Project Report

for

Splay Tree

Version 1.0 approved

Group 26

Prepared by:
Jason Aguirre, Gavin Doherty, and
Afolabi Abayomi

<CSC-212: Data Structure & Abstractions>

<November 20, 21>

Introduction

The goal of this project is to sort strings into splay trees and visualize them using Graphviz. We will be sorting strings(words) into our splay tree which will require us to use the STL string library, which will allow us to compare strings. Graphviz is a graph visualization software that will allow us to create a DOT file to visualize a splay tree after an operation has been performed. We plan on taking in a text file with strings separated by commas as inputs to our splay tree program. The user will be prompted with various options to manipulate the data structure; these options will include search, insert, and delete. The search function will also return the repeat count that each node will carry. Inserting a node that already exists will increase the repeat count of that node, as well as return the updated repeat count to the user. When the user is satisfied with their manipulations, they will be able to output a text file with graphviz commands that will accurately visualize the splay tree in its current state with each node containing its key, repeat count, and arrows pointing towards its children, if any. Our splay tree data structure will be implemented using a class.

1.1 Methods

A splay tree is a modified form of binary search tree (BST) that allows recently searched data to be accessible in O(1) time if accessed again; which is an improvement in comparison to the worst-case time complexity of a regular BST which is O(n). This is achieved by moving the recently accessed node to the root of the tree while keeping the binary tree structure intact. The structure remains intact when the left subtree of a node contains only nodes with keys lesser than the node's key, the right subtree of a node contains only nodes with keys greater than the node's key,

and the left and right subtree each are also binary search trees. A splay tree is self-adjusting meaning that it does not carry any balancing instructions. The tree adjusts itself by taking advantage of a method referred to as “splaying”.

Splaying is a series of rotations that takes a recently accessed node to the root of the tree depending on the position of the node relative to its parents and grandparents, if any. In the context of binary search trees, a parent represents the node that a leaf node is connected to, while grandparent represents the parent of the parent. These rotations take on one of three patterns; zig/zag, zig-zig/zag-zag, and zig-zag/zag-zig. In layman's terms zig represents a right rotation and zag represents a left rotation. While zig and zag represent single rotations, which occurs when a node doesn't have a grandparent node, zig-zig/zag-zag and zig-zag/zag-zig are double rotations which occur when a node does have a grandparent node. The order in which the zig and zag appear in any of the double rotations represents the sequence of rotations it will make. For example, a zig-zag rotation means that the node will rotate right before rotating left. A zig rotation will occur when no grandparent node exists and the current node is a left child. A zag rotation will occur when no grandparent node exists and the current node is a right child. A zig-zig rotation will occur when the current node is a left child and the current node’s parent is also a left child. A zag-zag rotation will occur when the current node is a right child and the current node’s parent is also a right child. A zig-zag rotation will occur when the current node is a right child and the current node’s parent is a left child. A zag-zig rotation will occur when the current node is a left child and the current node’s parent is a right child.

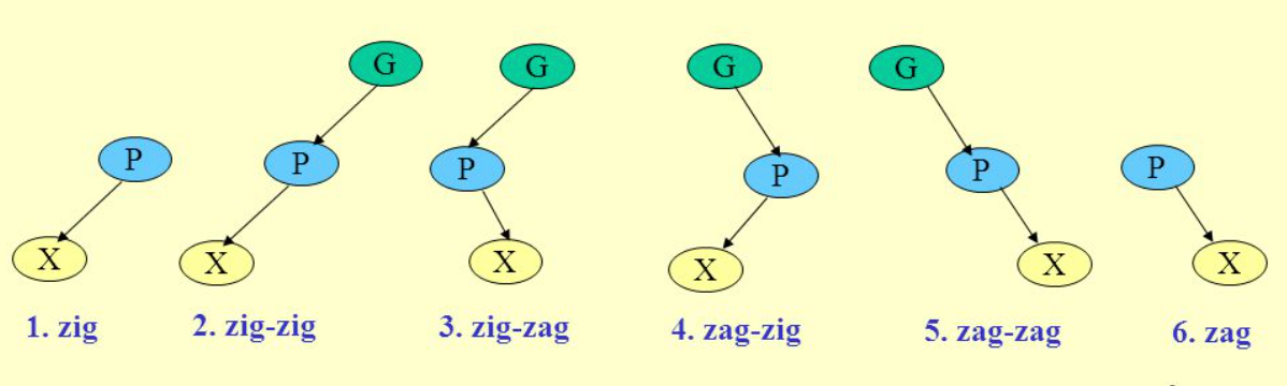

Figure 1. Representation of splay rotations in relation to position of node X.

https://slideplayer.com/slide/5962440/

Other operations that can be applied to splay trees include search, insert and delete. In the worst case, if nodes are accessed in ascending order, the complexity of splay tree operations would be O(n). Otherwise; splay, search, insert, and delete have an amortized time complexity of O(log n).

1.2 Implementation

Splay function

The goal of the splay function is to determine what kind of rotation the current node should make. The function parses the data values throughout the tree by the value of the key. By comparing values of the key of the current node to the values of the key of the current node's parents, the function will call either rightRotate or leftRotate. These functions will return the pointer to the left child or right child respectively which become the current node. This node is once again compared to the values of the key of the current node's parents. This function will run recursively until a null pointer is reached, it will then return the current node which will now be the root node.

rightRotate and leftRotate

```
//Zig rotation of tree to take the left child node and making it the parent node
    node* rightRotate(node* curr) { //takes in the memory address of the node we give it
        node* leftChild = curr->left; //creates a node called leftChild giving it the memory addr
        curr->left = leftChild->right; //now taking the curr left and setting it to leftChilds ri
        leftChild->right = curr; //leftChilds right child is now the current node at the top
        return leftChild;
    }

//Zag rotation of tree to take the left child node and making it the parent node
    node* leftRotate(node* curr) { //takes in the memory address of the node we give it
        node* rightChild = curr->right;//creates a node called rightChild giving it the memory ad
        curr->right = rightChild->left;//now taking the curr right and setting it to RightChilds
        rightChild->left = curr;//leftChilds right child is now the current node at the top
        return rightChild;
    }
```

The goal of rightRotate is to perform the zig rotation we described in the methods section while the leftRotate function performs the zag rotation that we also described in the methods section. rightRotate sets the left pointer of root to the right child of leftChild, then change leftChild's right pointer to point to root. In this case, leftChild is the node we are rotating. leftRotate sets the right pointer of curr to the left child of rightChild, then change rightChild's left pointer to point to root. In this case, rightChild is the node we are rotating.

Insertion

```
//inserts node with value of key into appropriate location in tree using insertHelper
    void insert(string key) {
        int duplicate;
        if (root == NULL) { //unless root is NULL then the newly created node is the first node
            root = createNode(key);
        }
        root = insertHelper(root, key);
        if (root->data == key) { //once the data in the curr node == key we return the mem address of cu
            cout<<key<< " is already in the tree \n";
            duplicate ++;
            cout<<"Number of duplicates: "<< duplicate << endl;
        }

    }
```

```
node* insertHelper(node* curr, string key) {
    curr = splay(curr, key);
    if (curr->data == key) { //once the data in the curr node == key we return the mem address of curr
        return curr;
    }

    node* newNode = createNode(key); //here we create a new node filled with the key of parameter w/
                                     //left and right pointers being NULL

    if (curr->data > key) { //if currdata is greater than key, insert new node pushes curr node to the right
        newNode->right = curr; //and curr left child is now left of newNode
        newNode->left = curr->left;
        curr->left = NULL;
    } else { //curr data is less than key. curr becomes the left child of newNode
        newNode->left = curr; // cur right child is now right of NewNode
        newNode->right = curr->right;
        curr->right = NULL;
    }

    return newNode;
}
```

The goal of the insertion function is to insert a new node into the tree. We first check if the tree is empty, if it is then we just set the new node as the root. If not, we first splay the tree to bring the closest leaf node to the root, then check if the key of the root is equal to the key we are inserting. If it is, then we just return root since the root key is already the key we are inserting. If not, we create a new node with the key we are inserting. If the root's key is greater, make root the right child of newNode and copy the left child of root to the newnode. If the root's key is smaller, make root the right child of newNode and copy the left child of root to the newNode (root is represented as curr in our code). We then return newNode as the root of the tree.

Delete

The goal of the delete function is to delete a node in the tree while keeping the binary search invariance intact. If the current node is NULL then return the current node, enter comparisons of key value to child nodes of the current node. If the key is greater than, call deleteNode again with current nodes right child If the key is less than, call deleteNode again with current nodes left child Repeat this cycle until either: Both left and right children of parent are NULL (return NULL) Left Child is NULL (return right child) Right Child is NULL (return left child) At which point we can return and delete our current node.

Search

The goal of the search function is to find a given key inside the tree. We first call the splay function to bring the closest leaf node up to the root. If the given key is equal to the key of the root, the key was found in the tree and is placed at the root. If the given key is not equal to the key of the root, then the last accessed leaf node will be at the root.

Team Contributions

Name	Contact	Contributions
Gavin Doherty	gavin_doherty@uri.edu, 7819874802	Provided splay tree class, added command line arguments, added deletion.
Jason Aguirre	jason_aguirre@uri.edu, 4013451706	Implemented Graphviz tool, created presentation, wrote project report, some source code troubleshooting.
Afolabi Abayomi	aabyomi786@uri.edu, 708-979-0102	Project report template and writing