FINDING ALL THE CLIQUE IN A GRAPH :

#include <bits/stdc++.h>

using namespace std;

/\*\*\*template\*\*\*/

#define ll long long

#define dd double

#define scl(n) scanf("%lld",&n)

#define sci(n) scanf("%d",&n)

#define scd(n) scanf("%lf",&n)

#define pi pair<ll,ll>

#define pb push\_back

#define mp make\_pair

#define uu first

#define vv second

#define FOR(i,n) for(ll i=1;i<=n;i++)

#define LOOP(i,n) for(ll i=0;i<n;i++)

#define FOOR(i,a,b) for(ll i=a;i<=b;i++)

#define sorted(s) sort(s.begin(),s.end())

#define reversed(s) reverse(s.begin(),s.end())

#define contains(a,b) (find((a).begin(), (a).end(), (b)) != (a).end())

#define maxn 500

#define pii 3.1415926536

#define

/\*\*\*template\*\*\*/

ll n,m,best;

vector<ll>graph[maxn];

vector<pi>degree;

ll silver[maxn];

bool dis[maxn];

void cle()

{

FOOR(i,0,n)

{

graph[i].clear();

dis[i]=false;

}

best=0;

degree.clear();

}

void in\_silver()

{

LOOP(i,n)

{

scl(silver[i]);

best=max(best,silver[i]);

}

}

void in\_edges()

{

ll u,v;

FOR(i,m)

{

scl(u);

scl(v);

u--;v--;

graph[u].pb(v);

graph[v].pb(u);

}

}

void making\_degeneracy\_ordering()

{

LOOP(i,n)

{

degree.pb(mp(graph[i].size(),i));

}

sorted(degree);

}

void chain(vector<ll>cur\_nodes,vector<ll>sel,ll pos,vector<ll>clique)

{

if (pos >= cur\_nodes.size())

{

LOOP(i,clique.size())

{

cout<<clique[i]<<" ";

}

cout<<endl;

return ;

}

chain(cur\_nodes, sel, pos+1, clique);

bool possible = true;

LOOP(i,sel.size())

{

if (!contains(graph[cur\_nodes[pos]], sel[i]))

{

possible = false;

}

}

sel.push\_back(cur\_nodes[pos]);

if (possible)

{

vector <ll>clique2;

clique2=clique;

clique2.pb(cur\_nodes[pos]);

chain(cur\_nodes, sel, pos + 1,clique2);

}

}

int main()

{

while(scanf("%lld%lld",&n,&m)==2)

{

cle();

in\_silver();

in\_edges();

making\_degeneracy\_ordering();

LOOP(i,n)

{

ll cur\_node = degree[i].vv;

vector<ll> cur\_nodes;

LOOP(j,graph[cur\_node].size())

{

int adj = graph[cur\_node][j];

if (!dis[adj])

{

cur\_nodes.pb(adj);

}

}

vector<ll> sel;

vector<ll>clique;

clique.pb(cur\_node);

chain(cur\_nodes, sel, 0, clique);

dis[cur\_node] = true;

}

}

return 0;

}

MATRIX FAST EXPO :

#include <bits/stdc++.h>

using namespace std;

/\*\*\*template\*\*\*/

#define ll long long

#define dd double

#define scl(n) scanf("%lld",&n)

#define pi pair<ll,ll>

#define pb push\_back

#define mp make\_pair

#define uu first

#define vv second

#define FOR(i,n) for(ll i=1;i<=n;i++)

#define LOOP(i,n) for(ll i=0;i<n;i++)

#define FOOR(i,a,b) for(ll i=a;i<=b;i++)

#define LOP(i,a,b) for(ll i=a;i<b;i++)

#define mstt(a,b) memset((a),(b),sizeof (a))

#define mat\_size 3

long long mod, col\_num = mat\_size, row\_num = mat\_size;

long long mat[mat\_size][mat\_size], temp[mat\_size][mat\_size], res[mat\_size][mat\_size];

void matpower(long long pwr)

{

long long i, j, k, s;

if (pwr == 0)

{

for (i = 0; i < row\_num; i++)

{

for (j = 0; j < col\_num; j++)

{

if (i != j)

{

res[i][j] = 0;

}

else

{

res[i][j] = 1;

}

}

}

return ;

}

matpower(pwr/2);

for (i = 0; i < col\_num; i++)

{

for (j = 0; j < row\_num; j++)

{

temp[i][j] = res[i][j];

}

}

for (i = 0; i < col\_num; i++)

{

for (j = 0; j < row\_num; j++)

{

s = 0;

for (k = 0; k < row\_num; k++)

{

s = (s + (temp[i][k]\*temp[k][j])%mod)%mod;

}

res[i][j] = s;

}

}

if (pwr%2 == 0)

{

}

else

{

for (i = 0; i < row\_num; i++)

{

for (j = 0; j < row\_num; j++)

{

temp[i][j] = res[i][j];

}

}

for (i = 0; i < row\_num; i++)

{

for (j = 0; j < col\_num; j++)

{

s = 0;

for (k = 0; k < col\_num; k++)

s = (s + (mat[i][k]\*(temp[k][j]))%mod)%mod;

res[i][j] = s;

}

}

}

}

int main ()

{

mat[0][0] = 2;

mat[0][1] = 2;

mat[0][2] = 3;

mat[1][0] = 2;

mat[1][1] = 2;

mat[1][2] = 4;

mat[2][0] = 5;

mat[2][1] = 5;

mat[2][2] = 5;

mod=1000; /\* declaring the base matrix by putting value in mat array \*/

matpower(5);

for(int i=0;i<mat\_size;i++)

{

for(int j=0;j<mat\_size;j++)

{

printf("%d ",res[i][j]);// Now, res array will have mat^5

}

printf("\n");

}

return 0;

}

# PERSISTENT SEGMENT TREE :

struct node

{

node \*left,\*right;

ll sum;

node(ll a = 0, node \*b = NULL, node \*c = NULL) :

sum(a), left(b), right(c) {} // \*\* Constructor

void build(ll l,ll r) // We are not initializing values for now.

{

if(l == r)

{

return;

}

left = new node(); // Create new node for Left child

right = new node();// We are creating nodes only when necessary!

ll mid = l + r >> 1;

left -> build(l, mid);

right -> build(mid+1, r);

}

node \*update(ll l,ll r,ll idx,ll v)

{

if(r < idx || l > idx)

{

return this; // Out of range, use this node.

}

if(l == r) // Leaf Node, create new node and return that.

{

node \*ret = new node(sum, left, right);

ret -> sum += v;

return ret;

// we first cloned our current node, then added v to the value.

}

ll mid = l + r >> 1;

node \*ret = new node(sum); //Create a new node, as idx in in [l, r]

ret -> left = left -> update(l, mid, idx, v);

ret -> right = right -> update(mid+1, r, idx, v);

// Note that 'ret -> left' is new node's left child,

// But 'left' is current old node's left child.

// So we call to update idx in left child of old node.

// And use it's return node as new node's left child. Same for right.

ret -> sum = ret -> left -> sum + ret -> right -> sum; // Update value.

return ret; // Return the new node to parent.

}

ll query(ll l,ll r,ll i,ll j)

{

if(r < i || l > j)

{

return 0; // out of range

}

if(i <= l && r <= j) // completely inside

{

return sum; // return value stored in this node

}

ll mid = l + r >> 1;

return left -> query(l,mid,i,j) + right -> query(mid+1,r,i,j);

}

}\*root[600005];

int main()

{

root[0] = new node();

root[0] -> build(1,8);

root[1]=root[0] -> update(1,8,1,2);

root[2]=root[1] -> update(1,8,2,1);

root[3]=root[2] -> update(1,8,3,3);

cout<<root[0]->query(1,8,1,8)<<" "<<root[1]->query(1,8,1,8)<<" "<<root[2]->query(1,8,1,8)<<" "<<root[3]->query(1,8,1,8)<<" ";

}

2d segment tree:

#include <bits/stdc++.h>

using namespace std;

#define ll long long

ll a[5][5];

ll t[20][20];

ll n,m;

void build\_y (int vx, int lx, int rx, int vy, int ly, int ry)

{

if (ly == ry)

if (lx == rx)

t[vx][vy] = a[lx][ly];

else

t[vx][vy] = max(t[vx\*2][vy] , t[vx\*2+1][vy]);

else

{

int my = (ly + ry) / 2;

build\_y (vx, lx, rx, vy\*2, ly, my);

build\_y (vx, lx, rx, vy\*2+1, my+1, ry);

t[vx][vy] =max( t[vx][vy\*2] , t[vx][vy\*2+1]);

}

}

void build\_x (int vx, int lx, int rx)

{

if (lx != rx)

{

int mx = (lx + rx) / 2;

build\_x (vx\*2, lx, mx);

build\_x (vx\*2+1, mx+1, rx);

}

build\_y (vx, lx, rx, 1, 1, m);

}

int sum\_y (int vx, int vy, int tly, int try\_, int ly, int ry)

{

if (ly > ry)

return 0;

if (ly == tly && try\_ == ry)

return t[vx][vy];

int tmy = (tly + try\_) / 2;

return max(sum\_y (vx, vy\*2, tly, tmy, ly, min(ry,tmy))

, sum\_y (vx, vy\*2+1, tmy+1, try\_, max(ly,tmy+1), ry));

}

int sum\_x (int vx, int tlx, int trx, int lx, int rx, int ly, int ry)

{

if (lx > rx)

return 0;

if (lx == tlx && trx == rx)

return sum\_y (vx, 1, 1, m, ly, ry);

int tmx = (tlx + trx) / 2;

return max(sum\_x (vx\*2, tlx, tmx, lx, min(rx,tmx), ly, ry)

, sum\_x (vx\*2+1, tmx+1, trx, max(lx,tmx+1), rx, ly, ry));

}

void update\_y (int vx, int lx, int rx, int vy, int ly, int ry, int x, int y, int new\_val)

{

if (ly == ry)

{

if (lx == rx)

t[vx][vy] = new\_val;

else

t[vx][vy] = t[vx\*2][vy] + t[vx\*2+1][vy];

}

else

{

int my = (ly + ry) / 2;

if (y <= my)

update\_y (vx, lx, rx, vy\*2, ly, my, x, y, new\_val);

else

update\_y (vx, lx, rx, vy\*2+1, my+1, ry, x, y, new\_val);

t[vx][vy] = t[vx][vy\*2] + t[vx][vy\*2+1];

}

}

void update\_x (int vx, int lx, int rx, int x, int y, int new\_val)

{

if (lx != rx)

{

int mx = (lx + rx) / 2;

if (x <= mx)

update\_x (vx\*2, lx, mx, x, y, new\_val);

else

update\_x (vx\*2+1, mx+1, rx, x, y, new\_val);

}

update\_y (vx, lx, rx, 1, 1, m, x, y, new\_val);

}

int main()

{

n=4,m=4;

for(ll i=1; i<=4; i++)

{

for(ll j=1; j<=4; j++)

{

scanf("%lld",&a[i][j]);

}

}

build\_x(1,1,n);

for(ll i=1; i<=4\*n; i++)

{

for(ll j=1; j<=4\*m; j++)

{

cout<<t[i][j]<<" ";

}

cout<<endl;

}

cout<<sum\_x(1,1,n,2,2,1,2)<<endl;

cout<<sum\_x(1,1,n,2,2,1,2)<<endl;

}