///ASIGNACIÓN DE TAREAS CON PESO

Pseudo

1) First sort jobs according to finish time.

2) Now apply following recursive process.

// Here arr[] is array of n jobs

findMaximumProfit(arr[], n) {

a) if (n == 1) return arr[0];

b) Return the maximum of following two profits.

(i)Maximum profit by excluding current job, i.e.,

findMaximumProfit(arr, n - 1)

(ii) Maximum profit by including the current job

}

The algorithm is :

Sort the jobs by non - decreasing finish times.

For each i from 1 to n, determine the maximum value of the schedule from the subsequence of jobs[0..i].Do this by comparing the inclusion of job[i] to the schedule to the exclusion of job[i] to the schedule, and then taking the max.

#include <iostream>

#include <algorithm>

using namespace std;

// A job has start time, finish time and profit.

struct Job{

int start, finish, profit;

};

bool myfunction(Job s1, Job s2){

return (s1.finish < s2.finish);

}

// A Binary Search based function to find the latest job

// (before current job) that doesn't conflict with current

// job. "index" is index of the current job. This function

// returns -1 if all jobs before index conflict with it.

// The array jobs[] is sorted in increasing order of finish

// time.

int binarySearch(Job jobs[], int index) {

// Initialize 'lo' and 'hi' for Binary Search

int lo = 0, hi = index - 1;

// Perform binary Search iteratively

while (lo <= hi) {

int mid = (lo + hi) / 2;

if (jobs[mid].finish <= jobs[index].start){

if (jobs[mid + 1].finish <= jobs[index].start)

lo = mid + 1;

else

return mid;

}

else

hi = mid - 1;

}

return -1;

}

// The main function that returns the maximum possible

// profit from given array of jobs

int findMaxProfit(Job arr[], int n)

{

// Sort jobs according to finish time

sort(arr, arr + n, myfunction);

// Create an array to store solutions of subproblems. table[i]

// stores the profit for jobs till arr[i] (including arr[i])

int \*table = new int[n];

table[0] = arr[0].profit;

// Fill entries in table[] using recursive property

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

{

// Find profit including the current job

int inclProf = arr[i].profit;

int l = binarySearch(arr, i);

if (l != -1)

inclProf += table[l];

// Store maximum of including and excluding

table[i] = max(inclProf, table[i - 1]);

}

// Store result and free dynamic memory allocated for table[]

int result = table[n - 1];

delete[] table;

return result;

}

// Driver program

int main()

{

Job arr[] = { { 3, 10, 20 },{ 1, 2, 50 },{ 6, 19, 100 },{ 2, 100, 200 } };

int n = sizeof(arr) / sizeof(arr[0]);

cout << "Optimal profit is " << findMaxProfit(arr, n);

return 0;

}

///ASIGNACION DE TAREAS SIN PESO

#include <bits/stdc++.h>

using namespace std;

// A job has start time, finish time and profit.

struct Activitiy{

int start, finish;

};

bool activityCompare(Activitiy s1, Activitiy s2)

return (s1.finish < s2.finish);

// Returns count of maximum set of activities that can be done by a single person, one at a time.

void printMaxActivities(Activitiy arr[], int n)

{

// Sort jobs according to finish time

sort(arr, arr + n, activityCompare);

cout << "Following activities are selected \n";

// The first activity always gets selected

int i = 0;

cout << "(" << arr[i].start << ", " << arr[i].finish << "), ";

// Consider rest of the activities

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

{

// If this activity has start time greater than or

// equal to the finish time of previously selected

// activity, then select it

if (arr[j].start >= arr[i].finish)

{

cout << "(" << arr[j].start << ", "

<< arr[j].finish << "), ";

i = j;

}

}

}

// Driver program

int main()

{

Activitiy arr[] = { { 5, 9 },{ 1, 2 },{ 3, 4 },{ 0, 6 },

{ 5, 7 },{ 8, 9 } };

int n = sizeof(arr) / sizeof(arr[0]);

printMaxActivities(arr, n);

return 0;

}

///GEOMETRÍA

#include <cstdio>

#include <cmath>

#include <vector>

using namespace std;

struct point {

long double x, y;

point() {};

point(double a, double b){

x = a;

y = b;

}

};

struct vec{

///Parece lo mismo que un punto, pero es un vector

double x, y;

vec() {};

vec(point a, point b){

x = b.x - a.x;

y = b.y - a.y;

}

};

struct line {

point p1, p2;

long double a, b, c;

line() {};

line(point ini, point fin){

///De dos puntos lo llevo a la forma ax+by+c = 0

a = ini.y - fin.y;

b = fin.x - ini.x;

c = (ini.x\*fin.y) - (ini.y\*fin.x);

///Guardo esos dos puntos en caso de requerir un "segmento"

p1.x = ini.x;

p1.y = ini.y;

p2.x = fin.x;

p2.y = fin.y;

}

};

bool lineVsline(line l, line s){

double det = l.a\*s.b - l.a\*s.b;

return det != 0;

}

bool lineVSsegment(line l, line s){

int v1, v2;

v1 = l.a\*s.p1.x + l.b\*s.p1.y + l.c;

v2 = l.a\*s.p2.x + l.b\*s.p2.y + l.c;

if (v1\*v2 <= 0){

return true;

}

return false;

}

bool segmentvssegment(line a, line b){

///Los primeros dos ifs "grandes" se pueden cambiar por "si somos colineales (El area formada por los 4 puntos es 0)

///y los segmentos no se traslapan(como en el if de adentro) return false;

if (a.p1.x == a.p2.x && a.p2.x == b.p1.x && b.p1.x == b.p2.x)

{

if (max(a.p1.y, a.p2.y) < min(b.p1.y, b.p2.y) || min(a.p1.y, a.p2.y) > max(b.p1.y, b.p2.y))

return false;

}

if (a.p1.y == a.p2.y && a.p2.y == b.p1.y && b.p1.y == b.p2.y)

{

if (max(a.p1.x, a.p2.x) < min(b.p1.x, b.p2.x) || min(a.p1.x, a.p2.x) > max(b.p1.x, b.p2.x))

return false;

}

int v1, v2;

v1 = a.a\*b.p1.x + a.b\*b.p1.y + a.c;

v2 = a.a\*b.p2.x + a.b\*b.p2.y + a.c;

if (v1\*v2 <= 0)

{

v1 = b.a\*a.p1.x + b.b\*a.p1.y + b.c;

v2 = b.a\*a.p2.x + b.b\*a.p2.y + b.c;

if (v1\*v2 <= 0)

return true;

}

return false;

}

point intersection(line line1, line line2) {

///La llamo sí y solo sí se intersectan

long double det, detx, dety;

point ans;

det = line1.a\*line2.b - line2.a\*line1.b;

detx = line1.c\*line2.b - line2.c\*line1.b;

dety = line1.a\*line2.c - line2.a\*line1.c;

ans = point(-1 \* detx / det, -1 \* dety / det);

return ans;

}

double angle(point a, point b, point p){

///Regresa el ángulo apb ( vertice en "p")

vec pa = vec(p, a), pb = vec(p, b);

double dot = pa.x\*pb.x + pa.y\*pb.y;

double norm\_pa = pa.x\*pa.x + pa.y\*pa.y;

double norm\_pb = pb.x\*pb.x + pb.y\*pb.y;

return acos(dot / sqrt(norm\_pa\*norm\_pb));

}

bool leftturn(point a, point b, point c){

///Me regresa si el giro de los puntos fue hacia la izquierda o hacia la derecha

double ans = a.x\*b.y;

ans += b.x\*c.y;

ans += c.x\*a.y;

ans -= a.y\*b.x;

ans -= b.y\*c.x;

ans -= c.y\*a.x;

return ans > 0;

}

long double area(vector<point> pp){

///Si no funciona es muy probable que ea por los arreglos

pp.push\_back(pp[0]);

long double a = 0;

for (int i = 0; i<pp.size(); i++){

a += pp[i].x\*pp[i + 1].y;

a -= pp[i].y\*pp[i + 1].x;

}

return a / 2.0;

}

bool pInPolygon(point p, vector<point> pp){

pp.push\_back(pp[0]);

double suma = 0, pi = acos(-1);

bool a;

for (int i = 0; i<pp.size() - 1; i++)/\*\*El "-1" es una conjetura,se usa porque al meter el primero el tamaño aumenta, si no funciona usar arrays\*\*/

{

a = leftturn(pp[i], pp[i + 1], p);

if (a == false)

suma -= angle(pp[i], pp[i + 1], p);

else

suma += angle(pp[i], pp[i + 1], p);

}

if (abs(abs(suma) - (2.0)\*pi) < 0.0000001) return true;

else return false;

}

bool isConvex(vector<point> pp){

pp.push\_back(pp[0]);

bool dir = leftturn(pp[0], pp[1], pp[2]), actual;

for (int i = 0; i<pp.size() - 1; i++)

{

actual = leftturn(pp[i], pp[i + 1], pp[i + 2]);

if (actual != dir) return false;

}

return true;

}

///CRIBA DE ERATOSTENES

#include <cstdio>

#include <cmath>

using namespace std;

bool marcados[1000000 + 5];

int primos[1000000 + 5], cant, n;

int main()

{

scanf("%d", &n);

for (int i = 2; i\*i <= n; i++) {

if (marcados[i] == false) {

for (int j = i\*i; j <= n; j += i)

marcados[j] = true;

}

}

for (int i = 2; i <= n; i++)

if (marcados[i] == false) {

primos[cant++] = i;

}

printf("%d primos entre 1 y %d\n", cant, n);

for (int i = 0; i<cant; i++) printf("%d\n", primos[i]);

}

///EXPONENCIACION BINARIA Y nCk

#include <cstdio>

using namespace std;

long long factoriales[10000000], inversos[100000], p = 1000000000 + 7;

long long exp\_bin(long long a, long long n) {

///a^n Exponenciación binaria

long long k = a;

long long r = 1;

while (n>0){

if (n % 2 == 1)

r = (k\*r) % p;

k = (k\*k) % p;

n = n / 2;

}

return r;

}

long long nCk(long long n, long long k){

///P tiene que ser primo, requiere inicializar arreglo de factoriales e inversos, si k> p se va a la verga, calculamos el inverso multiplicativo mod p en lugar de dividir, surge de que a^p-1 es congruente a 1 mod p si p es primo, entonces a ^(p-2) es congruente con a^-1 mod p, a^-1 es el inverso multiplicativo de a, entonces elevamos a^(p-2); pequeño teorema de Fermat.

long long ans;

ans = (factoriales[n] \* inversos[n - k]) % p;

ans = (ans \* inversos[k]) % p;

return ans;

}

int main()

{

///inveros[i] guarda el inverso multiplicativo de i! mod p

factoriales[0] = 1;

inversos[0] = exp\_bin(factoriales[0], p - 2) % p;

for (long long i = 1; i <= 1000; i++){

factoriales[i] = (i\*factoriales[i - 1]) % p;

inversos[i] = exp\_bin(factoriales[i], p - 2) % p;

}

printf("%lld", nCk(1, 1));

return 0;

}

///LONGEST INCREASING SUBSEQUENCE

#include <bits/stdc++.h>

using namespace std;

int main() {

int n;

scanf("%d", &n);

vector<int> nums;

nums.resize(n);

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

scanf("%d", &nums[i]);

printf("%d ", nums[i]); }

//obtener el tamaño de la LIS

multiset<int> ms;

multiset<int>::iterator it;

vector<int> lis;

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

int aux = nums[i];

ms.insert(aux);

it = ms.find(aux);

it++;

if (it != ms.end())

ms.erase(it);

else

lis.push\_back(aux);

} printf("\n");

for (int i = 0; i<lis.size(); i++) {

printf("%d ", lis[i]);

} printf("\n");

printf("\nLis tam: %d\n", ms.size());

}

int lis() {

/\*arr es el arreglo donde están los elementos, aux[i] guarda el último índice

con que se pudo hacer una subsecuencia de longitud i, n es el tamaño del conjunto\*/

int ini, fin, med, i, ul = 1, newl, res = 0;

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

ini = 1;

fin = res;

while (ini <= fin) {

med = (ini + fin) / 2;

if (arr[aux[med]]<arr[i])

ini = med + 1;

else

fin = med - 1;

}

newl = ini;

aux[newl] = i;

if (newl>res)

res = newl;

}

return res;

}

///LONGEST SUM CONTIGUOS SUBARRAY

int maxSubArraySum(int a[], int size){

int max\_so\_far = INT\_MIN, max\_ending\_here = 0,

start = 0, end = 0, s = 0;

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

{

max\_ending\_here += a[i];

if (max\_so\_far < max\_ending\_here){

max\_so\_far = max\_ending\_here;

start = s;

end = i; }

if (max\_ending\_here < 0){

max\_ending\_here = 0;

s = i + 1;

}

}

cout << "Maximum contiguous sum is "

<< max\_so\_far << endl;

cout << "Starting index " << start

<< endl << "Ending index " << end << endl;

}

///LONGEST COMMON SUBSTRING

int LCSubStr(char \*X, char \*Y, int m, int n){

// Create a table to store lengths of longest common suffixes of

// substrings. Notethat LCSuff[i][j] contains length of longest

// common suffix of X[0..i-1] and Y[0..j-1]. The first row and

// first column entries have no logical meaning, they are used only

// for simplicity of program

int LCSuff[m + 1][n + 1];

int result = 0; // To store length of the longest common substring

/\* Following steps build LCSuff[m+1][n+1] in bottom up fashion. \*/

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

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

if (i == 0 || j == 0)

LCSuff[i][j] = 0;

else if (X[i - 1] == Y[j - 1]){

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

result = max(result, LCSuff[i][j]);

}

else LCSuff[i][j] = 0;

}

}

return result;

}

///LONGEST COMMON SUBSEQUENCE

#include<bits/stdc++.h>

/\* Returns length of LCS for X[0..m-1], Y[0..n-1] \*/

int lcs(char \*X, char \*Y, int m, int n){

int L[m + 1][n + 1], i, j;

/\* Following steps build L[m+1][n+1] in bottom up fashion. Note

that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] \*/

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

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

if (i == 0 || j == 0)

L[i][j] = 0;

else if (X[i - 1] == Y[j - 1]) L[i][j] = L[i - 1][j - 1] + 1;

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

}

}

/\* L[m][n] contains length of LCS for X[0..n-1] and Y[0..m-1] \*/

return L[m][n];

}

///KNAPSACK 0 1 SIN DUPLICADOS

#include<stdio.h>

#include <algorithm>

// Returns the maximum value that can be put in a knapsack of capacity W

int knapSack(int W, int wt[], int val[], int n)

{

// Base Case

if (n == 0 || W == 0)

return 0;

// If weight of the nth item is more than Knapsack capacity W, then

// this item cannot be included in the optimal solution

if (wt[n - 1] > W)

return knapSack(W, wt, val, n - 1);

// Return the maximum of two cases:

// (1) nth item included

// (2) not included

else return max(val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1),

knapSack(W, wt, val, n - 1)

);

}

// Driver program to test above function

int main()

{

int val[] = { 60, 100, 120 };

int wt[] = { 10, 20, 30 };

int W = 50;

int n = sizeof(val) / sizeof(val[0]);

printf("%d", knapSack(W, wt, val, n));

return 0;

}

///KNAPSACK 0 1 CON DUPLICADOS

#include<bits/stdc++.h>

using namespace std;

// Returns the maximum value with knapsack of W capacity

int unboundedKnapsack(int W, int n, int val[], int wt[]) {

// dp[i] is going to store maximum value

// with knapsack capacity i.

int dp[W + 1];

memset(dp, 0, sizeof dp);

int ans = 0;

// Fill dp[] using above recursive formula

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

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

if (wt[j] <= i)

dp[i] = max(dp[i], dp[i - wt[j]] + val[j]);

return dp[W];

}

// Driver program

int main(){

int W = 100;

int val[] = { 10, 30, 20 };

int wt[] = { 5, 10, 15 };

int n = sizeof(val) / sizeof(val[0]);

cout << unboundedKnapsack(W, n, val, wt);

return 0;

}

///FORMAS DIFERENTES DE SUMAR N CON UN CONJUNTO DADO

#include <cstdio>

using namespace std;

int coins[] = { 1, 5, 10, 25, 50 }, solutions[10000], n;

void dp(){

for (int i = 0; i<10000; i++) solutions[i] = 1;

for (int i = 1; i<5; i++)

for (int j = coins[i]; j <= 10000; j++)

solutions[j] += solutions[j - coins[i]];

}

///MINIMA CANTIDAD DE MONEDAS PARA SUMAR N

#include <cstdio>

#include <algorithm>

using namespace std;

int total[100000 + 5], coins[10] = { 1, 3, 5, 6 }, n, tc, aux;

int main(){

scanf("%d", &tc);

for (int i = 1; i <= 100005; i++)

total[i] = i;

for (int i = 0; i <= 3; i++){

for (int j = 1; j <= 100005; j++){

if (j >= coins[i]){

aux = total[j - coins[i]];

total[j] = min(aux + 1, total[j]);

}

}

}

}

///CONTADOR DE APARICIONES string a en string b (como subsecuencia)

int findSubsequenceCount(string S, string T)

{

int m = T.length(), n = S.length();

// T can't appear as a subsequence in S

if (m > n)

return 0;

// mat[i][j] stores the count of occurrences of T(1..i) in S(1..j).

int mat[m + 10][n + 10];

// Initializing first column with all 0s. An empty

// string can't have another string as suhsequence

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

mat[i][0] = 0;

// Initializing first row with all 1s. An empty

// string is subsequence of all.

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

mat[0][j] = 1;

// Fill mat[][] in bottom up manner

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

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

// If last characters don't match, then value

// is same as the value without last character

// in S.

if (T[i - 1] != S[j - 1])

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

// Else value is obtained considering two cases.

// a) All substrings without last character in S

// b) All substrings without last characters in

// both.

else

mat[i][j] = (mat[i][j - 1] + mat[i - 1][j - 1]) % 1000000007;

}

}

// uncomment this to print matrix mat

// for (int i = 1; i <= m; i++, cout << endl)

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

// cout << mat[i][j] << " ";

return mat[m][n];

}

///DIJKSTRA

typedef pair<int, int> ii; // <peso,nodo>

int main() {

int s = 0; // NODO ORIGEN!

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

pq.push(ii(0, s));

while (!pq.empty()) {

ii front = pq.top(); pq.pop();

int d = front.first; // peso

int u = front.second; // nodo

if (d > dist[u]) continue;

for (int j = 0; j < AdjList[u].size(); j++) {

ii v = AdjList[u][j]; // todas las salidas del nodo u

if (dist[u] + v.second < dist[v.first]) {

dist[v.first] = dist[u] + v.second;

pq.push(ii(dist[v.first], v.first));

}

}

}

}

///KRUSKALL

#include <cstdio>

#include <utility>

#include <cmath>

#include <algorithm>

using namespace std;

int tc, M, N, papas[505], niv[505];

int arr[505], X, Y, X1, Y1, X2, Y2, dist;

pair <int, pair<int, int> > puntos[505];

pair <int, pair<int, int> > pesos[250000 + 5];

int papa(int a){

if (a == papas[a])

return a;

else

papa(papas[a]);

}

void une(int a, int b){

int pa = papa(a);

int pb = papa(b);

if (niv[pa] > niv[pb])

papas[pb] = pa;

else if (niv[pa] < niv[pb])

papas[pa] = pb;

else{

papas[pb] = pa;

niv[pa]++;

}

}

int main(){

scanf("%d", &tc);

while (tc--){

for (int i = 0; i <= 501; i++){

niv[i] = 1;

papas[i] = i;

}

scanf("%d%d", &M, &N);

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

puntos[i].first = i;

scanf("%d%d", &puntos[i].second.first,

&puntos[i].second.second);

}

int c = 0;

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

X1 = puntos[i].second.first; Y1 = puntos[i].second.second;

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

if (puntos[i].first != puntos[j].first){

X2 = puntos[j].second.first;

Y2 = puntos[j].second.second;

X = (X2 - X1) \* (X2 - X1);

Y = (Y2 - Y1) \* (Y2 - Y1);

dist = X + Y;

pesos[c].first = dist;

pesos[c].second.first = puntos[i].first;

pesos[c].second.second = puntos[j].first;

c++;

}

}

}

sort(pesos, pesos + c);

int k = 0;

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

if (papa(pesos[i].second.first)!=papa(pesos[i].second.second)){

une(pesos[i].second.first, pesos[i].second.second);

arr[k] = pesos[i].first;

k++;

}

}

double aux = sqrt(arr[k - M]);

printf("%.2lf\n", aux);

}

}

///GRUNDY GERARDO

#include <bits/stdc++.h>

using namespace std;

int grundy[15][15];

vector<vector<bool> > asignados;

int mov[4][2] = { { -2,1 },{ -2,-1 },{ 1,-2 },{ -1,-2 } };

int Mex(set<int> s) {

int n = 0;

while (s.find(n) != s.end()) {

n++;

}

return n;

}

bool dentro(int x, int y) {

return (x >= 0 && x<15 && y >= 0 && y<15);

}

int grund(int x, int y) {

//ya esta calculado

if (asignados[x][y])

return grundy[x][y];

//no se ha calculado

set<int> v;

for (int i = 0; i<4; i++) { // revisar cada movimiento

int xx, yy;

xx = x + mov[i][0];

yy = y + mov[i][1];

if (dentro(xx, yy)) {

int aux = grund(xx, yy);

v.insert(aux);

}

}

int res = Mex(v);

grundy[x][y] = res;

asignados[x][y] = 1;

return res;

}

void printG() {

for (int i = 0; i<15; i++) {

for (int j = 0; j<15; j++) {

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

}

printf("\n");

}

}

int main() {

asignados.assign(15, vector<bool>(15, false));

//solo los primeros 4 cuadros son 0

asignados[0][0] = 1;

asignados[0][1] = 1;

asignados[1][0] = 1;

asignados[1][1] = 1;

int tc;

scanf("%d", &tc);

while (tc--) {

int m;

scanf("%d", &m);

int xxor = 0;

while (m--) {

int x, y;

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

int res = grund(x - 1, y - 1);

xxor ^= res;

}

if (xxor == 0)

printf("Second\n");

else

printf("First\n");

}

}

///GRUNDY CARLOS

#include <cstdio>

using namespace std;

int dir[5][5] = { { 1,-2 },{ -1,-2 },{ -2,-1 },{ -2,1 } }, tablero[20][20];

bool usados[20][20];

bool dentro(int i, int j){

if (i >= 1 && i <= 15 && j >= 1 && j <= 15) return true;

else return false; }

void calcular(int i, int j){

int mex[6] = { 0,0,0,0,0,0 };

usados[i][j] = true;

for (int k = 0; k <= 3; k++){

if (dentro(i + dir[k][0], j + dir[k][1])){

if (!usados[i + dir[k][0]][j + dir[k][1]]){

calcular(i + dir[k][0], j + dir[k][1]);

}

mex[tablero[i + dir[k][0]][j + dir[k][1]]] = 1;

}

}

for (int k = 0; k<5; k++){

if (mex[k] == 0){

tablero[i][j] = k;

break;

}

}

}

int main(){

tablero[1][1] = tablero[1][2] = tablero[2][1] = tablero[2][2] = 0;

usados[1][1] = usados[1][2] = usados[2][1] = usados[2][2] = true;

for (int i = 1; i <= 15; i++)

for (int j = 1; j <= 15; j++)

calcular(i, j);

int g, i, j, k, ans;

scanf("%d", &g);

while (g--){

ans = 0;

scanf("%d", &k);

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

scanf("%d%d", &i, &j);

ans ^= tablero[i][j];

}

if (ans != 0)

printf("First\n");

else printf("Second\n");

}

}

///PRIORITY QUEUE (HEAP SORT)

#include <cstdio>

#include <algorithm>

using namespace std;

int arr[10000], n, tam, num;

void insertar(int numero){

///el parámetro de esta función es el número que queremos insertar

tam++;

int vertice = tam;

arr[vertice] = numero;

int papa = vertice / 2;

while (vertice != 1 && arr[vertice] > arr[papa]){

swap(arr[papa], arr[vertice]);

vertice = papa;

papa = vertice / 2;}

}

void borrar(){

swap(arr[1], arr[tam]); ///Aquí mandamos al de la posición tam a la raíz, para volver a acomodar el montículo

tam--;

int vertice = 1;

int iz = vertice \* 2, der = vertice \* 2 + 1;

///Si la raiz está en 1 entonces los hijos de un nodo i serán i\*2 y (i\*2)+1; y su padre será i/2, si la raiz está en 0 entonces los hijos de un nodo i serán (i\*2)+1 y (i+1)\*2; y su padre será (i-1)/2

while ((iz <= tam && arr[iz]>arr[vertice]) ||

(der <= tam && arr[der]>arr[vertice])) {

///iz<=tam y der<=tam se utilizan para checar que esos nodos aún sean parte de nuestro montículo

if (der <= tam && arr[der] > arr[iz]){

swap(arr[der], arr[vertice]);

vertice = der; }

else{

swap(arr[iz], arr[vertice]);

vertice = iz; }

iz = vertice \* 2;

der = vertice \* 2 + 1;

}

}

///SEGMENT TREE PARA MAXIMO EN RANGO

#include <bits/stdc++.h>

using namespace std;

class segmentTree {

public:

//int arr[1000000+5], tree[5000000+5];

vector<int> arr, tree, lazy;

int N;

void build(int node, int ss, int se){

int mid;

if (ss == se)

scanf("%d", &tree[node]);

else{

mid = (ss + se) / 2;

build(2 \* node, ss, mid);

build(2 \* node + 1, mid + 1, se);

tree[node] = max(tree[2 \* node], tree[2 \* node + 1]);

}

}

void update(int node, int ss, int se, int index, int val){

int mid;

if (ss == se)

tree[node] = val;

else{

mid = (ss + se) / 2;

if (index <= mid)

update(node \* 2, ss, mid, index, val);

else

update(node \* 2 + 1, mid + 1, se, index, val);

tree[node] = (tree[node \* 2] + tree[node \* 2 + 1]);

}

}

void update\_range(int node, int ss, int se, int qs, int qe, int val){

int mid = (ss + se) / 2;

if (lazy[node] != 0){

tree[node] = lazy[node];

///for RANGESUM use = (se-ss+1)\*lazy[node]

if (ss != se){

lazy[node \* 2] = lazy[node];

lazy[node \* 2 + 1] = lazy[node];

}

lazy[node] = 0;

}

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

return;

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

tree[node] = val;///for RANGESUM use = (se-ss+1)\*val

if (ss != se){

lazy[node \* 2] = val;

lazy[node \* 2 + 1] = val;

}

}

else{

update\_range(node \* 2, ss, mid, qs, qe, val);

update\_range(node \* 2 + 1, mid + 1, se, qs, qe, val);

tree[node] = max(tree[node \* 2], tree[node \* 2 + 1]);

}

}

long long query(int node, int ss, int se, int qs, int qe){

if (lazy[node]){

tree[node] = lazy[node];///for RANGESUM use = (se-ss+1)\*lazy[node]

if (ss != se){

lazy[node \* 2] = lazy[node];

lazy[node \* 2 + 1] = lazy[node];

}

lazy[node] = 0;

}

int mid;

if (qe < ss || qs > se)

return -100000000000; /// for RANGESUM return 0, rangeProduct 1, RangeMin return max

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

return tree[node];

else{

mid = (ss + se) / 2;

return max(query(node \* 2, ss, mid, qs, qe),

query(node \* 2 + 1, mid + 1, se, qs, qe));

}

}

void build(int n) {

arr.resize(n + 5);

lazy.resize(5 \* n + 5);

tree.resize(5 \* n + 5);

build(1, 1, n);

N = n;

}///Falta sobrecarga para los updates

long long query(int qs, int qe) {

return query(1, 1, N, qs, qe);

}

};

int main()

{

int tc, N, K;

scanf("%d", &tc);

while (tc--){

segmentTree st;

scanf("%d%d", &N, &K);

st.build(N);

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

printf("%lld ", st.query(i, i + K - 1));

}

printf("\n");

}

return 0;

}

///UNION FIND DISJOINT SET

#include <bits/stdc++.h>

using namespace std;

class uf {

public:

//int papas[1000005], sizes[1000005], niv[1000005], cant\_conjuntos;

vector<int> papas, sizes, niv;

int cant\_conjuntos;

uf(int n){

papas.resize(1000005);sizes.resize(1000005);niv.resize(1000005);

cant\_conjuntos = n;

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

sizes[i] = 1;

papas[i] = i;

niv[i] = 0;

}

}

int papa(int a){

if (papas[a] != a)

papas[a] = papa(papas[a]);

return papas[a];

}

void une(int a, int b){

int pa = papa(a), pb = papa(b);

if (pa != pb){

if (niv[pa] < niv[pb])

papas[pa] = pb;

else if (niv[pa] > niv[pb])

papas[pb] = pa;

else{

papas[pa] = pb;

niv[pb]++;

}

int tam = sizes[pa] + sizes[pb];

sizes[pa] = sizes[pb] = tam;

cant\_conjuntos--;

}

}

bool mismo(int a, int b){

int pa = papa(a);

int pb = papa(b);

return pa == pb;

}

int tam(int a){

//regresa el taño del conjunto al que pertenece “a”

int pa = papa(a);

return sizes[pa];

}

int cant\_sets()

return cant\_conjuntos;

};