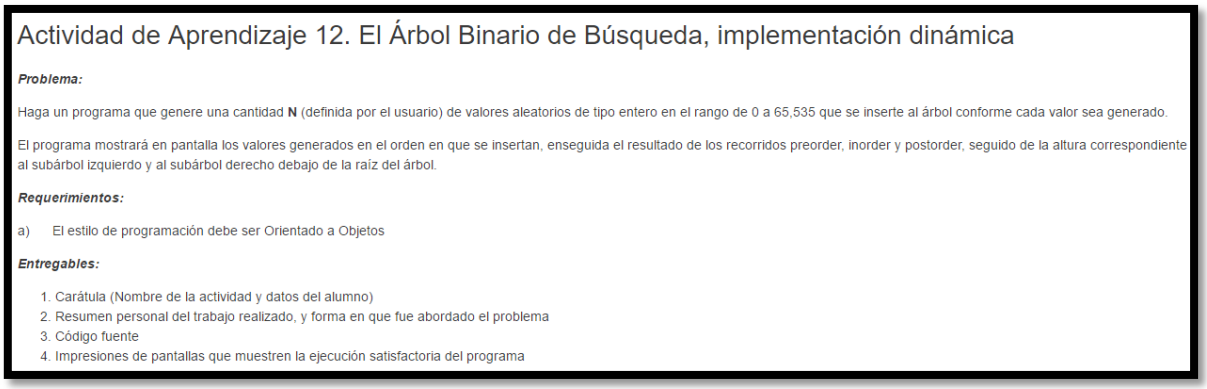
CENTRO UNIVERSITARIO DE CIENCIAS EXÁCTAS E INGENIERÍAS

Arboles Binarios

Tarea 12

Aldo Alexandro Vargas Meza 213495653

28/04/2017



Actividad de Aprendizaje 12. El Árbol Binario de Búsqueda, implementación dinámica

Problema:

Haga un programa que genere una cantidad **N** (definida por el usuario) de valores aleatorios de tipo entero en el rango de 0 a 65,535 que se inserte al árbol conforme cada valor sea generado.

El programa mostrará en pantalla los valores generados en el orden en que se insertan, enseguida el resultado de los recorridos preorder, inorder y postorder, seguido de la altura correspondiente al subárbol izquierdo y al subárbol derecho debajo de la raíz del árbol.

Requerimientos:

a) El estilo de programación debe ser Orientado a Objetos

Entregables:

1. Carátula (Nombre de la actividad y datos del alumno)
2. Resumen personal del trabajo realizado, y forma en que fue abordado el problema
3. Código fuente
4. Impresiones de pantallas que muestren la ejecución satisfactoria del programa

La actividad busca la implementación de un árbol de búsqueda binario con valores enteros, estos deben de estar en un rango de 0 – 65535 y se deben de mostrar conforme se inserten, después mostrar los diferentes órdenes que aplican a el árbol binario.

Para la implementación, se diseñó un nuevo modelo de Nodo, el cual cumple con las necesidades de un Árbol Binario.

Función Main

Dentro de la función main, hay una definición de función para la impresión la cual le agrega un separador a los datos a la hora de hacer los ordenamientos.

Fuera de eso, la clase main consiste en un while que enlaza el menú del programa, el cual implementa los aspectos a revisar del programa.

Clase Nodo

Para el manejo de los datos por medio de punteros, fue necesaria la entidad nodo, que consta del tipo de dato, y dos direcciones hacia otros elementos, que se manejaran como hojas. Siguiendo esta lógica, se programó una clase Nodo, que contiene ambas direcciones hacía la izquierda y derecha en forma de puntero y un dato almacenado. Esta entidad conformará cada uno de los datos de la lista ligada.

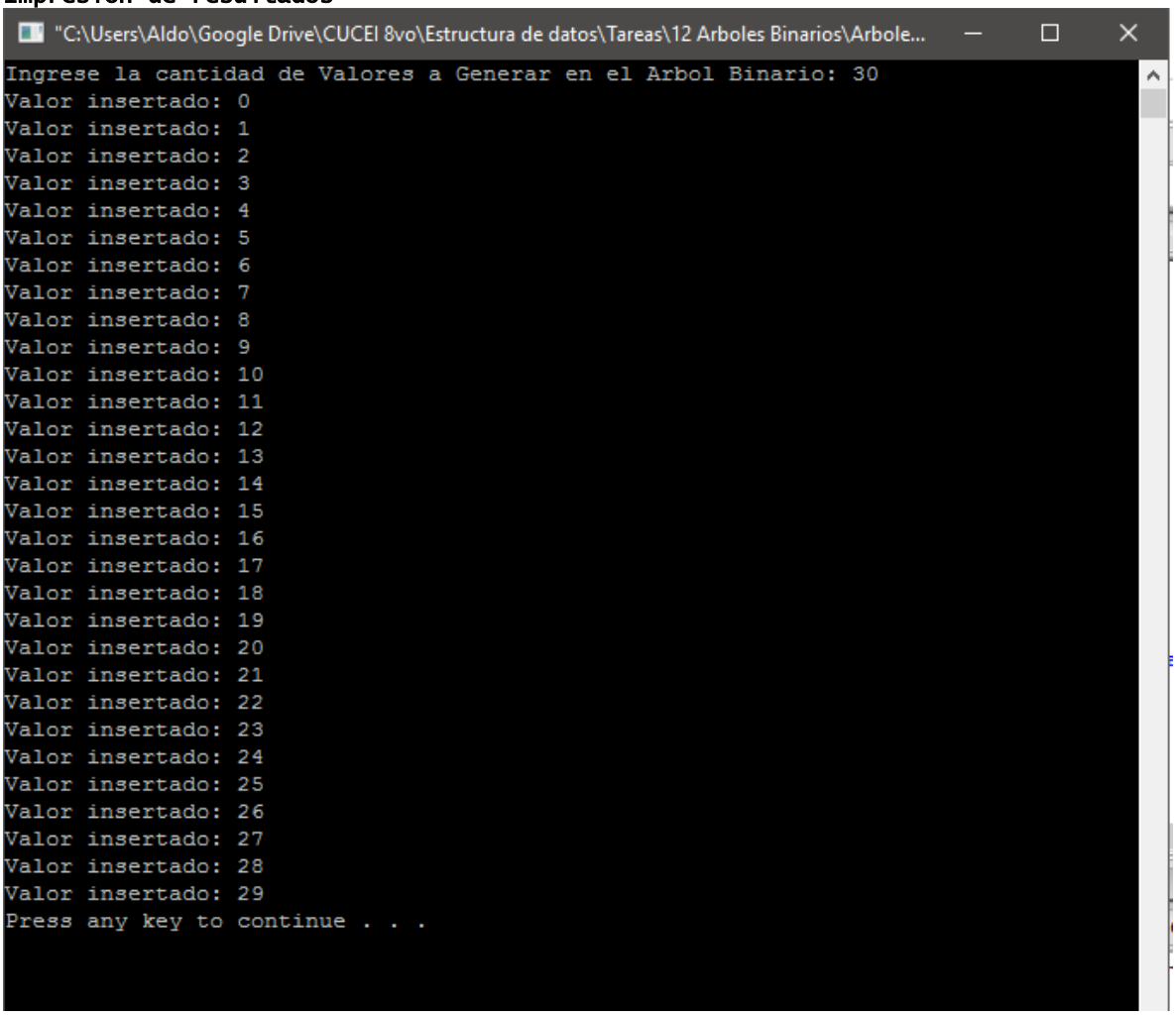
Además de dos constructores, la entidad Nodo entre sus métodos contiene getters y setters para los atributos y una función especial de impresión.

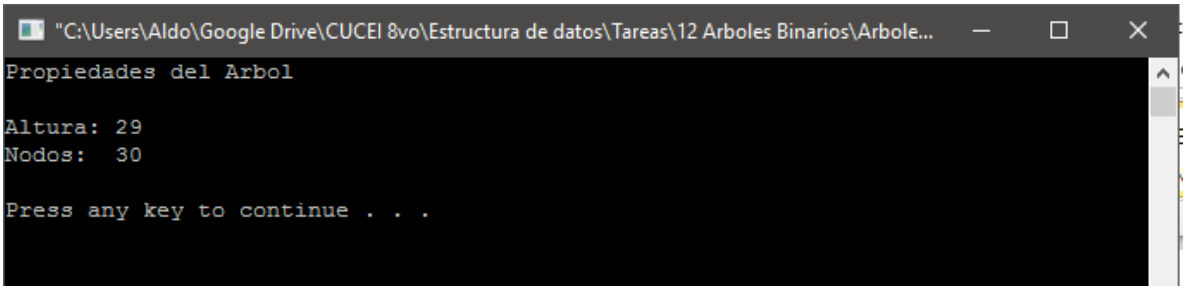
Clase Árbol

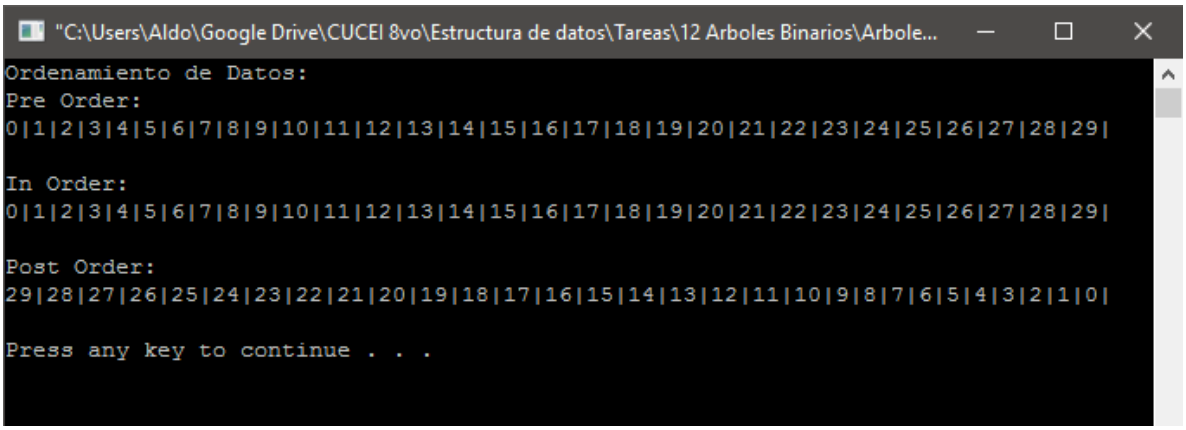
La clase árbol es una entidad que se podría pensar como una lista con diferentes niveles los cuales hacen que la implementación de este sea relativamente sencilla debido a su misma estructura de decisión cierto – falso.

Contiene funciones parecidas a la lista, y se maneja enteramente con un nodo principal, que es compuesto por un puntero del tipo Nodo dentro de los atributos privados.

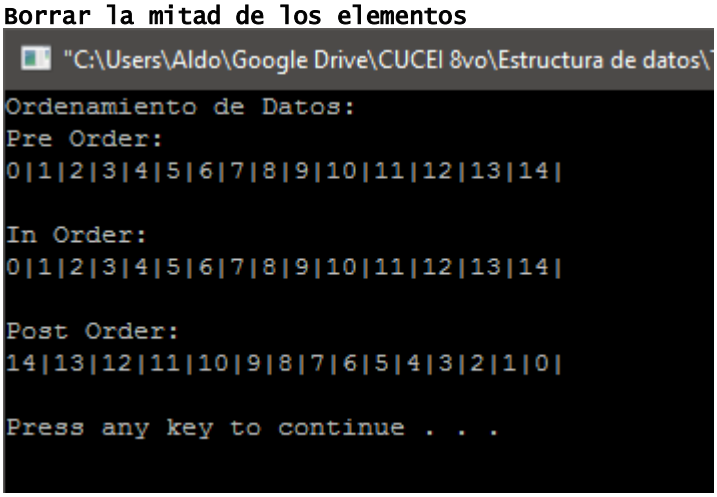
Impresión de resultados







Borrar la mitad de los elementos



Al momento de hacer la inserción con los números aleatorios, a pesar de ciclar correctamente en donde se aplicaba el numero aleatorio, las inserciones eran incorrectas el 90% de los resultados. Por esa razón el programa se simuló con números enteros no aleatorios.

Códigos

```
#include <iostream>
#include<stdlib.h>
#include<time.h>

#include "binaryTree.h"

using namespace std;

void Mostrar(int d) {
    cout << d << "|";
    }

int main() {
    int maxV = 0;
    binaryTree t;

    int opc = 0;

    while(opc != 5) {
        switch(opc) {
            case 0:
                cout<< "Ingrese la cantidad de Valores a Generar en el Arbol
Binario: ";
                cin >> maxV;

                for(int i=0; i<=maxV-1; i++) {
                    cout<<"Valor insertado: "<<i<<endl;
                    t.treeInsert(i);
                    }
                system("pause");
                system("cls");
                opc++;
                break;

            case 1:
                cout<< "Propiedades del Arbol" <<endl<<endl;

                cout<< "Altura: " << t.treeHeight() <<endl;
                cout<< "Nodos:  " << t.nodeCounter() <<endl<<endl;

                system("pause");
                system("cls");
                opc++;

                break;

            case 2:
                cout<<"Ordenamiento de Datos: "<<endl;

                cout<< "Pre Order: "<<endl;
                t.treePreOrder(Mostrar);
                cout<<endl<<endl;

                cout<< "In Order: "<<endl;
                t.treeInOrder(Mostrar);
                cout<<endl<<endl;

                cout<< "Post Order: "<<endl;
                t.treePostOrder(Mostrar);
                cout<<endl<<endl;

                system("pause");
                system("cls");
                opc++;


                break;

            case 3:
                cout<<"Borrado de la Mitad de los elementos: "<<endl;
```

```
                for(int i=maxV/2; i<maxV; i++) {
                    t.treeDelete(i);
                    }

                system("pause");
                system("cls");
                opc++;

                break;

            case 4:
                cout<<"Ordenamiento de Datos: "<<endl;

                cout<< "Pre Order: "<<endl;
                t.treePreOrder(Mostrar);
                cout<<endl<<endl;

                cout<< "In Order: "<<endl;
                t.treeInOrder(Mostrar);
                cout<<endl<<endl;

                cout<< "Post Order: "<<endl;
                t.treePostOrder(Mostrar);
                cout<<endl<<endl;

                system("pause");
                system("cls");
                opc++;
                break;



            }

        }
    return 0;
    }

#ifndef BINARYTREE_H_INCLUDED
#define BINARYTREE_H_INCLUDED

#include "node.h"

class binaryTree {
      private:
      int i;
      int height;
      Node* root;
      Node* actual;

      void pruneTree(Node* node);
      void auxCounter(Node* n);
      void auxHeight(Node* n, int e);

      public:

      binaryTree();
        ~binaryTree();

        void treeInsert(const int elem);
        void treeDelete(const int elem);

        bool isSearch(const int elem);
        bool isEmpty(Node* n);
        bool isLeaf(Node* n);

        const int nodeCounter();
        const int treeHeight();

        int getElemHeight(const int elem);
        int getActElem();
```

```
        void goToRoot();

             void treePreOrder(void (*func)(int) , Node* nodo= nullptr, bool
r=true);
        void treeInOrder(void (*func)(int) , Node* nodo= nullptr, bool r=true);
        void treePostOrder(void (*func)(int) , Node* nodo= nullptr, bool r=true);

};

#endif // BINARYTREE_H_INCLUDED

#include "binaryTree.h"

void binaryTree::pruneTree(Node* node) {
    if(node) {
        pruneTree(node->getSubIzq());
        pruneTree(node->getSubDer());
        delete node;
        node = nullptr;
        }

    }

void binaryTree::auxCounter(Node* n) {
    i++;
    if(n->getSubIzq())
        auxCounter(n->getSubIzq());
    if(n->getSubDer())
        auxCounter(n->getSubDer());

    }

void binaryTree::auxHeight(Node* n, int e) {
    if(n->getSubIzq())
        auxHeight(n->getSubIzq(), e+1);
    if(n->getSubDer())
        auxHeight(n->getSubDer(), e+1);

    if(isLeaf(n) && e > height)
        height = e;
    }

binaryTree::binaryTree() {
    root = nullptr;
    actual = nullptr;
    }

binaryTree::~binaryTree() {
    pruneTree(root);
    }

void binaryTree::treeInsert(const int elem) {
    Node* father = nullptr;
    actual = root;

    while(!isEmpty(actual) && elem != actual->getData() ) {
        father = actual;

        if(elem < actual->getData())
            actual = actual->getSubIzq();
        else
            actual = actual->getSubDer();
        }

    if(!isEmpty(actual))
        return;

    if(isEmpty(father))
        root = new Node(elem, nullptr,nullptr);

    else if(elem < father->getData())
        father->setSubDer(new Node(elem,nullptr,nullptr));
```

```
    else if(elem > father->getData())
        father->setSubDer(new Node(elem,nullptr,nullptr));

    }

void binaryTree::treeDelete(const int elem) {
    Node* father;
    Node* node;
    int aux;

    actual = root;

    while(!isEmpty(actual)) {
        if(elem == actual->getData()) {
            if(isLeaf(actual)) {
                if(father) {
                    if(father->getSubDer() == actual)
                        father->setSubDer(nullptr);
                    else if(father->getSubIzq() == actual)
                        father->setSubIzq(nullptr);
                    }
                delete actual;
                return;
                }

            else {
                father = actual;

                if(actual->getSubDer()) {
                    node = actual->getSubDer();
                    while(node->getSubIzq()) {
                        father = node;
                        node = node->getSubIzq();
                        }
                    }
                else {
                    node = actual->getSubIzq();
                    while(node->getSubDer()) {
                        father = node;
                        node = node->getSubDer();
                        }
                    }

                aux = actual->getData();
                actual->setData(node->getData());
                node->setData(aux);
                actual = node;
                }

            }
        else {
            father = actual;
            if(elem < actual->getData())
                actual = actual->getSubIzq();
            else if(elem > actual->getData())
                actual = actual->getSubDer();
            }


        }
    }

bool binaryTree::isSearch(const int elem) {
    actual = root;

    while(!isEmpty(actual)) {
        if(elem == actual->getData())
            return true;
        else if(elem < actual->getData())
            actual = actual->getSubIzq();
        else if(elem > actual->getData())
```

```
            actual = actual->getSubDer();

        }
    return false;
    }

bool binaryTree::isEmpty(Node* n) {
    return (n == nullptr);
    }

bool binaryTree::isLeaf(Node* n) {
    return (n->getSubDer() == nullptr && n->getSubIzq() == nullptr);
    }

const int binaryTree::nodeCounter() {
    i = 0;

    auxCounter(root);
    return i;
    }

const int binaryTree::treeHeight() {
    height = 0;

    auxHeight(root,0);
    return height;
    }

int binaryTree::getElemHeight(const int elem) {
    int altura = 0;
    actual = root;

    while(!isEmpty(actual)) {
        if(elem == actual->getData())
            return altura;
        else {
            altura++;
            if(elem < actual->getData())
                actual = actual->getSubIzq();
            else if(elem > actual->getData())
                actual = actual->getSubDer();

            }

        }
    return -1;
    }

int binaryTree::getActElem() {
    return actual->getData();

    }

void binaryTree::goToRoot() {
    actual = root;
    }

void binaryTree::treePreOrder(void (*func)(int), Node* node, bool r) {
    if(r)
        node = root;
    func(node->getData());
    if(node->getSubIzq())
        treePreOrder(func,node->getSubIzq(), false);
    if(node->getSubDer())
        treePreOrder(func,node->getSubDer(), false);
    }

void binaryTree::treeInOrder(void (*func)(int), Node* node, bool r) {
    if(r)
        node = root;
    if(node->getSubIzq())
        treeInOrder(func, node->getSubIzq(),false);
```

```
    func(node->getData());
    if(node->getSubDer())
        treeInOrder(func, node->getSubDer(), false);
    }

void binaryTree::treePostOrder(void (*func)(int), Node* node, bool r) {
    if(r)
        node = root;
    if(node->getSubIzq())
        treePostOrder(func, node->getSubIzq(),false);
    if(node->getSubDer())
        treePostOrder(func, node->getSubDer(), false);

    func(node->getData());
    }

#ifndef NODE_H_INCLUDED
#define NODE_H_INCLUDED
#include <iostream>

using namespace std;


class Node {

    private:
        int data;

        Node* subIzq;
        Node* subDer;

    public:

        Node();
        Node(int d, Node* sI, Node* sD);

        int  getData();
        Node* getSubIzq();
        Node* getSubDer();
        Node* getParent();

        void setData(int d);
        void setSubIzq(Node* sI);
        void setSubDer(Node* sD);
        void setParent(Node* p);

        void printNode();
    };

#endif // NODE_H_INCLUDED

#include "node.h"

Node::Node() {
    subIzq = nullptr;
    subDer = nullptr;
    }

Node::Node(int d,  Node* sI, Node* sD) {
    data = d;
    subIzq = sI;
    subDer = sD;
    }


int Node::getData() {
    return data;
    }

Node* Node::getSubIzq() {
    return subIzq;
    }
```

```
Node* Node::getSubDer() {
    return subDer;
    }


void Node::setData(int d) {
    data = d;
    }

void Node::setSubIzq(Node* sI) {
    subIzq = sI;
    }

void Node::setSubDer(Node* sD) {
    subDer = sD;
    }


void Node::printNode() {
    cout<< "_____"<<endl;
    cout<< "Addr: " << this << " -> Izq: " << subIzq <<" -> Der: " << subDer <<
endl;
    cout<< "Valor: " << data <<endl;
    }
```