# Implementation and Analysis of Splay Trees

Hitesh Kumar
PES1201701511
PES UNIVERSITY
Bangalore, India
hiteshkumarhk.info@gmail.com

***Abstract*—A splay tree is a self-balancing binary search tree with the additional property that recently accessed elements are quick to access again. It performs basic operations such as insertion, look-up and removal in O(log n) amortized time . Thus splay trees are as efficient as balanced trees when total running time is the measure of interest. In addition, for sufficiently long access sequences, splay trees are as efficient, to within a constant factor, as static optimum search trees. The efficiency of splay trees comes not from an explicit structural constraint, as with balanced trees, but from applying a simple restructuring heuristic, called splaying, In a splay tree, every operation is performed at the root of the tree. All the operations in splay tree are involved with a common operation called "Splaying".In a splay tree, splaying an element rearranges all the elements in the tree so that splayed element is placed at the root of the tree.**

## I. Introduction

Splay trees are always better than Binary search trees ,when your application seals with a lot of data in tree but , will need access to a subset of the data very frequently than others . In this case the data you access frequently will come near the root as a result of the spaly. Because of this any node can be accessed with less time.On the positive side is that we get log(n) in both these data structures theoretically. Splay trees have an average of log(n) over a number of operations.Any long sequence of operations will take at most O(n log n ) time ,but individual operations might take as much as O(n) time in splay trees. Two varieties to approach splay trees are : Bottom up : First search the tree and rotate at the same iteration. Top down: first search the tree and rotate in another iteration. There are three caes : i) ZIG ii) ZIG-ZAG ii)ZIG-ZIG.

## II. Implementation

The splay tree has been implemented using following functions :

### *A. SPLAY*

Splaying is what keeps the splay tree roughly balanced. To splay a node, splaying steps are repeatedly performed on it until it rises to the top. To decide what kind of splaying step to perform, the tree looks at three possibilities:a)The node's parent is the root .b)The node is the left child of a right child (or the right child of a left child) c) The node is the left child of a left child (or the right child of a right child) If the node's parent is the root, we only need one rotation to make it the root. If the node is the left child of the root, we perform a right rotation, and if the node is the right child of the root, we perform a left rotation. This is exactly the same as the rotations in an AVL Tree.. This is sometimes known as the "zig" case. If the node is the left child of a right child, we need to perform two rotations. Let N be the node we are trying to splay, P is its parent, and G is its grandparent. It first rotates N and P right, then rotates N and G left. If the node is the right child of a left child, it does the opposite. It first rotates N and P left, then N and G right. This is sometimes referred to as the "zig-zag" case.

### *B. INSERTION*

To insert a node, find its appropriate location at the bottom of the tree using binary search. Then perform splay on that node.

### *C. DELETION*

Deletion is the only operation that has some wiggle room for the implementer. After all, there's no obvious node to splay when you're removing a node. A typical implementation is the programmer can switch the node to be deleted with the right-most node in its left subtree or the left-most node in its right subtree. Then the node can be deleted without any consequence to the tree (since it has no children). Another way is the node to be deleted is first splayed, making it a root node. Then it is deleted. We are left with two separate trees that are then joinedtogether using the join operation, which we will see later. Regardless, a typical implementation will eventually splay the parent of the deletednode. This is similarly up to the discretion of the implementer. .

### *D. FIND*

Find is simple once splay is implemented. To search for a node, use binary search down the tree to locate the node. Then, perform splay on that node to bring it to the top of the tree. This method returns 1 or 0 if the value is in the tree or not. If the node is present, it should be splayed to the

top, otherwise the last found node should be splayed. This function is implemented by first calling splay, then checking if the roots value is equal to the value to be found.

## III. APPLICATIONS

Splay trees are typically used in the implementation of caches, memory allocators, garbage collectors, data compression, ropes. Splay trees are used in Windows NT (in the virtual memory, networking, and file system code), the gcc compiler and GNU C++ library, the sed string editor, Fore Systems network routers, the most popular implementation of Unix malloc, Linux loadable kernel modules, and in much other software.