Contents

1. Geometry

1.1 一些公式

1.1.1 Heron's Formula

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

1.1.2 四面体内接球球心

假设 s_i 是第 i 个顶点相对面的面积，则有

$$\begin{cases} x = \dfrac{s_1x_1 + s_2x_2 + s_3x_3 + s_4x_4}{s_1+s_2+s_3+s_4} \\ y = \dfrac{s_1y_1 + s_2y_2 + s_3y_3 + s_4y_4}{s_1+s_2+s_3+s_4} \\ z = \dfrac{s_1z_1 + s_2z_2 + s_3z_3 + s_4z_4}{s_1+s_2+s_3+s_4} \end{cases}$$

体积可以使用 1/6 混合积求, 内接球半径为

$$r = \frac{3V}{s_1+s_2+s_3+s_4}$$

1.1.3 三角形内心

$$\vec{I} = \frac{a\vec{A} + b\vec{B} + c\vec{C}}{a+b+c}$$

1.1.4 三角形外心

$$\vec{O} = \frac{\vec{A} + \vec{B} - \frac{\vec{BC}\cdot\vec{CA}}{\vec{AB}\times\vec{BC}}\vec{AB}^T}{2}$$

1.1.5 三角形垂心

$$\vec{H} = 3\vec{G} - 2\vec{O}$$

1.1.6 三角形偏心

$$\frac{-a\vec{A} + b\vec{B} + c\vec{C}}{-a+b+c}$$

内角的平分线和对边的两个外角平分线交点, 外切圆圆心. 剩余两点的同理.

1.1.7 三角形内接外接圆半径

$$r = \frac{2S}{a+b+c},\ R = \frac{abc}{4S}$$

1.1.8 Pick's Theorem 格点多边形面积

$S = I + \frac{B}{2} - 1$. I 内部点, B 边界点。

1.1.9 Euler's Formula 多面体与平面图的点、边、面

For convex polyhedron: $V - E + F = 2$.

For planar graph: $|F| = |E| - |V| + n + 1$, n : #(connected components).

1.2 三角公式

$$\sin(a \pm b) = \sin a \cos b \pm \cos a \sin b$$
$$\cos(a \pm b) = \cos a \cos b \mp \sin a \sin b$$
$$\tan(a \pm b) = \frac{\tan(a) \pm \tan(b)}{1 \mp \tan(a)\tan(b)}$$
$$\tan(a) \pm \tan(b) = \frac{\sin(a \pm b)}{\cos(a)\cos(b)}$$
$$\sin(a) + \sin(b) = 2\sin(\frac{a+b}{2})\cos(\frac{a-b}{2})$$
$$\sin(a) - \sin(b) = 2\cos(\frac{a+b}{2})\sin(\frac{a-b}{2})$$
$$\cos(a) + \cos(b) = 2\cos(\frac{a+b}{2})\cos(\frac{a-b}{2})$$
$$\cos(a) - \cos(b) = -2\sin(\frac{a+b}{2})\sin(\frac{a-b}{2})$$
$$\sin(na) = n\cos^{n-1}a\sin a - \tbinom{n}{3}\cos^{n-3}a\sin^3 a + \tbinom{n}{5}\cos^{n-5}a\sin^5 a - \dots$$
$$\cos(na) = \cos^n a - \binom{n}{2}\cos^{n-2}a\sin^2 a + \binom{n}{4}\cos^{n-4}a\sin^4 a - \dots$$

1.2.1 超球坐标系

$$\begin{aligned} x_1 &= r\cos(\phi_1) \\ x_2 &= r\sin(\phi_1)\cos(\phi_2) \\ &\cdots \\ x_{n-1} &= r\sin(\phi_1)\cdots\sin(\phi_{n-2})\cos(\phi_{n-1}) \\ x_n &= r\sin(\phi_1)\cdots\sin(\phi_{n-2})\sin(\phi_{n-1}) \\ \phi_{n-1} &\in [0, 2\pi] \\ \forall i = 1..n-1 \phi_i &\in [0, \pi] \end{aligned}$$

1.2.2 三维旋转公式

绕着 $(0,0,0) - (ux, uy, uz)$ 旋转 θ, (ux, uy, uz) 是单位向量

$$R = \begin{matrix} \cos\theta + u_x^2(1-\cos\theta) & u_xu_y(1-\cos\theta) - u_z\sin\theta & u_xu_z(1-\cos\theta) + u_y\sin\theta \\ u_yu_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_yu_z(1-\cos\theta) - u_x\sin\theta \\ u_zu_x(1-\cos\theta) - u_y\sin\theta & u_zu_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{matrix}.$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

1.2.3 立体角公式

$$\phi : \text{二面角}$$
$$\Omega = (\phi_{ab} + \phi_{bc} + \phi_{ac})\ \text{rad} - \pi\ \text{sr}$$
$$\tan\left(\frac{1}{2}\Omega/\text{rad}\right) = \frac{\left|\vec{a}\,\vec{b}\,\vec{c}\right|}{abc + \left(\vec{a}\cdot\vec{b}\right)c + (\vec{a}\cdot\vec{c})\,b + \left(\vec{b}\cdot\vec{c}\right)a}$$
$$\theta_s = \frac{\theta_a + \theta_b + \theta_c}{2}$$

1.2.4 常用体积公式

- 棱锥 Pyramid $V = \frac{1}{3}Sh$.
- 球 Sphere $V = \frac{4}{3}\pi R^3$.
- 棱台 Frustum $V = \frac{1}{3}h(S_1 + \sqrt{S_1S_2} + S_2)$.
- 椭球 Ellipsoid $V = \frac{4}{3}\pi abc$.
- 球缺 Spherical cap $\frac{\pi}{3}(3R - H)H^2$

1.2.5 扇形与圆弧重心

扇形重心与圆心距离为 $\frac{4r\sin(\theta/2)}{3\theta}$, 圆弧重心与圆心距离为 $\frac{4r\sin^3(\theta/2)}{3(\theta - \sin(\theta))}$.

1.2.6 高维球体积

$$V_2 = \pi R^2,\ S_2 = 2\pi R$$
$$V_3 = \frac{4}{3}\pi R^3,\ S_3 = 4\pi R^2$$
$$V_4 = \frac{1}{2}\pi^2 R^4,\ S_4 = 2\pi^2 R^3$$
$$\text{Generally},\ V_n = \frac{2\pi}{n}V_{n-2},\ S_{n-1} = \frac{2\pi}{n-2}S_{n-3}$$
$$\text{Where},\ S_0 = 2,\ V_1 = 2,\ S_1 = 2\pi,\ V_2 = \pi$$

1.3 距离

欧式距离

$$\sqrt{\sum_{i=1}^{n}(x_{1,i} - x_{2,i})^2}$$

曼哈顿距离

$$\sum_{i=1}^{n}\left|x_{1,i} - x_{2,i}\right|$$

切比雪夫距离

$$\max_{i=1}^{n}\left\{\left|x_{1,i} - x_{2,i}\right|\right\}$$

曼哈顿距离与切比雪夫距离转换:

- 曼哈顿坐标系是通过切比雪夫坐标系旋转 45° 后，再缩小到原来的一半得到的。
- 将一个点 (x, y) 的坐标变为 $(x + y, x - y)$ 后，原坐标系中的曼哈顿距离等于新坐标系中的切比雪夫距离。
- 将一个点 (x, y) 的坐标变为 $(\frac{x+y}{2}, \frac{x-y}{2})$ 后，原坐标系中的切比雪夫距离等于新坐标系中的曼哈顿距离。

1.4 Pick 定理

给定顶点均为整点的简单多边形，皮克定理说明了其面积 A 和内部格点数目 i、边上格点数目 b 的关系：

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

推广：

- 取格点的组成图形的面积为一单位。在平行四边形格点，皮克定理依然成立。套用于任意三角形格点，皮克定理则是 $A = 2 \times i + b - 2$。
- 对于非简单的多边形 P，皮克定理 $A = i + \frac{b}{2} - \chi(P)$，其中 $\chi(P)$ 表示 P 的欧拉特征数 $\chi(P) = V - E + F$。
- 皮克定理和欧拉公式（$V - E + F = 2$）等价。

2. Graph

2.1 图论基本知识

2.1.1 树链的交

2.1.2 带修改MST

维护少量修改的最小生成树，可以缩点缩边使暴力复杂度变低. (银川 21: 求有 16 个'某两条边中至少选一条'的限制条件的最小生成树)

找出必须边　将修改边标 $-\infty$, 在MST上的其余边为必须边, 以此缩点.

找出无用边　将修改边标 ∞, 不在MST上的其余边为无用边, 删除之.

假设修改边数为 k, 操作后图中最多剩下 $k + 1$ 个点和 $2k$ 条边.

2.1.3 差分约束

$x_r - x_l \le c$:`add(l, r, c)` $x_r - x_l \ge c$:`add(r, l, -c)`

2.1.4 李超线段树

添加若干条线段或直线 $(a_i, b_i) \to (a_j, b_j)$, 每次求 $[l, r]$ 上最上面的那条线段的值. 思想是让线段树中一个节点只对应一条直线, 如果在这个区间加入一条直线, 如果一段比原来的优, 一段比原来的劣, 那么判断一下两条线的交点, 判断哪条直线可以完全覆盖一段一半的区间, 把它保留, 另一条直线下传到另一半区间. 时间复杂度 $O(n\log n)$.

2.1.5 Segment Tree Beats

区间 min, max, 区间求和. 以区间取 min 为例, 额外维护最大值 m, 严格次大值 s 以及最大值个数 t. 现在假设我们要让区间 $[L,R]$ 对 x 取 min, 先在线段树中定位若干个节点, 对于每个节点分三种情况讨论: 1, 当 $m \le x$ 时, 显然这一次修改不会对这个节点产生影响, 直接退出; 2, 当 $se < x < ma$ 时, 显然这一次修改只会影响到所有最大值, 所以把 num 加上 $t*(x-ma)$, 把 ma 更新为 x, 打上标记退出; 3, 当 $se \ge x$ 时, 无法直接更新着一个节点的信息, 对当前节点的左儿子和右儿子递归处理. 单次操作均摊复杂度 $O(\log^2 n)$.

2.1.6 二分图

最小点覆盖=最大匹配数. 独立集与覆盖集互补. 最小点覆盖构造方法: 对二分图流图求割集, 跨过的边指示最小点覆盖. **Hall**定理 $G=(X,Y,E), |M|=|X| \Leftrightarrow \forall S \subseteq X, |S| \le |A(S)|$.

2.1.7 稳定婚姻问题

男士按自己喜欢程度从高到底依次向每位女士求婚, 女士遇到更喜欢的男士时就接受他, 并抛弃以前的配偶. 被抛弃的男士继续按照列表向剩下的女士依次求婚, 直到所有人都有配偶. 算法一定能得到一个匹配, 而且这个匹配一定是稳定的. 时间复杂度 $O(n^2)$.

2.1.8 三元环

对于无向边 (u,v), 如果 $\deg_u < \deg_v$, 那么连有向边 (u,v)(以点标号为第二关键字). 枚举 x 暴力即可. 时间复杂度 $O(m\sqrt{m})$.

2.1.9 图同构

令 $F_t(i) = (F_{t-1}(i)*A + \sum_{i\to j} F_{t-1}(j)*B + \sum_{j\to i} F_{t-1}(j)*C + D*(i-a)) \bmod P$, 枚举点 a, 迭代 K 次后求得的就是 a 点所对应的 $hash$ 值, 其中 K,A,B,C,D,P 为 $hash$ 参数, 可自选.

2.1.10 竞赛图 Landau's Theorem

n 个点竞赛图点按出度按升序排序, 前 i 个点的出度之和不小于 $\frac{i(i-1)}{2}$, 度数总和等于 $\frac{n(n-1)}{2}$. 否则可以用优先队列构造出方案.

2.1.11 Ramsey Theorem R(3,3)=6, R(4,4)=18

6 个人中存在 3 人相互认识或者相互不认识.

2.1.12 树的计数 Prufer序列

树和其prufer编码一一对应, 一颗 n 个点的树, 其prufer编码长度为 $n-2$, 且度数为 d_i 的点在prufer 编码中出现 d_i-1 次.

由树得到序列: 总共需要 $n-2$ 步, 第 i 步在当前的树中寻找具有最小标号的叶子节点, 将与其相连的点的标号设为Prufer序列的第 i 个元素 p_i, 并将此叶子节点从树中删除, 直到最后得到一个长度为 $n-2$ 的Prufer 序列和一个只有两个节点的树.

由序列得到树: 先将所有点的度赋初值为 1, 然后加上它的编号在Prufer序列中出现的次数, 得到每个点的度; 执行 $n-2$ 步, 第 i 步选取具有最小标号的度为 1 的点 u 与 $v=p_i$ 相连, 得到树中的一条边, 并将 u 和 v 的度减一. 最后再把剩下的两个度为 1 的点连边, 加入到树中.

相关结论: n 个点完全图, 每个点度数依次为 $d_1,d_2,\dots,d_n$, 这样生成树的棵树为: $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\dots(d_n-1)!}$.

左边有 n_1 个点, 右边有 n_2 个点的完全二分图的生成树棵树为 $n_1^{n_2-1} \times n_2^{n_1-1}$.

m 个连通块, 每个连通块有 c_i 个点, 把他们全部连通的生成树方案数: $(\sum c_i)^{m-2}\prod c_i$

2.1.13 有根树的计数

首先, 令 $S_{n,j} = \sum_{1\le j\le n/j}$; 于是 $n+1$ 个结点的有根树的总数为 $a_{n+1} = \frac{\sum_{j=1}^{n} ja_jS_{n-j}}{n}$. 注: $a_1=1, a_2=1, a_3=2, a_4=4, a_5=9, a_6=20, a_9=286, a_{11}=1842$.

2.1.14 无根树的计数

n 是奇数时, 有 $a_n - \sum_i^{n/2} a_i a_{n-i}$ 种不同的无根树.

n 时偶数时, 有 $a_n - \sum_i^{n/2} a_i a_{n-i} + \frac{1}{2}a_{n/2}(a_{n/2}+1)$ 种不同的无根树.

2.1.15 生成树计数 Kirchhoff's Matrix-Tree Thoerem

Kirchhoff Matrix $T = Deg - A$, Deg 是度数对角阵, A 是邻接矩阵. 无向图度数矩阵是每个点度数; 有向图度数矩阵是每个点入度.

邻接矩阵 $A[u][v]$ 表示 $u \longrightarrow v$ 边个数, 重边按照边数计算, 自环不计入度数.

无向图生成树计数: $c = |K$ 的任意1个 $n-1$ 阶主子式 $|$

有向图外向树计数: $c = |$ 去掉根所在的那阶得到的主子式 $|$

2.1.16 有向图欧拉回路计数 BEST Thoerem

$$\mathrm{ec}(G) = t_w(G)\prod_{v\in V}(\deg(v)-1)!$$

其中 deg 为入度 (欧拉图中等于出度), $t_w(G)$ 为以 w 为根的外向树的个数. 相关计算参考生成树计数.

欧拉连通图中任意两点外向树个数相同: $\mathrm{t_v}(G) = \mathrm{t_w}(G)$.

2.1.17 Tutte Matrix

Tutte matrix A of a graph $G=(V,E)$:

$$A_{ij} = \begin{cases} x_{ij} & \text{if } (i,j)\in E \text{ and } i<j \\ -x_{ij} & \text{if } (i,j)\in E \text{ and } i>j \\ 0 & \text{otherwise} \end{cases}$$

where x_{ij} are indeterminates. The determinant of this skew-symmetric matrix is then a polynomial (in the variables x_{ij}, $i<j$): this coincides with the square of the pfaffian of the matrix A and is non-zero (as a polynomial) if and only if a perfect matching exists.

2.1.18 Edmonds Matrix

Edmonds matrix A of a balanced ($|U|=|V|$) bipartite graph $G=(U,V,E)$:

$$A_{ij} = \begin{cases} x_{ij} & (u_i,v_j)\in E \\ 0 & (u_i,v_j)\notin E \end{cases}$$

where the x_{ij} are indeterminates. G 有完美匹配当且仅当关于 x_{ij} 的多项式 $det(A_{ij})$ 不恒为 0. 完美匹配的个数等于多项式中单项式的个数.

2.1.19 有向图无环定向, 色多项式

图的色多项式 $P_G(q)$ 对图 G 的 q-染色计数.

Triangle K_3 : $x(x-1)(x-2)$

Complete graph K_n : $x(x-1)(x-2)\cdots(x-(n-1))$

Tree with n vertices : $x(x-1)^{n-1}$

Cycle C_n : $(x-1)^n + (-1)^n(x-1)$

acyclic orientations of an n-vertex graph G is $(-1)^n P_G(-1)$.

2.2 2 SAT

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <queue>
#include <cmath>
#include <vector>
#include <algorithm>
#include <cstdlib>
using namespace std;
int n,m,h[2000001],p,tot,t;
int
  ↪ ans[2000001],cnt,bel[2000001],dfn[2000001],low[2000001],inz[20
char c[2000001];
struct pp
{
    int to,ne;
}b[2000001];
void add(int x,int y)
{
    b[++p].to=y;
    b[p].ne=h[x];
    h[x]=p;
}
void tarjan(int x)
{
    dfn[x]=low[x]=++cnt;
    z[++t]=x;
    inz[x]=1;
    for(int i=h[x];i;i=b[i].ne)
    {
        int v=b[i].to;
        if(!dfn[v])
        {
            tarjan(v);
            low[x]=min(low[x],low[v]);
        }
        else if(inz[v])
        {
            low[x]=min(low[x],dfn[v]);
        }
    }
    if(dfn[x]==low[x])
    {
        tot++;
        do
        {
            bel[z[t]]=tot;
            inz[z[t]]=0;
        }while(z[t--]!=x);
    }
}
bool solve()
{
    int i;
    for(i=1;i<=2*n;i++)
    {
        if(!dfn[i])
        tarjan(i);
    }
    for(i=1;i<=n;i++)
```

```
    {
        if(bel[i]==bel[i+n])
        {
            return 0;
        }
    }
    return 1;
}
int main()
{
    int i;
    scanf("%d%d",&n,&m);
        p=0;
        for(i=1;i<=m;i++)
        {
            int xx,yy,x,y;
            scanf("%d%d%d%d",&xx,&x,&yy,&y);
            add(xx+(x^1)*n,yy+y*n);
            add(yy+(y^1)*n,xx+x*n);
        }
        if(solve())
        {
            printf("POSSIBLE\n");
            for(int i=1;i<=n;i++)
            {
                if(bel[i]<bel[i+n])
                printf("0 ");
                else
                printf("1 ");
            }
        }
        else
        {
            printf("IMPOSSIBLE\n");
        }
    return 0;
}
```

2.3 极大团

```
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

const int maxn = 129;

int n, m;
int S;
int some[maxn][maxn], all[maxn][maxn], none[maxn][maxn], g[maxn][maxn];
void dfs(int d, int an, int sn, int nn) {
    if(!sn&&!nn) ++S;
    if(S>1000) return; //题意表明S超过1000就输出Impossible
    int u=some[d][0];
    for(int i=0; i<sn; ++i) {
        int v=some[d][i];
        if(g[u][v]) continue;
        int tsn=0, tnn=0;
        //for(int j=0; j<an; ++j) all[d+1][j]=all[d][j];
        //all[d+1][an]=v;  //可以不写这两行
        for(int j=0; j<sn; ++j) if(g[v][some[d][j]])
            some[d+1][tsn++]=some[d][j];
        for(int j=0; j<nn; ++j) if(g[v][none[d][j]])
            none[d+1][tnn++]=none[d][j];
        dfs(d+1,an+1,tsn,tnn);
        some[d][i]=0, none[d][nn++]=v;
    }
}

int main() {
    ios::sync_with_stdio(false);
    while(cin>>n>>m) {
        S=0;
        memset(g,0,sizeof(g));
        for(int i=0; i<m; ++i) {
            int a, b;
            cin>>a>>b;
            g[a][b]=g[b][a]=1;
        }
        for(int i=0; i<n; ++i) some[0][i]=i+1; //some的初始化
        dfs(0,0,n,0);
        if(S>1000) cout<<"Too many maximal sets of friends."<<endl;
        else cout<<S<<endl;
    }
}
```

3. Data Structure

3.1 LCT 动态树

```
namespace LCT {
    int ch[N][2], f[N], sum[N], val[N], tag[N], dat[N]; // dat 维护的链信息, val 点上信息
    inline void PushUp(int x) {
        dat[x] = dat[ch[x][0]] ^ dat[ch[x][1]] ^ val[x];
    }
    inline void PushRev(int x) {swap(ch[x][0], ch[x][1]); tag[x] ^= 1;}
    inline void PushDown(int x) {
        if (tag[x] == 0) return ;
        PushRev(ch[x][0]); PushRev(ch[x][1]); tag[x] = 0;
    }
    inline bool Get(int x) {return ch[f[x]][1] == x;} // 是父亲的哪个儿子
    inline bool IsRoot(int x) {return (ch[f[x]][1] != x && ch[f[x]][0] != x);} // 是否是当前 Splay 的根
    inline void Rotate(int x) { // Splay 旋转
        int y = f[x], z = f[y], k = Get(x);
        if (!IsRoot(y)) ch[z][Get(y)] = x;
        ch[y][k] = ch[x][k ^ 1]; f[ch[x][k ^ 1]] = y;
        ch[x][k ^ 1] = y; f[y] = x; f[x] = z;
        PushUp(y); PushUp(x);
    }
    void Updata(int x) { // Splay 中从上到下 PushDown
        if (!IsRoot(x)) Updata(f[x]);
        PushDown(x);
    }
    inline void Splay(int x) { // Splay 上把 x 转到根
        Updata(x);
        for (int fa; fa = f[x], !IsRoot(x); Rotate(x)) {
            if (!IsRoot(fa)) Rotate(Get(fa) == Get(x) ? fa : x);
        }
        PushUp(x);
    }
    inline void Access(int x) { // 辅助树上打通 x 到根的路径(即 x 到根变为实链)
        for (int p = 0; x; p = x, x = f[x]) {
            Splay(x); ch[x][1] = p; PushUp(x);
        }
    }
    inline void MakeRoot(int x) { // 钦定 x 为辅助树根
        Access(x); Splay(x); PushRev(x);
    }
    inline int FindRoot(int x) { // 找 x 所在辅助树根
        Access(x); Splay(x);
        while (ch[x][0]) PushDown(x), x = ch[x][0];
        Splay(x); // 不加复杂度会假
        return x;
    }
    inline void Split(int x, int y) { // 把 x 到 y 的路径提出来，并以 y 为 Splay 根
        MakeRoot(x); Access(y); Splay(y);
    }
    inline bool Link(int x, int y) { // 连接 x,y 两点
        MakeRoot(x);
        if (FindRoot(y) == x) return false;
        f[x] = y;
        return true;
    }
    inline bool Cut(int x, int y) { // x,y 断边
        MakeRoot(x);
        if (FindRoot(y) == x && f[y] == x && !ch[y][0]) {
            f[y] = ch[x][1] = 0; PushUp(x);
            return true;
        }
        return false;
    }
}
```

3.2 KD Tree

```cpp
// KDTree 二维平面邻域查询 K 远点对 n=1e5 k=100
priority_queue<ll, vector<ll>, greater<ll> >q; // 小根堆
namespace KDTree {
    struct node {
        int X[2];
        int &operator[](const int k) {return X[k];}
    } p[N];
    int nowd;
    bool cmp(node a, node b) {return a.X[nowd] < b.X[nowd];}
    int lc[N], rc[N], L[N][2], R[N][2]; // lc/rc 左右孩子; L/R 对应超矩形各个维度范围
    inline ll sqr(int x) {return 1ll * x * x;}
    void pushup(int x) { // 更新该点所代表空间范围
        L[x][0] = R[x][0] = p[x][0];
        L[x][1] = R[x][1] = p[x][1];
        if (lc[x]) {
            umin(L[x][0], L[lc[x]][0]); umax(R[x][0], R[lc[x]][0]);
            umin(L[x][1], L[lc[x]][1]); umax(R[x][1], R[lc[x]][1]);
        }
        if (rc[x]) {
            umin(L[x][0], L[rc[x]][0]); umax(R[x][0], R[rc[x]][0]);
            umin(L[x][1], L[rc[x]][1]); umax(R[x][1], R[rc[x]][1]);
        }
    }
    int build(int l, int r) {
        if (l > r) return 0;
        int mid = (l + r) >> 1;
        // >>> 方差建树
        db av[2] = {0, 0}, va[2] = {0, 0}; // av 平均数, va 方差
        for (int i = l; i <= r; ++i) av[0] += p[i][0], av[1] += p[i][1];
        av[0] /= (r - l + 1); av[1] /= (r - l + 1);
        for (int i = l; i <= r; ++i) {
            va[0] += sqr(av[0] - p[i][0]);
            va[1] += sqr(av[1] - p[i][1]);
        }
        if (va[0] > va[1]) nowd = 0;
        else nowd = 1; // 找方差大的维度划分
        // >>> 轮换建树 nowd=dep%D
        nth_element(p + l, p + mid, p + r + 1, cmp); // 以该维度中位数分割
        lc[mid] = build(l, mid - 1); rc[mid] = build(mid + 1, r);
        pushup(mid);
        return mid;
    }
    ll dist(int a, int x) { // 估价函数，点 a 到树上 x 点对应空间最远距离
        return max(sqr(p[a][0] - L[x][0]), sqr(p[a][0] - R[x][0])) +
               max(sqr(p[a][1] - L[x][1]), sqr(p[a][1] - R[x][1]));
    }
    void query(int l, int r, int a) { // 点 a 邻域查询
        if (l > r) return ;
        int mid = (l + r) >> 1;
        ll t = sqr(p[mid][0] - p[a][0]) + sqr(p[mid][1] - p[a][1]);
        if (t > q.top()) q.pop(), q.push(t); // 更新答案
        ll disl = dist(a, lc[mid]), disr = dist(a, rc[mid]);
        if (disl > q.top() && disr > q.top()) // 两边都有机会更新，优先搜大的
            (disl > disr) ? (query(l, mid - 1, a), query(mid + 1, r, a)) : (query(mid + 1, r, a), query(l, mid - 1, a));
        else
            (disl > q.top()) ? query(l, mid - 1, a) : query(mid + 1, r, a);
    }
}
using namespace KDTree;
int main() {
    red(n); red(k); k *= 2;
    for (int i = 1; i <= k; ++i) q.push(0);
    for (int i = 1; i <= n; ++i) red(p[i][0]), red(p[i][1]);
    build(1, n);
    for (int i = 1; i <= n; ++i) query(1, n, i);
    printf("%lld\n", q.top());
}
```

```cpp
// 动态 KDTree 维护空间权值 (单点修改 & 空间查询)
// 时间复杂度 O(log n) ~ O(n^(1-1/k))
#define sqr(x) ((x) * (x))
namespace KDT {
    struct dat {
        int X[2];
        int &operator[](const int k) {return X[k];}
    } p[N];
    db alp = 0.725; // 重构常数
    int nowd;
    bool cmp(int a, int b) {return p[a][nowd] < p[b][nowd];}
    //  root: 根 cur: 总点数 d: 当前分割维度 lc/rc: 左右儿子 L/R: 当前空间范围 siz: 子树大小  sum/val 空间的值, 单点的值
    int root, cur, d[N], lc[N], rc[N], L[N][2], R[N][2], siz[N], sum[N], val[N];
    int g[N], t; // 用于重构的临时数组
    void pushup(int x) {
        siz[x] = siz[lc[x]] + siz[rc[x]] + 1;
        sum[x] = sum[lc[x]] + sum[rc[x]] + val[x];
        L[x][0] = R[x][0] = p[x][0];
        L[x][1] = R[x][1] = p[x][1];
        if (lc[x]) {
            umin(L[x][0], L[lc[x]][0]); umax(R[x][0], R[lc[x]][0]);
            umin(L[x][1], L[lc[x]][1]); umax(R[x][1], R[lc[x]][1]);
        }
        if (rc[x]) {
            umin(L[x][0], L[rc[x]][0]); umax(R[x][0], R[rc[x]][0]);
            umin(L[x][1], L[rc[x]][1]); umax(R[x][1], R[rc[x]][1]);
        }
    }
    int build(int l, int r) { // 对 g[1...t] 进行建树 , 对应点都是 g[x]。方差建树
        if (l > r) return 0;
        int mid = (l + r) >> 1;
        db av[2] = {0, 0}, va[2] = {0, 0};
        for (int i = l; i <= r; ++i) av[0] += p[g[i]][0], av[1] += p[g[i]][1];
        av[0] /= (r - l + 1); av[1] /= (r - l + 1);
        for (int i = l; i <= r; ++i) va[0] += sqr(av[0] - p[g[i]][0]), va[1] += sqr(av[1] - p[g[i]][1]);
        if (va[0] > va[1]) d[g[mid]] = nowd = 0;
        else d[g[mid]] = nowd = 1;
        nth_element(g + l, g + mid, g + r + 1, cmp);
        lc[g[mid]] = build(l, mid - 1); rc[g[mid]] = build(mid + 1, r);
        pushup(g[mid]);
        return g[mid];
    }
    void expand(int x) { // 将子树展开到临时数组里
        if (!x) return;
        expand(lc[x]);
        g[++t] = x;
        expand(rc[x]);
    }
    void rebuild(int &x) { // x 所在子树重构
        t = 0; expand(x);
        x = build(1, t);
    }
    bool chk(int x) {return alp * siz[x] <= (db)max(siz[lc[x]], siz[rc[x]]);} // 判断失衡
    void insert(int &x, int a) { // 插入点 a , p[a],val[a] 为其信息
        if (!x) { x = a; pushup(x); d[x] = rand() & 1; return; }
        if (p[a][d[x]] <= p[x][d[x]]) insert(lc[x], a);
        else insert(rc[x], a);
        pushup(x);
        if (chk(x)) rebuild(x); // 失衡暴力重构
    }
```

```cpp
    dat Lt, Rt; // 询问一块空间的值 (为了减小常数把参数放在外面)
    int query(int x) {
        if (!x || Rt[0] < L[x][0] || Lt[0] > R[x][0] || Rt[1] < L[x][1]
        || Lt[1] > R[x][1]) return 0; // 结点为空或与询问区间无交
        if (Lt[0] <= L[x][0] && R[x][0] <= Rt[0] && Lt[1] <= L[x][1]
        && R[x][1] <= Rt[1]) return sum[x]; // 区间完全覆盖
        int ret = 0;
        if (Lt[0] <= p[x][0] && p[x][0] <= Rt[0] && Lt[1] <= p[x][1]
        && p[x][1] <= Rt[1]) ret += val[x]; // 当前点在区间内
        return query(lc[x]) + query(rc[x]) + ret;
    }
}
using namespace KDT;
int main() {
    int n; read(n);
    for (int op;;) {
        read(op);
        switch (op) {
        case 1:
            ++cur; read(p[cur][0]); read(p[cur][1]); read(val[cur]);
            insert(root, cur);
            break;
        case 2:
            read(Lt[0]); read(Lt[1]); read(Rt[0]); read(Rt[1]);
            printf("%d\n", query(root));
            break;
        case 3: | return 0; break;
        }
    }
    return 0;
}
```

3.3 李超线段树

```cpp
// 李超线段树 对于 (x1,y1) (x2,y2) -> y=0*x+max(y1,y2) [x1,x1]
#define ls (x<<1)
#define rs (x<<1|1)
typedef long long ll;
typedef double db;
const int N = 100010;
const int M = 40000;
struct line {
    db k, b;
} lin[N];
db val(int id, db X) {return lin[id].k * X + lin[id].b;}
int D[N << 2], n, id;
void modify(int L, int R, int id, int l = 1, int r = M - 1, int x = 1) { // 线 lin[id], 范围 [L, R]
    if (L <= l && r <= R) {
        int mid = (l + r) >> 1, lid = D[x];
        db lst = val(D[x], mid), now = val(id, mid);
        if (l == r) {if (now > lst) D[x] = id; return ;}
        if (lin[id].k > lin[D[x]].k) {
            if (now > lst) D[x] = id, modify(L, R, lid, l, mid, ls); // id->lid
            else modify(L, R, id, mid + 1, r, rs);
        } else if (lin[id].k < lin[D[x]].k) {
            if (now > lst) D[x] = id, modify(L, R, lid, mid + 1, r, rs); // id->lid
            else modify(L, R, id, l, mid, ls);
        } else if (lin[id].b > lin[D[x]].k) D[x] = id;
        return ;
    }
    int mid = (l + r) >> 1;
    if (L <= mid) modify(L, R, id, l, mid, x << 1);
    if (R > mid) modify(L, R, id, mid + 1, r, x << 1 | 1);
}
int gmax(int x, int y, int ps) {
    if (val(x, ps) > val(y, ps)) return x;
    if (val(x, ps) < val(y, ps)) return y;
    return (x < y) ? x : y;
}
int query(int ps, int l = 1, int r = M - 1, int x = 1) { // 查 x=ps
    if (l == r) return D[x];
    int mid = (l + r) >> 1, ret = D[x], t = 0;
    if (ps <= mid)
        t = query(ps, l, mid, ls);

    else
        t = query(ps, mid + 1, r, rs);
    return gmax(ret, t, ps);
}
```

3.4 吉司机线段树

```cpp
/*
 * seg-beats 吉司机线段树
 * 区间最值操作
 * 支持 区间取min, 区间取max, 区间加减, 区间求和, 区间最小/大
 * 复杂度 O(m log n)
 */
#define ls (x << 1)
#define rs (x << 1 | 1)
#define mid ((l + r) >> 1)
typedef long long ll;
const int N = 500010;
const int inf = 0x3f3f3f3f;
struct datmn {
    int fi, se, cnt; // 最小值, 次小值, 最小值个数
    datmn() {fi = se = inf; cnt = 0;}
    void ins(int x, int c) {
        if (x < fi) se = fi, cnt = c, fi = x;
        else if (x == fi) cnt += c;
        else if (x < se) se = x;
    }
    friend datmn operator+(const datmn &a, const datmn &b) {
        datmn r = a; r.ins(b.fi, b.cnt); r.ins(b.se, 0); return r;
    }
};
struct datmx {
    int fi, se, cnt;
    datmx() {fi = se = -inf; cnt = 0;}
    void ins(int x, int c) {
        if (x > fi) se = fi, cnt = c, fi = x;
        else if (x == fi) cnt += c;
        else if (x > se) se = x;
    }
    friend datmx operator+(const datmx &a, const datmx &b) {
        datmx r = a; r.ins(b.fi, b.cnt); r.ins(b.se, 0); return r;
    }
};

struct node {
    datmn mn; datmx mx;
    ll sum; int addmn, addmx, add, len;
} t[N << 2];
int n, m, a[N];
void pushup(int x) {
    t[x].mx = t[ls].mx + t[rs].mx;
    t[x].mn = t[ls].mn + t[rs].mn;
    t[x].sum = t[ls].sum + t[rs].sum;
}
void build(int l = 1, int r = n, int x = 1) {
    t[x].add = t[x].addmn = t[x].addmx = 0;
    t[x].len = r - l + 1;
    if (l == r) {
        t[x].mx = datmx(); t[x].mx.ins(a[l], 1);
        t[x].mn = datmn(); t[x].mn.ins(a[l], 1);
        t[x].sum = a[l];
        return;
    }
    build(l, mid, ls); build(mid + 1, r, rs);
    pushup(x);
}
void update(int x, int vn, int vx, int v) { // vn: addmn, vx: addmx, v: add
    // 所有数相同特判, 此时最大值 tag 和最小值 tag 应该相同且不等于其他值 tag
    if (t[x].mn.fi == t[x].mx.fi) {
        if (vn == v) vn = vx;
```

```cpp
        else vx = vn;
        t[x].sum += (ll)vn * t[x].mn.cnt;
    } else t[x].sum += (ll)vn * t[x].mn.cnt + (ll) vx * t[x].mx.cnt + (ll)v * (t[x].len - t[x].mn.cnt - t[x].mx.cnt);
    if (t[x].mn.se == t[x].mx.fi) t[x].mn.se += vx; // 次小值 = 最大值，应该用最大值 tag 处理
    else if (t[x].mn.se != inf) t[x].mn.se += v;
    if (t[x].mx.se == t[x].mn.fi) t[x].mx.se += vn; // 次大值同理
    else if (t[x].mx.se != -inf) t[x].mx.se += v;
    t[x].mn.fi += vn; t[x].mx.fi += vx;
    t[x].addmn += vn; t[x].addmx += vx; t[x].add += v;
}
void pushdown(int x) {
    int mn = min(t[ls].mn.fi, t[rs].mn.fi);
    int mx = max(t[ls].mx.fi, t[rs].mx.fi);
    update(ls, (mn == t[ls].mn.fi) ? t[x].addmn : t[x].add, (mx == t[ls].mx.fi) ? t[x].addmx : t[x].add, t[x].add);
    update(rs, (mn == t[rs].mn.fi) ? t[x].addmn : t[x].add, (mx == t[rs].mx.fi) ? t[x].addmx : t[x].add, t[x].add);
    t[x].add = t[x].addmn = t[x].addmx = 0;
}
void modifyadd(int L, int R, int v, int l = 1, int r = n, int x = 1) {
    if (r < L || R < l) return ;
    if (L <= l && r <= R) return update(x, v, v, v);
    pushdown(x);
    modifyadd(L, R, v, l, mid, ls);
    modifyadd(L, R, v, mid + 1, r, rs);
    pushup(x);
}
void modifymin(int L, int R, int v, int l = 1, int r = n, int x = 1) {
    if (r < L || R < l) return ;
    if (L <= l && r <= R && v > t[x].mx.se) {
        if (v >= t[x].mx.fi) return ;
        update(x, 0, v - t[x].mx.fi, 0);
        return;
    }
    pushdown(x);
    modifymin(L, R, v, l, mid, ls);
    modifymin(L, R, v, mid + 1, r, rs);
    pushup(x);
}
void modifymax(int L, int R, int v, int l = 1, int r = n, int x = 1) {
    if (r < L || R < l) return ;
    if (L <= l && r <= R && v < t[x].mn.se) {
        if (v <= t[x].mn.fi) return;
        update(x, v - t[x].mn.fi, 0, 0);
        return;
    }
    pushdown(x);
    modifymax(L, R, v, l, mid, ls);
    modifymax(L, R, v, mid + 1, r, rs);
    pushup(x);
}
int querymax(int L, int R, int l = 1, int r = n, int x = 1) {
    if (r < L || R < l) return -inf;
    if (L <= l && r <= R) return t[x].mx.fi;
    pushdown(x);
    return max(querymax(L, R, l, mid, ls), querymax(L, R, mid + 1, r, rs));
}
int querymin(int L, int R, int l = 1, int r = n, int x = 1) {
    if (r < L || R < l) return inf;
    if (L <= l && r <= R) return t[x].mn.fi;
    pushdown(x);
    return min(querymin(L, R, l, mid, ls), querymin(L, R, mid + 1, r, rs));
}
ll querysum(int L, int R, int l = 1, int r = n, int x = 1) {
    if (r < L || R < l) return 0;
    if (L <= l && r <= R) return t[x].sum;
    pushdown(x);
    return querysum(L, R, l, mid, ls) + querysum(L, R, mid + 1, r, rs);
}
```

3.5 FHQ Treep

```cpp
// fhq - treap 简易模板
#define ls(p) t[p].l
#define rs(p) t[p].r
#define mid ((l+r)>>1)
using namespace std;
const int N = 100010;
mt19937 rd(random_device{}());
struct node {
    int l, r, siz, rnd, val, tag;
} t[N]; int tot, root;
/* 节点回收
int cyc[N],cyccnt;
inline void delnode(int p) {cyc[++cyccnt]=p;}
inline void newnode(int val) {
    int id=(cyccnt>0)?cyc[cyccnt--]:++tot;
    t[id]={0,0,1,(int)(rd()),val}; return id;
}
*/
inline int newnode(int val) { t[++tot] = {0, 0, 1, (int)(rd()), val}; return tot;}
inline void updata(int p) {
    t[p].siz = t[ls(p)].siz + t[rs(p)].siz + 1;
    /* maintain */
}
inline void pushtag(int p, int vl) { /* tag to push */ }
inline void pushdown(int p) {
    if (t[p].tag != std_tag) {
        if (ls(p)) pushtag(ls(p), t[p].tag);
        if (rs(p)) pushtag(rs(p), t[p].tag);
        t[p].tag = std_tag;
    }
}
int merge(int p, int q) {
    if (!p || !q) return p + q;
    if (t[p].rnd < t[q].rnd) {
        pushdown(p);
        rs(p) = merge(rs(p), q);
        updata(p); return p;
    } else {
        pushdown(q);
        ls(q) = merge(p, ls(q));
        updata(q); return q;
    }
}
void split(int p, int k, int &x, int &y) {
    if (!p) x = 0, y = 0;
    else {
        pushdown(p);
        if (t[ls(p)].siz >= k) y = p, split(ls(p), k, x, ls(p));
        else x = p, split(rs(p), k - t[ls(p)].siz - 1, rs(p), y);
        updata(p);
    }
}
int build(int l, int r) { // build tree on a[l..r], return the root
    if (l > r) return 0;
    return merge(build(l, mid - 1), merge(newnode(a[mid]), build(mid + 1, r)));
}
```

3.6 哈希表

```cpp
typedef long long ll;
const int M = 19260817;
const int MAX_SIZE = 2000000;
struct Hash_map {
    struct data {
        int nxt;
        ll key, value; // (key,value)
    } e[MAX_SIZE];
    int head[M], size;
    inline int f(ll key) { return key % M; }
    ll &operator[](const ll &key) {
        int ky = f(key);
        for (int i = head[ky]; i != -1; i = e[i].nxt)
            if (e[i].key == key) return e[i].value;
```

```cpp
        return e[++size] = data{head[ky], key, 0}, head[ky]
          ↪ = size, e[size].value;
    }
    void clear() {
        memset(head, -1, sizeof(head));
        size = 0;
    }
    Hash_map() {clear();}
};
```

4. String

Good Luck && Have Fun!