https://www.overleaf.com/8163716681syrzymzwjjtn

TIPS:

1. visita todas las componentes del grafo!

inverse(a) = (inverse(prime % a) \* (prime - prime/a)) % prime

#include <bits/stdc++.h>

#include <iostream>

#include <algorithm>

#include <stdio.h>

#include <string>

#include <vector>

#include <map>

#include <math.h>

#include <numeric>

#include <queue>

#include <stack>

#include <utility>

#include <queue>

#include <set>

#include <iomanip>

using namespace std;

typedef int64\_t ll;

typedef pair<int,ll> pill;

typedef pair<int,int> pii;

#define INF 1000000000000

#define MOD 1000000007

#define loop(i,a,b) for(int i = a; i < b; ++i)

const int maxN = 100005;

void solve(){

}

int main(){

ios\_base::sync\_with\_stdio(0); cin.tie(0);

//if(fopen("case.txt", "r")) freopen("case.txt", "r", stdin);

int t = 1;

//int t; cin>>t;

loop(i, 0, t){

solve();

}

return 0;

}

vi adj[maxN];

int papa[maxN];

bool visited[maxN];

int level[maxN];

DFS

void dfs(int f, int nd){

for(int son : adj[nd]){

if(son == f) continue

//si eres back edge

if(visited[son] && level[son] < level[nd]){

}

//si eres front edje

else if(!visited[son]){

dfs(nd, son);

}

}

}

void recorridoInicial(int nd){

for(pill h: arbol[nd])

if(h.first != papa[nd])

{

papa[h.first] = nd;

++cantHijos[nd];

recorridoInicial(h.first);

}

}

Puentes

// =========== Puentes

/\*

dp[u] = numero de back edges que van de un "hijo" de u a un "ancestro" de u

puente entre a y b syss

dp[u] = 0

calculo de dp[u] = Suma(dp[hijos]) - backedgesQueLLeganAU + backEdgedSalenU

\*/

int dp[maxN];

int llegan[maxN];

bool hayPuente;

void dfs(int f, int nd){

visited[nd] = true;

level[nd] = level[f] + 1;

int salen = 0;

for(int son : adj[nd]){

if(son == f) continue;

//si eres back edge

if(visited[son] && level[son] < level[nd]){

++salen;

++llegan[son];

roads.push\_back(pii(nd, son));

}

//si eres front edje

else if(!visited[son]){

dfs(nd, son);

dp[nd] += dp[son];

roads.push\_back(pii(nd, son));

}

}

dp[nd] = dp[nd] - llegan[nd] + salen;

if(nd != 1 && dp[nd] == 0)

hayPuente = true;

}

loop(i, 0, m){

int a, b;

cin>>a>>b;

adj[a].push\_back(b);

adj[b].push\_back(a);

}

BFS

void bfs(int i){

queue<pii> c;

c.push(pii(i,0));

int nd, v;

fill(visited, visited + n + 5, 0);

while(!c.empty()){

nd = c.front().first; v = c.front().second;

c.pop();

if(!visited[nd]){

visited[nd] = true;

//meto a mis hijos con distancia mas uno

for(int h: adj[nd])

c.push(pii(h, v+1));

}

}

}

//=================== LCA

//asumiento que los nodos van de 1 a n

// si van de 0 a n-1 cambia el create LCA Table en el for nd = 0; nd < n

//siempre asigna a la raiz como su propio padre

// y su nivel como 1 o 0 dependiendo de que te convenga

//en main agrega

// assignParents(raiz, raiz);

// createLCATable();

struct node{

int level;

}nodes[maxN+1];

Const int LogN = 18;

int table[LogN+1][maxN+1], n;

void assignParents(int nd, int fa = nd){

for(int son: adj[nd]){

if(son != fa){

nodes[son].level = 1 + nodes[nd].level;

table[0][son] = nd;

//aqui puedes aprovechar para agregar funcionalidad que ocupes para el

//problema especifico

assignParents(son, nd);

}

}

}

//creates LCA Binary Lifting table

void createLCATable(){

int mid;

for(int k = 1; k<=LogN; ++k){

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

mid = table[k-1][nd];

table[k][nd] = table[k-1][mid];

}

}

}

//to get the k-th father of node i

int jump(int i, int k){

int x=0;

while(k > 0){

if(k&1)

i = table[x][i];

++x;

k>>=1;

}

return i;

}

int lca(int i, int j){

//to put i and j in the same level

if(nodes[i].level< nodes[j].level)

j = jump(j, nodes[j].level-nodes[i].level);

else

i = jump(i, nodes[i].level-nodes[j].level);

//these happens if one of them was lca of them

if(i==j)

return i;

//these is to put both just below the LCA

for(int d = LogN; d>=0; --d){

if(table[d][i] != table[d][j]){

i = table[d][i];

j = table[d][j];

}

}

//as they were just below the LCA, the LCA is the parent of either of the nodes

return table[0][i];

}

//linear sieve criba lineal

vector<ll> prime;

bool primo[maxP];

map<pii, int> mapeo;

void sieve () {

fill (primo, primo + maxP, true);

primo[0] = primo[1] = false;

for (int i = 2; i < maxP; ++i) {

if (primo[i]) prime.push\_back(i);

for (int j = 0; j < prime.size() && i \* prime[j] < maxP; ++j) {

primo[i \* prime[j]] = false;

if (i % prime[j] == 0) break;

}

}

}

//=============== Exponenciacion bianria

ll binPow(ll base, ll exp){

ll r = 1;

while (exp > 0){

if( exp & 1)

r = (r\*base) % MOD;

exp >>= 1;

base = (base \* base) % MOD;

}

return r;

}

Fenwick Tree

void update(int pos, ll x){

while(pos <= l){

bit[pos] = bit[pos] + x;

pos += pos&(-pos);

}

}

ll query(int pos){

ll suma = 0;

while(pos > 0){

suma = suma + bit[pos];

pos -= pos&(-pos);

}

return suma;

}

ll query(int pos1, int pos2){

return query(pos2) - query(pos1-1);

}

//gets the greatest common divisor of two numbers

int gcd(int a, int b){

if(a == 0)

return b;

else

return gcd(b%a, a);

}

/\*

convierte un string a un string con caracteres ordenados

y guarda en bucket la cantidad de veces que esta una letra en el string inicial

regresa el tamanio del string final

a - string inicial

bucket - vector que tiene cuantos hay de cada letra

s - string que guarda la lista de caracteres ordenados

ejemplo:

IN: aaaazzzbbb

Out: s = abz

bucket = {4, 3, 3}

regresa 3

\*/

int bucketString(string a, vi &bucket, string &s){

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

int sizeS = 1;

s = a.substr(0, 1);

bucket.push\_back(0);

for(char c : a){

if( c == s[sizeS-1])

++bucket[sizeS-1];

else

{

s += c;

bucket.push\_back(1);

++sizeS;

}

}

}

Maximum Bipartite Matching

Todo se basa en:

El emparejamiento bipartito actual es máximo si y sólo si no hay un “augmenting path” en el grafo.

Un augmenting path es un camino que alterna entre aristas libres y ocupadas y empieza y termina con un nodo libre.

Algoritmo:

1) Initialize Maximal Matching M as empty.

2) While there exists an Augmenting Path p

Remove matching edges of p from M and add not-matching edges of p to M

(This increases size of M by 1 as p starts and ends with a free vertex)

3) Return M.

// ================ MAX BIPARTITE MATCHING

//instructions:

// create bipartite graph normally

// run dfs to determine the side of each vertex

// run maxBPM()

// global variable

// n total size of both sides

const int maxN = 505;

int n, finalNode;

int visited[maxN], side[maxN], match[maxN], parent[maxN];

vector<int> adj[maxN];

#define noOne maxN+1

#define LEFT 0

#define RIGHT 1

void dfs(int v, int parity){

visited[v] = true;

side[v] = parity&1? LEFT : RIGHT;

for(int son : adj[v]){

if(!visited[son])

dfs(son, parity^1);

}

}

bool BFS(){

queue<int> bfs;

bool newPath = false;

fill(visited, visited+n, 0);

//free vertex from left side into the bfs

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

if(match[i] == noOne && side[i] == LEFT){

bfs.push(i);

visited[i] = true;

}

}

//perform bfs

while(!bfs.empty()){

int v = bfs.front();

bfs.pop();

// we are on the right side looking for

// a matching edge

if(side[v] == RIGHT){

//if its a free vertex I've already finished

if(match[v] == noOne){

newPath = true;

finalNode = v;

break;

}

else{

int myMatch = match[v];

//I mark it as visited and add it to bfs

//I'm the only one who can visit it in bfs

visited[myMatch] = true;

parent[myMatch] = v;

bfs.push(myMatch);

}

}

// we are on the left side looking for

// a non-matching edge

else{

for(int son : adj[v]){

//if it's visited already, I don't visit it

// i don't check whether it's my match or not because

// only my match can push me to bfs and it's already visited

if(visited[son]) continue;

//I mark it as visited and add it to bfs

visited[son] = true;

parent[son] = v;

bfs.push(son);

}

}

}

return newPath;

}

int maxBPM(){

int sizeMatching = 0;

fill(match, match+n, noOne);

while(BFS()){

++sizeMatching;

int v = finalNode;

int parity = 1;

while(parent[v] != v){

//we reassign matching nodes in odd number steps

if(parity){

match[v] = parent[v];

match[parent[v]] = v;

}

parity &= 1;

v = parent[v];

}

}

return sizeMatching;

}

//=================== FIN MAXIMUM BIPARTITE MATCHING

int main(){

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

if(!visited[i])

dfs(i, 1);

cout<<maxBPM()<<endl;

return 0;

}

//=== precalcula factorial

ll fact[maxN], invFact[maxN];

ll binPow(ll base, ll exp){

ll r = 1;

while (exp > 0){

if( exp & 1)

r = (r\*base) % MOD;

exp >>= 1;

base = (base \* base) % MOD;

}

return r;

}

ll comb(int num, int k){

if (num < 0 || num < k) return 0;

return (((fact[num]\*invFact[k])%MOD)\*invFact[num-k])%MOD;

}

fact[0] = invFact[0] = 1;

for(int i = 1; i <= 2\*1e5 + 5; ++i){

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

invFact[i] = binPow(fact[i], MOD-2);

}

%string a int numero

int toInt(string numero){

int resp = 0;

for(char c : numero){

resp = 10\*resp + (c-'0');

}

return resp;

}

Struct matriz

struct matrix{

vector< vector<ll> > m;

ll mod, sz;

ll mod2;

matrix (ll n,ll modc) : sz(n),m(n), mod(modc) {

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

m[i].resize(n);

mod2=mod\*mod;

}

matrix operator\*(matrix b)

{

matrix ans(sz,mod);

for(int i=0;i<sz;i++)

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

for(int u=0;u<sz;u++)

{

ans.m[i][u]+=m[i][j]\*b.m[j][u];

if(ans.m[i][u]>=mod2)

ans.m[i][u]-=mod2;

}

for(int i=0;i<sz;i++)

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

ans.m[i][j]%=mod;

return ans;

}

matrix pow(ll e)

{

if(e==1)

return \*this;

matrix x =pow(e/2);

x=(x\*x);

if(e&1)

x=(x\*(\*this));

return x;

}

};

Loca People

#include <iostream>

#include <stack>

using namespace std;

const int maxN = 1005;

int n, m;

typedef pair<int,int> pii;

stack<pii> alturas;

int extension[maxN];

int entreFilas[maxN][maxN];

int mat[maxN][maxN];

void leeAlturas(){

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

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

int h; cin>>h;

if(h == 1){

if(i > 0)

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

else

{

mat[i][j] = 1;

}

}

else{

mat[i][j] = 0;

}

}

}

void clear(){

stack<pii> empty;

swap(empty, alturas);

fill(extension, extension + maxN, 0);

}

void agrega(int h){

int como = 0;

while(!alturas.empty() && alturas.top().first >= h){

como += alturas.top().second + 1;

int tope = alturas.top().first;

alturas.pop();

//tu extension es todo lo que yo puedo comerme

extension[tope] = max( extension[tope], como);

}

alturas.push(pii(h, como));

}

int main(){

ios\_base::sync\_with\_stdio(0); cin.tie(0);

cin>>n>>m;

leeAlturas();

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

clear();

//destruyendo edificios para calcular maxima extension de cada altura

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

agrega(mat[i][j]);

agrega(0);

//extension maxima para cada altura es la maxima entre la tuya y la de las alturas mayores

for(int j = i; j >=1; --j){

extension[j] = max(extension[j], extension[j+1]);

}

//calculo entre filas

// tu area maxima entre la fila j e i con edificios que empiezan en el piso i

// es el maximo entre los que llegan hasta j-1 y los de la altura maxima

entreFilas[i][i] = extension[1];

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

entreFilas[j][i] = max(entreFilas[j+1][i], (i-j+1)\*extension[i-j+1]);

}

//ahora el maximo entre la fila j e i, o tiene piso en i, o ya esta guardado en el area maxima entre j e i-1

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

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

}

}

int q; cin>>q;

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

int a, b; cin>>a>>b;

--a;--b;

cout<<entreFilas[a][b]<<"\n";

}

}

Xbox Live Points

ll arre[maxN];

ll xbox(int length){

ll suma = 0, maxi = -10000000000;

int aux;

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

suma += arre[i];

maxi = max(maxi, suma);

if(suma < 0)

suma = 0;

}

return maxi;

}

Maximum Bipartite Matching

Si el matching no es maximo, existe un augmenting path del grafo (camino alternante entre aristas del match y no del match, el inicio y el final del camino son vertices no en el match)

Asi que mientras encuentres match, puedes hacer el match mas grande

Lista adj,

Si yo estoy en match, y mi vecino tambien, la arista es match

Si yo no estoy en match o mi vecino no, la arista no es match

Con dfs digo si estás en un conjunto o en el otro

Bool enA[] o enIzq[]

Via Augmenting paths:

int match[maxN] .//te dice con quien estas asociado

* BFS:

Arreglo papas[

Para cada nodo en A, si eres free, te meto a BFS

En BFS: while no vacia && no final

Pop();

If(visited) continue;

Visited[] = true

Papa[nd] = papa

Si soy de B,

Si soy free, me marco como final, break;

Si match, meto a Bfs a mis hijos de A unidos con aristas de match

Si soy de A, (soy match)

Meto a bfs a mis hijos de B unidos con aristas free

Cuando salgo, if vacia, el matching es maximo

Else

Vacio el arreglo match

While(yo no sea mi propio papa

Si soy de b, asocio a mi papa conmigo y viceversa

Final = papa;

Si soy de a, final = papa;

Repito

Bipartite graph

Tienes dos grupos de nodos

La gráfica se da por aristas que conecta a dos nodos de diferentes grupos

Max bipartite matching

Maxima cantidad de parejas que pueden emparejarse, donde un nodo sólo puede pertenecer a una pareja

Maximum vertex cover

Conjunto de nodos tal que para toda arista, esta está ligada a un nodo del conjunto

independent set

Conjunto de nodos tal que no hay dos adyacentes

En gráficas bipartitas

Min vertex cover = max bipartite matching’

Además

Max independent set + min vertex cover = # nodos de la gráfica

Así max indp set = #nodos - maxBipartite

Si hay dos tipos de eventos, y un evento excluye a algunos del otro tipo, puedes obtener el maximo que coexisten obteniendo el max Independent Set = vertices – tamanio max Bipartite matching

Idea del MST

Prim, kruskal

Teoream 11.1? del algorithms

Si no estás en el óptimo, construyo uno igual óptimo contigo

Te uno, en el simple path se forma un ciclo, elijo cualquiera d elos del arbol, lo corto y se separa en dos componentes y contigo las uno.

Strongly connected components

Suffix array

KMP