Tecnológico de Monterrey

Edgar Cruz Vázquez (A01730577)

TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)

Act. 3.3 – Árbol Desplegado: Implementando un Splay Tree

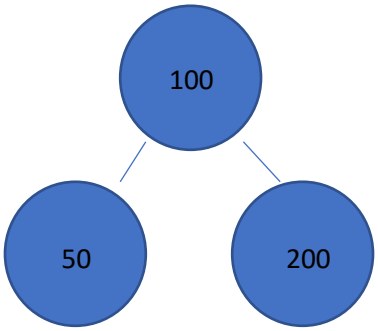
Instrucciones: Con el código proporcionado por tu profesor de Splay Tree, documenta y explica el proceso y funcionabilidad del programa.

Una de las características de un Splay Tree es el cual al momento de buscar un dato o número el número requerido lo mueve hasta la raíz o hasta el primer nodo haciendo que los demás se reacomoden, para poder ver mejor este árbol mostraremos el código y explicaremos paso a paso el proceso que se toma para hacer tal manipulación de datos:

Primero inicializamos los nodos que ocuparemos:

```
int main()
{
    node* root = newNode(100);
    root->left = newNode(50);
    root->right = newNode(200);
```

Primero inicializamos un nuevo nodo y lo que hacemos es que la raíz (primer nodo), tome el valor de 100 y que su lado izquierdo tenga el dato 50 y el derecho 200, resultando en el siguiente árbol:



Y para reafirmar que si es así usando la función search(); y la función preOrder(); imprimimos y verificaremos:

```
root = search(root, 100);
cout << "Preorder traversal of the modified Splay tree is \n";
preOrder(root);
return 0;
```

```
Preorder traversal of the modified Splay tree is
100 50 200
```

Por lo que logramos ver el árbol no sufrió ningún cambio al momento de imprimir y usar el preorden, debido a que buscamos el dato 100 y este estaba posicionado en el nodo raíz, pero ¿qué pasara si buscamos un dato diferente de 100 y lo imprimimos?

```
root = search(root, 50);
cout << "Preorder traversal of the modified Splay tree is \n";
preOrder(root);
return 0;
```

Tecnológico de Monterrey

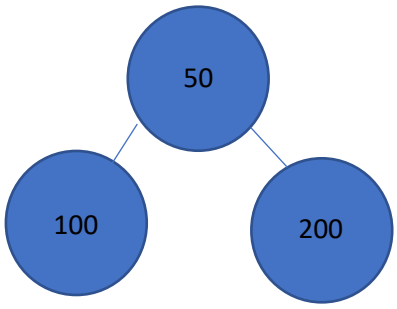
Edgar Cruz Vázquez (A01730577)

TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)

Act. 3.3 – Árbol Desplegado: Implementando un Splay Tree

```
Preorder traversal of the modified Splay tree is
50 100 200
```

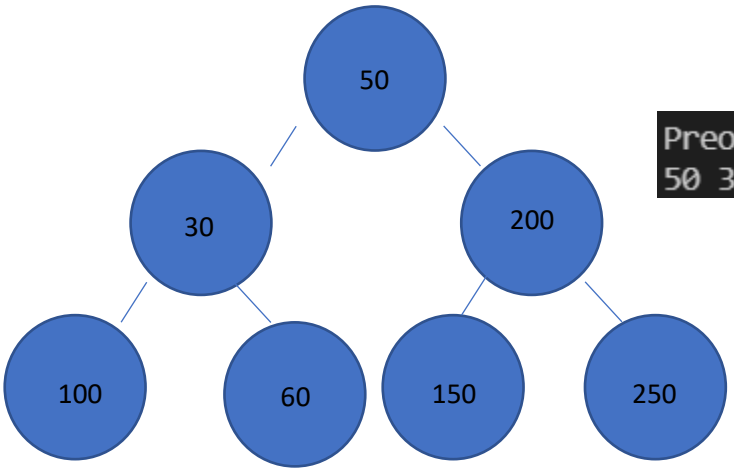
Vemos que, al momento de imprimir el árbol, este presenta un orden totalmente distinto, resultando en la siguiente secuencia:



Uno pensara solamente cambian de lugar, pero no es así, ya que si en caso llegáramos a agrandar el árbol y volvemos a hacer el mismo proceso este no solamente cambiara de lugar, sino que también modificara otros nodos como en el siguiente ejemplo:

```
int main()
{
    node* root = newNode(100);
    root->left = newNode(50);
    root->left->left = newNode(30);
    root->left->right = newNode(60);
    root->right = newNode(200);
    root->right->right = newNode(250);
    root->right->left = newNode(150);
```

En esta parte agrandamos el árbol de tal forma que este contiene 7 datos de forma balanceada:



Pero al aplicar la misma metodología de buscar el dato 50 al imprimir obtenemos este resultado:

```
Preorder traversal of the modified Splay tree is
50 30 100 60 200 150 250
```

Con esto dándonos cuenta que no solo se movieron tanto el 100 como el 50, también el 30 cambio de lugar por lo cual la función search(); es mas que un simple cambio de lugar, por lo cual esta actividad explicara el proceso y función del código de Slay Tree.

Tecnológico de Monterrey

Edgar Cruz Vázquez (A01730577)

TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)

Act. 3.3 – Árbol Desplegado: Implementando un Splay Tree

El sarch(); recibe como parámetros el árbol por referencia y un entero el cual es el dato a buscar y devuelve una función llamada splay(); que recibe los mismos parámetros que search();.

```
node* search(node* root, int key)
{
    return splay(root, key);
}
```

```
node* splay(node* root, int key)
```

Al entrar a la función primero recibe una condición en la cual pregunta si el árbol esta vacío o el dato (key) a buscar es el mismo del que se encuentra el primer nodo para terminar devolviendo el árbol, para este caso nos saltaremos esa condición.

```
if (root == NULL || root->key == key)
    return root;
```

Pasa a la siguiente condición en donde se pregunta si el valor del primer nodo es mayor al valor a buscar, queriendo esto decir que el dato se encuentra en el subárbol del lado izquierdo entrando a esta parte.

```
if (root->key > key)
```

Una vez adentro dentro de la condición existen otras tres condicionales que la primera preguntaba si el dato que debería estar en el lado izquierdo no existe, devolviendo el árbol y terminando la función, al ser falsa esta pasa a la siguiente condición.

```
if (root->left == NULL) return root;
```

La siguiente pregunta si el valor que se encuentra del lado izquierdo es mayor al valor a encontrar, este al ser falso ya que tanto el dato del lado izquierdo como el dato a buscar son los mismos (50) pasa a la siguiente condición que también se saltara ya que es la misma comparación de la anterior solamente que en vez de preguntar si es menor esta pregunta si es mayor, siendo esta otra vez falsa.

```
if (root->left->key > key)
```

```
else if (root->left->key < key) // Zig-Zag (Left Right)
```

Tecnológico de Monterrey

Edgar Cruz Vázquez (A01730577)

TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)

Act. 3.3 – Árbol Desplegado: Implementando un Splay Tree

Al no entrar a ninguna de las tres condiciones esta forza un return para acabar el proceso en la cual dentro del return se ejecuta una operación ternaria volviendo a preguntar si el subárbol del lado izquierdo esta vacia y en caso que no, entre a una función llamada rightRotate(); recibiendo como parámetro el árbol por referencia.

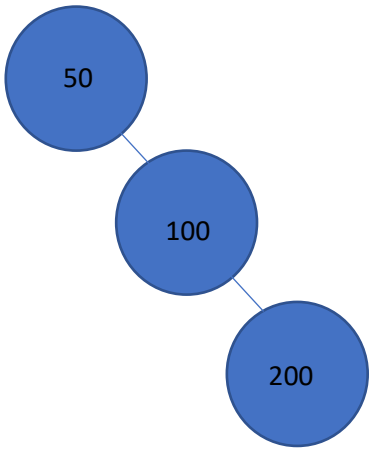
```
return (root->left == NULL) ? root : rightRotate(root);
```

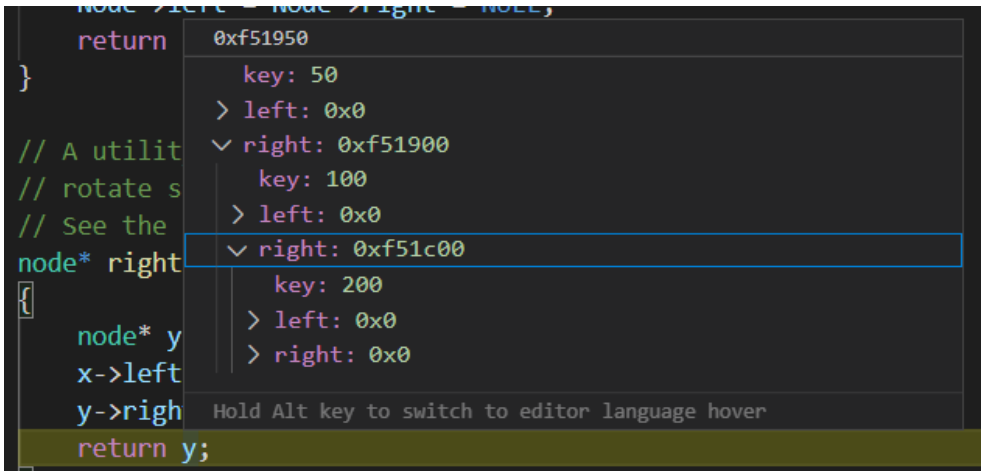
```
node* rightRotate(node* x)
```

Esta al ser falsa la comparación se entra a la función rightRotate(); se crea un nodo auxiliar que su primer nodo será el valor que guarda el árbol de su parte izquierda (50), actualizamos la parte izquierda del árbol original por la parte izquierda del árbol auxiliar el cual es un NULL y la parte derecha del árbol auxiliar es igual a los datos restantes del árbol original (100, 200), al final retornando el árbol auxiliar.

```
node* y = x->left;
x->left = y->right;
y->right = x;
return y;
```

Por lo cual si nos ponemos a pensar detenidamente la idea de cómo se movía presentada anteriormente es errónea ya que se pensó que el árbol guardaba la misma estructura (balanceada), quedando un árbol desbalanceado:





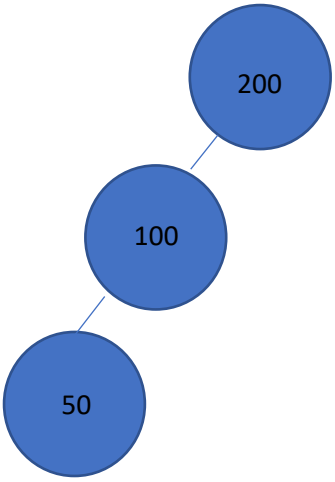
Por lo que vemos en este ejemplo es como si el dato a buscar va al primer nodo y los restantes cambian al subárbol derecho y si hiciéramos el search(); pero ahora con el 200, lo que nos debería de salir sería 200, 100, 50 todos colgados en el subárbol izquierdo quedando de la siguiente manera.

Tecnológico de Monterrey

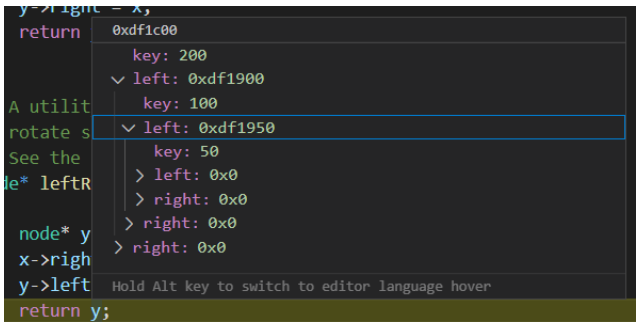
Edgar Cruz Vázquez (A01730577)

TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)

Act. 3.3 – Árbol Desplegado: Implementando un Splay Tree



Comprobamos el resultado:



Logramos ver que es cierto, los datos se movieron al subárbol izquierdo y el dato a buscar se movió al primer nodo.

Al final entrando a la función preOrder(); el cual lo hace es ir recorriendo el árbol en forma preorden e ir imprimiendo los datos.

```
preOrder(root);
```

```
void preOrder(node* root)
{
    if (root != NULL)
    {
        cout << root->key << " ";
        preOrder(root->left);
        preOrder(root->right);
    }
}
```

Lo cual lo que hace básicamente es primero imprimir el primer nodo y de ahí pregunta primero por la parte izquierda y después por la parte derecha y si hay dato los va imprimiendo.