.

First to Penalty

Contents

1 Template

```cpp
#include "bits/stdc++.h"
//assert(x>0) si falla da RTE
using namespace std;
#define endl '\n'
#define DBG(x) cerr<<#x<< "=" << (x) << endl;
#define RAYA cerr<<"======================"<<endl;
#define RAYAS cerr<<"......................"<<endl;
//#define DBG(x) ;
//#define RAYA ;
//#define RAYAS ;

//--------------------SOLBEGIN---------------------------------
int main() {
  ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
  int tC;

  cin >> tC;
  while (tC--) {

  }

}
//--------------------EOSOLUTION---------------------------------
```

2 Data structures

2.1 struct

```cpp
#include "bits/stdc++.h"
using namespace std;
#define endl "\n"

struct unidad{
  int costo;
  int origen;
  int destino;
};

bool comp(const unidad a, const unidad b){
  if(a.costo == b.costo){
    if(a.origen == b.origen){
      return a.destino < b.destino;
    }
    return a.origen < b.origen;
  }
  return a.costo < b.costo;
}

int main()
{
  ios_base::sync_with_stdio(false);cout.tie(NULL);cin.tie(NULL);

  vector<unidad> arr;
  unidad aux;

  int n;
  cin>>n;

  while(n--){
    cin>>aux.origen>>aux.destino>>aux.costo;
    arr.push_back(aux);
  }

  sort(arr.begin(), arr.end(), comp);

  for(unidad p : arr){
    cout<<p.origen<<"␣"<<p.destino<<"␣"<<p.costo<<endl;
  }

  return 0;
}
```

2.2 segTree

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<long long int> elem;
class segmentTree{
  public:
  vector<long long int>sT;
    segmentTree(long long int size);
    long long int create(long long int left, long long int right, long
        long int indice);
    long long int query(long long int left,long long int right,long long
```

```
        int q_left,long long int q_right,long long int ind);
    long long int update(long long int left,long long int right,long
        long int val,long long int up_index,long long int indice);

};

long long int segmentTree::query(long long int left,long long int right,
    long long int q_left,long long int q_right,long long int indice){
  long long int neutro = 1000000001;
  if ((q_left>right) || (q_right<left)){
    return neutro;
  }

  if (left>=q_left && right<=q_right){
    return sT[indice];
  }

  long long int mid=(right+left)/2;
  long long int leftSon=query(left, mid, q_left, q_right, (2*indice+1));
  long long int rightSon=query(mid+1, right, q_left, q_right, (2*indice
      +2));

  return min(leftSon, rightSon);
  // return(leftSon+rightSon);
}

long long int segmentTree::update(long long int left,long long int right
    ,long long int val,long long int up_index, long long int indice){
  long long int mid=(right+left)/2;
  if(left==right){
    sT[indice]=val;
    return sT[indice];
  }

  if((left<=up_index) && (up_index<=mid)){
    sT[indice]=min(sT[(2*indice+2)] , update(left, mid, val, up_index,
        (2*indice+1)));
  }
  else{
    sT[indice]=min(sT[(2*indice+1)] , update(mid+1, right, val, up_index
        , (2*indice+2)));
  }
  //aqui era val, pero si lo dejabas asi siempre ibas a subir val, lo
      cual puede que te cause bugs
  return sT[indice];
}

long long int segmentTree::create(long long int left, long long int
    right, long long int indice ){
  long long int mid=(right+left)/2;
  if(left==right){
    sT[indice]=elem[left];
    return sT[indice];
  }

  long long int leftSon=create(left, mid, (2*indice+1));
  long long int rightSon=create(mid+1, right, (2*indice+2));

  sT[indice]=min(leftSon, rightSon);
  // sT[indice]=leftSon + rightSon;
  // return (leftSon + rightSon);
  return min(leftSon, rightSon);
}

segmentTree::segmentTree(long long int size) {
  long long int neutro = 1000000001;

  long long int pot=2;
  while(pot<size){
    pot = pot * 2;
  }
  while(elem.size()<pot){
    elem.push_back(neutro);
  }

  for(int i = 0; i<2*pot-1; i++) sT.push_back(neutro);

  create(0, pot-1, 0);

  // for(int x:sT){
  //   cout<<x<<endl;
  // }

  // int res = query(0,7,0,4,0);
  // cout<<"res: "<<res<<endl;
}
```

```

int main() {
  long long int n,aux,q;

  cin>>n>>q;
  long long int auxn=n;

  while(n){
    cin>>aux;
    elem.push_back(aux);
    n--;
  }

  segmentTree st(auxn);


  long long int l,r,v;
  while(q--){
    cin>>v>>l>>r;

    if(v==1){
      // r--;
      st.update(0,elem.size()-1,r,l-1,0);
    }else{
      // r--;
      cout<<st.query(0,elem.size()-1,l-1,r-1,0)<<endl;
    }

  }

  return 0;
}
```

3 Graphs

3.1 bfs

```
#include<vector>
#include<bitset>
#include<queue>
#include<stack>

using namespace std;
#define GS 400040

vector<int> graph[GS];
bitset <GS> vis;
//anchura O(V+E)
void dfs(int curr) {
  queue<int> fringe;
  fringe.push(curr);
  while (fringe.size()) {
    curr = fringe.front(); fringe.pop();
    if (!vis[curr]) {
      vis[curr] = 1;
      for (int h : graph[curr]) fringe.push(h);
    }
  }
}
```

3.2 dfs

```
#include<vector>
#include<bitset>
#include<queue>
#include<stack>

using namespace std;
#define GS 400040

vector<int> graph[GS];
bitset <GS> vis;
//anchura O(V+E)
void dfs(int curr) {
  stack<int> fringe;
  fringe.push(curr);
  while (fringe.size()){
    curr = fringe.top(); fringe.pop();
    if (!vis[curr]) {
      vis[curr] = 1;
      for (int h : graph[curr]) fringe.push(h);
    }
  }
}
```

4 Math

4.1 Coeficientes binomiales. (Combinaciones n en k)

Una combinación es una forma de elegir k elementos de un grupo de tamaño n, sin importar el orden en que los eliges.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k$$

$$\binom{n}{k} = \binom{n}{n-k}$$

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

$$k\binom{n}{k} = n\binom{n-1}{k-1}$$

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

$$\sum_{k=0}^{n} (-1)^k \binom{n}{k} = 0$$

$$\binom{n+m}{t} = \sum_{k=0}^{t} \binom{n}{k}\binom{m}{t-k}$$

$$\sum_{j=k}^{n} \binom{j}{k} = \binom{n+1}{k+1}$$

$n=0$				1			
$n=1$			1		1		
$n=2$		1		2		1	
$n=3$		1		3		3	1
$n=4$	1		4		6		4	1
$n=5$	1	5		10		10	5	1
$n=6$	1	6	15		20	15	6	1
	0	1	2	3	4	5	6

$n=0$	1						
$n=1$	1	1					
$n=2$	1	2	1				
$n=3$	1	3	3	1			
$n=4$	1	4	6	4	1		
$n=5$	1	5	10	10	5	1	
$n=6$	1	6	15	20	15	6	1
	0	1	2	3	4	5	6

4.2 Numeros catalanes

Los números de Catalan son una secuencia de números enteros que aparecen en varios problemas combinatorios y estructurales en matemáticas. Se utilizan para contar estructuras específicas que siguen reglas particulares, como el número de maneras de emparejar paréntesis correctamente, las maneras de dividir un polígono convexo en triángulos, o las maneras de construir árboles binarios completos.
Los números de Catalan también se aplican en la cuenta de caminos específicos, aunque no para cualquier tipo de camino. En particular, se usan para contar caminos restringidos en una cuadrícula.
Un ejemplo clásico es el conteo de caminos que van desde el punto $(0,0)$ hasta el punto (n,n) en una cuadrícula, avanzando solo hacia la derecha o hacia abajo, pero con la condición de que el camino nunca debe cruzar la diagonal principal (es decir, que en todo momento debe estar por encima o sobre la línea $y = x$. En este contexto, el número de caminos que cumplen esta restricción para una cuadrícula de tamaño n es el n-ésimo número de Catalan.

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1}$$

$$C_n = \frac{2(2n-1)}{n+1} C_{n-1}$$

$$C_n \approx \frac{4^n}{n^{3/2}\sqrt{\pi}}$$

4.3 Separadores o barras y estrellas

En combinatoria, el método de separadores (o barras y estrellas) es una técnica para contar formas de dividir n objetos idénticos en k grupos.
Imagina que tienes n caramelos idénticos y quieres repartirlos entre k niños. Para resolver esto, colocas $k-1$ separadores entre los caramelos, dividiendo el total en k partes. Cada parte representa la cantidad de caramelos que cada niño recibe, y puede ser cero.
El numero de formas de hacer esto es $\binom{n+k-1}{k-1}$ ya que estamos eligiendo posiciones para los separadores entre los $n+k-1$ espacios disponibles.

$$\left(\binom{n}{k}\right) = \binom{n-1}{k-1} = \binom{n+k-1}{k-1}$$

4.4 Permutaciones

Una permutación es una forma de elegir y organizar k elementos de un grupo de tamaño n, donde el orden sí importa. (i.e. 1,2,3 es diferente a 3,1,2)

$$P(n,k) = \frac{n!}{(n-k)!}$$

Permutaciones objetos repetidos

Si existen elementos iguales entre los que hay que elegir (i.e. CARTA tiene dos 'A'), definamos la cantidad que ocurre el i-esimo elemento como am_i (as in amount)-

$$P(n,k) = \frac{P(n,k)}{e_1!e_2!...} = \frac{n!}{(n-k)!e_1!e_2!...}$$

4.5 Derangement

Un Derangement es una permutación que no deja ningún elemento en su lugar original.

$$Derangement(n) = \begin{cases} 0 & \text{si } n = 1. \\ 1 & \text{si } n = 2. \\ (n-1)(Derangement(n-1) + Derangement(n-2)) & \text{otherwise.} \end{cases}$$

$$Derangement(n) = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!}$$

4.6 Numeros de Fibonacci

$$F_n = F_{n-1} + F_{n-2}$$

$$F_{2n+1} = F_n^2 + F_{n+1}^2$$

$$F_{2n} = F_{n+1}^2 - F_{n-1}^2$$

$$\sum_{i=1}^{n} F_i = F_{n+2} - 1$$

$$F_{n+i}F_{n+j} - F_nF_{n+i+j} = (-1)^n F_iF_j$$

4.7 Cantidad de divisores

Para encontrar la cantidad de divisores de un número, se usa su *factorización prima*. Supongamos que un número N se factoriza como:

$$N = p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k}$$

donde $(p_1, p_2, \ldots, p_k)$ son los factores primos distintos de N, y $(e_1, e_2, \ldots, e_k)$ son sus respectivos exponentes.
Para obtener la cantidad total de divisores de N, se toma cada exponente, se le suma uno, y luego se multiplican todos estos valores:

$$\text{Cantidad de divisores} = (e_1 + 1) \times (e_2 + 1) \times \cdots \times (e_k + 1)$$

Para fines de programacion competitiva, considera que N tiene aproximadamente $\sqrt[3]{N}$ divisores.

4.8 Respuesta modulo m

En programacion competitiva, es comun encontrar problemas cuya respuesta pueda exceder el limite de long long int de C++ (10^{18}), por lo que para mantener la respuesta en un rango aceptable se pide calcularla modulo m.
Para calcular una respuesta modulo m, es necesario construir el codigo tomando en cuenta las siguientes propiedades.

$$(a+b)\%m = (a\%m + b\%m)\%m$$

$$(a-b)\%m = (a\%m - b\%m + m)\%m$$

$$(a*b)\%m = (a\%m * b\%m)\%m$$

En el caso de divisiones es necesario calcular el inverso modular del divisor, expresado por b^{-1}, esto esta codificado en modinverse.

$$(a/b)\%m = (a\%m * (b^{-1})\%m)\%m$$

4.9 bits

```cpp
#include <iostream>
#include <bitset> // For bitset representation

using namespace std;

// Function to demonstrate basic bit manipulation operations
```

```cpp
void basicBitManipulation() {
    int num = 29; // Example number: 29 in binary is 11101
    int k = 3;    // Bit position to manipulate (0-indexed)

    // AND operation
    int andResult = num & (1 << k);
    cout << "AND: " << (num & (1 << k)) << " (Is k-th bit set? " << (
        andResult != 0) << ")" << endl;

    // OR operation
    int orResult = num | (1 << k);
    cout << "OR: " << orResult << " (Set k-th bit)" << endl;

    // XOR operation
    int xorResult = num ^ (1 << k);
    cout << "XOR: " << xorResult << " (Toggle k-th bit)" << endl;

    // NOT operation
    int notResult = ~num;
    cout << "NOT: " << notResult << " (Invert all bits)" << endl;

    // Left shift
    int leftShiftResult = num << 1; // Multiply by 2
    cout << "Left Shift: " << leftShiftResult << endl;

    // Right shift
    int rightShiftResult = num >> 1; // Divide by 2
    cout << "Right Shift: " << rightShiftResult << endl;
}

// Function to demonstrate checking, setting, and clearing bits
void checkSetClearBits() {
    int num = 29; // 11101
    int k = 2;    // Check and manipulate the k-th bit

    // Check if the k-th bit is set
    bool isSet = (num & (1 << k)) != 0;
    cout << "Is k-th bit set? " << (isSet ? "Yes" : "No") << endl;

    // Set the k-th bit
    num = num | (1 << k);
    cout << "Set k-th bit: " << num << endl;

    // Clear the k-th bit
    num = num & ~(1 << k);
    cout << "Clear k-th bit: " << num << endl;

    // Toggle the k-th bit
    num = num ^ (1 << k);
    cout << "Toggle k-th bit: " << num << endl;
}

// Function to demonstrate counting bits
void countBits() {
    int num = 29; // 11101

    // Count set bits
    int count = 0;
    for (int i = 0; i < 32; i++) {
        count += (num >> i) & 1; // Check each bit
    }
    cout << "Number of set bits: " << count << endl;

    // Using bitset for representation
    bitset<32> b(num);
    cout << "Binary representation: " << b << endl;
}

int main() {
    cout << "Basic Bit Manipulation Operations:" << endl;
    basicBitManipulation();

    cout << "\nChecking, Setting, and Clearing Bits:" << endl;
    checkSetClearBits();

    cout << "\nCounting Bits:" << endl;
    countBits();

    return 0;
}
```

4.10 gcd_lcm

```cpp
#include <iostream>
#include <algorithm> // For std::swap
#include <numeric>   // For std::gcd (C++17)
```

```

using namespace std;

// Function to count the number of trailing zeroes in the binary
    representation of n
unsigned long long int trailing_zeroes(unsigned long long int n) {
    unsigned long long int bits = 0;
    while (n && (n & 1) == 0) {
        ++bits;
        n >>= 1;
    }
    return bits;
}

// Efficient GCD calculation using the Euclidean algorithm
unsigned long long int gcd(unsigned long long int a, unsigned long long
    int b) {
    if (a == 0 || b == 0) {
        return a | b; // Return non-zero value
    }

    unsigned long long int shift = trailing_zeroes(a | b);
    a >>= trailing_zeroes(a); // Remove factors of 2 from a

    do {
        b >>= trailing_zeroes(b); // Remove factors of 2 from b
        if (a > b) {
            swap(a, b);
        }
        b -= a; // Subtract a from b
    } while (b != 0);

    return a << shift; // Restore factors of 2
}

// LCM calculation using GCD
unsigned long long int lcm(unsigned long long int a, unsigned long long
    int b) {
    if (a == 0 && b == 0) {
        return 0; // LCM of 0 and 0 is undefined
    }
    return (a / gcd(a, b)) * b; // Calculate LCM
}

// Function to count the number of integers less than a that are coprime
    to a
unsigned long long int countCoprimes(unsigned long long int a) {
    unsigned long long int count = 0;
    for (unsigned long long int i = 1; i < a; i++) {
        if (gcd(i, a) == 1) {
            count++;
        }
    }
    return count;
}

int main() {
    unsigned long long int a;
    cin >> a;

    while (a != 0) {
        unsigned long long int coprimeCount = countCoprimes(a);
        cout << coprimeCount << endl;
        cin >> a;
    }

    return 0;
}
```

4.11 min_sum_sq

```
#include <iostream> // For input and output
#include <vector>   // For using the vector container
#include <cmath>    // For the sqrt function
#include <limits>   // For std::numeric_limits

using namespace std;

// Vector to store results of subproblems
vector<int> vec(60000, -1);

// Recursive function to find the minimum number of perfect squares that
    sum up to 'num'
int recurF(int num, int cont) {
    // Variable to store the result; initialized to a large value
    int res = numeric_limits<int>::max(); // Use a large value to find
```

```
        the minimum

    // Base case: if num is 1, we need one square (1^2)
    if (num == 1) return cont + 1;

    // Base case: if num is 0, we don't need any squares
    if (num == 0) return cont;

    // Calculate the integer square root of num
    int sq = (int)sqrt(num);

    // If num is a perfect square, we can return the count of squares
        used + 1
    if (num == sq * sq) return cont + 1;

    // Increment count for the current square we're considering
    cont++;

    // Iterate over all perfect squares less than or equal to num
    for (int i = sq; i > 0; i--) {
        // Check if the result for the subproblem (num - i*i) is already
            computed
        if (vec[num - i * i] != -1) {
            // If it is, use the stored result to update res
            res = min(res, vec[num - i * i] + cont);
        } else {
            // Otherwise, compute the result recursively
            int ans = recurF(num - i * i, cont);
            // Update res with the minimum value found
            res = min(res, ans);
            // Store the computed result for the subproblem
            vec[num - i * i] = ans - cont; // Store only the difference
        }
    }

    return res; // Return the minimum count of perfect squares
}

int main() {
    int m; // Variable to hold the input number
    cout << "Enter a number: ";
    cin >> m; // Read the input number

    // Call the recursive function and output the result
    cout << "Minimum number of perfect squares for " << m << " is: " <<
        recurF(m, 0) << endl;

    return 0; // Indicate successful completion of the program
}
```

4.12 mod

```
#include <iostream>
#include <vector>
#include <numeric> // For std::accumulate
#include <cmath>   // For std::pow

using namespace std;

const long long MOD = 1000000007;

// Function to perform modular addition
long long sumaMOD(long long a, long long b) {
    return (a + b) % MOD;
}

// Function to perform modular multiplication
long long multMOD(long long a, long long b) {
    return (a * b) % MOD;
}

// Function to perform modular subtraction
long long restaMOD(long long a, long long b) {
    return ((a - b) % MOD + MOD) % MOD;
}

// Function to perform modular division
long long divMOD(long long a, long long b) {
    // Fermat's Little Theorem for finding modular inverse
    // Since MOD is prime, use pow for the modular inverse
    return multMOD(a, static_cast<long long>(pow(b, MOD - 2)));
}

int main() {
    int n;
    cin >> n;
```

```
35
36      long long answer = 1;
37
38      // Calculate factorial using modular multiplication
39      for (int i = 2; i <= n; i++) {
40          answer = multMOD(answer, i);
41      }
42
43      // Count the number of trailing zeros
44      int cont = 0;
45      while (answer % 10 == 0) {
46          answer /= 10;
47          cont++;
48      }
49
50      cout << cont << endl;
51      return 0;
52  }
```

4.13 perm_comb

```
1   #include <iostream>
2
3   // Function to calculate factorial
4   long long factorial(int n) {
5       long long result = 1;
6       for (int i = 1; i <= n; i++) {
7           result *= i;
8       }
9       return result;
10  }
11
12  // Function to calculate permutations P(n, r)
13  long long permutation(int n, int r) {
14      return factorial(n) / factorial(n - r);
15  }
16
17  // Function to calculate combinations C(n, r)
18  long long combination(int n, int r) {
19      return factorial(n) / (factorial(r) * factorial(n - r));
20  }
21
22  int main() {
23      int n = 5;
24      int r = 3;
25      std::cout << "P(" << n << ",␣" << r << ")␣=␣" << permutation(n, r)
            << std::endl;
26      std::cout << "C(" << n << ",␣" << r << ")␣=␣" << combination(n, r)
            << std::endl;
27      return 0;
28  }
```

5 Geometry

6 Strings

6.1 Common

```
1   #include <iostream>
2   #include <string>
3   #include <algorithm>  // For sort, reverse, etc. // For next_permutation
4   #include <sstream>    // For stringstream (to convert between string and
        int)
5   #include <vector>     // For splitting a string
6   #include <cctype>     // For isdigit()
7
8   using namespace std;
9
10  // Convert string to integer
11  int stringToInt(const string& str) {
12      return stoi(str);  // stoi - string to integer
13  }
14
15  // Convert integer to string
16  string intToString(int num) {
17      return to_string(num);  // to_string - integer to string
18  }
19
20  // Convert char to integer (ASCII value)
21  int charToInt(char ch) {
22      return ch - '0';  // Convert character to digit
23  }
24
25  // Convert integer to char
26  char intToChar(int num) {
27      return num + '0';  // Convert digit to character
```

```
}

// Sort a string
string sortString(string str) {
    sort(str.begin(), str.end());
    return str;
}

// Reverse a string
string reverseString(string str) {
    reverse(str.begin(), str.end());
    return str;
}

// Search a substring in a string
bool searchString(const string& str, const string& sub) {
    return str.find(sub) != string::npos;  // npos means not found
}

// Split a string by a delimiter
vector<string> splitString(const string& str, char delimiter) {
    vector<string> tokens;
    string token;
    stringstream ss(str);  // Use stringstream to tokenize

    while (getline(ss, token, delimiter)) {
        tokens.push_back(token);
    }
    return tokens;
}

// Check if a string contains only digits
bool isNumeric(const string& str) {
    return all_of(str.begin(), str.end(), ::isdigit);
}

bool hasPrefix(const string& str, const string& prefix) {
    if (str.size() < prefix.size()) {
        return false;
    }
    return str.compare(0, prefix.size(), prefix) == 0;
}

bool hasSuffix(const string& str, const string& suffix) {
    if (str.size() < suffix.size()) {
        return false;
    }
    return str.compare(str.size() - suffix.size(), suffix.size(), suffix
        ) == 0;
}

// O(m x n), KMT is O(m + n)
vector<int> findAllOccurrences(const string& str, const string& sub) {
    vector<int> positions;
    size_t pos = str.find(sub);
    while (pos != string::npos) {
        positions.push_back(pos);
        pos = str.find(sub, pos + 1);  // Search from the next position
    }
    return positions;
}

vector<int> countCharFrequency(const string& str) {
    vector<int> freq(256, 0);  // For all ASCII characters
    for (char ch : str) {
        freq[ch]++;
    }
    return freq;
}

void generateSubsets(const string& str) {
    int n = str.size();
    for (int i = 0; i < (1 << n); ++i) {  // 2^n possible subsets
        string subset = "";
        for (int j = 0; j < n; ++j) {
            if (i & (1 << j)) {
                subset += str[j];
            }
        }
        cout << subset << endl;
    }
}

bool isPalindrome(const string& str) {
    int n = str.size();
    for (int i = 0; i < n / 2; ++i) {
```

```
        if (str[i] != str[n - i - 1]) {
            return false;
        }
    }
    return true;
}

//
    ================================================================================


string longestCommonPrefix(vector<string>& strs) {
    if (strs.empty()) return "";

    string prefix = strs[0];
    for (int i = 1; i < strs.size(); ++i) {
        while (strs[i].find(prefix) != 0) {
            prefix = prefix.substr(0, prefix.size() - 1);  // Reduce the
                prefix
            if (prefix.empty()) return "";
        }
    }
    return prefix;
}

int main() {
    string str = "hello world";
    str.erase(0, 6);  // Removes the first 6 characters, str becomes "
        world"

    string str = "hello world";
    str.replace(0, 5, "hi");  // Replaces "hello" with "hi", resulting
        in "hi world"

    string str = "abc";
    sort(str.begin(), str.end());  // Sort to start with the
        lexicographically smallest permutation
    do {
        cout << str << endl;
    } while (next_permutation(str.begin(), str.end()));
    // ================================================================

    // String to integer and vice versa
    string strNum = "12345";
    int num = stringToInt(strNum);
    cout << "String to integer: " << num << endl;

    string strFromInt = intToString(num);
    cout << "Integer to string: " << strFromInt << endl;

    // Char to integer and vice versa
    char ch = '7';
    int chToInt = charToInt(ch);
    cout << "Char to integer: " << chToInt << endl;

    char intToCh = intToChar(9);
    cout << "Integer to char: " << intToCh << endl;

    // Sorting a string
    string unsorted = "dcba";
    string sorted = sortString(unsorted);
    cout << "Sorted string: " << sorted << endl;

    // Reversing a string
    string original = "hello";
    string reversed = reverseString(original);
    cout << "Reversed string: " << reversed << endl;

    // Searching a substring
    string mainString = "C++ string manipulation";
    string subString = "string";
    bool found = searchString(mainString, subString);
    cout << "Substring found: " << (found ? "Yes" : "No") << endl;

    // Splitting a string
    string sentence = "C++ is awesome!";
    vector<string> words = splitString(sentence, ' ');
    cout << "Splitting sentence: ";
    for (size_t i = 0; i < words.size(); ++i) {
        cout << words[i] << " ";
    }
    cout << endl;

    // Checking if a string is numeric
    string numericStr = "123456";
    cout << "Is the string numeric? " << (isNumeric(numericStr) ? "Yes"
```

```
        : "No") << endl;

    return 0;
}
```

6.2 KMP

```
#include <iostream>
#include <string>
#include <algorithm>  // For sort, reverse, etc. // For next_permutation
#include <sstream>    // For stringstream (to convert between string and
    int)
#include <vector>     // For splitting a string
#include <cctype>     // For isdigit()

using namespace std;

void computeLPSArray(const string& pattern, vector<int>& lps) {
    int M = pattern.size();
    int length = 0; // length of the previous longest prefix suffix
    lps[0] = 0;     // LPS[0] is always 0
    int i = 1;      // i starts from 1 since lps[0] is already known

    while (i < M) {
        if (pattern[i] == pattern[length]) {
            length++;
            lps[i] = length;
            i++;
        } else {  // (pattern[i] != pattern[length])
            if (length != 0) {
                length = lps[length - 1];
                // do not increment i here
            } else {
                lps[i] = 0;
                i++;
            }
        }
    }
}

void KMPSearch(const string& text, const string& pattern) {
    int N = text.size();
    int M = pattern.size();

    // Create the LPS table
    vector<int> lps(M);
    computeLPSArray(pattern, lps);

    int i = 0;  // index for text[]
    int j = 0;  // index for pattern[]

    while (i < N) {
        if (text[i] == pattern[j]) {
            i++;
            j++;
        }

        // A match is found
        if (j == M) {
            cout << "Pattern␣found␣at␣index␣" << i - j << endl;
            j = lps[j - 1];  // Prepare for the next match, using LPS
        }

        // Mismatch after j matches
        else if (i < N && text[i] != pattern[j]) {
            // Don't match lps[0..lps[j-1]] characters, they will match
                anyway
            if (j != 0) {
                j = lps[j - 1];  // Fall back using LPS
            } else {
                i++;  // Move to the next character in the text
            }
        }
    }
}
```

7 Flow

8 Miscellaneous

8.1 Backtracking

```
#include <iostream>
#include <vector>
using namespace std;

```

```cpp
bool isSafe(vector<vector<int>>& board, int row, int col, int N) {
    for (int i = 0; i < row; i++) {
        if (board[i][col] == 1) {
            return false;
        }
    }

    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j] == 1) {
            return false;
        }
    }

    for (int i = row, j = col; i >= 0 && j < N; i--, j++) {
        if (board[i][j] == 1) {
            return false;
        }
    }

    return true;
}

bool solveNQueensUtil(vector<vector<int>>& board, int row, int N) {
    if (row == N) {
        return true;
    }

    for (int col = 0; col < N; col++) {
        if (isSafe(board, row, col, N)) {
            board[row][col] = 1;

            if (solveNQueensUtil(board, row + 1, N)) {
                return true;
            }

            board[row][col] = 0;
        }
    }

    return false;
}

void printBoard(vector<vector<int>>& board, int N) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (board[i][j] == 1) {
                cout << "Q ";
            } else {
                cout << ". ";
            }
        }
        cout << endl;
    }
}

void solveNQueens(int N) {
    vector<vector<int>> board(N, vector<int>(N, 0));

    if (solveNQueensUtil(board, 0, N)) {
        printBoard(board, N);
    } else {
        cout << "No solution exists for " << N << " queens." << endl;
    }
}

int main() {
    int N;
    cout << "Enter the value of N (for N x N board): ";
    cin >> N;

    solveNQueens(N);

    return 0;
}
```

8.2 Binary Search

```cpp
#include <iostream>
#include <vector>

using namespace std;

int binarySearch(const vector<int>& arr, int target) {
    int left = 0;
    int right = arr.size() - 1;

```

```
    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == target) {
            return mid;
        }
        if (arr[mid] < target) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
        }
    }
    return -1;
}

int firstOccurrence(const vector<int>& arr, int target) {
    int left = 0, right = arr.size() - 1, result = -1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == target) {
            result = mid;
            right = mid - 1;
        } else if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return result;
}

int lastOccurrence(const vector<int>& arr, int target) {
    int left = 0, right = arr.size() - 1, result = -1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == target) {
            result = mid;     // Save result
            left = mid + 1;   // Search in the right half for last
                occurrence
        } else if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return result;
}
```

8.3 Dijsktra

```
#include <iostream>
#include <vector>
#include <queue>
#include <utility> // for std::pair
#include <limits>  // for std::numeric_limits

using namespace std;

const int INF = numeric_limits<int>::max(); // Represents infinity

// Dijkstra's algorithm function
void dijkstra(int start, const vector<vector<pair<int, int>>>& graph) {
    int n = graph.size(); // Number of nodes in the graph

    // Distance vector to store the shortest distance from the start
        node
    vector<int> distance(n, INF);
    distance[start] = 0;

    // Priority queue to process nodes (min-heap), stores pairs {
        distance, node}
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq
        ;
    pq.push({0, start}); // Distance to start node is 0

    while (!pq.empty()) {
        int dist = pq.top().first;  // Current distance
        int node = pq.top().second; // Current node
        pq.pop();

        // If this distance is already greater than the stored distance,
```

```
            skip
        if (dist > distance[node]) continue;

        // Explore neighbors
        for (const auto& neighbor : graph[node]) {
            int nextNode = neighbor.first;
            int weight = neighbor.second;

            // If a shorter path to nextNode is found
            if (distance[node] + weight < distance[nextNode]) {
                distance[nextNode] = distance[node] + weight;
                pq.push({distance[nextNode], nextNode});
            }
        }
    }

    // Print the shortest distances from the start node
    cout << "Shortest distances from node " << start << ":\n";
    for (int i = 0; i < n; ++i) {
        if (distance[i] == INF)
            cout << "Node " << i << ": INF (unreachable)\n";
        else
            cout << "Node " << i << ": " << distance[i] << "\n";
    }
}

int main() {
    // Number of nodes (0-indexed graph)
    int n = 5;

    // Graph represented as an adjacency list
    // graph[node] = vector of {neighbor, weight}
    vector<vector<pair<int, int>>> graph(n);

    // Adding edges to the graph
    // Example graph (directed, weighted edges):
    graph[0].push_back({1, 10}); // Edge from 0 to 1 with weight 10
    graph[0].push_back({4, 5});  // Edge from 0 to 4 with weight 5
    graph[1].push_back({2, 1});  // Edge from 1 to 2 with weight 1
    graph[1].push_back({4, 2});  // Edge from 1 to 4 with weight 2
    graph[2].push_back({3, 4});  // Edge from 2 to 3 with weight 4
    graph[3].push_back({0, 7});  // Edge from 3 to 0 with weight 7
    graph[3].push_back({2, 6});  // Edge from 3 to 2 with weight 6
    graph[4].push_back({1, 3});  // Edge from 4 to 1 with weight 3
    graph[4].push_back({2, 9});  // Edge from 4 to 2 with weight 9
    graph[4].push_back({3, 2});  // Edge from 4 to 3 with weight 2

    int startNode = 0; // Starting node for Dijkstra's algorithm

    // Run Dijkstra's algorithm from the start node
    dijkstra(startNode, graph);

    return 0;
}
```

8.4 knapsack

```
#include <iostream>
#include <vector>
using namespace std;

int knapsackDP(int W, vector<int>& weights, vector<int>& values, int n)
    {
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

    for (int i = 1; i <= n; ++i) {
        for (int w = 0; w <= W; ++w) {
            if (weights[i - 1] <= w) {
                dp[i][w] = max(dp[i - 1][w], values[i - 1] + dp[i - 1][w
                    - weights[i - 1]]);
            } else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }

    return dp[n][W];
}

int main() {
    vector<int> values = {60, 100, 120};
    vector<int> weights = {10, 20, 30};
    int W = 50;
    int n = values.size();

    cout << "Maximum value in knapsack = " << knapsackDP(W, weights,
```

```
        values, n) << endl;

    return 0;
}
```

8.5 laberinto

```
#include <bits/stdc++.h>
using namespace std;
#define max 1002

char mapa[max][max];
bool visited[max][max];
queue<pair<pair<int, int>, int>> adj;
char fatherMove[max][max];
int nivel = 0;
string mov="";

int addEdge(int r, int c, int maxR, int maxC, int nivel){
    if (mapa[r][c]=='B'){
        return 1;
    }
    if(visited[r][c]) return 0;
    if(0<=r+1 && r+1<maxR && !visited[r+1][c]){
        adj.push({{r+1, c}, nivel + 1});
        fatherMove[r+1][c]='D';
    }
    if(0<=r-1 && r-1<maxR && !visited[r-1][c]){
        adj.push({{r-1, c}, nivel + 1});
        fatherMove[r-1][c]='U';
    }
    if(0<=c+1 && c+1<maxC && !visited[r][c+1]){
        adj.push({{r, c+1}, nivel + 1});
        fatherMove[r][c+1]='R';
    }
    if(0<=c-1 && c-1<maxC && !visited[r][c-1]){
        adj.push({{r, c-1}, nivel + 1});
        fatherMove[r][c-1]='L';
    }
    return 0;
}

void moves(int r, int c){
    if (mapa[r][c] == 'A'){
        return;
    }
    mov += fatherMove[r][c];

    if(fatherMove[r][c] == 'U'){
        moves(r+1, c);
    }
    else if(fatherMove[r][c] == 'D'){
        moves(r-1, c);
    }
    else if(fatherMove[r][c] == 'R'){
        moves(r, c-1);
    }
    else if(fatherMove[r][c] == 'L'){
        moves(r, c+1);
    }
}

int main(){
    int row;
    int col;
    cin>>row>>col;

    for(int i=0 ; i<row ; i++){
        for(int j=0 ; j<col ; j++){
            cin>>mapa[i][j];
            if(mapa[i][j]=='#') visited[i][j]=true;
            else visited[i][j]=false;
            fatherMove[i][j]='X';
        }
    }

    for(int i=0 ; i<row ; i++){
        for(int j=0 ; j<col ; j++){
            if (mapa[i][j]=='A'){
                adj.push({{i,j}, 0});
                while(adj.size() != 0){
                    int topR=adj.front().first.first;
                    int topC=adj.front().first.second;
                    int level=adj.front().second;
                    adj.pop();
                    if (addEdge(topR, topC, row, col, level)==1){
```

```
                        moves(topR, topC);
                        cout<<"YES"<<endl;
                        cout<<level<<endl;
                        reverse(mov.begin(),mov.end());
                        cout<<mov;
                        return 0;
                    }
                    visited[topR][topC]=true;
                }
                cout<<"NO";
                return 0;
            }

        }
    }
}
```

8.6 lcs

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

string printLCS(string X, string Y) {
    int m = X.length();
    int n = Y.length();

    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (X[i - 1] == Y[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
            } else {
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
            }
        }
    }

    string lcs = "";
    int i = m, j = n;
    while (i > 0 && j > 0) {
        if (X[i - 1] == Y[j - 1]) {
            lcs = X[i - 1] + lcs;
            i--;
            j--;
        } else if (dp[i - 1][j] > dp[i][j - 1]) {
            i--;
        } else {
            j--;
        }
    }

    return lcs;
}

int main() {
    string X = "ABCBDAB";
    string Y = "BDCAB";

    cout << "LCS:␣" << printLCS(X, Y) << endl;

    return 0;
}
```

8.7 most_data_structures

```
#include <iostream>
#include <vector>
#include <deque>
#include <map>
#include <set>
#include <unordered_map>
#include <stack>
#include <queue>
#include <algorithm> // for std::binary_search // for std::sort // for
    lower_bound and upper_bound

using namespace std;

int main() {
    // 1. Vector - dynamic array with additional methods
    vector<int> v = {5, 1, 3, 8, 6};

    sort(v.begin(), v.end()); // Sorting vector in ascending order
```

```
    cout << "Sorted vector: ";
    for (int x : v) cout << x << " "; // Output: 1 3 5 6 8
    cout << endl;

    v.push_back(10); // Adding an element at the end
    v.pop_back();    // Removing the last element (10)
    v.erase(v.begin()); // Erasing the first element (1)
    v.size();
    v.front(); //return first element
    v.back(); //return last element
    cout << "Vector after pop_back and erase: ";
    for (int x : v) cout << x << " "; // Output: 3 5 6 8
    cout << endl;

    const int value_to_find = 4;
    bool found = binary_search(v.begin(), v.end(), value_to_find);
    if (found) {
        cout << value_to_find << " is present in the sorted vector." <<
            endl;
    } else {
        cout << value_to_find << " is not present in the sorted vector."
             << endl;
    }

    int value = 30;
    // Find the first element >= value
    vector<int>::iterator lower = lower_bound(v.begin(), v.end(), value)
        ;
    if (lower != v.end())
        cout << "Lower bound of " << value << " is at position: " << (
            lower - v.begin()) << endl;

    // Find the first element > value
    vector<int>::iterator upper = upper_bound(v.begin(), v.end(), value)
        ;
    if (upper != v.end())
        cout << "Upper bound of " << value << " is at position: " << (
            upper - v.begin()) << endl;

    //
        ================================================================================


    // 2. Deque - double-ended queue with more operations
    deque<int> d = {10, 20, 30};
    d.push_front(5);  // Adding element at the front
    d.push_back(40);  // Adding element at the back

    d.pop_front(); // Removing element from the front
    d.pop_back();  // Removing element from the back
    d.size();

    cout << "Deque after pop_front and pop_back: ";
    for (int x : d) cout << x << " "; // Output: 20 30
    cout << endl;

    //
        ================================================================================


    // 3. Map - ordered key-value pairs
    map<string, int> mp;
    mp["apple"] = 3;
    mp["banana"] = 5;
    mp["orange"] = 2;

    map<string, int>::iterator it = mp.find("banana"); // Finding a key
        in the map
    if (it != mp.end()) {
        cout << "Found banana, value = " << it->second << endl; //
            Output: 5
        mp.erase(it); // Erase key-value pair for "banana"
    }

    cout << "Map after erasing 'banana':" << endl;
    for (pair<string, int> p : mp) {
        cout << p.first << ": " << p.second << endl; // Output: apple:
            3, orange: 2
    }

    // sort by val
    // Vector to store pairs of the map
    vector<pair<string, int>> vec(mp.begin(), mp.end());

    // Sorting the vector based on the values of the map
    sort(vec.begin(), vec.end(), [](const pair<string, int>& a, const
```

```
        pair<string, int>& b) {
        return a.second < b.second; // Compare based on the value (
            second element of pair)
    });

    //
        ==========================================================================


    // 4. Set - unique sorted elements with additional operations
    set<int> s = {4, 1, 9, 1, 3}; // Duplicates are ignored

    s.insert(2); // Insert element
    s.erase(4);  // Erase element 4
    s.size();

    if (s.find(2) != s.end()) { // Finding an element
        cout << "2 is in the set" << endl; // Output: 2 is in the set
    }

    cout << "Set elements after insert and erase: ";
    for (int x : s) cout << x << " "; // Output: 1 2 3 9
    cout << endl;

    multiset<int> ms = {4, 1, 9, 1, 3}; // Duplicates are present

    ms.insert(2); // Insert element
    ms.erase(4);  // Erase element 4
    ms.count(4); // times in multiset
    ms.size();

    if (ms.find(2) != ms.end()) { // Finding an element
        cout << "2 is in the set" << endl; // Output: 2 is in the set
    }

    int multiset_value = 30;
    // lower_bound: finds the first element >= value
    multiset<int>::iterator ms_lower = s.lower_bound(multiset_value);
    if (ms_lower != s.end())
        cout << "Lower bound of " << multiset_value << " is: " << *
            ms_lower << endl;

    // upper_bound: finds the first element > value
    multiset<int>::iterator ms_upper = s.upper_bound(multiset_value);
    if (ms_upper != s.end())
        cout << "Upper bound of " << multiset_value << " is: " << *
            ms_upper << endl;

    //
        ==========================================================================


    // 5. Unordered_map - hash table based key-value pairs
    unordered_map<string, int> um;
    um["cat"] = 1;
    um["dog"] = 2;
    um["bird"] = 3;

    um.erase("dog"); // Erasing key-value pair for "dog"

    cout << "Unordered map after erasing 'dog':" << endl;
    for (unordered_map<string, int>::iterator p = um.begin(); p != um.
        end(); ++p) {
        cout << p->first << ": " << p->second << endl; // Output: cat:
            1, bird: 3
    }

    //
        ==========================================================================


    // 6. Stack - LIFO structure with additional methods
    stack<int> st;
    st.push(10);
    st.push(20);
    st.push(30);

    cout << "Stack top: " << st.top() << endl; // Output: 30
    st.pop(); // Removing the top element
    cout << "Stack top after pop: " << st.top() << endl; // Output: 20

    //
        ==========================================================================


    // 7. Queue - FIFO structure with more operations
```

```cpp
    queue<int> q;
    q.push(1);
    q.push(2);
    q.push(3);

    cout << "Queue front: " << q.front() << endl; // Output: 1
    q.pop(); // Remove front element
    cout << "Queue front after pop: " << q.front() << endl; // Output: 2

    //
        ========================================================================


    // 8. Priority Queue - max heap (largest element first) by default
        with more operations
    priority_queue<int> pq;
    pq.push(50);
    pq.push(10);
    pq.push(30);

    cout << "Priority queue top (max heap): " << pq.top() << endl; //
        Output: 50
    pq.pop();
    cout << "Priority queue top after pop: " << pq.top() << endl; //
        Output: 30

    // Min heap priority queue (smallest element first)
    priority_queue<int, vector<int>, greater<int>> minHeap;
    minHeap.push(50);
    minHeap.push(10);
    minHeap.push(30);

    cout << "Priority queue top (min heap): " << minHeap.top() << endl;
        // Output: 10
    minHeap.pop();
    cout << "Priority queue top after pop: " << minHeap.top() << endl;
        // Output: 30

    return 0;
}
```

8.8 num_prime_factors

```cpp
#include <iostream>
#include <vector>
#include <bitset>
#include <set>
#include <cmath>

using namespace std;

// Constant to define the maximum limit for prime calculation
const int MAX = 3000;
// Function to generate a list of prime numbers using the Sieve of
    Eratosthenes
vector<int> primes() {
    bitset<MAX> bits;
    bits[0] = bits[1] = 1; // Mark 0 and 1 as non-prime
    int sqrt_max = static_cast<int>(sqrt(MAX)) + 1;
    vector<int> primes_vec;

    // Sieve of Eratosthenes
    for (int i = 2; i <= sqrt_max; i++) {
        if (bits[i]) continue; // Skip if i is marked as non-prime
        for (int j = i * i; j < MAX; j += i) {
            bits[j] = 1; // Mark multiples of i as non-prime
        }
    }

    // Collecting prime numbers
    for (int i = 2; i < bits.size(); i++) {
        if (!bits[i]) {
            primes_vec.push_back(i); // Add prime number to the vector
        }
    }
    return primes_vec;
}

int main() {
    int n;
    vector<int> primes_v = primes(); // Generate list of primes

    cin >> n; // Input number
    int count = 0; // Count of numbers with exactly two distinct prime
        factors
```

```
42
43      // Loop through all numbers from n down to 1
44      while (n > 0) {
45          vector<int> factors; // To store factors of the current number
46          set<int> factorsSet; // To store unique factors
47          int m = n; // Copy of n for factorization
48
49          // Factorization using the list of prime numbers
50          for (int x : primes_v) {
51              while (m % x == 0) {
52                  factors.push_back(x); // Store factor
53                  m /= x; // Reduce m by the factor
54              }
55          }
56
57          // Insert factors into the set for uniqueness
58          for (int x : factors) {
59              factorsSet.insert(x);
60          }
61
62          // Check if the number has exactly two distinct prime factors
63          if (factorsSet.size() == 2) {
64              count++;
65          }
66          n--; // Decrement n to check the next number
67      }
68
69      cout << count; // Output the count
70      return 0;
71  }
```

8.9 primes_in_range

```
1   #include <iostream>
2   #include <vector>
3   #include <bitset>
4   #include <set>
5   #include <cmath>
6
7   using namespace std;
8
9   // Constant to define the maximum limit for prime calculation
10  const int MAX = 3000;
11
12  // Function to generate a list of prime numbers using the Sieve of
        Eratosthenes
13  vector<int> primes() {
14      bitset<MAX> bits;
15      bits[0] = bits[1] = 1; // Mark 0 and 1 as non-prime
16      int sqrt_max = static_cast<int>(sqrt(MAX)) + 1;
17      vector<int> primes_vec;
18
19      // Sieve of Eratosthenes
20      for (int i = 2; i <= sqrt_max; i++) {
21          if (bits[i]) continue; // Skip if i is marked as non-prime
22          for (int j = i * i; j < MAX; j += i) {
23              bits[j] = 1; // Mark multiples of i as non-prime
24          }
25      }
26
27      // Collecting prime numbers
28      for (int i = 2; i < bits.size(); i++) {
29          if (!bits[i]) {
30              primes_vec.push_back(i); // Add prime number to the vector
31          }
32      }
33      return primes_vec;
34  }
35
36  int main() {
37      int m, l, u;
38      vector<int>primes_v=primes();
39
40      cin>>m;
41
42      while(m){
43          cin>>l>>u;
44
45          vector<int>::iterator mi, mx;
46
47          mi=upper_bound(primes_v.begin(), primes_v.end(),l-1);
48          mx=upper_bound(primes_v.begin(), primes_v.end(),u);
49
50          if (u==1) {
51              cout<<0<<endl;
52          }
```

```
        else{
            cout<<(int)(mx-primes_v.begin())-(int)(mi-primes_v.begin())
                <<endl;
        }
        m--;
    }
}
```

8.10 weird_input

```
#include <bits/stdc++.h>
using namespace std;
int r=101;
int c=101;
int mapa[101][101];
int ady[101][101];

int main(){
      // For getting input from input.txt file
    freopen("input.txt", "r", stdin);

    // Printing the Output to output.txt file
    freopen("output.txt", "w", stdout);

    int rows, cols;
    cin>>rows>>cols;
    string inp;
    for(int i=0 ; i<rows ; i++){
        cin>>inp;
        for(int j=0 ; j<cols ; j++){
            mapa[i][j]=inp[j]-'0';
        }
    }

    int pintura;

    for(int i=0 ; i<rows ; i++){
        for(int j=0 ; j<cols ; j++){
            if (mapa[i][j] == 0) continue; else pintura += 2;

            if((j-1)<0) {pintura +=mapa[i][j];}
            else{
                if (ady[i][j-1] == 0){
                    pintura += abs(mapa[i][j]-mapa[i][j-1]);
                }
            }
            if((j+1)>=cols) {pintura +=mapa[i][j];}
            else{
                if (ady[i][j+1] == 0){
                    pintura += abs(mapa[i][j]-mapa[i][j+1]);
                }
            }
            if((i-1)<0) {pintura +=mapa[i][j];}
            else{
                if (ady[i-1][j] == 0){
                    pintura += abs(mapa[i][j]-mapa[i-1][j]);
                }
            }
            if((i+1)>=rows) {pintura +=mapa[i][j];}
            else{
                if (ady[i+1][j] == 0){
                    pintura += abs(mapa[i][j]-mapa[i+1][j]);
                }
            }
            ady[i][j]=1;
        }
    }

    cout<<pintura;
    return 0;
}
```

9 Testing