Tecnológico de Monterrey
Edgar Cruz Vázquez (A01730577)
TC1031.11: Programación de estructuras y algoritmos fundamentales (GPO 11)

Act 3.1 – Operaciones avanzadas en un BST

En forma individual implementa y agrega las funcionalidades avanzadas al ADT del BST realizado con las funcionalidades fundamentales:

- visit();
- height();
- ancestors();
- whatlevelaml();

	CASOS DE PRUEBA ACT 3.1				
No.	Precondiciones	Resumen	Pasos	Resultado Esperado	Resultado obtenido
1	Tener el código y tener un bst válido	El usuario introduce un modo invalido en visit();	1. Iniciar el programa. 2. Introducir un modo no válido	Una vez insertado el modo el programa desplegara un texto que no es válido.	El programa despliega un texto diciendo que no es válido el modo.
2	Tener el código y tener un bst válido	El usuario solamente introduce un dato invalido en la función whatlevel	1. Correr el programa. 2. Introducir un dato invalido en la función whatlevelaml	El programa debería imprimir un -1 ya que no encontró el dato.	El programa imprime un -1
3	Tener el código y tener un bst válido	El usuario introduce un árbol nulo en ancestors	1. Correr el programa 2. Introducir un árbol NULL en la función ancestors();	El programa retornara un false	El programa no devuelve nada (false)
4	Tener el código y tener un bst válido	El usuario introduce un árbol NULL en la función height	1. Correr el programa. 2. Introducir un árbol NULL en height();	El programa retornara 0.	El programa imprime 0 ya que el árbol era NULL.

Tecnológico de Monterrey
Edgar Cruz Vázquez (A01730577)
TC1031.11: Programación de estructuras y algoritmos fundamentales (GPO 11)

Act 3.1 – Operaciones avanzadas en un BST

Explicación de las funciones implementadas:

```
public:
    bst* insert (bst *, int);
    int height(bst*);
    void visit(int);
    void inorder (bst *);
    void preorder (bst *);
    void postorder (bst *);
    void lbyl (bst *);
    bool ancestors(bst*,int);
    void whatlevelaml (int);
    void searchvalue (bst *,int);
```

- Bst* insert (bst *, int);

```
bst* bst_tree::insert(bst* r, int v){
    if(r == NULL){
        r = new bst;
        r->dato = v;
        r->left = NULL;
        r->right = NULL;
        return r;
    }

    else if (v < r->dato){
        r->left = insert(r->left, v);
    }

    else if (v >= r->dato){
        r->right = insert(r->right, v);
    } return r;
}
```

Esta función, en resumen, recibe como parámetros el árbol y un entero el cual viene representando el dato a añadir, primero entra en un if en caso de que el árbol este vacío o este haya encontrado en alguna parte del subárbol (izquierda o derecha) un fin para ahí añadir el nuevo nodo con el dato.

En caso de que este no se encuentre vacío el dato se ira moviendo conforme a las reglas de acomodación de datos del árbol, ya sea irse al subárbol derecho o al subárbol izquierdo.

Tecnológico de Monterrey
Edgar Cruz Vázquez (A01730577)
TC1031.11: Programación de estructuras y algoritmos fundamentales (GPO 11)

Act 3.1 – Operaciones avanzadas en un BST

```
int bst_tree::height(bst* t){
    int h = 0;
    if (t != NULL){
        int l_height = height(t->left);
        int r_height = height(t->right);
        int max_height = max(l_height, r_height);
        h = max_height + 1;
    }
    return h;
}
```

Esta función devuelve un entero y tiene como parámetros el árbol, el cual el número que devuelve significa la cantidad de niveles que tiene un árbol, para esta función se ocupa la recursividad para ir calculando nivel por nivel, sumándole a cada nivel uno, así obteniendo el resultado.

```
void bst_tree::visit(int m){
    if (m == 1)
        return preorder(r);
    else if (m == 2)
        return inorder(r);
    else if (m == 3)
        return postorder(r);
    else if (m == 4)
        return lbyl(r);
    else{
        cout << "modo de recorrido no valido." << endl;
    }
}
```

```
void bst_tree::preorder(bst* t){
    if(t == NULL)
        return;
    cout << t->dato << " ";
    preorder(t->left);
    preorder(t->right);
}
```

```
void bst_tree::inorder(bst* t){
    if(t == NULL)
        return;
    inorder(t->left);
    cout << t->dato << " ";
    inorder(t->right);
}
```

```
void bst_tree::postorder(bst* t){
    if(t == NULL)
        return;
    postorder(t->left);
    postorder(t->right);
    cout << t->dato << " ";
}
```

```
void bst_tree::lbyl(bst* t){
    if(t == NULL)
        return;
    queue<bst*> q;
    q.push(t);
    while (!q.empty()){
        int nodes = q.size();
        while (nodes > 0){
            bst* temp = q.front();
            cout << temp->dato << " ";
            q.pop();
            if(temp->left != NULL)
                q.push(temp->left);
            if(temp->right != NULL)
                q.push(temp->right);
            nodes--;
        }
    }
}
```

Para la función de visit();, este recibe como parámetro un int que el entero que ingrese el usuario representara el modo en el cual se imprimirá el árbol, ya sea 1, 2, 3, 4. Siendo 1 preorden, 2 inorder, 3 postorden y el número 4 la impresión por niveles, esta última no se aplico la recursividad como los tres anteriores, en la impresión por niveles se aplicaron conceptos y funciones del queue para así poder obtener el resultado deseado, esta función no fue hecha individualmente, se utilizo el apoyo de link de página web para su aplicación, link de la página:

Desplazamiento del árbol de orden de nivel (ichi.pro)

Tecnológico de Monterrey
Edgar Cruz Vázquez (A01730577)
TC1031.11: Programación de estructuras y algoritmos fundamentales (GPO 11)

Act 3.1 – Operaciones avanzadas en un BST

```
bool bst_tree::ancestors(bst* t, int v){
    if(t == NULL)
        return false;
    if(t->dato == v)
        return true;
    if(ancestors(t->left, v) || ancestors(t->right, v)){
        cout << t->dato << " ";
        return true;
    }
    return false;
}
```

Esta función lo que obtiene son los ancestros de un dato a buscar, los ancestros son los datos que están arriba del dato a buscar, en caso que se busque el primer dato solamente nos retornara un true ya que este no tiene datos ancestros, en caso se busque otro dato que se encuentre en un nivel diferente este entrara en condicional if y se llamara recursivamente tanto para el subárbol derecho como el izquierdo hasta encontrar el dato, ya encontrado el dato se irán imprimiendo los datos anteriores dependiendo el subárbol.

```
void bst_tree::whatlevelaml(int v){
    return searchvalue(r, v);
}
```

La última función a explicar es whatlevelaml();, el cual recibe como parámetro un entero que significa el dato a buscar y si lo encuentra muestra el nivel en el cual se encuentra y en caso no devuelve un -1, al entrar a la función, esta devuelve otra función el cual se llama searchvaule();, que tiene como parámetros el árbol y el valor a buscar, este pregunta primero si el árbol es NULL (no encontró el dato), en caso contrario se usara recursivamente la función para buscar el dato a encontrar (ya sea el subárbol derecho o izquierdo), una vez encontrado el dato se medirá la altura del árbol en el cual se encontró el número, para este caso se ajustó de tal forma que ya conocemos los niveles del árbol (3

```
void bst_tree::searchvalue(bst* t, int v){
    if (t == NULL)
        cout << -1 << endl;

    else if (v == t->dato){
        int ac;
        ac = height(t);
        if (ac == 3){
            cout << 0 << endl;
        }

        else if (ac == 2){
            cout << 1 << endl;
        }

        else if (ac == 1){
            cout << 2 << endl;
        }
    }


    else if (v >= t->dato){
        return searchvalue(t->right, v);
    }

    else if (v < t->dato){
        return searchvalue(t->left, v);
    }
}
```

Tecnológico de Monterrey
Edgar Cruz Vázquez (A01730577)
TC1031.11: Programación de estructuras y algoritmos fundamentales (GPO 11)

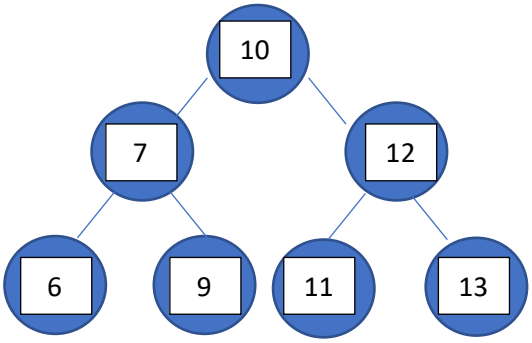
Act 3.1 – Operaciones avanzadas en un BST

niveles), pero este se puede ajustar para que pueda
recibir cualquier tipo de árbol, siendo en

El primer nodo, este nos devolverá 3 entonces lo igualaremos a 0, si es 2 devolveremos 1 y en caso sea 1 devolveremos 3.

FOTOS DE CASOS PRUEBA:

```
//arbol
r = bst.insert(r, 10);
r = bst.insert(r, 12);
r = bst.insert(r, 7);
r = bst.insert(r, 6);
r = bst.insert(r, 9);
r = bst.insert(r, 13);
r = bst.insert(r, 11);
```



Caso prueba 1:

```
//Caso prueba 1:
bst.visit(6);
```

```
modo de recorrido no valido.
```

Caso prueba 2:

```
//Caso prueba 2:
bst.whatlevelaml(30);
```

```
-1
```

Caso prueba 3:

```
//Caso prueba 3:
bst.ancestors(r, 30);
```

Tecnológico de Monterrey
Edgar Cruz Vázquez (A01730577)
TC1031.11: Programación de estructuras y algoritmos fundamentales (GPO 11)

Act 3.1 – Operaciones avanzadas en un BST

Caso prueba 4:

```
//Caso prueba 4:
cout << bst.height(r) << endl;
```

```
0
```

FUNCIÓN MAIN () { }

```
cout << "uso de funcion visit, orden por nivel" << endl;
bst.visit(4);

cout << endl;
cout << endl;

cout << "uso de la funcion height" << endl;
cout << bst.height(r) << endl;

 cout << endl;

cout << "uso de la funcion ancestors" << endl;
bst.ancestors(r,13);

cout << endl;
cout << endl;

cout << "uso de la funcion whatlevelaml" << endl;
bst.whatlevelaml(13);
```

```
uso de funcion visit, orden por nivel
10 7 12 6 9 11 13

uso de la funcion height
3

uso de la funcion ancestors
12 10

uso de la funcion whatlevelaml
2
```