

**INFORME DE EVALUACIÓN DE DESEMPEÑO**

CURSO DE FUNDAMENTOS DE PROGRAMACION COMPETITIVA – CC217

Carrera de Ciencias de la computación

Sección: CC52

Alumno:

U201922128

Daniel Ulises Barrionuevo Gutiérrez

Junio 2023

**CONTENIDO**

1. Índice
2. Resumen
3. Objetivo del estudiante (Student Outcome)
4. Diseña una solución basada en computación para cumplir con el conjunto de requerimientos en el contexto de sistemas de información
5. Implementa una solución basada en computación para cumplir con el conjunto de requerimientos en el contexto de sistemas de información
6. Evalúa una solución basada en computación para cumplir con el conjunto de requerimientos en el contexto de sistemas de información
7. Conclusiones
8. Recomendaciones
9. Anexos
10. Bibliografía
11. **ÍNDICE**
12. **RESUMEN**

* La problemática que se pudo encontrar al momento de desarrollar cada solución de los ejercicios fue variada, ya que cada uno se aplicaba a diferentes sectores y contextos tales como educación, problemas de la vida diaria, salud, etc. Esto debido a que la tarea de programar es de gran ayuda en casi cualquier sector si se sabe donde y como aplicarla para hallar las soluciones de manera mas sencilla

1. **OBJETIVO DEL ESTUDIANTE**

* Considero que se ha logrado el objetivo, ya que gracias a las clases brindadas a través de todo el ciclo académico pude adquirir los conocimientos necesarios para diseñar, implementar y llevar a cabo una solución valida para cada problema que se presentó, aplicando diferentes estructuras de datos para cada problema, sabiendo diferenciar en que caso es mejor aplicar cada una y sus ventajas y desventajas.

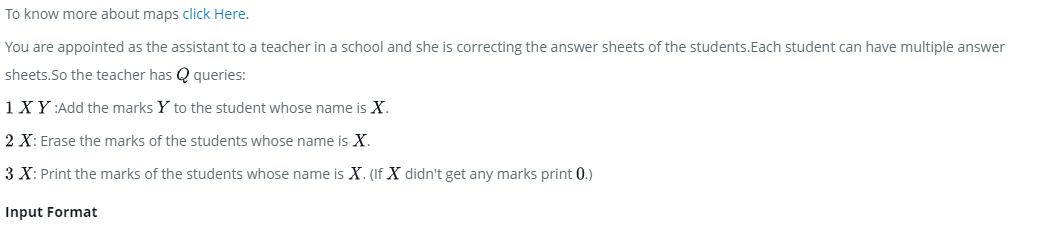
1. **DISEÑA UNA SOLUCIÓN BASADA EN COMPUTACIÓN PARA CUMPLIR CON EL CONJUNTO DE REQUERIMIENTOS EN EL CONTEXTO DE SISTEMAS DE INFORMACIÓN**

* En el trabajo presentado se hizo uso de múltiples plataformas de programación, entre las cuales están leetcode, hackerrank, codeforces, etc. Ya que estos aparte de ser más conocidas por la comunidad de programadores, presenta ejercicios variados entre los cuales en cada uno se aplica un método se solución diferente.
* Durante este trabajo se dividieron los ejercicios de cada tipo, esto debido a que cada uno tenía diferentes oportunidades para poder desarrollarlos. Sin embargo, al momento de presentar dudas nos apoyamos entre nosotros

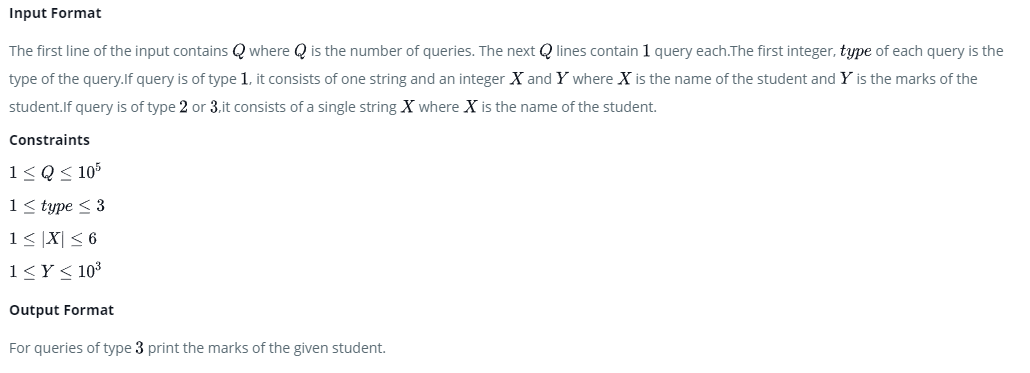
1. **IMPLEMENTA UNA SOLUCIÓN BASADA EN COMPUTACIÓN PARA CUMPLIR CON EL CONJUNTO DE REQUERIMIENTOS EN EL CONTEXTO DE SISTEMAS DE INFORMACIÓN**

**Pregunta 1: (Map):**

**Enunciado:**



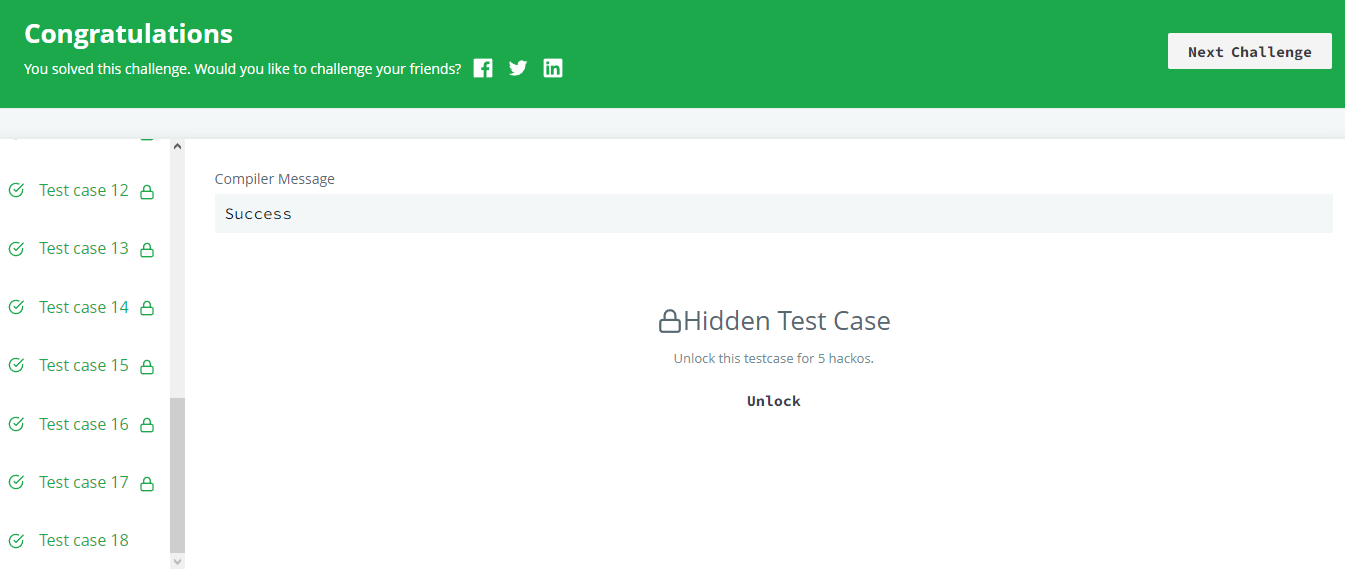
**Entrada:**

****

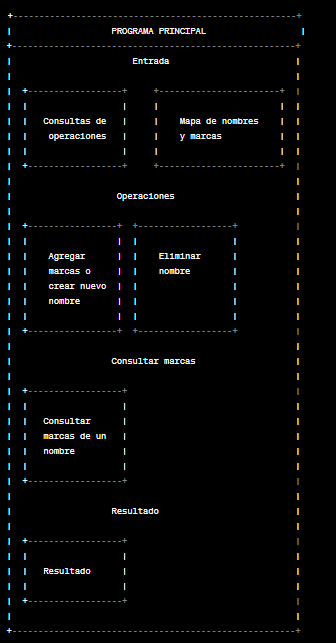
**Código:**

|  |
| --- |
| **#include <cmath>**  **#include <cstdio>**  **#include <vector>**  **#include <iostream>**  **#include <set>**  **#include <map>**  **#include <algorithm>**  **using namespace std;**  **int main() {**  ***/\* Enter your code here. Read input from STDIN. Print output to STDOUT \*/***  **int q;**  **cin >> q;**    **map<string, int> marks;**  **while (q--) {**  **int type;**  **cin >> type;**    **if (type == 1) {**  **string name;**  **int mark;**  **cin >> name >> mark;**  **marks[name] += mark;**  **} else if (type == 2) {**  **string name;**  **cin >> name;**  **marks.erase(name);**  **} else if (type == 3) {**  **string name;**  **cin >> name;**  **if (marks.count(name)) {**  **cout << marks[name] << endl;**  **} else {**  **cout << 0 << endl;**  **}**  **}**  **}**  **return 0;**  **}** |

**Captura:**

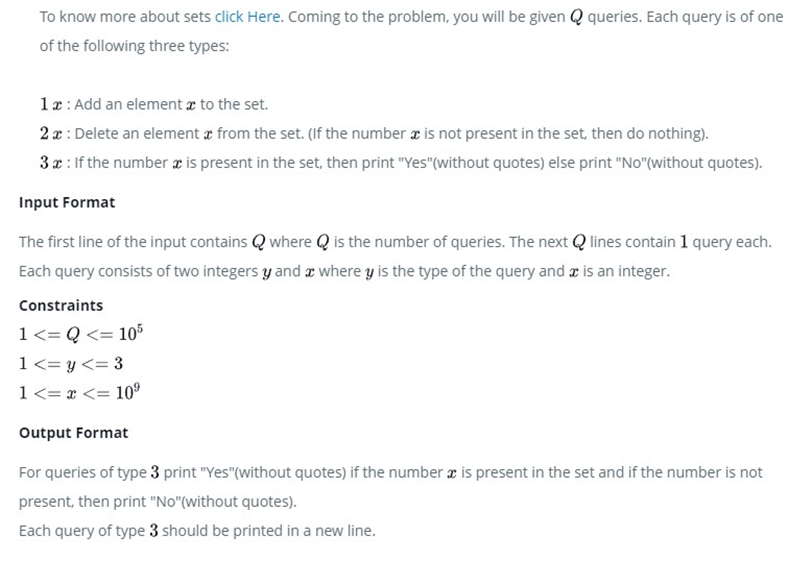
****

**Explicación:** El programa principal recibe consultas de entrada y realiza operaciones en un mapa que registra nombres y marcas. Existen tres tipos de operaciones: (1) agregar marcas a un nombre o crear un nuevo nombre con marcas, (2) eliminar un nombre del registro, y (3) consultar las marcas de un nombre. Las operaciones de adición y consulta garantizan un resultado de 0 para nombres no registrados. Todo esto se logra utilizando una estructura de datos de mapa en C++, que proporciona inserciones, borrados y búsquedas eficientes.

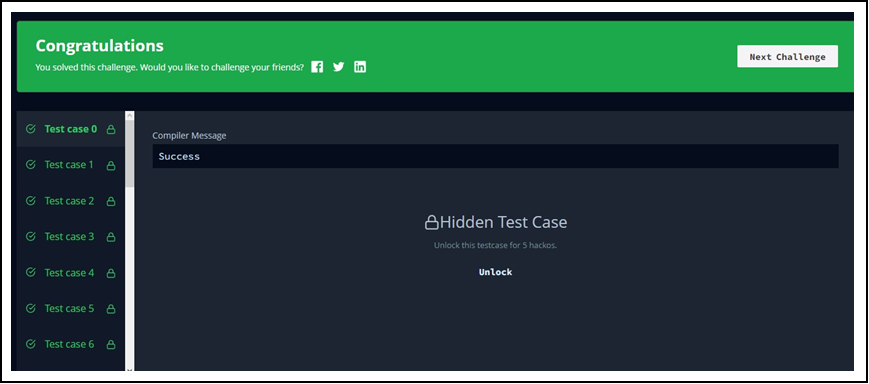


**Pregunta 2: (Set):**

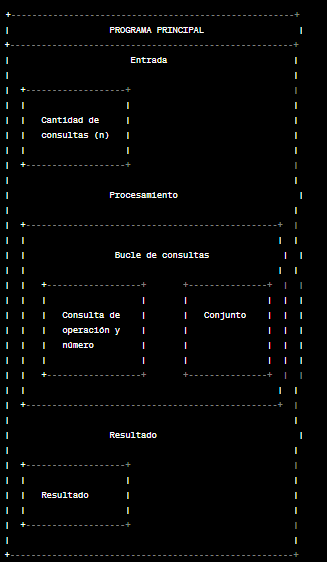
**Enunciado:**

**Código:**

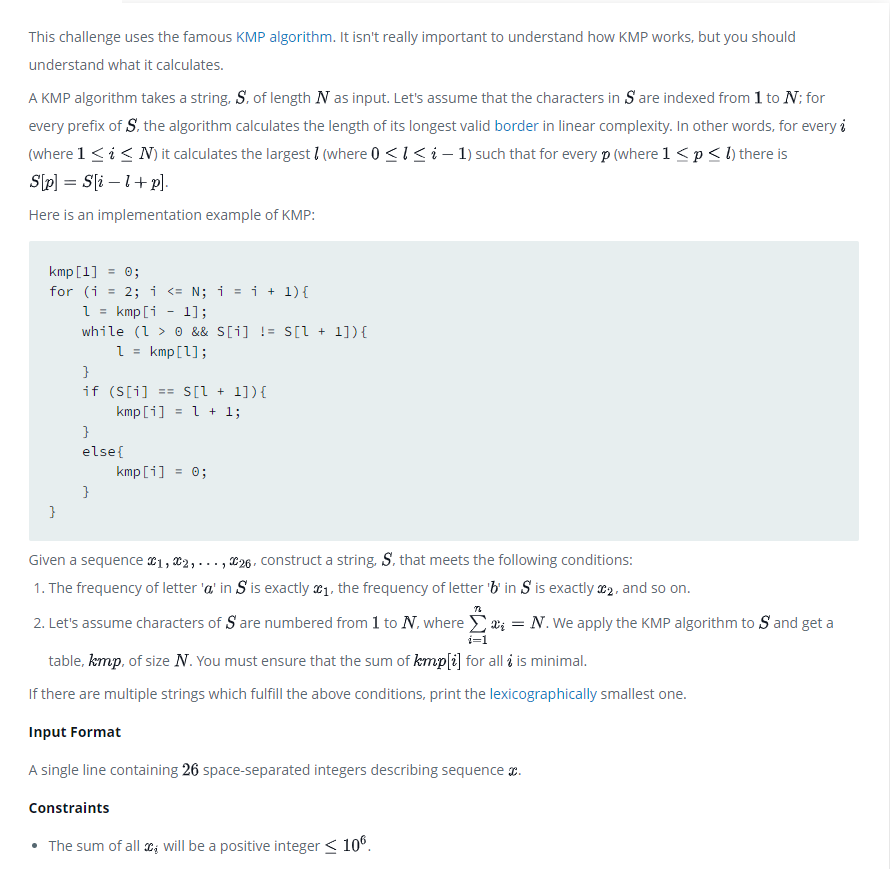
|  |
| --- |
| **#include <iostream>**  **#include <set>**  **using namespace std;**  **int main() {**  **/\* Enter your code here. Read input from STDIN. Print output to STDOUT \*/**    **set<int> s;**  **int n;**  **do {**  **cin >> n;**  **} while (n<1 || n>100000);**    **for (int i = 0; i < n; i++) {**  **int tipo, num;**  **cin >> tipo >> num;**  **if (tipo == 1) {**  **s.insert(num);**  **}**  **else if (tipo == 2) {**  **s.erase(num);**  **}**  **else if (tipo == 3) {**  **if (s.find(num) != s.end()) {**  **cout << "Yes" << endl;**  **}**  **else {**  **cout << "No" << endl;**  **}**  **}**  **}**  **return 0;** |

**Captura:**

**Explicación:**

Este programa principal realiza operaciones en un conjunto basado en consultas de entrada. Primero, lee un número n que indica la cantidad de consultas. Luego, procesa cada consulta que tiene un tipo y un número. Hay tres tipos de operaciones: (1) insertar el número en el conjunto, (2) borrar el número del conjunto, y (3) verificar si el número existe en el conjunto y responder con "Yes" o "No". El conjunto garantiza que cada número es único y permite operaciones eficientes de inserción, borrado y búsqueda.

**Pregunta 3: (KMP Algorithm):**

**Enunciado:**

**Entrada:**

|  |
| --- |
| **2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0** |

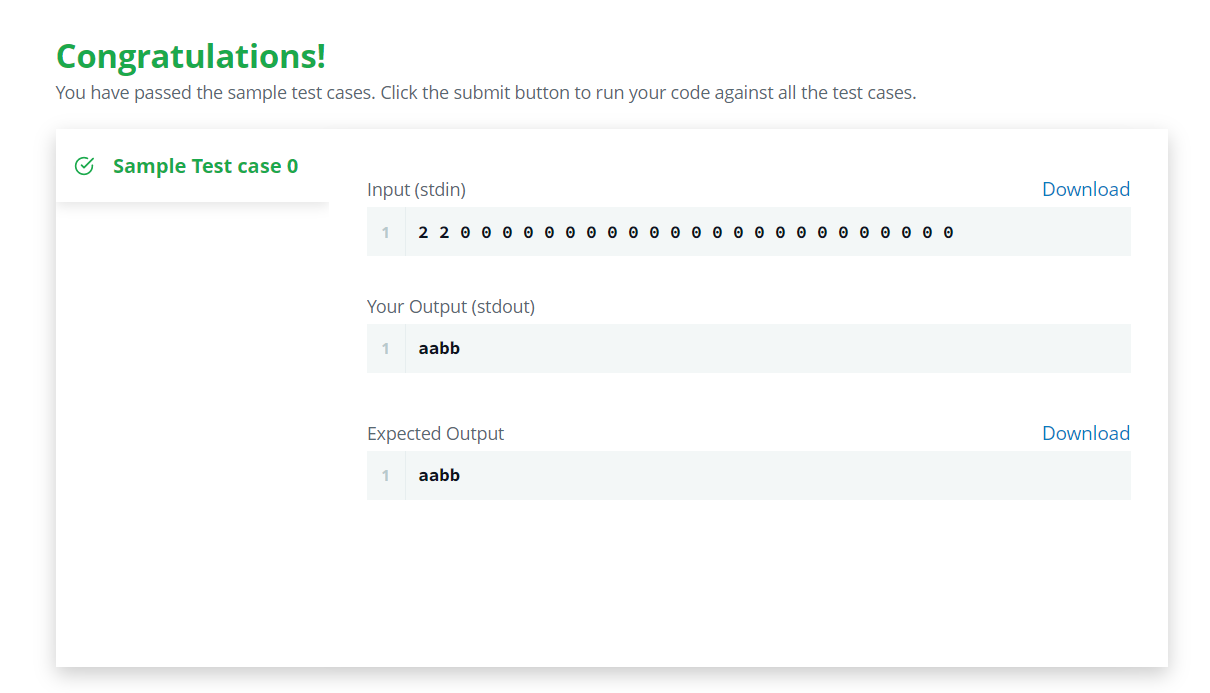
**Salida:**

|  |
| --- |
| **aabb** |

**Código:**

|  |
| --- |
| **#include <bits/stdc++.h>**  **using namespace std;**  **string ltrim(const string &);**  **string rtrim(const string &);**  **vector<string> split(const string &);**  **string kmp(vector<int> x) {**  **string s = "";**  **int min\_freq = INT\_MAX, min\_index = -1;**  **for (int i = 0; i < 26; i++) {**  **if (x[i] > 0 && x[i] < min\_freq) {**  **min\_freq = x[i];**  **min\_index = i;**  **}**  **}**  **for (int i = 0; i < min\_freq; i++) {**  **s += (char)(min\_index + 'a');**  **}**  **x[min\_index] -= min\_freq;**  **for (int i = 0; i < 26; i++) {**  **if (x[i] > 0) {**  **s += (char)(i + 'a');**  **x[i]--;**  **break;**  **}**  **}**  **for (int i = 0; i < 26; i++) {**  **if (i == min\_index && x[i] > 0) {**  **s += string(x[i], (char)(i + 'a'));**  **x[i] = 0;**  **}**  **else if (x[i] > 0) {**  **s += string(x[i], (char)(i + 'a'));**  **}**  **}**  **return s;**  **}**  **int main()**  **{**  **ofstream fout(getenv("OUTPUT\_PATH"));**  **string x\_temp\_temp;**  **getline(cin, x\_temp\_temp);**  **vector<string> x\_temp = split(rtrim(x\_temp\_temp));**  **vector<int> x(26);**  **for (int i = 0; i < 26; i++) {**  **int x\_item = stoi(x\_temp[i]);**  **x[i] = x\_item;**  **}**  **string result = kmp(x);**  **fout << result << "\n";**  **fout.close();**  **return 0;**  **}**  **string ltrim(const string &str) {**  **string s(str);**  **s.erase(**  **s.begin(),**  **find\_if(s.begin(), s.end(), not1(ptr\_fun<int, int>(isspace)))**  **);**  **return s;**  **}**  **string rtrim(const string &str) {**  **string s(str);**  **s.erase( find\_if(s.rbegin(),s.rend(),not1(ptr\_fun<int,int>(isspace))).base(),**  **s.end()**  **);**  **return s;**  **}**  **vector<string> split(const string &str) {**  **vector<string> tokens;**  **string::size\_type start = 0;**  **string::size\_type end = 0;**  **while ((end = str.find(" ", start)) != string::npos) {**  **tokens.push\_back(str.substr(start, end - start));**  **start = end + 1;**  **}**  **tokens.push\_back(str.substr(start));**  **return tokens;**  **}** |

**Captura:**

****

**Explicación:**

1. La entrada al programa es una lista de 26 números enteros que representan las frecuencias de cada carácter alfabético minúscula (a-z). La entrada es leída y se almacena en el vector "x".

2. Se inicializa una cadena vacía "s", y se busca el carácter que tiene la menor frecuencia en la lista de entrada, excluyendo los que tienen frecuencia 0. Se agrega ese carácter con la menor frecuencia a la cadena "s", tantas veces como la frecuencia indica.

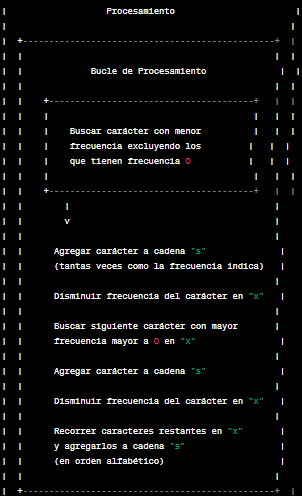
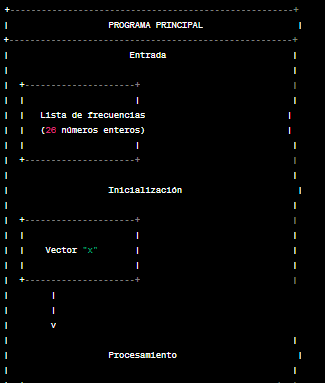
3. Luego, se disminuye la frecuencia de ese carácter en el vector "x" por la cantidad que se usó.

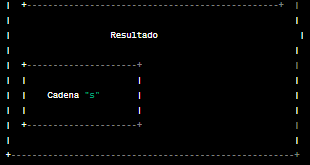
4. Ahora, se busca el siguiente carácter en orden alfabético que tenga una frecuencia mayor a 0 en el vector "x", y se añade a la cadena "s".

5. Después de eso, se disminuye la frecuencia de ese carácter en el vector "x" en 1.

6. A continuación, se recorren los caracteres restantes en el vector "x" y se añaden a la cadena "s" en orden alfabético, incluyendo el carácter que tiene la frecuencia más alta y que no pudo ser colocado en la segunda posición.

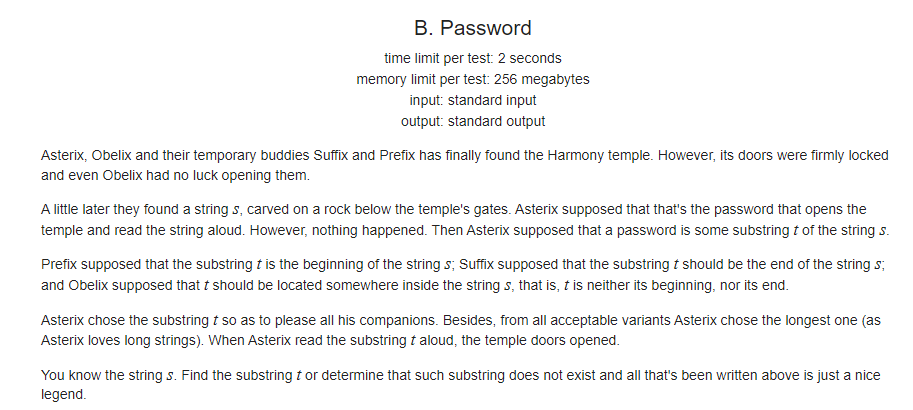
7. Finalmente, se retorna la cadena "s" que contiene la cadena lexicográficamente más pequeña que se puede formar bajo la restricción dada.

****

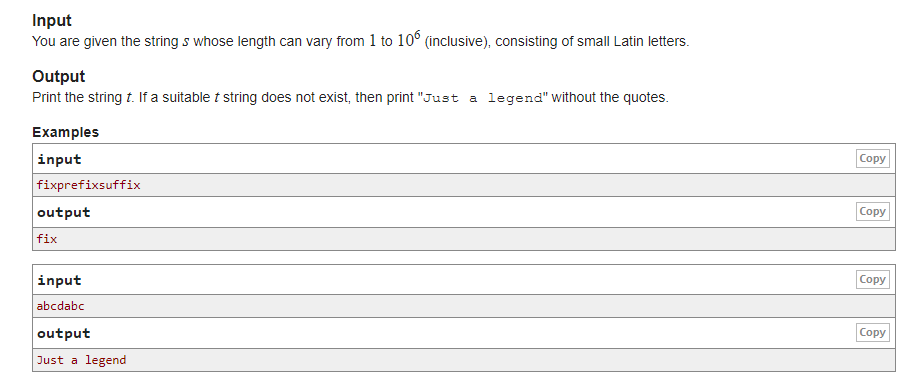
****

**Pregunta 4: (Z Algorithm):**

**Enunciado:**

****

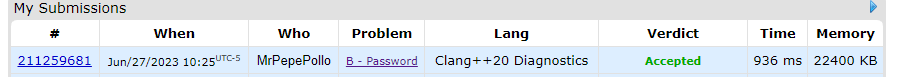
**Entrada:**

****

**Código:**

|  |
| --- |
| **#include <bits/stdc++.h>**  **using namespace std;**  **vector<int> calcularZ(string s) {**  **int n = s.length();**  **vector<int> Z(n, 0);**  **int left = 0, right = 0;**  **for (int i = 1; i < n; i++) {**  **if (i > right) {**  **left = right = i;**  **while (right < n && s[right - left] == s[right]) {**  **right++;**  **}**  **Z[i] = right - left;**  **right--;**  **}**  **else {**  **int k = i - left;**  **if (Z[k] < right - i + 1) {**  **Z[i] = Z[k];**  **}**  **else {**  **left = i;**  **while (right < n && s[right - left] == s[right]) {**  **right++;**  **}**  **Z[i] = right - left;**  **right--;**  **}**  **}**  **}**  **return Z;**  **}**  **string findSubstring(string s) {**  **int n = s.length();**  **vector<int> Z = calcularZ(s);**  **int maxZ = 0;**  **for (int i = 1; i < n; i++) {**  **if (Z[i] == n - i && maxZ >= n - i) {**  **return s.substr(0, n - i);**  **}**  **maxZ = max(maxZ, Z[i]);**  **}**  **return "Just a legend";**  **}**  **int main() {**  **string s;**  **cin >> s;**  **string t = findSubstring(s);**  **cout << t << endl;**  **return 0;**  **}** |

**Captura:**

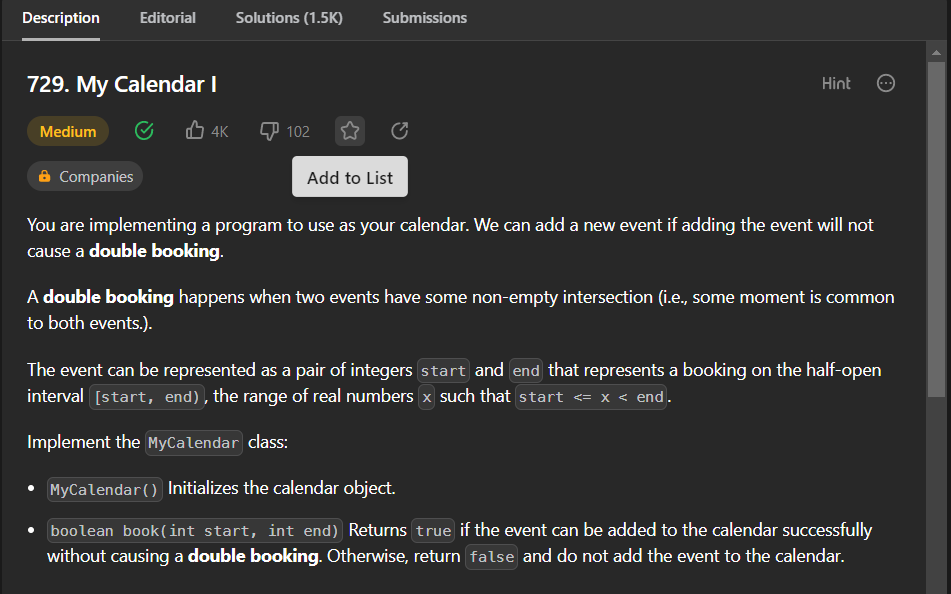
****

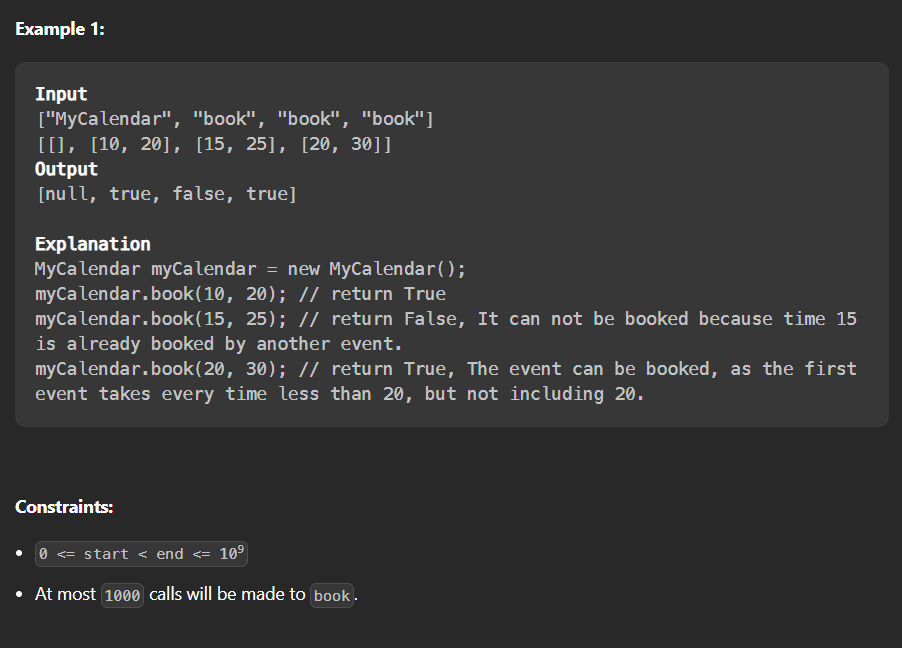
**Explicación:**

Este programa utiliza el algoritmo Z para encontrar la subcadena más larga que se encuentra al principio y al final de una cadena de entrada. La función calcularZ genera la matriz Z, que para cada índice en la cadena, denota la longitud del segmento más largo que comienza desde ese índice y es igual al segmento inicial. Luego, la función findSubstring encuentra la subcadena buscada, verificando la matriz Z: si la longitud del segmento es igual a la longitud del segmento restante y hay un segmento del mismo tamaño que aparece en otro lugar, se devuelve la subcadena. Si no se encuentra tal subcadena, se devuelve "Just a legend".

|  |
| --- |
|  |

**Pregunta 5: (Segment Tree):**

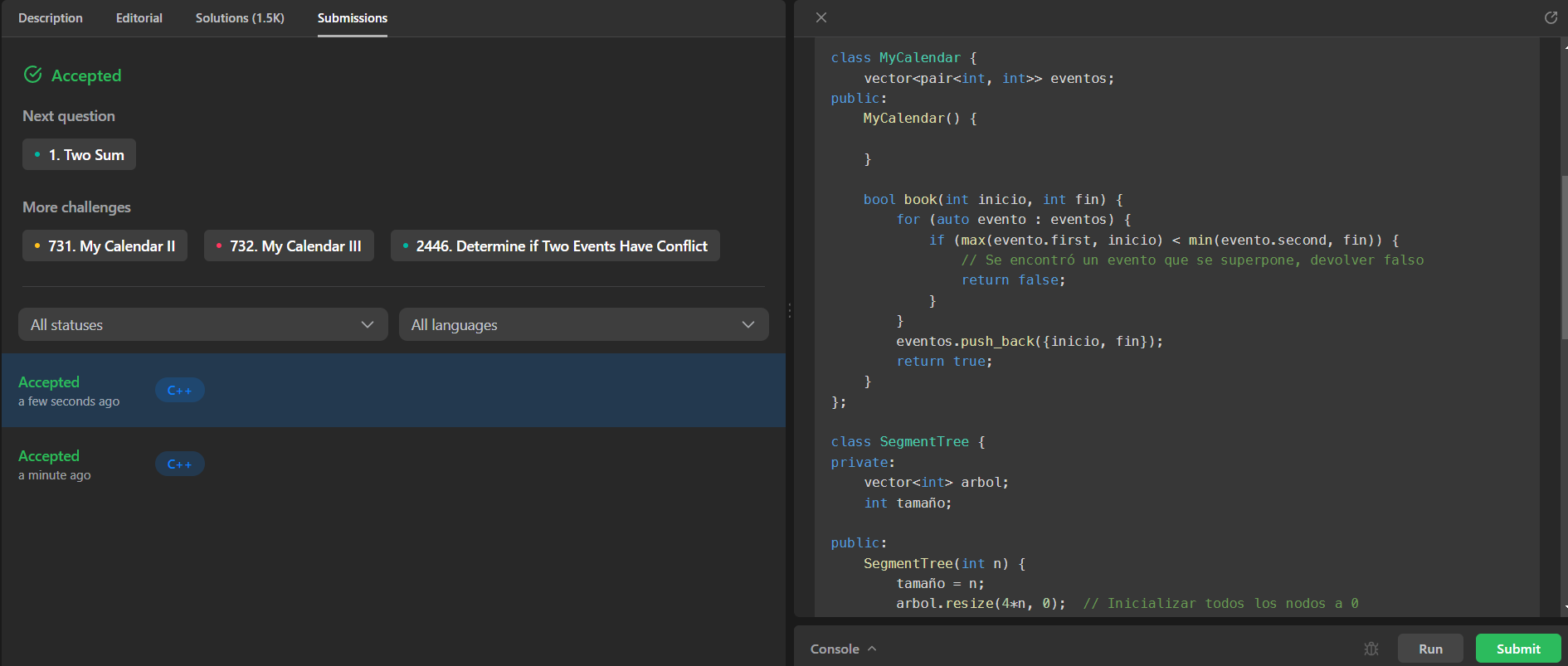
**Enunciado:**

**Entrada: **

**Código:**

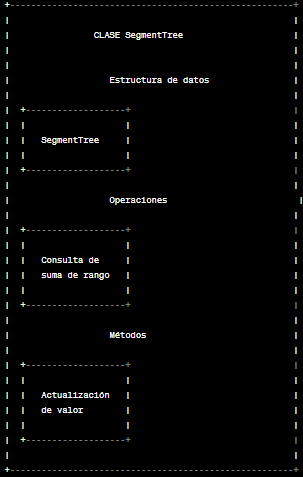
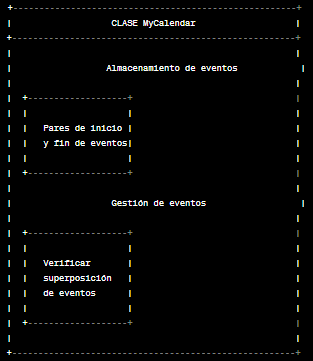
|  |
| --- |
| **class MyCalendar {**  **vector<pair<int, int>> eventos;**  **public:**  **MyCalendar() {**    **}**    **bool book(int inicio, int fin) {**  **for (auto evento : eventos) {**  **if (max(evento.first, inicio) < min(evento.second, fin)) {**  **// Se encontró un evento que se superpone, devolver falso**  **return false;**  **}**  **}**  **eventos.push\_back({inicio, fin});**  **return true;**  **}**  **};**  **class SegmentTree {**  **private:**  **vector<int> arbol;**  **int tamaño;**    **public:**  **SegmentTree(int n) {**  **tamaño = n;**  **arbol.resize(4\*n, 0); // Inicializar todos los nodos a 0**  **}**    **void actualizar(int idx, int val) {**  **actualizarAux(0, 0, tamaño-1, idx, val);**  **}**    **int consulta(int l, int r) {**  **return consultaAux(0, 0, tamaño-1, l, r);**  **}**    **void actualizarAux(int nodo, int nodoL, int nodoR, int idx, int val) {**  **if (nodoL == nodoR) {**  **// Nodo hoja**  **arbol[nodo] = val;**  **} else {**  **int mitad = (nodoL + nodoR) / 2;**  **if (idx <= mitad) {**  **// Si idx está en el hijo izquierdo, recursión en el hijo izquierdo**  **actualizarAux(2\*nodo+1, nodoL, mitad, idx, val);**  **} else {**  **// Si no, recursión en el hijo derecho**  **actualizarAux(2\*nodo+2, mitad+1, nodoR, idx, val);**  **}**  **// El nodo interno tendrá la suma de ambos hijos**  **arbol[nodo] = arbol[2\*nodo+1] + arbol[2\*nodo+2];**  **}**  **}**    **int consultaAux(int nodo, int nodoL, int nodoR, int l, int r) {**  **if (r < nodoL || nodoR < l) {**  **// Completamente fuera del segmento**  **return 0;**  **}**  **if (l <= nodoL && nodoR <= r) {**  **// Completamente dentro del segmento**  **return arbol[nodo];**  **}**  **// Parcialmente dentro y parcialmente fuera**  **int mitad = (nodoL + nodoR) / 2;**  **int p1 = consultaAux(2\*nodo+1, nodoL, mitad, l, r);**  **int p2 = consultaAux(2\*nodo+2, mitad+1, nodoR, l, r);**  **return p1 + p2;**  **}**  **};** |

**Captura:**

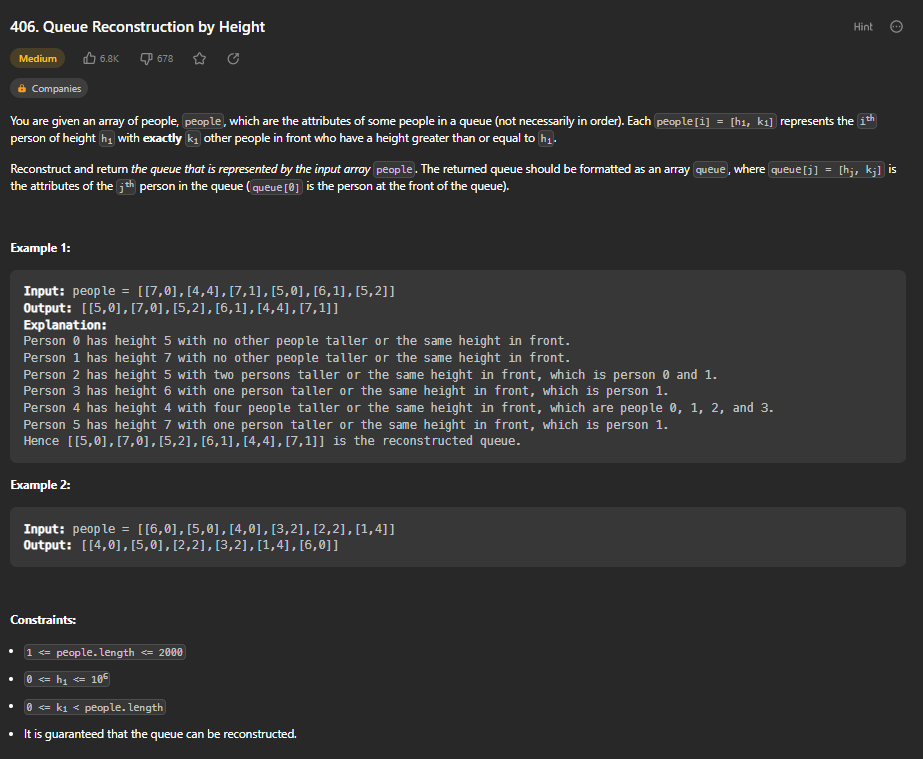
****

**Explicación:**

La clase MyCalendar almacena eventos sin superposiciones usando pares de inicio y fin. Si un nuevo evento se superpone con uno existente, se rechaza. La clase SegmentTree es una implementación de árbol de segmentos, permitiendo consultas de suma de rango y actualizaciones de valor eficientes. Los nodos almacenan sumas de segmentos del arreglo. Los métodos recursivos actualizan y consultan estos segmentos. En resumen, MyCalendar gestiona reservas de eventos y SegmentTree permite operaciones de suma y actualización eficientes en un arreglo.

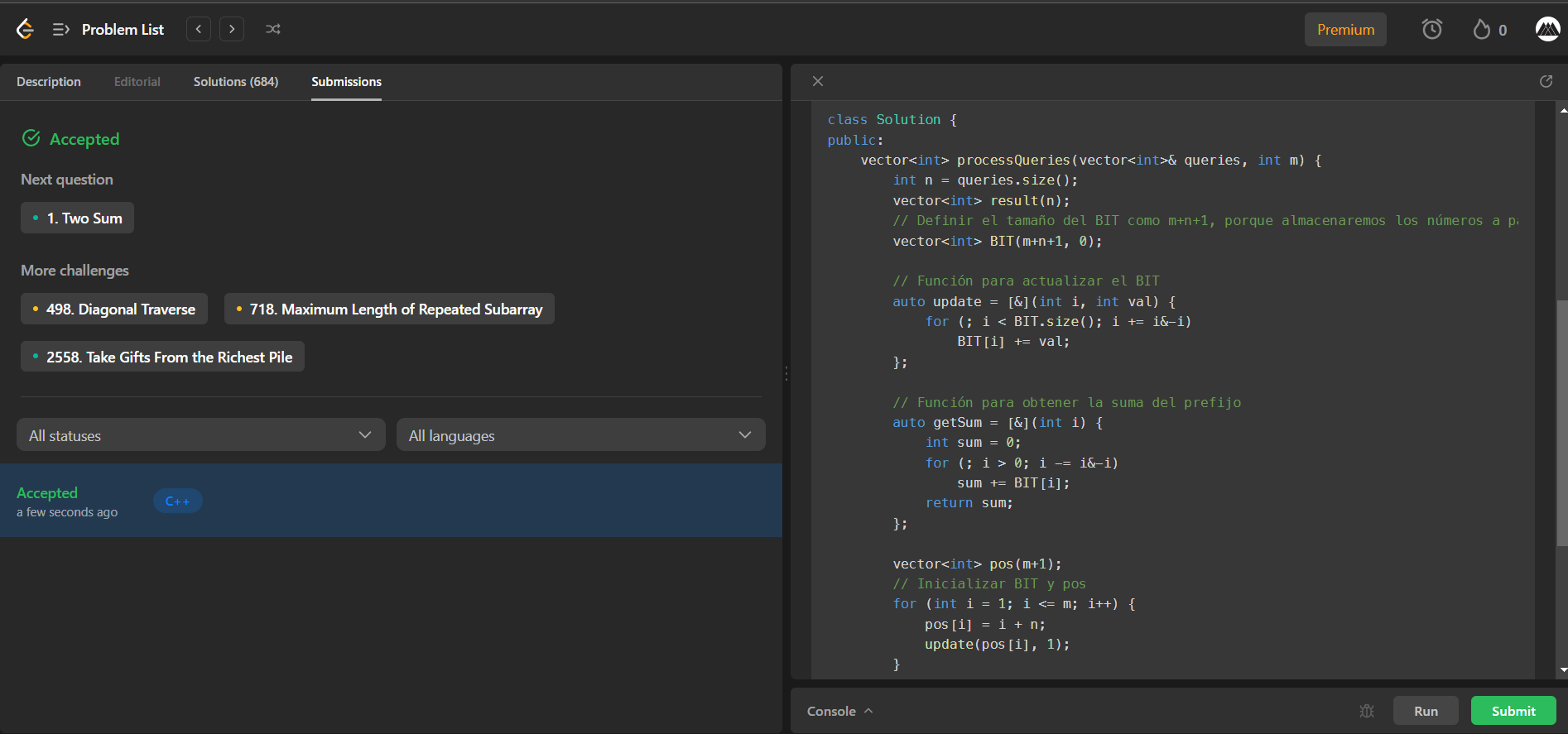


**Pregunta 6: (Árbol Fenwick):**

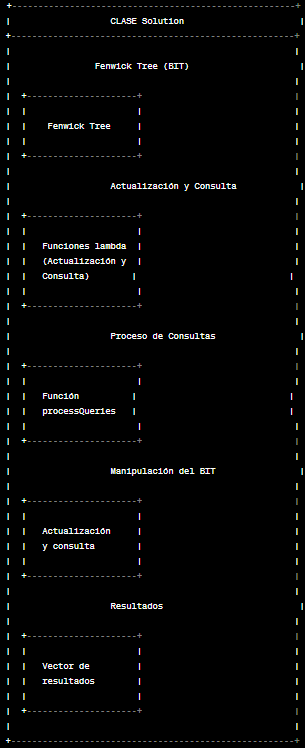
**Enunciado:**

**Código:**

|  |
| --- |
| **class Solution {**  **public:**  **vector<int> processQueries(vector<int>& queries, int m) {**  **int n = queries.size();**  **vector<int> result(n);**  **// Definir el tamaño del BIT como m+n+1, porque almacenaremos los números a partir del índice m+1**  **vector<int> BIT(m+n+1, 0);**  **// Función para actualizar el BIT**  **auto update = [&](int i, int val) {**  **for (; i < BIT.size(); i += i&-i)**  **BIT[i] += val;**  **};**  **// Función para obtener la suma del prefijo**  **auto getSum = [&](int i) {**  **int sum = 0;**  **for (; i > 0; i -= i&-i)**  **sum += BIT[i];**  **return sum;**  **};**  **vector<int> pos(m+1);**  **// Inicializar BIT y pos**  **for (int i = 1; i <= m; i++) {**  **pos[i] = i + n;**  **update(pos[i], 1);**  **}**  **for (int i = 0; i < n; i++) {**  **result[i] = getSum(pos[queries[i]]) - 1;**  **update(pos[queries[i]], -1);**  **pos[queries[i]] = n - i;**  **update(pos[queries[i]], 1);**  **}**    **return result;**  **}**  **};** |

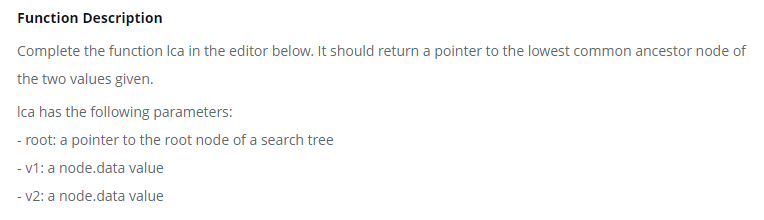
**Captura: Explicación:**

La clase Solution utiliza un fenwick tree (o BIT) para resolver consultas en un vector. El BIT se actualiza y consulta con funciones lambda, y una función processQueries procesa las consultas dadas, modificando el BIT y registrando los resultados. Cada consulta mueve un elemento al frente de la lista, y el BIT ayuda a rastrear las posiciones. La función retorna los resultados como un vector, indicando el índice inicial de cada consulta antes del movimiento. Esta solución proporciona un manejo eficiente de las consultas en escenarios de actualizaciones frecuentes.

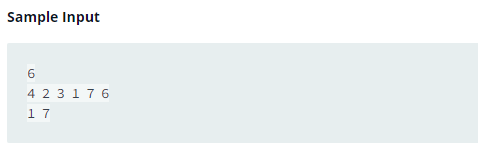
****

**Pregunta 7: (Árboles Ternarios (Tree)):**

**Enunciado:**

****

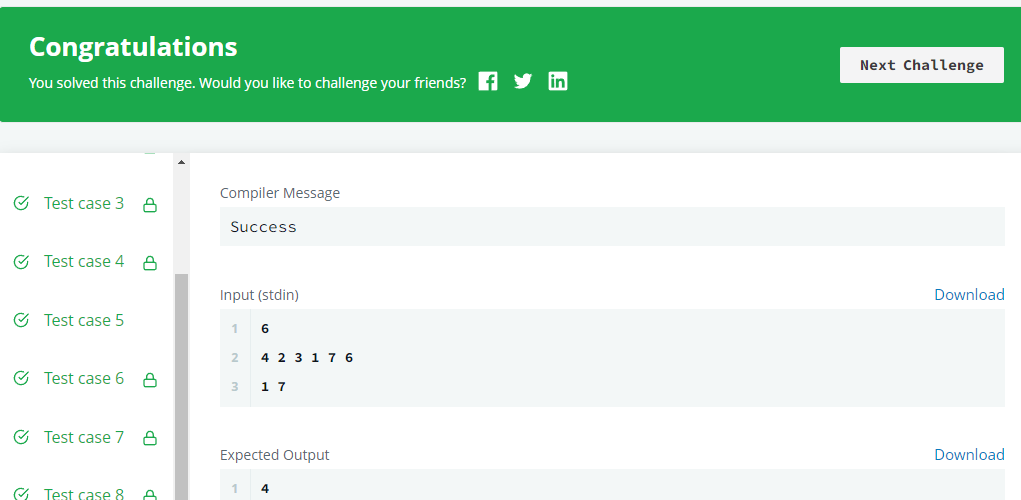
**Entrada:**

****

**Código:**

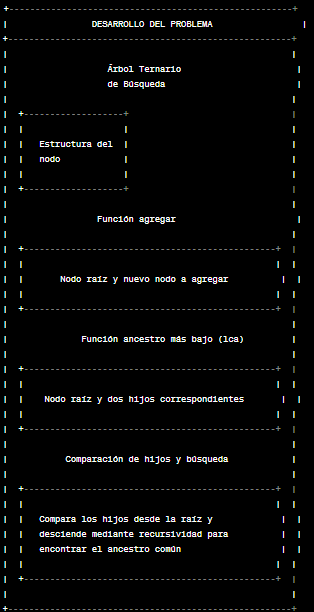
|  |
| --- |
| #include<iostream>  using namespace std;  class Node {  public:  int data;  Node\* left;  Node\* right;  Node\* equ;  Node(int d) {  data = d;  left = equ = right = NULL;  }  };  class Solution {  public:  Node\* insert(Node\* root, int data) {  if (root == NULL) {  return new Node(data);  }  else {  Node\* cur;  if (data <= root->data) {  cur = insert(root->left, data);  root->left = cur;  }  else if (data == root->data) {  cur = insert(root->equ, data);  root->equ = cur;  }  else if (data <= root->data) {  cur = insert(root->right, data);  root->right = cur;  }  return root;  }  }  Node\* lca(Node\* root, int v1, int v2) {  if (root == nullptr) return nullptr;  int data = root->data;  if (v1 < data && v2 < data) return lca(root->left, v1, v2);  if (v1 > data && v2 > data) return lca(root->right, v1, v2);  return root;  }  }; |

**Captura:**

****

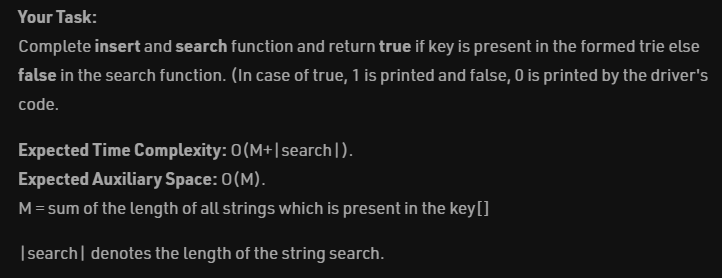
**Explicación:**

El desarrollo del problema hizo uso del árbol ternario de búsqueda, como primer paso se crea la estructura del nodo para luego crear la función de agregar y la del ancestro más bajo(lca). En la función lca pide como parámetros el nodo raíz y los dos hijos correspondientes, compara estos hijos comenzando desde la raíz y usando la recursividad va descendiendo por el árbol hasta encontrar el ancestro en común.

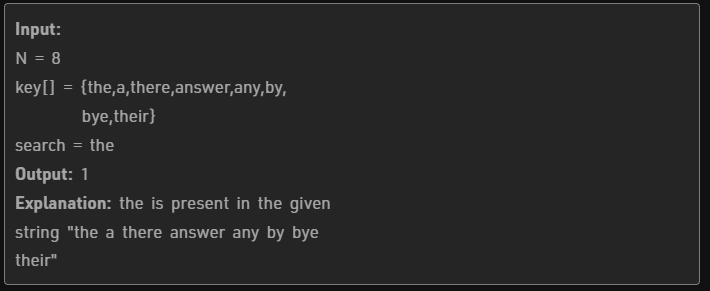
****

**Pregunta 8: (Trie):**

**Enunciado:**

****

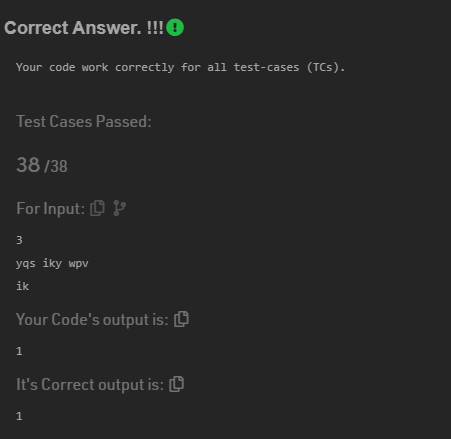
**Entrada:**

****

**Código:**

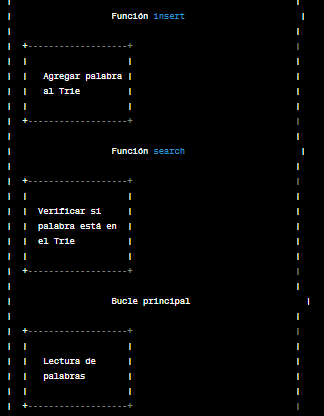
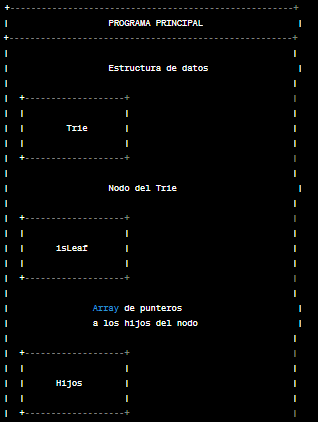
|  |
| --- |
| using namespace std;  #define ALPHABET\_SIZE (26)  #define CHAR\_TO\_INDEX(c) ((int)c - (int)'a')  struct TrieNode {  struct TrieNode \*children[ALPHABET\_SIZE];  // isLeaf is true if the node represents  // end of a word  bool isLeaf;  };  struct TrieNode \*getNode(void) {  struct TrieNode \*pNode = NULL;  pNode = (struct TrieNode \*) malloc(sizeof(struct TrieNode));  if (pNode) {  int i;  pNode->isLeaf = false;  for (i = 0; i < ALPHABET\_SIZE; i++) pNode->children[i] = NULL;  }  return pNode;  }  void insert(struct TrieNode \*, string);  bool search(struct TrieNode \*, string);  int main() {  int t;  cin >> t;  while (t--) {  int n;  cin >> n;  string keys[n];  for (int i = 0; i < n; i++) {  cin >> keys[i];  }  struct TrieNode \*root = getNode();  for (int i = 0; i < n; i++) insert(root, keys[i]);  string abc;  cin >> abc;  if (search(root, abc))  cout << "1\n";  else  cout << "0\n";  }  return 0;  }  void insert(struct TrieNode \*root, string key)  {  struct TrieNode\* var = root;  for (int i = 0; i < key.length(); i++) {  int index = key[i] - 'a';  if (!var->children[index])  var->children[index] = getNode();  var = var->children[index];  var->isLeaf = true;  }  }  bool search(struct TrieNode \*root, string key)  {  struct TrieNode\* var = root;  for (int i = 0; i < key.length(); i++) {  int index = key[i] - 'a';  if (!var->children[index])  return false;  var = var->children[index];  }  return(var->isLeaf);  } |

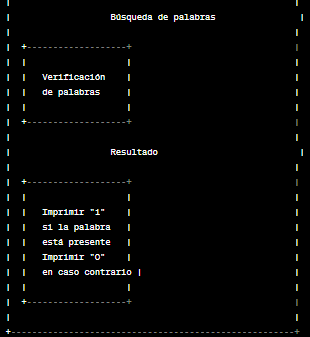
**Captura:**

****

**Explicación:**

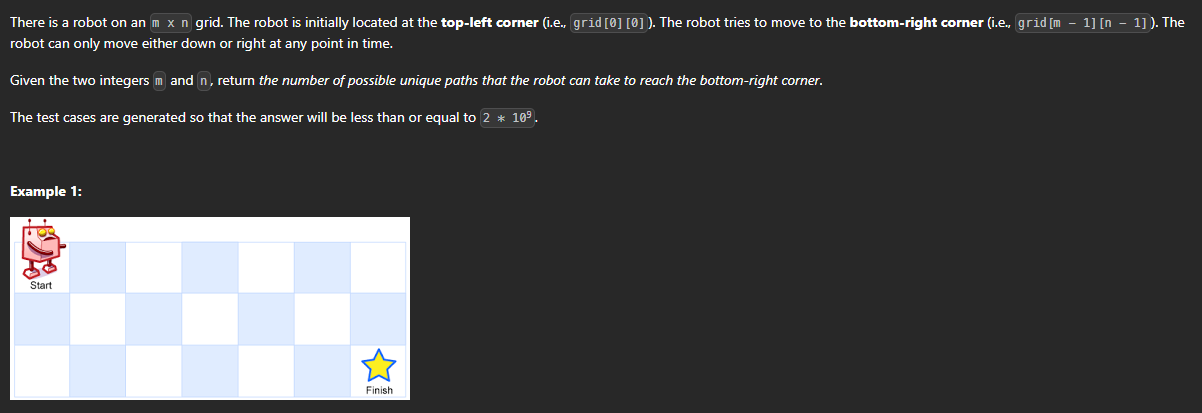
Este programa implementa una estructura de datos de Trie para realizar operaciones eficientes de inserción y búsqueda de palabras. Un nodo de Trie contiene una marca isLeaf para indicar el final de una palabra y un array de punteros a sus hijos. La función insert agrega una palabra al Trie, creando nodos como sea necesario, y la función search verifica si una palabra está en el Trie, recorriendo los nodos correspondientes a los caracteres de la palabra. En main, el programa recibe varias palabras, las inserta en el Trie y luego realiza una búsqueda, imprimiendo "1" si la palabra está presente y "0" en caso contrario.



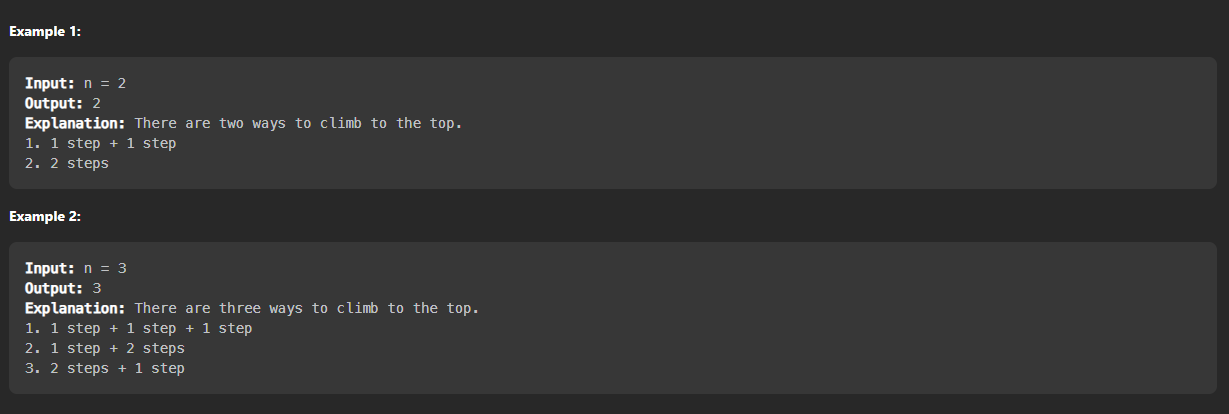


**Pregunta 9: (Programación Dinámica):**

**Enunciado:**

****

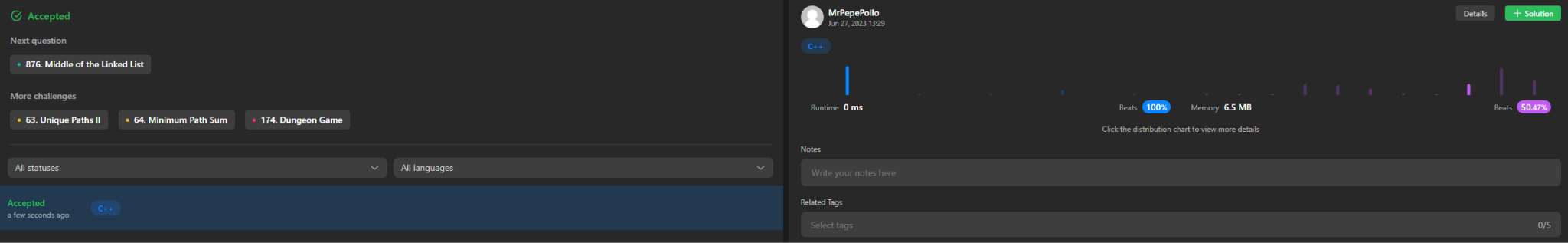
**Entrada:**

****

**Código:**

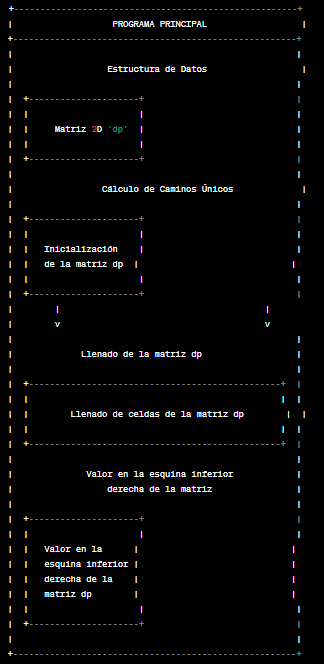
|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  int uniquePaths(int m, int n) {  vector<vector<int>> dp(m, vector<int>(n, 0));  for (int i = 0; i < m; i++) {  dp[i][0] = 1;  }    for (int j = 0; j < n; j++) {  dp[0][j] = 1;  }    for (int i = 1; i < m; i++) {  for (int j = 1; j < n; j++) {  dp[i][j] = dp[i - 1][j] + dp[i][j - 1];  }  }  return dp[m - 1][n - 1];  }  int main() {  int m, n;  cin >> m;  cin >> n;  int result = uniquePaths(m, n);  cout << result << endl;  return 0;  } |

**Captura:**

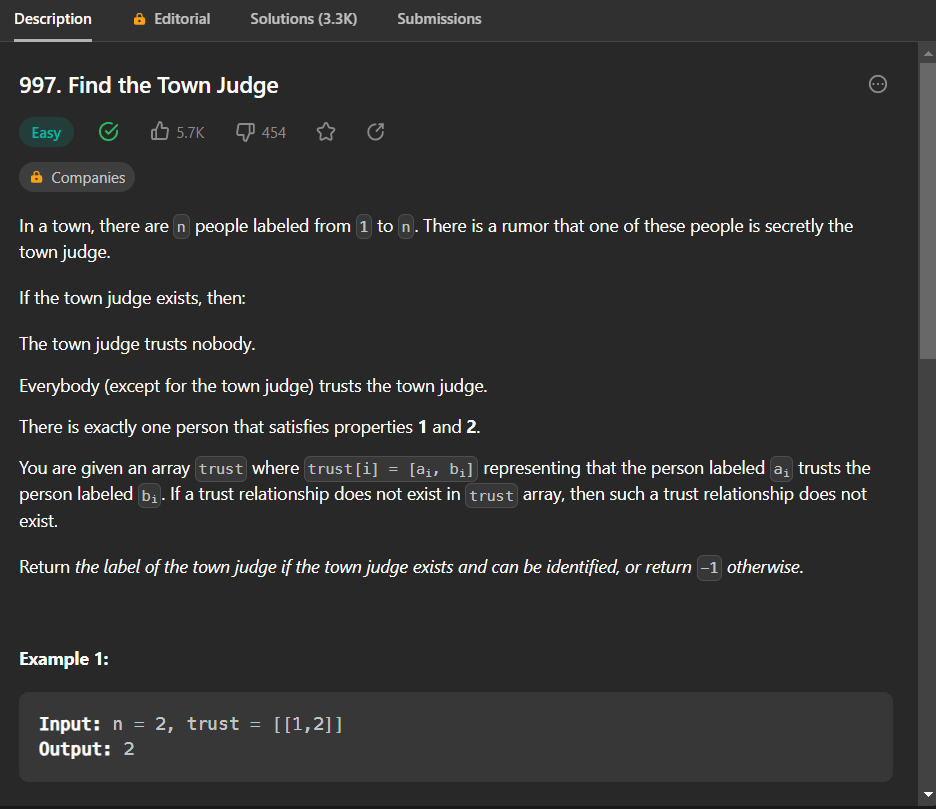
****

**Explicación:**

Este programa utiliza programación dinámica para calcular el número de caminos únicos de la esquina superior izquierda a la esquina inferior derecha de una cuadrícula de m\*n. Se crea una matriz 2D 'dp' donde dp[i][j] representa el número de caminos únicos para llegar a la celda (i, j). Todos los elementos de la primera fila y la primera columna se inicializan a 1 (solo hay un camino a lo largo de la fila o columna). Luego, cada celda restante se llena sumando los caminos desde la celda de arriba (dp[i-1][j]) y la celda de la izquierda (dp[i][j-1]). Finalmente, se devuelve el valor en la esquina inferior derecha.

****

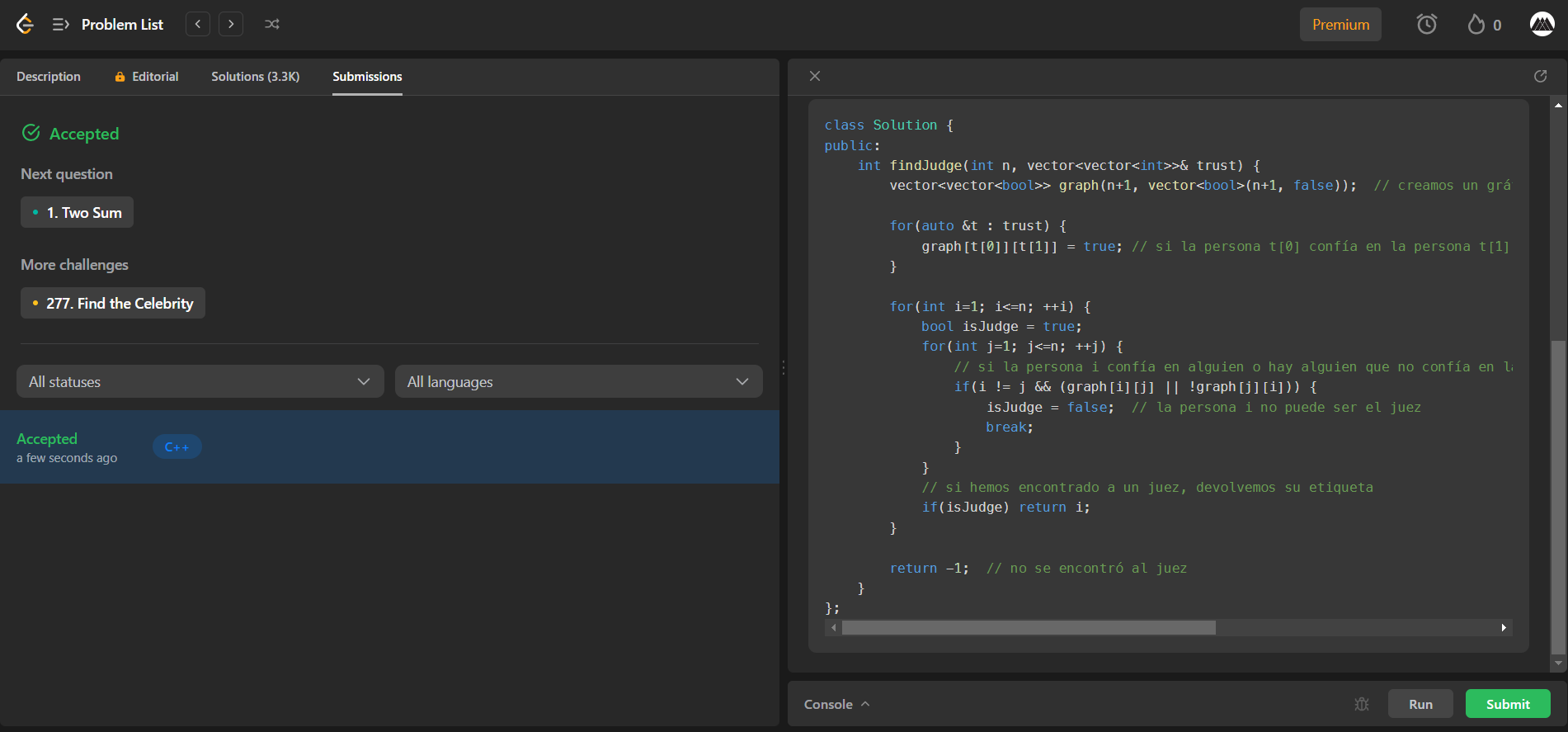
**Pregunta 10: (Grafos):**

**Enunciado:**

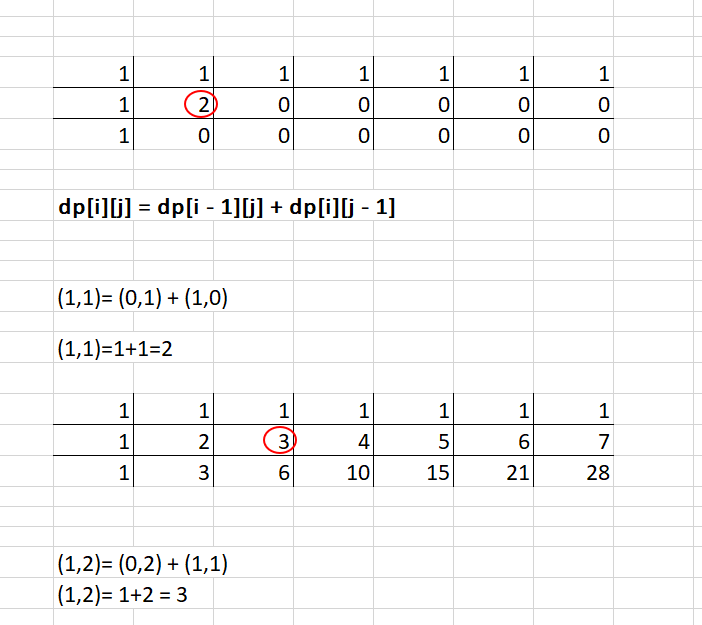
**Código:**

|  |
| --- |
| **class Solution {**  **public:**  **int findJudge(int n, vector<vector<int>>& trust) {**  **vector<vector<bool>> graph(n+1, vector<bool>(n+1, false)); // creamos un gráfico utilizando una matriz de adyacencia**    **for(auto &t : trust) {**  **graph[t[0]][t[1]] = true; // si la persona t[0] confía en la persona t[1], marcamos la celda correspondiente en el gráfico como verdadera**  **}**  **for(int i=1; i<=n; ++i) {**  **bool isJudge = true;**  **for(int j=1; j<=n; ++j) {**  **// si la persona i confía en alguien o hay alguien que no confía en la persona i**  **if(i != j && (graph[i][j] || !graph[j][i])) {**  **isJudge = false; // la persona i no puede ser el juez**  **break;**  **}**  **}**  **// si hemos encontrado a un juez, devolvemos su etiqueta**  **if(isJudge) return i;**  **}**  **return -1; // no se encontró al juez**  **}**  **};** |

**Captura:**

****

**Explicación:**

La clase Solution implementa un método para encontrar un "juez" en un grupo de personas basándose en la confianza. Se crea un gráfico de adyacencia para representar a quién confía cada persona. Luego se itera sobre cada persona: si alguien confía en la persona o la persona no confía en todos los demás, esa persona no puede ser el juez. Si una persona cumple con las condiciones de juez (todos confían en ella y ella no confía en nadie), se devuelve su etiqueta. Si no se encuentra un juez, se devuelve -1.

1. **EVALÚA UNA SOLUCIÓN BASADA EN COMPUTACIÓN PARA CUMPLIR CON EL CONJUNTO DE REQUERIMIENTOS EN EL CONTEXTO DE SISTEMAS DE INFORMACIÓN**

* El trabajo final aporto a la competencia abet ya que ayudo a mejorar mi capacidad critica para evaluar diferentes escenarios y en base a esto diseñar una solución usando las metodologías enseñadas en el curso cumpliendo con las restricciones establecidas.

1. **CONCLUSIONES**

* Finalmente, se puede concluir que gracias a las tan variadas estructuras de datos y arboles que se pueden usar en la programación, se nos permite hallar soluciones usando la programación a diferentes problemas enfocados en diferentes ámbitos y sectores.

1. **BIBLIOGRAFÍA**

**Binary Indexed Tree (Fenwick)**

[**https://leetcode.com/problems/queries-on-a-permutation-with-key**](https://leetcode.com/problems/queries-on-a-permutation-with-key)

**KMP**

[**https://www.hackerrank.com/challenges/kmp-problem/problem**](https://www.hackerrank.com/challenges/kmp-problem/problem)

**Grafos**

[**https://leetcode.com/problems/find-the-town-judge/**](https://leetcode.com/problems/find-the-town-judge/)

**Segment Tree:**

[**https://leetcode.com/problems/my-calendar-i/**](https://leetcode.com/problems/my-calendar-i/)

**programacion dinamica:**

[**https://leetcode.com/problems/unique-paths/description/**](https://leetcode.com/problems/unique-paths/description/)

**algortimo z:**

[**https://codeforces.com/contest/126/problem/B**](https://codeforces.com/contest/126/problem/B)

**set:**

[**https://www.hackerrank.com/challenges/cpp-sets/problem**](https://www.hackerrank.com/challenges/cpp-sets/problem)

**map:**

[**https://www.hackerrank.com/challenges/cpp-maps/problem**](https://www.hackerrank.com/challenges/cpp-maps/problem)

**arbol ternario:**

[**https://www.hackerrank.com/challenges/binary-search-tree-lowest-common-ancestor/problem?isFullScreen=true**](https://www.hackerrank.com/challenges/binary-search-tree-lowest-common-ancestor/problem?isFullScreen=true)

**trie:**

[**https://practice.geeksforgeeks.org/problems/trie-insert-and-search0651/1?utm\_source=geeksforgeeks&utm\_medium=article\_practice\_tab&utm\_campaign=article\_practice\_tab**](https://practice.geeksforgeeks.org/problems/trie-insert-and-search0651/1?utm_source=geeksforgeeks&utm_medium=article_practice_tab&utm_campaign=article_practice_tab)