**Heap**

int data[MAXN];

int size;

inline int lson(int a) {

return a<<1;

}

inline int rson(int a) {

return lson(a)+1;

}

inline int fa(int a) {

return a>>1;

}

int heapfy(int pos) {

while (true) {

int l = lson(pos);

int r = rson(pos);

int m = pos;

if (l <= size && data[l] < data[m]) {

m = l;

}

if (r <= size && data[r] < data[m]) {

m = r;

}

if (m == pos)

break;

swap(data[pos],data[m]);

pos = m;

}

}

void update(int pos, int val) {

data[pos] = val;

while (pos != 1 && data[fa(pos)] > data[pos]) {

swap(data[fa(pos)],data[pos]);

pos = fa(pos);

}

}

void push(int a) {

update(++size,a);

}

int extract() {

int ret = data[1];

swap(data[1],data[size--]);

heapfy(1);

return ret;

}

**Graham**

const int MAXN = 10001;

struct point

{

double x, y;

}pt[MAXN];

int ord[MAXN];

int n;

double det(double x1, double y1, double x2, double y2)

{

return x1\*y2-x2\*y1;

}

double cross(point a, point b, point c)

{

return det(b.x-a.x,b.y-a.y,c.x-a.x,c.y-a.y);

}

int tid;

int cmp(const void \*pa, const void \*pb)

{

int a = \*((int \*)pa);

int b = \*((int \*)pb);

if (b == tid || cross(p0,pt[a],pt[b])<0)

return 1;

return -1;

}

double sqr(double a)

{

return a\*a;

}

double dist(point p1, point p2)

{

return sqrt(sqr(p1.x-p2.x)+sqr(p1.y-p2.y));

}

void work()

{

//Graham-Scan-Algorithm

int stack[MAXN];

int size=0;

stack[++size] = ord[1];

stack[++size] = ord[2];

for (int i = 3; i <= n;)

{

if (cross(pt[stack[size-1]],pt[stack[size]],pt[ord[i]])>0)

stack[++size] = ord[i++];

else

--size;

}

double ans = 0;

for (int i = 2; i <= size; i++)

ans += dist(pt[stack[i]],pt[stack[i-1]]);

ans += dist(pt[stack[1]],pt[stack[size]]);

}

**Maxflow**

const int MAXN = 55;

const int INF = 1<<28;

const int MAXV = MAXN\*2;

const int MAXE = MAXV\*MAXV;

int vs, vt;

struct Edge {

int vtx, cap;

Edge \*nxt, \*rev;

Edge() {}

Edge(int iv, int ic, Edge \*in, Edge \*ir):

vtx(iv), cap(ic), nxt(in), rev(ir) { }

} epool[MAXE], \*adj[MAXV], \*eptr = epool, \*stae[MAXV], \*p[MAXV];

// epool静态存储各边数据的内存池 adj各点的邻接表

// eptr用来标记内存池写入位置的变量

// stae非递归增广过程中需要用到的存储增广路经的栈

// p非递归增广过程中用来记录每个点用了哪些边

inline void add(int u, int v, int cap) {

adj[u] = new (eptr++) Edge(v,cap,adj[u],eptr);

adj[v] = new (eptr++) Edge(u,0,adj[v],eptr-2);

}

int lv[MAXV], stap[MAXV];

bool label() {

int head, tail;

stap[head=tail=0] = vs;

memset(lv,-1,sizeof(lv));

lv[vs] = 0;

while (head<=tail) {

int u = stap[head++];

for (Edge \*t = adj[u]; t; t = t->nxt) {

int v = t->vtx;

if (t->cap && -1 == lv[v]) {

lv[v] = lv[u]+1;

if (v == vt)

return true;

stap[++tail] = v;

}

}

}

return false;

}

int aug() {

int ret = 0;

for (int i = vs; i <= vt; ++i)

p[i] = adj[i];

int stop = 1;

stap[stop] = vs;

int u, v;

while (stop) {

u = stap[stop];

if (u != vt) {

for (; p[u]; p[u] = p[u]->nxt)

if (p[u]->cap && lv[u]+1 == lv[v=p[u]->vtx])

break;

if (p[u]) {

stap[++stop] = v;

stae[stop] = p[u];

} else

--stop, lv[u] = -1;

} else {

int delta = INF;

for (int i = stop; i >= 2; --i)

if (stae[i]->cap < delta)

delta = stae[i]->cap;

ret += delta;

for (int i = stop; i >= 2; --i) {

stae[i]->cap -= delta;

stae[i]->rev->cap += delta;

if (!stae[i]->cap)

stop = i-1;

}

}

}

return ret;

}

int dinic() {

int ret = 0;

while (label())

ret += aug();

return ret;

}

**MAXFLOW-SAP**

onst int MAXN = 105;

const int MAXM = 5005;

const int INF = 1<<28;

int n, m;

const int MAXV = MAXN\*2;

const int MAXE = MAXM\*2+MAXV\*4;

struct Edge {

int vtx, cap;

Edge \*nxt, \*rev;

Edge() { }

Edge(int iv, int ic, Edge \*in, Edge \*ir):

vtx(iv), cap(ic), nxt(in), rev(ir) { }

} epool[MAXE], \*adj[MAXV];

Edge \*eptr = epool;

int vs, vt, vtot;

void add(int u, int v, int cap) {

adj[u] = new (eptr++) Edge(v,cap,adj[u],eptr);

adj[v] = new (eptr++) Edge(u,0,adj[v],eptr-2);

}

int ht[MAXV];

int hct[MAXV];

int aug(int u, int augc) {

if (vt == u)

return augc;

int rem = augc;

int minht = vtot-1;

for (Edge \*t = adj[u]; t; t = t->nxt) {

if (t->cap) {

int v = t->vtx;

if (ht[v]+1 == ht[u]) {

int d = aug(v,min(rem,t->cap));

rem -= d;

t->cap -= d;

t->rev->cap += d;

if (!rem)

return augc;

if (ht[vs] >= vtot)

return augc-rem;

}

minht = min(minht,ht[v]);

}

}

if (rem == augc) {

--hct[ht[u]];

if (!hct[ht[u]])

ht[vs] = vtot;

ht[u] = minht+1;

++hct[ht[u]];

}

return augc-rem;

}

int sap() {

int ret = 0;

hct[0] = vtot;

while (ht[vs] < vtot)

ret += aug(vs,INF);

return ret;

}

**MCMF**

const int MAXV = MAXN\*2;

const int MAXE = MAXV\*MAXV;

const int INF = 1<<28;

int vs, vt;

struct Edge {

int vtx, cap, cost;

Edge \*nxt, \*rev;

Edge() { }

Edge(int iv, int icap, int icost, Edge \*in, Edge \*ir):

vtx(iv), cap(icap), cost(icost), nxt(in), rev(ir) { }

} epool[MAXE], \*adj[MAXV], \*eptr = epool;

inline void add(int u, int v, int cap, int cost) {

adj[u] = new (eptr++) Edge(v,cap,cost,adj[u],eptr);

adj[v] = new (eptr++) Edge(u,0,-cost,adj[v],eptr-2);

}

int pre[MAXV];

Edge \*path[MAXV];

int flow;

bool inq[MAXV];

int dis[MAXV];

int spfa() {

for (int i = vs; i <= vt; ++i)

dis[i] = INF;

dis[vs] = 0;

int sum = 0;

deque<int> que;

que.push\_front(vs);

int u, v;

while (!que.empty()) {

while (dis[que.front()] > (sum/que.size())) {

que.push\_back(que.front());

que.pop\_front();

}

u = que.front();

que.pop\_front();

inq[u] = false;

sum -= dis[u];

for (Edge \*t = adj[u]; t; t = t->nxt)

if (t->cap && dis[u]+t->cost < dis[v=t->vtx]) {

int tmp = dis[v];

dis[v] = dis[u]+t->cost;

pre[v] = u;

path[v] = t;

sum += dis[v];

if (!inq[v]) {

if (que.empty() || dis[que.front()] > dis[v])

que.push\_front(v);

else

que.push\_back(v);

inq[v] = true;

} else {

sum -= tmp;

}

}

}

int delta = INF;

if (INF != dis[vt]) {

for (int v = vt; v != vs; v = pre[v])

if (path[v]->cap < delta)

delta = path[v]->cap;

flow += delta;

for (int v = vt; v != vs; v = pre[v]) {

path[v]->cap -= delta;

path[v]->rev->cap += delta;

}

}

if (INF == dis[vt])

return INF;

return dis[vt]\*delta;

}

int mcmf(int needed) {

flow = 0;

int ret = 0;

while (true) {

int d = spfa();

if (d != INF)

ret += d;

else

break;

}

if (flow < needed)

return INF;

return ret;

}

void init() {

memset(adj,0,sizeof(adj));

eptr = epool;

}

**2-SAT**

// 此题是一个标准的2-SAT 问题

// 解法详见 wc2003 伍昱论文

#include <iostream>

#include <cstdlib>

#include <cstdio>

#include <cstring>

using namespace std;

const int MAXN = 8001;

const int MAXM = 20001;

const int MAXV = MAXN\*2;

const int MAXE = MAXM\*6;

inline int cal(int a) {

if (a&1)

return a+1;

return a-1;

}

int e[MAXE], nxt[MAXE];

int adj[MAXV], tadj[MAXV];

int cnt;

inline void add(int u, int v) {

e[++cnt] = v;

nxt[cnt] = adj[u];

adj[u] = cnt;

e[++cnt] = u;

nxt[cnt] = tadj[v];

tadj[v] = cnt;

}

bool used[MAXV];

int color[MAXV];

int opp[MAXV];

bool del[MAXV];

int table[MAXV];

int n, m;

void load() {

n \*= 2;

cnt = table[0] = color[0] = 0;

memset(adj,0,sizeof(adj));

memset(tadj,0,sizeof(tadj));

memset(del,0,sizeof(del));

int a, b;

while (m--) {

scanf("%d%d", &a, &b);

add(a,cal(b));

add(b,cal(a));

}

}

void dfs(int u) {

used[u] = true;

for (int t = adj[u]; t; t = nxt[t]) {

int v = e[t];

if (!used[v])

dfs(v);

}

table[++table[0]] = u;

}

void kosaraju(int u) {

used[u] = false;

color[u] = color[0];

for (int t = tadj[u]; t; t = nxt[t]) {

int v = e[t];

if (used[v])

kosaraju(v);

}

}

void solve() {

// use the kosaraju algorithm to compute the strongly connected components and

// topsort the graph

for (int i = 1; i <= n; ++i)

if (!used[i])

dfs(i);

for (int i = n; i; --i)

if (used[table[i]]) {

++color[0];

kosaraju(table[i]);

}

// judge the feasibility

for (int i = 1; i < n; i += 2)

if (color[i] == color[i+1]) {

printf("NIE\n");

return;

}

// compute each subgraph's symmetry subgraph

for (int i = 1; i <= n; ++i)

opp[color[i]] = color[cal(i)];

// by the reverse order computed by topsort to construct a feasible solution

for (int i = color[0]; i; --i)

if (!del[i])

del[opp[i]] = true;

// simply output

for (int i = 1; i <= n; ++i)

if (!del[color[i]])

printf("%d\n", i);

}

int main() {

while (EOF != scanf("%d%d", &n, &m)) {

load();

solve();

}

return 0;

}

**TARRAY**

const int MAXN = 100;

int n;

int c[MAXN];

inline int lowbit(int a) {

return a&-a;

}

void ins(int a) {//添加

while (a <= n) {

++c[a];

a += lowbit(a);

}

}

void del(int a) {//删除

while (a <= n) {

--c[a];

a += lowbit(a);

}

}

int sel(int k) {//选第k小的数，选k大记录下总数然后转换成求k小即可

int ret = 0;

for (int p = 1<<(int(log2(n))); p; p >>= 1)

if (ret+p < n && c[ret+p] < k)

k -= c[ret+=p];

return ret+1;

}

int get(int a) {//求小于等于a的数的个数

int ret = 0;

while (a) {

ret += c[a];

a -= lowbit(a);

}

return ret;

}

int rank(int a) {//求比数a小的数的个数

return get(a-1);

}

int num(int l, int r) {//求区间【L,R】中元素的个数

return get(r)-get(l-1);

}

**TREAP**

// treap

class Treap {

public:

static const int Inf = INT\_MAX;

private:

struct Tnode {

// typedef Tnode\* Tptr;

int val, priority, size;

Tnode \*son[2];

Tnode(int ival, Tnode \*pnil):

val(ival), priority(rand()),size(1) { son[0] = son[1] = pnil; }

void resize() {

size = son[0]->size+son[1]->size+1;

}

} \*root, \*nil;

typedef Tnode\* Tptr;

void rotate(Tptr &ptr, bool side) {

Tptr tmp = ptr->son[side];

ptr->son[side] = tmp->son[!side];

tmp->son[!side] = ptr;

ptr->resize();

tmp->resize();

ptr = tmp;

}

void insert(Tptr &ptr, int val) {

if (nil == ptr) {

ptr = new Tnode(val,nil);

return ;

}

if (ptr->val == val)

return ;

bool side = val > ptr->val;

insert(ptr->son[side],val);

if (ptr->son[side]->priority < ptr->priority)

rotate(ptr,side);

else

ptr->resize();

}

void del\_node(Tptr &ptr, int val) {

if (nil == ptr)

return ;

if (ptr->val == val) {

bool side = ptr->son[1]->priority < ptr->son[0]->priority;

if (nil == ptr->son[side]) {

delete ptr;

ptr = nil;

return ;

}

rotate(ptr,side);

del\_node(ptr->son[!side],val);

} else {

del\_node(ptr->son[val>ptr->val],val);

}

ptr->resize();

}

int select(Tptr ptr, int kth) {

int lsize = ptr->son[0]->size;

if (lsize == kth)

return ptr->val;

if (kth < lsize)

return select(ptr->son[0],kth);

return select(ptr->son[1],kth-lsize-1);

}

int rank(Tptr ptr, int val) {

if (nil == ptr)

return 0;

int lsize = ptr->son[0]->size;

if (val <= ptr->val)

return rank(ptr->son[0],val);

return lsize+1+rank(ptr->son[1],val);

}

void dfs(Tptr ptr) {

if (ptr->son[0] != nil)

dfs(ptr->son[0]);

cout << ptr->val << ' ' << ptr->priority << ' ' << ptr->size << endl;

if (ptr->son[1] != nil)

dfs(ptr->son[1]);

}

public:

Treap() {

srand(time(0));

nil = new Tnode(0,0);

nil->size = 0;

nil->priority = Inf;

root = nil;

}

void ins(int a) {

insert(root,a);

}

void del(int a) {

del\_node(root,a);

}

int sel(int kth) {

if (kth > root->size)

return Inf;

return select(root,kth-1);

}

int ran(int val) {

return rank(root,val);

}

} tree;