Logotipo

Descripción generada automáticamente

# ARBOL AVL

**Fecha:** 19/06/2023

**Materia:** Estructura de datos

**Profesor/Profesora:** Ismael López Buenrostro

**Nombre:** Luis Rigoberto Ramírez Llamas

**Registro:** 21100298

**Carrera:** Drllo. Software

**Grupo:** 4E1

**Descripción del programa**

Se pretende realizar un árbol  AVL con opciones variadas

**Lista de requerimientos**

Se desea crear un programa que emplee un árbol AVL con las siguientes opciones:

* Agregar.
* Eliminar uno.
* Eliminar todo.
* Mostrar
  + Orden
  + Preorden
  + Postorden
* Contar.
* Calcular altura (desde la raíz)
* ​Buscar un elemento.

Notas:

1. Cada hoja deberá tener nombre, registro, fecha de nacimiento.
2. El campo para la fabricación del árbol será el registro.

Notas para revisión.

1. Debe ingresar por lo menos  diez datos antes de la revisión.
2. Se revisará cada una de las partes más de una vez.
3. No se revisará el programa en compiladores en línea, ni en Dev C++.
4. Programa que este copiado será descartado.
5. Al finalizar la entrega envíe el archivo a TEAMS de no ser así este no será tomado en cuenta.
6. No olvide el reporte.

**Contenido del PSP**

|  |  |  |  |
| --- | --- | --- | --- |
| Código fuente: | Estimado  (min) | Real  (min) | Error |
| Tiempo de Diseño | 120 | 90 | 25% |
| Tiempo de Codificación. | 360 | 300 | 17% |
| Tiempo estimado de fabricación | 480 | 390 | 19% |
| Total, de líneas de código nuevas. | 500 | 457 | 8% |
| Total, de líneas de código reusadas. | 100 | 89 | 11% |
| Total, de líneas de código del programa. | 580 | 546 | 6% |
| Cantidad de errores de compilación. | 20 | 16 | 20% |
| Cantidad de errores de ejecución. | 10 | 7 | 30% |

**Código fuente**

**AVL.cpp (main)**

#include <iostream>

#pragma once

#include "ArbolAVL.h"

#include "Nodo.h"

using namespace std;

int main()

{

Persona persona;

Nodo\* nodo = NULL;

ArbolAVL AVL;

char R = NULL;

while (R != 'S')

{

cout << "\nQue quieres hacer?\n[A] Insertar valor\n[B] Eliminar uno\n[C] Eliminar todo\n[D]Mostrar Int-Orden\n[E]Mostrar Pre-Orden\n[F]Mostrar Pos-Orden\n[G]Contar\n[H]Calcular altura\n[I]Buscar nodo\n[S] Salir\n" << endl;

cin >> R;

switch (R)

{

case 'A': //Insertar

{

AVL.Insertar(persona);

cout << "\nDatos insertados correctamente.\n" << endl;

}

break;

case 'B': //Eliminar uno

{

AVL.Eliminar(R);

}

break;

case 'C': //Eliminar todo

{

AVL.Eliminar(R);

}

break;

case 'D': //Mostrar intorden

cout << "Recorrido Int-Orden del Arbol AVL: \n";

AVL.Mostar(R);

break;

case 'E': //Mostrar preorden

{

cout << "Recorrido Pre-Orden del Arbol AVL: \n";

AVL.Mostar(R);

}

break;

case 'F': //Mostrar posorden

{

cout << "Recorrido Pro-Orden del Arbol AVL: \n";

AVL.Mostar(R);

}

break;

case 'G': //Contar

{

cout << "\nEl arbol cuenta con: " << AVL.Contar() << " nodos" << endl;

}

break;

case 'H': //Calcular altura

{

cout << "\nLa altura del arbol es de: " << AVL.CalcularAltura(nodo, NULL) << endl;

}

break;

case 'I': //Buscar elemento

{

AVL.BuscarNodo(nodo, NULL, R);

}

break;

case 'S':

{

cout << "\nQue tenga buen dia" << endl;

}

break;

default:

{

cout << "\nCreo que no se puede joven (-\_-) \n" << endl;

}

break;

}

}

return 0;

}

**Nodo.cpp**

#pragma once

#include <iostream>

#include <string>

using namespace std;

struct Persona

{

char nombre[50];

char fecha[12]; //DD/MM/AAAA

int registro = NULL;

};

class Nodo

{

public:

Persona persona;

Nodo\* der;

Nodo\* izq;

int altura;

Nodo(Persona persona);

};

**Nodo.h**

#include "Nodo.h"

Nodo::Nodo(Persona p)

{

persona = p;

der = NULL;

izq = NULL;

altura = 1;

}

**ArbolAVL.h**

#pragma once

#include <iostream>

#include "Nodo.h"

class ArbolAVL

{

Nodo\* raiz;

int TomarAltura(Nodo\* nodo);

int TomarBalance(Nodo\* nodo);

Nodo\* MinimoNodo(Nodo\* nodo);

Nodo\* RotacionDer(Nodo\* y);

Nodo\* RotacionDerIzq(Nodo\* nodo);

Nodo\* RotacionIzq(Nodo\* x);

Nodo\* RotacionIzqDer(Nodo\* nodo);

Nodo\* InsertarNodo(Nodo\* nodo, Persona persona);

Nodo\* EliminarNodo(Nodo\* nodo, int elimina);

void EliminarArbol(Nodo\* nodo);

void MostarInt(Nodo\* nodo);

void MostarPre(Nodo\* nodo);

void MostarPos(Nodo\* nodo);

public:

int cant = 0;

void BuscarNodo(Nodo\* nodo, int buscar, char R);

int CalcularAltura(Nodo\* nodo, int buscar);

ArbolAVL()

{

raiz = NULL;

}

void Insertar(Persona persona);

void Eliminar(char R);

void Mostar(char R);

int Contar();

};

**ArbolAVL.cpp**

#include "ArbolAVL.h"

#include "Nodo.h"

#include <iostream>

#include <string>

#include <string.h>

using namespace std;

int ArbolAVL::TomarAltura(Nodo\* nodo)

{

if (nodo == NULL)

{

return 0;

}

return nodo->altura;

}

int ArbolAVL::TomarBalance(Nodo\* nodo)

{

if (nodo == NULL)

{

return 0;

}

return (TomarAltura(nodo->izq) - TomarAltura(nodo->der));

}

Nodo\* ArbolAVL::RotacionDer(Nodo\* y)

{

Nodo\* x = y->izq;

Nodo\* T2 = x->der;

x->der = y;

y->izq = T2;

y->altura = 1 + max(TomarAltura(y->izq), TomarAltura(y->der));

x->altura = 1 + max(TomarAltura(x->izq), TomarAltura(x->der));

return x;

}

Nodo\* ArbolAVL::RotacionIzq(Nodo\* x)

{

Nodo\* y = x->der;

Nodo\* T2 = y->izq;

y->izq = x;

x->der = T2;

x->altura = 1 + max(TomarAltura(x->izq), TomarAltura(x->der));

y->altura = 1 + max(TomarAltura(y->izq), TomarAltura(y->der));

return y;

}

Nodo\* ArbolAVL::RotacionIzqDer(Nodo\* z)

{

z->izq = RotacionIzq(z->izq);

return RotacionDer(z);

}

Nodo\* ArbolAVL::RotacionDerIzq(Nodo\* z)

{

z->der = RotacionDer(z->der);

return RotacionIzq(z);

}

Nodo\* ArbolAVL::InsertarNodo(Nodo\* nodo, Persona persona)

{

if (nodo == NULL)

{

Nodo\* NewNodo = new Nodo(persona);

return NewNodo;

}

if (persona.registro < nodo->persona.registro)

{

nodo->izq = InsertarNodo(nodo->izq, persona);

}

else

{

if (persona.registro >= nodo->persona.registro)

{

nodo->der = InsertarNodo(nodo->der, persona);

}

else

{

return nodo;

}

}

nodo->altura = 1 + max(TomarAltura(nodo->izq), TomarAltura(nodo->der));

int BalanceFactor = TomarBalance(nodo);

if (BalanceFactor > 1)

{

if (persona.registro < nodo->izq->persona.registro)

{

return RotacionDer(nodo);

}

else

{

return RotacionIzqDer(nodo);

}

}

if (BalanceFactor < -1)

{

if (persona.registro > nodo->der->persona.registro)

{

return RotacionIzq(nodo);

}

else

{

return RotacionDerIzq(nodo);

}

}

return nodo;

}

void ArbolAVL::Insertar(Persona persona)

{

cant++;

cout << "Dame los datos de la persona\nRegistro:\t\t";

cin >> persona.registro;

cout << "Nombre:\t\t\t";

fgets(persona.nombre, 50, stdin);

fgets(persona.nombre, 50, stdin);

cout << "Fecha de nacimiento:\t";

fgets(persona.fecha, 12, stdin);

raiz = InsertarNodo(raiz, persona);

}

Nodo\* ArbolAVL::EliminarNodo(Nodo\* nodo, int elimina)

{

Nodo\* aux = NULL;

if (nodo == NULL)

{

cout << "No existe ese dato" << endl;

cant++;

return nodo;

}

if (elimina < nodo->persona.registro)

{

nodo->izq = EliminarNodo(nodo->izq, elimina);

}

if (elimina > nodo->persona.registro)

{

nodo->der = EliminarNodo(nodo->der, elimina);

}

if (nodo->izq == NULL || nodo->der == NULL)

{

if (nodo->izq != NULL)

{

aux = nodo->izq;

}

else

{

aux = nodo->der;

}

if (aux == NULL)

{

aux = nodo;

nodo = NULL;

}

else

{

\*nodo = \*aux;

}

delete aux;

}

else

{

aux = MinimoNodo(nodo->der);

nodo->persona = aux->persona;

nodo->der = EliminarNodo(nodo->der, aux->persona.registro);

}

if (nodo == NULL)

{

return nodo;

}

nodo->altura = 1 + max(TomarAltura(nodo->izq), TomarAltura(nodo->der));

int BalanceFactor = TomarBalance(nodo);

if (BalanceFactor > 1)

{

if (TomarBalance(nodo->izq) >= 0)

{

return RotacionDer(nodo);

}

else

{

nodo->izq = RotacionIzq(nodo->izq);

return RotacionDer(nodo);

}

}

if (BalanceFactor < -1)

{

if (TomarBalance(nodo->der) <= 0)

{

return RotacionIzq(nodo);

}

else

{

nodo->der = RotacionDer(nodo->der);

return RotacionIzq(nodo);

}

}

cout << "\nDato eliminado correctamente.\n" << endl;

return nodo;

}

Nodo\* ArbolAVL::MinimoNodo(Nodo\* nodo)

{

Nodo\* aux = nodo;

while (aux->izq != NULL)

{

aux = aux->izq;

}

return aux;

}

void ArbolAVL::EliminarArbol(Nodo\* nodo)

{

if (nodo == NULL)

{

return;

}

EliminarArbol(nodo->izq);

EliminarArbol(nodo->der);

delete nodo;

}

void ArbolAVL::Eliminar(char R)

{

if (raiz == NULL)

{

cout << "No se han ingresado datos para eliminar" << endl;

}

else

{

if(R == 'B')

{

int elimina;

cout << "Que dato quieres eliminar?" << endl;

cin >> elimina;

raiz = EliminarNodo(raiz, elimina);

cant--;

}

if (R == 'C')

{

EliminarArbol(raiz);

raiz = NULL;

cant = 0;

}

}

}

void ArbolAVL::MostarInt(Nodo\* nodo)

{

if (nodo != NULL)

{

MostarInt(nodo->izq);

cout << "\nRegistro:\t\t" << nodo->persona.registro << "\nNombre:\t\t\t" << nodo->persona.nombre << "Fecha de Nacimiento:\t" << nodo->persona.fecha << endl;

MostarInt(nodo->der);

}

}

void ArbolAVL::MostarPre(Nodo\* nodo)

{

if (nodo != NULL)

{

cout << "\nRegistro:\t\t" << nodo->persona.registro << "\nNombre:\t\t\t" << nodo->persona.nombre << "Fecha de Nacimiento:\t" << nodo->persona.fecha << endl;

MostarPre(nodo->izq);

MostarPre(nodo->der);

}

}

void ArbolAVL::MostarPos(Nodo\* nodo)

{

if (nodo != NULL)

{

MostarPos(nodo->izq);

MostarPos(nodo->der);

cout << "\nRegistro:\t\t" << nodo->persona.registro << "\nNombre:\t\t\t" << nodo->persona.nombre << "Fecha de Nacimiento:\t" << nodo->persona.fecha << endl;

}

}

void ArbolAVL::Mostar(char R)

{

if (R == 'D')

{

MostarInt(raiz);

}

if (R == 'E')

{

MostarPre(raiz);

}

if (R == 'F')

{

MostarPos(raiz);

}

}

int ArbolAVL::Contar()

{

return cant;

}

void ArbolAVL::BuscarNodo(Nodo\* nodo, int buscar, char R)

{

if (buscar == NULL)

{

cout << "Que nodo quiere buscar?" << endl;

cin >> buscar;

cout << "Desea ver la busqueda paso a paso?\n[S] Si\n[N] No" << endl;

cin >> R;

BuscarNodo(raiz, buscar, R);

return;

}

if (nodo == NULL)

{

cout << "No se encontro el dato en el arbol." << endl;

return;

}

if (buscar == nodo->persona.registro)

{

cout << "Registro:\t\t" << nodo->persona.registro << "\nNombre:\t\t\t" << nodo->persona.nombre << "Fecha de Nacimiento:\t" << nodo->persona.fecha << endl;

return;

}

if (buscar < nodo->persona.registro)

{

if (R == 'S')

{

cout << "Registro:\t\t" << nodo->persona.registro << "\nNombre:\t\t\t" << nodo->persona.nombre << "Fecha de Nacimiento:\t" << nodo->persona.fecha << endl;

}

BuscarNodo(nodo->izq, buscar, R);

}

else

{

if (R == 'S')

{

cout << "Registro:\t\t" << nodo->persona.registro << "\nNombre:\t\t\t" << nodo->persona.nombre << "Fecha de Nacimiento:\t" << nodo->persona.fecha << endl;

}

BuscarNodo(nodo->der, buscar, R);

}

}

int ArbolAVL::CalcularAltura(Nodo\* nodo, int altura) {

if (altura == NULL)

{

altura = 1;

CalcularAltura(raiz, altura);

}

else

{

if (nodo == NULL)

{

return 0;

}

int alturaIzquierda = CalcularAltura(nodo->izq, altura);

int alturaDerecha = CalcularAltura(nodo->der, altura);

return (altura + max(alturaIzquierda, alturaDerecha));

}

}