Student Name: A Z Hasnain Kabir
Student ID: 200042102

CSE 4303 Data Structures
Assignment on Splay Tree

Introduction

A splay tree is a self-adjusting binary search tree. Although other data structures such as AVL trees and Red Black trees are also self-balancing, the uniqueness of a splay tree is defined by its ability to splay. Splaying is a concept where left and right rotations occur to bring an element to the root. Through splaying, more Useful elements in a tree can be brought closer to the root of the tree to increase operations.

Splaying

Rotations in a splay tree are somewhat similar to an AVL tree. In total of six rotations are possible in a splay tree.

- Zig (right) rotation.
- Zag (left) rotation.
- Zig zag (right then left).
- Zag zig (left then right).
- Zig zig (right then right again).
- Zag zag (left then left again).

Operations on a Splay Tree

A splay tree contains operations such as insertion, deletion, searching, and splaying

Advantages of a Splay Tree

- In an AVL tree, we need to store the balance factor to determine when to make rotations. In Red Black Tree, we store the color of each node. In a splay tree, no balance factor or color is needed to be stored for splaying.
- Splay tree is similar to a binary search tree to some extent. However, it is significantly faster than a binary search tree. In fact, it is the fastest version of a binary search tree.
- Search time on a splay tree is reduced due to having more important nodes closer to the root. So, data access is much faster.

Disadvantages of a splay tree

Splay tree is not strictly balanced. It is roughly balanced. For this reason, if we consider the worst case when tree is linear, its time complexity will be O(n).

Code

```
#include<iostream>
using namespace std;

class node{
public:
    int key;
    node *left, *right;
};

node *TreeNode(int key)
{
    node *Node = new node();
    Node->key = key;
    Node->left = Node->right = NULL;
    return (Node);
}

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

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

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

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

    if (root->key > key)
    {

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

```

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

            root->left->left = splay(root->left->left, key);

            root = right_rotate(root);
        }
        else if (root->left->key < key)
        {
            root->left->right = splay(root->left->right, key);

            if (root->left->right != NULL)
                root->left = left_rotate(root->left);
        }

        return (root->left == NULL) ? root : right_rotate(root);
    }

    else
    {
        if (root->right == NULL)
            return root;

        if (root->right->key > key)
        {

            root->right->left = splay(root->right->left, key);

            if (root->right->left != NULL)
                root->right = right_rotate(root->right);
        }

        else if (root->right->key < key)
        {

            root->right->right = splay(root->right->right, key);
            root = left_rotate(root);
        }

        return (root->right == NULL) ? root : left_rotate(root);
    }
}

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

Complexity Analysis

In the case of a binary search tree, the time complexity for the worst case for insertion, deletion, and search operations is O(n). For the average case, complexity is O(log n). The objective of the splay tree was to reduce the cost of operations significantly. A splay tree conducts insertion, deletion, and search operations in O(log n) amortized time.

Usecases of a Splay tree

Problem

We are required to find the median from an ordered data stream containing integers when subsequent integer values are getting added to the stream.
For example,
in a stream of (2, 3, 4), the median is 3
in a stream of (2, 3), the median is 2.5

Solution

In this problem, we can use splay tree to find an efficient solution in O(log n) time complexity.