Logotipo, nombre de la empresa

Descripción generada automáticamenteÁrbol avl

Nombre: Oscar Alejandro Penilla Skakievich

Fecha: 7/12/22

Grupo:5D1

Registro: 20300701

**Descripción del programa:**

Se hará un árbol de datos en el cual se tendrán que contar, buscar, agregar, eliminar sus datos, además de poder hacer los recorridos en order, preorden y postorden

**Requerimientos:**

El programa debe de lograr hacer estas funciones

* Agregar. Añadir un elemento
* Eliminar uno. Eliminar un elemento
* Eliminar todos. Eliminar todos los elementos
* Buscar un elemento
* Contar. Cuenta la cantidad máxima de elementos
* Obtener  o mostrar. muestra en donde inician los elementos y en donde terminan
  + Preorden
  + Orden
  + Postorden

Además debe de estabilizar el mismo árbol como dicta el algoritmo de los arboles AVL

**PSP:**

|  |  |  |  |
| --- | --- | --- | --- |
| Contenido del PSP – avl | | | |
| Código fuente: | Estimado | Real | Error |
| (min) | (min) |
| Tiempo de Diseño | 30 | 60 | 100% |
| Tiempo de Codificación. | 100 | 600 | 500% |
| Tiempo estimado de fabricación | 130 | 660 | 408% |
| Total, de líneas de código nuevas. | 20 | 100 | 400% |
| Total, de líneas de código reusadas. | 500 | 499 | 0% |
| Total, de líneas de código del programa. | 520 | 599 | 15% |
| Cantidad de errores de compilación. | 70 | 100 | 43% |
| Cantidad de errores de ejecución. | 80 | 120 | 50% |

**Código fuente:**

#include <iostream>

using namespace std;

class hoja{

public:

hoja();

hoja(int x);

int info;

int alt;

hoja \*izq;

hoja \*der;

hoja \*padre;

};

hoja::hoja(){

info=0;

alt = 0;

izq=NULL;

der=NULL;

padre = NULL;

}

hoja::hoja(int x){

this->info=x;

alt= 0;

der=NULL;

izq=NULL;

padre=NULL;

}

//los de nodod depues tendrán que eliminarse, todo debe de ser entorno a hoja

class nodo{ //como no quiero que implosione mi codigo ccon un archivo .h, mejor agrego la lista aqui :c

public:

int x;

nodo();

nodo(int x);

nodo \*sig;

};

nodo::nodo(){

x=0;

sig=NULL;

}

nodo::nodo(int x){

this->x=x;

sig=NULL;

}

class lista{

private:

nodo \*i,\*a;

public:

lista();

void add(int x);

void del(int x);

void delall();

nodo\* busc(int x);

int cont();

void obte();

~lista();

};

lista::lista(){

a=NULL;

i=NULL;

}

void lista::add(int x){

if(i==NULL){

i = new nodo(x);

}else

a= new nodo(x);

nodo \*p=i;

while(p->sig !=NULL){

p=p->sig;

}

p->sig=a;

}

void lista::del(int x){

nodo\* p = NULL;

if(i){

a=busc(x);

if(a == i){

i=i->sig;

}else{

p=i;

while(p->sig != a){

p = p->sig;

}

p->sig = a->sig;

}

delete a;

}else{

cout << "No hay nada que eliminar"<<endl;

}

}

void lista::delall(){

if(i==NULL){

cout<<"Elementos eliminados"<<endl;

}

a=i;

while (i){

i=i->sig;

delete a;

a=i;

}

i=NULL;

a=NULL;

}

nodo\* lista::busc(int x){

a=i;

while(a){

if(a->x == x){

return a;

a=a->sig;

}

a=a->sig;

}

return NULL;

}

int lista::cont(){

int cont =0;

a=i;

while(a){

a=a->sig;

cont++;

}

return cont;

}

void lista::obte(){

a=i;

while(a){

// cout <<" Nodo: " << a <<", Valor del nodo: " <<a->x <<", A donde apunta "<< a->sig <<endl;

cout<< a->x << ", ";

a=a->sig;

}

cout << "\n";

}

lista::~lista(){

delall();

}

//hacemos nuestro arbol

class arbol{

private:

lista list;

hoja \*raiz, \*a, \*lob, \*p;

public:

hoja \*root;

arbol();

void add(int x, hoja \*nodo); //anadir un elemento

void del(int x, bool i, hoja \*nodo);//eliminar un elemento

void delall(hoja \*nodo);//eliminar todos los elementos

int busc(int x, bool i, hoja \*nodo);//buscar un elelemento por su valor

int contar(int cont, hoja \*nodo);//contrar los elementos

lista\* recorrer(char selec, lista &list, hoja \*nodo);//muestra en donde inician los elementos y en donde terminan

void orden(hoja \*nodo, lista &list);

void preorden(hoja \*nodo, lista &list);

void postorden(hoja \*nodo, lista &list);

//funciones para equilibrar el arbol

int equilibrio(hoja\* nodo);

void Actualizaralt(hoja\* nodo);

void rotacion(hoja\* nodo, bool izquierda);

void RotacionDoble(hoja\* nodo, bool izquierda);

void Balance(hoja\* nodo);

~arbol();

};

arbol::arbol(){

raiz=NULL;

a=NULL;

lob=NULL; //otra auxiliar

p = NULL;

//sigizq = a->izq;

//sigder = a->der;

}

void arbol::add(int x, hoja \*nodo = (hoja\*)1){

if(nodo == (hoja\*)1)

{

nodo= raiz;

}

if(raiz == NULL){ //si no tiene una raiz

raiz = new hoja(x);

raiz->padre = NULL;

}

else{ //si ya tiene una raiz

//se tiene que comparar si es mayor a la raiz, si es mayor se agrega a la derecha

if(x > nodo->info){

if(nodo->der == NULL){ //para saber si no tiene algo

nodo->der = new hoja(x);//como no tiene agregamos el valor

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

}

else{

add(x, nodo->der); //lo movemos ala derecha ya que ya sabemos que es mayor y si tiene algo

}

}

else{//si llega a ser menor

if(nodo->izq == NULL){ //para saber si no tiene algo

nodo->izq = new hoja(x);//como no tiene agregamos el valor

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

}

else{

add(x, nodo->izq); //lo movemos ala izquierda ya que ya sabemos que es menor y si tiene algo

}

}

}

Balance(nodo);

Actualizaralt(nodo);

root = raiz;

}

void arbol::del(int x, bool i, hoja \*nodo = (hoja\*)1){

if(nodo == (hoja\*)1)

{ nodo= raiz; }

i = busc(x,i, nodo);

if (i == 0){

cout << "Este dato no existe"<<endl;

}else if (i == 1) {

if (p == raiz) {

if (p->der == NULL && p->izq == NULL) {

delete p;

raiz = NULL;

p = NULL;

} else if (p->izq == NULL) {

raiz = p->der;

delete p;

p = raiz;

} else if (p->der == NULL) {

raiz = p->izq;

delete p;

p = raiz;

} else {

lob = p->izq;

while (lob->der != NULL) {

p = lob;

lob = lob->der;

}

raiz->info = lob->info;

if (lob->izq != NULL) {

p = lob->izq;

delete lob;

nodo->padre = NULL;

}

else {

delete lob;

p->izq = NULL;

nodo->padre = NULL;

}

}

}else if (p->izq == NULL && p->der == NULL) {

if (lob->izq == p){

lob->izq = NULL;}

else if (lob->der == p){

lob->der = NULL;}

delete p;

} else {

if (p->izq == NULL) {

lob->izq = p->der;

delete p;

p = NULL;

} else if (p->der == NULL){

lob->izq = p->izq;

delete p;

p = NULL;

} else{

lob = p;

while (lob->der != NULL)

lob = lob->der;

p->info = lob->info;

while (p->der != lob)

p = p->der;

delete lob;

p->der = NULL;

}

}

}

if (raiz == NULL){

root = NULL;

}

else{

cout <<"entra en el ordenamiento"<<endl;

Balance(nodo);

cout <<"balance"<<endl;

Actualizaralt(nodo);

cout <<"actualizar alt"<<endl;

root = raiz;

}

root = raiz;

}

void arbol::delall(hoja \*nodo = (hoja\*)1){

if(nodo == (hoja\*)1)

{

nodo = raiz;

}

if(raiz == NULL){

cout <<"Elementos eliminados"<<endl;

}

else{

if(nodo->izq != NULL){

delall(nodo->izq);

}

if(nodo->der != NULL){

delall(nodo->der);

}

delete nodo;

raiz = NULL;

}

}

int arbol::busc(int x, bool i, hoja \*nodo = (hoja\*)1){

if(nodo == (hoja\*)1)

{

nodo = raiz;

}

if (i == 1)

return 1;

if (nodo->info == x)

{

p = nodo;

i = 1;

}

else

{

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

lob = nodo;

i = 0;

}

if (nodo->izq != NULL)

i = busc(x, i, nodo->izq);

if (nodo->der != NULL)

i = busc(x, i, nodo->der);

return i;

}

int arbol::contar(int cont, hoja \*nodo = (hoja\*)1){

if(nodo == (hoja\*)1)

{

nodo= raiz;

}

if(raiz == NULL){

return 0;

}

else{

if (nodo == raiz){

cont++;

}

if(nodo->izq != NULL){

cont++;

cont = contar(cont, nodo->izq);

}

if(nodo->der != NULL){

cont++;

cont = contar(cont, nodo->der);

}

return cont;

}

}

lista\* arbol::recorrer(char selec, lista &list, hoja \*nodo = (hoja\*)1){

//cuando no hay nada que obtener

//cuando si hay algo, contando que deben aparecer todos unidos, sin espacios

if(nodo == (hoja\*)1)

{

nodo= raiz;

}

if(raiz == NULL){

cout<< "No hay elementos"<<endl;

return &list;

}

else{

switch(selec){

case 'a':

cout<<"Orden"<<endl;

orden(raiz, list);

break;

case 'b':

cout<<"Preorden"<<endl;

preorden(raiz, list);

break;

case 'c':

cout<<"Postorden"<<endl;

postorden(raiz, list);

break;

}

return &list;

}

}

void arbol::orden(hoja \*nodo, lista &list){

if(nodo->izq != NULL)

orden(nodo->izq, list /\*lista(El nombre)\*/);

list.add(nodo->info);

//agregar a la lista(nodo-<info)

if(nodo->der != NULL)

orden(nodo->der, list /\*lista(El nombre)\*/);

}

void arbol::preorden(hoja \*nodo, lista &list){

//agregar a la lista(nodo-<info)

list.add(nodo->info);

if(nodo->izq != NULL)

preorden(nodo->izq, list /\*lista(El nombre)\*/);

if(nodo->der != NULL)

preorden(nodo->der, list /\*lista(El nombre)\*/);

}

void arbol::postorden(hoja \*nodo, lista &list){

if(nodo->izq != NULL)

postorden(nodo->izq, list /\*lista(El nombre)\*/);

if(nodo->der != NULL)

postorden(nodo->der, list /\*lista(El nombre)\*/);

list.add(nodo->info);

//agregar a la lista(nodo-<info) listasimple.add(nodo->nfo)

}

int arbol::equilibrio(hoja\* nodo){

if(nodo == (hoja\*)1)

{ nodo= raiz; }

if (nodo == NULL)

return -1;

else

return nodo->alt;

}

void arbol::Actualizaralt(hoja\* nodo){

if(nodo == (hoja\*)1)

{ nodo= raiz; }

if (nodo != NULL)

nodo->alt = max(equilibrio(nodo->izq), equilibrio(nodo->der)) + 1;

}

void arbol::rotacion(hoja\* nodo, bool derecha){

if(nodo == (hoja\*)1)

{ nodo= raiz; }

bool existe = 0, r = 0;

hoja\* nodo1;

if (derecha) {

nodo1 = nodo->izq;

nodo->izq = nodo1->der;

nodo1->der = nodo;

}

else {

nodo1 = nodo->der;

nodo->der = nodo1->izq;

nodo1->izq = nodo;

}

p = raiz;

if (derecha == 1)

{

if (nodo->padre)

{

if (nodo->padre->der == nodo1->der)

{

nodo->padre->der = nodo1;

}

else

nodo->padre->izq = nodo1;

}

else

{

raiz = nodo1;

}

}

else

{

if (nodo->padre)

{

if (nodo->padre->izq == nodo1->izq)

{

nodo->padre->izq = nodo1;

}

else

nodo->padre->der = nodo1;

}

else

{

raiz = nodo1;

}

}

}

void arbol::RotacionDoble(hoja\* nodo, bool izquierda){

if(nodo == (hoja\*)1)

{ nodo= raiz; }

if (izquierda) {

rotacion(nodo->izq, false);

rotacion(nodo, true);

} else {// rotacion derecha

rotacion(nodo->der, true);

rotacion(nodo, false);

}

}

void arbol::Balance(hoja\* nodo){

if(nodo == (hoja\*)1)

{ nodo= raiz; }

if (nodo != NULL)

{

if (equilibrio(nodo->izq) - equilibrio(nodo->der) == 2)

{

if (equilibrio(nodo->izq->izq) >= equilibrio(nodo->izq->der))

rotacion(nodo, true);

else

RotacionDoble(nodo, true);

}

else if (equilibrio(nodo->der) - equilibrio(nodo->izq) == 2)

{

if (equilibrio(nodo->der->der) >= equilibrio(nodo->der->izq))

rotacion(nodo, false);

else

RotacionDoble(nodo, false);

}

}

}

arbol::~arbol(){

delall();

}

int main(){

//variables main

int op=0, x=0;

arbol ob1;

lista listaMain;

do{

cout<<"Elija una opcion"<<endl;

cout<<"1.-Agregar"<<endl;

cout<<"2.-Eliminar"<<endl;

cout<<"3.-Eliminar todos"<<endl;

cout<<"4.-Buscar"<<endl;

cout<<"5.-Contar"<<endl;

cout<<"6.-Recorrer"<<endl;

cout<<"7.-Salir"<<endl;

//

cin>> op;

switch(op){

case 1:

{

cout<<"1"<<endl;

cout<<"Valor a agregar:";

cin >> x;

ob1.add(x);

break;

}

case 2:

{

cout<<"2"<<endl;

cout<<"Valor a eliminar:";

cin >> x;

ob1.del(x, 0);

// cout<<"Valor eliminado"<<endl;

break;

}

case 3:

{

cout<<"3"<<endl;

ob1.delall();

break;

}

case 4:

{

cout<<"4"<<endl;

cout<<"Valor a buscar: ";

cin >> x;

// cout <<"hay????? "<<( ob1.busc(x, 0) )<<endl;

if(ob1.busc(x, 0) == 1){

cout << "Elemento encontrado: ";

cout << x <<endl;

}else{

cout << "Elemento no encontrado"<<endl;

}

break;

}

case 5:

{

cout<<"5"<<endl;

x = ob1.contar(0);

cout<<"Hay "<< x <<" elementos"<<endl;

break;

}

case 6:

{

char reco = 'S';

cout<<"6"<<endl;

cout<<"Elija el metodo de como recorrerá el arbol"<<endl;

cout<<"a.-Orden"<<endl;

cout<<"b.-Preorden"<<endl;

cout<<"c.-Postorden"<<endl;

cout<<"s.-Salir"<<endl;

cin >> reco;

ob1.recorrer(reco, listaMain);

listaMain.obte();

listaMain.delall();

break;

}

}

}while(op !=7);

return 0;

}