Act 3.1 - Operaciones avanzadas en un BST

Daniel Esparza Arizpe - A01637076

Diego Alberto Cisneros Fajardo - A01643454

Luis Fernando Li Chen - A00836174

Sadrac Aramburo Enciso - A01643639

Diego Michell Villa Durán - A00836723

Programación de estructuras de datos y algoritmos fundamentales

Grupo 605

Tecnológico de Monterrey

Lunes 09 de octubre 2023

Act 3.1 - Operaciones avanzadas en un BST

Casos de Prueba

1.-

```
Enter 1 for Preorder, 2 for Inorder, 3 for Postorder traversal: 2
1
3
4
5
6
7
9
Height of tree: 2
Enter the number whose ancestor(s) you want to find: 9
Ancestors of 9: 5 -> 7 -> 9
Enter number whose level you want to find: 9
9 is at level 2
```

2.-

```
Enter 1 for Preorder, 2 for Inorder, 3 for Postorder traversal: 1
5
3
1
4
7
6
9
Height of tree: 2
Enter the number whose ancestor(s) you want to find: 1
Ancestors of 1: 5 -> 3 -> 1
Enter number whose level you want to find: 55
-1
```

3.-

```
122  int main() {
123      Node* root = createNode(8);
124      insertNode(&root, 32);
125      insertNode(&root, 74);
126      insertNode(&root, 11);
127      insertNode(&root, 96);
128      insertNode(&root, 42);
129      insertNode(&root, 69);
130
131      int n;
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
./"insertion"
danielesparzarizpe@MacBook-Pro-de-Daniel-2 output % ./"insertion"
Enter 1 for Preorder, 2 for Inorder, 3 for Postorder traversal: 3
11
69
42
96
74
32
8
Height of tree: 4
Enter the number whose ancestor(s) you want to find: 96
Ancestors of 96: 8 -> 32 -> 74 -> 96
Enter number whose level you want to find: 96
96 is at level 3
```

4.-

```
122  int main() {
123      Node* root = createNode(2);
124      insertNode(&root, 6);
125      insertNode(&root, 4);
126      insertNode(&root, 9);
127      insertNode(&root, 5);
128      insertNode(&root, 14);
129      insertNode(&root, 25);
130
131      int n;
```

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

danielesparzarizpe@MacBook-Pro-de-Daniel-2 output % cd "/Users/daniele
danielesparzarizpe@MacBook-Pro-de-Daniel-2 output % ./"insertion"
Enter 1 for Preorder, 2 for Inorder, 3 for Postorder traversal: 2
2
4
5
6
9
14
25
Height of tree: 4
Enter the number whose ancestor(s) you want to find: 25
Ancestors of 25: 2 -> 6 -> 9 -> 14 -> 25
Enter number whose level you want to find: 25
25 is at level 4
```

Complejidad:

createNode(int data)
- Complejidad: O(1)
- La función simplemente crea un nuevo nodo y establece sus punteros izquierdo y derecho en NULL. No depende del tamaño de la entrada.

insertNode(Node root, int data)**
- Complejidad: En el peor caso, O(n), donde n es la altura del árbol.
- En el peor de los casos, la función debe recorrer la altura del árbol para encontrar el lugar adecuado para insertar el nuevo nodo.

height(Node* root)
- Complejidad: O(n), donde n es el número de nodos en el árbol.
- La función recorre todos los nodos del árbol para calcular la altura.

ancestors(Node* root, int target, vector<int>& path)

- Complejidad: O(n), donde n es el número de nodos en el árbol.
- La función explora todos los nodos en busca del nodo objetivo y almacena los nodos visitados en el vector `path`.

whatlevelamI(Node* root, int target, int level)
- Complejidad: O(n), donde n es el número de nodos en el árbol.
- La función explora todos los nodos para encontrar el nodo objetivo y devuelve su nivel en el árbol.

inOrder(Node* root)
- Complejidad: O(n), donde n es el número de nodos en el árbol.
- La función imprime todos los nodos en orden inOrder, visitando todos los nodos del árbol una vez.

postOrder(Node* root)
- Complejidad: O(n), donde n es el número de nodos en el árbol.
- La función imprime todos los nodos en orden postOrder, visitando todos los nodos del árbol una vez.

preOrder(Node* root)
- Complejidad: O(n), donde n es el número de nodos en el árbol.
- La función imprime todos los nodos en orden preOrder, visitando todos los nodos del árbol una vez.

visit(Node* root, int n)
- Complejidad: O(n), donde n es el número de nodos en el árbol.
- Dependiendo del valor de `n`, la función llama a una de las funciones de recorrido del árbol, cada una con complejidad O(n).

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

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

Node* createNode(int data) {
Node* newNode = new Node();
newNode->data = data;
newNode->left = NULL;
```

```
newNode->right = NULL;
return newNode;
}

void insertNode(Node** root, int data) {
if (*root == NULL) {
*root = createNode(data);
return;
}

if (data <= (*root)->data) {
if ((*root)->left == NULL) {
(*root)->left = createNode(data);
} else {
insertNode(&((*root)->left), data);
}
} else {
if ((*root)->right == NULL) {
(*root)->right = createNode(data);
} else {
insertNode(&((*root)->right), data);
}
}
}

int height(Node* root) {
if (root == NULL) {
return -1;
}
int leftHeight = height(root->left);
int rightHeight = height(root->right);
return max(leftHeight, rightHeight) + 1;
}

bool ancestors(Node* root, int target, vector<int>& path) {
if (root == NULL) {
return false;
}
path.push_back(root->data);
```

```
if (root->data == target) {
return true;
}
if (ancestors(root->left, target, path) || ancestors(root->right,
target, path)) {
return true;
}
path.pop_back();
return false;
}

int whatlevelamI(Node* root, int target, int level) {
if (root == NULL) {
return -1;
}
if (root->data == target) {
return level;
}
int leftLevel = whatlevelamI(root->left, target, level + 1);
if (leftLevel != -1) {
return leftLevel;
}
int rightLevel = whatlevelamI(root->right, target, level + 1);
return rightLevel;
}

void inOrder(Node* root) {
if (root == NULL) {
return;
}
inOrder(root->left);
cout << root->data << endl;
inOrder(root->right);
}

void postOrder(Node* root){
if(root == nullptr)
return;
```

```
postOrder(root->left);
postOrder(root->right);
cout << root->data << endl;
}

void preOrder(Node* root){
if(root == nullptr)
return;
cout << root->data << endl;
preOrder(root->left);
preOrder(root->right);
}

void visit(Node* root, int n) {
if (n == 1)
preOrder(root);

else if (n == 2)
inOrder(root);

else if (n == 3)
postOrder(root);

else{
cout << "Invalid choice" << endl;
}
}

int main() {
Node* root = createNode(2);
insertNode(&root, 6);
insertNode(&root, 4);
insertNode(&root, 9);
insertNode(&root, 5);
insertNode(&root, 14);
insertNode(&root, 25);

int n;
```

```
cout << "Enter 1 for Preorder, 2 for Inorder, 3 for Postorder
traversal: ";
cin >> n;
visit(root, n);

cout << "Height of tree: " << height(root) << endl;

int target;
cout << "Enter the number whose ancestor(s) you want to find: ";
cin >> target;

vector<int> path;
if (ancestors(root, target, path)) {
cout << "Ancestors of " << target << ": ";
for (int i = 0; i < path.size() - 1; i++) {
cout << path[i] << " -> ";
}
cout << path[path.size() - 1] << endl;
} else {
cout << target << " is not in the tree." << endl;
}

cout << "Enter number whose level you want to find: ";
cin >> target;

int level = whatlevelamI(root, target, 0);
if (level != -1) {
cout << target << " is at level " << level << endl;
} else {
cout << "-1" << endl;
}

return 0;
}
```