UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

Estructuras computacionales

Ingeniería en computación inteligente

Unidad IV: ESTRUCTURA DE ARBOLES + PROGRAMACION ARBOLES ATS (Parcial 3)

Nombre del maestro: Miguel Ángel Meza de Luna.

Nombre de los alumnos:

Ángel David Ortiz Quiroz ID: 261481

Ximena Rivera Delgadillo ID: 261261

Erick Iván Ramírez Reyes ID: 260806

Diego Emanuel Saucedo Ortega ID: 261230

Jose Luis Sandoval Perez ID: 261731

Carlos Daniel Torres Macias ID: 244543

Fecha de entrega: domingo 22 de Mayo del 2021.

CODIGO PROGRAMACION ARBOLES ATS

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

using namespace std;

struct Nodo{
      int dato;
      Nodo *der;
      Nodo *izq;
      Nodo *padre;
};
Nodo *arbol=NULL;

Nodo *crearNodo (int, Nodo *);
void insertar(Nodo *&,int,Nodo *);
void mostrar(Nodo *,int);
bool busqueda(Nodo *, int);
void preorden(Nodo *);
void inorden(Nodo *);
void postorden(Nodo *);
void eliminar(Nodo *,int);
void eliminarNodo(Nodo *);
Nodo *minimo(Nodo *);
void reemplazar(Nodo *, Nodo* );
void destruir(Nodo *);
```

```
void menu();

int main(){
	menu();

	getchar();
	return 0;
}
Nodo *crearNodo(int n,Nodo *padre){
	Nodo *nuevo_nodo = new Nodo();

	nuevo_nodo->dato=n;
	nuevo_nodo->der=NULL;
	nuevo_nodo->izq=NULL;
	nuevo_nodo->padre=padre;
	return nuevo_nodo;
}
void insertar(Nodo *&arbol, int n,Nodo *padre){
	if (arbol==NULL){
		Nodo *nuevo_nodo=crearNodo(n,padre);
		arbol=nuevo_nodo;
	}
	else{
		int valorRaiz=arbol->dato;
		if (n<valorRaiz){
			insertar(arbol->izq,n,padre);
		}
		else{
```

```
                insertar(arbol->der,n,padre);
            }
        }
}
void mostrar(Nodo *arbol,int cont){
        if (arbol==NULL){
            return;
        }
        else{
            mostrar(arbol->der,cont+1);
            for (int i=0;i<cont;i++){
                cout<<"  ";
            }
            cout<<arbol->dato<<endl;
            mostrar(arbol->izq,cont+1);
        }
}
bool busqueda(Nodo *arbol, int n){
        if(arbol==NULL){
            return false;
        }
        else if (arbol->dato==n){
            return true;
        }
        else if (n<arbol->dato){
            return busqueda(arbol->izq,n);
        }
        else{
```

```
            return busqueda(arbol->der,n);
      }
}
void preorden(Nodo *arbol){
      if (arbol==NULL){
            return;
      }
      else{
            cout<<arbol->dato<<" - ";
            preorden(arbol->izq);
            preorden(arbol->der);
      }
}
void inorden(Nodo *arbol){
      if (arbol=NULL){
            return;
      }
      else{
            inorden(arbol->izq);
            cout<<arbol->dato<<" - ";
            inorden(arbol->der);
      }
}
void postorden(Nodo *arbol){
      if (arbol==NULL){
            return;
      }
      else{
```

```
            postorden(arbol->izq);
            postorden(arbol->der);
            cout<<arbol->dato<<" - ";
      }
}
void eliminar(Nodo *arbol,int n){
      if (arbol==NULL){
            return;
      }
      else if(n<arbol->dato){
            eliminar(arbol->izq,n);
      }
      else if(n>arbol->dato){
            eliminar(arbol->der,n);
      }
      else{
            eliminar(arbol,n);
      }
}
Nodo *minimo(Nodo *arbol){
      if (arbol==NULL){
            return NULL;
      }
      if(arbol->izq){
            return minimo(arbol->izq);
      }
      else{
            return arbol;
```

```
	}
}
void eliminarNodo(Nodo *nodoEliminar){
	if (nodoEliminar->izq && nodoEliminar->der){
		Nodo *menor=minimo(nodoEliminar->der);
		nodoEliminar->dato=menor->dato;
		eliminarNodo(menor);
	}
	else if(nodoEliminar->izq){
		reemplazar(nodoEliminar,nodoEliminar->izq);
		destruir(nodoEliminar);
	}
	else if(nodoEliminar->der){
		reemplazar(nodoEliminar, nodoEliminar->der);
		destruir(nodoEliminar);
	}
	else{
		reemplazar(nodoEliminar,NULL);
		destruir(nodoEliminar);
	}
}
void reemplazar(Nodo *arbol, Nodo *nuevoNodo){
	if(arbol->padre){
		if(arbol->dato==arbol->padre->izq->dato){
			arbol->padre->izq=nuevoNodo;
		}
		else if(arbol->dato==arbol->padre->der->dato){
			arbol->padre->der=nuevoNodo;
```

```
            }
      }
      if(nuevoNodo){
            nuevoNodo->padre=arbol->padre;
      }
}

void destruir(Nodo *nodo){
      nodo->izq=NULL;
      nodo->der=NULL;
      delete nodo;
}

void menu(){
      int dato,cont=0,op;
      do{
            cout<<"\t.:MENU:."<<endl;
            cout<<"1. Insertar un nuevo nodo"<<endl;
            cout<<"2. Mostrar el arbol completo"<<endl;
            cout<<"3. Buscar un elemento en el arbol"<<endl;
            cout<<"4. Recorrer el arbol en PREORDEN"<<endl;
            cout<<"5. Recorrer el arbol en INORDEN"<<endl;
            cout<<"6. Recorrer el arbol en POSTORDEN"<<endl;
            cout<<"7. Eliminar un nodo del arbol"<<endl;
            cout<<"8. Salir"<<endl;
            cout<<"Opcion: ";
            cin>>op;
```

```
          switch (op){
           case 1:
                cout<<"\nInserte un numero: ";
                cin>>dato;
                insertar(arbol,dato,NULL);
                cout<<"\n";
                system("pause");
                break;
           case 2:
                cout<<"\nMostrando el arbol completo:\n\n";
                mostrar(arbol,cont);
                cout<<"\n";
                system("pause");
                break;
           case 3:
                cout<<"\nIngresa el elemento a buscar: ";
                cin>>dato;
                if (busqueda(arbol,dato)==true){
                     cout<<"\nElemento "<<dato<<" ha sido
encontrado."<<endl;
                }
                else{
                     cout<<"\nElemento no encontrado."<<endl;
                }
                break;
           case 4:
                cout<<"\nRecorriendo en Preorden: \n";
                preorden(arbol);
                cout<<"\n\n";
```

```
            system("pause");
            break;
      case 5:
            cout<<"\nRecorriendo en Inorden: \n";
            inorden(arbol);
            cout<<"\n\n";
            system("pause");
            break;
      case 6:
            cout<<"\nRecorrido en Postorden: \n";
            postorden(arbol);
            cout<<"\n\n";
            system("pause");
            break;
      case 7:
            cout<<"\nDigite el numero a eliminar: ";
            cin>>dato;
            eliminar(arbol,dato);
            cout<<"\n";
            system ("pause");
            break;
      case 8:
            cout<<"\nAdios";
            cout<<"\n";
            system ("pause");
            break;
}
system ("cls");
```

```
        }while(op!=8);

    }
```

PROBLEMAS APLICACIONES DE ARBOLES

Los problemas de los códigos presentados fueron extraídos del recurso utilizado en clase el libro de Joyanes c++.

Problema 16.2

Escribir un programa que lea un texto de longitud indeterminada y que produzca como resultado la lista de todas las palabras diferentes contenidas en el texto, así como su frecuencia de aparición. Hacer uso de la estructura árbol binario de búsqueda, cada nodo del árbol que tenga una palabra y su frecuencia.

```
#include <iostream>
#include <stdio.h>
#include <string>
#include <stdlib.h>
using namespace std;

struct Nodo{
    string palabra;
    int frecuencia;
    Nodo *izq;
    Nodo *der;
    Nodo *padre;
};

Nodo *raiz = NULL;
```

```
//uso de nodos
Nodo *crearNodo(string,Nodo *);
void insertarNodo(Nodo *&,string,Nodo *);
void recorrerInorden(Nodo *);
bool buscarNodo(Nodo*,string);


int main(){
    string h; char c;
    cout<<"Ingrese texto:\n";
    cout<<"(Para finalizar ingrese SPACE, despues ENTER seguido de
ctrl+Z y presione ENTER de nuevo)\n";
    fflush(stdin);
    while(scanf("%c",&c)!=EOF){ h+=c; };
    cout<<endl;


    string act="";
    for(auto x:h){
        if(x!=' '){ act+=x; }
        else{
            bool existe = buscarNodo(raiz,act);
            if(existe == false){
                insertarNodo(raiz,act,NULL);
            }
            act="";
        }
    }
    cout<<"Lista de repeticion:\n";
    recorrerInorden(raiz);
```

```
      cout << "\n        .:PROGRAMA REALIZADO POR :.          ";
      cout << "\nAngel David Ortiz Quiroz            ID:261481";
      cout << "\nErick Ivan Ramirez Reyes            ID:260806";
      cout << "\nXimena Rivera Delgadillo            ID:261261";
      cout << "\nJose Luis Sandoval Perez            ID:261731";
      cout << "\nDiego Emanuel Saucedo Ortega        ID:261230";
      cout<< "\nCarlos Daniel Torres Macias           ID: 244543";
      cout << "\n";

return 0;
}

Nodo *crearNodo(string n,Nodo *padre){

    Nodo *nuevo = new Nodo();

    nuevo->palabra = n;
    nuevo->frecuencia = 1;
    nuevo->izq = NULL;
    nuevo->der = NULL;
    nuevo->padre = padre;
    return nuevo;
}

void insertarNodo(Nodo *&arbol,string n, Nodo *padre){
    if(arbol==NULL){
        arbol = new Nodo();
```

```
        arbol = crearNodo(n,padre);
    }
    else if(n != arbol->palabra){
        string valorRaiz = arbol->palabra;

        if(n<valorRaiz){ insertarNodo(arbol->izq,n, arbol); }
        else{ insertarNodo(arbol->der, n, arbol); }
    }
}

void recorrerInorden(Nodo *arbol){
    if(arbol==NULL){return;}
    recorrerInorden(arbol->izq);
    cout<<arbol->palabra<<" "<<arbol->frecuencia<<endl;
    recorrerInorden(arbol->der);
}

bool buscarNodo(Nodo *arbol,string busca){
    if(arbol==NULL){ return false; }
    else if(busca==arbol->palabra){ arbol->frecuencia++; return
true; }
    else if(busca<arbol->palabra){buscarNodo(arbol->izq,busca);}
    else{ buscarNodo(arbol->der,busca); }//no esta vacio
    return false;
}
```

Problema 16.3

Escribir un programa que procese un árbol binario cuyos nodos contengan caracteres y a partir del siguiente menú de opciones:

I (seguido de un carácter) : Insertar un carácter

B (seguido de un carácter) : Buscar un carácter

RE : Recorrido en orden

RP : Recorrido en preorden

RT : Recorrido postorden

SA : Salir

```
// 16.3 arbol binario caracteres.cpp : Este archivo contiene la
función "main". La ejecución del programa comienza y termina ahí.
//
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
using namespace std;
struct Nodo {
      char dato;
      Nodo* der;
      Nodo* izq;
      Nodo *padre;
};
Nodo* arbol = NULL;

Nodo* crearNodo(char ,Nodo *);
void insertarNodo(Nodo*&,char,Nodo *);
bool busqueda(Nodo*,char);
```

```
void preorden(Nodo*);
void inorden(Nodo*);
void postorden(Nodo*);
void menu();

int main() {
      menu();

      getchar();
      return 0;
}

Nodo *crearNodo(char n,Nodo *padre){

    Nodo *nuevo = new Nodo();

    nuevo->dato = n;
    nuevo->izq = NULL;
    nuevo->der = NULL;
    nuevo->padre = padre;
    return nuevo;
}

void insertarNodo(Nodo *&arbol,char n, Nodo *padre){
    if(arbol==NULL){
        arbol = new Nodo();
        arbol = crearNodo(n,padre);
    }
```

```
    else{
        char valorRaiz = arbol->dato;

        if(n<valorRaiz){ insertarNodo(arbol->izq,n, arbol); }
        else{ insertarNodo(arbol->der, n, arbol); }
    }
}

bool busqueda(Nodo *arbol,char busca){
    if(arbol==NULL){ return false; }
    else if(busca==arbol->dato){ return true; }
    else if(busca<arbol->dato){busqueda(arbol->izq,busca);}
    else{ busqueda(arbol->der,busca); }//no esta vacio
    return false;
}
void preorden(Nodo* arbol) {
     if (arbol == NULL) {
          return;
     }
     else {
          cout << arbol->dato << " - ";
          preorden(arbol->izq);
          preorden(arbol->der);
     }
}
void inorden(Nodo* arbol) {
     if (arbol = NULL) {
          return;
```

```
	}
	else {
		inorden(arbol->izq);
		cout << arbol->dato << " - ";
		inorden(arbol->der);
	}
}
void postorden(Nodo* arbol) {
	if (arbol == NULL) {
		return;
	}
	else {
		postorden(arbol->izq);
		postorden(arbol->der);
		cout << arbol->dato << " - ";
	}
}
void menu() {
	int cont = 0;
	string op;
	do {
		cout << "\t.:MENU:." << endl;
		cout << "I: Insertar un nuevo nodo" << endl;
		cout << "B: Buscar un elemento en el arbol" << endl;
		cout << "RE: Recorrer el arbol en INORDEN" << endl;
		cout << "RP: Recorrer el arbol en PREORDEN" << endl;
		cout << "RT: Recorrer el arbol en POSTORDEN" << endl;
		cout << "SA: Salir" << endl;
```

```
            cout << "Seleccion--->";
            cin >> op;

            if(op=="I"){
             char c;
             cout << "\nInserte un caracter: ";
                 cin >> c;
                 insertarNodo(arbol,c,NULL);
                 cout << "\n";
            }//caso insertar

            if(op=="B"){
             char c;
                 cout << "\nIngresa caracter a buscar: ";
                 cin >> c;
                 if (busqueda(arbol, c) == true) {
                       cout << "\n" << c << " ha sido encontrado."
<< endl;
                 }
                 else {
                       cout << "\nElemento no encontrado." << endl;
                 }
            }//caso buscar

        if(op=="RE"){
             cout << "\nRecorriendo en Preorden: \n";
                 preorden(arbol);
                 cout << "\n\n";
```

```
        }//recorrido en orden


           if(op=="RP"){
            cout << "\nRecorriendo en Inorden: \n";
                inorden(arbol);
                cout << "\n\n";
           }//recorrido en preorden

           if(op=="RT"){
            cout << "\nRecorrido en Postorden: \n";
                postorden(arbol);
                cout << "\n\n";
           }//recorrido en postorden

           if(op=="SA"){
                cout << "\n        .:PROGRAMA REALIZADO POR :.
";
                cout << "\nAngel David Ortiz Quiroz
ID:261481";
                cout << "\nErick Ivan Ramirez Reyes
ID:260806";
                cout << "\nXimena Rivera Delgadillo
ID:261261";
                cout << "\nJose Luis Sandoval Perez
ID:261731";
                cout << "\nDiego Emanuel Saucedo Ortega
ID:261230";
cout<< “\nCarlos Daniel Torres Macias           ID: 244543”;
                cout << "\n";
```

```
            }//Salir

            system("pause");
            system("cls");
      } while (op != "SA");
}
```

Problema 16.5

Construir una función en la clase ArbolBinarioBusqueda que encuentre el nodo máximo.

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

using namespace std;

struct Nodo{
      int dato;
      Nodo *der;
      Nodo *izq;
      Nodo *padre;
};


Nodo *arbol=NULL;




Nodo *crearNodo (int, Nodo *);
void insertar(Nodo *&,int,Nodo *);
void mostrar(Nodo *,int);
```

```
bool busqueda(Nodo *, int);
void preorden(Nodo *);
void inorden(Nodo *);
void postorden(Nodo *);
void eliminar(Nodo *,int);
void eliminarNodo(Nodo *);
Nodo *minimo(Nodo *);
void reemplazar(Nodo *, Nodo* );
void destruir(Nodo *);
void menu();



int main(){
      menu();

      cout << "\n        .:PROGRAMA REALIZADO POR :.          ";
      cout << "\nAngel David Ortiz Quiroz            ID:261481";
      cout << "\nErick Ivan Ramirez Reyes            ID:260806";
      cout << "\nXimena Rivera Delgadillo            ID:261261";
      cout << "\nJose Luis Sandoval Perez            ID:261731";
      cout << "\nDiego Emanuel Saucedo Ortega        ID:261230";
      cout<< “\nCarlos Daniel Torres Macias           ID: 244543”;
      cout << "\n";

      getchar();
      return 0;
}
```

```
Nodo *crearNodo(int n,Nodo *padre){
      Nodo *nuevo_nodo = new Nodo();

      nuevo_nodo->dato=n;
      nuevo_nodo->der=NULL;
      nuevo_nodo->izq=NULL;
      nuevo_nodo->padre=padre;
      return nuevo_nodo;
}
void insertar(Nodo *&arbol, int n,Nodo *padre){
      if (arbol==NULL){
            Nodo *nuevo_nodo=crearNodo(n,padre);
            arbol=nuevo_nodo;
      }
      else{
            int valorRaiz=arbol->dato;
            if (n<valorRaiz){
                  insertar(arbol->izq,n,padre);
            }
            else{
                  insertar(arbol->der,n,padre);
            }
      }
}

void mostrar(Nodo *arbol,int cont){
      if (arbol==NULL){
            return;
```

```
	}
	else{
		mostrar(arbol->der,cont+1);
		for (int i=0;i<cont;i++){
			cout<<"  ";
		}
		cout<<arbol->dato<<endl;
		mostrar(arbol->izq,cont+1);
	}
}

bool busqueda(Nodo *arbol, int n){
	if(arbol==NULL){
		return false;
	}
	else if (arbol->dato==n){
		return true;
	}
	else if (n<arbol->dato){
		return busqueda(arbol->izq,n);
	}
	else{
		return busqueda(arbol->der,n);
	}
}

void preorden(Nodo *arbol){
	if (arbol==NULL){
```

```
            return;
      }
      else{
            cout<<arbol->dato<<" - ";
            preorden(arbol->izq);
            preorden(arbol->der);
      }
}

void inorden(Nodo *arbol){
      if (arbol=NULL){
            return;
      }
      else{
            inorden(arbol->izq);
            cout<<arbol->dato<<" - ";
            inorden(arbol->der);
      }
}
void postorden(Nodo *arbol){
      if (arbol==NULL){
            return;
      }
      else{
            postorden(arbol->izq);
            postorden(arbol->der);
            cout<<arbol->dato<<" - ";
      }
```

```
}
void eliminar(Nodo *arbol,int n){
      if (arbol==NULL){
            return;
      }
      else if(n<arbol->dato){
            eliminar(arbol->izq,n);
      }
      else if(n>arbol->dato){
            eliminar(arbol->der,n);
      }
      else{
            eliminar(arbol,n);
      }
}
Nodo *minimo(Nodo *arbol){
      if (arbol==NULL){
            return NULL;
      }
      if(arbol->izq){
            return minimo(arbol->izq);
      }
      else{
            return arbol;
      }
}
void eliminarNodo(Nodo *nodoEliminar){
      if (nodoEliminar->izq && nodoEliminar->der){
```

```
            Nodo *menor=minimo(nodoEliminar->der);
            nodoEliminar->dato=menor->dato;
            eliminarNodo(menor);
      }
      else if(nodoEliminar->izq){
            reemplazar(nodoEliminar,nodoEliminar->izq);
            destruir(nodoEliminar);
      }
      else if(nodoEliminar->der){
            reemplazar(nodoEliminar, nodoEliminar->der);
            destruir(nodoEliminar);
      }
      else{
            reemplazar(nodoEliminar,NULL);
            destruir(nodoEliminar);
      }
}
void reemplazar(Nodo *arbol, Nodo *nuevoNodo){
      if(arbol->padre){
            if(arbol->dato==arbol->padre->izq->dato){
                  arbol->padre->izq=nuevoNodo;
            }
            else if(arbol->dato==arbol->padre->der->dato){
                  arbol->padre->der=nuevoNodo;
            }
      }
      if(nuevoNodo){
            nuevoNodo->padre=arbol->padre;
```

```
	}
}
void destruir(Nodo *nodo){
	nodo->izq=NULL;
	nodo->der=NULL;
	delete nodo;
}

int Minimo(Nodo * raiz){

	if(raiz == NULL){
		cout<<" EL ARBOL ESTA VACIO";
	}

	Nodo* aux= raiz;
	aux=aux->izq;

	while(aux->izq !=NULL){
		aux=aux->izq;
	}


	cout<<"El nodo minimo es: "<<aux->dato<<endl;

	return aux->dato;

	cout<<"El nodo minimo es: "<<aux->dato<<endl;
```

```
}

int Maximo(Nodo * raiz){

      if(raiz == NULL){
            cout<<" EL ARBOL ESTA VACIO";
      }

      Nodo* aux= raiz;
      aux=aux->der;

      while(aux->der !=NULL){
            aux=aux->der;
      }

      cout<<"El nodo maximo es: "<<aux->dato<<endl;

      return aux->dato;

      cout<<"El nodo maximo es: "<<aux->dato<<endl;

}




void menu(){
```

```
int dato,cont=0,op;
do{
      cout<<"\t.:MENU:."<<endl;
      cout<<"1. Insertar un nuevo nodo"<<endl;
      cout<<"2. Mostrar el arbol completo"<<endl;
      cout<<"3. Buscar un elemento en el arbol"<<endl;
      cout<<"4. Recorrer el arbol en PREORDEN"<<endl;
      cout<<"5. Recorrer el arbol en INORDEN"<<endl;
      cout<<"6. Recorrer el arbol en POSTORDEN"<<endl;
      cout<<"7. Eliminar un nodo del arbol"<<endl;
      cout<<"8. Salir"<<endl;
      cout<<"9. Obtener nodo minimo"<<endl;
      cout<<"10. Obtener nodo maximo"<<endl;
      cout<<"Opcion: ";
      cin>>op;

    switch (op){
      case 1:
            cout<<"\nInserte un numero: ";
            cin>>dato;
            insertar(arbol,dato,NULL);
            cout<<"\n";
            system("pause");
            break;
      case 2:
            cout<<"\nMostrando el arbol completo:\n\n";
            mostrar(arbol,cont);
            cout<<"\n";
```

```
                system("pause");
                break;
            case 3:
                cout<<"\nIngresa el elemento a buscar: ";
                cin>>dato;
                if (busqueda(arbol,dato)==true){
                    cout<<"\nElemento "<<dato<<" ha sido
encontrado."<<endl;
                }
                else{
                    cout<<"\nElemento no encontrado."<<endl;
                }
                break;
            case 4:
                cout<<"\nRecorriendo en Preorden: \n";
                preorden(arbol);
                cout<<"\n\n";
                system("pause");
                break;
            case 5:
                cout<<"\nRecorriendo en Inorden: \n";
                inorden(arbol);
                cout<<"\n\n";
                system("pause");
                break;
            case 6:{
                cout<<"\nRecorrido en Postorden: \n";
                postorden(arbol);
                cout<<"\n\n";
```

```
            system("pause");
            break;
        }
        case 7:{
            cout<<"\nDigite el numero a eliminar: ";
            cin>>dato;
            eliminar(arbol,dato);
            cout<<"\n";
            system ("pause");
            break;
        }
        case 8:{
            cout<<"\nAdios";
            cout<<"\n";
            system ("pause");
            break;
        }
        case 9:{
            Minimo(arbol);
            system("pause");
            break;
        }
        case 10:{
            Maximo(arbol);
            system("pause");
            break;
        }
    }
```

```
         system ("cls");
      }while(op!=8);

}
```