

ACM/ICPC 比赛资料

---



图论和网络流

## 目录

术语 .....	4
有向树 .....	4
图论结论 .....	4
常见思维 .....	5
分数规划 .....	5
网络流 .....	6
最大流 .....	6
最大流 - Dinic - 非递归 .....	6
最小费用最大流 - spfa .....	8
网络流的应用 .....	9
最大权闭合图 Maximum Weight Closure .....	9
最小路径覆盖问题 .....	10
餐巾计划问题 .....	11
DancingParty .....	11
Transform Matrix[srm407 div1 最后一题] .....	11
Road[NOI2009 湖南省选] .....	12
建设乌托邦[有道难题 2010 总决赛题] .....	12
锦标赛[黑书] .....	12
志愿者招募 .....	12
最长 k 可重区间 .....	13
平面图最小割 .....	13
有向图强连通分量(SCC) -StronglyConnectedComponent .....	17
无向图的连通分量可以用 BFS/DFS/DisjointSet .....	17
Tarjan .....	17
Kosaraju - 邻接表 .....	17
缩点(Kosaraju + 模拟函数栈) + DAG DP .....	18
最大半连通子图 .....	21
无向图的连通性 .....	21
定义 .....	21

性质 .....	22
连通度 .....	22
割点和桥 .....	22
Tarjan 算法 .....	22
点双连通分支(块) .....	26
找从 s 到 t 的割点和桥 - $O(n + m)$ .....	29
无向图全局最小割 - Stoer-Wagner .....	29
欧拉图与欧拉路 .....	31
有向图和无向图和混合图的欧拉路 - 缺 .....	31
二分图 .....	31
二分图最大匹配 .....	31
KM 算法 - 时间复杂度 $O(n^3)$ .....	31
Hopcroft-Karp 算法 - 时间复杂度 $O( V ^{0.5} E )$ .....	32
二分图最佳匹配 - Kuhn-Munkr 算法 - 时间复杂度 $O(n^3)$ .....	35
稳定婚姻 - $O(n^2)$ .....	37
关键点和关键边 .....	38
算法 .....	38
关键点代码 .....	38
二分图判断 - bfs - $O(V + E)$ .....	40
一般图最大匹配 - 带花树 .....	41
团 .....	44
最大团 .....	44
伪代码 .....	44
代码 .....	45
极大团 - Bron-Lerbosch algorithm .....	46
伪代码 .....	46
代码 .....	46
图和树的同构 .....	48
有根树的同构 .....	48
无根树的同构 .....	48

图的同构 .....	48
SPFA & dijkstra & Floyd .....	49
SPFA 的两个优化 .....	49
SLF: Small Label First 策略 .....	49
LLL: Large Label Last 策略 .....	49
SPFA 求最短路 .....	49
A*求 k 短路(可以有环) .....	50
K 短路 Yen 无环 .....	52
SPFA 判断负权回路 .....	55
dijkstra(spfa)+heap .....	57
dijkstra $O(n^2)$ .....	60
dijkstra + stl_set .....	60
s->t 去掉某条边的最短路 .....	60
朴素 floyd 算法 .....	65
floyd 求无向图最小环 .....	65
树 .....	66
最近公共祖先 LCA – $n \log n$ dp .....	66
生成树 .....	66
Prim .....	66
次小生成树 .....	67
严格次小生成树 .....	69
最大度限制生成树(缺) .....	72
最优比例生成树(缺) .....	72
最小树形图 .....	72
时间复杂度 VE, 用带头节点和反向边的双向循环链表实现 .....	72
时间复杂度 VE, 用边表实现 .....	75
弦图 .....	77
弦图的判断 ( $E \log E$ ) .....	78
传递闭包 .....	80
dfs 递归转非递归 .....	81

# 术语

环(loop): 若一条边的两个顶点为同一顶点, 则此边称作环。(Loop: An edge connecting a vertex to itself.)

Walk: A sequence  $v_0e_1v_1...e_nv_l$

平凡图: 由一个孤立点组成的图叫做平凡图(又称为平凡树).

简单图: 没有环、且没有多重弧的图称为简单图。

Simple Path: a path in a graph which doesn't have repeating vertices.

定向图: 对无向图  $G$  的每条无向边指定一个方向得到的有向图。

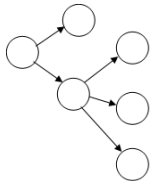
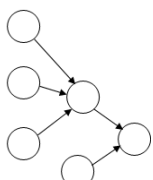
底图: 把有向图的每一条有向边的方向都去掉得到的无向图。

竞赛图: 有向图的底图是无向完全图, 则此有向图是竞赛图。性质: 竞赛图不存在环路 iff. 所有点的出度从小到大排列依次为  $0..n-1$

正则图: 如果图中的所有顶点的度数都相等, 则此图为正则图。

## 有向树

有向树: 去掉方向后称为一棵树的有向图称为有向树。

<p><b>外向树:</b> 一棵有向树 <math>T</math>, 若有且只有一个顶点入度为 <math>0</math>, 其余顶点入度都为 <math>1</math>, 则称 <math>T</math> 为外向树。<math>T</math> 中入度为 <math>0</math> 的节点被称为 <math>T</math> 的根节点, 出度为 <math>0</math> 的节点被称为 <math>T</math> 的叶节点, 每个节点 <math>u</math> 的有向边指向的节点称该节点(<math>u</math>)的子节点。</p>	
<p><b>内向树:</b> 一棵有向树 <math>T</math>, 若有且只有一个顶点出度为 <math>0</math>, 其余顶点出度都为 <math>1</math>, 则称 <math>T</math> 为内向树。<math>T</math> 中出度为 <math>0</math> 的节点被称为 <math>T</math> 的根节点, 入度为 <math>0</math> 的节点被称为 <math>T</math> 的叶节点, 每个节点 <math>u</math> 的有向边的反向边指向的节点称为该节点(<math>u</math>)的子节点。</p>	

注: 外向树和内向树都是有根树。

## 图论结论

点独立集(vertex independent set, VIS): 图的顶点集的子集, 其中任意两点不相邻

一般图的最大独立集是 NP-hard; 对于弦图(区间图), 可以根据 PEO 直接贪心。

对于二分图, 可以用最小覆盖集来求最大独立集; 对于树, 可以用 treedp 来求。

最小(点)覆盖: (“点”覆盖“边”)图的顶点集的子集, 若选中了点  $P$ , 则说他覆盖了所有以他为端点的边, 把所有的边都覆盖的所用顶点数最少的集合, 称为最小覆盖集。

最大(极大)独立集 + 最小(极小)(点)覆盖集 =  $V$

最大(极大)团 = 补图的最大(极大)独立集

二分图的最大独立集 =  $V$  - 二分图的最大匹配

二分图的最大(点权)独立集 = Sum - 二分图的最佳匹配

二分图的最小(边权)覆盖 = 二分图的最佳匹配

二分图的最小(点权)覆盖 = 最小割( $X$  -  $Y$  之间的边设为 inf)

二分图的最小覆盖 = 二分图的最大匹配 (König 定理, 证明参考最大流最小割定理)

Hall 定理: 设二分图  $G = \langle V_1, V_2, E \rangle$ ,  $|V_1| \leq |V_2|$ ,  $G$  中存在从  $V_1 \rightarrow V_2$  的完备匹配 iff.  $V_1$  中任意  $k$  ( $1 \leq k \leq |V_1|$ ) 个顶点至少与  $V_2$  中  $k$  个顶点相邻。

推论: 设二分图  $G = \langle V_1, V_2, E \rangle$ ,  $V_1$  中每个顶点至少关联  $t$  ( $t \geq 1$ ) 条边, 而  $V_2$  中每个顶点至多关联  $t$  条边, 则  $G$  中存在从  $V_1 \rightarrow V_2$  的完备匹配。

## 常见思维

树可以通过 dfs 转为序列 e.g. lca  $\rightarrow$  rmq, apple tree

## 分数规划

一般形式

$$\text{Minimize } \lambda = f(x) = \frac{a(x)}{b(x)}$$

其中, 解向量  $x$  在解空间  $S$  内,  $a(x)$  和  $b(x)$  都是连续的实值函数, 且  $\forall x \in S, b(x) > 0$ 。

(对于最大化目标函数, 只需令  $a' = -a, \lambda' = -\lambda$ )

假设  $\lambda^* = f(x^*)$  为该规划的最优解, 有  $0 = a(x^*) - \lambda^* \cdot b(x^*)$

令  $g(\lambda) = \min_{x \in S} \{a(x) - \lambda \cdot b(x)\}$ , 则  $g(\lambda)$  满足以下条件

**单调性定理:**  $g(\lambda)$  是一个严格减函数, 即对于  $\lambda_1 < \lambda_2$ , 有  $g(\lambda_1) > g(\lambda_2)$

**Dinkelbach 定理:** 设  $\lambda^*$  为原规划的最优解, 则  $\lambda = \lambda^*$  iff.  $g(\lambda) = 0$

**推论:** 设  $\lambda^*$  为原规划的最优解, 则

$$\begin{cases} g(\lambda) < 0 \Leftrightarrow \lambda > \lambda^* \\ g(\lambda) = 0 \Leftrightarrow \lambda = \lambda^* \\ g(\lambda) > 0 \Leftrightarrow \lambda < \lambda^* \end{cases}$$

于是可以对最优解  $\lambda^*$  进行二分逼近。

# 网络流

**求解可行流：**给定一个网络流图，初始时每个节点不一定流量平衡(每个节点可以有盈余或不足)，每条边的流量可以有上下界(当前流量不一定满足上下界约束)。可行流求解中没有源和汇的概念，算法的目的是寻找一个可以使所有节点都能平衡，所有边都能满足能量约束的方案，同时可能附加有最小费用的条件(最小费用可行流)

**求解最大流：**给定一个网络流图，其中有两个特殊的节点称为源和汇，给定的每个节点一定平衡，源可以产生无限大的流量，汇可以吸收无限大的流量。标准的最大流模型，初始解(一般为零流)一定是可行的，因此边上不能有以下界。算法的目的是寻找一个从源到汇的流量最大的方案，同时不破坏可行性约束，并可能附加有最小费用的条件(最小费用最大流)

**扩展的最大流：**在有上下界或有节点盈余的网络流图中求解最大流。实际上包括两个部分，显示消除下界、消除盈余，可能还需要消除不满足最优条件的流量(最小费用最大流)，找到一个可行流，在进一步得到最大流。因此我们实质上把扩展最大流的求解分为两个部分(求解可行流、求解最大流)

## 最大流

注：虽然最小割  $C = [S, T]$  中的边都是满流边，但满流边不一定是最小割  $C$  中的边。所以一定要用 dfs/bfs 的方法得到最小割  $C$

## 最大流 - Dinic – 非递归

时间复杂度  $O(V^2E)$ , 1s 能支持  $|V| = 10^4$ ,  $|E| = 10^5$  左右

//确保正确，上次使用 2012-9-9，对于 double 类型的网络流，需要把 LL 改成 double

```
struct Dinic {
    const static int maxn = ?; //5001
    const static int maxe = ?; // 60200
    const static int inf = 2000000000;

    struct node {
        int b, c;
        node *next, *anti;
    } *ge[maxn], pool[maxe], *pooltop;
    int dist[maxn], n;

    void init(int n) {
        pooltop = pool; this->n = n;
        for (int i = 0; i < n; ++i) ge[i] = 0;
    }
}
```

```

node *_insert(int a, int b, int c) {
    node *p = pooltop++; p->b = b; p->c = c; p->next = ge[a]; ge[a] = p; return p;
}

```

```

void insert2(int a, int b, int c) {
    node *p = _insert(a, b, c), *q = _insert(b, a, c);
    p->anti = q; q->anti = p;
}

```

```

bool bfs(int s, int t) {
    static int q[maxn], *qtop, *qbase;
    qbase = qtop = q;
    memset( dist, -1, sizeof(dist[0]) * (n+1) );
    dist[s] = 0; *qtop++ = s;
    for (; qbase != qtop; ++qbase) {
        for (node *p = ge[*qbase]; p; p = p->next) {
            if ( p->c && dist[p->b] == -1 ) { //double 的流改为 sign(p->c)
                dist[p->b] = dist[*qbase] + 1;
                *qtop++ = p->b;
                if (p->b == t) return true;
            }
        }
    }
    return false;
}

```

```

LL flow(int s, int t) {
    static int pre[maxn];
    static node *cur[maxn], *path[maxn];

    LL tot = 0;
    while ( bfs(s, t) ) {
        memcpy(cur, ge, sizeof(ge[0]) * (n+1) );
        for (int i = s; dist[s] != -1; ) {
            if ( i == t ) {
                int flow = inf;
                for (; i != s; i = pre[i])
                    flow = min(flow, path[i]->c);
                tot += flow;
                for (i = t; i != s; i = pre[i]) {
                    path[i]->c -= flow; path[i]->anti->c += flow;
                }
            }
            for (node *&p = cur[i]; p; p = p->next) {

```



```

        int v = p->b;
        if ( p->c && dist[v] == dist[i] + 1 ) { //double 的流改为 sign(p->c)
            pre[v] = i; path[v] = p; i = v; break;
        }
    }
    if ( cur[i] == 0 ) {
        dist[i] = -1; i = pre[i];
    }
}
return tot;
}
} flow;

```

## 最小费用最大流 – spfa

可以证明如果原图没有负环路，则在增广过程中不会出现负环路

```

const int maxn = 5001 * 2, maxe = 60200 * 5;
const int inf = 2000000000; // >= maxc * 2

```

```

struct node {
    int adj, cap, cost; node *next, *anti;
} *ge[maxn], pool[maxe], *pooltop;

```

```

void init(int n){
    pooltop = pool; for (int i = 0; i <= n; i++) ge[i] = 0;
}

```

```

inline node *_addEdge(int a, int b, int cap, int cost) {
    node *p = pooltop++; p->adj = b, p->cap = cap, p->cost = cost, p->next = ge[a], ge[a] = p;
    return p;
}

```

```

inline void addEdge(int a, int b, int cap, int cost) {
    node *p1 = _addEdge(a, b, cap, cost), *p2 = _addEdge(b, a, 0, -cost); //注意 a, b 顺序
    p1->anti = p2, p2->anti = p1;
}

```

```

inline void addBiEdge(int a, int b, int cap, int cost) {
    addEdge(a, b, cap, cost); addEdge(b, a, cap, cost);
}

```

```

complex<long long> augment(int n, int s, int t, int flowlimit){
    static int queue[maxn], *qbase, *qtop, inqueue[maxn], dist[maxn], pre[maxn];

```

```

static node *path[maxn];
qbase = qtop = queue;
#define enqueue(x) { *qtop++ = x; if (qtop==queue+maxn) qtop = queue; inqueue[x] = 1;}
#define dequeue(x) {x = *qbase++; if (qbase==queue+maxn) qbase = queue; inqueue[x] = 0;}
enqueue(s);
for (int i = 0; i <= n; i++) dist[i] = inf;
dist[s] = 0;
while (qbase != qtop){
    int u; dequeue(u);
    for (node *p = ge[u]; p; p = p->next){
        if (p->cap > 0 && dist[p->adj] > dist[u] + p->cost){
            dist[p->adj] = dist[u] + p->cost;
            pre[p->adj] = u; path[p->adj] = p;
            if (!inqueue[p->adj]) enqueue(p->adj);
        }
    }
}
long long flow = flowlimit, cost = 0;
if (dist[t] == inf) return complex<long long>(0, 0);

for (int i = t; i != s; i = pre[i])
    flow = min(flow, (long long)path[i]->cap);
for (int i = t; i != s; i = pre[i]){
    cost += flow * path[i]->cost;
    path[i]->cap -= flow; path[i]->anti->cap += flow;
}
return complex<long long>(flow, cost);
}

complex<long long> mincostmaxflow(int n, int s, int t, int flowlimit = inf){
    complex<long long> ret = 0, tmp;
    while ((tmp = augment(n, s, t, flowlimit)).real() > 0){
        ret += tmp;
        flowlimit -= tmp.real();
    }
    return ret;
}

```

## 网络流的应用

### 最大权闭合图 Maximum Weight Closure

定义一个有向图  $G=(V,E)$  的**闭合图(closure)**是该有向图的一个点集，且该点集的所有出边

都还指向该点集。即闭合图内任意点的任意后继也一定在闭合图中。更形式化地说，闭合图是这样的一个点集  $V' \subseteq V$ ，满足对于  $\forall u \in V'$  引出的  $\forall \langle u, v \rangle \in E$ ，必有  $v \in V'$  成立。还有一种等价定义为：满足对于  $\forall \langle u, v \rangle \in E$ ，若有  $\forall u \in V'$  成立，必有  $v \in V'$  成立，在布尔代数上这是一个“蕴含(imply)”运算。按照上面的定义，闭合图是允许超过一个连通快的。

给每个顶点  $v$  分配一个点权  $w_v$  (任意实数，可正可负)。**最大权闭合图(maximum weight closure)**，是一个点权之和最大的闭合图，即最大化  $\sum_{v \in V'} w_v$ 。

建图：

$$\begin{aligned} V_N &= V \cup \{s, t\} \\ E_N &= E \cup \{\langle s, v \rangle | v \in V, w_v > 0\} \cup \{\langle v, t \rangle | v \in V, w_v < 0\} \\ &\begin{cases} c(u, v) = \infty & \langle u, v \rangle \in E_N \\ c(s, v) = w_v & w_v > 0 \\ c(v, t) = -w_v & w_v < 0 \end{cases} \end{aligned}$$

## 最小路径覆盖问题

给定有向无环图  $G = (V, E)$ 。设  $P$  是  $G$  的一个简单路（顶点不相交）的集合。如果  $V$  中每个顶点恰好在  $P$  的一条路上，则称  $P$  是  $G$  的一个**路径覆盖**。 $P$  中路径可以从  $V$  的任何一个顶点开始，长度也是任意的，特别地，可以为 0。

$G$  的**最小路径覆盖**是  $G$  的所含路径条数最少的路径覆盖。

**分析：** 对于一个路径覆盖，有如下性质：

- 1、每个顶点属于且只属于一个路径。
- 2、路径上除终点外，从每个顶点出发只有一条边指向路径上的另一顶点。

所以我们可以把每个顶点理解成两个顶点，一个是出发，一个是目标，建立二分图模型。该二分图的任何一个匹配方案，都对应了一个路径覆盖方案。如果匹配数为 0，那么显然路径数=顶点数。每增加一条匹配边，那么路径覆盖数就减少一个，所以**路径数 = 顶点数 - 匹配数**。要想使路径数最少，则应最大化匹配数，所以要求二分图的最大匹配。

**注意**，此建模方法求最小路径覆盖仅适用于**有向无环图**，如果有环或是无向图，那么有可能求出的一些环覆盖，而不是路径覆盖。

## 最小路径覆盖例题

假设有  $n$  根柱子，现要按下述规则在这  $n$  根柱子中依次放入编号为 1, 2, 3, ..... 的球。

- (1) 每次只能在某根柱子的最上面放球。
- (2) 在同一根柱子中，任何 2 个相邻球的编号之和为完全平方数。

试设计一个算法，计算出在  $n$  根柱子上最多能放多少个球。例如，在 4 根柱子上最多可放 11 个球。

**题解：**（顺序）枚举答案转化为判定性问题，然后最小路径覆盖。在图中建立节点 1..A。如果对于  $i < j$  有  $i+j$  为一个完全平方数，连接一条有向边  $(i, j)$ 。该图是有向无环图，求最小路径覆盖。如果刚好满足最小路径覆盖数等于  $N$ ，那么  $A$  是一个可行解，在所有可行解中找到最大的  $A$ ，即为最优解。

**数学公式**(未证明正确性):  $f(n) = \left\lfloor \frac{1}{2}n^2 + n - \frac{1}{2} \right\rfloor$

## 餐巾计划问题

一个餐厅在相继的 $N$ 天里，每天需用的餐巾数不尽相同。假设第 $i$ 天需要 $r_i$ 块餐巾( $i=1, 2, \dots, N$ )。餐厅可以购买新的餐巾，每块餐巾的费用为 $p$ 分；或者把旧餐巾送到快洗部，洗一块需 $m$ 天，其费用为 $f$ 分；或者送到慢洗部，洗一块需 $n$ 天( $n>m$ )，其费用为 $s<f$ 分。每天结束时，餐厅必须决定将多少块脏的餐巾送到快洗部，多少块餐巾送到慢洗部，以及多少块保存起来延期送洗。但是每天洗好的餐巾和购买的新餐巾数之和，要满足当天的需求量。试设计一个算法为餐厅合理地安排好 $N$ 天中餐巾使用计划，使总的花费最小。

**题解：**把每天分为二分图两个集合中的顶点 $X_i, Y_i$ ，建立附加源 $S$ 汇 $T$ 。

- 1、从 $S$ 向每个 $X_i$ 连一条容量为 $r_i$ ，费用为0的有向边。
  - 2、从每个 $Y_i$ 向 $T$ 连一条容量为 $r_i$ ，费用为0的有向边。
  - 3、从 $S$ 向每个 $Y_i$ 连一条容量为无穷大，费用为 $p$ 的有向边。
  - 4、从每个 $X_i$ 向 $X_{i+1}$ ( $i+1 \leq N$ )连一条容量为无穷大，费用为0的有向边。
  - 5、从每个 $X_i$ 向 $Y_{i+m}$ ( $i+m \leq N$ )连一条容量为无穷大，费用为 $f$ 的有向边。
  - 6、从每个 $X_i$ 向 $Y_{i+n}$ ( $i+n \leq N$ )连一条容量为无穷大，费用为 $s$ 的有向边。
- 求网络最小费用最大流，费用流值就是要求的最小总花费。

## DancingParty

题意： $n$ 个男孩和 $n$ 个女孩参加舞会，每对异性之间最多跳一次舞，每轮舞蹈大家会分成 $n$ 对异性舞伴跳舞，每对异性之间可能喜欢也可能讨厌(喜欢关系是对称的)，每个人最多跟 $k$ 个讨厌的人跳舞，问最多能跳几轮？

题解：网络流，首先转化成判定能否跳 $x$ 轮的问题，那么显然 $k \leq x \leq n$ 。从源点向每个男孩、每个女孩向汇点引容量为 $x$ 的边，从每个男孩向每个喜欢的女孩连条容量为1的边，从每个男孩向中转站、中转站向每个女孩引容量为 $x$ 的边，判断是否存在可行流即可。

## Transform Matrix[srm407 div1 最后一题]

题目：两个 $n*m$ 的01矩阵 $A$ 和 $B$ ，对 $A$ 进行若干次操作，使得 $A$ 变成和 $B$ 一样。一次操作：交换 $A$ 中相邻两个位置的元素，每个格子 $(i,j)$ 最多被操作 $C_{ij}$ 次，问最少操作次数。

分析：

1.先不考虑 $C_{ij}$ 限制，一次操作 = 移动一个1，代价为1，给定1的初始和最终位置，用最少的步数移动。

对 $A$ 中所有为1的点 $i$ ，连边 $(s,i)$ ，容量1，费用0

对 $B$ 中所有为1的点 $i$ ，连边 $(i,t)$ ，容量1，费用0

相邻点连边，容量+inf，费用1

2.现在考虑 $C_{ij}$ 的限制(具体对应到实现会引发拆点，每个点 $\langle i, j \rangle$ 拆为两点，费用0，容量如下)

若 $A_{ij} == B_{ij}$ ，必被操作偶数次，点容量为 $C_{ij} / 2$

若 $A_{ij} != B_{ij}$ ，必被操作奇数次，点容量为 $(C_{ij} + 1) / 2$

## Road[NOI2009 湖南省选]

**题目：**数列  $h[1..n]$ ，通过修改，使之满足：相邻项差  $< d$ ，求最小修改代价  $\sum |\Delta h_i|$

**网络流解法：**令  $a[i] = h[i] - h[i+1]$  ( $1 \leq i \leq n-1$ )，则  $h[i]++$  iff.  $a[i-1]--$  &  $a[i]++$ ，问题转化为将  $a$  重新分配，使得每个  $a$  都落在  $[-d, d]$  范围内，其中  $a[1]$  和  $a[n-1]$  可以凭空  $+1/-1$ ，其余都是  $1$  在两个相邻的  $a$  之间“游走”

边	容量	费用	流量下界	注意
$s \rightarrow i$	$+\infty$	0	$a[i] > d ? a[i] - d : 0$	$1 \leq i \leq n-1$
$i \rightarrow t$	$+\infty$	0	$-d > a[i] ? -d - a[i] : 0$	$1 \leq i \leq n-1$
$i \rightarrow i+1$	$+\infty$	1	0	$1 \leq i \leq n-2$
$i \rightarrow i-1$	$+\infty$	1	0	$2 \leq i \leq n-1$
$s \rightarrow 1, s \rightarrow n-1$	$+\infty$	1	0	
$1 \rightarrow t, n-1 \rightarrow t$	$+\infty$	1	0	

求  $s \rightarrow t$  最小费用可行流即可。

## 建设乌托邦[有道难题 2010 总决赛题]

**题目：**一张图，有特殊点和一般点，每条边的端点至少有一个特殊点，每个点有权值  $L[i]$ ，特殊点权值可以修改，代价为  $c|\Delta L[i]|$ ，边  $(i,j)$  要计算代价，代价为  $e|L[i]-L[j]|$ ，求  $\min\{c \cdot \sum\{|\Delta L[i]|\} + e \cdot \sum\{|L[i]-L[j]|\}\}$ ， $L$  只能为  $1-20$  的整数。

**解法：**设  $cost[i][k]$  表示将特殊点  $i$  的权值改为  $k$ ， $i$  的点权代价 + 与周围非特殊点连边的边权代价。则  $cost[i][k] = c \cdot |k - L[i]| + e \cdot \sum\{|L[i] - L[j]| : j \text{ 为与 } i \text{ 相邻的非特殊点}\}$ 。下面顺序考虑特殊点之间的边权代价。

则可以建图， $G = \{V, E\}$ ， $V = \{s, t\} + \{(i, k) : 1 \leq i \leq n, 1 \leq k < 20\}$

E	容量	注意
$s \rightarrow (i, 1)$	$cost[i][1]$	$1 \leq i \leq n$
$(i, k) \rightarrow (i, k+1)$	$cost[i][k]$	$1 \leq i \leq n, 1 \leq k < 19$
$(i, 19) \rightarrow t$	$cost[i][20]$	$1 \leq i \leq n$
$(i, k) \leftrightarrow (j, k)$	$e$	$1 \leq k \leq 19$ ，原图有边 $(i, j)$

## 锦标赛[黑书]

**问题：**单循环，无平局，胜利(严格)最多夺冠，已知部分比赛结果，问谁可能夺冠。

**解法：**枚举让谁夺冠，让他剩下比赛都获胜，设他的胜场为  $k+1$ ，则其他人最多可以胜  $k$  场  
设选手  $i$  已经胜  $w[i]$  场，则建图：左边是剩下的比赛，右边是选手

从源点向每个比赛引边，容量  $1$ ，每个比赛向相应的两名选手引边，容量  $1$ ，从每个选手向汇点引边，容量  $k - w[i]$ ，判断网络是否可以满流即可。

## 志愿者招募

**问题：** $n$  天，第  $i$  天需要  $a[i]$  个志愿者，共  $m$  类志愿者，每种志愿者人数无限，第  $j$  类志愿

者从第  $s[i]$  天工作到第  $t[i]$  天，雇佣费用  $c[i]$ ，求最小花费。

**解法：** 3 3

2 3 4 //  $a[i]$

1 2 2 //  $[1, 2] c = 2$

2 3 5 //  $[2, 3] c = 5$

3 3 2 //  $[3, 3] c = 2$

得到  $n$  个不等式，含  $m$  个未知数( $x[i]$ 表示第  $i$  类人选  $x[i]$ 个，不等式左边为某天的已经有的人的最大工作量，右边为需要的工作量)

$x[1] \geq 2$

$x[1] + x[2] \geq 3$

$x[2] + x[3] \geq 4$

引入变量  $y[i] \geq 0 (1 \leq i \leq n)$ ，将不等式转化为等式

$x[1] - y[1] = 2$

$x[1] + x[2] - y[2] = 3$

$x[2] + x[3] - y[3] = 4$

$0 = 0$

相邻两个等式作差

$x[1] - y[1] = 2$

$x[2] + y[1] - y[2] = 1$

$x[3] - x[1] + y[2] - y[3] = 1$

$-x[2] - x[3] + y[3] = -4$

注意每个  $x[i]$  只出现 2 次，正负各一次，一个点流入一个点流出(因为每种人可以工作的时间是个区间)

我们把  $n$  个等式抽象为  $n$  个点， $x[i], y[i]$  抽象为边(连边的时候有点基尔霍夫电流定理的感觉)

如果等式  $i$  的值  $> 0$ ，则引边  $s \rightarrow i$ ，容量为等式  $i$  的值

如果等式  $i$  的值  $< 0$ ，则引边  $i \rightarrow t$ ，容量为 |等式  $i$  的值|

对于每个变量  $z$ ，若他在等式  $i$  中为  $+$ ，等式  $j$  中为  $-$ ，则连边  $i \rightarrow j$ ，费用为 1

求  $s \rightarrow t$  的最小费用最大流即可

## 最长 k 可重区间

**问题：**  $n$  个开区间，去若干个，使得没有超过  $k$  个区间覆盖同一点，问：取得的区间长度和的最大值

**解答：** 按照升序排序，离散化为点  $1..m$ ，每个端点建立一个节点，并增加  $s, t$

如果区间是  $(a, b)$ ，则连边  $\text{hash}(a) \rightarrow \text{hash}(b)$ ，容量 1，费用为区间长度

$s \rightarrow 1, m \rightarrow t$  连边，容量  $k$ ，费用 0；从  $i \rightarrow i+1$  连边，容量  $+\infty$ ，费用 0；求  $s \rightarrow t$  的最大费用最大流即可。

## 平面图最小割

//上次使用 2012.9.9，已经通过 hdu4280

```
typedef pair<int, int> Point;
```

```
#define x first
```

```

#define y second

const int inf = 1000000000;
static const int maxn = 100000 * 2 + 5;
static const int maxe = maxn * 2 + 5;

struct Graph {
    vector< pair<int, int> > ge[maxn]; int n;
    void init(int n){
        this->n = n; for (int i = 0; i < n; ++i) ge[i].clear();
    }
    void insert(int a, int b, int c) {
        ge[a].push_back( make_pair(b, c) );
        ge[b].push_back( make_pair(a, c) );
    }
    LL sssp(int s, int t) {
        set< pair<LL, int> > h;
        static LL dist[maxn];
        for ( int i = 0; i < n; ++i ) {
            dist[i] = i == s ? 0 : inf;
            h.insert( make_pair(dist[i], i) );
        }
        while ( !h.empty() ) {
            int u = h.begin()->second; h.erase( h.begin() );
            for (int k = 0; k < ge[u].size(); ++k) {
                int v = ge[u][k].first, d = ge[u][k].second;
                if ( dist[v] > dist[u] + d ) {
                    h.erase( make_pair(dist[v], v) );
                    dist[v] = dist[u] + d;
                    h.insert( make_pair(dist[v], v) );
                }
            }
        }
        return dist[t];
    }
} graph;

```

```

struct MaxFlowPlanar {
    Point p[maxn];
    int n, ecnt, fcnt;

    struct edge {
        int a, b, c, vis, find;
        edge *prev, *anti;
    }
}

```

```

double ang;
edge *next(){
    return this->anti->prev;
}
void init(int a, int b, int c, double ang, edge *anti) {
    this->a = a; this->b = b; this->c = c; this->ang = ang;
    this->vis = 0; this->anti = anti;
}
} e[maxe], *ptr[maxe];

struct Cmp {
    bool operator()(const edge* x, const edge* y) const {
        if ( x->a != y->a ) return x->a < y->a;
        return x->ang < y->ang;
    }
};

void init(Point p[], int n) {
    copy(p, p + n, this->p);
    this->n = n; ecnt = 0; fcnt = 0;
}

void insert2(int a, int b, int c) {
    int dy = p[b].y - p[a].y, dx = p[b].x - p[a].x;
    e[ecnt].init(a, b, c, atan2( dy, dx ), &e[ecnt ^ 1] ); ++ecnt;
    e[ecnt].init(b, a, c, atan2( -dy, -dx ), &e[ecnt ^ 1] ); ++ecnt;
}

LL maxflow() {
    for (int i = 0; i < ecnt; ++i) ptr[i] = e + i;
    sort( ptr, ptr + ecnt, Cmp() );
    for (int i = 0, j; i < ecnt; i = j) {
        for (j = i + 1; j < ecnt && ptr[i]->a == ptr[j]->a; ++j);
        ptr[i]->prev = ptr[j-1];
        for (int k = i + 1; k < j; ++k) ptr[k]->prev = ptr[k-1];
    }

    for (int i = 0; i < ecnt; ++i) {
        if ( ptr[i]->vis ) continue;
        ptr[i]->find = fcnt, ptr[i]->vis = 1;
        for ( edge *p = ptr[i]->next(); p != ptr[i]; p = p->next() )
            p->find = fcnt, p->vis = 1;
        ++fcnt;
    }
}

```



```

graph.init(fcnt);
int s = -1, t = -1;
for (int i = 0; i < ecnt; ++i) {
    if ( ptr[i]->c != inf ) {
        graph.insert(ptr[i]->find, ptr[i]->anti->find, ptr[i]->c);
    } else if (s == -1) {
        s = ptr[i]->find, t = ptr[i]->anti->find;
    }
}
return graph.sssp(s, t);
}
} flow;

void solve(){
    int n, m; cin >> n >> m;
    static Point p[maxn];
    int maxY = -inf, minY = inf;
    for (int i = 0; i < n; ++i) {
        scanf("%d%d", &p[i].x, &p[i].y);
        checkmin(minY, p[i].y); checkmax(maxY, p[i].y);
    }
    int s = min_element(p, p + n) - p, t = max_element(p, p + n) - p;
    p[n] = make_pair(p[s].x - 1, maxY + 1); p[n+1] = make_pair(p[t].x + 1, maxY + 1);

    flow.init(p, n + 2);
    flow.insert2(s, n, inf);
    flow.insert2(n, n+1, inf);
    flow.insert2(n+1, t, inf);
    for (int i = 0; i < m; ++i) {
        int a, b, c; scanf("%d%d%d", &a, &b, &c);
        flow.insert2(a-1, b-1, c);
    }
    cout << flow.maxflow() << endl;
}

int main(){ int re; cin >> re; while (re--) solve(); }

```

# 有向图强连通分量(SCC) –StronglyConnectedComponent

无向图的连通分量可以用 BFS/DFS/DisjointSet

## Tarjan

在实际测试中，tarjan 算法的运行效率比 Kosaraju 高 30%左右

注意：若原图不连通，可能需要枚举根。

dfn[u]: 节点 u 搜索的次序编号(时间戳)

low[u]: u 或 u 的子树能够通过非树枝边追述到的最早的搜索树中的节点编号  
则：

```
low[u] = min(  
    dfn[u],  
    low[v] if (u, v)为搜索树的树枝边, u 为 v 的父节点  
    dfn[v] if (u, v)为搜索树中节点的后向边(非横叉边)  
);
```

dfn[u] = low[u] iff. 以 u 为根的搜索树的子链上所有节点是一个强连通分量

伪代码

```
tarjan(u) {  
    dfn[u] = low[u] = ++index;  
    Stack.push(u);  
    for each (u, v) in E  
        if (v is not visited)  
            tarjan(v)  
            low[u] = min(low[u], low[v])  
        else if (v in Stack)  
            low[u] = min(low[u], dfn[v])  
    if (dfn[v] == low[u])  
        repeat  
            v = Stack.pop  
            print v  
        until u == v  
}
```

## Kosaraju – 邻接表

参考第 34 页，Kosaraju 类，有可能还需要加入类似的代码来构建新图，上次使用 2012.6.9

```
REP(i, nSCC) {  
    for (node *p = ge[i]; p; p = p->next){  
        int gi = group[i], gj = group[p->b];
```

```

        if ( gi != gj && joint[gi][gj] != ri){
            joint[gi][gj] = ri; _addedge(gg, gi, gj);
        }
    }
}

```

## 缩点(Kosaraju + 模拟函数栈) + DAG DP

//已经通过 APIO2009 ATM <http://61.187.179.132/JudgeOnline/problem.php?id=1179>

```

const int maxn = 500000 + 5;
const int inf = 2000000000;
int visit[maxn], order[maxn], group[maxn], cnt;

struct node { int adj; node *next;
} *ge[maxn], *ge_rev[maxn], *ge_group[maxn], pool[maxn * 10], *pooltop = pool;

/* void dfs(int u){
    visit[u] = 1;
    for (node *p = ge[u]; p; p = p->next){
        int v = p->adj;
        if(!visit[v]) dfs(v);
    }
    order[cnt++] = u;
}*/

void dfs(int u){
    static pair<int, node *> stk[maxn], *stktop;
    stktop = stk;
    node *p;
    *stktop++ = make_pair(u, ge[u]);
    while (stktop != stk){
        --stktop; u = stktop->first; p = stktop->second;
        visit[u] = 1;
        for (; p; p = p->next){
            if (!visit[p->adj]){
                *stktop++ = make_pair(u, p->next);
                *stktop++ = make_pair(p->adj, ge[p->adj]);
                break;
            }
        }
    }
    if (p == 0){
        order[cnt++] = u;
    }
}

```

```

}

/*void dfs_rev(int u){
    visit[u] = 1; group[u] = cnt;
    for (node *p = ge_rev[u]; p; p = p->next){
        int v = p->adj;
        if (!visit[v]) dfs_rev(v);
    }
}*/

void dfs_rev(int u){
    static pair<int, node *> stk[maxn], *stktop;
    stktop = stk; node *p;
    *stktop++ = make_pair(u, ge_rev[u]);
    while (stktop != stk){
        --stktop;
        u = stktop->first;
        p = stktop->second;
        if (p == ge_rev[u]){
            visit[u] = 1; group[u] = cnt;
        }
        for (; p = p->next){
            if (!visit[p->adj]){
                *stktop++ = make_pair(u, p->next);
                *stktop++ = make_pair(p->adj, ge_rev[p->adj]);
                break;
            }
        }
    }
}

int SCC(int n){
    cnt = 0; CLR(visit, 0, n+1);
    for (int i = 0; i < n; i++) //这里可能需要根据图的储存方式改为 i = 1..n
        if (!visit[i]) dfs(i);
    cnt = 0; CLR(visit, 0, n+1);
    for (int i = n-1; i >= 0; i--){
        int u = order[i];
        if (!visit[u]){
            dfs_rev(u); cnt++;
        }
    }
    return cnt;
}

```

```

#define addedge(ge, a, b){\
    node *_p = pooltop++; *_p->adj = b; *_p->next = ge[a]; ge[a] = *_p; }

int val[maxn], dest[maxn];
int group_val[maxn], group_dest[maxn];
int dp[maxn];

int main(){
    int n, m, a, b, s;
    while (scanf("%d%d", &n, &m) == 2){
        CLR(ge, 0, n); CLR(ge_rev, 0, n); CLR(dest, 0, n);
        pooltop = pool;
        for (int i = 0; i < m; i++){
            scanf("%d%d", &a, &b); a--; b--;
            addedge(ge, a, b); addedge(ge_rev, b, a);
        }
        for (int i = 0; i < n; i++){
            scanf("%d", &val[i]);
        }
        scanf("%d%d", &s, &a); --s;
        for (int i = 0; i < a; i++){
            scanf("%d", &b); dest[--b] = 1;
        }
        SCC(n);
        for (int i = 0; i < cnt; i++){
            group_val[i] = group_dest[i] = 0;
            dp[i] = -inf; ge_group[i] = 0;
        }

        s = group[s];
        for (int i = 0; i < n; i++){
            group_dest[ group[i] ] |= dest[i];
            group_val [ group[i] ] += val[i];
            for (node *p = ge[i]; p; p = p->next){
                if (group[i] == group[p->adj]) continue;
                addedge(ge_group, group[i], group[p->adj]);
            }
        }
        int maxval = 0;
        static int que[maxn], *qbase, *qtop, inqueue[maxn];
#define enqueue(x) { *qtop++ = x; if (qtop == que + maxn) qtop = que; inqueue[x] = 1; }
#define dequeue(x) { x = *qbase++; if (qbase == que + maxn) qbase = que; inqueue[x] = 0; }
        CLR(inqueue, 0, cnt); qbase = qtop = que;
    }
}

```

```

    *qtop++ = s;
    dp[s] = group_val[s];
    while (qbase != qtop){
        dequeue(s);
        for (node *p = ge_group[s]; p; p = p->next){
            if (dp[p->adj] < dp[s] + group_val[p->adj]){
                dp[p->adj] = dp[s] + group_val[p->adj];
                if (!linqueue[p->adj]) enqueue(p->adj);
            }
        }
    }
    REP(i, cnt) if (group_dest[i] && dp[i] > maxval) maxval = dp[i];
    cout << maxval << endl;
}
}

```

## 最大半连通子图

[ZJOI2007]<http://www.zybbbs.org/JudgeOnline/problem.php?id=1093>

- 一个有向图  $G = (V, E)$  称为是半连通的 iff. 对于任意  $u, v$  属于  $V$ , 满足  $u \rightarrow v$  或者  $v \rightarrow u$ , 即对于图中任意两点  $u, v$ , 存在一条  $u$  到  $v$  的有向路径或者从  $v$  到  $u$  的有向路径。
- 若  $G' = (V', E')$  满足,  $V'$  包含于  $V$ ,  $E'$  是  $E$  中所有跟  $V'$  关联的边, 则称  $G'$  是  $G$  的导出子图。
- 若  $G'$  是  $G$  的导出子图, 且  $G'$  为  $G$  的半连通子图。
- 若  $G'$  是  $G$  半连通子图中包含节点数最多的, 则称  $G'$  是  $G$  的最大半连通子图。
- 求有向图  $G$  的最大半连通子图所拥有的节点数和不同的最大半连通子图个数。

解法: 缩点 + DAG(最长路) DP

## 无向图的连通性

### 定义

设  $G = (V, E)$  是连通图

1. 若  $u$  是  $G$  的一个顶点, 且  $G - \{u\}$  不连通, 则称  $u$  是  $G$  的一个**割点**。  
若  $e$  是  $G$  的一条边, 且  $G - \{e\}$  不连通, 则称  $e$  是  $G$  的一条**割边**。
2. 若  $V_1$  是  $V$  的一个非空**真子集**,  $G - V_1$  不连通, 但对  $V_1$  的任何真子集  $V_2$  都有  $G - V_2$  连通, 则称  $V_1$  是  $G$  中的一个**点割集**。  
若  $E_1$  是  $E$  的一个非空**子集**,  $G - E_1$  不连通, 但对  $E_1$  的任何真子集  $E_2$  都有  $G - E_2$  连通, 则称  $E_1$  是  $G$  中的一个**边割集**。

## 性质

3. 割点构成只含一个点的点割集，割边构成只含一条边的边割集。
4. 对于不连通的图  $G$ ， $G$  的割点、割边、点割集、边割集分别是指  $G$  的某个连通分支的割点、割边、点割集、边割集。
5. 在集合的包含关系下，点割集和边割集都具有极小的性质。
6. 任何非平凡的连通图一定具有边割集。  
非平凡的连通图  $G$  存在点割集当且仅当  $G$  不是完全图。

## 连通度

7. 设  $G$  是一个非平凡的连通图，则我们称  $\kappa(G) = \min\{|V_1| : V_1 \text{ 是 } G \text{ 的点割集或 } G-V_1 \text{ 是平凡图}\}$  为  $G$  的**点连通度**，即  $\kappa(G)$  是使得  $G$  不连通或称为平凡图所必须删除的点的最少个数；称  $\lambda(G) = \min\{|E_1| : E_1 \text{ 是 } G \text{ 的割集}\}$  为  $G$  的**边连通度**，即  $\lambda(G)$  是使得  $G$  不连通所必须删除的边的最少条数。
8. 对于不连通图和平凡图，规定点连通度和边连通度都是 0
9. 定理：对于任意图  $G$ ，都有  $\kappa(G) \leq \lambda(G)$
10. 求图  $G$  的边连通度
  - a) 给定源汇点：每条边赋权值 1，求最大流
  - b) 不给定源汇点：用 **stoer-Wagner** 算法求全局最小割
11. 求图  $G$  的点连通度：构造一个流网络  $N$ 
  - a) 给定源汇点： $G$  中的每个顶点  $v$  拆成  $N$  中的两个点  $v_1, v_2$ ，连容量为 1 的边  $v_1 \rightarrow v_2$ ；对于  $G$  的每条有向边  $e=ab$  (无向边拆成两条)，在  $N$  中连弧  $a_2 \rightarrow b_1$ ，容量为  $\infty$ ；取  $s_2$  为源点， $t_1$  为汇点求最小割。
  - b) 不给定源汇点：枚举源汇点(已经通过 poj3921)

## 割点和桥

### Tarjan 算法

一个顶点  $u$  是割点，当且仅当满足(1)或(2)

(1) $u$  为树根，且  $u$  有多余 1 个子树

(2) $u$  不为树根，且存在树枝边  $(u, v)$ ，使得  $\text{dfn}(u) \leq \text{Low}(v)$

去掉  $u$  后分成连通块的个数

(1) $u$  为树根：子树的数目

(2) $u$  不为树根：满足  $\text{dfn}(u) \leq \text{Low}(v)$  的树枝边  $(u, v)$  的个数 + 1

一条无向边  $(u, v)$  是桥，当且仅当  $(u, v)$  为树枝边，且满足  $\text{dfn}(u) < \text{Low}(v)$

(前提是没有重边)

Tarjan( $u, f$ )

```

dfn[u] = Low[u] = ++index
for each (u, v) in E
    if v is not visited
        Tarjan(v, u)
        Low[u] = min(Low[u], Low[v])
        dfn[u] < Low[v] iff. (u, v)是桥
        if f != -1
            dfn(u) <= Low(v) iff. u 是割点
    else
        if v != f //v 不是 u 的父节点
            Low[u] = min(Low[u], dfn[v])
if f == -1
    u 是割点 iff. u 在搜索树中有至少两个子节点

```

## 求无向图的割点

参考 P27 的 tarjan 函数

//PKU1521 –  $O(n + m)$

const int maxn = 1000 + 9, maxm = 1000000 + 10;

struct node { int data; node \*next; } \*ge[maxn], pool[maxm], \*pooltop = pool;

int index = 0, existCutpoint;

int dfn[maxn], Low[maxn], subnetcnt[maxn];

```

void init(){
    memset(ge, 0, sizeof ge); memset(dfn, -1, sizeof dfn);
    for (int i = 0; i < maxn; i++) subnetcnt[i] = 1;
    pooltop = pool; index = 0; existCutpoint = 0;
}

```

void \_addedge(int a, int b){ node \*p = pooltop++; p->data = b; p->next = ge[a]; ge[a] = p;}

void addedge(int a, int b){ \_addedge(a, b); \_addedge(b, a);}

```

void tarjan(int u, int f){
    dfn[u] = Low[u] = ++index;
    int soncnt = 0;
    for (node *p = ge[u]; p; p = p->next){
        if (dfn[p->data] == -1){
            soncnt++;
            tarjan(p->data, u);
            Low[u] = min(Low[u], Low[p->data]);
            if (f != -1 && dfn[u] <= Low[p->data]) existCutpoint = 1, subnetcnt[u]++;
        }
    }
}

```



```

        else if (p->data != f)
            Low[u] = min(Low[u], dfn[p->data]);
    }
    if (f == -1)
        if (soncnt >= 2)
            existCutpoint = 1, subnetcnt[u] = soncnt;
}

int main(){
    int icalse = 0, a, b, ecnt = 0;
    for (;;) {
        init();
        ecnt = 0;
        while (scanf("%d", &a), a != 0) {
            scanf("%d", &b); addedge(a, b); ecnt++;
        }
        if (ecnt == 0) return 0;

        for (int i = 1; i <= 1000; i++)
            if (ge[i])
                tarjan(i, -1); break;

        printf("Network #%d\n", ++icalse);
        if (existCutpoint)
            for (int i = 1; i <= 1000; i++)
                if (ge[i] && subnetcnt[i] != 1)
                    printf("  SPF node %d leaves %d subnets\n", i, subnetcnt[i]);
        else
            printf("  No SPF nodes\n");
        printf("\n");
    }
}

```

## 求无向图的桥 - $O(E \log E)$

```

//已经通过 ZJU 2588 - 有重边 - Tarjan - 前向星
const int maxm = 100000 * 2 + 5, maxn = 10000 + 5;
struct Edge { int a, b, idx; } e[maxm];
bool operator < (const Edge& x, const Edge& y) { if (x.a == y.a) return x.b < y.b; return x.a < y.a; }
bool operator == (const Edge& x, const Edge& y) { return x.a == y.a && x.b == y.b; }
bool lessidx (const Edge& x, const Edge& y) { return abs(x.idx) < abs(y.idx); }
Edge *beg[maxn], *end[maxn];
int idx = 0, dfn[maxn], cnt, Low[maxn];

```

```

void Tarjan(int u, int f){
    dfn[u] = Low[u] = ++idx;
    for (Edge *p = beg[u]; p <= end[u]; p++){
        if (dfn[p->b] == -1){
            Tarjan(p->b, u); Low[u] = min(Low[u], Low[p->b]);
            if (dfn[u] < Low[p->b])
                if (p->idx > 0) p->idx = -p->idx, cnt++;
        }
        else if (p->b != f)
            Low[u] = min(Low[u], dfn[p->b]);
    }
}

int main(){
    int t; scanf("%d", &t);
    while (t--){
        int n, m, ecnt; scanf("%d%d", &n, &m);
        for (int i = 1; i <= m; i++){
            scanf("%d%d", &e[i].a, &e[i].b); e[i].idx = i;
            e[i+m].a = e[i].b; e[i+m].b = e[i].a; e[i+m].idx = i;
        }
        ecnt = 2 * m;
        sort(e + 1, e + 1 + ecnt);
        for (int i = 1; i <= n; i++){
            beg[i] = e + 1 + ecnt; end[i] = e; dfn[i] = -1;
        }
        e[0].a = -1; idx = 0; cnt = 0;
        for (int i = 1; i <= ecnt; i++){
            beg[e[i].a] = min(beg[e[i].a], &e[i]); end[e[i].a] = max(end[e[i].a], &e[i]);
            if (e[i] == e[i-1]) e[i].idx = e[i-1].idx = 0;
        }
        Tarjan(1, -1);
        sort(e + 1, e + 1 + ecnt, lessidx);
        printf("%d\n", cnt);
        if (cnt){
            for (int i = 1; i <= ecnt; i++)
                if (e[i].idx < 0){
                    printf("%d", -e[i].idx);
                    cnt--;
                    if (cnt) putchar(' ');
                }
            putchar('\n');
        }
        if (t) printf("\n");
    }
}

```

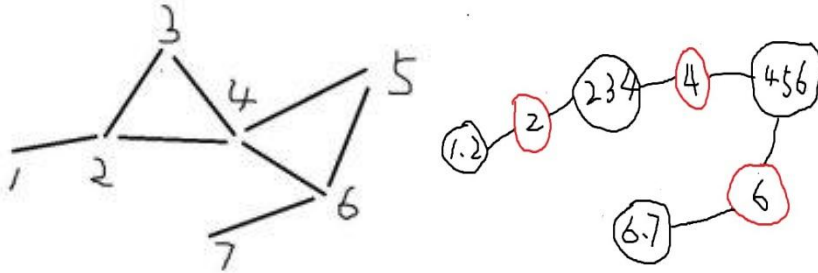
```

    }
}

```

## 点双连通分支(块)

在极大连通子图中，该连通分支的点连通度 $\geq 2$ ，即至少删除 2 点才能破坏其连通性



//已经通过 hdu4338(hdu 抽风，g++莫名其妙编译错误)，上次使用 2012-8-16

```
#define clr(a, v, n) memset( a, v, sizeof(a[0]) * (n+1) )
```

```
#define rep(i, n) for (int i = 0; i < (n); ++i)
```

```
#pragma comment(linker, "/STACK:102400000,102400000")
```

```
const int maxn = 100000 * 2 + 5, maxe = 200000 * 2 + 5;
```

```

struct Graph {
    struct node {
        int b; node *next;
    } *ge[maxn], pool[maxe], *pooltop;
    int n;

    void _addedge(node *ge[], int a, int b) {
        node *p = pooltop++; p->b = b; p->next = ge[a]; ge[a] = p;
    }

    void addedge2(int a, int b) {
        _addedge(ge, a, b); _addedge(ge, b, a);
    }

    void init(int n) {
        this->n = n; clr(ge, 0, n); pooltop = pool;
    }
};

```

```

struct Block : Graph {
    int idx, nBlock, dfn[maxn], low[maxn];
    int connectivity[maxn]; //点连通度
    int stk[maxn], *stktop;

```

```

vector<int> block[maxn];

void tarjan(int u, int f){
    dfn[u] = low[u] = ++idx; *stktop++ = u;
    int uf = 0;
    for (node *p = ge[u]; p; p = p->next) {
        if ( p->b == f && uf == 0 ) {
            ++uf; continue;
        } else if ( dfn[p->b] == -1 ) {
            tarjan(p->b, u); checkmin(low[u], low[p->b]);
            if ( dfn[u] <= low[p->b] ) {
                ++connectivity[u];
                block[nBlock].clear();
                for(int t; ){
                    block[nBlock].push_back( t = *--stktop );
                    if ( t == p->b ) break;
                };
                block[nBlock].push_back(u);
                ++nBlock;
            }
        } else {
            checkmin(low[u], dfn[p->b]);
        }
    }
}

void solve(){
    clr(dfn, -1, n); idx = 0; nBlock = 0; stktop = stk;
    rep(i, n) connectivity[i] = 1;
    rep(i, n) if ( dfn[i] == -1 ) {
        connectivity[i] = 0; tarjan(i, -1);
    }
}

};

struct Problem : Block {
    int belong[maxn], sz[maxn];
    int fa[20][maxn], tosz[20][maxn], dep[maxn], group[maxn], groupCnt;
    Graph graph;

    void dfs(int u, int f) {
        group[u] = groupCnt;
        fa[0][u] = f; tosz[0][u] = sz[u];
        for (int i = 1; i < 20; ++i) {

```

```

        if ( fa[i-1][u] != -1 ) {
            fa[i][u] = fa[i-1][ fa[i-1][u] ];
            totsiz[i][u] = totsiz[i-1][u] + totsiz[i-1][ fa[i-1][u] ];
        } else {
            fa[i][u] = -1;
            totsiz[i][u] = totsiz[i-1][u];
        }
    }
}

for (node *p = graph.ge[u]; p; p = p->next) {
    if (p->b == f) continue;
    dep[p->b] = dep[u] + 1;
    dfs(p->b, u);
}

}

void gao(){
    solve();
    graph.init(n + nBlock);
    rep(i, graph.n) belong[i] = i; // belong[i] == i iff. cutpoint
    rep(i, nBlock) rep( k, block[i].size() ){
        int j = block[i][k];
        if ( connectivity[j] >= 2 ) { // cutpoint
            graph.addedge2(n + i, j);
        } else {
            belong[j] = n + i; ++sz[n+i];
        }
    }
    rep(i, n) sz[i] = -1;
    rep(i, nBlock) sz[i+n] = block[i].size();
    clr(group, -1, graph.n); groupCnt = 0;
    rep(i, graph.n) if ( group[i] == -1 ) {
        dep[i] = 0; dfs(i, -1); ++groupCnt;
    }
}

int query(int a, int b) {
    if ( group[ belong[a] ] != group[ belong[b] ] ) return 0;
    if ( a == b ) return 1;
    int ans = (a == belong[a]) + (b == belong[b]);
    a = belong[a], b = belong[b];
    if ( dep[a] < dep[b] ) swap(a, b);
    for (int i = 19; i >= 0; --i) {
        if ( fa[i][a] != -1 && dep[ fa[i][a] ] >= dep[b] ) {
            ans += totsiz[i][a]; a = fa[i][a];
        }
    }
}

```

```

        }
    }
    if ( a == b ) {
        return ans += sz[a];
    }
    for (int i = 19; i >= 0; --i) {
        if ( fa[i][a] != fa[i][b] ) {
            ans += totsiz[i][a] + totsiz[i][b]; a = fa[i][a]; b = fa[i][b];
        }
    }
    return ans += sz[a] + sz[b] + sz[ fa[0][a] ];
}
} prob;

void solve(int n, int m) {
    prob.init(n);
    for (int a, b; m--;)
        scanf("%d%d", &a, &b), prob.addedge2(a, b);
    prob.gao();
    int q; cin >> q;
    static int ri = 0; printf("Case #d:\n", ++ri);
    for (int a, b; q--;)
        scanf("%d%d", &a, &b), printf( "%d\n", n - prob.query(a, b) );
    puts("");
}

```

## 找从 $s$ 到 $t$ 的割点和桥 - $O(n + m)$

找出  $s$ - $t$  的任何一条路径，与无向图的割点(桥)取交集即可  
求 DAG 上  $s$ - $t$  的桥参考第 61 页 BridgeST 类

## 无向图全局最小割 - Stoer-Wagner

算法框架：

1. 设当前找到的最小割 MinCut 为  $+\infty$
2. 在  $G$  中求出任意  $s$ - $t$  最小割  $C$ ， $\text{MinCut} = \min(\text{MinCut}, C)$
3. 在  $G$  作 Contract( $s, t$ )操作，得到  $G' = \{V', E'\}$ ，若  $|V'| > 1$ ，则  $G = G'$  并转 2，否则 MinCut 为原图的全局最小割

求  $G=(V, E)$  中任意  $s$ - $t$  最小割的算法

1. 令集合  $A = \{a\}$ ,  $a$  为  $V$  中任一点
2. 选取  $V - A$  中的  $w(A, x)$  最大的点  $x$  加入集合  $A$
3. 若  $|A| = |V|$  结束，否则转 2

令倒数第二个加入 A 的点为 s，最后一个加入 A 的点为 t，则 s-t 最小割为[V-{t}, {t}]

//已经通过 pku2914，时间复杂度  $O(n^3)$ ，上次使用：2012.5.3

```
const int maxn = 600, inf = 1 << 29;
int mat[maxn][maxn];

int mincut(int n){
    static int map[maxn], dist[maxn], visit[maxn];
    int ret = inf;
    for (int i = 0; i < n; i++) map[i] = i;
    while (n >= 2){
        for (int i = 0; i < n; i++){
            dist[i] = visit[i] = 0;
        }
        int cur = 0;
        visit[cur] = 1;
        for (int k = 1; k < n; k++){
            int maxval = -inf, maxiter;
            for (int i = 0; i < n; i++){
                if (!visit[i]){
                    dist[i] += mat[map[cur]][map[i]];
                    if (dist[i] > maxval) maxval = dist[i], maxiter = i;
                }
            }
            if (k == n - 1){
                ret = min(ret, maxval);
                int a = map[cur], b = map[maxiter];
                for (int i = 0; i < n; i++)
                    mat[map[i]][a] = mat[a][map[i]] += mat[b][map[i]];
                map[maxiter] = map[-n];
            }
            cur = maxiter; visit[cur] = 1;
        }
    }
    return ret;
}

int main(){
    int n, m, a, b, v;
    while (scanf("%d%d", &n, &m) != EOF){
        for (int i = 0; i <= n; i++)
            for (int j = 0; j <= n; j++)
                mat[i][j] = 0;
        while (m--){
            scanf("%d%d%d", &a, &b, &v), mat[a][b] = mat[b][a] += v;
        }
    }
}
```

```

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

```

## 欧拉图与欧拉路

1. 欧拉图：具有欧拉回路的图称为欧拉图。  
半欧拉图：具有欧拉通路而无欧拉回路的图持有恒威半欧拉图。
2. 定理：有向图  $G$  是欧拉图当且仅当  $G$  是强联通的且每个顶点的入度等于出度。  
有向图  $G$  是半欧拉图当且仅当  $G$  是单向联通的，且  $G$  中恰有两个奇度顶点，其中一个入度比出度大 1，另一个出度比入度大 1，而剩余顶点的入度都等于出度。
3. 求欧拉回路的 Fleury 算法(略)。

## 有向图和无向图和混合图的欧拉路 - 缺

## 二分图

## 二分图最大匹配

## KM 算法 - 时间复杂度 $O(n^3)$

```

struct Hungray {
    const static int maxnx = 64;
    const static int maxny = 64 * 64;

    int nx, ny, cx[maxnx], cy[maxny];
    vector<int> ge[maxnx];
    bool vis[maxnx];

    void init(int nx, int ny) {
        this->nx = nx; this->ny = ny;
        for (int i = 0; i < nx; ++i) ge[i].clear();
    }

    void addedge(int a, int b) {
        ge[a].push_back(b);
    }

    bool augment(int i) {

```



```

        if ( !vis[i] ) {
            vis[i] = true;
            for (vector<int>::const_iterator j = ge[i].begin(); j != ge[i].end(); ++j) {
                if ( cy[*j] == -1 || augment(cy[*j]) ) {
                    cx[i] = *j; cy[*j] = i; return true;
                }
            }
        }
        return false;
    }

    int gao() {
        int ret = 0;
        fill(cx, cx + nx, -1); fill(cy, cy + ny, -1);
        for (int i = 0; i < nx; ++i) {
            fill(vis, vis + nx, false);
            ret += augment(i);
        }
        return ret;
    }
};

```

## Hopcroft–Karp 算法 - 时间复杂度 $O(|V|^{0.5}|E|)$

对于两边各  $5e4$  个点， $2e5$  条边的二分图最大匹配可以在 1s 内得到解，上次使用 2012.6.8

```

#define REP(i, n) for (int i = 0; i < (n); ++i)
#define CLR(a, v, n) memset( a, v, sizeof(a[0]) * (n) )

```

```

struct node {
    int b; node *next;
    node &init(int b, node *next){
        this->b = b; this->next = next; return *this;
    }
};

```

```

struct HK {
    const static int maxnx = 10000 + 5; //左部
    const static int maxny = 10000 + 5; //右部
    const static int maxe = 100000 * 2 + 5;

    node *ge[maxnx], pool[maxe], *pooltop;
    int nx, ny;
    int cx[maxnx], cy[maxny]; //左部右部所匹配的元素
    int dx[maxnx], dy[maxny];

```

```

void init(int nx, int ny){
    this->nx = nx; this->ny = ny;
    pooltop = pool; CLR(ge, 0, nx);
}

node *_addedge(node *ge[], int a, int b){
    return ge[a] = &(*pooltop++).init(b, ge[a]);
}

node *addedge(int a, int b){
    return _addedge(ge, a, b);
}

bool bfs() {
    static int q[maxn], *qtop, *qbase;
    qbase = qtop = q;
    CLR( dx, 0, nx ); CLR(dy, 0, ny );
    REP(i, nx) if ( -1 == cx[i] ) *qtop++ = i;
    bool found = false;
    while ( qbase != qtop ){
        int a = *qbase++;
        for (node *p = ge[a]; p; p = p->next){
            int b = p->b;
            if ( dy[b] == 0 ){
                dy[b] = dx[a] + 1;
                if ( -1 == cy[b] ){
                    found = true;
                } else {
                    dx[ cy[b] ] = dy[b] + 1; *qtop++ = cy[b];
                }
            }
        }
    }
    return found;
}

bool dfs(int a){
    for (node *p = ge[a]; p; p = p->next){
        int b = p->b;
        if ( dx[a] + 1 == dy[b] ){
            dy[b] = 0; //表示 b 点已经被访问过了
            if ( -1 == cy[b] || dfs(cy[b]) ){
                cx[a] = b; cy[b] = a; return true;
            }
        }
    }
}

```

```

        }
    }
}
return false;
}

int MMG(){
    CLR(cx, -1, nx); CLR(cy, -1, ny);
    int ans = 0;
    while ( bfs() ) REP(i, nx) if (-1 == cx[i] && dfs(i)) ans++;
    return ans;
}
};

struct Kosaraju {
    const static int maxn = HK::maxnx + HK::maxny + 5;
    const static int maxe = (HK::maxe + maxn) * 2 + 5;

    node *ge[maxn], *grev[maxn], pool[maxe], *pooltop;
    int n, nScc, group[maxn], vis[maxn], order[maxn];

    void init(int n) {
        this->n = n; pooltop = pool; CLR(ge, 0, n); CLR(grev, 0, n);
    }

    node *_addedge(node *ge[], int a, int b){
        return ge[a] = &(*pooltop++).init(b, ge[a]);
    }

    void addedge(int a, int b){
        _addedge(ge, a, b); _addedge(grev, b, a);
    }

    void dfs(int a){
        vis[a] = 1;
        for (node *p = ge[a]; p; p = p->next) if ( !vis[p->b] ) dfs(p->b);
        order[nScc++] = a;
    }

    void dfsrev(int a){
        vis[a] = 1; group[a] = nScc;
        for (node *p = grev[a]; p; p = p->next) if ( !vis[p->b] ) dfsrev(p->b);
    }
}

```

```

int solve(){
    nScc = 0; CLR(vis, 0, n);
    REP(i, n) if (!vis[i]) dfs(i);
    nScc = 0; CLR(vis, 0, n);
    for (int i = n-1; i >= 0; --i) {
        int b = order[i];
        if ( !vis[b] ){
            dfsrev(b); ++nScc;
        }
    }
    return nScc;
}
};

struct KeyEdges : HK {
    Kosaraju scc;

    int solve(){
        MMG();
        int s = nx + ny, t = s + 1, ans = 0;
        scc.init(t + 1);
        REP(i, nx){
            for (node *p = ge[i]; p; p = p->next){
                if ( cx[i] != p->b ){
                    scc.addedge(i, nx + p->b);
                } else {
                    scc.addedge(nx + p->b, i);
                }
            }
            if (cx[i] == -1) scc.addedge(s, i); else scc.addedge(i, s);
        }
        REP(j, ny){
            if (cy[j] == -1) scc.addedge(j + nx, t); else scc.addedge(t, j + nx);
        }
        scc.solve();
        REP(i, nx) if ( cx[i] != -1 && scc.group[i] != scc.group[ cx[i] + nx ] ) ++ans;
        return ans;
    }
};

```

## 二分图最佳匹配 – Kuhn-Munkr 算法 – 时间复杂度 $O(n^3)$

已经通过 3 题，上次使用 2012.7.4:

只需传入 `w[][]` 和 `nx, ny` 的值, 下标从 1 开始(否则把所有循环改为 `0...n-1` 即可), 要求 `nx <= ny`

```
namespace optmatch{
    const int maxnx = 100, maxny = 100 * 100;
    int w[maxnx][maxny], nx, ny;
    int vx[maxnx], vy[maxny]; //visited
    int slack[maxny], Lx[maxnx], Ly[maxny], cy[maxny];

    bool dfs(int u){
        vx[u] = 1;
        FOR(v, 1, ny) {
            int wt = Lx[u] + Ly[v] - w[u][v];
            if (!vy[v] && wt == 0) {
                vy[v] = 1;
                if (cy[v] == -1 || dfs(cy[v])){
                    cy[v] = u; return 1;
                }
            } else {
                chkmin(slack[v], wt);
            }
        }
        return 0;
    }

    int km(){
        for (int i = 1; i <= nx; i++)
            Lx[i] = -inf;
        for (int j = 1; j <= ny; j++)
            Ly[j] = 0, cy[j] = -1;
        for (int i = 1; i <= nx; i++)
            for (int j = 1; j <= ny; j++)
                chkmax(Lx[i], w[i][j]);
        FOR(u, 1, nx) {
            FOR(j, 1, ny) slack[j] = inf;
            for (;;) {
                CLR(vx, 0, nx + 1); CLR(vy, 0, ny + 1);
                if (dfs(u)) break;
                int d = inf;
                FOR(j, 1, ny) if (!vy[j]) checkmin(d, slack[j]);
                FOR(i, 1, nx) if (vx[i]) Lx[i] -= d;
                FOR(j, 1, ny) if (vy[j]) Ly[j] += d; else slack[j] -= d;
            }
        }
        int ret = 0;
        FOR(j, 1, ny) if (cy[j] != -1) ret += w[cy[j]][j];
    }
}
```

```

        return ret;
    }
}

```

## 稳定婚姻 – $O(n^2)$

//已经通过南开 1710

```

const int maxn = 100 + 1;
int heroines[maxn][maxn]; //对第 i 号男性 第 j 爱的女性的编号
int rank[maxn][maxn]; //第 i 号男性, 在第 j 号女性心目中的排名

```

//Stable Marriage Problem

```

void GaleShapley(int n, int bf[], int gf[]){ //延迟认可算法
    static int profession[maxn]; //第 i 号男性已经向 profession[i] 名女性表白
    memset(gf, -1, (n+1) * sizeof(gf[0])); memset(bf, -1, (n+1) * sizeof(bf[0]));
    memset(profession, 0, (n+1) * sizeof(profession[0]));
    bool ok = false;
    int partner;
    while (!ok){
        ok = true;
        for (int i = 1; i <= n; i++){
            if (gf[i] == -1){
                ok = false;
                partner = heroines[i][++profession[i]];
                if (bf[partner] == -1 || rank[i][partner] < rank[bf[partner]][partner]){
                    if (bf[partner] != -1)
                        gf[bf[partner]] = -1;
                    gf[i] = partner; bf[partner] = i;
                }
            }
        }
    }
}

```

```

int gf[maxn]; //第 i 号男性的女朋友, -1 表示单身
int bf[maxn]; //第 i 号女性的男朋友, -1 表示单身

```

```

int main(){
    int x, n;
    int icalse = 0;
    while (cin >> n){
        for (int j = 1; j <= n; j++)
            for (int i = 1; i <= n; i++)
                cin >> rank[i][j];
    }
}

```

```

        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                cin >> x, heroines[i][x] = j;
    GaleShapley(n, bf, gf);
    cout << "Case " << ++icase << ":" << endl;
    for (int i = 1; i <= n; i++){
        cout << gf[i]; if (i != n) cout << " ";
    }
    cout << endl;
}
}

```

## 关键点 and 关键边

**关键点：**若删去这个点，就会使得最大匹配数减小。

**关键边：**若删去这条边，就会使得最大匹配数减小。

关键边的条数是小于最大匹配的。

## 算法

**关键点：**对  $X$  集合所有的点，如果  $cx[i] = 0$ ，表示  $i$  必为非关键点；否则用  $DelY$  记录  $cx[i]$ ，表示暂时删除与  $i$  匹配的点，沿  $i$  再去找一条增广路，若成功，说明  $DelY$  可有可无，存在与否不影响最大匹配数的大小，即非关键点，否则将  $DelY$  记录为关键点，这样可以找出所有  $Y$  集合中的关键点。在  $G$  的逆图中进行前面类似的操作，即可找出所有  $X$  集合中的关键点。

**关键边：**详见第 35 页 KM 求二分图最大匹配

## 关键点代码

```

//只需传入 nx, ny, g, 下标从 1 开始, 对 nx, ny 大小关系无特殊要求
const int maxn = 100;
int nx, ny, g[maxn][maxn], visy[maxn];
int cx[maxn], cy[maxn]; // i 与 cx[i] 匹配, cy[j] 与 j 匹配, 如不匹配 cx[i] or cy[j] = 0

int find(int u){
    for (int v = 1; v <= ny; v++){
        if (g[u][v] && !visy[v]){
            visy[v] = 1;
            if (!cy[v] || find(cy[v])){
                cx[u] = v; cy[v] = u; return 1;
            }
        }
    }
}

```

```

    }
    return 0;
}

int MaximumMatch(){
    int ret = 0;
    memset(cx, 0, sizeof cx); memset(cy, 0, sizeof cy);
    for (int i = 1; i <= nx; i++){
        if (!cx[i]){
            memset(visy, 0, sizeof visy); ret += find(i);
        }
    }
    return ret;
}

int keyx[maxn], keyy[maxn], visx[maxn];
int delX, delY;

int findY(int u){
    for (int v = 1; v <= ny; v++){
        if (v != delY && !visy[v] && g[u][v]){
            visy[v] = 1; if (!cy[v] || findY(cy[v])) return true;
        }
    }
    return 0;
}

int findX(int v){
    for (int u = 1; u <= nx; u++){
        if (u != delX && !visx[u] && g[u][v]){
            visx[u] = 1; if (!cx[u] || findX(cx[u])) return 1;
        }
    }
    return 0;
}

void keyPoints(){
    MaximumMatch();
    for (int i = 1; i <= nx; i++){
        if (!cx[i])
            keyx[i] = 0;
        else {
            delY = cx[i];
            memset(visy, 0, sizeof visy);

```



```

        keyy[delY] = !findY(i);
    }
}
for (int j = 1; j <= ny; j++){
    if (!cy[j])
        keyy[j] = 0;
    else {
        delX = cy[j];
        memset(visx, 0, sizeof visx);
        keyx[delX] = !findX(j);
    }
}
}
}

```

## 二分图判断 - bfs - $O(V + E)$

```

bool isbigraph(int n, int M){
    //判断一个无向图是否可以 2-染色(bfs), 权值<=M 的边不予考虑(已经通过 cugb1253)
    static int queue[maxn], *qbase, *qtop;
    static char flag[maxn];
    qbase = qtop = queue;
    CLR(flag + 1, -1, n);
    for (int i = 1; i <= n; i++){ //这层循环一定要有
        if (flag[i] == -1){
            *qtop++ = i;
            flag[i] = 0;
            for (; qbase != qtop;){
                int u = *qbase++;
                for (node *p = ge[u]; p; p = p->next){
                    if (p->val <= M) continue;
                    int v = p->adj;
                    if (flag[u] == flag[v]) return 0;
                    if (flag[v] == -1){
                        flag[v] = flag[u] ^ 1; *qtop++ = v;
                    }
                }
            }
        }
    }
    return 1;
}

```

# 一般图最大匹配 – 带花树

Ref. <http://hi.baidu.com/edwardmj/blog/item/022b9941915f95059313c6b3.html>

//Ural 1099, 此外通过 bzoj2184( $n = 1000, m = 3000$ )

```
const int maxn = 250;
```

```
int n;
```

```
bool g[maxn][maxn];
```

```
int match[maxn];
```

```
void createGraph(){
```

```
    int x, y;
```

```
    scanf("%d", &n);
```

```
    while (scanf("%d%d", &x, &y) != EOF)
```

```
        g[x][y] = g[y][x] = 1;
```

```
}
```

```
int father[maxn]; /*就是增广路的 father*/
```

```
int base[maxn]; /*该点属于哪朵花*/
```

```
int que[maxn], *qbeg, *qend;
```

```
bool inqueue[maxn];
```

```
char mark[maxn]; /* 不要用 bool */
```

```
void BlossomContract(int x, int y){ //O(n)
```

```
    static bool inblossom[maxn];
```

```
    fill(mark, mark + n + 1, 0);
```

```
    fill(inblossom, inblossom + n + 1, 0);
```

```
    #define pre(_i) father[match[_i]]
```

```
    int lca = -1;
```

```
    int i, j;
```

```
    for (i = x, j = y; i && j; i = pre(i), j = pre(j)){
```

```
        i = base[i]; j = base[j];
```

```
        mark[i]++; mark[j]++;
```

```
        if (mark[i] == 2) {lca = i; break;}
```

```
        if (mark[j] == 2) {lca = j; break;}
```

```
    }
```

```
    if (lca == -1){
```

```
        for (; i = pre(i)){
```

```
            i = base[i]; mark[i]++;
```

```
            if (mark[i] == 2) {lca = i; break;}
```

```
        }
```

```
    }
```

```
    if (lca == -1){
```

```
        for (; j = pre(j)){
```

```

        j = base[j]; mark[j]++;
        if (mark[j] == 2) {lca = j; break;}
    }
}
for (int i = x; base[i] != lca; i = pre(i)){
    if (base[pre(i)] != lca)
        father[pre(i)] = match[i];
    inblossom[base[i]] = 1;
    inblossom[base[match[i]]] = 1;
}
for (int i = y; base[i] != lca; i = pre(i)){
    if (base[pre(i)] != lca)
        father[pre(i)] = match[i];
    inblossom[base[i]] = 1;
    inblossom[base[match[i]]] = 1;
}
#undef pre
if (base[x] != lca) father[x] = y;
if (base[y] != lca) father[y] = x;
for (int i = 1; i <= n; i++){
    if (inblossom[base[i]]){
        base[i] = lca;
        if (!inqueue[i]){
            *qend++ = i; inqueue[i] = 1;
        }
    }
}
}

void change(int t){
    int x, y, z = t;
    while (z){
        y = father[z]; x = match[y];
        match[y] = z; match[z] = y;
        z = x;
    }
}

void augment(int s){ //O(E + kV), k - 奇环个数
    fill(father, father + n + 1, 0);
    fill(inqueue, inqueue + n + 1, 0);
    for (int i = 1; i <= n; i++)
        base[i] = i;
    qbeg = qend = que;
}

```

```

*qend++ = s; inqueue[s] = 1;
while (qbeg != qend){
    int x = *qbeg++;
    for (int y = 1; y <= n; y++){
        if (!g[x][y]) continue;
        if (base[x] == base[y]) continue;
        if (match[x] == y) continue;

        if (s == y || match[y] && father[match[y]]){
            BlossomContract(x, y);
        } else if (!father[y]){
            father[y] = x;
            if (match[y]){
                *qend++ = match[y];
                inqueue[match[y]] = 1;
            } else {
                change(y);
                return;
            }
        }
    }
}
}
}

```

```

void edmonds(){
    //时间复杂度  $O(VE + k * V^2) = O(V^3)$ 
    fill(match, match + n + 1, 0);
    for (int s = 1; s <= n; s++)
        if (!match[s])
            augment(s);
}

```

```

void output(){
    fill(mark, mark + n + 1, 0);
    int cnt = 0;
    for (int i = 1; i <= n; i++)
        if (match[i]) cnt++;
    printf("%d\n", cnt);
    for (int i = 1; i <= n; i++){
        if (!mark[i] && match[i]){
            mark[i] = 1; mark[match[i]] = 1;
            printf("%d %d\n", i, match[i]);
        }
    }
}

```

```

}

int main(){
    createGraph();
    edmonds();
    output();
}

```

## 团

### 最大团

### 伪代码

```

Var Max, Found, c[1..n]
Function dfs(U, Size)
    If  $|U| = 0$  then
        If Size > Max then
            Max := Size
            New record; Save it.
            Found := true
        End if
        Return
    End if
    While  $U \neq \emptyset$  do
         $i := \min\{j \mid v_j \in U\}$ 
        if  $\text{Size} + C[i] \leq \text{Max}$  or  $\text{Size} + |U| \leq \text{Max}$  then return
         $U := U \setminus \{v_i\}$ 
        dfs( $U \cap \text{Adj}(v_i)$ , size+1)
        if found = true then return
    end while
Function MaxClique
    Max := 0
    For i:=n downto 1 do
        Found := false
        Dfs( $S_i \cap \text{Adj}(v_i)$ , 1)
        C[i] := Max
    End for

```

其中

$S_i = \{v_i, v_{i+1}, \dots\}$

$C[i]$  :  $S_i$  的最大团中点的个数

Adj(v) – 与点 v 相邻的点的集合  
n 图的顶点数

## 代码

```
//已经通过 zju1492, hdu3585
const int maxn = 50;

int g[maxn][maxn];
int C[maxn], Max, Found;
int tmp[maxn], ans[maxn];

void dfs(int v[], int vc, int size){
    if (vc == 0){
        if (size > Max){
            Max = size; Found = true;
            for (int i = 0; i < size; i++) ans[i] = tmp[i];
        }
        return;
    }

    int *v2 = v + vc;
    while (vc != 0){
        if (size + vc <= Max) return;
        int i = v[--vc];
        if (size + C[i] <= Max) return;

        int vc2 = 0;
        for (int j = 0; j < vc; j++)
            if (g[i][v[j]])
                v2[vc2++] = v[j];
        tmp[size] = i;
        dfs(v2, vc2, size+1);
        if (Found) return;
    }
}

int maxclique(int n, int limit = maxn){
    //只要最大团点数大于等于 limit, 则返回 limit
    static int stk[maxn * maxn];
    int *v = stk, vc;

    Max = 0;
    for (int i = n-1; i >= 0; i--){
```

```

        Found = false; vc = 0;
        for (int j = n-1; j > i; j--)
            if (g[i][j])
                v[vc++] = j;
        tmp[0] = i;
        dfs(v, vc, 1);
        C[i] = Max;
        if (Max >= limit) return limit;
    }
    return C[0];
}

int main(){
    int n;
    while (cin >> n, n != 0){
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                cin >> g[i][j];
        cout << maxclique(n) << endl;
    }
}

```

## 极大团 - Bron-Lerbosch algorithm

### 伪代码

```

Init:  $P = V, R = X = \phi$ 
BronKerbosch( $R, P, X$ )
    If  $P$  and  $X$  are both empty:
        Report  $R$  as a maximal clique
    Choose a pivot vertex  $u$  in  $P \cup X$ 
    For each vertex  $v$  in  $P \setminus N(u)$ 
        BronKerbosch( $R \cup \{v\}, P \cap N(v), X \cap N(v)$ )
         $P := P \setminus \{v\}$ 
         $X := X \cup \{v\}$ 

```

### 代码

Pku2989

```

const int maxn = 128;
int joint[maxn][maxn], n, ans;

```

```

void BronKerbosch(int R[], int Rs, int P[], int Ps, int X[], int Xs){
    if (Ps == 0 && Xs == 0){
        ans++; return;
    }

    int v, u, i, P2[maxn], X2[maxn], Rs2, Ps2, Xs2, ret = 0;

    u = rand() % (Ps + Xs);
    u = ((u < Ps) ? P[u] : X[u - Ps]);

    i = 0;
    while (i < Ps){
        v = P[i];
        if (joint[v][u] == 0){
            Rs2 = Rs;
            R[Rs2++] = v;

            Ps2 = 0;
            for (int j = 0; j < Ps; j++)
                if (joint[P[j]][v])
                    P2[Ps2++] = P[j];

            Xs2 = 0;
            for (int j = 0; j < Xs; j++)
                if (joint[X[j]][v])
                    X2[Xs2++] = X[j];

            BronKerbosch(R, Rs2, P2, Ps2, X2, Xs2);
            if (ans > 1000) return; ///////
            X[Xs++] = v;

            swap(P[i], P[--Ps]);
        } else
            i++;
    }
}

```

```

int main(){
    int m, a, b;
    static int R[maxn], P[maxn], X[maxn];
    srand(2365362);
    while (scanf("%d%d", &n, &m) != EOF){
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)

```



```

        joint[i][j] = 0;
    while (m--){
        scanf("%d%d", &a, &b); a--; b--; joint[a][b] = joint[b][a] = 1;
    }
    for (int i = 0; i < n; i++) P[i] = i;

    ans = 0;
    BronKerbosch(R, 0, P, n, X, 0);

    if (ans > 1000)
        printf("Too many maximal sets of friends.\n");
    else
        printf("%d\n", ans);
}
}

```

## 图和树的同构

### 有根树的同构

可以使用一种类似树状递推的方法来计算 Hash 值:

对于一个节点  $v$ , 先求出它所有儿子节点的 Hash 值, 并从小到大**排序**, 记作  $H_1, H_2, \dots, H_D$ . 那么  $v$  的 Hash 值就可以计算为:

$$\text{Hash}(v) = (((\dots((((a \cdot p \text{ xor } H_1) \bmod q) \cdot p \text{ xor } H_2) \bmod q) \cdot p \text{ xor } \dots H_{D-1}) \bmod q) \cdot p \text{ xor } H_D) \cdot b \bmod q$$

换句话说, 就是从某个常数开始, 每次乘以  $p$ , 和一个元素异或, 再除以  $q$  取余, 再乘以  $p$ , 和下一个元素异或, 除以  $q$  取余.....一直进行到最后一个元素为止。最后把所得到的结果乘以  $b$ , 再对  $q$  取余。

现在, 只需要比较两棵树的 Hash 值, 即可分辨它们是否相同。时间复杂度为  $O(n \log n)$ 。

### 无根树的同构

取无根树的核(距叶子最远的点, 最多两个)作为根, 用有根树同构的方法解答。

### 图的同构

有一个实践中没有遇到过问题的算法:

先枚举一个起始点  $i$ , 所有结点的 Hash 值置为某个常数 (不妨取 1)。然后进行迭代, 每次一个节点新的 Hash 值计算为:

$$f_{j+1}(v) = \left[ a \cdot f_j(v) + b \cdot \sum_{w \in V, v \rightarrow w \in E} f_j(w) + c \cdot \sum_{w \in V, w \rightarrow v \in E} f_j(w) + g(v) \right] \bmod p$$

$$\text{这里 } g(v) = \begin{cases} d & v = i \\ 0 & v \neq i \end{cases}$$

然后迭代  $k$  次，计算出的  $f_k(i)$  就是  $i$  点的 Hash 值。

其实关键就是在于  $f_i(i)$  的计算和其他值不同，即可避免很多特殊情况。

## SPFA & dijkstra & Floyd

### SPFA 的两个优化

优化后只是实际效果很好，但是优化后他已经不是 bellman-ford 了，所以实际上已经不保证复杂度了，注意 SPFA 很容易退化。

#### SLF: Small Label First 策略

设队首元素为  $i$ ，队列中要加入节点  $j$ ，在  $d_j \leq d_i$  时加到队首，否则和普通的 SPFA 一样加到队尾。

#### LLL: Large Label Last 策略

设队列  $Q$  中的队首元素为  $i$ ，距离标号的平均值为

$$\bar{d} = \frac{\sum_{j \in Q} d_j}{|Q|}$$

每次出队时，若  $d_i > \bar{d}$ ，则把  $i$  移到队列末尾，如此反复，直到找到一个  $i$  使  $d_i \leq \bar{d}$ ，将其出队。

### SPFA 求最短路

```
//已经通过 cugb1077
const int maxn = 10000, maxe = 100000 * 2;
const LL inf = 1LL << 60;
struct node { int adj, w; node *next; } *ge[maxn+1], pool[maxe], *pooltop;

void init(int n){
    for (int i = 0; i <= n; i++) ge[i] = 0;
    pooltop = pool;
```

```

}

inline void addEdge(int a, int b, int w){
    node *p = pooltop++; p->adj = b; p->w = w; p->next = ge[a]; ge[a] = p;
}

inline void addBiEdge(int a, int b, int w){ addEdge(a, b, w); addEdge(b, a, w);}

LL spfa(int n, int s, int t){
    static int queue[maxn], *qbase, *qtop, inqueue[maxn];
    static LL dist[maxn];
#define enqueue(x) {*qtop++ = x; if (qtop == queue + maxn) qtop = queue; inqueue[x] = 1;}
#define dequeue(x) {x = *qbase++; if (qbase == queue + maxn) qbase = queue; inqueue[x] = 0;}
    qbase = qtop = queue; enqueue(s);
    for (int i = 0; i <= n; i++) dist[i] = inf;
    dist[s] = 0;
    while (qbase != qtop){
        int u; dequeue(u);
        for (node *p = ge[u]; p; p = p->next){
            if (dist[p->adj] > dist[u] + p->w){
                dist[p->adj] = dist[u] + p->w;
                if (!inqueue[p->adj]) enqueue(p->adj);
            }
        }
    }
    return dist[t];
}

```

## A\*求 k 短路(可以有环)

```

const int maxn = 10000 + 10, maxe = 100000, inf = 1000000000;
struct edge { int adj, w; edge *next;} *ge[maxn], *gerev[maxn], pool[maxe * 2], *pooltop;

inline void _adddedge(edge *&ge, int adj, int w){
    edge *p = pooltop++; p->adj = adj; p->w = w; p->next = ge; ge = p;
}

inline void addedge(int a, int b, int d){ _adddedge(ge[a], b, d); _adddedge(gerev[b], a, d); }

void spfa(edge *ge[], int n, int s, int dist[]){
    static int queue[maxn], *qbase, *qtop, inqueue[maxn];
#define enqueue(x) {*qtop++ = x; if (qtop == queue + maxn) qtop = queue; inqueue[x] = 1;}
#define dequeue(x) {x = *qbase++; if (qbase == queue + maxn) qbase = queue; inqueue[x] = 0;}
    qbase = qtop = queue; enqueue(s);
    for (int i = 0; i <= n; i++) dist[i] = inf;
}

```

```

dist[s] = 0;
while (qbase != qtop){
    int u; dequeue(u);
    for (edge *p = ge[u]; p; p = p->next){
        if (dist[p->adj] > dist[u] + p->w){
            dist[p->adj] = dist[u] + p->w;
            if (!inqueue[p->adj]) enqueue(p->adj);
        }
    }
}

}

struct node {
    int id, fx, gx;
    /* gx = hx(该点到终点的长度) + fx(当前走过的路径长度) */
    friend bool operator < (const node &a, const node &b){
        if (a.gx == b.gx) return a.fx > b.fx; else return a.gx > b.gx;
    }
} heap[2000010], *heaptop;

int kth_shortest(int n, int s, int t, int k, int dist[]){
    static int c[maxn];
    spfa(gerev, n, t, dist);
    if (dist[s] == inf) return -1;
    heaptop = heap;
    #define enheap(_id, _fx) { heaptop->id = _id; heaptop->fx = _fx; \
        heaptop->gx = heaptop->fx + dist[_id]; push_heap(heap, ++heaptop); }
    enheap(s, 0);
    for (int i = 0; i <= n; i++) c[i] = 0;
    while (heap != heaptop){
        int id = heap[0].id, fx = heap[0].fx, gx = heap[0].gx;
        ++c[id];
        if (c[id] > k) continue;
        if (c[t] == k) return fx;
        pop_heap(heap, heaptop--);
        for (edge *p = ge[id]; p; p = p->next)
            enheap(p->adj, fx + p->w);
    }
    return -1;
}

int dist[maxn]; /* dist[i]为点 i 到终点的距离 */

int main(){

```

```

int n, m, k, a, b, d, s, t;
while (scanf("%d%d%d", &n, &m, &k) != EOF){
    pooltop = pool;
    for (int i = 0; i <= n; i++)
        ge[i] = gerev[i] = 0;
    while (m--){
        scanf("%d%d%d", &a, &b, &d); --a; --b; addedge(a, b, d);
    }
    s = 0, t = n-1;
    cout << kth_shortest(n, s, t, k, dist) << endl;
}
}

```

## K 短路 Yen 无环

```

const int maxn = 50 + 5;
const int inf = 0x3f3f3f3f;

struct Node {
    int a, val, blockInd;
    Node *next;
};

struct Path {
    int n, dev;
    vector<int> node;
    vector<Node *> edge, block;
    int len; //路径长度
    bool operator < (const Path& p) const {
        return len != p.len ? len > p.len : node > p.node;
    }
    void print() const {
        for (int i = 0; i < n-1; i++)
            printf("%d-", node[i] + 1);
        printf("%d\n", node[n-1] + 1);
    }
};

struct PathLess{
    bool operator()(int i, int j) const { return p[i] < p[j]; }
    PathLess(Path p[]){ this->p = p; }
    private: const Path *p;
};

```

```

struct Graph {
    void init(int n) {
        this->n = n; pooltop = pool; blockInd = 1; memset(gInv, 0, sizeof gInv );
    }

    void addEdge(int a, int b, int v){ //实际内部是反向加边
        Node *p = pooltop++; p->blockInd = 0;
        p->a = a; p->val = v; p->next = gInv[b]; gInv[b] = p;
    }

    void yenLoopLess(int source, int sink, int k, vector<int> &res, vector<Path> &candidates){
        candidates.clear(); candidates.reserve(k);
        res.clear(); res.reserve(k);
        vector<int> q; q.reserve(k);
        #define enqueue(_x) { q.push_back( candidates.size() ); candidates.push_back(_x); \
            push_heap( q.begin(), q.end(), PathLess( &candidates[0] ) ); \
        }
        initSingleSrc(sink); dijkstra();
        if ( d[source] < inf ){
            Path sh = shortestPath(source);
            sh.dev = 1; sh.block.push_back( sh.edge[ sh.n-1-sh.dev ] );
            enqueue(sh);
        }
        while ( res.size() < k && !q.empty() ){
            const Path path = candidates[ q.front() ]; //不要加&
            res.push_back( q.front() );
            pop_heap( q.begin(), q.end(), PathLess(&candidates[0]) ); q.pop_back();
            for (int dev = path.dev; dev < path.node.size(); dev++){
                if ( dev == path.dev ){
                    for (int i = 0; i < path.block.size(); i++){
                        path.block[i]->blockInd = blockInd;
                    }
                } else {
                    path.edge[ path.n-1-dev ]->blockInd = blockInd;
                }
                initSingleSrc(sink); delSubpath(path, dev); dijkstra();
                if ( d[source] < inf ){
                    Path newP = shortestPath(source);
                    newP.dev = dev;
                    if ( dev == path.dev ){
                        newP.block = path.block;
                    } else {
                        newP.block.push_back( path.edge[path.n-1-dev] );
                    }
                }
            }
        }
    }
}

```

```

        newP.block.push_back( newP.edge[newP.n-1-dev] );
        enqueue(newP);
    }
}
++blockInd; //clear block
}
}
private:
    Node *gInv[maxn], pool[maxn * maxn], *pooltop;
    int n, blockInd;

    Path shortestPath(int v) const {
        Path res;
        res.len = d[v];
        for (; v != -1; v = f[v]){
            res.node.push_back(v); res.edge.push_back(e[v]);
        }
        res.n = res.node.size();
        return res;
    }

    // [[[以下是 dijkstra!]]]
    int d[maxn], f[maxn];
    Node *e[maxn];
    bool vis[maxn];

    void initSingleSrc(int s){
        for (int i = 0; i <= n; i++){
            d[i] = inf; f[i] = -1; vis[i] = 0; e[i] = 0;
        }
        d[s] = 0;
    }

    void delSubpath(const Path& path, int dev) { //特有函数，将路径首先加入 dij
        int pre = path.node[path.node.size()-1], cur;
        for (int i = 1; i < dev; i++, pre = cur){
            cur = path.node[path.node.size()-1-i];
            f[ cur ] = pre;
            e[ cur ] = path.edge[path.edge.size()-1-i];
            d[ cur ] = d[ pre ] + e[cur]->val;
            vis[ pre ] = true;
        }
    }
}

```

```

void dijkstra(){ //不能处理负权
    for (;){
        int u = -1;
        for (int i = 0; i < n; i++)
            if ( !vis[i] && (-1 == u || d[i] < d[u]) )
                u = i;
        if (-1 == u) return;
        vis[u] = true;
        for (Node *p = gInv[u]; p; p = p->next){
            int v = p->a, dv = d[u] + p->val;
            if ( p->blockInd == blockInd || vis[v] ) continue;
            if ( dv < d[v] || (dv == d[v] && u < f[v]) ){
                d[v] = dv; f[v] = u; e[v] = p;
            }
        }
    }
}

}g;

int main(){ // spoj MKTHPATH, 2012.5.1
    int n, m, k, s, t;
    while (cin >> n >> m >> k >> s >> t && n + m + k + s + t){
        g.init(n);
        while (m--){
            int a, b, c; scanf("%d%d%d", &a, &b, &c);
            g.addEdge(a-1, b-1, c);
        }
        vector<int> ksh;
        vector<Path> candidates;
        g.yenLoopLess(s-1, t-1, k, ksh, candidates);
        if (ksh.size() < k) {
            puts("None");
        } else {
            candidates[ ksh[k-1] ].print();
        }
    }
}

```

## SPFA 判断负权回路

```

//已经通过 PKU2949
const double eps = 1e-5;

const int maxn = 26 * 26;

```



```

int mat[maxn][maxn], adj[maxn][maxn], *adjtop[maxn];

double mid;
#define F(x) (-(x) - mid)

double dist[maxn];
int existnegativecycle;
int instack[maxn], visit[maxn];

void spfa(int x){
    instack[x] = true; visit[x] = true;
    for (int *p = adj[x], v; p != adjtop[x]; p++){
        v = *p;
        if (dist[x] + F(mat[x][v]) < dist[v]){
            dist[v] = dist[x] + F(mat[x][v]);
            if (!instack[v]) spfa(v); else existnegativecycle = 1;
        }
        if (existnegativecycle) return;
    }
    instack[x] = false;
}

int negative_cycle(){
    for (int i = 0; i < maxn; i++)
        dist[i] = 0, instack[i] = 0, visit[i] = 0;
    existnegativecycle = 0;

    for (int i = 0; i < maxn; i++)
        if (!visit[i] && adjtop[i] != adj[i]){
            spfa(i);
            if (existnegativecycle) return 1;
        }
    return 0;
}

double BinarySearch(){
    double left = 0, right = 1000;
    double ans = -1;
    while (right - left > eps){
        mid = (left + right) / 2;
        if (negative_cycle()){
            left = mid + eps;
        }
        else{

```

```

        right = mid - eps;
        ans = mid;
    }
}
if (fabs(ans) < 2 * eps){
    printf("No solution.\n");
}
else {
    printf("%.2f\n", ans);
}
}

int main(){
    int n;
    char s[2000];
    int a, b, slen;
    while (cin >> n && n){
        for (int i = 0; i < maxn; i++){
            for (int j = 0; j < maxn; j++){
                mat[i][j] = 0;
            }
        }
        while (n--){
            scanf("%s", s);
            slen = strlen(s);
            a = (s[0] - 'a') * 26 + s[1] - 'a';
            b = (s[slen-2] - 'a') * 26 + s[slen-1] - 'a';
            mat[a][b] = max(slen, mat[a][b]);
        }
        for (int i = 0; i < maxn; i++){
            int * &p = adjtop[i] = adj[i];
            for (int j = 0; j < maxn; j++){
                if (mat[i][j]){
                    *p++ = j;
                }
            }
        }
        BinarySearch();
    }
}

```

## dijkstra(spfa)+heap

//上次使用 2012.10.3

```

template< class Cmp = less<int> >
struct Heap {
    int hash[maxn], v[maxn], n; Cmp cmp;
    void init(){
        n = 0;
    }
    void up(int p) {
        int a = v[p];
        for (int q = p - 1 >> 1; q >= 0 && cmp(v[q], a); p = q, q = p - 1 >> 1) {
            v[p] = v[q]; hash[ v[p] ] = p;
        }
        v[p] = a; hash[ v[p] ] = p;
    }
    void down(int p) {
        int a = v[p];
        for (int q = p << 1 | 1; q < n; p = q, q = p << 1 | 1) {
            if ( q + 1 < n && cmp(v[q], v[q+1]) ) ++q;
            if ( cmp(v[q], a) ) break;
            v[p] = v[q]; hash[ v[p] ] = p;
        }
        v[p] = a; hash[ v[p] ] = p;
    }
    void push(int a) {
        v[n++] = a; up(n-1);
    }
    int top() const {
        return v[0];
    }
    int size() const {
        return n;
    }
    void pop(){
        swap(v[0], v[--n]); down(0);
    }
};

```

LL dist[maxn];

```

struct Cmp { bool operator()(int i, int j) const { return dist[i] > dist[j]; } };

```

```

struct Graph {
    struct node {
        int b, c; node *next;
    } *ge[maxn], pool[maxe * 2], *pooltop;

```

```

int n;
int order[maxn], ordercnt;
LL dpnext[maxn], dpprev[maxn];

node *addedge(int a, int b, int c) {
    node *p = pooltop++; p->b = b; p->c = c; p->next = ge[a]; ge[a] = p; return p;
}
void init(int n) {
    this->n = n; pooltop = pool;
    for (int i = 0; i < n; ++i) ge[i] = 0;
}
void sssp(int s) {
    static Heap<Cmp> h;
    h.init();
    for (int i = 0; i < n; ++i) {
        dist[i] = i == s ? 0 : inf;
        h.push(i);
    }
    ordercnt = 0;
    while ( h.size() ) {
        int a = h.top(); h.pop(); order[ordercnt++] = a;
        for ( node *p = ge[a]; p; p = p->next ) {
            int b = p->b, c = p->c;
            if ( dist[b] > dist[a] + c ) {
                dist[b] = dist[a] + c; h.up( h.hash[b] );
            }
        }
    }
}
void calc_dp() {
    for (int i = ordercnt - 1; i >= 0; --i) {
        int a = order[i];
        dpnext[a] = 1; dpprev[a] = 0;
        for (node *p = ge[a]; p; p = p->next) {
            int b = p->b, c = p->c;
            if ( dist[a] + c == dist[b] ) dpnext[a] = add_mod(dpnext[a], dpnext[b]);
        }
    }
    dpprev[0] = 1;
    for (int i = 0; i < ordercnt; ++i) {
        int a = order[i];
        for (node *p = ge[a]; p; p = p->next) {
            int b = p->b, c = p->c;
            if ( dist[a] + c == dist[b] ) dpprev[b] = add_mod(dpprev[b], dpprev[a]);
        }
    }
}

```

```

        }
    }
}
LL query(int x) const {
    return mul_mod(dpprev[x], dpnext[x]);
}
} g;

void solve(int n, int m, int q) {
    g.init(n);
    for (int i = 0, a, b, c; i < m; ++i) scanf("%d%d%d", &a, &b, &c), g.addedge(a, b, c);
    g.sssp(0); g.calc_dp();
    for (int x; q--;) scanf("%d", &x), printf("%010lld\n", g.query(x));
}

int main(){ for(int n, m, q; cin >> n >> m >> q; solve(n, m, q)); }
```

## dijkstra $O(n^2)$

参见 55 页，Yen 求 k 短路中的 **void dijkstra()**; 函数

## dijkstra + stl\_set

// $O(E \log V)$ , 参见 63 页 **void dijkstra(int n, int s, LL dist[])** 函数

## s->t 去掉某条边的最短路

```

//时间复杂度  $O(E \log E)$ 
const int maxn = 200000 + 5;
const int maxe = maxn * 2 + 5;
const LL inf = 1000000000000000LL;
#define x first
#define y second

struct Graph {
public:
    void init(int n){
        this->n = n; pooltop = pool;
        for (int i = 0; i <= n; i++) ge[i] = 0;
    }

    void addedge(int a, int b, int d, int flag1 = 0, int flag2 = 0){
```

```

        node *p = _addedge(a, b, d, flag1), q = _addedge(b, a, d, flag2);
        p->anti = q; q->anti = p;
    }
public:
    int n;
    struct node {
        int adj, d;
        node *anti, *next;
        int flag;
    } *ge[maxn];
private:
    node pool[maxe], *pooltop;

    node *_addedge(int a, int b, int d, int flag){
        node *p = pooltop++;
        p->adj = b; p->d = d; p->flag = flag;
        p->next = ge[a]; ge[a] = p;
        return p;
    }
};

```

```

struct BridgeST {
    typedef Graph::node node;
public:
    void init(int n, int s, int t, node *ge[]){
        this->n = n; this->s = s; this->t = t; this->ge = ge;
        for (int i = 0; i <= n; i++)
            low[i] = dfn[i] = inPath[i] = 0;
        idx = cur = 0;
    }
    void bridge(){
        bridge(s, 0);
    }
private:
    bool bridge(int u, node *f){
        bool canReachT = (u == t);
        low[u] = dfn[u] = ++idx;
        for (node *p = ge[u]; p; p = p->next){
            if ( p->flag ){
                if ( !dfn[p->adj] ){
                    int t = bridge(p->adj, p->anti);
                    canReachT |= t;
                    if (t) low[u] = min(low[u], low[p->adj]);
                }
            }
        }
    }
};

```

```

        } else {
            if ( p != f && dfn[p->adj] ){
                low[u] = min(low[u], dfn[p->adj]);
            }
        }
    }
    if ( low[u] == dfn[u] && canReachT){
        color(u, ++cur);
        inPath[u] = cur; List[cur] = u;
        if (f) fa[u] = f->adj; else fa[u] = -1;
    }
    return canReachT;
}

void color(int u, int c){
    belong[u] = c;
    for (node *p = ge[u]; p; p = p->next)
        if ( p->flag && belong[p->adj] == 0 )
            color(p->adj, c);
}

public:
    int cur; // s->t 有多少个连通分量
    int List[maxn], fa[maxn]; // < fa[ List[k] ], List[k] > 是一个桥
    int belong[maxn]; //属于第几个连通分量，与 t 近的标号小
    int inPath[maxn]; //该点是否是桥的"终点"
    LL ans[maxn];

private:
    int n, s, t, idx;
    int low[maxn]; //low[u]:u 或 u 的子树能通过非树枝边追述到的搜索树中最早的节点编号
    int dfn[maxn];
    node **ge;
};

struct ShorestPath { //数组下标从 1 开始
    typedef Graph::node node;

    void init(int n){
        this->n = n; g.init(n);
    }
    void addedge(int a, int b, int d){
        g.addedge(a, b, d);
    }
    void setST(int s, int t){
        this->s = s; this->t = t;
    }
};

```

```

void solve(){
    dijkstra(n, s, distS);
    if ( distS[t] == inf) {
        connected = false;
        return;
    } else {
        connected = true;
    }
    dijkstra(n, t, distT);
    buildDag(n, s, t);
    bridge.init(n, s, t, dag.ge);
    bridge.bridge();
    fill(vis, vis + n + 1, 0);
    for (int i = bridge.cur; i >= 1; i--){
        while ( !Q.empty() && Q.top().y > i ) Q.pop();
        bridge.ans[i] = Q.empty() ? -1 : -Q.top().x; // {fa[List[i]]->List[i]}
        dfs(bridge.List[i]);
    }
}

LL query(int a, int b){
    if ( !connected ) return -1;
    if ( bridge.belong[a] < bridge.belong[b] ) swap(a, b);
    if ( bridge.inPath[b] && bridge.fa[b] == a ){
        return bridge.ans[ bridge.inPath[b] ];
    } else {
        return distS[t];
    }
}

private:
void spfa(int n, int s, LL dist[]){
    //用 spfa 会超时
}

void dijkstra(int n, int s, LL dist[]){
    set< pair<LL, int> > h;
    for ( int i = 0; i <= n; i++ ){
        dist[i] = (i == s ? 0 : inf);
        h.insert( make_pair(dist[i], i) );
    }
    while ( !h.empty() ){
        int u = h.begin()->second; h.erase( h.begin() );
        for ( node *p = g.ge[u]; p; p = p->next ){
            int v = p->adj;
            if ( dist[v] > dist[u] + p->d ){
                h.erase( make_pair(dist[v], v) );
            }
        }
    }
}

```



```

        dist[v] = dist[u] + p->d;
        h.insert( make_pair(dist[v], v) );
    }
}
}
}

void buildDag(int n, int s, int t){
    dag.init(n);
    for(int i = 0; i <= n; i++){
        for ( node *p = g.ge[i]; p; p = p->next )
            if ( distS[i] + p->d == distS[p->adj] )
                dag.addedge(i, p->adj, p->d, 1, 0);
    }
}

void dfs(int u){
    if ( vis[u] ) return;
    vis[u] = 1;
    for ( node *p = g.ge[u]; p; p = p->next ){
        if ( bridge.fa[p->adj] != u && bridge.belong[p->adj] < bridge.belong[u] )
            Q.push( make_pair( -(distS[u]+p->d+distT[p->adj]), bridge.belong[p->adj] ) );
        for ( node *p = dag.ge[u]; p; p = p->next )
            if ( p->flag && !bridge.inPath[p->adj] )
                dfs(p->adj);
    }
}

private:
    int n, s, t;
    bool connected;
    Graph g, dag;
    BridgeST bridge;
    LL distS[maxn], distT[maxn];
    int vis[maxn];
    priority_queue< pair<LL, LL> > Q; //优先队列
};

```

ShorestPath solve;

```

int main(){
    int n, m;
    while ( cin >> n >> m ){
        solve.init(n);
        while (m--){
            int a, b, d;
            scanf("%d%d%d", &a, &b, &d);
            solve.addedge(a, b, d);
        }
    }
}

```

```

    }
    int s, t; scanf("%d%d", &s, &t);
    solve.setST(s, t);
    solve.solve();
    int q; scanf("%d", &q);
    while (q--){
        int a, b; scanf("%d%d", &a, &b);
        LL r = solve.query(a, b);
        if (r == -1) puts("Infinity"); else printf("%lld\n", r);
    }
}
}

```

## 朴素 floyd 算法

$f(i,j,k)$ : 从  $i$  到  $j$ , 中间点标号  $\leq k$ , 别忘了设置  $f(i,i) = 0$

## floyd 求无向图最小环

//注: 有向图的最小环可以直接由朴素 floyd 求得

```

const int inf = INT_MAX / 3, maxn = 300;
int mincircle(int g[maxn][maxn] /* 注意重边取最小值 */, int n, int path[maxn]){
    static int dis[maxn][maxn], int next[maxn][maxn];
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            dis[i][j] = g[i][j];
            next[i][j] = j; /* next[i][j]表示从 i 到 j 的最短路径上 i 的下一个节点 */
        }
    }
    int ret = 0; /* edge cnt */
    int minlen = inf; /* min circle length */
    for (int k = 0; k < n; k++){
        for (int i = 0; i < k; i++){
            if (g[k][i] == inf) continue; //剪枝
            for (int j = 0; j < k; j++){ /* 若要求环的边数>=3(如对于无向图), 则 j = 0..i-1 */
                int thislen = g[k][i] + dis[i][j] + g[j][k];
                if (thislen < minlen){
                    minlen = thislen; ret = 0;
                    path[ret++] = k;
                    for (int t = i; t != j; t = next[t][j]) path[ret++] = t;
                    path[ret++] = j;
                }
            }
        }
    }
}

```

```

        for (int i = 0; i < k; i++){
            if (dis[i][k] == inf) continue; //剪枝
            for (int j = 0; j < k; j++){
                if (dis[i][k] + dis[k][j] < dis[i][j])
                    dis[i][j] = dis[i][k] + dis[k][j], next[i][j] = next[i][k];
            }
        }
    }
    return ret;
}

```

## 树

### 最近公共祖先 LCA - $n \log n$ dp

```

int lca(int a, int b){
    if ( dep[a] > dep[b] ) swap(a, b);
    for (int i = 19; i >= 0; i--)
        if ( fa[i][b] != -1 && dep[ fa[i][b] ] >= dep[a] )
            b = fa[i][b];
    if ( a == b ) return a;
    for (int i = 19; i >= 0; i--)
        if ( fa[i][a] != -1 && fa[i][a] != fa[i][b] )
            a = fa[i][a], b = fa[i][b];
    return fa[0][a];
}

```

## 生成树

最小生成树的环切性质：在图  $G=(V, E)$  中，如果存在一个环，把环中权最大的边  $e$  删除得到图  $G' = (V, E - \{e\})$  (如果有多条最大边，则删除任意一条)，则  $G$  和  $G'$  中的最小生成树的权和相同。

### Prim

//已经通过 **cugb1036**

```

long long prim(int mat[maxn][maxn], int n){ //要求原图一定要是连通图，0 表示没有边
    static int dist[maxn], visit[maxn];
    for (int i = 0; i < n; i++)
        dist[i] = inf, visit[i] = 0;

```

```

int c = 0, miniter, minele; long long tot = 0;
dist[c] = 0; visit[c] = 1;
for (int i = 0; i < n-1; i++){ //n-1 times
    minele = inf;
    for (int j = 0; j < n; j++){
        if (!visit[j]){
            if (mat[c][j])
                dist[j] = min(dist[j], mat[c][j]);
            if (dist[j] < minele)
                minele = dist[j], miniter = j;
        }
    }
    tot += minele; c = miniter; visit[c] = 1;
}
return tot;
}

```

## 次小生成树

```

//复杂度 ElogE, 已经通过 ural1416, xjtu510
const int maxn = 1100, maxm = 1001000;
const LL inf = 0x3f3f3f3f3f3f3fLL;
const int maxdep = 20;

struct edge {
    int a, b, d, flag;
    friend bool operator < (const edge& x, const edge &y){ return x.d < y.d; }
} e[maxn];

struct node {
    int adj, val; node *next;
} *son[maxn], pool[maxn], *pooltop;

void addedge(node *ge[], int a, int b, int v){
    node *p = pooltop++; p->adj = b; p->val = v; p->next = ge[a]; ge[a] = p;
}

int fa[maxn], f[maxn][20], dep[maxn]; LL g[maxn][20];

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

void dfs(int u){
    static int stk[maxn], *stktop; stktop = stk; *stktop++ = u;
    while (stktop != stk){

```

```

u = *--stktop;
for (node *p = son[u]; p; p = p->next){
    if (dep[p->adj] != -1) continue;
    dep[p->adj] = dep[u] + 1;
    f[p->adj][0] = u; g[p->adj][0] = p->val;
    for (int k = 0; k < maxdep - 1; k++){
        int t = f[p->adj][k];
        if (t != -1 && f[t][k] != -1){
            f[p->adj][k+1] = f[t][k]; g[p->adj][k+1] = max( g[p->adj][k], g[t][k] );
        } else {
            f[p->adj][k+1] = -1;
        }
    }
    *stktop++ = p->adj;
}
}
}

```

```

int main(){ //时间复杂度 ElogE, 已通过 poj1679
    int n, m;
    while (scanf("%d%d", &n, &m) == 2){
        for (int i = 0; i < m; i++){
            scanf("%d%d%d", &e[i].a, &e[i].b, &e[i].d); --e[i].a; --e[i].b; e[i].flag = 0;
        }
        sort(e, e + m);
        for (int i = 0; i < n; i++)
            fa[i] = i, son[i] = 0;
        pooltop = pool;
        LL mst = 0;
        int cnt = 0;
        for (int i = 0; i < m; i++){
            int a = find(e[i].a), b = find(e[i].b), d = e[i].d;
            if (a != b){
                fa[a] = b; e[i].flag = 1; mst += e[i].d; cnt++;
                addedge(son, e[i].a, e[i].b, d); addedge(son, e[i].b, e[i].a, d);
            }
        }
        if (cnt != n - 1){
            printf("Cost: %d\n", -1);
            printf("Cost: %d\n", -1);
            continue;
        }
        for (int i = 0; i < n; i++) dep[i] = -1;
        for (int k = 0; k < maxdep; k++) f[0][k] = -1;
    }
}

```

```

dep[0] = 0;
dfs(0);

LL sec_mst = inf;
for (int i = 0; i < m; i++){
    if (e[i].flag == 0){
        int a = e[i].a, b = e[i].b;
        if (a == b) continue;
        if (dep[a] > dep[b]) swap(a, b);
        LL mdist = 0;
        for (int k = maxdep - 1; k >= 0; k--){
            if (dep[b] - dep[a] >= (1 << k)){
                checkmax(mdist, g[b][k]); b = f[b][k];
            }
        }
        if (a != b){
            for (int k = maxdep - 1; k >= 0; k--){
                if ((*f[a][k] != -1 && f[b][k] != -1 && *f[a][k] != f[b][k]) ){
                    checkmax(mdist, g[a][k]); a = f[a][k];
                    checkmax(mdist, g[b][k]); b = f[b][k];
                }
            }
            checkmax(mdist, g[a][0]); checkmax(mdist, g[b][0]);
        }
        sec_mst = min(sec_mst, e[i].d - mdist);
    }
}
sec_mst += mst;
if (sec_mst >= inf) sec_mst = -1;
printf("Cost: %lld\n", mst);
printf("Cost: %lld\n", sec_mst);
}
}

```

## 严格次小生成树

```

const int maxn = 110000, maxm = 300100, maxdep = 20;
const LL inf = 0x3f3f3f3f3f3f3fLL;

struct edge {
    int a, b, d, flag;
    friend bool operator < (const edge& x, const edge &y){ return x.d < y.d; }
} e[maxm];

```

```

struct node {
    int adj, val; node *next;
} *son[maxn], pool[maxn], *pooltop;

void addedge(node *ge[], int a, int b, int v){
    node *p = pooltop++; p->adj = b; p->val = v; p->next = ge[a]; ge[a] = p;
}

int fa[maxn], f[maxn][20], dep[maxn];
LL g[maxn][20], gsub[maxn][20];

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

void dfs(int u){
    static int stk[maxn], *stktop; stktop = stk; *stktop++ = u;
    while (stktop != stk){
        u = *--stktop;
        for (node *p = son[u]; p; p = p->next){
            if (dep[p->adj] != -1) continue;
            dep[p->adj] = dep[u] + 1;
            f[p->adj][0] = u; g[p->adj][0] = p->val; gsub[p->adj][0] = -1;
            for (int k = 0; k < maxdep - 1; k++){
                int t = f[p->adj][k];
                if (t != -1 && f[t][k] != -1){
                    f[p->adj][k+1] = f[t][k];
                    g[p->adj][k+1] = max( g[p->adj][k], g[t][k] );
                    gsub[p->adj][k+1] = max( gsub[p->adj][k], gsub[t][k] );
                    if(g[p->adj][k]!=g[p->adj][k+1])checkmax(gsub[p->adj][k+1],g[p->adj][k]);
                    if(g[t][k] != g[p->adj][k+1]) checkmax(gsub[p->adj][k+1], g[t][k]);
                } else {
                    f[p->adj][k+1] = -1;
                }
            }
            *stktop++ = p->adj;
        }
    }
}

int main(){ //时间复杂度 ElogE, 已经通过 bzoj1977
    int n, m;
    while (scanf("%d%d", &n, &m) == 2){
        for (int i = 0; i < m; i++){
            scanf("%d%d%d", &e[i].a, &e[i].b, &e[i].d); --e[i].a; --e[i].b; e[i].flag = 0;
        }
    }
}

```

```

sort(e, e + m);
for (int i = 0; i < n; i++)
    fa[i] = i, son[i] = 0;
pooltop = pool;
LL mst = 0;
int cnt = 0;
for (int i = 0; i < m; i++){
    int a = find(e[i].a), b = find(e[i].b), d = e[i].d;
    if (a != b){
        fa[a] = b; e[i].flag = 1; mst += e[i].d; cnt++;
        addedge(son, e[i].a, e[i].b, d); addedge(son, e[i].b, e[i].a, d);
    }
}
if (cnt != n - 1){
    TLE; continue;
}
for (int i = 0; i < n; i++) dep[i] = -1;
for (int k = 0; k < maxdep; k++) f[0][k] = -1;
dep[0] = 0;
dfs(0);

LL sec_mst = inf, strict_sec_mst = inf;
for (int i = 0; i < m; i++){
    if (e[i].flag == 0){
        int a = e[i].a, b = e[i].b;
        if (a == b) continue;
        if (dep[a] > dep[b]) swap(a, b);
        LL mdist = -1, mdistsub = -1;
#define check(b,k) {\
    if (g[b][k] > mdist) mdistsub = mdist, mdist = g[b][k];\
    g[b][k]==mdist ? checkmax(mdistsub, gsub[b][k]) : checkmax(mdistsub, g[b][k]);\
}

        for (int k = maxdep - 1; k >= 0; k--){
            if (dep[b] - dep[a] >= (1 << k)){
                check(b,k); b = f[b][k];
            }
        }
        if (a != b){
            for (int k = maxdep - 1; k >= 0; k--){
                if (f[a][k] != f[b][k]){
                    check(a,k); a = f[a][k]; check(b,k); b = f[b][k];
                }
            }
            check(a, 0); check(b, 0);

```



```

    }
    sec_mst = min(sec_mst, e[i].d - mdist);

    if (e[i].d != mdist)
        strict_sec_mst = min(strict_sec_mst, e[i].d - mdist);
    else
        if (mdistsub != -1)
            strict_sec_mst = min(strict_sec_mst, e[i].d - mdistsub);
    }
}
sec_mst += mst;
strict_sec_mst += mst;
if (sec_mst >= inf) sec_mst = -1;
if (strict_sec_mst >= inf) strict_sec_mst = -1;
cout << strict_sec_mst << endl;
}
}

```

## 最大度限制生成树(缺)

## 最优比例生成树(缺)

# 最小树形图

**注：**对于**不固定根**的最小树形图——新加一个点，和每个点连权相同的边，这个权 **huge** 大于原图所有边权和，这样这个图固定根的最小树形图和不固定根的最小树形图就是对应的了。若求得的结果  $\geq 2\text{huge}$ ，则不存在最小树形图；对于可行的根节点新树与虚拟根直接相连的节点的原节点。

## 时间复杂度 **VE**，用带头节点和反向边的双向循环链表实现

```

const int maxn = 1000;
const int maxe = 100000;
const double dbl_inf = 1e10;

struct node {
    int adj; double w;
    node *next, *prev, *anti;
} ge[maxn], ge_rev[maxn], pool[maxe], *pooltop;

```

```

void delpoint(node &ge){
    ge.prev = ge.next = &ge;
}

void init(int n){
    pooltop = pool;
    for (int i = 0; i <= n; i++)
        delpoint(ge[i]), delpoint(ge_rev[i]);
}

node *addedge(node &ge, int adj, double w){
    node *p = pooltop++; p->adj = adj; p->w = w;
    p->prev = &ge; p->next = ge.next; p->prev->next = p->next->prev = p;
    return p;
}

void addedge(int a, int b, double w){
    if (a == b) return; // of importance
    node *p = addedge(ge[a], b, w);
    node *q = addedge(ge_rev[b], a, w);
    p->anti = q; q->anti = p;
}

void deledge(node *p){
    p->prev->next = p->next;
    p->next->prev = p->prev;
}

bool bfs(int n, int r){
    static int q[maxn], *qtop, *qbase, flag[maxn];
    qtop = qbase = q; memset( flag, 0, sizeof(flag[0]) * (n+1) );
    *qtop++ = r; flag[r] = 1;
    while (qbase != qtop){
        int u = *qbase++;
        for (node *p = ge[u].next; p != &ge[u]; p = p->next)
            if (!flag[ p->adj ])
                *qtop++ = p->adj, flag[ p->adj ] = 1;
    }
    return qtop - q == n;
}

double zhuliu(int n, int r){
    static int pre[maxn], flag[maxn];
    static double mindist[maxn];

```

```

double res = 0;
for (;){
    for (int i = 0; i <= n; i++){
        pre[i] = i; mindist[i] = dbl_inf;
        if (i == r) continue;
        for (node *p = ge_rev[i].next; p != &ge_rev[i]; p = p->next)
            if ( p->w < mindist[i] )
                mindist[i] = p->w, pre[i] = p->adj;
    }
    memset(flag, -1, sizeof(flag[0]) * (n+1)); flag[r] = r;
    int i, j;
    for (i = 0; i <= n; i++){
        if (i == pre[i]) continue;
        if ( flag[i] != -1 ) continue;
        for (j = i; flag[j] == -1; j = pre[j])
            flag[j] = i;
        if (flag[j] != i) continue;
        i = j;
        res += mindist[i];
        for (node *p = ge_rev[i].next; p != &ge_rev[i]; p = p->next)
            p->w -= mindist[i], p->anti->w -= mindist[i];
        for (j = pre[i]; j != i; j = pre[j]){
            res += mindist[j];
            for (node *p = ge[j].next; p != &ge[j]; p = p->next)
                addedge(i, p->adj, p->w), deledge(p->anti);
            for (node *p = ge_rev[j].next; p != &ge_rev[j]; p = p->next)
                addedge(p->adj, i, p->w - mindist[j]), deledge(p->anti);
            delpoint(ge[j]); delpoint(ge_rev[j]);
        }
        break;
    }
    if (i > n){
        for (i = 0; i <= n; i++)
            if (pre[i] != i)
                res += mindist[i];
        return res;
    }
}
}

int main(){ //poj3164
    int n, m;
    static double x[maxn], y[maxn];
    while ( cin >> n >> m ){

```

```

    for (int i = 1; i <= n; i++)
        scanf("%lf%lf", &x[i], &y[i] );
    init(n);
    while (m--){
        int a, b; scanf("%d%d", &a, &b);
        addedge(a, b, sqrt( sqr(x[a] - x[b]) + sqr(y[a] - y[b]) ) );
    }
    if (bfs(n, 1)){
        printf("%.2f\n", zhuliu(n, 1) );
    } else {
        cout << "poor snoopy" << endl;
    }
}
}

```

## 时间复杂度 VE，用边表实现

```

//已经通过 hdu2121
const int maxn = 200, maxe = 20000, iinf = INT_MAX;

struct node {
    int a, b; int w;
    node *next;
} e[maxe], *ge[maxn];

bool judge(int n, int m, int r){
    static int q[maxn], *qtop, *qbase, flag[maxn];
    memset(flag, 0, sizeof (flag[0]) * n); qtop = qbase = q;
    *qtop++ = r; flag[r] = 1;
    while (qbase != qtop)
        for (node *p = ge[*qbase++]; p; p = p->next)
            if (!flag[ p->b ])
                *qtop++ = p->b, flag[ p->b ] = 1;
    return qtop - q == n;
}

LL zhuliu(int n, int m, int r){ //下标从 0 开始
    static int pre[maxn], mindist[maxn], flag[maxn];
    LL res = 0;
    for (;){
        for (int i = 0; i < n; i++)
            pre[i] = i, mindist[i] = iinf;
        for (int i = 0; i < m; i++){
            if (e[i].b == r || e[i].a == e[i].b) continue;

```

```

        if ( e[i].w < mindist[ e[i].b ] ){
            mindist[ e[i].b ] = e[i].w, pre[ e[i].b ] = e[i].a;
            //if ( e[i].a0 == r0 ) mark = e[i].b0; //要得到无根树最小的可行根时填此句
        }
    }
    memset( flag, -1, sizeof(flag[0]) * n ); flag[r] = r;
    int nn = 0;
    for (int i = 0, j; i < n; i++){
        if ( pre[i] == i || flag[i] != -1 ) continue;
        for ( j = i; flag[j] == -1; j = pre[j] )
            flag[j] = i;
        if ( flag[j] != i ) continue;
        i = j; res += mindist[i];
        memset( flag, -1, sizeof(flag[0]) * n );
        flag[i] = nn;
        for ( j = pre[i]; j != i; j = pre[j])
            flag[j] = nn, res += mindist[j];
        nn++;
        for ( j = 0; j < n; j++)
            if (flag[j] == -1)
                flag[j] = nn++;
        for ( j = 0; j < m; j++){
            if ( flag[ e[j].b ] == 0)
                e[j].w -= mindist[ e[j].b ];
            e[j].a = flag[ e[j].a ]; e[j].b = flag[ e[j].b ];
        }
        r = flag[r]; break;
    }
    if (nn == 0){
        for (int i = 0; i < n; i++)
            if (i != r)
                res += mindist[i];
        return res;
    }
    n = nn;
}
}

```

```

int main(){ //tju2248
    int n, m;
    while ( cin >> n >> m && n + m ){
        memset(ge, 0, sizeof(ge[0]) * n);
        for (int i = 0; i < m; i++){
            scanf( "%d%d%d", &e[i].a, &e[i].b, &e[i].w ); --e[i].a; --e[i].b;

```

```

        e[i].next = ge[ e[i].a ]; ge[ e[i].a ] = &e[i];
    }
    if (judge(n, m, 0)){
        cout << zhuliu(n, m, 0)<< endl;
    } else {
        puts("impossible");
    }
}
}
}

```

## 弦图

1. 子图(subgraph): 设  $G = (V, E)$ , 则图  $G' = (V', E')$ ,  $V' \subseteq V, E' \subseteq E$  为图  $G$  的子图。
2. 诱导子图(induced subgraph): 设  $G = (V, E)$ , 则图  $G' = (V', E')$ ,  $V' \subseteq V, E' = \{(u, v) \mid u, v \in V', (u, v) \in E\}$  为图  $G$  的诱导子图。
3. 团(clique): 图  $G$  的一个子图  $G' = (V', E')$ ,  $G'$  为关于  $V'$  的完全图。  
团数  $\omega(G)$ : 最大团的  $|V'|$
4. 最小染色(minimum coloring): 用最少的颜色给点染色, 使相邻点颜色不同。  
色数  $\chi(G)$
5.  $k$  染色问题: 判断一个图能否用  $k$  种颜色进行染色  
(2 染色问题 - 二分图判断; 3 染色问题 - NP)
6. 最大独立集(maximum independent set): 最大的一个点的子集是任何两点不相邻。  
独立数  $\alpha(G)$
7. 最小团覆盖(minimum clique cover): 用最小个数的团覆盖所有的点。  
最小团覆盖数  $\kappa(G)$
8. 性质:  $\omega(G) \leq \chi(G), \alpha(G) \leq \kappa(G)$
9. 弦(chord): 环中不相邻的两个点的边。
10. 弦图(chord graph): 图中任何长度  $\geq 4$  的环都存在弦。
11. 弦图性质: 弦图的任一个诱导子图一定是弦图; 弦图的任一个诱导子图不同构于  $C_n$  ( $n \geq 3$ ), 其中  $C_n$  为  $n$  个点  $n$  条边构成的环
12. 单纯点(simplicial vertex): 设  $N(v)$  表示与  $v$  相邻的点集, 一个点被称为单纯点, 如果  $\{v\} + N(v)$  的诱导子图为一个团。
13. 性质: 任何一个弦图都至少有一个单纯点, 不是完全图的弦图至少有两个不相邻的单纯点。
14. 完美消除序列 PEO(perfect elimination ordering): 一个点的序列(每个点出现且仅出现一次),  $v_1, \dots, v_n$ , 满足  $v_i$  在  $\{v_i, v_{i+1}, \dots, v_n\}$  的诱导子图中为一个单纯点。
15. 定理: 一个无向图为弦图当且仅当它有一个完美消除序列。
16. 弦图的极大团: 设第  $i$  个点在弦图的完美消除序列的第  $\text{rank}(i)$  个, 令  $N(v) = \{v \mid v \text{ 与 } u \text{ 相邻且 } \text{rank}(v) > \text{rank}(u)\}$  (注意  $N(v)$  的定义和前面不同), 则弦图的极大团一定是  $v \cup N(v)$  的形式。显然弦图最多有  $n$  个极大团。
17. 判断  $v \cup N(v)$  是否是极大团: 设  $\text{next}(u) = \arg\min\{\text{rank}(v) \mid v \in N(v)\}$ , 则若  $|N(v)| + 1 \leq |N(u)|$ , 则  $v \cup N(v)$  不是极大团。
18. 弦图的最小染色数: 完美消除序列从后向前一次给每个点染色。  $\text{color}[u] = \text{mex}\{\text{color}[v] \mid$

$v \in N(u) \}$  (此时  $\omega(G) = \chi(G)$ )

19. 弦图的**最大独立集**：完美消除序列从前向后能选就选。
20. 弦图的最小团覆盖：设最大独立集为 $\{P_1, \dots, P_t\}$ ，则最小团覆盖为 $\{P_1 \cup N(P_1), \dots, P_t \cup N(P_t)\}$
21. 完美图(perfect graph)：一个图是完美图若它的每个诱导子图都满足  $\omega(G) = \chi(G)$ 。
22. 伴完美图(co-perfect graph)：一个图是伴完美图若他的每个诱导子图都满足  $\alpha(G) = \kappa(G)$ 。
23. 区间图(interval graph)：将图的每一个顶点定义成一个区间，顶点间有边 iff.两区间相交。
24. 区间图  $\subset$  弦图  $\subset$  完美图(伴完美图)
25. 给定  $n$  个区间，所对应的**区间图**为  $G$ ，则  $G$  的一个**完美消除序列**：将所有的区间按照右端点从小到大排序。
26. 区间图的最小(点)覆盖问题：依据完美消除序列(按照右端点排序)，每次选择最靠右的点。
27. 区间图的最小(区间)覆盖问题：按照左端点排序，从左向右扫描，每次把右端点放到一个优先队列中，每次取出最大的右端点。
28. 例题：有  $n$  个积木，高度均为 1，第  $i$  个积木的高度范围为 $[L_i, R_i]$ ，选择一个积木的下落顺序是的积木的总高度尽可能小。(区间图的最小染色)
29. 极大团树(clique tree)：一个无向图  $G$  的极大团树定义为： $T$  的顶点为图  $G$  的左右极大团；包含每个点的所有极大团为  $T$  的一个连通子图。
30. 一个无向图是弦图 iff 它存在一个 clique tree
31. 构建弦图的一个 clique tree
  - a) 找出弦图的所有极大团。
  - b) 构图  $G'$ ，极大团为点，两个点之间的边权为两个极大团交集的点的个数。
  - c) 求  $G'$  的一个最大生成树。

## 弦图的判断 (ElogE)

```
struct edge {
    int a, b;
    friend bool operator < (const edge& x, const edge& y){ return x.a != y.a ? x.a < y.a : x.b < y.b;}
    friend bool operator == (const edge& x, const edge& y){ return x.a == y.a && x.b == y.b;}
} e[maxm], *start[maxn];
```

```
bool ischord(int n, int m){ //要求点的编号 1..n
    static int lbl[maxn], sigma[maxn], rank[maxn], L[maxn*2], R[maxn*2], U[maxn], D[maxn];
    for (int i = 1; i <= n; i++)
        sigma[i] = 0, rank[i] = 0, lbl[i] = 0;
    U[0] = D[0] = n; U[n] = D[n] = 0;
    for (int i = n+1; i <= n+n; i++)
        L[i] = i-1, R[i] = i+1;
    L[1+n] = R[n+n] = 0; R[0] = 1+n; L[0] = n+n;
    for (int i = 1; i < n; i++)
        L[i] = R[i] = i;
    #define removeLR(u) (R[L[u]] = R[u], L[R[u]] = L[u])
    #define removeUD(u) (U[D[u]] = U[u], D[U[u]] = D[u])
```

```

#define insertLR(u,a,b) do { int aa(a),bb(b); L[u] = aa, R[u] = bb, R[aa] = L[bb] = u; } while (0)
#define insertUD(u,a,b) do{ int aa(a),bb(b); U[u] = aa, D[u] = bb, D[aa] = U[bb] = u; } while(0)
#define rep_adj(p,u) for(edge *p = start[u]; p != e + m && p->a == (u); p++)
for (int i = n; i >= 1; i--){
    int uu = R[U[n]], u = uu - n;
    sigma[i] = u; rank[u] = i;
    removeLR(uu); if (l[u] == R[l[u]]) removeUD(l[u]);
    rep_adj(p, u) {
        int v = p->b, vv = v + n;
        if (rank[v] != 0) continue;
        removeLR(vv); insertLR(vv, l[v]+1, R[l[v]+1]);
        if (D[l[v]] != l[v] + 1) insertUD(l[v]+1, l[v], D[l[v]]);
        if (R[l[v]] == l[v]) removeUD(l[v]);
        ++l[v];
    }
}
rank[0] = n + 1; e[m].a = e[m].b = 0;
for (int i = 1; i <= n; i++){
    int u = sigma[i], v = 0;
    rep_adj(p, u)
        if ( i < rank[p->b] && rank[p->b] < rank[v] )
            v = p->b;
    if (v == 0) continue;
    rep_adj(p, u)
        if (rank[v] < rank[p->b])
            if ( e[m+1].a=v,e[m+1].b=p->b, !(*lower_bound(e, e + m, e[m+1]) == e[m+1]) )
                return false;
}
return true;
}

int main(){ //zju1015
    int n, m;
    while ( scanf("%d%d", &n, &m) == 2 && n + m ){
        int ecnt = 0;
        for (int i = 0; i < m; i++){
            int a, b; scanf("%d%d", &a, &b); if (a == b) continue;
            e[ecnt].a = a; e[ecnt].b = b; ecnt++; e[ecnt].a = b; e[ecnt].b = a; ecnt++;
        }
        sort(e, e + ecnt); ecnt = unique(e, e + ecnt) - e;
        for (int i = 1; i <= n; i++) start[i] = e + ecnt;
        for (int i = 0, j; i < ecnt; i = j){
            j = i + 1; while (j < ecnt && e[i].a == e[j].a) j++; start[ e[i].a ] = &e[i];
        }
    }
}

```



```

        puts(ischord(n, ecnt) ? "Perfect\n" : "Imperfect\n");
    }
}

```

## 传递闭包

求一个图的传递闭包有多少条边  $O(VE)$

```

#define clr(a, v, n) memset(a, v, sizeof(a[0]) * (n))

struct Graph {
    const static int maxn = 1000 + 5;
    const static int maxe = 10000 + 5;

    struct node {
        int b; node *next;
    } *ge[maxn], pool[maxe], *pooltop;
    int n;

    void init(int n) {
        clr(ge, 0, n + 1); pooltop = pool; this->n = n;
        clr(flag, -1, n + 1);
    }

    void insert(int a, int b) {
        node *p = pooltop++; p->b = b; p->next = ge[a]; ge[a] = p;
    }

    int flag[maxn], ind, cnt;
    void dfs(int a) {
        flag[a] = ind; ++cnt;
        for (node *p = ge[a]; p; p = p->next) {
            if ( flag[p->b] != ind ) dfs(p->b);
        }
    }

    int TransitiveClosureEdgeCount(){ // include edge (a->a)
        cnt = 0;
        for (int i = 1; i <= n; ++i) {
            ind = i;
            dfs(i);
        }
        return cnt;
    }
}

```

```

} graph;

void solve(int n, int m) {
    graph.init(n);
    while (m--) {
        int a, b; scanf("%d%d", &a, &b); graph.insert(a, b);
    }
    int ans = n * (n-1) / 2 - ( graph.TransitiveClosureEdgeCount() - n );
    cout << ans << endl;
}

```

## dfs 递归转非递归

```

void dfs(int u){
    //statement1
    for (node *p = ge[r]; p; p = p->next){
        //statement2
        dfs(p->adj);
    }
    //statement4
}

void dfs(int u){
    static pair<int, node *> stk[maxn], *stktop;
    stktop = stk; node *p;
    *stktop++ = make_pair(u, ge[u]);
    while (stktop != stk){
        --stktop; u = stktop->first; p = stktop->second;
        if (p == ge[u]){
            //statement1
        }
        for (; p; p = p->next){
            //statement2

            //dfs(p->adj)
            *stktop++ = make_pair(u, p->next);
            *stktop++ = make_pair(p->adj, ge[p->adj]);

            break;
        }
        if (p == 0){
            //statement4
        }
    }
}

```

}