**ANÁLISIS Y DISEÑO DE ALGORITMOS I**

**PROYECTO DE CURSADA 2022**

**Primera parte**

Grupo N° 53

Integrantes:

* Nahuel Fabian Flores Medrano - nffloresmedrano@gmail.com
* Abril Valentina Juarez - juarezabrilvalentina@gmail.com

Ayudante:

* \*A definir\*

*CLASE LISTA*

**Class** Lista[elemL]

**Basic Constructors** crear,agregar

**Effective**

**Type** Lista

**Operations**

Lista: ->Lista;

Agregar: elemL x Lista(L) x int(i) -> Lista

Pre: (i>=1) and (i<=cantidad\_elementos(L)+1);

Cantidad\_elementos: Lista x int -> int;

Verificar: Lista x elemL -> bool;

Es\_vacio: Lista ->bool;

Eliminar\_elemento: Lista(L) x elemL -> Lista;

Pre: not es\_vacio(L);

Iniciar\_iterador: Lista ->Lista;

Avanzar\_iterador: Lista -> Lista;

Elemento\_iterador: Lista -> elemL;

Final\_iterador: Lista -> bool;



End\_class

La estructura utilizada para implementar esta clase es un simple, con lugar para un elemento y un puntero al elemento siguiente. Al ser una lista simple se usa iteraciones lineales ya que para acceder a un nodo hay que pasar por sus antecesores.

*Detalles Funciones:*

* Lista: Constructor de la clase usado para asignar valores o estructuras necesarias para el uso de la clase, complejidad temporal- O(1)
* Agregar: Agregar elementos en la lista ya sea al principio, final o posición ingresada por el usuario. Por defecto agrega al principio de la lista para evitar recorrerla completamente. Según el método pedido para ingresar a la lista su complejidad es O(1) u O(n) siendo n la cantidad de elementos en la lista.
* Cantidad\_elementos: Devuelve la cantidad de elementos agregados a la lista almacenados en una variable. Complejidad de O (1).
* Verificar: Recorre la lista, para encontrar el valor solicitado. Complejidad de O(n) con n igual a la cantidad de elementos al momento de ejecutar la función.
* Es\_vacio: Pregunta si el puntero principal de la lista está apuntando a NULL, devolviendo un booleano como respuesta. Complejidad O(1).
* Eliminar\_elemento: Recorre el arreglo hasta encontrar el valor a eliminar deseado o fin de la lista. Complejidad O(n) con n igual a la cantidad de elementos.
* Iniciar\_iterador: Inicializa el iterador de la lista al primer elemento;
* Avanzar\_iterador: Avanza un paso el iterador de la lista, en caso de que no sea NULL;
* Elemento\_iterador: Retorna el elemento en donde el puntero del iterador está parado.;
* Final\_iterador: Retorna booleano indicando si se llegó a final de la lista.

*CLASE FILA*

**Class** Fila[elemL]

**Basic Constructors crear,** agregar

**Effective**

**Type** Fila

**Operations**

Fila: -> Fila;

Agregar: elemL x Fila(L) -> FIla

Cantidad\_elementos: Fila -> int;

Verificar: Fila -> bool;

Es\_vacio: Fila ->bool;

Eliminar\_elemento: Fila(F) -> Lista;

Pre: not es\_vacio(F);



End\_class

*Detalles Funciones:*

* Fila: Constructor de la clase usado para asignar valores o estructuras necesarias para el uso de la clase, complejidad temporal- O(1)
* Agregar: Agregar elementos al final de la fila. Complejidad es O(n) u O(n) siendo n la cantidad de elementos actuales en la fila.
* Cantidad\_elementos: Devuelve la cantidad de elementos agregados a la fila almacenados en una variable. Complejidad de O (1).
* Verificar: Retorna el primer elemento en la fila. Complejidad de O(1).
* Es\_vacio: Pregunta si el puntero principal de la fila está apuntando a NULL, devolviendo un booleano como respuesta. Complejidad O(1).
* Eliminar\_elemento: Elimina el primer elemento de la fila por su comportamiento. Complejidad O(1) .

*CLASE ARBIN*

**Class** Arbin[elemA]

**Basic Constructors** crear,agregar

**Effective**

**Type** Arbin

**Operations**

Arbin: ->Arbin;

Agregar: elemA x Arbin -> Arbin;

Verificar: Arbin x elemA -> bool;

Es\_vacio: Arbin ->bool;

Eliminar\_elemento: Arbin(A) x elemA -> bool;

Pre: not es\_vacio(A);

Altura\_Árbol: Arbin -> int;

Eliminar\_elemento: Arbin x elemA -> bool;

Es\_hoja: Arbin -> bool;



End\_class

La estructura utilizada para implementar esta clase Arbin o árbol binario de búsqueda, con espacio para un elemento y dos puenteros a nodos iguales con elementos mayor y menor al mismo.

*Detalles Funciones:*

* Arbin: Constructor de la clase usado para asignar valores o estructuras necesarias para el uso de la clase, complejidad temporal- O(1)
* Agregar: Función básica para agregar elementos en el árbol. Ya que es un Árbol ordenado, su complejidad es O(n) en el peor de los casos al parecerse a una lista.
* Cantidad\_elementos: función simple que devuelve la cantidad de elementos agregados al Árbol almacenados en una variable. Complejidad de O (1).
* Verificar: función que recorre el Árbol para encontrar el valor solicitado. Complejidad de O(n) en el peor de los casos.
* Es\_vacio: simple función que pregunta si el puntero principal del Árbol está apuntando al vacío, devolviendo un booleano como respuesta. Complejidad O(1).
* Eliminar\_elemento: función que recorre el Árbol hasta encontrar el valor a eliminar deseado o fin de la lista. Complejidad O(n) ya que tiene que verificar todos los elementos en el árbol.
* Altura\_arbol: Función que recorre todos los elementos del árbol para devolver la altura de la rama más larga. Complejidad O(n).
* Es\_Hoja:
* Altura\_Árbol: Arbin -> int;
* Eliminar\_elemento: Arbin x elemA -> bool;
* Es\_hoja: Arbin -> bool;

Costos/Beneficios en las estructuras

Lista: la estructura utilizada en este proyecto es una variable para almacenar el dato y un puntero al elemento siguiente. Se puede mejorar el tiempo de acceso realizándolo con un arreglo, pero implicaría tener que saber la cantidad de elementos que el usuario va a ingresar.

Fila:

Arbin: la estructura para el Arbin usado tiene problemas para ciertos casos a la hora de ingresar los elementos (ordenados), podría mejorarse utilizando ciertas formas de ordenar los elementos a la hora de agregarlos, balanceando los nodos de tal forma que la raíz del árbol siempre sea un valor medio entre el menor y el mayor elemento del árbol.

CÓDIGO LISTA

**Lista.h:**

#ifndef LISTA\_EJEMPLO\_H

#define LISTA\_EJEMPLO\_H

template <typename elemL> class Lista{

struct nodo\_lista {

elemL elemento;

nodo\_lista \* sig;

};

int cant;

void vaciar(nodo\_lista \* & primero);

public:

Lista();

~Lista();

nodo\_lista \* Primero, \*it;

void agregar(const elemL elemento , int posicion = 0);

int cantidad\_elementos();

bool verificar( const elemL elemento);

bool es\_vacia() const;

bool eliminar\_elemento(const elemL elemento);

void iniciar\_iterador();

void avanzar\_iterador();

elemL elemento\_iterador();

bool final\_iterador();

};

#endif

**Lista.cpp:**

#include "lista.h"

#include <iostream>

#include <string>

#include <stdlib.h>

using namespace std;

template <typename elemL> Lista<elemL>::Lista() {

this->Primero = NULL;

this->cant = 0;

}

template <typename elemL> Lista<elemL>::~Lista() {

vaciar(this->Primero);

}

template <typename elemL> void Lista<elemL>::vaciar(nodo\_lista \* & primero) {

nodo\_lista \* aux;

aux = primero;

while (aux != NULL) {

primero = primero->sig;

delete aux;

aux = primero;

}

}

template <typename elemL> void Lista<elemL>::agregar(const elemL elemento,int posicion){

nodo\_lista \* aux = new nodo\_lista;

aux->elemento = elemento;

aux->sig = NULL;

nodo\_lista \* l = this->Primero;

if (l != NULL){

if (posicion == 0){

nodo\_lista \* cursor = l;

while (cursor->sig != NULL){

cursor = cursor->sig;

}

cursor->sig = aux;

}

else{

if (posicion == 1){

aux->sig = l;

this->Primero = aux;

}

else{

while (posicion > 1){

l = l->sig;

posicion--;

}

aux->sig = l->sig;

l->sig = aux;

}

}

}

else{

this->Primero = aux;

}

this->cant++;

}

template <typename elemL> int Lista<elemL>::cantidad\_elementos(){

return this->cant;

}

template <typename elemL> bool Lista<elemL>::verificar( const elemL elemento){

nodo\_lista \* l = this->Primero;

while (l != NULL){

if (l->elemento == elemento){

return true;

}

l = l->sig;

}

return false;

}

template <typename elemL> bool Lista<elemL>::es\_vacia() const{

if (Primero != NULL) return false;

return true;

}

template <typename elemL> bool Lista<elemL>::eliminar\_elemento(const elemL elemento){

nodo\_lista \* cursor = Primero;

nodo\_lista \* aux = cursor;

bool found = false;

while ((cursor != NULL) && (!found)){

if (cursor->elemento == elemento){

if (cursor == Primero) {

aux = cursor;

Primero = cursor->sig;

delete aux;

}

else{

aux->sig = cursor->sig;

delete cursor;

}

cant--;

found = true;

}

else{

aux = cursor;

cursor = cursor->sig;

}

}

return found;

}

template <typename elemL> void Lista<elemL>::iniciar\_iterador(){

it = Primero;

}

template <typename elemL> void Lista<elemL>::avanzar\_iterador(){

if (it) it = it->sig;

else it = NULL;

}

template <typename elemL> elemL Lista<elemL>::elemento\_iterador(){

return it->elemento;

}

template <typename elemL> bool Lista<elemL>::final\_iterador(){

return (it == NULL);

}

template class Lista<unsigned int>;

template class Lista<float>;

template class Lista<string>;

template class Lista<char>;

template class Lista<bool>;

/////////////////////////////////////////////////////////////////////////////////

Código Fila

Fila.h

#ifndef FILA\_EJEMPLO\_H

#define FILA\_EJEMPLO\_H

template <typename elemL> class Fila{

struct nodo\_lista {

elemL elemento;

nodo\_lista \* sig;

};

int cant;

void vaciar(nodo\_lista \* & primero);

public:

Fila();

~Fila();

nodo\_lista \* Primero;

void agregar(const elemL elemento);

int cantidad\_elementos();

elemL verificar();

bool es\_vacia();

bool eliminar\_elemento();

};

#endif

Fila.cpp

#include "fila.h"

#include <iostream>

#include <string>

#include <stdlib.h>

using namespace std;

template <typename elemL> Fila<elemL>::Fila() {

this->Primero = NULL;

this->cant = 0;

}

template <typename elemL> Fila<elemL>::~Fila() {

vaciar(this->Primero);

}

template <typename elemL> void Fila<elemL>::vaciar(nodo\_lista \* & primero) {

nodo\_lista \* aux;

aux = primero;

while (aux != NULL) {

primero = primero->sig;

delete aux;

aux = primero;

}

}

template <typename elemL> void Fila<elemL>::agregar(const elemL elemento){

nodo\_lista \* aux = new nodo\_lista;

aux->elemento = elemento;

aux->sig = NULL;

nodo\_lista \* l = this->Primero;

/\*\*Recorro la lista de primero hasta el final y agrego el elemento\*/

if (l != NULL){

nodo\_lista \* cursor = l;

while (cursor->sig != NULL){

cursor = cursor->sig;

}

cursor->sig = aux;

} else {

this->Primero = aux;

}

this->cant++;

}

template <typename elemL> int Fila<elemL>::cantidad\_elementos(){

return this->cant;

}

template <typename elemL> elemL Fila<elemL>::verificar(){

return this->Primero->elemento;

}

template <typename elemL> bool Fila<elemL>::es\_vacia(){

return (this->Primero == NULL);

}

template <typename elemL> bool Fila<elemL>::eliminar\_elemento(){

nodo\_lista \* cursor = this->Primero;

while (cursor != NULL){

this->Primero = this->Primero->sig;

delete(cursor);

return true;

}

return false;

}

template class Fila<unsigned int>;

template class Fila<float>;

template class Fila<string>;

template class Fila<char>;

template class Fila<bool>;

/////////////////////////////////////////////////////////////////////////////////

Código Arbin

**Arbin.h:**

#ifndef ARBOL\_BINARIO\_H\_INCLUDED

#define ARBOL\_BINARIO\_H\_INCLUDED

#include "climits"

template <typename elemA> class Arbol{

struct nodo\_arbol {

elemA elemento;

nodo\_arbol \* izq;

nodo\_arbol \* der;

};

int cant;

private:

bool verificar(nodo\_arbol \* cursor, elemA elemento);

bool es\_hoja(nodo\_arbol \* cursor);

public:

Arbol();

~Arbol();

nodo\_arbol\* Raiz;

void agregar(nodo\_arbol \* &cursor,const elemA elemento);

int cantidad\_elementos();

bool verificar\_elemento(elemA elemento);

bool es\_vacio();

int altura\_arbol(nodo\_arbol \* cursor);

bool eliminar\_elemento(const elemA elemento);

void pre\_agregar(const elemA elemento);

void mostrar\_inorder(nodo\_arbol \* cursor);

void mostrar\_frontera(nodo\_arbol \* cursor);

void mostrar();

};

#endif

**Arbin.cpp:**

#include "arbol.h"

#include <cstdlib>

#include <iostream>

#include <algorithm>

using namespace std;

template <typename elemA> Arbol<elemA>::Arbol() {

this->Raiz = NULL;

this->cant = 0;

}

template <typename elemA> Arbol<elemA>::~Arbol() {

delete this->Raiz;

}

template <typename elemA> void Arbol<elemA>::agregar(nodo\_arbol \* &cursor,const elemA elemento){

if (cursor == NULL){

nodo\_arbol \* aux = new nodo\_arbol;

aux->der = NULL;

aux->izq = NULL;

aux->elemento = elemento;

cursor = aux;

cant++;

}

else{

if (cursor->elemento < elemento){

agregar(cursor->der,elemento);

}

else{

agregar(cursor->izq,elemento);

}

}

}

template <typename elemA> void Arbol<elemA>::pre\_agregar(const elemA elemento){

agregar(Raiz,elemento);

}

template <typename elemA> void Arbol<elemA>::mostrar\_inorder(nodo\_arbol \* cursor){

if (cursor != NULL){

mostrar\_inorder(cursor->izq);

cout << cursor->elemento << "-";

mostrar\_inorder(cursor->der);

}

}

template <typename elemA> bool Arbol<elemA>::verificar\_elemento(elemA elemento){

return verificar(this->Raiz, elemento);

}

template <typename elemA> bool Arbol<elemA>::verificar(nodo\_arbol \* cursor, elemA elemento){

if (cursor != NULL){

if (cursor->elemento == elemento){

return true;

}

else{

if (cursor->elemento < elemento){

return verificar(cursor->der,elemento);

}

else return verificar(cursor->izq,elemento);

}

}

else return false;

}

template <typename elemA> void Arbol<elemA>::mostrar(){

nodo\_arbol \* cursor = Raiz;

if (Raiz != NULL){

mostrar\_inorder(cursor);

}

else{ cout << "Arbol vacio" << endl;}

}

template <typename elemA> int Arbol<elemA>::cantidad\_elementos(){

return cant;

}

template <typename elemA> bool Arbol<elemA>::es\_vacio(){

return (Raiz == NULL);

}

template <typename elemA> int Arbol<elemA>::altura\_arbol(nodo\_arbol\* cursor){

if (cursor != NULL){

return 1+max(altura\_arbol(cursor->der),altura\_arbol(cursor->izq));

}

else return 0;

}

template <typename elemA> bool Arbol<elemA>::es\_hoja(nodo\_arbol\* cursor){

if ((cursor->der == NULL) && (cursor->izq == NULL)) return true;

else return false;

}

template <typename elemA> void Arbol<elemA>::mostrar\_frontera(nodo\_arbol \* cursor){

if (cursor != NULL){

if (es\_hoja(cursor)){ cout << cursor->elemento << "-";}

else{

mostrar\_frontera(cursor->der);

mostrar\_frontera(cursor->izq);

}

}

}

//-------------------------------------

template class Arbol<unsigned int>;

template class Arbol<float>;

template class Arbol<string>;

template class Arbol<char>;

template class Arbol<bool>;