## Template

#include<bits/stdc++.h>

#define pb push\_back

#define mp make\_pair

#define fs first

#define sd second

using namespace std;

typedef long long ll;

typedef unsigned long long ull;

typedef pair<ll,ll> pl;

typedef vector<ll> vl;

typedef vector<pair<ll,ll>> vll;

typedef vector<pair<ll,pl>> vlll;

typedef priority\_queue <ll, vector<ll>, greater<ll>> minh;

const int N = 1e6 + 3, Mod = 1e9 + 7;

const int maxN=1e3+3;

void solve(){

}

int main(){

ios\_base::sync\_with\_stdio(false);

cin.tie(nullptr);

cout.tie(nullptr);

int t=1;

cin>>t;

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

//cout<<"Case #"<<i+1<<": ";

solve();

}

}

## Data Structres

#include<bits/stdc++.h>

using namespace std;

typedef vector<int> vi;

const int N=1e6+3, Mod=1e9+7;

const int maxN=1e4+2;

//union find disjoint set

class dsu{

private:

vl p, rank, setSize;

int numSets;

public:

dsu(int n){

setSize.assign(n,1);

numSets=n;

rank.assign(n,0);

p.assign(n,0);

for(int i=0;i<n;i++)p[i]=i;

}

int findP(int i){

return (p[i]==i)?i:findP(p[i]);

}

bool isSameP(int a, int b){

return findP(a)==findP(b);

}

void unite(int a, int b){

if(!isSameP(a,b)){

numSets--;

int x=findP(a);

int y=findP(b);

if(rank[x]>rank[y]){

p[y]=x;

setSize[x]+=setSize[y];

}else{

p[x]=y;

setSize[y]+=setSize[x];

if(rank[x]==rank[y])rank[y]++;

}

}

}

int totSets(){

return numSets;

}

int sizeOfSet(int i){

return setSize[findP(i)];

}

};

//segment tree (single update)

int a[maxN]; //tree source

struct info {

int l, r, fl, fr, sz;

long long tot;

} d[maxN \* 4]; //tree storage

info operator + (info a, info b) {

info c;

c.l = a.l;

c.r = b.r;

c.sz = a.sz + b.sz;

c.fl = (a.fl == a.sz && a.r <= b.l) ? (a.fl + b.fl) : a.fl;

c.fr = (b.fr == b.sz && a.r <= b.l) ? (b.fr + a.fr) : b.fr;

c.tot = a.tot + b.tot + ((a.r <= b.l) ? (1ll \* a.fr \* b.fl) : 0);

return c;

} //merging two datas (ans)

void build(int k, int l, int r) { //(1,1,n)

if (l == r) {

d[k] = (info) {a[l], a[l], 1, 1, 1, 1ll};

} else {

int mid = (l + r) / 2;

build(k \* 2, l, mid);

build(k \* 2 + 1, mid + 1, r);

d[k] = d[k \* 2] + d[k \* 2 + 1]; //operator +

}

}

//rebuild, only with node x, change the array directly

void update(int k, int l, int r, int x) { //(1,1,n,node)

if (l == r) {

d[k] = (info) {a[l], a[l], 1, 1, 1, 1ll};

} else {

int mid = (l + r) / 2;

if (x <= mid) update(k \* 2, l, mid, x);

else update(k \* 2 + 1, mid + 1, r, x);

d[k] = d[k \* 2] + d[k \* 2 + 1]; //operator +

}

}

info query(int k, int l, int r, int x, int y) { //(1,1,n,leftq,rightq)

if (l == x && r == y) {

return d[k];

} else {

int mid = (l + r) / 2;

if (y <= mid) return query(k \* 2, l, mid, x, y);

else if (x > mid) return query(k \* 2 + 1, mid + 1, r, x, y);

else return query(k \* 2, l, mid, x, mid) + query(k \* 2 + 1, mid + 1, r, mid + 1, y); //operator +

}

}

//fenwick tree

class fenwick{

private:

vl ft;

public:

fenwick(int n){

ft.assign(n+1,0);

}

int rsq(int b){

int ret=0;

for(;b;b-=(b&(-b)))ret+=ft[b];

return ret;

}

int rsq(int a,int b){

return rsq(b)-(a==1?0:rsq(a-1));

}

void update(int i, int val){

for(;i<(int)ft.size();i+=(i&-i))ft[i]+=val;

}

};

## Algorithms

//Binary Search

ll binser(ll l,ll r,ll val){

if(l>=r)return l;

ll mid=(l+r)/2;

//printf(".%d",mid);

if(arr[mid]==val)return mid;

if(arr[mid]>val&&arr[mid-1]<val)return mid;

if(arr[mid]>val) return binser(l,mid-1,val);

if(arr[mid]<val) return binser(mid+1,r,val);

}

//fast C(N,K)

int n, k;

long long fact[N], invf[N], inv[N];

long long modpow(long long x, long long y) {

long long ret = 1;

while (y > 0) {

if (y & 1) ret = (ret \* x) % Mod;

y >>= 1;

x = (x \* x) % Mod;

}

return ret;

}

void preprocess() {

fact[0] = invf[0] = 1;

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

fact[i] = (fact[i - 1] \* i) % Mod;

invf[i] = modpow(fact[i], Mod - 2);

inv[i]=modpow(i,Mod-2);

}

}

long long C(int a, int b) {

if (a < b) return 0;

long long ret = (fact[a] \* invf[a - b]) % Mod;

ret = (ret \* invf[b]) % Mod;

return ret;

}

//Dijkstra

Chart

Description automatically generated

//Bellman Ford

Text

Description automatically generated

//Floyd Warshall

Text

Description automatically generated

//Articulation Point

Text, letter

Description automatically generated

//Tarjan SCC

Text, letter

Description automatically generated

//Toposort, lexicographically smallest (inside main)

ll n,m; cin>>n>>m;

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

ll x,y; cin>>x>>y;

adj[x].pb(y);

w[y]++;

}

vl ans;

priority\_queue <ll, vector<ll>, greater<ll>>pq;

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

if(w[i])continue;

pq.push(i);

}

while(!pq.empty()){

ll idx=pq.top(); pq.pop();

ans.pb(idx);

for(auto v: adj[idx]){

w[v]--;

if(!w[v])pq.push(v);

}

}

// Cycle-finding DFS

ll adj[maxN];

ll w[maxN];

int visit[maxN];

vl path;

ll ans=0;

void dfs(ll i){

path.pb(i);

visit[i]=1;

if(visit[adj[i]]==1){

int n=path.size()-1;

ll ret=w[path[n]];

while(path[n]!=adj[i]){

ret=min(ret,w[path[n]]); //or pb to cycle list

n--;

}

ret=min(ret,w[path[n]]);

ans+=ret;

}

if(visit[adj[i]]==0){

dfs(adj[i]);

}

visit[i]=2;

}

//Bipartite (Color 1,2)

vl adj[maxN];

int col[maxN];

int cnt[3];

bool f=1;

int xxor(int a){

if(a==1)return 2;

else return 1;

}

bool dfs(int u){

for(auto v:adj[u]){

if(col[v]==0){

col[v]=xxor(col[u]);

cnt[col[v]]++;

if(!dfs(v))return 0;

}

else if(col[v]!=xxor(col[u])){

f=0;

return 0;

}

}

return 1;

}

//KMP String

Text, letter

Description automatically generated

## Math

//Floyd’s Hare and Turtle

Text, letter

Description automatically generated

//Grundy Nim

Text

Description automatically generated

//Catalan Numbers

Table

Description automatically generated with low confidence

Text, letter

Description automatically generated