**Informe - Trabajo Final - Grupo 6**

**“Algoritmo y Estructura de Datos”**

****

**Alumnos:**

* Sanandres Rojas, Edwin Eduardo
* Barrionuevo Gutiérrez, Daniel Ulises

**Sección:**

SS33

**Profesor:**

Cueva Chavez, Walter

2022 - 1

**Caso de estudio:**

Nuestro trabajo final consiste en el desarrollo de un programa basado en el ya conocido Lastpass que guarda, administra y modifica diferentes tipos de contraseñas de diferentes empresas/redes sociales para cada tipo de usuario con diferentes planes mensuales. El proyecto ha sido desarrollado en consola usando la programación orientada a objetos usando el lenguaje de programación C++ en Visual Studio. El proyecto se utiliza como muestra de lo aprendido a lo largo del presente ciclo.

**Funcionalidades del proyecto:**

El presente proyecto presenta las siguientes estructuras de datos:

* **Árbol Binario:** Al trabajar con muchos clientes y necesitar buscar un cliente en específico para que pueda acceder al programa usamos un árbol binario que se ordena mediante una function que compara el id de los clientes.
* **Lista Enlazada Simple:** Se utiliza para guardar los formularios, puesto que, no sabemos la cantidad de formularios que necesitará cada cliente.
* **Hashtable:** Usamos esta estructura para guardar todas las contraseñas, el cual nos permite guardar la contraseña por empresa a la cual está asociada.
* **Map:** Usamos los maps en la bóveda, asociamos una contraseña ingresada por el usuario a una cuenta bancaria. Nos permite crear un vector de pares ordenados de una clave y un dato.
* **Set:** Lo usamos de apoyo para guardar el nombre de la empresa asociado con un número. Es una estructura de datos que no permite repetir un dato dentro de un arreglo.
* **Vector:** Nos permite guardar y acceder de manera eficaz a una cantidad limitada y pequeña de dispositivos.

A través de los menús el usuario podrá:

* Registrarse (cuenta empresarial o personal)
* Ingresar a su cuenta(personal o empresarial)
  + Crear contraseñas
  + Administrar contraseñas
  + Administrar dispositivos
  + Administrar formularios
  + Administrar bóveda bancaria
  + Administrar plan mensual
  + Administrar datos personales

**Big O y Descripción de funciones en las estructuras de datos:**

**Árbol Binario:**

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 | **template**<**class** **Generico**>  **class** **Arbol** {  **private:**  Nodo\_T<Generico>\* raiz;  **size\_t** cantidad;  function<**void**(Generico)> show;  function<**bool**(Generico, Generico)> compare;  function<**size\_t**(Generico, Generico)> doSomething;  **public:**  Nodo\_T<Generico>\* GetRaiz() { **return** raiz; } // 1  **void** setElemento(Generico a, Nodo\_T<Generico>\*& tmp) { tmp->setGenerico(a); } // 2  Arbol(function<**void**(Generico)> show,  function<**bool**(Generico, Generico)> compare,  function<**size\_t**(Generico, Generico)> doSomething) :  show(show), compare(compare), doSomething(doSomething) {  raiz = nullptr;  cantidad = **0**;  }  **void** setFunction(function<**void**(Generico)> a, function<**bool**(Generico, Generico)> b,  function<**size\_t**(Generico, Generico)> c) { // 3  show = a; compare = b; doSomething = c;  }  **void** **insert**(Generico e) { insert(e, raiz); } // O(log n) [Si está vacio -> O(1)]  **void** **insert**(Generico g, Nodo\_T<Generico>\*& tmp) {  **if** (tmp == nullptr) // 1 + 2 = 3  tmp = **new** Nodo\_T<Generico>(g); // 2 + O(log n)  **else** **if** (compare(g, tmp->elemento)) // 2 + O(log n)  **insert**(g, tmp->der);  **else** // O(log n)  **insert**(g, tmp->izq);  }  **void** replaceElement(Generico e) { replaceElement(e, raiz); } // O(log n)  **void** replaceElement(Generico g, Nodo\_T<Generico>\*& tmp) {  **if** (tmp != NULL)  {  **if** (doSomething(g, tmp->elemento) == **0**)  setElemento(g, tmp);  **else** **if** (doSomething(g, tmp->elemento) == -**1**)  replaceElement(g, tmp->izq);  **else** **if** (doSomething(g, tmp->elemento) == **1**)  replaceElement(g, tmp->der);  }  }  **void** print() { print(raiz); } // O(n)  **void** print(Nodo\_T<Generico>\* tmp) {  **if** (tmp == nullptr) **return**; // Recorre recursivamente cada hijo  print(tmp->izq); // 2  show(tmp->elemento); // 4  print(tmp->der); // 2  }  // Recorre el árbol en base a comparar valores lo que le hace recorrer solo los niveles del arbol  Generico search(Generico g) { **return** search(g, raiz); }  Generico search(Generico g, Nodo\_T<Generico>\*& tmp) { // O(log n)  **if** (tmp != NULL)  {  **if** (doSomething(g, tmp->elemento) == **0**) {  **return** tmp->elemento;  }  **else** **if** (doSomething(g, tmp->elemento) == -**1**) **return** search(g, tmp->izq);  **else** **if** (doSomething(g, tmp->elemento) == **1**) **return** search(g, tmp->der);  **else** {  **return** g;  }  }  }  }; |
| --- | --- |

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88 | **template**<**class** **Generico**>  **class** **Lista** {  **private:**  Nodo<Generico>\* inicio;  Nodo<Generico>\* fin;  **size\_t** cantidad;  function<**void**(Generico)> show;  **public:**  Lista(function<**void**(Generico)> show) : show(show) {  inicio = fin = nullptr;  cantidad = **0**;  }  Nodo<Generico>\* getInicio() { **return** inicio; } // 1  **void** push\_front(Generico e) { // 7 -> O(1)  Nodo<Generico>\* n = **new** Nodo<Generico>(e); // 2  **if** (cantidad == **0**) // 1 + 1 = 2  inicio = n; // 1  **else** { // 3  n->sig = inicio; // 2  inicio = n; // 1  }  ++cantidad; // 2  }  **void** push\_back(Generico e) { 7 -> O(1)  Nodo<Generico>\* nuevo = **new** Nodo<Generico>(e); // 2  **if** (inicio == nullptr)//1 + 2 -> 3  inicio = fin = nuevo; // 2  **else** { 3  fin->sig = nuevo; // 2  fin = nuevo; // 1  }  cantidad++; // 2  }  **void** search(**size\_t** n) { // 4n + 11 -> O(n)  Nodo<Generico>\* aux = inicio; // 2  **size\_t** c = **0**; // 2  **if** (inicio == nullptr || n <= **0** || n - **1** >= cantidad) // 4  **return**;  **while** (c != n - **1**) // n(4) -> 4n  {  aux = aux->sig; // 2  c++; // 2  }  show(aux->elemento); // 3  }  **void** delete\_front() { // 4 -> O(1)  **if** (inicio == nullptr) // 1 + 1 -> 2  **return**; // 1  **else** // 2  inicio = inicio->sig; // 2  }  **void** delete\_back() { //5n +3 -> O(n)  Nodo<Generico>\* aux = inicio; // 2  **if** (inicio == nullptr) // 1 +1 -> 2  **return**; // 1  **if** (inicio->sig == nullptr) { // 2 + 2 -> 4  inicio = nullptr; // 1  **return**; // 1  }  **while** (aux->sig->sig != nullptr) // (n-1)(3 + 2) -> 5n - 5  aux = aux->sig; // 2  aux->sig = nullptr; // 2  }  **void** delete\_by\_position(**size\_t** n) { // 9n + 12 -> O(n)  Nodo<Generico>\* aux = inicio; // 2  **int** c = **0**; // 2  **while** (c != n - **1**) { //n(4) -> 4n  aux = aux->sig; // 2  c++; // 2  }  **if** (aux->sig->sig == nullptr) { //3 + 2 -> 5  aux->sig = nullptr; // 2  }  **else** {  aux->sig = aux->sig->sig; // 3  }  cantidad--; // 2  cout << "Elemento eliminado!" << endl; // 3  print(); // 5n +2  }  **void** print() { //5n +2 O(n)  Nodo<Generico>\* aux = inicio; // 2  **while** (aux != nullptr) { // n(5) -> 5n  show(aux->elemento); // 3  aux = aux->sig; // 2  }  }  }; |
| --- | --- |

**HT:**

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | **template**<**class** Generico1, **class** Generico2>  **class** HT {  array < list<Entity<Generico1, Generico2>>, 5 > container;  function<**void**(Generico1, Generico2)> show;  function<**int**(Generico1)> index;  public:  HT(function<**void**(Generico1, Generico2)> show,  function<**int**(Generico1)> index) : show(show), index(index) { }  **int** getIndex(Generico1 key) { **return** index(key); } // 3  array < list<Entity<Generico1, Generico2>>, 5 > getContainer() { **return** container; } // 1  **void** setContainer(array < list<Entity<Generico1, Generico2>>, 5 > c) { container = c; } // 1  **void** add(Generico1 key, Generico2 value) { //O(1) ¨[Peor caso: O(n)]  container[getIndex(key)].push\_back(Entity<Generico1, Generico2>(key, value)); }  **void** display() { // O(n^2) 30n^2 +129  **for** (**size\_t** i = 0; i < container.size(); ++i) { 4 + 5(n + 3 + 5n +22 ) = 30n^2 +129  **for** (**auto** it : container[i]) { 2 + 5(n+4) = 5n + 22  cout << "**\t\t**"; // 1  show(it.getG1(), it.getG2()); // 3  }  }  }  **void** deleteg(Generico1 key) { // O(n^2) 23 + 5n^2  HT<Generico1, Generico2> aux(show,index); // 1 + 1 + 1 = 3  **for** (**size\_t** i = 0; i < container.size(); ++i) { // 1 + 2 + 5(3+n^2) = 18 + 5n^2  **for** (**auto** it : container[i]) {  **if** (it.getG1() != key) // 3 + n  {  aux.add(it.getG1(), it.getG2()); //n  }  }  }  **this**->setContainer(aux.getContainer()); //2  }  }; |
| --- | --- |

**Vector:**

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | **template** <**typename** T>  **class** **Vector**  {  T\* arr;  **int** capacity;  **int** current;  function<**void**(T)> show;  **public:**  Vector(function<**void**(T)> show) : show(show) {  arr = **new** T[**1**];  capacity = **1**;  current = **0**;  }  **void** push(T data) { // 5n + 13 -> O(n)  **if** (current == capacity) { // 1 + 8 +5n -> 5n +9  T\* temp = **new** T[capacity + **1**]; // 1 + 1 -> 2  **for** (**int** i = **0**; i < capacity; i++) { // 2 + n(3 + 2) = 5n +2  temp[i] = arr[i]; // 3  }  **delete**[] arr; // 1  capacity += **1**; //2  arr = temp; // 1  }  arr[current] = data; // 2  current++; // 2  }  **void** push(T data, **int** index) { // 5n + 15  **if** (index == capacity) // 2 + 5n +13 -> 5n + 15  push(data); // 5n + 13  **else** // 2  arr[index] = data; // 2  }  T get(**int** index) { // 3 -> O(1)  **if** (index <= current) // 1 + 2 -> 3  **return** arr[index];  }  **int** size() { **return** current; } // 1 -> O(1)  **int** getcapacity() { **return** capacity; } // 1 -> O(1)  **void** print() { // 5n + 2 -> O(n)  **for** (**int** i = **0**; i < current; i++) { // 2 + n(3+2) -> 5n + 2  show(arr[i]); // 3  }  }  **void** setT(T t, **int** index) { // 3 -> O(1)  **if** (index < current) // 1 + 2 -> 3  arr[index] = t; // 2  }  }; |
| --- | --- |

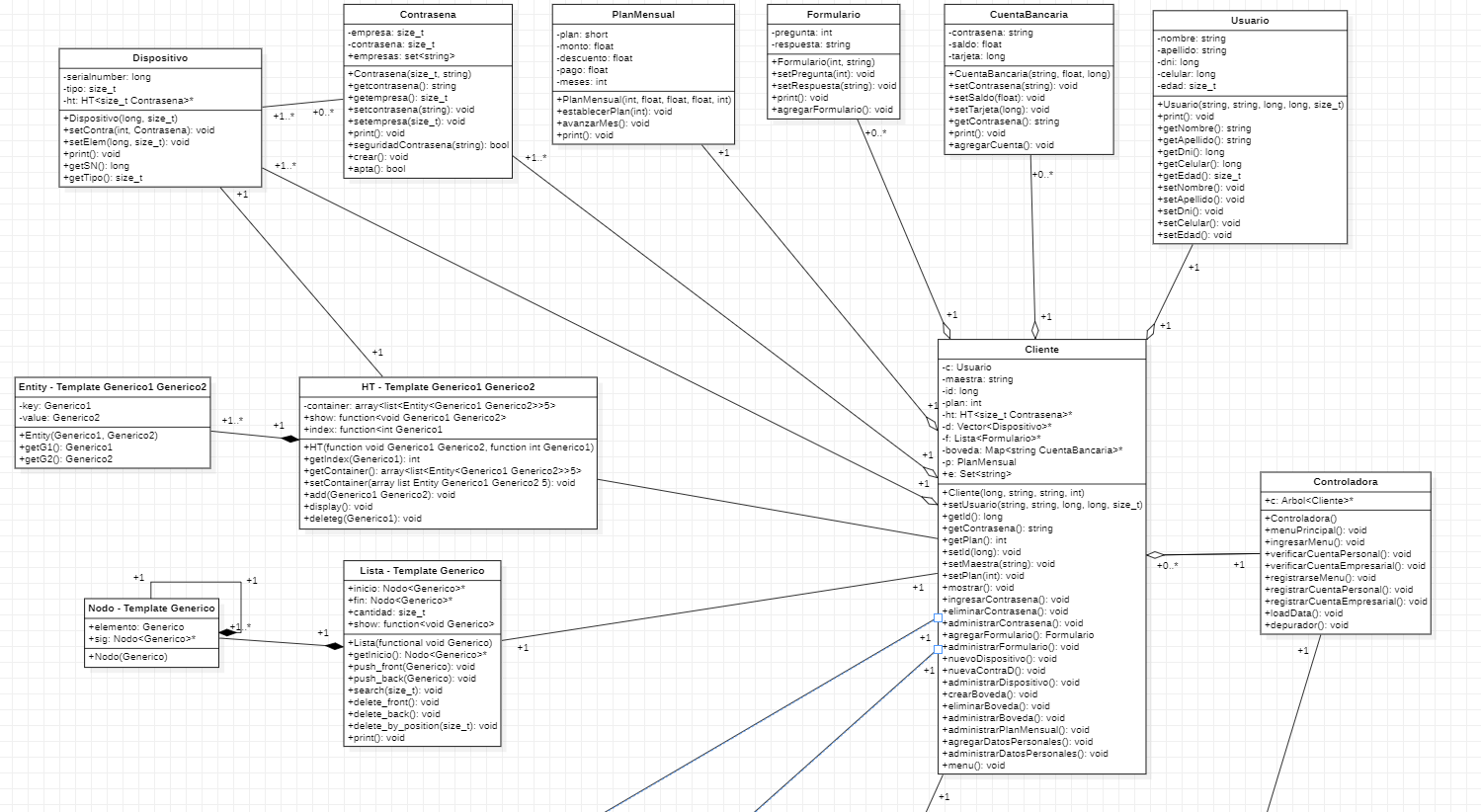
**Map:**

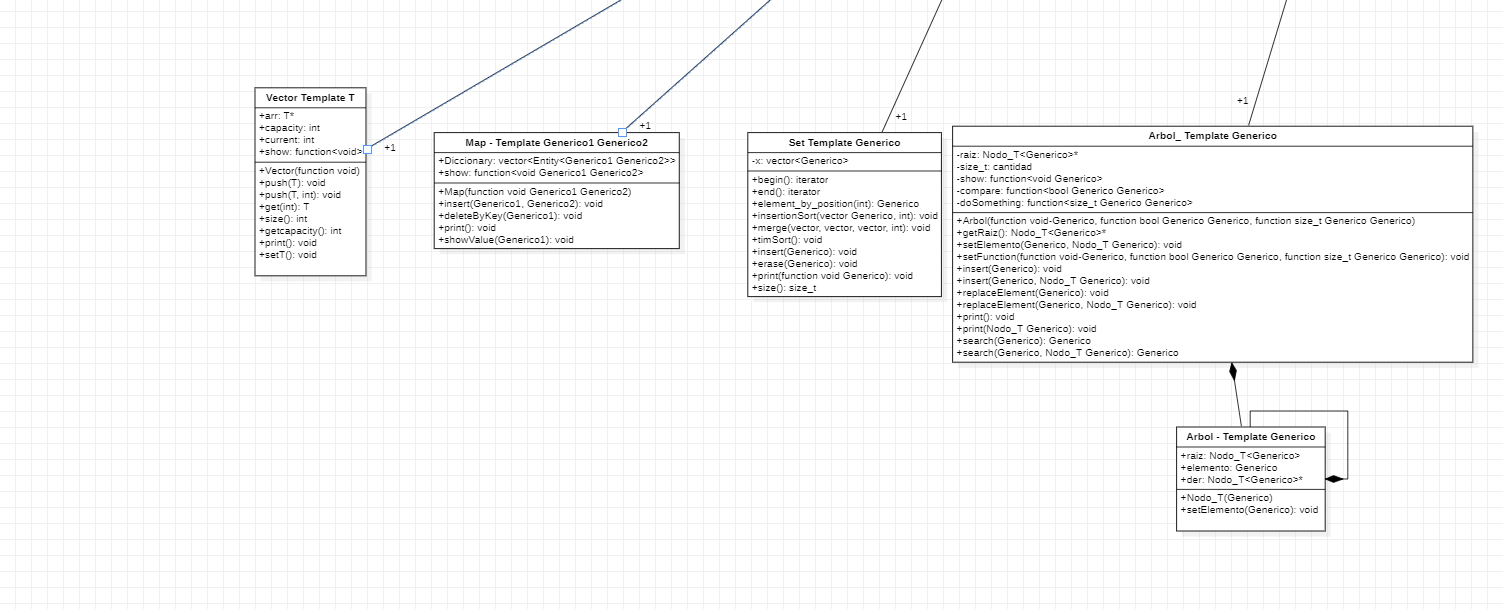
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | **template**<**class** **Generico1**, **class** **Generico2**>  **class** **Map** {  vector<Entity<Generico1, Generico2>> Diccionary;  function<**void**(Generico1, Generico2)> show;  **public:**  Map(function<**void**(Generico1, Generico2)> show) : show(show) { }  **void** insert(Generico1 a, Generico2 b) { // 5n +18 O(n)  Entity<Generico1, Generico2> aux(a, b); // 3  Diccionary.push\_back(aux); // 5n + 15  }  **void** deleteByKey(Generico1 a) { // n^2 + 5n + 2 -> O(n^2)  **for** (**size\_t** i = **0**; i < Diccionary.size(); i++) // 2 + n(2+ 3 +n)  {  **if** (a == Diccionary.at(i).getG1()) // 3 + n  {  Diccionary.erase(Diccionary.begin() + i); // n  }  }  }  **void** print() { // 9n + 2  **for** (**size\_t** i = **0**; i < Diccionary.size(); i++) // n(7 + 2) + 2 -> 9n + 2  show(Diccionary.at(i).getG1(), Diccionary.at(i).getG2()); // 4 + 3  }  **void** showValue(Generico1 key) { // 12n + 2 -> O(n)  **for** (**size\_t** i = **0**; i < Diccionary.size(); i++) // n(10 + 2) + 2 -> 12n +2  {  **if** (key == Diccionary.at(i).getG1()) // 3 + 7 -> 10  show(Diccionary.at(i).getG1(), Diccionary.at(i).getG2()); // 7  }  }  }; |
| --- | --- |

**Set:**

| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100 | **template**<**class** Generico>  **class** Set {  private:  vector<Generico> x;  public:  Set() {}  *//CLASS ITERATOR-------------------------*  **class** iterator {  **int** i;  vector<Generico> xd;  public:  iterator(**int** i = 0) : i(i) {}  iterator(**int** i, vector<Generico> arr) : i(i), xd(arr) {}  **void** **operator** ++ () { i++; } // 2  Generico **operator** \* () { **return** xd[i]; } // 2  **bool** **operator** != (iterator b) { **return** **this**->i != b.i; } // 4  };  *//---------------------------------------*  *//ITERATOR*  iterator begin() { **return** iterator(0, x); } // 3  iterator end() { **return** iterator(x.size()); } // 2  Generico element\_by\_position(**int** pos) { // 2 -> O(1)  **return** x[pos]; // 2  }  **void** insertionSort(vector<Generico>& data, **int** n) { // 3n^2 + 12n + 5 -> O(n^2)  **int** i, j; // 2  Generico tmp; // 1  **for** (i = 1; i < n; i++) { // 2 + n(2 + 3n + 10) = 3n^2 + 12n + 2  j = i; // 1  tmp = data[i]; // 2  **while** (j > 0 && tmp < data[j - 1]) { // 3(n) + 5  data[j] = data[j - 1]; // 3  j--; // 2  }  data[j] = tmp; // 2  }  }  **void** merge2(vector<Generico>& arr1,vector<Generico>& arr2,vector<Generico>& arr3,**int** n){  **size\_t** i = 0, j = 0, k = 0; // 6  **size\_t** mitad = n / 2; // 2  **while** (i < mitad && j < n - mitad) {  **if** (arr1[i] < arr2[j]) {  arr3[k] = arr1[i];  i++; k++;  }  **else** {  arr3[k] = arr2[j];  j++; k++;  }  }  **while** (i < mitad) {  arr3[k] = arr1[i];  i++; k++;  }  **while** (j < n - mitad) {  arr3[k] = arr2[j];  j++; k++;  }  }  **void** Tim\_Sort() {  **int** mitad = size() / 2;  **int** mitad2 = size() - mitad;  vector<Generico> arr1;  vector<Generico> arr2;  **for** (**int** i = 0; i < mitad; i++)  arr1.push\_back(x[i]);  **for** (**int** i = 0; i < mitad2; i++)  arr2.push\_back(x[i + mitad]);  insertionSort(arr1, arr1.size());  insertionSort(arr2, arr2.size());  merge2(arr1, arr2, x, x.size());  }  **void** insert(Generico e) { // 10n + 19  **bool** existe = false; // 2  **for** (**size\_t** i = 0; i < x.size(); i++) { // 2 + n(2 +3) -> 5n +2  **if** (x[i] == e) { // 2 + 1  existe = true;  }  }  **if** (existe == false) { // 2 + 5n + 15 -> 5n + 17  x.push\_back(e); // 5n + 15  }  Tim\_Sort();  }  **void** erase(Generico e) { //7n +2 -> O(n)  **for** (**size\_t** i = 0; i < x.size(); i++) { // 2 + n(2 + 5) -> 7n +2  **if** (x[i] == e) { // 2 + 3 -> 5  x.erase(x.begin() + i);  }  }  }  **void** print(function<**void**(Generico)> f) { // 4n + 2 -> O(n)  **for** (**size\_t** i = 0; i < x.size(); i++) { // 2 + n(2 + 2) = 4n +2  f(x[i]);  }  }  **size\_t** size() { // 2 -> O(1)  **return** x.size(); // 2  }  }; |
| --- | --- |

**Diagrama de clases:**

****

****

**Descripción de las tareas:**

Ambas personas trabajaron en las clases presentadas.

| **Nombre** | **Tareas** | **Autovaloración** |
| --- | --- | --- |
| Sanandres Rojas, Edwin Eduardo | Implementación del Map.  Implementación del árbol binario y sus funciones.  Implementación del  lista y sus funciones. | 50% |
| Barrionuevo Gutiérrez, Daniel Ulises | Implementación de la estructura set.  Creación del diagrama de clase.  Implementación del hashtable y sus funciones  Implementación del vector y sus funciones.  Creación de la implementación de lectura y escritura de archivos | 50% |