# Union Find Disjoint Set

vector<int> par,rnk,cnt; int numOfsets;

void SetInit(int n){

par.assign(n,-1); rnk.assign(n,0); cnt.assign(n,1); //par==parent

numOfsets=n; // if wanna count number of disjoint sets

}

int Find(int a){

int i=a,j=a,tmp;

while(par[i]!=-1){ i=par[i]; }

while(par[j]!=-1){ tmp=par[j]; par[j]=i; j=tmp; } //path compression

return i;

}

int Uni(int a, int b){

int A=Find(a),B=Find(b);

if(A!=B){

if(rnk[A]<rnk[B]) swap(A,B); // union using rank

if(rnk[A]==rnk[B]) rnk[A]++;

par[B]=A;

cnt[A]+=cnt[B]; // if we wanna count each set size

numOfsets--; // if wanna count number of disjoint sets

}

return cnt[A]; // if we wanna count each set size

}

# Graphic Sequence

// given a sequence of integers see if it’s a sequence of degrees of graph or not.

int a[10010]; long long sum,Min;;

int main(){

int n;

while(cin >> n && n){

for(int i=0 ; i<n ; i++) scanf("%d",&a[i]);

sort(a,a+n, ::greater<int>() );

bool possible=true; sum=0;

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

sum+=a[i]; Min=0;

for(int j=i+1 ; j<n; j++) Min+=min(a[j],i+1);

if(sum>i\*(i+1)+Min){

possible=false;

break;

}

}

if(!possible || sum%2) cout << "Not possible" << endl;

else cout << "Possible" << endl;

}

return 0;

}

# Segment Tree

vector<int> st, a;//st == segment tree, a == array

int Log2(int n){ int cnt=0; while(n/=2) cnt++; return cnt; }

void initSegT(int n){

//if n is power of 2 then arrlen=n else arrlen=first power of 2 that greater than n

int arrlen=(((n!=0) && !(n&(n-1)))? n : 1<<(Log2(n)+1)); //calc length of array

int stlen=4\*arrlen; // if you don’t wanna use vector, these must be global

a.clear();

a.assign(arrlen,0);// using vector can Cause TLE

st.assign(stlen,0);// length of segment tree

}

void updateSegT(int node, int idx){

if(node==0) return;

st[node]=(a[idx]<a[st[node]] ? idx : st[node]);

updateSegT(node/2,idx);

//if(node==0) return; st[node]=st[node]-diff; updateSegT(node/2,diff);

}

void updateArr(int idx, int val){

a[idx]=val;

updateSegT(st.size()/2+idx,idx);

//int diff=a[idx]-val; a[idx]=val; updateSegT(st.size()/2+idx,diff);

}

int Query(int node, int L, int R, int i, int j){

if(i>R || L>j) return -1; //return 0; // for range sum query

if(i<=L && R<=j) return st[node];

int p1 = Query(2\*node,L,(L+R)/2,i,j);

int p2 = Query(2\*node+1,(L+R)/2+1,R,i,j);

if(p1==-1) return p2; if(p2==-1) return p1;

return (a[p1] <= a[p2]) ? p1 : p2;

//return Query(2\*node,L,(L+R)/2,i,j)+Query(2\*node+1,(L+R)/2+1,R,i,j);

}

void printtree(){int l=1;

for(int i=1 ; i<st.size() ; i++){

if((i)== 1 << l) l++,cout << endl;

cout << st[i] << " ";

}cout << endl;}

int main(){

int n; cin >> n; initSegT(n);

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

int tmp; cin >> tmp;

updateArr(i,tmp);

}

int i,j;

while( cin >> i >> j){

cout << Query(1,0,a.size()-1,i,j) << endl;

}

# Fenwick Tree (counting inversions)

vector<int> tree,a,b; int n;

int64 read(int idx){

int64 sum=0;

while(idx>0){

sum+=tree[idx]; idx-=(idx & -idx);

}

return sum;

}

void update(int idx, int val){

int64 sum=0;

while(idx<n){

tree[idx]+=val; idx+=(idx & -idx);

}

}

// get largest value with cumulative sum less than or equal to x;

// for smallest, pass x-1 and add 1 to result

int getind(int x) {// \*\*\*Change Needed\*\*\*

int idx = 0, mask = TREE\_SIZE; //(must be a power of 2)

while(mask && idx < TREE\_SIZE) {

int t = idx + mask;

if(x >= tree[t]) {idx = t; x -= tree[t]; }

mask >>= 1;

}

return idx;

}

int main(){

while(cin >> n){

a.assign(n,0); b.assign(n,0); tree.assign(n,0);

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

cin >> a[i]; b[i]=a[i];

}

sort(b.begin(),b.end());

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

int rank=(int)(lower\_bound(b.begin(),b.end(),a[i])-b.begin());

a[i]=rank+1;

}

int64 invs=0;//num of inversions

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

invs+=read(a[i]-1);

update(a[i],1);

}

cout << invs << endl;

}

return 0;

}

# Maximum Subrectangle Sum

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

for(int j=0 ; j<n ; j++)

a[i][j]+=a[i-1][j];

int Max=0, ans=0;

for(int k=0 ; k<n ; k++){//calc

for(int i=0 ; i<n-k ; i++){ Max=0;

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

if(Max<0) Max=a[i+k][j]-a[i][j];

else Max+=a[i+k][j]-a[i][j];

if(Max>ans) ans=Max;

} } }

//sub array, finsh and start point p=(val, startidx, finishidx)

p ans=p(-1,0,0); int sum=0,id=1;

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

if(sum<0){sum=0; id=i;}

sum+=a[i];

p tmp=p(sum,id,i+1); ans=Max(ans,tmp);

}

# Optimal Array Multiplication Sequence (Print Path)

int n,a[10+5],p[10+5][10+5],dp[10+5][10+5];

int solve(int L, int R){

if(L==R){ return 0; }

if(dp[L][R]!=-1) return dp[L][R];

int Min=INF;

for(int i=L ; i<R ; i++){

int slv=solve(L,i)+solve(i+1,R)+a[(L-1)]\*a[i]\*a[R];

if(Min>slv) Min=slv; p[L][R]=i;

}

return dp[L][R]=Min;

}

//prints like this => (A1 x (A2 x A3))

void print(int L, int R){

if(L==R){ cout << "A" <<L; return; }

cout << "("; print(L,p[L][R]);

cout << " x ";

print(p[L][R]+1,R); cout << ")";

}

int main(){ int t=1;

while(cin >> n && n){

for(int i=1 ; i<=n ; i++)cin >> a[i-1] >> a[i];

memset(dp,-1,sizeof dp);

solve(1,n);//cout << solve(1,n) << endl;

printf("Case %d: ",t++); print(1,n); printf("\n");

}

return 0;

}

# LIS

struct e{// this is for Handeling redundancy

e(){}

e(int a, int b){a1=a; a2=b;}

bool operator==(const e& a) const {return a.a1==a1 && a.a1==a1;}

bool operator>(const e& a) const {

if(a.a1<a1) return 1;

else if(a.a1==a1 && a.a2>a2) return 1;

return 0; }

bool operator<(const e& a) const{

if(a.a1>a1) return 1;

else if(a.a1==a1 && a.a2<a2) return 1;

return 0; }

int a1,a2;

}a[200005];

int main(){

int n,tmp;

for(n=0 ; cin >> tmp ; n++){

path[n]=n;

a[n]=e(tmp,n);

}

set<e> st; set<e>::iterator it; st.clear();

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

st.insert(a[i]); it=st.find(a[i]);

if(it!=st.begin()){ it--; path[i]=(it->a2); it++; }

it++;

if(it!=st.end()){ st.erase(it)}

}

it=st.end(); it--; //find last element of LIS

cout<< st.size() <<endl << "-" << endl;

print(it->a2);

}

//print path, lis

void print(int i){

if(i==0 || i==path[i]){cout << a[i].a1 << endl; retu;}

print(path[i]); cout << a[i].a1 << endl; }

//lis[i]+lds[j]-1; ;)

----------------------------------

for(int i=0 ; i<n ; i++){ dp[i]=1;

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

if(a[i]>a[j] && dp[i]<dp[j]+1){

dp[i]=dp[j]+1;

} } }

# LCS

dp[MAX][MAX]={0};

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

for(int j=1 ; j<=n ; j++){

if(a[i-1]==b[j-1]) dp[i][j]=dp[i-1][j-1]+1;

else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);

}}

cout << dp[n][n] << endl;

# TSP

p a[15]; int n, dp[15][1<<15];

int solve(int pos, int bitset){

int& dpp=dp[pos][bitset]; //dpp = dp poniter

if(bitset==(1<<n)-1) return dist(a[pos],a[0]);

if(dpp!=-1) return dpp;

dpp=INF;

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

if(!(bitset&(1<<i))) dpp=min(dpp,solve(i,bitset|(1<<i))+dist(a[pos],a[i]));

}

return dpp;

}

int main(){

int tc; cin >> tc;

while(tc--){

cin >> a[0].X >> a[0].Y; cin >> n; n++;

for(int i=1 ; i<n ; i++) cin >> a[i].X >> a[i].Y;

memset(dp, -1, sizeof dp);

cout << solve(0,1) << endl;

}

return 0;

}

# Bitmask

bit&(1<<i) // bit i is 0 or 1

(bit>>j)&1// bit i is 0 or 1 // use this & multiplication to avoid TLE

bit|(1<<i) // set bit i to 1

bit^(1<<i) // toggle bit i

x & ( x – 1) // check if x is a power of 2

string stmp; bitset<12> tmp; //Debuging

tmp=bit; stmp=tmp.to\_string();

# Articulation Points & Bridges

#define INF (int) 1e6

#define MAX 100

#define pii pair<int,int>

using namespace std;

int n, dfsLow[MAX+10], dfsNum[MAX+10],parent[MAX+10] , lev, dfsRoot,rootChilds;

vector<int> adj[MAX+10]; set<pii> bridges; set<int> artPoints;

void dfs(int u){

dfsLow[u]=dfsNum[u]=lev++;

for(int i=0 ; i<adj[u].size() ; i++){

int v=adj[u][i];

if(dfsNum[v]==0){

if(u==dfsRoot) rootChilds++;

parent[v]=u; dfs(v);

if(dfsLow[v]>=dfsNum[u] && u!=dfsRoot)//u is articulation point

artPoints.insert(u);

if(dfsLow[v]>dfsNum[u])//u,v is bridge

bridges.insert(pii(min(u,v),max(u,v)));

dfsLow[u]=min(dfsLow[u],dfsLow[v]);

}

else if(parent[u]!=v)

dfsLow[u]=min(dfsLow[u],dfsNum[v]);

}

}

int main(){

while(cin >> n){

for(int i=0 ; i<MAX+10; i++) dfsLow[i]=parent[i]=dfsNum[i]=0, adj[i].clear();

bridges.clear(); artPoints.clear(); lev=1; int tmp,u,m;//initialization

for(int i=0 ; i<n ; i++){ // construct the graph

scanf("%d (%d", &u, &m); cin.ignore();

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

cin >> tmp; adj[u].push\_back(tmp);

}

}

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

if(dfsNum[i]==0){

dfsRoot=i; rootChilds=0; dfs(i);

if(rootChilds>=2) artPoints.insert(dfsRoot);

}

}

printf("%d critical links\n", bridges.size());

set<pii>::iterator itr; // print answer

for(itr=bridges.begin() ; itr!=bridges.end() ; itr++)

printf("%d - %d\n", itr->first, itr->second);

cout << endl;

}

return 0;

}

# Finding Strongly Connected Components

#define MAX 100000

using namespace std;

int dfsNum[MAX+10],dfsLow[MAX+10],vis[MAX+10],in[MAX+10],n,m,lev,ans; vector<int> SCC,adj[MAX+10];

void dfs(int u){

dfsLow[u]=dfsNum[u]=lev++; vis[u]=1; SCC.push\_back(u);

for(int i=0 ; i<adj[u].size() ; i++){

int v=adj[u][i];

if(dfsNum[v]==0) dfs(v);

if(vis[v]) dfsLow[u]=min(dfsLow[u], dfsLow[v]), in[v]--;

}

if(dfsLow[u]==dfsNum[u]){

// this prints all vertices v blong to SCC with dfsLow[v] == dfsLow[u]

bool flag=true;

for(int i=0, v ; !SCC.empty() ; i++){

v=SCC.back(); SCC.pop\_back(); vis[v]=0;

printf("%d ", v);

if(in[v]) flag=false;

if(v==u) break;

}

printf("\n");

if(flag) ans++;

// counts number of SCCs without indegree outside of other SCCs

}

}

int main(){

int tc; scanf("%d", &tc);; int x,y;

while(tc--){

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

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

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

memset(in,0,sizeof in); lev=1; ans=0;

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

scanf("%d %d", &x, &y); x--; y--;

adj[x].push\_back(y); in[y]++;

}

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

if(dfsNum[i]==0) dfs(i);

}

cout << ans << endl;

}

return 0;

}

# Kruskal’s Minimum Spanning Tree

#define INF (int)1e9

#define MAX 1000

#define pii pair<int,int>

#define piii pair<int, pii >

#define W first

#define U second.first

#define V second.second

using namespace std;

vector< piii > E;// e==edges array

int m,n;

int main(){

while(cin >> n){

E.clear(); int u,v,w;

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

cin >> u >> v >> w;

E.push\_back(piii(w,pii(u,v)));

}

sort(E.begin(), E.end()); SetInit(n); int ans=0; int j=0;

for(int i=0; i<E.size() && j<m-1; i++){

if(Find(E[i].U)!=Find(E[i].V)){

Uni(E[i].U,E[i].V); ans+=E[i].W; j++; marked.push\_back(i);

}

}

cout << (j==m-1 ? ans : -1) << endl;

}

return 0;

}

# BFS Topological Sort

//store indegree of vertice u in indegree[u]

fr(i,n) if(!indegree[i]) q.push(i);

while(!q.empty()){

int v = q.front(); q.pop();

cout << v + 1 << " " ;

int s = adjlist[v].size();

fr(i,s){

if(!(--indegree[ adjlist[v][i] ])) q.push(adjlist[v][i]);

}

}

# Floyd Warshal (Print Path)

#define INF (int)1e9

#define MAX 100

using namespace std;

int adj[MAX+10][MAX+10],path[MAX+10][MAX+10]; int n;

void print(int i,int j){

if(i!=j){

printf(" %d",i );

print(path[i][j],j);

}

}

int main(){

int tc; cin >> tc;

while(tc--){

cin >> n;

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

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

adj[i][j]=INF; if(i==j) adj[i][j]=0;

path[i][j]=j;//initial parent

}

}

for(int k=0 ; k<n ; k++){

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

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

if(adj[i][j]>adj[i][k]+adj[k][j]){

adj[i][j]=adj[i][k]+adj[k][j];

path[i][j]=path[i][k];//set parent

}

}

}

}

int s,d;

cin >> s >> d;

printf("%d euros\n",adj[s][d]);

//this prints the path even if source and distinaion are same

printf("%d",s); print(path[s][d],d); printf(" %d\n",d);

}

return 0;

}

# Edmonds Karp’s

//UVa 820 - Internet Bandwidth

#define INF (int)1e9

#define MAX 100+10

using namespace std;

int res[MAX][MAX],mf,f,s,t,n,m,par[MAX]; vector<int> dist,adj[MAX];

void agument(int v, int minEdge){

if(v==s) f=minEdge;

else if(par[v]!=-1){

agument(par[v],min(minEdge,res[par[v]][v]));

res[par[v]][v]-=f; res[v][par[v]]+=f;

}

}

int main(){

int tc=1;

while(cin >> n && n){

mf=0; memset(res,0,sizeof res); for(int i=0 ; i<n ; i++) adj[i].clear();

cin >> s >> t >> m; s--; t--;

int u,v,c;

while(m--){

cin >> u >> v >> c; u--; v--;

res[u][v]+=c; res[v][u]+=c;

adj[u].push\_back(v); adj[v].push\_back(u);

}

while(1){

f=0; memset(par,-1,sizeof par); dist.assign(n,INF);

dist[s]=0; queue<int> q; q.push(s);

while(!q.empty()){

int u=q.front(); q.pop();

if(u==t) break;

for(int i=0 ; i<adj[u].size(); i++){

int v=adj[u][i];

if(res[u][v]>0 && dist[v]==INF){

dist[v]=dist[u]+1; q.push(v); par[v]=u;

}

}

}

agument(t,INF);

if(f==0) break;

mf+=f;

}

printf("Network %d\n", tc++);

printf("The bandwidth is %d.\n\n", mf);

}

return 0;}

# Alternating Path Algorithm for Max Bipartite Matching

//UVa 11138 - Nuts and Bolts // O(V^2 + VE)

#define vi vector<int>

using namespace std;

vector< vi > adj; vector<int> owner, vis; int n,b;

int altpath(int u){

if(vis[u]) return 0; vis[u]=1;

for(int i=0 ; i<adj[u].size() ; i++){

int v=adj[u][i];

if(owner[v]==-1 || altpath(owner[v])){

owner[v]=u; return 1;

}

}

return 0;

}

int main(){

int tmp,tc,t=1; cin >> tc;

while(tc--){

cin >> n >> b; adj.assign(n+b,vi());

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

for(int j=0 ; j<b ; j++){

cin >> tmp; if(tmp==1) adj[i].push\_back(j+n);

}

}

int ans=0; owner.assign(n+b,-1);

for(int u=0 ; u<n ; u++){

vis.assign(n,0); ans+=altpath(u);

}

printf("Case %d: a maximum of %d matched\n", t++, ans);

}

return 0;

}