

**INFORME DE EVALUACIÓN DE DESEMPEÑO**

CURSO DE FUNDAMENTOS DE PROGRAMACION COMPETITIVA – CC217

Carrera de Ciencias de la Computación

Sección: CC51

Alumno:

U202115065 - Sovero Cubillas, John Davids

Junio 2023

Contenido

[1. RESUMEN 3](#_Toc138966846)

[2. OBJETIVO DEL ESTUDIANTE 3](#_Toc138966847)

[3. DISEÑA UNA SOLUCIÓN BASADA EN COMPUTACIÓN PARA CUMPLIR CON EL CONJUNTO DE REQUERIMIENTOS EN EL CONTEXTO DE SISTEMAS DE INFORMACIÓN 3](#_Toc138966848)

[4. IMPLEMENTA UNA SOLUCIÓN BASADA EN COMPUTACIÓN PARA CUMPLIR CON EL CONJUNTO DE REQUERIMIENTOS EN EL CONTEXTO DE SISTEMAS DE INFORMACIÓN 4](#_Toc138966849)

[5. EVALÚA UNA SOLUCIÓN BASADA EN COMPUTACIÓN PARA CUMPLIR CON EL CONJUNTO DE REQUERIMIENTOS EN EL CONTEXTO DE SISTEMAS DE INFORMACIÓN 44](#_Toc138966850)

[6. CONCLUSIONES 44](#_Toc138966851)

[7. RECOMENDACIONES 44](#_Toc138966852)

[8. ANEXOS 45](#_Toc138966853)

[9. BIBLIOGRAFÍA 45](#_Toc138966854)

1. RESUMEN

El contexto de este trabajo final se sitúa en el curso de Fundamentos de Programación Competitiva impartido para la carrera de Ciencias de la Computación por el profesor Peter Montalvo García en la sección CC51.

La problemática real que se aborda para el desarrollo de este trabajo final es el desarrollo de 10 problemas en total, extraídos de aplicaciones de programación competitiva, como HackerRank, Codeforces, Online Judge o LeetCode. Además, cada problema aborda los siguientes tipos de algoritmos: Map, Set, algoritmo KMP y Z, Segment Tree, Árbol Fenwick, Árboles Ternarios (Tree), Trie, Programación Dinámica y Grafos. Asimismo, estos problemas tienen un nivel de complejidad adecuado y sus soluciones fueron sustentadas mediante gráficos y herramientas que permitieron la comprensión del código.

1. OBJETIVO DEL ESTUDIANTE

He logrado el *student* *outcome* de la carrera de Ciencias de la Computación diseñando diversas soluciones a los 10 problemas de programación competitiva, implementándolas en el lenguaje de programación C++ con diversos algoritmos y enfoques y evaluando su solución, al enviarlas en las diferentes plataformas de programación competitiva y recibiendo aceptado como respuesta del juez, es decir, que cumple con todas las restricciones y por lo tanto es una solución válida al problema.

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

Mayormente los objetivos de los problemas encontrados eran hallar el máximo y mínimo de movimientos, averiguar si en una oración existen patrones de palabras, responder diversas consultas de cuanto suman ciertos elementos de un arreglo, averiguar cuantas coordenadas son posibles encontrar en un conjunto de segmentos brindados, encontrar el máximo árbol binario a partir de un arreglo de números, averiguar cuales son las palabras mas comunes en un arreglo de palabras, encontrar la mayor palabra palindrómica en una cadena y clonar un grafo mediante una lista de adyacencia.

Las herramientas que se utilizaron para extraer los problemas del trabajo Final son las siguientes páginas web:

* <https://codeforces.com/>
* <https://leetcode.com/>
* <https://atcoder.jp/>

Para el desarrollo de estos problemas se utilizó el editor de código Visual Studio Code y el copilador MinGW.

Plan de actividades:

|  |  |  |  |
| --- | --- | --- | --- |
| N° | Fecha | Autores | Descripción de actividades |
| 1 | 24/06/2023 | John Davids Sovero Cubillas | Búsqueda de los problemas a realizarse y desarrollo del ejercicio Set |
| 2 | 24/06/2023 | Luis Felipe Poma Astete | Desarrollo de los ejercicios Map, KMP y Arboles Ternarios |
| 3 | 25/06/2023 | John Davids Sovero Cubillas | Desarrollo de los ejercicios Segment Tree, Árbol de Fenwick y creación de gráficos para explicación de código |
| 4 | 25/06/2023 | Luis Felipe Poma Astete | Desarrollo de los ejercicios Trie, Programación Dinámica, Grafos y creación de gráficos para explicación de código |

1. IMPLEMENTA UNA SOLUCIÓN BASADA EN COMPUTACIÓN PARA CUMPLIR CON EL CONJUNTO DE REQUERIMIENTOS EN EL CONTEXTO DE SISTEMAS DE INFORMACIÓN
2. Map (<https://atcoder.jp/contests/arc087/tasks/arc087_a>)

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

|  |
| --- |
| #include <bits/stdc++.h> using namespace std;  int main() {       *// Declaramos la variables a utilizar*       int n; *// numero de elementos*       int changes = 0; *// número de cambios necesarios*       map<int, int> numbers; *// map donde se guardará la frecuencia de cada número*             cin >> n;        *// While para leer los elementos y guardarlos en el map*       while (n--)       {             *// Se lee la variable*             int x;             cin >> x;              *// Se aumenta la frecuencia del numero en 1*             numbers[x] += 1;              *// Si la frecuencia es mayor al numero es necesario un cambio*             if(numbers[x] > x) {                   changes++;                   numbers[x] -= 1;             }       }        *// for para obtener los cambios que se necesitaran realizar*       for (auto i: numbers){             *// si el numero es menor que la frecuencia sera necesario n cambios*             if (i.first > i.second){                   changes += i.second;             }       }        *// Se imprime en pantalla los cambios*       cout<<changes;       return 0; } |

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente con confianza media

Interfaz de usuario gráfica, Aplicación

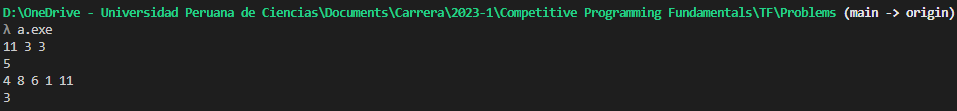
Descripción generada automáticamente

1. Set ([http://codeforces.com/problemset/problem/567/D](https://codeforces.com/problemset/problem/567/D))

Imagen que contiene Texto

Descripción generada automáticamente

|  |
| --- |
| #include <iostream> #include <cstdio> #include <set>  using namespace std; int n, m, q, len, sum; set<int> pos;  int main() {     cin >> n >> m >> len >> q;     pos.insert(0);     pos.insert(n + 1);     *// cantidad de barcos máximos que entran en el tablero*     sum = (n + 1) / (len + 1);      for (int i = 1; i <= q; ++i)     {         int x, l, r;         cin >> x;         *// Si la bala actual ya existe, continua a la siguiente iteracion*         if (pos.find(x) != pos.end())             continue;         *// Se obtiene la posicion de la bala mas cercana por encima de la actual*         auto it = pos.upper\_bound(x);         *// Se obtiene la posicion de la bala mas cercana por debajo de la actual*         l = \*prev(it);         r = \*it;         *// Restamos a la cantidad de barcos maximos los barcos que entran fuera de los lugares donde cayeron las balas*         sum -= (r - l) / (len + 1);         *// Sumamos la cantidad de barcos que entran en el rango entre la bala minima - actual y bala actual - maxima*         sum += (x - l) / (len + 1) + (r - x) / (len + 1);         *// Insertamos la posicion de la bala actual, se usa set en caso se dispare a la misma posicion varias veces*         pos.insert(x);         *// Si sum es menor que la cantidad de barcos maximos que deberian entrar en el tablero, se hizo trampa*         if (sum < m)         {             cout << i << '\n';             return 0;         }     }     *// No se hizo trampa*     cout << "-1\n";     return 0; } |



Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

1. Algoritmo KMP ([SPOJ.com - Problem NAJPF](https://www.spoj.com/problems/NAJPF/))

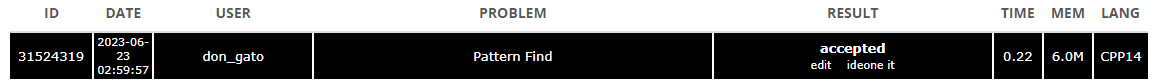
Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

|  |
| --- |
| #include <bits/stdc++.h> using namespace std;  *// Funcion para hallar el LPS del patron* vector<int> compute\_failure(const string &pattern) {       int n = pattern.size();       vector<int> failure(n);       failure[0] = 0;       int i = 1;       int pos = 0;       *// Recorremos la subcadena*       while (i < n)       {             if (pattern[i] == pattern[pos])             {                   pos++;                   failure[i] = pos;                   i++;             }             else             {                   if (pos != 0)                   {                         pos = failure[pos - 1];                   }                   else                   {                         failure[i] = 0;                         i++;                   }             }       }        *// Devolvemos la tabla de fallas*       return failure; }  *// Esta función busca la subcadena en la cadena de texto* vector<int> kmp(const string &text, const string &pattern) {       vector<int> matches;       int n = text.size();       int m = pattern.size();       *// Construimos la tabla de fallas para la subcadena*       vector<int> failure = compute\_failure(pattern);       int j = 0;       int i = 0;        *// Recorremos la cadena de texto*       while (i < n)       {             if (pattern[j] == text[i])             {                   j++;                   i++;             }             if (j == m)             {                   matches.push\_back(i - j);                   j = failure[j - 1];             }             else if (i < n && pattern[j] != text[i])             {                   if (j != 0)                   {                         j = failure[j - 1];                   }                   else                         i++;             }       }       *// Devolvemos las posiciones donde se encuentra la subcadena*       return matches; } int main() {       *// Se lee el numero de casos*       int t;       cin >> t;        *// Se recorre cada caso*       while (t--)       {             *// Se lee el texto y el patron*             string text, pattern;             cin >> text >> pattern;             *// Se crea un vector donde se almacenara los indices donde esta el patron*             vector<int> matches = kmp(text, pattern);              *// Si el tamaño del vector es 0 entondes no encontro coincidencias*             if (matches.size() == 0)             {                   cout << "Not Found";             }             else             {                   *// Se imprime el numero de coincidencias*                   cout << matches.size() << endl;                   for (auto i : matches)                   {                         *// Se imprime el indice de la coincidencia*                         cout << i + 1<< " ";                   }             }             cout << "\n";       }       return 0; } |

Texto

Descripción generada automáticamente



1. Algoritmo Z (<https://codeforces.com/contest/126/problem/B>)

Imagen que contiene Tabla

Descripción generada automáticamente

|  |
| --- |
| #include <bits/stdc++.h> using namespace std;  *// Función para calcular el arreglo Z* vector<int> calculateZ(const string &s) {       int n = s.length();       vector<int> z(n, 0); *// Inicializar el arreglo Z con ceros*       for (int i = 1, l = 0, r = 0; i < n; ++i)       {             if (i <= r)                   z[i] = min(r - i + 1, z[i - l]); *// Caso 1: i está dentro del rango de una subcadena ya procesada*             while (s[z[i]] == s[z[i] + i] && z[i] + i < n)                   ++z[i]; *// Caso 2: extender la subcadena a la derecha*             if (z[i] + i - 1 > r)                   r = z[i] + i - 1, l = i; *// Actualizar los límites de la subcadena procesada más a la derecha*       }       return z; }  void findSubstring(const string &s) {       int n = s.length();       vector<int> z = calculateZ(s); *// Obtener el arreglo Z*       vector<int> x(n, 0);           *// Arreglo para almacenar la longitud máxima de un prefijo de un sufijo en cada posición*       vector<int> y(n, 0);           *// Arreglo para almacenar la segunda longitud máxima de un prefijo de un sufijo en cada posición*       int max1 = 0, max2 = 0;        *// Variables para almacenar las longitudes máximas*        *// Calcular los valores máximos max1 y max2*       for (int i = n - 1; i > -1; --i)       {             if (z[i] == n - i)             {                   if (z[i] > max1)                         max2 = max1, max1 = z[i];                   else if (z[i] > max2)                         max2 = z[i];             }             x[i] = max1;             y[i] = max2;       }        *// Se verifica si los maximos no son iguales a 0*       if (max1 == 0 && max2 == 0)       {             puts("Just a legend");             return;       }        n = n - 1;        int ans = 0; *// Almacenar la longitud máxima de la subcadena t*        *// Verificar si existe una subcadena t que cumpla las condiciones y encontrar la longitud máxima*       for (int i = 1; i < n; ++i)       {             if (z[i] == n - i + 1)             {                   if (y[i] > ans)                         ans = y[i]; *// Si la longitud y[i] es mayor que ans, actualizar ans*             }             else             {                   int s = min(x[i], min(z[i], x[n - z[i] + 1])); *// Obtener el mínimo entre tres valores*                   if (s > ans)                         ans = s; *// Si s es mayor que ans, actualizar ans*             }       }        *// Si no se encontró una subcadena adecuada, imprimir "Just a legend" y devolver el arreglo Z*       if (ans == 0)       {             puts("Just a legend");             return;       }        *// Imprimir la subcadena t de longitud ans*       for (int i = 0; i < ans; ++i)             putchar(s[i]); }  int main() {       string s;       cin >> s;       findSubstring(s);       return 0; } |

Texto

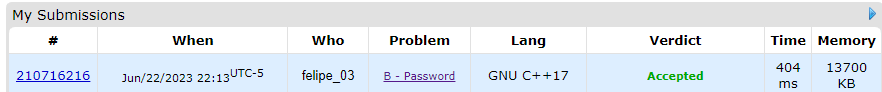
Descripción generada automáticamente

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Texto

Descripción generada automáticamente



1. Segment Tree (<https://leetcode.com/problems/range-sum-query-mutable/description/>)

Texto

Descripción generada automáticamente

|  |
| --- |
| #include <iostream> #include <vector>  using namespace std;  *// Estructura del árbol de segmentos* struct SegmentTree {     vector<int> tree; *// Array para almacenar el árbol*     int size;         *// Tamaño del árbol*      *// Constructor*     SegmentTree(int n)     {         *// Calcula el tamaño del árbol basado en el número de elementos*         size = 1;         while (size < n)             size \*= 2;         tree.resize(2 \* size); *// Redimensiona el árbol*     }      *// Actualiza el valor del elemento en la posición index con el valor val*     void update(int index, int val)     {         index += size;     *// Calcula la posición en el array del árbol*         tree[index] = val; *// Actualiza el valor del nodo hoja*          *// Propaga los cambios hacia arriba*         while (index > 1)         {             index /= 2;             tree[index] = tree[index \* 2] + tree[index \* 2 + 1];         }     }      *// Calcula la suma de valores en el rango [left, right)*     int query(int left, int right, int node, int nodeLeft, int nodeRight)     {         *// Si el rango está completamente fuera del rango del nodo actual*         if (nodeRight <= left || right <= nodeLeft)             return 0;          *// Si el rango está completamente cubierto por el nodo actual*         if (left <= nodeLeft && nodeRight <= right)             return tree[node];          *// Divide el rango en dos y realiza las consultas recursivas*         int mid = (nodeLeft + nodeRight) / 2;         int sumLeft = query(left, right, node \* 2, nodeLeft, mid);         int sumRight = query(left, right, node \* 2 + 1, mid, nodeRight);          return sumLeft + sumRight;     }      *// Calcula la suma de valores en el rango [left, right)*     int query(int left, int right)     {         return query(left, right, 1, 0, size);     } };   class NumArray { private:     SegmentTree st; public:        NumArray(vector<int>& nums): st(nums.size()) {          if(nums.size() > 0){             int n = nums.size();             for (int i = 0; i < n; i++)             {                 st.update(i, nums[i]);             }         }     }          void update(int index, int val) {         st.update(index, val);     }          int sumRange(int left, int right) {         return st.query(left, right+1);     } }; |

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

.

**Representación**

Diagrama

Descripción generada automáticamente

**Update:**

Diagrama

Descripción generada automáticamente

1. Árbol Fenwick (<https://codeforces.com/contest/369/problem/E> )

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

|  |
| --- |
| #include <bits/stdc++.h> using namespace std;  const int N = 3e5 + 10, M = 1e6 + 5; int lst[N], fen[M], ans[N], L[M]; vector<int> R[M], p[M];  int obtenerSiguiente(int index) {     *// index - LSB(index)*     return index + (index & -index); *// obtener el bit menos significativo (LSB)* }  void actualizar(int index, int val) {     while (index < M)     {         fen[index] += val;         index = obtenerSiguiente(index);     } }   int obtenerSuma(int ind) {     int res = 0;     for (; ind > 0; ind -= (ind & (-ind)))     {         res += fen[ind];     }     return res; }  int main() {     *// Para optimizar la entrada de variables*     ios::sync\_with\_stdio(0), cin.tie(0), cout.tie(0);         *// Obtenemos la cantidad de segmentos y querys*     int n, m;     cin >> n >> m;      for (int i = 1; i <= n; i++)     {         *// Obtenemos la posicion mas hacia la izquierda y derecha de cada segmento*         int l, r;         cin >> l >> r;         *// Aumentamos en 1 la posición mas hacia la izquierda del segmento en un arreglo*         L[l]++;         *// Insertamos en la posicion mas hacia la derecha del segmento, el valor de la posición mas hacia la izquierda del segmento*         R[r].push\_back(l);     }     for (int i = 1; i <= m; i++)     {         *// Obtenemos la cantidad de coordenadas a consultarse*         int cnt;         cin >> cnt;         for (int j = 1, a; j <= cnt; j++)         {             *// Para cada coordenada a consultarse guardamos en un arreglo el numero de query en el que se realizó*             cin >> a;             p[a].push\_back(i);         }     }      for (int i = 1; i < M; i++)     {         *// Insertamos en el arbol de fenwick la posicion mas hacia la izquierda de cada segmento, en caso no exista se inserta 0*         actualizar(i, L[i]);          for (auto x : p[i])         {             *// Aumentamos a un arreglo de respuestas la cantidad de veces que la coordenada está en el rango de los segmentos*             ans[x] += obtenerSuma(i) - obtenerSuma(lst[x]);             *// Guardamos la mayor coordenada consultada hasta el momennto que está en uno de los segmentos, para evitar que se repita en un segmento ya consultado*             lst[x] = i;         }         *// Si ya se consutó la posición mas a la derecha de un segmento, se elimina del arbol de fenwick este segmento*         for (auto x : R[i])         {             actualizar(x, -1);         }     }     *// Mostramos las respuestas para cada query*     for (int i = 1; i <= m; i++)     {         cout << ans[i] << endl;     } } |

**Ejemplo de codeforces:**

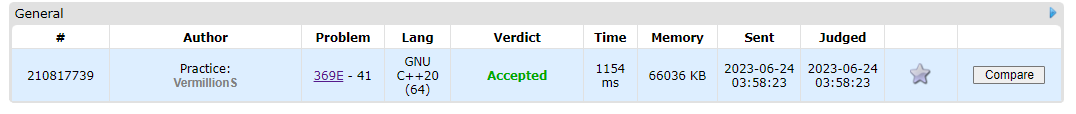
**Texto

Descripción generada automáticamente**

**Ejemplo para explicación de código:**

Texto

Descripción generada automáticamente



**Ingreso del 1:**

Diagrama

Descripción generada automáticamente

**Ingreso del 3:**

Diagrama

Descripción generada automáticamente

**Consulta del 3:**

sum(3) - sum(0) = (1 +1) - 0 = 2 veces puede estar en los segmentos la coordenada 3

Limite derecho = 3

**Consulta del 4:**

sum(4) - sum(**3**) = 2 - 2 = 0 veces puede estar en los segmentos la coordenada 4, porque ya está siendo ocupado por el 3

Como 4 es el fin de rango de un segmento se quita del árbol de fenwick la coordenada izquierda ingresada, que es el 3:

Diagrama

Descripción generada automáticamente

**Se agrega el 5:**

Por ser el inicio de un nuevo segmento se agrega el 5 y se propaga por todo el árbol de fenwick

**Diagrama

Descripción generada automáticamente**

**Consulta del 5:**

sum(5) - sum(3) = 2 - 1 = 1 vez puede estar en los segmentos, en este caso en el segmento 5 - 6, porque el 1 - 5 ya está siendo ocupado por la coordenada 3 de su consulta.

Como 5 es el fin de rango de un segmento se quita del árbol de fenwick la coordenada izquierda ingresada que es el 1 y se propaga:

Diagrama

Descripción generada automáticamente

Como 6 es el fin de rango de un segmento se quita del árbol de fenwick la coordenada izquierda ingresada que es el 5 y se propaga:

Diagrama

Descripción generada automáticamente

**Consulta del 8:**

sum(0) - sum(0) = 0 - 0 = 0 veces puede estar en los segmentos

1. Árboles Ternarios (Tree) ([Maximum Binary Tree - LeetCode](https://leetcode.com/problems/maximum-binary-tree/description/) )

Texto

Descripción generada automáticamente

|  |
| --- |
| */\*\*  \* Definition for a binary tree node.  \* struct TreeNode {  \*     int val;  \*     TreeNode \*left;  \*     TreeNode \*right;  \*     TreeNode() : val(0), left(nullptr), right(nullptr) {}  \*     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}  \*     TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}  \* };  \*/ class Solution { public:       // Funcion para hallar el indice del mayor elemento en un vector       int findIndexMax(const vector<int> &vector)       {             auto maxElement = max\_element(vector.begin(), vector.end());             return distance(vector.begin(), maxElement);       }        // Funcion principal recursiva para la construccion del arbol       // (Esta funcion se llama recursivamente para ir guardando el mayor valor del arreglo de cada rama)       TreeNode \*constructMaximumBinaryTree(vector<int> &nums)       {             // En caso envien un vector vacio             if (nums.empty())                   return nullptr;              // Se obtiene el indice del maximo elemento en el vector             int indexMax = findIndexMax(nums);              // Se ingresa los valores de la izquiera un arreglo left             vector<int> leftarr;             for (int i = 0; i < indexMax; i++)                   leftarr.push\_back(nums[i]);              // Se ingresa los valores de la derecha un arreglo right             vector<int> rightarr;             for (int i = indexMax + 1; i < nums.size(); i++)                   rightarr.push\_back(nums[i]);              // Se crea el nodo donde se guardaran los vectores left y right             TreeNode \*node = new TreeNode(nums[indexMax]);              // Se guarda a la izquierda del arbol el vector left             node->left = constructMaximumBinaryTree(leftarr);              // Se guarda a la right del arbol el vector right             node->right = constructMaximumBinaryTree(rightarr);              return node;       } };* |

Patrón de fondo

Descripción generada automáticamente

Patrón de fondo

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

1. Trie ([Top K Frequent Words - LeetCode](https://leetcode.com/problems/top-k-frequent-words/))

Texto

Descripción generada automáticamente

|  |
| --- |
| class Solution { public:     *// Se crea la clase de TrieNode*     class TrieNode     {     public:         unordered\_map<char, TrieNode \*> children; *// map donde se guardaran los demas TrieNode\**         int count;                                *// contador de frecuencias de cada palabra insertada*         bool isEndOfWord;                         *// bool para saber si termino la palabra*          TrieNode()         {             isEndOfWord = false;             count = 0;         }     };      *// Funcion para añadir palabra al TrieNode*     void insertWord(TrieNode \*node, string word)     {         TrieNode \*current = node;         for (char c : word)         {             if (current->children.find(c) == current->children.end())             {                 current->children[c] = new TrieNode();             }             current = current->children[c];         }         *// Al finalizar una palabra se aumenta el numero de frecuencia*         current->count += 1;         current->isEndOfWord = true;     }      *// Funcion para obtener las palabras del TrieNode*     void getWord(TrieNode \*node, string prefix, vector<pair<string, int>> &res)     {          *// Si se encuentra en el final de la palabra*         if (node->isEndOfWord)         {             *// Ingresa la palabra al vector como un pair de la palabra y su frecuencia*             res.push\_back(make\_pair(prefix, node->count));         }          *// Uso de iteradores para recorre el map*         auto iter = node->children.begin();         while (iter != node->children.end())         {             *// Se genera un char donde se guardar la letra del actual TrieNode*             char letter = iter->first;             *// Se llama recursivamente a la funcion para seguir recorriendo el map*             getWord(iter->second, prefix + letter, res);             iter++;         }     }      *// Funcion principal de LeetCode*     vector<string> topKFrequent(vector<string> &words, int k)     {         *// Declaramos la clase TrieNode e insertamos las palabras*         TrieNode \*trie = new TrieNode();         for (string word : words)             insertWord(trie, word);          *// Luego creamos un vector con pair para obtener las palabras con su respectiva frecuencia*         vector<pair<string, int>> result;         getWord(trie, "", result);          *// Creamos una variable lambda para ordenar el arreglo en base a su frecuencia*         auto compare = [&](pair<string, int> a, pair<string, int> b)         {             if (a.second != b.second)             {                 return a.second > b.second; *// Orden descendente por frecuencia*             }             else             {                 return a.first < b.first; *// Orden lexicográfico si tienen la misma frecuencia*             }         };          *// Ordenamos con la funcion sort y el lambda "compare"*         sort(result.begin(), result.end(), compare);          *// Posteriormente creamos el vector donde estara nuestra respuesta*         vector<string> answer;         for (int i = 0; i < k; i++)         {             answer.push\_back(result[i].first);         }         return answer;     } }; |

Interfaz de usuario gráfica

Descripción generada automáticamente

Imagen que contiene Forma

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamente

1. Programación Dinámica ([Longest Palindromic Substring - LeetCode](https://leetcode.com/problems/longest-palindromic-substring/description/)) ([Longest palindromic substring | Dynamic programming - YouTube](https://www.youtube.com/watch?v=UflHuQj6MVA&ab_channel=Techdose))

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

|  |
| --- |
| class Solution { public:     string longestPalindrome(string s)     {         int n = s.length();         vector<vector<bool>> dp(n, vector<bool>(n, false));          *// String*         *// 0 1 2 3 4 5 6 7*         *// a a a b b a b a*          *// Casos base: cada carácter individual es palindrómico*         for (int i = 0; i < n; i++)         {             dp[i][i] = true;         }          *// Matrix DP*         *//   0 1 2 3 4 5 6 7*         *// 0 1 0 0 0 0 0 0 0*         *// 1 0 1 0 0 0 0 0 0*         *// 2 0 0 1 0 0 0 0 0*         *// 3 0 0 0 1 0 0 0 0*         *// 4 0 0 0 0 1 0 0 0*         *// 5 0 0 0 0 0 1 0 0*         *// 6 0 0 0 0 0 0 1 0*         *// 7 0 0 0 0 0 0 0 1*          int start = 0;     *// Índice inicial de la subcadena palindrómica más larga encontrada*         int maxLength = 1; *// Longitud de la subcadena palindrómica más larga encontrada*          *// Casos base: subcadenas de longitud 2*         for (int i = 0; i < n - 1; i++)         {             if (s[i] == s[i + 1])             {                 dp[i][i + 1] = true;                 start = i;                 maxLength = 2;             }         }          *// Casos generales: subcadenas de longitud mayor a 2*         for (int len = 3; len <= n; len++)         { *// 'for' para ir aumentando el tamaño del palindromo*             for (int i = 0; i <= n - len; i++)             { *// 'for' para ir recorriendo a medida que el tamanio del palindromo aumente*                  *// variable j para ir comparando el palindromo*                 int j = i + len - 1;                  *// La comparacion de (s[i] == s[j]) sera de la siguiente manera*                  *// i - j - - - - -      - i - j - - - -*                 *// 0 1 2 3 4 5 6 7  ->  0 1 2 3 4 5 6 7*                 *// a a a b b a b a      a a a b b a b a*                  *// La comparacion  de  dp[i + 1][j - 1] == 1 sera de la siguiente manera*                 *// Tomando en cuenta el elemento en dicha posicion*                                 *// Matrix DP*                 *//   0 1 2 3 4 5 6 7*                 *// 0 1 1 0 0 0 0 0 0*                 *// 1 0 1 1 0 0 0 0 0*                 *// 2 0 0 1 1 0 0 0 0*                 *// 3 0 0 0 1 0 0 0 0*                 *// 4 0 0 0 0 1 1 0 0*                 *// 5 0 0 0 0 0 1 0 0*                 *// 6 0 0 0 0 0 0 1 1*                 *// 7 0 0 0 0 0 0 0 1*                  *// SI cumplen las anteriores condiciones entra en el if*                 if (s[i] == s[j] && dp[i + 1][j - 1])                 {                     *// Dicha posicion la marca como True para seguir recorriendo la matriz*                     dp[i][j] = true;                     *// Para luego obtener el palindromo se guarda la posicion(start) y tamanio(maxLength)*                     start = i;                     maxLength = len;                 }             }         }         *// Retornamos el palindromo maximo como un subString del string original*         return s.substr(start, maxLength);     } }; |

Imagen que contiene Gráfico de dispersión

Descripción generada automáticamente

Imagen que contiene Gráfico

Descripción generada automáticamente

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Imagen que contiene Gráfico

Descripción generada automáticamente

Patrón de fondo

Descripción generada automáticamente

Patrón de fondo

Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamente

1. Grafo ([Clone Graph - LeetCode](https://leetcode.com/problems/clone-graph/description/))

Texto

Descripción generada automáticamente

Escala de tiempo

Descripción generada automáticamente con confianza media

|  |
| --- |
| */\* // Definition for a Node. class Node { public:     int val;     vector<Node\*> neighbors;     Node() {         val = 0;         neighbors = vector<Node\*>();     }     Node(int \_val) {         val = \_val;         neighbors = vector<Node\*>();     }     Node(int \_val, vector<Node\*> \_neighbors) {         val = \_val;         neighbors = \_neighbors;     } }; \*/  class Solution { public:     // Funcion para recorrer el grafo mediante una variante DFS (recursivo) que ira guardando los valores     Node \*dfs(Node \*current, unordered\_map<Node \*, Node \*> &visited)     {         //Creamos las variables a utilizar         vector<Node \*> neighbors;         Node \*clone = new Node(current->val);         visited[current] = clone;          // recorremos los nodos vecinos del actual grafo         for (auto it : current->neighbors)         {             // Si el nodo existe se guarda ese nodo en el vector de neighbors             if (visited.find(it) != visited.end())             {                 neighbors.push\_back(visited[it]);             }             else             // En otro caso se sigue recorriendo                 neighbors.push\_back(dfs(it, visited));         }          // Los neighbors se guardan en el grafo clonado en el vector de neighbors y es retornado         clone->neighbors = neighbors;         return clone;     }      // Funcion principal de Leetcode     Node \*cloneGraph(Node \*node)     {         unordered\_map<Node \*, Node \*> visited;         // Si nos dan un nodo nulo         if (node == NULL)             return NULL;          // Si el nodo no tiene vecinos         if (node->neighbors.size() == 0)         {             Node \*clone = new Node(node->val);             return clone;         }          // Si existe el nodo y tiene vecinos se realiza la clonacion mediante dfs alterado         return dfs(node, visited);     } };* |

Patrón de fondo

Descripción generada automáticamente

Patrón de fondo

Descripción generada automáticamente

Patrón de fondo

Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamente

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

Este trabajo final aporta a la competencia ABET ya que nos permite desarrollar la capacidad de diseñar al tener que imaginar múltiples soluciones posibles eficientes para resolver cada uno de los problemas, es decir, analizar los requerimientos del problema, identificar las entradas y salidas esperadas y plantear los algoritmos y estructuras de datos adecuadas. Además, una vez diseñado estas soluciones se implementaron con ayuda del lenguaje de programación C++, traduciendo los diseños planteados a una solución en código ejecutable que resuelva el problema. Por último, este trabajo final nos permitió evaluar estas soluciones al enviarlas al juez de las plataformas de programación competitiva, que incluyó diferentes casos de prueba y casos límite para que se asegure de que la solución funcione adecuadamente en todos los escenarios y su complejidad temporal y espacial sea eficiente.

1. CONCLUSIONES

* Una de las conclusiones sobre el trabajo final del curso de Fundamentos de Programación Competitiva es que proporciona a los estudiantes la oportunidad de aplicar los conocimientos adquiridos en el curso en un contexto práctico y desafiante.
* Después de finalizar el trabajo final se puede concluir que este permite desarrollar el razonamiento cuantitativo al presentar problemas que requieren un análisis exhaustivo y manipulación de datos numéricos en diversos formatos.
* En conclusión, el trabajo final promueve el pensamiento algorítmico y la resolución de problemas ya que para realizar las soluciones de los ejercicios se tienen que aplicar diversos algorítmicos y estructuras de datos. Esto implica comprender el enfoque algorítmico y analizar el rendimiento y la eficiencia de las soluciones.

1. RECOMENDACIONES

* Para realizar este trabajo final es fundamental trabajar en equipo ya que de esta manera se pueden abordar los problemas de manera más eficiente. Recomiendo que se promueva la colaboración activa, donde los miembros puedan discutir y compartir ideas, plantear soluciones conjuntas y revisar el código de los demás. Esto no solo fomentará el aprendizaje mutuo, sino que también ayudará a abordar los problemas desde diferentes perspectivas y encontrar soluciones más sólidas.
* Otra recomendación que puedo brindar es analizar la complejidad de cada una de las soluciones de los problemas antes de implementarlas. Esto puede ayudar significativamente a ahorrar tiempo en la resolución ya que así uno podrá saber si el enfoque planteado será aceptado en el tiempo y espacio límite que brinda la plataforma.

1. ANEXOS

* <https://docs.google.com/document/d/1dWmW-uyyU6NYhAvjWDbf8KO8NA4RFdeTeRhcx-5DnB0/edit?usp=sharing>

1. BIBLIOGRAFÍA

Chen, Hao. (2023). LeetCode. Recuperado de: <https://leetcode.com/> [Consulta: 29 de junio de 2023].

Mirzayanov, Mike. (2010). Codeforces. Recuperado de: <https://codeforces.com/>. [Consulta: 29 de junio de 2023].