**Referencias**

ÍNDICE

Exponenciación binaria 2

Minimum Range Query 3

LCA Binary Lifting 4

BFS 6

Leer linea con enteros in saber cuántos son 6

Datos primitivos 7

Combinaciones Memo 8

Combinaciones mod primo 8

Fibonacci Exp Binaria 8

Algoritmo de Pollard Rho 9

Destruyendo Edificios 10

Loca People 11

Nieves Corriendo 13

Lectura rápida Java 16

Fotos 17

Keep it Healthy 19

Propiedades de Fibonacci 20

COMPU

# Exponenciación binaria

long long binPowLoop(long long base, long long exp){

long long r = 1;

while (exp > 0){

if( exp & 1)

r = r\*base;

exp >>= 1;

base = (base \* base);

}

return r;

}

long long binPowRecursive(long long a, long long b) {

if (b == 0)

return 1;

long long res = binpow(a, b / 2);

if (b % 2)

return res \* res \* a;

else

return res \* res;

}

long long binPowLoopMOD(long long base, long long exp, long MOD){

long long r = 1;

while (exp > 0){

if( exp & 1)

r = (r\*base) % MOD;

exp >>= 1;

base = (base \* base) % MOD;

}

return r;

}

long long binPowRecursiveMOD(long long a, long long b, long long MOD) {

if (b == 0)

return 1;

long long res = binpow(a, b / 2);

if (b % 2)

return (res \* res \* a) % MOD;

else

return (res \* res) % MOD;

}

# Minimum Range Query

#include<bits/stdc++.h>

using namespace std;

int getMid(int a, int b){return a+(b-a)/2;}

int constructSTSub(int arr[],int \*st,int si,int ss,int se){

if(se == ss)

st[si] = arr[ss];

else{

int mid = getMid(ss, se);

st[si] = min(constructSTSub(arr, st, 2\*si+1, ss, mid),

constructSTSub(arr, st, 2\*si+2, mid+1, se));

}

return st[si];

}

int \*constructST(int arr[],int n)

{

int x = (int)ceil(log2(n));

int maxSize = (int)pow(2, x+1) - 1;

int \*st = new int[maxSize];

constructSTSub(arr, st, 0, 0, n-1);

return st;

}

int RMQSub(int \*st, int si, int ss, int se, int qs, int qe){

if(qs<=ss && se<=qe)

return st[si];

else if(se<qs || qe<ss)

return INT\_MAX;

else{

int mid = getMid(ss, se);

return min(RMQSub(st, 2\*si+1, ss, mid, qs, qe),

RMQSub(st, 2\*si+2, mid+1, se, qs, qe));

}

}

/\* The functions returns the

min element in the range

from a and b \*/

int RMQ(int \*st , int n, int a, int b)

{

if(a > b)

swap(a,b);

if(a > n-1 || b < 0)

return -1;

return RMQSub(st, 0, 0, n-1,a, b);

}

int main()

{

int N;

cin>>N;

int A[N];

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

cin>>A[i];

int Q;

cin>>Q;

int \*segTree = constructST(A,N);

}

# LCA Binary Lifting

#include <iostream>

#include <vector>

#include <cmath>

#include <queue>

using namespace std;

const int maxN = 100, maxL=6;

vector<int> adj[maxN+1];

int level[maxN+1], table[maxL+1][maxN+1], l, n;

/\*

\*

\*/

void createLCATable(){

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

fill(table[i], table[i]+n+1, -1);

int visited[n+1];

fill(visited, visited+n+1, 0);

queue<int> q;

q.push(1);

visited[1] = 1;

level[1] = 0;

int node;

while(!q.empty())

{

node = q.front();

q.pop();

for(int i:adj[node])

{

if(!visited[i])

{

visited[i] = 1;

q.push(i);

table[0][i] = node;

level[i] = level[node]+1;

}

}

}

int mid;

for(node = 1; node <= n;++node)

{

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

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

if(mid != -1)

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

}

}

}

int walk(int i, int k){

int log = int(floor(log2(k)));

for(int d; d<=log && i!=-1; ++d)

if(1<<d & k)

i = table[d][i];

return i;

}

int lca(int i, int j){

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

j = walk(j, level[j]-level[i]);

else if(level[j] < level[i])

i = walk(i, level[i]-level[j]);

if(i==j)

return i;

for(int d = l; d>=0; --d)

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

i = table[d][i];

j = table[d][j];

}

return table[0][i];

}

int main() {

cin>>n;

int i, j;

for(int k = 0; k < n-1; ++k){

cin>>i>>j;

adj[i].push\_back(j);

adj[j].push\_back(i);

}

l = int(floor(log2(n)));

createLCATable();

return 0;

}

# BFS

Cola

Meter primer element a cola

Mientras cola no vacia{

Sacar elemento

Checar condicion o lo que buscas

Meter adyacentes

}

# Leer linea con enteros in saber cuántos son

string input;

int aux;

getline(cin, input);

input+=" ";

size = input.length();

while(ini < size){

fin = input.find(" ", ini);

aux = stoi(input.substr(ini, fin - ini));

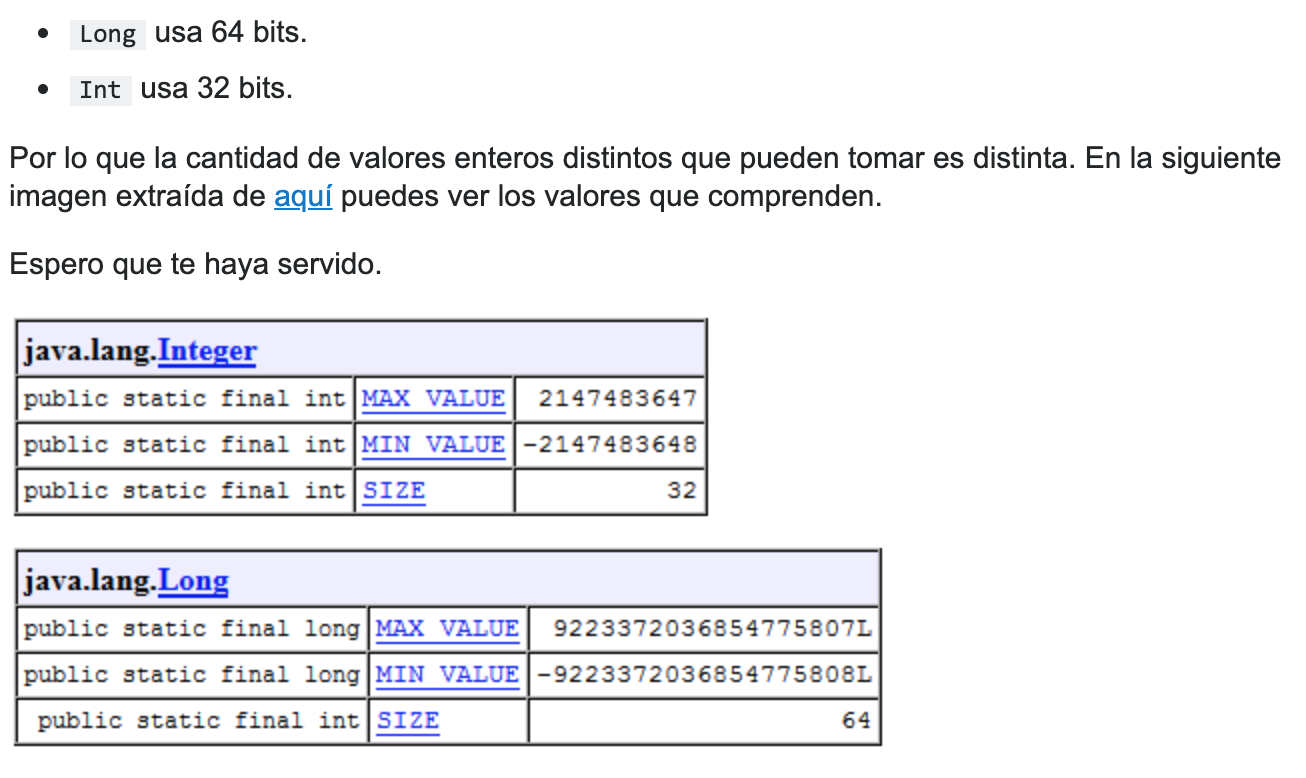
//lo que quieras hacer con él

ini = fin+1;

}

# Datos primitivos

Java y C++ (Variante 1)

109 < int < 1010

1018 < long < 1019

short 2 bytes

-215 a 215-1

-32767 a 32767

104 < short < 105

C++ variante 2

Int

-215 a 215-1

-32767 a 32767

104 < short < 105

Long

-232 a 232 – 1

-2147483648  a 2147483647

109 < long < 1010

Long long

1018 < long long < 1019

-9 223 372 036 854 775 808 a 9 223 372 036 854 775 807

-263 a 263 - 1

MATE

# Combinaciones Memo

Da las combinaciones de cierto renglón del triang de pascal

La entrada es el renglón, es decir, el total de elementos

long comb[a+2];

void combHasta(long a, long MOD){ //a es el renglon, el total de elementos

long aux1, aux2;

for(long i = comb[1] + 1; i<= a; ++i){

aux1 = 1;

aux2 = comb[1];

long j;

long mitad = i/2;

for(j = 1; j <= mitad; j++){

comb[j] = (aux1 + aux2) % MOD;

aux1 = aux2;

aux2 = comb[j+1];

}

for(j = mitad + 1; j <= i; j++)

comb[j] = comb[i-j];

}

}

# Combinaciones mod primo

**Sirve para calcular una específico**

long combMod(long n, long k){

if(n-k < k) k = n-k;

long comb = 1, inv; //inv = inverse of i

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

comb = (comb\*(n-i))%MOD;

inv = binPow(i+1,MOD-2); //inverse of k is k^p-2 because of Fermat's Little Theorem

comb = (comb\*inv)%MOD;

}

return comb;

}

# Fibonacci Exp Binaria

long\*\* identidad(){

long\*\* iden = new long\*[2];

iden[0] = new long[2];

iden[1] = new long[2];

iden[0][0] = 1;

iden[0][1] = 0;

iden[1][0] = 0;

iden[1][1] = 1;

return iden;

}

long\*\* multiplica(long\*\* a, long\*\* b){

long\*\* resp = new long\*[2];

resp[0] = new long[2];

resp[1] = new long[2];

resp[0][0]= ((a[0][0]\*b[0][0])%MOD + (a[0][1]\*b[1][0])%MOD)%MOD;

resp[0][1]= ((a[0][0]\*b[0][1])%MOD + (a[0][1]\*b[1][1])%MOD)%MOD;

resp[1][0]= ((a[1][0]\*b[0][0])%MOD + (a[1][1]\*b[1][0])%MOD)%MOD;

resp[1][1]= ((a[1][0]\*b[0][1])%MOD + (a[1][1]\*b[1][1])%MOD)%MOD;

return resp;

}

long\*\* binPow(long\*\* a, long b){

if (b == 0)

return identidad();

long\*\* res= binPow(a, b / 2);

if (b % 2)

return multiplica(multiplica(res,res), a);

else

return multiplica(res,res);

}

int fibo(long n) {

long\*\* fibo = new long\*[2];

fibo[0] = new long[2];

fibo[1] = new long[2];

fibo[0][0] = 0;

fibo[0][1] = 1;

fibo[1][0] = 1;

fibo[1][1] = 1;

if(n == 0)

return 0;

else{

long\*\* resp = binPow(fibo, n);

return resp[0][1];

}

}

# Algoritmo de Pollard Rho

**(Para factorizar números)**

#include<bits/stdc++.h>

#define abs\_val(a) (((a)>0)?(a):-(a))

typedef long long ll;

ll mulmod(ll a, ll b, ll c) { // returns (a \* b) % c, and minimize overflow

ll x = 0, y = a % c;

while (b > 0) {

if (b % 2 == 1) x = (x + y) % c;

y = (y \* 2) % c;

b /= 2;

}

return x % c;

}

ll gcd(ll a,ll b){ return !b ? a : gcd(b, a % b); } // standard gcd

ll pollard\_rho(ll n){

int i = 0, k = 2;

ll x = 3, y = 3; // random seed = 3, other values possible

while (1) {

i++;

x = (mulmod(x, x, n) + n - 1) % n;

ll d = gcd(abs\_val(y - x), n);

if (d != 1 && d != n) return d;

if (i == k) y = x, k \*= 2;

}

}

int main() {

ll n = ;

ll ans = pollard\_rho(n);

if (ans > n / ans) ans = n / ans;

printf("%lld %lld\n", ans, n / ans); // should be: 1112041493 1855607779

} // return 0;

PROBLEMAS IMPORTANTES

# Destruyendo Edificios

#include <iostream>

#include <stack>

#define etop extiende.top()

using namespace std;

int N;

long areaMax = 0;

stack< pair<int, int> > extiende;

void actualiza(long area){

if(area > areaMax)

areaMax = area;

}

void agrega(int altura){

long long area, comoDer = 0;

int comoIzq = 0;

while(!extiende.empty() && altura <= etop.first){

comoIzq += etop.second + 1;

area = (comoDer + etop.second + 1) \* etop.first;

actualiza(area);

comoDer += etop.second + 1;

extiende.pop();

}

extiende.push(make\_pair(altura, comoIzq));

}

int main() {

ios\_base::sync\_with\_stdio(0);

cin.tie(0);

int altura;

cin>>N;

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

cin>>altura;

agrega(altura);

}

agrega(0); //para contar el area abarcada por los ultimos edificios

cout<<areaMax<<endl;

return 0;

}

# Loca People

#include <bits/stdc++.h>

#define ptop pila.top()

using namespace std;

const int max\_n = 1002, max\_m = 1002;

int h[max\_n][max\_m], maxA[max\_n][max\_m], maxABase[max\_n][max\_m];

int n, m;

void leeAlturas(){

int aux;

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

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

cin>>aux;

h[i][j] = aux? (h[i-1][j] + 1) : 0;

maxA[i][j] = -1;

}

}

void maxAreaBase(int j){

int extiende[j+2], como, ancho, i, h\_;

fill(extiende, extiende+j+2, 0);

stack< pair<int, int> > pila;

for(i = 1; i <= m+1; ++i){

h\_ = h[j][i];

como = 0;

while(!pila.empty() && h\_ <= pila.top().first){

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

if(como > extiende[pila.top().first])

extiende[pila.top().first] = como;

pila.pop();

}

pila.push({h\_, como});

}

for(i = j; i >= 0; --i)

extiende[i] = max(extiende[i], extiende[i+1]);

for(i = j; i >= 0; --i)

extiende[i]\*=i;

for(i = 1; i<=j; ++i){

extiende[i] = max(extiende[i], extiende[i-1]);

maxABase[j-i+1][j] = extiende[i];

}

}

int maxArea(int i, int j){

if(i > j) return 0;

if(maxA[i][j] == -1)

maxA[i][j] = max(maxArea(i,j-1), maxABase[i][j]);

return maxA[i][j];

}

int main() {

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

cin>>n>>m;

leeAlturas();

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

maxAreaBase(j);

int p, a, b;

cin>>p;

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

cin>>a>>b;

cout<<maxArea(a, b)<<"\n";

}

return 0;

}

# Nieves Corriendo

#include <iostream>

#include <algorithm>

#include <climits>

using namespace std;

/\*

// para simular las dos carreras, considero dos caminos que inician ambos en la

// casilla (1,1) y en cada paso se tienen cuatro opciones, abajo-abajo,

// derecha-derecha, abajo-derecha, derecha-abajo

// la dp consiste en un bottom-up que encuentra el maximo puntaje entre las dos

// casillas actuales y el final

// cada estado se define en base a la diagonal en la que se encuentran los caminos

// y el numero de la casilla relativo a la diagonal.

// i.e. la casilla dos de la diagonal dos realmente es la casilla (1, 2)

// y la casilla uno de la diagonal dos es (2,1)

\*/

int main() {

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

int n;

cin>>n;

int mat[n+1][n+1];

int i, j;

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

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

cin>>mat[i][j];

/\*

// en pAnt vamos a guardar los datos de la diagonal que antes hicimos

// e iremos guardando los datos la actual en \*pAct

//

// d es el numero de la diagonal

// size el tamanio de la diagonal

// sumaAportada la suma que aportan las casillas actuales

// puntaje primero guarda el puntaje maximo de entre los caminos posibles a

// seguir y luego le sumamos sumaAportada para ser el puntaje maximo desde

// las casillas actuales hasta el final

// i y j pasan a representar la casilla en la diagonal del primer y segundo

// camino respectivamente

// spg i<=j

\*/

int ant[n+1][n+1], act[n+1][n+1];

int (\*pAnt)[n+1][n+1] = &ant;

int (\*pAct)[n+1][n+1] = &act;

int puntaje, d, size, sumaAportada;

ant[1][1] = mat[n][n];

for(int d = 2\*n - 2; d >= 1; --d){

size = d <= n? d: 2\*n - d;

for(i = 1; i <= size; ++i)

for(j = i; j <= size; ++j){

puntaje = INT\_MIN;

/\*

// las cosas son diferentes dependiendo de que lado de la diagonal

// principal estes

// si estas en la diagonal d y en la casilla x

// si d < n

// abajo: diagonal d+1 casilla x

// derecha: diagonal d+1 casilla x+1

// pos relativas al eje x y al eje y: (size-(x-1), x)

// si d >= n

// abajo: diagonal d+1 casilla x-1

// derecha: diagonal d+1 casilla x

// pos relativas al eje x y al eje y: (size-(x-1), x):

// (n - (x-1) , (n-size)+x)

\*/

if(d<n){

//der-abajo; abajo abajo; der der; abajo der

if(i!=j) //si son iguales aqui se romperia i<=j

puntaje = (\*pAnt)[i+1][j];

puntaje = max(puntaje, (\*pAnt)[i][j]);

puntaje = max(puntaje, (\*pAnt)[i+1][j+1]);

puntaje = max(puntaje, (\*pAnt)[i][j+1]);

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

}

else{

/\*

//der-abajo; abajo abajo; der der; abajo der

// si i > 1 si tiene abajo

// si j != size tiene derecha

\*/

if(i!=j)

puntaje = (\*pAnt)[i][j-1];

if(i > 1)

puntaje = max(puntaje, (\*pAnt)[i-1][j-1]);

if(size != j)

puntaje = max(puntaje, (\*pAnt)[i][j]);

if( i > 1 && size != j)

puntaje = max(puntaje, (\*pAnt)[i-1][j]);

sumaAportada = mat[n-i+1][n-size+i] + mat[n-j+1][n-size+j];

}

//si i == j solo sumas una vez la casilla

if(i==j)

sumaAportada/=2;

puntaje += sumaAportada;

(\*pAct)[i][j] = puntaje;

}

// \*pAct pasa a ser la anterior, y \*pAnt la usamos para la nueva

int (\*t)[n+1][n+1] = pAct;

pAct = pAnt;

pAnt = t;

}

cout<<(\*pAnt)[1][1]<<"\n";

return 0;

}

# Lectura rápida Java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Scanner;

import java.util.StringTokenizer;

public class Main{

    static class FastReader {

        BufferedReader br;

        StringTokenizer st;

        public FastReader(){

            br = new BufferedReader(new

                     InputStreamReader(System.in));

        }

        String next(){

            while (st == null || !st.hasMoreElements()){

                try  {

                    st = new StringTokenizer(br.readLine());

                }

catch (IOException  e) {

                    e.printStackTrace();

                }

            }

            return st.nextToken();

        }

        int nextInt()

            return Integer.parseInt(next());

        }

        long nextLong(){

            return Long.parseLong(next());

        }

        double nextDouble(){

            return Double.parseDouble(next());

        }

        String nextLine(){

            String str = "";

            try{

                str = br.readLine();

            }

            catch (IOException e) {

                e.printStackTrace();

            }

            return str;

        }

    }

    public static void main(String[] args) {

        FastReader s=new FastReader();

        int n = s.nextInt();

        int k = s.nextInt();

        int count = 0;

        while (n-- > 0)

            int x = s.nextInt();

            if (x%k == 0)

               count++;

        }

        System.out.println(count);

    }

}

# Fotos

#include <iostream>

#include <math.h>

#include <stdlib.h>

using namespace std;

int main(){

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

int r, c, a = 0, k;

cin>>r>>c>>k;

char x;

int i, j;

pair<int,int> arre[21];

for(i = 1; i <= r; ++i)

for(j = 1; j <= c; ++j){

cin>>x;

if(x == '\*'){

arre[a].first = i;

arre[a].second = j;

++a;

}

}

long mat[r\*c + 1];

mat[0] = 0;

for(i = 1; i <= r; ++i){

if(i & 1)

for(j = 1; j <= c; ++j){

mat[(i-1)\*c+j] = mat[(i-1)\*c+j-1];

mat[(i-1)\*c+j] += a\*1000;

for(int s = 0; s < a; ++s)

mat[(i-1)\*c+j]-= (abs(arre[s].first - i) + abs(arre[s].second - j));

}

else

for(j = c; j >= 1; --j){

mat[i\*c-j+1] = mat[i\*c-j];

mat[i\*c-j+1] += a\*1000;

for(int s = 0; s < a; ++s)

mat[i\*c-j+1]-= (abs(arre[s].first - i) + abs(arre[s].second - j));

}

}

cout<<'\n';

int aux,fijo, ini2, fin2, mitad2;

long ini1 = 0, fin1 = mat[r\*c], mitad1;

while(ini1 != fin1){

mitad1 = (ini1+fin1)/2;

aux = k;

ini2 = 1;

while(aux>0 && ini2 <= r\*c){

fijo = ini2-1;

fin2 = r\*c;

while(ini2 != fin2){

mitad2 = (ini2+fin2)/2;

if(mat[mitad2]-mat[fijo] > mitad1)

fin2 = mitad2;

else

ini2 = mitad2 + 1;

}

--aux;

}

if(ini2 == r\*c && (mat[ini2]-mat[fijo]) <= mitad1)

fin1 = mitad1;

else

ini1 = mitad1 + 1;

}

cout<<ini1<<'\n';

return 0;

}

# Keep it Healthy

#include <iostream>

#include <stack>

#define pt pila.top()

using namespace std;

int main() {

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

//n(filas) y m(columnas)

int max = 0, n, m;

cin>>n>>m;

int i, j, aux;

int h[n+1][m+2];

fill(h[0], h[0]+m+2, 0);

//el cero que va a sacar todas las alturas

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

h[i][m+1] = 0;

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

for(j = 1; j <= m; ++j){

cin>>aux;

h[i][j] = aux==0? 1+h[i-1][j]:0;

}

stack<pair<int, int> >pila;

int como, h\_, perimetro;

for(i = n; 2\*(i+m) > max && i > 0; --i)

for(j = 1; j <= m+1; ++j){

h\_ = h[i][j];

como = 0;

while(!pila.empty() && h\_ <= pt.first){

como += pt.second + 1;

//como es lo que se extiende la altura del tope

//checa si tienes que actualizar el maximo perimetro

if( pt.first>0 && (perimetro=2\*(pt.first + como)) > max)

max = perimetro;

pila.pop();

}

pila.push({h\_, como});

}

cout<<max<<"\n";

return 0;

}

# Propiedades de FibonacciMacintosh HD:private:var:folders:jv:39rqvvhn4451yhcn6t33xb640000gq:T:TemporaryItems:Captura de pantalla 2019-10-05 a las 16.27.45.png

