD

```
1   template <class SqrtTreeItem>
2   class SqrtTree {
3   private:
4       int n, lg, indexSz;
5       vector<SqrtTreeItem> v;
6       vector<int> clz, layers, onLayer;
7       vector<vector<SqrtTreeItem> > pref, suf, between;
8       //确立两数运算法则
9       inline SqrtTreeItem op(const SqrtTreeItem &a, const SqrtTreeItem &b) {
           return 1ll * a * b % p; }
10
11      inline int log2Up(int n) {
12          int res = 0;
13          while ((1 << res) < n) {
14              res++;
15          }
16          return res;
17      }
18
19      inline void buildBlock(int layer, int l, int r) {
20          pref[layer][l] = v[l];
21          for (int i = l + 1; i < r; i++) {
22              pref[layer][i] = op(pref[layer][i - 1], v[i]);
23          }
24          suf[layer][r - 1] = v[r - 1];
25          for (int i = r - 2; i >= l; i--) {
26              suf[layer][i] = op(v[i], suf[layer][i + 1]);
27          }
28      }
29
30      inline void buildBetween(int layer, int lBound, int rBound, int
           betweenOffs) {
31          int bSzLog = (layers[layer] + 1) >> 1;
32          int bCntLog = layers[layer] >> 1;
33          int bSz = 1 << bSzLog;
34          int bCnt = (rBound - lBound + bSz - 1) >> bSzLog;
35          for (int i = 0; i < bCnt; i++) {
36              SqrtTreeItem ans;
37              for (int j = i; j < bCnt; j++) {
38                  SqrtTreeItem add = suf[layer][lBound + (j << bSzLog)];
39                  ans = (i == j) ? add : op(ans, add);
40                  between[layer - 1][betweenOffs + lBound + (i << bCntLog) + j
                       ] = ans;
```

```
41              }
42          }
43      }
44
45      inline void buildBetweenZero() {
46          int bSzLog = (lg + 1) >> 1;
47          for (int i = 0; i < indexSz; i++) {
48              v[n + i] = suf[0][i << bSzLog];
49          }
50          build(1, n, n + indexSz, (1 << lg) - n);
51      }
52
53      inline void updateBetweenZero(int bid) {
54          int bSzLog = (lg + 1) >> 1;
55          v[n + bid] = suf[0][bid << bSzLog];
56          update(1, n, n + indexSz, (1 << lg) - n, n + bid);
57      }
58
59      inline void build(int layer, int lBound, int rBound, int betweenOffs) {
60          if (layer >= (int)layers.size()) {
61              return;
62          }
63          int bSz = 1 << ((layers[layer] + 1) >> 1);
64          for (int l = lBound; l < rBound; l += bSz) {
65              int r = min(l + bSz, rBound);
66              buildBlock(layer, l, r);
67              build(layer + 1, l, r, betweenOffs);
68          }
69          if (layer == 0) {
70              buildBetweenZero();
71          } else {
72              buildBetween(layer, lBound, rBound, betweenOffs);
73          }
74      }
75
76      inline void update(int layer, int lBound, int rBound, int betweenOffs,
           int x) {
77          if (layer >= (int)layers.size()) {
78              return;
79          }
80          int bSzLog = (layers[layer] + 1) >> 1;
81          int bSz = 1 << bSzLog;
82          int blockIdx = (x - lBound) >> bSzLog;
83          int l = lBound + (blockIdx << bSzLog);
84          int r = min(l + bSz, rBound);
85          buildBlock(layer, l, r);
86          if (layer == 0) {
87              updateBetweenZero(blockIdx);
88          } else {
89              buildBetween(layer, lBound, rBound, betweenOffs);
90          }
91          update(layer + 1, l, r, betweenOffs, x);
92      }
93
94      inline SqrtTreeItem query(int l, int r, int betweenOffs, int base) {
95          if (l == r) {
96              return v[l];
97          }
```

```
98          if (l + 1 == r) {
99              return op(v[l], v[r]);
100         }
101         int layer = onLayer[clz[(l - base) ^ (r - base)]];
102         int bSzLog = (layers[layer] + 1) >> 1;
103         int bCntLog = layers[layer] >> 1;
104         int lBound = (((l - base) >> layers[layer]) << layers[layer]) + base
                ;
105         int lBlock = ((l - lBound) >> bSzLog) + 1;
106         int rBlock = ((r - lBound) >> bSzLog) - 1;
107         SqrtTreeItem ans = suf[layer][l];
108         if (lBlock <= rBlock) {
109             SqrtTreeItem add =
110                 (layer == 0) ? (query(n + lBlock, n + rBlock, (1 << lg) - n,
                        n))
111                              : (between[layer - 1][betweenOffs + lBound + (
                                    lBlock << bCntLog) + rBlock]);
112             ans = op(ans, add);
113         }
114         ans = op(ans, pref[layer][r]);
115         return ans;
116     }
117
118 public:
119     inline SqrtTreeItem query(int l, int r) { return query(l, r, 0, 0); }
120
121     inline void update(int x, const SqrtTreeItem &item) {
122         v[x] = item;
123         update(0, 0, n, 0, x);
124     }
125
126     //下标从0开始
127     SqrtTree(const vector<SqrtTreeItem> &a)
128         : n((int)a.size()), lg(log2Up(n)), v(a), clz(1 << lg), onLayer(lg +
                1) {
129         clz[0] = 0;
130         for (int i = 1; i < (int)clz.size(); i++) {
131             clz[i] = clz[i >> 1] + 1;
132         }
133         int tlg = lg;
134         while (tlg > 1) {
135             onLayer[tlg] = (int)layers.size();
136             layers.push_back(tlg);
137             tlg = (tlg + 1) >> 1;
138         }
139         for (int i = lg - 1; i >= 0; i--) {
140             onLayer[i] = max(onLayer[i], onLayer[i + 1]);
141         }
142         int betweenLayers = max(0, (int)layers.size() - 1);
143         int bSzLog = (lg + 1) >> 1;
144         int bSz = 1 << bSzLog;
145         indexSz = (n + bSz - 1) >> bSzLog;
146         v.resize(n + indexSz);
147         pref.assign(layers.size(), vector<SqrtTreeItem>(n + indexSz));
148         suf.assign(layers.size(), vector<SqrtTreeItem>(n + indexSz));
149         between.assign(betweenLayers, vector<SqrtTreeItem>((1 << lg) + bSz))
                ;
150         build(0, 0, n, 0);
```

```
    }
};
```

12 主席树

12.1 BZOJ 3524

题意:
给出一个长度为 n 的序列 a_i, 每次询问一个区间 $[l,r]$ 中是否存在一个数出现的次数大于 $\frac{r-l+1}{2}$
如果存在，输出这个数，否则输出 0
做法:
直接判断左子树和是否大于 $r-l+1$ 或者右子树和是否大于即可，这两个条件不可能同时成立。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 5e5 + 10;
int n, q, m, a[N];

struct Hash {
    vector <int> a;
    void init() { a.clear(); }
    void add(int x) { a.push_back(x); }
    void gao() { sort(a.begin(), a.end()); a.erase(unique(a.begin(), a.end()
       ), a.end()); }
    int get(int x) { return lower_bound(a.begin(), a.end(), x) - a.begin() +
        1; }
}hs;

struct SEG {
    struct node {
        int ls, rs, cnt;
        void init() { ls = rs = cnt = 0; }
    }t[N * 40];
    int rt[N], tot;
    void init() { memset(rt, 0, sizeof rt); tot = 0; t[0].init(); }
    int newnode() {
        ++tot;
        t[tot].init();
        return tot;
    }
    void up(int id) {
        int ls = t[id].ls, rs = t[id].rs;
        t[id].cnt = 0;
        if (ls) t[id].cnt += t[ls].cnt;
        if (rs) t[id].cnt += t[rs].cnt;
    }
    void build(int &id, int l, int r) {
        id = newnode();
        if (l == r) return;
        int mid = (l + r) >> 1;
        build(t[id].ls, l, mid);
        build(t[id].rs, mid + 1, r);
    }
    void update(int &now, int pre, int l, int r, int pos, int v) {
        now = newnode();
        t[now] = t[pre];
        if (l == r) {
            t[now].cnt += v;
```

```
44              return;
45          }
46          int mid = (l + r) >> 1;
47          if (pos <= mid) update(t[now].ls, t[pre].ls, l, mid, pos, v);
48          else update(t[now].rs, t[pre].rs, mid + 1, r, pos, v);
49          up(now);
50      }
51      int query(int tl, int tr, int l, int r, int cnt) {
52          if (l == r) return l - 1;
53          int mid = (l + r) >> 1;
54          int lsum = t[t[tr].ls].cnt - t[t[tl].ls].cnt;
55          int rsum = t[t[tr].rs].cnt - t[t[tl].rs].cnt;
56          if (lsum > cnt) {
57              return query(t[tl].ls, t[tr].ls, l, mid, cnt);
58          } else if (rsum > cnt) {
59              return query(t[tl].rs, t[tr].rs, mid + 1, r, cnt);
60          } else {
61              return -1;
62          }
63      }
64  }seg;
65
66  int main() {
67      while (scanf("%d%d", &n, &q) != EOF) {
68          hs.init();
69          for (int i = 1; i <= n; ++i) scanf("%d", a + i), hs.add(a[i]);
70          hs.gao(); m = hs.a.size();
71          for (int i = 1; i <= n; ++i) a[i] = hs.get(a[i]);
72          seg.init(); seg.build(seg.rt[0], 1, m);
73          for (int i = 1; i <= n; ++i) seg.update(seg.rt[i], seg.rt[i - 1], 1,
                m, a[i], 1);
74          for (int i = 1, l, r; i <= q; ++i) {
75              scanf("%d%d", &l, &r);
76              int ans = seg.query(seg.rt[l - 1], seg.rt[r], 1, m, (r - l + 1)
                    >> 1);
77              if (ans == -1) ans = 0;
78              else ans = hs.a[ans];
79              printf("%d\n", ans);
80          }
81      }
82      return 0;
83  }
```

12.2 树状数组套主席树

BZOJ 1901
带修改区间第 k 小的数
时空复杂度 $O(n \log^2 n)$。

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   #define SZ(x) (int(x.size()))
4   const int N = 2e4 + 10;
5   int n, m, q, a[N];
6
7   struct Hash {
8       vector <int> a;
9       int& operator[](int x) { return a[x - 1]; }
```

```
10      int size() { return a.size(); }
11      void init() { a.clear(); }
12      void add(int x) { a.push_back(x); }
13      void gao() { sort(a.begin(), a.end()); a.erase(unique(a.begin(), a.end()
            ), a.end()); }
14      int get(int x) { return lower_bound(a.begin(), a.end(), x) - a.begin() +
            1; }
15  }hs;
16
17  struct SEG {
18      struct node {
19          int ls, rs, sum;
20          void init() { ls = rs = sum = 0; }
21      }t[N * 50];
22      int tot;
23      int newnode() {
24          ++tot;
25          t[tot].init();
26          return tot;
27      }
28      void init() { tot = 0; }
29      void update(int &rt, int l, int r, int pos, int v) {
30          if (!rt) rt = newnode();
31          t[rt].sum += v;
32          if (l == r) return;
33          int mid = (l + r) >> 1;
34          if (pos <= mid) update(t[rt].ls, l, mid, pos, v);
35          else update(t[rt].rs, mid + 1, r, pos, v);
36      }
37      int query(int l, int r, int k, vector<int> &L, vector<int> &R) {
38          if (l == r) return l;
39          int mid = (l + r) >> 1;
40          int lsum = 0, rsum = 0;
41          for (int i = 0; i < SZ(L); ++i) lsum += t[t[L[i]].ls].sum;
42          for (int i = 0; i < SZ(R); ++i) rsum += t[t[R[i]].ls].sum;
43          if (rsum - lsum >= k) {
44              for (int i = 0; i < SZ(L); ++i) L[i] = t[L[i]].ls;
45              for (int i = 0; i < SZ(R); ++i) R[i] = t[R[i]].ls;
46              return query(l, mid, k, L, R);
47          } else {
48              for (int i = 0; i < SZ(L); ++i) L[i] = t[L[i]].rs;
49              for (int i = 0; i < SZ(R); ++i) R[i] = t[R[i]].rs;
50              return query(mid + 1, r, k - (rsum - lsum), L, R);
51          }
52      }
53  }seg;
54
55  struct BIT {
56      int rt[N], n;
57      void init(int _n) { n = _n; memset(rt, 0, sizeof (rt[0]) * (n + 5)); seg
            .init(); }
58      int lowbit(int x) { return x & -x; }
59      void update(int x, int pos, int v) {
60          for (int i = x; i <= n; i += lowbit(i))
61              seg.update(rt[i], 1, m, pos, v);
62      }
63      vector <int> get(int x) {
64          vector <int> vec;
```

```
65          for (int i = x; i; i -= lowbit(i))
66              vec.push_back(rt[i]);
67          return vec;
68      }
69      int query(int l, int r, int k) {
70          vector <int> L(get(l - 1)), R(get(r));
71          return seg.query(1, m, k, L, R);
72      }
73  }bit;
74
75  struct E {
76      int op, x, y, z;
77      E() {}
78      E(int op, int x, int y, int z) : op(op), x(x), y(y), z(z) {}
79  }e[N];
80
81  int main() {
82      while (scanf("%d%d", &n, &q) != EOF) {
83          hs.init();
84          for (int i = 1; i <= n; ++i) {
85              scanf("%d", a + i);
86              hs.add(a[i]);
87          }
88          scanf("%d", &q);
89          for (int i = 1, x, y, z; i <= q; ++i) {
90              static char op[10];
91              scanf("%s", op);
92              scanf("%d%d", &x, &y);
93              if (op[0] == 'C') {
94                  e[i] = E(1, x, y, 0);
95                  hs.add(y);
96              } else if (op[0] == 'Q') {
97                  scanf("%d", &z);
98                  e[i] = E(2, x, y, z);
99              }
100         }
101         hs.gao(); m = hs.size();
102         for (int i = 1; i <= n; ++i) a[i] = hs.get(a[i]);
103         for (int i = 1; i <= q; ++i) if (e[i].op == 1) e[i].y = hs.get(e[i].
                y);
104         bit.init(n);
105         for (int i = 1; i <= n; ++i) {
106             bit.update(i, a[i], 1);
107         }
108         for (int i = 1; i <= q; ++i) {
109             if (e[i].op == 2) {
110                 printf("%d\n", hs[bit.query(e[i].x, e[i].y, e[i].z)]);
111             } else {
112                 bit.update(e[i].x, a[e[i].x], -1);
113                 a[e[i].x] = e[i].y;
114                 bit.update(e[i].x, a[e[i].x], 1);
115             }
116         }
117     }
118     return 0;
119 }
```

12.3 树上主席树

Luogu3302
题意:
支持两种操作:

- 查询 (x,y) 路径上第 k 小的权值
- 增加一条边 (x,y)，保证这条边之前不存在

强制在线
思路:
裸的树上主席树，对于加边，看作两颗树的合并，直接启发式暴力合并即可。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 8e4 + 10, M = 17;
int n, m, ce, q, a[N], fa[N][M], deep[N], lst;
vector <vector<int>> G;

struct Hash {
    vector <int> a;
    void init() { a.clear(); }
    void add(int x) { a.push_back(x); }
    void gao() { sort(a.begin(), a.end()); a.erase(unique(a.begin(), a.end()), a.end()); }
    int get(int x) { return lower_bound(a.begin(), a.end(), x) - a.begin() + 1; }
}hs;

struct UFS {
    int fa[N], sze[N];
    void init(int n) { for (int i = 0; i <= n; ++i) fa[i] = i, sze[i] = 1; }
    int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }
    void merge(int x, int y) {
        x = find(x); y = find(y);
        if (x != y) {
            if (sze[x] > sze[y]) swap(x, y);
            fa[x] = y;
            sze[y] += sze[x];
        }
    }
}ufs;

struct SEG {
    struct node {
        int ls, rs, sum;
        void init() { ls = rs = sum = 0; }
    }t[N * 300];
    int rt[N], tot;
    void init() { memset(rt, 0, sizeof rt); tot = 0; t[0].init(); }
    int newnode() {
        ++tot;
        t[tot].init();
        return tot;
    }
    void build(int &id, int l, int r) {
        id = newnode();
        if (l == r) return;
        int mid = (l + r) >> 1;
```

```
45          build(t[id].ls, l, mid);
46          build(t[id].rs, mid + 1, r);
47      }
48      void update(int &now, int pre, int l, int r, int pos, int v) {
49          now = newnode();
50          t[now] = t[pre];
51          t[now].sum += v;
52          if (l == r) return;
53          int mid = (l + r) >> 1;
54          if (pos <= mid) update(t[now].ls, t[pre].ls, l, mid, pos, v);
55          else update(t[now].rs, t[pre].rs, mid + 1, r, pos, v);
56      }
57      int query(int x, int y, int lca, int falca, int l, int r, int k) {
58          if (l == r) return l;
59          int lsum = t[t[x].ls].sum + t[t[y].ls].sum - t[t[lca].ls].sum - t[t[
                falca].ls].sum;
60          int mid = (l + r) >> 1;
61          if (lsum >= k) {
62              return query(t[x].ls, t[y].ls, t[lca].ls, t[falca].ls, l, mid, k
                    );
63          } else {
64              return query(t[x].rs, t[y].rs, t[lca].rs, t[falca].rs, mid + 1,
                    r, k - lsum);
65          }
66      }
67  }seg;
68
69  void dfs(int u) {
70      for (int i = 1; i < M; ++i) {
71          fa[u][i] = fa[fa[u][i - 1]][i - 1];
72      }
73      for (auto &v : G[u]) if (v != fa[u][0]) {
74          deep[v] = deep[u] + 1;
75          fa[v][0] = u;
76          seg.update(seg.rt[v], seg.rt[u], 1, m, a[v], 1);
77          dfs(v);
78      }
79  }
80
81  int querylca(int u, int v) {
82      if (deep[u] < deep[v]) swap(u, v);
83      int deg = deep[u] - deep[v];
84      for (int i = 0; i < M; ++i) {
85          if ((deg >> i) & 1) {
86              u = fa[u][i];
87          }
88      }
89      if (u == v) return u;
90      for (int i = M - 1; i >= 0; --i) {
91          if (fa[u][i] != fa[v][i]) {
92              u = fa[u][i];
93              v = fa[v][i];
94          }
95      }
96      return fa[u][0];
97  }
98
99  int main() {
```

```
100     int _T; scanf("%d", &_T);
101     scanf("%d%d%d", &n, &ce, &q);
102     G.clear(); G.resize(n + 1); ufs.init(n);
103     hs.init(); lst = 0;
104     for (int i = 1; i <= n; ++i) {
105         scanf("%d", a + i);
106         hs.add(a[i]);
107     }
108     hs.gao(); m = hs.a.size();
109     for (int i = 1; i <= n; ++i) a[i] = hs.get(a[i]);
110     for (int i = 1, u, v; i <= ce; ++i) {
111         scanf("%d%d", &u, &v);
112         G[u].push_back(v);
113         G[v].push_back(u);
114         ufs.merge(u, v);
115     }
116     seg.init(); seg.build(seg.rt[0], 1, m);
117     memset(deep, 0, sizeof deep);
118     for (int i = 1; i <= n; ++i) {
119         if (deep[i] == 0) {
120             deep[i] = 1;
121             fa[i][0] = 0;
122             seg.update(seg.rt[i], seg.rt[0], 1, m, a[i], 1);
123             dfs(i);
124         }
125     }
126     char op[10]; int x, y, z;
127     for (int i = 1; i <= q; ++i) {
128         scanf("%s", op);
129         if (*op == 'Q') {
130             scanf("%d%d%d", &x, &y, &z);
131             x ^= lst; y ^= lst; z ^= lst;
132             int lca = querylca(x, y);
133             lst = hs.a[seg.query(seg.rt[x], seg.rt[y], seg.rt[lca], seg.rt[
                    fa[lca][0]], 1, m, z) - 1];
134             printf("%d\n", lst);
135         } else {
136             scanf("%d%d", &x, &y);
137             x ^= lst; y ^= lst;
138             G[x].push_back(y);
139             G[y].push_back(x);
140             if (ufs.sze[ufs.find(x)] > ufs.sze[ufs.find(y)]) swap(x, y);
141             ufs.merge(x, y);
142             seg.update(seg.rt[x], seg.rt[y], 1, m, a[x], 1);
143             deep[x] = deep[y] + 1;
144             fa[x][0] = y;
145             dfs(x);
146         }
147     }
148     return 0;
149 }
```

13 本质不同子序列

13.1 牛客 4853C

题意：求长度为 k 的本质不同的子序列个数。

考虑 $f_{i,j}$ 表示前 i 个字符，长度为 j 的本质不同的子序列个数。
如果当前字符第一次出现，那么有 $f_{i,j} = f_{i-1,j} + f_{i-1,j-1}$。
否则，令 pre 为上一次出现的位置，转移有：$f_{i,j} = f_{i-1,j} + f_{i-1,j-1} - f_{pre-1,j-1}$。
时间复杂度 $O(nk)$。

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
constexpr int mod = 1e9 + 7;
constexpr int N = 1e3 + 10;
int n, K, pre[330]; ll f[N][N]; char s[N];

int main() {
    scanf("%d%d", &n, &K);
    scanf("%s", s + 1);
    memset(f, 0, sizeof f);
    memset(pre, -1, sizeof pre);
    f[0][0] = 1;
    for (int i = 1; i <= n; ++i) {
        int ch = s[i] - 'a';
        for (int j = 0; j < i; ++j) {
            if (pre[ch] == -1) {
                (f[i][j + 1] += f[i - 1][j + 1] + f[i - 1][j]) %= mod;
            } else {
                (f[i][j + 1] += f[i - 1][j + 1] + f[i - 1][j] - f[pre[ch] -
                   1][j] + mod) %= mod;
            }
        }
        f[i][0] = 1;
        pre[ch] = i;
    }
    printf("%lld\n", f[n][K]);
    return 0;
}
```

13.2 CCCC L3-020

题意：
给出一个字符串，至多删除 3 个字符，有多少种本质不同的子序列。
思路：
考虑 $f[i][j]$ 表示前 i 个字符，删去 j 个字符的本质不同子序列个数。
有转移方程：$f[i][j] = f[i-1][j] + f[i-1][j-1]$
考虑重复情况：比如 *aabab*，当递推到第 5 个位置的时候，删去第 3 位和第 4 位的 *ba*，但是保留着第 5 位的 *b*，和递推到第 3 个位置时，保留着第三位的 *b* 的情况是一样的，所以要减去。即 $f[i][j] -= f[d-1][j-(i-d)]$。
时间复杂度 $O(n)$。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e6 + 10;
int n, pos[220]; ll f[N][4]; char s[N];

int main() {
    scanf("%s", s + 1);
    n = strlen(s + 1);
    memset(pos, -1, sizeof pos);
    f[0][0] = 1;
    for (int i = 1; i <= n; ++i) {
```

```
        int d = pos[s[i]];
        pos[s[i]] = i;
        f[i][0] = f[i][1] = 1;
        for (int j = 1; j < 4; ++j) {
            f[i][j] += f[i - 1][j];
            f[i][j + 1] = f[i - 1][j];
            if (d != -1 && j - i + d >= 0)
                f[i][j] -= f[d - 1][j - (i - d)];
        }
    }
    printf("%lld\n", f[n][0] + f[n][1] + f[n][2] + f[n][3]);
    return 0;
}
```

14 KMP

14.1 CF 291E

题意:

给出一棵压缩字母树，问存在多少对 (u,v) 使得 u 是 v 的祖先，并且 $u \to v$ 的路径上组成的字符串能够跟给出的字符串 t 匹配上。

字符串总长小于等于 $3 \cdot 10^5$

思路:

考虑 *kmp* 匹配过程中跳 *fail* 过程的复杂度是均摊 $O(n+m)$，那么在树上跑 *kmp* 就不能暴力跑，我们可以优化以下跳 *fail* 的过程，发现跳 *fail* 的终点和当前匹配的字符有关，可以提前预处理好，一步跳到位。

```
#include <bits/stdc++.h>
#define SZ(x) (int(x.size()))
using namespace std;
typedef long long ll;
constexpr int N = 3e5 + 10;
int n, fa[N]; string s[N], t; ll res;
vector <vector<int>> G;

struct KMP {
    int Next[N], nx[N][27], len;
    void get_Next(string &s) {
        len = SZ(s);
        int i, j;
        j = Next[0] = -1;
        i = 0;
        while (i < len) {
            while (-1 != j && s[i] != s[j]) j = Next[j];
            Next[++i] = ++j;
        }
        for (int i = 0; i < len; ++i) {
            for (int j = 0; j < 26; ++j) {
                nx[i][j] = i;
                if (nx[i][j] != -1 && s[nx[i][j]] != 'a' + j) {
                    nx[i][j] = Next[nx[i][j]];
                    if (nx[i][j] != -1 && s[nx[i][j]] != 'a' + j)
                        nx[i][j] = nx[nx[i][j]][j];
                }
            }
        }
    }
    int Match(int &pos, string &s) {
        int cnt = 0;
```

```
        for (int i = 0; i < SZ(s); ++i) {
            int ch = s[i] - 'a';
            if (pos != -1) pos = nx[pos][ch];
            ++pos;
            if (pos == len) {
                ++cnt;
                pos = Next[pos];
            }
        }
        return cnt;
    }
}kmp;

void dfs(int u, int kmpPos) {
    for (auto &v : G[u]) {
        int pos = kmpPos;
        res += kmp.Match(pos, s[v]);
        dfs(v, pos);
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr); cout.tie(nullptr);
    cin >> n;
    G.clear(); G.resize(n + 1);
    for (int i = 2; i <= n; ++i) {
        cin >> fa[i] >> s[i];
        G[fa[i]].push_back(i);
    }
    cin >> t;
    kmp.get_Next(t);
    res = 0;
    dfs(1, 0);
    cout << res << "\n";
    return 0;
}
```

15 失配树

一个字符串 s，定义他的 k 前缀 pre_k 为 $s_{1\cdots k}$，定义它的 k 后缀 suf_k 为 $s_{|s|-k+1\cdots|s|}$，其中 $1 \le k \le |s|$。

定义 $Border(s)$ 为对于 $i \in [1,|s|)$, 满足 $pre_i = suf_i$ 的字符串 pre_i 的集合。$Border(s)$ 中的每个元素都称为字符串 s 的 *border*。

border 的性质：

- $Border(s)$ 中的所有元素都小于 s
- $Border(s)$ 中任意一个元素的 *border* 都是 $Border(s)$ 的一个子集
- 设 f_s 为 $Border(s)$ 中最大的元素, 那么有递推式 $Border(s) = Border(f_s) \bigcup \{f_s\}$

洛谷 P5829

给出一个字符串 s，有 q 次询问，每次询问给出 p 和 q，询问前缀 p 和前缀 q 的最长公共 *border* 的长度。

思路：

根据 *border* 的性质，两个前缀的公共 *border* 肯定直接沿着 *Next* 数组往上跳跳到一个相同的前缀即可，这个等价于树上找 *lca* 的过程，建树求 *lca* 即可，注意自己不是自己的 *border*。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e6 + 5;
int q; char s[N];

struct Graph {
    struct E { int to, nx, w; }e[N]; int h[N], cnt;
    void init(int n) { for (int i = 0; i <= n; ++i) h[i] = -1; cnt = -1; }
    void addedge(int u, int v, int w = 0) { e[++cnt] = { v, h[u], w}; h[u] =
        cnt; }
}G;

struct BorderTree {
    int Next[N], sze[N], top[N], son[N], deep[N], len;
    //0-index
    void get_Next(char *s) {
        int i, j;
        j = Next[0] = -1;
        i = 0;
        while (i < len) {
            while (-1 != j && s[i] != s[j]) j = Next[j];
            Next[++i] = ++j;
        }
    }
    void dfs(int u) {
        son[u] = 0; sze[u] = 1;
        for (int i = G.h[u]; ~i; i = G.e[i].nx) {
            int v = G.e[i].to;
            deep[v] = deep[u] + 1;
            dfs(v);
            sze[u] += sze[v];
            if (!son[u] || sze[v] > sze[son[u]]) son[u] = v;
        }
    }
    void gettop(int u, int tp) {
        top[u] = tp;
        if (son[u]) gettop(son[u], tp);
        for (int i = G.h[u]; ~i; i = G.e[i].nx) {
            int v = G.e[i].to;
            if (v == son[u]) continue;
            gettop(v, v);
        }
    }
    int query(int u, int v) {
        u = Next[u], v = Next[v];
        while (top[u] != top[v]) {
            if (deep[top[u]] < deep[top[v]]) swap(u, v);
            u = Next[top[u]];
        }
        if (deep[u] > deep[v]) swap(u, v);
        return u;
    }
    void build(char *s) {
        len = strlen(s);
        get_Next(s);
        G.init(len);
        for (int i = 1; i <= len; ++i) G.addedge(Next[i], i);
        Next[0] = 0;
```

```
58          deep[0] = 0;
59          dfs(0);
60          gettop(0, 0);
61      }
62  }borderTree;
63
64  int main() {
65      scanf("%s", s);
66      borderTree.build(s);
67      scanf("%d", &q);
68      while (q--) {
69          int u, v;
70          scanf("%d%d", &u, &v);
71          printf("%d\n", borderTree.query(u, v));
72      }
73      return 0;
74  }
```

16 AC 自动机

16.1 牛客 4010K

题意:

给出 n 个模式串 s_i, 每个模式串的代价为 p_i, 现在要求用这些模式串恰好组成一个文本串，使得代价和最小。

模式串总长 $\leq 5 \cdot 10^5$。

文本串长度 $\leq 5 \cdot 10^5$。

思路:

考虑模式串的 *len* 的种类是根号级别的，那么文本串中每个位置最多匹配上根号级别个模式串，暴力 dp 即可。

这里如果用结构体存 Trie 树上的节点可能会 *TLE*，因为暴力跳 $fail$ 进行 dp 转移的过程，需要访问很多次 $fail$ 数组，用数组会进 *cache*，加快访问速度。

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   using ll = long long;
4   const int N = 1e6 + 10, ALP = 26, INF = 0x3f3f3f3f;
5   const ll INFLL = 0x3f3f3f3f3f3f3f3f;
6   char s[N]; int n;
7
8   struct ACAM {
9       int nx[N][ALP];
10      int fail[N];
11      int len[N], val[N]; ll f[N];
12      int root, tot;
13      int que[N], ql, qr;
14      //节点从1开始
15      int newnode() {
16          ++tot;
17          memset(nx[tot], -1, sizeof nx[tot]);
18          len[tot] = 0; val[tot] = INF;
19          return tot;
20      }
21      void init() { tot = 0; root = newnode(); }
22      void insert(char *s, int _val) {
23          int now = root;
24          for (int i = 0; s[i]; ++i) {
```

```
25              if (nx[now][s[i] - 'a'] == -1)
26                  nx[now][s[i] - 'a'] = newnode();
27              now = nx[now][s[i] - 'a'];
28              len[now] = i + 1;
29          }
30          val[now] = min(val[now], _val);
31      }
32      void build() {
33          ql = 1, qr = 0;
34          for (int i = 0; i < ALP; ++i) {
35              if (nx[root][i] == -1) {
36                  nx[root][i] = root;
37              } else {
38                  fail[nx[root][i]] = root;
39                  que[++qr] = nx[root][i];
40              }
41          }
42          while (ql <= qr) {
43              int now = que[ql++];
44              for (int i = 0; i < ALP; ++i) {
45                  if (nx[now][i] == -1) {
46                      nx[now][i] = nx[fail[now]][i];
47                  } else {
48                      fail[nx[now][i]] = nx[fail[now]][i];
49                      que[++qr] = nx[now][i];
50                  }
51              }
52          }
53          //剪枝 缩减跳fail过程中的无效点
54          vector <int> vec;
55          for (int i = 1; i <= tot; ++i) vec.push_back(i);
56          sort(vec.begin(), vec.end(), [&](int x, int y) { return len[x] < len
                [y]; });
57          for (auto &it : vec) {
58              while (fail[it] != root) {
59                  if (val[fail[it]] != INF) break;
60                  fail[it] = fail[fail[it]];
61              }
62          }
63      }
64      ll query(char *s) {
65          int n = strlen(s + 1);
66          int now = root;
67          memset(f, 0x3f, sizeof f);
68          f[0] = 0;
69          for (int i = 1; i <= n; ++i) {
70              now = nx[now][s[i] - 'a'];
71              for (int j = now; j != root; j = fail[j]) {
72                  if (val[j] != INF) {
73                      f[i] = min(f[i], f[i - len[j]] + val[j]);
74                  }
75              }
76          }
77          return f[n] == INFLL ? -1 : f[n];
78      }
79  }acam;
80
81  int main() {
```

```
    scanf("%d", &n);
    acam.init();
    for (int i = 1, x; i <= n; ++i) {
        scanf("%s%d", s, &x);
        acam.insert(s, x);
    }
    acam.build();
    scanf("%s", s + 1);
    printf("%lld\n", acam.query(s));
    return 0;
}
```

17 FFT 匹配字符串

BZOJ 4259
题意:
给出两个字符串 S 和 T，其中有的字符为'*'，表示该字符可以匹配任意字符。
输出 S 在 T 中所有完全匹配位置的开头的下标。
思路:
考虑令'*' 位置为 0，那么式子 $\sum_{i=0}^{n-1}(a_i - b_j)^2 \cdot a_i \cdot b_j = 0$ 成立的 j 即为答案。
拆式子后为: $a_i^3 b_j + a_i b_j^3 - 2a_i^2 b_j^2$。
然后将 S 串翻转，发现该式子的下标之和为定值，是一个卷积形式，做三遍卷积即可。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;
const db PI = acos(-1.0);
const db eps = 1e-8;
const int N = 1e6 + 10;
char s[N], t[N];
int n, m, a[N], b[N]; db f[N];

//此处省略FFT模板

int main() {
    while (scanf("%d%d", &n, &m) != EOF) {
        scanf("%s%s", s, t);
        reverse(s, s + n);
        for (int i = 0; i < n; ++i) {
            if (s[i] == '*') a[i] = 0;
            else a[i] = s[i] - 'a' + 1;
        }
        for (int i = 0; i < m; ++i) {
            if (t[i] == '*') b[i] = 0;
            else b[i] = t[i] - 'a' + 1;
        }

        int len = 1;
        while (len < n + m) len <<= 1;

        for (int i = 0; i < len; ++i) x1[i] = x2[i] = Complex(0.0, 0.0);
        for (int i = 0; i < n; ++i) x1[i] = Complex(1ll * a[i] * a[i] * a[i
            ], 0.0);
        for (int i = 0; i < m; ++i) x2[i] = Complex(1ll * b[i], 0.0);
        fft.gao(x1, len, 1);
        fft.gao(x2, len, 1);
```

```
34          for (int i = 0; i < len; ++i) x1[i] = x1[i] * x2[i];
35          fft.gao(x1, len, -1);
36          for (int i = 0; i < m; ++i) f[i] += x1[i].x;
37
38          for (int i = 0; i < len; ++i) x1[i] = x2[i] = Complex(0.0, 0.0);
39          for (int i = 0; i < n; ++i) x1[i] = Complex(1ll * a[i], 0.0);
40          for (int i = 0; i < m; ++i) x2[i] = Complex(1ll * b[i] * b[i] * b[i
                ], 0.0);
41          fft.gao(x1, len, 1);
42          fft.gao(x2, len, 1);
43          for (int i = 0; i < len; ++i) x1[i] = x1[i] * x2[i];
44          fft.gao(x1, len, -1);
45          for (int i = 0; i < m; ++i) f[i] += x1[i].x;
46
47          for (int i = 0; i < len; ++i) x1[i] = x2[i] = Complex(0.0, 0.0);
48          for (int i = 0; i < n; ++i) x1[i] = Complex(1ll * a[i] * a[i], 0.0);
49          for (int i = 0; i < m; ++i) x2[i] = Complex(1ll * b[i] * b[i], 0.0);
50          fft.gao(x1, len, 1);
51          fft.gao(x2, len, 1);
52          for (int i = 0; i < len; ++i) x1[i] = x1[i] * x2[i];
53          fft.gao(x1, len, -1);
54          for (int i = 0; i < m; ++i) f[i] += -2.0 * x1[i].x;
55
56          vector <int> res;
57          for (int i = n - 1; i < m; ++i) if (int(f[i] + 0.2) == 0) res.
                push_back(i - (n - 1) + 1);
58          int sze = res.size();
59          printf("%d\n", sze);
60          for (int i = 0; i < sze; ++i)
61              printf("%d%c", res[i], " \n"[i == sze - 1]);
62      }
63      return 0;
64  }
```

18 Hash

18.1 BZOJ 4084

题意:
有两个字符串集合 S 和 T，S 中所有字符串长度为 n，T 中所有字符串长度为 m，保证 $n+m$ 为偶数。
询问有多少对 (i,j) 使得 S_i 和 T_j 拼接之后得到的字符串满足双旋转性。
双旋转性:
将字符串分成等长的两部分，这两部分循环同构。
思路:
考虑 $Rabin-Karp$:

$$H(s)=\sum_{i=0}^{|S|-1} base^i \cdot s[i]$$

注意到 $n \geq m$，那么拼接之后必然在 S_i 中的末尾 $n-\frac{n+m}{2}$ 的部分是固定拼接在 T_j 前面，并成为分割后的后半部分。

那么直接把 T_j 的 $Hash$ 值丢进 map，然后枚举 S_i 前 $\frac{n+m}{2}$ 部分的循环同构，处理一下 S_i 后半部分与 T_j 的拼接，然后计算答案即可。

如何枚举 S_i 前 $\frac{n+m}{2}$ 部分的循环同构?
考虑一个字符串 $s_1s_2s_3\cdots s_n$，我们要得到 $s_ns_1s_2\cdots s_{n-1}$ 的 $Hash$ 值。
那么就是先将 $H(s)$ 减去 $Base^{i-1}\cdot s_n$，再乘上 $Base$，再加上 s_n 即可。

```cpp
#include <bits/stdc++.h>
using namespace std;

#define ll long long
#define ull unsigned long long
const ull basep = 31;
int n, m, lens, lent;
vector <string> s, t;
vector <ull> Ht;
map <ull, int> used, has;
ull qmod(ull base, int n) {
    ull res = 1;
    while (n) {
        if (n & 1) {
            res = res * base;
        }
        base = base * base;
        n >>= 1;
    }
    return res;
}
void Hash(vector <string> &s, vector <ull> &H, int sze, int len, ull base) {
    H.clear(); H.resize(sze);
    for (int i = 0; i < sze; ++i) {
        H[i] = 0;
        ull tmp = base;
        for (int j = 0; j < len; ++j) {
            H[i] = H[i] + tmp * s[i][j];
            tmp *= basep;
        }
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    while (cin >> n >> m >> lens >> lent) {
        s.clear(); s.resize(n);
        t.clear(); t.resize(m);
        Ht.clear(); Ht.resize(m);
        for (int i = 0; i < n; ++i) cin >> s[i];
        for (int i = 0; i < m; ++i) cin >> t[i];
        Hash(t, Ht, m, lent, qmod(basep, lens - (lens + lent) / 2 + 1));
        has.clear(); for (int i = 0; i < m; ++i) ++has[Ht[i]];
        int need = (lens + lent) / 2;
        ll res = 0;
        ull D = qmod(basep, need);
        for (int i = 0; i < n; ++i) {
            used.clear();
            ull L = 0, R = 0, base;
            base = basep;
            for (int j = need; j < lens; ++j) {
                R = R + base * s[i][j];
                base *= basep;
            }
            base = basep;
            for (int j = 0; j < need; ++j) {
                L = L + base * s[i][j];
```

```
59                  base *= basep;
60              }
61              for (int j = need - 1; j >= 0; --j) {
62                  if (used.find(L) == used.end() && has.find(L - R) != has.end
                        ()) {
63                      used[L] = 1;
64                      res += has[L - R];
65                  }
66                  L -= D * s[i][j];
67                  L *= basep;
68                  L += basep * s[i][j];
69              }
70          }
71          cout << res << "\n";
72      }
73      return 0;
74  }
```

18.2 求两个串的 LCP

BZOJ 4892
考虑:

$$w_{str} = (s_0 p^{n-1} + s_1 p^{n-2} + \cdots + s_{n-1} p^0)$$

那么有:

$$\begin{aligned} w_{pre_{i-1}} &= (s_0 p^{i-1} + s_1 p^{i-2} + \cdots + a_{i-1} p^0) \\ w_{pre_j} &= (s_0 p^j + s_1 p^{j-1} + \cdots + s_j p^0) \end{aligned}$$

所以有:

$$\begin{aligned} w_{str_{i,j}} &= (s_i p^{j-i} + s_{i+1} p^{j-i-1} + \cdots + s_j p^0) \\ &= w_{pre_j} - w_{pre_{i-1}} p^{j-i+1} \end{aligned}$$

```
1   #include <bits/stdc++.h>
2   using namespace std;
3
4   const int N = 1e5 + 10;
5   int n, m;
6   char s[N], t[N];
7
8   typedef unsigned long long ull;
9   struct Hash {
10      static ull base[N];
11      static void init() {
12          base[0] = 1;
13          for (int i = 1; i < N; ++i)
14              base[i] = base[i - 1] * 131;
15      }
16      ull a[N];
17      //最好改成从1开始，因为查询区间Hash值是前缀和形式
18      inline void gao(char *s) {
19          a[0] = s[0];
20          for (int i = 1; s[i]; ++i) {
21              a[i] = a[i - 1] * 131 + s[i];
22          }
23      }
```

```
24      inline ull get(int l, int r) {
25          return a[r] - a[l - 1] * base[r - l + 1];
26      }
27  }hs, ht;
28  ull Hash::base[N] = {0};
29
30  //二分求两个串的lcp
31  inline int gao(int x, int y) {
32      int l = 1, r = min(n - x, m - y), res = 0;
33      while (r - l >= 0) {
34          int mid = (l + r) >> 1;
35          if (hs.get(x, x + mid - 1) == ht.get(y, y + mid - 1)) {
36              res = mid;
37              l = mid + 1;
38          } else {
39              r = mid - 1;
40          }
41      }
42      return res;
43  }
44
45  //倍增求两个串的lcp
46  inline int gao2(int x, int y) {
47      int res = 0;
48      for (int i = 1 << 17; i >= 1; i >>= 1) {
49          if (x + i - 1 < n && y + i - 1 < m && hs.get(x, x + i - 1) == ht.get
                (y, y + i - 1)) {
50              x += i;
51              y += i;
52              res += i;
53          }
54      }
55      return res;
56  }
57
58  //倍增高效求两个串的lcp
59  inline int gao3(int x, int y) {
60      int k = 0, p = 1;
61      while (p != 0) {
62          if (hs.get(x, x + k + p - 1) == ht.get(y, y + k + p - 1)) {
63              k += p;
64              p <<= 1;
65          } else {
66              p >>= 1;
67          }
68          while (x + k + p > n || y + k + p > m) p >>= 1;
69      }
70      return k;
71  }
72
73  inline int ok(int x) {
74      int y = 0;
75      for (int j = 1; j <= 4; ++j) {
76          int lcp = gao3(x, y);
77          x += lcp + 1, y += lcp + 1;
78          if (j == 4) return y > m;
79          if (y >= m) return 1;
80      }
```

```
81      return 0;
82  }
83
84  void gogogo() {
85      scanf("%s%s", s, t);
86      hs.gao(s); ht.gao(t);
87      n = strlen(s), m = strlen(t);
88      if (n < m) return (void) puts("0");
89      int res = 0;
90      for (int i = 0; i + m - 1 < n; ++i) {
91          res += ok(i);
92      }
93      printf("%d\n", res);
94  }
95
96  int main() {
97      Hash::init();
98      int T; scanf("%d", &T);
99      while (T--) gogogo();
100     return 0;
101 }
```

18.3 双模数 Hash

题意:
给出一个字符串 S，然后每次询问给出 k 个 S 的子串。
要求有多少个子集，使得每个串都不是其他串的前缀。
思路:
可以先将所有子串按字典序排序，然后用栈构建出前缀关系的树，然后就是树形 dp。
树中的关系是每个点是它子树中其他任意一点的前缀。
那么 dp 就是当前点不选，那么儿子任选，否则只选当前点。

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   using ll = long long;
4   using pLL = pair <ll, ll>;
5   using P = pair<ll, int>;
6   #define fi first
7   #define se second
8   const int N = 4e5 + 10;
9   int n, m, q, mod, cnt[N], f[N], sta[N]; char s[N];
10  vector <vector<int>> G;
11  struct Hash {
12      constexpr static ll seed[2] = {233, 13331};
13      constexpr static ll mod[2] = {998244353, 1000000007};
14      static ll base[2][N];
15      static void init() {
16          base[0][0] = base[1][0] = 1;
17          for (int i = 1; i < N; ++i) {
18              base[0][i] = base[0][i - 1] * seed[0] % mod[0];
19              base[1][i] = base[1][i - 1] * seed[1] % mod[1];
20          }
21      }
22      ll a[2][N];
23      inline void gao(char *s) {
24          a[0][0] = a[1][0] = 0;
25          for (int i = 1; s[i]; ++i) {
26              a[0][i] = (a[0][i - 1] * seed[0] % mod[0] + s[i]) % mod[0];
```

```
27              a[1][i] = (a[1][i - 1] * seed[1] % mod[1] + s[i]) % mod[1];
28          }
29      }
30      inline pLL get(int l, int r) {
31      //  return (a[0][r] - a[0][l - 1] * base[0][r - l + 1] % mod[0] + mod
            [0]) % mod[0] * mod[0] + (a[1][r] - a[1][l - 1] * base[1][r - l + 1]
            % mod[1] + mod[1]) % mod[1];
32          return pLL((a[0][r] - a[0][l - 1] * base[0][r - l + 1] % mod[0] +
               mod[0]) % mod[0], (a[1][r] - a[1][l - 1] * base[1][r - l + 1] %
               mod[1] + mod[1]) % mod[1]);
33      }
34  }hs;
35  ll Hash::base[2][N] = {{0}, {0}};
36  int querylcp(int a, int b, int c, int d) {
37      int l = 1, r = min(b - a + 1, d - c + 1), res = 0;
38      while (r - l >= 0) {
39          int mid = (l + r) >> 1;
40          if (hs.get(a, a + mid - 1) == hs.get(c, c + mid - 1)) {
41              res = mid;
42              l = mid + 1;
43          } else {
44              r = mid - 1;
45          }
46      }
47      return res;
48  }
49  struct node {
50      int l, r, len;
51      bool operator < (const node &other) const {
52          int lcp = querylcp(l, r, other.l, other.r);
53          if (lcp == len) {
54              if (len == other.len) return 0;
55              return 1;
56          }
57          if (lcp == other.len) return 0;
58          return s[l + lcp] < s[other.l + lcp];
59      }
60  }a[N];
61  bool equal(node &a, node &b) {
62      if (a.len != b.len) return 0;
63      int lcp = querylcp(a.l, a.r, b.l, b.r);
64      return lcp == a.len;
65  }
66  bool isprefix(node &a, node &b) {
67      int lcp = querylcp(a.l, a.r, b.l, b.r);
68      return lcp == a.len;
69  }
70  void dfs(int u) {
71      f[u] = 1;
72      for (auto &v: G[u]) {
73          dfs(v);
74          f[u] = 1ll * f[u] * f[v] % mod;
75      }
76      f[u] = (f[u] + cnt[u]) % mod;
77  }
78  int main() {
79      Hash::init();
80      while (scanf("%d%d", &n, &q) != EOF) {
```

```
        scanf("%s", s + 1);
        hs.gao(s);
        while (q--) {
            scanf("%d%d", &m, &mod);
            for (int i = 1; i <= m; ++i) {
                scanf("%d%d", &a[i].l, &a[i].r);
                a[i].len = a[i].r - a[i].l + 1;
            }
            sort(a + 1, a + 1 + m);
            *sta = 0;
            memset(cnt, 0, sizeof (cnt[0]) * (m + 10));
            cnt[0] = 1;
            G.clear(); G.resize(m + 1);
            for (int i = 1; i <= m; ++i) {
                if (*sta && equal(a[sta[*sta]], a[i])) {
                    ++cnt[sta[*sta]];
                    continue;
                }
                while (*sta && !isprefix(a[sta[*sta]], a[i])) --*sta;
                if (!*sta) G[0].push_back(i);
                else G[sta[*sta]].push_back(i);
                cnt[i] = 1;
                sta[++*sta] = i;
            }
            dfs(0);
            printf("%d\n", ((f[0] - 1) % mod + mod) % mod);
        }
    }
    return 0;
}
```

18.4 动态维护 Hash 值

CometOJ 64D

```
#include <bits/stdc++.h>
#include <bits/extc++.h>
using namespace std;
using namespace __gnu_pbds;
using ll = long long;
using ull = unsigned long long;
const int N = 5e5 + 10;
int n, m, q, rt, a[N], fa[N], deep[N];
vector <vector<int>> G;
cc_hash_table <ull, int> mp;

ull seed = 19260817;
ull base[N];
struct HashNode {
    ull val; int len;
    HashNode() { val = 0; len = 0; }
    HashNode(ull val, int len) : val(val), len(len) {}
    HashNode operator + (const HashNode &other) const {
        HashNode res = HashNode();
        res.len = len + other.len;
        res.val = val * base[other.len] + other.val;
        return res;
    }
}Ha[N];
```

```
25
26 struct SEG {
27     HashNode t[N << 2], res;
28     int done;
29     void build(int id, int l, int r) {
30         t[id] = HashNode(0, 1);
31         if (l == r) return;
32         int mid = (l + r) >> 1;
33         build(id << 1, l, mid);
34         build(id << 1 | 1, mid + 1, r);
35         t[id] = t[id << 1] + t[id << 1 | 1];
36     }
37     void update(int id, int l, int r, int pos, int v) {
38         if (l == r) {
39             t[id] = HashNode(v, 1);
40             return;
41         }
42         int mid = (l + r) >> 1;
43         if (pos <= mid) update(id << 1, l, mid, pos, v);
44         else update(id << 1 | 1, mid + 1, r, pos, v);
45         t[id] = t[id << 1] + t[id << 1 | 1];
46     }
47     void query(int id, int l, int r, int ql, int qr) {
48         if (done) return;
49         if (l >= ql && r <= qr) {
50             if (mp[(res + t[id]).val]) {
51                 res = res + t[id];
52                 return;
53             }
54             if (l == r) done = 1;
55         }
56         if (l == r) return;
57         int mid = (l + r) >> 1;
58         if (ql <= mid) query(id << 1, l, mid, ql, qr);
59         if (qr > mid) query(id << 1 | 1, mid + 1, r, ql, qr);
60     }
61 }seg;
62
63 void dfs(int u) {
64     mp[Ha[u].val] = u;
65     for (int i = 0, sze = G[u].size(); i < sze; ++i) {
66         int rk = i + 1, v = G[u][i];
67         deep[v] = deep[u] + 1;
68         Ha[v] = Ha[u] + HashNode(rk, 1);
69         dfs(v);
70     }
71 }
72
73 int main() {
74     base[0] = 1;
75     for (int i = 1; i < N; ++i) {
76         base[i]= base[i - 1] * seed;
77     }
78     scanf("%d%d%d", &n, &m, &q);
79     rt = 1;
80     G.clear(); G.resize(n + 10);
81     mp.clear();
82     for (int i = 1; i <= n; ++i) {
```

```
83          scanf("%d", fa + i);
84          if (fa[i] == 0) rt = i;
85          else {
86              G[fa[i]].push_back(i);
87          }
88      }
89      deep[rt] = 0;
90      Ha[rt] = HashNode(0, 1);
91      dfs(rt);
92      seg.build(1, 1, m);
93      for (int i = 1; i <= m; ++i) {
94          scanf("%d", a + i);
95          seg.update(1, 1, m, i, a[i]);
96      }
97      for (int _q = 1, op, x, l, r, t, k; _q <= q; ++_q) {
98          scanf("%d", &op);
99          if (op == 1) {
100             scanf("%d%d%d", &x, &l, &r);
101             seg.done = 0;
102             seg.res = Ha[x];
103             seg.query(1, 1, m, l, r);
104             printf("%d\n", mp[seg.res.val]);
105         } else {
106             scanf("%d%d", &t, &k);
107             a[t] = k;
108             seg.update(1, 1, m, t, a[t]);
109         }
110     }
111     return 0;
112 }
```

18.5 一维 Hash

```
1  namespace Hash {
2      ll base[N], seed = 131, mod = 1e9 + 7;
3      void init() {
4          base[0] = 1;
5          for (int i = 1; i < N; ++i)
6              base[i] = (base[i - 1] * 131) % mod;
7      }
8      struct ValNode {
9          ll val; int len;
10         ValNode() { val = 0; len = 0; }
11         ValNode(ll val, int len) : val(val), len(len) {}
12         ValNode operator + (const ValNode &other) const {
13             ValNode res = ValNode();
14             res.len = len + other.len;
15             res.val = val * base[other.len] + other.val;
16             res.val %= mod;
17             return res;
18         }
19     };
20     ValNode one = ValNode(base[1] * '1', 1);
21 }
```

18.6 二维 Hash

```
struct Hash2D { // 1-index
    static const ll px = 131, py = 233, mod = 998244353;
    static ll pwx[N], pwy[N];
    ll hv[N][N];
    static void init() {
        pwx[0] = pwy[0] = 1;
        for (int i = 1; i < N; ++i) {
            pwx[i] = pwx[i - 1] * px % mod;
            pwy[i] = pwy[i - 1] * py % mod;
        }
    }
    void init_hash(int n, int m, int a[][N]) {
        for (int i = 1; i <= n; ++i) {
            ll s = 0;
            for (int j = 1; j <= m; ++j) {
                s = (s * py + a[i][j]) % mod;
                hv[i][j] = (hv[i - 1][j] * px + s) % mod;
            }
        }
    }
    ll get(int x, int y, int dx, int dy) {
        --x; --y;
        ll ret = hv[x + dx][y + dy] + hv[x][y] * pwx[dx] % mod * pwy[dy]
                 - hv[x][y + dy] * pwx[dx] - hv[x + dx][y] * pwy[dy];
        return (ret % mod + mod) % mod;
    }
}ha, hb;
ll Hash2D::pwx[N], Hash2D::pwy[N];
```

19 SOSDP

CF1208F

题意：

给出一个序列 a_i，询问 $a_i \mid (a_j \& a_k)$ 这个式子的最大值，其中 $i < j < k$

思路：

考虑固定 a_i，然后我们只需要关心 a_i 那些二进制位上为 0 的位，从高位到低位确定。

比如说对于第 x 位，那么我们相当于固定了一个前缀，去找 $i < j < k$ 是否存在一个 a_j 以及一个 a_k，它们都有这样的前缀。

那么直接枚举子集标记前缀即可。但是这样复杂度不对。

我们可以倒着来，因为我们发现对于一个 (j,k)，那么肯定希望 j,k 越大越好，那么倒着标记即可，这样标记过了肯定不用再标记了，因为被标记过说明下标肯定大于等于当前的下标。

但是要注意下标等于的情况

```
#include <bits/stdc++.h>
using namespace std;
int n, a[N]; pII id[1 << 22];
void DFS(int mask, int x) {
    if (id[mask].fi != -1) return;
    if (id[mask].se == -1) id[mask].se = x;
    else if (id[mask].se != x) id[mask].fi = x;
    else return;
    for (int i = 21; i >= 0; --i) if ((mask >> i) & 1)
        DFS(mask ^ (1 << i), x);
}
void run() {
    memset(id, -1, sizeof id);
    for (int i = 1; i <= n; ++i) cin >> a[i];
```

```
    for (int i = n; i >= 1; --i) DFS(a[i], i);
    int res = 0;
    for (int i = 1; i <= n; ++i) {
        int now = 0;
        for (int j = 21; j >= 0; --j) if (((a[i] >> j) & 1) == 0) {
            if (id[now ^ (1 << j)].fi > i) {
                now = now ^ (1 << j);
            }
        }
        if (id[now].fi > i) res = max(res, a[i] | now);
    }
    cout << res << endl;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr); cout.tie(nullptr);
    cout << fixed << setprecision(20);
    while (cin >> n) run();
    return 0;
}
```

20 决策单调性

第一种是优化形如：

$$f_i = \min / \max w_{i,j}$$

且对于每一个 i 和它的最优决策点 j 都有单调性的方程，其实就是双指针。

第二种是优化形如：

$$f_i = \min / \max_{j=1}^{i-1} g_j + w_{i,j}$$

且记 f_i 的最优决策点为 p_i(也就是 f_i 从 $g_{p_i} + w_{i,p_i}$ 处转移最优)，若满足 $p_i \le p_{i+1}$，则该方程满足决策单调性。

20.1 二分栈

我们可以把 $g_j + w_{i,j}$ 视为关于 j 的函数。因为决策单调，所以对于栈中的任意相邻两个决策点，我们都可以通过二分找到一个临界值 k，使得序列中在 k 之前的时候，其中一个作为决策转移到 f_k 更优，而 k 以后另一个更优。可以借助函数图像来理解这个过程。

我们需要栈顶为当前的最优解。而如果栈中有不止一个元素，则可能存在一个 i，使得到 i 之后栈里面的决策比栈顶优了。这个时候，如何快速判断并弹掉栈顶呢?

上面提到的这个可二分的性质就派上用场了。对于当前的 j，如果当前栈顶下面与栈顶相邻的决策在 i 之前就比栈顶更优了，就要把栈顶弹掉。

二分栈是在线处理，但是 $w_{i,j}$ 要能够 $O(1)$ 计算，否则复杂度会退化。

BZOJ 4709

题意：

有 n 只贝壳，第 i 只贝壳的大小为 s_i，每次从一端取下一小段连续的贝壳，并选择一种贝壳大小 S，如果这种贝壳在这一小段中的个数为 t，那么获得收益 $S \cdot t^2$。

问如果操作使得收益最大。

显然，肯定是分成若干段，并且每段的开头和结尾的贝壳种类相同，选择的贝壳大小就是开头的贝壳种类。

那么转移方程有 $f_i = \max\{f_j + a_i \cdot (S_i - S_j + 1)^2\}$，其中 S_i 表示前 i 个元素中，a_i 的个数。

那么考虑 S_i 是递增的，并且 $(S_i - S_j + 1)^2$ 是以平方级的速度在增加，钦定 $j < k$，一旦 j 这个决策点超过了 k，之后肯定会一直比 k 要优。

所以可以用单调栈来维护这个东西，并且考虑离栈顶第二个元素比第一个元素优，就弹出第一个元素。这时候栈顶就是最优决策点。

但是还要考虑第二个元素劣与第一个元素，但是第三个元素优于第一个元素，我们考虑 $i < j < k < l$, 如果 i 超过 k 的时间小于 j 超过 k 的时间，那么 i 超过 l 的时间肯定小于 j 超过 l 的时间，所以这个时候 j 就可以直接弹栈。

```
#include <bits/stdc++.h>
#define SZ(x) (int(x.size()))
using namespace std;
typedef long long ll;
const int N = 1e5 + 10, M = 1e4 + 10;
int n, a[N], S[N], cnt[M];
ll f[N];

namespace DP {
    vector <int> sta[M];
    ll calc(int x, int y) { return f[x - 1] + 1ll * a[x] * y * y; }
    //x < y 表示需要多少步 x这个决策点就能够比y这个决策点优
    int better(int x, int y) {
        int l = 1, r = n, res = r + 1;
        while (r - l >= 0) {
            int mid = (l + r) >> 1;
            if (calc(x, mid - S[x] + 1) >= calc(y, mid - S[y] + 1)) {
                res = mid;
                r = mid - 1;
            } else {
                l = mid + 1;
            }
        }
        return res;
    }
    void gao() {
        for (int i = 1; i <= n; ++i) {
            int x = a[i];
            while (SZ(sta[x]) >= 2 && better(sta[x].end()[-2], sta[x].end()
               [-1]) <= better(sta[x].end()[-1], i)) sta[x].pop_back();
            sta[x].push_back(i);
            while (SZ(sta[x]) >= 2 && better(sta[x].end()[-2], sta[x].end()
               [-1]) <= S[i]) sta[x].pop_back();
            f[i] = calc(sta[x].end()[-1], S[i] - S[sta[x].end()[-1]] + 1);
        }
        printf("%lld\n", f[n]);
    }
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) scanf("%d", a + i), S[i] = ++cnt[a[i]];
    DP::gao();
    return 0;
}
```

20.2 二分队列

队头为当前最优决策点，维护队列中决策点的 *bound*，即决策边界点，当前的 i 超过这个边界点时即可以弹出队头。

每次要将当前 i 入队的时候，可以比较一下倒数队列中队尾倒数第二个元素的边界点在 $que[tail]$ 和 i 哪个更优，如果在 i 更优，那么 $que[tail]$ 可以直接出队，它不会做出贡献。

使用单调队列处理的问题一般是对于 $i > j$, 其决策点的关系 $p_i \geq p_j$。

Luogu P3515

题意:

给出一个长度为 n 的序列 a_i, 对于每个 $i \in [1,n]$, 求出最小的非负整数 p, 使得 $\forall j \in [1,n]$, 都有 $a_i \le a_j + p - \sqrt{|i-j|}$。

$$p_i \ge a_j - a_i + \sqrt{|i-j|}$$
$$p_i = \max_{j=1}^{n}\{a_j + \sqrt{|i-j|}\} - a_i$$

拆绝对值，有:

$$p_i = \max(\max_{j=1}^{i}\{a_j + \sqrt{i-j}\}, \max_{j=i}^{n}\{a_j + \sqrt{j-i}\}) - a_i$$

两部分为同一形式，分别做两遍即可。

因为 $f(x) = x$ 的增速大于 $f(x) = \sqrt{x}$，所以该式子具有决策单调性。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 5e5 + 10;
int n, a[N];
double ans[N];

namespace DP {
    int bound[N], que[N], head, tail;
    double sqr[N];
    void init() {
        for (int i = 0; i < N; ++i) sqr[i] = sqrt(i);
    }
    inline double calc(int i, int j) { return 1. * a[j] + sqr[i - j]; }
    int getBound(int x, int y) {
        int l = y, r = bound[x] == -1 ? n : bound[x];
        int res = r + 1;
        while (r - l >= 0) {
            int mid = (l + r) >> 1;
            if (calc(mid, x) <= calc(mid, y)) {
                res = mid;
                r = mid - 1;
            } else {
                l = mid + 1;
            }
        }
        return res;
    }
    void gao() {
        memset(bound, -1, sizeof bound);
        head = 1, tail = 0;
        for (int i = 1; i <= n; ++i) {
            while (head < tail && calc(bound[tail - 1], que[tail]) < calc(
               bound[tail - 1], i)) --tail;
            bound[tail] = getBound(que[tail], i); que[++tail] = i;
            while (head < tail && bound[head] <= i) ++head;
            ans[i] = max(ans[i], calc(i, que[head]));
        }
    }
}

int main() {
    DP::init();
```

```
    while (scanf("%d", &n) != EOF) {
        for (int i = 1; i <= n; ++i) scanf("%d", a + i);
        memset(ans, 0, sizeof ans);
        DP::gao();
        reverse(a + 1, a + 1 + n);
        reverse(ans + 1, ans + 1 + n);
        DP::gao();
        for (int i = n; i >= 1; --i)
            printf("%d\n", max((int)ceil(ans[i]) - a[i], 0));
    }
    return 0;
}
```

20.3 分治

对于转移过程是单调并且离线的，我们考虑分治。假设当前我们求解一段区间 $f_{l,r}$，而所有 $f_{l,r}$ 的最优决策点在 $[L,R]$ 之间。

对于 $[L,R]$ 的中点 mid，我们可以暴力扫一遍 $L-\min(mid,R)$，找到它的最优决策点 k。

因为决策单调，所以 $f_{l,mid-1}$ 的决策落在 $[L,k]$ 上，而 $f_{mid+1,r}$ 的决策落在 $[k,R]$ 上，变成了两个规模减半的小问题。

时间复杂度 $O(n\log n)$

CF 868F

题意：

定义一段数的代价为无序对 (i,j) 的数量，其中 (i,j) 满足 $a_i=a_j$。

现在给出 $n(1\le n\le 10^5)$ 个数 a_i，恰好分成 $k(1\le k\le 20)$ 段，使得每段代价之和最少。

思路：

注意到分的段数越多答案越小，所以可以看成最多分 k 段。

设 $f_{i,j}$ 为前 i 个数分成 j 段的最小花费，$w_{l,r}$ 为 $[l,r]$ 全在一段的费用。

有：

$$f_{i,j}=\min_{k=1}^{i}\{f_{k,j-1}+w_{k+1,i}\}$$

考虑 $w_{l,r}$ 可以 $O(1)$ 转移到 $w_{l-1,r},w_{l+1,r},w_{l,r-1},w_{l,r+1}$。

那么可以开个桶记录一下每种数出现的个数。

求解区间：| ← 预处理 → $|l\xrightarrow{\downarrow mid}r$

决策区间：$|L\xrightarrow{\downarrow k}R|$

为了保证分治的复杂度，当进入求解区间时，应该要确保 $[L,l-1]$ 的信息存在。

考虑进入子问题前，先预处理出子问题区间所需的贡献：

- 对于左边的子问题 $([l,mid-1],[L,k])$，和当前区间的 $[L,l-1]$ 是一致的，修改后还原回去即可。
- 对于右边的子问题 $([mid+1,r],[k,R])$，它要预处理的是 $[k,mid]$，当前是 $[L,l-1]$，先把右端点从 $l-1$ 移到 mid，再将左端点从 k 移到 $L-1$，然后递归下去处理，回溯上来的时候记得还原成 $[L,l-1]$ 即可。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5 + 10;
int n, k, a[N], w[N], vis[N];
ll tt[2][N], *f = tt[0], *g = tt[1];
void gao(int l, int r, int kl, int kr, ll W) {
    if (l > r) return;
    int mid = (l + r) >> 1;
    int p = mid < kr ? mid : kr; int k = kr;
    for (int i = l; i <= mid; ++i) W += w[a[i]]++;
    for (int i = kl; i <= p; ++i) {
```

```
13          W -= --w[a[i]];
14          if (f[mid] > W + g[i]) {
15              f[mid] = W + g[i];
16              k = i;
17          }
18      }
19      vis[mid] = k;
20      for (int i = kl; i <= p; ++i) W += w[a[i]]++;
21      for (int i = l; i <= mid; ++i) W -= --w[a[i]];
22      gao(l, mid - 1, kl, k, W);
23      for (int i = l; i <= mid; ++i) W += w[a[i]]++;
24      for (int i = kl; i < k; ++i) W -= --w[a[i]];
25      gao(mid + 1, r, k, kr, W);
26      for (int i = kl; i < k; ++i) W += w[a[i]]++;
27      for (int i = l; i <= mid; ++i) W -= --w[a[i]];
28  }
29
30  int main() {
31      while (scanf("%d%d", &n, &k) != EOF) {
32          for (int i = 1; i <= n; ++i) scanf("%d", a + i);
33          memset(w, 0, sizeof w);
34          g[0] = 0;
35          for (int i = 1; i <= n; ++i) {
36              g[i] = g[i - 1] + w[a[i]];
37              ++w[a[i]];
38          }
39          memset(w, 0, sizeof w);
40          while (--k) {
41              memset(f, 0x3f, sizeof tt[0]);
42              gao(1, n, 1, n, 0);
43              swap(f, g);
44          }
45          printf("%lld\n", g[n]);
46      }
47      return 0;
48  }
```

21 斜率优化

对于决策单调性的式子，考虑转移方程 $f_i = \max / \min_{j=1}^{i-1} g_j + w_{j+1,i}$, 考虑两个决策 $k < j$, 如果 j 比 k, 那么满足 $g_j + w_{j+1,i} \geq g_k + w_{k+1,i}$。

将 w 展开, 如果式子能化成 $\frac{y_j - y_k}{x_j - x_k} \leq k_i$ 的形式，那么就能够进行斜率优化。

- 当不等式为小于等于号，且不等式右边 k_i 单调递增时，单调队列维护斜率递增的上凸包即可。
- 当不等式为大于等于号，且不等式右边 k_i 单调递减时，单调队列维护斜率递减的下凸包即可。

还有一种是将每个决策函数看成一条直线，然后维护直线与直线的交点，求出每条直线的优势范围，如果每次询问的 x 也是单调的，那么也可以单调维护询问，但是这样需要满足斜率和询问都是单调的，才是单调的，如果询问的 x 不是单调的，直接凸优化即可。

21.1 洛谷 P2900

题意:

有 n 块土地，每块土地的大小为 $w_i \cdot h_i$, 购买一块土地的花费为 $w_i \cdot h_i$, 但是购买一堆土地的花费这堆土地中最大的长乘最大的宽

求将所有土地都买下的最小花费。

思路：

考虑一块土地如果长宽都小于等于另外一块，那么这一块土地肯定能够被并购掉，那么剔除掉这些土地，然后按长从大到小排序，容易发现对应的宽是从小到大的。

那么考虑 f_i 表示前 i 块土地的最小代价，有转移方程：

$$f_i = \min_{j=0}^{i-1}\{f_j + w_{j+1} \cdot h_i\}$$

考虑该式子有决策单调性，再考虑斜率优化，任取 j,k 满足 $0 \le k < j < i$，如果 j 比 k 更优，那么满足：

$$f_j + w_{j+1} \cdot h_i \le f_k + w_{k+1} \cdot h_i$$

注意到宽是递减的，有 $w_{k+1} \le w_{j+1}$，直接移项有：

$$\frac{f_j - f_k}{w_{k+1} - w_{j+1}} \le h_i$$

由于 h_i 是递增的，用单调队列维护斜率递增的上凸包即可。

```
#include <bits/stdc++.h>
using namespace std;
typedef double db;
typedef long long ll;
typedef pair<int, int> pII;
#define fi first
#define se second
const int N = 1e5 + 10;
int n; pII a[N];

namespace DP {
    ll f[N];
    int que[N], head, tail;
    db slope(int i, int j) { return (f[i] - f[j]) * 1.0 / (a[j + 1].fi - a[i + 1].fi); }
    void gao() {
        memset(f, 0, sizeof f);
        head = 1, tail = 0;
        que[++tail] = 0;
        for (int i = 1; i <= n; ++i) {
            while (head < tail && slope(que[head], que[head + 1]) <= a[i].se) ++head;
            f[i] = f[que[head]] + 1ll * a[que[head] + 1].fi * a[i].se;
            while (head < tail && slope(que[tail - 1], que[tail]) >= slope(que[tail], i)) -- tail;
            que[++tail] = i;
        }
        printf("%lld\n", f[n]);
    }
}

int main() {
    while (scanf("%d", &n) != EOF) {
        for (int i = 1; i <= n; ++i) scanf("%d%d", &a[i].fi, &a[i].se);
        sort(a + 1, a + 1 + n, [&](pII a, pII b) {
            if (a.fi != b.fi)
```

```
                return a.fi > b.fi;
            return a.se > b.se;
        });
        int _n = 1;
        for (int i = 2; i <= n; ++i) {
            if (a[i].se > a[_n].se)
                a[++_n] = a[i];
        }
        n = _n;
        DP::gao();
    }
    return 0;
}
```

21.2 洛谷 P3628

题意:

有 n 个人，每个人的战斗力为 x_i, 可以将连续的一段人进行分组，每个人属于一个小组，每一组的战斗力为 $aX^2 + bX + c$, X 为该组人战斗力之和，其中 $a < 0$

问如何分组能够使得最大和最大，求出最大和。

思路:

令 f_i 为安排前 i 个人的最大值, 并且令 $x_i = \sum_{j=1}^{i} x_j$:

$$\begin{aligned} f_i &= \max_{j=0}^{i-1}\{f_j + a(x_i - x_j)^2 + b(x_i - x_j) + c\} \\ &= \max_{j=0}^{i-1}\{f_j - 2ax_ix_j + ax_j^2 - bx_j\} + ax_i^2 + bx_i + c \end{aligned}$$

决策 $j > k$, 并且 j 比 k 优满足:

$$\begin{aligned} f_j - 2ax_ix_j + ax_j^2 - bx_j &\geq f_k - 2ax_ix_k + ax_k^2 - bx_k \\ \frac{f_j + ax_j^2 - bx_j - (f_k + ax_k^2 - bx_k)}{x_j - x_k} &\geq 2ax_i \end{aligned}$$

考虑 $a < 0, x_i$递增，所以不等式右边是递减的，直接用单调队列维护斜率递减的下凸包即可。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;
const int N = 1e6 + 10;
int n; ll a, b, c, x[N];

namespace DP {
    ll f[N], y[N];
    int que[N], head, tail;
    db slope(int i, int j) { return (y[i] - y[j]) * 1.0 / (x[i] - x[j]); }
    void gao() {
        memset(f, 0, sizeof f);
        head = 1, tail = 0;
        que[++tail] = 0;
        for (int i = 1; i <= n; ++i) {
            while (head < tail && slope(que[head], que[head + 1]) >= x[i] *
                a * 2) ++head;
            f[i] = f[que[head]] + a * (x[i] - x[que[head]]) * (x[i] - x[que[
                head]]) + b * (x[i] - x[que[head]]) + c;
```

```
            y[i] = f[i] + a * x[i] * x[i] - b * x[i];
            while (head < tail && slope(que[tail - 1], que[tail]) <= slope(
                que[tail], i)) -- tail;
            que[++tail] = i;
        }
        printf("%lld\n", f[n]);
    }
}

int main() {
    while (scanf("%d", &n) != EOF) {
        scanf("%lld%lld%lld", &a, &b, &c);
        for (int i = 1; i <= n; ++i) {
            scanf("%lld", x + i);
            x[i] += x[i - 1];
        }
        DP::gao();
    }
    return 0;
}
```

21.3 HDU 3480

题意:

给出 n 个数 a_i, 最多将数分成 m 个集合，每个集合的代价是 $(Max - Min)^2$，其中 Max 为集合中最大的数，Min 为集合中最小的数。

求如何分数，使得总代价最小。

先将数从小到大排序，显然是分成连续的段，考虑 $f_{i,j}$ 表示前 i 个数，分成 j 段的最小代价。

那么考虑斜率优化, 令 $0 \le k < j < i$, 如果 j 比 k 优，那么满足:

$$f_j + (a_i - a_{j+1})^2 \le f_k + (a_i - a_{k+1})^2$$

$$\frac{(f_j + a_{j+1}^2) - (f_k + a_{k+1}^2)}{a_{j+1} - a_{k+1}} \le 2a_i$$

因为 a_i 递增，如果维护斜率递增的上凸包即可。

但是被卡精度，就不能用浮点数存斜率，用分数的方式进行判断，但是要注意分母的正负，要强制保证分母是正的，这样乘到不等式另一边才不用变号，保证正确性。

```
#include <bits/stdc++.h>
using namespace std;
typedef int ll;
const int N = 1e4 + 10, INF = 0x3f3f3f3f;
int n, k; ll a[N], ff[N], gg[N], *f, *g;

namespace DP {
    struct frac {
        ll x, y;
        frac(ll x = 0, ll y = 0) : x(x), y(y) { }
        bool operator <= (const frac &other) const { return x * other.y <=
            other.x * y; }
        bool operator >= (const frac &other) const { return x * other.y >=
            other.x * y; }
    };
    ll y[N];
    int que[N], head, tail;
    //k < j < i
    frac slope(int k, int j) { return frac(y[j] - y[k], a[j + 1] - a[k + 1])
        ; }
```

```
18      void gao() {
19          head = 1, tail = 0;
20          que[++tail] = 0;
21          f[0] = 0;
22          for (int i = 1; i <= n; ++i) {
23              while (head < tail && slope(que[head], que[head + 1]) <= frac(2
                    * a[i], 1)) ++head;
24              f[i] = g[que[head]] + (a[i] - a[que[head] + 1]) * (a[i] - a[que[
                    head] + 1]);
25              y[i] = g[i] + a[i + 1] * a[i + 1];
26              while (head < tail && slope(que[tail - 1], que[tail]) >= slope(
                    que[tail], i)) --tail;
27              que[++tail] = i;
28          }
29      }
30  }
31
32  int main() {
33      int _T; scanf("%d", &_T);
34      for (int kase = 1; kase <= _T; ++kase) {
35          printf("Case %d: ", kase);
36          scanf("%d%d", &n, &k);
37          for (int i = 1; i <= n; ++i) scanf("%d", a + i);
38          sort(a + 1, a + 1 + n);
39          f = ff, g = gg;
40          g[0] = 0;
41          for (int i = 1; i <= n; ++i) {
42              g[i] = (a[i] - a[1]) * (a[i] - a[1]);
43          }
44          --k;
45          while (k--) {
46              DP::gao();
47              swap(f, g);
48          }
49          printf("%d\n", g[n]);
50      }
51      return 0;
52  }
```

21.4 牛客 890J

题意:
有 n 个矩形，高度相同的矩形可以拼起来。
现在需要拼出 k 块矩形，可以切掉一些矩形的一部分，使得它可以和别的矩形拼起来。
问拼出 k 块矩形需要切掉的最小面积。
思路:

- 将问题转化成保留最多的面积。
- 然后将矩形按高度从大到小排序
- 令 $f[i][k]$ 表示前 i 个矩形分成了 k 个保留的最大面积
- 那么转移就是:

$$\begin{aligned} f[i][k] &= f[j][k-1] + (sumw[i] - sumw[j]) \cdot h[i] \\ &= f[j][k-1] - sumw[j] \cdot h[i] + sumw[i] \cdot h[i] \end{aligned}$$

注意式子的前面部分：$f[j][k-1]-sumw[j]\cdot h[i]$ 是一个 $b-kx$ 的直线形式。
直接维护一个上凸壳进行转移即可
这个板子需要满足插入的直线的斜率具有单调性，询问的 x 值也具有单调性，复杂度为 $O(n)$。

```
#include <bits/stdc++.h>
using namespace std;

#define ll long long
const int N = 5e3 + 10;
const int INF = 0x3f3f3f3f;
int n, k;
struct node {
    int w, h;
    node() {}
    node(int w, int h) : w(w), h(h) {}
    void scan() {
        scanf("%d%d", &w, &h);
    }
    bool operator < (const node &other) const {
        return h > other.h;
    }
}a[N];
ll sum[N], f[N];

struct Line {
    struct node {
        ll a, b, x;
        node() {}
        node(ll a, ll b, ll x) : a(a), b(b), x(x) {}
    }t[N * 3]; int l, r, pos;
    void init() {
        l = N * 3 - 10, r = l - 1;
        pos = r;
    }
    void insert(ll a, ll b) {
        if (r < l) {
            t[--l] = node(a, b, INF);
            return;
        }
        ll x = -INF;
        while (l <= r) {
            node tmp = t[l];
            ll cross = (tmp.b - b) / (a - tmp.a);
            if (cross <= x) break;
            if (cross < t[l].x) {
                x = cross;
                break;
            } else {
                x = t[l].x;
                ++l;
            }
        }
        t[--l] = node(a, b, x);
    }
    ll query(ll x) {
        while (l < pos && t[pos - 1].x >= x) --pos;
        return 1ll * t[pos].a * x + t[pos].b;
    }
}L;
```

```
56
57 int main() {
58     while (scanf("%d%d", &n, &k) != EOF) {
59         for (int i = 1; i <= n; ++i) a[i].scan();
60         sort(a + 1, a + 1 + n);
61         sum[0] = 0;
62         ll tot = 0;
63         for (int i = 1; i <= n; ++i) {
64             sum[i] = sum[i - 1] + a[i].w;
65             tot += 1ll * a[i].w * a[i].h;
66         }
67         L.init();
68         for (int i = 1; i <= n; ++i) {
69             f[i] = 1ll * a[i].h * sum[i];
70             L.insert(-sum[i], f[i]);
71         }
72         for (int o = 2; o <= k; ++o) {
73             for (int i = 1; i <= n; ++i) {
74                 ll base = a[i].h * sum[i];
75                 f[i] = base + L.query(a[i].h);
76             }
77             L.init();
78             for (int i = 1; i <= n; ++i) {
79                 L.insert(-sum[i], f[i]);
80             }
81         }
82         printf("%lld\n", tot - f[n]);
83     }
84     return 0;
85 }
```

22 凸优化

22.1 短板

CF1303G
要 G++14 以上才能用

```
1  struct Line {
2      mutable ll k, m, p;
3      bool operator < (const Line& o) const { return k < o.k; }
4      bool operator < (ll x) const { return p < x; }
5  };
6
7  struct LineContainer : multiset<Line, less<>> {
8      ll div(ll a, ll b) { return a / b - ((a ^ b) < 0 && a % b); }
9      bool isect(iterator x, iterator y) {
10         if (y == end()) { x->p = INF; return false; }
11         if (x->k == y->k) x->p = x->m > y->m ? INF : -INF;
12         else x->p = div(y->m - x->m, x->k - y->k);
13         return x->p >= y->p;
14     }
15     void add(ll k, ll m) {
16         auto z = insert({k, m, 0}), y = z++, x = y;
17         while (isect(y, z)) z = erase(z);
18         if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
19         while ((y = x) != begin() && (--x)->p >= y->p)
20             isect(x, erase(y));
```

```
    }
    ll query(ll x) {
        if (empty()) return 0;
        auto it = lower_bound(x);
        assert(it != end());
        return (*it).k * x + (*it).m;
    }
};
```

22.2 长板

CF1303G
只能求最大值，要求最小值，可以将 (x,y) 取反成 $(-x,-y)$ 插入，然后将查询结果取反即可
支持 G++11

```
// upHull enables you to do the following two operations in amortized O(log n) time:
// 1. Insert a pair (a_i, b_i) into the structure
// 2. For any value of x, query the maximum value of a_i * x + b_i
// All values a_i, b_i, and x can be positive or negative.
struct UpHull {
    struct Point {
        db x, y;
        Point(db _x = 0, db _y = 0) : x(_x), y(_y) {}
    };
    struct segment {
        Point p;
        mutable Point next_p;
        segment(Point _p = {0, 0}, Point _next_p = {0, 0}) : p(_p), next_p(_next_p) {}
        bool operator < (const segment &other) const {
            // Sentinel value indicating we should binary search the set for a single x-value.
            if (sgn(p.y - INF) == 0)
                return sgn(p.x * (other.next_p.x - other.p.x) - (other.p.y - other.next_p.y)) <= 0;
            return sgn(p.x - other.p.x) == 0 ? sgn(p.y - other.p.y) < 0 : p.x < other.p.x;
        }
    };
    set<segment> segments;
    void clear() { segments.clear(); }
    int size() const { return segments.size(); }
    set<segment>::iterator prev(set<segment>::iterator it) const { return it == segments.begin() ? it : --it; }
    set<segment>::iterator next(set<segment>::iterator it) const { return it == segments.end() ? it : ++it; }
    //double类型使用
    //static db floor_div(db a, db b) { return a / b; }
    static db floor_div(db a, db b) { return a / b - ((a ^ b) < 0 && a % b != 0); }
    static bool bad_middle(const Point &a, const Point &b, const Point &c) {
        // This checks whether the x-value where b beats a comes after the x-value where c beats b. It's fine to round
        // down here if we will only query integer x-values. (Note: plain C++ division rounds toward zero)
        return sgn(floor_div(a.y - b.y, b.x - a.x) - floor_div(b.y - c.y, c.x - b.x)) >= 0;
```

```
33      }
34      bool bad(set<segment>::iterator it) const {
35          return it != segments.begin() && next(it) != segments.end() &&
                bad_middle(prev(it)->p, it->p, next(it)->p);
36      }
37      void insert(const Point &p) {
38          set<segment>::iterator next_it = segments.lower_bound(segment(p));
39          if (next_it != segments.end() && p.x == next_it->p.x)
40              return;
41          if (next_it != segments.begin()) {
42              set<segment>::iterator prev_it = prev(next_it);
43              if (p.x == prev_it->p.x)
44                  segments.erase(prev_it);
45              else if (next_it != segments.end() && bad_middle(prev_it->p, p,
                    next_it->p))
46                  return;
47          }
48          // Note we need the segment(p, p) here for the single x-value binary
                search.
49          set<segment>::iterator it = segments.insert(next_it, segment(p, p));
50          while (bad(prev(it)))
51              segments.erase(prev(it));
52          while (bad(next(it)))
53              segments.erase(next(it));
54          if (it != segments.begin())
55              prev(it)->next_p = it->p;
56          if (next(it) != segments.end())
57              it->next_p = next(it)->p;
58      }
59      void insert(db a, db b) { insert(Point(a, b)); }
60      // Queries the maximum value of ax + b.
61      db query(db x) {
62          if (size() == 0) return -INF;
63          set<segment>::iterator it = segments.upper_bound(segment(Point(x,
                INF)));
64          return it->p.x * x + it->p.y;
65      }
66  };
```

23 四边形不等式优化

四边形不等式

设函数 $w(x,y)$ 是定义在 Z 上的函数, 若对于任意 $a,b,c,d \in Z$，其中 $a \le b \le c \le d$，都有:

$$w(a,d) + w(b,c) \ge w(a,c) + w(b,d)$$

则称函数 w 满足四边形不等式。

推论

设函数 $w(x,y)$ 是定义在 Z 上的函数，若对于任意 $a,b \in Z$，其中 $a < b$，都有:

$$w(a,b+1) + w(a+1,b) \ge w(a,b) + w(a+1,b+1)$$

则称函数 w 满足四边形不等式。

定理一

对于任意 $a,b,c,d \in Z$, 如果函数 w 满足四边形不等式, 且 $w(a,b) \ge w(b,c)$, 则函数 f 也满足四边形不等式, 其中 f 满足:

$$f(x,y) = min(f(x,z) + f(z+1,y) + w(x,y) | x \le z < y)$$

特别的，我们令 $f(x,y) = w(x,y) = 0$

定理二

对于任意 $a,b,c,d \in Z$, 如果函数 w 满足四边形不等式, 且函数 f 满足:

$$f(x,y) = min(f(x,z) + f(z+1,y) + w(x,y) | x \le z < y)$$

特别的，我们令 $f(x,y) = w(x,y) = 0$

记 $P(x,y)$ 为令 $f(x,y)$ 取到最小值的 k 值。如果函数 f 满足四边形不等式, 那么对于任意 x,y，我们有:

$$P(x,y-1) \le P(x,y) \le P(x+1,y)$$

洛谷 P1880

题意: 一个环上有 n 个石子, 每次能够选择相邻的两堆进行合并，新的一堆的石子数是用于合并的两堆的石子数量的和，新石子堆的数量也是合并的代价，求最后合并成一堆的最小代价和最大代价。

思路:

考虑区间 DP，那么对于环的情况，我们将原序列复制一遍成 $2n$，那么环就是任意连续长度为 n 的子区间的代价最值。

令 $dp[i][j]$ 表示 i 到 j 之间石子合并的最大值 (最小值), 有转移方程:

$$dp[i][j] = \max / \min(dp[i][j], dp[i][k] + dp[k+1][j] + d(i,j) \mid i \le k < j)$$

其中 $d(i,j)$ 表示 i 到 j 之间的石子数量。

对于最小值，可以用四边形不等式优化, 对于 $dp[i][j]$, 只需要在区间 $[P[i][j-1], P[i+1][j]]$ 枚举 k 即可，时间复杂度 $O(n^2)$。

最大值不满足单调性，不能用四边形不等式优化, 但是最大值有一个性质，使最大值最优的决策 $P[i][j]$ 要么是 i，要么是 $j-1$。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 2e2 + 10, INF = 0x3f3f3f3f;
int n, a[N], S[N], Min[N][N], Max[N][N], PMin[N][N];

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", a + i);
        a[i + n] = a[i];
    }
    for (int i = 1; i <= n * 2; ++i) S[i] = S[i - 1] + a[i], PMin[i][i] = i;
    for (int len = 2; len <= n * 2; ++len) {
        for (int i = 1; i + len - 1 <= n * 2; ++i) {
            int j = i + len - 1;
            Min[i][j] = INF;
            Max[i][j] = max(Max[i][i] + Max[i + 1][j], Max[i][j - 1] + Max[j
                ][j]) + S[j] - S[i - 1];
            for (int k = PMin[i][j - 1]; k <= PMin[i + 1][j]; ++k) {
                int V = Min[i][k] + Min[k + 1][j] + S[j] - S[i - 1];
                if (V < Min[i][j]) {
                    Min[i][j] = V;
                    PMin[i][j] = k;
                } else if (V == Min[i][j]) {
                    PMin[i][j] = max(PMin[i][j], k);
                }
```

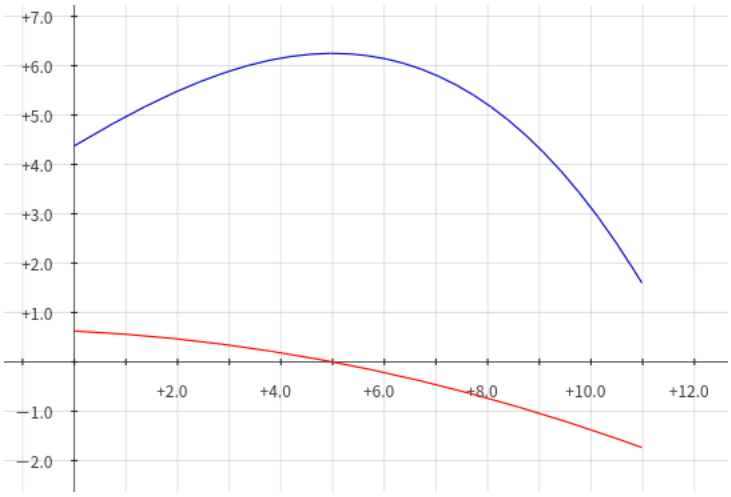
```
            }
        }
    }
    int resMin = INF, resMax = 0;
    for (int i = 1; i <= n; ++i) {
        resMin = min(resMin, Min[i][i + n - 1]);
        resMax = max(resMax, Max[i][i + n - 1]);
    }
    printf("%d\n%d\n", resMin, resMax);
    return 0;
}
```

24 WQS 二分

解决的问题：有 n 个物品，规定以若干方式选择物品有若干代价，需要在强制选出 C 个物品的前提下最大/最小化代价。

该类题目一般还满足如下特点，设 $g(x)$ 表示强制选 x 个物品的最值：

- 无法直接求出 $g(C)$ 的值
- 可以直接求出 g 的值，以及使 $g(x)$ 取到最值的 x
- g 是一个凸函数



考虑原函数的导数的关系进行理解，令蓝色曲线为 $g(x)$，红色曲线为 $g'(x)$。

设 $C = 7$，我们要求出 $g(7)$，因为 $g(x)$ 是凸的，所以 $g'(x)$ 是单调递减的，并且可以直接求出 $g(x)$ 的最值，并且它是导函数与 x 轴的交点，发现 $x = 5$ 的时候取到最值，但是它并不是我们想要的。

我们想要让导函数与 x 轴的交点移到 $x = 7$ 的位置，这样直接求 $g(x)$ 的最值就可以了。

令 $f(x) = g(x) + kx$，k 的现实意义是每多选一个物品，就要多付出 k 的代价，并且发现 $f'(x) = g'(x) + k$，那么 k 的函数意义就是让导函数向上平移了 k 个单位。

相当于导函数与 x 轴的交点右移，等价于 k 越大，交点也越向右，这样便是可二分的。

k 的值域就是导函数的值域，根据求得的交点和 C 的大小关系，调整二分的方向。

24.1 洛谷 P2619

题意：

给出一个无向带权连通图，每条边是黑色或白色，求一棵最小权的恰好有 k 条边的生成树。

考虑不管 k 条边限制的最优值就是求最小生成树，那么带上权去二分即可。

直观的解释就是，如果白色的边多了，那么就增加选白色边的代价，这样在最小生成树中白色的边就会相应减少，反之亦然。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5 + 10;
const ll INF = 0x3f3f3f3f3f3f3f3f;
int n, m, C;

struct Edge {
    int u, v, w, k, vis;
    Edge() {}
    Edge(int u, int v, int w, int vis) : u(u), v(v), w(w), vis(vis) {k = 0;}
    bool operator < (const Edge &other) const {
        if (k + w == other.k + other.w) return vis > other.vis;
        return k + w < other.k + other.w;
    }
}e[N];

struct UFS {
    int fa[N];
    void init() { memset(fa, 0, sizeof fa); }
    int find(int x) { return fa[x] == 0 ? x : fa[x] = find(fa[x]); }
    bool merge(int u, int v) {
        int fu = find(u), fv = find(v);
        if (fu != fv) {
            fa[fu] = fv;
            return true;
        }
        return false;
    }
}ufs;

namespace WQS {
    int check(int K) {
        for (int i = 1; i <= m; ++i) {
            e[i].k = e[i].vis * K;
        }
        sort(e + 1, e + 1 + m);
        ufs.init();
        int cnt_e = 0;
        for (int i = 1; i <= m; ++i) {
            int u = e[i].u, v = e[i].v;
            if (ufs.merge(u, v)) cnt_e += e[i].vis;
        }
        return cnt_e;
    }
    int calc(int K) {
        for (int i = 1; i <= m; ++i) {
            e[i].k = e[i].vis * K;
        }
        sort(e + 1, e + 1 + m);
        ufs.init();
        ll tot = 0;
        for (int i = 1; i <= m; ++i) {
            int u = e[i].u, v = e[i].v, w = e[i].w;
            if (ufs.merge(u, v)) tot += w + e[i].k;
        }
        return tot - C * K;
    }
```

```
59      void gao() {
60          int l = -200, r = 100, res = -200;
61          while (r - l >= 0) {
62              int mid = (l + r) >> 1;
63              if (check(mid) >= C) {
64                  l = mid + 1;
65                  res = mid;
66              } else {
67                  r = mid - 1;
68              }
69          }
70          printf("%d\n", calc(res));
71      }
72  }
73
74  int main() {
75      while (scanf("%d%d%d", &n, &m, &C) != EOF) {
76          for (int i = 1, u, v, w, vis; i <= m; ++i) {
77              scanf("%d%d%d%d", &u, &v, &w, &vis);
78              ++u, ++v;
79              e[i] = Edge(u, v, w, vis ^ 1);
80          }
81          WQS::gao();
82      }
83      return 0;
84  }
```

24.2 Gym 101981B

题意:

有 n 个人，他们所处的位置为 $a_i(0 = a_1 < a_2 < \cdots < a_n \le 10^9)$, 要恰好安排 k 个垃圾桶，令 $D(a_i)$ 表示第 i 个人到离它最近的垃圾桶的距离，现在要安排 k 个垃圾桶的位置，使得 $\sum_{i=1}^{n} D(a_i)$ 最少。

思路:

现在如果没有 k 这个限制，很好求，有这个限制套一个 WQS 二分即可。

再考虑怎么求 $g(x)$ 的最值?

显然垃圾桶一定可以安排在某个人的位置，所以可选的位置只有 n 个。

令 f_i 表示前 i 个位置的最短距离和，g_i 表示当 i 放了一个垃圾桶，前 i 个位置的最短距离和，S_i 表示 $\sum_{i=1}^{n} a_i$。

我们考虑转移:

$$g_i = \min_{j=0}^{i-1} f_j - (S_i - S_j) + (i-j)\cdot a_i$$
$$f_i = \min_{j=1}^{i-1} g_j + (S_i - S_j) - (i-j)\cdot a_j$$

考虑斜率优化，令 $j > k$, 有:

$$\frac{(f_j + S_j) - (f_k + S_k)}{j-k} \le a_i$$
$$\frac{(f_j - S_j + ja_j) - (f_k - S_k + ka_k)}{a_j - a_k} \le i$$

然后单调队列维护上凸包的斜率即可。

但是要注意这里还有一个垃圾桶个数的限制，这个转移的时候递推以下即可，并且我们发现当距离和相同的时候，垃圾桶个数越少越好，并且注意到 f_i, g_i 也有单调性，所以等价于维护斜率单调性的时候，如果存在 $f_i = f_j(j > i)$ 的情况，是不会剔除 f_i 的，相当于维护斜率单调性的时候的等于号需要去掉。

这里 *WQS* 二分的时候的带的权可以考虑为每多加一个垃圾桶需要多付出的代价，如果当前放的垃圾桶比较少，那么我们就减少代价，否则增加代价。

上下界可以取为 $[0, Sum[n]+1]$ 即可。其实应该是 $[-\infty, \infty]$，只是 $[0, Sum[n]+1]$ 和 $[-\infty, \infty]$ 能达到的效果是一样的，即下界为 0 时，肯定是放 n 个垃圾桶距离和取到最小值，上界为 $Sum[n]+1$ 时，肯定是放 1 个垃圾桶距离和取到最大值。

时间复杂度 $O(n \log n)$

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using db = double;
using pLI = pair<ll, int>;
#define fi first
#define se second
const int N = 3e5 + 10;
const ll INF = 2e18;
int n, C, a[N]; ll S[N];

namespace WQS {
    //0 前i个位置的最值
    //1 前i个位置，并且第i个位置是集结点的最值
    struct frac {
        db x, y;
        frac(db x = 0, db y = 0) : x(x), y(y) { }
        bool operator < (const frac &other) const { return x * other.y < other.x * y; }
        bool operator > (const frac &other) const { return x * other.y > other.x * y; }
    };
    ll f[2][N]; db y[2][N], x[2][N]; int cnt[2][N];
    int que[2][N], head[2], tail[2];
    frac slope(int k, int j, int id) { return frac(y[id][j] - y[id][k], x[id][j] - x[id][k]); }
    pLI check(ll k) {
        head[0] = 1; tail[0] = 0;
        head[1] = 1; tail[1] = 0;
        que[0][++tail[0]] = 0;
        for (int i = 1, j; i <= n; ++i) {
            //第i个位置是集结点的最值
            while (head[0] < tail[0] && slope(que[0][head[0]], que[0][head[0] + 1], 0) < frac(a[i], 1)) ++head[0];
            j = que[0][head[0]];
            f[1][i] = f[0][j] - (S[i] - S[j]) + 1ll * (i - j) * a[i] + k;
            cnt[1][i] = cnt[0][j] + 1;
            y[1][i] = f[1][i] - S[i] + 1ll * i * a[i];
            x[1][i] = a[i];
            while (head[1] < tail[1] && slope(que[1][tail[1] - 1], que[1][tail[1]], 1) > slope(que[1][tail[1]], i, 1)) --tail[1];
            que[1][++tail[1]] = i;
            //前i个位置的最值
            while (head[1] < tail[1] && slope(que[1][head[1]], que[1][head[1] + 1], 1) < frac(i, 1)) ++head[1];
            j = que[1][head[1]];
            f[0][i] = f[1][j] + (S[i] - S[j]) - 1ll * (i - j) * a[j];
            cnt[0][i] = cnt[1][j];
            y[0][i] = f[0][i] + S[i];
            x[0][i] = i;
            while (head[0] < tail[0] && slope(que[0][tail[0] - 1], que[0][
```

```
                tail[0]], 0) > slope(que[0][tail[0]], i, 0)) --tail[0];
            que[0][++tail[0]] = i;
        }
        return pLI(f[0][n], cnt[0][n]);
    }
    void gao() {
        ll l = 0, r = S[n] + 1, pos = 0;
        while (r - l >= 0) {
            ll mid = (l + r) >> 1;
            if (check(mid).se <= C) {
                r = mid - 1;
                pos = mid;
            } else {
                l = mid + 1;
            }
        }
        printf("%lld\n", check(pos).fi - 1ll * pos * C);
    }
}

int main() {
    while (scanf("%d%d", &n, &C) != EOF) {
        S[0] = 0;
        for (int i = 1; i <= n; ++i) scanf("%d", a + i), S[i] = S[i - 1] + a
            [i];
        WQS::gao();
    }
    return 0;
}
```

25 高维前缀和

子集:

$\forall i \in [0, 2^n - 1]$，求解 $\sum_{j \subset i} a_j$

```
for(int j = 0; j < n; j++)
    for(int i = 0; i < 1 << n; i++)
        if(i >> j & 1) f[i] += f[i ^ (1 << j)];
```

超集:

$\forall i \in [0, 2^n - 1]$，求解 $\sum_{i \subset j} a_j$

```
for(int j = 0; j < n; j++)
    for(int i = 0; i < 1 << n; i++)
        if(!(i >> j & 1)) f[i] += f[i ^ (1 << j)];
```

二维前缀和:

```
for(int j = 0; j < n; j++)
    for(int i = 0; i < 1 << n; i++)
        if(i >> j & 1) f[i] += f[i ^ (1 << j)];
```

三维前缀和:

```
for(int i = 1; i <= n; i++)
    for(int j = 1; j <= n; j++)
        for(int k = 1; k <= n; k++)
            a[i][j][k] += a[i - 1][j][k];
for(int i = 1; i <= n; i++)
```

```
    for(int j = 1; j <= n; j++)
        for(int k = 1; k <= n; k++)
            a[i][j][k] += a[i][j - 1][k];
for(int i = 1; i <= n; i++)
    for(int j = 1; j <= n; j++)
        for(int k = 1; k <= n; k++)
            a[i][j][k] += a[i][j][k - 1];
```

25.1 ARC 100E

题意:
给出 2^n 个数 a_i，之后对于 $\forall k \in [1, 2^n - 1]$，求出 $a_i + a_j$ 的最大值，并且满足 $i \ or \ j \le k$
思路:

- 我们对于每个 k 都求出最大的 $a_i + a_j$ 并且满足 $i \ or \ j = k$，最后答案就是一个前缀最大值
- 所以可以进一步转化为 $i \ or \ j \subset k$，那么就将问题转化为了子集问题
- 所以对于每个 k，求出其所有子集的最大值和次大值即可

```
#include <bits/stdc++.h>
using namespace std;
using pII = pair<int, int>;
#define fi first
#define se second
const int N = 1e6 + 10, INF = 0x3f3f3f3f;
int n, a[N], ans[N]; pII f[N];
int main() {
    scanf("%d", &n);
    a[1 << n] = -INF;
    for (int i = 0; i < 1 << n; ++i) scanf("%d", a + i);
    for (int i = 0; i < 1 << n; ++i) f[i] = pII(i, 1 << n);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < 1 << n; ++j) {
            if ((j >> i) & 1) {
                int k = j ^ (1 << i);
                if (a[f[k].fi] > a[f[j].fi]) {
                    f[j].se = f[j].fi;
                    f[j].fi = f[k].fi;
                //  cout << j << " " << f[j].fi << " " << f[j].se << endl;
                }
                if (f[k].fi != f[j].fi && a[f[k].fi] > a[f[j].se]) {
                    f[j].se = f[k].fi;
                }
                if (f[k].se != f[j].fi && a[f[k].se] > a[f[j].se]) {
                    f[j].se = f[k].se;
                }
            //  cout << j << " " << f[j].fi << " " << f[j].se << endl;
            }
        }
    }
    memset(ans, 0, sizeof ans);
    for (int i = 1; i < 1 << n; ++i) {
        ans[i] = ans[i - 1];
    //  cout << i << " " << f[i].fi << " " << f[i].se << endl;
        if (f[i].fi != f[i].se && f[i].se != (1 << n)) {
            ans[i] = max(ans[i], a[f[i].fi] + a[f[i].se]);
        }
```

```
39          printf("%d\n", ans[i]);
40      }
41      return 0;
42  }
```

25.2 opentrains 010413F

题意:

给出 n 个数，最多去掉 k 个数，使得剩下的数的 gcd 最大。

其中 $k \le \frac{n}{2}$

思路:

转化成至少保留 $n-k$ 个数，使得 gcd 最大，因为 $k \le \frac{n}{2}$，故每个数属于最优解的概率至少为 $\frac{1}{2}$，多随机几次，假设随机了 x 次，每次随机出来都不是最优解的概率为 $(\frac{1}{2})^x$，多随机几次，就能保证出答案。

然后枚举这个数的因数，看看剩下的数中出现的次数，大于等于 $n-k-1$ 就可以作为答案，可以大 $gcd(a[i],k)$ 处统计答案，然后用高维前缀和统计出现次数。

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   using ll = long long;
4   const int N = 3e5 + 10;
5   int n, m, f[N];
6   ll a[N], ans, fac[N], bk[N]; int tot;
7   vector <ll> vec;
8   mt19937 rd(time(0));
9   inline ll gcd(ll a, ll b) {
10      return b ? gcd(b, a % b) : a;
11  }
12  inline ll mul(ll a, ll b, ll p) {
13      return (a * b - (ll)(a / (long double)p * b + 1e-3) * p + p) % p;
14  }
15  inline ll qmod(ll base, ll n, ll p) {
16      ll res = 1;
17      base %= p;
18      while (n) {
19          if (n & 1) {
20              res = mul(res, base, p);
21          }
22          base = mul(base, base, p);
23          n >>= 1;
24      }
25      return res;
26  }
27
28  //此处需要Mill板子
29
30  inline int id(ll x) { return lower_bound(vec.begin(), vec.end(), x) - vec.
        begin(); }
31  void gao(ll x) {
32      if (x == 1) return;
33      mill.gao(x, vec);
34      sort(fac + 1, fac + 1 + tot);
35      tot = unique(fac + 1, fac + 1 + tot) - fac - 1;
36      memset(f, 0, sizeof f);
37      for (int i = 1; i <= n; ++i) {
38          ll G = gcd(x, a[i]);
39          ++f[id(G)];
40      }
41      int sze = vec.size();
```

```
    for (int i = 1; i <= tot; ++i) {
        int j, k;
        for (j = k = sze - 1; j >= 0; --j) if (vec[j] % fac[i] == 0) {
            ll goal = vec[j] / fac[i];
            while (vec[k] > goal) --k;
            f[k] += f[j];
        }
    }
    for (int i = sze - 1; i >= 0; --i) {
        if (f[i] >= m) {
            ans = max(ans, vec[i]);
            break;
        }
    }
}

int main() {
    while (scanf("%d%d", &n, &m) != EOF) {
        m = n - m;
        for (int i = 1; i <= n; ++i)
            scanf("%lld", a + i);
        ans = 1;
        for (int i = 1; i <= 8; ++i) {
            int x = (rd() % n + n) % n + 1;
            gao(a[x]);
        }
        printf("%lld\n", ans);
    }
    return 0;
}
```

26 状压 DP

26.1 带位置信息的状压

CometOJ 52C

题意:

给出 n 张牌，一个合法的排列当且仅当编号为 i 的牌不在位置 p_i 上。

一个合法的排列的价值为:

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}[pos_j < pos_i]\cdot|j-i|\cdot|pos_i - pos_j|$$

现在要求计算所有合法排列的价值和。

思路:

将贡献拆成 $|j-i|\cdot pos_i$ 和 $-|j-i|\cdot pos_j$

考虑 $f[mask].fi$ 表示前 $popcount(mask)$ 个位置已经放下了 $mask$ 状态的牌的贡献和。

转移的时候枚举 $mask$ 中一个已经存在的元素进行转移，并且钦定它放在 $popcount(mask)$ 个位置，那么之前的元素都在它前面，还没放的元素都在它后面。

并且转移的时候需要维护一下方案数，时间复杂度 $O(n^2\cdot 2^n)$

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using pLL = pair<ll, ll>;
#define fi first
```

```
#define se second
int n, p[20];
pLL f[(1 << 16) + 10];

int main() {
    int _T; cin >> _T;
    while (_T--) {
        scanf("%d", &n);
        for (int i = 1; i <= n; ++i) scanf("%d", p + i);
        memset(f, 0, sizeof f);
        f[0] = pLL(0, 1);
        for (int mask = 1; mask < (1 << n); ++mask) {
            int cnt = __builtin_popcount(mask);
            for (int i = 0; i < n; ++i) {
                if (((mask >> i) & 1) && p[i + 1] != cnt) {
                    ll fee = 0;
                    for (int j = 0; j < n; ++j) if (j != i) {
                        if ((mask >> j) & 1) {
                            if (i < j) fee += abs(i - j) * cnt;
                        } else {
                            if (i > j) fee -= abs(i - j) * cnt;
                        }
                    }
                    int pre = mask ^ (1 << i);
                    f[mask].se += f[pre].se;
                    f[mask].fi += f[pre].fi + fee * f[pre].se;
                }
            }
        }
        printf("%lld\n", f[(1 << n) - 1].fi);
    }
    return 0;
}
```

26.2 CometOJ 13D

生成树装压 dp。

题意:

有 n 个点，m 条双向边 (u_i, v_i, w_i)，表示连接了 u_i 和 v_i，通过的时间为 w_i。

令 $d(i,j)$ 表示 i 到 j 的最短时间，定义一个道路系统的复杂度为 $\sum_{i=1}^{n}\sum_{j=1}^{n} d(i,j)$。

删除一些道路，使得图连通的情况下，复杂度尽量高。

图中没有重边和自环，保证图连通, $1 \le n \le 14$。

思路:

最后的结果肯定是一棵生成树，因为边删的越多，复杂度越高。

考虑 $f[S][i]$ 表示点集 S 是一棵子树，第 i 个点是根，转移的时候枚举以 j 为根，包含的点集为 T，那么考虑可以从 $f[i][S-T]$ 和 $f[j][T]$ 转移过来，并且考虑 (i,j) 这条边产生的贡献。

考虑一条边 (i,j) 的转移，如果转移的贡献跟两边的子树大小有关，注意以 j 为根的子树是确定的，假设大小为 $Size_j$，但是另一半的子树还没有确定，但是大小是知道的，是 $n - Size_j$。

时间复杂度: $O(3^n \cdot n^2)$。

```
#include <bits/stdc++.h>
#define SZ(x) ((int)x.size())
using namespace std;
typedef long long ll;
const ll INFLL = 0x3f3f3f3f3f3f3f3f;
const int N = 110;
int n, m;
```

```
int dist[20][20];
ll f[1 << 15][15];
vector <int> vec[1 << 15];

int main() {
    while (scanf("%d%d", &n, &m) != EOF) {
        memset(dist, -1, sizeof dist);
        memset(f, -1, sizeof f);
        for (int S = 0, len = (1 << n); S < len; ++S) {
            for (int i = 0; i < n; ++i) if ((S >> i) & 1)
                vec[S].push_back(i + 1);
            if (SZ(vec[S]) == 1)
                f[S][*vec[S].begin()] = 0;
        }
        for (int i = 1, u, v, w; i <= m; ++i) {
            scanf("%d%d%d", &u, &v, &w);
            dist[u][v] = dist[v][u] = w;
        }
        ll res = 0;
        for (int S = 1; S < (1 << n); ++S) {
            for (auto &u : vec[S]) {
                // 枚举子集
                for (int T = (S - 1) & S; T != 0; T = (T - 1) & S) {
                    for (auto &v : vec[T]) {
                        if (dist[u][v] == -1 || f[T][v] == -1 || f[S - T][u]
                            == -1) continue;
                        f[S][u] = max(f[S][u], f[T][v] + f[S - T][u] + 1ll *
                            dist[u][v] * SZ(vec[T]) * (n - SZ(vec[T])));
                    }
                }
                if (S == (1 << n) - 1)res = max(res, f[S][u]);
            }
        }
        printf("%lld\n", res);
    }
    return 0;
}
```

27 数位 DP

2019Nowcoder 多校第七场 H. Pair
题意:
给出三个数 (A,B,C)，问有多少对 (x,y) 满足:

- $1 \le x \le A$
- $1 \le y \le B$
- $x \text{ and } y > C$ 或者 $x \text{ xor } y < C$

思路:
考虑 $f[i][j][k][l][m]$ 表示:

- 枚举到前 i 位
- x 是否卡到 A 的上界
- y 是否卡到 B 的上界

- *x and y* 是否已经大于 *C*
- *x and y* 是否已经小于 *C*

最后注意一下 0 的贡献，强行删去或者再加两位状态标记。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define N 100

ll A, B, C;
int a[N], b[N], c[N];
ll dp[N][3][3][2][2][2][2];

//位置 x&y x^y (=0 <1 >2)
ll DFS(int pos, int flag1, int flag2, int limit1, int limit2, int one1, int
    one2) {
    if (pos == -1 && (flag1 == 2 || flag2 == 1) && one1 == 1 && one2 == 1) {
        return 1ll;
    } else if (pos == -1) {
        return 0ll;
    }
    ll &res = dp[pos][flag1][flag2][limit1][limit2][one1][one2];
    if (res != -1) {
        return res;
    }
    res = 0;
    if (flag1 == 1 && flag2 == 2) {//x&y<C x^y>C 不合法
        return 0ll;
    }
    int l1 = limit1 ? a[pos] : 1;
    int l2 = limit2 ? b[pos] : 1;
    for (int i = 0; i <= l1; ++i) {
        for (int j = 0; j <= l2; ++j) {
            int x = i & j;
            int y = i ^j;
            int f1 = flag1;
            if (f1 == 0) {
                if (x > c[pos]) {
                    f1 = 2;
                } else if (x == c[pos]) {
                    f1 = 0;
                } else {
                    f1 = 1;
                }
            }
            int f2 = flag2;
            if (f2 == 0) {
                if (y > c[pos]) {
                    f2 = 2;
                } else if (y == c[pos]) {
                    f2 = 0;
                } else {
                    f2 = 1;
                }
            }
            res += DFS(pos - 1, f1, f2, limit1 && (i == a[pos]),
                       limit2 && (j == b[pos]), one1 | i, one2 | j);
        }
```

```
    }
    return res;
}

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        scanf("%lld %lld %lld", &A, &B, &C);
        for (int i = 30; i >= 0; --i) {
            if (A & (1ll << i)) {
                a[i] = 1;
            } else {
                a[i] = 0;
            }
            if (B & (1ll << i)) {
                b[i] = 1;
            } else {
                b[i] = 0;
            }
            if (C & (1ll << i)) {
                c[i] = 1;
            } else {
                c[i] = 0;
            }
        }
        memset(dp, -1, sizeof dp);
        printf("%lld\n", DFS(30, 0, 0, 1, 1, 0, 0));
    }
    return 0;
}
```

28 分治 DP

28.1 二维分治

CF 364E

题意:

给出一个 $n \cdot m$ 的 01 矩形，找出有多少个子矩形满足其矩形和为 k

$1 \le n, m \le 2500, 0 \le k \le 6$

思路:

考虑暴力就是枚举上下界再根据前缀和进行计数，复杂度是 $O(n^2m)$

我们考虑分治能够改变统计的东西，我们考虑横向切一刀，统计跨过 mid 的合法区间个数，那么我们统计一下 $up_{0,k}$ 表示从 mid 往上走矩形和 $\ge k$ 的第一个位置，$up_{1,k}$ 表示 $mid+1$ 往下走矩形和 $\ge k$ 的第一个位置，这个随着另一维度区间长度的缩小具有单调性，那么最后答案就是

$$\sum_{i=0}^{K}(up[0][i]-up[0][i+1]) * (up[1][K-k+1]-up[1][K-k])$$

注意要判断一下 $up_{0,k}$ 和 $up_{1,K-k}$ 这两个位置是否是矩形和恰好等于 k 和 $K-k$ 的。

时间复杂度 $O(n^2klogn)$

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 3e3 + 10;
int n, m, K, a[N][N], S[N][N], up[2][10];
```

```
6  ll res;
7
8  inline int get(int x1, int x2, int y1, int y2) {
9      if (x1 > x2 || y1 > y2) return 0;
10     x1 = max(x1, 1); y1 = max(y1, 1);
11     return S[x2][y2] - S[x1 - 1][y2] - S[x2][y1 - 1] + S[x1 - 1][y1 - 1];
12 }
13
14 void gao(int x1, int x2, int y1, int y2) {
15     if (x1 > x2 || y1 > y2) return;
16     if (x1 == x2 && y1 == y2) {
17         res += a[x1][y1] == K;
18         return;
19     }
20     if (x2 - x1 > y2 - y1) {
21         int mid = (x1 + x2) >> 1;
22         gao(x1, mid, y1, y2);
23         gao(mid + 1, x2, y1, y2);
24         for (int i = y1; i <= y2; ++i) {
25             for (int k = 0; k <= K + 1; ++k) up[0][k] = mid, up[1][k] = mid
                   + 1;
26             for (int j = y2; j >= i; --j) {
27                 for (int k = 0; k <= K + 1; ++k) {
28                     while (up[0][k] >= x1 && get(up[0][k], mid, i, j) < k)
                           --up[0][k];
29                     while (up[1][k] <= x2 && get(mid + 1, up[1][k], i, j) <
                           k) ++up[1][k];
30                 }
31                 for (int k = 0; k <= K; ++k) {
32                     if (get(up[0][k], mid, i, j) == k && get(mid + 1, up[1][
                           K - k], i, j) == K - k) {
33                         res += 1ll * (up[0][k] - up[0][k + 1]) * (up[1][K -
                               k + 1] - up[1][K - k]);
34                     }
35                 }
36             }
37         }
38     } else {
39         int mid = (y1 + y2) >> 1;
40         gao(x1, x2, y1, mid);
41         gao(x1, x2, mid + 1, y2);
42         for (int i = x1; i <= x2; ++i) {
43             for (int k = 0; k <= K + 1; ++k) up[0][k] = mid, up[1][k] = mid
                   + 1;
44             for (int j = x2; j >= i; --j) {
45                 for (int k = 0; k <= K + 1; ++k) {
46                     while (up[0][k] >= y1 && get(i, j, up[0][k], mid) < k)
                           --up[0][k];
47                     while (up[1][k] <= y2 && get(i, j, mid + 1, up[1][k]) <
                           k) ++up[1][k];
48                 }
49                 for (int k = 0; k <= K; ++k) {
50                     if (get(i, j, up[0][k], mid) == k && get(i, j, mid + 1,
                           up[1][K - k]) == K - k) {
51                         res += 1ll * (up[0][k] - up[0][k + 1]) * (up[1][K -
                               k + 1] - up[1][K - k]);
52                     }
53                 }
```

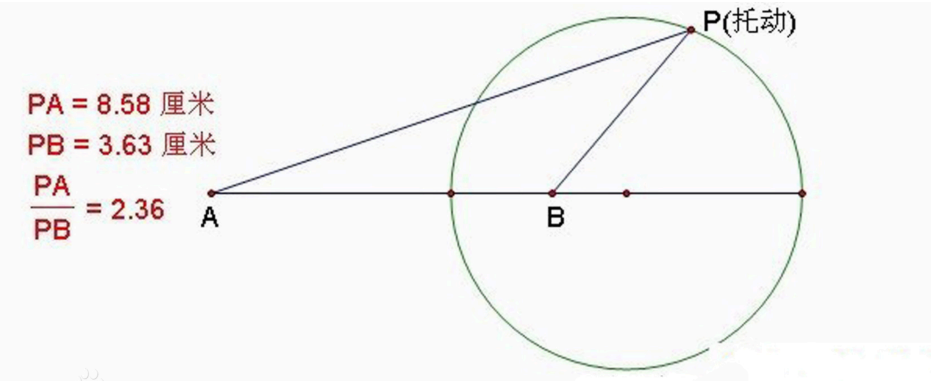
```
            }
        }
    }
}

int main() {
    scanf("%d%d%d", &n, &m, &K);
    for (int i = 1; i <= n; ++i) {
        static char s[N]; scanf("%s", s);
        for (int j = 0; j < m; ++j) {
            a[i][j + 1] = s[j] - '0';
            S[i][j + 1] = S[i - 1][j + 1] + S[i][j] - S[i - 1][j] + a[i][j +
                1];
        }
    }
    res = 0;
    gao(1, n, 1, m);
    printf("%lld\n", res);
    return 0;
}
```

29 基础知识

弧度制和角度制：$1rad = \frac{180}{\pi} \approx 57.3^\circ \quad 1^\circ = \frac{\pi}{180} \approx 0.01745rad$

包围凸包的圆角多边形: 弄一个圆角多边形包围凸包，并且使得凸包上的每个点到圆角多边形的距离为 d，那么该圆角多边形的周长为凸包的周长 + 半径为 d 的圆的周长

阿波罗尼斯圆:



令 B 为坐标原点, A 的坐标为 $(a,0)$, 则动点 $P(x,y)$ 满足 $\frac{PA}{PB} = k(k > 0且k \neq 1)$

且 $PA = \sqrt{(x-a)^2 + y^2}$, $PB = \sqrt{x^2 + y^2}$

整理得: $(k^2-1)(x^2+y^2) + 2ax - a^2 = 0$

- 当 $k > 0且k \neq 1$ 时, 它是圆
- 当 $k = 1$ 时, 轨迹是两点连线的中垂线

球缺:

用一个平面截去一个球所得部分叫球缺。

球缺面积：$2\pi h$。

球缺体积：$\pi \cdot h^2 \cdot (R - \frac{h}{3})$

球缺质心：匀质球缺的质心位于它的中轴线上，并且与底面的距离为 $C = \frac{(4 \cdot R - h) \cdot h}{12 \cdot R - 4 \cdot h} = \frac{(d^2 + 2 \cdot h^2) \cdot h}{3 \cdot d^2 + 4 \cdot h^2}$

其中，h 为球缺的高，R 为大圆半径，d 为球缺的底面直径。

30 皮克定理

给定顶点坐标均是整点 (或正方形格子点) 的简单多边形，皮克定理说明了其面积 S 和内部格点数目 i、边上格点数目 b 的关系：

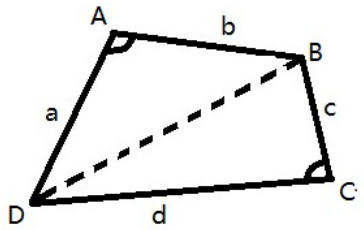
$$S = i + \frac{b}{2} - 1$$

31 欧拉公式

$$V - E + F = 2$$

其中 V 是顶点, E 是边, F 是面。
适用于所有多边形，无论维度。

32 婆罗摩笈多公式



已知四边形四边长 a, b, c, d，那么其面积为：

$$S = \sqrt{(p-a)(p-b)(p-c)(p-d) - abcdcos^2\theta}$$

其中 $p = \frac{a+b+c+d}{2}$。

33 点积

对于向量 a 和向量 b，有：

$$a = [a_1, a_2, \cdots, a_n]$$
$$b = [b_1, b_2, \cdots, b_n]$$

a 和 b 的点积公式为：

$$a \cdot b = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

几何意义
点积的几何意义用来表示或计算两个向量之间的夹角，以及向量 b 在向量 a 上的投影：

$$a \cdot b = |a| \cdot |b| \cos\theta$$

其含义为向量 a 在 b 上的投影长度和 $|b|$ 的乘积。
点积和两向量之间的夹角关系：
$a \cdot b > 0$，夹角范围：$[0°, 90°)$
$a \cdot b = 0$，两向量垂直
$a \cdot b < 0$，夹角范围：$(90°, 180°]$

34 叉积

我们令 $\overrightarrow{A}=(x_1,y_1,z_1),\overrightarrow{B}=(x_2,y_2,z_2)$，那么有：

$$\overrightarrow{A}\times\overrightarrow{B}=\begin{vmatrix} i & j & k \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix}=\begin{vmatrix} y_1 & z_1 \\ y_2 & z_2 \end{vmatrix} i-\begin{vmatrix} x_1 & z_1 \\ x_2 & z_2 \end{vmatrix} j+\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} k=(y_1z_2-z_1y_2)i-(x_1z_2-z_1x_2)j+(x_1y_2-y_1x_2)k$$

34.1 代数性质

对于任意三个向量 $\overrightarrow{A}$、$\overrightarrow{B}$、$\overrightarrow{C}$、$\overrightarrow{D}$ 有：

- $\overrightarrow{A}\times\overrightarrow{A}=\overrightarrow{0}$
- $\overrightarrow{A}\times\overrightarrow{0}=\overrightarrow{0}$
- $\overrightarrow{A}\times\overrightarrow{B}=-(\overrightarrow{B}\times\overrightarrow{A})$ (反交换律)
- $\overrightarrow{A}\times(\overrightarrow{B}+\overrightarrow{C})=\overrightarrow{A}\times\overrightarrow{B}+\overrightarrow{A}\times\overrightarrow{C}$，(加法的左分配律)
- $(\overrightarrow{A}+\overrightarrow{B})\times\overrightarrow{C}=\overrightarrow{A}\times\overrightarrow{C}+\overrightarrow{B}\times\overrightarrow{C}$，(加法的右分配律)
- $(\lambda\overrightarrow{A})\times\overrightarrow{B}=\lambda(\overrightarrow{A}\times\overrightarrow{B})=\overrightarrow{A}\times(\lambda\overrightarrow{B})$
- $\overrightarrow{A}\times\overrightarrow{B}+\overrightarrow{C}\times\overrightarrow{D}=(\overrightarrow{A}-\overrightarrow{C})\times(\overrightarrow{B}-\overrightarrow{D})+\overrightarrow{A}\times\overrightarrow{D}+\overrightarrow{C}\times\overrightarrow{B}$
- $$\left|\overrightarrow{A}\times\overrightarrow{B}\right|=\left|\overrightarrow{B}\times\overrightarrow{A}\right|$$
- $$\left|\overrightarrow{A}\times\overrightarrow{B}\right|^2=|\overrightarrow{A}|^2|\overrightarrow{B}|^2-(\overrightarrow{A}\cdot\overrightarrow{B})^2=\begin{vmatrix} \overrightarrow{A}\cdot\overrightarrow{A} & \overrightarrow{A}\cdot\overrightarrow{B} \\ \overrightarrow{A}\cdot\overrightarrow{B} & \overrightarrow{B}\cdot\overrightarrow{B} \end{vmatrix}$$

  (拉格朗日恒等式)

34.2 二维向量叉积

我们令 $\overrightarrow{A}=(x_1,y_1,0),\overrightarrow{B}=(x_2,y_2,0)$，那么有：

$$\overrightarrow{A}\times\overrightarrow{B}=(x_1y_2-y_1x_2)k$$

那么向量的模：

$$|\overrightarrow{A}\times\overrightarrow{B}|=|\overrightarrow{A}||\overrightarrow{B}|\sin\theta=(x_1y_2-x_2y_1)$$

其意义为两向量构成的平行四边形的有向面积。

34.3 二维向量叉积的几何意义

- $\overrightarrow{A}\times\overrightarrow{B}<0$ 时，$\overrightarrow{B}$ 对应的向量在 $\overrightarrow{A}$ 的顺时针方向 (右边)。
- $\overrightarrow{A}\times\overrightarrow{B}=0$ 时，$\overrightarrow{A}$、$\overrightarrow{B}$ 共线。
- $\overrightarrow{A}\times\overrightarrow{B}>0$ 时，$\overrightarrow{B}$ 在 $\overrightarrow{A}$ 的逆时针方向 (左边)。

35 直线

1. 一般式

$$Ax + By + C = 0$$

一般式说明了平面直角坐标系上一个一元二次方程表示一条直线。
斜率：$k = -\frac{A}{B}$。
法向量：$\overrightarrow{n} = (A, B)$。
方向向量：$\overrightarrow{d} = (B, -A)$。
x 轴上的截距：$-\frac{C}{A}$，y 轴上的截距：$-\frac{C}{B}$。

2. 点斜式

$$y - y_0 = k(x - x_0)$$

点斜式是由一个定点 $P(x_0, y_0)$ 和斜率 k 确定的直线方程。
斜率：k。
法向量：$\overrightarrow{n} = (k, -1)$。
方向向量：$\overrightarrow{d} = (1, k)$。
x 轴上的截距为：$-\frac{y_0}{k} + x_0$，y 轴上的截距为：$y_0 - kx_0$。

3. 斜截式

$$y = kx + b$$

斜截式是由斜率 k 和 y 轴上的截距 b 确定的直线方程。
斜率：k。
法向量：$\overrightarrow{n} = (k, -1)$。
方向向量：$\overrightarrow{d} = (1, k)$。
x 轴上的截距为：$-\frac{b}{k}$，y 轴上的截距为：$-b$。

4. 两点式

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$$

两点式是由已知的两个点 (x_1, y_1)、(x_2, y_2) 所确定的直线方程。
斜率：$k = \frac{y_2 - y_1}{x_2 - x_1}$
法向量：$\overrightarrow{n} = (y_2 - y_1, x_1 - x_2)$
方向向量：$\overrightarrow{d} = (x_2 - x_1, y_2 - y_1)$
x 轴上的截距：$\frac{x_1 y_2 - x_2 y_1}{y_2 - y_1}$，$y$ 轴上的截距：$\frac{x_2 y_1 - x_1 y_2}{x_2 - x_1}$
通过变形有：

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$$

等价于：

$$(y - y_1)(x_2 - x_1) = (x - x_1)(y_2 - y_1)$$

有：

$$(y_2 - y_1)x - (x_2 - x_1)y = y_1 x_2 - y_2 x_1$$

令：

$$\begin{aligned} a &= y_2 - y_1 \\ b &= x_2 - x_1 \\ c &= y_1 x_2 - y_2 x_1 \end{aligned}$$

这样就可以写成 $ax - by = c$ 的形式，并且当给出的两点均为整数时，a, b, c 均为整数。

5. 点向式

$$\frac{y - y_0}{b} = \frac{x - x_0}{a}$$

点向式是由已知的定点 $P(x_0, y_0)$ 和方向向量 $\vec{\mathrm{a}} = (a, b)$ 所确定的直线方程。
斜率：$k = \frac{b}{a}$
法向量：$\vec{n} = (b, -a)$
方向向量：$\vec{d} = (a, b)$
x 轴上的截距为：$\frac{x_1 y_2 - x_2 y_1}{y_2 - y_1}$，$y$ 轴上的截距为：$\frac{x_1 y_1 - x_1 y_2}{x_2 - x_1}$

6. 参数式

$$\begin{cases} x = x_0 + at \\ y = y_0 + bt \end{cases}$$

这里的参数是 t，是点向式的变式，也是由定点 $P(x_0, y_0)$ 和方向向量 $\vec{\mathrm{a}} = (a, b)$ 确定的。
斜率：$k = \frac{b}{a}$
法向量：$\vec{\mathrm{n}} = (b, -a)$
方向向量：$\vec{\mathrm{a}} = (a, b)$
x 轴上的截距为：$x_0 - \frac{a}{b} y_0$，y 轴上的截距为：$y_0 - \frac{b}{a} x_0$

7. 特别参数式

$$\begin{cases} x = x_0 + t cos\alpha \\ y = y_0 + t sin\alpha \end{cases}$$

这里的参数是 t，是参数式的特例，即以直线的倾角 α 为参数。斜率：$tan\alpha$
法向量：$\vec{\mathrm{n}} = (sin\alpha, -cos\alpha)$
方向向量：$\vec{\mathrm{a}} = (cos\alpha, sin\alpha)$
x 轴上的截距为：$x_0 - \frac{1}{tan\alpha} y_0$，$y$ 轴上的截距为：$y_0 - tan\alpha x_0$

8. 点法式

$$A(x - x_0) + B(y - y_0) = 0$$

点法式是由定点 $P(x_0, y_0)$ 和方向量 $\vec{\mathrm{n}} = (A, B)$ 直接确定的直线方程。
斜率：$-\frac{A}{B}$
法向量：$\vec{\mathrm{n}} = (A, B)$
方向向量：$\vec{\mathrm{a}} = (B, -A)$
x 轴上的截距为：$x_0 + \frac{B}{A} y_0$，y 轴上的截距为：$y_0 + \frac{A}{B} x_0$

9. 截距式

$$\frac{x}{a} + \frac{y}{b} = 1$$

截距式是由 x 轴上的截距 a 和 y 轴上的截距 b 直接确定的直线方程，当然必须要截距不为 0。

斜率：$k = -\frac{b}{a}$

法向量：$\vec{n} = (\frac{1}{a}, \frac{1}{b})$

方向向量：$\vec{a} = (a, -b)$

x 轴上的截距为：a，y 轴上的截距为：b

10. 点法式

$$xcos\theta + ysin\theta - p = 0$$

点法式是由直线与 y 轴的夹角 θ 与原点 O 到直线的距离 p，这里 $p > 0$，θ 是直线逆时针转向 y 轴的夹角。

斜率：$k = -tan\theta$

法向量：$\vec{n} = (cos\theta, sin\theta)$

方向向量：$\vec{a} = (sin\theta, -cos\theta)$

x 轴上的截距为：$\frac{p}{cos\theta}$，y 轴上的截距为：$\frac{p}{sin\theta}$

11. 切线式

$$x_0x + y_0y = r^2$$

切线式表达的几何意义是，以原点 O 为圆心，r 为半径，$P(x_0, y_0)$ 为切点的圆的切线。

斜率：$k = -\dfrac{x_0}{y_0}$

法向量：$\vec{n} = (x_0, y_0)$

方向向量：$\vec{a} = (y_0, -x_0)$

x 轴上的截距为：$\dfrac{r^2}{y_0}$，y 轴上的截距为：$\dfrac{r^2}{x_0}$

12. 直线与直线的位置关系

设有两条直线 l_1、l_2，斜率分别为 k_1、k_2，法向量分别为 $\vec{n_1}$、$\vec{n_2}$，放量向量分别为 $\vec{a_1}$、$\vec{a_2}$，那么则有：

两直线平行的判定：$l_1//l_2 \Leftrightarrow k_1 = k_2 \Leftrightarrow \vec{n_1}//\vec{n_2} \Leftrightarrow \vec{a_1}//\vec{a_2}$

两直线垂直的判定：$l_1 \perp l_2 \Leftrightarrow k_1 \cdot k_2 = -1 \Leftrightarrow \vec{n_1} \perp \vec{n_2} \Leftrightarrow \vec{a_1} \perp \vec{a_2}$

两直线的夹角 θ：$cos\theta = |\frac{k_1 - k_2}{1 + k_1k_2}| = |\frac{\vec{n_2} \cdot \vec{n_1}}{|\vec{n_1}| \cdot |\vec{n_2}|}| = |\frac{\vec{a_2} \cdot \vec{a_1}}{|\vec{a_1}| \cdot |\vec{a_2}|}|$

直线上两点间距离 d（主要用于求解弦长）：$d = \sqrt{1 + k^2}|x_2 - x_1| = \sqrt{1 + k^2}|y_2 - y_1| = |t_2 - t_1|$，这里的 t 就是参数方程中的 t，t 的几何意义便是直线上动点 (x, y) 到定点 (x_0, y_0) 的有向线段的数量。

总结：直线的斜率表达了直线与 x 轴的夹角，法向量是垂直于直线的向量，方向向量是平行于直线的向量，因此可以用斜率、法向量、方向向量来判断平行、垂直与夹角。

36 三角形相关

36.1 面积

- 叉积

$$\frac{1}{2}\overrightarrow{AB} \times \overrightarrow{AC}$$

- 海伦公式

$$S = \sqrt{p(p-a)(p-b)(p-c)}, \quad p = \frac{a+b+c}{2}$$

-

$$S = \frac{absinC}{2}$$

36.2 费马点

该点到三角形三个顶点的距离之和最小。

如果三角形有一个内角大于或等于 120°, 这个内角的顶点就是费马点；如果 3 个内角均小于 120°, 则在三角形内部对 3 边张角均为 120° 的点, 是三角形的费马点。

设三角形三条边为 a,b,c, 且不妨假设 $a \le b \le c$，那么费马点到三点的距离和为:

- 有一个内角大于 120°, 距离为 $a+b$
- 三个内角均小于 120°, 距离为 $\sqrt{\frac{a^2+b^2+c^2+4\sqrt{3}\cdot s}{2}}$

36.3 外心

三边中垂线交点，到三角形三个顶点距离相同, 外接圆圆心。

36.4 内心

角平分线的交点，到三角形三边的距离相同, 内切圆圆心。

36.5 垂心

三条高线的交点。

36.6 重心

三条中线的交点，到三角形三顶点距离的平方和最小的点，三角形内到三边距离之积最大的点

37 欧拉四面体公式

HDU 1411

$$V^2=\frac{1}{36}\begin{vmatrix} p^2 & \frac{p^2+q^2-n^2}{2} & \frac{p^2+r^2-m^2}{2} \\ \frac{p^2+q^2-n^2}{2} & q^2 & \frac{q^2+r^2-l^2}{2} \\ \frac{p^2+r^2-m^2}{2} & \frac{q^2+r^2-l^2}{2} & r^2 \end{vmatrix}$$

```
#include <bits/stdc++.h>
using namespace std;
using db = double;

// 求四面体的体积, 欧拉公式
// p, q, r 为AB, AC, AD, n, m, l 为BC, BD, CD
db V(db p, db q, db r, db n, db m, db l) {
    db rr = r * r, mm = m * m, nn = n * n;
    db pp = p * p, qq = q * q, ll = l * l;
    db x1 = (pp + qq - nn) / 2.0, x2 = (pp + rr - mm) / 2.0, x3 = (qq + rr -
        ll) / 2.0;
    db v = pp * (qq * rr - x3 * x3) - x1 * (x1  * rr - x2 * x3) + x2 * (x1 *
        x3 - qq * x2);
    return sqrt(v) / 6.0;
}

int main() {
    db p, q, r, n, m, l;
    while (cin >> p >> q >> r >> n >> m >> l) {
        printf("%.4lf\n", V(p, q, r, n, m, l));
```

```
    }
    return 0;
}
```

38 一切的开始

```
typedef double db;
const db eps = 1e-10;
const db PI = acos(-1.0L);
//long long类型时使用
//int sgn(db x) { if (x == 0) return 0; return x < 0 ? -1 : 1; }
int sgn(db x) { if (fabs(x) < eps) return 0; return x < 0 ? -1 : 1; }
db sqr(db x) { return x * x; }
db fixOut(db x) { if (sgn(x) == 0) return 0; return x; }
db toRad(db angle) { return angle / 180 * PI; }
```

39 二维几何

39.1 点、线

```
struct Point {
    db x, y;
    Point(db x = 0, db y = 0) : x(x), y(y) {}
    void scan() { db _x, _y; scanf("%lf%lf", &_x, &_y); x = _x, y = _y; }
    void print() { printf("%.11f %.11f\n", x, y); }
    bool operator == (const Point &b) const { return sgn(x - b.x) == 0 &&
       sgn(y - b.y) == 0; }
    bool operator < (const Point &b) const { return sgn(x - b.x) == 0 ? sgn(
       y - b.y) < 0 : x < b.x; }
    Point operator + (const Point &b) const { return Point(x + b.x, y + b.y)
       ; }
    Point operator - (const Point &b) const { return Point(x - b.x, y - b.y)
       ; }
    Point operator * (const db &b) const { return Point(x * b, y * b); }
    Point operator / (const db &b) const { return Point(x / b, y / b); }
    db operator ^ (const Point &b) const { return x * b.y - y * b.x; }
    db operator * (const Point &b) const { return x * b.x + y * b.y; }
    db len() { return hypot(x, y); }
    db len2() { return x * x + y * y; }
    db dis(Point b) { return hypot(x - b.x, y - b.y); }
    db dis2(Point b) { return (x - b.x) * (x - b.x) + (y - b.y) * (y - b.y);
        }
    int quad() {
        int _x = sgn(x), _y = sgn(y);
        if (_x > 0 && _y >= 0) return 1;
        if (_x <= 0 && _y > 0) return 2;
        if (_x < 0 && _y <= 0) return 3;
        if (_x >= 0 && _y < 0) return 4;
    }
    //求PA和PB构成的夹角 lightOJ 1203
    db getRad(Point a, Point b) {
        Point p = *this;
        return fabs(atan2(fabs((a - p) ^ (b - p)), (a - p) * (b - p)));
    }
    //单位法向量
```

```
31      Point normal() { return Point(-y, x) / len(); }
32      //化为长度为r的向量
33      Point trunc(db r) {
34          db l = len();
35          if (!sgn(l)) return *this;
36          r /= l;
37          return Point(x * r, y * r);
38      }
39      //逆时针旋转90度
40      Point rotleft() { return Point(-y, x); }
41      //顺时针旋转90度
42      Point rotright() { return Point(y, -x); }
43      //绕点p旋转rad弧度
44      Point rotate(Point p, db rad) {
45          Point v = (*this) - p;
46          db c = cos(rad), s = sin(rad);
47          return Point(p.x + v.x * c - v.y * s, p.y + v.x * s + v.y * c);
48      }
49  };
50
51  struct Line {
52      Point s, e;
53      Line() {}
54      Line(Point s, Point e) : s(s), e(e) {}
55      void scan() { s.scan(); e.scan(); }
56      void print() { s.print(); e.print(); }
57      bool operator == (const Line &b) const { return s == b.s && e == b.e; }
58      //根据一个点和倾斜角确定直线,0 <= angle <= PI
59      Line(Point p, db rad) {
60          s = p;
61          if (sgn(rad - PI / 2) == 0) {
62              e = (s + Point(0, 1));
63          } else {
64              e = (s + Point(1, tan(rad)));
65          }
66      }
67      //ax + by + c = 0
68      Line(db a, db b, db c) {
69          if (sgn(a) == 0) {
70              s = Point(0, -c / b);
71              e = Point(1, -c / b);
72          } else if (sgn(b) == 0) {
73              s = Point(-c / a, 0);
74              e = Point(-c / a, 1);
75          } else {
76              s = Point(0, -c / b);
77              e = Point(1, (-c - a) / b);
78          }
79      }
80      void adjust() { if (e < s) swap(s, e); }
81      db length() { return s.dis(e); }
82      //返回直线倾斜角 0 <= rad <= PI
83      db getAngle() {
84          db k = atan2(e.y - s.y, e.x - s.x);
85          if (sgn(k) < 0) k += PI;
86          if (sgn(k - PI) == 0) k -= PI;
87          return k;
88      }
```

```
89      //点和直线关系
90      //1 在左侧
91      //2 在右侧
92      //3 在直线上
93      int relationPoint(Point p) {
94          int c = sgn((p - s) ^ (e - s));
95          if (c < 0) return 1;
96          if (c > 0) return 2;
97          return 3;
98      }
99      //判断点是否在线段上
100     bool pointOnSeg(Point p) { return sgn((p - s) ^ (e - s)) == 0 && sgn((p
            - s) * (p - e)) <= 0; }
101     //判断两向量是否平行
102     bool parallel(Line b) { return sgn((e - s) ^ (b.e - b.s)) == 0; }
103     //两线段相交判断
104     //2 规范相交
105     //1 非规范相交
106     //0 不相交
107     int segCrossSeg(Line b) {
108         int d1 = sgn((e - s) ^ (b.s - s));
109         int d2 = sgn((e - s) ^ (b.e - s));
110         int d3 = sgn((b.e - b.s) ^ (s - b.s));
111         int d4 = sgn((b.e - b.s) ^ (e - b.s));
112         if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) return 2;
113         return ((d1 == 0 && sgn((b.s - s) * (b.s - e)) <= 0) ||
114                 (d2 == 0 && sgn((b.e - s) * (b.e - e)) <= 0) ||
115                 (d3 == 0 && sgn((s - b.s) * (s - b.e)) <= 0) ||
116                 (d4 == 0 && sgn((e - b.s) * (e - b.e)) <= 0));
117     }
118     //直线和线段相交判断
119     //2 规范相交
120     //1 非规范相交
121     //0 不相交
122     int lineCrossSeg(Line b) {
123         int d1 = sgn((e - s) ^ (b.s - s));
124         int d2 = sgn((e - s) ^ (b.e - s));
125         if ((d1 ^ d2) == -2) return 2;
126         return (d1 == 0 || d2 == 0);
127     }
128     //两直线关系
129     //0 平行
130     //1 重合
131     //2 相交
132     int lineCrossLine(Line b) {
133         if ((*this).parallel(b)) return b.relationPoint(s) == 3;
134         return 2;
135     }
136     //求两直线交点
137     //要保证两直线不平行或重合
138     Point crossPoint(Line b) {
139         db a1 = (b.e - b.s) ^ (s - b.s);
140         db a2 = (b.e - b.s) ^ (e - b.s);
141         return Point((s.x * a2 - e.x * a1) / (a2 - a1), (s.y * a2 - e.y * a1
                ) / (a2 - a1));
142     }
```

```
    //点到直线的距离
    db disPointToLine(Point p) { return fabs((p - s) ^ (e - s)) / length(); }
    //点到线段的距离
    db disPointToSeg(Point p) {
        if (sgn((p - s) * (e - s)) < 0 || sgn((p - e) * (s - e)) < 0)
            return min(p.dis(s), p.dis(e));
        return disPointToLine(p);
    }
    //线段到线段的距离
    //如果两线段相交，距离为0
    db disSegToSeg(Line b) {
        return min(min(disPointToSeg(b.s), disPointToSeg(b.e)),
                   min(b.disPointToSeg(s), b.disPointToSeg(e)));
    }
    //返回点p在直线上的投影
    Point lineProg(Point p) { return s + (((e - s) * ((e - s) * (p - s))) / ((e - s).len2())); }
    //返回点p关于直线的对称点
    Point symmetryPoint(Point p) {
        Point q = lineProg(p);
        return Point(q.x * 2 - p.x, q.y * 2 - p.y);
    }
};
```

39.2 圆

```
struct Circle {
    Point p; db r;
    Circle() {}
    Circle(Point p, db r) : p(p), r(r) {}
    Circle(db x, db y, db r) : p(Point(x, y)), r(r) {}
    //三角形的相关圆
    //外接圆 opt = 0 可处理三点共线的情况
    //内切圆 opt = 1
    //UVA 12304
    Circle(Point a, Point b, Point c, int opt = 0) {
        if (opt == 0) {
            Point p0 = (a + b) / 2;
            Point v0 = (b - a).rotright();
            Point p1 = (a + c) / 2;
            Point v1 = (c - a).rotright();
            db t = ((p1 - p0) ^ v1) / (v0 ^ v1);
            p = p0 + v0 * t;
            r = p.dis(a);
        } else {
            Line u, v;
            db m = atan2(b.y - a.y, b.x - a.x), n = atan2(c.y - a.y, c.x - a.x);
            u.s = a;
            u.e = u.s + Point(cos((n + m) / 2), sin((n + m) / 2));
            v.s = b;
            m = atan2(a.y - b.y, a.x - b.x), n = atan2(c.y - b.y, c.x - b.x);
            v.e = v.s + Point(cos((n + m) / 2), sin((n + m) / 2));
            p = u.crossPoint(v);
            r = Line(a, b).disPointToSeg(p);
```

```
29          }
30      }
31      //阿波罗尼斯圆
32      //HDU 5130
33      Circle(Point a, Point b, db k) {
34          db mu = k * k - 1;
35          p = Point(-(b.x - k * k * a.x), -(b.y - k * k * a.y)) / mu;
36          db L = (a - b).len();
37          r = L * k / (1 - k * k);
38      }
39      void scan() { p.scan(); scanf("%lf", &r); }
40      void print() { printf("%.10f %.10f %.10f\n", p.x, p.y, r); }
41      bool operator == (const Circle &b) const { return (p == b.p) && sgn(r -
            b.r) == 0; }
42      bool operator < (const Circle &b) const { return ((p < b.p) || ((p == b.
            p) && sgn(r - b.r) < 0)); }
43      db area() { return PI * r * r; }
44      db circumference() { return PI * r * 2; }
45      //根据圆心角求圆上的点
46      Point getPoint(db rad) { return Point(p.x + cos(rad) * r, p.y + sin(rad)
            * r); }
47      //点和圆的关系
48      //0 圆外
49      //1 圆上
50      //2 圆内
51      int relationPoint(Point b) {
52          db dis = b.dis(p);
53          if (sgn(dis - r) < 0) return 2;
54          if (sgn(dis - r) == 0) return 1;
55          return 0;
56      }
57      //线段和圆的关系
58      //比较的是圆心到线段的距离和半径的关系
59      int relationSeg(Line b) {
60          db dis = b.disPointToSeg(p);
61          if (sgn(dis - r) < 0) return 2;
62          if (sgn(dis - r) == 0) return 1;
63          return 0;
64      }
65      //直线和圆的关系
66      //比较的是圆心到直线的距离和半径的关系
67      int relationLine(Line b) {
68          db dis = b.disPointToLine(p);
69          if (sgn(dis - r) < 0) return 2;
70          if (sgn(dis - r) == 0) return 1;
71          return 0;
72      }
73      //两圆的关系
74      //5 相离
75      //4 外切
76      //3 相交
77      //2 内切
78      //1 内含
79      //UVA12304
80      int relationCircle(Circle b) {
81          db dis = p.dis(b.p);
82          if (sgn(dis - r - b.r) > 0) return 5;
```

```
        if (sgn(dis - r - b.r) == 0) return 4;
        db l = fabs(r - b.r);
        if (sgn(dis - r - b.r) && sgn(dis - l) > 0) return 3;
        if (sgn(dis - l) == 0) return 2;
        if (sgn(dis - l) < 0) return 1;
        //不可达分支
        return 0;
    }
    //求直线和圆的交点 返回值为交点个数
    int pointCrossLine(Line b, Point &p1, Point &p2) {
        if (!(*this).relationLine(b)) return 0;
        Point a = b.lineProg(p);
        db d = b.disPointToLine(p);
        d = sqrt(r * r - d * d);
        if (sgn(d) == 0) {
            p1 = p2 = a;
            return 1;
        }
        p1 = a + (b.e - b.s).trunc(d);
        p2 = a - (b.e - b.s).trunc(d);
        return 2;
    }
    //求两个圆的交点 返回值为交点个数
    //UVA12304
    int pointCrossCircle(Circle b, Point &p1, Point &p2) {
        int rel = relationCircle(b);
        if (rel == 1 || rel == 5) return 0;
        db d = p.dis(b.p);
        db l = (d * d + r * r - b.r * b.r) / (d * 2);
        db h = sqrt(r * r - l * l);
        Point tmp = p + (b.p - p).trunc(l);
        p1 = tmp + ((b.p - p).rotleft().trunc(h));
        p2 = tmp + ((b.p - p).rotright().trunc(h));
        if (rel == 2 || rel == 4) return 1;
        return 2;
    }
    //过一点作圆的切线(先判断点和圆的关系) 返回切线条数
    //UVA12304
    int tangentLine(Point q, Line &u, Line &v) {
        int x = relationPoint(q);
        if (x == 2) return 0;
        if (x == 1) {
            u = Line(q, q + (q - p).rotleft());
            v = u;
            return 1;
        }
        db d = p.dis(q);
        db l = r * r / d;
        db h = sqrt(r * r - l * l);
        u = Line(q, p + ((q - p).trunc(l) + (q - p).rotleft().trunc(h)));
        v = Line(q, p + ((q - p).trunc(l) + (q - p).rotright().trunc(h)));
        return 2;
    }
    //得到与直线u相切, 过点q, 半径为r1的圆
    //UVA12304
    static int getCircle(Line u, Point q, db r1, Circle &c1, Circle &c2) {
        db dis = u.disPointToLine(q);
        if (sgn(dis - r1 * 2) > 0) return 0;
```

```
141         if (sgn(dis) == 0) {
142             c1.p = q + ((u.e - u.s).rotleft().trunc(r1));
143             c2.p = q + ((u.e - u.s).rotright().trunc(r1));
144             c1.r = c2.r = r1;
145             return 2;
146         }
147         Line u1 = Line((u.s + (u.e - u.s).rotleft().trunc(r1)), (u.e + (u.e
                - u.s).rotleft().trunc(r1)));
148         Line u2 = Line((u.s + (u.e - u.s).rotright().trunc(r1)), (u.e + (u.e
                - u.s).rotright().trunc(r1)));
149         Circle cc = Circle(q, r1);
150         Point p1, p2;
151         if (!cc.pointCrossLine(u1, p1, p2)) cc.pointCrossLine(u2, p1, p2);
152         c1 = Circle(p1, r1);
153         if (p1 == p2) {
154             c2 = c1;
155             return 1;
156         }
157         c2 = Circle(p2, r1);
158         return 2;
159     }
160     //同时与直线u, v相切, 半径为r1的圆
161     static int getCircle(Line u, Line v, db r1, Circle &c1, Circle &c2,
            Circle &c3, Circle &c4){
162         if(u.parallel(v)) return 0;
163         Line u1 = Line(u.s + (u.e - u.s).rotleft().trunc(r1), u.e + (u.e - u
                .s).rotleft().trunc(r1));
164         Line u2 = Line(u.s + (u.e - u.s).rotright().trunc(r1), u.e + (u.e -
                u.s).rotright().trunc(r1));
165         Line v1 = Line(v.s + (v.e - v.s).rotleft().trunc(r1), v.e + (v.e - v
                .s).rotleft().trunc(r1));
166         Line v2 = Line(v.s + (v.e - v.s).rotright().trunc(r1), v.e + (v.e -
                v.s).rotright().trunc(r1));
167         c1.r = c2.r = c3.r = c4.r = r1;
168         c1.p = u1.crossPoint(v1);
169         c2.p = u1.crossPoint(v2);
170         c3.p = u2.crossPoint(v1);
171         c4.p = u2.crossPoint(v2);
172         return 4;
173     }
174     //同时与不相交圆cx, cy相切, 半径为r1的圆
175     //UVA12304
176     static int getCircle(Circle cx, Circle cy, db r1, Circle &c1, Circle &c2
            ) {
177         Circle x(cx.p, r1 + cx.r), y(cy.p, r1 + cy.r);
178         int t = x.pointCrossCircle(y, c1.p, c2.p);
179         if (!t) return 0;
180         c1.r = c2.r = r1;
181         return t;
182     }
183     //得到过a, b两点, 半径为r1的两个圆
184     static int getCircle(Point a, Point b, db r1, Circle &c1, Circle &c2) {
185         Circle x(a, r1), y(b, r1);
186         int t = x.pointCrossCircle(y, c1.p, c2.p);
187         if (!t) return 0;
188         c1.r = c2.r = r1;
189         return t;
190     }
```

```
191     //两个圆的公切线
192     //返回切线的条数, -1表示无穷条
193     //a[i] b[i]分别是第i条切线在圆A, 和圆B上的切点
194     //UVA 10674
195     static int tangentLineCC(Circle A, Circle B, vector<Point> &a, vector<
            Point> &b) {
196         int cnt = 0;
197         //交换引用的话, 要交换回去
198         int Swap = 0;
199         if(A.r < B.r) { swap(A, B), swap(a, b); Swap = 1; }
200         db d = (A.p - B.p).len();
201         db d2 = (A.p - B.p).len2();
202         db rdiff = A.r - B.r;
203         db rsum = A.r + B.r;
204         if(sgn(d2 - rdiff * rdiff) < 0) return 0; //内含
205         db base = atan2(B.p.y - A.p.y, B.p.x - A.p.x);
206         if(sgn(d2) == 0 && sgn(A.r - B.r) == 0) return -1; //无限多条
207         if(sgn(d2 - rdiff * rdiff) == 0) { // 内切,一条切线
208             a.push_back(A.getPoint(base)); b.push_back(B.getPoint(base)); ++
                    cnt;
209             return 1;
210         }
211         //有外公切线
212         db ang = acos((A.r - B.r) / d);
213         a.push_back(A.getPoint(base + ang)); b.push_back(B.getPoint(base +
                ang)); ++cnt;
214         a.push_back(A.getPoint(base - ang)); b.push_back(B.getPoint(base -
                ang)); ++cnt;
215         if(sgn(d2 - rsum * rsum) == 0) { //一条内公切线
216             a.push_back(A.getPoint(base)); b.push_back(B.getPoint(PI + base)
                    ); ++cnt;
217         } else if(sgn(d2 - rsum * rsum) > 0) { // 两条内公切线
218             db ang = acos((A.r + B.r) / d);
219             a.push_back(A.getPoint(base + ang)); b.push_back(B.getPoint(PI +
                    base + ang)); ++cnt;
220             a.push_back(A.getPoint(base - ang)); b.push_back(B.getPoint(PI +
                    base - ang)); ++cnt;
221         }
222         if (Swap) swap(a, b);
223         return cnt;
224     }
225     //求两圆相交面积
226     db areaIntersectCircle(Circle b) {
227         int rel = relationCircle(b);
228         if (rel >= 4) return 0.0;
229         if (rel <= 2) return min(area(), b.area());
230         db d = p.dis(b.p);
231         db hf = (r + b.r + d) / 2.0;
232         db ss = sqrt(hf * (hf - r) * (hf - b.r) * (hf - d)) * 2.0;
233         db a1 = acos((r * r + d * d - b.r * b.r) / (2.0 * r * d)) * r * r;
234         db a2 = acos((b.r * b.r + d * d - r * r) / (2.0 * b.r * d)) * b.r *
                b.r;
235         return a1 + a2 - ss;
236     }
237     //求圆和三角形pab的相交面积
238     //POJ 3675 HDU 3982 HDU 2892
239     db areaIntersectTriangle(Point a, Point b) {
```

```
240         if (sgn((p - a) ^ (p - b)) == 0) return 0.0;
241         Point q[5];
242         int len = 0;
243         q[len++] = a;
244         Line l(a, b);
245         Point p1, p2;
246         if (pointCrossLine(l, q[1], q[2]) == 2) {
247             if (sgn((a - q[1]) * (b - q[1])) < 0) q[len++] = q[1];
248             if (sgn((a - q[2]) * (b - q[2])) < 0) q[len++] = q[2];
249         }
250         q[len++] = b;
251         if (len == 4 && sgn((q[0] - q[1]) * (q[2] - q[1])) > 0) swap(q[1], q
                [2]);
252         db res = 0;
253         for (int i = 0; i < len - 1; ++i) {
254             if (relationPoint(q[i]) == 0 || relationPoint(q[i + 1]) == 0) {
255                 db arg = p.getRad(q[i], q[i + 1]);
256                 res += r * r * arg / 2.0;
257             } else {
258                 res += fabs((q[i] - p) ^ (q[i + 1] - p)) / 2.0;
259             }
260         }
261         return res;
262     }
263 };
```

39.3 多边形

```
1  struct Polygon {
2      vector <Point> p;
3      Polygon() { p.clear(); }
4      Polygon(int n) { p.clear(); p.resize(n); }
5      int sze() { return p.size(); }
6      Point& operator[](int x) { return p[(x + sze()) % sze()]; }
7      void add(Point q) { p.push_back(q); }
8      void scan(int n = -1) { if (n == -1) scanf("%d", &n); (*this) = Polygon(
           n); for (int i = 0; i < n; ++i) p[i].scan(); }
9      vector<Line> getline() {
10         vector <Line> l(sze());
11         for (int i = 0; i < sze(); ++i) {
12             int j = (i + 1) % sze();
13             l[i] = Line(p[i], p[j]);
14         }
15         return l;
16     }
17     //极角排序
18     //POJ 2007
19     struct cmpNorm {
20         Point p;
21         cmpNorm(Point p) : p(p) {}
22         bool operator () (Point a, Point b) {
23             int d = sgn((a - p) ^ (b - p));
24             if (d == 0) {
25                 return sgn(a.dis2(p) - b.dis2(p) < 0);
26             } else {
27                 return d > 0;
28             }
```

```
29          }
30      };
31      void norm() {
32          Point mi = *p.begin();
33          for (int i = 1; i < sze(); ++i) mi = min(mi, p[i]);
34          sort(p.begin(), p.end(), cmpNorm(mi));
35      }
36      void norm(Point mi) { sort(p.begin(), p.end(), cmpNorm(mi)); }
37      //Andrw 求凸包
38      //opt=1 严格 不会有三点共线
39      //opt=0 非严格 有三点共线
40      //LightOJ 1203
41      //凸包上的点不超过sqrt(m), m为坐标系范围
42      void convexHull(int opt = 1) {
43          sort(p.begin(), p.end());
44          Polygon res; res.p.resize(sze() * 2);
45          int top = -1;
46          for (int i = 0; i < sze(); ++i) {
47              while (top > 0 && sgn((res[top] - res[top - 1]) ^ (p[i] - res[
                    top - 1])) < opt) --top;
48              res[++top] = p[i];
49          }
50          int pre = top;
51          for (int i = sze() - 2; i >= 0; --i) {
52              while (top > pre && sgn((res[top] - res[top - 1]) ^ (p[i] - res[
                    top - 1])) < opt) --top;
53              res[++top] = p[i];
54          }
55          if (top > 0) res.p.resize(top);
56          p = res.p;
57      }
58      //返回点集直径的平方 需要先求凸包
59      //POJ 2187
60      db convexDimmeter2() {
61          if (sze() == 1) return 0;
62          if (sze() == 2) return p[0].dis2(p[1]);
63          db ans = 0;
64          for (int u = 0, v = 1; u < sze(); ++u) {
65              //一条直线贴住边p[u] - p[u + 1]
66              while (1) {
67                  // 当Area(p[u], p[u + 1], p[v + 1]) <= Area(p[u], p[u + 1],
                        p[v])时停止旋转
68                  // 即Cross(p[u + 1] - p[u], p[v + 1] - p[u]) - Cross(p[u +
                        1] - p[u], p[v] - p[u]) <= 0
69                  // 根据Cross(A, B) - Cross(A, C) = Cross(A, B - C)
70                  // 化简得Cross(p[u + 1] - p[u], p[v + 1] - p[v]) <= 0
71                  int diff = sgn((p[(u + 1) % sze()] - p[u]) ^ (p[(v + 1) %
                        sze()] - p[v]));
72                  if (diff <= 0) {
73                      // u 和 v 是对踵点
74                      ans = max(ans, p[u].dis2(p[v]));
75                      //diff == 0 时 u 和 v + 1也是对踵点
76                      if (diff == 0) ans = max(ans, p[u].dis2(p[(v + 1) % sze
                            ()]));
77                      break;
78                  }
79                  v = (v + 1) % sze();
```

```
            }
        }
        return ans;
    }
    //计算周长
    //LightOJ 1239
    db getCircumference() {
        db res = 0;
        for (int i = 0; i < sze(); ++i) {
            int j = (i + 1) % sze();
            res += p[i].dis(p[j]);
        }
        return res;
    }
    //计算有向面积
    //顺时针 负 逆时针 正
    //POJ 3348
    db getArea() {
        db res = 0;
        for (int i = 0; i < sze(); ++i) {
            int j = (i + 1) % sze();
            res += (p[i] ^ p[j]);
        }
        return res / 2;
    }
    //得到方向
    //1 逆时针 0 顺时针
    bool getDir() { return sgn(getArea()) > 0; }
    //得到重心
    //HDU 1115
    Point getBarycenter() {
        Point res(0, 0);
        db area = 0;
        for (int i = 1; i < sze() - 1; ++i) {
            db tmp = (p[i] - p[0]) ^ (p[i + 1] - p[0]);
            if (sgn(tmp) == 0) continue;
            area += tmp;
            res.x += (p[0].x + p[i].x + p[i + 1].x) * tmp;
            res.y += (p[0].y + p[i].y + p[i + 1].y) * tmp;
        }
        if (sgn(area)) res = res / (area * 3);
        return res;
    }
    //判断是不是凸多边形
    //HDU 2108
    bool isConvex() {
        bool s[3] = {0, 0};
        for (int i = 0; i < sze(); ++i) {
            int j = (i + 1) % sze();
            int k = (j + 1) % sze();
            s[sgn((p[j] - p[i]) ^ (p[k] - p[i])) + 1] = true;
            if (s[0] && s[2]) return false;
        }
        return true;
    }
    //判断点和凸包的关系 O(log n)
    //1 边上
```

```
137     //0 内部
138     //-1 外部
139     //UVA 10256
140     int pointInConvex(Point q) {
141         int l = 1, r = sze() - 2;
142         while (r - l >= 0) {
143             int mid = (l + r) >> 1;
144             int a1 = sgn((p[mid] - p[0]) ^ (q - p[0]));
145             int a2 = sgn((p[mid + 1] - p[0]) ^ (q - p[0]));
146             if (a1 >= 0 && a2 <= 0) {
147                 int a3 = sgn((q - p[mid]) ^ (q - p[mid + 1]));
148                 if (a3 < 0) return -1;
149                 else if (a1 || a2 || a3) return 0;
150                 return 1;
151             } else if (a1 < 0) {
152                 r = mid - 1;
153             } else {
154                 l = mid + 1;
155             }
156         }
157         return -1;
158     }
159     //判断点和任意多边形的关系
160     //3 点上
161     //2 边上
162     //1 内部
163     //0 外部
164     int relationPoint(Point q) {
165         for (int i = 0; i < sze(); ++i) {
166             if (p[i] == q) return 3;
167         }
168         vector<Line> l(getline());
169         for (int i = 0; i < (int)l.size(); ++i) {
170             if (l[i].pointOnSeg(q)) return 2;
171         }
172         int cnt = 0;
173         for (int i = 0; i < sze(); ++i) {
174             int j = (i + 1) % sze();
175             int k = sgn((q - p[j]) ^ (p[i] - p[j]));
176             int u = sgn(p[i].y - q.y);
177             int v = sgn(p[j].y - q.y);
178             if (k > 0 && u < 0 && v >= 0) cnt++;
179             if (k < 0 && v < 0 && u >= 0) cnt--;
180         }
181         return cnt != 0;
182     }
183     //向量u切割凸多边形
184     //注意向量方向 保留的是向量左边的部分
185     //HDU 3982
186     Polygon convexCut(Line u) {
187         Polygon res;
188         for (int i = 0; i < sze(); ++i) {
189             int j = (i + 1) % sze();
190             int d1 = sgn((u.e - u.s) ^ (p[i] - u.s));
191             int d2 = sgn((u.e - u.s) ^ (p[j] - u.s));
192             if (d1 >= 0) res.add(p[i]);
193             if (d1 * d2 < 0) res.add(u.crossPoint(Line(p[i], p[j])));
```

```
194         }
195         return res;
196     }
197     //多边形和圆交的面积
198     //HDU3982 HDU2892 HDU5130
199     db areaIntersectCircle(Circle c) {
200         db res = 0;
201         for (int i = 0; i < sze(); ++i) {
202             int j = (i + 1) % sze();
203             if (sgn((p[j] - c.p) ^ (p[i] - c.p)) >= 0) {
204                 res += c.areaIntersectTriangle(p[i], p[j]);
205             } else {
206                 res -= c.areaIntersectTriangle(p[i], p[j]);
207             }
208         }
209         return fabs(res);
210     }
211     //多边形和圆的关系
212     //2 圆完全在多边形内
213     //1 圆在多边形里面, 碰到了多边形边界
214     //0 其它
215     //POJ 1584
216     int relationCircle(Circle c) {
217         vector<Line>l(getline());
218         int x = 2;
219         if (relationPoint(c.p) != 1) return 0; //圆心不在内部
220         for (int i = 0; i < sze(); ++i) {
221             if (c.relationSeg(l[i]) == 2) return 0;
222             if (c.relationSeg(l[i]) == 1) return 1;
223         }
224         return x;
225     }
226     //判断两凸包是否有交点
227     //UVA 10256
228     bool ConvexPolygonDisjoint(Polygon &b) {
229         for (int i = 0; i < sze(); ++i)
230             if (b.pointInConvex(p[i]) >= 0)
231                 return true;
232         for (int i = 0; i < b.sze(); ++i)
233             if (pointInConvex(b[i]) >= 0)
234                 return true;
235         for (int i = 0; i < sze(); ++i)
236             for(int j = 0; j < b.sze(); ++j) {
237                 int _i = (i + 1) % sze();
238                 int _j = (j + 1) % b.sze();
239                 if (Line(p[i], p[_i]).segCrossSeg(Line(b[j], b[_j])))
240                     return true;
241             }
242         return false;
243     }
244     //两凸包最值距离
245     //求最大 就将Min改成Max即可
246     //POJ 3608
247     db disConvexToConvex(Polygon &q) {
248         int _p = 0, _q = 0;
249         for (int i = 1; i < sze(); ++i)
250             if (sgn(p[i].y - p[_p].y) < 0) _p = i;
```

```
        for (int i = 1; i < q.sze(); ++i)
            if (sgn(q[i].y - q[_q].y) > 0) _q = i;
        db ans = p[_p].dis(q[_q]);
        for(int i = 0; i < sze(); ++i) {
            while (1) {
                int diff = sgn(((p[(_p + 1) % sze()] - p[_p]) ^ (q[(_q + 1)
                    % q.sze()] - p[_p])) - ((p[(_p + 1) % sze()] - p[_p]) ^ (
                    q[_q] - p[_p])));
                if (diff > 0) {
                    _q = (_q + 1) % q.sze();
                    continue;
                }
                ans = min(ans, Line(p[_p], p[(_p + 1) % sze()]).
                    disPointToSeg(q[_q]));
                ans = min(ans, Line(p[_p], p[(_p + 1) % sze()]).disSegToSeg(
                    Line(q[_q], q[(_q + 1) % q.sze()])));
                ans = min(ans, p[_p].dis(q[_q]));
                ans = min(ans, p[_p].dis(q[(_q + 1) % q.sze()]));
                _p = (_p + 1) % sze();
                break;
            }
        }
        return ans;
    }
    //凸包最大内接三角形面积 O(n^2)
    //HDU2202 CF682E
    Polygon convexMaxInnerTriangle() {
        if (sze() < 3) return Polygon();
        Polygon res(3);
        db ans = 0;
        for(int i = 0, j, k; i < sze(); ++i) {
            j = (i + 1) % sze();
            k = (j + 1) % sze();
            while((j != k) && (k != i)) {
                while(sgn(((p[j] - p[i]) ^ (p[(k + 1) % sze()] - p[i])) - ((
                    p[j] - p[i]) ^ (p[k] - p[i]))) > 0) k = (k + 1) % sze();
                db tmp = (p[j] - p[i]) ^ (p[k] - p[i]);
                if(tmp > ans) {
                    res[0] = p[i]; res[1] = p[j]; res[2] = p[k];
                    ans = tmp;
                }
                j = (j + 1) % sze();
            }
        }
        return res;
    }
    //凸包最小矩形覆盖
    //返回的矩形的点以逆时针给出
    //LuoguP3187 HDU5251
    Polygon convexMinRectangleCover() {
        Polygon res(4);
        db area = 1e18;
        int l = 1, r = 1, u = 1;
        for (int i = 0; i < sze(); ++i) {
            int _i = (i + 1) % sze();
            while (sgn(((p[_i] - p[i]) ^ (p[u] - p[i])) - ((p[_i] - p[i]) ^
                (p[(u + 1) % sze()] - p[i]))) < 0) u = (u + 1) % sze();
```

```
302             while (sgn((p[_i] - p[i]) * (p[r] - p[i]) - (p[_i] - p[i]) * (p
                    [(r + 1) % sze()] - p[i])) <= 0) r = (r + 1) % sze();
303             if (i == 0) l = r;
304             while (sgn((p[_i] - p[i]) * (p[l] - p[i]) - (p[_i] - p[i]) * (p
                    [(l + 1) % sze()] - p[i])) > 0) l = (l + 1) % sze();
305             Point A = Line(p[i], p[_i]).lineProg(p[l]);
306             Point B = Line(p[i], p[_i]).lineProg(p[r]);
307             Point v = (p[_i] - p[i]).rotleft().trunc(1);
308             db w = (B - A).len(), h = Line(p[i], p[_i]).disPointToLine(p[u])
                    ;
309             Point C = B + v * h;
310             Point D = A + v * h;
311             if (sgn(w * h - area) < 0) {
312                 area = w * h;
313                 res[0] = A; res[1] = B; res[2] = C; res[3] = D;
314             }
315         }
316         return res;
317     }
318     //得到上下凸壳
319     void getUpDownHull(Polygon &upHull, Polygon &downHull) {
320         sort(p.begin(), p.end());
321         upHull.p.clear(); downHull.p.clear();
322         for (int i = 0; i < sze(); ++i) {
323             while (upHull.sze() > 1 && sgn((p[i] - upHull[-1]) ^ (upHull[-2]
                        - upHull[-1])) >= 0) upHull.p.pop_back();
324             upHull.add(p[i]);
325             while (downHull.sze() > 1 && sgn((p[i] - downHull[-1]) ^ (
                    downHull[-2] - downHull[-1])) <= 0) downHull.p.pop_back();
326             downHull.add(p[i]);
327         }
328     }
329     //查询 a.x * b.x + a.y * b.y 的最大
330     //当前维护的是上凸壳 b.y > 0
331     //当前维护的是下凸壳 b.y < 0
332     //b.y = 0 上下凸壳都可
333     //其中a为凸包上的点, b为给定点
334     db querySlopeMax(Point b) {
335         int l = 1, r = sze() - 1, pos = 0;
336         while (r - l >= 0) {
337             int mid = (l + r) >> 1;
338             if (b * p[mid] > b * p[mid - 1]) {
339                 pos = mid;
340                 l = mid + 1;
341             } else {
342                 r = mid - 1;
343             }
344         }
345         return b * p[pos];
346     }
347 };
```

39.4 半平面交

- 半平面：平面的一半。一条直线会把平面分成两部分，就是两个半平面。对于半平面，常用的是用向量表示，默认向量的左侧为我们所需要的半平面。
- 半平面交：多个半平面求交集。其结果可能是一个凸多边形、无穷平面、直线、线段、点等。

- 多边形的核：如果多边形中存在一个区域使得在区域中可以看到多边形中任意位置（反之亦然），则这个区域就是多边形的核。可以用半平面交来求解。
- 极点：极点就是点 $(x,,y)$ 与原点的连线与 x 轴的夹角，其范围为 $[0,360]$

应用:

- 多个凸多边形面积交: BZOJ 2618
- 求多边形中可以放入的最大圆半径: POJ 3535
  考虑最大半径为 r，那么将多边形的核的每条边往里推 r，那么其多边形的核仍然存在，所以直接二分 r，然后半平面交判断其多边形的核是否存在即可

```
struct halfplane:public Line {
    db rad;
    //表示向量s -> e 逆时针(左侧)的半平面
    halfplane() {}
    halfplane(Point _s, Point _e) { s = _s, e = _e; }
    halfplane(Line v) { s = v.s, e = v.e; }
    void calcangle() { rad = atan2(e.y - s.y, e.x - s.x); }
    //将向量往逆时针方向平移d距离
    //POJ 3525
    Line move(db d) {
        db len = s.dis(e), dx = (s.y - e.y) / len * d, dy = (e.x - s.x) /
           len * d;
        return Line(s + Point(dx, dy), e + Point(dx, dy));
    }
    bool operator < (const halfplane &b) const { return rad < b.rad; }
};

struct halfplanes {
    vector <halfplane> hps, deq;
    halfplanes() { hps.clear(); }
    halfplanes(int n) { hps.clear(); hps.resize(n); }
    int st, ed;
    int sze() { return hps.size(); }
    halfplane& operator[] (int x) { return hps[(x + sze()) % sze()]; }
    void add(halfplane hp) { hps.push_back(hp); }
    void unique() {
        int m = 1;
        for (int i = 1; i < sze(); ++i) {
            if (sgn(hps[i].rad - hps[i - 1].rad) != 0) {
                hps[m++] = hps[i];
            } else if (sgn((hps[m - 1].e - hps[m - 1].s) ^ (hps[i].s - hps[m
                - 1].s)) > 0) {
                hps[m - 1] = hps[i];
            }
        }
        hps.resize(m);
    }
    bool halfPlaneIntersect() {
        for (int i = 0; i < sze(); ++i) hps[i].calcangle();
        sort(hps.begin(), hps.end()); unique();
        if (sze() < 2) return false;
        deq.resize(sze() + 5); st = 1, ed = 2;
        deq[1] = hps[0]; deq[2] = hps[1];
        for (int i = 2; i < sze(); ++i) {
            while (st < ed && hps[i].relationPoint(deq[ed].crossPoint(deq[ed
                - 1])) == 2) --ed;
```

```
            while (st < ed && hps[i].relationPoint(deq[st].crossPoint(deq[st
                + 1])) == 2) ++st;
            deq[++ed] = hps[i];
        }
        while (st < ed && deq[st].relationPoint(deq[ed].crossPoint(deq[ed -
            1])) == 2) --ed;
        if (st + 1 >= ed) return false;
        return true;
    }
    //得到多边形的核
    Polygon getConvex() {
        deq[st - 1] = deq[ed];
        Polygon po;
        for (int i = st; i <= ed; ++i)
            po.add(deq[i].crossPoint(deq[i - 1]));
        return po;
    }
};
```

39.5 K 圆覆盖

SPOJ - CIRUT
对于 $i \in [1,n]$，求被圆覆盖 i 次的面积并
该板子的 k 圆覆盖无法处理有相同圆的情况
时间复杂度 $O(n^2 \log n)$

```
#include <bits/stdc++.h>
using namespace std;
#define fi first
#define se second
typedef double db;
const db eps = 1e-8;
const db PI = acos(-1.0);
const int N = 1e3 + 10, INF = 0x3f3f3f3f;
int n;
db sqr(db x) { return x * x; }
int sgn(db x) { return fabs(x) < eps ? 0 : x < 0 ? -1 : 1; }

struct Point {
    db x, y;
    Point(db x = 0, db y = 0) : x(x), y(y) {}
    void scan() { db _x, _y; scanf("%lf%lf", &_x, &_y); x = _x, y = _y; }
    void print() { printf("%.11f %.11f\n", x, y); }
    bool operator == (const Point &b) const { return sgn(x - b.x) == 0 &&
        sgn(y - b.y) == 0; }
    bool operator < (const Point &b) const { return sgn(x - b.x) == 0 ? sgn(
        y - b.y) < 0 : x < b.x; }
    Point operator + (const Point &b) const { return Point(x + b.x, y + b.y)
        ; }
    Point operator - (const Point &b) const { return Point(x - b.x, y - b.y)
        ; }
    Point operator * (const db &b) const { return Point(x * b, y * b); }
    Point operator / (const db &b) const { return Point(x / b, y / b); }
    db operator ^ (const Point &b) const { return x * b.y - y * b.x; }
    db operator * (const Point &b) const { return x * b.x + y * b.y; }
    db len() { return hypot(x, y); }
    db dis(Point b) { return hypot(x - b.x, y - b.y); }
};
```

```
29
30 struct Circle {
31     Point p; db r;
32     Circle() {}
33     Circle(Point p, db r) : p(p), r(r) {}
34     Circle(db x, db y, db r) : p(Point(x, y)), r(r) {}
35     void scan() { p.scan(); scanf("%lf", &r); }
36     void print() { printf("%.10f %.10f %.10f\n", p.x, p.y, r); }
37     bool operator == (const Circle &b) const { return (p == b.p) && sgn(r -
          b.r) == 0; }
38     bool operator < (const Circle &b) const { return ((p < b.p) || ((p == b.
          p) && sgn(r - b.r) < 0)); }
39     db area() { return PI * r * r; }
40     db circumference() { return PI * r * 2; }
41     //两圆的关系
42     //5 相离
43     //4 外切
44     //3 相交
45     //2 内切
46     //1 内含
47     //UVA12304
48     int relationCircle(Circle b) {
49         db dis = p.dis(b.p);
50         if (sgn(dis - r - b.r) > 0) return 5;
51         if (sgn(dis - r - b.r) == 0) return 4;
52         db l = fabs(r - b.r);
53         if (sgn(dis - r - b.r) && sgn(dis - l) > 0) return 3;
54         if (sgn(dis - l) == 0) return 2;
55         if (sgn(dis - l) < 0) return 1;
56         //不可达分支
57         return 0;
58     }
59     //本圆是否被圆b包含
60     bool InnerCircle(Circle b) {
61         if (relationCircle(b) != 1) return 0;
62         return sgn(r - b.r) <= 0 ? 1 : 0;
63     }
64 };
65
66 struct CircleUnion {
67     vector <Circle> cir;
68     //ans[i] 表示被覆盖了i次的面积
69     vector <db> ans, pre;
70     int sze() { return cir.size(); }
71     CircleUnion(int n = 0) { cir.clear(); cir.resize(n); }
72     void scan(int n = -1) { if (n == -1) scanf("%d", &n); (*this) =
          CircleUnion(n); for (int i = 0; i < n; ++i) cir[i].scan(); }
73     void add(Circle c) { cir.push_back(c); }
74     Circle& operator[](int x) { return cir[x]; }
75     //返回 半径为r的圆 弧度为th的弓形的面积
76     db areaArc(db th, db r) { return 0.5 * r * r * (th - sin(th)); }
77     //去掉相同圆及内含圆
78     //求圆并的时候需要调用 然后答案就是 \sum ans[i]
79     void init_Unique() {
80         vector <bool> mark(sze() + 5, 0);
81         for (int i = 0; i < sze(); ++i) {
82             for (int j = 0; j < sze(); ++j) {
```

```
83                  if (i != j && !mark[j]) {
84                      if (cir[i] == cir[j] || cir[i].InnerCircle(cir[j])) {
85                          mark[i] = 1;
86                          break;
87                      }
88                  }
89              }
90          }
91          int cnt = 0;
92          for (int i = 0; i < sze(); ++i) {
93              if (!mark[i]) {
94                  cir[cnt++] = cir[i];
95              }
96          }
97          cir.resize(cnt);
98      }
99      void gao() {
100         ans.clear(); ans.resize(sze() + 5, 0);
101         pre.clear(); pre.resize(sze() + 5, 0);
102         vector < pair<db, int> > v;
103         for (int i = 0; i < sze(); ++i) {
104             v.clear();
105             v.push_back(make_pair(-PI, 1));
106             v.push_back(make_pair(PI, -1));
107             for (int j = 0; j < sze(); ++j) {
108                 if (i != j) {
109                     Point q = (cir[j].p - cir[i].p);
110                     db ab = q.len(), ac = cir[i].r, bc = cir[j].r;
111                     if (sgn(ab + ac - bc) <= 0) {
112                         v.push_back(make_pair(-PI, 1));
113                         v.push_back(make_pair(PI, -1));
114                         continue;
115                     }
116                     if (sgn(ab - ac + bc) <= 0) continue;
117                     if (sgn(ab - ac - bc) > 0) continue;
118                     db th = atan2(q.y, q.x), fai = acos((ac * ac + ab * ab -
                            bc * bc) / (2.0 * ac * ab));
119                     db a0 = th - fai;
120                     if (sgn(a0 + PI) < 0) a0 += PI * 2;
121                     db a1 = th + fai;
122                     if (sgn(a1 - PI) > 0) a1 -= PI * 2;
123                     if (sgn(a0 - a1) > 0) {
124                         v.push_back(make_pair(a0, 1));
125                         v.push_back(make_pair(PI, -1));
126                         v.push_back(make_pair(-PI, 1));
127                         v.push_back(make_pair(a1, -1));
128                     } else {
129                         v.push_back(make_pair(a0, 1));
130                         v.push_back(make_pair(a1, -1));
131                     }
132                 }
133             }
134             sort(v.begin(), v.end());
135             int cur = 0;
136             for (int j = 0; j < (int)v.size(); ++j) {
137                 if (cur > 0 && sgn(v[j].fi - pre[cur])) {
138                     ans[cur] += areaArc(v[j].fi - pre[cur], cir[i].r);
139                     ans[cur] += 0.5 * (Point(cir[i].p.x + cir[i].r * cos(pre
```

```
                        [cur]), cir[i].p.y + cir[i].r * sin(pre[cur])) ^
                                    Point(cir[i].p.x + cir[i].r * cos(v[j
                                        ].fi), cir[i].p.y + cir[i].r * sin
                                        (v[j].fi)));
                }
                cur += v[j].se;
                if (cur > 0) pre[cur] = v[j].fi;
            }
        }
        for (int i = 1; i < sze(); ++i)
            ans[i] -= ans[i + 1];
    }
};

int main() {
    while (scanf("%d", &n) != EOF) {
        CircleUnion cu; cu.scan(n);
        cu.gao();
        for (int i = 1; i <= n; ++i) {
            printf("[%d] = %.3f\n", i, cu.ans[i]);
        }
        return 0;
    }
}
```

39.6 多边形面积交

Nowcoder 1112J
题意：给出一个直角三角形和一个矩形，求面积交
只能求两个多边形
多边形的边一定是要按逆时针方向给出

```
namespace PolyIntersectArea {
    //ConvexPolygonIntersectArea
    db CPIA(Polygon a, Polygon b) {
        Polygon p, tmp;
        p = b;
        for (int i = 0; i < a.sze() && b.sze() > 2; i++) {
            int sflag = sgn((a[i + 1] - a[i]) ^ (p[0] - a[i])), eflag;
            tmp = Polygon(0);
            for (int j = 0; j < b.sze(); j++, sflag = eflag) {
                if (sflag >= 0) tmp.add(p[j]);
                eflag = sgn((a[i + 1] - a[i]) ^ (p[j + 1] - a[i]));
                if ((sflag ^ eflag) == -2)
                    tmp.add(Line(a[i], a[i + 1]).crossPoint(Line(p[j], p[j +
                        1])));
            }
            p = tmp;
            b.p.resize(tmp.sze());
        }
        if(b.sze() < 3) return 0.0;
        return p.getArea();
    }
    //SimplePolygonIntersectArea
    db SPIA(Polygon a, Polygon b) {
        Polygon t1(3), t2(3);
        db res = 0, num1, num2;
        t1[0] = a[0], t2[0] = b[0];
```

```
        for(int i = 2; i < a.sze(); i++) {
            t1[1] = a[i - 1], t1[2] = a[i];
            num1 = sgn((t1[1] - t1[0]) ^ (t1[2] - t1[0]));
            if(num1 < 0) swap(t1[1], t1[2]);
            for(int j = 2; j < b.sze(); j++) {
                t2[1] = b[j - 1], t2[2] = b[j];
                num2 = sgn((t2[1] - t2[0]) ^ (t2[2] - t2[0]));
                if(num2 < 0) swap(t2[1], t2[2]);
                res += CPIA(t1, t2) * num1 * num2;
            }
        }
        return res;
    }
}

int main() {
    int x[4], y[4];
    while (scanf("%d%d%d%d", x, y, x + 1, y + 1) != EOF) {
        scanf("%d%d%d%d", x + 2, y + 2, x + 3, y + 3);
        vector <Polygon> poly(2);
        poly[0].add(Point(x[0], y[0]));
        poly[0].add(Point(x[0], y[1]));
        poly[0].add(Point(x[1], y[0]));
        poly[1].add(Point(x[2], y[2]));
        poly[1].add(Point(x[2], y[3]));
        poly[1].add(Point(x[3], y[3]));
        poly[1].add(Point(x[3], y[2]));
        for (int i = 0; i < 2; ++i) {
            if (poly[i].getDir() == 0)
                reverse(poly[i].p.begin(), poly[i].p.end());
        }
        printf("%.8f\n", PolyIntersectArea::CPIA(poly[0], poly[1]));
    }
    return 0;
}
```

39.7 多边形面积并

Gym101673A
题意：给出给出 n 个一般多边形，求 n 个多边形分别的面积和，以及面积并
时间复杂度 $O(m^2 \log m)$，m 为总点数
要求多边形的点按逆时针给出
LuoguP4406, 牛客 1112J

```
namespace PolyUnion {
    Point dir(Line ln) { return ln.e - ln.s; }
    db pos(Point p, Line ln){ return ((p - ln.s) * dir(ln)) / dir(ln).len2()
       ; }
    db gao(vector <Polygon> po) {
        int n = po.size();
        db res = 0;
        for (int i = 0; i < n; ++i) {
            for(int ii = 0; ii < po[i].sze(); ++ii) {
                Point A = po[i][ii], B = po[i][(ii + 1) % po[i].sze()];
                Line AB = Line(A,B);
                vector<pair<db, int> > c;
                for (int j = 0; j < n; ++j) if (i != j) {
                    for(int jj = 0; jj < po[j].sze(); ++jj) {
```

```
                        Point C = po[j][jj], D = po[j][(jj + 1) % po[j].sze
                            ()];
                        Line CD = Line(C,D);
                        int f1 = sgn((B - A) ^ (C - A));
                        int f2 = sgn((B - A) ^ (D - A));
                        if (!f1 && !f2) {
                            if (i < j && sgn((dir(AB) * dir(CD))) > 0) {
                                c.push_back(make_pair(pos(C, AB), 1));
                                c.push_back(make_pair(pos(D, AB), -1));
                            }
                            continue;
                        }
                        db s1 = (D - C) ^ (A - C);
                        db s2 = (D - C) ^ (B - C);
                        db t = s1 / (s1 - s2);
                        if (f1 >= 0 && f2 < 0) c.push_back(make_pair(t, 1));
                        if (f1 < 0 && f2 >= 0) c.push_back(make_pair(t,-1));
                    }
                }
                c.push_back(make_pair(0.0, 0));
                c.push_back(make_pair(1.0, 0));
                sort(c.begin(), c.end());
                db s = 0.5 * (A ^ B), z = min(max((db)c[0].second, (db)0.0),
                    (db)1.0);
                for(int j = 1, k = c[0].second; j < (int)c.size(); ++j) {
                    db w = min(max(c[j].first, (db)0.0), (db)1.0);
                    if (k == 0) res += s * (w - z);
                    k += c[j].second;
                    z = w;
                }
            }
        }
        return fabs(res);
    }
}

int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        vector <Polygon> poly;
        poly.resize(n);
        db tot = 0;
        for (int i = 0; i < n; ++i) {
            poly[i].scan();
            tot += fabs(poly[i].getArea());
        }
        printf("%.10f %.10f\n", tot, PolyUnion::gao(poly));
    }
    return 0;
}
```

40 三维几何

40.1 点、线、面

```
struct Point3 {
    db x, y, z;
```

```
3       Point3(db x = 0, db y = 0, db z = 0) : x(x), y(y), z(z) {}
4       void scan() { scanf("%lf%lf%lf", &x, &y, &z); }
5       void print() { printf("%.5f %.5f %.5f\n", x, y, z); }
6       bool operator == (const Point3 &b) const { return sgn(x - b.x) == 0 &&
            sgn(y - b.y) == 0 && sgn(z - b.z) == 0; }
7       bool operator < (const Point3 &b) const { return sgn(x - b.x) == 0 ? (
            sgn(y - b.y == 0) ? sgn(z - b.z) < 0 : y < b.y) : x < b.x; }
8       Point3 operator + (const Point3 &b) const { return Point3(x + b.x, y + b
            .y, z + b.z); }
9       Point3 operator - (const Point3 &b) const { return Point3(x - b.x, y - b
            .y, z - b.z); }
10      Point3 operator * (const db &k) const { return Point3(x * k, y * k, z *
            k); }
11      Point3 operator / (const db &k) const { return Point3(x / k, y / k, z /
            k); }
12      db operator * (const Point3 &b) const { return x * b.x + y * b.y + z * b
            .z; }
13      Point3 operator ^ (const Point3 &b) const { return Point3(y * b.z - z *
            b.y, z * b.x - x * b.z, x * b.y - y * b.x); }
14      db len() { return sqrt(x * x + y * y + z * z); }
15      db len2() { return x * x + y * y + z * z; }
16      db dis(Point3 b) { return sqrt((x - b.x) * (x - b.x) + (y - b.y) * (y -
            b.y) + (z - b.z) * (z - b.z)); }
17      db dis2(Point3 b) { return (x - b.x) * (x - b.x) + (y - b.y) * (y - b.y)
             + (z - b.z) * (z - b.z); }
18      db rad(Point3 a, Point3 b) {
19          Point3 p = (*this);
20          return acos(((a - p) * (b - p)) / (a.dis(p) * b.dis(p)));
21      }
22      //变换长度
23      Point3 trunc(db r) {
24          db l = len();
25          if (!sgn(l)) return *this;
26          r /= l;
27          return (*this) * r;
28      }
29  };
30
31  struct Line3 {
32      Point3 s, e;
33      Line3() {}
34      Line3(Point3 s, Point3 e) : s(s), e(e) {}
35      void scan() { s.scan(); e.scan(); }
36      void print() { s.print(); e.print(); }
37      bool operator == (const Line3 &b) { return s == b.s && e == b.e; }
38      db len() { return s.dis(e); }
39      //点到直线距离
40      db disPointToLine(Point3 b) { return ((e - s) ^ (b - s)).len() / s.dis(e
            ); }
41      //点到线段距离
42      db disPointToSeg(Point3 b) {
43          if (sgn((b - s) * (e - s)) < 0 || sgn((b - e) * (s - e)) < 0)
44              return min(s.dis(b), e.dis(b));
45          return disPointToLine(b);
46      }
47      //点b 在直线上的投影
48      Point3 lineProg(Point3 b) { return s + (((e - s) * ((e - s) * (b - s)))
            / ((e - s).len2())); }
```

```
49      //点b 绕此向量逆时针rad角度
50      Point3 rotate(Point3 b, db rad) {
51          if (sgn(((s - b) ^ (e - b)).len()) == 0) return b;
52          Point3 f1 = (e - s) ^ (b - s);
53          Point3 f2 = (e - s) ^ f1;
54          db len = ((s - b) ^ (e - b)).len() / s.dis(e);
55          f1 = f1.trunc(len); f2 = f2.trunc(len);
56          Point3 h = b + f2;
57          Point3 pp = h + f1;
58          return h + ((b - h) * cos(rad)) + ((pp - h) * sin(rad));
59      }
60      bool pointOnSeg(Point3 b) { return sgn(((s - b) ^ (e - b)).len()) == 0
            && sgn((s - b) * (e - b)) == 0; }
61  };
62
63  struct Plane {
64      Point3 a, b, c, o;
65      Plane() {}
66      Plane(Point3 a, Point3 b, Point3 c, Point3 o) : a(a), b(b), c(c), o(o)
            {}
67      void scan() { a.scan(); b.scan(); c.scan(); o.scan(); }
68      //ax + by + cz + d = 0
69      Plane(db _a, db _b, db _c, db _d) {
70          o = Point3(_a, _b, _c);
71          if (sgn(_a) != 0)
72              a = Point3((-_d - _c - _d) / _a, 1, 1);
73          else if (sgn(_b) != 0)
74              a = Point3(1, (-_d - _c - _a) / _b, 1);
75          else if (sgn(_c) != 0)
76              a = Point3(1, 1, (-_d - _a - _b) / _c);
77      }
78      Point3 pvec() { return (b - a) ^ (c - a); }
79      //点是否在平面上
80      bool pointOnPlane(Point3 b) { return sgn((b - a) * o) == 0; }
81      //两平面夹角
82      db radPlane(Plane f) { return acos(o * f.o) / (o.len() * f.o.len()); }
83      //平面和直线的交点，返回值是交点个数
84      int crossLine(Line3 u, Point3 &p) {
85          db x = o * (u.e - a);
86          db y = o * (u.s - a);
87          db d = x - y;
88          if (sgn(d) == 0) return 0;
89          p = ((u.s * x) - (u.e * y)) / d;
90          return 1;
91      }
92      //点到平面最近点(投影)
93      Point3 pointToPlane(Point3 b) {
94          Line3 u = Line3(b, b + o);
95          crossLine(u, b);
96          return b;
97      }
98      //平面和平面的交线
99      int crossPlane(Plane f, Line3 &u) {
100         Point3 oo = o ^ f.o;
101         Point3 v = o ^ oo;
102         db d = fabs(f.o * v);
103         if (sgn(d) == 0) return 0;
104         Point3 q = a + (v * (f.o * (f.a - a)) / d);
```

```
105         u = Line3(q, q + oo);
106         return 1;
107     }
108 };
```

40.2 三维凸包

- 求重心不要用去重随机
- 求表面多边形数不要用扰动法

40.2.1 传统法

```
1   struct CH3D {
2       struct face {
3           //表示凸包一个面上的三个点的编号
4           int a, b, c;
5           //表示该面是否属于最终的凸包上的面
6           bool ok;
7       };
8       vector<Point3> P;
9       //凸包表面的三角形
10      vector <face> F;
11      //凸包表面的三角形数
12      int num;
13      vector <vector<int> > g;
14      CH3D(int n = 0) { P.clear(); P.resize(n); }
15      void scan(int n = -1) { if (n == -1) scanf("%d", &n); (*this) = CH3D(n);
            for (int i = 0; i < n; ++i) P[i].scan(); }
16      int sze() { return P.size(); }
17      Point3& operator [](int x) {return P[(x + sze()) % sze()]; }
18      void add(Point3 q) { P.push_back(q); }
19      //三角形面积 * 2
20      db areaTriangle(Point3 a, Point3 b, Point3 c) { return ((b - a) ^ (c - a
            )).len(); }
21      //四面体有向面积 * 6
22      db volume(Point3 a, Point3 b, Point3 c, Point3 d) { return ((b - a) ^ (c
            - a)) * (d - a); }
23      //正: 点在面同向
24      db dblcmp(Point3 &p, face &f) {
25          Point3 p1 = P[f.b] - P[f.a];
26          Point3 p2 = P[f.c] - P[f.a];
27          Point3 p3 = p - P[f.a];
28          return (p1 ^ p2) * p3;
29      }
30      void deal(int p, int a, int b) {
31          int f = g[a][b];
32          face add;
33          if (F[f].ok) {
34              if (dblcmp(P[p], F[f]) > eps)
35                  dfs(p, f);
36              else {
37                  add.a = b;
38                  add.b = a;
39                  add.c = p;
40                  add.ok = true;
41                  g[p][b] = g[a][p] = g[b][a] = num;
```

```
42                  F.push_back(add); ++num;
43              }
44          }
45      }
46      //递归搜索所有应该从凸包内删除的面
47      void dfs(int p, int now) {
48          F[now].ok = false;
49          deal(p, F[now].b, F[now].a);
50          deal(p, F[now].c, F[now].b);
51          deal(p, F[now].a, F[now].c);
52      }
53      bool same(int s, int t) {
54          Point3 &a = P[F[s].a];
55          Point3 &b = P[F[s].b];
56          Point3 &c = P[F[s].c];
57          return fabs(volume(a, b, c, P[F[t].a])) < eps &&
58                 fabs(volume(a, b, c, P[F[t].b])) < eps &&
59                 fabs(volume(a, b, c, P[F[t].c])) < eps;
60      }
61      //构建三维凸包 O(n^2)
62      void create() {
63          F.clear(); num = 0;
64          g.clear(); g.resize(sze() + 1, vector<int>(sze() + 1, 0));
65          face add;
66          //保证前4个点不共面
67          int ok = false;
68          for (int i = 1; i < sze(); ++i) {
69              if (!(P[0] == P[i])) {
70                  swap(P[1], P[i]);
71                  ok = true;
72                  break;
73              }
74          }
75          if (!ok) return;
76          ok = false;
77          for (int i = 2; i < sze(); ++i) {
78              if (((P[1] - P[0]) ^ (P[i] - P[0])).len() > eps) {
79                  swap(P[2], P[i]);
80                  ok = true;
81                  break;
82              }
83          }
84          if (!ok) return;
85          ok = false;
86          for (int i = 3; i < sze(); ++i) {
87              if (fabs(((P[1] - P[0]) ^ (P[2] - P[0])) * (P[i] - P[0])) > eps)
                    {
88                  swap(P[3], P[i]);
89                  ok = true;
90                  break;
91              }
92          }
93          if (!ok) return;
94
95          for (int i = 0; i < 4; ++i) {
96              add.a = (i + 1) % 4;
97              add.b = (i + 2) % 4;
98              add.c = (i + 3) % 4;
```

```
            add.ok = true;
            if (dblcmp(P[i], add) > 0) swap(add.b, add.c);
            g[add.a][add.b] = g[add.b][add.c] = g[add.c][add.a] = num;
            F.push_back(add); ++num;
        }
        for (int i = 4; i < sze(); ++i) {
            for (int j = 0; j < num; ++j) {
                if (F[j].ok && dblcmp(P[i], F[j]) > eps) {
                    dfs(i, j);
                    break;
                }
            }
        }
        int tmp = num;
        num = 0;
        for (int i = 0; i < tmp; ++i) {
            if (F[i].ok) {
                F[num++] = F[i];
            }
        }
        F.resize(num);
    }
    //表面积
    //POJ3528 LuoguP4724
    db getArea() {
        db res = 0;
        if (sze() == 3) {
            Point3 p = (P[1] - P[0]) ^ (P[2] - P[0]);
            return p.len() / 2;
        }
        for (int i = 0; i < num; ++i)
            res += areaTriangle(P[F[i].a], P[F[i].b], P[F[i].c]);
        return res / 2.0;
    }
    //体积
    db volume() {
        db res = 0;
        Point3 tmp = Point3(0, 0, 0);
        for (int i = 0; i < num; ++i)
            res += volume(tmp, P[F[i].a], P[F[i].b], P[F[i].c]);
        return fabs(res / 6.0);
    }
    //表面三角形个数
    int triangleNum() { return num; }
    //表面多边形个数
    //HDU3662
    int polygonNum() {
        int res = 0;
        for (int i = 0; i < num; ++i) {
            int ok = 1;
            for (int j = 0; j < i; ++j) {
                if (same(i, j)) {
                    ok = 0;
                    break;
                }
            }
            res += ok;
        }
```

```
157         return res;
158     }
159     //重心
160     //HDU4273
161     Point3 getBarycenter() {
162         Point3 ans = Point3(0, 0, 0);
163         Point3 o = Point3(0, 0, 0);
164         db all = 0;
165         for (int i = 0; i < num; ++i) {
166             db vol = volume(o, P[F[i].a], P[F[i].b], P[F[i].c]);
167             ans = ans + (((o + P[F[i].a] + P[F[i].b] + P[F[i].c]) / 4.0) *
                    vol);
168             all += vol;
169         }
170         ans = ans / all;
171         return ans;
172     }
173     //点到面的距离
174     //HDU4273 HDU4266
175     db disPointToFace(Point3 p, int i) {
176         db tmp1 = fabs(volume(P[F[i].a], P[F[i].b], P[F[i].c], p));
177         db tmp2 = ((P[F[i].b] - P[F[i].a]) ^ (P[F[i].c] - P[F[i].a])).len();
178         return tmp1 / tmp2;
179     }
180 };
```

40.2.2 扰动法

```
1  struct CH3D {
2      db rdeps() {return (1. * rand() / RAND_MAX - 0.5) * eps;}
3      vector <Point3> P;
4      vector <vector<int> > vis;
5      struct face { int v[3]; };
6      vector <face> f, g;
7      Point3 Normal(face f) { return (P[f.v[1]] - P[f.v[0]]) ^ (P[f.v[2]] - P[
           f.v[0]]); }
8      bool See(face f, Point3 b) { return ((b - P[f.v[0]]) * Normal(f)) > 0; }
9      //三角形面积 * 2
10     db areaTriangle(face f) { return Normal(f).len(); }
11     //四面体有向面积 * 6
12     db volume(Point3 a, Point3 b, Point3 c, Point3 d) { return ((b - a) ^ (c
           - a)) * (d - a); }
13     CH3D(int n = 0) { P.clear(); P.resize(n); }
14     void scan(int n = -1) { if (n == -1) scanf("%d", &n); (*this) = CH3D(n);
           for (int i = 0; i < n; ++i) P[i].scan(); }
15     int sze() { return P.size(); }
16     Point3& operator [](int x) {return P[(x + sze()) % sze()]; }
17     void add(Point3 q) { P.push_back(q); }
18     void create() {
19         //防止出现四点共面
20         //扰动
21         for (int i = 0; i < sze(); ++i) P[i].x += rdeps(), P[i].y += rdeps()
               , P[i].z += rdeps();
22         //随机
23 //      sort(P.begin(), P.end());
24 //      P.erase(unique(P.begin(), P.end()), P.end());
25 //      random_shuffle(P.begin(), P.end());
```

```
26          f.clear(); f.resize(sze() * 2 + 5);
27          g.clear(); g.resize(sze() * 2 + 5);
28          vis.clear(); vis.resize(sze() + 1, vector<int>(sze() + 1, 0));
29          int cnt = 0, num = 0;
30          f[cnt++] = (face){0, 1, 2}; f[cnt++] = (face){2, 1, 0};
31          for (int i = 3; i < sze(); ++i) {
32              for (int j = 0, v; j < cnt; ++j) {
33                  if (!(v = See(f[j], P[i]))) g[num++] = f[j];
34                  for (int k = 0; k < 3; ++k) vis[f[j].v[k]][f[j].v[(k + 1) %
                        3]] = v;
35              }
36              for (int j = 0, x, y; j < cnt; ++j) {
37                  for (int k = 0; k < 3; ++k) {
38                      x = f[j].v[k], y = f[j].v[(k + 1) % 3];
39                      if (vis[x][y] && !vis[y][x]) g[num++] = (face){x, y, i};
40                  }
41              }
42              for (int j = 0; j < num; ++j) f[j] = g[j];
43              cnt = num; num = 0;
44          }
45          f.resize(cnt); g.clear();
46      }
47      //表面积
48      //BZOJ 1209
49      db getArea() {
50          db res = 0;
51          for (int i = 0; i < (int)f.size(); ++i)
52              res += areaTriangle(f[i]);
53          return fabs(res / 2.0);
54      }
55      //体积
56      db getVolume() {
57          db res = 0;
58          Point3 tmp = Point3(0, 0, 0);
59          for (int i = 0; i < (int)f.size(); ++i) {
60              res += volume(tmp, P[f[i].v[0]], P[f[i].v[1]], P[f[i].v[2]]);
61          }
62          return fabs(res / 6.0);
63      }
64      //表面三角形个数
65      int triangleNum() { return f.size(); }
66      bool same(int s, int t) {
67          Point3 &a = P[f[s].v[0]];
68          Point3 &b = P[f[s].v[1]];
69          Point3 &c = P[f[s].v[2]];
70          return fabs(volume(a, b, c, P[f[t].v[0]])) < eps &&
71                 fabs(volume(a, b, c, P[f[t].v[1]])) < eps &&
72                 fabs(volume(a, b, c, P[f[t].v[2]])) < eps;
73      }
74      //表面多边形个数
75      //HDU3662
76      int polygonNum() {
77          int res = 0;
78          for (int i = 0; i < (int)f.size(); ++i) {
79              int ok = 1;
80              for (int j = 0; j < i; ++j) {
81                  if (same(i, j)) {
82                      ok = 0;
```

```
                    break;
                }
            }
            res += ok;
        }
        return res;
    }
    //重心
    //HDU4273
    Point3 getBarycenter() {
        Point3 ans = Point3(0, 0, 0);
        Point3 o = Point3(0, 0, 0);
        db all = 0;
        for (int i = 0; i < (int)f.size(); ++i) {
            db vol = volume(o, P[f[i].v[0]], P[f[i].v[1]], P[f[i].v[2]]);
            ans = ans + (((o + P[f[i].v[0]] + P[f[i].v[1]] + P[f[i].v[2]]) /
                4.0) * vol);
            all += vol;
        }
        ans = ans / all;
        return ans;
    }
    //点到面的距离
    //HDU4273 HDU4266
    db disPointToFace(Point3 p, int i) {
        db tmp1 = fabs(volume(P[f[i].v[0]], P[f[i].v[1]], P[f[i].v[2]], p));
        db tmp2 = ((P[f[i].v[1]] - P[f[i].v[0]]) ^ (P[f[i].v[2]] - P[f[i].v
            [0]])).len();
        return tmp1 / tmp2;
    }
};
```

40.3 判断四点共面

构造行列式

$$\begin{vmatrix} x_2-x_1 & y_2-y_1 & z_2-z_1 \\ x_3-x_2 & y_3-y_2 & z_3-z_2 \\ x_4-x_3 & y_4-y_3 & z_4-z_2 \end{vmatrix} = 0$$

即其中一个向量可以由其他两个构成。

51node 1265

```
#include <bits/stdc++.h>
using namespace std;

struct Point {
    double x, y, z;
    Point() {}
    Point(double _x, double _y, double _z) { x = _x, y = _y, z = _z; }
    void input() { scanf("%lf %lf %lf", &x, &y, &z); }
    Point operator - (const Point &other) const { return {x - other.x, y -
        other.y, z - other.z}; }
} t[10], p[10];

int main() {
    int _T;
    scanf("%d", &_T);
    while (_T--) {
```

```
        for (int i = 1; i <= 4; ++i) t[i].input();
        for (int i = 1; i <= 3; ++i) p[i] = t[i] - t[i + 1];
        double res = 0;
        res += p[1].x * (p[2].y * p[3].z - p[2].z * p[3].y);
        res -= p[1].y * (p[2].x * p[3].z - p[2].z * p[3].x);
        res += p[1].z * (p[2].x * p[3].y - p[2].y * p[3].x);
        if (res == 0.0) {
            puts("Yes");
        } else {
            puts("No");
        }
    }
    return 0;
}
```

41 凸包

41.1 最大空凸包

时间复杂度 $O(n^3)$
HDU6219 POJ1259

```
#include <bits/stdc++.h>
using namespace std;
typedef double db;
const int N = 110;
const db eps = 1e-8;
int sgn(db x) { if (fabs(x) < eps) return 0; return x < 0 ? -1 : 1; }

struct Point {
    db x, y;
    Point(db x = 0, db y = 0) : x(x), y(y) {}
    void scan() { db _x, _y; scanf("%lf%lf", &_x, &_y); x = _x, y = _y; }
    void print() { printf("%.11f %.11f\n", x, y); }
    bool operator == (const Point &b) const { return sgn(x - b.x) == 0 &&
       sgn(y - b.y) == 0; }
    bool operator < (const Point &b) const { return sgn(x - b.x) == 0 ? sgn(
       y - b.y) < 0 : x < b.x; }
    Point operator + (const Point &b) const { return Point(x + b.x, y + b.y)
       ; }
    Point operator - (const Point &b) const { return Point(x - b.x, y - b.y)
       ; }
    Point operator * (const db &b) const { return Point(x * b, y * b); }
    Point operator / (const db &b) const { return Point(x / b, y / b); }
    db operator ^ (const Point &b) const { return x * b.y - y * b.x; }
    db operator * (const Point &b) const { return x * b.x + y * b.y; }
    db len() { return hypot(x, y); }
    db len2() { return x * x + y * y; }
    db dis(Point b) { return hypot(x - b.x, y - b.y); }
}s[N], p[N];

bool cmp(Point a, Point b) {
    if (sgn(a ^ b) != 0) return sgn(a ^ b) > 0;
    return a.len2() < b.len2();
}

int n, dp[N][N];

```

```
int gao(int m) {
    int res = 0;
    memset(dp, 0, sizeof dp);
    for (int i = 2; i <= m; ++i) {
        int now = i - 1;
        while (now >= 1 && sgn(p[i] ^ p[now]) == 0) --now;
        int ok = false;
        if (now == i - 1) ok = true;
        while (now >= 1) {
            int S = p[now] ^ p[i];
            int k = now - 1;
            while (k >= 1 && sgn((p[now] - p[i]) ^ (p[k] - p[now])) > 0) --k
                ;
            if (k >= 1) S += dp[now][k];
            if (ok) dp[i][now] = S;
            res = max(res, S);
            now = k;
        }
        if (!ok) continue;
        for (int j = 1; j <= i - 1; ++j) {
            dp[i][j] = max(dp[i][j], dp[i][j - 1]);
        }
    }
    return res;
}

int main() {
    int _T;
    scanf("%d", &_T);
    while (_T--) {
        scanf("%d", &n);
        for (int i = 1; i <= n; ++i) s[i].scan();
        int res = 0;
        for (int i = 1; i <= n; i++) {
            int m = 0;
            for (int j = 1; j <= n; j++) {
                if (s[j].y > s[i].y || (s[j].y == s[i].y && s[j].x >= s[i].x
                    ))
                    p[++m] = s[j] - s[i];
            }
            sort(p + 1, p + m + 1, cmp);
            res = max(res, gao(m));
        }
        printf("%.1f\n", res / 2.0);
    }
    return 0;
}
```

41.2 线段树维护凸包

LuoguP3309
题意:
维护两种操作:

- $A\ x\ y$, 加入向量 $\overrightarrow{(x,y)}$
- $Q\ x\ y\ l\ r$, 询问第 L 个加入的向量到第 R 个向量之间的向量与 $\overrightarrow{(x,y)}$ 点积的最大值

思路:

线段树维护凸包，发现插入的时候当且仅当 $pos = r$ 的时候这个线段树节点才算插入完成，这时候再求凸壳即可

总点数为 $O(n \log n)$

然后考虑询问，有 $res = x_1x_2 + y_1y_2$，有 $y_2 = -\frac{x_1}{y_1}x_2 + \frac{c}{y_1}$，相当于一条斜率为 $-\frac{x_1}{y_1}$ 的直线，要使得纵截距最大，答案肯定在凸包上，并且具有单调性

令 $x_1 >= 0$, 那么根据 y_1 的正负讨论；

- $y_1 = 0$, 那么肯定是凸包最左或者最右的点，上凸壳或者下凸壳都可
- $y_1 > 0$, 答案在上凸壳
- $y_1 < 0$, 答案在下凸壳

总的时间复杂度 $O(n \log^2 n)$

```
const int N = 4e5 + 10;
struct SEG {
    Polygon t[N << 2], upHull[N << 2], downHull[N << 2];
    void build(int id, int l, int r) {
        t[id].p.clear();
        if (l == r) return;
        int mid = (l + r) >> 1;
        build(id << 1, l, mid);
        build(id << 1 | 1, mid + 1, r);
    }
    void ins(int id, int l, int r, int pos, Point p) {
        t[id].add(p);
        if (pos == r) t[id].getUpDownHull(upHull[id], downHull[id]);
        if (l == r) return;
        int mid = (l + r) >> 1;
        if (pos <= mid) ins(id << 1, l, mid, pos, p);
        else ins(id << 1 | 1, mid + 1, r, pos, p);
    }
    db query(int id, int l, int r, int ql, int qr, Point p) {
        if (l >= ql && r <= qr) {
            if (p.y > 0) return upHull[id].querySlopeMax(p);
            else return downHull[id].querySlopeMax(p);
        }
        int mid = (l + r) >> 1;
        db Max = -1e18;
        if (ql <= mid) Max = max(Max, query(id << 1, l, mid, ql, qr, p));
        if (qr > mid) Max = max(Max, query(id << 1 | 1, mid + 1, r, ql, qr,
           p));
        return Max;
    }
}seg;

int main() {
    ll lst = 0;
    int n = 0;
    int q; char dataTp[10];
    scanf("%d%s", &q, dataTp);
    seg.build(1, 1, q);
    for (int i = 1; i <= q; ++i) {
        char op[5]; Point p; int l, r;
        scanf("%s", op);
        p.scan();
        p.x ^= lst; p.y ^= lst;
        if (op[0] == 'A') {
```

```
            ++n;
            seg.ins(1, 1, q, n, p);
        } else {
            scanf("%d%d", &l, &r);
            l ^= lst; r ^= lst;
            lst = seg.query(1, 1, q, l, r, p);
            printf("%lld\n", lst);
            if (dataTp[0] == 'E') lst = 0;
            lst &= 0x7fffffff;
        }
    }
    return 0;
}
```

41.3 时间线段树维护凸包

CF678F
题意:
维护三种操作:

- 1 a b, 往集合中加入点 (a,b)
- 2 i, 删除第 i 次操作添加的点
- 3 q, 查询集合中的点 (x,y) 与 $(q,1)$ 点积的最大值

随着一次加入和一次删除，每个点会有一个生存周期，以此建立时间线段树，将点插入到所属区间中，总的点数为 $O(n\log n)$
最后构建凸包,并且将每个查询可以拆成 $O(\log n)$ 个询问,每次可以在上凸壳上二分,询问复杂度为 $O(\log^2 n)$
总的时间复杂度为 $O(n\log^2 n)$，该算法为离线算法

```
struct SEG {
    Polygon t[N << 2], upHull[N << 2], downHull[N << 2];
    void build(int id, int l, int r) {
        t[id].p.clear();
        if (l == r) return;
        int mid = (l + r) >> 1;
        build(id << 1, l, mid);
        build(id << 1 | 1, mid + 1, r);
    }
    void ins(int id, int l, int r, int ql, int qr, Point q) {
        if (l >= ql && r <= qr) return t[id].add(q);
        int mid = (l + r) >> 1;
        if (ql <= mid) ins(id << 1, l, mid, ql, qr, q);
        if (qr > mid) ins(id << 1 | 1, mid + 1, r, ql, qr, q);
    }
    void buildConvex(int id, int l, int r) {
        t[id].getUpDownHull(upHull[id], downHull[id]);
        if (l == r) return;
        int mid = (l + r) >> 1;
        buildConvex(id << 1, l, mid);
        buildConvex(id << 1 | 1, mid + 1, r);
    }
    db query(int id, int l, int r, int pos, Point q) {
        db Max = -9e18;
        if (q.y > 0) Max = max(Max, upHull[id].querySlopeMax(q));
        else Max = max(Max, downHull[id].querySlopeMax(q));
        if (l == r) return Max;
        int mid = (l + r) >> 1;
```

```
29          if (pos <= mid) Max = max(Max, query(id << 1, l, mid, pos, q));
30          else Max = max(Max, query(id << 1 | 1, mid + 1, r, pos, q));
31          return Max;
32      }
33  }seg;
```

41.4 动态维护凸壳

```
1   struct DCH {
2       //维护上凸壳插入(x, y)
3       //维护下凸壳插入(x, -y)
4       struct Point {
5           db x, y;
6           Point () {}
7           Point (db _x, db _y): x(_x), y(_y) {}
8           bool operator == (const Point &t) const { return sgn(x - t.x) == 0
                && sgn(y - t.y) == 0; }
9           Point operator - (const Point &t) const { return Point(x - t.x, y -
                t.y); }
10          db operator ^ (const Point &t) const { return x * t.y - y * t.x; }
11          db dis(Point b) { return hypot(x - b.x, y - b.y); }
12      };
13      struct node {
14          Point p;
15          mutable Point next_p;
16          int opt;
17          //0 set中排序使用
18          //1 查询使用
19          node(Point p = Point(0, 0), Point next_p = Point(0, 0), int opt = 0)
                : p(p), next_p(next_p), opt(opt) {}
20          bool operator < (const node &other) const {
21              if (opt) {
22                  return sgn(p.x * (other.next_p.x - other.p.x) + p.y * (other
                        .next_p.y - other.p.y)) <= 0;
23              } else {
24                  return sgn(p.x - other.p.x) == 0 ? sgn(p.y - other.p.y) < 0
                        : p.x < other.p.x;
25              }
26          }
27      };
28      //点逆时针排序
29      set <node> st;
30      db area;
31      double perimeter;
32      typedef set <node>::iterator IT;
33      DCH() { st.clear(); area = 0; perimeter = 0; }
34      void init() { st.clear(); area = 0; perimeter = 0; }
35      int size() { return st.size(); }
36      //判断点是否在凸壳内
37      //如果要判断点是否在凸包内，维护上凸壳和下凸壳，点要满足同时在两个凸壳内
38      //CF 70D
39      bool inside(const Point &p) const {
40          if (st.empty()) return 0;
41          if (p.x < st.begin()->p.x) return 0;
42          if (p.x > (--st.end())->p.x) return 0;
43          IT lef = st.lower_bound(Point(p.x, -INF));
44          if (p.x == lef->p.x) return p.y <= lef->p.y;
```

```
45          IT rig = lef--;
46          return sgn((p - lef->p) ^ (rig->p - lef->p)) >= 0;
47      }
48      void insert(Point p) {
49          if (inside(p)) return;
50          st.insert(node(p, p));
51          IT itr = st.lower_bound(p);
52          if (itr != st.begin() && itr != (--st.end())) {
53              IT lef = itr; lef--;
54              IT rig = itr; rig++;
55              lef->next_p = itr->p;
56              itr->next_p = rig->p;
57              area += (lef->p - p) ^ (rig->p - p);
58              perimeter += p.dis(lef->p);
59              perimeter += p.dis(rig->p);
60              Point tmpp = lef->p;
61              perimeter -= tmpp.dis(rig->p);
62          }
63          if (itr == st.begin() && itr != (--st.end())) {
64              IT rig = itr; rig++;
65              itr->next_p = rig->p;
66              area += (rig->p.x - p.x) * (p.y + rig->p.y);
67              perimeter += p.dis(rig->p);
68          }
69          if (itr != st.begin() && itr == (--st.end())) {
70              IT lef = itr; lef--;
71              lef->next_p = itr->p;
72              area += (p.x - lef->p.x) * (p.y + lef->p.y);
73              perimeter += p.dis(lef->p);
74          }
75          if (itr != st.begin()) {
76              IT now = itr; now--;
77              while (now != st.begin()) {
78                  IT tmp = now; tmp--;
79                  if (sgn((tmp->p - p) ^ (now->p - p)) >= 0) {
80                      tmp->next_p = itr->p;
81                      area += (tmp->p - p) ^ (now->p - p);
82                      perimeter -= p.dis(now->p);
83                      Point tmpp = now->p;
84                      perimeter -= tmpp.dis(tmp->p);
85                      perimeter += p.dis(tmp->p);
86                      st.erase(now--);
87                  } else break;
88              }
89              if (now->p.x == p.x) {
90                  if (now != st.begin()) {
91                      IT tmp = now; tmp--;
92                      tmp->next_p = itr->p;
93                  }
94                  perimeter -= p.dis(now->p);
95                  if ((int)st.size() > 2 && now != st.begin()) {
96                      IT pre = now;
97                      if (now == st.begin()) pre = (--st.end());
98                      else --pre;
99                      Point tmpp = now->p;
100                     perimeter -= tmpp.dis(pre->p);
101                     perimeter += p.dis(pre->p);
102                 }
```

```
                st.erase(now);
            }
        }
        if (itr != (--st.end())) {
            IT now = itr; now++;
            while (now != (--st.end())) {
                IT tmp = now; tmp++;
                if (sgn((now->p - p) ^ (tmp->p - p)) >= 0) {
                    itr->next_p = tmp->p;
                    area += (now->p - p) ^ (tmp->p - p);
                    perimeter -= p.dis(now->p);
                    Point tmpp = now->p;
                    perimeter -= tmpp.dis(tmp->p);
                    perimeter += p.dis(tmp->p);
                    st.erase(now++);
                } else break;
            }
        }
    }
    void insert(db x, db y = 0) { insert(Point(x, y)); }
    //查询max(ax + by)
    //LOJ2197 正确性没什么问题，常数有点大
    //y >= 0 在上凸壳中查
    //y < 0 在下凸壳中查
    db query(db a, db b) {
        if (size() == 0) return -INF;
        IT it = st.upper_bound(node(Point(a, b), Point(0, 0), 1));
        return it->p.x * a + b * it->p.y;
    }
    //传入下凸壳，用于计算整个凸包的周长
    //LightOJ 1239
    db calcPerimeter(DCH &down) {
        Point up_begin = st.begin()->p;
        Point down_begin = down.st.begin()->p;
        Point up_end = (--st.end())->p;
        Point down_end = (--down.st.end())->p;
        down_begin.y *= -1;
        down_end.y *= -1;
        db res = perimeter + down.perimeter;
        if (!(up_begin == down_begin)) res += up_begin.dis(down_begin);
        if (!(up_end == down_end)) res += up_end.dis(down_end);
        return res;
    }
    //传入下凸壳，用于计算整个凸包的面积
    //POJ 3348
    db calcArea(DCH &down) {
        db res = area + down.area;
        return res / 2;
    }
};
```

41.5 动态维护凸包

```
struct DCH {
    struct Point {
        db x, y;
        double alpha;
```

```
5         Point () {}
6         Point (db _x, db _y, double _alpha = 0): x(_x), y(_y), alpha(_alpha)
              {}
7         Point operator - (const Point &t) const { return Point(x - t.x, y -
              t.y); }
8         bool operator == (const Point &t) const { return sgn(x - t.x) == 0
              && sgn(y - t.y) == 0; }
9         bool operator < (const Point &t) const { return alpha < t.alpha; }
10        db operator ^ (const Point &t) const { return x * t.y - y * t.x; }
11        db dis(Point t) { return hypot(x - t.x, y - t.y); }
12    };
13    //点逆时针排序
14    set <Point> st;
15    db area;
16    double perimeter;
17    double Ox, Oy;
18    typedef set <Point>::iterator IT;
19    void init() { st.clear(); }
20    int size() { return st.size(); }
21    double calcAlpha(Point p) { return atan2(1.0 * p.y - Oy, 1.0 * p.x - Ox)
          ; }
22    Point getPrev(Point p) {
23        IT it = st.lower_bound(p);
24        if (sgn(it->x - p.x) == 0 && sgn(it->y - p.y) == 0) return p;
25        if (it != st.begin()) return *--it;
26        return *--st.end();
27    }
28    Point getNext(Point p) {
29        IT it = st.upper_bound(p);
30        if (it == st.end()) return *st.begin();
31        return *it;
32    }
33    //判断点是否在凸包内
34    //Cf70D
35    bool isInside(Point p) {
36        if (size() < 3) return 0;
37        p.alpha = calcAlpha(p);
38        Point pre = getPrev(p);
39        Point nx = getNext(p);
40        return sgn((nx - pre) ^ (p - pre)) >= 0;
41    }
42    //计算周长
43    //LuoguP2521
44    db calcPerimeter() {
45        if (size() < 2) return 0;
46        if (size() == 2) {
47            Point q[2] = { *st.begin(), *++st.begin()};
48            return q[0].dis(q[1]) * 2;
49        }
50        return perimeter;
51    }
52    //计算面积
53    //POJ3348
54    db calcArea() {
55        if (size() < 3) return 0;
56        return area / 2;
57    }
58    //无法处理三点共线的情况，需要满足前三个点一定能构成凸包
```

```
59      void insert(Point p) {
60          if (size() < 2) {
61              p.alpha = size();
62              st.insert(p);
63              return;
64          }
65          if (size() == 2) {
66              Point q[3] = { *st.begin(), *++st.begin(), p };
67              double rands[] = { 0.4352341254, 0.8242147544, 0.3614575554 };
68              double sum = 0;
69              Ox = 0; Oy = 0;
70              for (int i = 0; i < 3; ++i) {
71                  sum += rands[i];
72                  Ox += rands[i] * q[i].x;
73                  Oy += rands[i] * q[i].y;
74              }
75              Ox /= sum;
76              Oy /= sum;
77              st.clear();
78              for (int i = 0; i < 3; ++i) {
79                  q[i].alpha = calcAlpha(q[i]);
80                  st.insert(q[i]);
81              }
82              IT it = st.begin();
83              for (int i = 0; i < 3; ++i) {
84                  q[i] = *it;
85                  ++it;
86              }
87              perimeter = 0;
88              area = 0;
89              perimeter += q[0].dis(q[1]) + q[1].dis(q[2]) + q[2].dis(q[0]);
90              area += (q[0] ^ q[1]) + (q[1] ^ q[2]) + (q[2] ^ q[0]);
91              return;
92          }
93          p.alpha = calcAlpha(p);
94          if (isInside(p)) return;
95          while (true) {
96              Point next1 = getNext(p);
97              st.erase(next1);
98              Point next2 = getNext(p);
99              if (sgn((next2 - p) ^ (next1 - p)) < 0) {
100                 st.insert(next1);
101                 break;
102             }
103             Point pre = getPrev(next1);
104             perimeter -= pre.dis(next1);
105             perimeter -= next1.dis(next2);
106             perimeter += pre.dis(next2);
107             area -= pre ^ next1;
108             area -= next1 ^ next2;
109             area += pre ^ next2;
110         }
111         while (true) {
112             Point prev1 = getPrev(p);
113             st.erase(prev1);
114             Point prev2 = getPrev(p);
115             if (sgn((p - prev2) ^ (prev1 - prev2)) < 0) {
116                 st.insert(prev1);
```

```
                break;
            }
            Point nx = getNext(prev1);
            perimeter -= prev1.dis(nx);
            perimeter -= prev2.dis(prev1);
            perimeter += prev2.dis(nx);
            area -= prev1 ^ nx;
            area -= prev2 ^ prev1;
            area += prev2 ^ nx;
        }
        Point pre = getPrev(p);
        Point nx = getNext(p);
        st.insert(p);
        perimeter -= pre.dis(nx);
        perimeter += pre.dis(p);
        perimeter += p.dis(nx);
        area -= pre ^ nx;
        area += pre ^ p;
        area += p ^ nx;
    }
    void insert(db x, db y) { insert(Point(x, y)); }
};
```

42 极角排序

42.1 HDU 3629

题意:

给出 n 个点，没有三点共线，问选择四个点能够组成凸四边形的个数。

思路:

考虑总数为 $\binom{n}{4}$，那么我们求反面，即四个点组成的凹包，那么形状必然是一个点在一个三角形里面。

那么我们枚举每个点 i，看看剩下的点能够组成多少个包含该点 i 的三角形，那么这就是包含点 i 的四点凹包的个数。

再反面考虑一次，我们对于每个点 i，考虑剩下的点能够组成多少个三角形不包含该点。

如果三个点不能包含点 i，那么必然存在一条过点 i 的直线，使得这三个点都在直线的一侧。

那么将剩下的点以点 i 为中心点进行极角排序，依次处理每个点 j，往后扫，扫到点 k 满足 j 和 k 的夹角大于 $180°$。

那么点 $[j+1,k-1]$ 中的每个点都可以和点 j 组成不包含点 i 的三角形，个数为 $\binom{k-j-1}{2}$，注意 k 具有单调性，指针不回退即可。

时间复杂度 $O(n^2 \log n)$

```
#include <bits/stdc++.h>
using namespace std;
typedef double db;
typedef long long ll;
const db eps = 1e-10;
const db PI = acos(-1.0);
int sgn(db x) { if (fabs(x) < eps) return 0; return x < 0 ? -1 : 1; }
int n;

struct Point {
    db x, y;
    Point(db x = 0, db y = 0) : x(x), y(y) {}
    void scan() { db _x, _y; scanf("%lf%lf", &_x, &_y); x = _x, y = _y; }
};

```

```
16 vector <Point> po;
17
18 ll C(int n, int m) {
19     if (n < m) return 0;
20     ll res = 1;
21     for (int i = n; i > n - m; --i)
22         res = res * i;
23     for (int i = 1; i <= m; ++i)
24         res /= i;
25     return res;
26 }
27
28 inline db calc(Point a, Point b) {
29     return atan2(b.y - a.y, b.x - a.x);
30 }
31
32 ll gao() {
33     ll res = 0;
34     for (int i = 0; i < n; ++i) {
35         vector <db> vec;
36         for (int j = 0; j < n; ++j) if (i != j) {
37             vec.push_back(calc(po[i], po[j]));
38             if (vec.end()[-1] < -eps) vec.end()[-1] += PI * 2;
39         }
40         sort(vec.begin(), vec.end());
41         for (int j = 0; j < n - 1; ++j)
42             vec.push_back(vec[j] + PI * 2);
43         ll now = C(n - 1, 3);
44         int r = 1;
45         for (int l = 0; l < n - 1; ++l) {
46             while (fabs(vec[r] - vec[l]) - PI < -eps) ++r;
47             now -= C(r - l - 1, 2);
48         }
49         res -= now;
50     }
51     return res;
52 }
53
54 int main() {
55     int _T; scanf("%d", &_T);
56     while (_T--) {
57         scanf("%d", &n);
58         po.resize(n);
59         for (int i = 0; i < n; ++i) po[i].scan();
60         ll res = C(n, 4) + gao();
61         printf("%lld\n", res);
62     }
63     return 0;
64 }
```

42.2 HDU 5784

题意:
给出 n 个点，统计锐角三角形的个数，可能有三点共线，但是没有重点。
思路:
考虑反面，即求直角、钝角、平角三角形的个数。
锐角三角形个数 $=\binom{n}{3}$ – 锐角个数 – 直角个数 – 三点共线个数

枚举点，求出其他点与该点构成的向量，做极角排序。

根据叉积等于 0，可以求出三点贡献的个数，但是要注意一组三点共线会被计算三次。

接下来对于每个向量，考虑其与左边所有向量 (叉积小于 0) 中夹角为直角和钝角的个数，为方便计算，此时我们先求出所有向量个数，再减去左边锐角 (点积大于 0) 个数。

对于环，将长度增常一倍，破环为链。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e4 + 5;
int n;

struct Point {
    ll x, y;
    Point(ll x = 0, ll y = 0) : x(x), y(y) {}
    Point operator - (const Point &b) const { return Point(x - b.x, y - b.y); }
    ll operator * (const Point &b) const { return x * b.x + y * b.y; }
    ll operator ^ (const Point &b) const { return x * b.y - y * b.x; }
    bool operator < (const Point &b) const {
        if (y * b.y <= 0) {
            if (y > 0 || b.y > 0) return y < b.y;
            if (y == 0 && b.y == 0) return x < b.x;
        }
        return x * b.y - y * b.x > 0;
    }
}p[N], q[N << 1];

ll gao() {
    ll line = 0, res = 0;
    for (int i = 0; i < n; ++i) {
        int tot = 0;
        for (int j = 0; j < n; ++j) {
            if (i == j) continue;
            q[tot++] = p[j] - p[i];
        }
        int tmp = 0;
        sort(q, q + tot);
        for (int j = 0; j < tot; ++j) q[tot + j] = q[j];
        //三点共线个数
        for (int j = 0; j < tot - 1; ++j) {
            if ((q[j] ^ q[j + 1]) == 0) ++tmp;
            else tmp = 0;
            line += tmp;
        }
        int cnt1 = 0, cnt2 = 0;
        for (int j = 0; j < tot; ++j) {
            //左边锐角个数
            while (cnt1 <= j || (cnt1 - tot < j && (q[cnt1] ^ (q[j])) < 0 &&
                (q[cnt1] * q[j]) > 0)) cnt1++;
            //左边角个数
            while (cnt2 <= j || (cnt2 - tot < j && (q[cnt2] ^ q[j]) < 0))
               cnt2++;
            res += cnt2 - cnt1;
        }
    }
    return res + line / 3;
}

```

```
int main () {
    while (~scanf("%d", &n)) {
        for (int i = 0; i < n; ++i) {
            scanf("%lld%lld", &p[i].x, &p[i].y);
        }
        ll ans = 1ll * n * (n - 1) * (n - 2) / 6 - gao();
        printf("%lld\n", ans);
    }
    return 0;
}
```

42.3 Hihocoder 1879

题意:
统计所有锐角三角形组成的面积和。
思路:
考虑锐角个数为 A，直角和钝角个数为 B。
那么锐角三角形个数为 $\frac{A-2B}{3}$
那么利用叉积，维护前缀和，计算面积即可。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef __int128 LL;
const int N = 5e3 + 10;
const ll mod = 998244353;
int n;
LL sumx[N], sumy[N];

ll qpow(ll base, ll n) {
    ll res = 1;
    while (n) {
        if (n & 1) res = res * base % mod;
        base = base * base % mod;
        n >>= 1;
    }
    return res;
}

struct Point {
    LL x, y;
    Point() {}
    Point(LL x, LL y) : x(x), y(y) {}
    Point operator-(const Point &other) const { return {x - other.x, y - other.y}; }
    Point operator+(const Point &other) const { return {x + other.x, y + other.y}; }
    LL operator^(const Point &other) const { return x * other.y - y * other.x; }
    bool operator<(const Point &other) const {
        if (y < 0 && other.y > 0) return true;
        if (y > 0 && other.y < 0) return false;
        if (y == 0 && other.y == 0) {
            return (x < 0 && other.x > 0);
        } else {
            return ((*this) ^ other) > 0;
        }
    }
```

```
}p[N], qrr[N];

ll gao() {
    LL res1 = 0, res2 = 0;
    for (int i = 1; i <= n; ++i) {
        int cnt = 0;
        for (int j = 1; j <= n; ++j) if (i != j) {
            qrr[++cnt] = p[j] - p[i];
        }
        sort(qrr + 1, qrr + 1 + cnt);
        for (int j = 1; j <= cnt; ++j) {
            qrr[j + cnt] = qrr[j];
        }
        sumx[0] = sumy[0] = 0;
        cnt <<= 1;
        for (int j = 1; j <= cnt; ++j) {
            sumx[j] = (sumx[j - 1] + qrr[j].x) % mod;
            sumy[j] = (sumy[j - 1] + qrr[j].y) % mod;
        }
        int q = 1, s = 1, t = 1;
        for (int j = 1; j < n; ++j) {
            q = max(q, j);
            Point _90 = Point(-qrr[j].y, qrr[j].x);
            Point _180 = Point(-qrr[j].x, -qrr[j].y);
            //重合分界线
            while (q < cnt && (qrr[q + 1] ^ qrr[j]) == 0) ++q;
            //(q, s] 锐角范围
            while (s < cnt && (s < q || ((qrr[s + 1] ^ _90) > 0 && (qrr[s +
                1] ^ qrr[j]) <= 0))) ++s;
            //(s, t] 直角钝角范围
            while (t < cnt && (t < s || ((qrr[t + 1] ^ _180) > 0 && (qrr[t +
                 1] ^ _90) <= 0))) ++t;
            res1 = (res1 + (sumy[s] - sumy[q] + mod) % mod * (qrr[j].x % mod
                ) % mod -
                    (sumx[s] - sumx[q] + mod) % mod * (qrr[j].y) % mod + mod
                        ) % mod;
            res2 = (res2 + (sumy[t] - sumy[s] + mod) % mod * (qrr[j].x % mod
                ) % mod -
                    (sumx[t] - sumx[s] + mod) % mod * (qrr[j].y) % mod + mod
                        ) % mod;
        }
    }
    return ((res1 - res2 * 2 % mod + mod) % mod * qpow(3, mod - 2) % mod) %
        mod;
}

int main() {
    int _T; scanf("%d", &_T);
    while (_T--) {
        scanf("%d", &n);
        for (int i = 1; i <= n; ++i) {
            ll x, y;
            scanf("%lld%lld", &x, &y);
            p[i].x = x;
            p[i].y = y;
        }
        printf("%lld\n", gao());
    }
```

```
87      return 0;
88  }
```

43 旋转卡壳

43.1 BZOJ 2739

题意:
给出一个凸包，求离每个点最远的点。
思路:
根据旋转卡壳求凸包的直径，知道离每个点最远的点有决策单调性，套决策单调性 dp 的分治做法即可。

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   typedef long long ll;
4   const int N = 1e6 + 10;
5   int n, ans[N];
6
7   struct Point {
8       ll x, y; int id;
9       ll dis(const Point &b) { return (x - b.x) * (x - b.x) + (y - b.y) * (y -
            b.y); }
10  }a[N];
11
12  bool check(int id, int i, int j) {
13      if (i < id || i > n + id) return 0;
14      if (j < id || j > n + id) return 1;
15      ll dis1 = a[id].dis(a[i]);
16      ll dis2 = a[id].dis(a[j]);
17      if (dis1 == dis2) return a[i].id < a[j].id;
18      return dis1 > dis2;
19  }
20
21  void gao(int l, int r, int L, int R) {
22      if (l > r) return;
23      int mid = (l + r) >> 1;
24      int pos = 0;
25      for (int i = L; i <= R; ++i) {
26          if (check(mid, i, pos))
27              pos = i;
28      }
29      ans[mid] = pos > n ? pos - n : pos;
30      gao(l, mid - 1, L, pos);
31      gao(mid + 1, r, pos, R);
32  }
33
34  int main() {
35      int _T; scanf("%d", &_T);
36      while (_T--) {
37          scanf("%d", &n);
38          for (int i = 1; i <= n; ++i) {
39              scanf("%lld%lld", &a[i].x, &a[i].y);
40              a[i].id = i;
41              a[i + n] = a[i];
42          }
43          gao(1, n, 1, n << 1);
44          for (int i = 1; i <= n; ++i)
45              printf("%d\n", ans[i]);
```

```
    }
    return 0;
}
```

44 模拟退火

44.1 HDU 3007

最小圆覆盖

```
#include <bits/stdc++.h>
using namespace std;
typedef double db;
const int N = 510;
struct Point {
    db x, y;
    Point () {}
    Point (db x, db y) : x(x), y(y) {}
    void scan() { scanf("%lf%lf", &x, &y); }
    db dis(const Point &p) const { return hypot(x - p.x, y - p.y); }
}arr[N];
int n;

void gao() {
    //pd越高, eps越高 炼的越久 pd < 1
    db t = 100, pd = 0.95, eps = 1e-7;
    db res = 1e18;
    Point now = arr[1]; int tmp = 1;
    while (t > eps) {
        for (int i = 1; i <= n; ++i)
            if (now.dis(arr[tmp]) < now.dis(arr[i]))
                tmp = i;
        db dis = now.dis(arr[tmp]);
        res = min(res, dis);
        now.x += (arr[tmp].x - now.x) / dis * t;
        now.y += (arr[tmp].y - now.y) / dis * t;
        t *= pd;
    }
    printf("%.2f %.2f %.2f\n", now.x, now.y, res);
}

int main() {
    while (scanf("%d", &n), n) {
        for (int i = 1; i <= n; ++i) arr[i].scan();
        gao();
    }
    return 0;
}
```

44.2 Gym 101981D

最小球覆盖

```
#include <bits/stdc++.h>
using namespace std;
#define db double
const int N = 1e2 + 10;
const db eps = 1e-8;
```

```
int n;
struct node {
    db x, y, z;
    node() { x = y = z = 0; }
    void scan() { scanf("%lf%lf%lf", &x, &y, &z); }
}a[N];

db dis(node a, node b) {
    return sqrt(1.0 * (a.x - b.x) * (a.x - b.x) + 1.0 * (a.y - b.y) * (a.y -
        b.y) + 1.0 * (a.z - b.z) * (a.z - b.z));
}

db solve() {
    db step = 100000, pd = 0.98, ans = 1e30;
    node c = node();
    int s = 1;
    while (step > eps) {
        for (int i = 1; i <= n; ++i) {
            if (dis(c, a[s]) < dis(c, a[i])) {
                s = i;
            }
        }
        db mt = dis(c, a[s]);
        ans = min(ans, mt);
        c.x += (a[s].x - c.x) / mt * step;
        c.y += (a[s].y - c.y) / mt * step;
        c.z += (a[s].z - c.z) / mt * step;
        step *= pd;
    }
    return ans;
}

int main() {
    while (scanf("%d", &n) != EOF) {
        for (int i = 1; i <= n; ++i) {
            a[i].scan();
        }
        printf("%.16f\n", solve());
    }
    return 0;
}
```

44.3 多边形费马点

题意：二维平面上给出 n 个点，现在你要找个点使得这个点到 n 个点的欧几里得距离之和最小。
opentrains 1491F HDU2440 HDU 3694

```
namespace PolygonFermatPoint {
    int Move[][2] = {-1, 0, 1, 0, 0, 1, 0, -1};
    db calc(Point p, Polygon &po) {
        db res = 0;
        for (int i = 0; i < po.sze(); ++i)
            res += po[i].dis(p);
        return res;
    }
    Point gao(Polygon &po) {
        db t = 1000, pd = 0.90;
        Point now(0, 0);
        for (int i = 0; i < po.sze(); ++i) now = now + po[i];
```

```
13          now = now / po.sze();
14          db res = calc(now, po);
15          while (t > eps) {
16              int ok = 1;
17              while (ok) {
18                  ok = false;
19                  for (int i = 0; i < 4; ++i) {
20                      Point nx = now + Point(Move[i][0] * t, Move[i][1] * t);
21                      db tmp = calc(nx, po);
22                      if (tmp < res) {
23                          res = tmp;
24                          now = nx;
25                          ok = 1;
26                      }
27                  }
28              }
29              t *= pd;
30          }
31          if (sgn(now.x) == 0) now.x = 0;
32          if (sgn(now.y) == 0) now.y = 0;
33          return now;
34      }
35  }
36
37  int main() {
38      int _T; cin >> _T;
39      for (int kase = 1; kase <= _T; ++kase) {
40          if (kase > 1) puts("");
41          Polygon po; po.scan();
42          po.convexHull();
43          Point p = PolygonFermatPoint::gao(po);
44          db res = 0;
45          for (int i = 0; i < po.sze(); ++i)
46              res += p.dis(po[i]);
47          printf("%.0f\n", round(res));
48      }
49      return 0;
50  }
```

45 最小圆覆盖

定理 1：如果点 p 不在集合 S 的最小覆盖圆内，则 p 一定在 $S \cup \{p\}$ 的最小覆盖圆上。
根据这个定理，我们可以分三次确定前 i 个点的最小覆盖圆。

- 1. 令前 $i-1$ 个点的最小覆盖圆为 C
- 2. 如果第 i 个点在 C 内，则前 i 个点的最小覆盖圆也是 C
- 3. 如果不在，那么第 i 个点一定在前 i 个点的最小覆盖圆上，接着确定前 $i-1$ 个点中还有哪两个在最小覆盖圆上。因此，设当前圆心为 P_i，半径为 0，做固定了第 i 个点的前 i 个点的最小圆覆盖。
- 4. 固定了一个点：不停地在范围内找到第一个不在当前最小圆上的点 P_j，设当前圆心为 $(P_i + P_j)/2$，半径为 $\frac{1}{2}|P_iP_j|$，做固定了两个点的，前 j 个点外加第 i 个点的最小圆覆盖。
- 5. 固定了两个点：不停地在范围内找到第一个不在当前最小圆上的点 P_k，设当前圆为 P_i, P_j, P_k 的外接圆。

Luogu 1742

```cpp
#include <bits/stdc++.h>
using namespace std;
using db = double;
const int N = 1e5 + 10;
const db eps = 1e-10;
mt19937 rnd(time(0));
int sgn(db x) { if (fabs(x) < eps) return 0; return x < 0 ? -1 : 1; }

struct Point {
    db x, y;
    Point(db x = 0, db y = 0) : x(x), y(y) {}
    void scan() { db _x, _y; scanf("%lf%lf", &_x, &_y); x = _x, y = _y; }
    void print() { printf("%.11f %.11f\n", x, y); }
    bool operator == (const Point &b) const { return sgn(x - b.x) == 0 &&
       sgn(y - b.y) == 0; }
    bool operator < (const Point &b) const { return sgn(x - b.x) == 0 ? sgn(
       y - b.y) < 0 : x < b.x; }
    Point operator + (const Point &b) const { return Point(x + b.x, y + b.y)
       ; }
    Point operator - (const Point &b) const { return Point(x - b.x, y - b.y)
       ; }
    Point operator * (const db &b) const { return Point(x * b, y * b); }
    Point operator / (const db &b) const { return Point(x / b, y / b); }
    db operator ^ (const Point &b) const { return x * b.y - y * b.x; }
    db operator * (const Point &b) const { return x * b.x + y * b.y; }
    db len() { return hypot(x, y); }
    db len2() { return x * x + y * y; }
    db dis(Point b) { return hypot(x - b.x, y - b.y); }
    //顺时针旋转90度
    Point rotright() { return Point(y, -x); }
}p[N];

struct Circle {
    Point p; db r;
    Circle() {}
    Circle(Point p, db r) : p(p), r(r) {}
    Circle(db x, db y, db r) : p(Point(x, y)), r(r) {}
    //三角形的相关圆
    //外接圆 opt = 0 可处理三点共线的情况
    Circle(Point a, Point b, Point c, int opt = 0) {
        if (opt == 0) {
            Point p0 = (a + b) / 2;
            Point v0 = (b - a).rotright();
            Point p1 = (a + c) / 2;
            Point v1 = (c - a).rotright();
            db t = ((p1 - p0) ^ v1) / (v0 ^ v1);
            p = p0 + v0 * t;
            r = p.dis(a);
        }
    }
};

int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        for (int i = 1; i <= n; ++i) p[i].scan();
        shuffle(p + 1, p + 1 + n, rnd);
        Circle cir(0, 0, 0);
```

```
        //cir.r 存的是半径的平方
        for (int i = 1; i <= n; ++i) {
            //p_i不在前i - 1个点的最小覆盖圆上
            if (sgn((cir.p - p[i]).len2() - cir.r) > 0) {
                cir.p = p[i];
                cir.r = 0;
                for (int j = 1; j < i; ++j) {
                    if (sgn((cir.p - p[j]).len2() - cir.r) > 0) {
                        cir.p = (p[i] + p[j]) / 2;
                        cir.r = (p[j] - cir.p).len2();
                        for (int k = 1; k < j; ++k) {
                            if (sgn((cir.p - p[k]).len2() - cir.r) > 0) {
                                cir = Circle(p[i], p[j], p[k]);
                                cir.r = (p[k] - cir.p).len2();
                            }
                        }
                    }
                }
            }
        }
        printf("%.10f\n", sqrt(cir.r));
        printf("%.10f %.10f\n", cir.p.x, cir.p.y);
    }
    return 0;
}
```

46 最小球覆盖

Gym 101981D

```
#include<bits/stdc++.h>
using namespace std;
#define db double
const int N = 2e5 + 10;
const db eps = 1e-8;
struct Tpoint{
    db x,y,z;
    Tpoint() {}
    void scan() { scanf("%lf%lf%lf", &x, &y, &z); }
}pt[N], outer[4], res;
int npoint, nouter;
db radius;

inline db dist(Tpoint p1,Tpoint p2) {
    double dx = p1.x - p2.x, dy = p1.y - p2.y, dz = p1.z - p2.z;
    return (dx * dx + dy * dy + dz * dz);
}

inline db dot(Tpoint p1,Tpoint p2) {
    return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z;
}

void ball() {
    Tpoint q[3];
    db m[3][3], sol[3], L[3], det;
    res.x = res.y = res.z = radius = 0.;
    switch(nouter) {
        case 1:res = outer[0];break;
```

```
29          case 2:
30              res.x = (outer[0].x + outer[1].x) / 2;
31              res.y = (outer[0].y + outer[1].y) / 2;
32              res.z = (outer[0].z + outer[1].z) / 2;
33              radius = dist(res,outer[0]);
34              break;
35          case 3:
36              for(int i = 0; i < 2; i++) {
37                  q[i].x = outer[i+1].x - outer[0].x;
38                  q[i].y = outer[i+1].y - outer[0].y;
39                  q[i].z = outer[i+1].z - outer[0].z;
40              }
41              for(int i = 0; i < 2; i++) {
42                  for(int j = 0;j < 2;j++) {
43                      m[i][j] = dot(q[i], q[j]) * 2;
44                  }
45              }
46              for(int i = 0; i < 2; i++) sol[i] = dot(q[i], q[i]);
47              if(fabs(det = m[0][0] * m[1][1] - m[0][1] * m[1][0]) < eps)
48                  return;
49              L[0] = (sol[0] * m[1][1] - sol[1] * m[0][1]) / det;
50              L[1] = (sol[1] * m[0][0] - sol[0] * m[1][0]) / det;
51              res.x = outer[0].x + q[0].x * L[0] + q[1].x * L[1];
52              res.y = outer[0].y + q[0].y * L[0] + q[1].y * L[1];
53              res.z = outer[0].z + q[0].z * L[0] + q[1].z * L[1];
54              radius = dist(res, outer[0]);
55              break;
56          case 4:
57              for(int i = 0; i < 3; i++) {
58                  q[i].x = outer[i + 1].x - outer[0].x;
59                  q[i].y = outer[i + 1].y - outer[0].y;
60                  q[i].z = outer[i + 1].z - outer[0].z;
61                  sol[i] = dot(q[i], q[i]);
62              }
63              for(int i = 0; i < 3; i++) {
64                  for(int j = 0; j < 3; j++) {
65                      m[i][j] = dot(q[i], q[j]) * 2;
66                  }
67              }
68              det = m[0][0] * m[1][1] * m[2][2]
69              + m[0][1] * m[1][2] * m[2][0]
70              + m[0][2] * m[2][1] * m[1][0]
71              - m[0][2] * m[1][1] * m[2][0]
72              - m[0][1] * m[1][0] * m[2][2]
73              - m[0][0] * m[1][2] * m[2][1];
74              if(fabs(det) < eps)
75                  return;
76              for(int j = 0; j < 3; j++) {
77                  for(int i = 0; i < 3; i++)
78                      m[i][j] = sol[i];
79                  L[j] = (m[0][0] * m[1][1] * m[2][2]
80                          + m[0][1] * m[1][2] * m[2][0]
81                          + m[0][2] * m[2][1] * m[1][0]
82                          - m[0][2] * m[1][1] * m[2][0]
83                          - m[0][1] * m[1][0] * m[2][2]
84                          - m[0][0] * m[1][2] * m[2][1]) / det;
85
86                  for(int i = 0; i < 3; i++)
```

```
                    m[i][j] = dot(q[i], q[j]) * 2;
            }
            res = outer[0];
            for(int i = 0; i < 3; i++) {
                res.x += q[i].x * L[i];
                res.y += q[i].y * L[i];
                res.z += q[i].z * L[i];
            }
            radius=dist(res,outer[0]);
    }
//  cout << nouter <<"***" << radius<<"**"<<endl;
}

void minball(int n) {
    ball();
    if(nouter < 4) {
        for(int i = 0; i < n; i++) {
            if(dist(res,pt[i]) - radius > eps) {
                outer[nouter] = pt[i];
                ++nouter;
                minball(i);
                --nouter;
                if(i > 0){
                    Tpoint Tt = pt[i];
                    memmove(&pt[1], &pt[0], sizeof(Tpoint) * i);
                    pt[0] = Tt;
                }
            }
        }
    }
}

db smallest_ball() {
    radius = -1;
    for(int i = 0; i < npoint; i++) {
        if(dist(res,pt[i]) - radius > eps) {
            nouter = 1;
            outer[0] = pt[i];
            minball(i);
        }
    }
    return sqrt(radius);
}

int main() {
    scanf("%d",&npoint);
    for(int i = 0; i < npoint; i++) {
        pt[i].scan();
    }
    db ans = smallest_ball();
    printf("%.15f\n",ans);
    return 0;
}
```

47 平面最小三角形

BZOJ 3707

题意:

给处 n 个点，求出任选三个点组成三角形面积的最小值

思路:

先将所有点排序。

搞出任意两点组成的线段，那么以这条线段为底，面积最小的三角形就是按这条线段旋转坐标系，离 y 轴最近的点和这条线段组成的三角形

将这些线段按照斜率排序，那么容易发现第一条线段肯定是连续的两点，并且其它点到这条线段的距离往两边走是单调增的，因为点的排序可以看作是以 $x = 0$ 这条直线为对称轴往两边走的

然后考虑枚举下一条直线的时候，相对关系只有当前点对发生了变化，交换一下即可

```
namespace minTriangle {
    struct E {
        int a, b; double slop;
        bool operator < (const E &other) const {
            return slop < other.slop;
        }
    };
    db gao(vector <Point> &p) {
        int n = p.size();
        sort(p.begin(), p.end());
        vector <E> vec;
        //rk[i]表示排名为i的点的标号
        //pos[i]表示点i的排名
        vector <int> rk(n), pos(n);
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                vec.push_back({i, j, atan2((p[j].y - p[i].y), (p[j].x - p[i
                    ].x))});
            }
        }
        sort(vec.begin(), vec.end());
        for (int i = 0; i < n; ++i) rk[i] = pos[i] = i;
        db ans = 1e60;
        for (int i = 0; i < (int)vec.size(); ++i) {
            int a = vec[i].a, b = vec[i].b;
            if (pos[a] > pos[b]) swap(a, b);
            if (pos[a] > 0) ans = min(ans, fabs((p[b] - p[a]) ^ (p[rk[pos[a]
                - 1]] - p[a])));
            if (pos[b] < n - 1) ans = min(ans, fabs((p[b] - p[a]) ^ (p[rk[
                pos[b] + 1]] - p[a])));
            swap(pos[a], pos[b]); swap(rk[pos[a]], rk[pos[b]]);
        }
        return ans / 2;
    }
}

int main() {
    int n; scanf("%d", &n);
    vector <Point> p(n);
    for (int i = 0; i < n; ++i) p[i].scan();
    printf("%.2f\n", minTriangle::gao(p));
    return 0;
}
```

48 平面最近点对

Luogu P1429

48.1 分治法

```
//先要按照x排序
db closePair(Polygon &A, int l, int r){
    if (r - l <= 5) {
        db ans = 1e20;
        for (int i = l; i <= r; ++i)
            for (int j = i + 1; j <= r; ++j)
                ans = min(ans, A[i].dis(A[j]));
        return ans;
    }
    int mid = (l + r) >> 1;
    db ans = min(closePair(A, l, mid), closePair(A, mid + 1, r));
    vector <Point> B;
    for (int i = l; i <= r; ++i) {
        if (sgn(fabs(A[i].x - A[mid].x) - ans) <= 0)
            B.push_back(A[i]);
    }
    sort(B.begin(), B.end(), [&](Point k1, Point k2) { return k1.y < k2.y;
       });
    for (int i = 0; i < (int)B.size(); ++i)
        for (int j = i + 1; j < (int)B.size() && sgn(B[j].y - B[i].y - ans)
           < 0; ++j)
            ans = min(ans, B[i].dis(B[j]));
    return ans;
}

int main() {
    Polygon po; po.scan();
    sort(po.p.begin(), po.p.end());
    printf("%.4f\n", closePair(po, 0, po.sze() - 1));
    return 0;
}
```

48.2 随机旋转法

```
mt19937 rnd(time(0));
namespace closePair {
    db calc(Polygon &p, db angle) {
        angle = angle / 180 * PI;
        Polygon tmp;
        for (int i = 0; i < p.sze(); ++i) {
            tmp.add(p[i].rotate(Point(0, 0), angle));
        }
        sort(tmp.p.begin(), tmp.p.end());
        db res = 1e18;
        for (int i = 0; i < p.sze(); ++i) {
            for (int j = i + 1; j <= i + 5 && j < p.sze(); ++j) {
                res = min(res, tmp[i].dis(tmp[j]));
            }
        }
        return res;
    }
    db gao(Polygon &p, int cnt = 2) {
        db res = calc(p, 0);
        for (int i = 0; i < cnt; ++i) {
            res = min(res, calc(p, rnd() % 360));
```

```
        }
        return res;
    }
};

int main() {
    Polygon po; po.scan();
    printf("%.4f\n", closetPair::gao(po, 3));
    return 0;
}
```

49 辛普森积分

设 $f(x)$ 为原函数，$g(x) = Ax^2 + Bx + C$ 为拟合后的函数，有：

$$\begin{aligned}
\int_a^b f(x)dx &\approx \int_a^b Ax^2 + Bx + C \\
&= \frac{A}{3}(b^3 - a^3) + \frac{B}{2}(b^2 - a^2) + C(a - b) \\
&= \frac{(b-a)}{6}[2A(b^2 + ab + a^2) + 3B(b + a) + 6C] \\
&= \frac{(b-a)}{6}(2Ab^2 + 2Aab + 2Aa^2 + 3Bb + 3Ba + 6C) \\
&= \frac{(b-a)}{6}[Aa^2 + Ba + C + Ab^2 + Bb + C + 4A(\frac{a+b}{2})^2 + 4B(\frac{a+b}{2}) + 4C] \\
&= \frac{(b-a)}{6}(f(a) + f(b) + 4f(\frac{a+b}{2}))
\end{aligned}$$

那么 Simpson 公式为：

$$\int_a^b f(x)dx \approx \frac{(b-a)(f(a) + f(b) + 4f(\frac{a+b}{2}))}{6}$$

49.1 洛谷 4525

题意：计算：

$$\int_L^R \frac{cx + d}{ax + b}dx$$

```
#include <bits/stdc++.h>
using namespace std;
using db = double;
const db eps = 1e-6;
db a, b, c, d, l, r;

db F(db x){ return (c * x + d) / (a * x + b); }

//区间[a,b]上的辛普森值，已知区间长度l和端点及中点处的函数值A，B，C
db S(db l,db A,db B,db C){
    return (A + 4 * C + B) * l / 6;
}

db Simpson(db a,db b,db eps,db A,db B,db C){
    db l = b - a;
    db c = a + l / 2;
    db D = F(a + l / 4), E = F(a + 3 * l / 4);
```

```
    db L = S(l/2, A, C, D), R = S(l / 2, C, B, E), AB = S(l, A, B, C);
    if(fabs(L + R - AB) <= 15 * eps) return L + R + (L + R - AB) / 15;
    else return Simpson(a, c, eps / 2, A, C, D) + Simpson(c, b, eps / 2, C,
       B, E);
}

int main(){
    scanf("%lf%lf%lf%lf%lf%lf",&a, &b, &c, &d, &l, &r);
    printf("%.6f\n",Simpson(l, r, eps, F(l), F(r), F((l + r) / 2)));
    return 0;
}
```

49.2 圆的面积并

BZOJ 2178 SPOJ-CIRU

```
#include <bits/stdc++.h>
using namespace std;
#define fi first
#define se second
typedef double db;
typedef pair<db, db> pDD;
const db eps = 1e-8;
const int N = 1e3 + 10, INF = 0x3f3f3f3f;
int n;
db sqr(db x) { return x * x; }
int sgn(db x) { return fabs(x) < eps ? 0 : x < 0 ? -1 : 1; }

struct Circle{
    db x, y, r;
    void scan() { scanf("%lf%lf%lf", &x, &y, &r); }
    bool operator < (const Circle &b) const { return r > b.r; }
    db disCircletoCircle(Circle b) { return hypot(x - b.x, y - b.y); }
};

struct CircleUnion {
    vector <Circle> cir;
    vector <pair<db, db> > sta;
    int sze() { return cir.size(); }
    CircleUnion(int n = 0) { cir.clear(); cir.resize(n); }
    void scan(int n = -1) { if (n == -1) scanf("%d", &n); (*this) =
       CircleUnion(n); for (int i = 0; i < n; ++i) cir[i].scan(); }
    void add(Circle c) { cir.push_back(c); }
    Circle& operator[](int x) { return cir[x]; }
    //扫描线部分
    db f(db x){
        int top = 0;
        for (int i = 0; i < sze(); ++i) if (fabs(cir[i].x - x) < cir[i].r) {
            db len = sqrt(sqr(cir[i].r) - sqr(cir[i].x - x));
            sta[++top] = pDD(cir[i].y - len, cir[i].y + len);
        }
        if (!top) return 0;
        sort(sta.begin() + 1, sta.begin() + top + 1);
        int tmp = 1;
        for (int i = 2; i <= top; ++i) {
            if (sgn(sta[i].fi - sta[tmp].se) <= 0)
                sta[tmp].se = max(sta[tmp].se, sta[i].se);
            else
```

```
42                  sta[++tmp] = sta[i];
43          }
44          top = min(top, tmp);
45          db res = 0;
46          for (int i = 1; i <= top; ++i)
47              res += sta[i].se - sta[i].fi;
48          return res;
49      }
50      //辛普森积分
51      db Simpson(db L, db M, db R, db fL, db fM, db fR, int dep){
52          db M1 = (L + M) / 2, M2 = (M + R) / 2, fM1 = f(M1), fM2 = f(M2);
53          db g1 = (M - L) * (fL + 4 * fM1 + fM) / 6;
54          db g2 = (R - M) * (fM + 4 * fM2 + fR) / 6;
55          db g = (R - L) * (fL + 4 * fM + fR) / 6;
56          //迭代次数和精度控制
57          if (dep > 11 && fabs(g - g1 - g2) < eps) return g;
58          return Simpson(L, M1, M, fL, fM1, fM, dep + 1) + Simpson(M, M2, R,
                fM, fM2, fR, dep + 1);
59      }
60      //去除内含圆
61      void Unique(){
62          int cnt = 0;
63          sort(cir.begin(), cir.end());
64          for (int i = 0; i < sze(); i++) {
65              int ok = 1;
66              for (int j = 0; j < cnt; j++)
67                  if (cir[i].r + cir[i].disCircletoCircle(cir[j]) <= cir[j].r)
                        {
68                      ok = 0;
69                      break;
70                  }
71              if (ok) cir[cnt++] = cir[i];
72          }
73          cir.resize(cnt);
74      }
75      //获得圆并面积
76      db gao() {
77          if (sze() == 0) return 0;
78          Unique();
79          sta.clear(); sta.resize(sze() + 5);
80          db l = cir[0].x - cir[0].r;
81          db r = cir[0].x + cir[0].r;
82          for (int i = 1; i < sze(); ++i) {
83              l = min(l, cir[i].x - cir[i].r);
84              r = max(r, cir[i].x + cir[i].r);
85          }
86          db mid = l + (r - l) / 2;
87          return Simpson(l, mid, r, f(l), f(mid), f(r), 0);
88      }
89  };
90
91  int main() {
92      while (scanf("%d", &n) != EOF) {
93          CircleUnion cu; cu.scan(n);
94          printf("%.3f\n", cu.gao());
95          return 0;
96      }
97  }
```

50 格林公式

平面单连通区域：设 D 是平面内一区域, 若属于 D 内任一简单闭曲线内部都属于 D, 则 D 为平面单连通区域。简单来说，就是没有洞的区域

正方向：当 xOy 平面上的曲线起点与终点重合时, 则称曲线为闭曲线, 设平面的闭曲线 L 围成平面区域 D, 并规定当一个人沿闭曲线 L 环行时, 区域 D 总是在此人的左侧, 称此人行走的方向为曲线 L 关于区域 D 的正方向, 反之为负方向。

格林公式：设 D 是一个平面单连通区域，L 是它取正向的轮廓线（分段光滑），P,Q 在 D 上具有一阶连续偏导数，则有:

$$\int\int_D \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) \mathrm{d}x\mathrm{d}y = \oint_L P\mathrm{d}x + Q\mathrm{d}y$$

令 $Q = x$, $P = -y$, 有:

$$\int\int_D 1\mathrm{d}x\mathrm{d}y = \frac{1}{2}\oint_L -y\mathrm{d}x + x\mathrm{d}y$$

这样转化之后，我们就可以把圆的面积转化成跟轮廓线有关的计算了

考虑圆的参数方程:

$$\begin{cases} x = x_0 + r\cos t \\ y = y_0 + r\sin t \end{cases}$$

带入有:

$$\begin{aligned}
&\int_L -y\mathrm{d}x + x\mathrm{d}y \\
=& \int_L -(y_0 + r\sin t)\mathrm{d}(x_0 + r\cos t) + (x_0 + r\cos t)\mathrm{d}(y_0 + r\sin t) \\
=& \int_L ((y_0 + r\sin t)(r\sin t) + (x_0 + r\cos t)(r\cos t))dt \\
=& r\int_L ((y_0 + r\sin t)\sin t + (x_0 + r\cos t)\cos t)dt \\
=& r\int_L (r + y_0\sin t + x_0\cos t)dt \\
=& r(rt + x_0\sin t - y_0\cos t)
\end{aligned}$$

然后利用牛顿-莱布尼兹公式即可。

注意按此公式所求的值的闭区域 D 的两倍

UOJ 419

题意：有 n 个圆，输出 n 个数，第 i 个数字表示前 i 个圆的面积并

动态维护圆的轮廓线，然后用格林公式即可，格林公式计算的是轮廓线所围成的闭区域的面积，即一段圆弧和这段圆弧两端水平连成的弦组成的面积。

```
#include <bits/stdc++.h>
#define fi first
#define se second
using namespace std;
typedef double db;
const int N = 2e3 + 10;
const db PI = acos(-1.0);
inline db sqr(db x) { return 1. * x * x; }
```

```
int n;

struct Point {
    db x, y;
    Point() { }
    Point(db _x, db _y): x(_x), y(_y) {}
    void scan() { scanf("%lf%lf", &x, &y); }
    Point operator + (const Point& b) const { return Point(x + b.x, y + b.y); }
    Point operator - (const Point& b) const { return Point(x - b.x, y - b.y); }
    db len() { return hypot(x, y); }
    db ang() { return atan2(y, x); }
};

struct Circle {
    Point p; db r;
    void scan() { p.scan(); scanf("%lf", &r); }
    db oint(db t1, db t2) {
        return r * (r * (t2 - t1) + p.x * (sin(t2) - sin(t1)) - p.y * (cos(t2) - cos(t1)));
    }
    bool in(const Circle &b) const { return (p - b.p).len() + b.r <= r; }
    bool out(const Circle &b) const { return r + b.r <= (p - b.p).len(); }
}c[N];

namespace CircleUnion {
    struct node {
        db l, r;
        node(db l = 0, db r = 0) : l(l), r(r) {}
        bool operator < (const node &b) const { return r < b.r; }
    };
    set <node> s[N]; db res;
    bool vis[N];
    typedef set <node>::iterator IT;
    inline void fix(db &ang){
        if(ang < -PI) ang += PI * 2;
        if(ang > PI) ang -= PI * 2;
    }
    void remove(int id, db l, db r) {
        if (l > r) {
            remove(id, -PI, r);
            remove(id, l, +PI);
            return;
        }
        for (IT it = s[id].lower_bound(node(0, l)), tmp; it != s[id].end()
            && it->l < r; it = tmp) {
            db nl = it->l, nr = it->r;
            tmp = it, ++tmp;
            s[id].erase(it), res -= c[id].oint(nl, nr);
            if (nl < l) s[id].insert(node(nl, l)), res += c[id].oint(nl, l);
            if (nr > r) s[id].insert(node(r, nr)), res += c[id].oint(r, nr);
        }
    }
    void del(int i, int j) {
        db dis = (c[i].p - c[j].p).len(), ang, cur, l, r;
        cur = (c[j].p - c[i].p).ang();
        //求圆心到交点的线和两圆心连线之间的角度
```

```
63          ang = acos((sqr(dis) + sqr(c[i].r) - sqr(c[j].r)) / (c[i].r * dis *
                2));
64          fix(l = cur - ang); fix(r = cur + ang);
65          remove(i, l, r);
66      }
67      void gao() {
68          res = 0;
69          for (int i = 1; i <= n; ++i) {
70              vis[i] = 1;
71              for (int j = 1; j < i; ++j) if (vis[j]) {
72                  if (c[j].r >= c[i].r && c[j].in(c[i])) { vis[i] = 0; break;
                        }
73                  if (c[i].r > c[j].r && c[i].in(c[j])) {
74                      vis[j] = 0;
75                      remove(j, -PI, +PI);
76                  }
77              }
78              if (!vis[i]) {
79                  printf("%.10f\n", res * 0.5);
80                  continue;
81              }
82              res += c[i].r * c[i].r * PI * 2;
83              s[i].clear();
84              s[i].insert(node(-PI, PI));
85              for (int j = 1; j < i; ++j) if (vis[j] && !c[i].out(c[j])) del(i
                    , j), del(j, i);
86              printf("%.10f\n", res * 0.5);
87          }
88      }
89  }
90
91  int main() {
92      scanf("%*d%d", &n);
93      for (int i = 1; i <= n; ++i) c[i].scan();
94      CircleUnion::gao();
95      return 0;
96  }
```

51 闵可夫斯基和

定义:

两个图形 A, B 的闵可夫斯基和 $C = \{a + b \mid a \in A, b \in B\}$

简单来说就是从原点向图形 A 内部的每一个点做向量，将图形 B 沿每个向量移动，所有的最终位置的并就是闵可夫斯基和 (具有交换律)。

51.1 LuoguP4557

题意:

两个凸包 A, B，每次移动 B，问 B 是否和 A 有交点

思路:

令 $a \in A, b \in B$, 则有移动向量 w 使得存在 $b + w = a$，那么 w 满足 $w = a - b$

那么构造闵可夫斯基和 $C = \{a + (-b)\}$，然后判断每次给出的移动向量是否在该闵可夫斯基和的凸包里即可

```
1  Polygon minKowSki(Polygon A, Polygon B) {
2      //做闵可夫斯基和之前要对A, B分别求凸包
3      A.convexHull(); B.convexHull();
4      Polygon res;
```

```
    res.add(A[0] + B[0]);
    int i = 0, j = 0;
    while (i < A.sze() && j < B.sze()) {
        if (sgn((A[i + 1] - A[i]) ^ (B[j + 1] - B[j])) >= 0) {
            res.add(res[-1] + (A[i + 1] - A[i])); ++i;
        } else {
            res.add(res[-1] + (B[j + 1] - B[j])); ++j;
        }
    }
    while (i < A.sze()) { res.add(res[-1] + (A[i + 1] - A[i])); ++i; }
    while (j < B.sze()) { res.add(res[-1] + (B[j + 1] - B[j])); ++j; }
    //再求一次凸包可以去除掉边上的点
    res.convexHull();
    return res;
}

int main() {
    Polygon A, B, C;
    int n, m, q; scanf("%d%d%d", &n, &m, &q);
    A.scan(n); B.scan(m);
    for (int i = 0; i < m; ++i) {
        B[i].x = -B[i].x;
        B[i].y = -B[i].y;
    }
    C = minKowSki(A, B);
    for (int i = 1; i <= q; ++i) {
        Point p; p.scan();
        if (C.pointInConvex(p) >= 0) puts("1");
        else puts("0");
    }
    return 0;
}
```

51.2 HDU 6541

题意:

给出 n 个点 A_i，每次询问给出一个点 P_k, 要找出一个区间 $[l,r]$ 使得 $2\cdot\sum_{i=l}^{r}S_i$ 最大，S_i 表示 $\triangle OP_kA_i$ 的有向面积

思路:

要求的东西等价于 $P_k.x\cdot\sum_{i=l}^{r}A_i.y-P_k.y\cdot\sum_{i=l}^{r}A_i.x$

利用分治，每一层对于 $[l,mid-1]$ 的每个后缀以及 $[mid+1,r]$ 的每个前缀搞出凸包，然后闵可夫斯基和合并即可

凸包的总点数是 $O(n\log n)$

最后再对合并出的所有点求一遍凸包即可，那么对于每个询问的答案所在点肯定在凸包上

```
Polygon minKowSki(Polygon &A, Polygon &B) {
    A.convexHull(); B.convexHull();
    Polygon res;
    res.add(A[0] + B[0]);
    int i = 0, j = 0;
    while (i < A.sze() && j < B.sze()) {
        if (sgn((A[i + 1] - A[i]) ^ (B[j + 1] - B[j])) >= 0) {
            res.add(res[-1] + (A[i + 1] - A[i])); ++i;
        } else {
            res.add(res[-1] + (B[j + 1] - B[j])); ++j;
        }
```

```
12      }
13      while (i < A.sze()) { res.add(res[-1] + (A[i + 1] - A[i])); ++i; }
14      while (j < B.sze()) { res.add(res[-1] + (B[j + 1] - B[j])); ++j; }
15      //再求一次凸包可以去除掉边上的点
16      res.convexHull();
17      return res;
18  }
19
20  void gao(vector <Point> &vec, Polygon &res, int l, int r) {
21      if (l == r) {
22          res.add(vec[l]);
23          return;
24      }
25      int mid = (l + r) >> 1;
26      gao(vec, res, l, mid);
27      gao(vec, res, mid + 1, r);
28      Polygon A, B;
29      Point pre(0, 0);
30      for (int i = mid; i >= l; --i) {
31          pre = pre + vec[i];
32          A.add(pre);
33      }
34      pre = Point(0, 0);
35      for (int i = mid + 1; i <= r; ++i) {
36          pre = pre + vec[i];
37          B.add(pre);
38      }
39      Polygon tmp = minKowSki(A, B);
40      for (int i = 0; i < tmp.sze(); ++i) res.add(tmp[i]);
41  }
42
43  int main() {
44      int n, q;
45      while (scanf("%d%d", &n, &q) != EOF) {
46          vector <Point> vec(n + 1);
47          for (int i = 1; i <= n; ++i) vec[i].scan();
48          Polygon res;
49          gao(vec, res, 1, n);
50          Polygon upHull, downHull;
51          res.getUpDownHull(upHull, downHull);
52          for (int i = 1; i <= q; ++i) {
53              Point p; p.scan();
54              p.y *= -1;
55              swap(p.x, p.y);
56              db ans = 0;
57              if (p.y > 0) ans = max(ans, upHull.querySlopeMax(p));
58              else ans = max(ans, downHull.querySlopeMax(p));
59              printf("%lld\n", ans);
60          }
61      }
62      return 0;
63  }
```