# ACM/ICPC Template Library

BJTU_Azwraith,
Beijing Jiao Tong University
2013.10

# 目录

# 图论

## 一 连通性问题

### 1. 有向图强连通分量 Tarjan O(N)

注意初始化，insert(int, int) 加入单向边

```
const int maxn = ____, maxm = ____;
struct edge
{
    int next, tar;
}e[maxm];
struct vtx
{
    int dfn, low, belong;
    bool instack;
}ver[maxn];
int hd[maxn], cnt, ind, sta[maxn], top, bcnt;
void insert(int a, int b)
{
    e[cnt].next = hd[a];  hd[a] = cnt;  e[cnt].tar = b; cnt++;
}

void tarjanSCC(int u)
{
    ver[u].dfn = ver[u].low = ind++;
    ver[u].instack = true;
    sta[top++] = u;
    for (int i = hd[u]; i != -1; i = e[i].next)
    {
        int v = e[i].tar;
        if (!ver[v].dfn)
        {
            tarjanSCC(v);
            ver[u].low = min(ver[u].low, ver[v].low);
        }
        if (ver[v].instack)
            ver[u].low = min(ver[u].low, ver[v].dfn);
    }

    if (ver[u].dfn == ver[u].low)
    {
        bcnt++;
```



```
        int j = -1;
        do
        {
            j = sta[--top];
            ver[j].belong = bcnt;
            ver[j].instack = false;
        }   while (j != u);
    }
}
void tarjanInit()
{
    memset(hd, -1, sizeof(hd));
    memset(ver, 0, sizeof(ver));
    ind = top = 1;
    cnt = 0;
}
```

### 2. 无向图边双连通分量 Tarjan O(N)

注意初始化

insert(int, int) 加入双向边

tarjan(int n) 调用

点的编号从 1 开始，双连通分量的编号也从 1 开始

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <queue>
#include <stack>
#include <ctime>
using namespace std;
/// 无向图 判断割顶 桥 求边双连通分量（环）
const int maxn = 200010, maxm = 1000010;
struct edge
{
    int next, s, t, isbridge;
}e[2 * maxm];
struct vtx
{
    int dfn, low, belong;
    bool iscut;
}ver[maxn];

int hd[maxn], ecnt, ind, bcnt;

/// insert(a, b) 创建双向边
```

```
void insert(int a, int b)
{
    e[ecnt].next = hd[a];  hd[a] = ecnt;  e[ecnt].s = a;  e[ecnt].t = b; ecnt++;
    e[ecnt].next = hd[b];  hd[b] = ecnt;  e[ecnt].s = b;  e[ecnt].t = a; ecnt++;
}

stack <int> sta;
/// 第一次调用时 fa = -1
void dfs(int u, int fa, int lastedge)
{
    ver[u].dfn = ver[u].low = ind++;
    int child = 0;
    for (int i = hd[u]; i != -1; i = e[i].next)
    {
        int v = e[i].t;

        if (!ver[v].dfn)
        {
            sta.push(v);
            child++;
            dfs(v, u, i);
            ver[u].low = min(ver[u].low, ver[v].low);
            if (ver[v].low >= ver[u].dfn)
                ver[u].iscut = true;
            if (ver[v].low > ver[u].dfn)
            {
                e[i].isbridge = true;
                ++bcnt;
                while (sta.top() != v)
                {
                    ver[sta.top()].belong = bcnt;
                    sta.pop();
                }
                ver[sta.top()].belong = bcnt;
                sta.pop();
            }
        }
        else if (ver[v].dfn < ver[u].dfn && lastedge != (i^1))
            ver[u].low = min(ver[u].low, ver[v].dfn);
    }
    if (fa < 0 && child == 1) ver[u].iscut = 0;
}

void tarjan(int n)
{
    for (int i = 1; i <= n; i++) if (!ver[i].dfn)
    {
        sta.push(i);
        dfs(1, -1, -1);
        if (!sta.empty())
        {
            bcnt++;
            while (!sta.empty())
            {
                ver[sta.top()].belong = bcnt;
                sta.pop();
            }
        }
    }
}

void init()
{
    memset(hd, -1, sizeof(hd));
    memset(ver, 0, sizeof(ver));
    memset(e, 0, sizeof(e));
    ind = 1;
    ecnt = bcnt = 0;
}
```

## 3. 无向图点双连通分量 Tarjan O(N) 【未验】

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <map>
#include <set>
#include <stack>
#include <vector>
#include <string>
#include <iostream>
#include <algorithm>

using namespace std;

typedef long long LL;
typedef vector<int> VI;
typedef pair<int,int> PII;

#define MP make_pair
#define PB push_back
```

```
#define eps 1e-8
#define inf 0x3f3f3f3f

#define Maxm 200010
#define Maxn 10010

struct edge {
    int u, v, next;
} e[Maxm];

int cnt, head[Maxn], n, m, block[Maxm];
void addedge(int u, int v) {
   e[cnt].u = u, e[cnt].v = v, e[cnt].next = head[u], head[u] = cnt ++ ;
   e[cnt].u = v, e[cnt].v = u, e[cnt].next = head[v], head[v] = cnt ++ ;
}
void Init() {
    int u, v;
    memset(head, -1, sizeof(head));
    cnt = 0;
    for (int i = 0; i < m; i ++ ) {
        scanf("%d%d", &u, &v);
        addedge(u, v);
    }
}
int deep, dfn[Maxn], low[Maxn], n1, iscut[Maxn];
stack<int> sta;
void tarjan(int u, int pre) {
    int v, child = 0;
    dfn[u] = low[u] = ++ deep;
    for (int i = head[u]; i != -1; i = e[i].next) {
        if (i == pre) continue;
        v = e[i].v;
        if (dfn[v] < dfn[u]) {
            sta.push(i);
            if (!dfn[v]) {
                child ++ ;
                tarjan(v, i ^ 1);
                low[u] = min(low[u], low[v]);
                if (low[v] >= dfn[u]) {
                    while (sta.top() != i) {
                        block[sta.top() / 2] = n1;
                        sta.pop();
                    }
                    block[i / 2] = n1 ++ ;
                    sta.pop();
                    iscut[u] = 1;
                }
            }
            else low[u] = min(low[u], low[v]);
        }
    }
    if (pre < 0 && child == 1) iscut[u] = -1;
}
VI lin[Maxn * 3];
int fa[Maxn * 3], sum[Maxn * 3], h[Maxn * 3];
bool vis[Maxn * 3], color[Maxn * 3];
void dfs(int t) {
    vis[t] = true;
    if (color[t])
        sum[t] ++ ;
    for (int i = 0; i < lin[t].size(); i ++ )
        if (!vis[lin[t][i]]) {
            fa[lin[t][i]] = t;
            sum[lin[t][i]] = sum[t];
            h[lin[t][i]] = h[t] + 1;
            dfs(lin[t][i]);
        }
}
void solve() {
    memset(dfn, 0, sizeof(dfn));
    memset(low, 0, sizeof(low));
    memset(iscut, -1, sizeof(iscut));
    memset(color, 0, sizeof(color));
    memset(sum, 0, sizeof(sum));
    memset(h, 0, sizeof(h));
    n1 = 0;
    for (int i = 1; i <= n; i ++ )
        if (!dfn[i]) {
            deep = 0;
            while (!sta.empty()) sta.pop();
            tarjan(i, -1);
        }
    for (int i = 1; i <= n; i ++ )
        if (iscut[i] == 1) {
            color[n1] = true;
            iscut[i] = n1 ++ ;
        }
    for (int i = 0; i < n1; i ++ )
        lin[i].clear();
    for (int i = 0; i < cnt / 2; i ++ )
        if (iscut[e[i * 2].u] == -1 && iscut[e[i * 2].v] == -1)
continue;
        else {
            int u, v;
            if (iscut[e[i * 2].u] != -1) {
                u = iscut[e[i * 2].u];
```

```
                v = block[i];
                lin[u].PB(v);
                lin[v].PB(u);
            }
            if (iscut[e[i * 2].v] != -1){
                u = iscut[e[i * 2].v];
                v = block[i];
                lin[u].PB(v);
                lin[v].PB(u);
            }
        }
    memset(vis, 0, sizeof(vis));
    for (int i = 0; i < n1; i ++ )
        if (!vis[i]) {
            fa[i] = -1;
            dfs(i);
        }
}
int anc[Maxn * 3][20];
void lca_prepare() {
    memset(anc, -1, sizeof(anc));
    for (int i = 0; i < n1; i ++ )
        anc[i][0] = fa[i];
    for (int j = 1; ; j ++ ) {
        bool flag = false;
        for (int i = 0; i < n1; i ++ ) {
            if (anc[i][j - 1] != -1)
                anc[i][j] = anc[anc[i][j - 1]][j - 1];
            if (anc[i][j] != -1)
                flag = true;
        }
        if (!flag) break;
    }
}
int query_lca(int u, int v) {
    int dif = abs(h[u] - h[v]), i = 0;
    if (h[u] < h[v]) swap(u, v);
    while (dif) {
        if (dif & 1) u = anc[u][i];
        i ++ ;
        dif >>= 1;
    }
    for (i = 19; i >= 0; i -- )
        if (anc[u][i] == anc[v][i]) continue;
        else u = anc[u][i], v = anc[v][i];
    if (u == v) return u;
    else return anc[u][0];
}
void query() {
    int q, u, v;
    scanf("%d", &q);
    for (int i = 0; i < q; i ++ ) {
        scanf("%d%d", &u, &v);
        u = block[u - 1];
        v = block[v - 1];
        int ans = sum[u] + sum[v];
        int lca = query_lca(u, v);
        if (fa[lca] != -1) ans -= sum[fa[lca]];
        ans -= sum[lca];
        printf("%d\n", ans);
    }
}
int main() {
    while (scanf("%d%d", &n, &m) != EOF) {
        if (n == 0 && m == 0) break;
        Init();
        solve();
        lca_prepare();
        query();
    }
    return 0;
}
```

## 4. 应用：2-sat

```
bool twoSat(int n)
{
    for (int i = 0; i < 2 * n; i++)
        if (!ver[i].dfn)   tarjanSCC(i);
    for (int i = 0; i < n; i++)
        if (ver[i * 2].belong == ver[i * 2 + 1].belong)
            return false;
    return true;
}

/// 用法参考下面的 main() 函数
/// 建图前, 先 tarjanInit() 建图时用 insert(u, v) u-->v
/// 对立点为 i 与 (i ^ 1)
int main()
{
    int m;
    while (~scanf("%d%d", &n, &m))
    {
        tarjanInit();
```

```
        for (int i = 0; i < m; i++)
        {
            int a1, a2, c1, c2;
            scanf("%d%d%d%d", &a1, &a2, &c1, &c2);
            insert(a1 + c1 * n, a2 + (1 - c2) * n); // x1 --> ~x2
            insert(a2 + c2 * n, a1 + (1 - c1) * n); // x2 --> ~x1
        }
        if (twoSat(n))
            printf("YES\n");
        else
            printf("NO\n");
    }
    return 0;
}
```

## 5．关于无向图连通性

[点连通度与边连通度]

在一个无向连通图中，如果有一个顶点集合，删除这个顶点集合，以及这个集合中所有顶点相关联的边以后，原图变成多个连通块，就称这个点集为**割点集合**。一个图的**点连通度**的定义为，最小割点集合中的顶点数。

类似的，如果有一个边集合，删除这个边集合以后，原图变成多个连通块，就称这个点集为**割边集合**。一个图的**边连通度**的定义为，最小割边集合中的边数。

**[双连通图、割点与桥]**

如果一个无向连通图的点连通度大于 1，则称该图是**点双连通的(point biconnected)**，简称**双连通**或**重连通**。一个图有割点，当且仅当这个图的点连通度为 1，则割点集合的唯一元素被称为**割点(cut point)**，又叫**关节点(articulation point)**。

如果一个无向连通图的边连通度大于 1，则称该图是**边双连通的(edge biconnected)**，简称双连通或重连通。一个图有桥，当且仅当这个图的边连通度为 1，则割边集合的唯一元素被称为**桥(bridge)**，又叫**关节边(articulation edge)**。

可以看出，点双连通与边双连通都可以简称为双连通，它们之间是有着某种联系的，下文中提到的双连通，均既可指点双连通，又可指边双连通。

**[双连通分支]**

在图 G 的所有子图 G'中，如果 G'是双连通的，则称 G'为**双连通子图**。如果一个双连通子图 G'它不是任何一个双连通子图的真子集，则 G'为**极大双连通子图**。**双连通分支(biconnected component)**，或**重连通分支**，就是图的极大双连通子图。特殊的，点双连通分支又叫做**块**。

**[求割点与桥]**

该算法是 R.Tarjan 发明的。对图深度优先搜索，定义 DFS(u)为 u 在搜索树（以下简称为树）中被遍历到的次序号。定义 Low(u)为 u 或 u 的子树中能通过非父子边追溯到的最早的节点，即 DFS 序号最小的节点。根据定义，则有：

Low(u)=Min { DFS(u) DFS(v) (u,v)为后向边(返祖边) 等价于 DFS(v)<DFS(u)且 v 不为 u 的父亲节点 Low(v) (u,v)为树枝边(父子边) }

一个顶点 u 是割点，当且仅当满足(1)或(2) (1) u 为树根，且 u 有多于一个子树。(2) u 不为树根，且满足存在(u,v)为树枝边(或称父子边，即 u 为 v 在搜索树中的父亲)，使得 DFS(u)<=Low(v)。

一条无向边(u,v)是桥，当且仅当(u,v)为树枝边，且满足 DFS(u)<Low(v)。

**[求双连通分支]**

下面要分开讨论点双连通分支与边双连通分支的求法。

对于点双连通分支，实际上在求割点的过程中就能顺便把每个点双连通分支求出。建立一个栈，存储当前双连通分支，在搜索图时，每找到一条树枝边或后向 边(非横叉边)，就把这条边加入栈中。如果遇到某时满足

DFS(u)<=Low(v)，说明 u 是一个割点，同时把边从栈顶一个个取出，直到遇到了边 (u,v)，取出的这些边与其关联的点，组成一个点双连通分支。割点可以属于多个点双连通分支，其余点和每条边只属于且属于一个点双连通分支。

对于边双连通分支，求法更为简单。只需在求出所有的桥以后，把桥边删除，原图变成了多个连通块，则每个连通块就是一个边双连通分支。桥不属于任何一个边双连通分支，其余的边和每个顶点都属于且只属于一个边双连通分支。

**[构造双连通图]**

一个有桥的连通图，如何把它通过加边变成边双连通图？方法为首先求出所有的桥，然后删除这些桥边，剩下的每个连通块都是一个双连通子图。把每个双连通子图收缩为一个顶点，再把桥边加回来，最后的这个图一定是一棵树，边连通度为 1。

统计出树中度为 1 的节点的个数，即为叶节点的个数，记为 leaf。则至少在树上添加(leaf+1)/2 条边，就能使树达到边二连通，所以至少添加 的边数就是(leaf+1)/2。具体方法为，首先把两个最近公共祖先最远的两个叶节点之间连接一条边，这样可以把这两个点到祖先的路径上所有点收缩到一 起，因为一个形成的环一定是双连通的。然后再找两个最近公共祖先最远的两个叶节点，这样一对一对找完，恰好是(leaf+1)/2 次，把所有点收缩到了一 起。

# 二 网络流和匹配

## 1. 最大流 Dinic

```
typedef _____ Ftype; // 流的数据类型
const int maxn = _____, maxm = _____;
const Ftype inf = _____;

// 使用之前必须 init(n)
struct FlowNetwork
{
    struct edge
    {
        int s, t, nxt;
        Ftype c, f;
    } e[maxm];

    int n, m, s, t;
    int hd[maxn], cur[maxn], dep[maxn];
    Ftype max_flow;
    bool vis[maxn];
    void init(int n)
    {
        this->n = n;
        memset(hd, -1, sizeof hd);
        m = max_flow = 0;
    }
    void AddEdge(int from, int to, Ftype cap)
    {
        e[m] = (edge){from, to, hd[from], cap, 0};hd[from] = m; m++;
        e[m] = (edge){to, from, hd[to], 0, 0};hd[to] = m; m++;
    }
    bool BFS()
    {
        memset(vis, 0, sizeof(vis));
        queue <int> Q;
        Q.push(s);
        dep[s] = 0, vis[s] = 1;
        while (!Q.empty())
        {
            int x = Q.front();
            Q.pop();
            for (int i = hd[x]; i != -1; i = e[i].nxt)
            {
                if (!vis[e[i].t] && e[i].c > e[i].f)
                {
                    vis[e[i].t] = 1;
                    dep[e[i].t] = dep[x] + 1;
                    Q.push(e[i].t);
                }
            }
        }
        return vis[t];
    }

    int DFS(int x, Ftype a)
    {
        if (x == t || a == 0)    return a;
```

```
        Ftype flow = 0, f;
        for (; cur[x] != -1; cur[x] = e[cur[x]].nxt)
        {
            int i = cur[x];
            if (dep[x] + 1 == dep[e[i].t]
&& (f = DFS(e[i].t, min(a, e[i].c - e[i].f))) > 0)
            {
                e[i].f += f;
                e[i^1].f -= f;
                flow += f;
                a -= f;
                if (a == 0)    break;
            }
        }
        return flow;
    }

    vector <int> minCut() // 最小割边集
    {
        BFS();
        vector <int> ans;
        for (int i = 0; i < m; i += 2)
            if (vis[e[i].s] && !vis[e[i].t]) ans.push_back(i);
        return ans;
    }

    Ftype maxFlow(int s, int t) // 最大流
    {
        this->s = s; this->t = t;
        Ftype flow = 0;

        while (BFS())
        {
            memcpy(cur, hd, sizeof hd);
            flow += DFS(s, inf);
        }
        return max_flow += flow;
    }
    void reset() // 重置（不删边）
    {
        max_flow = 0;
        for (int i = 0; i < m; i++) e[i].f = 0;
    }
};
```



## 2. 最小费用最大流 SPFA 增广（稀疏图较快）

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

const int maxn = _____, maxm = _____;
typedef _____ Ftype;
Ftype inf = _____;
struct edge
{
    int from, to;
    Ftype cap, flow, cost;
    int next;
};

struct MCMF
{
    int n, m, s, t, hd[maxn], inq[maxn], p[maxn];
    edge e[maxm];
    Ftype a[maxn], d[maxn];

    void init(int n)
    {
        this->n = n;
        memset(hd, -1, sizeof hd);
        m = 0;
    }
    void addEdge(int from, int to, Ftype cap, Ftype cost)
    {
        e[m] = (edge){from, to, cap, 0, cost, hd[from]}; hd[from] =
m; m++;
        e[m] = (edge){to, from, 0, 0, -cost, hd[to]}; hd[to] = m; m+
+;
    }

    bool SPFA(int s, int t, Ftype &flow, Ftype &cost)
    {
        for (int i = 0; i < n; i++) d[i] = inf;
        memset(inq, 0, sizeof inq);
        d[s] = 0, inq[s] = 1, p[s] = 0, a[s] = inf;
        deque <int> Q;
        Q.push_back(s);
        while (!Q.empty())
```

```
        {
            int u = Q.front(); Q.pop_front();
            inq[u] = false;
            for (int i = hd[u]; i != -1; i = e[i].next)
            {
                if (e[i].cap > e[i].flow && d[e[i].to] > d[u] +
e[i].cost)
                {
                    d[e[i].to] = d[u] + e[i].cost;
                    p[e[i].to] = i;
                    a[e[i].to] = min(a[u], e[i].cap - e[i].flow);
                    if (!inq[e[i].to])
                    {
                        if (d[e[i].to] <= d[u])
Q.push_front(e[i].to);
                        else Q.push_back(e[i].to);
                        inq[e[i].to] = true;
                    }
                }
            }
        }
        if (d[t] == inf) return false;
        flow += a[t];
        cost += d[t] * a[t];
        int u = t;
        while (u != s)
        {
            e[p[u]].flow += a[t];
            e[p[u]^1].flow -= a[t];
            u = e[p[u]].from;
        }
        return true;
    }
    Ftype minCost(int s, int t)
    {
        Ftype flow = 0, cost = 0;
        while (SPFA(s, t, flow, cost))
            ;
        return cost;
    }
};
```

## 3. 二分图最大匹配 匈牙利算法 O(NM)

```
#include <iostream>
#include <cstdio>
#include <cstring>
const int maxn = 1010, maxm = 510*510;
struct edge
{
    int s, t, next;
} e[maxm];
struct vtx
{
    int used, match;
}ver[maxn];
int hd[maxn], ecnt;
bool maxMatchDFS(int v)
{
    for (int i = hd[v]; i != -1; i = e[i].next) if(!ver[u].used)
    {
        int u = e[i].t, w = ver[e[i].t].match;
        ver[u].used = true;
        if (w == -1 || (!ver[w].used && maxMatchDFS(w)))
        {
            ver[v].match = u, ver[u].match = v;
            return true;
        }
    }
    return false;
}
int maxMatch(int n)
{
    int match = 0;
    for (int i = 1; i <= n; i++)
    {
        for (int j = n+1; j <= (n<<1); j++) ver[j].used = 0;
        if (maxMatchDFS(i)) match++;
    }
    return match;
}
void init(int n)
{
    memset(hd, -1, sizeof hd);
    ecnt = 0;
    for (int i = 1; i <= (n<<1); i++)
        ver[i].match = -1, ver[i].used = false;
}
void addEdge(int a, int b)
{
    e[ecnt].s = a, e[ecnt].t = b; e[ecnt].next = hd[a];
                                            hd[a] = ecnt++;
}
```

## 4．混合图欧拉回路建边

**基础知识**

欧拉回路是图 G 中的一个回路，经过每条边有且仅一次，称该回路为欧拉回路。具有欧拉回路的图称为欧拉图，简称 E 图。

无向图中存在欧拉回路的条件：每个点的度数均为偶数。

有向图中存在欧拉回路的条件：每个点的入度=出度。

欧拉路径比欧拉回路要求少一点：

无向图中存在欧拉路径的条件：每个点的度数均为偶数或者有且仅有 2 个度数为奇数的点。

有向图中存在欧拉路径的条件：除了 2 个点外，其余的点入度=出度，且在这 2 个点中，一个点的入度比出度大 1，另一个出度比入度大 1。

欧拉路径的输出：经典的套圈算法。

下面来重点讲讲混合图的欧拉回路问题。

混合图就是边集中有有向边和无向边同时存在。这时候需要用网络流建模求解。

建模：

把该图的无向边随便定向，计算每个点的入度和出度。如果有某个点出入度之差为奇数，那么肯定不存在欧拉回路。 因为欧拉回路要求每点入度 = 出度，也就是总度数为偶数，存在奇数度点必不能有欧拉回路。

好了，现在每个点入度和出度之差均为偶数。那么将这个偶数除以 2，得 x。也就是说，对于每一个点，只要将 x 条边改变方向（入>出就是变入，出>入就是变出），就能保证出 = 入。如果每个点都是出 = 入，那么很明显，该图就存在欧拉回路。

现在的问题就变成了：我该改变哪些边，可以让每个点出 = 入？构造网络流模型。

首先，有向边是不能改变方向的，要之无用，删。一开始不是把无向边定向了吗？定的是什么向，就把网络构建成什么样，边长容量上限 1。另新建 s 和 t。对于入 > 出的点 u，连接边(u, t)、容量为 x，对于出 > 入的点 v，连接边(s, v)，容量为 x（注意对不同的点 x 不同）。

之后，察看从 S 发出的所有边是否满流。有就是能有欧拉回路，没有就是没有。欧拉回路是哪个？察看流值分配，将所有流量非 0（上限是 1，流值不是 0 就是 1）的边反向，就能得到每点入度 = 出度的欧拉图。

由 于是满流，所以每个入 > 出的点，都有 x 条边进来，将这些进来的边反向，OK，入 = 出了。对于出 > 入的点亦然。那么，没和 s、t 连接的点怎么办？和 s 连接的条件是出 > 入，和 t 连接的条件是入 > 出，那么这个既没和 s 也没和 t 连接的点，自然早在开始就已经满足入 = 出了。那么在网络流过程中，这些点属于“中间点”。我们知道中间点流量不允许有累积的，这样，进去多少就出来多少，反向之后，自然仍保持平衡。

所以，就这样，混合图欧拉回路问题，解了。

例：HDU3472

题意：给出一些单词，其中有些单词反转之后也是有意义的单词，问是否能将所有单词首尾相连，每个单词用 1 次且仅用 1 次。

解：这题是混合路的欧拉路径问题。

1.首先判断图的连通性，若不连通，无解。

2.然后任意定向无向边，计算每个点 i 的入度和出度之差 deg[i]。若 deg[i]为奇数，无解。

3.设立源点 s 和汇点 t，若某点 i 入度<出度，连边(s,i,-deg[i]/2)，若入度>出度，连边(i,t,deg[i]/2)；对于任意定向的无向边(i,j,1)。

4.若有两个度数为奇数的点，假设存在欧拉路径，添加一条容量为 1 的边，构成欧拉回路，不影响结果。若全为偶数，直接最大流。

5.若从 S 发出的边全部满流，证明存在欧拉回路(路径)，否则不存在。

ps：若要求输出路径，将网络中有(无)流量的边反向，加上原图的有向边，用套圈算法即可。

## 5．全局最小割

```
1 #include <iostream>
2 #include <cstdio>
3 #include <cstring>
4 #include <algorithm>
5 using namespace std;
6 const int INF = 0x3f3f3f3f;
7 const int maxn = 510;
```

```
 8 struct UMinCut
 9 {
10     int c[maxn][maxn], n;
11     int cut;
12     int id[maxn], b[maxn];
13     void init(int _n)
14     {
15         memset(c, 0, sizeof c);
16         n = _n;
17     }
18     int minCut()
19     {
20         for (int i = 0; i < n; i++) id[i] = i;
21         cut = INF;
22         for (; n > 1; n--)
23         {
24             memset(b, 0, sizeof b);
25             for (int i = 0; i + 1 < n; i++)
26             {
27                 int p = i + 1;
28                 for (int j = i + 1; j < n; j++)
29                 {
30                     b[id[j]] += c[id[i]][id[j]];
31                     if (b[id[p]] < b[id[j]]) p = j;
32                 }
33                 swap(id[i+1], id[p]);
34             }
35             cut = min(cut, b[id[n-1]]);
36             for (int i = 0; i < n-2; i++)
37             {
38                 c[id[i]][id[n-2]] += c[id[i]][id[n-1]];
39                 c[id[n-2]][id[i]] += c[id[n-1]][id[i]];
40             }
41         }
42         return cut;
43     }
44 } ccc;
45
46 int main()
47 {
48     int n, m;
49
50     while (~scanf("%d%d", &n, &m))
51     {
52         ccc.init(n);
53         for (int i = 0; i < m; i++)
54         {
55             int u, v, w;
56             scanf("%d%d%d", &u, &v, &w);
57             ccc.c[u][v] += w;
58             ccc.c[v][u] += w;
59         }
60         printf("%d\n", ccc.minCut());
61     }
62
63     return 0;
64 }
```

# 三 最短路

## 1. Dijkstra 复杂度上限 O(M log N)

```
1  memset(use, 0x3f, sizeof(use));
2  priority_queue < pair < int, int > > q;
3  use[s] = 0;
4  q.push(mp(0, s));
5  while (!q.empty())
6  {
7      int cur = q.top().B;
8      q.pop();
9      if (use[cur] != -q.top().A)
10         continue;
11     for (unsigned i = 0; i < g[cur].size(); i++)
12         if (use[g[cur][i].B] > use[cur] + g[cur][i].A)
13         {
14             use[g[cur][i].B] = use[cur] + g[cur][i].A;
15             q.push(mp(-use[g[cur][i].B], g[cur][i].B));
16         }
17 }
```

## 2. SPFA 估计复杂度 O(2 M) 带 SLF 优化

```
1  deque < int > q;
2  use[s] = 0;
3  q.push_back(s);
4  memset(inq, 0, sizeof(inq));
5  inq[s] = true;
6  while (!q.empty())
7  {
8      int cur = q.front();
9      inq[cur] = false;
```

```
10      q.pop_front();
11      for (unsigned i = 0; i < g[cur].size(); i++)
12          if (use[g[cur][i].B] > use[cur] + g[cur][i].A)
13          {
14              use[g[cur][i].B] = use[cur] + g[cur][i].A;
15              if (!inq[g[cur][i].B])
16              {
17                  if (!q.empty() && use[g[cur][i].B] <=
18  use[q.front()])
19                      q.push_front(g[cur][i].B);
20                  else q.push_back(g[cur][i].B);
21                  inq[g[cur][i].B] = true;
22              }
23          }
24  }
```

### 3．差分约束系统

根据二元约束关系建图，设源点 D[s] = 0.

| 约束关系表示 | 问题 | 算法 |
| --- | --- | --- |
| D[j] – D[i] <= C | 约束关系下的最大值 | 最短路 |
| D[j] – D[i] >= C | 约束关系下的最小值 | 最长路 |

### 4．次短路径

次短路径可以看作是 k 短路径问题的一种特殊情况，求 k 短路径有 Yen 算法等较为复杂的方法，对于次短路径，可以有更为简易的方法。下面介绍一种求两个顶点之间次短路径的解法。

我们要对一个有向赋权图(无向图每条边可以看作两条相反的有向边)的顶点 S 到 T 之间求次短路径，首先应求出 S 的单源最短路径。遍历有向图，标记出可以在最短路径上的边，加入集合 K。然后枚举删除集合 K 中每条边，求从 S 到 T 的最短路径，记录每次求出的路径长度值，其最小值就是次短路径的长度。

在这里我们以为次短路径长度可以等于最短路径长度，如果想等，也可以看作是从 S 到 T 有不止一条最短路径。如果我们规定求从 S 到 T 大于最短路径长度的次短路径，则答案就是每次删边后大于原最短路径的 S 到 T 的最短路径长度的最小值。

用 Dijkstra+堆求单源最短路径，则每次求最短路径时间复杂度为 O(Nlog(N+M) + M)，所以总的时间复杂度为 O(NM*log(N+M) + M^2)。该估计是较为悲观的，因为一般来说，在最短路径上的边的条数要远远小于 M，所以实际效果要比预想的好。

# 四 生成树

## 次小生成树

类比上述次短路径求法，很容易想到一个“枚举删除最小生成树上的每条边，再求最小生成树”的直观解法。如果用 Prim+堆，每次最小生成树时间复杂度为 O(N*log(N+M) + M)，枚举删除有 O(N) 条边，时间复杂度就是 O(N^2log(N+M) + N*M)，当图很稠密时，接近 O(N^3)。这种方法简易直观，但我们有一个更简单，而且效率更高的 O(N^2+M)的解法，下面介绍这种方法。

首先求出原图最小生成树，记录权值之和为 MinST。枚举添加每条不在最小生成树上的边(u,v)，加上以后一定会形成一个环。找到环上权值第二大的边(即除了(u,v)以外的权值最大的边)，把它删掉，计算当前生成树的权值之和。取所有枚举修改的生成树权值之和的最小值，就是次小生成树。

具体实现时，更简单的方法是从每个节点 i 遍历整个最小生成树，定义 F[j]为从 i 到 j 的路径上最大边的权值。遍历图求出 F[j]的值，然后对于添加每条不在最小生成树中的边(i,j)，新的生成树权值之和就是 MinST + w(i,j) - F[j]，记录其最小值，则为次小生成树。

该算法的时间复杂度为 O(N^2 + M)。由于只用求一次最小生成树，可以用

最简单的 Prim，时间复杂度为 O(N^2)。算法的瓶颈不在求最小生成树，而在 O(N^2+M)的枚举加边修改，所以用更好的最小生成树算法是没有必要的。

## 五 树的直径

（调用时语句：dis[bfs(bfs(1))]; 注意初始化等问题）

```
const int maxn = 100010;
vector<int> g[maxn];
int dis[maxn], vis[maxn];

int bfs(int start)
{
    queue <int> q;
    q.push(start);
    vis[start] = 1;
    memset(dis, 0x3f, sizeof dis);
    memset(vis, 0, sizeof vis);
    dis[start] = 0;
    int p = 0;
    while (!q.empty())
    {
        int cur = q.front();
        p = cur;
        q.pop();
        for (int i = 0; i < (int)g[cur].size(); i++)
        {
            if (!vis[g[cur][i]])
            {
                q.push(g[cur][i]);
                vis[g[cur][i]] = 1;
                dis[g[cur][i]] = dis[cur] + 1;
            }
        }
    }
    return p;
}
```



# 计算几何

## 一 二维几何

### 1. 基本操作【部分未验证】

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

const double eps = 1e-11;

struct point
{
    double x, y;
    point () {}
    point (double _x, double _y) : x(_x), y(_y) {}

    point operator+ (point b)
    {
        return point(x + b.x, y + b.y);
    }
    point operator- (point b)
    {
        return point(x - b.x, y - b.y);
    }
    point operator* (double k)
    {
        return point(k*x, k*y);
    }
    point operator/ (double k)
    {
        return point(x/k, y/k);
    }

    double dot(point b)
    {
        return x * b.x + y * b.y;
    }
    double cross(point b)
```

```
    {
        return x * b.y - y * b.x;
    }

    double dist()
    {
        return sqrt(x*x + y*y);
    }
    double dist2()
    {
        return x*x+y*y;
    }

    inline int getQuad()
    {
        if (x > eps && y > -eps) return 1;
        else if (x < eps && y > eps) return 2;
        else if (x < -eps && y < eps) return 3;
        else return 4;
    }

    void rotate90()
    {
        double nx = x, ny = y;
        x = -ny, y = nx;
    }
};

struct line
{
    point a, b;
    line () {}
    line (point _a, point _b): a(_a), b(_b) {}
};
typedef line segment;

double distSegmentPoint(point, segment);
double distLinePoint(point, line);
bool judSegmentInt(segment, segment);
point getLineInt(line, line);

struct circle
{
    point centre;
    double r;

    circle (point _c, double _r = 0)
    {
        centre = _c, r = _r;
    }
    circle (point a, point b)
    {
        centre.x = (a.x + b.x) / 2;
        centre.y = (a.y + b.y) / 2;
        r = (a-b).dist() / 2;
    }

    circle (point a, point b, point c)
    {
        point u1((a.x + b.x) / 2, (a.y + b.y) / 2), v1;
        v1.x = u1.x - (b.y - a.y), v1.y = u1.y + (b.x - a.x);

        point u2((a.x + c.x) / 2, (a.y + c.y) / 2), v2;
        v2.x = u2.x - (c.y - a.y), v2.y = u2.y + (c.x - a.x);

        centre = getLineInt(line(u1, v1), line(u2, v2));
        r = (centre - a).dist();
    }

    bool inCircle(point px)
    {
        if ((px - centre).dist() < r + eps)
            return true;
        else
            return false;
    }

    bool judIntSegment(segment s)
    {
        return distSegmentPoint(centre, s) < r + eps &&
            (r < (centre - s.a).dist() + eps || r < (centre -
s.b).dist() + eps);
    }

    bool judIntCircle(circle b)
    {
        double dis = (centre - b.centre).dist();
        return dis < r + b.r + eps && fabs(r - b.r) < dis + eps;
    }

    int getIntLine(line l, vector<point> &ret)
    {
        double x = (l.a - centre).dot(l.b - l.a);
        double y = (l.b - l.a).dist2();
        double d = x*x - y*((l.a - centre).dist2() - r*r);
        if (d < -eps) return 0;
        if (d < 0) d = 0;
        point q1 = l.a - ((l.b - l.a)*(x/y));
```

```
        point q2 = l.b - (l.a*(sqrt(d)/y));
        ret.push_back(q1);
        ret.push_back(q2);
        return 2;
    }

    int getIntCircle(circle &c2, vector <point> &ret)
    {
        double x = (centre - c2.centre).dist2();
        double y = ((r*r - c2.r*c2.r) / x + 1.0) / 2.0;
        double d = r*r/x - y*y;
        if (d < -eps) return 0;

        point q1 = centre + (c2.centre - centre) * y;
        if (d < eps)
        {
            ret.push_back(q1);
            return 1;
        }
        point q2 = (c2.centre - centre) * sqrt(d);
        ret.push_back(q1 - q2);
        ret.push_back(q1 + q2);
        return 2;
    }

    int getTangentFromP(Point p, vector <point> &ret)
    {
        double x = (p - centre).dist2();
        double d = x - r*r;
        if (d < -eps) return 0;

        if (d < 0) d = 0;
    }

};

double distSegmentPoint(point p, segment s)
{
    if ((s.b - s.a).dot(p - s.a) < eps) return (p - s.a).dist();
    if ((s.a - s.b).dot(p - s.b) < eps) return (p - s.b).dist();
    return distLinePoint(p, s);
}

double distLinePoint(point p, line l)
{
    return abs((l.b - l.a).cross(p - l.a)) / (l.b - l.a).dist();
}

bool onSegment(segment a, point b)
{
    return (min(a.a.x, a.b.x) < b.x + eps) && (max(a.a.x, a.b.x) >
b.x - eps)
            && (min(a.a.y, a.b.y) < b.y + eps) && (max(a.a.y, a.b.y)
> b.y - eps);
}
bool judSegmentInt(segment a, segment b)
{
    double d1, d2, d3, d4;
    d1 = (b.a - a.a).cross(a.b - a.a);
    d2 = (b.b - a.a).cross(a.b - a.a);
    d3 = (a.a - b.a).cross(b.b - b.a);
    d4 = (a.b - b.a).cross(b.b - b.a);

    if (((d1 > eps && d2 < -eps) || (d1 < -eps && d2 > eps))
        && ((d3 > eps && d4 < -eps) || (d3 < -eps && d4 > eps)))
        return true;

    if (fabs(d1) < eps)     return onSegment(a, b.a);
    if (fabs(d2) < eps)     return onSegment(a, b.b);
    if (fabs(d3) < eps)     return onSegment(b, a.a);
    if (fabs(d4) < eps)     return onSegment(b, a.b);
    return false;
}
point getLineInt(line a, line b)
{
    point ret = a.a;

    double t = ((a.a.x - b.a.x) * (b.a.y - b.b.y) - (a.a.y - b.a.y)
* (b.a.x - b.b.x))
                /((a.a.x - a.b.x) * (b.a.y - b.b.y) - (a.a.y - a.b.y)
* (b.a.x - b.b.x));

    ret.x += (a.b.x - a.a.x) * t;
    ret.y += (a.b.y - a.a.y) * t;
    return ret;
}

point getProj(line a, point p)
{
    return a.a + ((a.b - a.a)*((a.b - a.a).dot(p - a.a) / (a.b -
a.a).dist()));
}
```

## 2. 凸包 O(N logN)

```
#include <iostream>
#include <cstdio>
#include <cmath>
#include <cstring>
#include <algorithm>
#define MAXN 10001
#define eps 1e-9
#define SQ(x) ((x)*(x))
using namespace std;

struct point
{
    double x, y;
    point (double _x = 0, double _y = 0)
    {
        x = _x, y = _y;
    }
    double operator * (point b)
    {
        return x * b.x + y * b.y;
    }

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

    point operator - (point b)
    {
        return point(x - b.x, y - b.y);
    }
};
point pts[MAXN];
point _Basis;
double direction(point a, point b, point c)
{
    return (c - a).cross(b - a);
}

bool cmp(point a, point b)
{
    if (fabs(direction(_Basis, a, b)) < eps)
        return SQ(a - _Basis) < SQ(b - _Basis) - eps;
    return direction(_Basis, a, b) < -eps;
}

void getConvex(point p[], int n, point *convex, int &hv)
{

    int Basis = 0, Top;
    for (int i = 0; i < n; i++)
    {
        if (fabs(p[i].y - p[Basis].y) < eps)
            if (p[i].x < p[Basis].x - eps)
                Basis = i;
        Basis = p[i].y < p[Basis].y-eps ? i : Basis;
    }
    _Basis = p[Basis];
    swap(p[0], p[Basis]);
    sort(p + 1, p + n, cmp);
    for (int i = 0; i < 3; i++)
        convex[i] = p[i];
    Top = 2;
    for (int i = 3; i < n; i++)
    {
        while (direction(convex[Top - 1], convex[Top], p[i]) > -eps
&& Top > 0)
            Top--;
        convex[++Top] = p[i];
    }
    hv = Top+1;
}

double getConvexArea(point conv[], int num) // 计算凸包面积
{
    double ans = 0;
    for (int i = 0; i < num; i++)
        ans += direction(point(0, 0), conv[i], conv[(i + 1) % num]);
    return fabs(ans / 2.0);     // 计算结果必须 / 2.0
}
```

## 3. 旋转卡壳

```
// 用以求凸包上距离最远两点的距离
// 如果有三点共线的情况，取消代码注释部分
// 返回 ans 最大两点距离平方
double getMaxSQDistance(/*point p[], int n,*/ point conv[], int num)
{
    /*
    if (num <= 1)
    {
        double ret = 0;
        for (int i = 0; i < n; i++)
```

```
            ret = max(ret, SQ(p[i] - p[0]));
        return ret;
    }*/
    int q = 1;
    double ans = 0;
    for (int i = 0; i < num; i++)
    {
        while (abs(direction(conv[i], conv[i + 1], conv[q]))
                < abs(direction(conv[i], conv[i + 1], conv[q + 1]))-eps)
            q = (q + 1) % num;
        ans = max(ans, max(SQ(conv[i] - conv[q]), SQ(conv[i + 1] - conv[q + 1])));
    }
    return ans;
}
```

## 4. 半平面交 O(N logN)

```
// POJ 2451 Uyuw's Concert
// POJ 3335
// POJ 1279
#include <iostream>
#include <vector>
#include <cstdio>
#include <cmath>
#include <cstring>
#include <algorithm>
#define SQ(x) ((x)*(x))
#define A first
#define B second
using namespace std;

const double eps = 1e-9;
const double pi = acos(-1.0);
const int maxn = 1010;

struct point
{
    double x, y;
    point (double _x = 0, double _y = 0)
    {
        x = _x, y = _y;
    }
    point operator * (double b)
    {
        return point(x*b, y*b);
    }

    inline double dot(point b)
    {
        return x * b.x + y * b.y;
    }

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

    point operator - (point b)
    {
        return point(x - b.x, y - b.y);
    }

    inline int getQuad()
    {
        if (x > eps && y > -eps) return 1;
        else if (x < eps && y > eps) return 2;
        else if (x < -eps && y < eps) return 3;
        else return 4;
    }
    inline double dist()
    {
        return x*x + y*y;
    }
};
typedef pair <point, point> Line;

point pts[maxn];
Line line[maxn];

double direction(point a, point b, point c)
{
    return (c - a).cross(b - a);
}

bool cmpPointbyAngle(point a, point b)
{
    int q1 = a.getQuad(), q2 = b.getQuad();
    if (q1 == q2)
    {
        if (fabs(a.cross(b)) < eps)
            return a.dist() < b.dist();

        return a.cross(b) > eps;
```

```
    }
    return q1 < q2;
}

bool cmpLinebyAngle(Line a, Line b)
{
    point v1 = a.B - a.A, v2 = b.B - b.A;

    return cmpPointbyAngle(v1, v2);
}

bool onLeft(Line L, point p)
{
    return (L.B - L.A).cross(p - L.A) > eps;  // 注意: 如果相交的点也算作半平面交的解 将此句改为 (L.B - L.A).cross(p - L.A) > -eps
}

bool isParallel(Line a, Line b) /// 平行且同向
{
    return fabs((a.B - a.A).cross(b.B - b.A)) < eps &&
(a.B-a.A).dot(b.B - b.A)>eps;
}

point lineInt(Line a, Line b)
{   // 注意 事先必须判断两直线不平行
    point ret = a.A;

    double t = ((a.A.x - b.A.x) * (b.A.y - b.B.y) - (a.A.y - b.A.y)
* (b.A.x - b.B.x))
              /((a.A.x - a.B.x) * (b.A.y - b.B.y) - (a.A.y - a.B.y)
* (b.A.x - b.B.x));

    ret.x += (a.B.x - a.A.x) * t;
    ret.y += (a.B.y - a.A.y) * t;
    return ret;
}

point intersect[maxn];
Line my_deque[maxn];

int halfplaneIntersection(Line *L, int n, vector<point> &poly)
{
    sort(L, L + n, cmpLinebyAngle);

    int first, last;

    point *p = intersect;
    Line *q = my_deque;
    q[first = last = 0] = L[0];

    /** p 为双端队列对应的交点 p[i-1] = q[i] 交 q[i-1] */

    for (int i = 1; i < n; i++)
    {
        /** 消除无用直线，加入新直线 */
        while (first < last && !onLeft(L[i], p[last-1])) last--;
        while (first < last && !onLeft(L[i], p[first])) first++;
        q[++last] = L[i];

        /** 判断平行，平行时保留内侧直线 */
        if (first < last && isParallel(q[last], q[last - 1]))
        {
            last--;
            if (onLeft(q[last], L[i].A)) q[last] = L[i];
        }
        if (first < last)
            p[last-1] = lineInt(q[last-1], q[last]);
    }
    while (first < last && !onLeft(q[first], p[last-1])) last--;
    if (last - first <= 1) return 0;
    p[last] = lineInt(q[last], q[first]);
    for (int i = first; i <= last; i++) poly.push_back(p[i]);
    return (int)poly.size();
}
```

## 5. 最小包围圆 O(N)

```
#include <iostream>
#include <cstdio>
#include <cmath>
#include <cstring>
#include <algorithm>
#define MAXN 50001
#define eps 1e-9
#define SQ(x) ((x)*(x))
using namespace std;

int n;

struct point
{
    double x, y;
    point (double _x = 0, double _y = 0)
    {
        x = _x, y = _y;
    }
```

```
    double operator * (point b)
    {
        return x * b.x + y * b.y;
    }
    double cross (point b)
    {
        return x * b.y - y * b.x;
    }

    point operator - (point b)
    {
        return point(x - b.x, y - b.y);
    }
};

point lineInt(point a1, point a2, point b1, point b2)
{   // 注意 事先必须判断两直线不平行
    point ret = a1;

    double t = ((a1.x - b1.x) * (b1.y - b2.y) - (a1.y - b1.y) *
(b1.x - b2.x))
               /((a1.x - a2.x) * (b1.y - b2.y) - (a1.y - a2.y) *
(b1.x - b2.x));

    ret.x += (a2.x - a1.x) * t;
    ret.y += (a2.y - a1.y) * t;
    return ret;
}

double dist(point a, point b)
{
    return sqrt(SQ(a - b));
}

struct circle
{
    point centre;
    double r;

    circle (point _c, double _r = 0)
    {
        centre = _c, r = _r;
    }
    circle (point a, point b)
    {
        centre.x = (a.x + b.x) / 2;
        centre.y = (a.y + b.y) / 2;
        r = dist(a, b) / 2;
    }
    circle (point a, point b, point c)
    {
        point u1((a.x + b.x) / 2, (a.y + b.y) / 2), v1;
        v1.x = u1.x - (b.y - a.y), v1.y = u1.y + (b.x - a.x);

        point u2((a.x + c.x) / 2, (a.y + c.y) / 2), v2;
        v2.x = u2.x - (c.y - a.y), v2.y = u2.y + (c.x - a.x);

        centre = lineInt(u1, v1, u2, v2);
        r = dist(centre, a);
    }

    bool inCircle(point px)
    {
        if (sqrt(SQ(px - centre)) < r + eps)
            return true;
        else
            return false;
    }
};

double direction(point a, point b, point c)
{
    return (c - a).cross(b - a);
}

point pt[MAXN];
int Basis, Top;

circle minCoverDisc(point p[], int n)
{
    random_shuffle(p, p + n);

    if (n == 1)
        return circle(p[0], 0);
    circle ret(p[0], p[1]);

    for (int i = 2; i < n; i++)
    {
        if (ret.inCircle(p[i]))
            continue;
        ret = circle(p[0], p[i]);
        for (int j = 1; j < i; j++)
        {
            if (ret.inCircle(p[j]) || i == j)
                continue;
            ret = circle(p[i], p[j]);

            for (int k = 0; k < j; k++)
```

```
            {
                if (ret.inCircle(p[k]) || k == j || k == i)
                    continue;
                ret = circle(p[i], p[j], p[k]);
            }
        }
    }
    return ret;
}

int main()
{
    int n;
    //freopen("10005.in", "r", stdin);
    while (1)
    {
        scanf("%d", &n);
        if (!n) break;
        for (int i = 0; i < n; i++)
        {
            scanf("%lf%lf", &pt[i].x, &pt[i].y);
        }

        double rr = 0;
        circle c = minCoverDisc(pt, n);
        scanf("%lf", &rr);

        if (rr < c.r - eps)
        {
            printf("There is no way of packing that polygon.\n");
        }
        else
        {
            printf("The polygon can be packed in the circle.\n");
        }
        printf("(%.3f, %.3f)   r = %.11f\n", c.centre.x, c.centre.y,
c.r);
    }
    return 0;
}
```



# 二 三维几何

## 1. 基本操作（点和线等）

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <vector>
using namespace std;
const double eps = 0;
struct point3D
{
    double x, y, z;
    point3D (double _x = 0, double _y = 0, double _z = 0)
    {
        x = _x, y = _y, z = _z;
    }
    point3D operator + (point3D b)
    {
        return point3D(x + b.x, y + b.y, z + b.z);
    }

    point3D operator - (point3D b)
    {
        return point3D(x - b.x, y - b.y, z - b.z);
    }
    point3D operator * (double k)
    {
        return point3D(k*x, k*y, k*z);
    }
    point3D operator / (double k)
    {
        return point3D(x/k, y/k, z/k);
    }

    point3D cross(point3D b)
    {
         return point3D(y * b.z - z * b.y,
                        z * b.x - x * b.z,
                        x * b.y - y * b.x);
    }
    double dot(point3D b)
    {
        return x * b.x + y * b.y + z * b.z;
```

```
    }
    double dist2()
    {
        return x * x + y * y + z * z;
    }
    double dist()
    {
        return sqrt(dist2());
    }
    void read()
    {
        scanf("%lf%lf%lf", &x, &y, &z);
    }
    void write()
    {
        printf("%.6lf %.6lf %.6lf", x, y, z);
    }
};

double disLP(point3D p1, point3D p2, point3D q)
{
    return ((p2 - p1).cross(q - p1)).dist() / (p2 - p1).dist();
}

double disLL(point3D p1, point3D p2, point3D q1, point3D q2,
                                        point3D &is1, point3D &is2)
{
    point3D p = q1 - p1, u = p2 - p1, v = q2 - q1;
    double d = u.dist2() * v.dist2() - u.dot(v) * u.dot(v);
    if (fabs(d) < eps) return disLP(q1, q2, p1);
    double s = (p.dot(u) * v.dist2() - p.dot(v) * u.dot(v)) / d;
    double t = -(u.dist2() * p.dot(v) - p.dot(u) * v.dot(u)) / d;
    is1 = p1 + (u * s);
    is2 = q1 + (v * t);
    return disLP(q1, q2, p1+(u * s));
}

vector<point3D> isPL(point3D p, point3D o, point3D q1, point3D q2)
{
    double a = o.dot(q2-p);
    double b = o.dot(q1-p);
    double d = a - b;
    vector <point3D> retv;
    if (fabs(d) < eps) return retv;
    retv.push_back((q1 * a - q2 * b)/d);
    return retv;
}

bool isPP(point3D p1, point3D o1, point3D p2, point3D o2,
pair<point3D, point3D> L)
{
    point3D e = o1.cross(o2);
    point3D v = o1.cross(e);
    double d = o2.dot(v);
    if (fabs(d) < eps) return false;
    point3D q = p1 + (v * (o2.dot(p2 - p1)) / d);
    L.first = q, L.second = q + e;
    return true;
}
```

## 2. 凸包 O(N^2)

来源于网络 验过

```
/*
模板来源：互联网
功能：
    1. 求三维凸包
    2. 三维凸包求表面积
    3. 三维凸包求面数
    4. 求质心位置

*/
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <algorithm>
using namespace std;
const int PR = 1e-8;
const int eps = 1e-8;
#define N 510
struct TPoint
{
    double x,y,z;
    TPoint(){}
    TPoint(double _x,double _y,double _z):x(_x),y(_y),z(_z){}
    TPoint operator-(const TPoint p) {return
TPoint(x-p.x,y-p.y,z-p.z);}
    TPoint operator*(const TPoint p) {return
TPoint(y*p.z-z*p.y,z*p.x-x*p.z,x*p.y-y*p.x);}//叉积
    double operator^(const TPoint p) {return x*p.x+y*p.y+z*p.z;}//点
积
};
struct fac
{
    int a,b,c;//凸包一个面上的三个点的编号
```

```
    bool ok;//该面是否是最终凸包中的面
};
struct T3dhull
{
    int n;//初始点数
    TPoint ply[N];//初始点
    int trianglecnt;//凸包上三角形数
    fac tri[N];//凸包三角形
    int vis[N][N];//点 i 到点 j 是属于哪个面
    double dist(TPoint a){return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);}//两点长度
    double area(TPoint a,TPoint b,TPoint c){return dist((b-a)*(c-a));}//三角形面积*2
    double volume(TPoint a,TPoint b,TPoint c,TPoint d){return (b-a)*(c-a)^(d-a);}//四面体有向体积*6
    double ptoplane(TPoint &p,fac &f)//正: 点在面同向
    {
        TPoint m=ply[f.b]-ply[f.a],n=ply[f.c]-ply[f.a],t=p-ply[f.a];
        return (m*n)^t;
    }
    void deal(int p,int a,int b)
    {
        int f=vis[a][b];
        fac add;
        if(tri[f].ok)
        {
            if((ptoplane(ply[p],tri[f]))>PR) dfs(p,f);
            else
            {
                add.a=b,add.b=a,add.c=p,add.ok=1;
                vis[p][b]=vis[a][p]=vis[b][a]=trianglecnt;
                tri[trianglecnt++]=add;
            }
        }
    }
    void dfs(int p,int cnt)//维护凸包，如果点 p 在凸包外更新凸包
    {
        tri[cnt].ok=0;
        deal(p,tri[cnt].b,tri[cnt].a);
        deal(p,tri[cnt].c,tri[cnt].b);
        deal(p,tri[cnt].a,tri[cnt].c);
    }
    bool same(int s,int e)//判断两个面是否为同一面
    {
        TPoint a=ply[tri[s].a],b=ply[tri[s].b],c=ply[tri[s].c];
        return fabs(volume(a,b,c,ply[tri[e].a]))<PR
            &&fabs(volume(a,b,c,ply[tri[e].b]))<PR
            &&fabs(volume(a,b,c,ply[tri[e].c]))<PR;
    }
    void construct()//构建凸包
    {
        int i,j;
        trianglecnt=0;
        if(n<4) return ;
        bool tmp=true;
        for(i=1;i<n;i++)//前两点不共点
        {
            if((dist(ply[0]-ply[i]))>PR)
            {
                swap(ply[1],ply[i]); tmp=false; break;
            }
        }
        if(tmp) return;
        tmp=true;
        for(i=2;i<n;i++)//前三点不共线
        {
            if((dist((ply[0]-ply[1])*(ply[1]-ply[i])))>PR)
            {
                swap(ply[2],ply[i]); tmp=false; break;
            }
        }
        if(tmp) return ;
        tmp=true;
        for(i=3;i<n;i++)//前四点不共面
        {

if(fabs((ply[0]-ply[1])*(ply[1]-ply[2])^(ply[0]-ply[i]))>PR)
            {
                swap(ply[3],ply[i]); tmp=false; break;
            }
        }
        if(tmp) return ;
        fac add;
        for(i=0;i<4;i++)//构建初始四面体
        {
            add.a=(i+1)%4,add.b=(i+2)%4,add.c=(i+3)%4,add.ok=1;
            if((ptoplane(ply[i],add))>0) swap(add.b,add.c);
            vis[add.a][add.b]=vis[add.b][add.c]=vis[add.c]
[add.a]=trianglecnt;
            tri[trianglecnt++]=add;
        }
        for(i=4;i<n;i++)//构建更新凸包
        {
            for(j=0;j<trianglecnt;j++)
            {
                if(tri[j].ok&&(ptoplane(ply[i],tri[j]))>PR)
                {
                    dfs(i,j); break;
```

```
                }
            }
        }
        int cnt=trianglecnt;
        trianglecnt=0;
        for(i=0;i<cnt;i++)
        {
            if(tri[i].ok)
                tri[trianglecnt++]=tri[i];
        }
    }
    double area()//表面积
    {
        double ret=0;
        for(int i=0;i<trianglecnt;i++)
            ret+=area(ply[tri[i].a],ply[tri[i].b],ply[tri[i].c]);
        return ret/2.0;
    }
    double volume()//体积
    {
        TPoint p(0,0,0);
        double ret=0;
        for(int i=0;i<trianglecnt;i++)

ret+=volume(p,ply[tri[i].a],ply[tri[i].b],ply[tri[i].c]);
        return fabs(ret/6);
    }
    int facetri() {return trianglecnt;}//表面三角形数
    int facepolygon()//表面多边形数
    {
        int ans=0,i,j,k;
        for(i=0;i<trianglecnt;i++)
        {
            for(j=0,k=1;j<i;j++)
            {
                if(same(i,j)) {k=0;break;}
            }
            ans+=k;
        }
        return ans;
    }

    TPoint massPoint()  // 凸包重心
    {
        TPoint ret(0, 0, 0);
        double sumv = 0.0;
        for (int i = 0; i < trianglecnt; i++)
        {
            TPoint cur;
            cur = ply[tri[i].a] + ply[tri[i].b] + ply[tri[i].c];
            cur.x /= 4.0, cur.y /= 4.0, cur.z /= 4.0;
            double v = volume(TPoint(0,0,0), ply[tri[i].a],
ply[tri[i].b], ply[tri[i].c]);
            cur.x *= v, cur.y *= v, cur.z *= v;
            sumv += v;
            ret = ret + cur;
        }

        ret.x /= sumv, ret.y /= sumv, ret.z /= sumv;
        return ret;
    }

}hull;
```

### 3. 凸包切割 O(N^2)

参考东大模板

# 其他算法专题

### Dancing Links X 精确覆盖

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

const int maxr = 1010, maxc = 1010, inf = 0x3f3f3f3f;

int findcnt = 0;
struct DancingLinks
{
    struct Node
    {
        int L, R, U, D, Col, Row;
        void setLink(int _l, int _r, int _u, int _d, int _col, int
_row)
        {
            L = _l, R = _r, U = _u, D = _d, Col = _col, Row = _row;
```

```
        }
    } mat[maxr*maxc];

    int head, nodecnt, S[maxc], c, r;
    int ans[maxr], ansdep;

    void init(int _r, int _c) // Set all nums in matrix 1
    {
        c = _c, r = _r;
        head = 0;
        nodecnt = c+r+1;
        memset(S, 0, sizeof S);

        for (int i = 1; i <= c; i++)
        {
            mat[i].setLink(i-1, i+1, i, i, i, 0);
        }
        mat[c].R = head;
        mat[1].L = 0;
        for (int i = 1; i <= r; i++)
        {
            mat[c+i].setLink(i+c, i+c, i-1+c, i+1+c, 0, i+c);
        }
        mat[c+1].U = 0;
        mat[c+r].D = 0;

        mat[head].setLink(c, 1, c+r, c+1, 0, 0);
    }

    void setRow(int row, const vector<int> cols) // Set 1 from the
top to the bottom, cols should be sorted.
    {
        for (int i = 0; i < (int)cols.size(); i++)
        {
            int rtail = mat[row+c].L, ctail = mat[cols[i]].U, rhead
= row+c, chead = cols[i];
            mat[nodecnt].setLink(rtail, rhead, ctail, chead, chead,
rhead);
            mat[rtail].R = nodecnt, mat[ctail].D = nodecnt;
            mat[rhead].L = nodecnt, mat[chead].U = nodecnt;
            S[cols[i]]++;
            nodecnt++;
        }
    }

    void removeColumn(int col)
    {
        mat[mat[col].L].R = mat[col].R;
        mat[mat[col].R].L = mat[col].L;

        for (int i = mat[col].D; i != col; i = mat[i].D)
        {
            for (int j = mat[i].R; j != i; j = mat[j].R)
            {
                if (mat[j].Col == 0) continue;
                mat[mat[j].U].D = mat[j].D;
                mat[mat[j].D].U = mat[j].U;
                S[mat[j].Col]--;
            }
        }
    }

    void resumeColumn(int col)
    {
        for (int i = mat[col].U; i != col; i = mat[i].U)
        {
            for (int j = mat[i].L; j != i; j = mat[j].L)
            {
                if (mat[j].Col == 0) continue;
                mat[mat[j].U].D = j;
                mat[mat[j].D].U = j;
                S[mat[j].Col]++;
            }
        }
        mat[mat[col].L].R = col;
        mat[mat[col].R].L = col;
    }

    bool dfs(int dep)
    {
        findcnt++;

        if (mat[head].R == head)
        {
            // one answer has been found
            ansdep = dep;
            return true;
        }
        int mins = inf, rc = -1;
        for (int i = mat[head].R; i != head; i = mat[i].R)
            if (mins > S[mat[i].Col]) mins = S[mat[i].Col], rc =
mat[i].Col;

        removeColumn(rc);


        for (int i = mat[rc].D; i != rc; i = mat[i].D)
        {
```

```
            for (int j = mat[i].R; j != i; j = mat[j].R)
            {
                if (mat[j].Col != 0)
                    removeColumn(mat[j].Col);
            }

            ans[dep] = mat[i].Row - c;
            if (dfs(dep+1)) return true;

            for (int j = mat[i].L; j != i; j = mat[j].L)
                if (mat[j].Col != 0)
                    resumeColumn(mat[j].Col);
        }
        resumeColumn(rc);
        return false;
    }
};

DancingLinks dl;

int main()
{
    int n, m;

    while (~scanf("%d%d", &n, &m))
    {
        dl.init(n, m);

        for (int i = 1; i <= n; i++)
        {
            int c;
            scanf("%d", &c);
            vector <int> v;
            for (int j = 0; j < c; j++)
            {
                int p;
                scanf("%d", &p);
                v.push_back(p);
            }
            sort(v.begin(), v.end());
            dl.setRow(i, v);
        }

        if (dl.dfs(0))
        {
            printf("%d ", dl.ansdep);
            for (int i = 0; i < dl.ansdep; i++)
            {
                if (i) printf(" ");
                printf("%d", dl.ans[i]);
            }
            printf("\n");
        }
        else
            printf("NO\n");
    }

    return 0;
}
```



# 语言和环境

## 一 python 可视化小工具

```
"""
  This file is a part of TkDraw

  Copyright (C) 2011  Rujia Liu

  This program is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this program; if not, write to the Free
Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA
02110-1301
  USA
"""

import sys
from Tkinter import *

def onMouseDown(event):
  statusText.set("%d,%d" % (event.x, event.y))

```

```
def onRightMouseDown(event):
  global scale, dx, dy
  scale = 1
  dx = dy = 0
  f = open(sys.argv[1], 'r')
  solo = False
  inblock = True
  draw("CLEAR")
  for line in f.readlines():
    line = line.strip()
    if line == '-s':
      if not solo:
        solo = True # start to draw a solo block
        draw("CLEAR")
      else:
        break # end of solo
    elif line == '-[':
      if inblock:
        draw("CLEAR") # clear previously drawn content
      inblock = True
    elif line == '-]':
      inblock = False
    else:
      if inblock:
        draw(line) # draw if we're inside a block
  f.close()

def buildGUI():
  global canvas, statusText

  canvas = Canvas(width=800, height=600, background='#ffffff')
  canvas.bind("<Button-1>", onMouseDown)
  canvas.bind("<Button-3>", onRightMouseDown)
  canvas.pack()

  statusText = StringVar()
  status = Label(root, textvariable=statusText, bd=1, relief=SUNKEN,
anchor=W)
  status.pack(side=BOTTOM, fill=X)

def draw(command):
  global scale, dx, dy
  cmd = command.split()
  if cmd[0] == "TRANSLATE":
    dx = float(cmd[1])
    dy = float(cmd[2])
  if cmd[0] == "SCALE":
    scale = float(cmd[1])
  if cmd[0] == 'LINE':
    x1 = (float(cmd[1]) + dx) * scale
    y1 = (float(cmd[2]) + dy) * scale
    x2 = (float(cmd[3]) + dx) * scale
    y2 = (float(cmd[4]) + dy) * scale
    canvas.create_line(x1, y1, x2, y2)
  if cmd[0] == 'CIRCLE':
    x = (float(cmd[1]) + dx) * scale
    y = (float(cmd[2]) + dy) * scale
    r = float(cmd[3]) * scale
    canvas.create_oval(x-r, y-r, x+r, y+r)
  if cmd[0] == 'CLEAR':
    all = canvas.find_all()
    for item in all:
      canvas.delete(item)

root = Tk()
buildGUI()
root.mainloop()
```

# 二 准备工作

## 配终端

gnome-terminal [--disable-factory] -t $TITLE -x