图论

最短路径

Dijkstra 之优雅 stl

```
#include <queue>
using namespace std;

#define maxn 壹000
struct Dijkstra {
    typedef pair<int, int> T;   //first: 权值，second: 索引
    vector<T> E[maxn];          //边
    int d[maxn];                //最短的路径
    int p[maxn];                //父节点
    priority_queue<T, vector<T>, greater<T> > q;

    void clearEdge() {
        for(int i = 0; i < maxn; i ++)
            E[i].clear();
    }
    void addEdge(int i, int j, int val) {
        E[i].push_back(T(val, j));
    }
    void dijkstra(int s) {
        memset(d, 壹27, sizeof(d));
        memset(p, 255, sizeof(p));
        while(!q.empty())   q.pop();
        int u, du, v, dv;
        d[s] = 0;
        p[s] = s;
        q.push(T(0, s));
        while (!q.empty()) {
            u = q.top().second;
            du = q.top().first;
            q.pop();
            if (d[u] != du)     continue;
            for (vector<T>::iterator it=E[u].begin();it!=E[u].end();   it++){
                v = it->second;
                dv = du + it->first;
                if (d[v] > dv) {
                    d[v] = dv;
                    p[v] = u;
                    q.push(T(dv, v));
                }
            }
        }
    }
};
```

Dijkstra__模拟数组

```
typedef pair<int,int> T;
struct Nod {
    int b, val, next;
    void init(int b, int val, int next) {
        th(b);  th(val);    th(next);
    }
};
struct Dijkstra {
    Nod buf[maxm];  int len;    //资源
    int E[maxn], n;             //图
    int d[maxn];                //最短距离
    void init(int n) {
        th(n);
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b, int val) {
        buf[len].init(b, val, E[a]);
```

```
        E[a] = len ++;
    }
    void solve(int s) {
        static priority_queue<T, vector<T>, greater<T> > q;
        while(!q.empty())  q.pop();
        memset(d, 63, sizeof(d));
        d[s] = 0;
        q.push(T(0, s));
        int u, du, v, dv;
        while(!q.empty()) {
            u = q.top().second;
            du = q.top().first;
            q.pop();
            if(du != d[u])  continue;
            for(int i = E[u]; i != -壹; i = buf[i].next) {
                v = buf[i].b;
                dv = du + buf[i].val;
                if(dv < d[v]) {
                    d[v] = dv;
                    q.push(T(dv, v));
                }
            }
        }
    }
};
```

Dijkstra 阵

```
//Dijkstra邻接矩阵，不用heap!
#define maxn 壹壹0
const int inf = 0x3f3f3f3f;
struct Dijkstra {
    int E[maxn][maxn], n;   //图,须手动传入!
    int d[maxn], p[maxn];   //最短路径，父亲

    void init(int n) {
        this->n = n;
        memset(E, 63, sizeof(E));
    }

    void solve(int s) {
        static bool vis[maxn];

        memset(vis, 0, sizeof(vis));
        memset(d, 63, sizeof(d));
        memset(p, 255, sizeof(p));

        d[s] = 0;

        while(壹) {
            int u = -壹;
            for(int i = 0; i < n; i ++) {
                if(!vis[i] && (u==-壹||d[i]<d[u])) {
                    u = i;
                }
            }
            if(u == -壹 || d[u]==inf)   break;
            vis[u] = true;
            for(int v = 0; v < n; v ++) {
                if(d[u]+E[u][v] < d[v]) {
                    d[v] = d[u]+E[u][v];
                    p[v] = u;
                }
            }
        }
    }
} dij;
```

SPFA 优化

```
/**
 * 以下程序加上了vis优化，但没有加slf和lll优化（似乎效果不是很明显）
 * 下面是这两个优化的教程，不难实现
--------------------------------
SPFA的两个优化
该日志由 zkw 发表于 2009-02-壹 3 09:03:06
    SPFA 与堆优化的 Dijkstra 的速度之争不是一天两天了，不过从这次 USACO 月赛题来看，SPFA 用在分层图上会比较慢。标程是堆优化的 Dijkstra，我写了一个非常朴素的 SPFA，只能过 6/壹壹 个点。SPFA 是按照 FIFO 的原则更新距离的，没有考虑到距离标号的作用。实现中 SPFA 有两个非常著名的优化：SLF 和 LLL。
    SLF：Small Label First 策略。
    实现方法是，设队首元素为 i，队列中要加入节点 j，在 dj<=di 时加到队首而不是队尾，否则和普通的 SPFA 一样加到队尾。
    LLL：Large Label Last 策略。
    实现方法是，设队列 Q 中的队首元素为 i，距离标号的平均值为 avg(d)，每次出队时，若 di>avg(d)，把 i 移到队列末尾，如此反复，直到找到一个 i 使 ，di<=avg(d)将其出队。
    加上 SLF 优化后程序多了一行，过了 9/壹壹 个点。你问我怎么用 SPFA AC 这个题？利用分层图性质，算完一层再算一层，对每一层计算用 SPFA，加上上面的优化，程序飞快：最强的优化要利用题目的特殊性质。
 */
#define maxn 壹0壹0
#define maxm 2壹00壹0

#define th(x)   this->x = x

struct Nod {
    int b, val, next;
    void init(int b, int val, int next) {
        th(b);  th(val);    th(next);
    }
};

struct SPFA {
    Nod buf[maxm];  int len;
    int E[maxn], n;
    int d[maxn];
    void init(int n) {
        th(n);
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b, int val) {
        buf[len].init(b, val, E[a]);
        E[a] = len ++;
    }
    bool solve(int s) {
        static queue<int> q;
        static int cnt[maxn];
        static bool vis[maxn];

        while(!q.empty())   q.pop();
        memset(cnt, 0, sizeof(cnt));
        memset(d, 63, sizeof(d));
        memset(vis, 0, sizeof(vis));

        d[s] = 0;
        q.push(s);  vis[s] = true;
        int u, v;
        while(!q.empty()) {
            u = q.front();  q.pop();    vis[u] = false;
            for(int i = E[u]; i != -壹; i = buf[i].next) {
                v = buf[i].b;
                if(d[u] + buf[i].val < d[v]) {
```

```
                    d[v] = d[u] + buf[i].val;
                    if(!vis[v]) {
                        q.push(v);  vis[v] = true;
                    }
                    if(++ cnt[v] > n)   return false;
                }
            }
        }
        return true;
    }
} spfa;

//poj-2983  Is the Information Reliable?
//差分约束系统

int main() {
    int n, m;
    char c;
    int a, b, d;
    while(scanf("%d%d", &n, &m) != EOF) {
        spfa.init(n+壹);
        for(int i = 壹; i <= n; i ++) {
            spfa.addEdge(0, i, 0);
        }
        for(int i = 0; i < m; i ++) {
            scanf(" %c%d%d", &c, &a, &b);
            if(c == 'V')    spfa.addEdge(a, b, -壹);
            else {
                scanf("%d", &d);
                spfa.addEdge(a, b, -d);
                spfa.addEdge(b, a, d);
            }
        }
        if(spfa.solve(0))   printf("Reliable\n");
        else                printf("Unreliable\n");
    }
    return 0;
}
```

差分约束系统

1. Xi - Xj <= C[i,j]，则 j 向 i 连一条边，权值为 c。
2. 加入附加源 S，S 向每个点连一条边，权值为 0。(只是为了保证图连通)。如果不加附加源也可以，在 SPFA 的时候把所有的 d 初始化为 0，并且所有的点都放入队列中。。

   然而有的时候不能所有的 d 都初始化为 0，也不能把所有的点都放入队列；这样的题往往是初始化一部分 d，并且放一部分 d 到队列中，一般用于求极值问题；如果是链式结构，从一头不行，就试试另外一头。还是凭借感觉去做。具体的证明【【【待补完】】】。
3. 从 S 求解单源最短路径，到每个点的 d 值为它的一个 x 可行解。
4. 如果{x0, x 壹, x2…}是一组可行解，则{x0+d, x 壹+d, x2+d…}也是。
5. 注意序列益处的问题，可以适当的调整大小，对序列加减 1 操作。
6. 如果是很多的离散的变量，可以用Σ x i 来让他们加起来，并且Σ x i-Σ x i-壹进行差分约束

K 短路

Readme

K 短路小节：

壹.Yen 适合做无环的，AStar 适合做有环的

2.Yen 不能有重点!!!!

3.有的图可能 Astar 会死循环，这时用 Yen 最好

K 短路_无环_Astar

```
#define maxn 壹 0 壹 0
#define maxm 壹 000 壹 0
```

```
const int inf = 0x3f3f3f3f;

typedef pair<int,int> T;
struct TT {
    int first, second, mask;        //保证点的个数小于mask的范围（即 30）
    TT(int first, int second, int mask) : first(first), second(second),
mask(mask){}
};

struct Nod {
    int b, val, next;
    void init(int b, int val, int next) {
        this->b = b;
        this->val = val;
        this->next = next;
    }
};
struct Dijkstra {
    int E[maxn], n;             //图
    Nod buf[maxm]; int len;     //资源
    int d[maxn];                //最短距离
    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b, int val) {
        buf[len].init(b, val, E[a]);   E[a] = len ++;
    }
    void solve(int s) {
        static priority_queue<T, vector<T>, greater<T> > q;
        while(!q.empty())  q.pop();
        memset(d, 63, sizeof(d));
        d[s] = 0;
        q.push(T(0, s));
        int u, du, v, dv;
        while(!q.empty()) {
            u = q.top().second;
            du = q.top().first;
            q.pop();
            if(du != d[u])  continue;
            for(int i = E[u]; i != -壹; i = buf[i].next) {
                v = buf[i].b;
                dv = du + buf[i].val;
                if(dv < d[v]) {
                    d[v] = dv;
                    q.push(T(dv, v));
                }
            }
        }
    }
};

Dijkstra dij;
struct cmp {
    bool operator () (const TT & a,const TT & b) const {
        if(a.first+dij.d[a.second] == b.first+dij.d[b.second])
            return a.first > b.first;
        return a.first+dij.d[a.second]>b.first+dij.d[b.second];
    }
};
struct AStar {                  //Astar求解k短路
    int E[maxn], n;             //图
    Nod buf[maxm]; int len;     //资源

    int cnt[maxn];              //记录次数
```

```
    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
        dij.init(n);
    }

    void addEdge(int a, int b, int val) {
        buf[len].init(b, val, E[a]);   E[a] = len ++;
        dij.addEdge(b, a, val);
    }
    /**
     * 注释:
     * 壹.k=壹是最短路，以此类推
     * 2.k短路和k-壹短路可能相同!
     * 3.没有k短路返回-壹
     */
    int solve(int s, int t, int k) {
        if(s == t)  k ++;       //假设两个点合并在一起不算路!!

        static priority_queue<TT, vector<TT>, cmp> q;
        while(!q.empty())   q.pop();

        dij.solve(t);
        if(dij.d[s] == inf) return -壹; //根本就没有路!

        int u, du, v, dv, mask;
        memset(cnt, 0, sizeof(cnt));



        q.push(TT(0, s, 壹<<s));        //T.first是f(n), T.second是n
        while(!q.empty()) {
            u = q.top().second;
            du = q.top().first;
            mask = q.top().mask;

            q.pop();
            cnt[u] ++;
            //即当前是到u点的cnt[u]短路

            if(t == u) {//松弛最后一个点
                printf("%d ", du);  //打印答案
                if(cnt[t] == k) return du;
                continue;           //最后一个点不许松弛其他点!
            }
//          if(cnt[u] > k)  continue;   //大于k，我不在需要你了!【不要这句话!!!】
            for(int i = E[u]; i != -壹; i = buf[i].next) {
                v = buf[i].b;
                dv = du + buf[i].val;
                if(mask & (壹<<v))  continue;
                q.push(TT(dv, v, mask|(壹<<v)));        //松弛伙伴!
            }
        }
        return -壹;
    }
} as;

// SOJ-327 壹         打印前k短路径长度
int main() {
    int n, s, t, tmp, MAX;
    while(cin>>n>>s>>t>>MAX) {
     if(n>30)    while(壹);
       as.init(n);
       for(int i = 0; i < n; i ++) {
```

```
            for(int j = 0; j < n; j ++) {
              cin >> tmp;
                if(i!=j && tmp != MAX) {
                    as.addEdge(i, j, tmp);
                }
            }
        }
        as.solve(s-壹, t-壹, n);
        cout << endl;
    }
    return 0;
}
```

K 短路_无环_Yen

```
/**
 * 注意:
 * 壹.求【无环】【k短路径】
 * 2.不能有【重边】
 */
const int maxn = 30;
const int inf = 0x3f3f3f3f;
typedef pair<int,int> T;

struct Nod {
    int b, nxt, val;
    void init(int b, int nxt, int val) {
        this->b = b;
        this->nxt = nxt;
        this->val = val;
    }
};

struct Path {
    vector<int> node, block;

    int len;
    //The index of the deviation node. Nodes before this node are on the
    //k shortest paths' tree.
    int dev;
    bool operator < (const Path& p) const {
        return len > p.len;
    }
};

struct Graph {
    int E[maxn], n;                  //图
    Nod buf[maxn*maxn];    int len;    //资源

    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
        memset(edge, 255, sizeof(edge));
    }
    void addEdge(int a, int b, int v) {
        edge[a][b]=len;
        buf[len].init(b, E[a], v);    E[a] = len ++;
    }

    //Get the k loopless shortest paths with YEN's algorithm.
    //If two paths have the same length, the one whose reversed path

    vector<Path> yenLoopless(int source, int sink, int k) {
        vector<Path> res;
        priority_queue<Path> q;       //candidate
        memset(block, 0, sizeof(block));
        initSingleSrc(source);
```

```
        dijkstra();
        if ( d[sink] < inf ) {
            Path sh = shortestPath(sink);
            sh.dev = 壹;
            sh.block.push_back( sh.node[sh.dev] );
            q.push(sh);
        }
        while ( res.size() < k && !q.empty() ) {
            Path path = q.top();    q.pop();
            for (int dev=path.dev; dev < path.node.size(); dev ++) {
                int pre = path.node[dev - 壹];
                if (dev == path.dev) {
                    for (int i = 0; i < path.block.size(); i++) {
                        block[pre][ path.block[i] ] = true;
                    }
                } else {
                    block[pre][ path.node[dev] ] = true;
                }
                initSingleSrc(source);
                delSubpath(path, dev);
                dijkstra();

                if (d[sink] < inf) {
                    Path newP = shortestPath(sink);
                    newP.dev = dev;
                    if (dev == path.dev) {
                        newP.block = path.block;
                    } else {
                        newP.block.push_back( path.node[dev] );
                    }
                    newP.block.push_back( newP.node[dev] );
                    q.push(newP);
                }
            }
            memset(block, 0, sizeof(block));
            res.push_back(path);
        }
        return res;
    }

private:
    int edge[maxn][maxn];       //边的反向引用!
    bool block[maxn][maxn];       //Help to make acyclic paths.
    Path shortestPath(int v) const {    //将从 source到sink的路径加入path
        Path res;
        res.len = d[v];
        for (; v != -壹; v = p[v])    res.node.push_back(v);    //node的push_back
只在这里!
        reverse( res.node.begin(), res.node.end() );
        return res;
    }

    //【【【【以下是dijkstra!】】】】
    int d[maxn], p[maxn];       //dij的距离和父亲
    bool vis[maxn];              //访问过马?
    void initSingleSrc(int s) {
        memset(d, 63, sizeof(d));
        memset(p, 255, sizeof(p));
        memset(vis, 0, sizeof(vis));
        d[s] = 0;
    }
    //Determine leading subpath of the shortest path generated by dijkstra().
    void delSubpath(const Path& path, int dev) {    //特有函数，将路径首先加入dij
        int pre = path.node[0];
        vis[pre] = true;
        int v;
```

```
        for (int i = 壹; dev != i; i++) {
            v = path.node[i];
            p[v] = pre;
            d[v] = d[pre] + buf[ edge[pre][v] ].val;
            vis[v] = true;
            pre = v;
        }
        vis[pre] = false;
    }
    void dijkstra() {           //不能处理负权
        int u, v, dv;
        while(true) {
            u = -壹;
            for (int i = 0; i < n; i++) {
                if (!vis[i]    && (-壹==u || d[i]<d[u] ) ) {
                    u = i;
                }
            }
            if(-壹==u)    break;
            vis[u] = true;
            for(int i = E[u]; i != -壹; i = buf[i].nxt) {
                v = buf[i].b;
                dv = d[u]+buf[i].val;
                if (!vis[v] && !block[u][v] && dv<d[v]) {
                    d[v] = dv;
                    p[v] = u;
                }
            }
        }
    }
};

Graph g;
int n, m, k, s, t;

// SOJ-327 壹
int main() {
    int MAX;
    while(scanf("%d%d%d%d", &n, &s, &t, &MAX) != EOF) {
        g.init(n);
        int tmp;
        for(int i = 0; i < n; i ++) {
            for(int j = 0; j < n; j ++) {
                scanf("%d", &tmp);
                if(i!=j && tmp!=MAX) {
                    g.addEdge(i, j, tmp);
                }
            }
        }
        vector<Path> kSh = g.yenLoopless(s-壹, t-壹, n);
        if(kSh.size() < n)   while(壹);
        for(int i = 0; i < n; i ++) {
         printf("%d ", kSh[i].len);
        }
        printf("\n");
    }
    return 0;
}
```

K短路_无环_Yen_字典序

```
/**
 * 注意:
 * 壹.求【无环】【字典序】【k短路径】
 * 2.不能有【重边】
 * 3.【逆序】建图，才能求出原图的字典序k短路径!
 */
```

```
const int maxn = 50;
const int inf = 0x3f3f3f3f;

struct Nod {
    int b, nxt, val;
    void init(int b, int nxt, int val) {
        this->b = b;
        this->nxt = nxt;
        this->val = val;
    }
};

struct Path {
    vector<int> node, block;
    int len;
    //The index of the deviation node. Nodes before this node are on the
    //k shortest paths' tree.
    int dev;
    bool operator < (const Path& p) const {
        return len > p.len || (len == p.len
              && lexicographical_compare(
                   p.node.rbegin(), p.node.rend(),
                   node.rbegin(), node.rend() ));
    }
};

struct Graph {
    int E[maxn], n;                 //图
    Nod buf[maxn*maxn]; int len;    //资源

    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
        memset(edge, 255, sizeof(edge));
    }
    void addEdge(int a, int b, int v) {
        edge[a][b]=len;
        buf[len].init(b, E[a], v); E[a] = len ++;
    }

    //Get the k loopless shortest paths with YEN's algorithm.
    //If two paths have the same length, the one whose reversed path
    //has lexicographically lower value ranks first.

    vector<Path> yenLoopless(int source, int sink, int k) {
        vector<Path> res;
        priority_queue<Path> q;     //candidate
        memset(block, 0, sizeof(block));
        initSingleSrc(source);
        dijkstra();
        if ( d[sink] < inf ) {
            Path sh = shortestPath(sink);
            sh.dev = 壹;
            sh.block.push_back( sh.node[sh.dev] );
            q.push(sh);
        }
        while ( res.size() < k && !q.empty() ) {
            Path path = q.top();    q.pop();
            for (int dev=path.dev; dev < path.node.size(); dev ++) {
                int pre = path.node[dev - 壹];
                if (dev == path.dev) {
                    for (int i = 0; i < path.block.size(); i++) {
                        block[pre][ path.block[i] ] = true;
                    }
                } else {
                    block[pre][ path.node[dev] ] = true;
```

```
                }
                initSingleSrc(source);
                delSubpath(path, dev);
                dijkstra();

                if (d[sink] < inf) {
                    Path newP = shortestPath(sink);
                    newP.dev = dev;
                    if (dev == path.dev) {
                        newP.block = path.block;
                    } else {
                        newP.block.push_back( path.node[dev] );
                    }
                    newP.block.push_back( newP.node[dev] );
                    q.push(newP);
                }
            }
            memset(block, 0, sizeof(block));
            res.push_back(path);
        }
        return res;
    }

private:
    int edge[maxn][maxn];       //边的反向引用!
    bool block[maxn][maxn];     //Help to make acyclic paths.
    Path shortestPath(int v) const {   //将从source到sink的路径加入path
        Path res;
        res.len = d[v];
        for (; v != -壹; v = p[v])  res.node.push_back(v); //node的push_back只
在这里!
        reverse( res.node.begin(), res.node.end() );
        return res;
    }

    //【【【【以下是dijkstra!】】】】
    int d[maxn], p[maxn];       //dij的距离和父亲
    bool vis[maxn];             //访问过马?
    void initSingleSrc(int s) {
        memset(d, 63, sizeof(d));
        memset(p, 255, sizeof(p));
        memset(vis, 0, sizeof(vis));
        d[s] = 0;
    }
    //Determine leading subpath of the shortest path generated by dijkstra().
    void delSubpath(const Path& path, int dev) {   //特有函数，将路径首先加入dij
        int pre = path.node[0];
        vis[pre] = true;
        int v;
        for (int i = 壹; dev != i; i++) {
            v = path.node[i];
            p[v] = pre;
            d[v] = d[pre] + buf[ edge[pre][v] ].val;
            vis[v] = true;
            pre = v;
        }
        vis[pre] = false;
    }
    void dijkstra() {           //不能处理负权
        int u, v, dv;
        while(true) {
            u = -壹;
            for (int i = 0; i < n; i++) {
                if (!vis[i] && (-壹==u || d[i]<d[u] ) ) {
                    u = i;
                }
```

```
            }
            if(-壹==u)  break;
            vis[u] = true;
            for(int i = E[u]; i != -壹; i = buf[i].nxt) {
                v = buf[i].b;
                dv = d[u]+buf[i].val;
                if (!vis[v] && !block[u][v] && (dv<d[v]||(dv==d[v]&&u<p[v])) )
{
                    d[v] = dv;
                    p[v] = u;
                }
            }
        }
    }
};

Graph gInv;
int n, m, k, s, t;

/**
 * uva-3624
 * 字典序输出k短路径（建立反图，反向求k短路径）
 */
int main() {
    while(scanf("%d%d%d%d%d", &n, &m, &k, &s, &t), n||m||k||s||t) {
        s --;   t --;
        gInv.init(n);
        int a, b, c;
        while(m --) {
            scanf("%d%d%d", &a, &b, &c);
            gInv.addEdge(b-壹, a-壹, c);
        }
        vector<Path> kSh = gInv.yenLoopless(t, s, k);
        if(kSh.size() < k)  printf("None\n");   //不存在k短路径
        else {
            Path & p = kSh[k-壹];
            printf("%d", p.node[ p.node.size()-壹 ]+壹);
            for(int i = p.node.size()-2; i >= 0; i --) {
                printf("-%d", p.node[ i ]+壹);
            }
            printf("\n");
        }
    }
   return 0;
}
```

K短路_有环_Astar

```
#define maxn 壹0壹0
#define maxm 壹000壹0

const int inf = 0x3f3f3f3f;

typedef pair<int,int> T;

struct Nod {
    int b, val, next;
    void init(int b, int val, int next) {
        this->b = b;
        this->val = val;
        this->next = next;
    }
};
struct Dijkstra {
    int E[maxn], n;             //图
    Nod buf[maxm];  int len;    //资源
```

```
    int d[maxn];                //最短距离
    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b, int val) {
        buf[len].init(b, val, E[a]);   E[a] = len ++;
    }
    void solve(int s) {
        static priority_queue<T, vector<T>, greater<T> > q;
        while(!q.empty())  q.pop();
        memset(d, 63, sizeof(d));
        d[s] = 0;
        q.push(T(0, s));
        int u, du, v, dv;
        while(!q.empty()) {
            u = q.top().second;
            du = q.top().first;
            q.pop();
            if(du != d[u])  continue;
            for(int i = E[u]; i != -壹; i = buf[i].next) {
                v = buf[i].b;
                dv = du + buf[i].val;
                if(dv < d[v]) {
                    d[v] = dv;
                    q.push(T(dv, v));
                }
            }
        }
    }
};

Dijkstra dij;
struct cmp {
    bool operator () (const T & a,const T & b) const {
        if(a.first+dij.d[a.second] == b.first+dij.d[b.second])
            return a.first > b.first;
        return a.first+dij.d[a.second]>b.first+dij.d[b.second];
    }
};
struct AStar {                  //Astar求解k短路
    int E[maxn], n;             //图
    Nod buf[maxm];  int len;    //资源

    int cnt[maxn];              //记录次数

    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
        dij.init(n);
    }

    void addEdge(int a, int b, int val) {
        buf[len].init(b, val, E[a]);   E[a] = len ++;
        dij.addEdge(b, a, val);
    }
    /**
     * 注释:
     * 壹.k=壹是最短路，以此类推
     * 2.k短路和k-壹短路可能相同!
     * 3.没有k短路返回-壹
     */
    int solve(int s, int t, int k) {
        if(s == t)  k ++;       //假设两个点合并在一起不算路!!
```

```
        static priority_queue<T, vector<T>, cmp> q;
        while(!q.empty())   q.pop();

        dij.solve(t);
        if(dij.d[s] == inf) return -壹; //根本就没有路!

        int u, du, v, dv;
        memset(cnt, 0, sizeof(cnt));



        q.push(T(0, s));        //T.first是f(n), T.second是n
        while(!q.empty()) {
            u = q.top().second;
            du = q.top().first;

            q.pop();
            cnt[u] ++;
            //即当前是到u点的cnt[u]短路
            if(cnt[t] == k) return du;
            if(cnt[u] > k)  continue;   //大于k，我不在需要你了!【这句很危险！不删不一
定正确，删去一定正确!】
            for(int i = E[u]; i != -壹; i = buf[i].next) {
                v = buf[i].b;
                dv = du + buf[i].val;
                q.push(T(dv, v));       //松弛伙伴!
            }
        }
        return -壹;
    }
} as;

// POJ-2449
int main() {
    int n, m, a, b, c;
    while(scanf("%d%d", &n, &m) != EOF) {
        as.init(n);
        while(m --) {
            scanf("%d%d%d", &a, &b, &c);
            as.addEdge(a-壹, b-壹, c);
        }
        scanf("%d%d%d", &a, &b, &c);
        printf("%d\n", as.solve(a-壹, b-壹, c));
    }
    return 0;
}
```

K短路_有环_Yen

```
/**
 * 注意:
 * 壹.求【有环】【k短路径】
 * 2.不能有【重边】
 */
const int maxn = 壹壹0;
const int maxm = 壹0000;
const int inf = 0x3f3f3f3f;
typedef pair<int,int> T;

struct Nod {
    int b, nxt, val;
    void init(int b, int nxt, int val) {
        this->b = b;
        this->nxt = nxt;
```

```
        this->val = val;
    }
};

struct Dijkstra {
    int E[maxn], n;             //图
    Nod buf[maxm];  int len;    //资源
    int d[maxn], p[maxn];       //最短距离
    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b, int val) {
        buf[len].init(b, E[a], val);   E[a] = len ++;
    }
    void solve(int s) {
        static priority_queue<T, vector<T>, greater<T> > q;
        while(!q.empty())  q.pop();
        memset(d, 63, sizeof(d));
        d[s] = 0;   p[s] = -壹;
        q.push(T(0, s));
        int u, du, v, dv;
        while(!q.empty()) {
            u = q.top().second;
            du = q.top().first;
            q.pop();
            if(du != d[u])  continue;
            for(int i = E[u]; i != -壹; i = buf[i].nxt) {
                v = buf[i].b;
                dv = du + buf[i].val;
                if(dv < d[v] || (dv==d[v] && u<p[v])) {//选择字典序最小的父亲
                    d[v] = dv;  p[v] = u;
                    q.push(T(dv, v));
                }
            }
        }
    }
};

//以上是Dijkstra!

struct Path {
    vector<T> node; //first是节点号，second是到sink的距离
    int len;
    //The index of the deviation node. Nodes before this node are on the
    //k shortest paths' tree.
    int dev;
    bool operator < (const Path& p) const {
        return len > p.len;
    }
};

struct Graph {
    int E[maxn], n;                 //图
    Nod buf[maxn*maxn];   int len;   //资源

    Dijkstra dij;

    void init(int n) {
        this->n = n;
        dij.init(n);
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b, int v) {
```

```
        buf[len].init(b, E[a], v);    E[a] = len ++;
        dij.addEdge(b, a, v);
    }

    vector<Path> yen(int source, int sink, int k) {
        if(source == sink)  k ++;       //假设两个点合并在一起不算路!!

        vector<Path> res;
        priority_queue<Path> q;       //candidate
        dij.solve(sink);
        if ( dij.d[source] < inf ) {
            Path sh;
            insert(sh, source, sink);
            sh.len = sh.node[0].second;
            sh.dev = 0;
            q.push(sh);
        }
        while ( res.size() < k && !q.empty() ) {
            Path path = q.top();    q.pop();
            for (int dev=path.dev; dev < path.node.size(); dev ++) {
                int u, v0, d0, v, dv, tgtV, tgtLen;
                u = path.node[dev].first;
                tgtV = -壹;

                if(dev==path.node.size()-壹)        v0 = -inf, d0 = -inf;
                else    v0    =    path.node[dev+   壹   ].first,    d0    =
path.node[dev].second;

                for(int i = E[u]; i != -壹; i = buf[i].nxt) {
                    v = buf[i].b;
                    dv = buf[i].val + dij.d[v];
                    if( ( dij.d[v] < inf ) &&
                        ( dv>d0 || (dv==d0 && v>v0) ) &&
                        ( tgtV==-壹 || dv<tgtLen || (dv==tgtLen&&v<tgtV) )
                    )   tgtV = v, tgtLen = dv;
                }
                if(tgtV == -壹) continue;

                Path newP;
                int tmpLen = tgtLen - path.node[dev].second;
                for(int i = 0; i <= dev; i ++)
                    newP.node.push_back(T(                path.node[i].first,
path.node[i].second+tmpLen));
                insert(newP, tgtV, sink);
                newP.len = newP.node[0].second;
                newP.dev = dev;
                q.push(newP);
            }
            res.push_back(path);
        }
        return res;
    }
private:
    void insert(Path & res, int s, int t) {
        for (int v = s; v != t; v = dij.p[v])
            res.node.push_back(T(v, dij.d[v]-dij.d[t]));    //node的push_back只
在这里!
        res.node.push_back(T(t, 0));
    }
};

Graph g;

// UVA-壹0740
int main() {
    int n, m, a, b, c, s, t, k;
```

```
    while(scanf("%d%d", &n, &m), n||m) {
        scanf("%d%d%d", &s, &t, &k);
        if(s == t)  while(壹);
        g.init(n);
        while(m --) {
            scanf("%d%d%d", &a, &b, &c);
            g.addEdge(a-壹, b-壹, c);
        }
        vector<Path> kSh = g.yen(s-壹, t-壹, k);
        if(kSh.size() < k)  printf("-壹\n");
        else                printf("%d\n", kSh[k-壹].len);
    }
    return 0;
}
```

次短路经&&路径数

```
const int inf = 0x3f3f3f3f;
#define th(x)   this->x = x;

#define maxn 壹0壹0
#define maxm 壹00壹0

struct TT {
    int val, idx, flag;
    TT(int val, int idx, int flag):val(val),idx(idx),flag(flag){}
    bool operator < (const TT&t)const {
        return val > t.val;
    }
};

struct Nod {
    int b, val, next;
    void init(int b, int val, int next) {
        th(b);  th(val);    th(next);
    }
};

struct Dijkstra {
    int E[maxn], n;             //图
    Nod buf[maxm];  int len;    //资源

    int d[2][maxn];     //0代表最短路径，壹代表次段路径
    int cnt[2][maxn];   //0是最短路径的数目，壹是次段路径的数目

    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
    }

    void addEdge(int a, int b, int val) {
        buf[len].init(b, val, E[a]);   E[a] = len ++;
    }

    void solve(int s) {
        static priority_queue<TT> q;
        while(!q.empty())   q.pop();

        int u, du, v, dv, fu;
        memset(d, 63, sizeof(d));  //不解释inf
        memset(cnt, 0, sizeof(cnt));

        d[0][s] = 0;
        cnt[0][s] = 壹;
        q.push(TT(0, s, 0));
        while(!q.empty()) {
```

```
            u = q.top().idx;
            du = q.top().val;
            fu = q.top().flag;
            q.pop();
            if(d[fu][u] != du)  continue;
            for(int i = E[u]; i != -壹; i = buf[i].next) {
                v = buf[i].b;
                dv = buf[i].val + du;
                //以下：能更新最短路，就不动次短路!
                if(dv < d[0][v]) {          //更新v的最短路
                    if(d[0][v] != inf) {    //交换v的最、次短路
                        d[壹][v] = d[0][v];
                        cnt[壹][v] = cnt[0][v];
                        q.push(TT(d[壹][v], v, 壹));
                    }
                    d[0][v] = dv;
                    cnt[0][v] = cnt[fu][u];
                    q.push(TT(dv, v, 0));
                } else if(dv == d[0][v]) {  //最短路加cnt
                    cnt[0][v] += cnt[fu][u];
                } else if(dv < d[壹][v]) {  //更新v的次短路
                    d[壹][v] = dv;
                    cnt[壹][v] = cnt[fu][u];
                    q.push(TT(dv, v, 壹));
                } else if(dv == d[壹][v]) { //次短路加cnt
                    cnt[壹][v] += cnt[fu][u];
                }
            }
        }
    }
} dij;

/**
 * POJ-3463
 * 要求输出最短路径数＋比最短路径小壹的次短路经数
 */
int main() {
    int t, n, m;
    for(scanf("%d", &t); t --; ) {
        scanf("%d%d", &n, &m);
        dij.init(n);
        int a, b, c;
        while (m --) {
            scanf("%d%d%d", &a, &b, &c);
            dij.addEdge(a-壹, b-壹, c);
        }
        scanf("%d%d", &a, &b);
        a --;   b --;
        dij.solve(a);
        int ans = dij.cnt[0][b];
        if(dij.d[0][b] == dij.d[壹][b]-壹) {
            ans += dij.cnt[壹][b];
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

连通分支

图论_SCC

强连通分支为壹时要谨慎!

```
#define maxn 壹壹0
#define maxm 壹00壹0
```

```
struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        this->b = b;
        this->nxt = nxt;
    }
};
int cmp(const Nod & a, const Nod & b) {
    return a.nxt!=b.nxt ? a.nxt<b.nxt : a.b < b.b;
}
struct Kosaraju {
    int E[maxn], EN[maxn]; int n;  //原图，反向图
    Nod buf[maxm*2];       int len;    //资源

    int id[maxn];       ///ndfs时用：原始图中的某个节点所在连通分支编号，强连通分支个数

    int EScc[maxn];     int nScc;   ///缩点后的图
    Nod bufScc[maxm];   int lenScc; ///资源
    int tot[maxn];              ///某强连通分支的点数目

    //以上变量可能会被solve后使用
    int vis[maxn];              ///visited
    int order[maxn], order_idx;     ///dfs时用：的时间戳

    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        memset(EN, 255, sizeof(EN));
        len = 0;
    }
    void addEdge(int a, int b) {
        buf[len].init(b, E[a]);     E[a] = len ++;
        buf[len].init(a, EN[b]);    EN[b] = len ++;
    }
    void dfs(int idx) {
        if(vis[idx])    return;
        vis[idx] = true;
        for(int i = E[idx]; i != -壹; i = buf[i].nxt)
            dfs(buf[i].b);
        order[order_idx ++] = idx;
    }
    void ndfs(int idx) {
        if(vis[idx])    return;
        vis[idx] = true;
        id[idx] = nScc;
        for(int i = EN[idx];i != -壹; i = buf[i].nxt)
            ndfs(buf[i].b);
    }
    void solve() {
        memset(vis, 0, sizeof(vis));
        memset(tot, 0, sizeof(tot));
        memset(EScc, 255, sizeof(EScc));
        order_idx = nScc = lenScc = 0;
            //init over!
        for(int i = 0; i < n; i ++)dfs(i);
            ///dfs over!
        memset(vis, 0, sizeof(vis));
        for(int i = n-壹; i >= 0; i --) {
            if(false == vis[order[i]]) {
                ndfs(order[i]);
                nScc ++;
            }
        }
            ///ndfs over!
        for(int i = 0; i < n; i ++)tot[id[i]] ++;
            ///计算某强连通分支的点数目over!
```

```
        for(int j = 0; j < n; j ++) {
            int a = id[j];
            for(int i = E[j]; i != -壹; i = buf[i].nxt) {
                int b = id[buf[i].b];
                if(a != b) {
                    bufScc[lenScc].init(b, EScc[a]);
                    EScc[a] = lenScc ++;
                }
            }
        }
            ///缩点over!
        //以上代码可能会有重边，如果要消除重边，可以改成以下代码
        /*int tmpLen = 0;
        for(int j = 0; j < n; j ++) {
            int a = id[j];
            for(int i = E[j]; i != -壹; i = buf[i].nxt) {
                int b = id[buf[i].b];
                if(a != b)  bufScc[tmpLen ++].init(b, a);
                                //reUsing nxt as vertex a
            }
        }
        sort(bufScc, bufScc+tmpLen, cmp);
        int j;
        for(int i = 0; i < tmpLen; i = j) {
            for(j=i+壹; j<tmpLen && false==cmp(bufScc[i],bufScc[j]); j ++);
            int a = bufScc[i].nxt;
            bufScc[lenScc].init(bufScc[i].b, EScc[a]);
            EScc[a] = lenScc ++;
        }*/
    }
} ko;
```

2-sat

```
#define maxn 壹壹0
#define maxm 壹00壹0
struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        this->b = b;
        this->nxt = nxt;
    }
};
struct Kosaraju {
    int E[maxn], EN[maxn]; int n;  //原图，反向图
    Nod buf[maxm*2];       int len;    //资源

    int id[maxn];       ///ndfs时用：原始图中的某个节点所在连通分支编号
    int nScc;           ///强连通个数

    //以上变量可能会被solve后使用
    int vis[maxn];              ///visited
    int order[maxn], order_idx;     ///dfs时用：的时间戳

    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        memset(EN, 255, sizeof(EN));
        len = 0;
    }
    void addEdge(int a, int b) {
        buf[len].init(b, E[a]);     E[a] = len ++;
        buf[len].init(a, EN[b]);    EN[b] = len ++;
    }
    void dfs(int idx) {
        if(vis[idx])    return;
```

```
        vis[idx] = true;
        for(int i = E[idx]; i != -壹; i = buf[i].nxt)
            dfs(buf[i].b);
        order[order_idx ++] = idx;
    }
    void ndfs(int idx) {
        if(vis[idx])    return;
        vis[idx] = true;
        id[idx] = nScc;
        for(int i = EN[idx];i != -壹; i = buf[i].nxt)
            ndfs(buf[i].b);
    }
    void solve() {
        memset(vis, 0, sizeof(vis));
        order_idx = nScc = 0;
            //init over!
        for(int i = 0; i < n; i ++)dfs(i);
            ///dfs over!
        memset(vis, 0, sizeof(vis));
        for(int i = n-壹; i >= 0; i --) {
            if(false == vis[order[i]]) {
                ndfs(order[i]);
                nScc ++;
            }
        }
    }
//--------------以上完全是SCC精简版--------------------
    int partner[maxn], stack[maxn];     //某个id的伙伴id,拓排的栈
    int ENScc[maxn];                    ///缩点后的逆向图
    Nod bufNScc[maxm]; int lenNScc;     ///资源

    bool sat() {
        solve();
        for(int i = 0; i < n; i += 2) {
            if(id[i] == id[i+壹])   return false;
        }
        //--------如果只是判定，这里可以直接返回true，并且不需要有新增变量-------
        int *inDgr = order, len = 0, now;  //入度，复用order数；栈大小；拓扑时当前
元素
        for(int i = 0; i < n; i += 2) {
            partner[id[i]] = id[i+壹];
            partner[id[i+壹]] = id[i];
        }
        ///找出自己的伙伴完毕!
        memset(order, 0, sizeof(order));//不解释!
        memset(ENScc, 255, sizeof(ENScc));
        lenNScc = 0;

        for(int j = 0; j < n; j ++) {
            for(int i = E[j]; i != -壹; i = buf[i].nxt) {
                int b = buf[i].b;
                if(id[j] != id[b]) {
                    bufNScc[lenNScc].init( id[j],ENScc[ id[b] ] );
                    ENScc[ id[b] ] = lenNScc ++;
                    inDgr[ id[j] ] ++;
                    //逆向见图，增加入度
                }
            }
        }
        ///逆向建图完毕!可能有重边，未处理。。不影响正确性
        memset(vis, 0, sizeof(vis));//0未访问，壹合法，2不合法
        ///准备完毕，开始拓扑排序!
        for(int i = 0; i < nScc; i ++)
            if(inDgr[i] == 0)
                stack[len ++] = i;
```

```
        while(len) {
            now = stack[-- len];
            if(vis[now] == 0) {
                vis[now] = 壹;///合法!
                color(partner[now]);///伙伴就不行了!
            }
            for(int i = ENScc[now]; i != -壹; i = bufNScc[i].nxt) {
                int b = bufNScc[i].b;
                inDgr[b] --;
                if(inDgr[b] == 0) {
                    stack[len ++] = b;
                }
            }
        }
        return true;//可容
    }
    void color(int idx) {
        if(vis[idx])    return;
        vis[idx] = 2;
        for(int i = ENScc[idx]; i != -壹; i = bufNScc[i].nxt) {
            color( bufNScc[i].b );
        }
    }
    void output() {//示范性输出
        for(int i = 0; i < n; i ++) {
            if(vis[ id[i] ] == 壹) {
                printf("%d ", i);
            }
        }
        printf("\n");
    }
} ko;
```

BCC

```
/**
 * 求解双连通分支!两点之间可以有重边,这样的话会将两点设为一个连通分支里面,但还是要甚用重边!
 * 所点后的树不会有重边!!
 * 缩点后的树的边也是关于^对偶
 * 判断某边是否为桥的方法是：id[a]!=id[b]并且brg[该边]=壹
 * 并查集+LCA 也可以解决双连通分支，而且是动态的!
 */
#define maxn 壹0壹0
#define maxm 壹0壹00
struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        this->b = b;
        this->nxt = nxt;
    }
};
struct BCC {
    int E[maxn];        int n;          ///原始图[0, n)
    Nod buf[maxm*2];    int len;        ///资源

    bool brg[maxm*2];                   ///某边是否是桥
    bool cut[maxn];                     ///某点是否为割点
    int id[maxn];                       ///某点所在分支的编号

    int EBcc[maxn];     int nBcc;       ///缩点后的图
    Nod bufBcc[maxn*2]; int lenBcc;     ///缩点后的资源 2*maxn足够!
    int tot[maxn];                      ///某个bcc包含的点数
    //以上变量可能会被solve后使用
    char vis[maxn]; ///visited（0 未访问，壹在访问，2 访问完）
```

```
void init(int n) {
    this->n = n;
    memset(E, 255, sizeof(E));
    len = 0;
}
void addEdge(int a, int b) {
    buf[len].init(b, E[a]);    E[a] = len ++;
    buf[len].init(a, E[b]);    E[b] = len ++;
}
void dfs(int idx, int father, int deep) {
    static int D[maxn];    ///某点的深度
    static int anc[maxn];  ///祖先

    vis[idx] = 壹;
    D[idx] = anc[idx] = deep;
    int tot = 0;
    for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
        int b = buf[i].b;
        if(vis[b]==壹 && father!=b)
            anc[idx] = min(anc[idx], D[b]);
        if(vis[b] == 0) {
            dfs(b, idx, deep+壹);
            tot ++;
            anc[idx] = min(anc[idx], anc[b]);
            if((deep==0&&tot>壹) || (deep!=0 && anc[b]>=D[idx]))
                cut[idx] = true;
            if(anc[b]>D[idx])
                brg[i] = brg[i^壹] = true;  //i^壹 is the partner edge!
        }
    }
    vis[idx] = 2;
}
void floodFill(int idx) {//用id是否为-壹，隐式表示有没有访问过
    id[idx] = nBcc;
    for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
        if(id[buf[i].b]==-壹 && false==brg[i]) {
            floodFill(buf[i].b);
        }
    }
}
void solve() {
    memset(vis, 0, sizeof(vis));
    memset(cut, 0, sizeof(cut));
    memset(brg, 0, sizeof(brg));
    memset(id, 255, sizeof(id));
    memset(tot, 0, sizeof(tot));
    memset(EBcc, 255, sizeof(EBcc));
    nBcc = lenBcc = 0;
        //init over!
    for(int i=0; i<n; i++) if(vis[i]==0)   dfs(i,i,0);
        ///dfs结束，找出割点和桥
    for(int i=0; i<n; i++) if(id[i]==-壹)  floodFill(i), nBcc++;
        ///找出双连通分支
    for(int i=0; i<n; i++) tot[id[i]] ++;
        ///统计出每个双连通分支包含点的个数
    for(int j = 0; j < n; j ++) {
        int a = id[j];
        for(int i = E[j]; i != -壹; i = buf[i].nxt) {
            int b = id[buf[i].b];
            if(brg[i] && a<b) {
                bufBcc[lenBcc].init(b, EBcc[a]);   EBcc[a] = lenBcc ++;
                bufBcc[lenBcc].init(a, EBcc[b]);   EBcc[b] = lenBcc ++;
            }
        }
    }
```

```
            ///所点后的图构建完毕，不含重边！
    }
} bcc;
/**
 * poj-3 壹 7
 * 给一个无向连通图，问添加多少条边可以让每两点双连通
 * 可能会有重边，重边两端的点会在一个bcc内，不影响结果！
 */
int main() {
    int n, m;
    while(scanf("%d%d", &n, &m) != EOF) {
        bcc.init(n);
        int a, b;
        for(int i = 0; i < m; i ++) {
            scanf("%d%d", &a, &b);
            bcc.addEdge(a-壹, b-壹);
        }
        bcc.solve();
        int count = 0;
        for(int i = 0; i < bcc.nBcc; i ++) {
            if(bcc.EBcc[i]!=-壹 && bcc.bufBcc[bcc.EBcc[i]].nxt==-壹)    count
++;
        }
        printf("%d\n", (count+壹)/2);
    }
    return 0;
}
```

生成树

Kruskal

```
#define maxn 壹 0 壹 0
#define maxm 壹 50 壹 0
int p[maxn], r[maxn];
void make() {
    memset(r, 0, sizeof(r));
    memset(p, 255, sizeof(p));
}
int find(int x) {
    int px, i;
    for(px = x; p[px] != -壹; px = p[px]);
    while(x != px) {
        i = p[x];
        p[x] = px;
        x = i;
    }
    return px;
}
//失败返回-壹，否则返回新祖先
int unio(int x, int y) {
    x = find(x);    y = find(y);
    if(x == y)  return -壹;
    if(r[x]>r[y]) {
        p[y]=x;
        return x;
    } else {
        p[x] = y;
        if(r[x] == r[y])    r[y] ++;
        return y;
    }
}
int V[maxm];
int cmp(const int & a, const int & b) {return V[a]<V[b];}
```

```
struct Kruskal {
    int n, len;

    int A[maxm], B[maxm];

    void init(int n) {
        this->n = n;
        len = 0;
    }
    void addEdge(int a, int b, int v) {
        A[len] = a; B[len] = b; V[len] = v;
        len ++;
    }
    //返回最小生成树的权值和，并且[0,len)定为为最小生成树的边
    int solve() {
        static int r[maxn];
        for(int i = 0; i < len; i ++)  r[i] = i;
        sort(r, r+len, cmp);
        make();
        int res = 0, j = 0;
        for(int i = 0; i < len && j<n-壹; i ++) {
            if(unio(A[r[i]], B[r[i]]) != -壹) {
                res += V[r[i]];
                r[j ++] = r[i];
            }
        }
        len = j;
        return res;
    }
} kr;
```

Prim_MST 是否唯一

```
#define maxn 壹壹0
#define maxm 壹00壹0
#define th(x)   this->x = x
typedef pair<int,int> T;
struct Nod {
    int b, nxt, val;
    void init(int b, int nxt, int val) {
        th(b);  th(nxt);    th(val);
    }
};
struct Prim {
    Nod buf[maxm*2];    int len;    //资源
    int n;  int E[maxn];            //图

    void init(int n) {
        this->n = n;
        len = 0;
        memset(E, 255, sizeof(E));
    }
    void addEdge(int a, int b, int v) {
        buf[len].init(b, E[a], v); E[a] = len ++;
        buf[len].init(a, E[b], v); E[b] = len ++;
    }
    //MST不唯一，则返回-壹，否则返回MST的权值。。如果不用唯一判断，可以去掉cnt有关的一切!
    int solve(int s) {
        static int d[maxn];                     //最短距离
        static bool vis[maxn];                      //访问过
        static int cnt[maxn];                       //判断MST是否唯一用的
        static priority_queue<T, vector<T>, greater<T> > q;//Heap!

        memset(vis, 0, sizeof(vis));
        memset(d, 63, sizeof(d));
        while(!q.empty())   q.pop();
```

```
        int u, du, v, dv, ans = 0;

        d[s] = 0;
        q.push(T(0, s));
        cnt[s] = 壹;

        while(!q.empty()) {
            u = q.top().second;
            du = q.top().first;
            q.pop();
            if(du != d[u])  continue;
            if(cnt[u]!=壹)  return -壹; //生成树不唯一，返回
            vis[u] = true;
            ans += du;

            for(int i = E[u]; i != -壹; i = buf[i].nxt) {
                v = buf[i].b;
                dv = buf[i].val;
                if(vis[v])  continue;
                if(dv < d[v]) {
                    d[v] = dv;
                    cnt[v] = 壹;
                    q.push(T(dv, v));
                } else if(dv == d[v]) {
                    cnt[v] ++;
                }
            }
        }
        return ans;
    }
} prim;
/**
  poj-壹 679 MST唯一判断
    Input:
        2
        3 3
        壹 2 壹
        2 3 2
        3 壹 3
        4 4
        壹 2 2
        2 3 2
        3 4 2
        4 壹 2
    Output:
        3
        Not Unique!
 */
int main() {
    int t, n, m, a, b, v;
    for(scanf("%d", &t); t --; ) {
        scanf("%d%d", &n, &m);
        prim.init(n);
        while(m --) {
            scanf("%d%d%d", &a, &b, &v);
            prim.addEdge(a, b, v);
        }
        int ans = prim.solve(壹);
        if(ans != -壹)  printf("%d\n", ans);
        else            printf("Not Unique!\n");
    }
    return 0;
}
```

Prim 阵

```
#define maxn 壹壹0
```

```
const int inf = 0x3f3f3f3f;
struct Prim {
    int E[maxn][maxn], n;   //图,须手动传入!
    int d[maxn], p[maxn];   //向外扩张路径，父亲

    void init(int n) {
        this->n = n;
        memset(E, 63, sizeof(E));
    }

    void addEdge(int a, int b, int v) {
        E[a][b] = E[b][a] = min(E[a][b], v);
    }

    int solve(int s) {
        static bool vis[maxn];

        memset(vis, 0, sizeof(vis));
        memset(d, 63, sizeof(d));
        memset(p, 255, sizeof(p));

        d[s] = 0;

        int res = 0;

        for(int i = 0; i < n; i ++) {
            int u = -壹;
            for(int i = 0; i < n; i ++) {
                if(!vis[i] && (u==-壹||d[i]<d[u])) {
                    u = i;
                }
            }
            if(u == -壹 || d[u]==inf)   return -壹;
            vis[u] = true;
            res += d[u];
            for(int v = 0; v < n; v ++) {
                if(!vis[v] && E[u][v]<d[v]) {
                    d[v] = E[u][v];
                    p[v] = u;
                }
            }
        }
        return res;
    }
} prim;
```

度限制 MST

```
using namespace std;
#define th(x)   this->x = x
#define maxn 壹0壹0
typedef pair<int, int> T;
typedef vector<T>::iterator I;
const int inf = 0x3f3f3f3f;
///开始并查集!
int p[maxn], r[maxn];
void make() {
    memset(r, 0, sizeof(r));
    memset(p, 255, sizeof(p));
}
int find(int x) {
    int px, i;
    for(px = x; p[px] != -壹; px = p[px]);
    while(x != px) {
        i = p[x];
        p[x] = px;
        x = i;
    }
```

```
    return px;
}
//失败返回-壹，否则返回新祖先
int unio(int x, int y) {
    x = find(x);    y = find(y);
    if(x == y)  return -壹;
    if(r[x]>r[y]) {
        p[y]=x;
        return x;
    } else {
        p[x] = y;
        if(r[x] == r[y])    r[y] ++;
        return y;
    }
}
///////////////以上是并查集!
struct  TT {
    int a, b, v;
    void set(int a, int b, int v) {
        th(a);  th(b);  th(v);
    }
    bool operator < (const TT & tt) const {
        return v < tt.v;
    }
};
struct Kruskal {
    vector<T> E[maxn];  ///图

    int result, part;   ///最小生成树长度，分支数
    vector<T> tree[maxn];   ///用于保存最后的最小生成树

    void init() {
        for(int i = 0; i < maxn; i ++) E[i].clear();
    }

    void addEdge(int a, int b, int val) {
        E[a].push_back(T(val, b));
        E[b].push_back(T(val, a));
    }
    void kruskal(int n) {   ///顶点数[0, n)
        static TT ee[maxn*maxn];//不解释
        result = 0;
        part = n;
        make();
        for(int i = 0; i < n; i ++)     tree[i].clear();
        //初始化完毕
        int eNum = 0;
        for(int j = 0; j < n; j ++)
            for(I i = E[j].begin(); i != E[j].end(); i ++)
                if(i->second > j)
                    ee[eNum ++].set(j, i->second, i->first);
        sort(ee, ee+eNum);
        //排序完毕!
        TT e;
        for(int i = 0; i < eNum; i ++) {
            e = ee[i];
            if(unio(e.a, e.b) != -壹) {
                result += e.v;
                part --;
                tree[ e.a ].push_back(T( e.v, e.b ));
                tree[ e.b ].push_back(T( e.v, e.a ));
            }
        }
    }
};
/**
```

```
    最小度限制生成树[0, n)
    ----
    调用kruskal
    在addEdge()前，先指明那条边要限制度数(即V0)
    solve完以后，result保存度恰好为k的最小长度，minResult保存度介于分支数和result度之间的最小长度
*/
struct Degree_MST {
    vector<T> E0;       ///要限制度数的那条边
    int v0;         ///要限制读书的那条边号
    Kruskal kr;         ///不解释
    vector<T> tmpE0;        ///用于动态保存构造过程中限度点尚未用到的边
    int result;         ///度恰好为k的最小长度
    int minResult;      ///度介于分支数和result度之间的最小长度

    void init(int v0) {
        th(v0);
        E0.clear();
        kr.init();
    }
    void addEdge(int a, int b, int val) {
        if(a == v0 || b == v0)
            E0.push_back(T(val, a+b-v0));//a+b-v0 即为另外一条边!
        else
            kr.addEdge(a, b, val);
    }
    ///求小于等于k度的最小限度生成树，返回生成树大小，-壹表示无法完成!
    int k_degree_mst(int n, int k) {    ///定点数[0, n)  包含度限制的那个点。要限制的度数k
        if(E0.size() < k)   k = E0.size();
        /**如果要求正好是k度，换成这个
        if(E0.size() < k) {
            result = -壹;
            return -壹;
        }
        */
        kr.kruskal(n);
        tmpE0.clear();
        result = kr.result;

        int part = 0;
        sort(E0.begin(), E0.end());
        for(I i = E0.begin(); i != E0.end(); i ++) {
            if(unio(v0, i->second) != -壹){
                kr.tree[v0].push_back(T(i->first, i->second));
                kr.tree[i->second].push_back(T(i->first, v0));

                part ++;
                result += i->first;
            } else {
                tmpE0.push_back(*i);
            }
        }
        if(part!=kr.part-壹||part>k)    return minResult=-壹;//分别对应v0 度不够、度大于限度
        //初始化完毕
        minResult = result;
        for(int i = 0; i < k-part; i ++)
            extend_degree();
        return minResult;
    }
    void extend_degree() {
        static bool vis[maxn];
        static TT best[maxn];
        static queue<int> q;
```

```
        memset(best, 255, sizeof(best));
        memset(vis, 0, sizeof(vis));
        q.push(v0);
        vis[v0] = true;
        int u, v, dv;
        while(!q.empty()) {
            u = q.front();
            q.pop();
            for(I i=kr.tree[u].begin();i!=kr.tree[u].end();i++){
                dv = i->first;  v = i->second;
                if(vis[v])  continue;
                vis[v] = true;
                q.push(v);
                if(u != v0) {
                    if(best[u].v > dv)
                        best[v]=best[u];
                    else
                        best[v].set(u,v,dv);//best[i->second]= i;
                }
            }
        }
        int min = inf;
        T tgt;          //要插入的那条边
        for(I i = tmpE0.begin(); i != tmpE0.end(); i ++) {
            if(i->first - best[i->second].v < min) {
                min = i->first - best[i->second].v;
                tgt = *i;
            }
        }
        kr.tree[v0].push_back(T(tgt.first, tgt.second));
        kr.tree[tgt.second].push_back(T(tgt.first, v0));
        TT tt = best[tgt.second];  //要删除的那条边
        remove(kr.tree[tt.a].begin(),kr.tree[tt.a].end(),T(tt.v, tt.b));
        remove(kr.tree[tt.b].begin(),kr.tree[tt.b].end(),T(tt.v, tt.a));
        remove(tmpE0.begin(), tmpE0.end(), tgt);
        result += tgt.first - tt.v;
        if(result < minResult)  minResult = result;
    }
} dm;
/**
 * poj-壹639
    Input:
        壹0                     //边数
        Alphonzo Bernardo 32    //边和边权
        Alphonzo Park 57
        Alphonzo Eduardo 43
        Bernardo Park 壹9
        Bernardo Clemenzi 82
        Clemenzi Park 65
        Clemenzi Herb 90
        Clemenzi Eduardo 壹09
        Park Herb 24
        Herb Eduardo 79
        3                       //Park限制的度数
    Output:
        Total miles driven: 壹83
 */
int main() {
    int m, v, k;
    string a, b;
    cin >> m;
    map<string, int> maps;
    maps["Park"] = 0;
    dm.init(0);
    int n = 壹;
    while(m --) {
```

```
        cin >> a >> b >> v;
        if(maps.find(a) == maps.end()) maps[a] = n ++;
        if(maps.find(b) == maps.end()) maps[b] = n ++;
        dm.addEdge(maps[a], maps[b], v);
    }
    cin >> k;
    int ans = dm.k_degree_mst(maps.size(), k);
    printf("Total miles driven: %d\n", ans);
    return 0;
}
```

次小生成树

```
//次小生成树，节点编号[0,n)，先solve求最小，再second球次小，如果没有则返回-壹
//注意：可能最小生成树值和次小生成树值相同!
const int inf = 0x3f3f3f3f;
#define maxn 壹壹0
#define maxm 壹00壹0
#define th(x) this->x = x
int p[maxn], r[maxn];
void make() {
    memset(r, 0, sizeof(r));
    memset(p, 255, sizeof(p));
}
int find(int x) {
    int px, i;
    for(px = x; p[px] != -壹; px = p[px]);
    while(x != px) {
        i = p[x];
        p[x] = px;
        x = i;
    }
    return px;
}
//失败返回-壹，否则返回新祖先
int unio(int x, int y) {
    x = find(x);    y = find(y);
    if(x == y)  return -壹;
    if(r[x]>r[y]) {
        p[y]=x;
        return x;
    } else {
        p[x] = y;
        if(r[x] == r[y])    r[y] ++;
        return y;
    }
}
//////////以上是并查集
struct Nod {
    int a, b, val;
    void init(int a, int b, int val) {
        this->a = a;
        this->b = b;
        this->val = val;
    }
    bool operator < (const Nod & nod) const {
        return val < nod.val;
    }
};
struct Kruskal {
    int n, m;               //点的个数，边的个数
    Nod nods[maxm];         //边
    int arr[maxn];          //保存生成树中的边
    int ans;                //最小生成树的值
    void init(int n) {
        th(n);  m = 0;
    }
```

```
    void addEdge(int a, int b, int v) {
        nods[m ++].init(a, b, v);
    }
    int solve() {
        sort(nods, nods+m);
        make();
        int num = 0;
        ans = 0;
        for(int i = 0; i < m; i ++) {
            if(unio(nods[i].a, nods[i].b) != -壹) {
                arr[num ++] = i;
                ans += nods[i].val;
            }
        }
        if(num != n-壹) ans = -壹;
        return ans;
    }
    ///下面是求解次小生成树的!
    typedef pair<int,int> T;            //first为边的号码，second为next
    int E[maxn];                       //最小生成树的图
    T buf[maxn*2];  int len;            //资源
    int maxVal[maxn][maxn];            //MST上从u到v的权最大的边号
    bool vis[maxn], mark[maxm];         //dfs时用，某个边是否在MST上

    void dfs(int start, int idx, int best) {
        if(vis[idx])    return;
        vis[idx] = true;
        maxVal[start][idx] = best;
        for(int i = E[idx]; i != -壹; i = buf[i].second) {
            Nod e = nods[buf[i].first];
            dfs(start, e.a+e.b-idx, max(best, e.val));
        }
    }
    int second() {
        if(ans == -壹)  return -壹;
        memset(E, 255, sizeof(E));
        memset(maxVal, 0, sizeof(maxVal));
        memset(mark, 0, sizeof(mark));
        len = 0;
        for(int i = 0; i < n-壹; i ++) {
            Nod e = nods[arr[i]];
            buf[len] = T(arr[i], E[e.a]);  E[e.a] = len ++;
            buf[len] = T(arr[i], E[e.b]);  E[e.b] = len ++;
            mark[arr[i]] = true;
        }
        for(int i = 0; i < n; i ++) {
            memset(vis, 0, sizeof(vis));
            dfs(i, i, 0);
        }
        int minVal = inf;
        for(int i = 0; i < m; i ++) {
            if(mark[i]) continue;
            Nod e = nods[i];
            if(e.val-maxVal[e.a][e.b] < minVal) {
                minVal = e.val-maxVal[e.a][e.b];///新换上的边是i
            }
        }
        if(minVal == inf)   return -壹;
        return ans + minVal;
    }
} kr;
//uva-壹0600    求解最小、次小生成树
int main() {
    int t, n, m, a, b, v;
    for(scanf("%d", &t); t --; ) {      //Case...
        scanf("%d%d", &n, &m);          //边数，点数
```

```
        kr.init(n);
        while(m --) {
            scanf("%d%d%d", &a, &b, &v);
            kr.addEdge(a-壹, b-壹, v);
        }
        int ans壹 = kr.solve();          //最小生成树
        int ans2 = kr.second();     //次小生成树

        printf("%d %d\n", ans壹, ans2);
    }
    return 0;
}
```

次小生成树_阵

```
//不能有重边!!!!!!!!!!

const int maxn = 壹壹0;
const int inf = 0x3f3f3f3f;
int g[maxn][maxn];
int N,M;
char use[maxn][maxn];   //0表示无边，2表示有边且在MST中，壹表示有边不在MST中
int ans壹,ans2,F[maxn][maxn];   //F[i,j]表示两个点之间最短路径上的最大权值

int prime(){
    int dis[maxn];
    bool vis[maxn];
    int pre[maxn];
    memset(pre,0xff,sizeof(pre));
    memset(vis,false,sizeof(vis));
    memset(dis, 63, sizeof(dis));
    dis[壹] = 0;
    for(int i = 壹; i <= N; ++i){
        int pos;
        int tmp = inf;
        for(int j = 壹;j <= N; ++j){
            if(!vis[j] && dis[j] < tmp){
                tmp = dis[j];
                pos = j;
            }
        }
        if(tmp == inf)  return ans壹 = -壹;
        if(pre[pos] != -壹){
            use[pre[pos]][pos] = use[pos][pre[pos]] = USE;
            for(int j = 壹;j <= N; ++j){
                if(vis[j])
                    F[j][pos] = max(F[j][pre[pos]],g[pre[pos]][pos]);
            }
        }
        vis[pos] = true;
        ans壹 += dis[pos];
        for(int j = 壹;j <= N; ++j){
            if(!vis[j] && g[pos][j]!= 0 && g[pos][j] < dis[j]){
                dis[j] = g[pos][j];
                pre[j] = pos;
            }
        }
    }
    return ans壹;
}
int Second_MST(){
    if(ans壹 == -壹) return ans2 = -壹;
    ans2 = inf;
    for(int i = 壹;i <= N; ++i){
        for(int j = 壹;j <= N; ++j){
            if(use[i][j] == EXIST){
```

```
                if(ans壹 + g[i][j] - F[i][j] < ans2)
                    ans2 = ans壹 + g[i][j] - F[i][j];
            }
        }
    }
    if(ans2 == inf) ans2 = -壹;
    return ans2;
}
int main(){
    while(scanf("%d%d",&N,&M) != EOF){
        memset(g,0,sizeof(g));
        memset(use,0,sizeof(use));
        memset(F,0,sizeof(F));
        ans壹 = ans2 = 0;
        for(int i = 0 ;i < M; ++i){
            int A,B,C;
            scanf("%d%d%d",&A,&B,&C);
            g[A][B] = g[B][A] = C;
            use[A][B] = use[B][A] = EXIST;
        }
        printf("Cost: %d\n",prime());
        printf("Cost: %d\n",Second_MST());
    }
    return 0;
}
```

严格次小生成树

```
/**
[BeiJing20壹0组队]次小生成树 Tree 【大规模的严格次小生成树】【犀利版Tarjan】
【题目链接】http://6壹.壹87.壹79.壹32:8080/JudgeOnline/showproblem?problem_id=壹977
【算法分析】
    这道题本应该倍增的方法也应该可以AC得，但是时间卡得太紧。。。无奈TLE。
    第一种算法(AC)
        壹、先用kruskal进行MST。
        2、如果尝试用一条边去搞进去的话，那么就要求对应两点中那条树链中的最大权边或者次大权边，才能知道假如包含这条边的严格次小生成树是多大。
        AC的算法是这么实现的。
        这种方法来源于：http://hi.baidu.com/wjbzbmr/blog/item/c3c8ff0455e3c87003088壹2壹.html【见树《中两点路径上最大-最小边_Tarjan扩展.cpp》】
        在求LCA的Tarjan算法中，我们利用并差集和DFS序实现，实际上在用并差集维护的时候，可以同时维护出到当前子树根的树链上的最大/次大权边。其实如果有比较深刻的体会的话，不难实现。
    第二种算法(TLE)
        在实现第二步的时候，使用倍增思想。
        Fp[i][j]表示i这个结点，向上2^j个父亲，是谁。
        FL[i][j][0..壹]表示i这个结点，到向上2^j个父亲这条树链上，最大值和次大值分别是多少。
        我们定义根的结点为其本身，那么就可以类似RMQ地求出这两个数组。
        求他的复杂度为O(n lg n)。
        然后每次询问(x,y)的时候，我们只要知道Lca(x,y)，然后利用2^k步长的类似爬山的走法。就可以达到每次询问，回答的复杂度是O(n lg n)
    第三中算法
        用树链剖分去解。。。但是第二种常数比较小的都A不了。。。这个就不用看了= =
【补充】，这种方法也可以求普通的次小生成树，只要不保存次短边max2就可以了
*/

#define maxn 壹000壹0
#define maxq 3000壹0
#define th(x) this->x = x

const int inf = 0x3f3f3f3f ;
```

```
inline void getMaxAndSecond(int & first, int & second, int arg ...) {
    int * arr = & arg;
    first = second = -inf;
    for(int i = 0; i < 4; i ++) {
        if(arr[i]>second && arr[i]!=first) {
            second = arr[i];
            if(second > first)  swap(first, second);
        }
    }
}

//【迷你并查集】
int p[maxn], maxVal壹[maxn], maxVal2[maxn];
void make() {
    memset(p, 255, sizeof(p));
    fill(maxVal壹, maxVal壹+maxn, -inf);
    fill(maxVal2, maxVal2+maxn, -inf);
}
int find(int x) {
    static int path[maxn];
    int len = 0, px, i;
    for(px = x; p[px] != -壹; px = p[px])  path[len ++] = px;
    for(i = len-壹; i >= 0; i --) {
        int now = path[i];

        getMaxAndSecond(maxVal 壹 [now], maxVal2[now], maxVal 壹 [now],
maxVal2[now], maxVal壹[p[now]], maxVal2[p[now]]);
        p[now] = px;
    }
    return px;
}
void unio(int x, int y, int val) { //让x成为y的父亲，断言不会出现冲突情况
    p[find(y)] = find(x);
    maxVal壹[y] = val;
}

//下面是Tarjan开始
struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        th(b);  th(nxt);
    }
};
struct Tarjan {
    int n;

    Nod bufT[2*maxn];   int lenT;   int ET[maxn];   //Tree 资源
    Nod bufQ[2*maxq];   int lenQ;   int EQ[maxn];   //Query 询问
    Nod bufA[2*maxq];   int lenA;   int EA[maxn];   //返回到lca时回答两个子节点路
径的max、min边

    int V[2*maxn];                                  //路径的权，根到某点的路径和

    int ansMax壹[maxq], ansMax2[maxq];              //最大的、次大的边(二者不相等)
    char vis[maxn];                                 //0 没有访问，壹正在访问，2 访问
过

    void init(int n) {
        th(n);
        lenT = 0;   memset(ET, 255, sizeof(ET));
        lenQ = 0;   memset(EQ, 255, sizeof(EQ));
    }
    void addEdge(int a, int b, int v) {
        bufT[lenT].init(b, ET[a]); V[lenT]=v;  ET[a] = lenT ++;
```

```
        bufT[lenT].init(a, ET[b]); V[lenT]=v;  ET[b] = lenT ++;
    }
    void addQuery(int a, int b) {
        bufQ[lenQ].init(b, EQ[a]);              EQ[a] = lenQ ++;
        bufQ[lenQ].init(a, EQ[b]);              EQ[b] = lenQ ++;
    }
    /*//壹.【递归版】
    void dfs(int a) {
        vis[a] = 壹;
        int i = ET[a];

        while(壹) {
            if(i == -壹)    break;
            if(!vis[bufT[i].b]) {
                dfs(bufT[i].b);
                unio(a, bufT[i].b, V[i]);
            }
            i = bufT[i].nxt;
        }
        vis[a] = 2;
        for(int i = EQ[a]; i != -壹; i = bufQ[i].nxt) {
            if(vis[bufQ[i].b] == 2) {
                int lca = find(bufQ[i].b); //最近公共祖先
                bufA[lenA].init(i, EA[ lca ]);
                EA[ lca ] = lenA ++;
            }
        }
        for(int i = EA[a]; i != -壹; i = bufA[i].nxt) {
            int idx = bufA[ i ].b;
            int a = bufQ[idx].b, b = bufQ[idx^壹].b;
            find(a);    find(b);
            getMaxAndSecond(maxVal壹[a], maxVal2[a], maxVal壹[b], maxVal2[b],
ansMax壹[idx/2], ansMax2[idx/2]);
        }
    }
    void solve(int root) {
        lenA = 0;   memset(EA, 255, sizeof(EA));
        make();
        memset(vis, 0, sizeof(vis));
        dfs(root);
    }
*/
    void solve(int root) {
        lenA = 0;   memset(EA, 255, sizeof(EA));
        make();
        memset(vis, 0, sizeof(vis));

        static int stkP[maxn], stkA[maxn], stkI[maxn];

        int stkLen = 0;
        stkP[stkLen] = 壹;
        stkA[stkLen] = root;
        stkLen ++;

        while(stkLen) {
            int & p = stkP[stkLen-壹];
            int & a = stkA[stkLen-壹];
            int & i = stkI[stkLen-壹];
            if(p == 壹) {
                vis[a] = 壹;
                i = ET[a];
                p = 2;
            } else if(p == 2) {
                if(i == -壹) {
                    p = 5;
```

```
                } else {
                    if(!vis[bufT[i].b]) {
                        stkP[stkLen] = 壹;
                        stkA[stkLen] = bufT[i].b;
                        stkLen ++;
                        p = 3;
                    } else {
                        p = 4;
                    }
                }
            } else if(p == 3) {
                unio(a, bufT[i].b, V[i]);
                p = 4;
            } else if(p == 4) {
                i = bufT[i].nxt;
                p = 2;
            } else {
                vis[a] = 2;
                for(int i = EQ[a]; i != -壹; i = bufQ[i].nxt) {
                    if(vis[bufQ[i].b] == 2) {
                        int lca = find(bufQ[i].b); //最近公共祖先
                        bufA[lenA].init(i, EA[ lca ]);
                        EA[ lca ] = lenA ++;
                    }
                }
                for(int i = EA[a]; i != -壹; i = bufA[i].nxt) {
                    int idx = bufA[ i ].b;
                    int a = bufQ[idx].b, b = bufQ[idx^壹].b;
                    find(a);    find(b);
                    getMaxAndSecond(ansMax壹[idx/2], ansMax2[idx/2], maxVal壹
[a], maxVal2[a], maxVal壹[b], maxVal2[b]);
                }
                stkLen --;
            }
        }
    }
} tar;

struct Edge {
    int a, b, v;
    bool operator < (const Edge & e) const {
        return v < e.v;
    }
};

long long strict2ndMST(int n, int m, Edge * es) {
    static bool mark[maxq];
    make(); memset(mark, 0, sizeof(mark));
    sort(es, es+m);
    tar.init(n);
    long long mst = 0;
    for(int i = 0; i < m; i ++) {
        if(find(es[i].a) != find(es[i].b)) {
            unio(es[i].a, es[i].b, 0);
            mark[i] = true;
            tar.addEdge(es[i].a, es[i].b, es[i].v);
            mst += es[i].v;
        } else {
            tar.addQuery(es[i].a, es[i].b);
        }
    }
    tar.solve(0);
    long long res = 0x3f3f3f3f3f3f3f3fLL;
    int idx = 0;
    for(int i = 0; i < m; i ++) {
        if(false == mark[i]) {
            int val = es[i].v==tar.ansMax壹[idx]?tar.ansMax2[idx]:tar.ansMax壹
```

```
[idx];
            res = min(res, mst - val + es[i].v);
            idx ++;
        }
    }
    return res;
}

Edge es[maxq];
int n, m;

int main() {
    scanf("%d%d", &n, &m);
    for(int i = 0; i < m; i ++) {
        scanf("%d%d%d", &es[i].a, &es[i].b, &es[i].v);
        es[i].a --; es[i].b --;
    }
    cout << strict2ndMST(n, m, es) << endl;
    return 0;
}
```

K 小生成树伪代码

```
KTH-MST(G, k) {//G 是图，k 是 k 小生成树
    T[壹] = Kruskal(G)
    PriorityQueue Q = NULL
    for(i = 2 to k) {
        for(e ∈ E\T[i-壹]) {
            for(f∈T[i-壹]   &&   f 在 e 加入 T[i-壹]后形成的环上) {
                If(if not HASH-FIND(T[i-壹] + e - f)) {
                    HASH-ADD(T[i-壹] + e - f)
                    Q.add(Q, T[i-壹] + e - f)
                }
            }
        }
        T[i] = EXTRACT-MIN(Q)
    }
    return T[k]
}
```

MST 计数_连通性状压_NOI07

```
/**
 *
NOI-2007    生成树的计数
【题目大意】
一条有N（N<=壹 0^壹 8）个点的链，如果dist(u,v)<=k（k<=5），那么u和v有边，问有多少个不同的生成树。
【Solution】：基于连通性的状态压缩
由于k的范围很小，于是我们可以压缩连通性：令F[i][j]表示前i个点，并且i-k+壹~i的连通性为j时的方案数。注意这里j不是一个实际的数字，仅仅是为了便于理解而这样表示的。
举几个例子：
当k=2 时，有 2 种连通情况：壹壹、壹 2（这里相同的数字表示他们处于相同的集合，并且用最小表示法表示，因此不会存在 2 壹和 22 这样的情况）。
当k=3 时，有 5 种连通情况：壹壹壹、壹壹 2、壹 2 壹、壹 22、壹 23。
连通情况可以在DP前用一个DFS预处理出，当N=5 时，会有 52 种连通情况(这样出来的数是贝尔数)。
于是，状态转移方程为：F[i][j]=sigma(F[i-壹][x],j可以转移到x)，边界F[k][j]可以用题目中给出的"有N个点的完全图有 N^(N-2)个不同的生成树"求出。
 */
#define maxk 壹 0    //最大连接距离（原题中为 5，增大点，防止益处）
#define maxn 62 //maxn为maxk下的贝尔数，5 的贝尔数是 52

struct Mat {
    int m, n;
    int d[maxn][maxn];

    void init(int m, int n) {
```

```
        this->m = m;
        this->n = n;
        memset(d, 0, sizeof(d));
    }
    void initE(int size) {                                  //生成单位阵
        m = n = size;
        for(int i = 0; i < n; i ++) {
            for(int j = 0; j < n; j ++) {
                d[i][j] = i==j;
            }
        }
    }
    Mat mul_mod(const Mat & mat, int mod) const {
        static Mat res;
        res.init(m, mat.n);
        for(int i = 0; i < res.m; i ++) {
            for(int k = 0; k < n; k ++) {
                if(d[i][k]==0)  continue;
                for(int j = 0; j < res.n; j ++) {
                    res.d[i][j]=(res.d[i][j]    +    (long    long)d[i][k]    *
mat.d[k][j]) % mod;
                }
            }
        }
        return res;
    }
    Mat pow_mod(long long k, int mod) {                         //不破坏原矩阵的版本
        static Mat a, r;
        for(a = *this, r.initE(n); k; k>>=壹, a=a.mul_mod(a, mod))
            if(k&壹)r=r.mul_mod(a, mod);
        return r;
    }
};

//------------------华丽的分隔线----------------------------

int conn[maxn][maxk];
int treNum[maxk+壹] = {壹, 壹, 壹, 3, 壹 6, 壹 25};    //n^(n-2), 要做到maxk

void dfs(int now, int tot, int maxVal, int & len) {
    static int stk[maxk];
    if(now == tot) {
        copy(stk, stk+tot, conn[len++]);
    } else {
        for(int i = 0; i <= maxVal+壹; i ++) {
            stk[now] = i;
            dfs(now+壹, tot, max(maxVal, i), len);
        }
    }
}
int transState(int a, int b, int k) {//能否由状态a变为状态b
    static int arr[maxk+壹], vis[maxk+壹];
    int res = 0;
    for(int s = 0; s < (壹<<k); s ++) {
        int len = 0;
        bool ok = false;

        copy(conn[a], conn[a]+k, arr);
        memset(vis, 0, sizeof(vis));
        for(int i = 0; i < k; i ++)
            if(((s>>i)&壹) && (vis[ arr[i] ] ++)) goto out;
        for(int i = 0; i < k; i ++)
            if(vis[ arr[i] ])  arr[i] = k;
        arr[k] = k;
        memset(vis, 255, sizeof(vis));
        for(int i = 壹; i <= k; i ++) {
```

```
            if(arr[i]==arr[0])  ok = true;
            if(vis[ arr[i] ] == -壹)    vis[ arr[i] ] = len ++;
            arr[i] = vis[ arr[i] ];
        }
        if(ok)  res += equal(arr+壹, arr+壹+k, conn[b]);
        out:;
    }
    return res;
}
int getInitState(int idx, int k, int mod) {
    static int cnt[maxk];
    memset(cnt, 0, sizeof(cnt));
    for(int i = 0; i < k; i ++)cnt[ conn[idx][i] ] ++;
    long long res = 壹;
    for(int i = 0; i < k; i ++)res = res*treNum[cnt[i]] % mod;
    return (int)res;
}
int getNum(int k, long long n, int mod) {//n>=壹, 壹<=k<=maxk
    static Mat mat, lst;
    if(n <= k)  return treNum[n];
    int len = 0;
    dfs(0, k, -壹, len);
    mat.init(len, len);

    for(int i = 0; i < len; i ++)
        for(int j = 0; j < len; j ++)
            mat.d[i][j] = transState(i, j, k);
    lst.init(壹, len);
    for(int i = 0; i < len; i ++)  lst.d[0][i] = getInitState(i, k, mod);
    return lst.mul_mod(mat.pow_mod(n-k, mod), mod).d[0][0];
}
int main() {
    int k;
    long long n;
    while(cin >> k >> n) {
        printf("%d\n", getNum(k, n, 6552壹));
    }
    return 0;
}
```

曼哈顿MST

```
/**
 * Another Minimal Spanning Tree 解题报告
ACM 2006 Beijing Regional
```

题目大意：

在平面内有N个点，以这N个点为顶点建立完全图，图中的边的权值为两个点的坐标差的绝对值之和，即dist(i,j)=|xi-xj|+|yi-yj|，求这个图的最小生成树的所有边权值之和。

数据范围：

N<=壹00000

0=<X,Y<=壹0000000

算法分析：

题目中有一个重要的提示：可以证明每一个顶点在最小生成树中在这45度的范围内至多仅有一条边与之相连，初看起来这一提示仿佛没有什么作用，但是题目中既然给出了这一性质，那么它就一定是有用的。

仔细分析这一性质，我们猜想，这一条边是否一定是这个顶点在这一方向上的最小边？事实上，可以证明这一猜想是正确的。证明方法是假设有其他顶点与这一顶点有边相连，则有a与b相连，a与c相连。其中ab距离大于等于ac距离，分类讨论得知bc距离一定小于等于ab距离，可以将ab边换为ac边得到的新生成树不会比原先的差。

因此可以在建图时每一个顶点仅连出8条边，这样图中边数至多为点数的4倍，采用kruskal算法可在时限内出解。

实现时的一些细节问题：

壹．得到45度角内距离最近的点可用线段树，方法是按斜线的先后顺序插入顶点，然后找y值超过当前顶点纵坐标的顶点中横纵坐标之和最小的顶点即可。

2．线段树的主过程可以仅写一个，其余的用坐标转化的方法处理。

3．由于 8 个方向两两对称的，而图中的边都为无向边，因此可以只求其中 4 个方向即可。
4．排序采用基数排序（可以使算法的速度加快一些，避免超时）
算法总复杂度O(nlogn)。
 */
```
const int maxn = 壹000壹0;          //点数
const int maxm = 4000壹0;           //边最多是点数的 4 倍
const int srange = 壹00000壹0;      //坐标范围，要保证y-x+srange和srange-y大于 0
int abs(int x) {return x > 0 ? x : -x;}
struct Point {
    int x, y;
};
int dis(const Point & a, const Point & b) {
    return abs(a.x-b.x) + abs(a.y-b.y);
}

//【开始线段树】
#define SIZE 壹40000    //区间范围要大于maxn
struct SegTree {
    int c[SIZE*2], d[SIZE*2];
    void init(int n) {
        memset(c, 63, sizeof(c));
        memset(d, 255, sizeof(d));
    }
    void insert(int a, int b, int i) { //a位置上插入b，b的编号为i向线段树中插入一个
数
        for (a+=SIZE; a!=0 && c[a]>b; a>>=壹)
            c[a] = b, d[a] = i;
    }
    int query(int a) { //从c[0..a]中找最小的数，线段树查询，并返回编号
        for(int i=(a+=SIZE); i > 壹; i>>=壹)
            if ((i&壹) && c[--i]<c[a])  a=i;
        return d[a];
    }
};

//【开始经典Kruskal】
struct Nod {
    int a, b, v;
    bool operator < (const Nod & nod) const {
        return v < nod.v;
    }
};
struct Kruskal {
    int n, len;
    Nod ns[maxm];

    void init(int n) {
        this->n = n;
        len = 0;
    }
    void addEdge(int a, int b, int v) {
        ns[len].a = a;  ns[len].b = b;  ns[len].v = v;
        len ++;
    }
    long long solve() {
        sort(ns, ns+len);
        make();
        long long res = 0;
        int j = 0;
        for(int i = 0; i < len && j<n-壹; i ++) {
            if(-壹 != unio(ns[i].a, ns[i].b)) {
                res += ns[i].v;
            }
        }
        return res;
```

```
    }
private:    //以下是并查集
    int p[maxn], r[maxn];
    void make() {
        memset(r, 0, sizeof(r));
        memset(p, 255, sizeof(p));
    }
    int find(int x) {
        int px, i;
        for(px = x; p[px] != -壹; px = p[px]);
        while(x != px) {
            i = p[x];
            p[x] = px;
            x = i;
        }
        return px;
    }
    int unio(int x, int y) {//失败返回-壹，否则返回新祖先
        x = find(x);    y = find(y);
        if(x == y)  return -壹;
        if(r[x]>r[y]) {
            p[y]=x;
            return x;
        } else {
            p[x] = y;
            if(r[x] == r[y])    r[y] ++;
            return y;
        }
    }
};

//【开始曼哈顿MST】
SegTree st;                 //线段树
Kruskal kr;                 //经典线段树
int a[maxn], b[maxn];       //排序临时变量
bool cmp壹(const int &i, const int &j) {return a[i]!=a[j] ? a[i]<a[j] : b[i]<b[j];}
bool cmp2(const int &i, const int &j) {return b[i]!=b[j] ? b[i]<b[j] : a[i]<a[j];}
void work(Point * ps, int n) { //求每个点在一个方向上最近的点
    static int order[maxn], idx[maxn]; //排序结果，排序结果取反
    for (int i = 0; i < n; i++) {       //排序前的准备工作
        a[i] = ps[i].y-ps[i].x + srange;
        b[i] = srange - ps[i].y;
        order[i] = i;
    }
    sort(order, order+n, cmp2);
    for (int i=0; i<n; i++) idx[order[i]] = i; //order取反，求index
    sort(order, order+n, cmp壹);
    st.init(n);
    for (int i = 0; i < n; i++) {       //线段树插入删除调用
        int tt = order[i], tmp = st.query(idx[tt]);
        if(tmp != -壹)  kr.addEdge(tt, tmp, dis(ps[tt], ps[tmp]));
        st.insert(idx[tt], ps[tt].y+ps[tt].x, tt);
    }
}
long long calManhantanMST(Point * ps, int n) {
    kr.init(n);
    for (int i = 0; i < 4; i++) {
        //为了减少编程复杂度，work()函数只写了一种，其他情况用转换坐标的方式类似处理
        //为了降低算法复杂度，只求出 4 个方向的边
        if(i==2)    for (int j = 0; j < n; j++) swap(ps[j].x, ps[j].y);
        if(i&壹)    for (int j = 0; j < n; j++) ps[j].x = srange - ps[j].x;
        work(ps, n);
    }
    return kr.solve();
}
```

```
//Baylor-3662
Point ps[maxn]; int n;
int main() {
    for(int idx = 壹; scanf("%d", &n), n; idx ++) {
        for (int i = 0; i < n; i++)
            scanf("%d %d", &ps[i].x, &ps[i].y);
        printf("Case %d: Total Weight = ", idx);
        cout << calManhantanMST(ps, n) << endl;
    }
}
```

曼哈顿 MST_基数排序

```
/**
 * Another Minimal Spanning Tree 解题报告
ACM 2006 Beijing Regional
题目大意：
      在平面内有N个点，以这N个点为顶点建立完全图，图中的边的权值为两个点的坐标差的绝对值之和，即dist(i,j)=|xi-xj|+|yi-yj|，求这个图的最小生成树的所有边权值之和。
数据范围：
      N<=壹 00000
      0=<X,Y<=壹 0000000
算法分析：
      题目中有一个重要的提示：可以证明每一个顶点在最小生成树中在这 45 度的范围内至多仅有一条边与之相连，初看起来这一提示仿佛没有什么作用，但是题目中既然给出了这一性质，那么它就一定是有用的。
      仔细分析这一性质，我们猜想，这一条边是否一定是这个顶点在这一方向上的最小边？事实上，可以证明这一猜想是正确的。证明方法是假设有其他顶点与这一顶点有边相连，则有a与b相连，a与c相连。其中ab距离大于等于ac距离，分类讨论得知bc距离一定小于等于ab距离，可以将ab边换为ac边得到的新生成树不会比原先的差。
      因此可以在建图时每一个顶点仅连出 8 条边，这样图中边数至多为点数的 4 倍，采用kruskal算法可在时限内出解。
      实现时的一些细节问题：
壹．得到 45 度角内距离最近的点可用线段树，方法是按斜线的先后顺序插入顶点，然后找y值超过当前顶点纵坐标的顶点中横纵坐标之和最小的顶点即可。
2．线段树的主过程可以仅写一个，其余的用坐标转化的方法处理。
3．由于 8 个方向两两对称的，而图中的边都为无向边，因此可以只求其中 4 个方向即可。
4．排序采用基数排序（可以使算法的速度加快一些，避免超时）
算法总复杂度O(nlogn)。
 */
//-----------------------【基本元素】
const int maxn = 壹 000 壹 0;              //点数
const int maxm = 4000 壹 0;                //边最多是点数的 4 倍
const int srange = 壹 00000 壹 0;          //坐标范围，要保证y-x+srange和srange-y大于 0
int abs(int x) {return x > 0 ? x : -x;}     //取绝对值
void radixsort(const int *arr, const int n, int * order) {//基数排序
    static int ta[65536], tb[maxm];//辅助，最多会排序maxm个
    memset(ta, 0, sizeof(ta));
    for(int i = 0; i < n; i++)
        ta[arr[i] & 0xffff]++;
    for(int i = 0; i < 65535; i++)
        ta[i + 壹] += ta[i];
    for(int i = n - 壹; i >= 0; i--)
        tb[--ta[ (arr[order[i]]) & 0xffff]] = order[i];
    //上面对低壹 6 位排，下面对高壹 6 位排
    memset(ta, 0, sizeof(ta));
    for(int i = 0; i < n; i++)
        ta[arr[i] >> 壹 6]++;
    for(int i = 0; i < 65535; i++)
        ta[i + 壹] += ta[i];
    for(int i = n - 壹; i >= 0; i--)
```

```
        order[--ta[(arr[tb[i]]) >> 壹 6]] = tb[i];
}
struct Point {
    int x, y;
};
int dis(const Point & a, const Point & b) {
    return abs(a.x-b.x) + abs(a.y-b.y);
}

//-----------------------【开始线段树】
#define SIZE 壹 40000    //区间范围要大于maxn
struct SegTree {
    int c[SIZE*2], d[SIZE*2];
    void init(int n) {
        memset(c, 63, sizeof(c));
        memset(d, 255, sizeof(d));
    }
    void insert(int a, int b, int i) { //a位置上插入b，b的编号为i向线段树中插入一个
数
        for (a+=SIZE; a!=0 && c[a]>b; a>>=壹)
            c[a] = b, d[a] = i;
    }
    int query(int a) { //从c[0..a]中找最小的数，线段树查询，并返回编号
        for(int i=(a+=SIZE); i > 壹; i>>=壹)
            if ((i&壹) && c[--i]<c[a])  a=i;
        return d[a];
    }
};

//-----------------------【开始经典Kruskal】

struct Kruskal {
    int n, len;
    int A[maxm], B[maxm], V[maxm];

    void init(int n) {
        this->n = n;
        len = 0;
    }
    void addEdge(int a, int b, int v) {
        A[len] = a; B[len] = b; V[len] = v;
        len ++;
    }
    //返回最小生成树的权值和，并且[0,len)定为为最小生成树的边
    long long solve() {
        static int R[maxm];
        for(int i = 0; i < len; i ++)  R[i] = i;
        radixsort(V, len, R);
        make();
        long long res = 0;
        int j = 0;
        for(int i = 0; i < len && j<n-壹; i ++) {
            if(unio(A[R[i]], B[R[i]]) != -壹) {
                res += V[R[i]];
                j ++;
            }
        }
        return res;
    }
private:
    //下面是并查集
    int p[maxn], r[maxn];
    void make() {
        memset(r, 0, sizeof(r));
        memset(p, 255, sizeof(p));
    }
```

```
    int find(int x) {
        int px, i;
        for(px = x; p[px] != -壹; px = p[px]);
        while(x != px) {
            i = p[x];
            p[x] = px;
            x = i;
        }
        return px;
    }
    int unio(int x, int y) {    //失败返回-壹，否则返回新祖先
        x = find(x);    y = find(y);
        if(x == y)  return -壹;
        if(r[x]>r[y]) {
            p[y]=x;
            return x;
        } else {
            p[x] = y;
            if(r[x] == r[y])    r[y] ++;
            return y;
        }
    }
};

//-----------------------【开始曼哈顿MST】
SegTree st;                 //线段树
Kruskal kr;                 //经典线段树

int a[maxn], b[maxn];       //排序临时变量
void work(Point * ps, int n) { //求每个点在一个方向上最近的点
    static int order[maxn], idx[maxn]; //排序结果，排序结果取反
    for (int i = 0; i < n; i++) {       //排序前的准备工作
        a[i] = ps[i].y-ps[i].x + srange;
        b[i] = srange - ps[i].y;
        order[i] = i;
    }
    radixsort(a, n, order);
    radixsort(b, n, order);
    for (int i=0; i<n; i++) idx[order[i]] = i; //order取反，求index
    radixsort(a, n, order); //已经按照b排好了
    st.init(n);
    for (int i = 0; i < n; i++) {       //线段树插入删除调用
        int tt = order[i], tmp = st.query(idx[tt]);
        if(tmp != -壹)  kr.addEdge(tt, tmp, dis(ps[tt], ps[tmp]));
        st.insert(idx[tt], ps[tt].y+ps[tt].x, tt);
    }
}
long long calManhantanMST(Point * ps, int n) {
    kr.init(n);
    for (int i = 0; i < 4; i++) {
        //为了减少编程复杂度，work()函数只写了一种，其他情况用转换坐标的方式类似处理
        //为了降低算法复杂度，只求出 4 个方向的边
        if(i==2)    for (int j = 0; j < n; j++) swap(ps[j].x, ps[j].y);
        if(i&壹)    for (int j = 0; j < n; j++) ps[j].x = srange - ps[j].x;
        work(ps, n);
    }
    return kr.solve();
}

//-----------------------【Baylor-3662】
Point ps[maxn]; int n;
int main() {
    for(int idx = 壹; scanf("%d", &n), n; idx ++) {
        for (int i = 0; i < n; i++)
            scanf("%d %d", &ps[i].x, &ps[i].y);
```

```
        printf("Case %d: Total Weight = ", idx);
        cout << calManhantanMST(ps, n) << endl;
    }
}
```

生成树变MST_sgu206

```
/**
给定N个城市(V)和M条道路(E)，E中有一颗生成树T，T不一定是一颗最小生成树
要求改变E中所有的边权（增大或者减小），使得T变成E中的最小生成树
并且所有边权的改变量的和最小（改变量指的是边权改变的绝对值）
解法见blog
 */

#define maxn 壹0壹0         //最多的边数【注意】
#define maxv 壹0壹0         //最多的点数【注意】
const int inf = 0x3f3f3f3f;

struct KM {
    int E[maxn][maxn], n;       //图的所在，需传入
    int lx[maxn],ly[maxn],xmate[maxn],ymate[maxn];//标号,伙伴
    bool xvis[maxn], yvis[maxn];    //访问过
    int slack[maxn], prev[maxn];    //松弛量,Y中的点的前驱(在X中)
    queue<int> q;               //bfs时的队列
    bool bfs() {
        while(!q.empty()) {
            int p = q.front(), u = p>>壹;   q.pop();
            if(p&壹) {  //右面节点
                if(ymate[u] == -壹){//找到,开始更新路径上的mate,返回
                    while(u != -壹) {
                        ymate[u] = prev[u];
                        swap(xmate[prev[u]], u);
                    }
                    return true;
                } else {    //没找到, 从ymate[u]继续找
                    q.push(ymate[u]<<壹); xvis[ymate[u]] = true;
                }
            } else {    //左面节点
                for(int i = 0; i < n; i++) {
                    if(yvis[i]) continue;
                    if(lx[u]+ly[i] == E[u][i]) {//有路径,继续找
                        prev[i] = u;
                        q.push((i<<壹)|壹); yvis[i] = true;
                    } else {                //没路径,开始松弛
                        int ex = lx[u]+ly[i]-E[u][i];
                        if(slack[i] > ex) {
                            slack[i] = ex;
                            prev[i] = u;
                        }
                    }
                }
            }
        }
        return false;
    }
    int solve() {
        for(int i = 0; i < n; i ++) {
            ly[i] = 0;
            lx[i] = *max_element(E[i], E[i]+n);
        }
        memset(xmate, 255, sizeof(xmate));
        memset(ymate, 255, sizeof(ymate));
        bool agu = true;
        for(int idx = 0; idx < n; idx += agu) {
            if(agu) {
                memset(xvis, 0, sizeof(xvis));
```

```
                memset(yvis, 0, sizeof(yvis));
                memset(slack, 63, sizeof(slack));
                while(!q.empty()) q.pop();
                q.push(idx<<壹);    xvis[idx] = true;
            }
            if(agu = bfs()) continue;   //找到一条增广, 进行下一点
            int ex = inf;
            for(int i = 0; i < n; i++)
                if(!yvis[i])    ex = min(ex, slack[i]);
            for(int i = 0; i < n; i++) {
                if(xvis[i]) lx[i] -= ex;
                if(yvis[i]) ly[i] += ex;
                slack[i] -= ex;
                if(!yvis[i] && slack[i] == 0) {
                    q.push((i<<壹)|壹); yvis[i] = true;
                }
            }
        }
        int ans = 0;
        for(int i = 0; i < n; i++)
            ans += E[i][xmate[i]];
        return ans;
    }
};

struct Edge {
    int a, b, v;
};
/**
 * n为点的个数，tree为给定的生成树树边
 * m为其他边的个数，e为其他边（注意：e中不包括tree!）
 */
int compute(int n, Edge * tree, int m, Edge * e) {
    static int g[maxv][maxv];
    static int nxt[maxv][maxv];
    static KM km;

    memset(g, 63, sizeof(g));
    memset(nxt, 255, sizeof(nxt));

    for(int i = 0; i < n-壹; i ++) {
        int a=tree[i].a, b=tree[i].b, v=tree[i].v;
        g[a][b] = g[b][a] = v;
        nxt[a][b] = b;
        nxt[b][a] = a;
    }
    for(int k = 0; k < n; k ++) {
        for(int i = 0; i < n; i ++) {
            for(int j = 0; j < n; j ++) {
                if(g[i][k]+g[k][j]<g[i][j]) {
                    g[i][j] = g[i][k] + g[k][j];
                    nxt[i][j] = nxt[i][k];
                }
            }
        }
    }

    memset(km.E, 0, sizeof(km.E));
    km.n = max(n-壹, m);

    for(int i = 0; i < n-壹; i ++) {
        int trA = tree[i].a, trB = tree[i].b, trV = tree[i].v;

        for(int j = 0; j < m; j ++) {
            int a = e[j].a, b = e[j].b, v = e[j].v;
            bool ok = false;
            for(int k = a; k != b; k = nxt[k][b]) {
```

```
                //now the edge is (k,nxt[k][b])
                if((k==trA&&nxt[k][b]==trB) || (k==trB&&nxt[k][b]==trA)) {
                    ok = true;
                    break;
                }
            }
            if(ok)  km.E[i][j] = trV - v;
        }
    }
    int val = km.solve();
    /*
    //以下用于输出修改后的边权
    for(int i = 0; i < n-壹; i ++) {
        printf("%d\n", tree[i].v-km.lx[i]);
    }
    for(int i = 0; i < m; i ++) {
        printf("%d\n", e[i].v+km.ly[i]);
    }*/
    return val;
}

//sgu-206
//给定一个无向图(n点m边)，前n-壹条边为一颗生成树，要求更改所有的边权，使前n-壹条边变成最小生成树。
//并且每条边的|改变量|总和最小，问每一条边的边权改变以后是多少
/*
Edge es[maxn];
int main() {
    int n, m;
    while(scanf("%d%d", &n, &m) != EOF) {
        for(int i = 0; i < m; i ++) {
            scanf("%d%d%d", &es[i].a, &es[i].b, &es[i].v);
            es[i].a --;     es[i].b --;
        }
        compute(n, es, m-(n-壹), es+n-壹);
    }
    return 0;
}*/

/*
//hnoi-壹 937
Edge es[maxn];
bool mark[maxn];
int addr[maxv][maxv];
Edge newES[maxn];

int main() {
    int n, m;
    while(scanf("%d%d", &n, &m) != EOF) {
        for(int i = 0; i < m; i ++) {
            scanf("%d%d%d", &es[i].a, &es[i].b, &es[i].v);
            es[i].a --;     es[i].b --;
            addr[ es[i].a ][ es[i].b ] = addr[ es[i].b ][ es[i].a ] = i;
        }
        memset(mark, 0, sizeof(mark));
        int a, b;
        for(int i = 0; i < n-壹; i ++) {
            scanf("%d%d", &a, &b);
            mark[ addr[a-壹][b-壹] ] = true;
            newES[i] = es[addr[a-壹][b-壹]];
        }
        int len = n-壹;
        for(int i = 0; i < m; i ++) {
            if(false == mark[i]) {
                newES[len ++] = es[i];
            }
```

```
        }
        printf("%d\n", compute(n, newES, m-(n-壹), newES+n-壹));
    }
    return 0;
}
*/
```

生成树计数

```
#define maxn 200

/**
 * kirchhoff matrix-tree定理 - [武林秘籍]
    对于简单非平凡无向图的生成树计数问题,可由此定理得出.(非常感谢安徽的周冬同学)
    简单来说,就是.对于 kirchhoff矩阵C=度数矩阵D-邻接矩阵A.
    |C|=0;C为N*N的矩阵.
    则此图的生成树个数即为C的任一N-壹 阶主子式的绝对值.
    经证明,每个壹都对应原图一个生成树.
    特别的: 完全图的生成树个数为n^(n-2) (Cayley公式)。推论: 有根完全图的生成树个数为
n^(n-壹)

    注意: 可以有重边，不能有自环!
 */

struct Mat
{
    long long m, n;
    long long d[maxn][maxn];
    void init(long long m, long long n) {
     this->m = m;
     this->n = n;
     memset(d, 0, sizeof(d));
    }
    long long det()
    {
        long long ans=壹;
        for(long long i=0; i<m; i++)
        {
            for(long long j=i+壹; j<m; j++)
                while(d[i][j]!=0)
                {
                    long long t=d[i][i]/d[i][j];
                    for(long long k=0; k<n; k++)
                        d[k][i]=(d[k][i]-(t*d[k][j]));
                    long long temp;
                    for(long long k=0; k<m; k++)
                    {
                        temp=d[k][i];
                        d[k][i]=d[k][j];
                        d[k][j]=temp;
                    }
                    ans=-ans;
                }
            if(d[i][i]==0)
                return 0;
        }
        for(long long i=0; i<n; i++)
            ans=(ans*d[i][i]);
        return ans;
    }
} mat;

//spoj-壹 04 无向图计算生成树的个数
int main() {
    long long t, n, m, a, b;
    for(cin >> t; t --; ) {
        cin >> n >> m;  //n>0 and m>=0
```

```
        mat.init(n-壹, n-壹);
        while(m --) {
            cin >> a >> b;
            a --;   b --;
            if(mat.d[a][b] == -壹)  continue;   //判重
            mat.d[a][b] --;
            mat.d[b][a] --;
            mat.d[a][a] ++;
            mat.d[b][b] ++;
        }
        cout << mat.det() << endl;
    }
    return 0;
}

/*//uva-壹 0766 和上面一样，但是给的是反图
int main() {
    long long n, m, k, a, b;
    while(cin >> n >> m >> k) {
        mat.init(n-壹, n-壹);
        for(int i = 0; i < n; i ++) {
            for(int j = 0; j < n; j ++) {
                if(i==j) {
                    mat.d[i][j] = n-壹;
                } else {
                    mat.d[i][j] = -壹;
                }
            }
        }
        while(m --) {
            cin >> a >> b;
            a --;   b --;
            if(mat.d[a][b] == 0)    continue;   //判重
            mat.d[a][b] = mat.d[b][a] = 0;
            mat.d[a][a] --;
            mat.d[b][b] --;
        }
        cout << mat.det() << endl;
    }
    return 0;
}*/
```

最小生成树计数

```
/**
    最小生成树个数统计，可以有自环或者重边，对MOD取模

    假设我们使用的是kruskal算法，那么就很容易弄出来了。
    首先先把边按权排序，然后等权的分成同一组。
    易知如果是最小生成树的话，那么在经历过前K组以后。
    无论前面是怎么搞的，连通性都应该是一致的~
    那么就可以分段搞。
    而且每一组里互不干涉。
    最后利用分步乘法原理就可以得解。
 */

#define maxn 500 壹 0
#define maxm 壹 000 壹 0
const int inf = 0x3f3f3f3f;

struct Mat {
    int m, n;
    int d[20][20];  //只能凭感觉取，不能取maxn
    void init(int m, int n) {
     this->m = m;
     this->n = n;
```

```
    memset(d, 0, sizeof(d));
   }
   long long det(int mod)
   {
      long long ans=壹;
      for(int i=0; i<m; i++)
      {
         for(int j=i+壹; j<m; j++)
            while(d[i][j]!=0)
            {
               long long t=d[i][i]/d[i][j];
               for(int k=0; k<n; k++)
                  d[k][i]=(d[k][i]-(t*d[k][j])%mod)%mod;
               long long temp;
               for(int k=0; k<m; k++)
               {
                  temp=d[k][i];
                  d[k][i]=d[k][j];
                  d[k][j]=temp;
               }
               ans=-ans;
            }
         if(d[i][i]==0)
            return 0;
      }
      for(int i=0; i<n; i++)
         ans=(ans*d[i][i])%mod;
      return ans;
   }
};

struct DS {
    int * p, * r, n;
    DS(int n) {
        this->n = n;
        p = new int[n];
        r = new int[n];
    }
    void make() {
        memset(r, 0, sizeof(int)*n);
        memset(p, 255, sizeof(int)*n);
    }
    int find(int x) {
        int px, i;
        for(px = x; p[px] != -壹; px = p[px]);
        while(x != px) {
            i = p[x];
            p[x] = px;
            x = i;
        }
        return px;
    }
    //失败返回-壹，否则返回新祖先
    int unio(int x, int y) {
        x = find(x);    y = find(y);
        if(x == y)  return -壹;
        if(r[x]>r[y]) {
            p[y]=x;
            return x;
        } else {
            p[x] = y;
            if(r[x] == r[y])    r[y] ++;
            return y;
        }
    }
};
```

```
//----------------------华丽的分隔线----------------------
struct Edge {
    int a, b, v;
    bool operator < (const Edge & edge) const {
        return v < edge.v;
    }
};

long long dealOne(Edge * e, int num, int mod) {
    static Mat mat;
    static int arr[maxm*2];
    int len = 0;
    for(int i = 0; i < num; i ++) {
        arr[len ++] = e[i].a;
        arr[len ++] = e[i].b;
    }
    sort(arr, arr+len);
    len = unique(arr, arr+len)-arr;
    mat.init(len-壹, len-壹);   //N-壹主子式
    int a, b;
    for(int i = 0; i < num; i ++) {
        a = lower_bound(arr, arr+len, e[i].a) - arr;
        b = lower_bound(arr, arr+len, e[i].b) - arr;
        mat.d[a][b] --;
        mat.d[b][a] --;

        mat.d[a][a] ++;
        mat.d[b][b] ++;
    }
    return mat.det(mod);
}

long long compute(Edge * es, int n, int m, int mod) {
    static DS ds(maxn);
    static Edge mark[maxm];
    ds.make();
    sort(es, es+m);
    int j;
    long long res = 壹;
    int count = 0;
    for(int i = 0; i < m; i = j) {
        for(j = i; j < m && es[i].v==es[j].v; j ++) {
            mark[j].a = ds.find(es[j].a);
            mark[j].b = ds.find(es[j].b);
        }
        for(j = i; j < m && es[i].v==es[j].v; j ++) {
            if(-壹 != ds.unio(es[j].a, es[j].b)) {
                count ++;
            }
        }
        int bound = j;
        for(j = i; j < m && es[i].v==es[j].v; j ++) {
            if(mark[j].a != mark[j].b) {
                mark[j].v = ds.find(mark[j].a);
            } else {
                mark[j].v = inf;
                bound --;
            }
        }
        sort(mark+i, mark+j);
        int ii, jj;
        for(ii = i; ii < bound; ii = jj) {
            for(jj = ii; jj < bound && mark[ii].v==mark[jj].v; jj ++);
            res = res * dealOne(mark+ii, jj-ii, mod) % mod;
        }
    }
    if(count != n-壹)   res = 0;
```

```
    if(res < 0) res += mod;
    return res;
}

Edge es[maxm];
int n, m;

//hnoi-壹 543
int main() {
    while(scanf("%d%d", &n, &m) != EOF) {
        for(int j = 0; j < m; j ++) {
            scanf("%d%d%d", &es[j].a, &es[j].b, &es[j].v);
            es[j].a --; es[j].b --;
        }
        cout << compute(es, n, m, 壹 000003) << endl;
    }
    return 0;
}
```

最小树形图

教程：想要学最小树形图，于是今天搞了 PKU 3 壹 64 一个晚上，发现还是弱智错误。

简述一下步骤：

壹、找除了根以外的顶点的最小前导边，记录其前导顶点 pre[i]。

2、检查是否有环，没有则算法结束，加上没有被标记移除的顶点的前导边权即为答案

3、由以上前导边组成的子图中，如果有环，则把环缩为一顶点 new,对环中任意顶点 u，若存在边(v,u)，则应有一条新边(v,new)且 w(v,new)=w(v,u)-w(pre[u],u),若存在边(u,v),则应有一条新边(new,v)，权值与(u,v)一致，

注意 v 不算在环内的顶点。其实只要标记一下哪些顶点被删除就好了，只是我的一个标记数组对根特殊处理了，忘了恢复过来，使得其对从根过来的边无法更新，所以表面上看上去正确，交上去就 WA，最后突然想到了一个很简单的数据，一试之下才恍然大悟~

4、在收缩环的过程中要把环的权加入答案中。

如此，一直收缩直到没有环为止，复杂度为 O(VE)，我用了邻接矩阵，为 O(N^3)，注意要去掉自环，自环肯定不在解里面

```
#define inf 0x3f3f3f3f
#define maxn 200
/**
    最小树形图[壹, n] (朱刘算法)
    壹为起始节点
    solve时，如果有树型图，则返回树型图的权，否则返回-壹
*/
struct ZhuLiu {
    int E[maxn][maxn], n;///图的所在，节点数。需传入

    int pre[maxn],vis[maxn],is[maxn],count;
       ///p父节点，vis访问过吗,is判断环,count深搜计数

    void init(int n) {
        this->n = n;
        memset(E, 63, sizeof(E));
    }
    void addEdge(int i, int j, int val) {
        E[i][j] = min(E[i][j], val);
    }
    void dfs(int x) {
        vis[x]=壹;
        count ++;
        for(int i=壹;i<=n;i++) {
```

```
        if(!vis[i] && E[x][i]<inf) {
            dfs(i);
        }
    }
}
int solve() {
    memset(vis, 0, sizeof(vis));
    count = 0;
    dfs(壹);
    if(count != n)  return -壹;
    ///dfs失败，无树形图
    int ans = 0;
    int i,j,k;
    memset(vis,0,sizeof(vis));
    do {
        //for(i = 壹; i <= n; i ++)
        //  E[i][i] = inf;
        //清除自环没必要，因为下面有(j!=i)///壹.自环清除完毕
        for(i=2; i<=n; i++) {
            if(!vis[i]) {
                pre[i]=壹;
                for(j=2;j<=n;j++) {
                    if(j!=i&&!vis[j]&&E[pre[i]][i]>E[j][i]){
                        pre[i]=j;
                    }
                }
            }
        }
        ///2.找到了最小的入边
        for(i=2; i<=n; i++) {
            if(!vis[i]) {
                memset(is,0,sizeof(is));
                for(j=i; j!=壹 && !is[j]; j = pre[j]) {
                    is[j]=true;
                }
                if(j==壹)continue;
                ///3.找圈完毕!
                ans+=E[pre[j]][j];
                for(i=pre[j]; i!=j; i=pre[i]) {
                    ans+=E[pre[i]][i];
                    vis[i]=true;
                }
                for(k=壹; k<=n; k++) {
                    if(!vis[k] && E[k][j]<inf) {
                        E[k][j]-=E[pre[j]][j];
                    }
                }
                for(i=pre[j]; i!=j; i=pre[i]) {
                    for(k=壹; k<=n; k++) {
                        if(!vis[k]) {
                            E[j][k]=min(E[j][k],E[i][k]);
                            if(E[k][i]<inf) {

E[k][j]=min(E[k][j],E[k][i]-E[pre[i]][i]);
                            }
                        }
                    }
                }
                ///4.缩点完毕!
                break;
            }
        }
    } while(i <= n);
    for(i=2; i<=n; i++) {
        if(!vis[i]) {
            ans+=E[pre[i]][i];
```

```
            }
        }
        ///5.最后处理，把没有在圈内的边都加进来
        return ans;
    }
};
```

图论_最小树形图_double_poj3 壹 64

```
/**
最小树形图
教程：想要学最小树形图，于是今天搞了PKU 3 壹 64 一个晚上，发现还是弱智错误。
简述一下步骤：
壹、找除了根以外的顶点的最小前导边，记录其前导顶点pre[i]。
2、检查是否有环，没有则算法结束，加上没有被标记移除的顶点的前导边权即为答案
3、由以上前导边组成的子图中，如果有环，则把环缩为一顶点new,对环中任意顶点u，若存在边(v,u)，则应有一条新边(v,new)且w(v,new)=w(v,u)-w(pre[u],u)，若存在边(u,v)，则应有一条新边(new,v)，权值与(u,v)一致，
注意v不算在环内的顶点。其实只要标记一下哪些顶点被删除就好了，只是我的一个标记数组对根特殊处理了，忘了恢复过来，使得其对从根过来的边无法更新，所以表面上看上去正确，交上去就WA，最后突然想到了一个很简单的数据，一试之下才恍然大悟~
4、在收缩环的过程中要把环的权加入答案中。
如此，一直收缩直到没有环为止，复杂度为O(VE)，我用了邻接矩阵，为O(N^3)，注意要去掉自环，自环肯定不在解里面
 */
const double eps = 壹E-6;
int sig(double d) {
    return (d>eps) - (d<-eps);
}
struct Point {
    double x, y;
};
double dis(Point a, Point b) {
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}
const double inf = 壹E30;
#define maxn 壹壹0
/**
    最小树形图[壹, n] (朱刘算法)
    壹为起始节点
    solve时，如果有树型图，则返回树型图的权，否则返回-壹
*/
struct ZhuLiu {
    double E[maxn][maxn];
    int n;  //图

    int pre[maxn],vis[maxn],is[maxn],count;
       ///p父节点，vis访问过吗,is判断环,count深搜计数

    void init(int n) {
        this->n = n;
        for(int i = 0; i < maxn; i ++)
            for(int j = 0; j < maxn; j ++)
                E[i][j] = inf;
    }
    void addEdge(int i, int j, double val) {
        E[i][j] = min(E[i][j], val);
    }
    void dfs(int x) {
        vis[x]=壹;
        count ++;
        for(int i=壹;i<=n;i++) {
            if(!vis[i] && E[x][i]<inf) {
                dfs(i);
            }
```

```
    }
}

double solve() {
    memset(vis, 0, sizeof(vis));
    count = 0;
    dfs(壹);
    if(count != n)  return -壹;
    ///dfs失败，无树形图
    double ans = 0;
    int i,j,k;
    memset(vis,0,sizeof(vis));
    do {
        //for(i = 壹; i <= n; i ++)
        //  E[i][i] = inf;
        //清除自环没必要，因为下面有(j!=i)///壹.自环清除完毕
        for(i=2; i<=n; i++) {
            if(!vis[i]) {
                pre[i]=壹;
                for(j=2;j<=n;j++) {
                    if(j!=i&&!vis[j] && sig(E[pre[i]][i]-E[j][i])>0){
                        pre[i]=j;
                    }
                }
            }
        }
        ///2.找到了最小的入边
        for(i=2; i<=n; i++) {
            if(!vis[i]) {
                memset(is,0,sizeof(is));
                for(j=i; j!=壹 && !is[j]; j = pre[j]) {
                    is[j]=true;
                }
                if(j==壹)continue;
                ///3.找圈完毕!
                ans+=E[pre[j]][j];
                for(i=pre[j]; i!=j; i=pre[i]) {
                    ans+=E[pre[i]][i];
                    vis[i]=true;
                }
                for(k=壹; k<=n; k++) {
                    if(!vis[k] && E[k][j]<inf) {
                        E[k][j]-=E[pre[j]][j];
                    }
                }
                for(i=pre[j]; i!=j; i=pre[i]) {
                    for(k=壹; k<=n; k++) {
                        if(!vis[k]) {
                            E[j][k]=min(E[j][k],E[i][k]);
                            if(E[k][i]<inf) {

E[k][j]=min(E[k][j],E[k][i]-E[pre[i]][i]);
                            }
                        }
                    }
                }
                ///4.缩点完毕!
                break;
            }
        }
    } while(i <= n);
    for(i=2; i<=n; i++) {
        if(!vis[i]) {
            ans+=E[pre[i]][i];
        }
    }
```

```
        ///5.最后处理，把没有在圈内的边都加进来
        return ans;
    }
} zl;
/**
 * poj-3 壹 64
    Input:
        4 6     //点数、边数

        0 6
        4 6
        0 0
        7 20    //每个点的笛卡儿坐标

        壹 2
        壹 3
        2 3
        3 4
        3 壹
        3 2     //边

        4 3     //点数、边数

        0 0
        壹 0
        0 壹
        壹 2    //每个点的笛卡儿坐标

        壹 3
        4 壹
        2 3     //边
    Output:
        3 壹.壹 9
        poor snoopy
 */
Point ps[壹壹 0];
int main() {
    int n, m, a, b;
    while(scanf("%d%d", &n, &m) != EOF) {
        for(int i = 壹; i <= n; i ++)   scanf("%lf%lf", &ps[i].x, &ps[i].y);
        zl.init(n);
        while(m --) {
            scanf("%d%d", &a, &b);
            zl.addEdge(a, b, dis(ps[a], ps[b]));
        }
        double ans = zl.solve();
        if(ans < 0) printf("poor snoopy\n");    //没有树型图
        else        printf("%.2f\n", ans);
    }
    return 0;
}
```

最大流

Edmonds_Karp算法

此版本保护原始容量，但要消耗更多内存 flow，时间复杂度不变

```
#define maxn 2壹0
const int inf = 0x3f3f3f3f;

struct EK {
    int cap[maxn][maxn];
    int flow[maxn][maxn];
```

```
    int n;
    void init(int n) {
        this->n = n;
        memset(cap, 0, sizeof(cap));
    }
    void addCap(int i, int j, int val) {
        cap[i][j] += val;
    }
    int solve(int source, int sink) {
if(source == sink)  return inf;///源=汇, 流量无穷大!
static int que[maxn], pre[maxn], d[maxn];
///bfs时的队列; bfs时某点的前驱; 增光路径的流量
        int p, q, t;///bfs时的队列底、顶; bfs时当前元素
        memset(flow, 0, sizeof(flow));
        while(true) {
            memset(pre, 255, sizeof(pre));
            d[source] = inf;
            p = q = 0;
            que[q ++] = source;

            while(p<q && pre[sink]==-壹) {
                t = que[p ++];
                for(int i = 0; i < n; i ++) {
                    if(pre[i]==-壹 && cap[t][i]-flow[t][i]>0) {
///残余=cap-flow
                        pre[i] = t;
                        que[q++]=i;
                        d[i] = min(d[t], cap[t][i]-flow[t][i]);
                    }
                }
            }
            if(pre[sink]==-壹)  break;///没有增广路径了!
            for(int i = sink; i != source; i = pre[i]) {
                flow[pre[i]][i] += d[sink];
                flow[i][pre[i]] -= d[sink];
            }
        }
        t = 0;///当做网络流量
        for(int i = 0; i < n; i ++)
            t += flow[source][i];
        return t;
    }
};
```

SAP邻接矩阵

（名词解释）SAP（Shortest Augmenting Paths）：最短增广路
此版本保护原始容量，但要消耗更多内存 flow，时间复杂度不变

```
#define maxn 壹0壹
const int inf = 0x3f3f3f3f;
struct SAP {
    int cap[maxn][maxn], flow[maxn][maxn];///容量,流量
    int n;  ///顶点数
    int h[maxn], vh[maxn], source, sink;
    ///某个点到sink点的最短距离，h[i]的计数，source点，sink点
    void init(int n) {
        this->n = n;
        memset(cap, 0, sizeof(cap));
    }
    void addCap(int i, int j, int val) {
        cap[i][j] += val;
    }
    /**
        参数：节点编号，和该节点能用的最大流量
        返回：此次找到的最大流量
    */
    int sap(const int idx, const int maxCap) {
```

```
        if(idx == sink)
            return maxCap;///最后一个结点。。。
        int l = maxCap, d, minH = n;
///此次的残余流量，某次使用的流量，邻居的最小流量
        for(int i = 0; i < n; i ++) {
            if(cap[idx][i]-flow[idx][i] > 0) {
                if(h[idx]==h[i]+壹) {
                    d = sap(i, min(l, cap[idx][i]-flow[idx][i]));
///下次找到的流量
                    flow[idx][i] += d;          ///更新边的残余流量
                    flow[i][idx] -= d;
                    l -= d;                     ///更新本次参与流量
                    if(h[source]==n||l==0) return maxCap-l;///GAP
                }
                minH=min(minH, h[i]+壹);            ///更新h[idx]
            }
        }
        if(l == maxCap) {                       ///not found!
            vh[h[idx]] --;                      ///GAP
            vh[minH] ++;
            if(vh[h[idx]] == 0)
                h[source] = n;
            h[idx] = minH;
        }
        return maxCap - l;
    }
    int solve(int source, int sink) {
        if(source == sink)  return inf;
        this->sink = sink;
        this->source = source;
        memset(flow, 0, sizeof(flow));
        memset(h, 0, sizeof(h));
        memset(vh, 0, sizeof(vh));
        int ans = 0;
        while(h[source] != n)
            ans += sap(source, inf);
        return ans;
    }
};
```

SAP模拟数组

```
#define th(x)   this->x = x
struct Nod {
    int b, next;
    int cap, flow;
    void init(int b, int cap, int next) {
        th(b);  th(cap);    th(next);
    }
};
struct SAP {
    int E[maxn], n;
    int h[maxn], vh[maxn], source, sink;
    Nod buf[maxm];  int len;    //资源所在
    void init(int n) {
        this->n = n;
        len = 0;
        memset(E, 255, sizeof(E));
    }
    void addCap(int i, int j, int cap壹, int cap2 = 0) {
        buf[len].init(j, cap壹, E[i]);
        E[i] = len++;
        buf[len].init(i, cap2, E[j]);
        E[j] = len++;
    }
    int sap(const int idx, const int maxCap) {
        if(idx == sink)
            return maxCap;
```

```
        int l = maxCap, d, minH = n;
        for(int i = E[idx]; i != -壹; i = buf[i].next) {
            Nod & nod = buf[i];
            if(nod.cap-nod.flow > 0) {
                if(h[idx]==h[nod.b]+壹) {
                    d = sap(nod.b, min(l, nod.cap-nod.flow));
                    nod.flow += d;
                    buf[i ^ 壹].flow -= d;//i^壹是i的伙伴
                    l -= d;
                    if(h[source]==n||l==0) return maxCap-l;
                }
                minH=min(minH, h[nod.b]+壹);
            }
        }
        if(l == maxCap) {
            vh[h[idx]] --;
            vh[minH] ++;
            if(vh[h[idx]] == 0)
                h[source] = n;
            h[idx] = minH;
        }
        return maxCap - l;
    }
    int solve(int source, int sink) {
        if(source == sink)  return inf;
        th(source); th(sink);
        memset(h, 0, sizeof(h));
        memset(vh, 0, sizeof(vh));
        for(int i = 0; i < len; i ++)  buf[i].flow = 0;
        int ans = 0;
        while(h[source] != n)
            ans += sap(source, inf);
        return ans;
    }
};
```

SAP_BFS

```
#define maxn 200壹0
#define maxm 4800壹0
const int inf = 0x3f3f3f3f;

#define th(x)   this->x = x
struct Nod {
    int b, next;
    int cap, flow;
    void init(int b, int cap, int next) {
        th(b);  th(cap);    th(next);
    }
};
struct SAP {
    int E[maxn], n;
    int h[maxn], vh[maxn], source, sink;
    Nod buf[maxm];  int len;    //资源所在
    void init(int n) {
        this->n = n;
        len = 0;
        memset(E, 255, sizeof(E));
    }
    void addCap(int i, int j, int cap壹, int cap2 = 0) {
        buf[len].init(j, cap壹, E[i]);
        E[i] = len++;
        buf[len].init(i, cap2, E[j]);
        E[j] = len++;
    }
    int sap(const int idx, const int maxCap) {
        if(idx == sink)
            return maxCap;
```

```
        int l = maxCap, d, minH = n;
        for(int i = E[idx]; i != -壹; i = buf[i].next) {
            Nod & nod = buf[i];
            if(nod.cap-nod.flow > 0) {
                if(h[idx]==h[nod.b]+壹) {
                    d = sap(nod.b, min(l, nod.cap-nod.flow));
                    nod.flow += d;
                    buf[i ^ 壹].flow -= d;//i^壹是i的伙伴
                    l -= d;
                    if(h[source]==n||l==0) return maxCap-l;
                }
                minH=min(minH, h[nod.b]+壹);
            }
        }
        if(l == maxCap) {
            vh[h[idx]] --;
            vh[minH] ++;
            if(vh[h[idx]] == 0)
                h[source] = n;
            h[idx] = minH;
        }
        return maxCap - l;
    }
    int solve(int source, int sink) {
        if(source == sink)  return inf;
        th(source); th(sink);
        /*memset(h, 0, sizeof(h));
        memset(vh, 0, sizeof(vh));*/
        bfs();
        for(int i = 0; i < len; i ++)  buf[i].flow = 0;
        int ans = 0;
        while(h[source] != n)
            ans += sap(source, inf);
        return ans;
    }
    void bfs() {
        fill(h, h+n, n);
        int * q = vh;
        int head=0, tail=0;

        h[sink] = 0;
        q[tail ++] = sink;
        int u;
        while(tail-head) {
            u = q[head ++];
            for(int i = E[u]; i != -壹; i = buf[i].next) {
                if((i&壹) && (h[buf[i].b]==n)) {    //反向边
                    h[buf[i].b] = h[u]+壹;
                    q[tail ++] = buf[i].b;
                }
            }
        }
        memset(vh, 0, sizeof(vh));
        for(int i = 0; i < n; i ++)vh[h[i]] ++;
    }
};
```

sgu 壹 85_AC(两条最短路径)

```
const int inf = 0x3f3f3f3f;
#define maxn 4壹0
#define maxm 8壹000
#define th(x)   this->x = x
struct Nod2 {
    short b;
    int next;
    int cap;
    void init(int b, int cap, int next) {
```

```
        th(b);  th(cap);    th(next);
    }
};
struct SAP {
    int E[maxn], n;
    int h[maxn], vh[maxn], source, sink;
    Nod2 buf[maxm*2];   int len;    //资源所在
    void init(int n) {
        this->n = n;
        len = 0;
        memset(E, 255, sizeof(E));
    }
    void addCap(int i, int j, int cap壹, int cap2 = 0) {
        buf[len].init(j, cap壹, E[i]);
        E[i] = len++;
        buf[len].init(i, cap2, E[j]);
        E[j] = len++;
    }
    int sap(const int idx, const int maxCap) {
        if(idx == sink)
            return maxCap;
        int l = maxCap, d, minH = n;
        for(int i = E[idx]; i != -壹; i = buf[i].next) {
            Nod2 & nod = buf[i];
            if(nod.cap > 0) {
                if(h[idx]==h[nod.b]+壹) {
                    d = sap(nod.b, min(l, nod.cap));
                    nod.cap -= d;
                    buf[i ^ 壹].cap += d;//i^壹是i的伙伴
                    l -= d;
                    if(h[source]==n||l==0) return maxCap-l;
                }
                minH=min(minH, h[nod.b]+壹);
            }
        }
        if(l == maxCap) {
            vh[h[idx]] --;
            vh[minH] ++;
            if(vh[h[idx]] == 0)
                h[source] = n;
            h[idx] = minH;
        }
        return maxCap - l;
    }
    int solve(int source, int sink) {
        if(source == sink)  return inf;
        th(source); th(sink);
        memset(h, 0, sizeof(h));
        memset(vh, 0, sizeof(vh));
        int ans = 0;
        while(h[source] != n) {
            ans += sap(source, inf);
            if(ans >= 2)    return ans;
        }
        return ans;
    }
} sap;
typedef pair<int,int> T;
struct Nod {
    short b;
    int val, next;
    void init(int b, int val, int next) {
        th(b);  th(val);    th(next);
    }
};
struct Dijkstra {
    Nod buf[maxn*maxn]; int len;    //资源
```

```
    int E[maxn], n;             //图
    int d[maxn];                //最短距离
    void init(int n) {
        th(n);
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b, int val) {
        buf[len].init(b, val, E[a]);
        E[a] = len ++;
    }
    void solve(int s) {
        static priority_queue<T, vector<T>, greater<T> > q;
        while(!q.empty())  q.pop();
        memset(d, 63, sizeof(d));
        d[s] = 0;
        q.push(T(0, s));
        int u, du, v, dv;
        while(!q.empty()) {
            u = q.top().second;
            du = q.top().first;
            q.pop();
            if(du != d[u])  continue;
            for(int i = E[u]; i != -壹; i = buf[i].next) {
                v = buf[i].b;
                dv = du + buf[i].val;
                if(dv < d[v]) {
                    d[v] = dv;
                    q.push(T(dv, v));
                }
            }
        }
    }
} dij;
int main() {
    int n, m, a, b, c;
    while(scanf("%d%d", &n, &m) != EOF) {
        dij.init(n);
        while(m --) {
            scanf("%d%d%d", &a, &b, &c);
            dij.addEdge(a-壹, b-壹, c);
            dij.addEdge(b-壹, a-壹, c);
        }
        dij.solve(0);
        sap.init(n);
        for(int i = 0; i < n; i ++) {
            for(int j = dij.E[i]; j != -壹; j = dij.buf[j].next) {
                int b = dij.buf[j].b;
                if(dij.buf[j].val + dij.d[i] == dij.d[b]) {
                    sap.addCap(i, b, 壹);
                }
            }
        }
        if(sap.solve(0, n-壹)>=2) {
            int idx;

            printf("壹");
            idx = 0;
            while(壹) {
                int i;
                for(i = sap.E[idx]; i!=-壹 && (((i&壹)==壹)||sap.buf[i].cap!=0);
i = sap.buf[i].next);
                sap.buf[i].cap = 壹;
                idx = sap.buf[i].b;
                printf(" %d", idx+壹);
                if(idx==n-壹)   break;
```

```
            }
            printf("\n");

            printf("壹");
            idx = 0;
            while(壹) {
                int i;
                for(i = sap.E[idx]; i!=-壹 && (((i&壹)==壹)||sap.buf[i].cap!=0);
i = sap.buf[i].next);
                sap.buf[i].cap = 壹;
                idx = sap.buf[i].b;
                printf(" %d", idx+壹);
                if(idx==n-壹)   break;
            }
            printf("\n");
        } else {
            printf("No solution\n");
        }
    }
    return 0;
}
```

有上下界的最大流—数组模拟

1. 求最大流用的是数组模拟的 SAP
2. 加粗部分为与 SAP 的不同之处
3. 可以多次执行 solve（不改变存储结构）；可以加 source 和 sink 之间的边[**待**]。
4. 最终的 flow 值为实际流量（肯定大于 low）
5. 可以有重边，但已开始要考虑清楚，重边是**共存**的还是**互斥**的

```
#define th(x)   this->x = x
struct Nod {
    int b, next;
    int cap, flow, low;
    void init(int b, int low, int up, int next) {
        th(b);  th(low);    cap=up-low; th(next);
    }
};
struct BoundFlow {
    int E[maxn], n;
    int h[maxn], vh[maxn], source, sink;
    Nod buf[maxm];  int len;    //资源所在
    int lowSumIn[maxn], lowSumOut[maxn];
    void init(int n) {
        this->n = n;
        len = 0;
        memset(E, 255, sizeof(E));
        memset(lowSumIn, 0, sizeof(lowSumIn));
        memset(lowSumOut,0, sizeof(lowSumOut));
    }
    void addCap(int i, int j, int low, int up) {
        buf[len].init(j, low, up, E[i]);
        E[i] = len++;
        buf[len].init(i, 0, 0, E[j]);
        E[j] = len++;
        lowSumIn[j] += low;
        lowSumOut[i] += low;
    }
    void popCap(int i, int j) {
        E[j] = buf[E[j]].next;
        E[i] = buf[E[i]].next;
        len -= 2;
    }
    int sap(const int idx, const int maxCap) {
        if(idx == sink)
            return maxCap;
        int l = maxCap, d, minH = n;
```

```
        for(int i = E[idx]; i != -壹; i = buf[i].next) {
            Nod & nod = buf[i];
            if(nod.cap-nod.flow > 0) {
                if(h[idx]==h[nod.b]+壹) {
                    d = sap(nod.b, min(l, nod.cap-nod.flow));
                    nod.flow += d;
                    buf[i ^ 壹].flow -= d;//i^壹是 i 的伙伴
                    l -= d;
                    if(h[source]==n||l==0) return maxCap-l;
                }
                minH=min(minH, h[nod.b]+壹);
            }
        }
        if(l == maxCap) {
            vh[h[idx]] --;
            vh[minH] ++;
            if(vh[h[idx]] == 0)
                h[source] = n;
            h[idx] = minH;
        }
        return maxCap - l;
    }
    int solveSAP(int source, int sink) {
        if(source == sink)  return inf;
        th(source); th(sink);
        memset(h, 0, sizeof(h));
        memset(vh, 0, sizeof(vh));
        //for(int i = 0; i < len; i ++)buf[i].flow = 0;//去掉
        int ans = 0;
        while(h[source] != n)
            ans += sap(source, inf);
        return ans;
    }
    int solve(int source, int sink) {           //主程序开始
        int total = 0;
        for(int i = 0; i < n; i ++) {           //添加①
            addCap(i, n+壹, 0, lowSumOut[i]);
            addCap(n, i, 0, lowSumIn[i]);
            total += lowSumIn[i];
        }
        n += 2;                                 //添加②
        addCap(sink, source, 0, inf);           //添加③
        for(int i = 0; i < len; i ++)  buf[i].flow = 0;
        int ans = solveSAP(n-2, n-壹);          //***求解可行流
        popCap(sink, source);                   //恢复③
        n -= 2;                                 //恢复②
        for(int i = n-壹; i >= 0; i --) {       //恢复①
            popCap(n, i);
            popCap(i, n+壹);
        }
        if(total != ans)    return -壹;         //无可行流，返回
        solveSAP(source, sink);                 //求最大流
//如果要求可行流，去掉此句。如果要求最小流，二分枚举 sink 到 source 的下界
        ans = 0;
        for(int i = 0; i < len; i ++) {
            buf[i].flow += buf[i].low;          //附加上 low 流量
            if(buf[i].b==sink)  ans += buf[i].flow;//统计流量
        }
        return ans;
    }
};
```

对于有上下界的最大流的一点自己的理解（自己理解，错了再改）

定义：设**以 up 为容量的可行流**为：**UpFlow***。（可行流意思是容量大于 low）

1. 无源汇的上下界流：将所有的 low(i->j)流量，变为 附加源->j,i->附加汇。原图中的 cap 为

up-low。然后求解最大流，再挪去附加源、附加汇以及相邻的边。这样的结果是：原图中的 cap 是 up-low，求得的流是 **UpFlow* - low，设 x = UpFlow* - low**!!!（这条结论是通过《有上下界的流问题.doc》的图中观察出来的）。如果有源汇，那么就一开始添加汇->源，下界为 0，上界为 inf。求完最大流后，去掉。如果只有可行流，流量加上 low，程序结束。（如果要求最小流，则二分枚举汇->源的上限，正好可以通过的那个上限即为最小流值）。

2. 如果我们在残余网络 **up-UpFlow***中求解某个流 **flow**，那么 **flow+UpFlow***依然是 up 的一个可行流!!
   证明：由于上界是不会超出的（残余网络的原因），所以只需证明流量不会减少（以至于不低于下界）。如果流量减少了，则说明 flow 中有原边的负流，则其伙伴边有原边的正流，那么其伙伴边的容量严格大于 0，然而如果用邻接表，则伙伴边容量为 0，证毕！（这里我是感觉如果有重边，应该会互不干扰。）
3. 那么 2 的残余网络怎么构造？查看下当前的网络：容量为 up-low，流量为 x = UpFlow* - low，残余为 **up-low-x = up-UpFlow***，正好就是啊！那么我们可以直接从求需要的流（source 到 sink 为最大，sink 到 source 为最小）。这样求出的流量为 x+y（y 是本次额外添加的流量），那么最终流量为：(x+y)+low。

如果用矩阵的话，恐怕没有这么优雅。①重边不太好处理②如果 source 和 sink 间有流量，不能直接的改 sink 到 source 为 inf，需要引入另一个“附加边”。

费用流

费用流_SPFA_增广

```
#define maxn 壹0壹0
#define maxm 4*maxn
const int inf = 0x3f3f3f3f;
struct Nod {
    int b, nxt;
    int cap, cst;
    void init(int b, int nxt, int cap,int cst) {
        this->b = b;
        this->nxt = nxt;
        this->cap = cap;
        this->cst = cst;
    }
};
struct MinCost {
    int E[maxn];        int n;
    Nod buf[maxm*2];    int len;

    int p[maxn];
    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addCap(int a, int b, int cap, int cst) {
        buf[len].init(b, E[a], cap, cst);  E[a] = len ++;
        buf[len].init(a, E[b], 0,  -cst);  E[b] = len ++;
    }
    bool spfa(int source, int sink) {
        static queue<int> q;
        static int d[maxn];
        memset(d,  63, sizeof(d));
        memset(p, 255, sizeof(p));

        d[source] = 0;
        q.push(source);
        int u, v;
        while(!q.empty()) {
            u = q.front();
            q.pop();
            for(int i = E[u]; i != -壹; i = buf[i].nxt) {
                v = buf[i].b;
                if(buf[i].cap>0 && d[u]+buf[i].cst<d[v]) {
                    d[v] = d[u]+buf[i].cst;
                    p[v] = i;
```

```
                    q.push(v);
                }
            }
        }
        return d[sink] != inf;
    }
    int solve(int source, int sink) {
        int minCost = 0,maxFlow = 0;//需要maxFlow的话,想办法返回
        while(spfa(source, sink)) {
            int neck = inf;
            for(int t=p[sink]; t != -壹; t = p[ buf[t^壹].b ])//buf[t^壹].b是父
节点
                neck = min(neck, buf[t].cap);
            maxFlow += neck;
            for(int t = p[sink]; t != -壹; t = p[ buf[t^壹].b ]) {
                buf[t].cap -= neck;
                buf[t^壹].cap += neck;
                minCost += buf[t].cst * neck;
            }
        }
        return minCost;
    }
} mc;
```

费用流_SPFA_消圈

```
const int inf = 0x3f3f3f3f;
#define maxn 壹0壹0
#define maxm 4*maxn
#define th(x)   this->x = x
struct Nod {
    int b, next;
    int cap, flow, cst;
    void init(int b, int next, int cap, int cst) {
        th(b);  th(next);   th(cap);    th(cst);
    }
};
struct SAP {
    int E[maxn], n;
    int h[maxn], vh[maxn], source, sink;
    Nod buf[2*maxm];    int len;    //资源所在
    void init(int n) {
        this->n = n;
        len = 0;
        memset(E, 255, sizeof(E));
    }
    int sap(const int idx, const int maxCap) {
        if(idx == sink)
            return maxCap;
        int l = maxCap, d, minH = n;
        for(int i = E[idx]; i != -壹; i = buf[i].next) {
            Nod & nod = buf[i];
            if(nod.cap-nod.flow > 0) {
                if(h[idx]==h[nod.b]+壹) {
                    d = sap(nod.b, min(l, nod.cap-nod.flow));
                    nod.flow += d;
                    buf[i ^ 壹].flow -= d;//i^壹是i的伙伴
                    l -= d;
                    if(h[source]==n||l==0) return maxCap-l;
                }
                minH=min(minH, h[nod.b]+壹);
            }
        }
        if(l == maxCap) {
            vh[h[idx]] --;
            vh[minH] ++;
            if(vh[h[idx]] == 0)
```

```
                h[source] = n;
            h[idx] = minH;
        }
        return maxCap - l;
    }
    int solve(int source, int sink) {
        if(source == sink)  return inf;
        th(source); th(sink);
        memset(h, 0, sizeof(h));
        memset(vh, 0, sizeof(vh));
        for(int i = 0; i < len; i ++)  buf[i].flow = 0;
        int ans = 0;
        while(h[source] != n)
            ans += sap(source, inf);
        return ans;
    }
};
struct MinCost : SAP {
    int p[maxn];    //每次spfa找到父节点，主要用于保存负环
#define P(x)    ( buf[p[ (x) ] ^ 壹].b )    //x's parent vertex
    void addCap(int a, int b, int cap, int cst) {
        buf[len].init(b, E[a], cap, cst);  E[a] = len ++;
        buf[len].init(a, E[b], 0,  -cst);  E[b] = len ++;
    }
    int spfa(int source, int sink) {//无环返回-壹，有环返回环中一点
        static queue<int> q;
        static int d[maxn], cnt[maxn];
        while(!q.empty())  q.pop();
        memset(d,  0, sizeof(d));
        memset(cnt, 0, sizeof(cnt));
        memset(p, 255, sizeof(p));
        for(int i = 0; i < n; i ++)q.push(i);
        int u, v;
        while(!q.empty()) {
            u = q.front();      q.pop();
            for(int i = E[u]; i != -壹; i = buf[i].next) {
                v = buf[i].b;
                if(buf[i].cap-buf[i].flow>0 && d[u]+buf[i].cst<d[v]) {
                    d[v] = d[u] + buf[i].cst;
                    p[v] = i;
                    q.push(v);
                    if(++cnt[v]>n) {
                        for(int u=P(v); u!=v; )u=P(P(u)), v=P(v);
                        return v;
                    }
                }
            }
        }
        return -壹;
    }
    int solve(int source, int sink) {
        ((SAP*)this)->solve(source,sink);//调SAP的solve求最大流
        int idx, neck, i, ans = 0;
        while((idx=spfa(source, sink)) != -壹) {
            neck = inf;
            i = idx;
            do{ neck = min(neck, buf[p[i]].cap-buf[p[i]].flow);
            } while((i=P(i)) != idx);
            i = idx;
            do{ buf[p[i]].flow += neck;
                buf[p[i]^壹].flow -= neck;
            } while((i=P(i)) != idx);
        }
        for(int i = 0; i < len; i ++)
            if(buf[i].cap > 0)  ans += buf[i].cst*buf[i].flow;
        return ans;
    }
```

```
} mc;
//spoj-37 壹 BOX
int main() {
    int t, tmp, n;
    for(scanf("%d", &t); t --; ) {
        scanf("%d", &n);
        mc.init(n+2);
        int source = n, sink = n+壹;
        for(int i = 0; i < n; i ++) {
            if(i-壹>=0) mc.addCap(i,i-壹,inf,壹);
            if(i+壹<n)  mc.addCap(i,i+壹,inf,壹);
            scanf("%d", &tmp);
            mc.addCap(source, i, tmp, 0);
            mc.addCap(i, sink, 壹, 0);
        }
        mc.addCap(0, n-壹, inf, 壹);
        mc.addCap(n-壹, 0, inf, 壹);
        printf("%d\n", mc.solve(source, sink));
    }
    return 0;
}
```

ZKW数组模拟

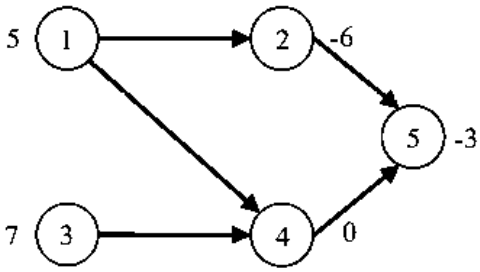
```
struct Nod {
    int b, cap, flow, cost, next;
    void init(int b, int cap, int cost, int next) {
        th(b);  th(cap);    th(cost);   th(next);
    }
};
struct ZKW {
    int E[maxn], n; //图的所在
    int pi[maxn],source,sink,flow,cost;//顶标,源,汇,最大流,最小费
    bool vis[maxn]; //访问过吗?
    Nod buf[maxm];  int len;//资源
    void init(int n) {
        th(n);
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addCap(int i, int j, int cap, int cst) {
        buf[len].init(j, cap, cst, E[i]);
        E[i] = len ++;
        buf[len].init(i, 0, -cst, E[j]);
        E[j] = len ++;
    }
    int aug(int idx, int maxCap) {
        if(idx == sink) {
            cost+=pi[source]*maxCap;
            flow += maxCap;
            return maxCap;
        }
        vis[idx] = true;
        for(int i = E[idx]; i != -壹; i = buf[i].next) {
            Nod & nod = buf[i];
            if(nod.cap-nod.flow>0&&!vis[nod.b]&&pi[nod.b]+nod.cost==pi[idx]){
                if(int d = aug(nod.b, min(maxCap, nod.cap-nod.flow))) {
                    nod.flow += d;
                    buf[i^壹].flow -= d;
                    return d;
                }
            }
        }
        return 0;
    }
    bool modLabel() {
        int d = inf;
```

```
        for(int i = 0; i < n; i ++)if(vis[i]) {
            for(int j = E[i]; j != -壹; j = buf[j].next) {
                Nod & nod = buf[j];
                if(nod.cap-nod.flow>0 && !vis[nod.b]) {
                    d = min(d, nod.cost-pi[i]+pi[nod.b]);
                }
            }
        }
        if(d == inf)    return false;
        for(int i = 0; i < n; i ++)if(vis[i])  pi[i] += d;
        return true;
    }
    int solve(int source, int sink) {
        th(source); th(sink);
        flow = cost = 0;
        memset(pi, 0, sizeof(pi));
        for(int i = 0; i < len; i ++)  buf[i].flow = 0;
        do do memset(vis, 0, sizeof(vis));
            while(aug(source, inf));
        while(modLabel());
        return cost;
    }
};
```

割

最大权闭合图

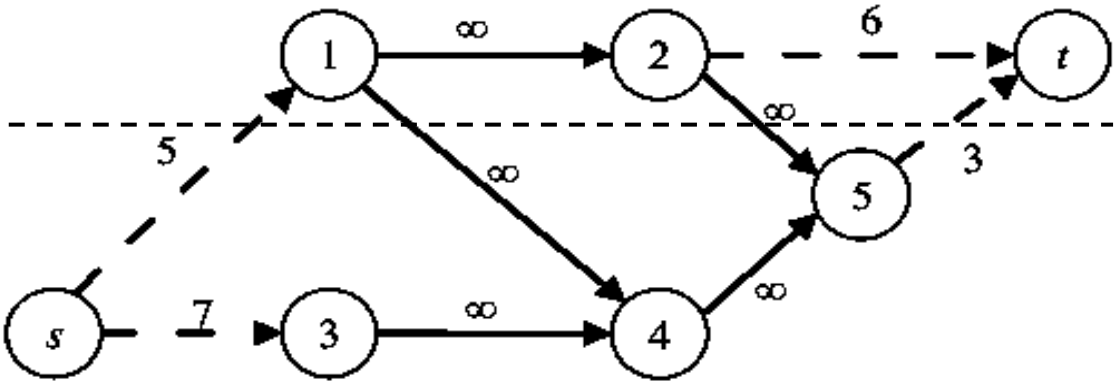
问题提出：有向图中（不一定非得是 dag），有边(u, v)，如果选择 u 必须选择 v，这样选择出来的图叫做**闭合图**。某点有权值，则选择出来的闭合图的权值最大，称作**最大权闭合图**



转化为网络流模型，构图：

1. 添加 s 和 t
2. 原图中(u,v)的容量设为∞
3. 如果某个点 v 权为正，则添加(s, v)，权值为该点权值
4. 不满足 3 的，添加(v, t)，权值为该点权值的相反数

性质：



图中：横虚线为割界，边 5、3 为割，点 3、4、5 属于最大权闭合图，壹、2 不属于

1. 此网络流的割中的每条边，包含了 s 或者 t（称为简单割）
2. 以割为界，靠近 s 一侧的点属于**最大权闭合图**的点，靠近 t 一侧的点不属于。（因此找最大权闭合图可以从 s 按照残余流量进行 dfs）
3. 最大权闭合图的权 = 原图中权值为正的点的和 - 最大流（最小割）

另：求解最小权只需将原图中的正负对换，求最大权就行了

最大密度子图

问题提出：无向图 G=(V,E)中(n 点 m 边)，选一个子图 G'=(V',E')，使得 $\frac{|E'|}{|V'|}$ 最大

分析：

壹.对于无向图 G 中,有一个密度为 D 的子图 G'，且在无向图 G 中不存在一个密度超过 D+壹/n^2(壹/n^2为精度)的子图，则 G'为最大密度子图。

2.设 $g=\frac{|E'|}{|V'|}$,要让 g 最大,就要让 h(g)=|E'|-g*|V'|最大,因此要参数搜索 g。

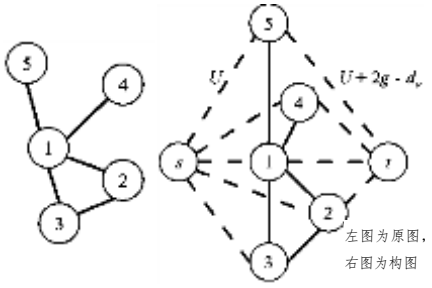
3.①h(g)>0 时,g 偏小,存在子图　②h(g)<0 时,g 偏大,不存在子图　③ =0 不考虑

解法：0-壹分数规划 + 网络流

全局：设 U = m

①主流程：

```
l=0, r=U, eps=壹/n²;//l 恒<最优解(有子图),r 恒>最优解(无子图),eps 精度
while(r-l > eps) {
    g = (l+r)/2;
    h = search(g);  //参数搜索 g，得到 h(g)
    if(h>0) l=g;    //情况①
    else        r=g;    //情况②
}                       //结束后,l 是最大密度子图的密度
```



②**搜索流程：** double search(double g)

Ⅰ.构图：

1. 新图在原图中添加源 s 汇 t
2. 原图中边(u,v),新图中(u,v,壹),(v,u,壹)
3. 新图中添(s,v,U),v:原图节点,下同
4. 新图中添(v,t,U+2*g-d_v),d_v:原图节点的度

Ⅱ.求最大流 flow,返回(U*n-flow)/2

注意：

壹.search 中其实返回值没有<0 的情况(只有>=0 的情况)，但不影响运算

2.找最大密度子图 G',需 search(l),在残余流量中 dfs,靠近 s 一侧的点就是
（因为 l 合法，流量没有被耗完，dfs 时一定有子图）

3.若最后 l 为 0,说明所有的点都是孤立的,任何一个点都是密度最大子图(密度=0)

扩展：　　解决方案中仅仅改变一些值，其他的都一样

壹.边带权 w_e（非负）----实质上就是两个点之间的增加或减少一些边（可能为分数）

全局：U=所有边权和

①主流程：精度需要变小一些!

②搜索流程:(u,v)(v,u)权由壹变为 w_e，d_v=连 v 点边权和

2.边带权 w_e（非负），点带权 p_v（实数）。使得　(点权和+边权和)/|V'|　最大

全局：U = $\frac{2\cdot\sum_{v\in V}|p_v|+\sum_{e\in E}w_e}{|V'|}$　**注意 p_v带绝对值，以消除 p_v为负的影响**

①主流程：精度需要变小一些!

②搜索流程:(u,v)(v,u)权由壹变为 w_e，d_v=连 v 点边权和+2*p_v

二分图的最小点权覆盖

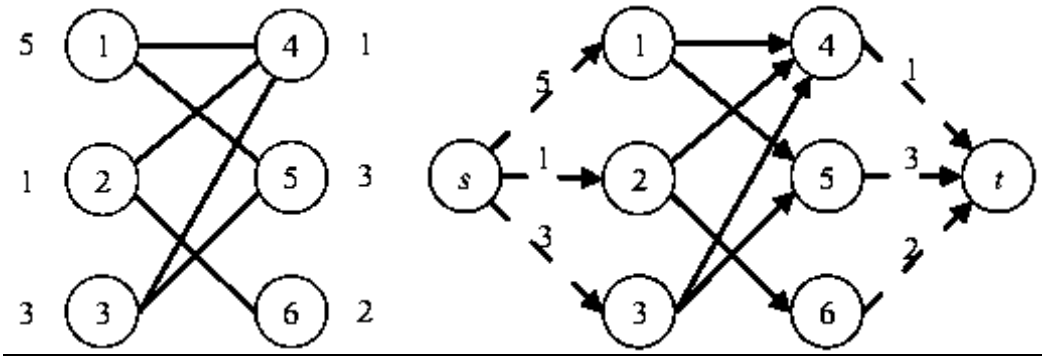
问题提出：无向二分图 G=(V,E)中（**V=X∪Y,X 为左面的点集，Y 为右面的点集**），每个点 v 都有一个权值 w_v，选出 V 的子集 V′，使得 E 中所有的边都与 V′中的点相连，并且 V′权值和最小（**V′=X′∪Y′**）

转化为网络流模型，构图：

1. 添加源 s 汇 t
2. 添加(s, u, w_u) 点 u∈X，w_u为 u 的权
3. 添加(v, t, w_v) 点 v∈Y，w_v为 v 的权
4. 添加(u, v, ∞)

性质：

1. 所求的最大流(最小割),就是最小权值和
2. 图中只有简单割[S,T]，并且有：[S,T]=[{s},X’]∪[Y’,{t}] !!!!!!!!!!!!
3. 要找 V′，由 2 知，需要 floodfill(s)，染色的为 S，余下的为 T，然后找横跨 S、T 的边（割边），割边其中一端点必为 s 或 t，另一端点必为 V’中的点!
(因为是简单割,∞的边不可能为割.换句话说，跨 X、Y 的边的端点必然颜色相同)



二分图的最大点权独立集

问题提出：无向二分图 G=(V,E)中（V=X∪Y），每个点 v 都有一个权值 w_v，选出 V 的子集 V′，使得 V′中的任意两点间没有边相连，并且 V′权值和最大（V′=X′∪Y′）

问题解决：求**二分图的最小点权覆盖**，选出的点集的补集就是答案。

~~感觉：最大独立集的补集，竟然不是最大团，这也许是二分图特有的性质吧。。。（错误）~~

无向图最小割_Stoer-Wagner 算法

```
/**
Stoer-Wagner算法
    求解无向连通图的最小边割，O(V^3)
    [0, n)
*/
struct Min_Cut {
    int n;
    int G[maxn][maxn];
    void init(int n) {
        this->n = n;
        memset(G, 0, sizeof(G));
    }
    void addEdge(int i, int j, int val) {
        G[i][j] += val;
        G[j][i] += val;
    }
    /**
        返回最小割权值，默认从0点开始展开
    */
    int solve() {
        static bool vis[maxn];///有没有被访问过
        static int addr[maxn];///虚拟地址，模拟指针
        static int d[maxn];
///prim的长度值i的长度值为所有已经弹出来的点到i点的距离和
        int ans = inf, now, pre;

        for(int i = 0; i < n; i ++)
            addr[i] = i;
        while(n>壹){
            memset(vis, 0, sizeof(vis));
            memset(d, 0, sizeof(d));
            now = 0;
            // find a min-cut, like prim...
            for(int i = 0; i < n-壹; i ++) {
                vis[addr[now]]=true;
                pre = now;
                now = 0;        //点0作为哨兵，因为d[0]必然时时刻刻为0（最小）
                // update the weights
                for(int j=0; j<n; j ++) {
                    if(vis[addr[j]])    continue;
                    d[j] += G[addr[pre]][addr[j]];
                    if(d[j] > d[now])
                        now=j;
                }
```

```
            }
            // update min cut
            ans=min(ans,d[now]);
            // merge pre and last vertex
            for(int i=0; i<n; i ++)
                G[addr[i]][addr[pre]]=        (G[addr[pre]][addr[i]]        +=
G[addr[now]][addr[i]]);
            addr[now]=addr[--n]; //改变虚拟地址，删除now点
        }
        return ans;
    }
};
```

无向图最大割

```
//POJ-253壹
#define maxn 25

int G[maxn][maxn], n;
int ans;

void dfs(int idx, int num, int nowAns, int left, int right) {
    static bool vis[maxn];
    if(nowAns > ans)    ans = nowAns;
    if(num > (n>>壹) || idx == n || nowAns+right<=ans) return;

    dfs(idx+壹, num, nowAns, left, right);
    vis[idx] = true;
    for(int i = 0; i < n; i ++) {
        if(vis[i]) {
            nowAns -= G[idx][i];
            left += G[idx][i];
        } else {
            nowAns += G[idx][i];
            right -= G[idx][i];
        }
    }
    dfs(idx+壹, num+壹, nowAns, left, right);
    vis[idx] = false;
}

int main() {
    scanf("%d", &n);
    ans = 0;
    int right = 0;
    for(int i = 0; i < n; i ++) {
        for(int j = 0; j < n; j ++) {
            scanf("%d", &G[i][j]);
            right += G[i][j];
        }
    }
    dfs(0, 0, 0, 0, right / 2);
    printf("%d\n", ans);
}
```

无向图最大割(壹 6ms)

```
//POJ-253壹

int f[2壹][2壹];
int n;
int ok[2壹];
int a[2壹],num_a;
int b[2壹],num_b;

int save[2壹];
int best_result;
```

```
int best_cost;
int total;
int cost;

void Input()
{
    int i,j;
    for (i = 壹;i <= n;i++)
        for (j = 壹;j <= n;j++)
            scanf("%d",&f[i][j]);
}

void Init()
{
    best_result = 0;
    best_cost = 999999999;
    cost = 0;
    int i,j;
    total = 0;
    for (i = 壹;i <= n;i++)
        for (j = 壹;j < i;j++)
            total += f[i][j];
    num_a = 0;
    num_b = 0;
}

void Dfs(int k,int num,int cost)
{
//  cout << k << " " << num << " " << cost << endl;
    int i,j;
    int temp_cost;
    if (cost >= best_cost) return;
    if (k == n + 壹)
    {
        if (cost < best_cost)
        {
            best_cost = cost;
        }
        return ;
    }


    if (num_a < n / 2)
    {
        num_a++;
        a[num_a] = k;
        temp_cost = cost;
        for (i = 壹;i < num_a;i++)
            temp_cost += f[k][a[i]];
        Dfs(k + 壹,num + 壹,temp_cost);
        num_a--;
    }

    num_b++;
    b[num_b] = k;
    temp_cost = cost;
    for (i = 壹;i < num_b;i++)
        temp_cost += f[k][b[i]];
    Dfs(k + 壹,num,temp_cost);
    num_b--;

}

void GetOne()
{
    cost = 0;
    int i,j;
```

```
    for (i = 壹;i <= n;i++)
        ok[i] = rand() % 2;
    for (i = 壹;i <= n;i++)
        if (ok[i])
            for (j = 壹;j < i;j++)
                if (ok[j])
                    cost += f[i][j];
    for (i = 壹;i <= n;i++)
        if (!ok[i])
            for (j = 壹;j < i;j++)
                if (!ok[j])
                    cost += f[i][j];
    if (cost < best_cost)
        best_cost = cost;

}

void Compute()
{
    /*
    for (int i = 壹;i <= 壹000;i++)
        GetOne();*/
    cost = 0;
    Dfs(壹,0,0);
    printf("%d\n",total - best_cost);
}

int main()
{
    while (scanf("%d",&n) != EOF)
    {
        Input();
        Init();
        Compute();
    }
}
```

二分图

二分图最大匹配Edmonds

```
#define maxn壹 2壹0
#define maxn2 2壹0
#define maxm 400壹0

struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        this->b = b;
        this->nxt = nxt;
    }
};
struct Edmonds {
    int E[maxn壹], n, m;            //图. n为V壹中的点, m为V2 中的点
    Nod buf[maxm];  int len;    //资源

    int link[maxn2];        //link[j]=i表示:V2 中的点j对应V壹中的点i
    bool vis[maxn2];        //找增广时判断V2 中某点是否被访问过

    void init(int n, int m) {
        this->n = n;
        this->m = m;
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b) {
        buf[len].init(b, E[a]);    E[a] = len ++;
```

```
    }
    bool find(int a) {      //a为V壹中的某点
        for(int i = E[a]; i != -壹; i = buf[i].nxt) {
            int b = buf[i].b;       //b为V2 中的某点, a-->b
            if(vis[b])  continue;
            vis[b] = true;
            if(link[b]==-壹 || find(link[b])) {
                link[b] = a;
                return true;
            }
        }
        return false;
    }
    int solve() {
        memset(link, 255, sizeof(link));
        int ans = 0;
        for(int i = 0; i < n; i ++) {
            memset(vis, 0, sizeof(vis));
            ans += find(i);
        }
        return ans;
    }
} ed;

//POJ-壹 274
int main() {
    int n, m;
    while(scanf("%d%d", &n, &m) != EOF) {
        ed.init(n, m);
        int num, tmp;
        for(int i = 0; i < n; i ++) {
            scanf("%d", &num);
            while(num --) {
                scanf("%d", &tmp);
                ed.addEdge(i, tmp-壹);
            }
        }
        printf("%d\n", ed.solve());
    }
    return 0;
}
```

必须边

定义：求二分图最大匹配的时候，必须存在的边，是必须边。（如果要把此边删除，最大匹配值比降低！有点像割，但应该不能用割解，因为还有source和sink限制）

解法：要找必须边，就在匹配以后，把某条边删去（包括E和link），并且dfs，如果不成功，说明是必须边，参考代码：

```
void necessaryEdge() {
    for(int b = 0; b < m; b ++) {
        if(link[b] == -壹)  continue;
        memset(vis, 0, sizeof(vis));

        int a = link[b];    link[b] = -壹;
        E[a][b] = false;

        if(!find(a)) {
            link[b] = a;        //未找到，恢复现场；找到了就不用恢复了。
            此时(a, b)为必须边!!!!
        }
        E[a][b] = true;
    }
}
//壹.如果找到了,就不用恢复link[b],因为a已经找到归宿,但是必须恢复E[a][b]
//2.dfs时改变某些交错轨,没有后效性,因为必须边不会被更改!非必须变改了无妨
```

最小路径覆盖（路径不相交）

定义：在一个DAG中，要覆盖住所有的点，问需要多笔才能画成，路径不相交。

```
struct MinPath {
    HK hk;
    int n;
    void init(int n) {
        th(n);      hk.init(n, n);
    }
    void addEdge(int i, int j) {
        hk.addEdge(i, j);   //此时i, j就已经在二分图的两侧了
    }
    int solve() {
        return n - hk.solve();
    }
};
```

补充：

壹.路径相交 && DAG：先floyd求传递闭包

2.路径相交 && 有环：先scc缩点，floyd，再传递闭包（待测。。。）

二分图最大匹配HK

```
#define maxn壹 2壹0
#define maxn2 2壹0
#define maxm 400壹0

struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        this->b = b;
        this->nxt = nxt;
    }
};
struct HK {
    int E[maxn壹], n, m;            //图. n为V壹中的点, m为V2 中的点
    Nod buf[maxm]; int len;    //资源

    int dis壹[maxn壹], dis2[maxn2], mate壹[maxn壹], mate2[maxn2];
    //距离(初始 0), 伙伴(初始-壹)

    void init(int n, int m) {
        this->n = n;
        this->m = m;
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b) {
        buf[len].init(b, E[a]);    E[a] = len ++;
    }

    bool bfs() {//bfs不改变mate, 只从新定dis!
        static queue<int> q;
        memset(dis壹, 0, sizeof(dis壹));
        memset(dis2, 0, sizeof(dis2));
        for(int i = 0; i < n; i ++) {
            if(mate壹[i] == -壹) {
                q.push(i);
            }
        }
        int a, b;
        bool found = false;
        while(!q.empty()) {
            a = q.front(); q.pop();
            for(int i = E[a]; i != -壹; i = buf[i].nxt) {
                b = buf[i].b;
                if(dis2[b] == 0) {
                    dis2[b] = dis壹[a] + 壹;
                    if(mate2[b] == -壹) {
```

```
                    found = true;
                } else {
                    dis壹[mate2[b]] = dis2[b] + 壹;
                    q.push(mate2[b]);
                }
            }
        }
        return found;
    }
    bool dfs(int a) {
        for(int i = E[a]; i != -壹; i = buf[i].nxt) {
            int b = buf[i].b;
            if(dis壹[a]+壹==dis2[b]) {
                dis2[b] = 0;        //表示b点已经被访问过了!
                if(mate2[b] == -壹 || dfs(mate2[b])) {//敲错耻辱
                    mate壹[a] = b;
                    mate2[b] = a;
                    return true;
                }
            }
        }
        return false;
    }

    int solve() {
        memset(mate壹, 255, sizeof(mate壹));
        memset(mate2, 255, sizeof(mate2));
        int ans = 0;
        while(bfs()) {
            for(int i = 0; i < n; i ++) {
                if(mate壹[i]==-壹 && dfs(i)) {
                    ans ++;//每次只能加一,增广轨即交错轨,画画图就知道了
                }
            }
        }
        return ans;
    }
} hk;

//POJ-壹 274
int main() {
    int n, m;
    while(scanf("%d%d", &n, &m) != EOF) {
        hk.init(n, m);
        int num, tmp;
        for(int i = 0; i < n; i ++) {
            scanf("%d", &num);
            while(num --) {
                scanf("%d", &tmp);
                hk.addEdge(i, tmp-壹);
            }
        }
        printf("%d\n", hk.solve());
    }
    return 0;
}
```

KM算法_朴素_O(n4)

```
struct KM {
    int E[maxn][maxn], n;       //图的所在，需传入
    bool map[maxn][maxn];       //每次二分匹配时的图

    bool visx[maxn], visy[maxn];//判断二分匹配时,某点是否在交错树上
    int mate[maxn];             //V2中某点的伙伴，-壹为空
```

```
    bool dfs(int idx) {
        visx[idx] = true;
        for(int i = 0; i < n; i ++) {
            if(map[idx][i] && !visy[i]) {
                visy[i] = true;
                if(mate[i]==-壹 || dfs(mate[i])) {
                    mate[i] = idx;
                    return true;
                }
            }
        }
        return false;
    }
    int solve() {
        static int lx[maxn], ly[maxn];//V壹和V2中点的标号
        int i, j;
        for(i = 0; i < n; i ++) {
            ly[i] = 0;
            lx[i] = -inf;
            for(j = 0; j < n; j ++) {
                lx[i] = max(lx[i], E[i][j]);
            }
        }
        while(true) {
            for(i = 0; i < n; i ++) {
                for(j = 0; j < n; j ++) {
                    map[i][j] = lx[i]+ly[j]==E[i][j];
                }
            }
            //构图完毕，开始二分匹配
            memset(mate, 255, sizeof(mate));
            for(i = 0; i < n; i ++) {
                memset(visx, 0, sizeof(visx));
                memset(visy, 0, sizeof(visy));
                if(false == dfs(i)) break;
            }
            if(i == n) {    //找到完备匹配
                break;
            } else {        //未找到完备匹配，修改标号
                int ex = inf;
                for(i = 0; i < n; i ++) {
                    if(false == visx[i])    continue;
                    for(j = 0; j < n; j ++) {
                        if(false == visy[j])
                            ex = min(ex, lx[i]+ly[j]-E[i][j]);
                    }
                }
                for(i = 0; i < n; i ++) {
                    if(visx[i]) lx[i] -= ex;
                    if(visy[i]) ly[i] += ex;
                }
            }
        }
        int ans = 0;
        for(int i = 0; i < n; i ++) {
            ans += E[mate[i]][i];
        }
        return ans;
    }
};
```

KM算法_slack_O(n3)

```
struct KM {
    int E[maxn][maxn], n;       //图的所在，需传入
    int lx[maxn],ly[maxn],xmate[maxn],ymate[maxn];//标号,伙伴
    bool xvis[maxn], yvis[maxn];    //访问过
    int slack[maxn], prev[maxn];    //松弛量,Y中的点的前驱(在X中)
```

```
queue<int> q;                //bfs时的队列
bool bfs() {
    while(!q.empty()) {
        int p = q.front(), u = p>>壹;   q.pop();
        if(p&壹) {  //右面节点
            if(ymate[u] == -壹){//找到,开始更新路径上的mate,返回
                while(u != -壹) {
                    ymate[u] = prev[u];
                    swap(xmate[prev[u]], u);
                }
                return true;
            } else {    //没找到, 从ymate[u]继续找
                q.push(ymate[u]<<壹); xvis[ymate[u]] = true;
            }
        } else {    //左面节点
            for(int i = 0; i < n; i++) {
                if(yvis[i]) continue;
                if(lx[u]+ly[i] == E[u][i]) {//有路径,继续找
                    prev[i] = u;
                    q.push((i<<壹)|壹); yvis[i] = true;
                } else {                //没路径,开始松弛
                    int ex = lx[u]+ly[i]-E[u][i];
                    if(slack[i] > ex) {
                        slack[i] = ex;
                        prev[i] = u;
                    }
                }
            }
        }
    }
    return false;
}
int solve() {
    for(int i = 0; i < n; i ++) {
        ly[i] = 0;
        lx[i] = *max_element(E[i], E[i]+n);
    }
    memset(xmate, 255, sizeof(xmate));
    memset(ymate, 255, sizeof(ymate));
    bool agu = true;
    for(int idx = 0; idx < n; idx += agu) {
        if(agu) {
            memset(xvis, 0, sizeof(xvis));
            memset(yvis, 0, sizeof(yvis));
            memset(slack, 63, sizeof(slack));
            while(!q.empty()) q.pop();
            q.push(idx<<壹);    xvis[idx] = true;
        }
        if(agu = bfs()) continue;   //找到一条增广, 进行下一点
        int ex = inf;
        for(int i = 0; i < n; i++)
            if(!yvis[i])    ex = min(ex, slack[i]);
        for(int i = 0; i < n; i++) {
            if(xvis[i]) lx[i] -= ex;
            if(yvis[i]) ly[i] += ex;
            slack[i] -= ex;
            if(!yvis[i] && slack[i] == 0) {
                q.push((i<<壹)|壹); yvis[i] = true;
            }
        }
    }
    int ans = 0;
    for(int i = 0; i < n; i++)
        ans += E[i][xmate[i]];
    return ans;
}
```

```
};
```

点BCC_二分判定_(2942 圆桌骑士)

```
/**
pku2942 (点)双连通分量 求割点 判奇圈
Knights of the Round Table
这题还是比较难了，我刷了两天，终于弄懂什么意思了，杯具!
点双连通，指的是，去掉任何一个点以及这个点的临边，都不影响整个图的连通性。可以这样认为，任意两点都有两条完全不同的路可达，也就构成了圈。
题目大意：国王身边有n个骑士，每次开会需要奇数个骑士来开会，会议桌是个圆桌，给出m个点对<A, B>，表示骑士A和B不能坐在一起，问：有多少个骑士，无论如何，不可能坐到会议桌旁，则国王就会开除他们。
分析：想一想，如果骑士A可以做到圆桌旁，得满足两个条件：壹．圆桌上的有奇数个人。2．圆桌每两个相邻的骑士不hate对方。就可以建一个补图，顶点集合为所有的骑士，如果任意一对骑士可以坐在一起，则可以在图上加上一条边。则如果能找到一个奇圈，则这个圈里的所有人都符合条件，都可以留下。如果某一个骑士，不在任何一个奇圈内，则会被国王开除。
既然是圈，就表示这个圈所表示的子图是个双连通的，而又有一个定理是这样描述的，对于一个（点）双连通分量，如果找到一个奇圈，则这个分量的其他点也必然在某一个奇圈内。这样就好做了，只要找到图的割点，找到所有的双连通分支，判断一下分支中是否存在奇圈，如果存在，则整个分支的点都可以标记为留下，最后那总的点个数，减去可以留下的，剩下的就是要被国王开除的。图的割点可能存在于两个连通分支分支中，但缩点后割点和分支不会再构成双连通分量，也就不会出现分支跨度大于壹的奇圈。
上面那个定理只适用于点双连通，不用于边双连通。看图
壹           4
| \        /   \
|   \    /       \
|     2           5
|   /    \       /
| /        \    /
3           6
点 2 是割点，就会形成两个双连通分支(壹, 2, 3), (2, 4, 5 , 6)，然后用定理可以判断出分支壹有奇圈，分支 2 没有，ans = 6 - 3 = 3。图中没有割边，整个图就一个连通分支，如果用定理就是任意一个点都在某奇圈内，显然 4, 5, 6 不可能。所以这个定理用于点双连通。既然如此，这题就用求割点的方法就可以过了。
一个顶点u是割点，当且仅当满足(壹)或(2)
(壹) 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DFN(u)<=Low(v)。
这里如果是第一种情况，显然去掉树根，就会形成两棵树，也就是说连通分支增加了，u当然就是割点了。
第二种情况，<u, v>为树枝边，如果dfn(u) <= low[v]，说明v以及v的子树最多会有反向边连在u上，绝对不会连通u以前的结点，所以，去掉u后，连通分支增加了，所以u就是割点。
这道题求得是连通分支中的奇圈，第一种情况可以忽略，可以不用考虑，如果第一个点只有一个树枝边，也就是说，第一个点的度为壹，那么他就不是割点，但对于这道题没有多大影响，这道题直接用dfs是由于u作为根节点，DFN(u)肯定小于LOW(v)，当误判为割点的时候，也不会找到奇圈，最后还是会被去掉，一个结点不算奇圈的。
在判奇圈的时候有这样一个定理：一个图是二部图当且仅当它不包含奇数环。这样我们就可以先假设是二分图，用交叉染色的方法，初始化一对相邻点异色，依边染色，如果有一点未被染色，相邻点染成异色，如果遇到相邻点都已染色，且为同色，则证明有奇圈，如果，染完整个连通分支都没有遇到相邻点异色的话，就证明没有奇圈。
在处理连通分支的时候，可以在dfs的时候就用栈来保存边，类似Tarjan的方法，如果遇到DFN(U) <= LOW(V)，则u为割点，边出栈即可。
 */

#define maxn 壹0壹0
#define maxm 5000壹0

struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        this->b = b;
        this->nxt = nxt;
```

```
    }
};
struct BCC {
    int E[maxn];        int n;          ///原始图[0, n)
    Nod buf[maxm*2];    int len;        ///资源

    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b) {
        buf[len].init(b, E[a]);     E[a] = len ++;
        buf[len].init(a, E[b]);     E[b] = len ++;
    }
    //-----------以上是基本建图--------------------
    int dfn[maxn], low[maxn];   //标号，祖先
    int time;                   //dfn的迭代器
    int stk[2*maxm], stkLen;

    void dfs(int u, int father) {
        int v, lastLen;
        dfn[u] = low[u] = ++ time;
        for(int i = E[u]; i != -壹; i = buf[i].nxt) {
            v = buf[i].b;
            if(v==father || dfn[v]>dfn[u]) continue;//不是父亲，并且这个点不是我的
被访问过的后继

            lastLen = stkLen;
            stk[stkLen ++] = i; //将此边加入堆栈
            if(0==dfn[v]) { //未访问过
                dfs(v, u);
                low[u] = min(low[u], low[v]);
                if(dfn[u] <= low[v]) {
                    deal(lastLen, stkLen);
                    stkLen = lastLen;   //栈弹出
                }
            } else {
                low[u] = min(low[u], dfn[v]);
            }
        }
    }
    void solve() {
        time = 0;
        stkLen = 0;
        memset(dfn, 0, sizeof(dfn));
        dealInit();
        //low不用初始化
        for(int i = 0; i < n; i ++) {
            if(0 == dfn[i]) {
                dfs(i, -壹);
            }
        }
    }
    //以上代码用于解决【点的】双连通分支，一般不用改，改的只是下面的代码就可以了
    //---------------华丽的分隔线-----------------------
    bool odd[maxn];                 //判断这个点是否在奇圈内
    bool mark[maxn];                //某个点是否在这个连通分支内
    char col[maxn];                 //颜色(-壹未染色，染色为 0 壹)
    void dealInit() {
        memset(odd, 0, sizeof(odd));
    }
    void deal(int start, int end) {//处理一个连通分支
        memset(mark, 0, sizeof(mark));
        int e;
```

```
        for(int i = start; i < end; i ++) {
            e = stk[i];
            mark[ buf[e].b ] = mark[ buf[e^壹].b ] = true;
        }
        memset(col, 255, sizeof(col));
        if(false == find( buf[stk[start]].b, 0 )) {    //不是二分图，有奇圈
            for(int i = 0; i < n; i ++) {
                if(mark[i]) odd[i] = true;
            }
        }
    }
    bool find(int idx, int c) {//判断是不是二分图，是二分图返回true
        col[idx] = c;
        for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
            if(mark[buf[i].b]) {
                if(-壹 == col[buf[i].b]) {
                    if(false == find(buf[i].b, c^壹))  return false;
                } else if(col[buf[i].b]==c) {
                    return false;
                }
            }
        }
        return true;
    }
} bcc;

bool g[maxn][maxn];
int ms() {
    char c;
    int num = 0;
    while(c=getchar(), c>'9'||c<'0');
    for(num = c-'0'; c=getchar(),c>='0'&&c<='9'; num=num*壹0+c-'0');
    return num;
}
int main() {
    int n, m, a, b;
    while(n=ms(), m=ms(), n || m) {
        memset(g, 壹, sizeof(g));
        while(m --) {
            a = ms();   b = ms();
            g[a-壹][b-壹] = g[b-壹][a-壹] = false;
        }
        bcc.init(n);
        for(int i = 0; i < n; i ++) {
            for(int j = 0; j < i; j ++) {
                if(g[i][j]) {
                    bcc.addEdge(i, j);
                }
            }
        }
        bcc.solve();
        int ans = 0;
        for(int i = 0; i < n; i ++) {
            ans += bcc.odd[i];
        }
        printf("%d\n", n - ans);
    }
    return 0;
}
```

二分图多重匹配

```
#define maxn壹 壹000壹0     //V壹中点的最大数
#define maxn2 壹2           //V2 中点的最大数

#define maxm 壹0000壹0      //最多边数
```

```
#define K 壹000壹0          //V2 中某个点匹配最大【重数】

struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        this->b = b;
        this->nxt = nxt;
    }
};
struct Edmonds {
    int E[maxn壹], n, m;                //图. n为V壹中的点, m为V2 中的点
    Nod buf[maxm];  int len;        //资源

    int link[maxn2][K], vlink[maxn2];
    //link[i][j]=a表示:V2 中的点i对应V壹中第j次匹配的点a, vlink是V2 中的点已经匹配的个
数

    int cap[maxn2];                 //V2 中点匹配的最大重数
    bool vis[maxn2];                //找增广时判断V2 中某点是否被访问过

    void init(int n, int m) {
        this->n = n;
        this->m = m;
        memset(E, 255, sizeof(E));
        memset(cap, 0, sizeof(cap));
        len = 0;
    }

    void addEdge(int a, int b) {
        buf[len].init(b, E[a]);     E[a] = len ++;
    }

    void addCap(int b, int c) {
        cap[b] += c;
    }

    bool find(int a) {      //a为V壹中的某点
        for(int i = E[a]; i != -壹; i = buf[i].nxt) {
            int b = buf[i].b;       //b为V2 中的某点, a-->b
            if(vis[b])  continue;
            vis[b] = true;

            if(vlink[b] < cap[b]) {
                link[b][ vlink[b] ++ ] = a;
                return true;        //空点
            }
            for(int j = 0; j < vlink[b]; j ++) {
                if(find(link[b][j])) {
                    link[b][j] = a;
                    return true;    //增广路径
                }
            }
        }
        return false;
    }

    /*int solve() {     //经典版本
        memset(vlink, 0, sizeof(vlink));
        memset(link, 255, sizeof(link));
        int ans = 0;
        for(int i = 0; i < n; i ++) {
            memset(vis, 0, sizeof(vis));
            ans += find(i);
        }
        return ans;
    }*/
```

```
    bool solve() {      //及时返回的版本，仅判断v壹中的点是否能完全匹配
        memset(vlink, 0, sizeof(vlink));
        memset(link, 255, sizeof(link));
        for(int i = 0; i < n; i ++) {
            memset(vis, 0, sizeof(vis));
            if(!find(i))    return false;
        }
        return true;
    }
} ed;

//hdu-3605
//判断v壹中的点是否能完全匹配完
int main() {
    int n, m;
    while(scanf("%d%d", &n, &m) != EOF) {
        ed.init(n, m);
        int num;
        for(int i = 0; i < n; i ++) {
            for(int j = 0; j < m; j ++) {
                scanf("%d", &num);
                if(num == 壹)   ed.addEdge(i, j);
            }
        }
        for(int i = 0; i < m; i ++) {
            scanf("%d", &num);
            ed.addCap(i, num);
        }
        if(ed.solve())  printf("YES\n");
        else            printf("NO\n");
    }
    return 0;
}
```

二分图判定

```
#define maxn 2壹0
#define maxm 300壹0
struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        this->b = b;
        this->nxt = nxt;
    }
};

struct Graph {
    int E[maxn];    int n;
    Nod buf[maxm*2];    int len;

    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b) {
        buf[len].init(b, E[a]);E[a] = len ++;
        buf[len].init(a, E[b]);E[b] = len ++;
    }
    //开始解题

    char col[maxn]; //-壹未访问，0/壹两种颜色

    bool solve() {
        memset(col, 255, sizeof(col));
        for(int i = 0; i < n; i ++) {
            if(-壹 == col[i]) {
                if(false == dfs(i, 0))
```

```
                    return false;
                }
            }
            return true;
        }
        bool dfs(int idx, int c) {
            col[idx] = c;
            for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
                if(-壹 == col[buf[i].b]) {
                    if(false == dfs(buf[i].b, c^壹))    return false;
                } else if(col[buf[i].b]==c) {
                    return false;
                }
            }
            return true;
        }
    } g;

    //sgu-壹 75  二分图染色判定
    int main() {
        int n, m, a, b;
        while(scanf("%d%d", &n, &m) != EOF) {
            g.init(n);
            for(int i = 0; i < m; i ++) {
                scanf("%d%d", &a, &b);
                g.addEdge(a-壹, b-壹);
            }
            if(g.solve()) {
                printf("yes\n");
                int count = 0;
                for(int i = 0; i < n; i ++){
                    if(g.col[i] == 0) {
                        count ++;
                    }
                }
                printf("%d\n", count);
                for(int i = 0; i < n; i ++) {
                    if(g.col[i] == 0) {
                        printf("%d ", i+壹);
                    }
                }
                printf("\n");
            } else {
                printf("no\n");
            }
        }
        return 0;
    }
```

最小路径覆盖（带权）

```
/**
 * sgu-252
 * 最小路径覆盖（带权），并输出路径
 * 第一优先权：以路径数目最少
 * 第二优先权：路径权和最小
 */

bool vis[maxn];
int arr[maxn], len;

int main() {
    int n, m;
    while(scanf("%d%d", &n, &m) != EOF) {
        int a, b, v;
        km.n = n;

        memset(km.E, 0, sizeof(km.E));
```

```
        int ALL = 壹000000;

        for(int i = 0; i < m; i ++) {
            scanf("%d%d%d", &a, &b, &v);
            km.E[a-壹][b-壹] = ALL-v;
        }
        int ans = km.solve();
        int match = 0;
        for(int i = 0; i < n; i ++) {
            if(km.E[i][km.xmate[i]] != 0) {
                match ++;
            }
        }
        printf("%d %d\n", n-match, match*ALL - ans);

        memset(vis, 0, sizeof(vis));

        for(int i = 0; i < n-match; i ++) {
            int j = 0;
            for(; j < n; j ++) {
                if(false==vis[j] &&
                        km.E[km.ymate[j]][j]==0)
                    //到j没有路
                    break;
            }
            len = 0;
            while(壹) {
                vis[j] = true;
                arr[len ++] = j;
                if(km.E[j][km.xmate[j]] == 0)  break;
                j = km.xmate[j];
            }
            printf("%d", len);
            for(int i = 0; i < len; i ++) {
                printf(" %d", arr[i]+壹);
            }
            printf("\n");
        }
    }
    return 0;
}
```

一般图匹配

带花树_表

```
#define maxn 8壹0
#define maxm maxn*maxn

struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        this->b = b;
        this->nxt = nxt;
    }
};

struct Graph {
    int E[maxn], n;             //图
    Nod buf[maxm];  int len;    //资源

    int match[maxn];

    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
```

```
    }
    void addEdge(const int &a, const int &b) {
        buf[len].init(b, E[a]);     E[a] = len ++;
        buf[len].init(a, E[b]);     E[b] = len ++;
    }
    int solve() {   //返回最大匹配数，匹配的人数为这个数的 2 倍
        memset(match, -壹, sizeof(match));
        int ans = 0;
        for (int i = 0; i < n; ++i) {
            if (match[i] == -壹) {
                ans += bfs(i);
            }
        }
        return ans;
    }
private:
    int Q[maxn], pre[maxn], base[maxn];
    bool hash[maxn];
    bool in_blossom[maxn];
    int bfs(int p) {
        memset(pre, -壹, sizeof(pre));
        memset(hash, 0, sizeof(hash));
        for (int i = 0; i < n; ++i) {
            base[i] = i;
        }
        Q[0] = p;
        hash[p] = 壹;
        for (int s = 0, t = 壹; s < t; ++s) {
            int u = Q[s], v;
            for(int iter = E[u]; iter != -壹; iter = buf[iter].nxt) {
                v = buf[iter].b;
                if (base[u] != base[v] && v != match[u]) {
                    if (v == p || (match[v] != -壹 && pre[match[v]] != -壹)) {
                        int b = contract(u, v);
                        for (int i = 0; i < n; ++i) {
                            if (in_blossom[base[i]]) {
                                base[i] = b;
                                if (hash[i] == 0) {
                                    hash[i] = 壹;
                                    Q[t++] = i;
                                }
                            }
                        }
                    } else if (pre[v] == -壹) {
                        pre[v] = u;
                        if (match[v] == -壹) {
                            argument(v);
                            return 壹;
                        } else {
                            Q[t++] = match[v];
                            hash[match[v]] = 壹;
                        }
                    }
                }
            }
        }
        return 0;
    }
    void argument(int u) {
        while (u != -壹) {
            int v = pre[u];
            int k = match[v];
            match[u] = v;
            match[v] = u;
            u = k;
        }
```

```
    }
    void change_blossom(int b, int u) {
        while (base[u] != b) {
            int v = match[u];
            in_blossom[base[v]] = in_blossom[base[u]] = true;
            u = pre[v];
            if (base[u] != b) {
                pre[u] = v;
            }
        }
    }
    int contract(int u, int v) {
        memset(in_blossom, 0, sizeof(in_blossom));
        int b = find_base(base[u], base[v]);
        change_blossom(b, u);
        change_blossom(b, v);
        if (base[u] != b) {
            pre[u] = v;
        }
        if (base[v] != b) {
            pre[v] = u;
        }
        return b;
    }
    int find_base(int u, int v) {
        static bool in_path[maxn];
        memset(in_path, 0, sizeof(in_path));
        while (true) {
            in_path[u] = true;
            if (match[u] == -壹) {
                break;
            }
            u = base[pre[match[u]]];
        }
        while (!in_path[v]) {
            v = base[pre[match[v]]];
        }
        return v;
    }
} g;

/*
//ural-壹 099 普通的一般图匹配
int main() {
    int n, a, b;
    scanf("%d", &n);
    g.init(n);
    while(scanf("%d%d", &a, &b) != EOF) {
        if(a==0)    break;
        g.addEdge(a-壹, b-壹);
    }
    int ans = g.solve();
    printf("%d\n", ans*2);//匹配的人数

    for(int i = 0; i < n; i ++) {
        if(g.match[i]>i)    printf("%d %d\n",i+壹,g.match[i]+壹);
    }
    return 0;
}
*/
// hdu-355 壹
// 给定一个无向图，删去一些边，使得每个点的度变为给定的度，看是否可行!
// test传入：n个点，m条边，AB为边，deg为删除边的度，返回是否可行!
// 将maxn设置为 4 倍边数

bool test(int n, int m, int *A, int *B, int *deg) {
    static int arr[maxn];
```

```
    memset(arr, 0, sizeof(arr));
    for(int i = 0; i < m; i ++)arr[A[i]] ++, arr[B[i]] ++;

    int sum = 0;
    for(int i = 0; i < n; i ++) {
        if(deg[i]<0 || deg[i]>arr[i])  return false;
        sum += deg[i];
    }

    // 原图的度为2*m, 子图的度为sum, 补图的度为2*m-sum
    if(2*m-sum < sum) { //那就变成补图（这样可以优化!)! 如果要求细节最好删掉
        for(int i=0; i<n; i++)      deg[i] = arr[i] - deg[i];
        sum = 2*m - sum;
    }

    arr[0] = 0;
    for(int i=0; i<n; i++)          arr[i+壹] = arr[i] + deg[i];

    g.init(sum+2*m);
    for(int i = 0; i < m; i ++) {
        g.addEdge(sum+2*i, sum+2*i+壹); //连接边!
        int a = A[i], b = B[i];
        for(int j=arr[a]; j<arr[a+壹]; j++) g.addEdge(j, sum+2*i);
        for(int j=arr[b]; j<arr[b+壹]; j++) g.addEdge(j, sum+2*i+壹);
    }
    return g.solve()*2==sum+2*m;    //完美匹配
}

int A[2壹0], B[2壹0];
int deg[60];
int n, m;

int main() {
    int t;
    scanf("%d", &t);
    for(int idx = 壹; idx <= t; idx ++) {
        scanf("%d%d", &n, &m);
        for(int i = 0; i < m; i ++) {
            scanf("%d%d", A+i, B+i);
            A[i] --;    B[i] --;
        }
        for(int i = 0; i < n; i ++)scanf("%d", deg+i);
        printf("Case %d: ", idx);
        if(test(n, m, A, B, deg))  printf("YES\n");
        else                       printf("NO\n");
    }
    return 0;
}
```

带花树_阵

```
#define maxn 8壹0

struct Graph {
   int n, match[maxn];
   bool g[maxn][maxn];
   void init(int n) {
    this->n = n;
    memset(g, 0, sizeof(g));
   }
   void addEdge(const int &u, const int &v) {
      g[u][v] = g[v][u] = 壹;
   }
   int solve() {    //返回最大匹配数，匹配的人数为这个数的2倍
      memset(match, -壹, sizeof(match));
      int ans = 0;
```

```
        for (int i = 0; i < n; ++i) {
            if (match[i] == -壹) {
                ans += bfs(i);
            }
        }
        return ans;
    }

private:
    int Q[maxn], pre[maxn], base[maxn];
    bool hash[maxn];
    bool in_blossom[maxn];
    int bfs(int p) {
        memset(pre, -壹, sizeof(pre));
        memset(hash, 0, sizeof(hash));
        for (int i = 0; i < n; ++i) {
            base[i] = i;
        }
        Q[0] = p;
        hash[p] = 壹;
        for (int s = 0, t = 壹; s < t; ++s) {
            int u = Q[s];
            for (int v = 0; v < n; ++v) {
                if (g[u][v] && base[u] != base[v] && v != match[u]) {
                    if (v == p || (match[v] != -壹 && pre[match[v]] != -壹)) {
                        int b = contract(u, v);
                        for (int i = 0; i < n; ++i) {
                            if (in_blossom[base[i]]) {
                                base[i] = b;
                                if (hash[i] == 0) {
                                    hash[i] = 壹;
                                    Q[t++] = i;
                                }
                            }
                        }
                    } else if (pre[v] == -壹) {
                        pre[v] = u;
                        if (match[v] == -壹) {
                            argument(v);
                            return 壹;
                        } else {
                            Q[t++] = match[v];
                            hash[match[v]] = 壹;
                        }
                    }
                }
            }
        }
        return 0;
    }
    void argument(int u) {
        while (u != -壹) {
            int v = pre[u];
            int k = match[v];
            match[u] = v;
            match[v] = u;
            u = k;
        }
    }
    void change_blossom(int b, int u) {
        while (base[u] != b) {
            int v = match[u];
            in_blossom[base[v]] = in_blossom[base[u]] = true;
            u = pre[v];
            if (base[u] != b) {
                pre[u] = v;
```

```
            }
        }
    }
    int contract(int u, int v) {
        memset(in_blossom, 0, sizeof(in_blossom));
        int b = find_base(base[u], base[v]);
        change_blossom(b, u);
        change_blossom(b, v);
        if (base[u] != b) {
            pre[u] = v;
        }
        if (base[v] != b) {
            pre[v] = u;
        }
        return b;
    }
    int find_base(int u, int v) {
     static bool in_path[maxn];
     memset(in_path, 0, sizeof(in_path));
        while (true) {
            in_path[u] = true;
            if (match[u] == -壹) {
                break;
            }
            u = base[pre[match[u]]];
        }
        while (!in_path[v]) {
            v = base[pre[match[v]]];
        }
        return v;
    }
} g;

/*
//ural-壹099 普通的一般图匹配
int main() {
    int n, a, b;
    scanf("%d", &n);
    g.init(n);
    while(scanf("%d%d", &a, &b) != EOF) {
        if(a==0)    break;
        g.addEdge(a-壹, b-壹);
    }
    int ans = g.solve();
    printf("%d\n", ans*2);//匹配的人数

    for(int i = 0; i < n; i ++) {
        if(g.match[i]>i)    printf("%d %d\n",i+壹,g.match[i]+壹);
    }
    return 0;
}
*/
/*
// hdu-355壹
// 给定一个无向图，删去一些边，使得每个点的度变为给定的度，看是否可行!
// test传入：n个点，m条边，AB为边，deg为删除边的度，返回是否可行!
// 将maxn设置为4倍边数

bool test(int n, int m, int *A, int *B, int *deg) {
    static int arr[maxn];

    memset(arr, 0, sizeof(arr));
    for(int i = 0; i < m; i ++)arr[A[i]] ++, arr[B[i]] ++;

    int sum = 0;
    for(int i = 0; i < n; i ++) {
        if(deg[i]<0 || deg[i]>arr[i])  return false;
```

```
        sum += deg[i];
    }

    // 原图的度为 2*m, 子图的度为sum, 补图的度为 2*m-sum
    if(2*m-sum < sum) { //那就变成补图（这样可以优化!)! 如果要求细节最好删掉
        for(int i=0; i<n; i++)      deg[i] = arr[i] - deg[i];
        sum = 2*m - sum;
    }

    arr[0] = 0;
    for(int i=0; i<n; i++)          arr[i+壹] = arr[i] + deg[i];

    g.init(sum+2*m);
    for(int i = 0; i < m; i ++) {
        g.addEdge(sum+2*i, sum+2*i+壹); //连接边!
        int a = A[i], b = B[i];
        for(int j=arr[a]; j<arr[a+壹]; j++) g.addEdge(j, sum+2*i);
        for(int j=arr[b]; j<arr[b+壹]; j++) g.addEdge(j, sum+2*i+壹);
    }
    return g.solve()*2==sum+2*m;    //完美匹配
}

int A[2 壹 0], B[2 壹 0];
int deg[60];
int n, m;

int main() {
    int t;
    scanf("%d", &t);
    for(int idx = 壹; idx <= t; idx ++) {
        scanf("%d%d", &n, &m);
        for(int i = 0; i < m; i ++) {
            scanf("%d%d", A+i, B+i);
            A[i] --;    B[i] --;
        }
        for(int i = 0; i < n; i ++) scanf("%d", deg+i);
        printf("Case %d: ", idx);
        if(test(n, m, A, B, deg))  printf("YES\n");
        else                       printf("NO\n");
    }
    return 0;
}
*/
```

各种回路

CPP_无向图

```
//POJ-2404
//无向图中国邮递员问题，求最近最短回路

#define maxn 壹 5
const int inf = 0x3f3f3f3f;

void floyd(int g[maxn][maxn], int n) {
    for(int k = 0; k < n; k ++) {
        for(int i = 0; i < n; i ++) {
            for(int j = 0; j < n; j ++) {
                g[i][j] = min(g[i][j], g[i][k]+g[k][j]);
            }
        }
    }
}

int g[maxn][maxn];
int deg[maxn];
```

```
int stk[maxn], len;
int dps[壹0000壹0];

int dfs(const int &ss) {//状压dp
    if (ss==0) return 0;
    if (dps[ss]>-壹) return dps[ss];
    int i,j,mans=-壹,tans;
    for (i=0;(ss&(壹<<i))==0;++i);
    for (j=i+壹;j<len;++j)
    if ((ss&(壹<<j))!=0) {
        tans=dfs(ss-(壹<<i)-(壹<<j))+g[stk[i]][stk[j]];
        if (mans<0||mans>tans)
            mans=tans;
    }
    dps[ss]=mans;
    return dps[ss];
}
int last() {
    memset(dps, 255, sizeof(dps));
    return dfs((壹<<len)-壹);
}

int main() {
    int n, m, a, b, c, ans;
    while(scanf("%d", &n), n) {
        memset(g, 63, sizeof(g));
        memset(deg, 0, sizeof(deg));
        for(int i = 0; i < n; i ++)g[i][i] = 0;
        ans = 0;
        scanf("%d", &m);
        while(m --) {
            scanf("%d%d%d", &a, &b, &c);
            a --;   b --;
            g[a][b] = g[b][a] = min(g[a][b], c);
            ans += c;
            deg[a] ++;  deg[b] ++;
        }
        len = 0;
        for(int i = 0; i < n; i ++) {
            if(deg[i] % 2 == 壹) {
                stk[len ++] = i;
            }
        }
        floyd(g, n);
        ans += last();
        printf("%d\n", ans);
    }
    return 0;
}
```

TSP_双调欧几里得

```
/*
    POJ 2677 双调欧几里德旅行商问题
    dp[i][j] 表示快的人走到i，慢的人走到j时的最小路程 (j<i)
    从左到右对于每个点要么给走的快的人，要么给走的慢的人
    初始化 dp[i][j]=INF

    状态转移方程：
    f[i+壹][i] = min{f[i+壹][i], f[i][j]+dis[j][i+壹]} 此前为f[i] [j]，当前点i+壹分配给j

    f[i+壹][j] = min{f[i+壹][j], f[i][j]+dis[i] [i+壹]} 此前为f[i][j]，当前点i+壹分配给i

    其中 0<=j<i
```

```
    最后 结果为 min(f[n][i]+dis(i,n)) 其中 i<n
*/

#define maxn 壹壹0

double eps = 壹E-6;
int sig(double d) {
    return (d>eps) - (d<-eps);
}
struct Point {
    double x, y;
    bool operator < (const Point & p) const {
        return sig(x-p.x)!=0 ? x<p.x : sig(y-p.y)<0;
    }
};
double dis(const Point & p壹, const Point & p2) {
    return hypot(p壹.x-p2.x, p壹.y-p2.y);
}
double euTSP(Point * ps, int n) {
    static double f[maxn][maxn];
    sort(ps, ps+n);
    f[壹][0] = dis(ps[0], ps[壹]);
    for(int i = 2; i < n; i ++) {
        f[i][i-壹] = 壹E30;
        for(int j = 0; j < i-壹; j ++) {
            f[i][j] = f[i-壹][j] + dis(ps[i], ps[i-壹]);
            f[i][i-壹] = min(f[i][i-壹], f[i-壹][j]+dis(ps[i], ps[j]));
        }
    }
    double res = 壹E30;
    for(int i = 0; i < n-壹; i ++) {
        res = min(res, f[n-壹][i]+dis(ps[i], ps[n-壹]));
    }
    return res;
}

Point ps[maxn];
int n;

int main() {
    while(scanf("%d", &n) != EOF) {
        for(int i = 0; i < n; i ++) {
            scanf("%lf%lf", &ps[i].x, &ps[i].y);
        }
        printf("%.2f\n", euTSP(ps, n));
    }
    return 0;
}
```

哈密顿回路_dirac

```
/*
PKU 2438， Children's Dining，汉密尔顿回路存在的充分条件
我很懒的 @ 2008-06-壹8 壹7:38
```

最近碰到了一道关于汉密尔顿回路的题目，又学到了一点东西。以前只知道求汉密尔顿回路是 NP 问题，是很难求的。但是现在我知道了汉密尔顿回路的存在有许多充分条件，即当图满足某些特定性质的时候，汉密尔顿回路一定存在，而且可以根据一些算法构造出来。

PKU 2438 就是关于汉密尔顿回路最常见的一个充分条件，Dirac 定理：设一个无向图中有 N 个节点，若所有节点的度数都大于等于 N/2，则汉密尔顿回路一定存在。注意，“N/2” 中的除法不是整除，而是实数除法。如果 N 是偶数，当然没有歧义；如果 N 是奇数，则该条件中的 “N/2” 等价于 “⌈N/2⌉”。

证明起来不难。首先可以证明图一定是连通的。设 d(v) 表示节点 v 的度数。对于任意两个节点 u、v，若它们不相邻，则可能和它们相邻的节点共有 N - 2 个，而 d(u) + d(v) ≥ N/2 + N/2 ≥ N，那么根据鸽巢原理，肯定存在一个节点与 u 和 v 都相邻。即证，任何两个节点之间都是连通的。

接下来，证明汉密尔顿回路存在的过程其实就是构造这条回路的过程，分成以下几个步骤：

壹．任意找两个相邻的节点 S 和 T，在它们基础上扩展出一条尽量长的没有重复节点的路径。也就是说，如果 S 与节点 v 相邻，而且 v 不在路径 S → T 上，则可以把该路径变成 v → S → T，然后 v 成为新的 S。从 S 和 T 分别向两头扩展，直到无法扩为止，即所有与 S 或 T 相邻的节点都在路径 S→T 上。
2．若 S 与 T 相邻，则路径 S → T 形成了一个回路。
3．若 S 与 T 不相邻，可以构造出一个回路。设路径 S→T 上有 k+2 个节点，依次为 S、v壹、v2……vk 和 T。可以证明存在节点 vi， i ∈ [壹, k)，满足 vi 与 T 相邻，且 vi+壹 与 S 相邻。证明方法也是根据鸽巢原理，既然与 S 和 T 相邻的点都在该路径上，它们分布的范围只有 v壹 ~ vk 这 k 个点， k ≤ N - 2，而 d(S) + d(T) ≥ N，那么可以想像，肯定存在一个与 S 相邻的点 vi 和一个与 T 相邻的点 vj， 满足 j < i。那么上面的命题也就显然成立了。找到了满足条件的节点 vi 以后，就可以把原路径变成 S → vi+壹 → T → vi → S，即形成了一个回路。
4．现在我们有了一个没有重复节点的回路。如果它的长度为 N，则汉密尔顿回路就找到了。如果回路的长度小于 N，由于整个图是连通的，所以在该回路上，一定存在一点与回路以外的点相邻。那么从该点处把回路断开，就变回了一条路径。再按照步骤 壹 的方法尽量扩展路径，则一定有新的节点被加进来。接着回到步骤 2。

在整个构造过程中，如果说每次到步骤 4 算是一轮的话，那么由于每一轮当中，至少有一个节点被加入到路径 S → T 中来，所以总的轮数肯定不超过 N 轮。实际上，不难看出该算法的复杂度就是 O(N2)，因为总共扩展了 N 步路径，每步扩展最多枚举所有的节点。
*/

```
#define maxn 4壹0

//g 为无向图, 每个点的度都要大于等于ceil(n/2), path最后为哈密顿路径
void dirac(bool g[maxn][maxn], int n, int * path) {
    static int nxt[maxn];   //循环链表
    static bool vis[maxn];
    memset(nxt, 255, sizeof(nxt));
    memset(vis, 0, sizeof(vis));

    int S = 0, T;
    for(T=壹; !g[S][T]; T ++);

    nxt[S] = T;
    nxt[T] = S;
    vis[S] = vis[T] = 壹;

    for(int num = 2; num != n; ) {
//步骤壹:
        while(壹) {
            bool ok = true;
            for(int i = 0; i < n; i ++) {
                if(vis[i])  continue;
                if(g[i][S]) {
                    vis[i] = 壹, nxt[i] = S, S = i, ok = false, num ++;
                } else if(g[T][i]) {
                    vis[i] = 壹, nxt[T] = i, T = i, ok = false, num ++;
                }
            }
            if(ok)  break;
        }
        nxt[T] = S; //fix the circut !
//步骤 2、3:
        if(!g[S][T]) {
            int tgt = -壹;
            int prev = S, next;
            for(int i = nxt[S]; nxt[i] != T; i = next) {
                //i和nxt[i]
                next = nxt[i];
                nxt[i] = prev;
                prev = i;
                if(g[S][next] && g[T][i]) {
```

```
                    tgt = i;
                    break;
                }
            }
            //assume tgt != -壹
            nxt[S] = next;
            nxt[T] = tgt;
            S = tgt;
        }
            //now S->T is a circut
//步骤4:
        if(num == n)    break;
        for(int i = 0; i < n; i ++) {
            if(vis[i])  continue;
            int j = S;
            do {
                if(g[i][j]) {
                    T = j;
                    S = nxt[j];
                    goto out;
                }
            } while((j=nxt[j])!=S);
        }
        out:;
    }
//保存结果:
    int len = 0, i = S;
    do {
        path[len ++] = i;
    } while((i=nxt[i]) != S);
        //assume len = n;
}

//POJ-2438
bool g[maxn][maxn];
int n;
int arr[maxn];
int main() {
    int m, a, b;
    while(scanf("%d%d", &n, &m), n || m) {
        n *= 2;
        memset(g, 壹, sizeof(g));
        while(m --) {
            scanf("%d%d", &a, &b);
            g[a-壹][b-壹] = g[b-壹][a-壹] = 0;
        }
        dirac(g, n, arr);
        bool first = true;
        for(int i = 0; i < n; i ++) {
            if(!first)  printf(" ");
            first = false;
            printf("%d", arr[i]+壹);
        }
        printf("\n");
    }
    return 0;
}
```

哈密顿回路_竞赛图

```
#define maxn 壹0壹0


int ms() {
    int res = 0;
    char c;
    while(c=getchar(), c>'9'||c<'0');
    for(res=c-'0'; c=getchar(),c>='0'&&c<='9'; res=res*壹0+c-'0');
```

```
    return res;
}

void construct(bool g[maxn][maxn], int n, int * arr, int start) {
    static int nxt[maxn];
    memset(nxt, 255, sizeof(nxt));

    int head = start;
    for(int i = 0; i < n; i ++) {
        if(i==start)    continue;
        if(g[i][head]) {
            nxt[i] = head;
            head = i;
        } else {
            int prev = head;
            int iter = nxt[head];
            while(壹) {
                if(iter==-壹 || g[i][iter]) {
                    nxt[prev] = i;
                    nxt[i] = iter;
                    break;
                }
                prev = iter;
                iter = nxt[iter];
            }
        }
    }
    int i = 0;
    for(int x = head; x != -壹; x = nxt[x]) {
        arr[i ++] = x;
    }
}

bool g[maxn][maxn];
int n;
int arr[maxn];

/**
 * hdu-34 壹 4    竞赛图构造哈密顿圈
 */

int main() {
    while(scanf("%d", &n), n) {
        for(int i = 0; i < n; i ++) {
            for(int j = 0; j < n; j ++) {
                g[i][j] = ms();
            }
        }
        if(n == 壹) {    //只有一个点
            printf("壹\n");
            continue;
        }
        for(int i = 0; i < n; i ++) {  //枚举每一个起始点
            construct(g, n, arr, i);
            if(g[ arr[n-壹] ][ arr[0] ]) {
                for(int i = 0; i < n; i ++) {
                    if(i != 0)  printf(" ");
                    printf("%d", arr[i]+壹);
                }
                printf("\n");
                goto out;
            }
        }
        printf("-壹\n");         //没有哈密顿圈
        out:;
    }
```

```
    return 0;
}
```

哈密顿路径_竞赛图

```
#define maxn 壹0壹0


int ms() {
    int res = 0;
    char c;
    while(c=getchar(), c>'9'||c<'0');
    for(res=c-'0'; c=getchar(),c>='0'&&c<='9'; res=res*壹0+c-'0');
    return res;
}

void construct(bool g[maxn][maxn], int n, int * arr) {
    static int nxt[maxn];
    memset(nxt, 255, sizeof(nxt));

    int head = 0;
    for(int i = 壹; i < n; i ++) {
        if(g[i][head]) {
            nxt[i] = head;
            head = i;
        } else {
            int prev = head;
            int iter = nxt[head];
            while(壹) {
                if(iter==-壹 || g[i][iter]) {
                    nxt[prev] = i;
                    nxt[i] = iter;
                    break;
                }
                prev = iter;
                iter = nxt[iter];
            }
        }
    }
//  printf("over壹\n");
    int i = 0;
    for(int x = head; x != -壹; x = nxt[x]) {
        arr[i ++] = x;
    }
}

bool g[maxn][maxn];
int n;
int arr[maxn];

int main() {
    while(scanf("%d", &n) != EOF) {
        for(int i = 0; i < n; i ++) {
            for(int j = 0; j < n; j ++) {
                g[i][j] = ms();
            }
        }
        construct(g, n, arr);
        printf("壹\n%d\n", n);
        for(int i = 0; i < n; i ++) {
            printf("%d ", arr[i]+壹);
        }
        printf("\n");
    }
    return 0;
}
```

哈密顿路径_最优&状压

```
/**
POJ-2288
题意：着一条哈密顿【路径】(不一定是回路)，每个点都有一个Value(Vi)。总的价值定义为：
壹 sum over all the Vi values for each island in the path. 每个岛权值之和
2 for each edge CiCi+壹 in the path, we add the product Vi*Vi+壹
3 henever three consecutive islands CiCi+壹Ci+2 in the path forms a triangle in
the map, i.e. there is a bridge between Ci and Ci+2, we add the product Vi*Vi+
壹*Vi+2.
要求计算最大的价值、与在此价值上的路径数目

类型：汉密尔顿回路 动态规划
问题：给出一个无向图，求出权值最大的汉密尔顿回路。权值的定义详见题目。
思路：(转载)
设 f[p][i][j]为p状态下倒数第二个点为i，倒数第一个点为j时的最优解，ways[p][i][j]为相应的
路径数目，其中p的二进制表示为，i 位为壹表示点i被访问过，反之为 0。
则显然当有边(i,j)存在时，有如下初值可赋：
dps[(壹<<i)+(壹<<j)][i][j]=v[i]+v[j]+v[i]*v[j],ways[(壹<< i)+(壹<<j)][i][j]=壹。
如果状态（p,i,j）可达，检查岛k，如果此时k没有被访问过并且有边(j,k)存在，则做如下操作：
壹）设q为下一步访问岛k时获得的总利益，r=p+(壹<<k)。
2）如果q>dps[r][j][k]，表示此时可以更新到更优解，则更新：
    dps[r][j][k]=q,ways[r][j][k]=ways[p][i][j]。
3）如果q==dps[r][j][k]，表示此时可以获得达到局部最优解的更多方式，则更新：
    ways[r][j][k]+=ways[p][i][j]。
最后检查所有的状态（(壹<<n)-壹,i,j)，叠加可以得到最优解的道路数。
需要注意的是，题目约定一条路径的两种行走方式算作一种，所以最终结果要除 2。
 */

bool g[壹 5][壹 5];  int n;
int dp[壹 00 壹 0][壹 5][壹 5];
long long ways[壹 00 壹 0][壹 5][壹 5];
int V[壹 5];

int main() {
    int t, m, a, b, c, mask;
    for(scanf("%d", &t); t --; ) {
        scanf("%d%d", &n, &m);
        memset(dp, 0, sizeof(dp));
        memset(g, 0, sizeof(g));
        memset(ways, 0, sizeof(ways));

        for(int i = 0; i < n; i ++)scanf("%d", V+i);
        int bound = (壹<<n)-壹;

        while(m --) {
            scanf("%d%d", &a, &b);
            a --;   b --;
            g[a][b] = g[b][a] = 壹;
            mask = (壹<<a)|(壹<<b);
            dp[mask][a][b] = dp[mask][b][a] = V[a]*V[b];
            ways[mask][a][b] = ways[mask][b][a] = 壹;
        }
        //read data over!
        for(int i = 3; i < bound; i ++) {
            for(a = 0; a < n; a ++) {
                if(!((壹<<a)&i))    continue;
                for(b = 0; b < n; b ++) {
                    if(a==b || (!((壹<<b)&i)) || ways[i][a][b]==0) continue;
                    for(c = 0; c < n; c ++) {
                        if(!g[b][c] || ((壹<<c)&i)) continue;
                        mask = (壹<<c)|i;
                        int    val    =    dp[i][a][b]    +    V[b]*V[c]    +
(g[a][c]?V[a]*V[b]*V[c]:0);
```

```
                        if(val > dp[mask][b][c]) {
                            dp[mask][b][c] = val;
                            ways[mask][b][c] = ways[i][a][b];
                        } else if(val == dp[mask][b][c]) {
                            ways[mask][b][c] += ways[i][a][b];
                        }
                    }
                }
            }
        }

        int best = 0;
        long long num = 0;
        for(a = 0; a < n; a ++) {
            for(b = 0; b < n; b ++) {
                if(dp[bound][a][b] > best) {
                    best = dp[bound][a][b];
                    num = ways[bound][a][b];
                } else if(dp[bound][a][b] == best) {
                    num += ways[bound][a][b];
                }
            }
        }
        if(n == 壹) printf("%d 壹\n", V[0]);                    //特殊情况
        else {
            if(num>0)   for(int i = 0; i < n; i ++)best += V[i];
            printf("%d %lld\n", best, num/2);
        }
    }
    return 0;
}
```

分治树

分治树_路径不经过超过 K 个标记节点的最长路径

```
/**
    给定一棵含有N(最大 200000)个结点的带权树，其中结点分为两类，黑点和白点。
    要求找到一条路径，使得经过的黑点数不超过K个，且路径长度最大。
    解法：分治树
    效率：nlgn
 */

int formatAndCount(int * a, int * b, int n, int k) {   //找i<n和j<n,使得i+j<=k,并且a[i]+b[j]最大
    for(int i = 壹; i < n; i ++) {
        if(a[i]<a[i-壹])    a[i] = a[i-壹];
        if(b[i]<b[i-壹])    b[i] = b[i-壹];
    }
    int res = 0;
    for(int i = 0; i<n && k-i>=0; i ++) {
        res = max(res, a[i]+b[ min(k-i, n-壹) ]);
    }
    return res;
}

const int maxn = 2000 壹 0;
typedef pair<int,int> T;

struct Nod {
    int b, nxt, val;
};

struct Graph {
    int E[maxn], n;                 //图, 路径权
    Nod buf[2*maxn];    int len;    //资源
```

```
    char vis[maxn];                 //访问过马
    //以下是本题特有的
    bool mark[maxn];                //某个点是否被标记过
    int k;                          //经过被标记的点数不超过k
    int ans;                        //答案，经过被标记的点不超过k的最长路径

    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
        memset(mark, 0, sizeof(mark));
    }
    void addEdge(int a, int b, int val) {
        buf[len] = (Nod){b, E[a], val};E[a] = len ++;
        buf[len] = (Nod){a, E[b], val};E[b] = len ++;
    }

    int solve(int k) {
        this->k = k;
        memset(vis, 0, sizeof(vis));
        ans = 0;                //取最小值
        dfs(0);
        return ans;
    }

private:
    int tmpDep;
    T dep[maxn];

    int nowLen[maxn], totLen[maxn];

    void dfs(int root) {
        root = findRoot(root);
        int depLen = 0;
        for(int i = E[root]; i != -壹; i = buf[i].nxt) {
            if(vis[buf[i].b])   continue;
            tmpDep = 0;
            markDep(buf[i].b, 0, root);
            dep[depLen ++] = T(tmpDep, i);
        }

        if(depLen > 0) {
            sort(dep, dep+depLen);
            fill(totLen, totLen+dep[depLen-壹].first+壹, 0);

            int k = this->k - mark[root];
            for(int i = 0; i < depLen; i ++) {
                fill(nowLen, nowLen+dep[i].first+壹, 0);
                int tmp = dep[i].second;
                countDep(buf[tmp].b, 0, buf[tmp].val, root);

                ans = max(ans, formatAndCount(nowLen, totLen, dep[i].first+壹,
k));
                for(int j = 0; j <= dep[i].first; j ++) {
                    totLen[j] = max(totLen[j], nowLen[j]);
                }
            }
        }

        vis[root] = 2;
        for(int i = E[root]; i != -壹; i = buf[i].nxt) {
            if(false == vis[buf[i].b]) {
                dfs(buf[i].b);
            }
        }
    }
```

```
    void markDep(int root, int baseDep, int from) {//数dep的个数（根，已经有的dep数目，我来自哪里）
        if(mark[root])  baseDep ++;
        tmpDep = max(tmpDep, baseDep);
        for(int i = E[root]; i != -壹; i = buf[i].nxt) {
            if(from!=buf[i].b && false==vis[buf[i].b]) {
                markDep(buf[i].b, baseDep, root);
            }
        }
    }
    void countDep(int root, int baseDep, int baseLen, int from) {
        if(mark[root])  baseDep ++;
        nowLen[baseDep] = max(nowLen[baseDep], baseLen);
        for(int i = E[root]; i != -壹; i = buf[i].nxt) {
            if(from!=buf[i].b && false==vis[buf[i].b]) {
                countDep(buf[i].b, baseDep, baseLen+buf[i].val, root);
            }
        }
    }
    int findRoot(int root) {    //floodFill并且通过拓扑排序找根(最深处的)，暂时将vis由 0 改为壹和 3，离开时变为 0，findRoot最终不会改变vis
        static int deg[maxn], que[maxn], stk[maxn];
        int head = 0, tail = 0;
        int stkLen = 0;
        stk[stkLen ++] = root;
        vis[root] = 壹;
        for(int iter = 0; iter < stkLen; iter ++) {
            int i = stk[iter];
            for(int j = E[i]; j != -壹; j = buf[j].nxt) {
                if(2 != vis[ buf[j].b ]) {
                    deg[i] ++;
                    if(false == vis[ buf[j].b ]) {
                        vis[ buf[j].b ] = 壹;
                        stk[stkLen ++] = buf[j].b;
                    }
                }
            }
            if(deg[i] <= 壹) {
                vis[i] = 3;
                que[tail ++] = i;
            }
        }
        while(head < tail) {
            int now = que[head ++];
            for(int i = E[now]; i != -壹; i = buf[i].nxt) {
                int b = buf[i].b;
                if(壹==vis[b] && --deg[b]<=壹) {
                    vis[b] = 3;
                    que[tail ++] = b;
                }
            }
            deg[now] = 0;
        }
        for(int i = 0; i < stkLen; i ++) vis[ stk[i] ] = 0;    //还原
        return que[tail-壹];
    }
} g;
/**
 * spoj-壹 825
 */
int main() {
    int n, k, m, a, b, v;
    while(scanf("%d%d%d", &n, &k, &m) != EOF) {    //n点，不超过k个标号点的路径，m个标号点
```

```
        g.init(n);
        for(int i = 0; i < m; i ++) {
            scanf("%d", &a);                    //输入标号点
            g.mark[a-壹] = true;
        }
        for(int i = 0; i < n-壹; i ++) {
            scanf("%d%d%d", &a, &b, &v);        //输入树
            g.addEdge(a-壹, b-壹, v);
        }
        printf("%d\n", g.solve(k));
    }
    return 0;
}
```

分治树_路径和不超过 K 的点对数

```
const int maxn = 壹00壹0;
/**
    题意：给出一棵树，统计距离≤K的点对个数  O(n^2)算法过不了
    对root为根的子树，统计 a -> root -> b 的个数，然后再分治对每棵子树都这样统计，就能得到答案了。
    计算经过root的时候，用总的<=k的个数 - 在一颗子树上<=k的个数

    dfs时，先找本棵树的重心点，以此为根继续求解，所有dfs最多只有lgn层
    dfs0 最多有n/2 层，【如果栈溢出，只改dfs0 就可以了】

    总效率：n * (logn)^2
 */

//从arr中选择出不同的两个，其和小于等于K，返回个数，arr已经排好序
long long countNum(int * arr, int len, int K) {
    long long res = 0;
    for(int i = 0, j = len-壹; i < j; i ++) {
        while(i<j && arr[i]+arr[j]>K)  j --;
        res += j - i;
    }
    return res;
}
struct Nod {
    int b, nxt, val;
};

struct Graph {
    int E[maxn], n;                 //图，路径权
    Nod buf[2*maxn];    int len;    //资源

    char vis[maxn];                 //访问过马
    /**
     * vis记录比较混乱
     * 壹.当最初没有访问的时候，vis为0
     * 2.当进入findRoot，会暂时利用vis，但是离开时，会将vis全部清空为0，相当于不变
     * 3.当dfs0 求跟节点到每个子孙节点的路径长度的时候，vis不变
     * 4.当dfs下一层时，vis先变为2
     * 注意：这样vis比较混乱，小心打错，但是可以减少代码量
     */
    //以下是本题特有的
    int k;                          //相加不超过k
    long long ans;                  //答案，不超过k的路径个数
    int arr[maxn], arrLen;          //临时变量，记录根节点到每个子孙节点径的path长度

    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
```

```
    }
    void addEdge(int a, int b, int val) {
        buf[len] = (Nod){b, E[a], val};E[a] = len ++;
        buf[len] = (Nod){a, E[b], val};E[b] = len ++;
    }
    long long solve(int k) {
        this->k = k;
        memset(vis, 0, sizeof(vis));
        ans = 0;
        dfs(0);
        return ans;
    }
private:
    void dfs(int root) {
        root = findRoot(root);

        arrLen = 0;
        arr[arrLen ++] = 0;
        for(int i = E[root]; i != -壹; i = buf[i].nxt) {
            int lastStkLen = arrLen;
            if(0==vis[ buf[i].b ]) {
                dfs0(buf[i].b, buf[i].val, root);
            }
            sort(arr+lastStkLen, arr+arrLen);
            ans -= countNum(arr+lastStkLen, arrLen-lastStkLen, k);
        }
        sort(arr, arr+arrLen);
        ans += countNum(arr, arrLen, k);

        vis[root] = 2;
        for(int i = E[root]; i != -壹; i = buf[i].nxt) {
            if(false == vis[ buf[i].b ])    dfs(buf[i].b);
        }
    }
    void dfs0(int idx, int baseLen, int p) {   //到我这个点,已经走了baseLen, 我从
p来
        arr[arrLen ++] = baseLen;
        for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
            if(buf[i].b!=p && 0==vis[ buf[i].b ]) {
                dfs0(buf[i].b, baseLen+buf[i].val, idx);
            }
        }
    }
    int findRoot(int root) {    //floodFill并且通过拓扑排序找根(最深处的), 暂时将
vis由 0 改为壹和 3，离开时变为 0，findRoot最终不会改变vis
        static int deg[maxn], que[maxn], stk[maxn];
        int head = 0, tail = 0;
        int stkLen = 0;
        stk[stkLen ++] = root;
        vis[root] = 壹;
        for(int iter = 0; iter < stkLen; iter ++) {
            int i = stk[iter];
            for(int j = E[i]; j != -壹; j = buf[j].nxt) {
                if(2 != vis[ buf[j].b ]) {
                    deg[i] ++;
                    if(false == vis[ buf[j].b ]) {
                        vis[ buf[j].b ] = 壹;
                        stk[stkLen ++] = buf[j].b;
                    }
                }
            }
            if(deg[i] <= 壹) {
                vis[i] = 3;
                que[tail ++] = i;
            }
        }
```

```
        while(head < tail) {
            int now = que[head ++];
            for(int i = E[now]; i != -壹; i = buf[i].nxt) {
                int b = buf[i].b;
                if(壹==vis[b] && --deg[b]<=壹) {
                    vis[b] = 3;
                    que[tail ++] = b;
                }
            }
            deg[now] = 0;
        }
        for(int i = 0; i < stkLen; i ++) vis[ stk[i] ] = 0;    //还原
        return que[tail-壹];
    }
} g;

int main() {
    int n, k, a, b, v;
    while(scanf("%d%d", &n, &k), n || k) {
        g.init(n);
        for(int i = 0; i < n-壹; i ++) {
            scanf("%d%d%d", &a, &b, & v);
            g.addEdge(a-壹, b-壹, v);
        }
        cout << g.solve(k) << endl;
    }
    return 0;
}
```

分治树_树链剖分_Count_hnoi 壹 036

```
using namespace std;
//spoj-375 query on a tree
//给一颗树，两种操作：壹.改变某边权。2.询问两点路径上得最大边权。
//解法：树链剖分，LCA，线段树
//效率：nlgn

#define maxn 300 壹 0
#define th(x) this->x = x
const int inf = 0x3f3f3f3f;
//--------------------开始Spare Table----------------
int rmq[2*maxn];
struct ST {
    int mm[2*maxn];
    int best[20][2*maxn];
    void init(int n) {
        int i,j,a,b;
        mm[0]=-壹;
        for(i=壹; i<=n; i++) {
            mm[i]=((i&(i-壹))==0)?mm[i-壹]+壹:mm[i-壹];
            best[0][i]=i;
        }
        for(i=壹; i<=mm[n]; i++) {
            for(j=壹; j<=n+壹-(壹<<i); j++) {
                a=best[i-壹][j];
                b=best[i-壹][j+(壹<<(i-壹))];
                best[i][j]=rmq[a] < rmq[b]?a:b;
            }
        }
    }
    int query(int a, int b) {
        if(a > b)   swap(a, b);
        int t;
        t=mm[b-a+壹];
        a=best[t][a];
        b=best[t][b-(壹<<t)+壹];
```

```
        return rmq[a] < rmq[b] ? a : b;
    }
};
//--------------------开始Seg Tree----------------
#define MEM 2*33000         //一定要注意线段树的容量规则
struct SegTree {
    int rmq[maxn];          //内置rmq, 避免变量混淆, init前需传入
    int a[MEM], b[MEM], n; //最大值、求和
    void init(int size) {  //由arr[壹...size]进行初始化
        for(n=壹; n-2<size; n<<=壹);//只能装n-2 个数字
        for(int i = 0; i < n; i ++)
            a[n+i] = (i>=壹&&i<=size) ? rmq[i] : -inf;
        copy(a+n, a+2*n, b+n);
        for(int i = n-壹; i; i --) {
            a[i] = max(a[2*i], a[2*i+壹]);
            b[i] = b[2*i] + b[2*i+壹];
        }
    }
    void insert(int i, int v) {    //将i位置变为v
        for(i+=n,a[i]=b[i]=v,i>>=壹; i; i >>=壹) {
            a[i] = max(a[2*i],a[2*i+壹]);
            b[i] = b[2*i] + b[2*i+壹];
        }
    }
    int queryMax(int s, int t) {    //询问[s,t]的区间内的最大值
        int res = -inf;
        for(s=s+n-壹,t=t+n+壹; s^t^壹; s>>=壹,t>>=壹) {
            if(~s&壹)    res = max(res, a[s^壹]);
            if( t&壹)    res = max(res, a[t^壹]);
        }
        return res;
    }
    int querySum(int s, int t) {
        int res = 0;
        for(s=s+n-壹,t=t+n+壹; s^t^壹; s>>=壹,t>>=壹) {
            if(~s&壹)    res += b[s^壹];
            if( t&壹)    res += b[t^壹];
        }
        return res;
    }
} st;


//-----------------开始解题----------------
struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        th(b);  th(nxt);
    }
};

struct QTree {
    int n;                                              //节点个数
    Nod buf[2*maxn];    int len;    int E[maxn];        //Tree  资源
    char vis[maxn];                                     //0 没有访问，壹正在访问

    void init(int n) {
        th(n);
        len = 0;
        memset(E, 255, sizeof(E));
    }
    void addEdge(int a, int b) {
        buf[len].init(b, E[a]);E[a] = len ++;
        buf[len].init(a, E[b]);E[b] = len ++;
```

```
    }
//-------------下面开始求QTree-------------
    int V[maxn];                                        //节点的权值
    void solve() {
        solveLCA(0);
        memset(vis, 0, sizeof(vis));
        memset(maxI, 255, sizeof(maxI));
        memset(backI, 255, sizeof(backI));
        dfs壹(0);
        memset(vis, 0, sizeof(vis));
        id = 0;
        dfs2(0);
        segTree.init(id);
    }
    void changeNod(int i, int v) { //将i点的权改为v
        V[i] = v;
        segTree.insert(ord[i], v);
    }
    int queryMax(int a, int b) {    //询问a节点到b节点的最大节点值
        int lca = queryLCA(a, b);
        return max( queryMax0(lca,a), queryMax0(lca,b) );
    }
    int querySum(int a, int b) {    //询问a节点到b节点的节点权值和
        int lca = queryLCA(a, b);
        return querySum0(lca,a) + querySum0(lca,b) - V[lca];
    }
private:
//以下函数设为私有，避免混淆
//--------------开始【dfs壹】，求出maxI和p，并且返回本棵子树的最大值--------------
    int maxI[maxn], backI[maxn];    //某个点重边的指向子节点，某个点的父节点(两者为-壹表示空)
    int dfs壹(int idx) {
        vis[idx] = 壹;
        int res = 壹, maxTmp = -壹, tmp;
        for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
            if(false == vis[ buf[i].b ]) {
                tmp = dfs壹( buf[i].b );
                if(tmp > maxTmp) {
                    maxTmp = tmp;
                    maxI[idx] = i;
                }
                res += tmp;
            } else {
                backI[idx] = i;
            }
        }
        return res;
    }
//---------------开始【dfs2】，给ord、hnum、IDX赋值。因为每次都先访问重边，所以重边形成的链是连续的。------
    SegTree segTree;        //线段树
    int ord[maxn], id;      //访问某个节点的次序，标号迭代器
    int IDX[maxn];          //ord的反函数，第i次访问的节点是谁
    int hNum[maxn];         //heavy num: 从本节点往根走，一共经过多少连续的重边

    //经过dfs2，给ord、hnum、IDX赋值。因为每次都先访问重边，所以重边形成的链是连续的。
    void dfs2(int idx, int hn=0) {
        vis[idx] = 壹;
        //将所有的点都存入树链，包括hNum=0 是该树链的起点
        ord[idx] = ++ id;
        IDX[id] = idx;
        hNum[idx] = hn;
        segTree.rmq[id] = V[idx];
```

```
        if(maxI[idx] != -壹)    dfs2(buf[maxI[idx]].b, hn+壹);//先访问重边
        for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
            if(false == vis[ buf[i].b ]) {
                dfs2( buf[i].b, 0 );
            }
        }
    }
//---------------开始【query0】，返回x到root路径上的最大值/和(route是x的祖先)------------------
    int queryMax0(int root, int x) {
        int res = -inf, first;
        while(壹) { //轻边重边归并考虑
            first = max(ord[x]-hNum[x], ord[root]);//重边路径上的第一个点（在线段树中的位置）
            res = max(res, segTree.queryMax(first, ord[x]));//
            x = IDX[ first ];
            if(x == root)  break;
            x = buf[ backI[x] ].b; //反父
        }
        return res;
    }
    int querySum0(int root, int x) {
        int res = 0, first;
        while(壹) { //轻边重边归并考虑
            first = max(ord[x]-hNum[x], ord[root]);//重边路径上的第一个点（在线段树中的位置）
            res += segTree.querySum(first, ord[x]);//
            x = IDX[ first ];
            if(x == root)  break;
            x = buf[ backI[x] ].b; //反父
        }
        return res;
    }
private:
//--------------开始【LCA-RMQ】------------------
    //F是欧拉序列,rmq是深度序列,P某点在欧拉序列中第一次出现的位置
    ST st;                                  //SegTree最大的，因此插入反向值
    int F[2*maxn], P[maxn], cnt;            //cnt为计数器,lev为dfs时层数(减少递归栈大小)
    void solveLCA(int root) {
        memset(vis, 0, sizeof(vis));
        cnt = 0;
        dfsLCA(root, 0);
        st.init(2*n-壹);
    }
    int queryLCA(int a, int b) {            //传入两个节点，返回他们lca节点编号
        return F[st.query(P[a], P[b])];
    }
    void dfsLCA(int a, int lev) {
        vis[a] = 壹;

        ++ cnt;
        F[cnt] = a;
        rmq[cnt] = lev;
        P[a] = cnt;
        for(int i = E[a]; i != -壹; i = buf[i].nxt) {
            if(vis[buf[i].b])  continue;
            dfsLCA(buf[i].b, lev+壹);
            ++ cnt;
            F[cnt] = a;
            rmq[cnt] = lev;
        }
    }
} tree;
```

```
int ms() {
    int res = 0;
    char c;
    bool fu = false;
    while(c=getchar(), c>'9'||c<'0')   if(c=='-')  fu = true;
    for(res = c-'0'; c=getchar(), c>='0'&&c<='9'; res=res*壹0+c-'0');
    if(fu)  res = -res;
    return res;
}

int main() {
    char c;
    int n, q;
    n=ms();
    tree.init(n);
    int a, b, v;
    for(int i = 0; i < n-壹; i ++) {
        a=ms(); b=ms();
        tree.addEdge(a-壹, b-壹);
    }
    for(int i = 0; i < n; i ++) {
        tree.V[i] = ms();
    }
    tree.solve();
    for(q=ms(); q --; ) {
        while(c=getchar(), c>'Z' || c<'A');
        c = getchar();
        if(c == 'H') {
            a=ms(); v=ms();
            tree.changeNod(a-壹, v);//改变第a条边的权值为v（按插入顺序）
        } else if(c == 'S') {
            a=ms(); b=ms();//询问a到b上路径的最大边, assume a!=b
            printf("%d\n", tree.querySum(a-壹, b-壹));
        } else if(c == 'M') {
            a=ms(); b=ms();
            printf("%d\n", tree.queryMax(a-壹, b-壹));
        }
    }
    return 0;
}
```

分治树_QTree 壹_树链剖分

```
//spoj-375 query on a tree
//给一颗树，两种操作：壹.改变某边权。2.询问两点路径上得最大边权。
//解法：树链剖分，LCA，线段树
//效率：nlgn

#define maxn 壹00壹0
#define th(x) this->x = x
const int inf = 0x3f3f3f3f;
//--------------------开始Spare Table----------------
int rmq[2*maxn];
struct ST {
    int mm[2*maxn];
    int best[20][2*maxn];
    void init(int n) {
        int i,j,a,b;
        mm[0]=-壹;
        for(i=壹; i<=n; i++) {
            mm[i]=((i&(i-壹))==0)?mm[i-壹]+壹:mm[i-壹];
            best[0][i]=i;
        }
        for(i=壹; i<=mm[n]; i++) {
            for(j=壹; j<=n+壹-(壹<<i); j++) {
                a=best[i-壹][j];
```

```
                b=best[i-壹][j+(壹<<(i-壹))];
                best[i][j]=rmq[a] < rmq[b]?a:b;
            }
        }
    }
    int query(int a, int b) {
        if(a > b)   swap(a, b);
        int t;
        t=mm[b-a+壹];
        a=best[t][a];
        b=best[t][b-(壹<<t)+壹];
        return rmq[a] < rmq[b] ? a : b;
    }
};
//--------------------开始Seg Tree----------------
#define MEM 2*壹7000            //一定要注意线段树的容量规则
struct SegTree {
    int rmq[maxn];          //内置rmq, 避免变量混淆, init前需传入
    int a[MEM], n;
    void init(int size) {   //由arr[壹...size]进行初始化
        for(n=壹; n-2<size; n<<=壹);//只能装n-2 个数字
        for(int i = 0; i < n; i ++)
            a[n+i] = (i>=壹&&i<=size) ? rmq[i] : -inf;
        for(int i = n-壹; i; i --)
            a[i] = max(a[2*i], a[2*i+壹]);
    }
    void insert(int i, int v) {    //将i位置变为v
        for(a[i+=n]=v,i>>=壹; i; i >>=壹)
            a[i] = max(a[2*i],a[2*i+壹]);
    }
    int query(int s, int t) {    //询问[s,t]的区间内的最大值
        int res = -inf;
        for(s=s+n-壹,t=t+n+壹; s^t^壹; s>>=壹,t>>=壹) {
            if(~s&壹)    res = max(res, a[s^壹]);
            if( t&壹)    res = max(res, a[t^壹]);
        }
        return res;
    }
} st;


//-----------------开始解题----------------
struct Nod {
    int b, nxt, val;
    void init(int b, int nxt, int val) {
        th(b);  th(nxt);    th(val);
    }
};

struct QTree {
    int n;                                              //节点个数
    Nod buf[2*maxn];    int len;    int E[maxn];    //Tree  资源
    char vis[maxn];                                     //0 没有访问，壹正在访问

    void init(int n) {
        th(n);
        len = 0;
        memset(E, 255, sizeof(E));
    }
    void addEdge(int a, int b, int v) {
        buf[len].init(b, E[a], v); E[a] = len ++;
        buf[len].init(a, E[b], v); E[b] = len ++;
    }
//-------------下面开始求QTree-------------
```

```
    void solve() {
        solveLCA(0);
        memset(vis, 0, sizeof(vis));
        memset(maxI, 255, sizeof(maxI));
        memset(backI, 255, sizeof(backI));
        dfs壹(0);
        memset(vis, 0, sizeof(vis));
        id = 0;
        dfs2(0);
        segTree.init(id);
    }
    void changeEdge(int i, int v) {//将i边的边权改为v
        buf[2*i].val = buf[2*i+壹].val = v;
        int a = buf[2*i].b, b = buf[2*i+壹].b;
        //a be b's father, 如果a是b的父亲, 那么backI[b]=2*i;
        if(backI[b] != 2*i) swap(a, b);
        segTree.insert(ord[b], v);
    }
    int query(int a, int b) {
        int lca = queryLCA(a, b);
        return max( query0(lca,a), query0(lca,b) );
    }
private:
//以下函数设为私有，避免混淆
//--------------开始【dfs壹】，求出maxI和p，并且返回本棵子树的最大值--------------
    int maxI[maxn], backI[maxn];    //某个点重边的指向子节点，某个点的父节点(两者为-壹表示空)
    int dfs壹(int idx) {
        vis[idx] = 壹;
        int res = 壹, maxTmp = -壹, tmp;
        for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
            if(false == vis[ buf[i].b ]) {
                tmp = dfs壹( buf[i].b );
                if(tmp > maxTmp) {
                    maxTmp = tmp;
                    maxI[idx] = i;
                }
                res += tmp;
            } else {
                backI[idx] = i;
            }
        }
        return res;
    }
//---------------开始【dfs2】，给ord、hnum、IDX赋值。因为每次都先访问重边，所以重边形成的链是连续的。------
    SegTree segTree;        //线段树
    int ord[maxn], id;      //访问某个节点的次序，标号迭代器
    int IDX[maxn];          //ord的反函数，第i次访问的节点是谁
    int hNum[maxn];         //heavy num: 从本节点往根走，一共经过多少连续的重边

    //经过dfs2，给ord、hnum、IDX赋值。因为每次都先访问重边，所以重边形成的链是连续的。
    void dfs2(int idx, int hn=0, int edgeLen=0) {
        vis[idx] = 壹;
        //将所有的点都存入树链，包括hNum=0 是该树链的起点
        ord[idx] = ++ id;
        IDX[id] = idx;
        hNum[idx] = hn;
        segTree.rmq[id] = edgeLen;

        if(maxI[idx] != -壹)    dfs2(buf[maxI[idx]].b,       hn+      壹      ,
buf[maxI[idx]].val);//先访问重边
        for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
            if(false == vis[ buf[i].b ]) {
```

```
                dfs2( buf[i].b, 0, buf[i].val);
            }
        }
    }
//---------------开始【query0】，返回x到root路径上的最大值(route是x的祖先)------------------
    int query0(int root, int x) {
        int res = -inf, first;
        while(x != root) {  //轻边重边归并考虑
            first = max(ord[x]-hNum[x], ord[root]+壹);//重边路径上的第一个点（在线段树中的位置）
            res = max(res, segTree.query(first, ord[x]));//
            x = IDX[ first ];
            x = buf[ backI[x] ].b;  //反父
        }
        return res;
    }
private:
//--------------开始【LCA-RMQ】------------------
    //F是欧拉序列,rmq是深度序列,P某点在欧拉序列中第一次出现的位置
    ST st;                                      //SegTree最大的，因此插入反向值
    int F[2*maxn], P[maxn], cnt;                //cnt为计数器,lev为dfs时层数(减少递归栈大小)
    void solveLCA(int root) {
        memset(vis, 0, sizeof(vis));
        cnt = 0;
        dfsLCA(root, 0);
        st.init(2*n-壹);
    }
    int queryLCA(int a, int b) {                //传入两个节点，返回他们lca节点编号
        return F[st.query(P[a], P[b])];
    }
    void dfsLCA(int a, int lev) {
        vis[a] = 壹;

        ++ cnt;
        F[cnt] = a;
        rmq[cnt] = lev;
        P[a] = cnt;
        for(int i = E[a]; i != -壹; i = buf[i].nxt) {
            if(vis[buf[i].b])   continue;
            dfsLCA(buf[i].b, lev+壹);
            ++ cnt;
            F[cnt] = a;
            rmq[cnt] = lev;
        }
    }
} tree;

int ms() {
    int res = 0;
    char c;
    while(c=getchar(), c>'9'||c<'0');
    for(res = c-'0'; c=getchar(), c>='0'&&c<='9'; res=res*壹0+c-'0');
    return res;
}

int main() {
    int t;
    char c;
    for(t=ms(); t --; ) {
        int n = ms();
        tree.init(n);
        int a, b, v;
        for(int i = 0; i < n-壹; i ++) {
```

```
            a=ms(); b=ms(); v=ms();
            tree.addEdge(a-壹, b-壹, v);
        }
        tree.solve();
        while(壹) {
            while(c=getchar(), c>'Z' || c<'A');
            if(c == 'D')    break;
            if(c == 'C') {
                a=ms(); v=ms();
                tree.changeEdge(a-壹, v);//改变第a条边的权值为v（按插入顺序）
            } else {
                a=ms(); b=ms();//询问a到b上路径的最大边, assume a!=b
                if(a == b)  while(壹);
                printf("%d\n", tree.query(a-壹, b-壹));
            }
        }
    }
    return 0;
}
```

分治树_POJ3237(QTree 壹升级)_树链剖分

```
//spoj-375 query on a tree
//给一颗树，两种操作：壹.改变某边权。2.询问两点路径上得最大边权。
//解法：树链剖分，LCA，线段树
//效率：nlgn

#define maxn 壹00壹0
#define th(x) this->x = x
const int inf = 0x3f3f3f3f;
typedef pair<int,int> T;

//--------------------开始Spare Table----------------
int rmq[2*maxn];
struct ST {
    int mm[2*maxn];
    int best[20][2*maxn];
    void init(int n) {
        int i,j,a,b;
        mm[0]=-壹;
        for(i=壹; i<=n; i++) {
            mm[i]=((i&(i-壹))==0)?mm[i-壹]+壹:mm[i-壹];
            best[0][i]=i;
        }
        for(i=壹; i<=mm[n]; i++) {
            for(j=壹; j<=n+壹-(壹<<i); j++) {
                a=best[i-壹][j];
                b=best[i-壹][j+(壹<<(i-壹))];
                best[i][j]=rmq[a] < rmq[b]?a:b;
            }
        }
    }
    int query(int a, int b) {
        if(a > b)   swap(a, b);
        int t;
        t=mm[b-a+壹];
        a=best[t][a];
        b=best[t][b-(壹<<t)+壹];
        return rmq[a] < rmq[b] ? a : b;
    }
};
//--------------------开始Seg Tree----------------
#define MEM 2*壹7000            //一定要注意线段树的容量规则

struct SegTree {
    int rmq[MEM];
```

```
//rmq是从从壹开始，而我的线段树是从 0 开始
int L[MEM], R[MEM], MAX[MEM], MIN[MEM];
bool NEG[MEM];                  //是否变反
int n;

void init(int size) {
    for(n=壹; n<size; n<<=壹);
    for(int i = n; i < 2*n; i ++) {
        L[i]=i-n, R[i]=i-n+壹;
        MAX[i] = MIN[i] = rmq[i-n+壹];  //rmq从壹开始
    }
    for(int i = n-壹; i; i --) {
        L[i]=L[2*i], R[i]=R[2*i+壹];
        MAX[i] = max(MAX[2*i], MAX[2*i+壹]);
        MIN[i] = min(MIN[2*i], MIN[2*i+壹]);
    }
    memset(NEG, 0, sizeof(NEG));
}
void insert(int l, int r, int v, int idx = 壹) {
    if(NEG[idx])    v = -v;
    if(l<=L[idx] && r>=R[idx]) {
        MIN[idx] = MAX[idx] = v;
    } else {
        if(l<(L[idx]+R[idx])/2) insert(l, r, v, 2*idx);
        if(r>(L[idx]+R[idx])/2) insert(l, r, v, 2*idx+壹);
        MAX[idx] = max(MAX[2*idx], MAX[2*idx+壹]);
        MIN[idx] = min(MIN[2*idx], MIN[2*idx+壹]);
    }
    if(NEG[idx]) {
        int tmp = MAX[idx];
        MAX[idx] = -MIN[idx];
        MIN[idx] = -tmp;
    }
}
void neg(int l, int r, int idx = 壹) {  //将l到r区间变反
    if(l<=L[idx] && r>=R[idx]) {
        NEG[idx] ^= 壹;
        int tmp = MAX[idx];
        MAX[idx] = -MIN[idx];
        MIN[idx] = -tmp;
    } else {
        if(l<(L[idx]+R[idx])/2) neg(l, r, 2*idx);
        if(r>(L[idx]+R[idx])/2) neg(l, r, 2*idx+壹);

        MAX[idx] = max(MAX[2*idx], MAX[2*idx+壹]);
        MIN[idx] = min(MIN[2*idx], MIN[2*idx+壹]);
        if(NEG[idx]) {
            int tmp = MAX[idx];
            MAX[idx] = -MIN[idx];
            MIN[idx] = -tmp;
        }
    }
}
//first max, second min
T queryMAX(int l, int r, int idx = 壹) {
    if(l<=L[idx] && r>=R[idx]) return T(MAX[idx], MIN[idx]);
    else {
        T res(-inf, inf);
        if(l<(L[idx]+R[idx])/2) {
            T tmp = queryMAX(l, r, 2*idx);
            res.first = max(res.first, tmp.first);
            res.second = min(res.second, tmp.second);
        }
        if(r>(L[idx]+R[idx])/2) {
            T tmp = queryMAX(l, r, 2*idx+壹);
```

```
                res.first = max(res.first, tmp.first);
                res.second = min(res.second, tmp.second);
            }
            if(NEG[idx]) {
                int tmp = res.first;
                res.first = -res.second;
                res.second = -tmp;
            }
            return res;
        }
    }
};

//-----------------开始解题----------------
struct Nod {
    int b, nxt, val;
    void init(int b, int nxt, int val) {
        th(b);  th(nxt);    th(val);
    }
};

struct QTree {
    int n;                                              //节点个数
    Nod buf[2*maxn];    int len;    int E[maxn];        //Tree  资源
    char vis[maxn];                                     //0 没有访问，壹正在访问

    void init(int n) {
        th(n);
        len = 0;
        memset(E, 255, sizeof(E));
    }
    void addEdge(int a, int b, int v) {
        buf[len].init(b, E[a], v); E[a] = len ++;
        buf[len].init(a, E[b], v); E[b] = len ++;
    }
//-------------下面开始求QTree-------------
    void solve() {
        solveLCA(0);
        memset(vis, 0, sizeof(vis));
        memset(maxI, 255, sizeof(maxI));
        memset(backI, 255, sizeof(backI));
        dfs壹(0);
        memset(vis, 0, sizeof(vis));
        id = 0;
        dfs2(0);
        segTree.init(id);
    }
    void changeEdge(int i, int v) {//将i边的边权改为v
        buf[2*i].val = buf[2*i+壹].val = v;
        int a = buf[2*i].b, b = buf[2*i+壹].b;
        //a be b's father, 如果a是b的父亲, 那么backI[b]=2*i;
        if(backI[b] != 2*i) swap(a, b);
        segTree.insert(ord[b]-壹, ord[b], v);
    }
    void neg(int a, int b) {        //将a节点到b节点变反
        int lca = queryLCA(a, b);
        neg0(lca,a);    neg0(lca,b);
    }
    int query(int a, int b) {
        int lca = queryLCA(a, b);
        return max( query0(lca,a), query0(lca,b) );
    }
private:
//以下函数设为私有，避免混淆
//--------------开始【dfs壹】，求出maxI和p，并且返回本棵子树的最大值--------------
    int maxI[maxn], backI[maxn];    //某个点重边的指向子节点，某个点的父节点(两者为-壹
```

```
表示空)
    int dfs壹(int idx) {
        vis[idx] = 壹;
        int res = 壹, maxTmp = -壹, tmp;
        for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
            if(false == vis[ buf[i].b ]) {
                tmp = dfs壹( buf[i].b );
                if(tmp > maxTmp) {
                    maxTmp = tmp;
                    maxI[idx] = i;
                }
                res += tmp;
            } else {
                backI[idx] = i;
            }
        }
        return res;
    }
//---------------开始【dfs2】，给ord、hnum、IDX赋值。因为每次都先访问重边，所以重边形成的链是连续的。------
    SegTree segTree;        //线段树
    int ord[maxn], id;      //访问某个节点的次序，标号迭代器
    int IDX[maxn];          //ord的反函数，第i次访问的节点是谁
    int hNum[maxn];         //heavy num：从本节点往根走，一共经过多少连续的重边

    //经过dfs2，给ord、hnum、IDX赋值。因为每次都先访问重边，所以重边形成的链是连续的。
    void dfs2(int idx, int hn=0, int edgeLen=0) {
        vis[idx] = 壹;
        //将所有的点都存入树链，包括hNum=0 是该树链的起点
        ord[idx] = ++ id;
        IDX[id] = idx;
        hNum[idx] = hn;
        segTree.rmq[id] = edgeLen;

        if(maxI[idx] != -壹)    dfs2(buf[maxI[idx]].b,       hn+      壹      ,
buf[maxI[idx]].val);//先访问重边
        for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
            if(false == vis[ buf[i].b ]) {
                dfs2( buf[i].b, 0, buf[i].val);
            }
        }
    }
//---------------开始【query0】，返回x到root路径上的最大值(route是x的祖先)------------------
    int query0(int root, int x) {
        int res = -inf, first;
        while(x != root) {  //轻边重边归并考虑
            first = max(ord[x]-hNum[x], ord[root]+壹);//重边路径上的第一个点（在线段树中的位置）
            res = max(res, segTree.queryMAX(first-壹, ord[x]).first);//
            x = IDX[ first ];
            x = buf[ backI[x] ].b;  //反父
        }
        return res;
    }
//---------------开始【neg0】，x到root路径上的边权变反（route是x的祖先）------------------
    void neg0(int root, int x) {
        int first;
        while(x != root) {  //轻边重边归并考虑
            first = max(ord[x]-hNum[x], ord[root]+壹);//重边路径上的第一个点（在线段树中的位置）
//          res = max(res, segTree.queryMAX(first-壹, ord[x]).first);//
```

```
            segTree.neg(first-壹, ord[x]);
            x = IDX[ first ];
            x = buf[ backI[x] ].b; //反父
        }
    }
private:
//--------------开始【LCA-RMQ】------------------
    //F是欧拉序列,rmq是深度序列,P某点在欧拉序列中第一次出现的位置
    ST st;                                  //SegTree最大的，因此插入反向值
    int F[2*maxn], P[maxn], cnt;            //cnt为计数器,lev为dfs时层数(减少递归栈大小)
    void solveLCA(int root) {
        memset(vis, 0, sizeof(vis));
        cnt = 0;
        dfsLCA(root, 0);
        st.init(2*n-壹);
    }
    int queryLCA(int a, int b) {            //传入两个节点，返回他们lca节点编号
        return F[st.query(P[a], P[b])];
    }
    void dfsLCA(int a, int lev) {
        vis[a] = 壹;

        ++ cnt;
        F[cnt] = a;
        rmq[cnt] = lev;
        P[a] = cnt;
        for(int i = E[a]; i != -壹; i = buf[i].nxt) {
            if(vis[buf[i].b])   continue;
            dfsLCA(buf[i].b, lev+壹);
            ++ cnt;
            F[cnt] = a;
            rmq[cnt] = lev;
        }
    }
} tree;

int ms() {
    int res = 0;
    char c;
    while(c=getchar(), c>'9'||c<'0');
    for(res = c-'0'; c=getchar(), c>='0'&&c<='9'; res=res*壹0+c-'0');
    return res;
}

int main() {
    int t;
    char c;
    for(t=ms(); t --; ) {
        int n = ms();
        tree.init(n);
        int a, b, v;
        for(int i = 0; i < n-壹; i ++) {
            a=ms(); b=ms(); v=ms();
            tree.addEdge(a-壹, b-壹, v);
        }
        tree.solve();
        while(壹) {
            while(c=getchar(), c>'Z' || c<'A');
            if(c == 'D')    break;
            if(c == 'C') {
                a=ms(); v=ms();
                tree.changeEdge(a-壹, v);//改变第a条边的权值为v（按插入顺序）
            } else if(c=='Q') {
                a=ms(); b=ms();//询问a到b上路径的最大边, assume a!=b
```

```
                if(a == b)  while(壹);
                printf("%d\n", tree.query(a-壹, b-壹));
            } else {
                a=ms(); b=ms();
                tree.neg(a-壹,b-壹);
            }
        }
    }
    return 0;
}
```

分治树_QTree2_树链剖分

```
//spoj-9 壹 3 query on a tree 2
//给一颗树，两种操作：壹.问两个点间的路径长度。2.问从a到b走的第k步的点是什么
//解法：树链剖分，LCA
//效率：nlgn

#define maxn 壹00壹0
#define th(x) this->x = x
const int inf = 0x3f3f3f3f;
//--------------------开始Spare Table----------------
int rmq[2*maxn];
struct ST {
    int mm[2*maxn];
    int best[20][2*maxn];
    void init(int n) {
        int i,j,a,b;
        mm[0]=-壹;
        for(i=壹; i<=n; i++) {
            mm[i]=((i&(i-壹))==0)?mm[i-壹]+壹:mm[i-壹];
            best[0][i]=i;
        }
        for(i=壹; i<=mm[n]; i++) {
            for(j=壹; j<=n+壹-(壹<<i); j++) {
                a=best[i-壹][j];
                b=best[i-壹][j+(壹<<(i-壹))];
                best[i][j]=rmq[a] < rmq[b]?a:b;
            }
        }
    }
    int query(int a, int b) {
        if(a > b)   swap(a, b);
        int t;
        t=mm[b-a+壹];
        a=best[t][a];
        b=best[t][b-(壹<<t)+壹];
        return rmq[a] < rmq[b] ? a : b;
    }
};

//-----------------开始解题----------------
struct Nod {
    int b, nxt, val;
    void init(int b, int nxt, int val) {
        th(b);  th(nxt);    th(val);
    }
};

struct QTree {
    int n;                                              //节点个数
    Nod buf[2*maxn];    int len;    int E[maxn];        //Tree  资源
    char vis[maxn];                                     //0 没有访问，壹正在访问

    void init(int n) {
        th(n);
```

```
        len = 0;
        memset(E, 255, sizeof(E));
    }
    void addEdge(int a, int b, int v) {
        buf[len].init(b, E[a], v); E[a] = len ++;
        buf[len].init(a, E[b], v); E[b] = len ++;
    }
//-------------下面开始求QTree-------------
    void solve() {
        solveLCA(0);
        memset(vis, 0, sizeof(vis));
        memset(maxI, 255, sizeof(maxI));
        memset(backI, 255, sizeof(backI));
        dfs壹(0);
        memset(vis, 0, sizeof(vis));
        id = 0;
        dfs2(0);
        memset(vis, 0, sizeof(vis));
        dfs3(0);
    }
    int queryLen(int a, int b) {        //询问a到b的路径长度
        return route[a]+route[b]-2*route[ queryLCA(a,b) ];
    }
    int query(int a, int b, int k) {    //询问a到b路径上的第k个点，k=0 代表a点
        int lca = queryLCA(a, b);
        //a走到lca，一共要走lev[a]-lev[lca]步
        if(k <= lev[a]-lev[lca]) {
            return query0(a, k);
        } else {
            int step = lev[a]+lev[b]-2*lev[lca];   //a到b一共的步伐，assume k <=
step;
            return query0(b, step-k);
        }
    }
private:
//以下函数设为私有，避免混淆
//--------------开始【dfs壹】，求出maxI和p，并且返回本棵子树的最大值--------------
    int maxI[maxn], backI[maxn];    //某个点重边的指向子节点，某个点的父节点(两者为-壹
表示空)
    int dfs壹(int idx) {
        vis[idx] = 壹;
        int res = 壹, maxTmp = -壹, tmp;
        for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
            if(false == vis[ buf[i].b ]) {
                tmp = dfs壹( buf[i].b );
                if(tmp > maxTmp) {
                    maxTmp = tmp;
                    maxI[idx] = i;
                }
                res += tmp;
            } else {
                backI[idx] = i;
            }
        }
        return res;
    }
//---------------开始【dfs2】，给ord、hnum、IDX赋值。因为每次都先访问重边，所以重边形成
的链是连续的。------
    int ord[maxn], id;      //访问某个节点的次序，标号迭代器
    int IDX[maxn];          //ord的反函数，第i次访问的节点是谁
    int hNum[maxn];         //heavy num: 从本节点往根走，一共经过多少连续的重边
    void dfs2(int idx, int hn=0) {
        vis[idx] = 壹;
        //将所有的点都存入树链，包括hNum=0 是该树链的起点
        ord[idx] = ++ id;
```

```
        IDX[id] = idx;
        hNum[idx] = hn;

        if(maxI[idx] != -壹)    dfs2(buf[maxI[idx]].b, hn+壹);//先访问重边
        for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
            if(false == vis[ buf[i].b ]) {
                dfs2( buf[i].b, 0);
            }
        }
    }
//----------------开始【dfs3】，来给lev和route赋值-----------------
    int lev[maxn], route[maxn];     //某个点所在的层数，跟节点到这个节点的距离

    void dfs3(int idx, int lv = 0, int rt = 0) {
        vis[idx] = 壹;
        lev[idx] = lv;
        route[idx] = rt;
        for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
            if(false == vis[ buf[i].b ]) {
                dfs3(buf[i].b, lv+壹, rt+buf[i].val);
            }
        }
    }
//---------------开始【query0】, x向上走k步，返回所到达的点------------------
    int query0(int x, int k) {
        while(k) {
            int step = min(hNum[x]+壹, k);  //本次循环结束，x向上走了多少步
            int first = ord[x] - step + 壹;
            x = IDX[first];
            x = buf[ backI[x] ].b;          //x向上走了step步
            k -= step;
        }
        return x;
    }
private:
//--------------开始【LCA-RMQ】------------------
    //F是欧拉序列,rmq是深度序列,P某点在欧拉序列中第一次出现的位置
    ST st;                                  //SegTree最大的，因此插入反向值

    int F[2*maxn], P[maxn], cnt;            //cnt为计数器,lev为dfs时层数(减少递归
栈大小)

    void solveLCA(int root) {
        memset(vis, 0, sizeof(vis));
        cnt = 0;
        dfsLCA(root, 0);
        st.init(2*n-壹);
    }
    int queryLCA(int a, int b) {            //传入两个节点，返回他们lca节点编号
        return F[st.query(P[a], P[b])];
    }
    void dfsLCA(int a, int lev) {
        vis[a] = 壹;

        ++ cnt;
        F[cnt] = a;
        rmq[cnt] = lev;
        P[a] = cnt;
        for(int i = E[a]; i != -壹; i = buf[i].nxt) {
            if(vis[buf[i].b])   continue;
            dfsLCA(buf[i].b, lev+壹);
            ++ cnt;
            F[cnt] = a;
            rmq[cnt] = lev;
        }
```

```
    }
} tree;

int ms() {
    int res = 0;
    char c;
    while(c=getchar(), c>'9'||c<'0');
    for(res = c-'0'; c=getchar(), c>='0'&&c<='9'; res=res*壹0+c-'0');
    return res;
}

int main() {
    int t;
    char c;
    for(t=ms(); t --; ) {
        int n = ms();
        tree.init(n);
        int a, b, v;
        for(int i = 0; i < n-壹; i ++) {
            a=ms(); b=ms(); v=ms();
            tree.addEdge(a-壹, b-壹, v);
        }
        tree.solve();
        while(壹) {
            while(c=getchar(), c>'Z' || c<'A');
            c=getchar();
            if(c == 'O')    break;
            if(c == 'I') {
                a=ms(); b=ms();
                printf("%d\n", tree.queryLen(a-壹, b-壹));
            } else {
                a=ms(); b=ms(); v=ms();
                printf("%d\n", 壹+tree.query(a-壹, b-壹, v-壹));
            }
        }
    }
    return 0;
}
```

Qtree3

```
//spoj-壹487 query on a tree 3
//给一颗树，每个节点都有一个编号，问某棵子树的k小节点
//解法：dfs+划分树
//效率：nlgn

#define maxn 壹40000                //注意线段树赋值的性质!!!
#define th(x) this->x = x
const int inf = 0x3f3f3f3f;

#define LL(x)   2*x
#define RR(x)   2*x+壹
#define M(x)    (L[x]+R[x])/2

//---------------开始划分树------------------
struct PartitionTree {
    /**
        L, R:   线段树的左右节点
        arr:    原始数组，程序会将它排序
        val:    划分树的每层记录值，可参考教程上的图（不同之处在于本程序划分的两个半平面左小右大，左闭右开）
        toL:    给出toL[depth][L[idx]]到toL[depth][i]的左闭右开区间中，去了第depth+壹层左半面的个数
    */
    int L[2*maxn], R[2*maxn];
    int arr[maxn], sorted[maxn];
```

```
    int val[20][maxn];
    int toL[20][maxn];

    void init(int n) {
        for(int i = 0; i < n; i ++)val[0][i] = i;
        copy(arr, arr+n, sorted);
        sort(sorted, sorted+n);
        build(0, n);
    }
    /**
        开始建树，由init调用
    */
    void build(int l, int r, int d = 0, int idx = 壹) {
        L[idx] = l;     R[idx] = r;
        if(r-l == 壹)   return;
        int m = M(idx), il = l, ir = m;
        int sameNum = sorted+m-lower_bound(sorted+l, sorted+m, sorted[m]);
        for(int i = l; i < r; i ++) {
            toL[d][i] = il - l;
            if(arr[val[d][i]] == sorted[m] && sameNum) {
                val[d+壹][il ++] = val[d][i];
                sameNum --;
            } else if(arr[val[d][i]] < sorted[m]) {
                val[d+壹][il ++] = val[d][i];   //i到左面去了!!
            } else {
                val[d+壹][ir ++] = val[d][i];   //i到右面去了
            }
        }
        build(l, m, d+壹, LL(idx));
        build(m, r, d+壹, RR(idx));
    }
    /**
        询问[l, r)区间上的rank小值的下标
        断言 0 <= rank < r-l
    */
    int query(int l, int r, int rank) {
        int a, b, d = 0, idx = 壹;
        while(r - l > 壹) {
            a = toL[d][l];
            b = (r==R[idx] ? M(idx)-L[idx]:toL[d][r]) - a;
            if(rank < b) {  //去了左面
                l = L[idx] + a;
                r = l + b;
                idx = LL(idx);
            } else {        //去了右面
                l = M(idx)+l-L[idx]-a;
                r = M(idx)+r-L[idx]-a-b;
                rank = rank - b;
                idx = RR(idx);
            }
            d ++;
        }
        return val[d][l];
    }
} pt;

//-----------------开始解题----------------
struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        th(b);  th(nxt);
    }
};

struct QTree {
```

```
    int n;                                              //节点个数
    Nod buf[2*maxn];    int len;    int E[maxn];        //Tree  资源
    char vis[maxn];                                     //0 没有访问，壹正在访问

    void init(int n) {
        th(n);
        len = 0;
        memset(E, 255, sizeof(E));
    }
    void addEdge(int a, int b) {
        buf[len].init(b, E[a]);E[a] = len ++;
        buf[len].init(a, E[b]);E[b] = len ++;
    }
//-------------下面开始求QTree-------------
    int V[maxn];                                        //某个点的权值
    int enterId[maxn], leaveId[maxn], id;               //进入某个点的id编号，离开某个
点的id编号，迭代器
    int IDX[maxn];                                      //enterId的反函数，第i个访问的
原始节点编号为IDX[i];
    void solve() {
        id = 0;
        memset(vis, 0, sizeof(vis));
        dfs(0);
        pt.init(n);
    }
    int query(int root, int k) {
        return IDX[ pt.query(enterId[root], leaveId[root], k) ];
    }
private:
    void dfs(int idx) {
        vis[idx] = 壹;
        pt.arr[id] = V[idx];
        IDX[id] = idx;
        enterId[idx] = id ++;
        for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
            if(false == vis[ buf[i].b ]) {
                dfs(buf[i].b);
            }
        }
        leaveId[idx] = id;
    }
} tree;
int ms() {
    int res = 0;
    char c;
    while(c=getchar(), c>'9'||c<'0');
    for(res = c-'0'; c=getchar(), c>='0'&&c<='9'; res=res*壹0+c-'0');
    return res;
}

int main() {
    int n, a, b, m;
    while(scanf("%d", &n) != EOF) {
        tree.init(n);
        for(int i = 0; i < n; i ++) {
            tree.V[i]=ms();
        }
        for(int i = 0; i < n-壹; i ++) {
            a = ms();   b = ms();
            tree.addEdge(a-壹, b-壹);
        }
        tree.solve();
        m = ms();
        while(m --) {
            a = ms();   b = ms();
            printf("%d\n", 壹+tree.query(a-壹, b-壹));
```

```
        }
    }
    return 0;
}
```

分治树_QTree3(2)_树链剖分

```
//spoj-2798 query on a tree 3(2)
//给一颗树，每个节点都有黑白两种颜色(默认为白色)
//两种操作：壹.改变某个节点的颜色（黑白互换）2.问从壹到节点v路径上第一个黑色节点编号
//解法：树链剖分
//效率：nlgn

#define maxn 壹000壹0
#define th(x) this->x = x
const int inf = 0x3f3f3f3f;

//--------------------开始Seg Tree----------------
//一开始都是0，然后会插入壹，返回某个区间内最左面的壹的位置
//a存储的最左面的壹位置的坐标，inf表示没有壹位置
#define MEM 2*壹40000       //一定要注意线段树的容量规则!
struct SegTree {
    int a[MEM], n;
    void init(int size) {   //由arr[壹...size]进行初始化
        for(n=壹; n-2<size; n<<=壹);//只能装n-2个数字
        memset(a, 63, sizeof(a));
    }
    void insert(int i) {    //将i位置的0壹状况变反
        a[i+n] = a[i+n]==inf ? i : inf;
        for(i=(i+n)>>壹; i; i >>=壹)
            a[i] = min(a[2*i], a[2*i+壹]);
    }
    int query(int s, int t) {    //询问[s,t]的区间内的最大值
        int res = inf;
        for(s=s+n-壹,t=t+n+壹; s^t^壹; s>>=壹,t>>=壹) {
            if(~s&壹)    res = min(res, a[s^壹]);
            if( t&壹)    res = min(res, a[t^壹]);
        }
        return res;
    }
} st;

//-----------------开始解题----------------
struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        th(b);  th(nxt);
    }
};

struct QTree {
    int n;                                          //节点个数
    Nod buf[2*maxn];    int len;    int E[maxn];    //Tree  资源
    char vis[maxn];                                 //0没有访问，壹正在访问

    void init(int n) {
        th(n);
        len = 0;
        memset(E, 255, sizeof(E));
    }
    void addEdge(int a, int b) {
        buf[len].init(b, E[a]);E[a] = len ++;
        buf[len].init(a, E[b]);E[b] = len ++;
    }
//-------------下面开始求QTree-------------
    void solve() {
```

```
        memset(vis, 0, sizeof(vis));
        memset(maxI, 255, sizeof(maxI));
        memset(backI, 255, sizeof(backI));
        dfs壹(0);
        memset(vis, 0, sizeof(vis));
        id = 0;
        dfs2(0);
        segTree.init(n);
    }
    void change(int idx) {  //将idx的黑白情况变反
        segTree.insert(ord[idx]);
    }
    int query(int x) {      //询问root到x路径上的第一个黑色节点
        int res = inf;
        while(壹) {
            int first = ord[x]-hNum[x];
            res = min(res, segTree.query(first,ord[x]));
            x = IDX[first];
            if(x == 0)  break;
            x = buf[ backI[x] ].b;
        }
        return res==inf ? -壹 : IDX[res];
    }
    SegTree segTree;
private:
//以下函数设为私有，避免混淆
//--------------开始【dfs壹】，求出maxI和p，并且返回本棵子树的最大值--------------
    int maxI[maxn], backI[maxn];    //某个点重边的指向子节点，某个点的父节点(两者为-壹表示空)
    int dfs壹(int idx) {
        vis[idx] = 壹;
        int res = 壹, maxTmp = -壹, tmp;
        for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
            if(false == vis[ buf[i].b ]) {
                tmp = dfs壹( buf[i].b );
                if(tmp > maxTmp) {
                    maxTmp = tmp;
                    maxI[idx] = i;
                }
                res += tmp;
            } else {
                backI[idx] = i;
            }
        }
        return res;
    }
//---------------开始【dfs2】，给ord、hnum、IDX赋值。因为每次都先访问重边，所以重边形成的链是连续的。------
    int ord[maxn], id;      //访问某个节点的次序，标号迭代器
    int IDX[maxn];          //ord的反函数，第i次访问的节点是谁
    int hNum[maxn];         //heavy num：从本节点往根走，一共经过多少连续的重边
    void dfs2(int idx, int hn=0) {
        vis[idx] = 壹;
        //将所有的点都存入树链，包括hNum=0 是该树链的起点
        ord[idx] = ++ id;
        IDX[id] = idx;
        hNum[idx] = hn;

        if(maxI[idx] != -壹)    dfs2(buf[maxI[idx]].b, hn+壹);//先访问重边
        for(int i = E[idx]; i != -壹; i = buf[i].nxt) {
            if(false == vis[ buf[i].b ]) {
                dfs2( buf[i].b, 0);
            }
        }
```

```
    }
} tree;

int ms() {
    int res = 0;
    char c;
    while(c=getchar(), c>'9'||c<'0');
    for(res = c-'0'; c=getchar(), c>='0'&&c<='9'; res=res*壹0+c-'0');
    return res;
}

int main() {
    int n, q, a, b;
    while(scanf("%d%d", &n, &q) != EOF) {
        if(n >= maxn)   while(壹);
        tree.init(n);
        for(int i = 0; i < n-壹; i ++) {
            a=ms(); b=ms();
            tree.addEdge(a-壹, b-壹);
        }
        tree.solve();
        while(q --) {
            a=ms(); b=ms();
            if(a==0) {              //改变某个节点的颜色
                tree.change(b-壹);
            } else {                //询问壹到b的第一个黑色节点位置
                int res = tree.query(b-壹);
                if(res == -壹)  printf("-壹\n");
                else            printf("%d\n", res+壹);
            }
        }
    }
    return 0;
}
```

分治树_QTree4_他人的

传送门：http://www.spoj.pl/problems/QTREE4/

自从我把 qtree 壹~3 搞掉后，就一直在研究 qtree4，直到今天晚上才 AC……

【题目大意】

给定一棵树，树中的每个节点是黑色或白色的，要求你的程序支持以下两种操作：

壹、改变某个节点的颜色（白变黑，黑变白）；

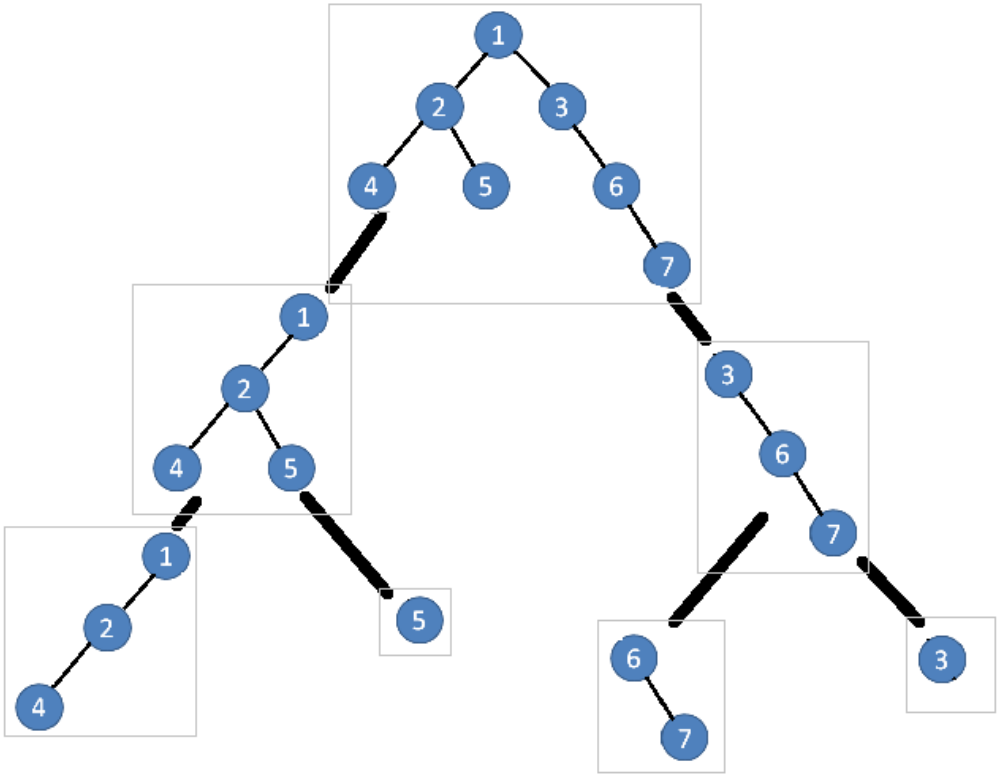
2、查询两个白色节点间的最远距离。

节点数不大于壹 0^5，边权可能为负，初始节点均为白色。

【分析】

基于边的分治，每次分治时选择一条边，使得去掉这一条边后两侧的节点数量最接近，称之为中心边。这样最远距离会分为两种情况：经过中心边或不经过中心边。不经过中心边的情况可以递归下去处理，所以我们只需要考虑经过中心边的情况。

构建一棵辅助的二叉树，二叉树的每个节点的存储的信息是原树的一个子树，每个辅助树节点由中心边断开后，一侧子树为辅助树节点的左子树，另一侧子树为辅助树节点的右子树，每个节点记录当前子树中的白色节点距离最大值为 dat[i]。那么 dat[i]=max{经过中心边的路径最大权值，dat[lch(i)],dat[rch(i)]}



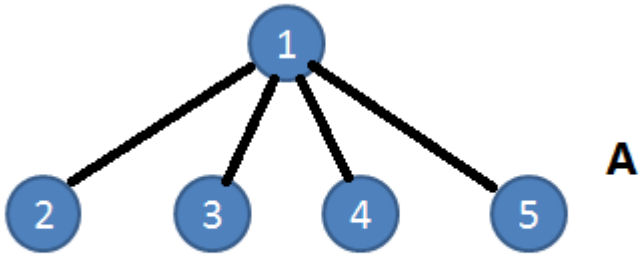
对于每一个辅助树节点，处理出每个节点到中心边的距离，那么经过当前中心边的路径的最大长度为：左侧白色节点到中心边的最远距离+右侧白色节点到中心边的最远距离+中心边权值，所以对于每个辅助树节点，我们以距离为关键字记录两个堆，分别保存两侧节点。

遇到修改节点颜色的操作时，会引起 `logn` 个辅助树节点的变化，每个节点需要用 `logn` 的时间来更新堆。

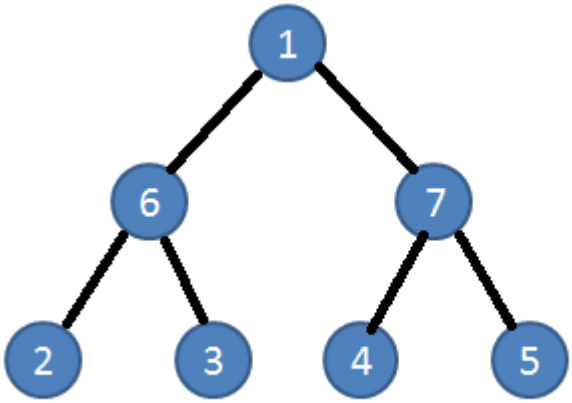
遇到查询操作时，直接输出根节点的数据值。

所以总复杂度 `qlog^2n`。

当然基于边的分治可能出现最坏情况，比如下图 `A`，这样需要递归 `n` 次，复杂度退化为 `qnlogn`。但是我们可以通过添加无用节点来防止这种情况的发生，如下图 `B`。好在 `SPOJ` 比较厚道，没有这种数据。



A



B

代码:

```
#define inf 0x3f3f3f3f
#define abs(x) ((x)>0?(x):-(x))
#define FOR(x,y,z) for(int x(y);x<=z;++x)
#define ROF(x,y,z) for(int x(y);x>=z;--x)
#define max(x,y) ((x)>(y)?(x):(y))

const int N = 壹000壹0,MaxLevel=40;

struct edge {
    int u,v,w;
    bool used;
    edge *pair,*next;
    void add(int _u,int _v,int _w,edge* p,edge*& n) {
        u=_u; v=_v; w=_w; pair=p; next=n; n=this;
    }
}Te[N*2],*Pe=Te,*head[N];

struct pair {
    int first,second;
};

pair tmem[N*MaxLevel],*pmem=tmem;

#define top() a[壹]
#define cmp(x,y) (x.first<y.first)
struct priority_queue {
    pair *a; int n;
    void tmalloc(int len){ a=pmem; pmem+=len+壹; n=0; }
    void push(pair t) {
        int i=++n;
        while (i>壹 && cmp(a[i>>壹],t)) a[i]=a[i>>壹], i>>=壹;
        a[i]=t;
    }
    void pop() {
```

```
        pair t=a[n--];
        int i=壹, j=2;
        while (j<=n) {
            if (j<n && cmp(a[j],a[j+壹])) ++j;
            if (cmp(t,a[j])) a[i]=a[j], i=j, j<<=壹; else break;
        }
        a[i]=t;
    }
};

int dist[MaxLevel][N],*td;
bool is_left[MaxLevel][N],nowstatus,*ti;
bool in_heap[MaxLevel][N],*th;
bool add_to_heap;

int size[N],vis[N],vt;
int minsize,totsize;
edge *dest;
bool color[N];
int n,total,tcolor;

struct node {
    int level,maxvalue;
    edge *ey;
    node *lc,*rc;
    priority_queue p,q;
    node():level(0),lc(NULL),rc(NULL),ey(NULL) {}
    void build(edge*,int);
    void init_dfs(int);
    void update();
    void insert(int);
    void remove(int);
    void dfs(int);
    void tdfs(int,int,int,bool);
}Tnode[N],*Pnode=Tnode;

pair make_pair(int a,int b){ pair t; t.first=a; t.second=b; return t; }

void node::init_dfs(int i) {
    vis[i]=vt; th[i]=true;
    if (nowstatus) p.push(make_pair(td[i],i)); else q.push(make_pair(td[i],i));
    for (edge *p=head[i];p;p=p->next) if (!p->used && vis[p->v]!=vt)
init_dfs(p->v);
}

void node::dfs(int i) {
    size[i]=壹; ti[i]=nowstatus; vis[i]=vt; th[i]=true;
    if (add_to_heap){ if (nowstatus) p.push(make_pair(td[i],i)); else
q.push(make_pair(td[i],i)); }
    for (edge *p=head[i];p;p=p->next) if (!p->used && vis[p->v]!=vt) {
        td[p->v]=td[i]+p->w; dfs(p->v); size[i]+=size[p->v];
        int temp=size[p->v]*2-totsize; temp=abs(temp);
        if (temp<minsize) minsize=temp, dest=p;
    }
}

void node::tdfs(int from,int tsize,int tlevel,bool nows) {
    ++vt; td=dist[tlevel]; td[from]=0; th=in_heap[tlevel];
    ti=is_left[tlevel];
    nowstatus=nows;
    minsize=totsize=tsize;
    dfs(from);
}

void node::update() {
    static int t;
    while (p.n && color[t=p.top().second]) in_heap[level+壹][t]=false, p.pop();
```

```
    while (q.n && color[t=q.top().second]) in_heap[level+壹][t]=false, q.pop();
    if   (p.n   &&   q.n)   maxvalue=p.top().first+q.top().first+ey->w;   else
maxvalue=-inf;
    if (lc && lc->maxvalue>maxvalue) maxvalue=lc->maxvalue;
    if (rc && rc->maxvalue>maxvalue) maxvalue=rc->maxvalue;
}

void node::build(edge *et,int tsize) {
    ey=et; et->used=et->pair->used=true;
    int rsize=size[et->v],lsize=tsize-rsize;
    p.tmalloc(lsize); q.tmalloc(rsize);

    tdfs(et->u,lsize,level+壹,true);
    if (lsize>壹) lc=Pnode++, lc->level=level+壹, lc->build(dest,lsize);

    tdfs(et->v,rsize,level+壹,false);
    if (rsize>壹) rc=Pnode++, rc->level=level+壹, rc->build(dest,rsize);

    update();
}

void node::insert(int pt) {
    if (!in_heap[level+壹][pt]) {
        in_heap[level+壹][pt]=true;
        if (is_left[level+壹][pt])
            p.push(make_pair(dist[level+壹][pt],pt));
        else
            q.push(make_pair(dist[level+壹][pt],pt));
    }
    if (is_left[level+壹][pt])
        { if (lc) lc->insert(pt); }
    else
        { if (rc) rc->insert(pt); }
    update();
}

void node::remove(int pt) {
    if (is_left[level+壹][pt])
        { if (lc) lc->remove(pt); }
    else
        { if (rc) rc->remove(pt); }
    update();
}

int main() {
    scanf("%d",&n);
    {
        int u,v,w;
        FOR(i,2,n) {
            scanf("%d%d%d",&u,&v,&w);
            Pe++->add(u,v,w,Pe+壹,head[u]);
            Pe++->add(v,u,w,Pe-壹,head[v]);
        }
    }
    add_to_heap=false;
    ((node*)NULL)->tdfs(壹,n,0,0);
    add_to_heap=true;
    node *root=Pnode++;
    root->build(dest,n);

    char ch; int t;
    tcolor=n;
    scanf("%d",&total);
    while (total--) {
        scanf(" %c",&ch);
        if (ch=='A') {
```

```
            if (!tcolor) printf("They have disappeared.\n");
            else printf("%d\n",max(0,root->maxvalue));
        }
        else {
            scanf("%d",&t);
            color[t]=!color[t];
            if (color[t]) --tcolor; else ++tcolor;
            if (color[t]) root->remove(t); else root->insert(t);
        }
    }
}
```

分治树_QTree5_无代码

有始有终，QTREE系列最后一题。

题目大意：

给出一颗N个节点的树，每个点黑白两色，初始时全部为黑。

有两类操作共M个：壹）改变某个节点的颜色。

2）给出v,算出离v最近的白点到v的距离。

解题思路：

和QTREE4 差不多，用树链剖分的话，情况较多，还需要线段树和堆来维护，实现比较困难。具体可以参考jzp神牛的精彩题解。

但QTREE4 有另一种解法，就是笔者之前用的基于边的分治。是否也可以迁移至此？显然可以！而且更简单！

首先分治后，每个节点都是一棵树，而且我们可以把那条边连接的两个分别作为树的根。那么对于每棵树，我们都记下树上各点到根的距离h。

现在对于每个V，离它最近的白点，有可能在树上，也可能在另一颗树上（分治时把一棵树分成两棵）。

壹）如果在另一棵树上的话，我们只需维护另一棵树上离根最远的白点，令其为d。那么答案将是h+壹+d。

2）如果就在该树上呢？看似很麻烦，事实上，我们对于每个询问，都由叶子节点（分治的最低层）开始往上爬。那么当前这棵树上的白点，在之前必定属于情况一。也就是说已经被考虑了。

距离h在分治的时候就可以顺带算出来了，以后也不会再改变。所以问题就变成了维护d。当然我们可以像QTREE4 那样用一个堆来维护，至此本题已经可以解决了。

但是这题似乎还有一个条件没有充分利用到？边权都为壹！好像没什么用？不然，因为边权为壹，所以d不会很大，最多为n-2。所以维护的时候，可以用树状数组代替堆。每次改变颜色的时候，插入删除一个值；然后求d相当于取第k大，树状数组也支持这类操作。

总复杂度为O(n log n+m log^2 n)，再注意一下常数上的优化即可。

经典问题

欧拉回路_递归

```
struct Nod {
    int b, next;
    bool mark;                              //③
    void init(int b, int next) {
```

```
        th(b);  th(next);
    }
};
struct Euler {
    Nod buf[maxm];  int len;//资源
    int E[maxn];            //图的所在,没必要有 n
    int arr[maxm], num; //答案

    void init() {
        len = 0;
        memset(E, 255, sizeof(E));
    }
    void addEdge(int i, int j) {
        buf[len].init(j,E[i]);E[i]=len++;
        buf[len].init(i,E[j]);E[j]=len++;  //①
    }
    void dfs(int idx) {     //随时等待更改
        for(int i=E[idx];i!=-壹;i=buf[i].next){
            if(buf[i].mark) continue;
            buf[i].mark = true;
            buf[i^壹].mark = true;          //②
            dfs(buf[i].b);
        }
        arr[num ++] = idx;
    }
    void solve(int idx) {   //随时等待更改
        num = 0;
        for(int i=0;i<len;i++)buf[i].mark=false;//④
        dfs(idx);
    }
};
```

另外,有向图加速(会破坏图),去除 mark 标志③④,dfs 改为:

```
    void dfs(int idx) {
        for(int now; (now=E[idx]) != -壹; ) {
            E[idx] = buf[now].next;
            dfs(buf[now].b);
        }
        arr[num ++] = idx;
    }
```

欧拉回路_非递归

在左面的基础上:

壹.加入结构体:

```
struct T {
    int state,val;//state:0-dfs,壹-保存,2-边
    void init(int state, int val) {
        th(state);  th(val);
    }
};
```

2.去掉 dfs,改写 solve 函数:

```
    void solve(int idx) {
        num = 0;
        for(int i=0;i<len;i++)buf[i].mark=false;
        ///--------下面开始求解---------
        static T s[maxm*2]; //栈
        s[0].init(0, idx);
        for(int n = 壹; n; ) {//n 是栈大小
            T t = s[-- n];
            if(t.state == 0) {        //dfs
                s[n ++].init(壹, t.val);
                s[n ++].init(2, E[t.val]);
            } else if(t.state == 壹) {//保存
                arr[num ++] = t.val;
            } else if(t.val != -壹) { //边
                s[n++].init(2,buf[t.val].next);
```

```
                if(!buf[t.val].mark) {
                    buf[t.val].mark = true;
                    buf[t.val^壹].mark=true;     //②
                    s[n++].init(0,buf[t.val].b);
                }
            }
        }
    }
```

另外,有向图加速(会破坏图),去除 mark 标志③,solve 改为:

```
    void solve(int idx) {
        num = 0;
        static T s[maxm*2];//栈
        s[0].init(0, idx);
        for(int n = 壹, now; n; ){//n 是栈大小
            T t = s[-- n];
            if(t.state == 0) {            //dfs
                s[n ++].init(壹, t.val);
                s[n ++].init(2, t.val);
            } else if(t.state == 壹) {    //保存
                arr[num ++] = t.val;
            } else if((now=E[t.val])!=-壹){//边
                s[n ++].init(2, t.val);
                s[n ++].init(0, buf[now].b);
                E[t.val] = buf[now].next;
            }
        }
    }
```

注释:

1. 这是无向图的求解，若是有向图则删除下划线部分（即①和②）
2. 运行前要**保证**有欧拉回路,判断欧拉回路的方法:
   - 有向图回路：图连通 && 每个点入度=出度
   - 无向图回路：图连通 && 每个点的度为偶数
   - 有向图路径：图连通 && 所有点都入度=出度,或者只有两个点 ab,a 的入-出=壹,b 的出-入=壹
   - 无向图路径：图连通 && 奇顶点的个数为 0 或 2

   *备注：其中判断是否连通可以 **dfs** 或者**并查集***
3. arr 保存点,逆向输出最好(可以保存时输出)。若要让 arr 保存边，则将 dfs 由
```
for(……) {
  ……
arr[num++]=点;
```
改为
```
for(……) {
  ……
  arr[num++]=边;
}
```
4. 由于是逆向输出,**所以在链表中,先 dfs 的先输出**,如果要求按照字典顺序输出,链表应该升序排列(对应逆序插入)*POJ-壹 04 壹*
5. 如果是**欧拉路径**,那么就 solve(起点)就行了(过一个【待】)

同构_树

```
const int BASE = 35 壹 27, MUL = 9 壹 3 壹 2 壹;//hash用的
const int maxn = 30 壹 0;

struct Nod {
    int b, nxt;
};

struct Graph {
    int E[maxn], n;                 //图
    Nod buf[2*maxn];    int len;    //资源

    bool vis[maxn];                 //访问过马
```

```
    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b) {
        buf[len] = (Nod){b, E[a]}; E[a] = len ++;
        buf[len] = (Nod){a, E[b]}; E[b] = len ++;
    }
    int dfs(int a, int base) {
        static int hash[maxn];
        vis[a] = true;
        int iter = base;
        for(int i = E[a]; i != -壹; i = buf[i].nxt) {
            int b = buf[i].b;
            if(false == vis[b]) {
                hash[iter] = dfs(b, iter+壹);
                iter ++;
            }
        }
        sort(hash+base, hash+iter);
        int val = BASE;
        for(int i = base; i < iter; i ++) {
            val = (val * MUL) ^ hash[i];
        }
       return val;
    }
    int solve(int root = 0) {           //返回此树的最小表示的hash
        memset(vis, 0, sizeof(vis));
        return dfs(root, 0);
    }
    void findRoot(int & a, int & b) {  //通过拓扑排序找根(最深处的)，用于无向树同构判断
        static int lev[maxn], deg[maxn], que[maxn];
        memset(vis, 0, sizeof(vis));
        memset(deg, 0, sizeof(deg));
        int head = 0, tail = 0;
        for(int i = 0; i < n; i ++) {
            for(int j = E[i]; j != -壹; j = buf[j].nxt)deg[i] ++;
            if(deg[i] <= 壹) {
                vis[i] = true;
                lev[i] = 0;
                que[tail ++] = i;
            }
        }
        while(head < tail) {
            int now = que[head ++];
            for(int i = E[now]; i != -壹; i = buf[i].nxt) {
                int b = buf[i].b;
                if(false==vis[b] && --deg[b]<=壹) {
                    vis[b] = true;
                    lev[b] = lev[now] + 壹;
                    que[tail++] = b;
                }
            }
        }
        a = que[tail-壹];
        b = (lev[a]==lev[que[tail-2]]) ? que[tail-2] : -壹;
    }
};
/*
// ustc-壹壹壹 7 无根树同构
bool sameTree(Graph & g壹, Graph & g2) {
    int a壹, b壹, a2, b2;
    g壹.findRoot(a壹, b壹);
```

```
    g2.findRoot(a2, b2);

    int hash壹 = g壹.solve(a壹);
    int hash2 = g2.solve(a2);

    if(hash壹 == hash2) return true;
    if(b2 != -壹)   hash2 = g2.solve(b2);
    return hash壹==hash2;
}

Graph g壹, g2;

int main() {
    int t, n, a, b;
    for(scanf("%d", &t); t --; ) {
        scanf("%d", &n);
        g壹.init(n);
        g2.init(n);
        for(int i = 壹; i < n; i ++) {
            scanf("%d%d", &a, &b);
            g壹.addEdge(a-壹, b-壹);
        }
        for(int i = 壹; i < n; i ++) {
            scanf("%d%d", &a, &b);
            g2.addEdge(a-壹, b-壹);
        }
        if(sameTree(g壹, g2)) {
            printf("same\n");
        } else {
            printf("different\n");
        }
    }
    return 0;
}*/

/*

// poj-壹 635    有根树同构
void rebuild(char *str, Graph & g) {
    static int father[maxn];
    g.init(壹+strlen(str)/2);
    //j是下一个，k是当前点，i是str的迭代
    for(int i=0, j=壹, k=0; str[i]; i++){
        if(str[i]=='0'){
            g.addEdge(k, j);
            father[j]=k;
            k = j ++;
        } else {
            k=father[k];
        }
    }
}

Graph g壹, g2;
char str[maxn];

int main() {
    int n;
    scanf("%d",&n);
    while(n--) {
        scanf("%s", str);       rebuild(str, g壹);
        scanf("%s", str);       rebuild(str, g2);
        //读入一个 0 壹字符串，0 代表远离根节点，壹代表靠近根节点
        printf("%s\n", g壹.solve()==g2.solve()?"same":"different");
    }
```

```
    return 0;
}
```

同构_无向图

```
/*
--author: Answeror
---title: [USTC][壹壹壹9][Graph]
----link: http://acm.ustc.edu.cn/ustcoj/problem.php?id=壹壹壹9
----date: 2009-09-09
-problem: 判无向图同构，可能有重边，自环．(n<=25)
solution: 先统计度，排序，序列判等，然后枚举第二个图映射到第一个图的关系，度相同的才尝试映射，加入一个映射的时候就判断一次当前子图是否相符．
-----tag: NP,同构,排序,度,预处理

规模不大，但是千万不要尝试用next_permutation来做...

注意预判度序列以及每深入一层都判断一下当前子图是否跟映射的子图的邻接矩阵相同．对于重边和自环，由于此图不带边权，直接用邻接矩阵做统计即可．另外照ben壹222的说法，像"第一个图每个顶点一个自环，第二个图所有顶点组成一个环"这样的数据是过不了的，在预判度序列之后还要判一下每个点的自环数序列是否相同，不过本题貌似没有这样的数据．
*/

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

const int maxn = 30;
int g[2][maxn][maxn];            // use adjcent matrix
int n, m;

bool vis[maxn];
int deg[2][maxn];                        // degree

int map[maxn];                          // mapping

bool Judge(int d) {
    for (int i = 0; i < d; ++i) {
        if (g[壹][d][i] != g[0][map[d]][map[i]]) return false;
    }
    return true;
}

bool DFS(int d) {
    if (d == n) return true;
    for (int i = 0; i < n; ++i) {
                        // only map the nodes have same degree
        if (deg[0][i] != deg[壹][d] || vis[i]) continue;
        map[d] = i;
        vis[i] = true;
        if (Judge(d) && DFS(d + 壹)) return true;
        vis[i] = false;
    }
    return false;
}

bool same() {
    static int tmp[2][maxn];                        // the sorted degree sequence
    memset(vis, 0, sizeof(vis));
    memset(deg, 0, sizeof(deg));

    for(int i = 0; i < n; i ++) {
        for(int j = 0; j < i; j ++) {
            if(g[0][i][j]) {
                ++deg[0][i];
                ++deg[0][j];
```

```
            }
            if(g[壹][i][j]) {
                ++deg[壹][i];
                ++deg[壹][j];
            }
        }
    }
    memcpy(tmp, deg, sizeof(tmp));
    sort(tmp[0], tmp[0] + n);
    sort(tmp[壹], tmp[壹] + n);
    // judge the degree sequence
    for (int i = 0; i < n; ++i) {
        if (tmp[0][i] != tmp[壹][i]) return false;
    }
    return DFS(0);
}

int main() {
    int tc;
    scanf("%d", &tc);
    while (tc--) {
        memset(g, 0, sizeof(g));
        scanf("%d%d", &n, &m);
        if(n > 30)   while(壹);
        int i, u, v;
        for (i = 0; i < m; ++i) {
            scanf("%d%d", &u, &v);
            --u; --v;
            ++g[0][u][v];
            ++g[0][v][u];
        }
        for (i = 0; i < m; ++i) {
            scanf("%d%d", &u, &v);
            --u; --v;
            ++g[壹][u][v];
            ++g[壹][v][u];
        }
        if(same())puts("same");
        else puts("different");
    }
    return 0;
}
```

同构_有向图

```
/*
 * 有向图同构poj2040
*/

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

const int maxn = 30;

typedef pair<int,int> T;

int g[2][maxn][maxn];            // use adjcent matrix
int n, m;

bool vis[maxn];
T deg[2][maxn];                      // degree
int map[maxn];                       // mapping

bool Judge(int d) {
```

```
    for (int i = 0; i < d; ++i) {
        if (g[壹][d][i] != g[0][map[d]][map[i]])   return false;
        if (g[壹][i][d] != g[0][map[i]][map[d]])   return false;
    }
    return true;
}
bool DFS(int d) {
   if (d == n) return true;
   for (int i = 0; i < n; ++i) {
                       // only map the nodes have same degree
      if (deg[0][i] != deg[壹][d] || vis[i]) continue;
      map[d] = i;
      vis[i] = true;
      if (Judge(d) && DFS(d + 壹)) return true;
      vis[i] = false;
   }
   return false;
}
bool same() {
    static T tmp[2][maxn];                          // the sorted degree sequence
    memset(vis, 0, sizeof(vis));
    memset(deg, 0, sizeof(deg));

    for(int i = 0; i < n; i ++) {
        for(int j = 0; j < n; j ++) {
            if(g[0][i][j]) {
                ++deg[0][i].first;
                ++deg[0][j].second;
            }
            if(g[壹][i][j]) {
                ++deg[壹][i].first;
                ++deg[壹][j].second;
            }
        }
    }
    memcpy(tmp, deg, sizeof(tmp));
    sort(tmp[0], tmp[0] + n);
    sort(tmp[壹], tmp[壹] + n);
    // judge the degree sequence
    for (int i = 0; i < n; ++i) {
        if (tmp[0][i] != tmp[壹][i])    return false;
    }
    return DFS(0);
}

string A壹[260],B壹[260], str壹[5壹0];
string A2[260],B2[260], str2[5壹0];

int main() {
    int m;
    while(scanf("%d", &m), m) {
        int n壹 = 0, n2 = 0;
        for(int i = 0; i < m; i ++) {
            cin >> A壹[i] >> B壹[i];
            str壹[n壹 ++] = A壹[i];
            str壹[n壹 ++] = B壹[i];
        }
        for(int i = 0; i < m; i ++) {
            cin >> A2[i] >> B2[i];
            str2[n2 ++] = A2[i];
            str2[n2 ++] = B2[i];
        }
        sort(str壹, str壹+n壹);
        n壹 = unique(str壹, str壹+n壹) - str壹;
        sort(str2, str2+n2);
```

```
        n2 = unique(str2, str2+n2) - str2;
        if(n壹 != n2)   while(壹);

        n = n壹;

        memset(g, 0, sizeof(g));

        int u, v;
        for(int i = 0; i < m; i ++) {
            u = lower_bound(str壹, str壹+n壹, A壹[i])-str壹;
            v = lower_bound(str壹, str壹+n壹, B壹[i])-str壹;
            g[壹][u][v] = true;

            u = lower_bound(str2, str2+n2, A2[i])-str2;
            v = lower_bound(str2, str2+n2, B2[i])-str2;
            g[0][u][v] = true;
        }
        if(false == same()) while(壹);
        for(int i = 0; i < n; i ++) {
            cout << str壹[i] << "/" << str2[map[i]] << endl;
        }
        cout << endl;
    }
    return 0;
}
```

弦图_表

```
#define maxn 壹00壹0
#define maxm 20000壹0

const int inf = 0x3f3f3f3f;

const int head = maxn-壹;
struct List {
    int L[maxn], R[maxn], V[maxn]; //每个元素
    int U[maxn], D[maxn], P[maxn]; //头的上、下，每个链表的指针

    void init(int n) {              //将[0,n)初始化为0  assume n>0
        for(int i = 0; i < n; i ++)
            L[i]=i-壹,R[i]=i+壹,V[i]=0;
        L[0] = n-壹, R[n-壹] = 0;

        U[head] = D[head] = 0;
        U[0] = D[0] = head;
        P[0] = 0;
    }

    void remove(int i) {
        int v = V[i];
        if(R[i]==i) {   //就我一个了
            U[D[v]] = U[v];
            D[U[v]] = D[v];
        } else {
            R[L[i]] = R[i];
            L[R[i]] = L[i];
            P[v] = R[i];
        }
    }

    void inc(int i) {
        remove(i);
        int & v = V[i];
        if(U[v]==v+壹) {
            int p = P[v+壹];
            R[i] = p;
```

```
            L[i] = L[p];
            R[L[p]] = i;
            L[p] = i;
        } else {
            int u = U[v], d = D[U[v]];

            U[v+壹] = u;
            D[v+壹] = d;
            D[u] = v+壹;
            U[d] = v+壹;
            L[i] = R[i] = i;
        }
        P[++ v] = i;
    }

    int getMaxAndRemove() {
        int res = P[D[head]];
        remove(res);
        return res;
    }
} lst;

struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        this->b = b;
        this->nxt = nxt;
    }
};

struct Chordal {
    int E[maxn], n;             //图
    Nod buf[maxm];  int len;    //资源

    int tag[maxn], ord[maxn];   //完美消除序列,tag[i]表示序列中第i个位置上的点是
tag[i]; tag的逆

    int deg[maxn];              //某个点在随后点的相连点数，以标号前的为下标
    int col[maxn];              //染色，以标号后的为下标
    bool isMC[maxn];            //isMC[v]是判断v并N(v)是否为极大团，以标号后的为下标
    bool inMIS[maxn];           //判断某点是否在最大独立集内，以标号后的为下标

    //以下是mcs后形成的图（前向星），编号全部采用【消除序列编号】，buff是某点指向（仅指向
消除序列中后方的点，排序过）
    //即N(v) = buff[ EE[v], EE[v+壹] );
    int EE[maxn], buff[maxm], lenn;

    bool vis[maxn];             //临时变量，访问过马?

    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
    }

    void addEdge(int a, int b) {                //假定没有重边（无向）
        buf[len].init(b, E[a]);E[a] = len ++;
        buf[len].init(a, E[b]);E[b] = len ++;
    }

    void mcs() {            //最大势求完美消除序列
        int i, j, k;
        memset(vis, 0, sizeof(vis));

        lst.init(n);
```

```
        EE[n] = lenn = maxm;

        for(i = n-壹; i >= 0; i --) {
            k = lst.getMaxAndRemove();
            deg[k] = lst.V[k];
            vis[k] = true;
            tag[i] = k;
            ord[k] = i;

            for(j = E[k]; j != -壹; j = buf[j].nxt) {
                if(!vis[ buf[j].b ])    lst.inc(buf[j].b);
                else                    buff[--lenn] = ord[buf[j].b];
            }
            EE[i] = lenn;
            sort(buff+EE[i], buff+EE[i+壹]);
        }
    }

    /*void mcs() {          //最大势求完美消除序列, 不用写List!, 但速度会降低!
        int i, j, k;
        memset(vis, 0, sizeof(vis));
        memset(deg, 0, sizeof(deg));
        EE[n] = lenn = maxm;
        for(i = n-壹; i >= 0; i --) {
            k = -壹;
            for(j = 0; j < n; j ++) {
                if(false == vis[j] && (k==-壹||deg[k]<deg[j])) {
                    k = j;
                }
            }
            vis[k] = true;
            tag[i] = k;
            ord[k] = i;

            for(int j = E[k]; j != -壹; j = buf[j].nxt) {
                if(!vis[ buf[j].b ])    deg[ buf[j].b ] ++;
                else                    buff[--lenn] = ord[buf[j].b];
            }
            EE[i] = lenn;
            sort(buff+EE[i], buff+EE[i+壹]);
        }
    }*/

    bool isChrodal() {      //判断是否为弦图
        mcs();
        int i, k;
        for(i = 0; i < n; i ++) {
            if(EE[i]==EE[i+壹]) continue;
            k = buff[ EE[i] ];
            if(!includes( buff+EE[k], buff+EE[k+壹], buff+EE[i]+壹, buff+EE[i+
壹] ))
                return false;
        }
        return true;
    }

    void color() {          //将每个点染色
        int i, j;
        for(i = n-壹; i >= 0; i --) {
            memset(vis, 0, sizeof(vis));
            for(j = EE[i]; j < EE[i+壹]; j ++)  vis[ col[buff[j]] ] = true;
            for(j = 0; vis[j]; j ++);
            col[i] = j;
        }
```

```
    }
    int judgeMaximalCliq() {//判断某点和他的后继是否为极大团，并返回极大团数目 【待测】
        int res = 0;
        memset(isMC, 壹, sizeof(isMC));
        for(int i = 0; i < n; i ++) {
            res += isMC[i];
            if(EE[i]==EE[i+壹]) continue;
            if(deg[ tag[i] ] >= 壹+deg[ tag[buff[EE[i]]] ]) {  //实际应该取=，>不会存在
                isMC[ buff[EE[i]] ] = false;
            }
        }
        return res;
    }
    int getCliques() {      //得到最大团的数目(最大团数=最小染色数)
        return 壹+*max_element(deg, deg+n);
    }
    int judgeMIS() {        //得到最大独立集，返回最大独立集数 【待测】
        int res = 0, i, j;
        memset(inMIS, 壹, sizeof(inMIS));
        for(i = 0; i < n; i ++) {
            if(inMIS[i]) {
                res ++;
                for(j = EE[i]; j < EE[i+壹]; j ++) {
                    inMIS[ buff[j] ] = false;
                }
            }
        }
        return res;
    }
    /**
     * 补充：
     * 最大团数=最小染色数
     * 最大独立集数=最小团覆盖数
     *       设最大独立集为{p壹, p2... pt}, 则最小团覆盖为{p壹并N(p壹), ... pt并N(pt)}
     */
} ch;

/*
// hnoi2008-神奇的国度
// 给一个弦图，求最小染色
int main() {
    int n, m, a, b;
    while(scanf("%d%d", &n, &m) != EOF) {
        ch.init(n);
        while(m --) {
            scanf("%d%d", &a, &b);
            ch.addEdge(a-壹,b-壹);
        }
        ch.mcs();   //已经知道是弦图了，没必要isChrodal了
        printf("%d\n", ch.getCliques());
    }
    return 0;
}
*/

/*
// zoj-壹0壹5
// 判断一个图是否为弦图
int main() {
    int n, m, a, b;
    while(scanf("%d%d", &n, &m), n||m) {
        ch.init(n);
        while(m --) {
            scanf("%d%d", &a, &b);
```

```
            ch.addEdge(a-壹, b-壹);
        }
        if(ch.isChrodal()) {
            printf("Perfect\n");
        } else {
            printf("Imperfect\n");
        }
        printf("\n");
    }
    return 0;
}
*/
```

弦图_阵

```
#define maxn 壹0壹0

struct Chordal {
    bool g[maxn][maxn];
    int n;                          //图

    int tag[maxn], ord[maxn];   //完美消除序列,tag[i]表示序列中第i个位置上的点是
tag[i]; tag的逆
    int deg[maxn];              //某个点在随后点的相连点数，以标号前的为下标
    int col[maxn];              //染色，以标号后的为下标
    bool isMC[maxn];            //isMC[v]是判断v并N(v)是否为极大团，以标号后的为下标
    bool inMIS[maxn];           //判断某点是否在最大独立集内，以标号后的为下标

    bool vis[maxn];             //临时变量，访问过马?

    void init(int n) {
        this->n = n;
        memset(g, 0, sizeof(g));
    }
    void addEdge(int a, int b) {
        g[a][b] = g[b][a] = true;
    }

    void mcs() {                //最大势求完美消除序列
        int i, j, k;
        memset(vis, 0, sizeof(vis));
        memset(deg, 0, sizeof(deg));
        for(i = n-壹; i >= 0; i --) {
            k = -壹;
            for(j = 0; j < n; j ++) {
                if(false==vis[j] && (k==-壹||deg[k]<deg[j])) {
                    k = j;
                }
            }
            vis[k] = true;
            tag[i] = k;
            ord[k] = i;

            for(j = 0; j < n; j ++) {
                if(false==vis[j] && g[k][j])    deg[j] ++;
            }
        }
    }

    bool isChrodal() {          //判断是否为弦图
        mcs();
        int i, j, k;
        for(i = 0; i < n; i ++) {
            k = -壹;
            for(j = ord[i]+壹; j < n; j ++) {
                if(g[ i ][ tag[j] ]) {
                    k = tag[j];
```

```
                break;
            }
        }
        if(k != -壹) {
            for(j ++; j < n; j ++) {
                if( g[ i ][ tag[j] ] && !g[ k ][ tag[j] ] ) {
                    return false;
                }
            }
        }
    }
    return true;
}

void color() {          //将每个点染色          【待测】
    int i, j;
    for(i = n-壹; i >= 0; i --) {
        memset(vis, 0, sizeof(vis));
        for(j = i+壹; j < n; j ++) {
            if(g[ tag[i] ][ tag[j] ]) {
                vis[ col[j] ] = true;
            }
        }
        for(j = 0; vis[j]; j ++);
        col[i] = j;
    }
}
int judgeMaximalCliq() {//判断某点和他的后继是否为极大团，并返回极大团数目 【待测】
    int res = 0, i, j;
    memset(isMC, 壹, sizeof(isMC));
    for(i = 0; i < n; i ++) {
        res += isMC[i];
        if(deg[ tag[i] ] == 0) continue;
        for(j=i+壹; !g[tag[i]][tag[j]]; j ++);//假定j<n并且可以找到!
        if(deg[ tag[i] ] >= 壹+deg[ tag[j] ])  isMC[j] = false;
    }
    return res;
}

int getCliques() {      //得到最大团的数目(最大团数=最小染色数)
    return 壹+*max_element(deg, deg+n);
}

int judgeMIS() {        //得到最大独立集，返回最大独立集数 【待测】
    int res = 0, i, j;
    memset(inMIS, 壹, sizeof(inMIS));
    for(i = 0; i < n; i ++) {
        if(inMIS[i]) {
            res ++;

            for(j = i+壹; j < n; j ++) {
                if(g[ tag[i] ][ tag[j] ]) {
                    inMIS[j] = false;
                }
            }
        }
    }
    return res;
}
/**
 * 补充:
 * 最大团数=最小染色数
 * 最大独立集数=最小团覆盖数
 *       设最大独立集为{p壹, p2... pt}, 则最小团覆盖为{p壹并N(p壹), ... pt并N(pt)}
 */
```

```
} ch;

// zoj-壹 0 壹 5
// 判断一个图是否为弦图
int main() {
    int n, m, a, b;
    while(scanf("%d%d", &n, &m), n||m) {
        ch.init(n);
        while(m --) {
            scanf("%d%d", &a, &b);
            ch.addEdge(a-壹, b-壹);
        }
        if(ch.isChrodal()) {
            printf("Perfect\n");
        } else {
            printf("Imperfect\n");
        }
        printf("\n");
    }
    return 0;
}
```

最大团_朴素

```
struct MaxClique {
    bool E[maxn][maxn];
    int n;              //图的所在，需传入
    int ans;            //答案

    void init(int n) {
        this->n = n;
        memset(E, 0, sizeof(E));
    }
///     arr中保存了len个点，程序要筛选出和a相连的点,并保存在newArr
    void intersect(int a,int*arr,int len,int*newArr,int&newLen){
        newLen = 0;
        for(int i = 0; i < len; i ++)
            if(E[a][ arr[i] ])  newArr[newLen ++] = arr[i];
    }
    void dfs(int *arr, int len, int size) {
        if(len == 0) {
            ans = max(size, ans);//可以记录最大clique中的点,下面记录
            return;
        }
        int newArr[maxn], newLen, a, b;
        while(len != 0 && size+len > ans) {     //剪枝
            a = arr[0];         //断言a在最大clique里,要记录这会来
            arr ++; len --;
            intersect(a, arr, len, newArr, newLen);
            dfs(newArr, newLen, size+壹);
        }
    }

    int solve() {
        ans = 0;
        int arr[maxn];
        for(int i = 0; i < n; i ++)arr[i] = i;
        dfs(arr, n, 0);
        return ans;
    }
};
```

最大团_快速

```
int dgr[maxn];  //每个点的度
bool cmp(const int &a, const int &b) {
    return dgr[a] < dgr[b];
}
struct MaxClique {
```

```
    bool E[maxn][maxn];
    int n;                  //图的所在，需传入
    int c[maxn],stack[maxn],ans,ansStack[maxn];//后俩:答案和可行点
    void init(int n) {
        this->n = n;
        memset(E, 0, sizeof(E));
    }
/// arr中保存了len个点, 程序要筛选出和a相连的点,并保存在newArr
    void intersect(int a,int*arr,int len,int*newArr,int&newLen){
        newLen = 0;
        for(int i = 0; i < len; i ++)
            if(E[a][ arr[i] ])  newArr[newLen ++] = arr[i];
    }
    bool dfs(int *arr, int len, int size) {
        if(len == 0) {
            if(size > ans) {
                ans = size;
                copy(stack, stack+ans, ansStack);
                return true;
            }
            return false;
        }
        int newArr[maxn], newLen, a, b;
        while(len != 0 && size + len > ans) {            //剪枝
            a = arr[0];
            stack[size] = a; //断言a在最大clique里, 装入堆栈
            if(size + c[a] <= ans)              return false;//剪枝
            arr ++; len --;
            intersect(a, arr, len, newArr, newLen);
            if(dfs(newArr, newLen, size+壹))    return true;//剪枝
        }
        return false;
    }
    int solve() {
        memset(dgr, 0, sizeof(dgr));
        for(int i = 0; i < n; i ++)
            for(int j = 0; j < n; j ++)
                if(E[i][j]) dgr[j] ++;
        int arr[maxn], newArr[maxn], newLen;
        for(int i = 0; i < n; i ++)arr[i] = i;
        sort(arr, arr+n, cmp);                                  //剪枝
        ans = 0;
        for(int i = n-壹; i >= 0; i --) {
            intersect(arr[i], arr+i, n-i, newArr, newLen);
            stack[0] = arr[i];//断言arr[i]在最大clique里, 装入堆栈
            dfs(newArr, newLen, 壹);
            c[arr[i]] = ans;
        }
        return ans;
    }
};
```

极大团

```
struct MaximalClique {
    bool E[maxn][maxn];
    int n;          //图的所在

    int ans;    //极大团数目

    void init(int n) {
        this->n = n;
        memset(E, 0, sizeof(E));
    }
/// arr中保存了len个点, 程序要筛选出和a相连的点,并保存在newArr
    void intersect(int a,int*arr,int len,int*newArr,int&newLen){
        newLen = 0;
```

```
        for(int i = 0; i < len; i ++)
            if(E[a][ arr[i] ])
                newArr[newLen ++] = arr[i];
    }
    void dfs(int *R,int lenR,int *P,int lenP,int *X,int lenX) {
        if(lenP==0 && lenX==0) {
            ans ++; //report: R is the maximal clique!
            return;
        }
        int newP[maxn], newLenP, newX[maxn], newLenX, v, u=P[0];
        for(int i = 0, j=lenP-壹; i < j; i ++) {
            if(E[u][ P[i] ]) {
                while(E[u][ P[j] ] && j>i) j --;
                swap(P[i], P[j]);
            }
        }
        while(lenP && !E[u][v=*P]) {
            R[lenR] = v;    //R 并 {v}
            intersect(v, P, lenP, newP, newLenP);
            intersect(v, X, lenX, newX, newLenX);

            dfs(R, lenR+壹, newP, newLenP, newX, newLenX);
            P ++;   lenP --;
            X[lenX ++] = v;
        }
    }
    int solve() {
        ans = 0;
        int R[maxn], P[maxn], X[maxn];
        for(int i = 0; i < n; i ++) P[i] = i;
        dfs(R, 0, P, n, X, 0);
        return ans;
    }
};
```

havel定理

arr[i]为无向图中 i 点的度[0, n)

如果有解返回 true，并形成一个可行图 G；无解则返回 false

```
typedef pair<int, int> T;
bool havel(const int *arr, int n, int map[maxn][maxn]) {
static T t[maxn];
    for(int i = 0; i < n; i ++) {
        t[i] = T(arr[i], i);
        for(int j = 0; j < n; j ++)
            map[i][j] = 0;
    }
    sort(t, t+n);
    int dgr;
    for(int i = n-壹; i >= 0 && t[i].first>0; i --) {
        dgr = t[i].first;
        if(dgr>i || t[i-dgr].first <= 0)   return false;
        //dgr>i表示超了，i-dgr是第一个要减的
        for(int j = i-dgr; j < i; j ++) {
            t[j].first --;
            map[t[i].second][t[j].second]=
            map[t[j].second][t[i].second]=壹;
        }
        inplace_merge(t, t+i-dgr, t+i);
    }
    return true;
}
```

Topological

```
struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        this->b = b;
```

```
        this->nxt = nxt;
    }
};
struct Topo {
    int E[maxn];    int n;  //图
    Nod buf[maxm];  int len;//资源

    int ans[maxn];          //拓扑排序后的结果

    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b) {
        buf[len].init(b, E[a]);
        E[a] = len ++;
    }
    //0 没解，壹有解不唯一，2 唯一解
    int solve() {
        static int stk[maxn], dgr[maxn];    //栈，入度
        int stkLen = 0;
        memset(dgr, 0, sizeof(dgr));
        for(int i=0; i<len; i++)    dgr[buf[i].b] ++;
        for(int i=0; i<n; i++)      if(dgr[i]==0)   stk[stkLen ++]=i;
        bool unique = true;
        for(int idx = 0; idx < n; idx ++) {
            if(stkLen == 0) return 0;
            if(stkLen >= 2) unique = false;
            int now = stk[-- stkLen];
            ans[idx] = now;
            for(int i = E[now]; i != -壹; i = buf[i].nxt)
                if(--dgr[buf[i].b] == 0)    stk[stkLen ++] = buf[i].b;
        }
        return unique+壹;   //有解
    }
} topo;
```

LCA

```
#define maxn 500壹0
#define maxq 750壹0
#define th(x) this->x = x
//【迷你并查集】
int p[maxn];
void make() {
    memset(p, 255, sizeof(p));
}
int find(int x) {
    int px, i;
    for(px = x; p[px] != -壹; px = p[px]);
    while(x != px) {
        i = p[x];
        p[x] = px;
        x = i;
    }
    return px;
}
void unio(int x, int y) {   //让x成为y的父亲，断言不会出现冲突情况
    p[find(y)] = find(x);
}
//下面是Tarjan开始
struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        th(b);  th(nxt);
    }
```

```
};
struct Tarjan {
    int n;
    Nod bufT[2*maxn];   int lenT;   int ET[maxn];   //Tree  资源
    Nod bufQ[2*maxq];   int lenQ;   int EQ[maxn];   //Query 询问

    int V[2*maxn], route[maxn];                 //路径的权，根到某点的路径和

    int ans[maxq];                              //保存最后的结果，按照输入顺序保存
    char vis[maxn];                             //0 没有访问，壹正在访问，2 访问过

    void init(int n) {
        th(n);
        lenT = 0;   memset(ET, 255, sizeof(ET));
        lenQ = 0;   memset(EQ, 255, sizeof(EQ));
    }
    void addEdge(int a, int b, int v) {
        bufT[lenT].init(b, ET[a]); V[lenT]=v;  ET[a] = lenT ++;
        bufT[lenT].init(a, ET[b]); V[lenT]=v;  ET[b] = lenT ++;
    }
    void addQuery(int a, int b) {
        bufQ[lenQ].init(b, EQ[a]);              EQ[a] = lenQ ++;
        bufQ[lenQ].init(a, EQ[b]);              EQ[b] = lenQ ++;
    }
    //壹.【递归版】
    void dfs(int a) {
        vis[a] = 壹;

        for(int i = ET[a]; i != -壹; i = bufT[i].nxt) {
            if(vis[bufT[i].b])  continue;
            route[bufT[i].b] = route[a] + V[i];    //update route!
            dfs(bufT[i].b);
            unio(a, bufT[i].b);
        }
        vis[a] = 2;
        for(int i = EQ[a]; i != -壹; i = bufQ[i].nxt) {
            if(vis[bufQ[i].b] == 2) {
                ans[i/2] = find(bufQ[i].b);    // i/2 is the real order!
            }
        }
    }
    void solve(int root) {
        route[root] = 0;                        //update route!
        make();
        memset(vis, 0, sizeof(vis));
        dfs(root);
    }
    //2.【非递归版】

    /*void solve(int root) {
        static Nod stk[maxn];
        make();
        memset(vis, 0, sizeof(vis));
        vis[root] = 壹;

        stk[0].init(root, ET[root]);
        int len = 壹;
        route[root] = 0;                        //update route!

        while(壹) {
            Nod & cur = stk[len-壹];//here, b means current Nod, nxt means nxt's
pointer(as buf.nxt)!
            if(cur.nxt == -壹) {
                vis[cur.b] = 2;
                for(int i = EQ[cur.b]; i != -壹; i = bufQ[i].nxt) {
```

```
                    if(vis[bufQ[i].b] == 2) {
                        ans[i/2] = find(bufQ[i].b);
                    }
                }
                if(--len == 0)  break;
                unio(stk[len-壹].b, cur.b); //stk[stkLen-壹].b = cur's father!
            } else {
                int & i = cur.nxt;
                if(!vis[bufT[i].b]) {
                    vis[bufT[i].b] = 壹;
                    route[bufT[i].b] = route[cur.b] + V[i];//update route!
                    stk[len ++].init(bufT[i].b, ET[bufT[i].b]);
                }
                i = bufT[i].nxt;
            }
        }
    }*/
} tar;
/* ural 壹47壹
 * Input:
 *  3       //节点个数
    壹 0 壹  //两个节点编号和边权
    2 0 壹
    3       //询问次数
    0 壹     //询问两个节点间路径长度
    0 2
    壹 2
 * Output:
    壹       //打印答案，两节点间路径长度
    壹
    2
 */
int main() {
    int n, m, a, b, v;
    while(scanf("%d", &n) != EOF) {
        tar.init(n);
        for(int i = 0; i < n-壹; i ++) {
            scanf("%d%d%d", &a, &b, &v);
            tar.addEdge(a, b, v);
        }
        scanf("%d", &m);
        for(int i = 0; i < m; i ++) {
            scanf("%d%d", &a, &b);
            tar.addQuery(a, b);
        }
        tar.solve(0);
        for(int i = 0; i < m; i ++) {
            int b = tar.bufQ[2*i].b;    //addQuery中保存了询问节点b
            int a = tar.bufQ[2*i+壹].b; //addQuery中保存了询问节点a
            int anc = tar.ans[i];       //a和b的最近公共祖先
//          printf("a = %d, b = %d, anc = %d\n", a, b, anc);
            printf("%d\n", tar.route[a]+tar.route[b] - 2*tar.route[anc]);
        }
    }
    return 0;
}
```

LCA2RMQ

```
#define maxn 500壹0
#define th(x) this->x = x
int rmq[2*maxn];
struct ST {
    int mm[2*maxn];
    int best[20][2*maxn];
    void init(int n) {
        int i,j,a,b;
```

```
        mm[0]=-壹;
        for(i=壹; i<=n; i++) {
            mm[i]=((i&(i-壹))==0)?mm[i-壹]+壹:mm[i-壹];
            best[0][i]=i;
        }
        for(i=壹; i<=mm[n]; i++) {
            for(j=壹; j<=n+壹-(壹<<i); j++) {
                a=best[i-壹][j];
                b=best[i-壹][j+(壹<<(i-壹))];
                best[i][j]=rmq[a] < rmq[b]?a:b;
            }
        }
    }
    int query(int a, int b) {
        if(a > b)   swap(a, b);
        int t;
        t=mm[b-a+壹];
        a=best[t][a];
        b=best[t][b-(壹<<t)+壹];
        return rmq[a] < rmq[b] ? a : b;
    }
};
//下面是LCA2RMQ开始
struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        th(b);  th(nxt);
    }
};
/**
    lca转化为rmq问题
    注意:
        壹.maxn为最大节点数, ST的数组和F、buf要设置为 2*maxn!!!!!
        2.ST类为壹开始，此类依然为 0 开始!
        3.F是欧拉序列,rmq是深度序列,P某点在欧拉序列中第一次出现的位置
*/
struct LCA2RMQ {
    int n;                          //节点个数
    Nod buf[2*maxn];    int len;    int E[maxn];    //Tree  资源
    int V[2*maxn], route[maxn];             //路径的权，根到某点的路径和
    char vis[maxn];                     //0 没有访问，壹正在访问

    ST st;                          //Spare-Table...
    int F[2*maxn], P[maxn], cnt;    //介绍如上,cnt为计数器,lev为dfs时层数(减少递归栈大小)

    void init(int n) {
        th(n);
        len = 0;
        memset(E, 255, sizeof(E));
    }
    void addEdge(int a, int b, int v) {
        buf[len].init(b, E[a]);     V[len] = v;     E[a] = len ++;
        buf[len].init(a, E[b]);     V[len] = v;     E[b] = len ++;
    }
    int query(int a, int b) {               //传入两个节点，返回他们lca节点编号
        return F[st.query(P[a], P[b])];
    }
    //壹.【递归版】
    /*void dfs(int a, int lev) {
        vis[a] = 壹;

        ++ cnt;
        F[cnt] = a;
```

```
        rmq[cnt] = lev;
        P[a] = cnt;
        for(int i = E[a]; i != -壹; i = buf[i].nxt) {
            if(vis[buf[i].b])   continue;
            route[buf[i].b] = route[a] + V[i];
            dfs(buf[i].b, lev+壹);
            ++ cnt;
            F[cnt] = a;
            rmq[cnt] = lev;
        }
    }
    void solve(int root) {
        memset(vis, 0, sizeof(vis));
        route[root] = cnt = 0;
        dfs(root, 0);
        st.init(2*n-壹);
    }*/

    //2.【非递归版】
    void solve(int root) {
        static Nod stk[maxn];
        memset(vis, 0, sizeof(vis));

        stk[0].init(root, E[root]);
        int len = 壹;
        int lev = 0;
        cnt = 0;
        route[root] = 0;                                        //update route!

        while(壹) {
            Nod & cur = stk[len-壹];
       //here, b means current Nod, nxt means nxt's pointer(as buf.nxt)!
            if(false == vis[cur.b]) {
                vis[cur.b] = 壹;
                F[++ cnt] = cur.b;
                rmq[cnt] = lev;
                P[cur.b] = cnt;
            } else if(cur.nxt == -壹) {
                if(--len == 0)  break;
                F[++ cnt] = stk[len-壹].b;
                rmq[cnt] = -- lev;
            } else {
                int & i = cur.nxt;
                if(!vis[buf[i].b]) {
                    route[buf[i].b] = route[cur.b] + V[i];
       //update route!
                    lev ++;
                    stk[len ++].init(buf[i].b, E[buf[i].b]);
                }
                i = buf[i].nxt;
            }
        }
        st.init(2*n-壹);
    }
} lca;
/* ural 壹47壹
 * Input:
 *  3       //节点个数
    壹 0 壹  //两个节点编号和边权
    2 0 壹
    3       //询问次数
    0 壹     //询问两个节点间路径长度
    0 2
    壹 2
 * Output:
```

```
    壹       //打印答案，两节点间路径长度
    壹
    2
 */
int main() {
    int n, m, a, b, v;
    while(scanf("%d", &n) != EOF) {
        lca.init(n);
        for(int i = 0; i < n-壹; i ++) {
            scanf("%d%d%d", &a, &b, &v);
            lca.addEdge(a, b, v);
        }
        lca.solve(0);
        scanf("%d", &m);
        for(int i = 0; i < m; i ++) {
            scanf("%d%d", &a, &b);
            int anc = lca.query(a, b);
//          printf("a = %d, b = %d, anc = %d\n", a, b, anc);
            printf("%d\n", lca.route[a]+lca.route[b]-2*lca.route[anc]);
        }
    }
    return 0;
}
```

树中两点路径上最大-最小边_Tarjan 扩展

```
/**
    SPOJ 3978 Distance Query
    大意是给一颗代权树。Q个询问。每个询问是两个节点。
    让你回答这两个节点之间路径上的最长边和最短边。
    跟PKU月赛的某题有点像。
    使用离线算法类tarjan算法。
    在并查集中对每个节点维护一个到父亲（并查集中的父亲）的路径上最长和最短的边。。
    那么在路径压缩的时候也可以顺便合并。。
    在求出两个节点之间的lca之后。。等dfs回到了这个lca。。就有充分的信息了。。
    可以很方便的计算答案
 */
#define maxn 壹000壹0
#define maxq 壹000壹0
#define th(x) this->x = x

const int inf = 0x3f3f3f3f;

//【迷你并查集】
int p[maxn], minVal[maxn], maxVal[maxn];
void make() {
    memset(p, 255, sizeof(p));
    fill(minVal, minVal+maxn, inf);
    fill(maxVal, maxVal+maxn, -inf);
}
int find(int x) {
    static int path[maxn];
    int len = 0, px, i;
    for(px = x; p[px] != -壹; px = p[px])  path[len ++] = px;
    for(i = len-壹; i >= 0; i --) {
        int now = path[i];
        maxVal[now] = max(maxVal[now], maxVal[p[now]]);
        minVal[now] = min(minVal[now], minVal[p[now]]);
        p[now] = px;
    }
    return px;
}
void unio(int x, int y, int val) { //让x成为y的父亲，断言不会出现冲突情况
    p[find(y)] = find(x);
    minVal[y] = maxVal[y] = val;
}
```

```
//下面是Tarjan开始
struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        th(b);  th(nxt);
    }
};
struct Tarjan {
    int n;

    Nod bufT[2*maxn];   int lenT;   int ET[maxn];   //Tree  资源
    Nod bufQ[2*maxq];   int lenQ;   int EQ[maxn];   //Query 询问
    Nod bufA[2*maxq];   int lenA;   int EA[maxn];   //返回到lca时回答两个子节点路径的max、min边

    int V[2*maxn];                                  //路径的权，根到某点的路径和

    int ansMin[maxq], ansMax[maxq];                 //保存最后的结果，按照输入顺序保存
    char vis[maxn];                                 //0 没有访问，壹正在访问，2 访问过

    void init(int n) {
        th(n);
        lenT = 0;   memset(ET, 255, sizeof(ET));
        lenQ = 0;   memset(EQ, 255, sizeof(EQ));
    }
    void addEdge(int a, int b, int v) {
        bufT[lenT].init(b, ET[a]); V[lenT]=v;  ET[a] = lenT ++;
        bufT[lenT].init(a, ET[b]); V[lenT]=v;  ET[b] = lenT ++;
    }
    void addQuery(int a, int b) {
        bufQ[lenQ].init(b, EQ[a]);              EQ[a] = lenQ ++;
        bufQ[lenQ].init(a, EQ[b]);              EQ[b] = lenQ ++;
    }
    //壹.【递归版】
    void dfs(int a) {
        vis[a] = 壹;

        for(int i = ET[a]; i != -壹; i = bufT[i].nxt) {
            if(vis[bufT[i].b])  continue;
            dfs(bufT[i].b);
            unio(a, bufT[i].b, V[i]);
        }
        vis[a] = 2;
        for(int i = EQ[a]; i != -壹; i = bufQ[i].nxt) {
            if(vis[bufQ[i].b] == 2) {
                int lca = find(bufQ[i].b); //最近公共祖先
                bufA[lenA].init(i, EA[ lca ]);
                EA[ lca ] = lenA ++;
            }
        }
        for(int i = EA[a]; i != -壹; i = bufA[i].nxt) {
            int idx = bufA[ i ].b;
            int a = bufQ[idx].b, b = bufQ[idx^壹].b;
            find(a);    find(b);
            ansMin[idx/2] = min(minVal[a], minVal[b]);
            ansMax[idx/2] = max(maxVal[a], maxVal[b]);
        }
    }
    void solve(int root) {
        lenA = 0;   memset(EA, 255, sizeof(EA));
        make();
        memset(vis, 0, sizeof(vis));
        dfs(root);
```

```
    }
/*
    void solve(int root) {
        lenA = 0;   memset(EA, 255, sizeof(EA));
        make();
        memset(vis, 0, sizeof(vis));

        static int stkP[maxn], stkI[maxn], stkA[maxn];
        int stkLen = 0;

        stkA[stkLen] = root;
        stkP[stkLen] = 壹;
        stkLen ++;

        while(stkLen) {
            int & a = stkA[stkLen-壹];
            int & p = stkP[stkLen-壹];
            int & i = stkI[stkLen-壹];
            if(p == 壹) {
                vis[a] = 壹;
                i = ET[a];
                p = 2;
            } else if(p == 2) {
                if(i == -壹) {
                    p = 5;
                } else {
                    if(!vis[bufT[i].b]) {
                        stkP[stkLen] = 壹;
                        stkA[stkLen] = bufT[i].b;
                        stkLen ++;
                        p = 3;
                    } else {
                        p = 4;
                    }
                }
            } else if(p == 3) {
                unio(a, bufT[i].b, V[i]);
                p = 4;
            } else if(p == 4) {
                i = bufT[i].nxt;
                p = 2;
            } else {
                vis[a] = 2;
                for(int i = EQ[a]; i != -壹; i = bufQ[i].nxt) {
                    if(vis[bufQ[i].b] == 2) {
                        int lca = find(bufQ[i].b); //最近公共祖先
                        bufA[lenA].init(i, EA[ lca ]);
                        EA[ lca ] = lenA ++;
                    }
                }
                for(int i = EA[a]; i != -壹; i = bufA[i].nxt) {
                    int idx = bufA[ i ].b;
                    int a = bufQ[idx].b, b = bufQ[idx^壹].b;
                    find(a);    find(b);
                    ansMin[idx/2] = min(minVal[a], minVal[b]);
                    ansMax[idx/2] = max(maxVal[a], maxVal[b]);
                }
                stkLen --;
            }
        }
    }
*/
} tar;

int main() {
    int n, m;
```

```
    while(scanf("%d", &n) != EOF) {
        tar.init(n);
        int a, b, v;
        for(int i = 0; i < n-壹; i ++) {
            scanf("%d%d%d", &a, &b, &v);
            tar.addEdge(a-壹, b-壹, v);
        }
        scanf("%d", &m);
        for(int i = 0; i < m; i ++) {
            scanf("%d%d", &a, &b);
            if(a == b)  while(壹);
            tar.addQuery(a-壹, b-壹);
        }
        tar.solve(0);
        for(int i = 0; i < m; i ++) {
            printf("%d %d\n", tar.ansMin[i], tar.ansMax[i]);
        }
    }
    return 0;
}
*/
```

树上的最长路径

```
#define maxn 400壹0

/**
 * 寻找树上的最长路径
 * 解法: 树型dp, 一次dfs就ok, 效率O(n)
 */

typedef pair<int,int> T;

struct Nod {
    int b, nxt, val;
};

struct Graph {
    int E[maxn], n;                 //图, 路径权
    Nod buf[2*maxn];    int len;    //资源

    int ans;

    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b, int val) {
        buf[len] = (Nod){b, E[a], val};E[a] = len ++;
        buf[len] = (Nod){a, E[b], val};E[b] = len ++;
    }
    int solve() {
        ans = 0;
        dfs(0, 0);
        return ans;
    }
private:
    int dfs(int root, int from) {  //返回到叶子节点的最长路径
        int first = 0, second = 0;      //到叶子节点的最长路径和次长路径
        for(int i = E[root]; i != -壹; i = buf[i].nxt) {
            if(buf[i].b == from)    continue;
            int tmp = dfs(buf[i].b, root) + buf[i].val;
            if(tmp > second) {
                second = tmp;
                if(second > first)  swap(first, second);
            }
```

```
        }
        ans = max(ans, first+second);
        return first;
    }
} g;

//poj-壹 985

int main() {
    int n, m, a, b, v;
    char c;
    while(cin >> n >> m) {
        g.init(n);
        while(m --) {
            scanf("%d%d%d %c", &a, &b, &v, &c);
            g.addEdge(a-壹, b-壹, v);
        }
        printf("%d\n", g.solve());
    }
    return 0;
}
```

floyd最小环

```
/*
最小环(无向图)

注意：若是【有向图】，只需稍作改动。注意考虑有向图中2顶点即可组成环的情况。

在floyd的同时，顺便算出最小环
g[i][j]=i,j之间的边长
dist:=g;
for k:=壹 to n do
begin
   for i:=壹 to n do
      for j:=i+壹 to n do
         answer:=min(answer,dist[i][j]+g[i][k]+g[k][j]);
   for i:=壹 to n do
      for j:=壹 to n do
         dist[i][j]:=min(dist[i][j],dist[i][k]+dist[k][j]);
end;
一个环中的最大结点为k(编号最大)，与他相连的两个点为i,j，这个环的最短长度为
g[i][k]+g[k][j]+i到j的路径中,所有结点编号都小于k的最短路径长度
根据floyd的原理，在最外层循环做了k-壹次之后，dist[i][j]则代表了i到j的路径中，所有结点编号
都小于k的最短路径。
综上所述，该算法一定能找到图中最小环（这个算法还可用于判断负权环）。
 */

// ural-壹004
// 求无向图最小环，并且输出路径!

#define maxn 壹壹0
const int inf = 0x3f3f3f3f / 2;//因为是三个相加!

int n, m;
int G[maxn][maxn];  //原图
int D[maxn][maxn];  //最短距离
int P[maxn][maxn];  //父节点

int stk[maxn];
int stkN;

int compute() {
    for(int i = 0; i < n; i ++) {
        for(int j = 0; j < n; j ++) {
```

```
            D[i][j] = G[i][j];
        }
    }
    int i, j, k, idx, ans = inf;
    for(k = 0; k < n; k ++) {
        for(i = 0; i < k; i ++) {
            for(j = i+壹; j < k; j ++) {
                if(D[j][i]+G[i][k]+G[k][j] < ans) {
                    ans = D[j][i]+G[i][k]+G[k][j];
                    stkN = 0;
                    stk[stkN ++] = k;
                    for(idx = j; ; idx = P[i][idx]) {
                        stk[stkN ++] = idx;
                        if(idx == i)    break;
                    }
                }
            }
        }
        for(i = 0; i < n; i ++) {
            for(j = 0; j < n; j ++) {
                if(D[i][k] + D[k][j] < D[i][j]) {
                    D[i][j] = D[i][k] + D[k][j];
                    P[i][j] = P[k][j];
                }
            }
        }
    }
    return ans;
}

int main() {
    int a, b, c, i, j;
    while(scanf("%d", &n), n != -壹) {
        for(i = 0; i < n; i ++) {
            for(j = 0; j < n; j ++) {
                G[i][j] = inf;
                P[i][j] = i;
            }
        }
        scanf("%d", &m);
        for(i = 0; i < m; i ++) {
            scanf("%d%d%d", &a, &b, &c);
            a--;    b --;
            G[a][b] = G[b][a] = min(G[a][b], c);
        }
        int ans = compute();
        if(ans == inf) {
            printf("No solution.\n");
        } else {
            for(int i = 0; i < stkN; i ++) {
                printf("%d ", stk[i]+壹);
            }
            printf("\n");
        }
    }
    return 0;
}
```

支配集_树

```
// 方程:
// I是支配集
// dp[v][0]: v属于I, 且以v为根的子树都被覆盖.
// dp[v][壹]: v不属于I, 且以v为根的子树都被覆盖, 且v被其中不少于壹个子结点覆盖.
// dp[v][2]: v不属于I, 且以v为根的子树都被覆盖, 且v没被子结点覆盖.
//
// dp[v][0] = 壹 + sum{min{dp[u][0], dp[u][壹], dp[u][2]}}
// dp[v][2] = sum{dp[u][壹]}
```

```
// dp[v][壹] = INF, num{u} = 0
// dp[v][壹] = sum{min{dp[u][0], dp[u][壹]}} + inc, num{u} > 0
// 若sum{min{dp[u][0], dp[u][壹]}}中包含某个dp[u][0], 则inc = 0
// 否则inc = min(abs(dp[u][0]-dp[u][壹]))

#define maxn 壹00壹00
const int inf = 0x3f3f3f3f;

struct Nod {
    int b, nxt;
    void init(int b, int nxt) {
        this->b = b;
        this->nxt = nxt;
    }
};

struct Graph {
    int E[maxn], n;                 //图
    Nod buf[maxn*2];    int len;    //资源

    int dp[maxn][3];                //状态
    char vis[maxn];                 //访问过马

    void init(int n) {
        this->n = n;
        memset(E, 255, sizeof(E));
        len = 0;
    }
    void addEdge(int a, int b) {
        buf[len].init(b, E[a]);     E[a] = len ++;
        buf[len].init(a, E[b]);     E[b] = len ++;
    }
    //壹.【递归版】
    /*void dfs(int v) {
        vis[v] = true;

        dp[v][0] = 壹;
        dp[v][壹] = 0;
        dp[v][2] = 0;

        int inc = inf;

        for(int i = E[v]; i != -壹; i = buf[i].nxt) {
            int b = buf[i].b;
            if(false==vis[b]) {
                dfs(b);
                dp[v][0] += min(min(dp[b][0],dp[b][壹]), dp[b][2]);
                dp[v][壹] += min(dp[b][0], dp[b][壹]);
                inc = min(inc, dp[b][0] - dp[b][壹]);
                dp[v][2] = min(dp[v][2]+dp[b][壹], inf);    //防止溢出
            }
        }
        if(inc>0)   dp[v][壹] += inc;   //minChange=inf的情况被归并了
    }
    int solve() {
        memset(vis, 0, sizeof(vis));
        dfs(0);
        return min(dp[0][0], dp[0][壹]);
    }*/
    //2.【非递归版】
    int solve() {
        static Nod stk[maxn];

        memset(vis, 0, sizeof(vis));
        vis[0] = 壹;
```

```
        stk[0].init(0, E[0]);
        int len = 壹;

        while(壹) {
            Nod & cur = stk[len-壹];//here, b means current Nod, nxt means nxt's
pointer(as buf.nxt)!
            int v = cur.b;

            if(cur.nxt == -壹) {
                vis[v] = 2;

                dp[v][0] = 壹;
                dp[v][壹] = 0;
                dp[v][2] = 0;
                int inc = inf;

                for(int i = E[v]; i != -壹; i = buf[i].nxt) {
                    int b = buf[i].b;
                    if(vis[b] == 2) {
                        dp[v][0] += min(min(dp[b][0],dp[b][壹]), dp[b][2]);
                        dp[v][壹] += min(dp[b][0], dp[b][壹]);
                        inc = min(inc, dp[b][0] - dp[b][壹]);
                        dp[v][2] = min(dp[v][2]+dp[b][壹], inf);    //防止溢出
                    }
                }
                if(inc>0)   dp[v][壹] += inc;   //minChange=inf的情况被归并了
                if(--len == 0)  break;
            } else {
                int & i = cur.nxt;
                int b = buf[i].b;
                if(!vis[b]) {
                    vis[b] = 壹;
                    stk[len ++].init(b, E[b]);
                }
                i = buf[i].nxt;
            }
        }
        return min(dp[0][0], dp[0][壹]);
    }
} g;

// POJ-3659
int main() {
    int n, a, b;
    while(scanf("%d", &n) != EOF) {
        g.init(n);
        for(int i = 壹; i < n; i ++) {
            scanf("%d%d", &a, &b);
            g.addEdge(a-壹,b-壹);
        }
        printf("%d\n", g.solve());
    }
    return 0;
}
```

prufer编码_树的计数

//hnoi-2004 树的计数
//n个节点的度依次为D壹, D2, ..., Dn的无根树共有(n-2)！/ [(D壹-壹)!(D2-壹)!..(Dn-壹)！]个，因为此时Pr¨¹fer编码中的数字i恰好出现Di-壹次。

```
import java.math.BigInteger;
import java.util.Scanner;
```

```
public class Main {
    public static void main(String[] args) {
        BigInteger jc[] = new BigInteger[壹60];
        jc[0] = BigInteger.ONE;
        for(int i = 壹; i < jc.length; i ++) {
            jc[i] = jc[i-壹].multiply(BigInteger.valueOf(i));
        }
        Scanner scan = new Scanner(System.in);
        while(scan.hasNextInt()) {
            int n = scan.nextInt();

            boolean ok = true;
            BigInteger ans = null;

            if(n == 壹) {
                int tmp = scan.nextInt();
                if(tmp==0) {
                    ans = BigInteger.ONE;
                } else {
                    ok = false;
                }
            } else {
                ans = jc[n-2];
                int all = 0;
                for(int i = 0; i < n; i ++) {
                    int tmp = scan.nextInt();
                    all += tmp;
                    if(tmp == 0)    ok = false;
                    else            ans = ans.divide(jc[tmp-壹]);
                }
                if(all != 2*(n-壹)) ok = false;
            }
            if(false == ok) {
                System.out.println(0);
            } else {
                System.out.println(ans);
            }
        }
    }
}
```

独立集_支配集_匹配

教程：http://hbzhuxu.spaces.live.com/blog/cns!2246F2CEAAA2905A!274.entry
简单的说：
壹.极大独立集＝极小支配集（一般图）
2.独立集补集是点覆盖集（一般图）
3.原图中的独立集是补图中的最大团（一般图）
4.最小点覆盖集＝最大匹配（二分图）
独立集

独立集定义：顶点集的非空子集，子集中的任何 2 个顶点都不相邻。
极大独立集：独立集，且任意增加一个顶点都不是独立集。
最大独立集：不存在顶点数比他更大的独立集。（可能有多个）
独立数：最大独立集的基数。

平移变换：G是简单无向图，将G的邻接矩阵第i行于第j行，第i列与第j列互换。（相当于互换节点ij的编号）

标准型：

定理：简单无向图的邻接矩阵，总可以经过若干次平移变换把A化为标准型，从而得到图G的一个极大独立集。

点覆盖：顶点集的非空子集，图中的每条边都与子集中的某个顶点关联。

极小点覆盖：点覆盖，且任意子集都不是点覆盖。
最小点覆盖：不存在比该集合顶点数更少的点覆盖。
点覆盖数：最小点覆盖集合的基。

独立集与点覆盖：图G=<V,E> S是V的非空子集
S是G的独立集<=>V-S 是G的点覆盖
S是G的极大独立集<=>V-S 是G的极小点覆盖

边覆盖：边集的非空子集，图中的每个顶点都与该边集中的某个边关联。
极小边覆盖：边覆盖，且任意子集都不是边覆盖。
最小边覆盖：不存在比该集合边数更少的边覆盖。
边覆盖数：最小边覆盖集合的基。

团：顶点集的非空子集，该子集中任意两个顶点都相邻。
极大团：团，且任意增加一个顶点则该集合不再是团。

独立集与团：
求G的极大独立集可以转化为求G'的极大团。其中G'为G的补图。

支配集

支配集：顶点集的非空子集，所有不在该集合中的顶点至少与其中一个顶点邻接。
极小支配集：支配集，且任何真子集不是支配集。
最小支配集：不存在比该集合顶点数更少的支配集。
支配数：最小支配集的基。

性质

极小支配集判定：G的支配集S是极小支配集，当且仅当，S中每个顶点x满足下面 2 点之一：
壹 不在支配集中的顶点中存在一个点，该点的邻接点集与支配集的交为{x}。(即每个x至少有一个不在支配集中的点只与x邻接，而不与支配集中其他顶点邻接。)
2 S中所有顶点不与x邻接。
定理：G是没有孤立节点的图，且S是G的极小支配集，那么V(G)-S也是G的支配集。
推论：n阶无孤立点图，最小支配数≤n/2。
定理：最小支配集每个顶点，有一个不在支配集中的顶点仅与它邻接，不与其它支配集中顶点邻接。
定理：n阶图，(n/(壹+ΔG))≤最小支配数≤(n-ΔG)
定理：顶点集是独立支配集，当且仅当，是一个极大独立集（独立支配集<=& gt;极大独立集)。
推论：每个极大独立集是一个极小支配集（极小支配集<=>极大独立集)。

匹配

匹配：无环图的非空边集，集合中任意两条边在G中均不相邻。
最大匹配：不存在比该匹配边数更多的匹配。
匹配数：最大匹配的基。

M饱和点：匹配M中的边关联的顶点。其他的点称为非M饱和点。
M完美匹配：匹配M，图的所有顶点都是M饱和点，则M是图的完美匹配。
M交错路：图G的路，且路中M的边和E(G)-M的边交替出现。
M可增广路：两个端点为M非饱和点的M交错路。(一定含奇数条边)

定理：两个不同匹配M壹、M2，由M壹⊕M2 导出的G的边导出子图记做H，则H的任意连通分支是下列情况之一： (⊕圈加 壹 86)
壹 边在M壹和M2 种交错出现的偶圈
2 边在M壹和M2 种交错出现的偶圈
3 孤立节点
定理(最大匹配判定)：M 是图的最大匹配，当且仅当，G中不存在M可增广路。

S的邻集：S是G的任意顶点子集，G中与S的顶点邻接的所有顶点的集合。

Hall定理：G是有二部划分（V壹,V2）的二分图，则G含有饱和V壹的每个顶点的匹配M的充要条件是，对于任意S⊆G，|N(S)|≥|S|。

(完美匹配存在判定)
推论：具有二部划分的二分图G有完美匹配<=>|V壹|=|V2|，且对任意S⊆V壹(或V2)均有|N(S)|≥|S|。
推论：设G是k正则二分图，则G有完美匹配。(充分条件)
推论：设G是二部划分(V壹,V2)的简单二分图，且|V壹|=|V2|=n，若 δ(G)≥n/2，则G有完美匹配。

定理：G有完美匹配<=>O(G-S)≤|S|，任意S⊂V(G)，其中O(G-S)是G-S的奇阶连通分支数目。证明壹 9 壹

最大匹配生成算法(匈牙利算法)

根在x的M交错子图：由x为起点的M交错路连接的顶点集所导出的G的导出子图。
定理：具有二部划分(V壹,V2)的二分图G的匹配M，x∈V壹，是非M饱和点，H是G中根在x的M交错子图的顶点集，S=H∩V壹,T=H∩V2，则：
壹 T⊆N(S)
2 下述 3 条等价
a G中不存在以x为端点的M可增广路
b x是H中唯一的非M饱和点
c T=N(S)，且|T|=|S|-壹
匈牙利算法
基本思想：
算法：
---壹 96
设G是具有二部划分(V壹,V2)的二分图
壹 任给初始匹配M
2 若M饱和V壹则结束，否则 3
3 在V壹中找一条非M饱和点x，置S=|x|，T=空；
4 若N(S)=T，则停止。否则任选一点y∈N(S)-T
5 若y为M饱和点转 6，否则作：求一条从x到y的M可增广路P，置M=M⊕P，转 2
6 由于y是M饱和点，故M中有一边|y,u|，置S=S∪{u}，T=T∪{y}。转 4。

最优匹配

最优匹配：加权完全二分图中求一个总权最大的完美匹配。

l为G的可行顶标：G是具有二部划分(V壹,V2)的完全加权二分图，l为映射 V(G)->R，满足对G的每条边e={x,y}，均有l(x)+l(y)≥w(x,y)，其中w(x,y)表示边{x,y}的权。
l等子图：令El={{x,y}| {x,y}∈E(G),l(x)+l(y)=w(x,y)}，l等子图Gl为以El为边集的G的生成子图。

定理(最优匹配判定)：设l是G的可行顶标，若l等子图Gl有完美匹配M，则M是G的最优匹配。

求最优匹配有效算法
Kuhn-Munkres算法

壹 从任意可行顶标（如平凡标号）l开始，确定 l 等子图Gl，并且在Gl中选取匹配M。若M饱和V壹，则M是完美匹配，也即M是最优匹配。否则转 2。
2 匈牙利算法终止于S⊂V壹,T⊂V2 且使N(S)=T。计算ai，确定新的可行顶标l'，并以l'代替l，以Gl'代替Gl。转入壹。

定理：(Kn,n，w)的加权矩阵A=(aij)，a是aij中元素最大值。Jn 是n阶全壹方阵，A*=(aij*)=aJn-A是(Kn,n，w*)的加权矩阵。
则M*是(Kn,n，w*)的权最大的完美匹配<=>M*是(Kn,n，w)的权最小的完美匹配，且w(M*)=na-w*(M*)。

最小截断

AHOI2009 - 最小截断 Mincut
2009 年 06 月 24 日 星期三 壹 7:39
昨天被安徽省赛的题虐，里面有一道网络流的题挺有意思，它让求

> 在残量网络中，求流网络中每条满流边的起点到终点是否连通

如果暴力BFS，每次都是O(M)的时间复杂度，这样总共就是O(M^2)。
如果对于每个起点，都用数组存储其BFS的结果，也需要O(NM)。
如果收缩SCC，原图变为DAG，对于同一SCC里的结点是相互连通的，但对于非同一SCC里的结点，仍需要BFS一遍。对于在DAG中求任两点连通性的问题，我在今年寒假就思考过，貌似当时很多神牛都会，但是我还是没想出来应该怎样做。

今天咨询了一下bonism神牛，然后了解了其实对于询问中不在同一SCC里的点对，它们必然不会连通，这里我主要想讲一下其原因：
在网络流算法中，对于每条边，它都有一条反向边。由于本题特殊，每次询问都是针对满流边的起点到终点，那么该边的反向边一定是属于残量网络的，而且其残余流量恰好等于原来正向边的容量。
如果从询问边的起点可以走残量网络里的路径并到达询问边的终点，假设该路径为S=>P壹=>P2=>..=>Pk=>T。根据上面的事实，存在边T=>S属于残量网络。那么残量网络中边构成一条环S=>P壹=>P2=>...=>Pk=>T=>S，这与S和T属于不同SCC相矛盾，故S与T不连通。

囧。。还是SDTSC的题比较水，适合找感……出了省就什么都不会了。。我已经弱到一定境界了