2. SPLAY TREE

2.1 Introduction:

Splay tree is also Self-Balancing like AVL and Red-Black Trees.

The main idea of splay tree is to bring the recently accessed item to root of the tree, this makes the recently searched item to be accessible in O(1) time if accessed again.

The idea is to use locality of reference (In a typical application, 80% of the access are to 20% of the items). Imagine a situation where we have millions or billions of keys and only few of them are accessed frequently, which is very likely in many practical applications.

2.2 Set Operations

The most important tree operation is **splay(x)**, which moves an element x to the root of the tree. In case x is not present in the tree, the last element on the search path for x is moved instead.

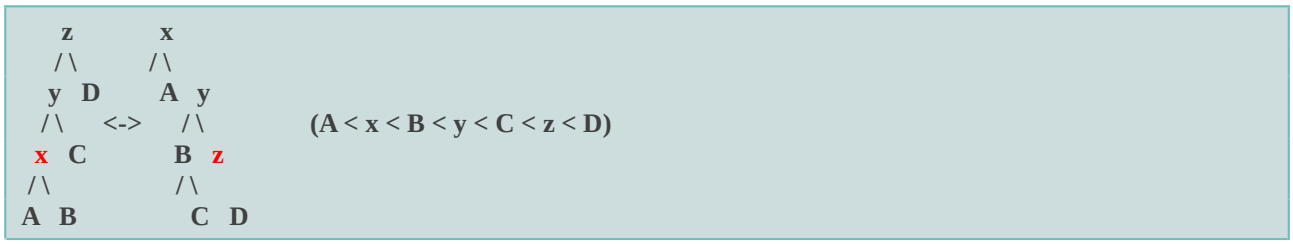
The run time for a splay(x) operation is proportional to the length of the search path for x: While searching for xwe traverse the search path top-down. Let y be the last node on that path. In a second step, we move y along that path by applying rotations as described below.

Rotation 1: Simple rotation (Zig and Zag)

The simple tree rotation used in AVL tree is also applied at the root of the splay tree, moving the splayed node **x** up to become the new tree root. Here we have **A < x < B < y < C**, and the splayed node is either **x** or **y** depending on which direction the rotation is. It is highlighted in red.

Rotation 2: Zig-Zig and Zag-Zag

Lower down in the tree rotations are performed in pairs so that nodes on the path from the splayed node to the root move closer to the root on average. In the "zig-zig" case, the splayed node is the left child of a left child or the right child of a right child ("zag-zag").



Rotation 3: Zig-Zag

In the "zig-zag" case, the splayed node is the left child of a right child or vice-versa. The rotations produce a subtree whose height is less than that of the original tree. Thus, this rotation improves the balance of the tree. In each of the two cases shown, **y** is the splayed node: