**East West University**

**Department of Computer Science and Engineering**

**CSE207 PROJECT**

**Course Code: CSE207**

**Course Title: Data Structures**

**Section:** 01

**Semester:** Spring 2021

**Project Title:**

**Implementation of Splay trees**

**Submitted By:**

**Group Number:15**

**Group Members:**

**Name:** Shafia Hasnin

**Id:** 2020-1-60-209

**Name:** Zarin Tasnim Nuzhat

**Id:** 2020-1-60-211

**Submitted To:**

**Dr. Maheen Islam,**

Associate Professor,

Department of CSE

**East West University.**

**6/2/2021**

**Splay trees**

**Introduction**

Splay trees are the self-balancing or self-adjusted binary search trees. In other words, we can say that the splay trees are the variants of the binary search trees. The prerequisite for the splay trees that we should know about the binary search trees.

There are more self-balancing trees but what makes the splay tree unique than other self -balancing trees is that it has one extra property called “Splaying”. A splay tree contains the same operations as a Binary search tree, i.e., Insertion, deletion and searching, but all these operations are followed by splaying. Splay trees are not strictly balanced trees, but they are roughly balanced trees.

**Data structure used**

Splay Trees are a simple and efficient data structure for storing an ordered set. The data structure of Splay Tree consists of a Binary Search Tree, with no additional fields. The Binary Search Tree is a data structure for representing tables and lists so that accessing, inserting, and deleting items are easy.

***Algorithm for Splaying:***

**SPLAY (T, n)**

**While n.parent! = NULL //node is not root**

**if n.parent == T.root //node is child of root, one rotation**

**if n == n.parent.left //left child**

**RIGHT\_ROTATE (T, n.parent)**

**else //right child**

**LEFT\_ROTATE (T, n.parent)**

**else //two rotations**

**p = n.parent**

**g = p.parent**

**if n.parent.left == n and p.parent.left == p //both are left children**

**RIGHT\_ROTATE (T, g)**

**RIGHT\_ROTATE (T, p)**

**else if n.parent.right == n and p.parent.right == p //both are right children**

**LEFT\_ROTATE (T, g)**

**LEFT\_ROTATE (T, p)**

**else if n.parent.right == n and p.parent.left == p**

**LEFT\_ROTATE(T, p)**

**RIGHT\_ROTATE (T, g)**

**else**

**RIGHT\_ROTATE (T, p)**

**LEFT\_ROTATE (T, g)**

***Algorithm for Searching:***

**SEARCH (T, n, x)**

**if x == n.data**

**SPLAY (T, n)**

**return n**

**else if x < n.data**

**return search (T, n.left, x);**

**else if x > n.data**

**return search (T, n.right, x);**

**else**

**return NULL**

***Algorithm for Insertion:***

**INSERT (T, n)**

**temp = T.root**

**y = NULL**

**while temp! = NULL**

**y = temp**

**if n.data < temp.data**

**temp = temp.left**

**else**

**temp = temp.right**

**n.parent = y**

**if y==NULL**

**T.root = n**

**else if n.data < y.data**

**y.left = n**

**else**

**y.right = n**

**SPLAY (T, n)**

***Algorithm for Deletion:***

**DELETE (T, n)**

**left\_subtree = new splay\_tree**

**right\_subtree = new splay\_tree**

**left\_subtree.root = T.root.left**

**right\_subtree = T.root.right**

**if left\_subtree.root! = NULL**

**left\_subtree.root.parent = NULL**

**if right\_subtree.root! = NULL**

**right\_subtree.root.parent = NULL**

**if left\_subtree.root!= NULL**

**m = MAXIMUM(left\_subtree, left\_subtree.root)**

**SPLAY (left\_subtree, m)**

**left\_subtree.root.right = right\_subtree.root**

**T.root = left\_subtree.root**

**else**

**T.root = right\_subtree.root**

**Conclusion**

There are many advantages of Splay Trees. Some of the most useful advantages are -

1. In the splay tree, we do not need to store the extra information. In contrast, in AVL trees, we need to store the balance factor of each node that requires extra space, and Red-Black trees also require storing one extra bit of information that denotes the color of the node, either Red or Black.
2. It provides better performance as the frequently accessed nodes will move nearer to the root node, due to which the elements can be accessed quickly in splay trees. It is used in the cache implementation, as the recently accessed data is stored in the cache so that we do not need to go to the memory for accessing the data, and it takes less time.

***Drawback:*** The major drawback of the splay tree would be that trees are not strictly balanced, i.e., they are roughly balanced. Sometimes the splay trees are linear, so it will take O (n) time complexity.