1. Introduction

Efficiently maintaining and manipulating sets of elements from a totally ordered universe is a fundamental problem in computer science. Specifically, many algorithms need a data structure that can efficiently support at least the following operations: insert, delete, predecessor, and successor, and perform these operations in the theoretical minimum bound of O(logn) comparisons per operation. A standard data structure that maintains a totally ordered set and supports these operations is a binary search tree (BST). Research over the years has focused on achieving the O(loglogn) competitive bound on O(logn) amortized complexity for access in a BST.

Multi Splay is one such data structure that achieves the competitive bound of O(loglogn). The Multi Splay is one such variant of the Splay Tree which is conjectured to be dynamically optimal. A Splay Tree is a Binary Search Tree which performs an extra operation called splaying. Splaying is an algorithm which along moves a node to the root upon access, such that the BST property is still maintained. Splay Tree have various useful properties which help us reduce the amortized cost of accessing elements in the BST. Some of these properties are Sequential Access Property, Dynamic Finger Property, Working Set Property, Dynamic Optimality, etc.

A Multi-Splay Tree is a Binary Search Tree that evolves over time, and preserves a tight relationship with something known as the reference tree. Each node of the Multi-Splay tree acts as a Splay tree, wherein the nodes have several different types of properties. Reference Tree is nothing but a balanced BST made up of say n nodes and each node in this reference tree has a preferred child. In totality, the reference tree is static but the preferred children will change over time. The reference tree is not explicitly part of our data structure, but is useful in understanding how it works. Implementation of the Multi-Splay Tree implies effectively Splaying a Splay Tree.

2. Functions Implemented

This section gives concise explanations to the various functions we have used in the code to implement the Multi-Splay Tree.

1. BuildTree(): It is a function to create the entire tree. The Tree T stores the key value, minDepth, depth andthe information of whether it is a root or not.

2. Rotate(): This function is for performing the rotation operation.

3. Splay(): This function is for performing the splay operation on the node

4. SwitchPath(): Like splay tree, this is also a self adjusting update operation, switch is performed to swap thepreferred child

5. Search(): To access a particular element using the Multi-Splay Algorithm in the Multi-Splay Tree

6. Display(): To display the entire tree.

3. Figures, Implementation and Algorithms

3.1. Figures

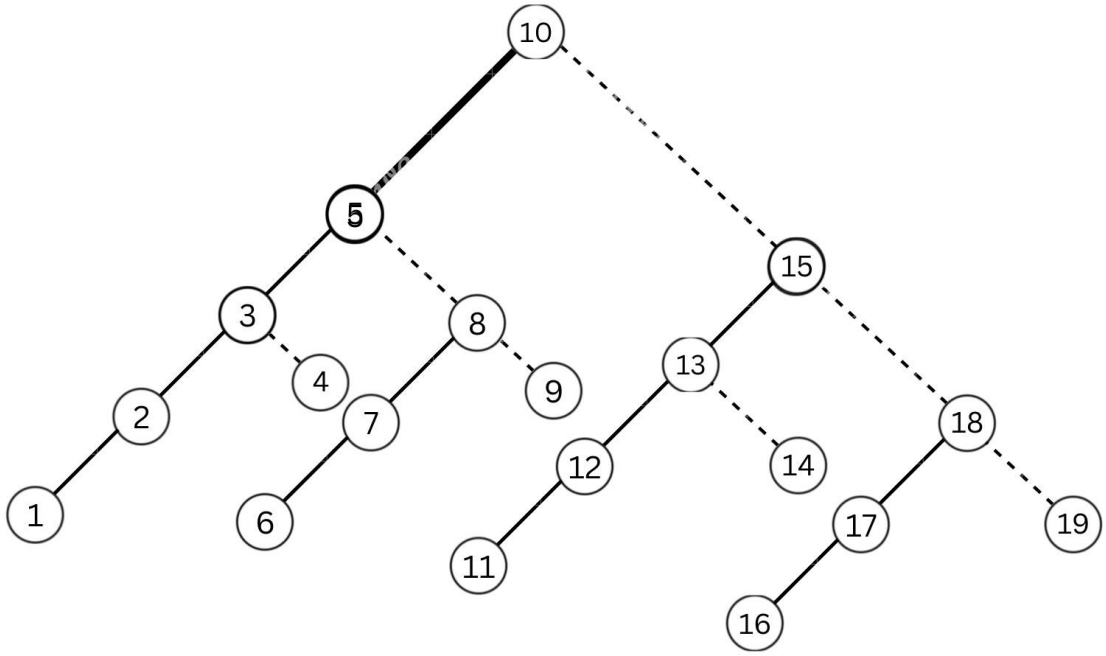
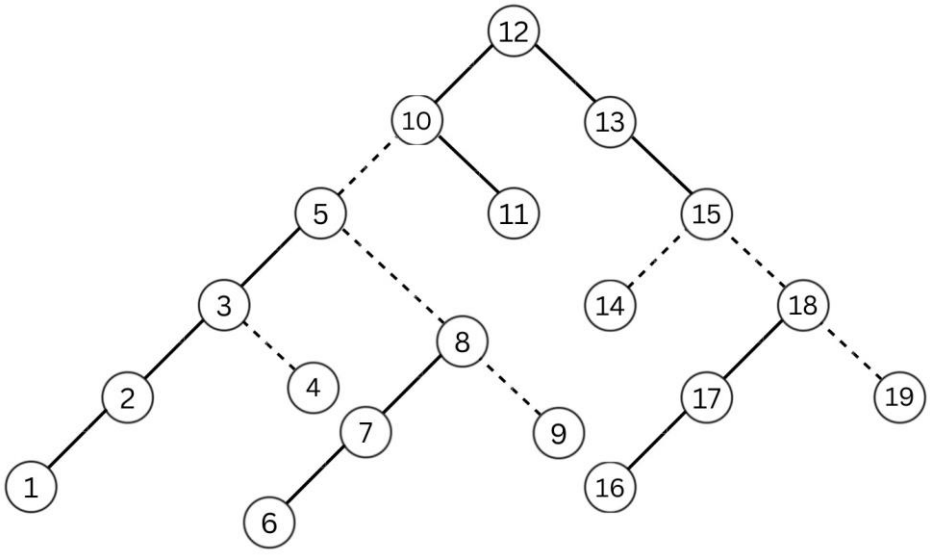
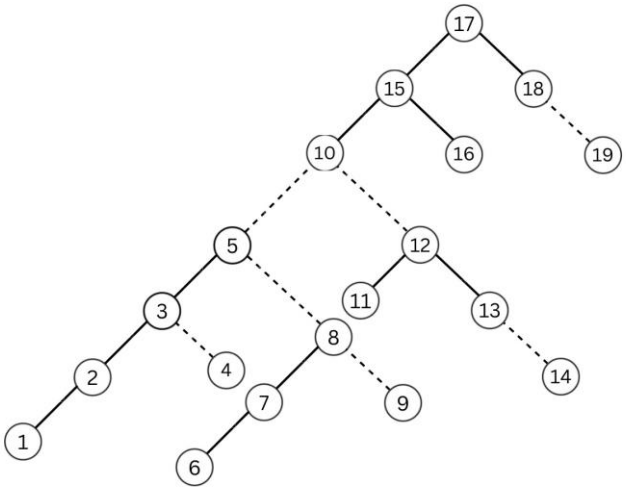


Figure 1: Original Tree.



(a) Accessing Node 12



(b) Now accessing node 17

Figure 2: Caption of the Figure.

3.2. Implementation

When we are splaying a particular node in the Multi-Splay Tree, the entire sub-splay tree associated with that node i.e. all the data associated with the data we want to access is splayed and brought to the root ensuring quick and efficient access.

In the figures above we wish to access the node 12 from our Multi-Splay Tree and when we perform the MultiSplay Algorithm it is evident from the Figure (2) that the entire sub-splay tree of the node 12 is also present at the root.

3.3. Algorithms

Like splay tree, there is a self-adjusting update algorithm that rotates a key to the root. This algorithm is called the multi-splay algorithm. In this section, we first explain the algorithm assuming we have the reference tree P, then we explain how to implement the corresponding operations in our actual representation T. As stated above, the preferred edges in P evolve over time. A switch at a node just swaps which child is the preferred one. For each access, switches are carried out from the bottom up, so that the accessed node v is on the same preferred path as the root of P. In addition, one last switch is carried out on the node that is accessed. In other words, traverse the path from v to the root doing a switch at each parent of a non-preferred child on the path, and then finally switch v.

Algorithm 1 Multi-Splay Algorithm

```
1: while node->parent != NULL do
2: splay(node)
3: end while
4: return
```

Algorithm 2 The Search() Function

Search until the end of the tree is reached
2: Search is implemented as per the Binary Search current =
current node
4: previous = parent node if
current==$NULL$ then
6: MultiSplay(previous) end if
8: MultiSplay(Current) return

---

4. Conclusions

With the help of this project we gained an idea about the working of a Multi-Splay Tree and the crucial dynamic optimality property associated with it. It is the only algorithm with O(loglogn) competitiveness in a BST.

Acknowledgements

Research Papers published by researchers at School of Computer Science, Carnegie Mellon University in Pittsburgh. [1–3]

References

[1] Chengwin Chris Wang. Multi-splay trees. *CMU-CS*, 2006.

[2] Jonathan Derryberry. Properties of multi-splay trees. 2009.

[3] Daniel Dominic Sleater. Dynamic optimality and multi-splay trees. *CMU-CS*, 2004.