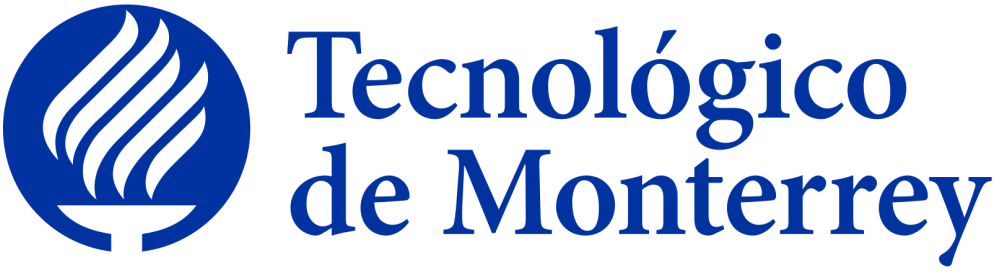
Instituto Tecnológico de Estudios Superiores de Monterrey



Campus Monterrey

Act 3.1 - Operaciones avanzadas en un BST

Programación de estructuras de datos y algoritmos fundamentales (Gpo 610)

Eduardo López Benítez

Fidel Morales Briones A01198630

Monterrey, Nuevo León, a 11 de octubre de 2023

Casos prueba:

Árbol 1:

```cpp
int main() {
    // Arbol 1
    cout << "Arbol 1 \n";
    BST<int> arbol1;
    arbol1.insert(5);
    arbol1.insert(3);
    arbol1.insert(2);
    arbol1.insert(1);
    arbol1.insert(0);
    arbol1.insert(6);

    // visit()
    cout << "Preorder: ";
    arbol1.visit(1);
    cout<<endl;

    cout << "Inorder: ";
    arbol1.visit(2);
    cout<<endl;

    cout << "Postorder: ";
    arbol1.visit(3);
    cout<<endl;

    cout << "Level by level: ";
    arbol1.visit(4);
    cout<<endl;

    // height()
    cout << "Altura de arbol: " << arbol1.height(arbol1.getRoot()) << endl;

    // ancestors()
    cout << "Ancestros de 6: ";
    arbol1.ancestors(arbol1.getRoot(), 6);

    // whatlevelamI()
    cout << "\nNivel de 6: " << arbol1.whatlevelamI(arbol1.getRoot(), 6) << endl;
```

```
PS C:\Users\T480\Desktop\tec\3
) { .\tarea3_1 }
Arbol 1
Preorder: 5 3 2 1 0 6
Inorder: 0 1 2 3 5 6
Postorder: 0 1 2 3 6 5
Level by level: 5 3 6 2 1 0
Altura de arbol: 5
Ancestros de 6: 5
Nivel de 6: 2
```

Árbol 2:

```
306     // Arbol 2
307     cout << "\nArbol 2 \n";
308     BST<int> arbol2;
309     arbol2.insert(10);
310     arbol2.insert(13);
311     arbol2.insert(21);
312     arbol2.insert(1);
313     arbol2.insert(5);
314     arbol2.insert(6);
315
316     // visit()
317     cout << "Preorder: ";
318     arbol2.visit(1);
319     cout<<endl;
320
321     cout << "Inorder: ";
322     arbol2.visit(2);
323     cout<<endl;
324
325     cout << "Postorder: ";
326     arbol2.visit(3);
327     cout<<endl;
328
329     cout << "Level by level: ";
330     arbol2.visit(4);
331     cout<<endl;
332
333     // height()
334     cout << "Altura de arbol: " << arbol2.height(arbol2.getRoot()) << endl;
335
336     // ancestors()
337     cout << "Ancestros de 1: ";
338     arbol2.ancestors(arbol2.getRoot(), 1);
339
340     // whatlevelamI()
341     cout << "\nNivel de 1: " << arbol2.whatlevelamI(arbol2.getRoot(), 1) << endl;
```

```
Arbol 2
Preorder: 10 1 5 6 13 21
Inorder: 1 5 6 10 13 21
Postorder: 6 5 1 21 13 10
Level by level: 10 1 13 5 21 6
Altura de arbol: 4
Ancestros de 1: 10
Nivel de 1: 2
```

Árbol 3:

```cpp
    // Arbol 3
    cout << "\nArbol 3 \n";
    BST<int> arbol3;
    arbol3.insert(1);
    arbol3.insert(22);
    arbol3.insert(21);
    arbol3.insert(14);
    arbol3.insert(5);
    arbol3.insert(0);

    // visit()
    cout << "Preorder: ";
    arbol3.visit(1);
    cout<<endl;

    cout << "Inorder: ";
    arbol3.visit(2);
    cout<<endl;

    cout << "Postorder: ";
    arbol3.visit(3);
    cout<<endl;

    cout << "Level by level: ";
    arbol3.visit(4);
    cout<<endl;

    // height()
    cout << "Altura de arbol: " << arbol3.height(arbol3.getRoot()) << endl;

    // ancestors()
    cout << "Ancestros de 5: ";
    arbol3.ancestors(arbol3.getRoot(), 5);

    // whatlevelamI()
    cout << "\nNivel de 5: " << arbol3.whatlevelamI(arbol3.getRoot(), 5) << endl;
```

```
Arbol 3
Preorder: 1 0 22 21 14 5
Inorder: 0 1 5 14 21 22
Postorder: 0 5 14 21 22 1
Level by level: 1 0 22 21 14 5
Altura de arbol: 5
Ancestros de 5: 14 21 22 1
Nivel de 5: 5
```

Árbol 4:

```
380     // Arbol 4
381     cout << "\nArbol 4 \n";
382     BST<int> arbol4;
383     arbol4.insert(11);
384     arbol4.insert(22);
385     arbol4.insert(33);
386     arbol4.insert(44);
387     arbol4.insert(55);
388     arbol4.insert(66);
389
390     // visit()
391     cout << "Preorder: ";
392     arbol4.visit(1);
393     cout<<endl;
394
395     cout << "Inorder: ";
396     arbol4.visit(2);
397     cout<<endl;
398
399     cout << "Postorder: ";
400     arbol4.visit(3);
401     cout<<endl;
402
403     cout << "Level by level: ";
404     arbol4.visit(4);
405     cout<<endl;
406
407     // height()
408     cout << "Altura de arbol: " << arbol4.height(arbol4.getRoot()) << endl;
409
410     // ancestors()
411     cout << "Ancestros de 44: ";
412     arbol4.ancestors(arbol4.getRoot(), 44);
413
414     // whatlevelamI()
415     cout << "\nNivel de 44: " << arbol4.whatlevelamI(arbol4.getRoot(), 44) << endl;
```

```
Arbol 4
Preorder: 11 22 33 44 55 66
Inorder: 11 22 33 44 55 66
Postorder: 66 55 44 33 22 11
Level by level: 11 22 33 44 55 66
Altura de arbol: 6
Ancestros de 44: 33 22 11
Nivel de 44: 4
```

Programa:

```
/*
Act 3.1 - Operaciones avanzadas en un BST
Fidel Morales Briones A01198630
11 de octubre de 2023
*/

#include <iostream>
using namespace std;

template <class T>
class Node {
    private:
        T data;
        Node<T>* left;
        Node<T>* right;

    public:

        Node(T data) {
            this->data = data;
            this->left = nullptr;
            this->right = nullptr;
        }

        T getData() {
            return data;
        }

        Node<T>* getLeft() {
            return left;
        }

        void setLeft(Node<T>* left) {
            this->left = left;
        }

        Node<T>* getRight() {
            return right;
        }

        void setRight(Node<T>* right) {
            this->right = right;
```

```
        }

};

template <class T>
class BST {
    private:
        Node<T>* root;

    public:
        BST() {
            root = nullptr;
        }

        Node<T>* getRoot() {
            return root;
        }

        /*
        * insert, inserta un nuevo nodo en el árbol
        *
        * Añade un nodo al árbol, utiliza el método recursivo
        * insertRec para encontrar la posición correcta
        *
        * @param data: dato a insertar
        * @return: no tiene
        * Complejidad tiempo: O(log(n))
        * Complejidad espacio: O(n)
        */
        void insert(T data) {
            root = insertRec(root, data);
        }

        /*
        * insertRec, método recursivo para insertar un nuevo nodo en el
árbol
        *
        * El nodo se actualizará en cada recursión, hasta encontrar la
posición
        * correcta para insertar el nuevo nodo, y se regresará el nodo
actualizado
        * para definirlo en el método insert
        *
```

```
        * @param data: dato a insertar
        * @param node: nodo actual
        * @return: *Nodo<T> a insertar
        * Complejidad tiempo: O(log(n))
        * Complejidad espacio: O(n)
        */
        Node<T> * insertRec(Node<T>* node, T data) {
            if (node == nullptr) {
                node = new Node<T>(data);
            }
            else {
                if (data < node->getData()) {
                    node->setLeft(insertRec(node->getLeft(), data));
                }
                else {
                    node->setRight(insertRec(node->getRight(), data));
                }
            }
            return node;
        }

        /*
        * visit, diferentes tipos de recorridos del árbol
        *
        * Se tendrán 4 opciones para recorrer el árbol:
        *   1. Preorder
        *   2. Inorder
        *   3. Postorder
        *   4. Level by level
        *
        * @param option: tipo de recorrido que se imprimirá
        * @return: en la terminal, se imprimirá el recorrido
seleccionado
        * Complejidad tiempo: O(n^2)
        * Complejidad espacio: O(n)
        */
        void visit(int option) {
            switch (option) {
                case 1:
                    preorder(root);
                    break;
                case 2:
                    inorder(root);
```

```
                    break;
                case 3:
                    postorder(root);
                    break;
                case 4:
                    levelbylevel(root);
                    break;
            }
        }

        /*
        * preorder, recorrido en preorder del árbol
        *
        * Con recursión, se imprime la raíz, después el subárbol izquierdo,
        * y finalmente el subárbol derecho. Se obtienen todos los nodos de la izquierda
        * y derecha hasta llegar a las hojas
        *
        * @param node: nodo raíz del árbol
        * @return: en la terminal, se imprimirá el recorrido
        * Complejidad tiempo: O(n)
        * Complejidad espacio: O(n)
        */
        void preorder(Node<T>* node) {
            if (node != nullptr) {
                cout << node->getData() << " ";
                preorder(node->getLeft());
                preorder(node->getRight());
            }
        }

        /*
        * inorder, recorrido en inorder del árbol
        *
        * Con recursión, se imprimen los datos del árbol en orden ascendente.
        * Recorre el subárbol izquierdo, después la raíz, y finalmente el subárbol derecho
        *
        * @param node: nodo raíz del árbol
        * @return: en la terminal, se imprimirá el recorrido
        * Complejidad tiempo: O(n)
```

```
        * Complejidad espacio: O(n)
        */
        void inorder(Node<T>* node) {
            if (node != nullptr) {
                inorder(node->getLeft());
                cout << node->getData() << " ";
                inorder(node->getRight());
            }
        }

        /*
        * postorder, recorrido en postorder del árbol
        *
        * Con recursión se recorre el sub[arbol izquierdo, después el
subárbol derecho, y finalmente
        * la raíz
        *
        * @param node: nodo raíz del árbol
        * @return: en la terminal, se imprimirá el recorrido
        * Complejidad tiempo: O(n)
        * Complejidad espacio: O(n)
        */
        void postorder(Node<T>* node) {
            if (node != nullptr) {
                postorder(node->getLeft());
                postorder(node->getRight());
                cout << node->getData() << " ";
            }
        }

        /*
        * levelbylevel, recorrido de cada nivel del árbol
        *
        * Con recursión se recorre cada nivel del árbol, imprimiendo
los datos
        * de los nodos de cada nivel
        *
        * @param node: nodo raíz del árbol
        * @return: en la terminal, se imprimirá el recorrido
        * Complejidad tiempo: O(n^2)
        * Complejidad espacio: O(n)
        */
        void levelbylevel(Node<T> *node) {
```

```
            int h = height(node);
            for (int i = 1; i <= h; i++) {
                printCurrentLevel(node, i);
            }
        }

        /*
        * printCurrentLevel, imprimir un nivel del árbol
        *
        * Se utiliza dentro de levelbylevel(), y sirve para imprimir
los nodos
        * del nivel actual del árbol, se utiliza el método height para
        * saber cuántas iteraciones hay que hacer
        *
        * @param node: nodo raíz del árbol
        * @param level: nivel actual del árbol
        * @return: en la terminal, se imprimirá el recorrido
        * Complejidad tiempo: O(n)
        * Complejidad espacio: O(n)
        */
        void printCurrentLevel(Node<T> *node, int level) {
            if (node == nullptr) {
                return;
            }
            if (level == 1) {
                cout << node->getData() << " ";
            }
            else if (level > 1) {
                printCurrentLevel(node->getLeft(), level - 1);
                printCurrentLevel(node->getRight(), level - 1);
            }
        }

        /*
        * height, se obtiene la altura del árbol
        *
        * Se busca en ambos subárboles la altura haciendo un recorrido
hasta llegar a una hoja,
        * y se regresa la altura más grande
        *
        * @param node: nodo raíz del árbol
        * @return: entero de altura del árbol
        * Complejidad tiempo: O(n)
```

```
        * Complejidad espacio: O(n)
        */
        int height(Node<T> *node) {
            if (node == nullptr) {
                return 0;
            }
            else {
                int leftHeight = height(node->getLeft());
                int rightHeight = height(node->getRight());
                if (leftHeight > rightHeight) {
                    return (leftHeight + 1);
                }
                else {
                    return (rightHeight + 1);
                }
            }
        }

        /*
        * ancestors, se obtienen los ancestros de un nodo
        *
        * Se busca en ambos subárboles la altura, y se regresa la
altura más grande
        * Se utiliza la recursividad hasta encontrar el nodo, y se en
la terminal
        * se imprimen los ancestros
        *
        * @param node: nodo raíz del árbol
        * @param data: dato del nodo a buscar los ancestros
        * @return: booleano, true si se encontró el dato, false si no
        * Complejidad tiempo: O(n)
        * Complejidad espacio: O(n)
        */
        bool ancestors(Node<T> *node, T data) {
            if (node == nullptr) {
                return false;
            }
            if (node->getData() == data) {
                return true;
            }
            if (ancestors(node->getLeft(), data) ||
ancestors(node->getRight(), data)) {
                cout << node->getData() << " ";
```

```
                return true;
            }
            return false;
        }

        /*
        * whatlevelamIUtil, método auxilar para obtener el nivel de un
nodo
        *
        * Se utiliza la recursividad hasta encontrar el nodo, y se
regresa el nivel
        * Se tendrá un nivel inicial de 1, y se irá incrementando en
cada recursión,
        * cuando la recursión termine este valor se retornará y será el
nivel de ese nodo
        *
        * @param node: nodo raíz del árbol
        * @param data: dato del nodo a buscar los ancestros
        * @param level: nivel actual del árbol
        * @return: entero del nivel del nodo
        * Complejidad tiempo: O(n)
        * Complejidad espacio: O(n)
        */
        int whatlevelamIUtil(Node<T> *node, T data, int level) {
            if (node == nullptr) {
                return 0;
            }
            if (node->getData() == data) {
                return level;
            }
            int downlevel = whatlevelamIUtil(node->getLeft(), data,
level + 1);
            if (downlevel != 0) {
                return downlevel;
            }
            downlevel = whatlevelamIUtil(node->getRight(), data, level
+ 1);
            return downlevel;
        }

        /*
        * whatlevelamI, método para conseguir el nivel en el que se
encuentra un nodo
```

```
        *
        * Se llama al método auxiliar, dentro de este método se
introducen los parámetros
        * iniciales del nodo raíz y el dato a buscar. La variable del
nivel pertenece al
        * método auxiliar pero se regresa en este método
        *
        * @param node: nodo raíz del árbol
        * @param data: dato del nodo a buscar los ancestros
        * @return: entero del nivel del nodo
        * Complejidad tiempo: O(n)
        * Complejidad espacio: O(n)
        */
        int whatlevelamI(Node<T>* node, T data) {
            return whatlevelamIUtil(node, data, 1);
        }

};

int main() {
    // Arbol 1
    cout << "Arbol 1 \n";
    BST<int> arbol1;
    arbol1.insert(5);
    arbol1.insert(3);
    arbol1.insert(2);
    arbol1.insert(1);
    arbol1.insert(0);
    arbol1.insert(6);

    // visit()
    cout << "Preorder: ";
    arbol1.visit(1);
    cout<<endl;

    cout << "Inorder: ";
    arbol1.visit(2);
    cout<<endl;

    cout << "Postorder: ";
    arbol1.visit(3);
    cout<<endl;
```

```
    cout << "Level by level: ";
    arbol1.visit(4);
    cout<<endl;

    // height()
    cout << "Altura de arbol: " << arbol1.height(arbol1.getRoot()) <<
endl;

    // ancestors()
    cout << "Ancestros de 6: ";
    arbol1.ancestors(arbol1.getRoot(), 6);

    // whatlevelamI()
    cout << "\nNivel de 6: " << arbol1.whatlevelamI(arbol1.getRoot(),
6) << endl;

    // Arbol 2
    cout << "\nArbol 2 \n";
    BST<int> arbol2;
    arbol2.insert(10);
    arbol2.insert(13);
    arbol2.insert(21);
    arbol2.insert(1);
    arbol2.insert(5);
    arbol2.insert(6);

    // visit()
    cout << "Preorder: ";
    arbol2.visit(1);
    cout<<endl;

    cout << "Inorder: ";
    arbol2.visit(2);
    cout<<endl;

    cout << "Postorder: ";
    arbol2.visit(3);
    cout<<endl;

    cout << "Level by level: ";
    arbol2.visit(4);
    cout<<endl;
```

```
    // height()
    cout << "Altura de arbol: " << arbol2.height(arbol2.getRoot()) <<
endl;

    // ancestors()
    cout << "Ancestros de 1: ";
    arbol2.ancestors(arbol2.getRoot(), 1);

    // whatlevelamI()
    cout << "\nNivel de 1: " << arbol2.whatlevelamI(arbol2.getRoot(),
1) << endl;

    // Arbol 3
    cout << "\nArbol 3 \n";
    BST<int> arbol3;
    arbol3.insert(1);
    arbol3.insert(22);
    arbol3.insert(21);
    arbol3.insert(14);
    arbol3.insert(5);
    arbol3.insert(0);

    // visit()
    cout << "Preorder: ";
    arbol3.visit(1);
    cout<<endl;

    cout << "Inorder: ";
    arbol3.visit(2);
    cout<<endl;

    cout << "Postorder: ";
    arbol3.visit(3);
    cout<<endl;

    cout << "Level by level: ";
    arbol3.visit(4);
    cout<<endl;

    // height()
    cout << "Altura de arbol: " << arbol3.height(arbol3.getRoot()) <<
endl;
```

```
    // ancestors()
    cout << "Ancestros de 5: ";
    arbol3.ancestors(arbol3.getRoot(), 5);

    // whatlevelamI()
    cout << "\nNivel de 5: " << arbol3.whatlevelamI(arbol3.getRoot(),
5) << endl;

    // Arbol 4
    cout << "\nArbol 4 \n";
    BST<int> arbol4;
    arbol4.insert(11);
    arbol4.insert(22);
    arbol4.insert(33);
    arbol4.insert(44);
    arbol4.insert(55);
    arbol4.insert(66);

    // visit()
    cout << "Preorder: ";
    arbol4.visit(1);
    cout<<endl;

    cout << "Inorder: ";
    arbol4.visit(2);
    cout<<endl;

    cout << "Postorder: ";
    arbol4.visit(3);
    cout<<endl;

    cout << "Level by level: ";
    arbol4.visit(4);
    cout<<endl;

    // height()
    cout << "Altura de arbol: " << arbol4.height(arbol4.getRoot()) <<
endl;

    // ancestors()
    cout << "Ancestros de 44: ";
    arbol4.ancestors(arbol4.getRoot(), 44);
```

```
    // whatlevelamI()
    cout << "\nNivel de 44: " << arbol4.whatlevelamI(arbol4.getRoot(),
44) << endl;

    return 0;
}
```

Bibliografía:

GeeksforGeeks. (2022). Get level of a node in a binary tree. *GeeksforGeeks*. https://www.geeksforgeeks.org/get-level-of-a-node-in-a-binary-tree/

GeeksforGeeks. (2023a). Level Order traversal breadth first Search or BFS of binary tree. *GeeksforGeeks*. https://www.geeksforgeeks.org/level-order-tree-traversal/

GeeksforGeeks. (2023b). Insertion in Binary Search Tree BST. *GeeksforGeeks*. https://www.geeksforgeeks.org/insertion-in-binary-search-tree/

GeeksforGeeks. (2023d). Print Ancestors of a given node in binary tree. *GeeksforGeeks*. https://www.geeksforgeeks.org/print-ancestors-of-a-given-node-in-binary-tree/

Pandey, D. (2022, 4 julio). Traversal of Binary Tree. *Scaler Topics*. https://www.scaler.com/topics/traversal-of-binary-tree/