UNIVERSIDAD AUTONOMA DE QUERETARO
Facultad de informática

Árbol binario

Estructura de datos

Carlos Noguez Juarez

315398

Grupo 35

FECHA 27/05/2024

Código:

```
#include <iostream>
#include <cstdlib>

using namespace std;

struct Node{
   int data;
   Node *father;
   Node *right;
   Node *left;
};

void menu();

Node *createNode(int, Node*);
void insertNode(Node *&, int, Node*);
void showTree(Node *, int );
bool searchNode(Node *, int);
void preOrden(Node *);
void inOrden(Node *);
void postOrden(Node *);
void deleteNode(Node*, int);
void deleteFindedNode(Node *);
void replaceNode(Node *, Node*);
void destroyNode(Node*);
Node* minimo(Node *);

Node *tree = NULL;

int main(){
   menu();
}

void menu(){
   system("cls");
   int option, n;
   bool loop = true;

   do{
      cout << endl;
      cout << "\tMENU ARBOL BINARIO DE BUSQUEDA" << endl;
      cout << "\n1. Insertar elementos a la estructura";
      cout << "\n2. Mostrar elementos";
      cout << "\n3. Buscar un elemento en la estructra";
```

```
    cout << "\n4. Recorrer arbol en pre orden";
    cout << "\n5. Recorrer arbol en in orden";
    cout << "\n6. Recorrer arbol en post orden";
    cout << "\n7. Eliminar un elemento";
    cout << "\n8. Salir";
    cout << "\nSelecciona una opcion: ";

    cin >> option;

    switch(option){
      case 1:
        cout << "\nInserta un numero entero: ";
        cin >> n;

        insertNode(tree, n , NULL);
        break;
      case 2:
        cout << "\nMostrar los elementos: \n\n";
        showTree(tree, 0);



        break;
      case 3:
        cout << "\nElemento a buscar:";
        cin >> n;

        searchNode(tree, n);
        break;
      case 4:
        cout << "\nRecorrer arbol en pre-orden: ";
        preOrden(tree);

        break;
      case 5:
        cout << "\nRecorrer arbol en in-orden: ";
        inOrden(tree);
        break;

      case 6:
        cout << "\nRecorrer arbol en post-orden: ";
        postOrden(tree);
        break;

      case 7:
        cout << "\nElemento a eliminar:";
        cin >> n;
```

```
            deleteNode(tree, n);
            showTree(tree, 0);

        case 8:
            loop = false;
            break;
        default:
            loop = false;
            break;
      }
   }
   while(loop);
}

Node *createNode(int n, Node*father){
   Node *newNode = new Node();

   newNode -> data = n;
   newNode -> father = father;
   newNode -> right = NULL;
   newNode -> left = NULL;

   return newNode;
}

void insertNode(Node *&tree, int n, Node *father){
   if(tree == NULL){
      Node *newNode = createNode(n, father);
      tree = newNode;
   }

   else{
      int rootValue = tree -> data;
      if(n < rootValue){
         insertNode(tree -> left, n, tree);
      }
      else{
         insertNode(tree -> right, n, tree);
      }
   }
}

void showTree(Node * tree, int auxY){
   int auxX = 0;
   if(tree == NULL){
      return;
   }
```

```
    else{
        showTree(tree -> right, auxY + 2);
        for (int i = 0; i < auxY; i++){
            cout << "    ";
        }
        cout << tree -> data << "\n";
        showTree(tree -> left, auxY + 2);
    }
}

bool searchNode(Node * tree, int n){
    if(tree == NULL){
        cout << "\nElemento no encontrado";
        return false;
    }

    else if(tree -> data == n){
        cout << "Elemento encontrado!";
        return true;
    }

    else if(n < tree -> data){
        return searchNode(tree -> left, n);
    }

    else{
        return searchNode(tree -> right, n);
    }
}

void preOrden(Node *tree){
    if(tree == NULL){
        return;
    }

    else{
        cout << "(" << tree -> data << ")" << "-";
        preOrden(tree -> left);
        preOrden(tree -> right);
    }
}

void inOrden(Node *tree) {
    if (tree == NULL) {
        return;
    } else {
```

```
        inOrden(tree->left);
        cout << "(" << tree->data << ")" << "-";
        inOrden(tree->right);
    }
}

void postOrden(Node *tree) {
    if (tree == NULL) {
        return;
    } else {
        postOrden(tree->left);
        postOrden(tree->right);
        cout << "(" << tree->data << ")" << "-";
    }
}

void deleteNode(Node *tree, int n){
    if(tree == NULL){
        return;
    }

    else if(n < tree -> data){
        deleteNode(tree -> left, n);
    }

    else if(n > tree -> data){
        deleteNode(tree -> right, n);
    }else {
        deleteFindedNode(tree);
    }

}

void deleteFindedNode(Node *aux_delete){
    if(aux_delete -> left && aux_delete -> right){
        Node *menor = minimo(aux_delete->right);
        aux_delete -> data = menor -> data;
        deleteFindedNode(menor);
    }

    else if(aux_delete -> left){
        replaceNode(aux_delete, aux_delete -> left);
        destroyNode(aux_delete);
    }

    else if(aux_delete -> right){
        replaceNode(aux_delete, aux_delete -> right);
```

```
      destroyNode(aux_delete);
   }
   else{
      replaceNode(aux_delete, NULL);
      destroyNode(aux_delete);
   }

}

Node* minimo(Node *node) {
   if(tree == NULL){
      return NULL;
   }
   if(node -> left){
      return minimo(node -> left);
   }else{
      return node;
   }
}

void replaceNode(Node * tree, Node * new_node){
   if(tree -> father){
      if(tree -> data == tree -> father -> left -> data) tree -> father -> left = new_node;
   }
   else if(tree -> data == tree -> father -> right -> data){
      tree -> father -> right = new_node;
   }
   if(new_node) new_node -> father = tree -> father;
}

void destroyNode(Node * node){
   node -> left= NULL;
   node -> right = NULL;

   delete node;
}
```