Programación de estructuras de datos y

algoritmos fundamentales (Gpo 14)

Act 3.3 - Árbol Desplegado: Implementando un Splay Tree

Alumno:

Alejandro Daniel González Carrillo A01570396

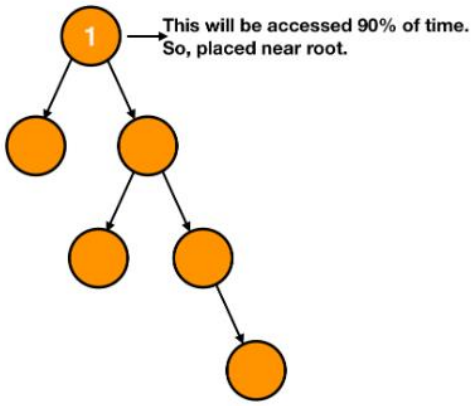
Profesores:
Luis Humberto González Guerra

Monterrey, Nuevo León. 20 de octubre de 2020.

Un splay tree es un arbol binario de bsuqueda que se ajusta acorde a lo que mas se accesa o lo que más se repite dentro de la busqueda. En el desarrollo del splay tree es posible insertar, buscar, borrar nodos y el mismo arbol binario se va a ajustando a las nuevas modificaciones. **Su peor caso de complejidad puede ser al realizar una operación de O(n)**. Un splay tree es recomendable usarlo cuando se sabe que se va a entrar al mismo nodo varias veces en un perido corto de tiempo. Entonces es muy funcional esta estructura ya que mantiene los nodos mas concurridos hasta arriba del arbol evitando que el programa gaste tiempo buscando entre todas las ramas.

Operation	Description	Complexity
Delete	Deletes a node given a key	$O(\log n)*$
Insert	Inserts a node with an associated key	$O(\log n)*$
Search	Searches for and returns a node using its key	$O(\log n)*$
Splay	Reorganises the tree, moving a particular node to the top	$O(\log n)*$

Alguna de las funciones que se utilizan al momento de estar utilizando un splay tree son **Delete, Insert, Search, Splay**. **Si se realiza cualquiera de estas operaciones el peor caso ahora es O(log(n)).** Delete como ya sabemos borra un nodo del arbol, Insert añade un nodo con su key asignada, Search hace una busqueda como cualquiera de todos los binary search trees que hemos hecho y finalmente el Splay que reorganiza los datos cada vez que se hace alguna modificación considero que podria estar dentro de cada funcion que modifique el arbol. Permitiendo que el nodo más concurrido siempre este hasta arriba.



Estuve leyendo algunos articulos y descubrí que existen diferentes **tipos de BST como el Regular BST, AVL tree, Splay tree, Red-Block tree**. En el articulo se menciona que la mejor manera o momento de usar un Splay BST es cuando los

nodos estan ordenados. **El Splay tree utiliza el principio de localidad, “90-10 rule”.**

El metodo utilizado en este arbol de busqueda binaria es el Splaying el cual realiza rotaciones para mover un nodo en especifico a la root.
El splay se realiza mediante operaciones llamadas zig, zig-zig y zig-zag dependiendo de donde esta el hijo o si tiene dos hijos. Los splay tree no siempre soy arboles balanceados y a veces estos cambian a arboles desbalanceados debido a las operaciones que se realizan.

En el codigo que nos brindo el profesor pude observar algunas cosas que son diferentes a lo que comunmente hacemos en clase. El nodo lo crean en una struct y no como una clase. Tenemos dos funciones de rotacion una para la derecha y una para la izquierda permitiendonos ir revisando los nodos de manera zig-zag o zig-zig. Tenemos el preOrder, inOrder, posOrder los cuales nos dejan reorganizar los datos de acuerdo a como los ocupemos desplegar. Para el metodo splay se utiliza recursividad, primero que nada se checa que el primer elemento exista. Si existe, entonces divide el arbol en dos partes primero va hacia el lado izquierdo solo si la key de la root es mayor que el del hijo izquierdo, sino entra al derecho. Este comienza ha hacer las rotaciones de manera zig-zig primero sino existe mas valores se pasa a zig-zag repitiendolo hasta que el hijo izquierdo del root sea nullptr. Y sucede igual para el hijo derecho.

Rotacion Derecha

```
node *rightRotate(node *x)
{
    node *y = x->left;
    x->left = y->right;
    y->right = x;
    return y;
}
```

Rotacion Izquierda

```
node *leftRotate(node *x)
{
    node *y = x->right;
    x->right = y->left;
    y->left = x;
    return y;
}
```

Splay Izquierdo

```
node *splay(node *root, int key)
{
    // Base cases: root is nullptr or
    // key is present at root
    if (root == nullptr || root->key == key)
        return root;

    // Key lies in left subtree
    if (root->key > key)
    {
        // Key is not in tree, we are done
        if (root->left == nullptr) return root;

        // Zig-Zig (Left Left)
        if (root->left->key > key)
        {
            // First recursively bring the
            // key as root of left-left
            root->left->left = splay(root->left->left, key);

            // Do first rotation for root,
            // second rotation is done after else
            root = rightRotate(root);
        }
        else if (root->left->key < key) // Zig-Zag (Left Right)
        {
            // First recursively bring
            // the key as root of left-right
            root->left->right = splay(root->left->right, key);

            // Do first rotation for root->left
            if (root->left->right != nullptr)
                root->left = leftRotate(root->left);
        }

        // Do second rotation for root
        return (root->left == nullptr)? root: rightRotate(root);
    }
```

Splay Derecho

```
else // Key lies in right subtree
{
    // Key is not in tree, we are done
    if (root->right == nullptr) return root;

    // Zag-Zig (Right Left)
    if (root->right->key > key)
    {
        // Bring the key as root of right-left
        root->right->left = splay(root->right->left, key);

        // Do first rotation for root->right
        if (root->right->left != nullptr)
            root->right = rightRotate(root->right);
    }
    else if (root->right->key < key)// Zag-Zag (Right Right)
    {
        // Bring the key as root of
        // right-right and do first rotation
        root->right->right = splay(root->right->right, key);
        root = leftRotate(root);
    }

    // Do second rotation for root
    return (root->right == nullptr)? root: leftRotate(root);
}
```

Search

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

Referencias

CodesDope. (2020). Splay Trees. Recuperado el 21 de octubre del 2020 en https://www.codesdope.com/course/data-structures-splay-trees/

Growing with the Web. (2015, 9 de junio). Splay Tree. Recuperado el 21 de octubre del 2020 en https://www.growingwiththeweb.com/data-structures/splay-tree/overview/

Cs Cornell. (2020). Splay Trees. Recuperado el 21 de octubre del 2020 en http://www.cs.cornell.edu/courses/cs3110/2011sp/Recitations/rec25-splay/splay.htm