

# Standard Source code Library

mayf3

October 9, 2014

# Contents

<b>1</b>	<b>图论</b>	<b>4</b>
1.1	最短路算法	4
1.1.1	k 最短路 (无环)	4
1.1.2	k 最短路	7
1.2	生成树	10
1.2.1	生成树计数	10
1.2.2	无向图的生成数计数-MatrixTree 定理	11
1.2.3	无向带权图最小生成树计数	11
1.2.4	带限制的最小生成树问题	11
1.2.5	K 度限制最小生成树	12
1.2.6	最小树形图, 输出边	15
1.2.7	最小树形图, $N \times N$	17
1.3	网络流	19
1.3.1	最大流	19
1.3.2	最大流 (fast)	21
1.3.3	最大流 (lby)	23
1.3.4	最小费用最大流	24
1.3.5	最小费用最大流 (fast)	26
1.3.6	混合图欧拉回路	29
1.3.7	stoer wagner 最小割集	31
1.3.8	最小割树	33
1.4	匹配	35
1.4.1	匈牙利算法 $O(V \times E)$	35
1.4.2	二分图匹配 $O(\sqrt{V} \times E)$	36
1.4.3	二分图最大权匹配	37
1.4.4	带花树	39
1.5	图	41
1.5.1	哈密尔顿回路	41
1.5.2	割点和桥	42
1.5.3	有向图割点	43
1.5.4	树链剖分	44
1.5.5	最大团	46
1.5.6	图平面化	46
1.5.7	LCA offline	53
1.5.8	LCA online	54
<b>2</b>	<b>数据结构</b>	<b>55</b>
2.1	平衡树	55
2.1.1	heap	55
2.1.2	BST	57
2.1.3	treap	59
2.1.4	splay	63
2.2	图上的数据结构	66
2.2.1	左偏树	66
2.2.2	支持子树操作的动态树	67
2.3	可持久化数据结构	70

2.3.1	函数式 treap	70
2.3.2	划分树	72
2.4	其他	73
2.4.1	矩形切割	73
<b>3</b>	<b>字符串算法</b>	<b>75</b>
3.1	基础	75
3.1.1	字符串循环节	75
3.1.2	manacher 求最长回文	76
3.1.3	字符串最小表示法	76
3.1.4	kmp	77
3.1.5	扩展 kmp	77
3.2	进阶	78
3.2.1	ac 自动机	78
3.2.2	Trie 图	79
3.2.3	后缀自动机	81
<b>4</b>	<b>计算几何</b>	<b>83</b>
4.1	平面几何基础	83
4.1.1	几何	83
4.1.2	圆	95
4.1.3	垂心, 内心, 外心	98
4.1.4	一般多边形	99
4.2	空间几何基础	103
4.2.1	三维几何	103
4.2.2	空间变换矩阵	112
4.3	凸包	115
4.3.1	凸包	115
4.3.2	三维凸包 $n*n$	117
4.3.3	三维凸包 $n*\log n$	121
4.3.4	动态凸包	126
4.3.5	两凸包间最短距离	127
4.4	平面	128
4.4.1	半平面交	128
4.4.2	动态半平面交	132
4.4.3	旋转卡壳	134
4.4.4	kd 树, 支持插入	135
4.4.5	knn 询问距离最近 K 个点	136
4.4.6	区域树 (查询区域内点数量)	139
4.5	面积交	144
4.5.1	圆与多边形面积交	144
4.5.2	k 多边形面积交	146
4.5.3	两个多边形面积交	148
4.5.4	k 圆面积交	151
4.5.5	矩形和多个圆并的面积交	153
4.6	其他	158
4.6.1	椭圆周长	158

<b>5</b>	<b>理论</b>	<b>158</b>
5.1	数学	158
5.1.1	数学结论	158
5.1.2	三次方程求解	160
5.1.3	辛普森积分	161
5.1.4	高斯消元	162
5.1.5	FFT	164
5.1.6	linear programming	165
5.1.7	线性递推式 $n*n*\log n$	169
5.1.8	线性递推式 +fft	170
5.1.9	牛顿插值法	174
5.2	数论	175
5.2.1	$O(n)$ 求逆元	175
5.2.2	中国剩余定理 (非互质)	176
5.2.3	取模	176
5.2.4	组合数计算	178
5.2.5	雅可比判别式 (二次剩余)	189
5.2.6	分解质因数	190
5.2.7	大数分解	191
5.2.8	米勒测试	193
5.2.9	pollard 分解质因数	194
5.2.10	pell 方程	196
5.2.11	二次剩余, 解数	197
5.2.12	二次剩余, 所有解	198
5.3	博弈论	203
5.3.1	nim 积	203
<b>6</b>	<b>其他</b>	<b>204</b>
6.0.2	高精度	204
6.0.3	计算器	208
6.0.4	最长公共上升子序列	211
6.0.5	罗马数字	213
6.0.6	最大子矩阵	213
6.0.7	精确覆盖	214
6.0.8	模糊覆盖	217
6.0.9	生成 N 皇后一组解	219
6.0.10	sgu313	220
6.0.11	MIPT015 最大矩阵	222
6.0.12	MIPT016 最大不相交矩阵	224
6.0.13	poj3016 可并堆	227

# 1 图论

## 1.1 最短路算法

### 1.1.1 k最短路（无环）

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <map>

using namespace std;

const int MAXN = 50 + 10; //number of vertices
const int MAXK = 200 + 10;
const int INF = 1000000000; //max dist

struct Tpath {
    int cnt, len, pos;
    int v[MAXN];
};

Tpath path[MAXK];
int g[MAXN][MAXN];
int len[MAXK], pos[MAXK], ans[MAXK];
bool used[MAXN];
int dist[MAXN], prev[MAXN], List[MAXN];
int N, M, K, S, T, cnt;

void Dijkstra() {
    int visited[MAXN];
    for (int i = 0; i <= N; ++i) dist[i] = INF, visited[i] = 0;
    dist[T] = 0;
    for (int k, i = T; i != N; i = k) {
        visited[i] = 1; k = N;
        for (int j = 0; j < N; ++j) {
            if (visited[j] || used[j]) continue;
            if (g[j][i] > -1 && dist[i] + g[j][i] < dist[j]) {
                dist[j] = dist[i] + g[j][i];
                prev[j] = i;
            }
            if (dist[j] < dist[k]) k = j;
        }
    }
}

void setPath(int v, Tpath &p) {
    p.len = 0;
```

```

while (1) {
    p.v[p.cnt++] = v;
    if (v == T) return;
    p.len += g[v][prev[v]]; v = prev[v];
}
}

void solve() {
    memset(used, 0, sizeof(used));
    Dijkstra();
    memset(ans, -1, sizeof(ans));
    if (dist[S] == INF) return;
    multimap<int, int> Q; Q.clear();
    path[0].cnt = 0; path[0].pos = 0; setPath(S, path[0]); Q.insert( make_pair(path[0].len, 0) );
    int tot = 1;
    for (int i = 0; i < K; ++i) {
        if (Q.empty()) return;
        multimap<int, int> :: iterator p = Q.begin();
        int x = (*p).second;
        ans[i] = path[x].len;
        if (i == K - 1) break;
        memset(used, 0, sizeof(used));
        Tpath cur; cur.cnt = 0; cur.len = 0;
        for (int sum = 0, j = 0; j + 1 < path[x].cnt; ++j) {
            cur.v[cur.cnt++] = path[x].v[j]; used[path[x].v[j]] = 1;
            if (j) sum += g[path[x].v[j - 1]][path[x].v[j]];
            if (j >= path[x].pos) {
                Dijkstra();
                int u = path[x].v[j];
                for (int v = 0; v < N; ++v)
                    if (g[u][v] > -1 && !used[v] && dist[v] < INF && v != path[x].v[j + 1]) {
                        Tpath tp = cur; tp.pos = j + 1; setPath(v, tp); tp.len += sum + g[u][v];
                        if (tot < K) path[tot] = tp, Q.insert( make_pair(tp.len, tot++) );
                        else {
                            multimap<int, int> :: iterator p = Q.end(); --p;
                            if (tp.len >= (*p).first) continue;
                            path[(*p).second] = tp; Q.insert( make_pair(tp.len, (*p).second) );
                            Q.erase(p);
                        }
                    }
            }
        }
        Q.erase(p);
    }
}

void DFS(int step, int u, int len) {
    if (!cnt) return;

```

```

if (u == T) {
    if (len == ans[K - 1]) {
        if (!(--cnt)) {
            for (int j = 0; j < step; ++j) {
                if (j) printf("-");
                printf("%d", List[j] + 1);
            }
            printf("\n");
        }
    }
    return;
}
Dijkstra();
int tmp[MAXN];
for (int i = 0; i < N; ++i) tmp[i] = dist[i];
for (int i = 0; i < N; ++i)
    if (g[u][i] > -1 && !used[i] && tmp[i] < INF && len + g[u][i] + tmp[i] <= ans[K - 1]) {
        used[i] = 1; List[step] = i;
        DFS(step + 1, i, len + g[u][i]);
        if (!cnt) return;
        used[i] = 0;
    }
}

int main() {
    scanf("%d%d%d%d", &N, &M, &K, &S, &T);
    --S; --T;
    if (S == T) ++K;
    memset(g, -1, sizeof(g));
    for (int i = 0; i < M; ++i) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        --u; --v;
        g[u][v] = w;
    }
    solve();
    if (ans[K - 1] == -1) printf("None\n");
    else {
        cnt = 0;
        for (int i = 0; i < K; ++i)
            if (ans[i] == ans[K - 1]) ++cnt;
        memset(used, 0, sizeof(used));
        used[S] = 1; List[0] = S;
        DFS(1, S, 0);
    }
    return 0;
}

```

### 1.1.2 k最短路

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <queue>

using namespace std;

const int MAXN = 1000 + 10; //number of vertices
const int MAXM = 100000 + 10; //number of edges
const int MAXK = 1000 + 10;
const int MAXH = 200000; //M + N log N
const int INF = 1000000000; //max dist

struct Theap {
    int idx, dep;
    int chd[3];
};

struct Tedge {
    int u, v, w, delta;
    bool inT;
};

Theap heap[MAXN];
Tedge edge[MAXM];
int first[MAXN], rfirst[MAXN], outdeg[MAXN], dist[MAXN], nextT[MAXN], list[MAXN], H1[MAXN], H2[MAXN];
int next[MAXM], rnext[MAXM];
int ans[MAXK];
int N, M, K, S, T, nlist, H, curedge;

void Dijkstra() {
    priority_queue < pair<int, int>, vector< pair<int, int> >, greater< pair<int, int> > > Q;
    for (int i = 0; i < N; ++i) dist[i] = INF;
    dist[T] = 0; Q.push( make_pair(0, T) );
    while (!Q.empty()) {
        int u = Q.top().second, d = Q.top().first;
        Q.pop();
        if (d > dist[u]) continue;
        for (int i = rfirst[u]; i != -1; i = rnext[i]) {
            int v = edge[i].u, w = edge[i].w;
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                Q.push( make_pair(dist[v], v) );
            }
        }
    }
}
```



```

}

void DFS(int u) {
    list[nlist++] = u;
    for (int i = rfirst[u]; i != -1; i = rnext[i]) {
        int v = edge[i].u, w = edge[i].w;
        if (!edge[i].delta && nextT[v] == -1) {
            nextT[v] = u; edge[i].inT = 1;
            DFS(v);
        }
    }
}

int buildH1(int Size, int dep) {
    if (!Size) return 0;
    if (edge[curedge].inT) curedge = next[curedge];
    int cur = H++;
    heap[cur].idx = curedge; curedge = next[curedge];
    heap[cur].chd[2] = 0;
    if (!dep) heap[cur].chd[0] = buildH1(Size - 1, dep + 1), heap[cur].chd[1] = 0;
    else {
        int half = (Size - 1) / 2;
        heap[cur].chd[0] = buildH1(half, dep + 1); heap[cur].chd[1] = buildH1(Size - 1 - half, dep + 1);
    }
    int i = cur;
    while (1) {
        int k = i;
        for (int j = 0; j < 2; ++j)
            if (heap[i].chd[j] && edge[heap[heap[i].chd[j]].idx].delta < edge[heap[k].idx].delta)
                if (k == i) break;
        swap(heap[k].idx, heap[i].idx); i = k;
    }
    return cur;
}

int buildH2(int a, int b) {
    if (!a) {
        heap[b].chd[0] = heap[b].chd[1] = 0; heap[b].dep = 1;
        return b;
    }
    int Next = heap[heap[a].chd[0]].dep >= heap[heap[a].chd[1]].dep;
    int cur = H++;
    heap[cur] = heap[a];
    if (edge[heap[b].idx].delta < edge[heap[a].idx].delta) {
        heap[b].chd[0] = heap[a].chd[0]; heap[b].chd[1] = heap[a].chd[1];
        heap[b].chd[Next] = buildH2(heap[b].chd[Next], cur);
        heap[b].dep = min(heap[heap[b].chd[0]].dep, heap[heap[b].chd[1]].dep) + 1;
        return b;
    }
}

```

```

    }
    else {
        heap[cur].chd[Next] = buildH2(heap[cur].chd[Next], b);
        heap[cur].dep = min(heap[heap[cur].chd[0]].dep, heap[heap[cur].chd[1]].dep) + 1;
        return cur;
    }
}

void solve() {
    Dijkstra();
    memset(ans, -1, sizeof(ans));
    if (dist[S] == INF) return;
    for (int i = 0; i < M; ++i) edge[i].delta = edge[i].w - dist[edge[i].u] + dist[edge[i].v];
    memset(nextT, -1, sizeof(nextT));
    nextT[T] = -2; nlist = 0;
    DFS(T);
    H = 1; heap[0].dep = 0;
    memset(H1, 0, sizeof(H1));
    for (int i = 0; i < N; ++i)
        if (dist[i] < INF) {
            int Size = outdeg[i];
            if (i != T) --Size;
            curedge = first[i];
            H1[i] = buildH1(Size, 0);
            if (H1[i]) {
                heap[H1[i]].chd[2] = heap[H1[i]].chd[0];
                heap[H1[i]].chd[0] = 0;
                heap[H1[i]].dep = 1;
            }
        }
    memset(H2, 0, sizeof(H2));
    H2[T] = H1[T];
    for (int i = 1; i < nlist; ++i) {
        int j = list[i];
        if (!H1[j]) H2[j] = H2[nextT[j]];
        else H2[j] = buildH2(H2[nextT[j]], H1[j]);
    }
    ans[0] = dist[S];
    priority_queue < pair<int, int>, vector< pair<int, int> >, greater< pair<int, int> > > Q;
    if (H2[S]) Q.push( make_pair(edge[heap[H2[S]].idx].delta, H2[S]) );
    for (int i = 1; i < K; ++i) {
        if (Q.empty()) break;
        int u = Q.top().second, d = Q.top().first;
        ans[i] = dist[S] + d;
        Q.pop();
        for (int j = 0; j < 3; ++j) {
            int v = heap[u].chd[j];
            if (v) Q.push( make_pair(d + edge[heap[v].idx].delta - edge[heap[u].idx].delta, v) );
        }
    }
}

```

```

    }
    int v = H2[edge[heap[u].idx].v];
    if (v) Q.push( make_pair(d + edge[heap[v].idx].delta, v) );
}
}

int main() {
    memset(first, -1, sizeof(first));
    memset(rfirst, -1, sizeof(rfirst));
    memset(outdeg, 0, sizeof(outdeg));
    scanf("%d%d", &N, &M);
    for (int i = 0; i < M; ++i) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        --u; --v;
        edge[i].u = u; edge[i].v = v; edge[i].w = w; edge[i].inT = 0;
        next[i] = first[u]; first[u] = i; rnext[i] = rfirst[v]; rfirst[v] = i;
        ++outdeg[u];
    }
    scanf("%d%d%d", &S, &T, &K);
    --S; --T;
    if (S == T) ++K;
    solve();
    printf("%d\n", ans[K - 1]);
    return 0;
}

```

## 1.2 生成树

### 1.2.1 生成树计数

一个完全图  $K_n$  有  $n^{n-2}$  棵生成树，即  $n$  个节点的带标号无根树有  $n^{n-2}$  个。

证明用到 **prufercode**：一棵  $n$  无根树的 **prufercode** 是这样转化的：每次选一个编号最小的叶节点，删除它，并把它所连的父亲节点的编号写下，直到这棵树剩下 2 个节点为止。那么生成的这  $n-2$  个数组成的序列就是这棵树的 **prufercode**。prufercode 和树是一一对应的，而一个长度为  $n-2$ ，每个数字的范围为  $[1, n]$  的序列一共有  $n^{n-2}$  种可能，所以  $n$  个节点的带标号无根树一共有  $n^{n-2}$  个。

以上是 Cayley 公式，它的一个应用： $n$  个节点，每个节点的度分别为  $d_1, d_2, \dots, d_n$ ，那么生成树的个数为  $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\dots(d_n-1)!}$ 。因为顶点  $i$  在序列中出现了  $d_i - 1$  次。

对于完全二分图，两边的顶点分别为  $n, m$ ，那么生成树的个数为  $n^{m-1} * m^{n-1}$ 。

### 1.2.2 无向图的生成数计数-MatrixTree 定理

给出一个无向图  $G = (V, E)$ ，求生成树个数。做法是构造一个  $n * n$  的 Kirchhoff 矩阵。矩阵的对角线  $(i, i)$  的位置填的是第  $i$  个顶点的度，对于  $G$  的边  $(v_i, v_j)$  在矩阵  $(i, j)$  和  $(j, i)$  的位置填 -1 (若  $(i, j)$  有  $k$  条重边，那么矩阵  $(i, j)$  和  $(j, i)$  的位置填  $-k$ )，然后生成树的个数就是  $n * n$  的矩阵的  $n - 1$  阶的行列式。具体做法就是删除任意的第  $r$  行  $r$  列，然后求矩阵的行列式。

### 1.2.3 无向带权图最小生成树计数

把所有的边按照边权从小到大排序，然后做 Kruskal。

假设已经处理了边权  $w_i < w$  的边，形成一个森林  $T$ ，现在考虑所有边权为  $w$  的边。

1、若一条边权为  $w$  的边  $(u, v)$  所连接的两个顶点在森林  $T$  中属于同一个块，那么  $(u, v)$  这条边是不可能存在于最小生成树的方案中，否则  $(u, v)$  可以存在于最小生成树的方案中。

2、把所有边权为  $w$ ，且根据森林  $T$  判断出可以存在于最小生成树方案中的边找出来，假设这些边集为  $E$ 。我们可以把森林  $T$  中的每一个块缩成一个点，那么用  $E$  中的边去连接  $T$ ，就形成了一些连通块。对于每一个连通块的方案数就是对这个连通块做一个生成树计数就可以了，然后把把这些连通块各自的方案数相乘就是选择边权为  $w$  的边的方案数。

3、算完边权为  $w$  的方案之后，就把这些边加入到  $T$  中，形成新的森林。

### 1.2.4 带限制的最小生成树问题

#### • Problem

无向带权连通图，每条边是黑色或白色。让你求一棵最小权的恰好有  $K$  条白色边的生成树。

#### • Solution

1、对于一个图，如果存在一棵生成树，它的白边数量为  $x$ ，那么就称  $x$  是合法白边数。所有的合法白边数组成一个区间  $[l, r]$ 。

2、对于一个图，如果存在一棵最小生成树，它的白边数量为  $x$ ，那么就称  $x$  是最小合法白边数。所有的最小合法白边数组成一个区间  $[l, r]$ 。

3、将所有白边追加权值  $x$  所得到的最小生成树，如果该树有  $a$  条白边。那么这棵树就是  $a$  条白边最小生成树的一个最优解。

所以可以二分得到一个最大的  $x$  使得所求的最小生成树的白边的最小值和最大值所组成的一个区间  $[l, r]$ ，若  $K \in [l, r]$ ，则该最小生成树就是最优解。其实只要求出最大的一个  $x$  使得最小生成树中最大白边数量不小于  $K$  即可。记录答案的时候，必须把枚举的  $x$  加上，然后在最后减去  $K * x$ ，如果直接在计算的时候加原来白边的长度的话，有可能超过  $K$  条边。

对于黑、白不同的边，他们内部的顺序是一样的，所以一开始将黑白边分别排序，这样在二分判断的时候只需要  $O(M)$  的时间复杂度去合并排序的边了，其中  $M$  是边数。总的时间复杂度为  $O(M \log W + N \log N)$  其中  $N$  是顶点数， $W$  是边权。

#### • Expansion

1、限制某个节点  $node$  的度数恰好为  $K$  的最小生成树。解法就是把和  $node$  关联的边标记为白边，其余的边为黑边，然后就转化为上面的经典问题了。

### 1.2.5 K度限制最小生成树

```
/*
    Find a minimum spanning tree whose vertex 1 has a degree limit D
*/
#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;

const int MAXN = 1000 + 1; //number of vertices + 1
const int MAXM = 100000; //number of edges
const int INF = 2000000000;

struct Tedge {
    int v, w, next;
};

Tedge edge[MAXM * 2], mst_edge[MAXM * 2];
int first[MAXN], mst_first[MAXN], dist[MAXN], heap[MAXN], pos[MAXN], maxw[MAXN], path[MAXN], p;
bool used[MAXN];
int N, M, D, cnt, num, ans;

inline void add_edge(Tedge& e, int& first, int i, int v, int w) {
    e.v = v; e.w = w; e.next = first; first = i;
}

void init() {
    memset(first, -1, sizeof(first));
    scanf("%d%d%d", &N, &M, &D);
    for (int i = 0; i < M; ++i) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        --u; --v;
        add_edge(edge[i * 2], first[u], i * 2, v, w);
        add_edge(edge[i * 2 + 1], first[v], i * 2 + 1, u, w);
    }
}

inline void moveup(int i) {
    int key = heap[i];
    while (i > 1 && dist[heap[i >> 1]] > dist[key])    heap[i] = heap[i >> 1], pos[heap[i]] = i, i >> 1;
    heap[i] = key; pos[key] = i;
}

inline void movedown(int i) {
```

```

    int key = heap[i];
    while ((i << 1) <= num) {
        int j = i << 1;
        if (j < num && dist[heap[j + 1]] < dist[heap[j]]) ++j;
        if (dist[key] <= dist[heap[j]]) break;
        heap[i] = heap[j]; pos[heap[i]] = i; i = j;
    }
    heap[i] = key; pos[key] = i;
}

void Prim(int u) {
    int minw = INF, s;
    num = 0;
    while (1) {
        used[u] = 1;
        for (int i = first[u]; i != -1; i = edge[i].next) {
            int v = edge[i].v, w = edge[i].w;
            if (!used[v] && (dist[v] == -1 || w < dist[v])) {
                dist[v] = w;
                prev[v] = u;
                if (pos[v] == -1) pos[v] = ++num, heap[num] = v;
                moveup(pos[v]);
            }
            else if (used[v] && v == 0 && w < minw) minw = w, s = i;
        }
        if (!num) break;
        u = heap[1]; heap[1] = heap[num--]; movedown(1);
        ans += dist[u];
        add_edge(mst_edge[cnt], mst_first[u], cnt, prev[u], dist[u]); ++cnt;
        add_edge(mst_edge[cnt], mst_first[prev[u]], cnt, u, dist[u]); ++cnt;
    }
    if (minw == INF) return;
    edge[s].w = -1; edge[s ^ 1].w = -1;
    s = edge[s ^ 1].v; ans += minw; --D;
    add_edge(mst_edge[cnt], mst_first[0], cnt, s, minw); ++cnt;
    add_edge(mst_edge[cnt], mst_first[s], cnt, 0, minw); ++cnt;
}

void DFS(int u) {
    used[u] = 1;
    for (int i = mst_first[u]; i != -1; i = mst_edge[i].next) {
        int v = mst_edge[i].v, w = mst_edge[i].w;
        if (w > -1 && !used[v]) {
            if (w > maxw[v]) maxw[v] = w, path[v] = i;
            if (maxw[u] > maxw[v]) maxw[v] = maxw[u], path[v] = path[u];
            DFS(v);
        }
    }
}

```

```

}

void work() {
    ans = cnt = 0;
    memset(mst_first, -1, sizeof(mst_first));
    memset(dist, -1, sizeof(dist));
    memset(pos, -1, sizeof(pos));
    memset(used, 0, sizeof(used));
    used[0] = 1;
    for (int i = first[0]; i != -1; i = edge[i].next)
        if (!used[edge[i].v]) Prim(edge[i].v);
    if (D < 0) {
        printf("NONE\n");
        return;
    }
    for (int i = 1; i < N; ++i)
        if (!used[i]) {
            printf("NONE\n");
            return;
        }
    memset(maxw, -1, sizeof(maxw));
    memset(used, 0, sizeof(used));
    used[0] = 1;
    for (int i = mst_first[0]; i != -1; i = mst_edge[i].next) DFS(mst_edge[i].v);
    for (int i = 0; i < D; ++i) {
        int minw = INF, s, x, y;
        for (int j = first[0]; j != -1; j = edge[j].next) {
            int v = edge[j].v, w = edge[j].w;
            if (w > -1 && maxw[v] > -1 && w - maxw[v] < minw) {
                minw = w - maxw[v]; s = v;
                x = path[v]; y = j;
            }
        }
        if (minw >= 0) break;
        ans += minw;
        mst_edge[x].w = mst_edge[x ^ 1].w = -1;
        add_edge(mst_edge[cnt], mst_first[0], cnt, s, edge[y].w); ++cnt;
        add_edge(mst_edge[cnt], mst_first[s], cnt, 0, edge[y].w); ++cnt;
        edge[y].w = edge[y ^ 1].w = -1;
        memset(used, 0, sizeof(used));
        used[0] = 1;
        for (int u = 0; u < N; ++u)
            if (path[u] == x) maxw[u] = -1;
        DFS(s);
    }

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

```

```

int main() {
    int c;
    for (scanf("%d", &c); c > 0; --c) {
        init();
        work();
    }
    return 0;
}

```

### 1.2.6 最小树形图, 输出边

```

#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;

const int maxn = 200 + 1; //number of vertices
const int maxnum = 200000001; //max weight

int map[maxn][maxn], dist[maxn][maxn], list[maxn][maxn];
int Q[maxn], c[maxn], d1[maxn], d2[maxn];
bool used[maxn];
int n, m, ans, x, y, p;

void init() {
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j)
            map[i][j] = maxnum;
        map[i][i] = 0;
    }
    for (int i = 0; i < m; ++i) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        --u; --v;
        if (w > map[u][v]) continue;
        map[u][v] = map[v][u] = w;
    }
}

void APSP() {
    for (int s = 0; s < n; ++s) {
        memset(used, 0, sizeof(used));
        for (int i = 0; i <= n; ++i) dist[s][i] = maxnum, list[s][i] = n;
        dist[s][s] = 0;
        for (int k, u = s, cnt = 0; u < n; u = k, ++cnt) {

```



```

        list[s][cnt] = u; used[u] = 1; k = n;
        for (int v = 0; v < n; ++v) {
            if (used[v]) continue;
            dist[s][v] = min(dist[s][v], dist[s][u] + map[u][v]);
            if (dist[s][v] < dist[s][k]) k = v;
        }
    }
}

void work() {
    ans = maxnum;
    for (int u = 0; u < n; ++u)
        if (dist[u][list[u][n - 1]] + dist[u][list[u][n - 2]] < ans) {
            ans = dist[u][list[u][n - 1]] + dist[u][list[u][n - 2]];
            x = y = u;
        }
    for (int u = 0; u < n; ++u)
        for (int v = u + 1; v < n; ++v)
            if (map[u][v] < maxnum) {
                for (int i = 0; i < n; ++i) d1[i] = dist[u][list[u][i]], d2[i] = dist[v][list[u][i]];
                for (int j = n - 1, i = n - 2; i >= 0; --i) {
                    if (d1[i] == d1[j]) {
                        if (d2[i] > d2[j]) j = i;
                        continue;
                    }
                    if (d2[i] > d2[j]) {
                        if (map[u][v] + d1[i] + d2[j] < ans) {
                            ans = map[u][v] + d1[i] + d2[j];
                            x = u; y = v;
                            p = map[u][v] + d2[j] - d1[i];
                        }
                        j = i;
                    }
                }
            }
    }

void print() {
    printf("%d\n", ans);
    if (x == y) {
        for (int i = 0; i < n; ++i) c[i] = dist[x][i];
        p = 1;
    }
    else {
        printf("%d %d\n", x + 1, y + 1);
        for (int i = 0; i < n; ++i)
            if (dist[x][i] * 2 + p < (dist[y][i] + map[x][y]) * 2 - p) c[i] = dist[x][i] * 2 + p;
    }
}

```

```

        else c[i] = (dist[y][i] + map[x][y]) * 2 - p;
    p = 2;
}
memset(used, 0, sizeof(used));
int t = 1;
Q[0] = x; used[x] = 1;
if (x != y) Q[t++] = y, used[y] = 1;
for (int h = 0; h < t; ++h) {
    int u = Q[h];
    for (int v = 0; v < n; ++v)
        if (!used[v] && c[v] == c[u] + map[u][v] * p) {
            used[v] = 1;
            if (u < v) printf("%d %d\n", u + 1, v + 1);
            else printf("%d %d\n", v + 1, u + 1);
            Q[t++] = v;
        }
}
}

int main() {
    init();
    APSP();
    work();
    print();
    return 0;
}

```

### 1.2.7 最小树形图, $N \times N$

```

const int MAXN = 1000 + 10; //number of vertices
const int INF = 1000000000;

struct Edge {
    int u, c;
    friend bool operator<(const Edge &a, const Edge &b) { return a.c < b.c; }
};

Edge edge[MAXN][MAXN];
int top[MAXN], add[MAXN];

int set_find(int *s, int x) {
    return s[x] == x ? x : s[x] = set_find(s, s[x]);
}

inline void set_union(int *s, int x, int y) {
    int fx = set_find(s, x), fy = set_find(s, y);
    if (fx != fy) s[fy] = fx;
}

```

```

void merge(int a, int b) {
    static Edge ret[MAXN];
    static bool used[MAXN];
    int size = 0;
    memset(used, 0, sizeof(used));
    for (int i = 0; i < top[a]; ++i) edge[a][i].c += add[a];
    for (int i = 0; i < top[b]; ++i) edge[b][i].c += add[b];
    while (top[a] > 0 || top[b] > 0) {
        int id = (top[a] > 0 && (top[b] == 0 || edge[a][top[a] - 1].c > edge[b][top[b] - 1].c)) ? a : b;
        if (!used[edge[id][top[id] - 1].u]) {
            ret[size] = edge[id][top[id] - 1];
            used[ret[size++].u] = true;
        }
        --top[id];
    }
    top[a] = size;
    for (int i = 0; i < size; ++i) edge[a][i] = ret[size - i - 1];
    add[a] = 0;
}

// find maximal branchings,  $O(N^2)$ 
// make sure that there exists a solution
// delete all edge(i, root) from c to find a rooted solution
int optimum_branchings(int c[MAXN][MAXN], int N) {
    static int S[MAXN], W[MAXN];
    static Edge enter[MAXN];
    memset(add, 0, sizeof(add));
    memset(top, 0, sizeof(top));
    for (int j = 0; j < N; ++j) {
        for (int i = 0; i < N; ++i)
            if (c[i][j] < INF) {
                edge[j][top[j]].u = i;
                edge[j][top[j]++].c = c[i][j];
            }
        sort(edge[j], edge[j] + top[j]);
        S[j] = W[j] = j;
        enter[j].u = -1;
    }
    int ret = 0;
    for (int u = 0; u < N; ) {
        if (top[u] == 0) {
            ++u;
            continue;
        }
        Edge e = edge[u][--top[u]], t;
        e.c += add[u];
        if (set_find(S, e.u) != u) {

```

```

        if (set_find(W, e.u) != set_find(W, u)) {
            set_union(W, e.u, u);
            enter[u++] = e;
        }
        else {
            int val = e.c, v;
            for (t = enter[set_find(S, e.u)]; t.u != -1; t = enter[set_find(S, t.u)]) val = min(val, t.c);
            add[u] += val - e.c;
            e.c -= val;
            for (v = set_find(S, e.u), t = enter[v]; t.u != -1; v = set_find(S, t.u), t = enter[t.u])
                add[v] += val - t.c;
            merge(u, v);
            set_union(S, u, v);
        }
    }
    ret += e.c;
}
return ret;
}

```

### 1.3 网络流

Graph modification : Given graph  $G$ , we have to find the flow from node  $s$  to  $t$  so that:  $F[u][v]$ : amount of flow passes through edge  $(u, v)$ .  $F[u][v]$  must satisfy:  $D[u][v] \leq F[u][v] \leq C[u][v]$ .

We add two new nodes  $s'$  and  $t'$  into the graph and rebuild the graph  $G$  to make new graph  $G'$ : Each edge  $(u, v)$  in the graph  $G$  will correspond to 3 edges in graph  $G'$ : edge  $(s', v)$  with capacity =  $D[u][v]$ , edge  $(u, t')$  with capacity =  $D[u][v]$  and edge  $(u, v)$  with capacity =  $C[u][v] - D[u][v]$ . In addition, we add new edge  $(t, s)$  with capacity = infinity ( i.e very very big ).

Theorem: There exists the flow in graph  $G$  that satisfy  $D[u][v] \leq F[u][v] \leq C[u][v]$  if and only if there exists one flow from  $s'$  to  $t'$  in graph  $G' = \text{Sum}(D[u][v])$  with edge  $(u, v)$  in  $G$ .

It's clear that if there exists such flow in graph  $G'$ , it must be the maxflow from  $s'$  to  $t'$ . So we only have to check whether maxflow from  $s'$  to  $t'$  is equal to  $\text{Sum}(D[u][v])$  or not. If there exists such one, now to define the flow graph in  $G$ , we just add  $D[u][v]$  into  $F'[u][v]$  (  $F'[u][v]$ : amount of flow passes through edge  $(u, v)$  in graph  $G'$  ) and  $F[u][v] = F'[u][v] + D[u][v]$ .

Maximum feasible flow: Modify  $C[v][u]$  into  $D[u][v]$ , then use any maxflow algorithm of ZERO lower bounds from  $S$  to  $T$  to get the Maximum feasible flow.

Minimum feasible flow: Modify  $C[v][u]$  into  $D[u][v]$ , then use any maxflow algorithm of ZERO lower bounds from  $T$  to  $S$  to get the Minimum feasible flow.

#### 1.3.1 最大流

```
#include "template.cpp"
```

```
const int N = 222;
const int E = N * N;
```

```
struct Cla{
```

```

struct Edge{
    int t, u;
    Edge *nx, *op;
}*e[N], mem[E], *C;
int n, S, T;
int vh[N], h[N];

void init(int num){
    n = num;
    C = mem;
    rep(i, n) e[i] = NULL;
}

void add_edge(int u, int v, int c){
    C->t = v, C->u = c, C->nx = e[u], e[u] = C++;
    C->t = u, C->u = 0, C->nx = e[v], e[v] = C++;
    e[u]->op = e[v];
    e[v]->op = e[u];
}

int aug(int p, int m){
    if (p == T) return m;
    int d = m;
    for(Edge *i = e[p]; i; i = i->nx){
        if (i->u && h[p] == h[i->t] + 1){
            int f = aug(i->t, min(i->u, d));
            i->u -= f, i->op->u += f, d -= f;
            if (h[S] == n || !d) return m - d;
        }
    }
    int w = d < m ? min(n, h[p] + 2) : n;
    for(Edge *i = e[p]; i; i = i->nx){
        if (i->u) w = min(w, h[i->t] + 1);
    }
    ++vh[w];
    --vh[h[p]] ? h[p] = w : h[S] = n;
    return m - d;
}

int maxflow(int s, int t){
    S = s, T = t;
    rep(i, n) vh[i] = 0, h[i] = 0;
    vh[0] = n;
    int ret = 0;
    while(h[S] != n) ret += aug(S, INT_MAX);
    return ret;
}
}graph;

```

### 1.3.2 最大流(fast)

```
struct cla {
    struct Edge {
        int a, b, c, f; //from, to, c, flow;
        int xv(int u) { return u == a ? b : a; } //other
        int xf(int u) { return u == a ? c - f : f; } //limit
        void add(int u, int xf){ u == a ? f += xf : f -= xf; }
    } e[maxn*maxn*2];
    // edge[maxn*maxn], * adj[maxn][maxn*2], *path[maxn];

    int n, m, up;
    int rn[maxn], p[maxn];
    vector<int> r[maxn];
    int h[maxn], cur[maxn];
    int hn[maxn*2], pre[maxn];

    void st(int x) {
        n=x;
        for(int i=0;i<=n;i++)rn[i]=0;
        for (int i = 0; i <= n; ++ i) r[i].clear();
        up = 0;
    }

    void addedge(int x, int y, int w) {
        e[up].a = x;
        e[up].b = y;
        e[up].c = w;
        //      r[x][rn[x]++] = up;
        //      r[y][rn[y]++] = up;
        r[x].push_back(up);
        r[y].push_back(up);
        ++ rn[x];
        ++ rn[y];
        up++;
    }

    void ch(int h0,int n) {
        if (hn[h0] != 0) return;
        for (int i = 0; i < n; i++)
            if (h[i] > h0)
            {
                hn[h[i]]--;
                hn[n]++;
                h[i] = n;
            }
    }
}
```

```

int maxflow(int s,int t) {
    int i, c = s, v, ex, flow = 0, oldh;
    fill(h,h+n+1,0);//memset
    fill(hn,hn+2*n+1,0);//memset
    hn[0]=n;
    for(i=0;i<up;i++) e[i].f=0;
    for (i = 0; i < n; i++) cur[i] = rn[i] - 1;
    while (h[s] < n) {
        for (; cur[c] >= 0; cur[c]--){
            if (e[r[c][cur[c]]].xf(c)>0 && h[c]==h[e[r[c][cur[c]]].xv(c)]+1) break;
        }

        if (cur[c] >= 0) {
            v = e[r[c][cur[c]]].xv(c);
            pre[v] = c;
            p[v] = r[c][cur[c]];
            c = v;
            if (c == t) {
                ex = oo3;
                for (; c != s; c = pre[c]){
                    ex = min(ex, e[p[c]].xf(pre[c]));
                }

                for (c = t; c != s; c = pre[c]) {
                    e[p[c]].add(pre[c], ex);
                }

                flow += ex;
            }
        }
        else {
            oldh = h[c];
            h[c] = n;
            for (i = 0; i < rn[c]; i++){
                if (e[r[c][i]].xf(c) > 0){
                    h[c] = min(h[c], h[e[r[c][i]].xv(c)] + 1);
                }
            }

            cur[c] = rn[c] - 1;
            hn[oldh]--;
            hn[h[c]]++;
            if (c != s) c = pre[c];
            ch(oldh,n);
        }
    }
    return flow;
}
}graph;

```

### 1.3.3 最大流 (lby)

```
const int inf = 0x3f3f3f3f;
const LL infLL = 0x3f3f3f3f3f3f3f3fLL;

const int maxn = 100000 + 5;
const int maxe = 200000 + 5;

class SAP {
    int n, psz, s, t;
    struct Edge {
        int v, r;
        Edge *next, *cp;
    } epool[maxe], *e[maxn], *cur[maxn];
    int dis[maxn], gap[maxn], pre[maxn];

    void bfs() {
        queue<int> que;
        memset(dis, 0x3f, sizeof(dis));
        memset(gap, 0, sizeof(gap));
        dis[t] = 0; que.push(t);
        while (!que.empty()) {
            int u = que.front(); que.pop(); ++gap[dis[u]];
            for (Edge *i = e[u]; i; i = i->next) {
                int v = i->v;
                if (i->cp->r && dis[v] == inf) {
                    dis[v] = dis[u] + 1;
                    que.push(v);
                }
            }
        }
    }

    int aug(int &u) {
        if (u == t) {
            int d = inf;
            for (int i = s; i != t; i = cur[i]->v)
                if (cur[i]->r < d) d = cur[i]->r, u = i;
            for (int i = s; i != t; i = cur[i]->v)
                cur[i]->r -= d, cur[i]->cp->r += d;
            return d;
        }
        for (Edge *i = e[u]; i; i = i->next) {
            int v = i->v;
            if (i->r && dis[u] == dis[v] + 1) {
                cur[u] = i; pre[v] = u; u = v;
                return 0;
            }
        }
    }
};
```



```

    }
    if (--gap[dis[u]]) {
        int d = n;
        for (Edge *i = e[u]; i; i = i->next) {
            int v = i->v;
            if (i->r && dis[v] < d) d = dis[v], cur[u] = i;
        }
        ++gap[dis[u]=d+1];
        if (u != s) u = pre[u];
    } else {
        dis[s] = n;
    }
    return 0;
}

public:
void init(int n) {
    this->n = n; psz = 0;
    memset(e, 0, sizeof(e));
}

void add_edge(int u, int v, int w) {
    Edge *i = epool + psz++;
    i->v = v; i->r = w; i->next = e[u]; e[u] = i;
    i->cp = epool + ((psz - 1) ^ 1);
    if (psz & 1) add_edge(v, u, w);
}

int max_flow(int s, int t) {
    int flow = 0;
    this->s = s, this->t = t;
    bfs();
    // memset(dis, 0, sizeof(dis));
    // memset(gap, 0, sizeof(gap)); gap[0] = n;
    for (int i = 0; i < n; ++i) cur[i] = e[i];
    for (int i = s; dis[s] < n; flow += aug(i));
    return flow;
}
} graph;

```

#### 1.3.4 最小费用最大流

```

#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
#include <climits>
#include <numeric>
#include <vector>
#include <queue>
using namespace std;

```

```

template<class Flow = int, class Cost = int>
struct MinCostFlow {
    struct Edge {
        int t;
        Flow f;
        Cost c;
        Edge*next, *rev;
        Edge(int _t, Flow _f, Cost _c, Edge*_next) :
            t(_t), f(_f), c(_c), next(_next) {
        }
    };

    vector<Edge*> E;

    int addV() {
        E.push_back((Edge*) 0);
        return E.size() - 1;
    }

    Edge* makeEdge(int s, int t, Flow f, Cost c) {
        return E[s] = new Edge(t, f, c, E[s]);
    }

    void addEdge(int s, int t, Flow f, Cost c) {
        Edge*e1 = makeEdge(s, t, f, c), *e2 = makeEdge(t, s, 0, -c);
        e1->rev = e2, e2->rev = e1;
    }

    pair<Flow, Cost> minCostFlow(int vs, int vt) { //flow, cost
        int n = E.size();
        Flow flow = 0;
        Cost cost = 0;
        // const Cost MAX_COST = numeric_limits<Cost>::max();
        // const Flow MAX_FLOW = numeric_limits<Flow>::max();
        const Cost MAX_COST = ~0U >> 1;
        const Flow MAX_FLOW = ~0U >> 1;
        for (;;) {
            vector<Cost> dist(n, MAX_COST);
            vector<Flow> am(n, 0);
            vector<Edge*> prev(n);
            vector<bool> inQ(n, false);
            queue<int> que;

            dist[vs] = 0;
            am[vs] = MAX_FLOW;
            que.push(vs);
            inQ[vs] = true;

```

```

while (!que.empty()) {
    int u = que.front();
    Cost c = dist[u];
    que.pop();
    inQ[u] = false;
    for (Edge*e = E[u]; e; e = e->next)
        if (e->f > 0) {
            Cost nc = c + e->c;
            if (nc < dist[e->t]) {
                dist[e->t] = nc;
                prev[e->t] = e;
                am[e->t] = min(am[u], e->f);
                if (!inQ[e->t]) {
                    que.push(e->t);
                    inQ[e->t] = true;
                }
            }
        }
}

if (dist[vt] == MAX_COST)
    break;

Flow by = am[vt];
int u = vt;
flow += by;
cost += by * dist[vt];
while (u != vs) {
    Edge*e = prev[u];
    e->f -= by;
    e->rev->f += by;
    u = e->rev->t;
}

return make_pair(flow, cost);
};

int main() {
    return 0;
}

```

### 1.3.5 最小费用最大流 (fast)

```

const int MAXN = 1000; //number of vertices
const int MAXM = 10000; //number of edges

```

```

const int MAXQ = 10000; //length of queue
const int INF = 2000000000; //max capacity

struct Tedge {
    int v, f, c, w, next;
};

Tedge edge[MAXM];
int first[MAXN], dist[MAXN], path[MAXN], heap[MAXN], pos[MAXN], phi[MAXN];
int Q[MAXQ];
bool inq[MAXN];
int N, M, S, T, cnt;

inline void add_edge(int u, int v, int c, int w) {
    edge[M].v = v; edge[M].f = 0; edge[M].c = c; edge[M].w = w; edge[M].next = first[u]; first[u] = M;
    edge[M].v = u; edge[M].f = 0; edge[M].c = 0; edge[M].w = -w; edge[M].next = first[v]; first[v] = M;
}

inline void moveup(int i) {
    int key = heap[i];
    for (int j; (j = i >> 1) > 0 && dist[heap[j]] > dist[key]; i = j) heap[i] = heap[j], pos[heap[i]] = j;
    heap[i] = key; pos[key] = i;
}

inline void movedown(int i) {
    int key = heap[i];
    for (int j; (j = i << 1) <= cnt; i = j) {
        if (j < cnt && dist[heap[j + 1]] < dist[heap[j]]) ++j;
        if (dist[key] <= dist[heap[j]]) break;
        heap[i] = heap[j]; pos[heap[i]] = i;
    }
    heap[i] = key; pos[key] = i;
}

bool find_path1() {
    for (int i = 0; i < N; ++i) dist[i] = INF, inq[i] = 0;
    dist[S] = 0; inq[S] = 1; Q[0] = S;
    for (int h = 0, t = 1; h != t; ) {
        int u = Q[h++];
        if (h == MAXQ) h = 0;
        for (int i = first[u]; i != -1; i = edge[i].next) {
            int v = edge[i].v;
            if (edge[i].f < edge[i].c && dist[u] + edge[i].w < dist[v]) {
                dist[v] = dist[u] + edge[i].w; path[v] = i;
                if (!inq[v]) {
                    inq[v] = 1;
                    if (h != t && dist[v] <= dist[Q[h]]) {
                        if ((--h) < 0) h = MAXQ - 1;
                    }
                }
            }
        }
    }
}

```

```

        Q[h] = v;
    }
    else {
        Q[t++] = v;
        if (t == MAXQ) t = 0;
    }
}
}
}
inq[u] = 0;
}
return dist[T] < INF;
}

bool find_path2() {
    for (int i = 0; i < N; ++i) pos[i] = -1, dist[i] = INF;
    cnt = 1; heap[1] = S; dist[S] = 0;
    while (cnt) {
        int u = heap[1]; heap[1] = heap[cnt--]; movedown(1);
        for (int i = first[u]; i != -1; i = edge[i].next) {
            int v = edge[i].v;
            if (edge[i].f < edge[i].c && dist[u] + edge[i].w - phi[u] + phi[v] < dist[v]) {
                dist[v] = dist[u] + edge[i].w - phi[u] + phi[v]; path[v] = i;
                if (pos[v] == -1) pos[v] = ++cnt, heap[cnt] = v;
                moveup(pos[v]);
            }
        }
    }
    return dist[T] < INF;
}

int Mincost() {
    int ret = 0;
    memset(phi, 0, sizeof(phi));
    bool first = 1;
    while (1) {
        if (first) {
            if (!find_path1()) break;
            first = 0;
        }
        else if (!find_path2()) break;
        int delta = INF;
        for (int u = T, i = path[u]; u != S; u = edge[i ^ 1].v, i = path[u]) delta = min(delta, edge[i].f);
        for (int u = T, i = path[u]; u != S; u = edge[i ^ 1].v, i = path[u]) edge[i].f += delta, edge[i].c -= delta;
        ret += delta * (dist[T] - phi[T]);
        for (int i = 0; i < N; ++i) phi[i] -= dist[i];
    }
    return ret;
}

```

```
}
```

### 1.3.6 混合图欧拉回路

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<ctime>
using namespace std;

const int INF=0x7FFFFFFF;
const int maxn=5000;
int n,m,num,s,t,all,correct,flow;
int a[maxn][3],adj[maxn],next[maxn],vt[maxn],h[maxn];
int vh[maxn],st[maxn],f[maxn],sum[maxn],sign[maxn][2],fa[maxn];

int maxflow(int x, int ff) {
    if (x==t) return ff;
    int minh=t-s+2;
    for (int i=1; i<=sum[x]; i++){
        if (f[st[x]]>0){
            //因为欧拉回路中两点之间可能有多条连边，用邻接矩阵可能会出现问題
            if (h[x]==h[adj[st[x]]]+1){
                int k=maxflow(adj[st[x]],min(ff,f[st[x]]));
                if (k>0){
                    f[st[x]]-=k;
                    f[st[x]^1]+=k;
                    return k;
                }
            }
            minh=min(minh,h[adj[st[x]]]+1);
            if (h[s]>t-s+1) return 0;
        }
        st[x]=next[st[x]];
        if (st[x]==0) st[x]=vt[x];
    }
    --vh[h[x]];
    if (vh[h[x]]==0) h[s]=t-s+2;
    h[x]=minh;
    ++vh[h[x]];
    return 0;
}

void connect(int x,int y) {
    ++all;
    ++sum[x];
    adj[all]=y;
```

```

    next[all]=vt[x];
    vt[x]=all;
}

int ask(int x) {
    if (fa[x]==x) return x;
    fa[x]=ask(fa[x]);
    return fa[x];
}

void solve() {
    for (int i=2; i<=n; i++)
        if (ask(i)!=ask(i-1)){
            printf("impossible\n");
            return;
        }
    //并查集判断图的联通性
    s=0; t=n+1; correct=0; flow=0;
    for (int i=1; i<=n; i++)
        if ((abs(sign[i][0]-sign[i][1]))%2==1){
            printf("impossible\n");
            return;
        } else if (sign[i][0]>sign[i][1]){
            connect(s,i);
            f[all]=(sign[i][0]-sign[i][1])/2;
            connect(i,s);
            //将入度大于出度的点连到起点s
            correct+=(sign[i][0]-sign[i][1])/2;
        } else if (sign[i][1]>sign[i][0]){
            connect(i,t);
            f[all]=(sign[i][1]-sign[i][0])/2;
            connect(t,i);
            //将出度大于入度的点连接到汇点t
        }
    for (int i=s; i<=t; i++) st[i]=vt[i];
    memset(h,0,sizeof(h));
    memset(vh,0,sizeof(vh));
    vh[0]=t-s+1;
    while (h[s]<=t-s+1) flow+=maxflow(s,INF);
    if (flow==correct) printf("possible\n"); else printf("impossible\n");
    //满流时成立
}

void prepare() {
    all=1;
    memset(sum,0,sizeof(sum));
    memset(vt,0,sizeof(vt));
    memset(sign,0,sizeof(sign));
}

```

```

    memset(f,0,sizeof(f));
    for (int i=1; i<=n; i++) fa[i]=i;
}

void init() {
    scanf("%d",&num);
    while (num--){
        scanf("%d%d",&n,&m);
        prepare();
        for (int i=1; i<=m; i++){
            scanf("%d%d%d",&a[i][0],&a[i][1],&a[i][2]);
            if (a[i][2]==0){
                if ((rand()%2)==0) swap(a[i][0],a[i][1]);
                //随机双向边的方向
                connect(a[i][0],a[i][1]);
                f[all]=1;
                //构图时只连接双向边，流的方向为随机后的方向
                connect(a[i][1],a[i][0]);
            }
            ++sign[a[i][0]][0]; ++sign[a[i][1]][1];
            int x=ask(a[i][0]), y=ask(a[i][1]);
            fa[x]=fa[y];
        }
        solve();
    }
}

int main() {
    srand(time(0));
    init();
    return 0;
}

```

### 1.3.7 stoer wagner最小割集

```

const int MAXN = 50 + 10; //number of vertices
const int MAXM = 500; //number of MinCut edges
const int INF = 1000000000; //max capacity

int map[MAXN][MAXN], a[MAXN][MAXN], idx[MAXN][MAXN]; //map, tmp map, idx of edge
int root[MAXN], q[MAXN], w[MAXN], pre[MAXN];
int list[MAXM]; //MinCut Edges
bool used[MAXN];
int N, M;

int mincut(int n) {
    memset(used, 0, sizeof(used));
    memset(w, 0, sizeof(w));

```



```

int last, cnt = 0;
for (int k, i = 0; i != n; i = k) {
    last = i; used[i] = 1; k = n;
    for (int j = 0; j < n; ++j) {
        if (used[j]) continue;
        w[j] += a[q[i]][q[j]]; pre[j] = i;
        if (w[j] > w[k]) k = j;
    }
}
return last;
}

int find(int x) {
    if (root[x] == x) return x;
    else return root[x] = find(root[x]);
}

int stoer_wagner() {
    memcpy(a, map, sizeof(map));
    for (int i = 0; i < N; ++i) q[i] = root[i] = i;
    int ret = INF;
    for (int i = 0; i < N - 1; ++i) {
        int t = mincut(N - i);
        ret = min(ret, w[t]);
        int s = pre[t];
        for (int j = 0; j < N - i; ++j)
            if (j != s && j != t) a[q[t]][q[j]] = (a[q[j]][q[t]] += a[q[j]][q[s]]);
        root[find(q[s])] = find(q[t]); q[s] = q[N - i - 1];
    }
    return ret;
}

void cal(int ans) {
    memcpy(a, map, sizeof(map));
    for (int i = 0; i < N; ++i) q[i] = root[i] = i;
    int t;
    for (int i = 0; i < N - 2; ++i) {
        t = mincut(N - i - 1);
        if (w[t] == ans) break;
        int s = pre[t];
        for (int j = 0; j < N - i - 1; ++j)
            if (j != s && j != t) a[q[t]][q[j]] = (a[q[j]][q[t]] += a[q[j]][q[s]]);
        root[find(q[s])] = find(q[t]); q[s] = q[N - i - 2];
    }
    t = find(q[t]);
    M = 0; //number of MinCut edges
    for (int i = 0; i < N; ++i)
        if (find(root[i]) == t)

```

```

        for (int j = 0; j < N; ++j)
            if (find(root[j]) != t && idx[i][j]) list[M++] = idx[i][j];
    }

```

### 1.3.8 最小割树

```

#include <cstdio>
#include <cstring>
#include <algorithm>

```

```

using namespace std;

```

```

const int maxn = 200 + 10;
const int maxm = maxn * maxn;
const int INF = 1000000000;

```

```

struct Tedge {
    int v, f, c, next;
};

```

```

Tedge edge[maxn];
int first[maxn], level[maxn], nedge[maxn], pedge[maxn], prev[maxn], queue[maxn], par[maxn], fl[maxn];
int a[maxn][maxn], cut[maxn][maxn];
int n, m, S, T;

```

```

inline void add_edge(int u, int v, int c1, int c2 = 0) {
    edge[m].v = v; edge[m].f = 0; edge[m].c = c1; edge[m].next = first[u]; first[u] = m++;
    edge[m].v = u; edge[m].f = 0; edge[m].c = c2; edge[m].next = first[v]; first[v] = m++;
}

```

```

bool newphase() {
    for (int i = 0; i < n; ++i) level[i] = n, nedge[i] = first[i];
    queue[0] = S; level[S] = 0;
    for (int h = 0, t = 1; h < t; ++h) {
        int u = queue[h];
        for (int i = first[u]; i != -1; i = edge[i].next)
            if (edge[i].f < edge[i].c && level[edge[i].v] == n) {
                level[edge[i].v] = level[u] + 1;
                if (edge[i].v == T) return 1;
                queue[t++] = edge[i].v;
            }
    }
    return 0;
}

```

```

bool find_path(int u) {
    for (int i = nedge[u]; i != -1; i = edge[i].next)

```

```

        if (edge[i].f < edge[i].c && level[edge[i].v] == level[u] + 1)
            if (edge[i].v == T || find_path(edge[i].v)) {
                pedge[edge[i].v] = nedge[u] = i;
                return 1;
            }
        nedge[u] = -1;
        return 0;
    }

int Dinic() {
    for (int i = 0; i < m; ++i) edge[i].f = 0;
    int ret = 0;
    while (newphase())
        while (find_path(S)) {
            int delta = INF;
            for (int u = T, i = pedge[u]; u != S; u = edge[i ^ 1].v, i = pedge[u])
                delta = min(delta, edge[i].c - edge[i].f);
            for (int u = T, i = pedge[u]; u != S; u = edge[i ^ 1].v, i = pedge[u])
                edge[i].f += delta, edge[i ^ 1].f -= delta;
            ret += delta;
        }
    return ret;
}

int main() {
    int N;
    scanf("%d", &N);
    for (int tst = 1; tst <= N; ++tst) {
        m = 0;
        memset(first, -1, sizeof(first));
        scanf("%d", &n);
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j) {
                scanf("%d", &a[i][j]);
                if (i < j && a[i][j]) add_edge(i, j, a[i][j], a[i][j]);
            }
        memset(cut, 0, sizeof(cut));
        memset(par, 0, sizeof(par));
        for (S = 1; S < n; ++S) {
            T = par[S];
            fl[S] = cut[S][T] = cut[T][S] = Dinic();
            for (int i = 1; i < n; ++i)
                if (i != S && level[i] != n && par[i] == T) par[i] = S;
            if (level[par[T]] != n) {
                par[S] = par[T];
                par[T] = S;
                fl[S] = fl[T];
                fl[T] = cut[S][T];
            }
        }
    }
}

```

```

    }
    for (int i = 0; i < S; ++i)
        if (i != T) cut[S][i] = cut[i][S] = min(cut[S][T], cut[T][i]);
}
// (i, par[i]) of value fl[i][par[i]] are the edges of GH cut tree
printf("Case #d:\n", tst);
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        if (j) printf(" ");
        printf("%d", cut[i][j]);
    }
    printf("\n");
}
}
return 0;
}

```

## 1.4 匹配

### 1.4.1 匈牙利算法 $O(V \cdot E)$

```

const int N = 100 + 10;

int n, m;
vector<int> E[N];
bool visit[N];
int v[N];

int find(int x) {
    int y;
    rep(i, E[x].size()){
        y = E[x][i];
        if (visit[y]) continue;
        visit[y] = true;
        if (v[y] == -1 || find(v[y])){
            v[y] = x;
            return true;
        }
    }
    return false;
}

int max_match() {
    memset(v, -1, sizeof(v));
    int res = 0;
    rep(i, n){
        Cls(visit);
        if (find(i)) res++;
    }
}

```

```

        return res;
    }

    int main(){
        while(~scanf("%d%d", &n, &m)){
            rep(i, n) E[i].clear();
            int x, y;
            rep(i, m){
                scanf("%d%d", &x, &y);
                x--, y--;
                E[x].PB(y);
            }
            printf("%d\n", max_match());
        }
        return 0;
    }

```

#### 1.4.2 二分图匹配 $O(\sqrt{V} \cdot E)$

```

const int MAXN = 1000; //number of vertices
const int MAXE = 10000; //number of edges

struct Tedge {
    int v, next;
};

Tedge edge[MAXE];
int first[MAXN], px[MAXN], py[MAXN], dx[MAXN], dy[MAXN], q[MAXN];
bool used[MAXN];
int N, E, len;

void init() {
    memset(first, -1, sizeof(first));
    E = 0;
}

inline void add_edge(int u, int v) {
    edge[E].v = v; edge[E].next = first[u]; first[u] = E++;
}

bool search(int u) {
    if (dx[u] > len) return 0;
    used[u] = 1;
    for (int i = first[u]; i != -1; i = edge[i].next) {
        int v = edge[i].v;
        if ((py[v] == -1 || !used[py[v]]) && dx[u] + 1 == dy[v]) {
            int tx = px[u], ty = py[v];
            px[u] = v; py[v] = u;

```

```

        if (ty == -1 || search(ty)) return 1;
        px[u] = tx; py[v] = ty;
    }
}
return 0;
}

void hopcroft() {
    memset(px, -1, sizeof(px));
    memset(py, -1, sizeof(py));
    while (1) {
        memset(dx, 0, sizeof(dx));
        memset(dy, 0, sizeof(dy));
        int t = len = 0;
        for (int i = 0; i < N; ++i)
            if (px[i] == -1) q[t++] = i, dx[i] = 1;
        for (int h = 0; h < t; ++h) {
            int u = q[h];
            for (int i = first[u]; i != -1; i = edge[i].next) {
                int v = edge[i].v;
                if (!dy[v]) {
                    dy[v] = dx[u] + 1;
                    if (py[v] != -1) q[t++] = py[v], dx[py[v]] = dy[v] + 1;
                    else len = max(len, dy[v]);
                }
            }
        }
        if (!len) break;
        memset(used, 0, sizeof(used));
        for (int i = 0; i < N; ++i)
            if (px[i] == -1) search(i);
    }
}
}

```

### 1.4.3 二分图最大权匹配

```

const int N = 105, inf = 0x3F3F3F3F;

int n;
int graph[N][N];
int match[N], slack[N], lx[N], ly[N];
bool vx[N], vy[N];

bool find(int x) {
    vx[x] = true;
    rep(y, n){
        if (vy[y]) continue;
        if (lx[x] + ly[y] == graph[x][y]){

```

```

        vy[y] = true;
        if (match[y] == -1 || find(match[y])){
            match[y] = x;
            return true;
        }
    }
    else slack[y] = min(slack[y], lx[x] + ly[y] - graph[x][y]);
}
return false;
}

int max_match(int n) {
    rep(i, n) {
        lx[i] = *max_element(graph[i], graph[i] + n);
        ly[i] = 0;
    }
    memset(match, -1, sizeof(match));
    rep(x, n){
        memset(slack, -1, sizeof(slack));
        while(true){
            Cls(vx);
            Cls(vy);
            if (find(x)) break;
            int sub = inf;
            rep(i, n) if (!vy[i]) sub = min(sub, slack[i]);
            rep(i, n) if (vx[i]) lx[i] -= sub;
            rep(i, n) {
                if (vy[i]) ly[i] += sub;
                else slack[i] -= sub;
            }
        }
    }
    int res = 0;
    rep(i, n) res += graph[match[i]][i];
    return res;
}

int min_match(int n) {
    rep(i, n) rep(j, n) graph[i][j] *= -1;
    return -max_match(n);
}

int main() {
    while(~scanf("%d", &n)){
        rep(i, n) rep(j, n) scanf("%d", &graph[i][j]);
        printf("%d\n", max_match(n));
    }
    return 0;
}

```

```
}
```

#### 1.4.4 带花树

```
const int N = 50, M = 150;
```

```
int n, m;  
int x[M], y[M]; int psz;  
int next[N], match[N], v[N];  
int f[N], rank[N];  
int ans[M];
```

```
VI E[N];  
deque<int> Q;
```

```
inline int find(int p) {return f[p]<0?p:f[p]=find(f[p]);}
```

```
void join(int x, int y){  
    x = find(x); y = find(y);  
    if (x != y) f[x] = y;  
}
```

```
int lca(int x, int y){  
    static int v[N];  
    static int stamp = 0;  
    ++stamp;  
    for (;;) {  
        if (x >= 0) {  
            x = find(x);  
            if (v[x] == stamp) return x;  
            v[x] = stamp;  
            if (match[x] >= 0) x = next[match[x]];  
            else x = -1;  
        }  
        swap(x, y);  
    }  
}
```

```
void group(int a, int p){  
    while (a != p) {  
        int b = match[a], c = next[b];  
        if (find(c) != p) next[c] = b;  
        if (v[b] == 2) Q.PB(b), v[b] = 1;  
        if (v[c] == 2) Q.PB(c), v[c] = 1;  
        join(a, b); join(b, c);  
        a = c;  
    }  
}
```



```

void aug(int s){
    Cls(v,0);
    Cls(next,-1);
    Cls(f,-1);
    Q.clear();
    Q.PB(s);
    v[s] = 1;
    while(!Q.empty() && match[s] == -1){
        int x = Q.front(); Q.pop_front();
        rep(i, E[x].size()){
            int y = E[x][i];
            if (match[x] == y || find(x) == find(y) || v[y] == 2) continue;
            if (v[y] == 1) {
                int p = lca(x, y);
                if (find(x) != p) next[x] = y;
                if (find(y) != p) next[y] = x;
                group(x, p);
                group(y, p);
            } else if (match[y] == -1) {
                next[y] = x;
                while (~y) {
                    int z = next[y];
                    int p = match[z];
                    match[y] = z; match[z] = y;
                    y = p;
                }
                break;
            } else {
                next[y] = x;
                Q.PB(match[y]);
                v[match[y]] = 1;
                v[y] = 2;
            }
        }
    }
}

int work(int k){
    psz = 0;
    rep(i, n) E[i].clear();
    rep(i, m){
        if (x[i] == x[k]) continue;
        if (x[i] == y[k]) continue;
        if (y[i] == x[k]) continue;
        if (y[i] == y[k]) continue;
        E[x[i]].PB(y[i]);
        E[y[i]].PB(x[i]);
    }
}

```

```

    }
    Cls(match,-1);
    rep(i,n) if (match[i]==-1) aug(i);
    int c = 0;
    rep(i,n) if (match[i]!=-1) c++;
    return c/2;
}

int main(){
    while(~scanf("%d%d", &n, &m)){
        rep(i,m) {
            scanf("%d%d",x+i,y+i);
            x[i]--,y[i]--;
        }
        x[m]=y[m]=n;
        int s=work(m);
        int tot=0;
        rep(i,m) if (work(i)!=s-1) ans[tot++]=i+1;
        printf("%d\n", tot);
        if (tot) rep(i,tot) printf(i==tot-1?"%d\n":"%d ",ans[i]);
        else puts("");
    }
}

```

## 1.5 图

### 1.5.1 哈密尔顿回路

求一个图的哈密尔顿回路是一个 NP 问题，只能用搜索解决。当定点数  $n$  比较小的时候，可以用状态压缩 Dp 解决。当  $n$  比较大的时候，只能用搜索解决。但是，关于哈密尔顿回路，有一个性质：

Ore 性质：对所有不邻接的不同顶点对  $x$  和  $y$ ，有

$$\deg(x) + \deg(y) \geq n$$

那么这个图一定存在哈密尔顿回路，且可以用一下方法求回路，时间复杂度接近  $O(n^2)$

1) 从任意一个顶点开始，在它的任意一端邻接一个顶点，构造一条越来越长的路径，直到不能再加长为止。设路径为

$$\gamma : y_1 - y_2 - \cdots - y_m$$

2) 检查  $y_1$  和  $y_m$  是否邻接。

- . a) 如果  $y_1$  和  $y_m$  不邻接，则转到 3，否则， $y_1$  和  $y_m$  是邻接的，转到 b。
- . b) 如果  $m = n$ ，则停止构造并输出哈密尔顿回路  $y_1 - y_2 - \cdots - y_m - y_1$ ，否则，转到 c。
- . c) 找出一个不在  $\gamma$  上的顶点  $z$  和在  $\gamma$  上的顶点  $y_k$ ，满足  $z$  和  $y_k$  是邻接的，将  $\gamma$  用下面的长度为  $m + 1$  的路径来替代

$$z - y_k - \cdots - y_m - y_1 - \cdots - y_{k-1}$$

. 转到 2)

3) 找出一个顶点  $y_k (1 < k < m)$ ，满足  $y_1$  和  $y_k$  是邻接的，且  $y_{k-1}$  和  $y_m$  也是邻接的，将  $\gamma$  用下面的路径

来替代

$$y_1 - \cdots - y_{k-1} - y_m - \cdots - y_k$$

. 转到2)

### 1.5.2 割点和桥

```
const int N = 1111, M = 1111111;
```

```
int n, m;
int root;
int low[N], dep[N];
bool cut[N], bri[N];
vector<int> E[N];
vector<int> id[N];
PII edge[N];
```

```
void dfs(int x, int f, int d){
    int e = 0, deg = 0;
    low[x] = dep[x] = d;
    rep(i, E[x].size()){
        int y = E[x][i];
        if (low[y] == -1){
            deg++;
            dfs(y, x, d + 1);
            low[x] = min(low[x], low[y]);
            if (low[y] > dep[x]) bri[id[x][i]] = true;
            cut[x] |= (x == root && deg > 1 || x != root && low[y] >= dep[x]);
        }
        else if (y != f || e){
            low[x] = min(low[x], dep[y]);
        }
        else e = 1;
    }
}
```

```
int main(){
    while(~scanf("%d%d",&n,&m)){
        rep(i, n) E[i].clear(), low[i] = dep[i] = -1, cut[i] = false;
        rep(i, n) id[i].clear();
        int x, y;
        rep(i, m){
            scanf("%d%d", &x, &y);
            x--, y--;
            bri[i] = false;
            edge[i] = MP(x, y);
            E[x].PB(y), id[x].PB(i);
            E[y].PB(x), id[y].PB(i);
        }
    }
}
```

```

    }
    dfs(root = 0, -1, 0);
}
return 0;
}

```

### 1.5.3 有向图割点

```

const int MAXN = 5000 + 10; //number of vertices
const int MAXM = 200000 + 10; //number of edges

struct Tedge {
    int v, next;
};

Tedge edge[MAXN], back[MAXN]; //back is opposite to edge
bool ontree[MAXN];
int first1[MAXN], first2[MAXN], id[MAXN], low[MAXN], stack[MAXN];
bool critical[MAXN]; //1 - the node is CutVertex, 0 - not
int N, M, cnt;

void DFS(int u) {
    id[u] = cnt; stack[cnt++] = u; low[u] = u;
    for (int i = first1[u]; i != -1; i = edge[i].next)
        if (id[edge[i].v] == -1) {
            ontree[i] = 1;
            DFS(edge[i].v);
        }
}

void update(int u) {
    for (int i = first1[u]; i != -1; i = edge[i].next)
        if (ontree[i] && id[low[u]] < id[low[edge[i].v]]) {
            low[edge[i].v] = low[u];
            update(edge[i].v);
        }
}

void CV() {
    cnt = 0;
    memset(id, -1, sizeof(id));
    memset(ontree, 0, sizeof(ontree));
    DFS(0);
    memset(critical, 0, sizeof(critical));
    critical[0] = 1;
    for (int i = cnt - 1; i >= 0; --i) {
        int u = stack[i];
        for (int j = first1[u]; j != -1; j = edge[j].next)

```

```

        if (ontree[j] && low[edge[j].v] == u) {
            critical[u] = 1;
            break;
        }
        for (int j = first2[u]; j != -1; j = back[j].next)
            if (id[low[back[j].v]] < id[low[u]]) low[u] = low[back[j].v];
        update(u);
    }
}

```

#### 1.5.4 树链剖分

```

const int N=50005,M=1<<16;

int n,m,q,tot;
int v[N];
int t[M*2];
VI E[N];
int fa[N],dep[N],son[N],sz[N];
int id[N],top[N];

void dfs(int x){
    sz[x]=1,son[x]=0;
    rep(i,E[x].size()){
        int y=E[x][i];
        if (y==fa[x]) continue;
        dep[y]=dep[x]+1;
        fa[y]=x;
        dfs(y);
        sz[x]+=sz[y];
        if (sz[y]>sz[son[x]]) son[x]=y;
    }
}

void dfs(int x,int p){
    id[x]=++tot,top[x]=p;
    if (son[x]) dfs(son[x],p);
    rep(i,E[x].size()){
        int y=E[x][i];
        if (y==fa[x]||y==son[x]) continue;
        dfs(y,y);
    }
}

int ask(int x){
    x=id[x];
    int ret=0;

```

```

    for(x+=M;x>=1) ret+=t[x];
    return ret;
}

void insert(int l,int r,int x){
    for(l+=M-1,r+=M+1;l^r^1;l>>=1,r>>=1){
        if (~l&1) t[l^1]+=x;
        if ( r&1) t[r^1]+=x;
    }
}

void add(int x,int y,int k){
    while(top[x]!=top[y]){
        if (dep[top[x]]<dep[top[y]]) swap(x,y);
        insert(id[top[x]],id[x],k);
        x=fa[top[x]];
    }
    if (dep[x]<dep[y]) swap(x,y);
    insert(id[y],id[x],k);
}

int main(){
    while(~scanf("%d%d%d",&n,&m,&q)){
        rep(i,n) scanf("%d",&v[i+1]);
        rep(i,n) E[i+1].clear();
        rep(i,m){
            int x,y;
            scanf("%d%d",&x,&y);
            E[x].PB(y);
            E[y].PB(x);
        }
        fa[1]=dep[1]=1;
        sz[0]=0,tot=0;
        dfs(1);
        dfs(1,1);
        Cls(t);
        fab(i,1,n) t[id[i]+M]=v[i];
        char ch;
        int x,y,k;
        rep(i,q){
            while((ch=getchar())&&ch!='D'&&ch!='Q'&&ch!='I');
            if (ch=='Q'){
                scanf("%d",&x);
                printf("%d\n",ask(x));
            }
            else{
                scanf("%d%d%d",&x,&y,&k);
                add(x,y,(ch=='I')?k:-k);
            }
        }
    }
}

```

```

    }
}
return 0;
}

```

### 1.5.5 最大团

```

const int MAXN = 100; //number of vertices

int a[MAXN][MAXN];
int f[MAXN];
int N, ans;

bool DFS(int q[], int t, int cnt) {
    if (t == 0) {
        if (cnt > ans) {
            ans = cnt;
            return 1;
        }
        return 0;
    }

    int tq[MAXN];
    for (int i = 0; i < t; ++i) if (cnt + t - i > ans){
        if (f[q[i]] + cnt <= ans) return 0;
        int k = 0;
        for (int j = i + 1; j < t; ++j)
            if (a[q[i]][q[j]]) tq[k++] = q[j];
        if (DFS(tq, k, cnt + 1)) return 1;
    }
    return 0;
}

void MaxClique() {
    ans = 0;
    int q[MAXN];
    for (int i = N - 1; i >= 0; --i) {
        int t = 0;
        for (int j = i + 1; j < N; ++j) if (a[i][j]) q[t++] = j;
        DFS(q, t, 1);
        f[i] = ans;
    }
}

```

### 1.5.6 图平面化

```

// vertices numbered from 1 to N
// No self-loops and no duplicate edges

```

```

typedef pair<int, int> T;
const int maxn = 10000 + 10;

struct node {
    int dep, fa, infc, used, vst, dfi, ec, lowp, bflag, flag, lowpoint;
};

int n, m, indee, p1, p2, p, ps;
int lk[maxn * 3][2], child[maxn * 3][3], bedg[maxn * 3][2], sdlist[maxn * 6][3],
    buk[maxn * 6][2], exf[maxn * 3][2], proots[maxn * 3][3], stk[maxn * 3][2], infap[maxn * 3];
int w1[maxn], w2[maxn], que[maxn];
node dot[maxn];

void init(T * ts) {
    ps = 0;
    for (int i = 1; i <= n; ++i) w1[i] = i;
    p1 = n;
    for (int i = 0; i < m; ++i) {
        int k1 = ts[i].first, k2 = ts[i].second;
        lk[++p1][0] = k2; lk[p1][1] = 0;
        lk[w1[k1]][1] = p1;
        w1[k1] = p1;
        lk[++p1][0] = k1; lk[p1][1] = 0;
        lk[w1[k2]][1] = p1;
        w1[k2] = p1;
    }
    for (int i = 1; i <= n; ++i) que[i] = i;
}

int deep(int a) {
    dot[a].used = 1; dot[a].dfi = ++indee;
    int t = lk[a][1];
    while (t != 0) {
        int tmp = lk[t][0];
        if (!dot[tmp].used) {
            dot[tmp].fa = a; dot[tmp].dep = dot[a].dep + 1; dot[tmp].ec = dot[a].dep; dot[tmp].lowp = dot[a].lowp;
            child[++p1][0] = tmp; child[p1][1] = 0;
            child[w1[a]][1] = p1;
            w1[a] = p1;
            int s = deep(tmp);
            if (s < dot[a].ec) dot[a].ec = s;
        }
        else if (dot[a].fa != tmp) {
            if (dot[a].lowp > dot[tmp].dep) dot[a].lowp = dot[tmp].dep;
            if (dot[a].dfi > dot[tmp].dfi) {
                bedg[++p2][0] = a; bedg[p2][1] = 0;
                bedg[w2[tmp]][1] = p2;
            }
        }
        t = lk[t][1];
    }
}

```



```

        w2[tmp] = p2;
    }
}
t = lk[t][1];
}
if (dot[a].ec > dot[a].lowp) dot[a].ec = dot[a].lowp;
return dot[a].ec;
}

void sortvtx() {
    for (int i = 1; i <= n; ++i) w1[i] = i;
    p1 = n; p2 = 0;
    for (int i = 1; i <= n; ++i) {
        buk[++p1][0] = i; buk[p1][1] = 0;
        buk[w1[dot[i].dfi]][1] = p1;
        w1[dot[i].dfi] = p1;
    }
    for (int i = n; i > 0; --i) {
        int tmp = buk[i][1];
        while (tmp != 0) {
            que[++p2] = buk[tmp][0];
            tmp = buk[tmp][1];
        }
    }
}

void getsdlist() {
    memset(buk, 0, sizeof(buk));
    for (int i = 1; i <= n; ++i) {
        w1[i] = w2[i] = i;
        buk[i][1] = 0;
    }
    p1 = p2 = n;
    for (int i = 1; i <= n; ++i) {
        buk[++p1][0] = i; buk[p1][1] = 0;
        buk[w1[dot[i].ec]][1] = w1[dot[i].ec] = p1;
    }
    for (int i = 1; i <= n; ++i) {
        int tmp = buk[i][1];
        while (tmp != 0) {
            int fa = dot[buk[tmp][0]].fa;
            sdlist[++p2][0] = i; sdlist[p2][1] = 0;
            sdlist[w2[fa]][1] = dot[buk[tmp][0]].infc = p2;
            sdlist[p2][2] = w2[fa]; w2[fa] = p2;
            tmp = buk[tmp][1];
        }
    }
}
}

```

```

void getnextvtx(int v, int v1, int &m, int &m1) {
    m = exf[v][v1 ^ 1];
    if (exf[m][0] == v) m1 = 0;
    else m1 = 1;
}

void addwei(int a) {
    int fa = dot[a - n].fa;
    ++p1;
    proots[p1][0] = a; proots[p1][1] = 0;
    proots[w1[fa]][1] = p1;
    proots[p1][2] = w1[fa]; w1[fa] = p1;
    infap[a] = p1;
}

void addsou(int a) {
    int fa = dot[a - n].fa;
    ++p1;
    proots[p1][0] = a; proots[p1][1] = proots[fa][1]; proots[p1][2] = fa;
    proots[fa][1] = p1;
    proots[proots[p1][1]][2] = p1;
    infap[a] = p1;
    if (w1[fa] == fa) w1[fa] = p1;
}

void walkup(int v, int w) {
    dot[w].bflag = v;
    int x = w, x1 = 1, y = w, y1 = 0;
    while (x != v) {
        if (dot[x].vst == v || dot[y].vst == v) break;
        dot[x].vst = v; dot[y].vst = v;
        int z1 = 0;
        if (x > n) z1 = x;
        if (y > n) z1 = y;
        if (z1 != 0) {
            int c = z1 - n, z = dot[c].fa;
            if (z != v) {
                if (dot[c].lowpoint < dot[v].dep) addwei(z1);
                else addsou(z1);
            }
            x = z; x1 = 1;
            y = z; y1 = 0;
        } else {
            getnextvtx(x, x1, x, x1);
            getnextvtx(y, y1, y, y1);
        }
    }
}

```

```

}

void getactivenext(int v, int v1, int &m, int &m1, int vt) {
    m = v; m1 = v1;
    getnextvtx(m, m1, m, m1);
    while (dot[m].bflag != vt && proots[m][1] == 0 && dot[m].ec >= dot[vt].dep && m != v) getnextv
}

void addstack(int a, int b) {
    stk[++ps][0] = a; stk[ps][1] = b;
}

void mergestack() {
    int t = stk[ps][0], t1 = stk[ps][1], k = stk[ps - 1][0], k1 = stk[ps - 1][1];
    ps -= 2;
    int s1, s = exf[t][1 ^ t1];
    if (exf[s][1] == t) s1 = 1;
    else s1 = 0;
    exf[k][k1] = s;
    exf[s][s1] = k;
    int tmp = dot[t - n].infc;
    sdlist[sdlist[tmp][2]][1] = sdlist[tmp][1]; sdlist[sdlist[tmp][1]][2] = sdlist[tmp][2];
    tmp = dot[t - n].fa;
    if (sdlist[tmp][1] == 0) dot[tmp].ec = dot[tmp].lowp;
    else dot[tmp].ec = min(dot[tmp].lowp, sdlist[sdlist[tmp][1]][0]);
    tmp = infap[t];
    int fa = dot[t - n].fa;
    proots[proots[tmp][2]][1] = proots[tmp][1];
    if (proots[tmp][1] != 0) proots[proots[tmp][1]][2] = proots[tmp][2];
    else w1[fa] = proots[tmp][2];
}

void embededg(int v, int v1, int w, int w1) {
    exf[v][v1] = w; exf[w][w1] = v;
}

void walkdown(int v) {
    ps = 0;
    int vt = dot[v - n].fa;
    for (int v2 = 0; v2 <= 1; ++v2) {
        int w, w1;
        getnextvtx(v, 1 ^ v2, w, w1);
        while (w != v) {
            if (dot[w].bflag == vt) {
                while (ps != 0) mergestack();
                embededg(v, v2, w, w1);
                dot[w].bflag = 0;
            }
        }
    }
}

```

```

        if (proots[w][1] != 0) {
            addstack(w, w1);
            int x, x1, y, y1, w2, w0 = proots[proots[w][1]][0];
            getactivenext(w0, 1, x, x1, vt);
            getactivenext(w0, 0, y, y1, vt);
            if (dot[x].ec >= dot[vt].dep) w = x, w1 = x1;
            else if (dot[y].ec >= dot[vt].dep) w = y, w1 = y1;
            else if (dot[x].bflag == vt || proots[x][1] != 0) w = x, w1 = x1;
            else w = y, w1 = y1;
            if (w == x) w2 = 0;
            else w2 = 1;
            addstack(w0, w2);
        }
        else if (w > n || dot[w].ec >= dot[vt].dep) getnextvtx(w, w1, w, w1);
        else {
            if (w <= n && dot[w].ec < dot[vt].dep && ps == 0) embededg(v, v2, w, w1);
            break;
        }
    }
    if (ps != 0) break;
}
}

bool chainvtx(int a) {
    for (int t = child[a][1]; t != 0; t = child[t][1]) {
        int tmp = child[t][0];
        exf[tmp][1] = tmp + n; exf[tmp][0] = tmp + n;
        exf[tmp + n][1] = tmp; exf[tmp + n][0] = tmp;
    }
    for (int t = bedg[a][1]; t != 0; t = bedg[t][1]) walkup(a, bedg[t][0]);
    for (int t = child[a][1]; t != 0; t = child[t][1]) walkdown(child[t][0] + n);
    for (int t = bedg[a][1]; t != 0; t = bedg[t][1]) if (dot[bedg[t][0]].bflag != 0) return false;
    return true;
}

bool judge(int N, int M, T * ts) {
    n = N;    m = M;
    if (n == 1) return true;
    if (m > 3 * n - 5) return false;
    init(ts);

    for (int i = 1; i <= n; ++i) {
        proots[i][1] = 0; proots[i + n][1] = 0;
        p = 0;
        child[i][1] = 0;
        buk[i][1] = 0; buk[i + n][1] = 0;
        sdlist[i][1] = 0; sdlist[i + n][1] = 0;
        dot[i].bflag = 0; dot[i + n].fflag = 0;
    }
}

```

```

}
for (int i = 1; i <= n; ++i) {
    w1[i] = i; w2[i] = i;
    child[i][1] = 0; bedg[i][1] = 0;
    dot[i].used = 0;
}
indee = 0; p1 = p2 = n;
for (int i = 1; i <= n; ++i) {
    if (!dot[i].used) {
        dot[i].dep = 1;
        deep(i);
    }
}
sortvtx();
getsdlist();
for (int i = 1; i <= n; ++i) {
    dot[i].lowpoint = dot[i].ec;
    dot[i].vst = 0; dot[i + n].vst = 0;
    proots[i][1] = 0;
    w1[i] = i;
}
p1 = n;
for (int i = 1; i <= n; ++i) if (!chainvtx(que[i])) return false;
return true;
}

T ts[maxn];
bool a[3001][3001];

int main() {
    int N, M;
    scanf("%d%d", &N, &M);
    int m = 0;
    for(int i = 0; i < M; i ++) {
        scanf("%d%d", &ts[i].first, &ts[i].second);
        ++ts[i].first; ++ts[i].second;
        if (ts[i].first == ts[i].second || a[ts[i].first][ts[i].second]) continue;
        a[ts[i].first][ts[i].second] = a[ts[i].second][ts[i].first] = 1;
        ts[m++] = ts[i];
    }
    M = m;
    if(judge(N, M, ts)) puts("YES");
    else puts("NO");

    return 0;
}

```

### 1.5.7 LCA offline

```
const int N = 1000, M = 10000; //number of vertices

int n, m, q;
vector<int> E[N];
vector<PII> Q[N];
int ans[N];
int f[N];
bool vis[N];

int find(int x){
    return (f[x] == x) ? f[x] : f[x] = find(f[x]);
}

void dfs(int x) {
    f[x] = x;
    int y;
    rep(i, E[x].size()){
        y = E[x][i];
        if (f[y] != -1) continue;
        dfs(y);
        f[y] = x;
    }
    vis[x] = true;
    int id;
    rep(i, Q[x].size()){
        y = Q[x][i].X, id = Q[x][i].Y;
        if (vis[y]) ans[id] = find(y);
    }
}

int main(){
    while(~scanf("%d", &n)){
        rep(i, n) E[i].clear(), Q[i].clear();
        rep(i, n) f[i] = -1, vis[i] = false;
        int x, y, c;
        m = n - 1;
        rep(i, m){
            scanf("%d%d", &x, &y);
            x--, y--;
            E[x].PB(y);
            E[y].PB(x);
        }
        scanf("%d", &q);
        rep(i, q){
            scanf("%d%d", &x, &y);
```

```

        x--, y--;
        Q[x].PB(MP(y, i));
        Q[y].PB(MP(x, i));
    }
    dfs(0);
    rep(i, q) printf("%d\n", ans[i] + 1);
}
return 0;
}

```

### 1.5.8 LCA online

```

const int N = 10000, M = 20;

int n, m, q;
int tot;
vector<int> E[N];
int dep[N];
int dist[N];
int anc[M][N];
int tin[N], tout[N];
bool vis[N];

void dfs(int x){
    vis[x] = true;
    tin[x] = tot++;
    int y;
    rep(i, E[x].size()){
        y = E[x][i];
        if (vis[y]) continue;
        dep[y] = dep[x] + 1;
        anc[0][y] = x;
        dfs(y);
    }
    tout[x] = tot++;
}

bool isanc(int x, int y){
    return tin[x] <= tin[y] && tout[y] <= tout[x];
}

int lca(int x, int y){
    if (isanc(x, y)) return x;
    fba(j, M - 1, 0){
        if (isanc(anc[j][x], y)) continue;
        x = anc[j][x];
    }
    return anc[0][x];
}

```

```

}

int main(){
    while(~scanf("%d%d", &n, &m)){
        rep(i, n) E[i].clear();
        rep(i, n) vis[i] = false;
        int x, y;
        rep(i, m){
            scanf("%d%d", &x, &y);
            x--, y--;
            E[x].PB(y);
            E[y].PB(x);
        }
        tot = 0;
        dep[0] = 1;
        rep(i, n) anc[i][0] = 0;
        dfs(0);
        REP(j, 1, M) rep(i, n) anc[j][i] = anc[j - 1][anc[j - 1][i]];
        scanf("%d", &q);
        while(q--){
            scanf("%d%d", &x, &y);
            x--, y--;
            printf("%d\n", lca(x, y) + 1);
        }
    }
    return 0;
}

```

## 2 数据结构

### 2.1 平衡树

#### 2.1.1 heap

```

#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;

typedef int T;

const int N = 10000 + 10;

T value[N];
int n;
int heap[N], pos[N], hn;

void heap_init() {

```



```

    hn = 0;
}

int heap_size() {
    return hn;
}

void heap_up(int x) {
    int p;
    while (x && value[heap[x]] < value[heap[p = (x - 1) >> 1]]) {
        swap(heap[x], heap[p]);
        swap(pos[heap[x]], pos[heap[p]]);
        x = p;
    }
}

void heap_down(int x) {
    int c;
    while ((c = (x << 1) + 1) < hn) {
        if (c + 1 < hn && value[heap[c + 1]] < value[heap[c]]) c++;
        if (value[heap[c]] < value[heap[x]]) {
            swap(heap[x], heap[c]);
            swap(pos[heap[x]], pos[heap[c]]);
            x = c;
        } else break;
    }
}

void heap_push(int i) {
    pos[heap[hn] = i] = hn;
    heap_up(hn++);
}

void heap_remove(int i) {
    int x = pos[i];
    pos[heap[x] = heap[--hn]] = x;
    heap_down(x);
}

int heap_top() {
    return heap[0];
}

int heap_pop() {
    int t;
    heap_remove(t = heap[0]);
    return t;
}

```

### 2.1.2 BST

\\*  
BST和Treap基本一样，Treap是用Max-heap作为平衡条件，BST用子树节点个数作为平衡条件  
一般情况下BST比Treap稍微平衡一点

```
\*
#include <cstdio>
#include <algorithm>
const int INF=0x7FFFFFFF;
const int kMaxN=1000001;

int n,total,head;
int key[kMaxN],size[kMaxN],le[kMaxN],ri[kMaxN];

void init(){
    scanf("%d\n",&n);
}

void RightRotate(int &head){
    int tmp=le[head];
    le[head]=ri[tmp];
    ri[tmp]=head;
    size[tmp]=size[head];
    size[head]=size[le[head]]+size[ri[head]]+1;
    head=tmp;
}

void LeftRotate(int &head){
    int tmp=ri[head];
    ri[head]=le[tmp];
    le[tmp]=head;
    size[tmp]=size[head];
    size[head]=size[le[head]]+size[ri[head]]+1;
    head=tmp;
}

void insert(int &head, int number){
    if (head==0){
        ++total;
        key[total]=number;
        size[total]=1;
        le[total]=ri[total]=0;
        head=total;
    } else{
        ++size[head];
        if (key[head]>number){
            insert(le[head],number);
            if (size[le[le[head]]]>size[ri[head]] || size[ri[le[head]]]>size[ri[head]])
```

```

        RightRotate(head);
    } else{
        insert(ri[head],number);
        if (size[le[ri[head]]]>size[le[head]] || size[ri[ri[head]]]>size[le[head]])
            LeftRotate(head);
    }
}
}

void del(int &head, int number){
    --size[head];
    if (key[head]>number) del(le[head],number); else
    if (key[head]<number) del(ri[head],number); else{
        if (size[head]==0){
            head=0;
            return;
        }
        if (size[le[head]]>size[ri[head]]){
            RightRotate(head);
            del(ri[head],number);
        } else{
            LeftRotate(head);
            del(le[head],number);
        }
    }
}

void solve(){
    char ch;
    int number;
    head=total=0;
    for (int i=0; i<n; i++){
        scanf("%c %d\n",&ch,&number);
        if (ch=='i'){
            insert(head,number);
        } else
        if (ch=='d'){
            del(head,number);
        }
    }
    printf("%d\n",size[head]);
}

int main(){
    init();
    solve();
    return 0;
}

```

### 2.1.3 treap

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <algorithm>
#include <ctype.h>
using namespace std;

typedef int T;

const int N = 1000010;

struct node {
    T d;
    int t, s;
    node *p, *l, *r;
} nodes[N], *next_alloc;

void init() {
    next_alloc = nodes;
}

node *node_new(T d) {
    node *x = next_alloc++;
    x->d = d;
    x->l = x->r = NULL;
    x->t = rand();
    x->s = 1;
    return x;
}

inline int size(node *x) {
    return x ? x->s : 0;
}

node *root(node *x) {
    while (x->p) x = x->p;
    return x;
}

void add_size(node *x, int d) {
    for ( ; x; x = x->p)
        x->s += d;
}

void rotate(node *x) {
    node *y = x->p;
```

```

node *g = y->p;

x->p = g;
if (g) (g->l == y ? g->l : g->r) = x;
y->p = x;

if (x == y->l) {
    y->l = x->r;
    if (x->r) x->r->p = y;
    x->r = y;
} else {
    y->r = x->l;
    if (x->l) x->l->p = y;
    x->l = y;
}

y->s = size(y->l) + size(y->r) + 1;
x->s = size(x->l) + size(x->r) + 1;
}

void adjust(node *x) {
    while (x->p && x->t < x->p->t)
        rotate(x);
}

/* return new root
 * duplicated elements are not allowed */
node *_insert(node *p, node *x) {
    if (!p)
        return x;

    while (1) {
        if (x->d < p->d) {
            if (!p->l) {
                add_size(x->p = p, 1);
                adjust(p->l = x);
                break;
            }
            p = p->l;
        } else if (x->d > p->d) {
            if (!p->r) {
                add_size(x->p = p, 1);
                adjust(p->r = x);
                break;
            }
            p = p->r;
        } else break;
    }
}

```

```

    return root(p);
}

/* return new root */
node *insert(node *p, T d) {
    return _insert(p, node_new(d));
}

node *kth(node *p, int k) {
    while (p) {
        if (k == size(p->l))
            return p;
        if (k < size(p->l))
            p = p->l;
        else
            k -= size(p->l) + 1, p = p->r;
    }
    return NULL;
}

/* # nodes with value < d */
int count(node *x, T d) {
    int res = 0;
    while (x) {
        if (d <= x->d)
            x = x->l;
        else
            res += size(x->l) + 1, x = x->r;
    }
    return res;
}

node *find(node *p, T d) {
    while (p) {
        if (d == p->d) return p;
        p = (d < p->d ? p->l : p->r);
    }
    return NULL;
}

node *lower_bound(node *x, T d) {
    node *y;
    if (!x)
        return NULL;
    if (d <= x->d && (y = lower_bound(x->l, d)))
        return y;
    if (d <= x->d)
        return x;
}

```

```

    return lower_bound(x->r, d);
}

node *upper_bound(node *x, T d) {
    node *y;
    if (!x)
        return NULL;
    if (d < x->d && (y = upper_bound(x->l, d)))
        return y;
    if (d < x->d)
        return x;
    return upper_bound(x->r, d);
}

node *next(node *x) {
    if (x->r) {
        x = x->r;
        while (x->l) x = x->l;
        return x;
    } else {
        while (x->p && x->p->r == x)
            x = x->p;
        return x->p;
    }
}

node *prev(node *x) {
    if (x->l) {
        x = x->l;
        while (x->r) x = x->r;
        return x;
    } else {
        while (x->p && x->p->l == x)
            x = x->p;
        return x->p;
    }
}

/* return new root */
node *erase(node *x) {
    while (x->l || x->r) {
        if (x->l && (!x->r || x->l->t < x->r->t))
            rotate(x->l);
        else
            rotate(x->r);
    }

    node *y = x->p;

```

```

    if (y) (y->l == x ? y->l : y->r) = NULL, add_size(y, -1);
    x->p = NULL;
    return y ? root(y) : NULL;
}

node *__merge(node *x, node *y) {
    node *l = y->l, *r = y->r;
    y->l = y->r = NULL, y->s = 1;

    if (r) x = __merge(x, r);
    x = _insert(x, y);
    if (l) x = __merge(x, l);
    return x;
}

/* return new root */
node *merge(node *x, node *y) {
    if (size(x) < size(y)) swap(x, y);
    if (!y) return x;
    return __merge(x, y);
}

void dfs(node *x) {
    putchar('(');
    if (x) {
        if (x->l) { dfs(x->l); putchar(' '); }
        printf("%d", x->d);
        if (x->r) { putchar(' '); dfs(x->r); }
    }
    putchar(')');
}

```

#### 2.1.4 splay

```

#include<cstdio>
#include<iostream>
#include<algorithm>
using namespace std;
const int MAX_N = 50000 + 10;
const int INF = ~0U >> 1;
struct Node {
    Node*ch[2], *p;
    int size, val, mx;
    int add;
    bool rev;
    Node() {
        size = 0;
        val = mx = -INF;
    }
}

```



```

        add = 0;
    }
    bool d() {
        return this == p->ch[1];
    }
    void setc(Node*c, int d) {
        ch[d] = c;
        c->p = this;
    }
    void addIt(int ad) {
        add += ad;
        mx += ad;
        val += ad;
    }
    void revIt() {
        rev ^= 1;
    }
    void relax();
    void upd() {
        size = ch[0]->size + ch[1]->size + 1;
        mx = max(val, max(ch[0]->mx, ch[1]->mx));
    }
} Tnull, *null = &Tnull;
Node mem[MAX_N], *C = mem;

void Node::relax() {
    if (add != 0) {
        for (int i = 0; i < 2; ++i) {
            if (ch[i] != null)
                ch[i]->addIt(add);
        }
        add = 0;
    }
    if (rev) {
        swap(ch[0], ch[1]);
        for (int i = 0; i < 2; ++i) {
            if (ch[i] != null)
                ch[i]->revIt();
        }
        rev = 0;
    }
}

Node*make(int v) {
    C->ch[0] = C->ch[1] = null;
    C->size = 1;
    C->val = v;
    C->mx = v;
}

```

```

    C->add = 0;
    C->rev = 0;
    return C++;
}

Node*build(int l, int r) {
    if (l >= r)
        return null;
    int m = (l + r) >> 1;
    Node*t = make(0);
    t->setc(build(l, m), 0);
    t->setc(build(m + 1, r), 1);
    t->upd();
    return t;
}

Node*root;

Node*rot(Node*t) {
    Node*p = t->p;
    p->relax();
    t->relax();
    int d = t->d();
    p->p->setc(t, p->d());
    p->setc(t->ch[!d], d);
    t->setc(p, !d);
    p->upd();
    if (p == root)
        root = t;
}

void splay(Node*t, Node*f = null) {
    while (t->p != f) {
        if (t->p->p == f)
            rot(t);
        else
            t->d() == t->p->d() ? (rot(t->p), rot(t)) : (rot(t), rot(t));
    }
    t->upd();
}

Node* select(int k) {
    for (Node*t = root;;) {
        t->relax();
        int c = t->ch[0]->size;
        if (k == c)
            return t;
        if (k > c)

```

```

        k -= c + 1, t = t->ch[1];
    else
        t = t->ch[0];
}
}

Node*&get(int l, int r) { //[l,r)
    Node*L = select(l - 1);
    Node*R = select(r);
    splay(L);
    splay(R, L);
    return R->ch[0];
}

int n, m;

int main() {
    cin >> n >> m;
    root = build(0, n + 2);
    root->p = null;
    for (int i = 0; i < m; ++i) {
        int k, l, r, v;
        scanf("%d%d%d", &k, &l, &r);
        Node*&t = get(l, r + 1);
        if (k == 1) {
            scanf("%d", &v);
            t->addIt(v);
            splay(t);
        } else if (k == 2) {
            t->revIt();
            splay(t);
        } else {
            printf("%d\n", t->mx);
        }
    }
}

```

## 2.2 图上的数据结构

### 2.2.1 左偏树

```

const int MAXN = 2000; //number of nodes

struct node {
    int key, npl, parent, left, right;
};

node heap[MAXN];

```

```

void init() {
    heap[0].npl = -1;
}

int merge(int a, int b) {
    if (!a) return b;
    if (!b) return a;
    if (heap[a].key > heap[b].key) swap(a, b);
    heap[a].right = merge(heap[a].right, b);
    heap[b].parent = a;
    if (heap[heap[a].right].npl > heap[heap[a].left].npl) swap(heap[a].left, heap[a].right);
    heap[a].npl = heap[heap[a].right].npl + 1;
    return a;
}

void remove(int x) {
    int q = heap[x].parent, p = merge(heap[x].left, heap[x].right);
    heap[p].parent = q;
    if (q && heap[q].left == x) heap[q].left = p;
    if (q && heap[q].right == x) heap[q].right = p;
    while (q) {
        if (heap[heap[q].left].npl < heap[heap[q].right].npl) swap(heap[q].left, heap[q].right);
        if (heap[heap[q].right].npl + 1 == heap[q].npl) break;
        heap[q].npl = heap[heap[q].right].npl + 1;
        p = q; q = heap[q].parent;
    }
}

```

### 2.2.2 支持子树操作的动态树

```

const int N=333333;

int n;
int pre[N],fa[N],fat[N],val[N],ma[N],ch[N][2];
bool black[N];
multiset<int> Q[N];
VI E[N];

inline void up(int x){ma[x]=max(max(val[x],*Q[x].rbegin()),max(ma[lch],ma[rch]));}

inline void rot(int id,int tp){
    static int k;
    k=pre[id];
    ch[k][tp^1]=ch[id][tp];
    if(ch[id][tp]) pre[ch[id][tp]]=k;
    if(pre[k]) ch[pre[k]][k==ch[pre[k]][1]]=id;
    pre[id]=pre[k];
    ch[id][tp]=k;
}

```

```

    pre[k]=id;
    up(k);
}

inline void splay(int x){
    if (!pre[x]) return;
    int tmp;
    for(tmp=x;pre[tmp];tmp=pre[tmp]);
    for(swap(fa[x],fa[tmp]);pre[x];rot(x,x==ch[pre[x]][0]));
    up(x);
}

inline int access(int x){
    int nt;
    for(nt=0;x;x=fa[x]){
        splay(x);
        if (rch){
            fa[rch]=x;
            pre[rch]=0;
            Q[x].insert(ma[rch]);
        }
        rch=nt;
        if (nt){
            fa[nt]=0;
            pre[nt]=x;
            Q[x].erase(Q[x].find(ma[nt]));
        }
        up(nt=x);
    }
    return nt;
}

void make(int x,int f){
    fat[x]=f;
    rep(i,E[x].size()) if (E[x][i]!=f) make(E[x][i],x);
    int t;
    up(x+n);up(x+2*n);
    fa[t=x+(1+black[x])*n]=x;
    Q[x].insert(*Q[t].rbegin());
    up(x);
    fa[x]=t=f+(1+black[x])*n;
    Q[t].insert(ma[x]);
}

void cut(int x,int f){
    access(f);
    splay(f);
    splay(x);

```

```

    Q[f].erase(Q[f].find(ma[x]));
    fa[x]=0;
    up(f);
}

void link(int x,int f){
    access(f);
    splay(f);
    splay(x);
    fa[x]=f;
    Q[f].insert(ma[x]);
    up(f);
}

int main(){
    while(~scanf("%d",&n)){
        Cls(pre);
        Cls(ch);
        Cls(fa);
        rep(i,n+1) E[i].clear();
        rep(i,n-1){
            int x,y;
            scanf("%d%d",&x,&y);
            E[x].PB(y);
            E[y].PB(x);
        }
        n++;
        rep(i,3*n+1) Q[i].clear();
        rep(i,3*n+1) ma[i]=val[i]=inf,Q[i].insert(inf);
        REP(i,1,n) scanf("%d",black+i);
        REP(i,1,n) scanf("%d",val+i);
        make(1,n);
        int q,k,x;
        scanf("%d",&q);
        rep(i,q){
            scanf("%d%d",&k,&x);
            if (k==0){
                for(x=access(x);lch;x=lch);
                splay(x);
                printf("%d\n",ma[rch]);
            }
            if (k==1){
                cut(x,fat[x]+(1+black[x])*n);
                cut(x+(1+black[x])*n,x);
                black[x]^=1;
                link(x+(1+black[x])*n,x);
                link(x,fat[x]+(1+black[x])*n);
            }
        }
    }
}

```

```

        if (k==2){
            access(x);
            splay(x);
            scanf("%d",&val+x);
            up(x);
        }
    }
}
return 0;
}

```

## 2.3 可持久化数据结构

### 2.3.1 函数式 treap

```

//By Lin
#include<cstdio>
#include<cstring>
#include<cstdlib>
using namespace std;

struct Node{
    int key,weight,size;
    Node *l,*r;
    Node(int _key , int _weight, Node *_l, Node* _r):
        key(_key),weight(_weight),l(_l),r(_r){
        size = 1;
        if ( l ) size += l->size;
        if ( r ) size += r->size;
    }
    Node *newnode(int key){
        return new Node(key,rand(),NULL,NULL);
    }
    inline int lsize(){ return l?l->size:0; }
    inline int rsize(){ return r?r->size:0; }
}*root[50005];

Node* Meger(Node *a , Node *b ){
    if ( !a || !b ) return a?a:b;
    return a->weight>b->weight?
        new Node(a->key,a->weight,a->l,Meger(a->r,b)):
        new Node(b->key,b->weight,Meger(a,b->l),b->r);
}

Node* Split_L(Node *a ,int size ){
    if ( !a || size == 0 ) return NULL;
    return a->lsize() < size?
        new Node(a->key,a->weight,a->l,Split_L(a->r,size-1-a->lsize())):
        Split_L(a->l,size);
}

```

```

}

Node* Split_R(Node *a ,int size ){
    if ( !a || size == 0 ) return NULL;
    return a->rsize() < size?
        new Node(a->key,a->weight,Split_R(a->l,size-1-a->rsize()),a->r):
        Split_R(a->r,size);
}

int Ask( Node *a ,int k ){
    if ( a->lsize() >= k ) return Ask(a->l,k);
    k -= a->lsize()+1;
    if ( k == 0 ) return a->key;
    return Ask(a->r,k);
}

int len = 0;

int main(){
    int d = 0 , cas;
    scanf("%d", &cas);
    root[0] = NULL;
    int cnt = 1,kind,v,p,c;
    char s[1005];
    while ( cas -- ) {
        scanf("%d", &kind );
        if ( kind == 1 ) {
            scanf("%d%s", &p , s );
            p-=d;
            Node *l = Split_L(root[cnt-1],p),
                *r = Split_R(root[cnt-1],len-p);
            for (int i = 0; s[i]; i++ ){
                l = Meger(l,new Node(s[i],rand(),NULL,NULL));
                len++;
            }
            root[cnt++] = Meger(l,r);
        }
        else if ( kind == 2){
            scanf("%d%d", &p , &c );
            p-=d,c-=d;
            Node *l = Split_L(root[cnt-1],p-1),
                *r = Split_R(root[cnt-1],len-p-c+1);
            len -= c;
            root[cnt++] = Meger(l,r);
        }
        else{
            scanf("%d%d%d", &v, &p , &c );
            v-=d,p-=d,c-=d;

```



```

        char ch;
        for (int i = p; i<p+c; i++) {
            printf("%c", ch = Ask(root[v],i) );
            if ( ch == 'c' ) d++;
        }
        puts("");
    }
}
return 0;
}

```

### 2.3.2 划分树

```

const int D = 18;
const int N = 100000 + 1000;

struct Tree{
    int n;
    int v[N];
    int val[D][N], to_left[D][N];
    LL sum_l[D][N];

    void build(int l, int r, int deep){
        if (l == r) return;
        int mid = (l + r) / 2, left_same = mid - l + 1;
        for(int i = l; i <= r; i++){
            if (val[deep][i] < v[mid]) left_same--;
        }
        int le = l, ri = mid + 1, same = 0;
        sum_l[deep][0] = 0;
        for(int i = l; i <= r; i++){
            to_left[deep][i] = (i == l) ? 0 : to_left[deep][i - 1];
            sum_l[deep][i] = sum_l[deep][i - 1];
            if (val[deep][i] < v[mid]){
                to_left[deep][i]++;
                sum_l[deep][i] += val[deep][i];
                val[deep + 1][le++] = val[deep][i];
            }
            else if (val[deep][i] > v[mid]){
                val[deep + 1][ri++] = val[deep][i];
            }
            else if (same < left_same){
                to_left[deep][i]++;
                sum_l[deep][i] += val[deep][i];
                val[deep + 1][le++] = val[deep][i];
                same++;
            }
        }
        else{

```

```

        val[deep + 1][ri++] = val[deep][i];
    }
}
build(l, mid, deep + 1);
build(mid + 1, r, deep + 1);
}

pair<int, LL> ask(int ask_l, int ask_r, int l, int r, int deep, int kth){
    if (l == r) return MP(val[deep][l], val[deep][l]);
    int mid = (l + r) / 2, s1, s2;
    if (l == ask_l){
        s1 = 0;
        s2 = to_left[deep][ask_r];
    }
    else{
        s1 = to_left[deep][ask_l - 1];
        s2 = to_left[deep][ask_r];
    }
    if (s2 - s1 >= kth){
        ask_l = l + s1, ask_r = ask_l + s2 - s1 - 1;
        return ask(ask_l, ask_r, l, mid, deep + 1, kth);
    }
    else{
        LL ret = sum_1[deep][ask_r] - sum_1[deep][ask_l - 1];
        kth = kth - (s2 - s1);
        s2 = ask_r - ask_l + 1 - (s2 - s1);
        ask_l = mid + ask_l - l + 1 - s1;
        ask_r = ask_l + s2 - 1;
        pair<int, LL> tmp = ask(ask_l, ask_r, mid + 1, r, deep + 1, kth);
        return MP(tmp.F, tmp.S + ret);
    }
}

void init(int n, int other[]){
    for(int i = 1; i <= n; i++){
        v[i] = other[i];
        val[0][i] = v[i];
    }
    sort(v + 1, v + n + 1);
    build(1, n, 0);
}
}tree;

```

## 2.4 其他

### 2.4.1 矩形切割

```

int n, m;
LL ans;

```

```

vi v[N][2];

void work(int lev, vi &a, vi &b);

bool is_in(vi &a, vi &b, vi &a2, vi &b2){
    rep(i, n){
        if (!(a2[i] <= a[i] && b[i] <= b2[i])) return false;
    }
    return true;
}

void dfs(vi &a, vi &b, vi &a2, vi &b2, int now_d, int lev){
    rep(i, n){
        if (a[i] == b[i]) return;
    }
    if (is_in(a, b, a2, b2)) return;
    if (now_d == n) return;
    int l = max(a[now_d], a2[now_d]);
    int r = min(b[now_d], b2[now_d]);
    int tmp_l = a[now_d], tmp_r = b[now_d];

    a[now_d] = l, b[now_d] = r;
    dfs(a, b, a2, b2, now_d + 1, lev);
    a[now_d] = tmp_l, b[now_d] = tmp_r;

    b[now_d] = l;
    work(lev + 1, a, b);
    b[now_d] = tmp_r;

    a[now_d] = r;
    work(lev + 1, a, b);
    a[now_d] = tmp_l;
}

void work(int lev, vi &a, vi &b){
    rep(i, n){
        if (a[i] == b[i]) return;
    }
    if (lev == m){
        LL ret = 1;
        rep(i, n) ret = ret * (b[i] - a[i]) % MD;
        ans = (ans + ret) % MD;
        return;
    }
    vi &a2 = v[lev][0], &b2 = v[lev][1];
    bool no_cover = false;
    rep(i, n){
        int l = max(a[i], a2[i]);

```

```

        int r = min(b[i], b2[i]);
        if (l >= r) no_cover = true;
    }
    if (no_cover) work(lev + 1, a, b);
    else dfs(a, b, a2, b2, 0, lev);
}

int main(){
    while(~scanf("%d%d", &m, &n)){
        rep(i, m){
            v[i][0].clear();
            rep(j, n){
                int x;
                scanf("%d", &x);
                v[i][0].PB(x);
            }
            v[i][1].clear();
            rep(j, n){
                int x;
                scanf("%d", &x);
                v[i][1].PB(x);
            }
            rep(j, n) if (v[i][0][j] > v[i][1][j]) swap(v[i][0][j], v[i][1][j]);
        }
        ans = 0;
        rep(i, m){
            work(i + 1, v[i][0], v[i][1]);
        }
        ans = (ans % MD + MD) % MD;
        printf("%d\n", (int)ans);
    }
    return 0;
}

```

## 3 字符串算法

### 3.1 基础

#### 3.1.1 字符串循环节

对于一个长度为  $n$  的字符串，枚举  $n$  的因子  $d$ ，然后判断  $d$  是否为循环节。只要字符串  $st[1..n-d]$  和  $st[d+1..n]$  相等，则  $d$  为循环节，这一个判断可以用 hash 来完成，时间复杂度为  $O(1)$ 。根据循环节判断的性质，求出 KMP 的 next 数组之后 (假设下标从 1 开始)，那么这个字符串的最小循环长度为  $n - next[n]$ 。

对于字符串的循环节还有这样一个规律: 如果  $\frac{n}{i}$  和  $\frac{n}{j}$  是循环节，且  $i, j$  互质，那么  $\frac{n}{i*j}$  也是循环节。

### 3.1.2 manacher 求最长回文

```
#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;

void manacher(char a[], int n, int rad[]) {
    int i, j, k;
    rad[0] = rad[n] = 0;
    for (i = 1, j = 0; i < n; i += k, j = max(0, j - k)) {
        while (i > j && i + j < n && a[i - j - 1] == a[i + j]) j++;
        for (rad[i] = j, k = 1; k <= rad[i] && rad[i - k] != rad[i] - k; k++)
            rad[i + k] = min(rad[i - k], rad[i] - k);
    }
}

const int N = 500010;
int n;
char a[N];
int rad[N];

int main() {
    scanf("%d %s", &n, a);
    manacher(a, n, rad);
    return 0;
}
```

### 3.1.3 字符串最小表示法

```
const int maxn = 1000000 + 10;

char s[maxn * 2];
int n;

int MinRepresentation() {
    n = strlen(s);
    for (int i = 0; i < n; ++i) s[n + i] = s[i];
    int i = 0, j = 1;
    while (i < n && j < n) {
        int k = 0;
        while (k <= n && s[i + k] == s[j + k]) ++k;
        if (k > n) break;
        if (s[i + k] < s[j + k]) j += k + 1;
        else i += k + 1;
        if (i == j) ++j;
    }
    return (i < j) ? i : j;
}
```

### 3.1.4 kmp

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 1000010;

int next[N];

int kmp(char *s, int n, char *t, int m) {
    int i, j;
    next[0] = -1;
    i = 0; j = -1;
    while (i < m) {
        if (j == -1 || t[i] == t[j]) {
            i++; j++;
            next[i] = (t[i] == t[j] ? next[j] : j);
        } else {
            j = next[j];
        }
    }
    i = j = 0;
    while (i < n && j < m) {
        if (j == -1 || s[i] == t[j]) {
            i++; j++;
        } else {
            j = next[j];
        }
    }
    return (j >= m ? i - m : -1);
}
```

### 3.1.5 扩展kmp

```
int ext[maxn]; // lcp(pat's suffix, pat)
int ex[maxn]; // lcp(pat's suffix, str)
//exp. str = "aaaba", pat = "aba", then ex[] = {1, 1, 3, 0, 1}, ext[] = {3, 0, 1}
//la = strlen(str), lb = strlen(pat);
void extkmp(char *str, char *pat, int ext[], int ex[]) {
    int p=0,k=1;
    while(pat[p] == pat[p+1]) p++;
    ext[0] = lb, ext[1] = p;
    for(int i=2;i<lb;i++){
        int x = k + ext[k] - i, y = ext[i - k];
        if (y < x) ext[i] = y;
        else{
            p = max(0, x);
        }
    }
}
```

```

        while (pat[p] == pat[p+i]) p++;
        ext[i] = p;
        k = i;
    }
}
p = k = 0;
while(str[p] && str[p] == pat[p]) p++;
ex[0] = p;
for(int i=1;i<la;i++){
    int x = k + ex[k] - i, y = ext[i - k];
    if (y < x) ex[i] = y;
    else{
        p = max(0, x);
        while (pat[p] && pat[p] == str[p+i]) p++;
        ex[i] = p;
        k = i;
    }
}
}
}

```

## 3.2 进阶

### 3.2.1 ac自动机

```

int root, idx;
struct trie_node{
    int next[size];
    int fail;
    bool flag;
    void init(){
        fail = -1, flag = false;
        memset(next, 0, sizeof(next));
    }
}trie[maxn * leng];
int q[maxn * leng];
void trie_init(){
    root = idx = 0;
    trie[root].init();
}
void insert(char *s){
    int i, j, p = root;
    for(i=0;s[i];i++){
        j = s[i] - 'A';
        if(!trie[p].next[j]){
            trie[++idx].init();
            trie[p].next[j] = idx;
        }
        p = trie[p].next[j];
    }
}

```

```

        trie[p].flag = true;
    }
    void build(){
        int j, p;
        q[0] = root;
        for(int l=0,h=1;l<h;){
            p = q[l++];
            for(j=0;j<size;j++){
                if(trie[p].next[j]){
                    q[h++] = trie[p].next[j];
                    if(trie[p].fail == -1)
                        trie[trie[p].next[j]].fail = root;
                    else{
                        trie[trie[p].next[j]].fail =
                            trie[trie[p].fail].next[j];

                        trie[trie[p].next[j]].flag |=
                            trie[trie[trie[p].fail].next[j]].flag;
                    }
                }
            }
            else{
                if(trie[p].fail != -1)
                    trie[p].next[j] = trie[trie[p].fail].next[j];
            }
        }
    }
}

```

### 3.2.2 Trie图

/\*

trie图用于多模式串匹配,一般的题目都是要求生成的字符串中不包含一些字符串  
建立trie图,然后从根节点开始走,只要不走到危险节点就可以了

注意: trie图的2个性质: 1、一个节点的危险性和它后缀节点相同 2、根节点的直接儿子的后缀是根节点

\*/

```

#include <string>
#include <queue>
#include <cstdio>
#include <cstring>
#include <iostream>
using namespace std;

```

```

class Trie{
public:
    int tot;
    const static int kMaxN=100;

public:

```



```

    char ch[kMaxN];
    bool danger[kMaxN];
    int g[kMaxN],next[kMaxN],suffix[kMaxN];

public:
    Trie(){
        tot=1;
        memset(g,255,sizeof(g));
        memset(danger,0,sizeof(danger));
    }
    int getson(int node, char c, bool flag);
    void insert(string st);
    void makegraph();
};

int Trie::getson(int node, char c, bool flag){
    while (true){
        int now=g[node];
        while (now!=-1 && ch[now]!=c) now=next[now];
        if (now!=-1 || !flag) return now;
        if (node==1) return node;
        node=suffix[node];
    }
}

void Trie::insert(string st){
    int len=st.size();
    int now=1,son;
    for (int i=0; i<len; i++){
        son=getson(now,st[i],false);
        if (son==-1){
            ++tot;
            ch[tot]=st[i];
            next[tot]=g[now];
            g[now]=tot;
            now=tot;
        } else now=son;
    }
    danger[now]=true;
}

void Trie::makegraph(){
    queue<int> q;
    q.push(1);
    while (!q.empty()){
        int node=q.front(); q.pop();
        for (int p=g[node]; p!=-1; p=next[p]){
            q.push(p);
        }
    }
}

```

```

        if (node==1){
            suffix[p]=1;
        } else{
            suffix[p]=getson(suffix[node],ch[p],true);
        }
        danger[p] |= danger[suffix[p]];
    }
}

int main(){
    string st;
    Trie graph;
    cin>>st;
    graph.insert(st);
    cin>>st;
    graph.insert(st);
    return 0;
}

```

### 3.2.3 后缀自动机

```

#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
#include <climits>
#include <numeric>
#include <vector>
using namespace std;

const int MAX_N = 1000000 + 10;
struct State {
    State*suf, *go[26], *nxt;
    int val, cnt;
    State() :
        suf(0), val(0) {
        memset(go, 0, sizeof go);
    }
}*root, *last;
State statePool[MAX_N * 2], *cur;
State*first[MAX_N] = { };

void init() {
    cur = statePool;
    root = last = cur++;
}

```

```

void extend(int w) {
    State*p = last, *np = cur++;
    np->val = p->val + 1;
    np->cnt = 1;
    while (p && !p->go[w])
        p->go[w] = np, p = p->suf;
    if (!p)
        np->suf = root;
    else {
        State*q = p->go[w];
        if (p->val + 1 == q->val) {
            np->suf = q;
        } else {
            State*nq = cur++;
            memcpy(nq->go, q->go, sizeof q->go);
            nq->val = p->val + 1;
            nq->suf = q->suf;
            q->suf = nq;
            np->suf = nq;
            while (p && p->go[w] == q)
                p->go[w] = nq, p = p->suf;
        }
    }
    last = np;
}

int main() {
    string str;
    cin >> str;
    init();
    int L = str.size();
    for (int i = 0; i < L; ++i) {
        extend(str[i] - 'a');
    }
    for (State*i = statePool; i != cur; ++i)
        i->nxt = first[i->val], first[i->val] = i;
    for (int it = L; it >= 0; --it) {
        for (State*i = first[it]; i; i = i->nxt)
            if (i->suf)
                i->suf->cnt += i->cnt;
    }
    // cout << root->go[0]->go[0]->cnt << endl;
    return 0;
}

```

## 4 计算几何

### 4.1 平面几何基础

#### 4.1.1 几何

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <algorithm>
#include <cmath>

#define Sqr(x) (x) * (x)
#define sign(x) ((x < -EPS) ? -1 : x > EPS)

using namespace std;

const double EPS = 1E-8;
const double INF = 1E+9;
const double PI = acos(-1.0);

typedef complex<double> Point;

bool operator<(const Point &a, const Point &b){
    int f = sign(a.X - b.X);
    if (f) return f < 0;
    return sign(a.Y - b.Y) < 0;
}

double cross(Point a, Point b){
    return a.X * b.Y - a.Y * b.X;
}

double cross(Point a, Point b, Point c){
    return cross(b - a, c - a);
}

double dot(Point a, Point b){
    return a.X * b.X + a.Y * b.Y;
}

double dot(Point a, Point b, Point c){
    return dot(b - a, c - a);
}

double dist(Point a, Point b){
    return abs(a - b);
}
```

```

Point rotate(Point v, double alpha){
    double c = cos(alpha), s = sin(alpha);
    return Point(v.X * c - v.Y * s, v.X * s + v.Y * c);
}

double fix(double a, double b = 0) {
    a -= b;
    if (sign(a) < 0) a += 2 * pi;
    if (sign(a - 2 * pi) >= 0) a -= 2 * pi;
    return a;
}

double angle(Point a, Point b){
    return fix(arg(b - a));
}

Point centroid(Point a[], int n){
    double area = 0;
    Point c = Point(0, 0);
    a[n] = a[0];
    for(int i = 0; i < n; i++){
        area += cross(a[i], a[i + 1]);
        c.x += (a[i].X + a[i + 1].X) * cross(a[i], a[i + 1]);
        c.y += (a[i].Y + a[i + 1].Y) * cross(a[i], a[i + 1]);
    }
    area = fabs(area) / 2;
    c.x /= 6 * area;
    c.y /= 6 * area;
    return c;
}

/*
 * convex hull
 */

Point __o;

bool cmp_p(Point a, Point b){
    int f = sign(a.X - b.X);
    if (f) return f < 0;
    return sign(a.Y - b.Y) < 0;
}

bool cmp(Point a, Point b){
    int f = sign(cross(o, a, b));
    if (f) return f > 0;
    return sign(abs(a - o) - abs(b - o)) < 0;
}

```

```

}

/*
 * find convex hull of p[n] in place
 * return # of points of resulting convex hull
 */
Point stack[1111]
int find_convex(Point p[], int n){
    __o = *min_element(p, p + n, cmp_p);
    sort(p, p + n, cmp);
    int top = 0;
    rep(i, n){
        while(top >= 2 && sign(cross(stack[top - 2], stack[top - 1], p[i])) <= 0) top--;
        stack[top++] = p[i];
    }
    rep(i, top) p[i] = stack[i];
    return top;
}

/*
 * rotate calipers
 */
double shadow_length(double alpha, Point a, Point b){
    double dx = a.x - b.x;
    double dy = a.y - b.y;
    double c = cos(alpha);
    double s = sin(alpha);
    return fabs(dx * c + dy * s);
}

/*
 * min area & min peri rectangle covering, using rotate calipers
 */
void rotate_calipers(Point ps[], int n, double &area, double &peri){
    area = peri = INF;
    n = find_convex(ps, n);
    ps[n] = ps[0];
    Point *q[4] = {NULL, NULL, NULL, NULL};
    for(int i = 0; i < n; i++){
        Point *p = &ps[i];
        if (!q[0] || q[0]->Y > p->Y || q[0]->Y == p->Y && q[0]->X > p->X) q[0] = p;
        if (!q[1] || q[1]->X < p->X || q[1]->X == p->X && q[1]->Y > p->Y) q[1] = p;
        if (!q[2] || q[2]->Y < p->Y || q[2]->Y == p->Y && q[2]->X < p->X) q[2] = p;
        if (!q[3] || q[3]->X > p->X || q[3]->X == p->X && q[3]->Y < p->Y) q[3] = p;
    }
    double alpha = 0;
    for(int k = 0; k < n + 5; k++){
        int bi = -1;

```

```

    double gap_min = INF;
    for(int i = 0; i < 4; i++){
        double gap = fix(angle(q[i][0], q[i][1]), alpha + i * PI / 2);
        if (gap < gap_min){
            gap_min = gap;
            bi = i;
        }
    }
    if (++q[bi] == ps + n) q[bi] = ps + 0;
    alpha = fix(alpha + gap_min);
    double a = shadow_length(alpha + PI / 2, *q[0], *q[2]);
    double b = shadow_length(alpha, *q[1], *q[3]);
    area = min(area, a * b);
    peri = min(peri, a + a + b + b);
}
}

/*
 * lines
 */
typedef pair<Point, Point> Line;

int parallel(Line a, Line b){
    return !sign(cross(a.Y - a.X, b.Y - b.X));
}

/*
 * same side : 1
 * at least one of a, b touches l : 0;;
 * other wise : -1
 */
int side(Line m, Point p, Point q){
    return sign(cross(m.X, m.Y, p)) * sign(cross(m.X, m.Y, q));
}

int on_line(Line l, Point p){
    return !sign(cross(l.p, l.q, p));
}

/*
 * u, v : line
 */
int coinside(Line u, Line v){
    return on_line(u, v.X) && on_line(u, v.Y);
}

/*
 * u, v : line segment, inclusive

```

```

    */
int intersected(Line u, Line v){
    return !parallel(u, v) && side(u, v.X, v.Y) <= 0 && side(v, u.X, u.Y) <= 0;
}

/*
 * u, v : line segment , exclusive
 */
int intersected_exclusive(Line u, Line v){
    return !parallel(u, v) && side(u, v.X, v.Y) < 0 && side(v, u.X, u.Y) < 0;
}

/*
 * intersection point
 * must check whether exist or not
 */
Point ip(Line u, Line v){
    double n = (u.p.y - v.p.y) * (v.q.x - v.p.x) - (u.p.x - v.p.x) * (v.q.y - v.p.y);
    double d = (u.q.x - u.p.x) * (v.q.y - v.p.y) - (u.q.y - u.p.y) * (v.q.x - v.p.x);
    double r = n / d;
    return Point(u.p.x + r * (u.q.x - u.p.x), u.p.y + r * (u.q.y - u.p.y));
}

bool inter(Line a, Line b, Point &p){
    double s1 = cross(a.F, a.S, b.F);
    double s2 = cross(a.F, a.S, b.S);
    if (!sign(s1 - s2)) return false;
    p = (s1 * b.S - s2 * b.F) / (s1 - s2);
    return true;
}

/*
 * if P on the Line segment l, inclusive
 */
int on_lineseg(Line l, Point p){
    return on_line(l, p) && sign(dot(p, l.p, l.q)) <= 0;
}

/*
 * if P on the segment l, exclusive
 */
int on_lineseg_exclusive(Line l, Point p){
    return on_line(l, p) && sign(dot(p, l.p, l.q)) < 0;
}

double dist_line_point(Line l, Point a){
    return fabs(cross(l.p, l.q, a)) / dist(l.p, l.q);
}

```



```

double dist_lineseg_point(Line l, Point a){
    if (on_lineseg(l, a)) return 0;
    if (on_line(l, a) || !sharp(l.p, a, l.q)) return min(dist(l.p, a), dist(l.q, a));
    return dist_line_point(l, a);
}

/*
 * u : line segment
 * ab : ray, if p is the resulting intersection point
 */
int intersected_lineseg_ray(Line u, Line v, Point &p){
    if (parallel(u, v)) return 0;
    p = ip(u, v);
    return on_lineseg(u, p) && (on_lineseg(v, p) || on_lineseg(Line(v.p, p), v.q));
}

/*
 * if point a inside polygon p[n]
 */
int inside_polygon(Point p[], int n, Point a){
    double sum = 0;
    for(int i = 0; i < n; i++){
        int j = (i + 1) % n;
        if (on_lineseg(Line(p[i], p[j]), a)) return 0;
        double angle = acos(dot(a, p[i], p[j]) / dist(a, p[i]) / dist(a, p[j]));
        sum += sign(cross(a, p[i], p[j])) * angle;
    }
    return sign(sum);
}

/*
 * if lineseg l strickly inside polygon p[n]
 */
int lineseg_inside_polygon(Point p[], int n, Line l){
    for(int i = 0; i < n; i++){
        int j = (i + 1) % n;
        Line l1(p[i], p[j]);
        if (on_lineseg_exclusive(l, p[i])) return 0;
        if (intersected_exclusive(l, l1)) return 0;
    }
    return inside_polygon(p, n, mp(l.p, l.q));
}

/*
 * if lineseg l intersect convex polygon p[n]
 */
int intersect_convex_lineseg(Point p[], int n, Line l){

```

```

    if (n < 3) return 0;
    Point q[4];
    int k = 0;
    q[k++] = l.p;
    q[k++] = l.q;
    for(int i = 0; i < n; i++){
        if (on_lineseg(l, p[i])){
            q[k++] = p[i];
        }
        else{
            int j = (i + 1) % n;
            Line a(p[i], p[j]);
            Point tmp = ip(a, l);
            if (on_lineseg(l, tmp) && on_lineseg(a, tmp)) q[k++] = tmp;
        }
    }
    sort(q, q + k);
    for(int i = 0; i + 1 < k; i++){
        if (inside_polygon(p, n, mp(q[i], q[i + 1]))) return 1;
    }
    return 0;
}

Line perpendicular(Line l, Point a){
    return Line(a, Point(a.x + l.p.y - l.q.y, a.y + l.q.x - l.p.x));
}

Point pedal(Line l, Point a){
    return ip(l, perpendicular(l, a));
}

Point mirror(Line l, Point a){
    Point p = pedal(l, a);
    return Point(p.x * 2 - a.x, p.y * 2 - a.y);
}

Point perpencenter(Point a, Point b, Point c){
    Line u = perpendicular(Line(b, c), a);
    Line v = perpendicular(Line(a, c), b);
    return ip(u, v);
}

/*
 * Inscribed circle center
 */
Point icc(Point A, Point B, Point C){
    double a = dist(B, C);
    double b = dist(C, A);

```

```

    double c = dist(A, B);
    double p = (a + b + c) / 2;
    double s = sqrt(p * (p - a) * (p - b) * (p - c));
    Point cp;
    cp.x = (a * A.x + b * B.x + c * C.x) / (a + b + c);
    cp.y = (a * A.y + b * B.y + c * C.y) / (a + b + c);
    return cp;
}

/*
 * Perpendicular bisector
 */
Line pb(Point a, Point b){
    return perpendicular(Line(a, b), mp(a, b));
}

/*
 * circumcircle center
 */
Point ccc(Point A, Point B, Point C){
    double a1 = B.x - A.x, b1 = B.y - A.y, c1 = (Sqr(a1) + Sqr(b1)) / 2;;
    double a2 = C.x - A.x, b2 = C.y - A.y, c2 = (Sqr(a2) + Sqr(b2)) / 2;;
    double d = a1 * b2 - a2 * b1;
    Point cp;
    cp.x = A.x + (c1 * b2 - c2 * b1) / d;
    cp.y = A.y + (a1 * c2 - a2 * c1) / d;
    return cp;
}

/*
 * translate l with distance e and direction s
 */
Line translate(Line l, double e, int s){
    double d = dist(l.p, l.q);
    double x = l.p.y - l.q.y;
    double y = l.q.x - l.p.x;
    x *= s * e / d;
    y *= s * e / d;
    l.p.x += x; l.p.y += y;
    l.q.x += x; l.q.y += y;
    return l;
}

/*
 * area of the part of convex polygon p[n] on the positive side of l
 */
double cut_area(Point *p, int n, Line l){
    int ai, bi;

```

```

Point ap, bp;
ai = bi = -1;
for(int i = 0; i < n; i++){
    Line v(p[i], p[i + 1]);
    if (parallel(v, l)) continue;
    Point cp = ip(v, l);
    if (cp == p[i] || on_lineseg_exclusive(v, cp)){
        if (ai == -1){
            ai = i;
            ap = cp;
        }
        else{
            bi = i;
            bp = cp;
        }
    }
}

Point *q = new Point[n + 2]; //XXX new here
int m = 0;
for(int i = 0; i < n; i++){
    if (sign(cross(l.p, l.q, p[i])) >= 0){
        q[m++] = p[i];
    }
    if (i == ai){
        q[m++] = ap;
    }
    if (i == bi){
        q[m++] = bp;
    }
    double res = area_polygon(q, m);
    free(q);
    return res;
}

}

/*
 * -----intersection points convex hull-----
 */
bool lcmp(Line u, Line v){
    int c = sign((u.p.x - u.q.x) * (v.p.y - v.q.y) - (v.p.x - v.q.x) * (u.p.y - u.q.y));
    return c < 0 || !c && sign(cross(u.p, u.q, v.p)) < 0;
}

/*
 * XXX sizeof(p) MUST be as large as n * 2
 * return # of points of resulting convex hull
 */
int ip_convex(Line l[], int n, Point p[]){

```

```

    for(int i = 0; i < n; i++){
        if (l[i].q < l[i].p) swap(l[i].p, l[i].q);
    }
    sort(l, l + n, lcmp);
    int n1 = 0;
    for(int i = 0, j = 0; i < n; i = j){
        while(j < n && parallel(l[i], l[j])) j++;
        if (j - i == 1){
            l[n1++] = l[i];
        }
        else{
            l[n1++] = l[i];
            l[n1++] = l[j - 1];
        }
    }
    n = n1;
    l[n + 0] = l[0];
    l[n + 1] = l[1];
    int m = 0;
    for(int i = 0, j = 0; i < n; i++){
        while(j < n + 2 && parallel(l[i], l[j])) j++;
        for(int k = j; k < n + 2 && parallel(l[j], l[k]); k++){
            p[m++] = ip(l[i], l[k]);
        }
    }
    return find_convex(p, m);
}

/*
 * -----circles-----
 */
struct Circle{
    Point o;
    double r;
    Circle(Point o = Point(), double r = 1) : o(o), r(r){}
    Circle(double x, double y, double r = 1) : o(x, y), r(r){}
};

int intersected_circle_line(Circle c, Line l){
    return sign(dist_line_point(l, c.o) - c.r) < 0;
}

int ip_circle_line(Circle c, Line l, Point &p1, Point &p2){
    Point a = l.p, b = l.q;
    double dx = b.x - a.x;
    double dy = b.y - a.y;
    double sdr = Sqr(dx) + Sqr(dy);
    double dr = sqrt(sdr);

```

```

double d, disc, x, y;
a.x -= c.o.x; a.y -= c.o.y;
b.x -= c.o.x; b.y -= c.o.y;
d = a.x * b.y - b.x * a.y;
disc = Sqr(c.r) * sdr - Sqr(d);
if (disc < -EPS) return 0;
if (disc < +EPS){
    disc = 0;
}
else{
    disc = sqrt(disc);
}
x = disc * dx * (dy > 0 ? 1 : -1);
y = disc * fabs(dy);
p1.x = (+d * dy + x) / sdr + c.o.x;
p2.x = (+d * dy - x) / sdr + c.o.x;
p1.y = (-d * dx + y) / sdr + c.o.y;
p2.y = (-d * dx - y) / sdr + c.o.y;
return disc > EPS ? 2 : 1;
}

int ip_circle_circle(const Circle &c1, const Circle &c2, Point &p1, Point &p2){
    double mx = c2.o.x - c1.o.x, sx = c2.o.x + c1.o.x, mx2 = Sqr(mx);
    double my = c2.o.y - c1.o.y, sy = c2.o.y + c1.o.y, my2 = Sqr(my);
    double sq = mx2 + my2, d = -(sq - Sqr(c1.r - c2.r)) * (sq - Sqr(c1.r + c2.r));
    if (!sign(sq)) return 0;
    if (d + EPS < 0) return 0;
    if (d < EPS){
        d = 0;
    }
    else{
        d = sqrt(d);
    }
    double x = mx * ((c1.r + c2.r) * (c1.r - c2.r) + mx * sx) + sx * my2;
    double y = my * ((c1.r + c2.r) * (c1.r - c2.r) + my * sy) + sy * mx2;
    double dx = mx * d, dy = my * d;
    sq *= 2;
    p1.x = (x + dy) / sq; p1.y = (y - dx) / sq;
    p2.x = (x - dy) / sq; p2.y = (y + dx) / sq;
    return d > EPS ? 2 : 1;
}

double circle_circle_intersection_area(Circle A, Circle B){
    double d, dA, dB, tx, ty;
    d = hypot(B.o.x - A.o.x, B.o.y - A.o.y);
    if ((d < EPS) || (d + A.r <= B.r) || (d + B.r <= A.r)){
        return Sqr(B.r < A.r ? B.r : A.r) * PI;
    }
}

```

```

    if (d >= A.r + B.r){
        return 0;
    }
    dA = tx = (Sqr(d) + Sqr(A.r) - Sqr(B.r)) / d / 2;
    ty = sqrt(Sqr(A.r) - Sqr(tx));
    dB = d - dA;
    return Sqr(A.r) * acos(dA / A.r) - dA * sqrt(Sqr(A.r) - Sqr(dA)) + Sqr(B.r) * acos(dB / B.r) -
}

/*
 * return 2 points of tangency of c and p
 */
void circle_tangents(Circle c, Point p, Point &a, Point &b){
    double d = Sqr(c.o.x - p.x) + Sqr(c.o.y - p.y);
    double para = Sqr(c.r) / d;
    double perp = c.r * sqrt(d - Sqr(c.r)) / d;
    a.x = c.o.x + (p.x - c.o.x) * para - (p.y - c.o.y) * perp;
    a.y = c.o.y + (p.y - c.o.y) * para + (p.x - c.o.x) * perp;
    b.x = c.o.x + (p.x - c.o.x) * para + (p.y - c.o.y) * perp;
    b.y = c.o.y + (p.y - c.o.y) * para - (p.x - c.o.x) * perp;
}

/*
 * +0 : on circle;
 * +1 : inside circle;
 * -1 : outside circle;
 */
int on_circle(Circle c, Point a){
    return sign(c.r - dist(a, c.o));
}

/*
 * minimum circle that covers 2 points
 */
Circle cc2(Point a, Point b){
    return Circle(mp(a, b), dist(a, b) / 2);
}

Circle cc3(Point p, Point q, Point r){
    Circle c;
    if (on_circle(c = cc2(p, q), r) >= 0) return c;
    if (on_circle(c = cc2(p, r), q) >= 0) return c;
    if (on_circle(c = cc2(q, r), p) >= 0) return c;
    c.o = ccc(p, q, r);
    c.r = dist(c.o, p);
    return c;
}

```

```

Circle min_circle_cover(Point p[], int n){
    if (n == 1) return Circle(p[0], 0);
    if (n == 2) return cc2(p[0], p[1]);
    random_shuffle(p, p + n);
    Point *ps[4] = {&p[0], &p[1], &p[2], &p[3]};
    Circle c = cc3(*ps[0], *ps[1], *ps[2]);
    while(true){
        Point *b = p;
        for(int i = 1; i < n; i++){
            if (dist(p[i], c.o) > dist(*b, c.o)) b = &p[i];
        }
        if (on_circle(c, *b) >= 0) return c;
        ps[3] = b;
        for(int i = 0; i < 3; i++){
            swap(ps[i], ps[3]);
            if (on_circle(c = cc3(*ps[0], *ps[1], *ps[2]), *ps[3]) >= 0) break;
        }
    }
}

```

#### 4.1.2 圆

```

struct Circle{
    Point o;
    double r;
    Circle(Point o = Point(), double r = 1) : o(o), r(r){}
    Circle(double x, double y, double r = 1) : o(x, y), r(r){}
};

```

```

int intersected_circle_line(Circle c, Line l){
    return sign(dist_line_point(l, c.o) - c.r) < 0;
}

```

```

int ip_circle_line(Circle c, Line l, Point &p1, Point &p2){
    Point a = l.p, b = l.q;
    double dx = b.x - a.x;
    double dy = b.y - a.y;
    double sdr = Sqr(dx) + Sqr(dy);
    double dr = sqrt(sdr);
    double d, disc, x, y;
    a.x -= c.o.x; a.y -= c.o.y;
    b.x -= c.o.x; b.y -= c.o.y;
    d = a.x * b.y - b.x * a.y;
    disc = Sqr(c.r) * sdr - Sqr(d);
    if (disc < -EPS) return 0;
    if (disc < +EPS){
        disc = 0;
    }
}

```



```

}
else{
    disc = sqrt(disc);
}
x = disc * dx * (dy > 0 ? 1 : -1);
y = disc * fabs(dy);
p1.x = (+d * dy + x) / sdr + c.o.x;
p2.x = (+d * dy - x) / sdr + c.o.x;
p1.y = (-d * dx + y) / sdr + c.o.y;
p2.y = (-d * dx - y) / sdr + c.o.y;
return disc > EPS ? 2 : 1;
}

int ip_circle_circle(const Circle &c1, const Circle &c2, Point &p1, Point &p2){
    double mx = c2.o.x - c1.o.x, sx = c2.o.x + c1.o.x, mx2 = Sqr(mx);
    double my = c2.o.y - c1.o.y, sy = c2.o.y + c1.o.y, my2 = Sqr(my);
    double sq = mx2 + my2, d = -(sq - Sqr(c1.r - c2.r)) * (sq - Sqr(c1.r + c2.r));
    if (!sign(sq)) return 0;
    if (d + EPS < 0) return 0;
    if (d < EPS){
        d = 0;
    }
    else{
        d = sqrt(d);
    }
    double x = mx * ((c1.r + c2.r) * (c1.r - c2.r) + mx * sx) + sx * my2;
    double y = my * ((c1.r + c2.r) * (c1.r - c2.r) + my * sy) + sy * mx2;
    double dx = mx * d, dy = my * d;
    sq *= 2;
    p1.x = (x + dy) / sq; p1.y = (y - dx) / sq;
    p2.x = (x - dy) / sq; p2.y = (y + dx) / sq;
    return d > EPS ? 2 : 1;
}

double circle_circle_intersection_area(Circle A, Circle B){
    double d, dA, dB, tx, ty;
    d = hypot(B.o.x - A.o.x, B.o.y - A.o.y);
    if ((d < EPS) || (d + A.r <= B.r) || (d + B.r <= A.r)){
        return Sqr((B.r < A.r) ? B.r : A.r) * PI;
    }
    if (d >= A.r + B.r){
        return 0;
    }
    dA = tx = (Sqr(d) + Sqr(A.r) - Sqr(B.r)) / d / 2;
    ty = sqrt(Sqr(A.r) - Sqr(tx));
    dB = d - dA;
    return Sqr(A.r) * acos(dA / A.r) - dA * sqrt(Sqr(A.r) - Sqr(dA)) + Sqr(B.r) * acos(dB / B.r) -
}

```

```

/*
 * return 2 points of tangency of c and p
 */
void circle_tangents(Circle c, Point p, Point &a, Point &b){
    double d = Sqr(c.o.x - p.x) + Sqr(c.o.y - p.y);
    double para = Sqr(c.r) / d;
    double perp = c.r * sqrt(d - Sqr(c.r)) / d;
    a.x = c.o.x + (p.x - c.o.x) * para - (p.y - c.o.y) * perp;
    a.y = c.o.y + (p.y - c.o.y) * para + (p.x - c.o.x) * perp;
    b.x = c.o.x + (p.x - c.o.x) * para + (p.y - c.o.y) * perp;
    b.y = c.o.y + (p.y - c.o.y) * para - (p.x - c.o.x) * perp;
}

/*
 * +0 : on circle;
 * +1 : inside circle;
 * -1 : outside circle;
 */
int on_circle(Circle c, Point a){
    return sign(c.r - dist(a, c.o));
}

/*
 * minimum circle that covers 2 points
 */
Circle cc2(Point a, Point b){
    return Circle(mp(a, b), dist(a, b) / 2);
}

Circle cc3(Point p, Point q, Point r){
    Circle c;
    if (on_circle(c = cc2(p, q), r) >= 0) return c;
    if (on_circle(c = cc2(p, r), q) >= 0) return c;
    if (on_circle(c = cc2(q, r), p) >= 0) return c;
    c.o = ccc(p, q, r);
    c.r = dist(c.o, p);
    return c;
}

Circle min_circle_cover(Point p[], int n){
    if (n == 1) return Circle(p[0], 0);
    if (n == 2) return cc2(p[0], p[1]);
    random_shuffle(p, p + n);
    Point *ps[4] = {&p[0], &p[1], &p[2], &p[3]};
    Circle c = cc3(*ps[0], *ps[1], *ps[2]);
    while(true){
        Point *b = p;

```

```

    for(int i = 1; i < n; i++){
        if (dist(p[i], c.o) > dist(*b, c.o)) b = &p[i];
    }
    if (on_circle(c, *b) >= 0) return c;
    ps[3] = b;
    for(int i = 0; i < 3; i++){
        swap(ps[i], ps[3]);
        if (on_circle(c = cc3(*ps[0], *ps[1], *ps[2]), *ps[3]) >= 0) break;
    }
}
}

```

#### 4.1.3 垂心, 内心, 外心

```

point ip(line u, line v) {
    double n = (u.p.y - v.p.y) * (v.q.x - v.p.x) - (u.p.x - v.p.x) * (v.q.y - v.p.y);
    double d = (u.q.x - u.p.x) * (v.q.y - v.p.y) - (u.q.y - u.p.y) * (v.q.x - v.p.x);
    double r = n / d;
    return point(u.p.x + r * (u.q.x - u.p.x), u.p.y + r * (u.q.y - u.p.y));
}

```

```

Line perpendicular(Line l, Point a){
    return Line(a, Point(a.x + l.p.y - l.q.y, a.y + l.q.x - l.p.x));
}

```

```

Point pedal(Line l, Point a){
    return ip(l, perpendicular(l, a));
}

```

```

Point mirror(Line l, Point a){
    Point p = pedal(l, a);
    return Point(p.x * 2 - a.x, p.y * 2 - a.y);
}

```

//垂心

```

Point perpencenter(Point a, Point b, Point c){
    Line u = perpendicular(Line(b, c), a);
    Line v = perpendicular(Line(a, c), b);
    return ip(u, v);
}

```

//内心

```

Point icc(Point A, Point B, Point C){
    double a = dist(B, C);
    double b = dist(C, A);
    double c = dist(A, B);
    double p = (a + b + c) / 2;
    double s = sqrt(p * (p - a) * (p - b) * (p - c));
}

```

```

    Point cp;
    cp.x = (a * A.x + b * B.x + c * C.x) / (a + b + c);
    cp.y = (a * A.y + b * B.y + c * C.y) / (a + b + c);
    return cp;
}

//外心
Point ccc(Point A, Point B, Point C){
    double a1 = B.x - A.x, b1 = B.y - A.y, c1 = (Sqr(a1) + Sqr(b1)) / 2;;
    double a2 = C.x - A.x, b2 = C.y - A.y, c2 = (Sqr(a2) + Sqr(b2)) / 2;;
    double d = a1 * b2 - a2 * b1;
    Point cp;
    cp.x = A.x + (c1 * b2 - c2 * b1) / d;
    cp.y = A.y + (a1 * c2 - a2 * c1) / d;
    return cp;
}

```

#### 4.1.4 一般多边形

```

/*
 * if point a inside polygon p[n]
 */
int inside_polygon(Point p[], int n, Point a){
    double sum = 0;
    for(int i = 0; i < n; i++){
        int j = (i + 1) % n;
        if (on_lineseg(Line(p[i], p[j]), a)) return 0;
        double angle = acos(dot(a, p[i], p[j]) / dist(a, p[i]) / dist(a, p[j]));
        sum += sign(cross(a, p[i], p[j])) * angle;
    }
    return sign(sum);
}

/*
 * if lineseg l strickly inside polygon p[n]
 */
int lineseg_inside_polygon(Point p[], int n, Line l){
    for(int i = 0; i < n; i++){
        int j = (i + 1) % n;
        Line l1(p[i], p[j]);
        if (on_lineseg_exclusive(l, p[i])) return 0;
        if (intersected_exclusive(l, l1)) return 0;
    }
    return inside_polygon(p, n, mp(l.p, l.q));
}

/*
 * if lineseg l intersect convex polygon p[n]

```

```

*/
int intersect_convex_lineseg(Point p[], int n, Line l){
    if (n < 3) return 0;
    Point q[4];
    int k = 0;
    q[k++] = l.p;
    q[k++] = l.q;
    for(int i = 0; i < n; i++){
        if (on_lineseg(l, p[i])){
            q[k++] = p[i];
        }
        else{
            int j = (i + 1) % n;
            Line a(p[i], p[j]);
            Point tmp = ip(a, l);
            if (on_lineseg(l, tmp) && on_lineseg(a, tmp)) q[k++] = tmp;
        }
    }
    sort(q, q + k);
    for(int i = 0; i + 1 < k; i++){
        if (inside_polygon(p, n, mp(q[i], q[i + 1]))) return 1;
    }
    return 0;
}

```

```

#define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))

```

```

Point isSS(Point p1, Point p2, Point q1, Point q2) {
    double a1 = cross(q1,q2,p1), a2 = -cross(q1,q2,p2);
    return (p1 * a2 + p2 * a1) / (a1 + a2);
}

```

```

vector<Point> convexCut(const vector<Point>&ps, Point q1, Point q2) {
    vector<Point> qs;
    int n = ps.size();
    for (int i = 0; i < n; ++i) {
        Point p1 = ps[i], p2 = ps[(i + 1) % n];
        int d1 = crossOp(q1,q2,p1), d2 = crossOp(q1,q2,p2);
        if (d1 >= 0)
            qs.push_back(p1);
        if (d1 * d2 < 0)
            qs.push_back(isSS(p1, p2, q1, q2));
    }
    return qs;
}

```

```

typedef double Tdata;
typedef Point Tpoint;

```

```

struct Tline {
    Tdata a, b, c;
    double ang;
    Tline() {}
    Tline(Tdata a, Tdata b, Tdata c) : a(a), b(b), c(c) { ang = atan2(b, -a); }
    void get() { scanf("%lf%lf%lf", &a, &b, &c); }
    bool operator <(Tline l) const { return sign(ang - l.ang) < 0 || sign(ang - l.ang) == 0 && sign
};

inline int side(Tline l, Tpoint p) { return sign(l.a * p.x + l.b * p.y + l.c); }

// change line from two point form to general form
// O(1)
// return line(general form)
inline Tline change_line(Tpoint a, Tpoint b) {
    Tdata tmp, A = a.y - b.y, B = b.x - a.x, C = cross(a, b);
    if (sign(A)) tmp = fabs(A);
    else tmp = fabs(B);
    return Tline(A / tmp, B / tmp, C / tmp);
}

// calculate the area of polygon
// O(N)
// be careful the sign of the area
Tdata cal_area(Tpoint *P, int N) {
    if (N < 3) return 0;
    Tdata ret = 0;
    P[N] = P[0];
    for (int i = 0; i < N; ++i) ret += cross(P[i], P[i + 1]);
    return ret / 2.0;
}

// intersection of half-planes
// O(N log N)
// ax + by + c >= 0
// P - points form the intersection, M - number of points
void inter_hplane(Tline *H, int N, Tpoint *P, int &M) {
    int *queue = new int[N + 1];
    sort(H, H + N);
    M = 0;
    for (int i = 0; i < N; ++i) if (!i || sign(H[i].ang - H[queue[M - 1]].ang)) queue[M++] = i;
    int h = 0, t = 2;
    for (int i = 2; i < M; ++i) {
        while (h + 1 < t && side(H[queue[i]], inter_point(H[queue[t - 1]], H[queue[t - 2]])) < 0) -
        while (h + 1 < t && side(H[queue[i]], inter_point(H[queue[h]], H[queue[h + 1]])) < 0) ++h;
        queue[t++] = queue[i];
    }
}

```

```

while (h + 1 < t && side(H[queue[h]], inter_point(H[queue[t - 1]], H[queue[t - 2]])) < 0) --t;
while (h + 1 < t && side(H[queue[t - 1]], inter_point(H[queue[h]], H[queue[h + 1]])) < 0) ++h;
M = 0;
for (int i = h; i < t; ++i) queue[M++] = queue[i];
queue[M] = queue[0];
for (int i = 0; i < M; ++i) P[i] = inter_point(H[queue[i]], H[queue[i + 1]]);
delete [] queue;
}

// get the core of polygon
// O(N log N)
Tpoint core_of_poly(Tpoint *P, int N) {
    Tline *H = new Tline[N];
    Tpoint *A = new Tpoint[N];
    int M;
    P[N] = P[0];
    for (int i = 0; i < N; ++i) H[i] = change_line(P[i], P[i + 1]);
    inter_hplane(H, N, A, M);
    Tpoint ret = A[0];
    delete [] H; delete [] A;
    return ret;
}

// get the length of segment in convex polygon
// O(N)
Tdata seg_in_convex_poly(Tpoint a, Tpoint b, Tpoint *P, int N) {
    int d1 = point_in_convex_poly(a, P, N), d2 = point_in_convex_poly(b, P, N);
    if (d2 == 1) swap(d1, d2), swap(a, b);
    if (d2 == 1) return dist(a, b);
    Tpoint p;
    P[N] = P[0];
    if (d1 == 1)
        for (int i = 0; i < N; ++i) {
            int d = inter_seg(a, b, P[i], P[i + 1], p);
            if (d == 1 || d == 2) return dist(a, p); // not including the boundaries, add "d == 3" for
        }
    else {
        int cnt = 0;
        Tpoint u, v;
        for (int i = 0; i < N; ++i) {
            int d = inter_seg(a, b, P[i], P[i + 1], p);
            if (d == 3) return 0; // on the boundaries
            if (cnt == 2) continue;
            if (d)
                if (!cnt) u = p, ++cnt;
                else if (u != p) v = p, ++cnt;
        }
        return cnt == 2 ? dist(u, v) : 0;
    }
}

```

```

    }
}

// get the centroid of polygon
// O(N)
Tpoint cal_centroid(Tpoint *P, int N) {
    P[N] = P[0];
    Tpoint c(0, 0);
    Tdata s = 0;
    for (int i = 0; i < N; ++i) {
        Tdata tmp = cross(P[i], P[i + 1]);
        c += (P[i] + P[i + 1]) * tmp; s += tmp;
    }
    return c / (3 * s);
}

```

## 4.2 空间几何基础

### 4.2.1 三维几何

```

#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <algorithm>
#include <cmath>

#define Sqr(x) (x) * (x)

using namespace std;

const double EPS = 1E-8;

inline int sign(double x){
    return x < -EPS ? -1 : x > EPS;
}

inline double frand(){
    return rand() / (RAND_MAX + 1.0);
}

/*
 * -----points & vectors-----
 */
struct Point3{
    double x, y, z;
    Point3(){ }
    Point3(double x, double y, double z) : x(x), y(y), z(z) {}
    void in(){

```



```

        scanf("%lf%lf%lf", &x, &y, &z);
    }
    void print(){
        printf("%lf %lf %lf", x, y, z);
    }
};
typedef Point3 Vector3;

inline Vector3 operator+(Point3 a, Point3 b){
    return Vector3(a.x + b.x, a.y + b.y, a.z + b.z);
}

inline Vector3 operator-(Point3 a, Point3 b){
    return Vector3(a.x - b.x, a.y - b.y, a.z - b.z);
}

inline Vector3 operator*(double t, Vector3 a){
    return Vector3(a.x * t, a.y * t, a.z * t);
}

inline Vector3 operator*(Vector3 a, double t){
    return Vector3(a.x * t, a.y * t, a.z * t);
}

inline Vector3 operator/(Vector3 a, double t){
    return Vector3(a.x / t, a.y / t, a.z / t);
}

inline Vector3 operator*(Vector3 a, Vector3 b){
    return Vector3(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x);
}

inline double operator^(Vector3 a, Vector3 b){
    return a.x * b.x + a.y * b.y + a.z * b.z;
}

inline double len(Vector3 a){
    return sqrt(a ^ a);
}

inline double len2(Vector3 a){
    return a ^ a;
}

inline int zero(Vector3 a){
    return !sign(a.x) && !sign(a.y) && !sign(a.z);
}

```

```

/*
 * -----lines, line segment & planes-----
 */
struct Line{
    Point3 p, q;
    Line(){ }
    Line(Point3 p, Point3 q) : p(p), q(q) {}
    double len2(){
        return (q - p) ^ (q - p);
    }
    double len(){
        return sqrt((q - p) ^ (q - p));
    }
};

/*
 * returns a vector that perps to u
 */
Vector3 perp_vector(Vector3 u){
    Vector3 v, n;
    while(true){
        v.x = frand();
        v.y = frand();
        v.z = frand();
        if (!zero(n = u * v)) return v;
    }
}

/*
 * check if point a inside line l
 */
int on_seg(Line l, Point3 a){
    return zero((a - l.p) * (a - l.q)) && sign((l.p - a) ^ (l.q - a)) <= 0;
}

/*
 * relation of a & b base on l
 * same side : +1;
 * opposite side : -1;
 * otherwise : 0;
 */
inline int side(Line l, Point3 a, Point3 b){
    return sign(((l.p - l.q) * (a - l.p)) ^ ((l.p - l.q) * (b - l.p)));
}

/*
 * intersetion point of plane(norm, A) and lineseg l
 * ret is the result

```

```

*/
int ip_plane_seg(Vector3 norm, Point3 a, Line l, Point3 &ret){
    double lhs = norm ^ (l.q - l.p);
    double rhs = norm ^ (a - l.p);
    double t = rhs / lhs;
    if (sign(t) >= 0 && sign(t- 1) <= 0){
        ret = l.p + t * (l.q - l.p);
        return 1;
    }
    return 0;
}

/*
 * check if 2 line-segments l1 & l2 touched with each other
 */
int touched_segs(Line l1, Line l2){
    if (zero((l1.q - l1.p) * (l2.q - l2.p))){
        return on_seg(l1, l2.p) || on_seg(l1, l2.q) || on_seg(l2, l1.p) || on_seg(l2, l1.q);
    }
    else{
        return side(l1, l2.p, l2.q) <= 0 && side(l2, l1.p, l1.q) <= 0;
    }
}

/*
 * return the projection of point a to line l
 */
Point3 project(Line l, Point3 a){
    double t = ((l.q - l.p) ^ (a - l.p)) / ((l.q - l.p) ^ (l.q - l.p));
    return l.p + t * (l.q - l.p);
}

/*
 * return the closest point in line l
 */
Point3 closest_point_seg(Line l, Point3 a){
    double t = ((l.q - l.p) ^ (a - l.p)) / ((l.q - l.p) ^ (l.q - l.p));
    return l.p + max(0.0, min(t, 1.0)) * (l.q - l.p);
}

/*
 * -----plane-----
 */
struct Plane{
    Point3 a;
    Vector3 n;
};

```

```

/*
 * check if the point in the plane
 */
int on_plane(Plane pl, Point3 p){
    return !sign(pl.n ^ (p - pl.a));
}

/*
 * return the distance between point and the plane
 */
double dist_plane_point(Plane pl, Point3 a){
    return fabs(pl.n ^ (a - pl.a)) / len(pl.n);
}

/*
 * closest point in the plane
 */
Point3 closest_point_plane(Plane pl, Point3 a){
    return a + ((pl.n ^ (pl.a - a)) / (pl.n ^ pl.n)) * pl.n;
}

/*
 * mappint from 3D point to 2D point
 */
Point3 to_plane(Point3 a, Point3 b, Point3 c, Point3 p){
    Vector3 norm, ydir, xdir;
    Point3 res;
    norm = (b - a) * (c - a);
    xdir = b - a;
    xdir = xdir / len(xdir);
    ydir = norm * xdir;
    ydir = ydir / len(ydir);
    res.x = (p - a) ^ xdir;
    res.y = (p - a) ^ ydir;
    res.z = 0;
    return res;
}

/*
 * given two lines in 3D space , find distance of closest approach
 */
double dist_line_line(Line l1, Line l2){
    Vector3 v = (l1.q - l1.p) * (l2.q - l2.p);
    if (zero(v)){
        if (zero((l1.q - l1.p) * (l2.p - l1.p))) return 0;
        return len((l2.p - l1.p) * (l2.q - l1.p)) / len(l2.p - l2.q);
    }
}

```

```

    return fabs((l1.p - l2.p) ^ v) / len(v);
}

/*
 * this is the same as dist_line_line, but it also return s the points of closest approach
 */
double closest_approach(Line l1, Line l2, Point3 &p, Point3 &q){
    double s = (l2.q - l2.p) ^ (l1.q - l1.p);
    double t = (l1.p - l2.p) ^ (l2.q - l2.p);
    double num, den, tmp;
    den = l1.len2() * l2.len2() - s * s;
    num = t * s - l2.len2() * ((l1.p - l2.p) ^ (l1.q - l1.p));
    if (!sign(den)){
        p = l1.p;
        q = l2.p + (l2.q - l2.p) * t / l2.len();
        if (!sign(s)) q = l1.p;
    }
    else{
        tmp = num / den;
        p = l1.p + (l1.q - l1.p) * tmp;
        q = l2.p + (l2.q - l2.p) * (t + s * tmp) / l2.len2();
    }
    return len(p - q);
}

/*
 * -----balls(spheres)-----
 */
struct Ball{
    Point3 o;
    double r;
    Ball(Point3 o = Point3(0, 0, 0), double r = 1) : o(o), r(r) {}
};

/*
 * ip between ball o and line l
 */
int ip_ball_line(Ball o, Line l, Point3 &p, Point3 &q){
    Vector3 v;
    Point3 d = project(l, o.o);
    if (len2(o.o - d) > o.r * o.r) return 0;
    v = sqrt((o.r * o.r - len2(o.o - d)) / l.len2()) * (l.p - l.q);
    p = d + v;
    q = d - v;
    return 1;
}

/*

```

```

* Given the latitude and longitude of two points in degrees
* calculates the distance over the sphere between them.
* Latitude is given in the range  $[-\pi/2, \pi/2]$  degrees,
* Longitude is given in the range  $[-\pi, \pi]$  degrees.
*/
double greatcircle(double lat1, double long1, double lat2, double long2){
    return acos(sin(lat1) * sin(lat2) + cos(lat1) * cos(lat2) * cos(long2 - long1));
}

/*
* Solves the determinant of a n*n matrix recursively
*/
double det(double m[4][4], int n){
    double s[4][4], res = 0, x;
    int i, j, skip, ssize;
    if (n == 2){
        return m[0][0] * m[1][1] - m[0][1] * m[1][0];
    }
    for(skip = 0; skip < n; skip++){
        for(i = 1; i < n; i++){
            for(j = 0, ssize = 0; j < n; j++){
                if (j == skip) continue;
                s[i - 1][ssize++] = m[i][j];
            }
        }
        x = det(s, n - 1);
        if (skip % 2){
            res -= m[0][skip] * x;
        }
        else{
            res += m[0][skip] * x;
        }
    }
    return res;
}

/*
* Given 4 points:
* Returns 0 if the points are coplanar
* Returns 1 if the points are not coplanar with:
*     o = center of sphere
*     r = radius of sphere
*/
int make_sphere(Point3 p[4], Ball o){
    double m[4][5], s[4][4], sol[5];
    int ssize, skip, i, j;
    for(i = 0; i < 4; i++){
        s[i][0] = p[i].x;

```

```

        s[i][1] = p[i].y;
        s[i][2] = p[i].z;
        s[i][3] = 1;
    }
    if (!sign(det(s, 4))) return 0;
    for(i = 0; i < 4; i++){
        m[i][0] = 0;
        m[i][0] += Sqr(m[i][1] = p[i].x);
        m[i][0] += Sqr(m[i][2] = p[i].y);
        m[i][0] += Sqr(m[i][3] = p[i].z);
        m[i][4] = 1;
    }
    for(skip = 0; skip < 5; skip++){
        for(i = 0; i < 4; i++){
            for(j = 0, ssize = 0; j < 5; j++){
                if (j == skip) continue;
                s[i][ssize++] = m[i][j];
            }
        }
        sol[skip] = det(s, 4);
    }
    for(i = 1; i < 5; i++){
        sol[i] /= (sol[0] * ((i % 2) ? 1 : -1));
    }
    for(i = 1; i < 4; i++){
        sol[4] += Sqr(sol[i] /= 2);
    }
    o.o.x = sol[1];
    o.o.y = sol[2];
    o.o.z = sol[3];
    o.r = sqrt(sol[4]);
    return 1;
}

/*
 * -----polygons-----
 */

/*
 * check if point A inside polygon p[n]
 */
int inside_polygon(Point3 *p, int n, Vector3 norm, Point3 A){
    if (sign(norm ^ (A - p[0]))) return 0;
    p[n] = p[0];
    for(int i = 0; i < n; i++){
        if (on_seg(Line(p[i], p[i + 1]), A)) return 1;
    }
    double sum = 0;

```

```

    for(int i = 0; i < n; i++){
        Vector3 a = p[i] - A;
        Vector3 b = p[i + 1] - A;
        sum += sign(norm ^ (a * b)) * acos((a ^ b) / (len(a) * len(b)));
    }
    return sign(sum);
}

/*
 * check if lineseg l touches polygon p[n] with normal vector norm
 */
int intersected_polygon_seg(Point3 *p, int n, Vector3 norm, Line l){
    p[n] = p[0];
    if (!sign((l.p - l.q) ^ norm)){
        if (sign(norm ^ (l.p - p[0]))) return 0;
        if (inside_polygon(p, n, norm, l.p) || inside_polygon(p, n, norm, l.q)) return 1;
        for(int i = 0; i < n; i++){
            if (touched_segs(l, Line(p[i], p[i + 1]))) return 1;
        }
        return 0;
    }
    Point3 ret;
    if (ip_plane_seg(norm, p[0], l, ret)){
        return inside_polygon(p, n, norm, ret);
    }
    return 0;
}

/*
 * normal vector of polygon p[n]
 */
Vector3 normal(Point3 *p, int n){
    Vector3 b, norm;
    p[n] = p[0];
    p[n + 1] = p[1];
    for(int i = 0; i < n; i++){
        norm = (p[i + 1] - p[i + 2]) * (p[i] - p[i + 1]);
        if (!zero(norm)) return norm;
    }
    return perp_vector(p[0] - p[1]);
}

/*
 * check if 2 polygons p[n] & q[m] touched with each other
 */
int touched_polygons(Point3 *p, int n, Point3 *q, int m){
    Vector3 norm;
    norm = normal(q, m);

```



```

    p[n] = p[0];
    for(int i = 0; i < n; i++){
        if (intersected_polygon_seg(q, m, norm, Line(p[i], p[i + 1]))) return 1;
    }
    norm = normal(p, n);
    q[m] = q[0];
    for(int i = 0; i < m; i++){
        if (intersected_polygon_seg(p, n, norm, Line(q[i], q[i + 1]))) return 1;
    }
    return 0;
}

/*
 * new add by myf
 */

struct Plane3{
    Point3 a, b, c;
    Plane3(){}
    Plane3(Point3 a, Point3 b, Point3 c) : a(a), b(b), c(c) {}
};

double triple(Point3 a, Point3 b, Point3 c){
    return a ^ (b * c);
}

double polygon_volume(Plane3 *p, int n){
    double volume = 0.0;
    for(int i = 0; i < n; i++){
        volume += triple(p[i].a, p[i].b, p[i].c);
    }
    return fabs(volume) / 6.0;
}

```

#### 4.2.2 空间变换矩阵

```

const int N = 4;
const int MD = 1000000007;
const int INF = 0x3f3f3f3f;
const double PI = acos(-1.0);
const double EPS = 1E-6;

struct Matrix{
    int n, m;
    double v[N][N];
    Matrix(){
        n = m = 4;
        rep(i, 4) rep(j, 4) v[i][j] = (i == j);
    }
};

```

```

    }
    Matrix(int n, int m) : n(n), m(m){
        rep(i, n) rep(j, m) v[i][j] = 0;
    }
};

int n;
Matrix ret;
char s[11];

Matrix operator * (Matrix a, Matrix b){
    Matrix c(a.n, b.m);
    rep(i, c.n){
        rep(j, c.m){
            rep(k, a.m){
                c.v[i][j] += a.v[i][k] * b.v[k][j];
            }
        }
    }
    return c;
}

Matrix translate(){
    Matrix ret;
    double x;
    rep(i, 3){
        scanf("%lf", &x);
        ret.v[i][3] += x;
    }
    return ret;
}

Matrix scale(){
    Matrix ret;
    double x;
    rep(i, 3){
        scanf("%lf", &x);
        ret.v[i][i] *= x;
    }
    return ret;
}

Matrix rotate(){
    Matrix ret;
    double x, y, z, d;
    scanf("%lf%lf%lf%lf", &x, &y, &z, &d);
    double len = sqrt(Sqr(x) + Sqr(y) + Sqr(z));
    x /= len; y /= len; z /= len;

```

```

d = d * PI / 180.0;
ret.v[0][0] = (1 - cos(d)) * x * x + cos(d);
ret.v[0][1] = (1 - cos(d)) * x * y - sin(d) * z;
ret.v[0][2] = (1 - cos(d)) * x * z + sin(d) * y;

ret.v[1][0] = (1 - cos(d)) * y * x + sin(d) * z;
ret.v[1][1] = (1 - cos(d)) * y * y + cos(d);
ret.v[1][2] = (1 - cos(d)) * y * z - sin(d) * x;

ret.v[2][0] = (1 - cos(d)) * z * x - sin(d) * y;
ret.v[2][1] = (1 - cos(d)) * z * y + sin(d) * x;
ret.v[2][2] = (1 - cos(d)) * z * z + cos(d);

return ret;
}

Matrix pow(Matrix now, int n){
    Matrix ret;
    while(n){
        if (n & 1) ret = now * ret;
        now = now * now;
        n >>= 1;
    }
    return ret;
}

Matrix dfs(int lev){
    Matrix now, tmp;
    while(true){
        scanf("%s", s);
        if (s[1] == 'r'){ // translate
            tmp = translate();
        }
        else if (s[1] == 'c'){ // scale
            tmp = scale();
        }
        else if (s[1] == 'o'){ // rotate
            tmp = rotate();
        }
        else if (s[1] == 'e'){ // repeat
            int k;
            scanf("%d", &k);
            tmp = dfs(lev + 1);
            tmp = pow(tmp, k);
        }
        else if (s[1] == 'n'){ // end
            break;
        }
    }
}

```

```

        now = tmp * now;
    }
    return now;
}

void solve(){
    Matrix now;
    rep(i, n){
        now.n = 4, now.m = 1;
        rep(j, 3) scanf("%lf", &now.v[j][0]);
        now.v[3][0] = 1;
        now = ret * now;
        rep(i, 3) if (fabs(now.v[i][0]) < EPS) now.v[i][0] = 0;
        printf("%.2f %.2f %.2f\n", now.v[0][0], now.v[1][0], now.v[2][0]);
    }
}

int main(){
    while(~scanf("%d", &n)){
        if (!n) break;
        ret = dfs(0);
        solve();
        puts("");
    }
    return 0;
}

```

## 4.3 凸包

### 4.3.1 凸包

```

// find the convex hull
Point __o;

bool cmp_p(Point a, Point b){
    int f = sign(a.X - b.X);
    if (f) return f < 0;
    return sign(a.Y - b.Y) < 0;
}

bool cmp(Point a, Point b){
    int f = sign(cross(o, a, b));
    if (f) return f > 0;
    return sign(abs(a - o) - abs(b - o)) < 0;
}

Point stack[1111]

int find_convex(Point p[], int n){

```

```

    __o = *min_element(p, p + n, cmp_p);
    sort(p, p + n, cmp);
    int top = 0;
    rep(i, n){
        while(top >= 2 && sign(cross(stack[top - 2], stack[top - 1], p[i])) <= 0) top--;
        stack[top++] = p[i];
    }
    rep(i, top) p[i] = stack[i];
    return top;
}

// -----intersection points convex hull-----
bool lcmp(Line u, Line v){
    int c = sign((u.p.x - u.q.x) * (v.p.y - v.q.y) - (v.p.x - v.q.x) * (u.p.y - u.q.y));
    return c < 0 || !c && sign(cross(u.p, u.q, v.p)) < 0;
}

/*
 * XXX sizeof(p) MUST be as large as n * 2
 * return # of points of resulting convex hull
 */
int ip_convex(Line l[], int n, Point p[]){
    for(int i = 0; i < n; i++){
        if (l[i].q < l[i].p) swap(l[i].p, l[i].q);
    }
    sort(l, l + n, lcmp);
    int n1 = 0;
    for(int i = 0, j = 0; i < n; i = j){
        while(j < n && parallel(l[i], l[j])) j++;
        if (j - i == 1){
            l[n1++] = l[i];
        }
        else{
            l[n1++] = l[i];
            l[n1++] = l[j - 1];
        }
    }
    n = n1;
    l[n + 0] = l[0];
    l[n + 1] = l[1];
    int m = 0;
    for(int i = 0, j = 0; i < n; i++){
        while(j < n + 2 && parallel(l[i], l[j])) j++;
        for(int k = j; k < n + 2 && parallel(l[j], l[k]); k++){
            p[m++] = ip(l[i], l[k]);
        }
    }
    return find_convex(p, m);
}

```

```

}

typedef double Tdata;
typedef Point Tpoint;
// get the diameter of convex polygon
// O(N)
// p1, p2 are the points forming diameter
Tdata diam_convex_poly(Tpoint *P, int N, Tpoint &p1, Tpoint &p2) {
    if (N == 1) {
        p1 = p2 = P[0];
        return 0;
    }
    double ret = -INF;
    for (int j = 1, i = 0; i < N; ++i) {
        while (sign(cross(P[i], P[i + 1], P[j + 1]) - cross(P[i], P[i + 1], P[j]))) > 0) j = (j + 1) % N;
        ret = max(ret, max(dist2(P[i], P[j]), dist2(P[i + 1], P[j + 1])));
    }
    return ret;
}

```

#### 4.3.2 三维凸包 n\*n

```

typedef double Tdata;

const int MAXN = 1000 + 10;
const int MAXF = MAXN * 6;
const double EPS = 1E-6;

inline int sign(Tdata x) { return x < -EPS ? -1 : x > EPS ? 1 : 0; }

struct Tpoint {
    Tdata x, y, z;

    Tpoint() {}
    Tpoint(Tdata x, Tdata y, Tdata z) : x(x), y(y), z(z) {}
    void get() { scanf("%lf%lf%lf", &x, &y, &z); }
    bool operator <(Tpoint p) const {
        int s = sign(x - p.x); if (s) return s < 0;
        s = sign(y - p.y); if (s) return s < 0;
        return sign(z - p.z) < 0;
    }
    bool operator ==(Tpoint p) const { return !sign(x - p.x) && !sign(y - p.y) && !sign(z - p.z); }
    void operator -=(Tpoint p) { x -= p.x; y -= p.y; z -= p.z; }
    void operator +=(Tpoint p) { x += p.x; y += p.y; z += p.z; }
    void operator *=(Tdata c) { x *= c; y *= c; z *= c; }
    void operator /=(Tdata c) { x /= c; y /= c; z /= c; }
    Tpoint operator +(Tpoint p) const { return Tpoint(x + p.x, y + p.y, z + p.z); }
    Tpoint operator -(Tpoint p) const { return Tpoint(x - p.x, y - p.y, z - p.z); }
}

```

```

    Tpoint operator *(Tdata c) const { return Tpoint(x * c, y * c, z * c); }
    Tpoint operator /(Tdata c) const { return Tpoint(x / c, y / c, z / c); }
};

inline Tdata sqr(Tdata x) { return x * x; }

inline Tdata norm2(Tpoint p) { return sqr(p.x) + sqr(p.y) + sqr(p.z); }

inline Tdata norm(Tpoint p) { return sqrt(norm2(p)); }

inline Tpoint cross(Tpoint a, Tpoint b) { return Tpoint(a.y * b.z - b.y * a.z, a.z * b.x - b.z * a.x, a.x * b.y - b.x * a.y); }

inline Tpoint cross(Tpoint o, Tpoint a, Tpoint b) { return cross(a - o, b - o); }

inline Tdata det(Tpoint a, Tpoint b, Tpoint c) {
    #define D2(a, b, x, y) (a.x * b.y - a.y * b.x)
    return a.x * D2(b, c, y, z) - a.y * D2(b, c, x, z) + a.z * D2(b, c, x, y);
    #undef D2
}

inline Tdata dot(Tpoint a, Tpoint b) { return a.x * b.x + a.y * b.y + a.z * b.z; }

inline double volume(Tpoint p, Tpoint a, Tpoint b, Tpoint c) { return det(a - p, b - p, c - p); }

struct Chull3D {
    Tpoint P[MAXN];
    int face[MAXF][3];
    int del[MAXF];
    int lnk[MAXN][MAXN];
    bool used[MAXN];
    int N, F, face_num;
    Tdata vol, area;
    Tpoint cen;

    inline int vol_sgn(int o, int a, int b, int c) {
        Tdata v = volume(P[o], P[a], P[b], P[c]);
        return sign(v);
    }

    inline void add_face(int a, int b, int c) {
        face[F][0] = a; face[F][1] = b; face[F][2] = c; del[F] = 0;
        lnk[a][b] = lnk[b][c] = lnk[c][a] = F++;
    }

    inline bool can_see(int p, int f) { return vol_sgn(p, face[f][0], face[f][1], face[f][2]) < 0; }

    //return 0 if all in one plane or line
    bool find_tet() {

```

```

    for (int i = 1; i < N; ++i) if (P[i].x < P[0].x) swap(P[i], P[0]);
    for (int i = 2; i < N; ++i) if (P[i].x > P[1].x) swap(P[i], P[1]);
    for (int i = 3; i < N; ++i)
        if (fabs(norm2(cross(P[0], P[1], P[i]))) > fabs(norm2(cross(P[0], P[1], P[2])))) swap(P[i], P[2]);
    if (cross(P[0], P[1], P[2]) == Tpoint(0, 0, 0)) return 0;
    for (int i = 4; i < N; ++i)
        if (fabs(volume(P[0], P[1], P[2], P[i])) > fabs(volume(P[0], P[1], P[2], P[3]))) swap(P[i], P[3]);
    if (!vol_sgn(0, 1, 2, 3)) return 0;
    for (int i = 0; i < 4; ++i) {
        int a = (i + 1) % 4, b = (i + 2) % 4, c = (i + 3) % 4;
        if (vol_sgn(i, a, b, c) < 0) swap(b, c);
        add_face(a, b, c);
    }
    return 1;
}

void add(int p, int f) {
    if (del[f]) return;
    del[f] = 1;
    for (int i = 0; i < 3; ++i) {
        int opp = lnk[face[f][(i + 1) % 3]][face[f][i]];
        if (!del[opp]) {
            if (can_see(p, opp)) add(p, opp);
            else add_face(face[f][i], face[f][(i + 1) % 3], p);
        }
    }
}

bool coplanar(int f1, int f2, int p1, int p2) {
    int vs[4], m = 0;
    for (int i = 0; i < 3; ++i) {
        int v = face[f1][i];
        if (v != p1 && v != p2) vs[m++] = v;
    }
    for (int i = 0; i < 3; ++i) vs[m++] = face[f2][i];
    return vol_sgn(vs[0], vs[1], vs[2], vs[3]) == 0;
}

int cal_face() {
    int E = 0, V = 0;
    memset(used, 0, sizeof(used));
    for (int i = 0; i < F; ++i)
        if (!del[i])
            for (int j = 0; j < 3; ++j) {
                int k = lnk[face[i][(j + 1) % 3]][face[i][j]];
                if (!del[k] && !coplanar(i, k, face[i][j], face[i][(j + 1) % 3])) ++E, used[face[i][j]] = 1;
            }
    for (int i = 0; i < N; ++i) if (used[i]) ++V;
}

```



```

    return 2 + E / 2 - V;
}

double cal_volume() {
    double ret = 0;
    for (int i = 0; i < F; ++i)
        if (!del[i]) {
            Tpoint a = P[face[i][0]], b = P[face[i][1]], c = P[face[i][2]];
            ret += volume(Tpoint(0, 0, 0), a, b, c);
        }
    return fabs(ret) / 6.0;
}

double cal_area() {
    double ret = 0;
    for (int i = 0; i < F; ++i)
        if (!del[i]) {
            Tpoint a = P[face[i][0]], b = P[face[i][1]], c = P[face[i][2]];
            ret += fabs(norm(cross(a, b, c)) / 2.0);
        }
    return ret;
}

Tpoint cal_centroid() {
    Tpoint ret = Tpoint(0, 0, 0);
    for (int i = 0; i < F; ++i)
        if (!del[i]) {
            Tpoint a = P[face[i][0]], b = P[face[i][1]], c = P[face[i][2]];
            ret += (a + b + c) * volume(Tpoint(0, 0, 0), a, b, c);
        }
    return ret / cal_volume() / 24.0;
}

void get() {
    scanf("%d", &N);
    for (int i = 0; i < N; ++i) P[i].get();
    sort(P, P + N);
    N = unique(P, P + N) - P; F = 0;
    vol = area = 0;
    memset(del, 0, sizeof(del));
    if (!find_tet()) return;
    random_shuffle(P + 4, P + N);
    for (int i = 4; i < N; ++i)
        for (int j = 0; j < F; ++j)
            if (!del[j] && can_see(i, j)) {
                add(i, j);
                break;
            }
}

```

```

        face_num = cal_face(); vol = cal_volume(); area = cal_area(); cen = cal_centroid();
    }
};

```

### 4.3.3 三维凸包 $n \cdot \log n$

```
typedef double Tdata;
```

```

const int MAXN = 1000 + 10;
const int MAXF = MAXN * 6;
const int MAXM = MAXN * 12;
const double EPS = 1E-6;

```

```
inline int sign(Tdata x) { return x < -EPS ? -1 : x > EPS ? 1 : 0; }
```

```

struct Tpoint {
    Tdata x, y, z;

```

```
    Tpoint() {}
```

```
    Tpoint(Tdata x, Tdata y, Tdata z) : x(x), y(y), z(z) {}
```

```
    void get() { scanf("%lf%lf%lf", &x, &y, &z); }
```

```

    bool operator <(Tpoint p) const {
        int s = sign(x - p.x); if (s) return s < 0;
        s = sign(y - p.y); if (s) return s < 0;
        return sign(z - p.z) < 0;
    }

```

```
    bool operator ==(Tpoint p) const { return !sign(x - p.x) && !sign(y - p.y) && !sign(z - p.z); }
```

```
    void operator -=(Tpoint p) { x -= p.x; y -= p.y; z -= p.z; }
```

```
    void operator +=(Tpoint p) { x += p.x; y += p.y; z += p.z; }
```

```
    void operator *=(Tdata c) { x *= c; y *= c; z *= c; }
```

```
    void operator /=(Tdata c) { x /= c; y /= c; z /= c; }
```

```
    Tpoint operator +(Tpoint p) const { return Tpoint(x + p.x, y + p.y, z + p.z); }
```

```
    Tpoint operator -(Tpoint p) const { return Tpoint(x - p.x, y - p.y, z - p.z); }
```

```
    Tpoint operator *(Tdata c) const { return Tpoint(x * c, y * c, z * c); }
```

```
    Tpoint operator /(Tdata c) const { return Tpoint(x / c, y / c, z / c); }
```

```
};
```

```
inline Tdata sqr(Tdata x) { return x * x; }
```

```
inline Tdata norm2(Tpoint p) { return sqr(p.x) + sqr(p.y) + sqr(p.z); }
```

```
inline Tdata norm(Tpoint p) { return sqrt(norm2(p)); }
```

```
inline Tpoint cross(Tpoint a, Tpoint b) { return Tpoint(a.y * b.z - b.y * a.z, a.z * b.x - b.z * a.x, a.x * b.y - b.x * a.y); }
```

```
inline Tpoint cross(Tpoint o, Tpoint a, Tpoint b) { return cross(a - o, b - o); }
```

```
inline Tdata det(Tpoint a, Tpoint b, Tpoint c) {
```

```

#define D2(a, b, x, y) (a.x * b.y - a.y * b.x)
return a.x * D2(b, c, y, z) - a.y * D2(b, c, x, z) + a.z * D2(b, c, x, y);
#undef D2
}

inline Tdata dot(Tpoint a, Tpoint b) { return a.x * b.x + a.y * b.y + a.z * b.z; }

inline double volume(Tpoint p, Tpoint a, Tpoint b, Tpoint c) { return det(a - p, b - p, c - p); }

struct Tedge {
    int v;
    Tedge *prev, *next, *opp;
    Tedge() {}
    Tedge(int v, Tedge *prev, Tedge *next, Tedge *opp) : v(v), prev(prev), next(next), opp(opp) {}
};

struct Chull3D {
    Tpoint P[MAXN];
    int face[MAXF][3];
    Tedge mem[MAXM], *elist[MAXM];
    Tedge *Fcon[MAXN];
    Tedge *Pcon[MAXF];
    int del[MAXF];
    int mark[MAXN];
    map<int, int> lnk[MAXN];
    int N, F, nfree, col, face_num;
    Tdata vol, area;
    Tpoint cen;

    void alloc_memory() {
        nfree = 12 * N;
        Tedge *e = mem;
        for (int i = 0; i < nfree; ++i) elist[i] = e++;
    }

    inline int vol_sgn(int o, int a, int b, int c) {
        Tdata v = volume(P[o], P[a], P[b], P[c]);
        return sign(v);
    }

    inline void add_face(int a, int b, int c) {
        face[F][0] = a; face[F][1] = b; face[F][2] = c; del[F] = 0; Pcon[F] = NULL;
        lnk[a][b] = lnk[b][c] = lnk[c][a] = F++;
    }

    inline void add_edge(int i, int j) {
        Tedge *a = elist[--nfree], *b = elist[--nfree];
        *a = Tedge(j, NULL, Fcon[i], b); *b = Tedge(i, NULL, Pcon[j], a);
    }
};

```

```

    if (Fcon[i] != NULL) Fcon[i]->prev = a;
    Fcon[i] = a;
    if (Pcon[j] != NULL) Pcon[j]->prev = b;
    Pcon[j] = b;
}

```

```

inline bool can_see(int p, int f) { return vol_sgn(p, face[f][0], face[f][1], face[f][2]) < 0; }

```

```

//return 0 if all in one plane or line
bool find_tet() {
    for (int i = 1; i < N; ++i) if (P[i].x < P[0].x) swap(P[i], P[0]);
    for (int i = 2; i < N; ++i) if (P[i].x > P[1].x) swap(P[i], P[1]);
    for (int i = 3; i < N; ++i)
        if (fabs(norm2(cross(P[0], P[1], P[i]))) > fabs(norm2(cross(P[0], P[1], P[2])))) swap(P[i], P[2]);
    if (cross(P[0], P[1], P[2]) == Tpoint(0, 0, 0)) return 0;
    for (int i = 4; i < N; ++i)
        if (fabs(volume(P[0], P[1], P[2], P[i])) > fabs(volume(P[0], P[1], P[2], P[3]))) swap(P[i], P[3]);
    if (!vol_sgn(0, 1, 2, 3)) return 0;

    for (int i = 0; i < 4; ++i) {
        int a = (i + 1) % 4, b = (i + 2) % 4, c = (i + 3) % 4;
        if (vol_sgn(i, a, b, c) < 0) swap(b, c);
        add_face(a, b, c);
    }
    return 1;
}

```

```

void update(int f1, int f2) {
    for (Tedge *l = Pcon[f1]; l != NULL; l = l->next) {
        int v = l->v;
        if (mark[v] != col && can_see(v, f2)) {
            mark[v] = col;
            add_edge(v, f2);
        }
    }
}

```

```

bool coplanar(int f1, int f2, int p1, int p2) {
    int vs[4], m = 0;
    for (int i = 0; i < 3; ++i) {
        int v = face[f1][i];
        if (v != p1 && v != p2) vs[m++] = v;
    }
    for (int i = 0; i < 3; ++i) vs[m++] = face[f2][i];
    return vol_sgn(vs[0], vs[1], vs[2], vs[3]) == 0;
}

```

```

int cal_face() {

```

```

    int E = 0, V = 0;
    memset(mark, 0, sizeof(mark));
    for (int i = 0; i < F; ++i)
        if (!del[i])
            for (int j = 0; j < 3; ++j) {
                int k = lnk[face[i][(j + 1) % 3]][face[i][j]];
                if (!del[k] && !coplanar(i, k, face[i][j], face[i][(j + 1) % 3])) ++E, mark[face[i][j]] = 1;
            }
    for (int i = 0; i < N; ++i) if (mark[i]) ++V;
    return 2 + E / 2 - V;
}

double cal_volume() {
    double ret = 0;
    for (int i = 0; i < F; ++i)
        if (!del[i]) {
            Tpoint a = P[face[i][0]], b = P[face[i][1]], c = P[face[i][2]];
            ret += volume(Tpoint(0, 0, 0), a, b, c);
        }
    return fabs(ret) / 6.0;
}

double cal_area() {
    double ret = 0;
    for (int i = 0; i < F; ++i)
        if (!del[i]) {
            Tpoint a = P[face[i][0]], b = P[face[i][1]], c = P[face[i][2]];
            ret += fabs(norm(cross(a, b, c)) / 2.0);
        }
    return ret;
}

Tpoint cal_centroid() {
    Tpoint ret = Tpoint(0, 0, 0);
    for (int i = 0; i < F; ++i)
        if (!del[i]) {
            Tpoint a = P[face[i][0]], b = P[face[i][1]], c = P[face[i][2]];
            ret += (a + b + c) * volume(Tpoint(0, 0, 0), a, b, c);
        }
    return ret / cal_volume() / 24.0;
}

void get() {
    scanf("%d", &N);
    for (int i = 0; i < N; ++i) P[i].get();
    sort(P, P + N);
    N = unique(P, P + N) - P; F = 0;
}

```

```

alloc_memory();
vol = 0; area = 0;
memset(del, 0, sizeof(del));
for (int i = 0; i < N; ++i) lnk[i].clear();
if (!find_tet()) return;

random_shuffle(P + 4, P + N);
for (int i = 0; i < N; ++i) Fcon[i] = NULL;
for (int i = 4; i < N; ++i)
    for (int j = 0; j < F; ++j)
        if (can_see(i, j)) add_edge(i, j);

col = 0;
int flag = 0;
memset(mark, 0, sizeof(mark));
for (int i = 4; i < N; ++i) {
    ++flag;
    for (Tedge *j = Fcon[i]; j != NULL; j = j->next) if (!del[j->v]) del[j->v] = flag;
    for (Tedge *next, *j = Fcon[i]; j != NULL; j = next) {
        int u = j->v;
        next = j->next;
        Tedge *p = j->opp->prev, *n = j->opp->next;
        if (p != NULL) p->next = n;
        else Pcon[u] = n;
        if (n != NULL) n->prev = p;
        elist[nfree++] = j; elist[nfree++] = j->opp;
        for (int k = 0; k < 3; ++k) {
            int v = lnk[face[u][(k + 1) % 3]][face[u][k]];
            if (!del[v]) {
                add_face(face[u][k], face[u][(k + 1) % 3], i);
                ++col;
                update(u, F - 1); update(v, F - 1);
            }
        }
    }
    for (Tedge *next, *l = Pcon[u]; l != NULL; l = next) {
        next = l->next;
        Tedge *p = l->opp->prev, *n = l->opp->next;
        if (p != NULL) p->next = n;
        else Fcon[l->v] = n;
        if (n != NULL) n->prev = p;
        elist[nfree++] = l; elist[nfree++] = l->opp;
    }
}
}
face_num = cal_face(); vol = cal_volume(); area = cal_area(); cen = cal_centroid();
};

```

#### 4.3.4 动态凸包

```
//By Lin
#include<cstdio>
#include<cstring>
#include<map>
#define mp(x,y) make_pair(x,y)
#define foreach(i,n) for(__typeof(n.begin()) i = n.begin(); i!=n.end(); i++)
#define X first
#define Y second
using namespace std;
typedef long long LL;
typedef pair<int,int> pii;

map<int,int> up[2];
map<int,int>::iterator iter,p,q;

int strcmp( pii a, pii b, pii c){
    LL ret = ((LL)b.X-a.X)*((LL)c.Y-a.Y)-((LL)b.Y-a.Y)*((LL)c.X-a.X);
    return ret>0?1:(ret==0?0:-1);
}

bool pan( map<int,int> &g ,int x,int y){
    if ( g.size() == 0 ) return false;
    if ( g.find(x) != g.end() ) return y>=g[x];
    if ( g.begin()->X > x || (--g.end()->X < x ) return false;
    iter = g.lower_bound(x);
    p = q = iter;
    p--;
    return strcmp(*p,*q,mp(x,y))>=0;
}

void insert( map<int,int> &g, int x,int y){
    if ( pan(g,x,y) ) return;
    g[x] = y;
    iter = g.find(x);
    while ( iter != g.begin() ){
        p = iter;
        p--;
        if ( p == g.begin() ) break;
        q = p;
        q--;
        if ( strcmp(*q,*iter,*p)>=0 ) g.erase(p);
        else break;
    }
    iter = g.find(x);
    while ( true ){
        p = iter;
```

```

        p++;
        if ( p == g.end() ) break;
        q = p;
        q++;
        if ( q == g.end() ) break;
        if ( strcmp(*iter,*q,*p)>=0 ) g.erase(p);
        else break;
    }
}

int main(){
    int cas;
    scanf("%d", &cas );
    while ( cas -- ){
        int k,x,y
        scanf("%d%d%d", &k, &x, &y );
        if ( k == 1 )
            insert( up[0], x,y ),
            insert( up[1], x,-y);
        else
            printf( pan(up[0],x,y)&&pan(up[1],x,-y)?"YES\n":"NO\n" );
    }
    return 0;
}

```

#### 4.3.5 两凸包间最短距离

```

const int maxn = 10000 + 10;
const double PI = acos(-1.0);
const double EPS = 1E-6;

struct Tpoint {
    double x, y;
};

Tpoint a[maxn], b[maxn];
int n, m;

inline double cross(double X1, double Y1, double X2, double Y2) {
    return X1 * Y2 - X2 * Y1;
}

double Area(Tpoint *a, int n) {
    double ret = 0;
    a[n] = a[0];
    for (int i = 0; i < n; ++i) ret += cross(a[i].x, a[i].y, a[i + 1].x, a[i + 1].y);
    return ret;
}

```



```

inline double Dist(Tpoint A, Tpoint B) {
    return sqrt((A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y));
}

inline double DistP2S(Tpoint P, Tpoint A, Tpoint B) {
    if ((B.x - A.x) * (P.x - A.x) + (B.y - A.y) * (P.y - A.y) < 0) return Dist(P, A);
    if ((A.x - B.x) * (P.x - B.x) + (A.y - B.y) * (P.y - B.y) < 0) return Dist(P, B);
    return fabs(cross(P.x - A.x, P.y - A.y, P.x - B.x, P.y - B.y)) / Dist(A, B);
}

double MinDist() {
    if (Area(a, n) < 0) reverse(a, a + n);
    if (Area(b, m) < 0) reverse(b, b + m);

    int p1 = 0, p2 = 0;
    for (int i = 0; i < n; ++i)
        if (a[i].x < a[p1].x) p1 = i;
    for (int i = 0; i < m; ++i)
        if (b[i].x > b[p2].x) p2 = i;

    int cnt = 0;
    double ret = dist(a[p1], b[p2]);
    while (cnt < n) {
        ret = min(ret, DistP2S(a[p1], b[p2], b[(p2 + 1) % m]));
        ret = min(ret, DistP2S(a[(p1 + 1) % n], b[p2], b[(p2 + 1) % m]));
        ret = min(ret, DistP2S(b[p2], a[p1], a[(p1 + 1) % n]));
        ret = min(ret, DistP2S(b[(p2 + 1) % m], a[p1], a[(p1 + 1) % n]));

        if (cross(a[(p1 + 1) % n].x - a[p1].x, a[(p1 + 1) % n].y - a[p1].y, b[p2].x - b[(p2 + 1) % m].x, b[p2].y - b[(p2 + 1) % m].y) < 0)
            else p2 = (p2 + 1) % m;
    }

    return ret;
}

```

## 4.4 平面

### 4.4.1 半平面交

```

#include <cstdio>
#include <iostream>
#include <algorithm>
#include <climits>
#include <cstring>
#include <cmath>
#define foreach(e,x) for(__typeof(x.begin()) e=x.begin();e!=x.end();++e)
using namespace std;

```

```

struct Point {
    long double x, y;
    Point() {
    }
    Point(long double _x, long double _y) :
        x(_x), y(_y) {
    }
    Point operator+(const Point&p) const {
        return Point(x + p.x, y + p.y);
    }
    Point operator-(const Point&p) const {
        return Point(x - p.x, y - p.y);
    }
    Point operator*(long double d) const {
        return Point(x * d, y * d);
    }
    Point operator/(long double d) const {
        return Point(x / d, y / d);
    }
    long double det(const Point&p) const {
        return x * p.y - y * p.x;
    }
    long double dot(const Point&p) const {
        return x * p.x + y * p.y;
    }
    Point rot90() const {
        return Point(-y, x);
    }
    void read() {
        cin >> x >> y;
    }
    void write() const {
        printf("%lf %lf", x, y);
    }
};

#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))

const long double EPS = 1e-12;
inline int sign(long double a) {
    return a < -EPS ? -1 : a > EPS;
}

#define crossOp(p1,p2,p3) (sign(cross(p1,p2,p3)))

Point isSS(Point p1, Point p2, Point q1, Point q2) {
    long double a1 = cross(q1,q2,p1), a2 = -cross(q1,q2,p2);
    return (p1 * a2 + p2 * a1) / (a1 + a2);
}

```

```

}

struct Border {
    Point p1, p2;
    long double alpha;
    void setAlpha() {
        alpha = atan2(p2.y - p1.y, p2.x - p1.x);
    }
    void read() {
        p1.read();
        p2.read();
        setAlpha();
    }
};

int n;
const int MAX_N_BORDER = 20000 + 10;
Border border[MAX_N_BORDER];

bool operator<(const Border&a, const Border&b) {
    int c = sign(a.alpha - b.alpha);
    if (c != 0)
        return c == 1;
    return crossOp(b.p1, b.p2, a.p1) >= 0;
}

bool operator==(const Border&a, const Border&b) {
    return sign(a.alpha - b.alpha) == 0;
}

const long double LARGE = 10000;

void add(long double x, long double y, long double nx, long double ny) {
    border[n].p1 = Point(x, y);
    border[n].p2 = Point(nx, ny);
    border[n].setAlpha();
    n++;
}

Point isBorder(const Border&a, const Border&b) {
    return isSS(a.p1, a.p2, b.p1, b.p2);
}

Border que[MAX_N_BORDER];
int qh, qt;

bool check(const Border&a, const Border&b, const Border&me) {
    Point is = isBorder(a, b);

```

```

    return crossOp(me.p1,me.p2,is) > 0;
}

void convexIntersection() {
    qh = qt = 0;
    sort(border, border + n);
    n = unique(border, border + n) - border;
    for (int i = 0; i < n; ++i) {
        Border cur = border[i];
        while (qh + 1 < qt && !check(que[qt - 2], que[qt - 1], cur))
            --qt;
        while (qh + 1 < qt && !check(que[qh], que[qh + 1], cur))
            ++qh;
        que[qt++] = cur;
    }
    while (qh + 1 < qt && !check(que[qt - 2], que[qt - 1], que[qh]))
        --qt;
    while (qh + 1 < qt && !check(que[qh], que[qh + 1], que[qt - 1]))
        ++qh;
}

void calcArea() {
    static Point ps[MAX_N_BORDER];
    int cnt = 0;
    if (qt - qh <= 2) {
        puts("0.0");
        return;
    }
    for (int i = qh; i < qt; ++i) {
        int next = i + 1 == qt ? qh : i + 1;
        ps[cnt++] = isBorder(que[i], que[next]);
    }
    long double area = 0;
    for (int i = 0; i < cnt; ++i) {
        area += ps[i].det(ps[(i + 1) % cnt]);
    }
    area /= 2;
    area = fabsl(area);
    cout.setf(ios::fixed);
    cout.precision(1);
    cout << area << endl;
}

int main() {
    cin >> n;
    for (int i = 0; i < n; ++i) {
        border[i].read();
    }
}

```

```

    add(0, 0, LARGE, 0);
    add(LARGE, 0, LARGE, LARGE);
    add(LARGE, LARGE, 0, LARGE);
    add(0, LARGE, 0, 0);
    convexIntersection();
    calcArea();
}

```

#### 4.4.2 动态半平面交

```

#include <cstdio>
#include <set>
#include <map>

using namespace std;

const int N = 100000 + 10;
const int inf = 100001;
const double eps = 1e-10;

set<int> S;
int B[N];
double xval[N];
map<double, int> X;

void cal(int a1, int b1, int a2, int b2, double& x, double& y) {
    if (a2 == -inf) {
        x = 0;
        y = b1;
    }
    else {
        x = (b2 - b1) / double(a1 - a2);
        y = a1 * x + b1;
    }
}

int main() {
    int n, a, b, i;
    double x, y;
    char cmd[2];
    set<int>::iterator itr;
    S.insert(-inf);
    S.insert(0);
    B[inf] = 0;
    xval[inf] = inf;
    B[0] = 0;
    X[0] = 0;
    xval[0] = 0;
}

```

```

scanf("%d", &n);
for (i=1; i<=n; i++) {
    scanf("%s", cmd);
    if (cmd[0] == 'S') {
        scanf("%d %d", &a, &b);
        itr = S.lower_bound(a);
        if (*itr == a) {
            if (B[*itr] >= b) continue; else B[*itr] = b;
        }
        else {
            itr--;
            cal(a, b, *itr, B[*itr], x, y);
            itr++;
            if (y < *itr * x + B[*itr] + eps) continue;
            itr = S.insert(a).first;
            B[*itr] = b;
        }
        itr--;
        while (itr != S.begin() && *itr * xval[*itr] + B[*itr] < a * xval[*itr] + b + eps) {
            x = xval[*itr];
            X.erase(x);
            S.erase(itr--);
        }
        cal(a, b, *itr, B[*itr], x, y);
        itr = S.find(a);
        itr++;
        X.erase(xval[*itr]);
        xval[-a] = x;
        X[x] = a;
        itr++;
        while (itr != S.end() && *itr * xval[*itr] + B[*itr] < a * xval[*itr] + b + eps) {
            itr--;
            S.erase(itr++);
            x = xval[*itr];
            X.erase(x);
            itr++;
        }
        itr--;
        cal(a, b, *itr, B[*itr], x, y);
        xval[*itr] = x;
        X[x] = *itr;
    }
    else {
        scanf("%lf", &x);
        x = x * x;
        map<double, int>::iterator itr;
        if (x == 0) itr = X.lower_bound(0);
        else {

```

```

        itr = X.lower_bound(x);
        itr--;
    }
    printf("%.0lf\n", itr->second * x + B[-itr->second]);
}
}
return 0;
}

```

#### 4.4.3 旋转卡壳

```

double fix(double a, double b = 0) {
    a -= b;
    if (sign(a) < 0) a += 2 * pi;
    if (sign(a - 2 * pi) >= 0) a -= 2 * pi;
    return a;
}

double angle(Point a, Point b){
    return fix(arg(b - a));
}

double shadow_length(double alpha, Point a, Point b){
    double dx = a.x - b.x;
    double dy = a.y - b.y;
    double c = cos(alpha);
    double s = sin(alpha);
    return fabs(dx * c + dy * s);
}

void rotate_calipers(Point ps[], int n, double &area, double &peri){
    area = peri = INF;
    n = find_convex(ps, n);
    ps[n] = ps[0];
    Point *q[4] = {NULL, NULL, NULL, NULL};
    for(int i = 0; i < n; i++){
        Point *p = &ps[i];
        if (!q[0] || q[0]->Y > p->Y || q[0]->Y == p->Y && q[0]->X > p->X) q[0] = p;
        if (!q[1] || q[1]->X < p->X || q[1]->X == p->X && q[1]->Y > p->Y) q[1] = p;
        if (!q[2] || q[2]->Y < p->Y || q[2]->Y == p->Y && q[2]->X < p->X) q[2] = p;
        if (!q[3] || q[3]->X > p->X || q[3]->X == p->X && q[3]->Y < p->Y) q[3] = p;
    }
    double alpha = 0;
    for(int k = 0; k < n + 5; k++){
        int bi = -1;
        double gap_min = INF;
        for(int i = 0; i < 4; i++){
            double gap = fix(angle(q[i][0], q[i][1]), alpha + i * PI / 2);

```

```

        if (gap < gap_min){
            gap_min = gap;
            bi = i;
        }
    }
    if (++q[bi] == ps + n) q[bi] = ps + 0;
    alpha = fix(alpha + gap_min);
    double a = shadow_length(alpha + PI / 2, *q[0], *q[2]);
    double b = shadow_length(alpha, *q[1], *q[3]);
    area = min(area, a * b);
    peri = min(peri, a + a + b + b);
}
}

```

#### 4.4.4 kd树,支持插入

```

const int N = 500005, K = 2, D=6;
const LL inf = ((ULL)1<<63)-1;
//const int inf=~0U>>1;

struct kd{
    T x[K];
    kd(){rep(i,K)x[i]=0;}
} t[N];
int l[N],r[N];
int a[D],n,tot,root;

void insert(int &cur,kd p, int d) {
    if (!cur){
        cur=++tot;
        rep(i,K) t[cur].x[i] = p.x[i];
        l[cur]=r[cur]=0;
        return;
    }
    T dx = p.x[d] - t[cur].x[d];
    if (++d==K) d=0;
    insert(dx<0?l[cur]:r[cur],p,d);
}

T dis2(kd a,kd b) {
    T s=0;
    rep(i,K) s+=Sqr(a.x[i]-b.x[i]);
    return s;
}

void query(int cur, kd p, LL &ret, int d) {
    if (!cur) return;

```



```

    ret = min(ret, dis2(t[cur],p));
    T dx = p.x[d] - t[cur].x[d];
    if (++d == K) d = 0;
    if (dx < 0) {
        query(l[cur],p,ret,d);
        if (ret > Sqr(dx)) query(r[cur],p,ret,d);
    } else {
        query(r[cur],p,ret,d);
        if (ret > Sqr(dx)) query(l[cur],p,ret,d);
    }
}
}

```

```

void work() {
    root = tot = 0;
    T ans = inf, ret=0;
    kd p;
    rep(i,n){
        p.x[0] = (p.x[0] * a[0] + a[1]) % a[2];
        p.x[1] = (p.x[1] * a[3] + a[4]) % a[5];
        query(root, p, ans, 0);
        insert(root, p, 0);
        ret += ans * (i > 0);
    }
    printf("%I64d\n", ret);
}

```

```

int main() {
    int test;
    scanf("%d", &test);
    rep(cas,test){
        scanf("%d", &n);
        rep(i,D) scanf("%d", &a[i]);
        work();
    }
    return 0;
}

```

#### 4.4.5 knn询问距离最近K个点

```
double cross(Point a,Point b,Point c){return (b.X-a.X)*(c.Y-a.Y)-(c.X-a.X)*(b.Y-a.Y);}
```

```
double dot(Point a,Point b,Point c){return (b.X-a.X)*(c.X-a.X)+(b.Y-a.Y)*(c.Y-a.Y);}
```

```

bool inpoly(Point a, Point *p, int n){
    int wn = 0;
    rep(i,n){
        Point p1 = p[i], p2 = p[(i + 1) % n];
        double s = cross(a, p1, p2);
    }
}

```

```

        if (!s && dot(a, p1, p2) <= 0) return true;
        double d1 = p1.Y - a.Y, d2 = p2.Y - a.Y;
        if (s > 0 && d1 <= 0 && d2 > 0) ++wn;
        if (s < 0 && d2 <= 0 && d1 > 0) --wn;
    }
    return wn != 0;
}

const int N = 20000, M = 20;

int n, m, r;
Point p[N], poly[M];

const int K = 2;
struct kd {
    LL x[K];
    int id;
}t[N];

double dis2(kd a, kd b){
    double s = 0;
    rep(i,K) s += Sqr(a.x[i] - b.x[i]);
    return s;
}

struct cmpk {
    int k;
    cmpk(int _k): k(_k) {}
    bool operator()(kd a, kd b){ return a.x[k] < b.x[k]; }
};

void build(int l, int r, int d){
    if (r - l <= 1) return;
    int mid = (l + r) >> 1;
    nth_element(t + l, t + mid, t + r, cmpk(d));
    if (++d == K) d = 0;
    build(l, mid, d); build(mid + 1, r, d);
}

typedef priority_queue<pair<double, int> > heap;
void knn(int l, int r, int d, kd p, size_t k, heap &h){
    if (r - l < 1) return;
    int mid = (l + r) >> 1;
    h.push(make_pair(dis2(p, t[mid]), t[mid].id));
    if (h.size() > k) h.pop();
    double dx = p.x[d] - t[mid].x[d];
    if (++d == K) d = 0;
    if (dx < 0) {

```

```

        knn(l, mid, d, p, k, h);
        if (h.top().first > Sqr(dx)) knn(mid + 1, r, d, p, k, h);
    } else {
        knn(mid + 1, r, d, p, k, h);
        if (h.top().first > Sqr(dx)) knn(l, mid, d, p, k, h);
    }
}

void solve(){
    scanf("%d", &m);
    rep(i,m) {
        int x,y;
        scanf("%d%d",&x,&y);
        poly[i]=MP(x,y);
    }
    int cnt = 0;
    rep(i,n){
        if (inpoly(p[i], poly, m)) {
            t[cnt].x[0] = p[i].X; t[cnt].x[1] = p[i].Y;
            t[cnt++].id = i + 1;
        }
    }
    build(0, cnt, 0);
    int q;
    scanf("%d", &q);
    while (q--) {
        kd p;
        scanf("%lld%lld", &p.x[0], &p.x[1]);
        heap h;
        knn(0, cnt, 0, p, 2, h);
        int a, b;
        b = h.top().second; h.pop();
        a = h.top().second;
        printf("%d %d\n", a, b);
    }
}

int main(){
    int dat;
    scanf("%d", &dat);
    rep(cas,dat){
        printf("Case #%d:\n", cas+1);
        scanf("%d",&n);
        rep(i,n){
            int x,y;
            scanf("%d%d",&x,&y);
            p[i]=MP(x,y);
        }
    }
}

```

```

        scanf("%d", &r);
        rep(id,r){
            printf("Region %d\n", id+1);
            solve();
        }
    }
}

```

#### 4.4.6 区域树 (查询区域内点数量)

```

/* MIPT Range Query
 * surport 3 types of operations:
 *   add x, y
 *   delete x, y
 *   count [x1, x2] * [y1, y2] */
#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;

/***** point *****/

struct point {
    int x, y;
};

bool operator<(point p, point q) {
    return p.x < q.x || p.x == q.x && p.y < q.y;
}

bool operator==(point p, point q) {
    return p.x == q.x && p.y == q.y;
}

bool xcmp(point *a, point *b) {
    return a->x < b->x || a->x == b->x && a->y < b->y;
}

bool ycmp(point *a, point *b) {
    return a->y < b->y || a->y == b->y && a->x < b->x;
}

/***** binary indexed tree *****/

void ta_init(int *ta, int n) {
    memset(ta, 0, sizeof(*ta) * (n + 1));
}

```

```

void ta_add(int *ta, int n, int k, int d) {
    while (k <= n) {
        ta[k] += d;
        k += k & -k;
    }
}

int ta_sum(int *ta, int n, int k) {
    int res = 0;
    while (k) {
        res += ta[k];
        k -= k & -k;
    }
    return res;
}

/***** range tree *****/

struct node {
    int size;
    int x1, x2;
    node *l, *r;
    int *la, *lb;
    int *ta;
};

struct range_tree {
    node *root;
    int size;
    point **yl;
};

const int N = 100010;
const int M = N * 20; // N log N
const int INF = 2010000000;

range_tree __rt;
node nodes[N << 1], *next;
point *xs[N], *ys[N], *yt[N];
int links[M << 1], *ln;
int ts[M], *tn;

node *__build(point **xl, point **yl, point **yt, int n) {
    int i, d, na, nb;
    node *p;
    point **ya, **yb;

    p = next++;

```

```

p->x1 = xl[0]->x;
p->x2 = xl[n - 1]->x;
p->size = n;
p->ta = tn; tn += n + 1;
ta_init(p->ta, n + 1);

if (n > 1) {
    d = n / 2;
    ya = yt;
    yb = yt + d;
    na = d;
    nb = n - d;
    p->la = ln; ln += n + 1;
    p->lb = ln; ln += n + 1;
    p->la[n] = na;
    p->lb[n] = nb;
    for (i = n - 1; i >= 0; i--) {
        if (xcmp(yl[i], xl[d]))
            ya[--na] = yl[i];
        else
            yb[--nb] = yl[i];
        p->la[i] = na;
        p->lb[i] = nb;
    }
    p->l = __build(xl, yt, yl, d);
    p->r = __build(xl + d, yt + d, yl + d, n - d);
} else {
    p->l = p->r = NULL;
}

return p;
}

/* NOTE: no duplicated points are allowed
 *      only one range_tree can be maintained at a time */
range_tree *range_tree_build(point *p, int n) {
    range_tree *rt = &__rt;

    for (int i = 0; i < n; i++) {
        xs[i] = &p[i];
        ys[i] = &p[i];
    }
    sort(xs, xs + n, xcmp);
    sort(ys, ys + n, ycmp);
    ln = links;
    tn = ts;
    next = nodes;
    rt->root = n ? __build(xs, ys, yt, n) : NULL;
}

```

```

    for (int i = 0; i < n; i++)
        ys[i] = &p[i];
    sort(ys, ys + n, ycmp);
    rt->yl = ys;
    rt->size = n;

    return rt;
}

int __query(node *p, int x1, int x2, int lb, int ub) {
    if (!p || x2 < p->x1 || p->x2 < x1 || lb >= ub)
        return 0;

    if (x1 <= p->x1 && p->x2 <= x2)
        return ta_sum(p->ta, p->size, ub) - ta_sum(p->ta, p->size, lb);

    return __query(p->l, x1, x2, p->la[lb], p->la[ub]) +
        __query(p->r, x1, x2, p->lb[lb], p->lb[ub]);
}

int range_tree_query(range_tree *rt, int x1, int x2, int y1, int y2) {
    int lb, ub;
    point a, b;
    node *root = rt->root;

    if (!root) return 0;

    a.x = -INF; a.y = y1;
    b.x = +INF; b.y = y2;
    lb = lower_bound(rt->yl, rt->yl + rt->size, &a, ycmp) - rt->yl;
    ub = upper_bound(rt->yl, rt->yl + rt->size, &b, ycmp) - rt->yl;
    return __query(root, x1, x2, lb, ub);
}

int range_tree_add(range_tree *rt, int x, int y, int d) {
    int i;
    point a;
    node *p = rt->root;

    if (!p) return 0;

    a.x = x; a.y = y;
    i = lower_bound(rt->yl, rt->yl + rt->size, &a, ycmp) - rt->yl;
    if (i == rt->size || rt->yl[i]->x != x || rt->yl[i]->y != y)
        return 0;
    if ((ta_sum(p->ta, p->size, i) < ta_sum(p->ta, p->size, i + 1)) ^ (d < 0))
        return 0;

```

```

while (p) {
    ta_add(p->ta, p->size, i + 1, d);
    if (p->size <= 1) break;
    if (p->la[i] != p->la[i + 1])
        i = p->la[i], p = p->l;
    else
        i = p->lb[i], p = p->r;
}
return 1;
}

/***** main *****/

struct query {
    char t;
    point p;
    int x1, y1, x2, y2;
};

int main() {
    char cmd[10];
    int n, m;
    static query qs[N];
    static point p[N];

    n = m = 0;
    while (scanf("%s", cmd) != EOF) {
        switch (qs[m].t = cmd[0]) {
            case 'A':
                scanf("%d %d", &qs[m].p.x, &qs[m].p.y);
                p[n++] = qs[m].p;
                break;
            case 'D':
                scanf("%d %d", &qs[m].p.x, &qs[m].p.y);
                break;
            case 'C':
                scanf("%d %d %d %d", &qs[m].x1, &qs[m].y1, &qs[m].x2, &qs[m].y2);
                break;
        }
        m++;
    }

    sort(p, p + n);
    n = unique(p, p + n) - p;
    range_tree *rt = range_tree_build(p, n);

    for (int i = 0; i < m; i++) {

```



```

query *q = &q[s[i];
switch (q->t) {
    case 'A':
        if (range_tree_add(rt, q->p.x, q->p.y, +1)) {
            puts("ADDED");
        } else {
            puts("ALREADY EXISTS");
        }
        break;
    case 'D':
        if (range_tree_add(rt, q->p.x, q->p.y, -1)) {
            puts("DELETED");
        } else {
            puts("NOT FOUND");
        }
        break;
    case 'C':
        printf("%d\n", range_tree_query(rt, q->x1, q->x2, q->y1, q->y2));
        break;
}
}

return 0;
}

```

## 4.5 面积交

### 4.5.1 圆与多边形面积交

```

Point p[3];
double r;

double cross(Point a, Point b){
    return a.X * b.Y - a.Y * b.X;
}

double cross(Point a, Point b, Point c){
    return cross(b - a, c - a);
}

double dot(Point a, Point b){
    return a.X * b.X + a.Y * b.Y;
}

double dot(Point a, Point b, Point c){
    return dot(b - a, c - a);
}

double len(Line l){

```

```

    return abs(l.S - l.F);
}

double dis(Point p, Line l){
    return fabs(cross(p, l.F, l.S) / len(l));
}

bool inter(Line a, Line b, Point &p){
    double s1 = cross(a.F, a.S, b.F);
    double s2 = cross(a.F, a.S, b.S);
    if (!sign(s1 - s2)) return false;
    p = (s1 * b.S - s2 * b.F) / (s1 - s2);
    return true;
}

Vec dir(Line l){
    return l.S - l.F;
}

Vec normal(Vec v){
    return Vec(-v.Y, v.X);
}

Vec unit(Vec v){
    return v / abs(v);
}

bool onseg(Point p, Line l){
    return sign(cross(p, l.F, l.S)) == 0 && sign(dot(p, l.F, l.S)) <= 0;
}

double arg(Vec a, Vec b){
    double d = arg(b) - arg(a);
    if (d > PI) d -= 2 * PI;
    if (d < -PI) d += 2 * PI;
    return d;
}

double area(Point a, Point b){
    double s1 = 0.5 * cross(a, b);
    double s2 = 0.5 * arg(a, b) * r * r;
    return fabs(s1) < fabs(s2) ? s1 : s2;
}

double area(){
    double s = 0;
    rep(i, n){
        Point O(0, 0), A = p[i], B = p[(i + 1) % 3];

```

```

    Line AB(A, B);
    double d = dis(O, AB);
    if (sign(d - r) >= 0){
        s += area(A, B);
    }
    else{
        Point P;
        inter(AB, Line(O, O + normal(dir(AB))), P);
        Vec v = sqrt(r * r - d * d) * unit(dir(AB));
        Point P1 = P - v, P2 = P + v;
        if (!onseg(P1, AB) && !onseg(P2, AB)){
            s += area(A, B);
        }
        else{
            s += area(A, P1);
            s += area(P1, P2);
            s += area(P2, B);
        }
    }
}
return fabs(s);
}

void init(){
    scanf("%d%d", &n, &r);
    rep(i, n){
        double x, y;
        scanf("%lf%lf", &x, &y);
        p[i] = Point(x, y);
    }
}

int main(){
    init();
    printf("%.12lf\n", area());
    return 0;
}

```

#### 4.5.2 k多边形面积交

```

int n;
int v[MAXN]; // the number of vertexes
point p[MAXN][MAXV];
pair<double, int> c[MAXN * MAXV * 2];
double tot[MAXN + 1];

double pos(point p, line ln) {
    return dcmp(ln.second.X - ln.first.X) ?

```

```

        (p.X - ln.first.X) / (ln.second.X - ln.first.X) :
        (p.Y - ln.first.Y) / (ln.second.Y - ln.first.Y);
    }

double area() {
    memset(tot, 0, sizeof(tot));
    for (int i = 0; i < n; ++i)
        for (int ii = 0; ii < v[i]; ++ii) {
            point A = p[i][ii], B = p[i][(ii + 1) % v[i]];
            line AB = line(A, B);
            int m = 0;
            for (int j = 0; j < n; ++j) if (i != j)
                for (int jj = 0; jj < v[j]; ++jj) {
                    point C = p[j][jj], D = p[j][(jj + 1) % v[j]];
                    line CD = line(C, D);
                    int f1 = dcmp(cross(A, B, C));
                    int f2 = dcmp(cross(A, B, D));
                    if (!f1 && !f2) {
                        if (i < j && dcmp(dot(dir(AB), dir(CD))) > 0) {
                            c[m++] = make_pair(pos(C, AB), 1);
                            c[m++] = make_pair(pos(D, AB), -1);
                        }
                    } else {
                        double s1 = cross(C, D, A);
                        double s2 = cross(C, D, B);
                        double t = s1 / (s1 - s2);
                        if (f1 >= 0 && f2 < 0) c[m++] = make_pair(t, 1);
                        if (f1 < 0 && f2 >= 0) c[m++] = make_pair(t, -1);
                    }
                }
            c[m++] = make_pair(0.0, 0);
            c[m++] = make_pair(1.0, 0);
            sort(c, c + m);
            double s = cross(A, B), z = min(max(c[0].first, 0.0), 1.0);
            for (int j = 1, k = c[0].second; j < m; ++j) {
                double w = min(max(c[j].first, 0.0), 1.0);
                tot[k] += s * (w - z);
                k += c[j].second;
                z = w;
            }
        }
    return tot[0];
}

/*
tot[0] is the aera of union
tot[n - 1] is the aera of intersection
tot[k - 1] - tot[k] is the aera of region covered by k times

```

```
*/
```

### 4.5.3 两个多边形面积交

```
#include <stdio.h>
#include <math.h>
#include <algorithm>
#include <vector>
#include <list>
using namespace std;

const int N = 1005;
const double eps = 1e-8;

inline int sign(double x) {
    return x < -eps ? -1 : x > eps;
}

struct point {
    double x, y;
    point(double x = 0, double y = 0) : x(x), y(y) {}

    bool operator<(const point &p) const {
        return sign(x - p.x) * 2 + sign(y - p.y) < 0;
    }
    bool operator==(const point &p) const {
        return !sign(x - p.x) && !sign(y - p.y);
    }
};

struct triangle {
    point p[3];
};

struct line {
    point p, q;
    line() {}
    line(point p, point q) : p(p), q(q) {}
};

double cross(point a, point b, point c) {
    return (b.x - a.x) * (c.y - b.y) - (b.y - a.y) * (c.x - b.x);
}

/* same side: 1; at least one of a, b touches l: 0; otherwise -1 */
int side(line m, point p, point q) {
    return sign(cross(m.p, m.q, p)) * sign(cross(m.p, m.q, q));
}
```

```

bool parallel(line u, line v) {
    return !sign((u.p.x - u.q.x) * (v.p.y - v.q.y) - (v.p.x - v.q.x) * (u.p.y - u.q.y));
}

/* u, v: line segment, inclusive */
bool intersected(line u, line v) {
    return !parallel(u, v) && side(u, v.p, v.q) <= 0 && side(v, u.p, u.q) <= 0;
}

/* intersection point */
point ip(line u, line v) {
    double n = (u.p.y - v.p.y) * (v.q.x - v.p.x) - (u.p.x - v.p.x) * (v.q.y - v.p.y);
    double d = (u.q.x - u.p.x) * (v.q.y - v.p.y) - (u.q.y - u.p.y) * (v.q.x - v.p.x);
    double r = n / d;
    return point(u.p.x + r * (u.q.x - u.p.x), u.p.y + r * (u.q.y - u.p.y));
}

bool inside_triangle(point a, point b, point c, point p) {
    if (cross(a, b, c) < 0) swap(b, c);
    return sign(cross(a, b, p)) > 0 && sign(cross(b, c, p)) > 0 && sign(cross(c, a, p)) > 0;
}

/* stores n - 2 splited triangles into T */
void triangulate(point p[], int n, triangle T[]) {
    list<point> P(p, p + n);
    list<point>::iterator a, b, c, q;
    triangle t;

    int m = 0;
    if (P.size() < 3) return;

    for (a = b = P.begin(), c = ++b, ++c; c != P.end(); a = b, c = ++b, ++c) if (sign(cross(*a, *b, *
        for (q = P.begin(); q != P.end(); q++) {
            if (q == a) { ++q; ++q; continue; }
            if (inside_triangle(*a, *b, *c, *q)) break;
        }
        if (q == P.end()) {
            t.p[0] = *a;
            t.p[1] = *b;
            t.p[2] = *c;
            T[m++] = t;

            P.erase(b);
            b = a;
            if (b != P.begin()) b--;
        }
    }
}

```

```

}

double area_polygon(point A[], int n) {
    double area = 0;
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        area += A[i].x * A[j].y - A[i].y * A[j].x;
    }
    return area / 2;
}

point o;

bool acmp(point a, point b) {
    return cross(o, a, b) > 0;
}

double triangles_intersection(triangle a, triangle b) {
    point p[15]; int n = 0;
    triangle T[2] = { a, b };

    for (int t = 0; t < 2; t++)
        for (int i = 0; i < 3; i++)
            if (inside_triangle(T[t].p[0], T[t].p[1], T[t].p[2], T[!t].p[i]))
                p[n++] = T[!t].p[i];

    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++) {
            line l1(T[0].p[i], T[0].p[(i + 1) % 3]);
            line l2(T[1].p[j], T[1].p[(j + 1) % 3]);
            if (intersected(l1, l2))
                p[n++] = ip(l1, l2);
        }

    if (!n) return 0;

    sort(p, p + n);
    n = unique(p, p + n) - p;

    if (n >= 3) {
        o = p[0];
        sort(p + 1, p + n, acmp);
        return area_polygon(p, n);
    }
    return 0;
}

triangle t1[N], t2[N];

```

```

double polygon_intersection(point p[], int n, point q[], int m) {
    triangulate(p, n, t1);
    triangulate(q, m, t2);

    double area = 0;
    for (int i = 0; i < n - 2; i++)
        for (int j = 0; j < m - 2; j++)
            area += triangles_intersection(t1[i], t2[j]);

    return area;
}

int n, m;
point p[N], q[N];

int main() {
    double a1, a2;
    scanf("%d %d", &n, &m);
    for (int i = 0; i < n; i++) scanf("%lf %lf", &p[i].x, &p[i].y);
    for (int i = 0; i < m; i++) scanf("%lf %lf", &q[i].x, &q[i].y);
    if ((a1 = area_polygon(p, n)) < 0) reverse(p, p + n);
    if ((a2 = area_polygon(q, m)) < 0) reverse(q, q + m);
    printf("area of the union of two polygon is: %lf\n", fabs(a1) + fabs(a2) - polygon_intersection(p, n, q, m));
    return 0;
}

```

#### 4.5.4 k圆面积交

```

/* Spoj CIRUT
 * Given n circles, find the area of all k-union regions
 * NOTE: No duplicated circles are allowed!
 * O(n ^ 2 log(n)) */
#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <math.h>

#define Sqr(x) (x)*(x)

using namespace std;

const double eps = 1e-8, inf = 1e+9, pi = acos(-1.0);

inline int sign(double x) { return x < -eps ? -1 : x > eps; }

struct point {
    double x, y;
}

```



```

};

struct circle {
    point o;
    double r;
};

struct event {
    double a;
    int t;
    point p;
    event() {}
    event(double a, int t, point p) : a(a), t(t), p(p) {}
    bool operator<(const event e) const { return a < e.a; }
};

inline int ip_circle_circle(const circle &c1, const circle &c2, point &p1, point &p2) {
    double mx = c2.o.x - c1.o.x, sx = c2.o.x + c1.o.x, mx2 = sqr(mx);
    double my = c2.o.y - c1.o.y, sy = c2.o.y + c1.o.y, my2 = sqr(my);
    double sq = mx2 + my2, d = -(sq - sqr(c1.r - c2.r)) * (sq - sqr(c1.r + c2.r));
    if (!sign(sq)) return 0;
    if (d + eps < 0) return 0;
    if (d < eps) d = 0; else d = sqrt(d);
    double x = mx * ((c1.r + c2.r) * (c1.r - c2.r) + mx * sx) + sx * my2;
    double y = my * ((c1.r + c2.r) * (c1.r - c2.r) + my * sy) + sy * mx2;
    double dx = mx * d, dy = my * d; sq *= 2;
    p1.x = (x + dy) / sq; p1.y = (y - dx) / sq;
    p2.x = (x - dy) / sq; p2.y = (y + dx) / sq;
    return d > eps ? 2 : 1;
}

inline double fix(double a, double b = 0) {
    a -= b;
    if (sign(a) < 0) a += 2 * pi;
    return a;
}

inline double angle(point a, point b) {
    return fix(atan2(b.y - a.y, b.x - a.x));
}

inline int contains(const circle &c1, const circle &c2) {
    return c1.r > c2.r && sign(sqr(c1.o.x - c2.o.x) + sqr(c1.o.y - c2.o.y) - sqr(c1.r - c2.r)) <= 0
}

inline double cross(point a, point b, point c) {
    return (b.x - a.x) * (c.y - b.y) - (b.y - a.y) * (c.x - b.x);
}

```

```

const int N = 1000 + 10;
int n, en;
circle cs[N];
event events[N + N];
point o;
double area[N];

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%lf %lf %lf", &cs[i].o.x, &cs[i].o.y, &cs[i].r);
    memset(area, 0, sizeof area);
    for (circle *a = cs; a < cs + n; a++) {
        int cover = 1;
        en = 0;
        for (circle *b = cs; b < cs + n; b++) if (a != b) {
            if (contains(*b, *a)) cover++;
            point p1, p2;
            if (ip_circle_circle(*a, *b, p1, p2) >= 2) {
                events[en++] = event(angle(a->o, p1), -sign(cross(a->o, b->o, p1)), p1);
                events[en++] = event(angle(a->o, p2), -sign(cross(a->o, b->o, p2)), p2);
            }
            if ((events[en - 2].a < events[en - 1].a) ^ (events[en - 2].t > events[en - 1].t)) cover++;
        }
        sort(events, events + en);
        events[en] = events[0];
        for (int i = 0; i < en; i++) {
            event *e1 = &events[i];
            event *e2 = &events[i + 1];
            cover += e1->t;
            double da = fix(e2->a, e1->a);
            area[cover] += cross(o, e1->p, e2->p) + sqr(a->r) * (da - sin(da));
        }
        if (!en) area[cover] += sqr(a->r) * pi * 2;
    }
    for (int i = 1; i < n; i++)
        area[i] -= area[i + 1];
    for (int i = 1; i <= n; i++)
        printf("[%d] = %.3lf\n", i, area[i] / 2);
    return 0;
}

```

#### 4.5.5 矩形和多个圆并的面积交

```

/* Given n circles and a rectangle, find the area of intersection of
 * the rectangle and the union of these circles
 * O(n ^ 3) */

```

```

#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <math.h>
using namespace std;

inline double sqr(double x) {
    return x * x;
}

const double eps = 1e-6;

inline int sign(double x) {
    return x < -eps ? -1 : x > eps;
}

inline bool deq(double x, double y) {
    return !sign(x - y);
}

struct point {
    double x, y;
};

struct circle {
    point o;
    double r;
};

inline int contains(const circle &a, const circle &b) {
    return a.r >= b.r && sqr(a.o.x - b.o.x) + sqr(a.o.y - b.o.y) <= sqr(a.r - b.r);
}

inline int ip_circle_circle(const circle &c1, const circle &c2, point &p1, point &p2) {
    double mx = c2.o.x - c1.o.x, sx = c2.o.x + c1.o.x, mx2 = sqr(mx);
    double my = c2.o.y - c1.o.y, sy = c2.o.y + c1.o.y, my2 = sqr(my);
    double sq = mx2 + my2, d = -(sq - sqr(c1.r - c2.r)) * (sq - sqr(c1.r + c2.r));
    if (!sign(sq)) return 0;
    if (d + eps < 0) return 0;
    if (d < eps) d = 0; else d = sqrt(d);
    double x = mx * ((c1.r + c2.r) * (c1.r - c2.r) + mx * sx) + sx * my2;
    double y = my * ((c1.r + c2.r) * (c1.r - c2.r) + my * sy) + sy * mx2;
    double dx = mx * d, dy = my * d; sq *= 2;
    p1.x = (x + dy) / sq; p1.y = (y - dx) / sq;
    p2.x = (x - dy) / sq; p2.y = (y + dx) / sq;
    return d > eps ? 2 : 1;
}

```

```

inline int ip_circle_y(const circle &c, double y, double &x1, double &x2) {
    double d = sqr(c.r) - sqr(y - c.o.y);
    if (sign(d) < 0) return 0;
    d = sign(d) ? d : 0;
    double dx = sqrt(d);
    x1 = c.o.x - dx;
    x2 = c.o.x + dx;
    return d > eps ? 2 : 1;
}

inline int ip_circle_x(const circle &c, double x, double &y1, double &y2) {
    double d = sqr(c.r) - sqr(x - c.o.x);
    if (sign(d) < 0) return 0;
    d = sign(d) ? d : 0;
    double dy = sqrt(d);
    y1 = c.o.y - dy;
    y2 = c.o.y + dy;
    return d > eps ? 2 : 1;
}

inline double area_bow(double r, double l) {
    double tri = l * sqrt(max(0.0, sqr(r) - sqr(l / 2))) / 2;
    double theta = 2 * asin(l / (r + r));
    double sector = theta * r * r / 2;
    return sector - tri;
}

struct arc {
    circle *c;
    double x1, x2;
    int t;
};

bool operator<(const arc &a, const arc &b) {
    return a.x1 + a.x2 < b.x1 + b.x2;
}

const int N = 200 + 10;

circle c[N];
int n, n1, n2;
double ys[N * N + N * 7 + 2];
int ysn;
double X, Y;
arc a[N * 2];
int an;

double solve(double ya, double yb) {

```

```

an = 0;
for (int i = 0; i < n; i++) {
    double x1, x2, x3, x4;
    if (ip_circle_y(c[i], ya, x1, x3) && ip_circle_y(c[i], yb, x2, x4)) {
        if (x1 + x2 > X + X || x3 + x4 < 0) continue;
        a[an++] = (arc) { &c[i], x1, x2, 0 }; // XXX g++ only!!
        a[an++] = (arc) { &c[i], x3, x4, 1 }; // XXX g++ only!!
    }
}

sort(a, a + an);

double x1, x2, x3, x4, res = 0;
for (int i = 0, b = 0; i < an; i++) {
    if (a[i].t == 0) {
        if (b == 0) {
            x1 = max(0.0, a[i].x1);
            x2 = max(0.0, a[i].x2);
            if (a[i].x1 + a[i].x2 > 0) res += area_bow(a[i].c->r, hypot(ya - yb, x1 - x2));
        }
        b++;
    } else {
        b--;
        if (b == 0) {
            x3 = min(X, a[i].x1);
            x4 = min(X, a[i].x2);
            res += (x3 + x4 - x1 - x2) * (yb - ya) / 2;
            if (a[i].x1 + a[i].x2 < X + X) res += area_bow(a[i].c->r, hypot(ya - yb, x3 - x4));
        }
    }
}
return res;
}

int main() {
    while (scanf("%lf %lf %d %d", &X, &Y, &n1, &n2), X > 0 || Y > 0 || n1 || n2) {
        for (int i = n2; i < n2 + n1; i++) {
            scanf("%lf %lf", &c[i].o.x, &c[i].o.y);
            c[i].r = 0.58;
        }
        for (int i = 0; i < n2; i++) {
            scanf("%lf %lf", &c[i].o.x, &c[i].o.y);
            c[i].r = 1.31;
        }
        n = n1 + n2;

        for (int i = 0; i < n; i++)
            if (c[i].o.x + c[i].r < 0 ||

```

```

        c[i].o.x - c[i].r > X ||
        c[i].o.y + c[i].r < 0 ||
        c[i].o.y - c[i].r > Y)
    c[i--] = c[--n];

for (int i = 0; i < n; i++)
    for (int j = i + 1; j < n; j++)
        if (contains(c[i], c[j]))
            c[j--] = c[--n];

ysn = 0;
ys[ysn++] = 0;
ys[ysn++] = Y;
for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
        point p1, p2;
        int num = ip_circle_circle(c[i], c[j], p1, p2);
        if (num >= 1) ys[ysn++] = p1.y;
        if (num >= 2) ys[ysn++] = p2.y;
    }
    ys[ysn++] = c[i].o.y;
    ys[ysn++] = c[i].o.y - c[i].r;
    ys[ysn++] = c[i].o.y + c[i].r;

    double y1, y2;
    if (ip_circle_x(c[i], 0, y1, y2)) {
        ys[ysn++] = y1;
        ys[ysn++] = y2;
    }
    if (ip_circle_x(c[i], X, y1, y2)) {
        ys[ysn++] = y1;
        ys[ysn++] = y2;
    }
}
sort(ys, ys + ysn);
ysn = unique(ys, ys + ysn, deq) - ys;

double ans = 0;
for (int i = 0; i + 1 < ysn; i++)
    if (sign(0 - ys[i]) <= 0 && sign(ys[i + 1] - Y) <= 0)
        ans += solve(ys[i], ys[i + 1]);
printf("%.2lf\n", X * Y - ans + eps);
}
return 0;
}

```

## 4.6 其他

### 4.6.1 椭圆周长

```
double const pi = atan2(0, -1.0);

double cal(double a, double b) {
    double e2 = 1.0 - b * b / a / a;
    double e = e2;
    double ret = 1.0;
    double xa = 1.0, ya = 2.0;
    double t = 0.25;
    for (int i = 1; i <= 10000; ++i) {
        ret -= t * e;
        t = t * xa * (xa + 2) / (ya + 2) / (ya + 2);
        xa += 2.0;
        ya += 2.0;
        e *= e2;
    }
    return 2.0 * pi * a * ret;
}

int main() {
    int _ca = 1;
    double a, b;
    int T;
    for (scanf("%d", &T); T--;) {
        scanf("%lf %lf", &a, &b);
        if (a < b) swap(a, b);
        printf("Case %d: %.10lf\n", _ca++, cal(a, b));
    }
    return 0;
}
```

## 5 理论

### 5.1 数学

#### 5.1.1 数学结论

五边形定理

五边形数  $n * (3 * n + 1) / 2$

fibonacci数性质:

$f[n] = f[n - 1] + f[n - 2]$

$f[n + m + 1] = f[n] * f[m] + f[n + 1] * f[m + 1]$

$\gcd(f[n], f[n + 1]) = 1$

$\gcd(f[n], f[n + 2]) = 1$

$\gcd(f[n], f[m]) = f[\gcd(n, m)]$

$$\begin{aligned}
f[n+1]*f[n+1]-f[n]*f[n+2] &= (-1)^n \\
\sigma\{f[i]^2, 1 \leq i \leq n\} &= f[n]*f[n+1] \\
\sigma\{f[i], 0 \leq i \leq n\} &= f[n+2] - 1 \\
\sigma\{f[2*i-1], 1 \leq i \leq n\} &= f[2*n] \\
\sigma\{f[2*i], 1 \leq i \leq n\} &= f[2*n+1]-1 \\
\sigma\{(-1)^i*f[i], 0 \leq i \leq n\} &= (-1)^n*(f[n+1]-f[n])+1 \\
f[2*n-1] &= f[n]^2 - f[n-2]^2 \\
f[2*n+1] &= f[n]^2 + f[n+1]^2 \\
3*f[n] &= f[n+2] + f[n-2] \\
f[n] &= c(n-1,0) + c(n-2,1) + \dots + c(n-1-m,m) \quad (m \leq n-1-m) \\
\sigma\{f[i]*i, 1 \leq i \leq n\} &= n*f[n+2] - f[n+3] + 2
\end{aligned}$$

catalan数性质:

凸多边形三角剖分数

简单有序根树的计数

(0,0) 走到 (n,n) 经过的点(a,b)满足a<=b的路径数

乘法结合问题

$$c[n+1] = (4 * n - 2) / (n + 1) * c[n]$$

$$c[n] = (2*n)!/(n!)/((n+1)!)$$

第一类stirling数性质

有正有负, 其绝对值是n个元素的项目分作k个环排列的数量,s[n,k] (n个人分成k组, 每组再按特定顺序排列)

$$s[n][0] = 0, s[1][1] = 1;$$

$$s[n+1][k] = s[n][k-1] + n * s[n][k]$$

$$|s[n][1]| = (n-1)!$$

$$s[n][k] = (-1)^{(n+k)} * |s[n][k]|$$

$$s[n][n-1] = -C(n,2)$$

$$x*(x-1)*(x-2)..(x-n+1) = \sigma\{s[n][k] * x^k\}$$

第二类stirling数性质

n个元素的集定义k个等价类的方法数目(n个人分成k组的方法数)

$$s[n][n] = s[n][1] = 1$$

$$s[n][k] = s[n-1][k-1] + k * s[n-1][k]$$

$$s[n][n-1] = C(n,2)$$

$$s[n][2] = 2^{(n-1)} - 1$$

$$s[n][k] = 1/(k!) \sigma\{(-1)^{k-j} * C(k,j) * j^n, 1 \leq j \leq k\}$$

bell数性质

$$B[n] = \sigma\{s[n][k], 1 \leq k \leq n\}$$

$$B[n+1] = \sigma\{C(n,k)*B[k], 0 \leq k \leq n\}$$

$$B[p+n] = B[n] + B[n+1] \pmod{p}$$

$$B[p^m+n] = B[n] + B[n+1] \pmod{p}$$

多项式性质

f(x)不存在重根 $\Leftrightarrow \gcd(f(x), f'(x))$ 的次数小于1次

多项式gcd可以用来判断两多项式是否有公共根

数论



$a^n \% b = a^{(n \% \phi(b) + \phi(b)) \% b} \quad (n \geq \phi(b))$   
 lucas定理  $c(n, m) = c(n \% p, m \% p) * c(n / p, m / p) \% p$   
 lucas函数 满足  $f(n, m) = f(n \% p, m \% p) * f(n / p, m / p) \% p$ , 可以猜测满足

原根

$2, 4, p^k, 2 * p^k$  存在原根, 存在原根则原根数量为  $\phi(\phi(n))$   
 验证原根  $x = \phi(n)$ ,  $x = p_1^{a_1} * p_2^{a_2} \dots p_k^{a_k}$   
 原根满足  $t^{(x / \phi)} \neq 1 \pmod n$

罗马数字性质

### 5.1.2 三次方程求解

```

#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <math.h>
using namespace std;

const double PI = acos(-1.0);

typedef struct {
    int n;          // Number of solutions
    double x[3];    // Solutions
} Result;

// a * x ^ 3 + b * x ^ 2 + c * x + d = 0
Result solve_cubic(double a, double b, double c, double d) {
    Result s;
    long double a1 = b / a, a2 = c / a, a3 = d / a;
    long double q = (a1 * a1 - 3 * a2) / 9.0, sq = -2 * sqrt(q);
    long double r = (2 * a1 * a1 * a1 - 9 * a1 * a2 + 27 * a3) / 54.0;
    double z = r * r - q * q * q;
    double theta;

    if (z <= 0) {
        s.n = 3;
        theta = acos(r / sqrt(q * q * q));
        s.x[0] = sq * cos(theta / 3.0) - a1 / 3.0;
        s.x[1] = sq * cos((theta + 2.0 * PI) / 3.0) - a1 / 3.0;
        s.x[2] = sq * cos((theta + 4.0 * PI) / 3.0) - a1 / 3.0;
    } else {
        s.n = 1;
        s.x[0] = pow(sqrt(z) + fabs(r), 1 / 3.0);
        s.x[0] += q / s.x[0];
        s.x[0] *= (r < 0) ? 1 : -1;
        s.x[0] -= a1 / 3.0;
    }
}
  
```

```

    return s;
}

```

### 5.1.3 辛普森积分

```

#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <math.h>
using namespace std;

//{{{start
double f(double x){
    return 8 * sqrt(r * r - x * x) * sqrt(R * R - x * x);
}

double simpson(double width, double fa, double fb, double fc){
    return (fb + fa + 4 * fc) * width / 6;
}

double asr(double a, double b, double eps, double A){
    double c = (a + b) / 2;
    double fa = f(a), fb = f(b), fc = f(c);
    double L = simpson(c - a, fa, fc, f((c + a) / 2));
    double R = simpson(b - c, fc, fb, f((b + c) / 2));
    if (fabs(L + R - A) <= 15 * eps) return L + R + (L + R - A) / 15;
    return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
}

double asr(double a, double b, double eps){
    return asr(a, b, eps, simpson(b - a, f(a), f(b), f((b + a) / 2)));
}
//end}}}

/* simpson integral of f at [a, b] */
double simpson(double (*f)(double), double a, double b) {
    int n = (int)(10000 * (b - a)); n -= n % 2;
    double A = 0, B = 0, d = (b - a) / n;
    for (int i = 1; i < n; i += 2)
        A += f(a + i * d);
    for (int i = 2; i < n; i += 2)
        B += f(a + i * d);
    return (f(a) + f(b) + 4 * A + 2 * B) * d / 3;
}

/* romberg integral of f at [a, b] */
double romberg(double (*f)(double), double l, double r) {
    const int N = 18;

```

```

double a[N][N], p[N];
p[0] = 1;
for (int i = 1; i < N; i++)
    p[i] = p[i - 1] * 4;
a[0][0] = (f(l) + f(r)) / 2;
for (int i = 1, n = 2; i < N; i++, n <= 1) {
    a[i][0] = 0;
    for (int j = 1; j < n; j += 2)
        a[i][0] += f((r - l) * j / n + 1);
    a[i][0] += a[i - 1][0] * (n / 2);
    a[i][0] /= n;
}
for (int j = 1; j < N; j++)
    for (int i = 0; i < N - j; i++)
        a[i][j] = (a[i + 1][j - 1] * p[j] - a[i][j - 1]) / (p[j] - 1);
return a[0][N - 1] * (r - l);
}

/* helper function of adaptive_simpsons */
double adaptive_simpsons_aux(double (*f)(double), double a, double b, double eps,
    double s, double fa, double fb, double fc, int depth) {
    double c = (a + b) / 2, h = b - a;
    double d = (a + c) / 2, e = (c + b) / 2;
    double fd = f(d), fe = f(e);
    double sl = (fa + 4 * fd + fc) * h / 12;
    double sr = (fc + 4 * fe + fb) * h / 12;
    double s2 = sl + sr;
    if (depth <= 0 || fabs(s2 - s) <= 15 * eps)
        return s2 + (s2 - s) / 15;
    return adaptive_simpsons_aux(f, a, c, eps / 2, sl, fa, fc, fd, depth - 1) +
        adaptive_simpsons_aux(f, c, b, eps / 2, sr, fc, fb, fe, depth - 1);
}

/* Adaptive Simpson's Rule, integral of f at [a, b], max error of eps, max depth of depth */
double adaptive_simpsons(double (*f)(double), double a, double b, double eps, int depth) {
    double c = (a + b) / 2, h = b - a;
    double fa = f(a), fb = f(b), fc = f(c);
    double s = (fa + 4 * fc + fb) * h / 6;
    return adaptive_simpsons_aux(f, a, b, eps, s, fa, fb, fc, depth);
}

```

#### 5.1.4 高斯消元

//在异或方程里，要求最小改变次数，那就从后往前面枚举，先枚举只有变量之后，前面的变量就确定了

```

void gauss(int n, int m, double p[M][M]){
    static double tmp[M][M];
    static double *b[M];
    rep(i, n){

```

```

        rep(j, m){
            tmp[i][j] = p[i][j];
        }
    }
    rep(i, n){
        b[i] = tmp[i];
    }
    rep(i, n){
        REP(j, i, n){
            if (sign(fabs(b[j][i]) - fabs(b[i][i])) > 0) swap(b[i], b[j]);
        }
        rep(j, n){
            if (i == j) continue;
            double rate = b[j][i] / b[i][i];
            rep(k, m) b[j][k] -= b[i][k] * rate;
        }
        double rate = b[i][i];
        rep(j, m) b[i][j] /= rate;
    }
    rep(i, n){
        rep(j, m){
            p[i][j] = b[i][j];
        }
    }
}

```

//整数答案不超过LL，可以用辗转相除法做高斯消元

```

void gauss(int n, int m, T p[M][M], T& ret){
    static T tmp[M][M];
    static T *b[M];
    rep(i, n){
        rep(j, m){
            tmp[i][j] = p[i][j];
        }
    }
    rep(i, n){
        b[i] = tmp[i];
    }
    rep(i, n){
        REP(j, i, n){
            if (abs(b[j][i]) > abs(b[i][i])) swap(b[i], b[j]);
        }
        if (b[i][i] == 0){
            ret = 0;
            return;
        }
        REP(j, i + 1, n){
            if (b[j][i] == 0) continue;

```

```

        while(b[j][i]){
            if (abs(b[i][i]) > abs(b[j][i])) swap(b[i], b[j]);
            LL rate = b[j][i] / b[i][i];
            REP(k, i, m) b[j][k] = b[j][k] - b[i][k] * rate;
        }
    }
    ret *= b[i][i];
    ret = abs(ret) % MD;
}
rep(i, n){
    rep(j, m){
        p[i][j] = b[i][j];
    }
}
}

```

### 5.1.5 FFT

```

typedef complex<double> Comp;
typedef Comp cp;

const double PI = acos(-1);
const Comp I(0, 1);

const int N = 1<<18;
Comp tmp[N];
Comp a[N] = { }, b[N] = { };
int n,m,d;
LL ans;
LL c[N];
int v[N];

void fft(Comp *a,int n,int f=1){
    double arg = PI;
    for(int k = n >> 1; k; k >>= 1, arg *= 0.5) {
        cp wm(cos(arg), f * sin(arg)), w(1, 0);
        for (int i = 0; i < n; i += k, w *= wm) {
            int p = i << 1;
            if (p >= n) p -= n;
            for (int j = 0; j < k; ++j) tmp[i + j] = a[p + j] + w * a[p + k + j];
        }
        rep(i,n) a[i] = tmp[i];
    }
}

int calc(int n){
    fft(a,n,1);
    fft(b,n,1);
}

```

```

    rep(i,n) a[i] = a[i]*b[i];
    fft(a,n,-1);
    rep(i,n) a[i] /= n;
}

int main(){
    int T;
    scanf("%d",&T);
    rep(cas,T){
        scanf("%d",&n);
        rep(i,n) scanf("%d",&v[i]);
        int ma=0;
        rep(i,n) ma=max(ma,v[i]);
        rep(i,N) a[i]=b[i]=Comp(0,0);
        rep(i,n) b[v[i]]=a[v[i]]+=Comp(1,0);
        int top=1;
        while(top<=ma*2) top*=2;
        calc(top);
        LL ans=(LL)n*(n-1)*(n-2)/6;
        rep(i,top) c[i]=(LL)(a[i].real()+0.4);
        rep(i,n) c[v[i]*2]--;
        rep(i,top) c[i]/=2;
        //rep(i,top) cout<<i <<' '<<c[i]<<endl;
        REP(i,1,top) c[i]+=c[i-1];
        rep(i,n) ans-=c[v[i]];
        printf("%.7lf\n",ans/((double)n*(n-1)*(n-2)/6));
    }
    return 0;
}

```

### 5.1.6 linear programming

```

/*
maximize
    c1 * x1 + c2 * x2 + ... + cn * xn
subject to
    a1,1 * x1 + a1,2 * x2 + ... + a1,n * xn <= b1
    .
    .
    .
    am,1 * x1 + am,2 * x2 + ... + am,n * xn <= bm

```

1. minimize the object function:  $c_i \Rightarrow -c_i$ ;
  2. exist  $a_i(x_1, x_2, \dots, x_n) = b_i$ :  $a_i(x_1, x_2, \dots, x_n) = b_i \Rightarrow a_i(x_1, x_2, \dots, x_n) \leq b_i \ \& \ a_i(x_1, x_2, \dots, x_n) \geq b_i$ ;
  3. exist  $a_i(x_1, x_2, \dots, x_n) \geq b_i$ :  $a_i(x_1, x_2, \dots, x_n) \geq b_i \Rightarrow -a_i(x_1, x_2, \dots, x_n) \leq -b_i$ ;
  4. exist  $x_i$  which don't have the limitation of  $x_i \geq 0$ : change  $x_i$  into  $(x_{i1} - x_{i2})$ , add  $x_{i1} \geq 0, x_{i2} \geq 0$ ;
- \*/

```

const double EPS = 1E-10;
const int MAXSIZE = 2000; //m + n
const int INF = 1000000000;

class LinearProgramming {
    double A[MAXSIZE + 1][MAXSIZE + 1];
    double b[MAXSIZE + 1], c[MAXSIZE + 1];
    double origC[MAXSIZE + 1];
    bool inB[MAXSIZE + 1];
    int N[MAXSIZE + 1 + 1], B[MAXSIZE + 1 + 1];
    int n, m;
    double v;

    void read() {
        scanf("%d%d", &n, &m);
        for (int i = 1; i <= n; ++i) scanf("%lf", &c[i]);
        for (int i = 1; i <= m; ++i) {
            for (int j = 1; j <= n; ++j) scanf("%lf", &A[n + i][j]);
            scanf("%lf", &b[n + i]);
        }
    }

    void pivot(int l, int e) {
        double key = A[l][e];
        b[e] = b[l] / key;
        for (int i = 1; i <= N[0]; ++i)
            if (N[i] != e) A[e][N[i]] = A[l][N[i]] / key;
        A[e][l] = 1.0 / key;

        for (int i = 1; i <= B[0]; ++i) {
            if (B[i] == l) continue;
            double tmp = A[B[i]][e];
            b[B[i]] = b[B[i]] - A[B[i]][e] * b[e];
            for (int j = 1; j <= N[0]; ++j)
                if (N[j] != e) A[B[i]][N[j]] = A[B[i]][N[j]] - A[e][N[j]] * tmp;
            A[B[i]][l] = -tmp * A[e][l];
        }

        v += b[e] * c[e];
        for (int i = 1; i <= N[0]; ++i)
            if (N[i] != e) c[N[i]] = c[N[i]] - A[e][N[i]] * c[e];
        c[l] = -A[e][l] * c[e];

        for (int i = 1; i <= N[0]; ++i)
            if (N[i] == e) N[i] = 1;
        for (int i = 1; i <= B[0]; ++i)
            if (B[i] == l) B[i] = e;
    }
}

```

```

//false stands for unbounded
bool opt() {
    while (1) {
        int l, e;
        double maxUp = -1; //^2» 0£i
        for (int ie = 1; ie <= N[0]; ++ie) {
            int te = N[ie];
            if (c[te] <= EPS) continue;
            double delta = INF;
            int tl = MAXSIZE + 1;
            for (int i = 1; i <= B[0]; ++i)
                if (A[B[i]][te] > EPS) {
                    double temp = b[B[i]] / A[B[i]][te];
                    if (delta == INF || temp < delta || temp == delta && B[i] < tl) {
                        delta = temp;
                        tl = B[i];
                    }
                }
            if (tl == MAXSIZE + 1) return 0;
            if (delta * c[te] > maxUp) {
                maxUp = delta * c[te];
                l = tl; e = te;
            }
        }
        if (maxUp == -1) break;
        pivot(l, e);
    }
    return 1;
}

void delete0() {
    int p = 1;
    while (p <= B[0] && B[p]) ++p;
    if (p <= B[0]) {
        int i = 1;
        while (i <= N[0] && fabs(A[0][N[i]]) < EPS) ++i;
        pivot(0, N[i]);
    }
    p = 1;
    while (p <= N[0] && N[p]) ++p;
    for (int i = p; i < N[0]; ++i) N[i] = N[i + 1];
    --N[0];
}

bool initialize() {
    N[0] = B[0] = 0;
    for (int i = 1; i <= n; ++i) N[++N[0]] = i;
}

```



```

for (int i = 1; i <= m; ++i) B[++B[0]] = n + i;
v = 0;

int l = B[1];
for (int i = 2; i <= B[0]; ++i)
    if (b[B[i]] < b[l]) l = B[i];
if (b[l] >= 0) return 1;

memcpy(origC, c, sizeof(double) * (n + m + 1));
N[++N[0]] = 0;
for (int i = 1; i <= B[0]; ++i) A[B[i]][0] = -1;
memset(c, 0, sizeof(double) * (n + m + 1));
c[0] = -1;
pivot(l, 0);
opt();
if (v < -EPS) return 0;
delete0();

memcpy(c, origC, sizeof(double) * (n + m + 1));
memset(inB, 0, sizeof(bool) * (n + m + 1));
for (int i = 1; i <= B[0]; ++i) inB[B[i]] = 1;
for (int i = 1; i <= n + m; ++i)
    if (inB[i] && c[i] != 0) {
        v += c[i] * b[i];
        for (int j = 1; j <= N[0]; ++j) c[N[j]] -= A[i][N[j]] * c[i];
        c[i] = 0;
    }
return 1;
}

public: void simplex() {
    read();
    if (!initialize()) {
        printf("Infeasible\n");
        return;
    }
    if (!opt()) {
        printf("Unbounded\n");
        return;
    }
    else printf("Max value is %lf\n", v);

    bool inN[MAXSIZE + 1];
    memset(inN, 0, sizeof(bool) * (n + m + 1));
    for (int i = 1; i <= N[0]; ++i) inN[N[i]] = 1;
    for (int i = 1; i <= n; ++i)
        if (inN[i]) printf("x%d = %lf\n", i, 0.0);
        else printf("x%d = %lf\n", i, b[i]);
}

```

```

    }
};

```

### 5.1.7 线性递推式 $n \times n \times \log n$

```

/* f[i] = a[i], i < m;
 * f[n] = b[0] * f[n - m] + ... + b[m - 1] * f[n - 1];
 * given a[], b[], m, n; find f[n]
 * O(M ^ 2 log N)
 * !!!!m = 1 特判
 */

const int M = 222;
const int MD=1000000007;

LL n;
int u,d;
int p[M],q[M];
bool use[M];
LL a[M],b[M];

int calc(LL n,int m,LL a[],LL c[],int p=MD){
    LL v[M]={1%p},u[M<<1],msk=!n;
    for(LL i=n;i>1;i>>=1) msk<<=1;
    for(LL x=0;msk;msk>>=1,x<<=1){
        fill_n(u,m<<1,0);
        int b=!!(n&msk);
        x|=b;
        if (x<m) u[x]=1%p;
        else{
            rep(i,m) for(int j=0,t=i+b;j<m;++j,++t) u[t]+=v[i]*v[j],u[t]%=p;
            fba(i,(m<<1)-1,m) for(int j=0,t=i-m;j<m;++j,++t) u[t]+=c[j]*u[i],u[t]%=p;
        }
        copy(u,u+m,v);
    }
    LL ret=0;
    rep(i,m) ret+=v[i]*a[i],ret%=p;
    return ret;
}

int main(){
    while(~scanf("%I64d",&n)){
        Cls(a);
        Cls(b);
        Cls(use);
        scanf("%d",&u);
        rep(i,u) scanf("%d",p+i);
        scanf("%d",&d);
    }
}

```

```

    rep(i,d) scanf("%d",q+i);
    int top=0;
    rep(i,d) top=max(top,q[i]+1),use[q[i]]=true;
    b[0]=1;
    REP(i,1,top){
        rep(j,u) if (i>=p[j]) b[i]+=b[i-p[j]],b[i]%=MD;
    }
    rep(i,top) if (!use[i]) b[i]=0;
    a[0]=1;
    REP(i,1,top){
        fab(j,1,i) a[i]+=a[i-j]*b[j],a[i]%=MD;
    }
    reverse(b,b+top);
    printf("%d\n",calc(n,top-1,a,b));
}
return 0;
}

```

#### 5.1.8 线性递推式+fft

```

/* f[i] = a[i], i < m;
 * f[n] = b[0] * f[n - m] + ... + b[m - 1] * f[n - 1];
 * given a[], b[], m, n; find f[n]
 * O(M ^ 2 log N)
 * !!!!m = 1 特判
 */
using namespace std;

typedef long long LL;
typedef long long ULL;
//typedef complex<double> Comp;

const int N = 10000;
const int MD = 5767169;
const long double EPS = 0.05;
const double PI = acos(-1.0);

typedef complex<double> Comp;
typedef Comp cp;

const Comp I(0, 1);
const int M = 1 << 15;

Comp fft_a[M], fft_b[M];
Comp tmp[M];

int n, a, b, p, q;

```

```

int fac[MD], inv[MD];

inline int ex_gcd(int a, int b, int &x, int &y){
    if (!a){
        x = 0;
        y = 1;
        return b;
    }
    int g = ex_gcd(b % a, a, x, y);
    int t = y;
    y = x;
    x = t - (b / a) * y;
    return g;
}

inline int qmod(int x, int n){
    int ret = 1, y = x % MD;
    while(n){
        if (n & 1) ret = (LL)ret * y % MD;
        y = (LL)y * y % MD;
        n >>= 1;
    }
    return ret;
}

inline int lucas(int n, int m){
    int ret = 1;
    while(n && m && ret){
        int nn = n % MD, mm = m % MD;
        if (nn < mm) return 0;
        else ret = (LL)ret * fac[nn] % MD * inv[mm] % MD * inv[nn - mm] % MD;
        n /= MD, m /= MD;
    }
    return ret;
}

void init(){
    fac[0] = 1;
    inv[0] = 1;
    inv[1] = 1;
    for(int i = 2; i < MD; i++){
        inv[i] = -(LL)(MD / i) * inv[MD % i] % MD;
    }
    for(int i = 1; i < MD; i++){
        fac[i] = ((LL)fac[i - 1] * i) % MD;
        inv[i] = ((LL)inv[i - 1] * inv[i]) % MD;
    }
}

```

```

void fft(Comp *a, int n, int f = 1){
    double arg = PI;
    for(int k = n >> 1; k; k >>= 1, arg *= 0.5) {
        cp wm(cos(arg), f * sin(arg)), w(1, 0);
        for (int i = 0; i < n; i += k, w *= wm) {
            int p = i << 1;
            if (p >= n) p -= n;
            for (int j = 0; j < k; ++j) tmp[i + j] = a[p + j] + w * a[p + k + j];
        }
        rep(i,n) a[i] = tmp[i];
    }
}

```

```

void calc(Comp *a, Comp *b, int n){
    fft(a, n, 1);
    fft(b, n, 1);
    rep(i, n) a[i] = a[i] * b[i];
    fft(a, n, -1);
    rep(i, n) a[i] /= n;
}

```

```

int calc(int n, int m){
    static int v[(M << 1) - 1], u[(M << 1) - 1], t[(M << 1) - 1];
    int size = 1;
    while (size <= 2 * m - 2) size *= 2;
    rep(i, 2 * m - 1) v[i] = 0;
    v[0] = 1;
    int now = 1;
    while(n){
        if (now < m){
            rep(i, 2 * m - 1) u[i] = 0;
            u[now] = 1;
        }
        else{
#ifdef FFT
            rep(i, size) fft_a[i] = fft_b[i] = Comp(0, 0);
            rep(i, m){
                fft_a[i] = Comp(u[i], 0);
                fft_b[i] = Comp(u[i], 0);
            }
            calc(fft_a, fft_b, size);
            rep(i, m * 2 - 1) u[i] = (LL)(fft_a[i].real() + EPS) % MD;
#else
            for(int i = 0; i < m; i++) t[i] = u[i], u[i] = 0;
            for(int i = 0; i < m; i++){
                for(int j = 0; j < m; j++){
                    u[i + j] = (u[i + j] + (LL)t[i] * t[j]) % MD;

```

```

        }
    }
#endif

    for(int i = 2 * m - 2; i >= m; i--){
        u[i - p] = (u[i - p] + (LL)a * u[i]) % MD;
        u[i - q] = (u[i - q] + (LL)b * u[i]) % MD;
        u[i] = 0;
    }
}

if (n & 1){
#ifdef FFT
    rep(i, size) fft_a[i] = fft_b[i] = Comp(0, 0);
    rep(i, m){
        fft_a[i] = Comp(u[i], 0);
        fft_b[i] = Comp(v[i], 0);
    }
    calc(fft_a, fft_b, size);
    rep(i, m * 2 - 1) v[i] = (LL)(fft_a[i].real() + EPS) % MD;
#else
    for(int i = 0; i < m; i++) t[i] = v[i], v[i] = 0;
    for(int i = 0; i < m; i++){
        for(int j = 0; j < m; j++){
            v[i + j] = (v[i + j] + (LL)t[i] * u[j]) % MD;
        }
    }
#endif
}

for(int i = 2 * m - 2; i >= m; i--){
    v[i - p] = (v[i - p] + (LL)a * v[i]) % MD;
    v[i - q] = (v[i - q] + (LL)b * v[i]) % MD;
    v[i] = 0;
}

}

now <<= 1;
n >>= 1;
}

return v[m - 1];
}

int main(){
    init();
    while(~scanf("%d%d%d%d", &a, &b, &p, &q, &n)){
        if (p > q) swap(p, q), swap(a, b);
        int ret;
        if (p == q){
            if (n % p != 0){
                ret = 0;
            }
            else{

```

```

        ret = qmod(a + b, n);
    }
}
else if (q <= 1000){
    n += q - 1;
    ret = calc(n, q);
}
else{
    ret = 0;
    int x, y;
    int d = ex_gcd(p, q, x, y);
    if (n % d != 0){
        puts("0");
        continue;
    }
    x = ((LL)x * n / d % (q / d) + (q / d)) % (q / d);
    y = (n - p * x) / q;
    while(y >= 0){
        ret = (ret + (LL)qmod(a, x) * qmod(b, y) % MD * lucas(x + y, x) % MD) % MD;
        x += q / d;
        y -= p / d;
    }
}
printf("%d\n", (ret % MD + MD) % MD);
}
return 0;
}

```

### 5.1.9 牛顿插值法

```

#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;

typedef long long ll;

const ll R = 10000000007;
const int N = 1000 + 10;

int n, q;
ll fs[N][N], xs[N], ys[N];

inline ll modular(ll x, ll y) {
    return (x % y + y) % y;
}

ll exgcd(ll m, ll n, ll &a, ll &b) {

```

```

    if (!n) return a = 1, b = 0, m;
    ll d = exgcd(n, m % n, b, a);
    b -= m / n * a;
    return d;
}

ll invert(ll x, ll m) {
    ll a, b;
    exgcd(x, m, a, b);
    return modular(a, m);
}

ll f(ll x) {
    ll y = ys[0];
    ll p = 1;
    for (int i = 1; i < n; i++) {
        p = modular(p * (x - xs[i - 1]), R);
        y = modular(y + p * fs[0][i], R);
    }
    return y;
}

int main() {
    scanf("%d %d", &n, &q);
    for (int i = 0; i < n; i++)
        scanf("%lld %lld", &xs[i], &ys[i]);
    for (int i = 0; i < n; i++)
        fs[i][i] = ys[i];
    for (int d = 1; d < n; d++)
        for (int i = 0, j = d; j < n; i++, j++)
            fs[i][j] = modular(fs[i + 1][j] - fs[i][j - 1], R) * invert(d, R) % R;
    while (q--) {
        ll x; scanf("%lld", &x);
        printf("%lld\n", f(x));
    }
    return 0;
}

```

## 5.2 数论

### 5.2.1 $O(n)$ 求逆元

```

inv[1]=1;
rep(i, MD){
    if (i < 2) continue;
    inv[i] = - MD / i * (LL) inv[MD % i] % MD;
    inv[i] = (inv[i] % MD + MD) % MD;
}

```



### 5.2.2 中国剩余定理(非互质)

```
LL exgcd(LL a,LL b,LL &x,LL &y) {
    if (!a){
        x = 0;
        y = 1;
        return b;
    }
    LL g = exgcd(b % a, a, x, y);
    LL t = y;
    y = x;
    x = t - (b / a) * y;
    return g;
}

LL CRT(const vector<LL>& m,const vector<LL>& b) {
    bool flag = false;
    LL x, y, i,d,result,a1,m1,a2,m2,Size=m.size();
    m1 = m[0]; a1 = b[0];
    for(i = 1; i < Size; ++i){
        m2 = m[i]; a2 = b[i];
        d = exgcd(m1, m2, x, y );
        if((a2-a1) % d != 0) flag = true;
        result = (x * ((a2-a1) / d ) % m2 + m2 ) % m2;
        a1 = a1 + m1 * result; //对于求多个方程
        m1 = (m1 * m2) / d;    //lcm(m1,m2)最小公倍数
        a1 = (a1 % m1 + m1) % m1;
    }
    if (flag) return -1;
    else return a1;
}
```

### 5.2.3 取模

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cmath>
using namespace std;

typedef long long ll;
typedef long long LL;

ll gcd(ll x, ll y) {
    return !y ? x : gcd(y, x % y);
}

ll modular(ll a, ll b) {
    return (a % b + b) % b;
}
```

```

}

/** a * x + b * y == gcd(a, b) */
LL exgcd(LL a, LL b, LL &x, LL &y) {
    if (!a) {
        x = 0;
        y = 1;
        return b;
    }
    LL g = exgcd(b % a, a, x, y);
    LL t = y;
    y = x;
    x = t - (b / a) * y;
    return g;
}

/** x * y % m == 1 */
ll invert(ll x, ll m) {
    ll a, b;
    exgcd(x, m, a, b);
    return modular(a, m);
}

/** x % m == a && x % n == b */
ll modular_system(ll m, ll a, ll n, ll b) {
    ll g, k, l;
    g = exgcd(m, n, k, l);
    if ((a - b) % g) return -1;
    k *= (b - a) / g;
    k = modular(k, n / g);
    return modular(k * m + a, m / g * n);
}

/** x % m[i] == r[i] */
ll modular_system_array(ll m[], ll r[], int k) {
    ll M = m[0], R = r[0];
    for (int i = 1; R != -1 && i < k; i++) {
        R = modular_system(M, R, m[i], r[i]);
        M = M / gcd(M, m[i]) * m[i];
    }
    return R;
}

/** a * x % m == b */
ll modular_equation(ll a, ll m, ll b) {
    return modular_system(m, b, a, 0) / a % m;
}

```

```

/** calculate  $r = x^y \% m$  */
ll modular_pow(ll x, ll y, ll m) {
    ll r = 1 % m;
    for (; y; y >>= 1, x = x * x % m)
        if (y & 1) r = r * x % m;
    return r;
}

//  $a^x = b \pmod n$ , n is prime
int mod_log(int a, int b, int n) {
    int m = (int)ceil(sqrt(n)), inv = Inv(mod_exp(a, m, n), n);
    id[0] = 0; mexp[0] = 1;
    for (int i = 1; i < m; ++i) id[i] = i, mexp[i] = (long long)mexp[i - 1] * a % n;
    sort(id, id + m, logcmp); sort(mexp, mexp + m);
    for (int i = 0; i < m; ++i) {
        int j = lower_bound(mexp, mexp + m, b) - mexp;
        if (j < m && mexp[j] == b) return i * m + id[j];
        b = (long long)b * inv % n;
    }
    return -1;
}

//  $A^x = B \pmod C$ 
// return x, -1 means no solution
int Dlog(int A, int B, int C) {
    map<int, int> Hash;
    int D = 1 % C;
    for(int j = D, i = 0; i <= 100; j = (long long)j * A % C, ++i)
        if (j == B) return i;
    int d = 0;
    for (int g; (g = gcd(A, C)) != 1; ) {
        if (B % g) return -1;
        ++d; C /= g; B /= g; D = (long long)D * A / g % C;
    }
    int M = (int)ceil(sqrt((double)C));
    for (int j = 1 % C, i = 0; i <= M; j = (long long)j * A % C, ++i)
        if (Hash.find(j) == Hash.end()) Hash[j] = i;
    for (int j = mod_exp(A, M, C), i = 0; i <= M; D = (long long)D * j % C, ++i) {
        int tmp = Inval(D, B, C);
        if (tmp >= 0 && Hash.find(tmp) != Hash.end()) return i * M + Hash[tmp] + d;
    }
    return -1;
}

```

#### 5.2.4 组合数计算

```

#include <cstdio>
#include <cmath>

```

```

#include <memory.h>

typedef long long T;

///Lib functions
T GCD(T a, T b) {
    return b ? GCD(b, a % b) : a;
}

T extendGCD(T a, T b, T & x, T & y) {
    if (!b)
        return x = 1, y = 0, a;
    T res = extendGCD(b, a % b, x, y), tmp = x;
    x = y, y = tmp - (a / b) * y;
    return res;
}

///for  $x^k$ 
T power(T x, T k) {
    T res = 1;
    while (k) {
        if (k & 1)
            res *= x;
        x *= x, k >>= 1;
    }
    return res;
}

///for  $x^k \bmod m$ 
T powerMod(T x, T k, T m) {
    T res = 1;
    while (x %= m, k) {
        if (k & 1)
            res *= x, res %= m;
        x *= x, k >>= 1;
    }
    return res;
}

/*****
Inverse in mod  $p^t$  system
*****/
T inverse(T a, T p, T t = 1) {
    T pt = power(p, t);
    T x, y;
    y = extendGCD(a, pt, x, y);
    return x < 0 ? x += pt : x;
}

```

```

/*****
Linear congruence theorem
x = a (mod p)
x = b (mod q)
for gcd(p, q) = 1, 0 <= x < pq
*****/
T linearCongruence(T a, T b, T p, T q) {
    T x, y;
    y = extendGCD(p, q, x, y);
    while (b < a)
        b += q / y;
    x *= b - a, x = p * x + a, x %= p * q;
    if (x < 0)
        x += p * q;
    return x;
}

/*****
prime table
O(n)
*****/
const int PRIMERANGE = 1000000;
int prime[PRIMERANGE + 1];
int getPrime() {
    memset(prime, 0, sizeof(int) * (PRIMERANGE + 1));
    for (int i = 2; i <= PRIMERANGE; i++) {
        if (!prime[i])
            prime[++prime[0]] = i;
        for (int j = 1; j <= prime[0] && prime[j] <= PRIMERANGE / i; j++) {
            prime[prime[j] * i] = 1;
            if (i % prime[j] == 0)
                break;
        }
    }
    return prime[0];
}

/*****
get factor of n
O(sqrt(n))
factor[][0] is prime factor
factor[][1] is factor generated by this prime
factor[][2] is factor counter

need: Prime Table
*****/
///you should init the prime table before

```

```

int factor[100][3], facCnt;
int getFactors(int x) {
    facCnt = 0;
    int tmp = x;
    for (int i = 1; prime[i] <= tmp / prime[i]; i++) {
        factor[facCnt][1] = 1, factor[facCnt][2] = 0;
        if (tmp % prime[i] == 0)
            factor[facCnt][0] = prime[i];
        while (tmp % prime[i] == 0)
            factor[facCnt][2]++, factor[facCnt][1] *= prime[i], tmp /= prime[i];
        if (factor[facCnt][1] > 1)
            facCnt++;
    }
    if (tmp != 1)
        factor[facCnt][0] = tmp, factor[facCnt][1] = tmp, factor[facCnt++][2] = 1;
    return facCnt;
}

```

```

/*****
Easy Combination for
C(n, k) that not exceeds limit of T
*****/
T combination(int n, int k) {
    T res = 1, g;
    if (k > n)
        return 0;
    if (n - k < k)
        k = n - k;
    for (int i = 0; i < k; i++) {
        if (res % (i + 1) == 0)
            res /= i + 1, res *= n - i;
        else if ((n - i) % (i + 1) == 0)
            res *= (n - i) / (i + 1);
        else {
            g = GCD(res, i + 1), res /= g;
            res *= (n - i) / ((i + 1) / g);
        }
    }
    return res;
}

```

```

/*****
C(n, k) mod m for k < 50,
O(k*k*lgk) m * m < T_MAX
*****/
T combinationModS(T n, T k, T m) {
    ///larger gate for more optimization
    ///too large gate may overflow somewhere

```

```

const T gate = 1LL << 50;
if (k > n || m == 1)
    return 0;
if (n - k < k)
    k = n - k;
T d[k], tmp = 1, i = 0, j, h, g;
for (i = 0, j = n - k + 1; n - i >= j; i++) {
    d[i] = n - i;
    while (gate / d[i] >= j && n - i != j)
        d[i] *= j, j++;
}
for (j = 2, h = k; j <= h; h--) {
    tmp *= h;
    while (gate / tmp >= j && h != j)
        tmp *= j, j++;
    for (int s = 0; tmp != 1; s++) {
        g = GCD(tmp, d[s]);
        d[s] /= g, tmp /= g;
    }
}
int res = 1;
for (j = 0; j < i; j++)
    d[j] %= m, res *= d[j], res %= m;
return res;
}

/*****
C(n, k) mod m using prime table for sieving
O(n) m * m < TMAX
limited by primetable, n could not be too large
need: prime table
*****/
T combinationModPri(T n, T k, T m) {
    if (k > n || m == 1)
        return 0;
    T result = 1, cnt = 0, temp;
    for (int i = 1; i < prime[0] && prime[i] <= n; i++) {
        temp = n, cnt = 0;
        while (temp)
            temp /= prime[i], cnt += temp;
        temp = n - k;
        while (temp)
            temp /= prime[i], cnt -= temp;
        temp = k;
        while (temp)
            temp /= prime[i], cnt -= temp;
        temp = prime[i];
        while (cnt) {

```

```

        if (cnt & 1)
            result *= temp, result %= m;
        temp *= temp, cnt >>= 1, temp %= m;
    }
    if (result == 0)
        return 0;
}
return result;
}

/*****
C(n, k) mod m
O(k*lgm) m * m < T_MAX
*****/
T combinationModN(T n, T k, T m) {
    if (k > n || m == 1)
        return 0;
    k = (n - k < k) ? n - k : k;
    int pcnt = 0;
    T a = 1, b = 1, x, y, g;
    T pa = 1, pb = 1; //may over flow
    for (int i = 1; i <= k; i++) {
        a *= n - i + 1, b *= k - i + 1;
        while ((g = GCD(a, m)) > 1)
            pa *= g, a /= g;
        while ((g = GCD(b, m)) > 1)
            pb *= g, b /= g;
        g = GCD(pa, pb), pa /= g, pb /= g;
        while (pa % m == 0)
            pa /= m, pcnt++;
        while (pb % m == 0)
            pb /= m, pcnt--;
        b %= m, a %= m;
    }
    a *= pa / pb, a %= m;
    while (pcnt)
        return 0;
    extendGCD(b, m, x, y);
    if (x < m)
        x += m;
    x *= a, x %= m;
    return x;
}

/*****
C(n, k) mod p
O(k) p*p <= TMAX
*****/

```



```

T combinationModP(T n, T k, T p) {
    if (k > n)
        return 0;
    if (n - k < k)
        k = n - k;
    T a = 1, b = 1, x, y;
    int pcnt = 0;
    for (int i = 1; i <= k; i++) {
        x = n - i + 1, y = i;
        while (x % p == 0)
            x /= p, pcnt++;
        while (y % p == 0)
            y /= p, pcnt--;
        x %= p, y %= p, a *= x, b *= y;
        b %= p, a %= p;
    }
    if (pcnt)
        return 0;
    extendGCD(b, p, x, y);
    if (x < 0)
        x += p;
    a *= x, a %= p;
    return a;
}

/*****
C(n, k) mod p^t
O(k*lgn/lgp) p^2t < TMAX
*****/
T combinationModPt(T n, T k, T p, T t) {
    if (k > n)
        return 0;
    if (n - k < k)
        k = n - k;
    T pt = power(p, t);
    T a = 1, b = 1, x, y;
    int pcnt = 0;
    for (int i = 1; i <= k; i++) {
        x = n - i + 1, y = i;
        while (x % p == 0)
            pcnt++, x /= p;
        while (y % p == 0)
            pcnt--, y /= p;
        x %= pt, y %= pt, a *= x, b *= y;
        a %= pt, b %= pt;
    }
    if (pcnt >= t)
        return 0;

```

```

    extendGCD(b, pt, x, y);
    if (x < 0)
        x += pt;
    a *= x, a %= pt;
    return a * power(p, pcnt) % pt;
}

/*****
C(n, k) mod m
O(k*lgn/lgp) m * m < TMAX
p is factor of m (depends on the smallest one)
need:
prime table
factor table
combinationModPt()
linearCongruence
*****/
//you need to init the prime table
T combinationModLi(T n, T k, T m) {
    if (k > n || m == 1)
        return 0;
    getFactors(m);
    T a, b, p, q;
    for (int i = 0; i < facCnt; i++) {
        if (!i)
            a = combinationModPt(n, k, factor[i][0], factor[i][2]), p = factor[i][1];
        else
            b = combinationModPt(n, k, factor[i][0], factor[i][2]), q = factor[i][1];
        if (!i)
            continue;
        a = linearCongruence(a, b, p, q), p *= q;
    }
    return a;
}

/*****
C(n, k) mod p
Lucas's theorem for combination mod p
O(p * lgn/lgp)
*****/
T lucas(T n, T k, T p) {
    T res = 1;
    while (n && k && res) {
        res *= combinationModP(n % p, k % p, p);
        res %= p, n /= p, k /= p;
    }
    return res;
}

```

```

/*****
a = n * (n - 1) * ... * (n - k + 1)
b = m * (m - 1) * ... * (m - k + 1)
c = ? from input
if a * c / b is an integer
this function will calculate this value module p^t
and the c input is moduled by p^t, be sure that gcd(c, p^t) = 1
0(len * lgn/lgp) , p^2t < TMAX
*****/
//the parameter &pcnt caches the factors consists of p
T productQuotient(T n, T m, T len, T p, T pt, T & c, T & pcnt) {
    if (!c || n < len)
        return c = 0;
    T & a = c, b = 1, x, y;
    for (int i = 1; i <= len; i++) {
        x = n - i + 1, y = m - i + 1;
        while (x % p == 0)
            x /= p, pcnt++;
        while (y % p == 0)
            y /= p, pcnt--;
        x %= pt, y %= pt, a *= x, b *= y;
        a %= pt, b %= pt;
    }
    extendGCD(b, pt, x, y);
    if (x < 0)
        x += pt;
    a *= x, a %= pt;
    return a;
}

/*****
C(n, k) mod p^t
generalized Lucas's theorem for combination mod p
0(p^t * lgn/lgp), p^2t < TMAX
*****/
T generalizedLucas(T n, T k, T p, T t) {
    if (k > n)
        return 0;
    if (n - k < k)
        k = n - k;
    if (t == 1)
        return lucas(n, k, p);
    T pt = power(p, t);
    T c = 1, pcnt = 0, ktable[100], ntable[100], ltable[100];
    int cnt = 0;
    for (; n || k; cnt++) {
        ktable[cnt] = k, ntable[cnt] = n, ltable[cnt] = k % pt;

```

```

        n -= k % pt, k -= k % pt, n /= p, k /= p;
    }
    for (--cnt; c && cnt >= 0; cnt--)
        productQuotient(n table[cnt], k table[cnt], l table[cnt], p, pt, c, pcnt);
    if (!c || pcnt >= t)
        return 0;
    return c * power(p, pcnt) % pt;
}

/*****
    C(n, k) mod m
    O(min(k, p^t * lgn/lgp)) m * m < TMAX
    p^t is fractor of m
need:
prime table
factor table
generalizedLucas
linearCongruence
*****/
///you need to init the prime table
T combinationModLucas(T n, T k, T m) {
    if (m == 1 || k > n)
        return 0;
    if (n - k < k)
        k = n - k;
    getFactors(m);
    T a, b, p, q;
    for (int i = 0; i < facCnt; i++) {
        if (!i)
            a = generalizedLucas(n, k, factor[i][0], factor[i][2]), p = factor[i][1];
        else
            b = generalizedLucas(n, k, factor[i][0], factor[i][2]), q = factor[i][1];
        if (!i)
            continue;
        a = linearCongruence(a, b, p, q), p *= q;
    }
    return a;
}

/*****
*****/
const T PTMAX = 10000;
T facmod[PTMAX];
void initFacMod(T p, T t = 1) {
    T pt = power(p, t);
    facmod[0] = 1 % pt;
    for (int i = 1; i < pt; i++) {
        if (i % p)

```

```

        facmod[i] = facmod[i - 1] * i % pt;
    else
        facmod[i] = facmod[i - 1];
    }
}

///you should init the facmod[] before
T factorialMod(T n, T & pcnt, T p, T t = 1) {
    T pt = power(p, t), res = 1;
    T stepCnt = 0;
    while (n) {
        res *= facmod[n % pt], res %= pt;
        stepCnt += n / pt, n /= p, pcnt += n;
    }
    res *= powerMod(facmod[pt - 1], stepCnt, pt);
    return res %= pt;
}

T combinationModPtFac(T n, T k, T p, T t = 1) {
    if (k > n || p == 1)
        return 0;
    if (n - k < k)
        k = n - k;
    T pt = power(p, t), pcnt = 0, pmcnt = 0;
    if (k < pt)
        return combinationModPt(n, k, p, t);
    initFacMod(p, t);
    T a = factorialMod(n, pcnt, p, t);
    T b = factorialMod(k, pmcnt, p, t);
    b *= factorialMod(n - k, pmcnt, p, t), b %= pt;
    pcnt -= pmcnt;
    if (pcnt >= t)
        return 0;
    a *= inverse(b, p, t), a %= pt;
    return a * power(p, pcnt) % pt;
}

T combinationModFac(T n, T k, T m) {
    getFactors(m);
    T a, b, p, q;
    for (int i = 0; i < facCnt; i++) {
        if (!i)
            a = combinationModPtFac(n, k, factor[i][0], factor[i][2]), p = factor[i][1];
        else
            b = combinationModPtFac(n, k, factor[i][0], factor[i][2]), q = factor[i][1];
        if (!i)
            continue;
        a = linearCongruence(a, b, p, q), p *= q;
    }
    return a;
}

```

```

}

/*****
C(n, k) mod m with generalizedLucas and combinationModFac
O(min(k, p^t))
*****/
T combinationModPtAuto(T n, T k, T p, T t) {
    if (t > 6)
        return combinationModPtFac(n, k, p, t);
    return generalizedLucas(n, k, p, t);
}
T combinationModOdds(T n, T k, T m) {
    if (k > n || m == 1)
        return 0;
    getFactors(m);
    T a, b, p, q;
    for (int i = 0; i < facCnt; i++) {
        if (!i)
            a = combinationModPtAuto(n, k, factor[i][0], factor[i][2]), p = factor[i][1];
        else
            b = combinationModPtAuto(n, k, factor[i][0], factor[i][2]), q = factor[i][1];
        if (!i)
            continue;
        a = linearCongruence(a, b, p, q), p *= q;
    }
    return a;
}

```

### 5.2.5 雅可比判别式 (二次剩余)

```

//n为合数, 判别不一定正确
int Jacobi(int a, int n) {
    if (a == 0) return 0;
    if (a == 1) return 1;
    int s, n1, a1 = a, e = 0;
    while (!(a1 & 1)) a1 >>= 1, ++e;
    if (!(e & 1)) s = 1;
    else {
        int u = n % 8;
        if (u == 1 || u == 7) s = 1;
        else s = -1;
    }
    if (n % 4 == 3 && a1 % 4 == 3) s = -s;
    n1 = n % a1;
    if (a1 == 1) return s;
    return s * Jacobi(n1, a1);
}

```

### 5.2.6 分解质因数

```
#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;

const int M = 1000010;

int n, m, dn;
int fp[M], prime[M], pn;
int p[M], k[M], d[M];

/* O(M) */
void init() {
    pn = 0;
    for (int i = 2; i < M; i++) {
        if (!fp[i]) fp[i] = prime[pn++] = i;
        for (int j = 0; prime[j] * i < M; j++) {
            fp[prime[j] * i] = prime[j];
            if (i % prime[j] == 0) break;
        }
    }
}

/* assumes that n < M */
void factorize1(int n, int p[], int k[], int &m) {
    for (m = 0; n > 1; m++) {
        p[m] = fp[n], k[m] = 0;
        while (n % p[m] == 0)
            n /= p[m], k[m]++;
    }
}

/* assumes that n < M * M */
void factorize(int n, int p[], int k[], int &m) {
    m = 0;
    for (int i = 0; n >= M && prime[i] * prime[i] <= n; i++)
        if (n % prime[i] == 0) {
            p[m] = prime[i], k[m] = 0;
            while (n % p[m] == 0)
                n /= p[m], k[m]++;
            m++;
        }
    if (n < M)
        for (; n > 1; m++) {
            p[m] = fp[n], k[m] = 0;
            while (n % p[m] == 0)
```

```

        n /= p[m], k[m]++;
    }
    if (n > 1)
        p[m] = n, k[m++] = 1;
}

void find_divisors(int p[], int k[], int m, int d[], int &dn) {
    dn = 0; d[dn++] = 1;
    for (int i = 0; i < m; i++) {
        for (int j = 0, z = dn; j < k[i] * z; j++, dn++)
            d[dn] = d[dn - z] * p[i];
    }
    sort(d, d + dn);
}

int main(){
    init();
    scanf("%d", &n);
    factorize(n, p, k, m);
    find_divisors(p, k, m, d, dn);
    for(int i = 0; i < dn; i++){
        printf("%d ", d[i]);
    }
    puts("");
    return 0;
}

```

### 5.2.7 大数分解

```

#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <algorithm>
#include <vector>
using namespace std;

typedef long long ll;

ll modular_multiply(ll x, ll y, ll m) {
    if (x < y) swap(x, y);
    ll r = 0;
    while (y) {
        if (y & 0x1)
            r = (r + x) % m;
        x = (x << 1) % m;
        y >>= 1;
    }
    return r;
}

```



```

}

ll modular_power(ll x, ll y, ll m) {
    ll r = 1;
    while (y) {
        if (y & 0x1)
            r = modular_multiply(r, x, m);
        x = modular_multiply(x, x, m);
        y >>= 1;
    }
    return r;
}

int miller_rabin(ll n, int k = 10) {
    if (n <= 3)
        return n > 1;
    while (k--) {
        ll a = rand() % (n - 3) + 2;
        if (modular_power(a, n - 1, n) != 1)
            return 0;
    }
    return 1;
}

ll gcd(ll x, ll y) {
    return !y ? x : gcd(y, x % y);
}

ll f(ll x, ll m, ll c) {
    return (modular_multiply(x, x, m) + c) % m;
}

ll pollard_rho(ll n) {
    if (!(n & 1))
        return 2;
    while (1) {
        ll x = rand() % n;
        ll y = x;
        ll c = rand() % n;
        ll d = 1;
        while (1) {
            x = f(x, n, c);
            y = f(f(y, n, c), n, c);
            d = gcd(y > x ? y - x : x - y, n);
            if (d == n) break;
            if (d > 1) return d;
        }
    }
}

```

```

}

void factorize(ll n, vector<ll> &r) {
if (miller_rabin(n)) {
    if (n != 1)
        r.push_back(n);
} else {
    ll d = pollard_rho(n);
    factorize(d, r);
    factorize(n / d, r);
}
}

void find_divisors(ll n, vector<ll> &divisors) {
vector<ll> factors;
factorize(n, factors);
sort(factors.begin(), factors.end());
divisors.clear(); divisors.push_back(1);
for (int i = 0, j = 0; i < factors.size(); i = j) {
    while (j < factors.size() && factors[i] == factors[j]) j++;
    ll p = factors[i];
    int k = j - i;
    int o = divisors.size();
    for (int z = 0; z < k * o; z++)
        divisors.push_back(divisors[divisors.size() - o] * p);
}
}

int main() {
    ll n;
    while (scanf("%llu", &n), n) {
        vector<ll> factors;
        factorize(n, factors);
        for (int i = 0; i < factors.size(); i++)
            printf("%llu ", factors[i]);
        puts("");
    }
    return 0;
}

```

### 5.2.8 米勒测试

```

LL mul_mod(LL x, LL y, LL n) {
    if (!x) return 0;
    return (((x & 1) * y) % n + (mul_mod(x >> 1, y, n) << 1) % n) % n;
}

LL pow_mod(LL a, LL x, LL n) {

```

```

    LL ret = 1;
    while (x) {
        if (x & 1) ret = mul_mod(ret, a, n);
        a = mul_mod(a, a, n); x >>= 1;
    }
    return ret;
}

bool millerRabin(LL a, LL n) {
    LL r = 0, s = n - 1;
    while (!(s & 1)) {
        s >>= 1;
        ++r;
    }
    LL x = pow_mod(a, s, n);
    if (x == 1 || x == n - 1) return 1;
    for (int j = 1; j < r; ++j) {
        x = mul_mod(x, x, n);
        if (x == 1) return 0;
        if (x == n - 1) return 1;
    }
    return 0;
}

/*
    if n < 1,373,653, it is enough to test a = 2 and 3;
    if n < 9,080,191, it is enough to test a = 31 and 73;
    if n < 4,759,123,141, it is enough to test a = 2, 7, and 61;
    if n < 2,152,302,898,747, it is enough to test a = 2, 3, 5, 7, and 11;
    if n < 3,474,749,660,383, it is enough to test a = 2, 3, 5, 7, 11, and 13;
    if n < 341,550,071,728,321, it is enough to test a = 2, 3, 5, 7, 11, 13, and 17.
*/
bool isPrime(LL n) {
    for (int i = 2; i < 1000 && i < n; ++i)
        if (n % i == 0) return 0;
    if (!millerRabin(2, n)) return 0;
    if (!millerRabin(3, n)) return 0;
    if (!millerRabin(5, n)) return 0;
    if (!millerRabin(7, n)) return 0;
    return 1;
}

```

### 5.2.9 pollard 分解质因数

```

const int limit = 1000000; //limit of brute-force
const int maxfn = 100;
const int maxL = 10;

```

```

bool b[limit];
long long p[limit];
pair <long long, int> f[maxfn];
long long n;
int pn, fn;

void init() {
    pn = 0;
    memset(b, 1, sizeof(b));
    for (int i = 2; i < limit; ++i)
        if (b[i]) {
            p[pn++] = i;
            for (int j = i + i; j < limit; j += i) b[j] = 0;
        }
}

long long mod_mul(long long a, long long b, long long n) {
    if (a <= 0x7fffffff && b <= 0x7fffffff) return a * b % n;
    long long len = 61, ret = 0;
    for (long long p = 8; p < n; len -=4, p <= 4);
    for (long long dig = (1LL << len) - 1; b > 0; b >>= len) {
        if (b & dig) ret = (ret + a * (b & dig)) % n;
        a = (a << len) % n;
    }
    return ret;
}

long long mod_exp(long long a, long long b, long long n) {
    long long ret = 1;
    while (b) {
        if (b & 1) ret = mod_mul(ret, a, n);
        a = mod_mul(a, a, n); b >>= 1;
    }
    return ret;
}

long long gcd(long long a, long long b) {
    if (!b) return a;
    else return gcd(b, a % b);
}

bool Miller_Rabin(long long n) {
    if (n < 2) return 0;
    if (n == 2) return 1;
    if (!(n & 1)) return 0;
    for (int i = 0; i < 20 && p[i] < n; ++i)
        if (mod_exp(p[i], n - 1, n) != 1) return 0;
    return 1;
}

```

```

}

void factor(long long n) {
    fn = 0;
    for (int i = 0; i < pn && p[i] * p[i] <= n; ++i)
        if (n % p[i] == 0) {
            int cnt = 0;
            while (n % p[i] == 0) n /= p[i], ++cnt;
            f[fn++] = make_pair(p[i], cnt);
        }
    if (n == 1) return;
    long long x = 5, y = 2, k = 1, l = 1;
    while (!Miller_Rabin(n)) {
        while (1) {
            long long g = gcd((y - x + n) % n, n);
            if (g == 1) {
                if ((--k) == 0) y = x, l <= 1, k = 1;
                x = (mod_mul(x, x, n) + 1) % n;
                continue;
            }
            int cnt = 0;
            while (n % g == 0) n /= g, ++cnt;
            f[fn++] = make_pair(g, cnt);
            if (n == g) return;
            n /= g; x %= n; y %= n;
            break;
        }
    }
    f[fn++] = make_pair(n, 1);
}

```

### 5.2.10 pell方程

```

//  $x^2 - D * y^2 = 1$ ,  $xn + yn * \sqrt{d} = (x0 + y0 * \sqrt{d})^n$ 
//  $x^2 - D * y^2 = -1$ ,  $xn + yn * \sqrt{d} = (x0 + y0 * \sqrt{d})^{(2*n+1)}$ 
//
//  $a * x^2 - b * y^2 = c$ 
// get x0, y0 from  $x^2 - a * b * y^2 = 1$ 
// get x1, y1 from  $a * x^2 - b * y^2 = c$ 
//  $[xk] = [x0, by0]^{k-1} * [x1]$ 
//  $[yk] = [ay0, x0] \quad [y1]$ 
//
bool pell( int D, int& x, int& y ) {
    int sqrtD = sqrt(D + 0.0);
    if( sqrtD * sqrtD == D ) return false;
    int c = sqrtD, q = D - c * c, a = (c + sqrtD) / q;
    int step = 0;
    int X[] = { 1, sqrtD };

```

```

int Y[] = { 0, 1 };
while( true ) {
    X[step] = a * X[step^1] + X[step];
    Y[step] = a * Y[step^1] + Y[step];
    c = a * q - c;
    q = (D - c * c) / q;
    a = (c + sqrtD) / q;
    step ^= 1;
    if( c == sqrtD && q == 1 && step ) {
        x = X[0], y = Y[0];
        return true;
    }
}
}

```

### 5.2.11 二次剩余, 解数

```

int Euler(int a, int p) {
    int ret = 1, s = a, k = (p - 1) / 2;
    while (k) {
        if (k & 1) ret = (long long)ret * s % p;
        s = (long long)s * s % p;
        k >>= 1;
    }
    if (ret != 1) ret = 0;
    else ret = 2;
    return ret;
}

```

```

int cal(int p, int n, int d) {
    int pn = 1;
    for (int i = 0; i < n; ++i) pn *= p;
    d %= pn;
    if (d == 0) {
        int k = 1;
        for (int i = 0; i < n / 2; ++i) k *= p;
        return k;
    }
    int r, b = 0, pr, pb;
    while (d % p == 0) {
        d /= p;
        ++b;
    }
    if (b % 2 != 0) return 0;
    r = b / 2;
    pr = 1;
    for (int i = 0; i < r; ++i) pr *= p;
    if (p == 2) {

```

```

        n -= b;
        if (n < 2) return 1 * pr;
        if (n == 2 && d % 4 == 1) return 2 * pr;
        if (n > 2 && d % 8 == 1) return 4 * pr;
        return 0;
    }
    return pr * Euler(d, p);
}

// x^2 = d (% m)
int QuadraticResidue(int m, int d) {
    int ret = 1;
    for (int i = 2; i * i <= m; ++i)
        if (m % i == 0) {
            int j = 0, q = 1;
            while (m % i == 0) {
                m /= i;
                ++j;
                q *= i;
            }
            ret *= cal(i, j, d);
        }
    if (m > 1) ret *= cal(m, 1, d);
    return ret;
}

```

### 5.2.12 二次剩余, 所有解

```

#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <vector>
using namespace std;

typedef vector<int> VI;
typedef vector< pair<int, int> > VII;
#define pb(a) push_back(a)

inline int longMod(int a, int b, int c) {
    return (long long)a * b % c;
}

int pm(int a, int b, int m) {
    int r = 1;
    for (; b; b >>= 1, a = longMod(a, a, m))
        if (b & 1)
            r = longMod(r, a, m);
    return r;
}

```

```

}

int ext_gcd(int a, int b, int &x, int &y) {
int r;
if (!b) {
    x = 1, y = 0;
    return a;
}
int d = ext_gcd(b, a % b, y, x);
y = y - a / b * x;
return d;
}

// solve  $x^2 \equiv r \pmod{p}$ 
// p is an odd prime
// and  $r^{((p-1)/1)} \equiv 1 \pmod{p}$ 
// r1 and r2 is the result
void getSqr(int r, int p, int &r1, int &r2) {
int S = 0, Q = p - 1;
while (!(Q & 1)) {
Q >>= 1;
S++;
}
if (S == 1) {
    r1 = pm(r, (p + 1) >> 2, p);
    r2 = p - r1;
    return;
}
int i, j, c, R, dt, t, M, z = 1;
do {
    z = rand() % p;
} while (pm(z, (p - 1) >> 1, p) != p - 1);
c = pm(z, Q, p);
R = pm(r, (Q + 1) >> 1, p);
t = pm(r, Q, p);
M = S;
while (t != 1) {
    for (i = 1, dt = longMod(t, t, p); dt != 1; i++)
        dt = longMod(dt, dt, p);
    for (j = M - i - 1; j > 0; j--)
        c = longMod(c, c, p);
    R = longMod(R, c, p);
    c = longMod(c, c, p);
    t = longMod(t, c, p);
    M = i;
}
r1 = R;
r2 = p - R;

```



```

}

// know the last digit of the result
// get the whole result of
//  $x^2 \equiv r \pmod{p^c}$ ;
// p is odd prime
int getDigit(int r, int p, int c, int last) {
    int ans = last, w = p;
    int i, pp, a, b;
    for (pp = 1, i = 1; i <= c; i++)
        pp *= p;
    int x, y;
    for (i = 2; i <= c; i++) {
        a = (last << 1) % p;
        b = (r + pp - longMod(last, last, pp)) % pp / w % p;
        ext_gcd(a, p, x, y);
        a = (p + longMod(b, x, p)) % p;
        last = last + w * a;
        w *= p;
    }
    return last;
}

// Solve  $x^2 \equiv r \pmod{p^c}$ 
// p is odd prime
//  $c \geq 1, r > 0$ 
void getPowerSol(int r, int p, int c, int &r1, int &r2) {
    int last;
    int i, j;
    getSqr(r, p, r1, r2);
    r1 = getDigit(r, p, c, r1);
    r2 = getDigit(r, p, c, r2);
}

// Solve  $x^2 \equiv r \pmod{2^c}$ 
//  $r > 0$ 
void getPowerSol2(int r, int c, VI &sol) {
    sol.clear();
    if (c == 1) {
        sol.pb(r & 1);
        return;
    }
    if (c == 0) {
        sol.pb(0);
        sol.pb(1);
        return;
    }
    if (c == 2) {

```

```

    int t = r & 3;
    if (t == 3 || t == 2)
        return;
    else if (t == 1) {
        sol.pb(1);
        sol.pb(3);
    } else {
        sol.pb(0);
        sol.pb(2);
    }
    return;
}
int pp = (1 << c) - 1;
r &= pp;
if (r == 0) {
    r = 1 << ((c + 1) >> 1);
    sol.pb(0);
    while (r < pp) {
        sol.pb(r);
        r <<= 1;
    }
    return;
}
if (r & 1) {
    if ((r & 7) != 1) {
        return;
    }
    int i, x, d, f;
    x = 1, f = 4;
    for (i = 4; i <= c; i++) {
        x = x + (((r - x * x) >> (i - 1)) & 1) << (i - 2));
        f <<= 1;
    }
    sol.pb(x);
    sol.pb(-x);
    sol.pb(-x - f);
    sol.pb(x + f);
    for (i = 0; i < 4; i++)
        sol[i] &= pp;
} else {
    int fac = 1;
    while (!(r & 1) && c > 3) {
        if (r & 3) {
            return;
        }
    }
    r >>= 2, fac <<= 1, c -= 2;
    }
    getPowerSol2(r, c, sol);

```

```

        for (int i = 0; i < sol.size(); i++) {
            sol[i] *= fac;
            sol[i] &= pp;
        }
    }
}

void divide(int n, VII &ans) {
    ans.clear();
    int i, j = 0;
    for (i = 2; i <= n / i; i++) {
        if (n % i == 0) {
            ans.pb(make_pair(i, 0));
            while (n % i == 0) {
                n /= i;
                ans[j].second++;
            }
            j++;
        }
    }
    if (n > 1)
        ans.pb(make_pair(n, 1));
}

// Using Chinese Remainder Theorem
// (m1,m2)=1
void updateRemain(const VI &org, int m1, int r2, int m2, VI &ans) {
    int x, y, M = m1 * m2, r1, R;
    ext_gcd(m1, m2, x, y);
    for (int i = 0; i < org.size(); i++) {
        r1 = org[i];
        R = (longMod(r2 * x, m1, M) + longMod(r1 * y, m2, M)) % M;
        ans.pb(R);
    }
}

void solve(int r, int n, VI &res) {
    res.clear();
    VII fac;
    r = (r % n + n) % n;
    static VI rs[2];
    rs[0].clear(), rs[1].clear();
    rs[0].pb(0);
    int i, d = 0, p, j, pp, curM = 1, r1, r2;
    if (n == 1) {
        res.pb(0);
        return;
    }
}

```

```

divide(n, fac);
static VI tmp;
for (i = 0; i < fac.size(); i++) {
    tmp.clear();
    p = 1;
    for (j = 0; j < fac[i].second; j++)
        p *= fac[i].first;
    if (fac[i].first == 2)
        getPowerSol2(r, fac[0].second, tmp);
    else {
        if (pm(r, (fac[i].first - 1) >> 1, fac[i].first) != 1)
            return;
        getPowerSol(r, fac[i].first, fac[i].second, r1, r2);
        tmp.pb(r1);
        tmp.pb(r2);
    }
    for (j = 0; j < tmp.size(); j++)
        updateRemain(rs[d], curM, tmp[j], p, rs[d ^ 1]);
    rs[d].clear();
    curM *= p;
    d ^= 1;
}
res.assign(rs[d].begin(), rs[d].end());
for (i = 0; i < res.size(); i++)
    res[i] = (res[i] % n + n) % n;
}

int main() {
    VI res;
    solve(2, 24, res);
    return 0;
}

```

## 5.3 博弈论

### 5.3.1 nim积

```

#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;

int cache[20][20];

int mulx(int, int);

int mul(int x, int y) {
    int sum = 0;
    for (int i = 0; (1 << i) <= x; i++) if (x & (1 << i))

```

```

    for (int j = 0; (1 << j) <= y; j++) if (y & (1 << j))
        sum ^= mulx(i, j);
    return sum;
}

int mulx(int i, int j) {
    if (i > j)
        swap(i, j);
    if (i == 0)
        return 1 << j;
    if (cache[i][j] != -1)
        return cache[i][j];
    int x = 1 << i;
    int y = 1 << j;
    int k;
    for (k = 30; !((1 << k) & i); k--);
    int z = 1 << (1 << k);
    if (x == z) {
        if ((i & j) == 0)
            return x * y;
        else
            return mul(x * 3 / 2, y / x);
    }
    return mul(mul(y, z), x / z);
}

void init() {
    memset(cache, -1, sizeof cache);
}

```

## 6 其他

### 6.0.2 高精度

```

#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <algorithm>
using namespace std;

const int MAXLEN = 110, BASE = 10000;

struct bignum {
    int d[MAXLEN / 4], n, s;
    bignum(int x = 0) {
        if (x < 0) s = 1, x = -x; else s = 0;
        for (n = 0; x; x /= BASE)
            d[n++] = x % BASE;
    }
};

```

```

}
bignum(const char *str) {
    if (isdigit(*str)) s = 0;
    else s = (*str++ == '-');

    int k = strlen(str);
    n = (k + 3) / 4;
    fill(d, d + n, 0);
    for (int i = n - 1, j = 0; j < k; j++) {
        d[i] = d[i] * 10 + str[j] - '0';
        if ((j + 1 - k) % 4 == 0) i--;
    }
    while (n > 0 && d[n - 1] == 0) n--;
}

int read() {
    char str[MAXLEN];
    if (scanf("%s", str) == EOF) return 0;
    *this = bignum(str);
    return 1;
}

bignum(const bignum& x) {
    n = x.n;
    s = x.s;
    copy(x.d, x.d + n, d);
}

static bignum exp(int len) {
    bignum C;
    C.n = len + 1;
    fill(C.d, C.d + len, 0);
    C.d[len] = 1;
    return C;
}

int length() {
    if (!n) return 0;
    int res = 4 * n - 4;
    for (int x = d[n - 1]; x; x /= 10) res++;
    return res;
}

void println() const {
    if (s && n) putchar('-');
    if (n > 0) {
        printf("%d", d[n-1]);
        for (int i = n - 2; i >= 0; i--)
            printf("%04d", d[i]);
    }
}

```

```

        printf("\n");
    } else puts("0");
}

};

bignum operator-(bignum A) {
    A.s ^= 1;
    return A;
}

int cmp(const bignum& A, const bignum& B) {
    if (!A.n && !B.n) return 0;
    if (!A.n) return B.s ? +1 : -1;
    if (!B.n) return A.s ? -1 : +1;
    if (A.s ^ B.s) return A.s ? -1 : +1;
    if (A.s && B.s) return cmp(-B, -A);
    if (A.n != B.n) return A.n - B.n;

    for (int i = A.n - 1; i >= 0; i--) {
        if (A.d[i] < B.d[i]) return -1;
        if (A.d[i] > B.d[i]) return +1;
    }
    return 0;
}

bool operator< (const bignum& A, const bignum& B) { return cmp(A, B) < 0; }
bool operator> (const bignum& A, const bignum& B) { return cmp(A, B) > 0; }
bool operator==(const bignum& A, const bignum& B) { return cmp(A, B) == 0; }
bool operator!=(const bignum& A, const bignum& B) { return cmp(A, B) != 0; }
bool operator<=(const bignum& A, const bignum& B) { return cmp(A, B) <= 0; }
bool operator>=(const bignum& A, const bignum& B) { return cmp(A, B) >= 0; }

bignum operator+(const bignum&, const bignum&);
bignum operator-(const bignum&, const bignum&);
bignum operator*(const bignum&, const bignum&);
bignum operator/(const bignum&, const bignum&);
bignum operator%(const bignum&, const bignum&);

bignum& operator+=(bignum& a, const bignum& b) { return a = a + b; }
bignum& operator-=(bignum& a, const bignum& b) { return a = a - b; }
bignum& operator*=(bignum& a, const bignum& b) { return a = a * b; }
bignum& operator/=(bignum& a, const bignum& b) { return a = a / b; }
bignum& operator%=(bignum& a, const bignum& b) { return a = a % b; }

bignum operator+(const bignum& A, const bignum& B) {
    if (A.s && B.s) return -((-A) + (-B));
    if (A.s) return B - (-A);

```

```

    if (B.s) return A - (-B);
    bignum C; C.n = max(A.n, B.n);
    int carry = 0;
    for (int i = 0; i < C.n; i++) {
        carry += (i < A.n ? A.d[i] : 0) + (i < B.n ? B.d[i] : 0);
        C.d[i] = carry % BASE;
        carry /= BASE;
    }
    if (carry) C.d[C.n++] = carry;
    return C;
}

```

```

bignum operator-(const bignum& A, const bignum& B) {
    if (A.s && B.s) return -((-A) - (-B));
    if (A.s) return -((-A) + B);
    if (B.s) return A + (-B);
    if (A < B) return -(B - A);

    bignum C; C.n = max(A.n, B.n);
    int carry = 0;
    for (int i = 0; i < C.n; i++) {
        carry += (i < A.n ? A.d[i] : 0) - (i < B.n ? B.d[i] : 0);
        C.d[i] = (carry + BASE) % BASE;
        carry = carry < 0 ? -1 : 0;
    }
    while (C.n > 0 && C.d[C.n - 1] == 0) C.n--;
    return C;
}

```

```

bignum operator*(const bignum& A, const bignum& B) {
    bignum C; C.n = A.n + B.n; C.s = A.s ^ B.s;
    fill(C.d, C.d + C.n, 0);
    for (int i = 0; i < A.n; i++)
        for (int j = 0; j < B.n; j++) {
            C.d[i + j] += A.d[i] * B.d[j];
            C.d[i + j + 1] += C.d[i + j] / BASE;
            C.d[i + j] %= BASE;
        }
    while (C.n > 0 && C.d[C.n - 1] == 0) C.n--;
    return C;
}

```

```

void div(bignum A, bignum B, bignum &Q, bignum &R) {
    bignum q, p;
    int s = A.s ^ B.s;
    A.s = B.s = 0;
    R = A;
    while (R >= B) {

```



```

        q = bignum::exp(max(R.n - B.n - 1, 0));
        p = q * B;
        if (p * BASE <= R) { p *= BASE; q *= BASE; }
        int u = 1;
        int v = BASE;
        while (u + 1 < v) {
            int k = (u + v) / 2;
            if (p * k <= R) u = k; else v = k;
        }
        Q += q * u;
        R -= p * u;
    }
    R.s = Q.s = s;
}

bignum operator/(const bignum& A, const bignum& B) {
    bignum Q, R;
    div(A, B, Q, R);
    return Q;
}

bignum operator%(const bignum& A, const bignum& B) {
    bignum Q, R;
    div(A, B, Q, R);
    return R;
}

char a[MAXLEN], b[MAXLEN];

int main() {
    while (~scanf("%s %s", a, b)) {
        bignum x(a);
        bignum y(b);
        bignum z = x + y;
        z.println();
    }
    return 0;
}

```

### 6.0.3 计算器

```

#include <cstdio>
#include <cstring>
#include <cctype>
#include <algorithm>
using namespace std;

const int INTMAX = 0x7fffffff;

```

```

typedef long long ll;

int x, ok;
struct Token { int t; ll val; } la;

inline bool fit(ll x) { return -INTMAX-1 <= x && x <= INTMAX; }

Token next_token() {
    while (isspace(x)) x = getchar();
    if (x == EOF) return (Token){EOF}; // XXX g++ only!!

    if (strchr("()+-*/%^", x)) {
        Token res = {x};
        x = getchar();
        return res;
    }

    ll z;
    for (z = 0; isdigit(x); x = getchar()) {
        z = z * 10 + x - '0';
        if (!fit(z)) ok = 0;
    }
    return (Token){'n', z};
}

int pow(int x, int y) {
    ll r = 1;
    if (x == 0 && y == 0 || y < 0) { ok = 0; return 1; }
    if (x == 0 || x == 1) return x;
    if (x == -1) y %= 2;

    while (y-- > 0) {
        r *= x;
        if (!fit(r)) { ok = 0; return 1; }
    }
    return r;
}

void shift() { la = next_token(); }

bool match(int t) {
    if (la.t == t) {
        shift();
        return 1;
    }
    return 0;
}

```

```

int exp();
int term();
int factor();
int unit();

int exp() {
    ll ans = term();
    while (1) {
        if (match('+')) {
            ans += term();
        } else if (match('-')) {
            ans -= term();
        } else break;

        if (!fit(ans)) ok = 0;
    }
    return ans;
}

int term() {
    ll t, ans = factor();
    while (1) {
        if (match('*')) {
            ans *= factor();
        } else if (match('/')) {
            if (t = factor())
                ans /= t;
            else
                ok = 0;
        } else if (match('%')) {
            if (t = factor())
                ans %= t;
            else
                ok = 0;
        } else break;
        if (!fit(ans)) ok = 0;
    }
    return ans;
}

int factor() {
    ll ans = unit();
    if (match('^'))
        ans = pow(ans, factor());
    return ans;
}

int unit() {

```

```

ll ans, sign;

sign = 1;
while (match('-'))
    sign *= -1;

if (la.t == 'n') {
    ans = sign * la.val;
    shift();
    if (!fit(ans)) ok = 0;
    return ans;
}

if (match('(')) {
    ans = sign * exp();
    if (!fit(ans)) ok = 0;
    if (match(')'))
        return ans;
}

ok = 0;
return 1;
}

int main() {
    x = getchar(); shift();

    while (la.t != EOF) {
        ok = 1;
        int val = exp();

        if (ok)
            printf("%d\n", val);
        else
            puts("ERROR!");
    }

    return 0;
}

```

#### 6.0.4 最长公共上升子序列

```

#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;

```

```

#define rep(i, n) for(int i = 0; i < (n); i++)

int n, m;
int a[1010], b[1010];
int f[1010];

int LCIS() {
    memset(f, 0, sizeof(f));
    rep(i, n){
        int k = 0;
        rep(j, m){
            if(a[i] == b[j]){//如果a[i]==b[j]
                if(f[j]<f[k]+1){//就在0到j-1之间, 找一个b[k]小于a[i]的f[k]值最大的解
                    f[j]=f[k]+1;
                }
            }
            if(a[i]>b[j]){//0到j-1中, 对于小于a[i]的, 保存f值的最优解
                if(f[k]<f[j]){
                    k=j;
                }
            }
        }
    }
    int ans=0;
    rep(i, m){
        ans=max(ans,f[i]);
    }
    return ans;
}

int main() {
    int t;
    scanf("%d", &t);
    while(t--) {
        scanf("%d",&n);
        rep(i, n){
            scanf("%d",&a[i]);
        }
        scanf("%d",&m);
        rep(j, m){
            scanf("%d",&b[j]);
        }
        printf("%d\n",LCIS());
        if (t) printf("\n");
    }
    return 0;
}

```

### 6.0.5 罗马数字

```
map <string, int, less <string> > dict;
char nums[5000][20];

void gen_roman() {
    char *roman[13] = {"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"};
    int arab[13] = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
    string key;

    for (int i = 0; i < 5000; ++i) {
        nums[i][0] = 0;
        for (int n = i, j = 0; n; ++j)
            for (; n >= arab[j]; n -= arab[j])
                strcat(nums[i], roman[j]);
        key = nums[i];
        dict[key] = i;
    }
}

char *to_roman(int n) {
    if (n < 1 || n >= 5000) return 0;
    return nums[n];
}

int to_arabic(char *in) {
    string key = in;
    if (!dict.count(key)) return -1;
    return dict[key];
}
```

### 6.0.6 最大子矩阵

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 3005;
int n, m;
bool a[N][N];
int lc[N], rc[N], tc[N], ll[N], rr[N];

int main() {
    int i, j, l, u, d, r, ans, tt;
    ans = 0;
    memset(tc, 0, sizeof(tc));
    memset(ll, 0x3f, sizeof(ll));
    memset(rr, 0x3f, sizeof(rr));
```

```

for (i = 1; i <= n; i++) {
    lc[0] = rc[m+1] = 0;
    for (j = m; j >= 1; j--)
        rc[j] = (a[i][j] ? rc[j+1]+1 : 0);
    for (j = 1; j <= m; j++) {
        if (!a[i][j]) {
            tc[j] = lc[j] = 0;
            ll[j] = rr[j] = 0x3f3f3f3f;
        } else {
            tc[j]++;
            lc[j] = lc[j-1] + 1;
            ll[j] = min(ll[j], lc[j]);
            rr[j] = min(rr[j], rc[j]);
            l = j - ll[j] + 1;
            r = j + rr[j] - 1;
            u = i - tc[j] + 1;
            d = i;
            tt = (r - l + 1) * (d - u + 1);
            ans = max(ans, tt);
        }
    }
}
printf("%d\n", ans);
return 0;
}

```

### 6.0.7 精确覆盖

```

#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;

const int N = 9;
const int B = 3;
const int R = N * N * N + 5;
const int C = N * N * 4 + 5;
const int Z = R * 4 + C + 5;

struct node {
    int x, y;
    node *l, *r, *u, *d;
};

node nodes[Z], *next, *root, *row[R], *col[C];
int size[C];
int ans[N][N];

```

```

void init(int r, int c) {
    next = nodes;
    memset(row, 0, sizeof row);
    memset(size, 0, sizeof size);
    root = next++;
    root->l = root->r = root;
    for (int y = 0; y < c; y++) {
        node *p = next++;
        p->x = -1, p->y = y;
        p->r = root, p->l = root->l;
        p->r->l = p->l->r = p;
        col[y] = p->u = p->d = p;
    }
}

/* BETTER add from top to bottom, from left to right */
node *add(int x, int y) {
    node *p = next++;
    p->x = x, p->y = y;
    size[y]++;
    if (!row[x]) {
        row[x] = p->l = p->r = p;
    } else {
        p->r = row[x];
        p->l = row[x]->l;
        p->r->l = p->l->r = p;
    }
    p->d = col[y];
    p->u = col[y]->u;
    p->u->d = p->d->u = p;
    return p;
}

void cover(int c) {
    node *x = col[c], *y, *z;
    x->l->r = x->r;
    x->r->l = x->l;
    for (y = x->d; y != x; y = y->d)
        for (z = y->r; z != y; z = z->r) {
            z->u->d = z->d;
            z->d->u = z->u;
            size[z->y]--;
        }
}

void uncover(int c) {
    node *x = col[c], *y, *z;
    for (y = x->u; y != x; y = y->u)

```



```

    for (z = y->l; z != y; z = z->l) {
        z->u->d = z;
        z->d->u = z;
        size[z->y]++;
    }
    x->l->r = x;
    x->r->l = x;
}

int dfs(int dep) {
    node *x, *y, *z = NULL;
    for (x = root->r; x != root; x = x->r)
        if (!z || size[x->y] < size[z->y]) z = x;
    if (!z) return 1;
    cover(z->y);
    for (x = z->u; x != z; x = x->u) {
        int r = x->x;
        ans[r / N / N][r / N % N] = r % N;
        for (y = x->r; y != x; y = y->r)
            cover(y->y);
        if (dfs(dep + 1)) return 1;
        for (y = x->l; y != x; y = y->l)
            uncover(y->y);
    }
    uncover(z->y);
    return 0;
}

int main() {
    init(N * N * N, 4 * N * N);
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            for (int d = 0; d < N; d++) {
                int b = i / B * B + j / B;
                add(i * N * N + j * N + d, 0 * N * N + i * N + j);
                add(i * N * N + j * N + d, 1 * N * N + i * N + d);
                add(i * N * N + j * N + d, 2 * N * N + j * N + d);
                add(i * N * N + j * N + d, 3 * N * N + b * N + d);
            }
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++) {
            scanf("%1d", &ans[i][j]);
            if (--ans[i][j] != -1) {
                int x = i * N * N + j * N + ans[i][j];
                node *z = row[x];
                do {
                    cover(z->y);
                } while ((z = z->r) != row[x]);
            }
        }
}

```

```

    }
}
if (dfs(0)) {
    for (int i = 0; i < N; i++, puts(""))
        for (int j = 0; j < N; j++)
            printf("%c", '1' + ans[i][j]);
} else
    puts("no solution!");
return 0;
}

```

### 6.0.8 模糊覆盖

```

#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;

const int N = 50;
const int R = N;
const int C = 64
const int Z = R * C;

struct node {
    int x, y;
    node *l, *r, *u, *d;
};

node nodes[Z], *next, *root, *row[R], *col[C];
int size[C];
bool mark[C];

void init(int r, int c) {
    next = nodes;
    memset(row, 0, sizeof row);
    memset(size, 0, sizeof size);
    root = next++;
    root->l = root->r = root;
    for (int y = 0; y < c; y++) {
        node *p = next++;
        p->x = -1, p->y = y;
        p->r = root;
        p->l = root->l;
        p->r->l = p->l->r = p;
        col[y] = p->u = p->d = p;
    }
}

```

```

/* MUST add from top to bottom, from left to right */
node *add(int x, int y) {
    node *p = next++;
    p->x = x, p->y = y;
    size[y]++;
    if (!row[x]) {
        row[x] = p->l = p->r = p;
    } else {
        p->r = row[x];
        p->l = row[x]->l;
        p->r->l = p->l->r = p;
    }
    p->d = col[y];
    p->u = col[y]->u;
    p->u->d = p->d->u = p;
    return p;
}

void cover(node *x) {
    for (node *y = x->d; y != x; y = y->d) {
        y->l->r = y->r;
        y->r->l = y->l;
        size[x->y]--;
    }
}

void uncover(node *x) {
    for (node *y = x->u; y != x; y = y->u) {
        y->l->r = y->r->l = y;
        size[x->y]++;
    }
}

int h() {
    int res = 0;
    node *x, *y, *z;
    memset(mark, 0, sizeof mark);
    for (x = root->l; x != root; x = x->l) if (!mark[x->y]) {
        mark[x->y] = 1;
        res++;
        for (y = x->u; y != x; y = y->u)
            for (z = y->r; z != y; z = z->r)
                mark[z->y] = 1;
    }
    return res;
}

int dfs(int dep) {

```

```

node *x, *y, *z = NULL;
if (dep < h()) return 0;
for (x = root->r; x != root; x = x->r)
    if (!z || size[x->y] < size[z->y]) z = x;
if (!z) return 1;
if (!dep) return 0;
for (x = z->u; x != z; x = x->u) {
    cover(x);
    for (y = x->r; y != x; y = y->r)
        cover(y);
    if (dfs(dep - 1)) return 1;
    for (y = x->l; y != x; y = y->l)
        uncover(y);
    uncover(x);
}
return 0;
}

```

### 6.0.9 生成N皇后一组解

```

#include <cstdio>
#include <algorithm>
#include <cstring>

using namespace std;

const int maxn = 300 + 10;

int board[maxn], diagl[maxn * 2], diagr[maxn * 2];
int n, conflict;

void compute() {
    conflict = 0;
    memset(diagl, 0, sizeof(diagl));
    memset(diagr, 0, sizeof(diagr));
    for (int i = 1; i <= n; ++i) {
        int r = i, c = board[i];
        ++diagl[r - c + n]; ++diagr[r + c - 1];
    }
    for (int i = 1; i < (n << 1); ++i) {
        if (diagl[i]) conflict += diagl[i] - 1;
        if (diagr[i]) conflict += diagr[i] - 1;
    }
}

void del(int r, int c, int& sum) {
    if (--diagl[r - c + n]) --sum;
    if (--diagr[r + c - 1]) --sum;
}

```

```

}

void ins(int r, int c, int& sum) {
    if (++diagl[r - c + n] > 1) ++sum;
    if (++diagr[r + c - 1] > 1) ++sum;
}

void NQueens() {
    for (int i = 1; i <= n; ++i) board[i] = i;
    while (1) {
        random_shuffle(board + 1, board + n + 1);
        compute();
        while (conflict) {
            int tot = 0;
            for (int i = 1; i <= n; ++i)
                for (int j = i + 1; j <= n; ++j) {
                    int k = conflict;
                    del(i, board[i], k); del(j, board[j], k);
                    ins(i, board[j], k); ins(j, board[i], k);
                    if (k < conflict) {
                        swap(board[i], board[j]);
                        conflict = k; ++tot;
                    }
                    else {
                        ins(i, board[i], k); ins(j, board[j], k);
                        del(i, board[j], k); del(j, board[i], k);
                    }
                }
            if (!tot) break;
        }
        if (!conflict) break;
    }
    for (int i = 1; i <= n; ++i) printf("%d\n", board[i]);
}

int main(){
    scanf("%d", &n);
    NQueens();
    return 0;
}

```

#### 6.0.10 sgu313

/\*

There are L stations along a circular railway, numbered 1 through L.  
 Trains travel in both directions, and take 1 minute to get from a station to the neighbouring one.  
 There are n employee's houses along the railway, and n offices, each house or office located near a station.  
 You are to establish a one-to-one correspondence between houses and offices in such a way that t

```

*/
#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;

const int maxn = 50000 + 10;

struct Tobj {
    int loc, sign;
};

Tobj obj[maxn * 2];
long long tab[maxn * 2];
int stack[maxn * 2];
int match[maxn];
int n, l;

bool cmp(Tobj i, Tobj j) {
    return i.loc < j.loc;
}

int main() {
    scanf("%d%d", &n, &l);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &obj[i].loc);
        obj[i].sign = i;
    }
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &obj[n + i].loc);
        obj[n + i].sign = -i;
    }
    sort(obj + 1, obj + n * 2 + 1, cmp);
    int p = n + 1, delta = 0, now = 0;
    long long sum = 0, len = 1 - (obj[n * 2].loc - obj[1].loc);
    tab[delta + p] = len;
    for (int i = 2; i <= n * 2; ++i) {
        if (obj[i - 1].sign > 0) ++now;
        else --now;
        len = obj[i].loc - obj[i - 1].loc;
        sum += len * abs(now);
        tab[now + delta + p] += len;
    }
    for (int i = -n; i <= n; ++i) tab[i + p] += tab[i - 1 + p];
    long long ans = -1;
    int broken;
    for (int i = 1; i <= n * 2; ++i) {

```

```

    if (ans == -1 || sum < ans) {
        ans = sum;
        broken = i;
    }
    if (obj[i].sign > 0) {
        sum -= (long long)(1 - tab[delta + p]) - tab[delta + p];
        ++delta;
    }
    else {
        sum += (long long)(1 - tab[delta - 1 + p]) - tab[delta - 1 + p];
        --delta;
    }
}
int top = 0, i = broken;
do {
    stack[top++] = obj[i].sign;
    while (top >= 2 && (stack[top - 1] > 0) != (stack[top - 2] > 0)) {
        if (stack[top - 1] > 0) match[stack[top - 1]] = -stack[top - 2];
        else match[stack[top - 2]] = -stack[top - 1];
        top -= 2;
    }
    if ((++i) > n * 2) i = 1;
} while (i != broken);
printf("%I64d\n", ans);
for (int i = 1; i <= n; ++i) {
    if (i > 1) printf(" ");
    printf("%d", match[i]);
}
printf("\n");
return 0;
}

```

### 6.0.11 MIPT015 最大矩阵

```

/*
In the square  $0 \leq x \leq 100$ ,  $0 \leq y \leq 100$  there are N points with integer coordinates.
You should find out rectangle which has no any of given point inside and has maximum possible area.
Remark: Points are permitted to be on the border of the rectangle.
*/
#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;

const int maxn = 100 + 10;

struct Tpoint {

```

```

    int x, y;
};

Tpoint p[maxn];
int y[maxn];
int n;

inline bool cmp(Tpoint i, Tpoint j) {
    return i.x < j.x;
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        scanf("%d%d", &p[i].x, &p[i].y);
        y[i] = p[i].y;
    }
    y[n] = 0; y[n + 1] = 100;

    int ans = 0;

    sort(y, y + n + 2);
    int m = unique(y, y + n + 2) - y;
    for (int i = 1; i < m; ++i)
        if (y[i] - y[i - 1] > ans) ans = y[i] - y[i - 1];
    ans *= 100;

    sort(p, p + n, cmp);
    for (int i = 0; i < n; ++i)
        if (p[i].x > 0 && p[i].x < 100 && p[i].y > 0 && p[i].y < 100) {
            int up = 100, down = 0;
            for (int j = i + 1; j < n; ++j)
                if (p[j].x > p[i].x && p[j].y < up && p[j].y > down) {
                    if ((p[j].x - p[i].x) * (up - down) > ans) ans = (p[j].x - p[i].x) * (up - down);
                    if (p[j].y > p[i].y) up = p[j].y;
                    else down = p[j].y;
                }
            if ((100 - p[i].x) * (up - down) > ans) ans = (100 - p[i].x) * (up - down);
        }

    reverse(p, p + n);
    for (int i = 0; i < n; ++i)
        if (p[i].x > 0 && p[i].x < 100 && p[i].y > 0 && p[i].y < 100) {
            int up = 100, down = 0;
            for (int j = i + 1; j < n; ++j)
                if (p[j].x < p[i].x && p[j].y < up && p[j].y > down) {
                    if ((p[i].x - p[j].x) * (up - down) > ans) ans = (p[i].x - p[j].x) * (up - down);
                    if (p[j].y > p[i].y) up = p[j].y;
                }
        }
}

```



```

        else down = p[j].y;
    }
    if (p[i].x * (up - down) > ans) ans = p[i].x * (up - down);
}

printf("%d\n", ans);

return 0;
}

```

### 6.0.12 MIPT016 最大不相交矩阵

/\*  
 Big rectangle A is defined via inequalities  $0 \leq x \leq \text{MAXX}$ ,  $0 \leq y \leq \text{MAXY}$ , where x and y are cart.  
 Rectangles B and C have sides parrallel to the sides of A. Intersection of B and C has null area.

Please, write program which can determine maximum sum area of the rectangles B and C.  
 \*/

```

#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;

const int maxn = 100 + 10;

struct Tpoint {
    double x, y;
};

Tpoint p[maxn];
double x[maxn], y[maxn], f[maxn], g[maxn];
double MAXX, MAXY;
int n;

inline bool cmpx(Tpoint i, Tpoint j) {
    return i.x < j.x;
}

inline bool cmpy(Tpoint i, Tpoint j) {
    return i.y < j.y;
}

int main() {
    scanf("%lf%lf%d", &MAXX, &MAXY, &n);
    for (int i = 0; i < n; ++i) {
        scanf("%lf%lf", &p[i].x, &p[i].y);
    }
}

```

```

    x[i] = p[i].x; y[i] = p[i].y;
}
x[n] = 0; x[n + 1] = MAXX;
y[n] = 0; y[n + 1] = MAXY;

sort(x, x + n + 2);
int m = unique(x, x + n + 2) - x;
double area1 = 0, area2 = 0;
for (int i = 1; i < m; ++i)
    if (x[i] - x[i - 1] > area1) area2 = area1, area1 = x[i] - x[i - 1];
    else if (x[i] - x[i - 1] > area2) area2 = x[i] - x[i - 1];
double ans = (area1 + area2) * MAXY;

sort(y, y + n + 2);
m = unique(y, y + n + 2) - y;
area1 = 0, area2 = 0;
for (int i = 1; i < m; ++i)
    if (y[i] - y[i - 1] > area1) area2 = area1, area1 = y[i] - y[i - 1];
    else if (y[i] - y[i - 1] > area2) area2 = y[i] - y[i - 1];
if ((area1 + area2) * MAXX > ans) ans = (area1 + area2) * MAXX;

memset(f, 0, sizeof(f));
memset(g, 0, sizeof(g));
sort(p, p + n, cmpx);
for (int i = 0; i < n; ++i)
    if (p[i].x > 0 && p[i].x < MAXX && p[i].y > 0 && p[i].y < MAXY) {
        double up = MAXY, down = 0, area = 0;
        for (int j = i + 1; j < n; ++j)
            if (p[j].x > p[i].x && p[j].y < up && p[j].y > down) {
                if ((p[j].x - p[i].x) * (up - down) > area) area = (p[j].x - p[i].x) * (up - down);
                if (p[j].y > p[i].y) up = p[j].y;
                else down = p[j].y;
            }
        if ((MAXX - p[i].x) * (up - down) > area) area = (MAXX - p[i].x) * (up - down);
        f[i] = area;

        up = MAXY; down = area = 0;
        for (int j = i - 1; j >= 0; --j)
            if (p[j].x < p[i].x && p[j].y < up && p[j].y > down) {
                if ((p[i].x - p[j].x) * (up - down) > area) area = (p[i].x - p[j].x) * (up - down);
                if (p[j].y > p[i].y) up = p[j].y;
                else down = p[j].y;
            }
        if (p[i].x * (up - down) > area) area = p[i].x * (up - down);
        g[i] = area;
    }

for (int i = n - 2; i >= 0; --i)

```

```

        if (f[i + 1] > f[i]) f[i] = f[i + 1];
for (int i = 1; i < n; ++i)
    if (g[i - 1] > g[i]) g[i] = g[i - 1];

for (int i = 0; i < n; ++i)
    if (f[i] + g[i] > ans) ans = f[i] + g[i];

memset(f, 0, sizeof(f));
memset(g, 0, sizeof(g));
sort(p, p + n, cmpy);
for (int i = 0; i < n; ++i)
    if (p[i].x > 0 && p[i].x < MAXX && p[i].y > 0 && p[i].y < MAXY) {
        double right = MAXX, left = 0, area = 0;
        for (int j = i + 1; j < n; ++j)
            if (p[j].y > p[i].y && p[j].x < right && p[j].x > left) {
                if ((p[j].y - p[i].y) * (right - left) > area) area = (p[j].y - p[i].y) * (right - left);
                if (p[j].x > p[i].x) right = p[j].x;
                else left = p[j].x;
            }
        if ((MAXY - p[i].y) * (right - left) > area) area = (MAXY - p[i].y) * (right - left);
        f[i] = area;

        right = MAXX; left = 0; area = 0;
        for (int j = i - 1; j >= 0; --j)
            if (p[j].y < p[i].y && p[j].x < right && p[j].x > left) {
                if ((p[i].y - p[j].y) * (right - left) > area) area = (p[i].y - p[j].y) * (right - left);
                if (p[j].x > p[i].x) right = p[j].x;
                else left = p[j].x;
            }
        if (p[i].y * (right - left) > area) area = p[i].y * (right - left);
        g[i] = area;
    }

for (int i = n - 2; i >= 0; --i)
    if (f[i + 1] > f[i]) f[i] = f[i + 1];
for (int i = 1; i < n; ++i)
    if (g[i - 1] > g[i]) g[i] = g[i - 1];

for (int i = 0; i < n; ++i)
    if (f[i] + g[i] > ans) ans = f[i] + g[i];

printf("%.2lf\n", ans);

return 0;
}

```

### 6.0.13 poj3016 可并堆

```
/*
    Given a sequence A of length n, separate it into k parts and modify each element so that
    each part is increasing or decreasing strictly, and minimize the cost.
    modify a1, a2, ..., am into b1, b2, ..., bm, the cost is |a1 - b1| + |a2 - b2| + ... + |am - bm|
*/

#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;

const int maxn = 2000 + 10;
const int maxk = 10 + 5;

struct node {
    int key, npl, left, right;
};

node heap[maxn];
int a[maxn], root[maxn], cnt[maxn];
int cost[maxn][maxn];
int f[maxk][maxn];
int n, k, m;

int merge(int a, int b, int typ) {
    if (!a) return b;
    if (!b) return a;
    if ((heap[a].key - heap[b].key) * typ < 0) swap(a, b);
    heap[a].right = merge(heap[a].right, b, typ);
    if (heap[heap[a].right].npl > heap[heap[a].left].npl) swap(heap[a].left, heap[a].right);
    heap[a].npl = heap[heap[a].right].npl + 1;
    return a;
}

int main() {
    while (scanf("%d%d", &n, &k), n || k) {
        for (int i = 1; i <= n; ++i) scanf("%d", &a[i]);
        heap[0].npl = -1;
        for (int i = 1; i <= n; ++i) {
            int x, y;
            m = x = y = 0;
            cnt[0] = i - 1;
            for (int j = i; j <= n; ++j) {
                heap[j].key = a[j] - j; heap[j].left = heap[j].right = heap[j].npl = 0;
                ++m; root[m] = cnt[m] = j; x += a[j] - j; y -= a[j] - j;
            }
        }
    }
}
```

```

        while (m > 1 && heap[root[m] - 1].key > heap[root[m]].key) {
            if ((cnt[m] - cnt[m - 1]) & 1) x -= a[root[m]] - root[m];
            if ((cnt[m - 1] - cnt[m - 2]) & 1) x -= a[root[m - 1]] - root[m - 1];
            root[m - 1] = merge(root[m], root[m - 1], 1); --m;
            if ((cnt[m + 1] - cnt[m - 1]) & 1) x += a[root[m]] - root[m];
            if ((cnt[m + 1] - cnt[m]) & 1 && (cnt[m] - cnt[m - 1]) & 1) {
                y += (a[root[m]] - root[m]) * 2;
                root[m] = merge(heap[root[m]].left, heap[root[m]].right, 1);
            }
            cnt[m] = cnt[m + 1];
        }
        cost[i][j] = x + y;
    }
    m = x = y = 0;
    cnt[0] = i - 1;
    for (int j = i; j <= n; ++j) {
        heap[j].key = a[j] + j; heap[j].left = heap[j].right = heap[j].npl = 0;
        ++m; root[m] = cnt[m] = j; x -= a[j] + j; y += a[j] + j;
        while (m > 1 && heap[root[m] - 1].key < heap[root[m]].key) {
            if ((cnt[m] - cnt[m - 1]) & 1) x += a[root[m]] + root[m];
            if ((cnt[m - 1] - cnt[m - 2]) & 1) x += a[root[m - 1]] + root[m - 1];
            root[m - 1] = merge(root[m], root[m - 1], -1); --m;
            if ((cnt[m + 1] - cnt[m - 1]) & 1) x -= a[root[m]] + root[m];
            if ((cnt[m + 1] - cnt[m]) & 1 && (cnt[m] - cnt[m - 1]) & 1) {
                y -= (a[root[m]] + root[m]) * 2;
                root[m] = merge(heap[root[m]].left, heap[root[m]].right, -1);
            }
            cnt[m] = cnt[m + 1];
        }
        if (x + y < cost[i][j]) cost[i][j] = x + y;
    }
}
memset(f, -1, sizeof(f));
f[0][0] = 0;
for (int i = 1; i <= k; ++i)
    for (int j = i; j <= n; ++j)
        for (int p = i - 1; p < j; ++p)
            if (f[i - 1][p] != -1)
                if (f[i][j] == -1 || f[i - 1][p] + cost[p + 1][j] < f[i][j]) f[i][j] = f[i - 1][p] + cost[p + 1][j];
printf("%d\n", f[k][n]);
}
return 0;
}

```