Reflexión Actividad 3.1 Operaciones avanzadas en un BST

Héctor Robles Villarreal A01634105

Programación de estructuras de datos y algoritmos fundamentales

Carlos Augusto Ventura Molina

Grupo 11

Tecnológico de Monterrey Campus Guadalajara

Sábado 23 de Octubre de 2021

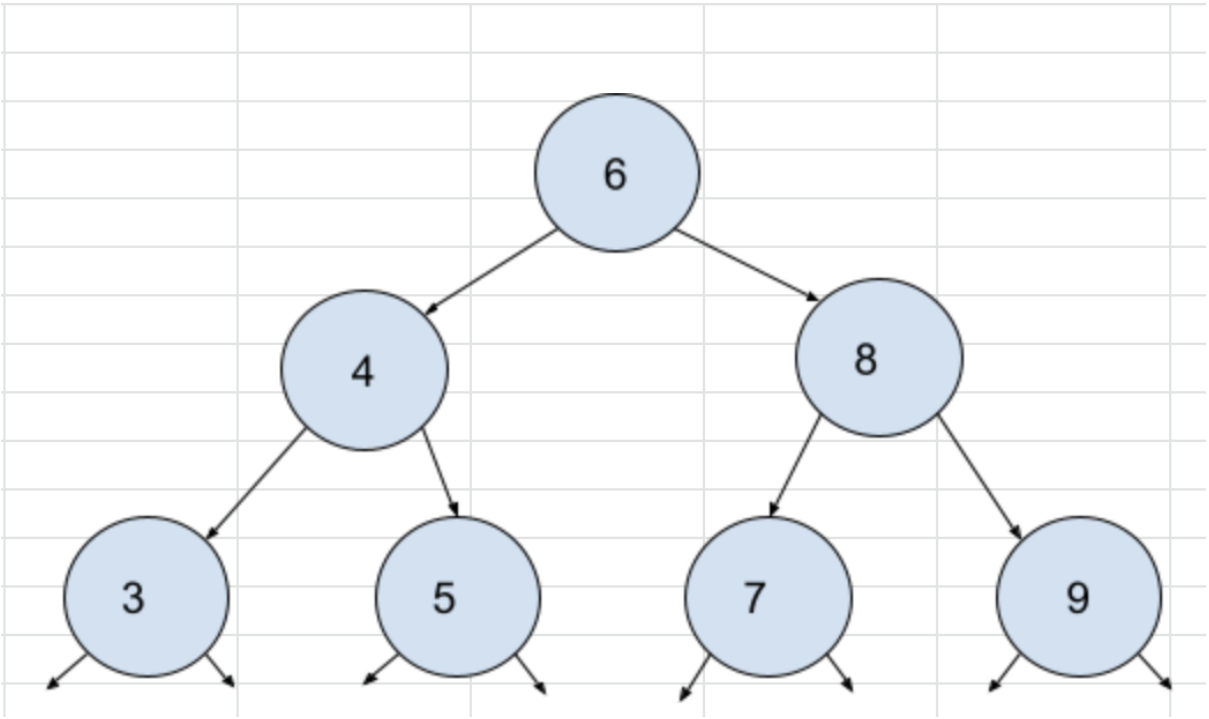
Casos de prueba y ejecución del código

Primero se crea un árbol de tipo avl

Después se insertan varios nodos a través del uso de la función insert

```
int main() {
    avl_tree tree;
    r = tree.insert(r, 3);
    r = tree.insert(r, 6);
    r = tree.insert(r, 8);
    r = tree.insert(r, 5);
    r = tree.insert(r, 9);
    r = tree.insert(r, 4);
    r = tree.insert(r, 7);
```

El árbol quedaría así ya que se balancea automáticamente con insert:



Ahora se manda a llamar a la función visit que despliega los datos del árbol en un orden específicado:

```
void avl_tree::visit(avl* t, int orden){
    if (orden == 1){
        preorder(t);
    }
    else if (orden ==2){
        inorder(t);
    }
    else if (orden == 3){
        postorder(t);
    }
```

```
    else if (orden == 4){
        static avl* const DELIMETER = nullptr;
        //Si la raíz está vacía no imrpime nada
        if (t == NULL){
            return;
        }
        //Inicializa una queue de nodos
        queue<avl*> qu;
        //Se introduce el nodo de parámetro
        qu.push(t);
        //Se mete un delimitador que indica que habrá un cambio de nivel
        qu.push(DELIMETER);
        while(true){
            avl* curr = qu.front();
            qu.pop();
            if(curr != DELIMETER){
                cout << curr->d <<" ";
                if(curr->l != NULL){
                    qu.push(curr->l);
                }
                if (curr->r != NULL){
                    qu.push(curr->r);
                }
            }
            else{
                cout << endl;
                if(qu.empty()) break;
                qu.push(DELIMETER);
            }
        }
    }
}
```

Lo que hace es primero, checar el número que se le pasó como opción para decidir en qué orden va a desplegar los datos. En caso de que sea un 4, crea una queue y la va llenando con

los datos, cada que detecta que cambia de nivel, se agrega un delimitador a la queue para poder ir imprimiendo de forma adecuada los datos.

Se manda a llamar cuatro veces, cada vez con un parámetro diferente para probar los diferentes órdenes, el 1 es Preorder, el 2 Inorder, el 3 Postorder y el 4 despliega los datos por nivel:

```
cout << "Datos del árbol ordenados en Preorder" << endl;
avl_tree::visit(r,1);
cout << endl;
cout << "Datos del árbol ordenados en Inorder : " << endl;
avl_tree::visit(r,2);
cout  << endl;
cout << "Datos del árbol ordenados en Postorder : " << endl;
avl_tree::visit(r,3);
cout << endl;
cout << "Datos del árbol ordenados por nivel: " << endl;
avl_tree::visit(r,4) ;
cout << endl;
```

Esta es la salida:

```
Datos del árbol ordenados en Preorder:
6 4 3 5 8 7 9
Datos del árbol ordenados en Inorder:
3 4 5 6 7 8 9
Datos del árbol ordenados en Postorder:
3 5 4 7 9 8 6
Datos del árbol ordenados por nivel:
6
4 8
3 5 7 9
```

Ahora se manda a llamar a la función height para saber la altura del árbol:

```
int avl_tree::height(avl* t) {
    int h = 0;
    if (t != NULL) {
        int l_height = height(t->l);
        int r_height = height(t->r);
        int max_height = max(l_height, r_height);
        h = max_height + 1;
    }
    return h;
```

La función básicamente usa recursividad para encontrar la altura del árbol y al final se le suma un 1 para encontrar la altura total, que es la alutra máxima entre el lado izquierdo y el derecho.

```
cout << "La altura del árbol r es de: ";
cout << avl_tree::height(r) << endl << endl;
```

Esta es la salida:

```
La altura del árbol r es de: 3
```

Ahora se hace uso de la función ancestors para obtener los ancestros de un nodo en específico

```
bool avl_tree::ancestors(avl *t, int dato){
    //Si el árbol está vacío no tiene ancestos
    if (t == NULL){
        return false;
    }
    //Regresa true cuando encuentra el dato
    if (t->d == dato){
        return true;
    }
    //Se usa recursión para encontrar el lado en el que se encuentra el dato
    //Posteriormente se recorre ese lado del árbol, desplegando los ancestros del dato
    if (ancestors(t->l,dato) || ancestors(t->r,dato)) {
        cout << t->d << "  ";
        return true;
    }
    //Regresa false si no encuentra el dato en el árbol
    return false;
}
```

Lo que hace es usar recursión para ir recorriendo los niveles del árbol hasta que llega al dato que se ingresó y así desplegar los ancestros. En caso de ser la raíz, no se imprime nada porque esta no tiene ancestros.

```
cout << endl;
cout << "Ancestros del nodo que tiene el 5: " << endl;
avl_tree::ancestors(r,5);
cout << endl;
cout << "Ancestros del nodo que tiene el 9: " << endl;
avl_tree::ancestors(r,9);
cout << endl;
cout << "Ancestros del nodo que tiene el 4: " << endl;
avl_tree::ancestors(r,4);
cout << endl;
cout << "Ancestros del nodo que tiene el 6: " << endl;
avl_tree::ancestors(r,6);
cout << endl;
```

Se piden los ancestros del nodo que tiene el 5, el 9, el 4 y ancestros de la raíz. Como en este caso la raíz es el nodo que tiene el 6, no se despliega nada, ya que la raíz no tiene ancestros.

Ahora se hace uso de la función whatlevelamI que indica en qué nivel se encuentra un nodo que contiene un dato en específico

```
int avl_tree::whatlevelamI(avl* t, int dato){
    //Si el nodo está vacío, regresa un 0
    if (t == NULL){
        return 0;
    }
    int level = 1;
    int nivel = 0;
    //Guarda el nivel del dato a buscar
    nivel = avl_tree::getLevel(t, dato, level);
    //Si no encuentra el dato, regresa un -1
    if (nivel == 0){
        return -1;
    }
    else{
        return nivel;
    }
}

int avl_tree::getLevel(avl *t, int dato, int level){
    //Si el nodo está vacío, regresa un 0
    if(t == NULL){
        return 0;
    }
    //Checa si el valor de ese nodo es el dato que se busca
    if (t->d == dato){
        return level;
    }
    //Llamada recursiva para buscar el árbob por el lado izquierdo
    int resultado = getLevel(t->l, dato, level+1);
    if (resultado != 0){
        return resultado;
    }
    //Si no encontro el dato en el lado izquierdo, lo busca en el derecho
    resultado = getLevel(t->r,dato,level+1);
    return resultado;
}
```

Esta manda llamar a otra función llamada getLevel que usa recursión para ir recorriendo los niveles del árbol hasta llegar al dato deseado y desplegar el nivel en el que se encuentra. En caso de no encontrar el dato, se regresa un -1.

```
cout << "El nivel en el que se encuentra el número 9 es el: ";
cout << avl_tree::whatlevelamI(r,9) <<endl;
cout << "El nivel en el que se encuentra el número 3 es el: ";
cout << avl_tree::whatlevelamI(r,3) <<endl;
cout << "El nivel en el que se encuentra el número 4 es el: ";
cout << avl_tree::whatlevelamI(r,4) <<endl;
cout << "El nivel en el que se encuentra el número 6 es el: ";
cout << avl_tree::whatlevelamI(r,6) <<endl;
cout << "El nivel en el que se encuentra el número 5 es el: ";
cout << avl_tree::whatlevelamI(r,5) <<endl;
cout << "El nivel en el que se encuentra el número 89 es el: ";
cout << avl_tree::whatlevelamI(r,89) <<endl;
```

Esta es la salida:

```
El nivel en el que se encuentra el número 9 es el: 3
El nivel en el que se encuentra el número 3 es el: 3
El nivel en el que se encuentra el número 7 es el: 2
El nivel en el que se encuentra el número 6 es el: 1
El nivel en el que se encuentra el número 5 es el: 3
El nivel en el que se encuentra el número 89 es el: -1
```

Notese que como el 89 no está dentro de los nodos de este árbol, la función regresa un -1, indicando que no encontró ese dato.