CS 4/56101

Design and Analysis of Algorithms

Kent State University

Dept. of Math & Computer Science

LECT-11

Splay Tree

2

Splay Tree

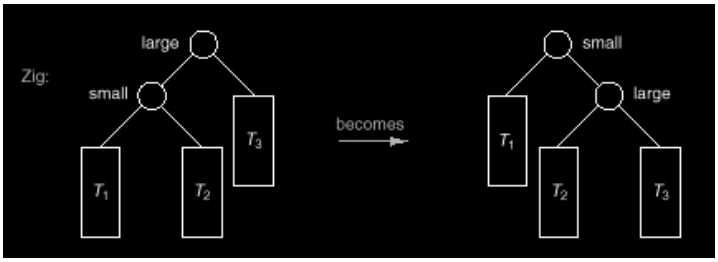
- Idea:
  - in a Binary Search Tree or even in an AVL tree where the new records are going?
  - The later a key/record is added the higher will be its access time!
  - Consider a hospital situation..
- Motivation
  - How can we keep frequently accessed/ recently used keys near the root?

Splay Tree

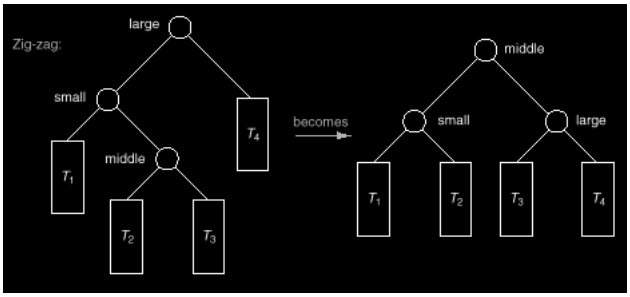
- A Self-adjusting Tree
  - A Binary Search Tree
  - Not so well balanced explicitly.
- Adjustments
  - Every time a new node is inserted, it is positioned at the root.
  - Every time a node is retrieved, it becomes the root.
- Performance
  - A sequence of **m** insertions or retrievals with splaying a binary search tree of size **n** will never need more than **m (1+3 log n) + log n** upward moves of a target node.

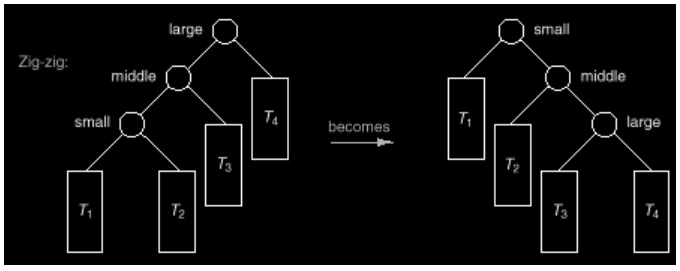
Idea of Zig and Zag

- Right-Rotation = Zig
- Left Rotation = Zag



Zig-Zag and Zig Zig

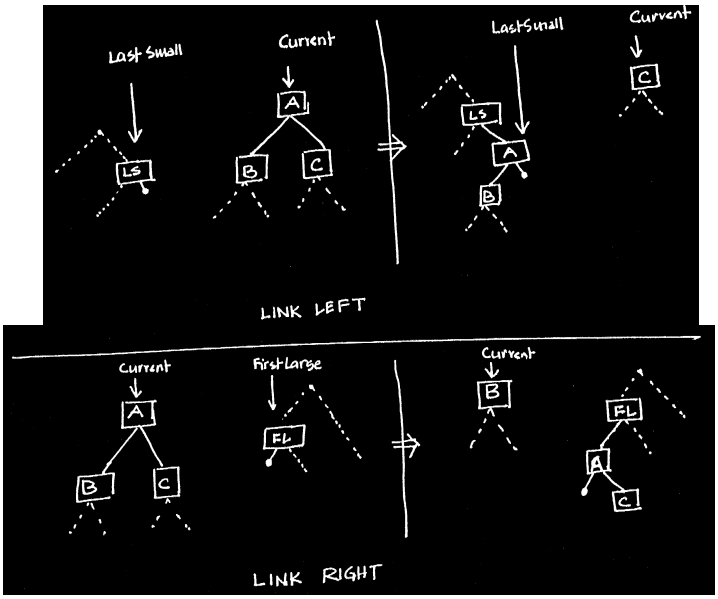




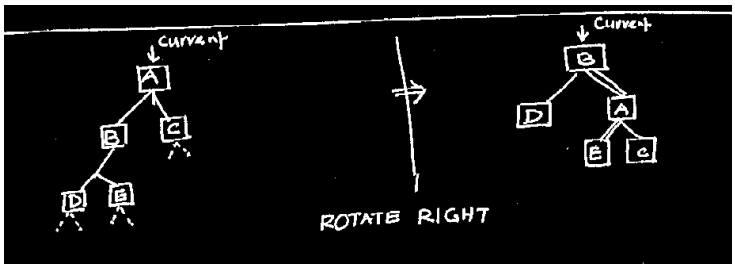
Basic Algorithm

- Start from the root and keep on splitting the tree applying zig/zag and eventually bringing the target at the root.
-
- Divide the tree in three parts:
  - Tree#1: all the nodes confirmed greater than the key.
  - Tree#3: all the nodes confirmed larger than the key.
  - Three#2: if the key is there it must be inside this tree.

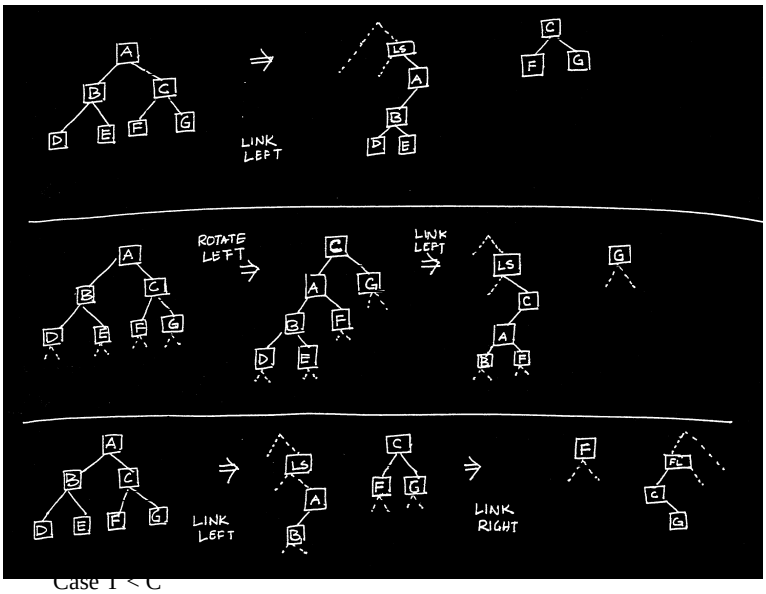
Some Operations



Some Operations (continued..)

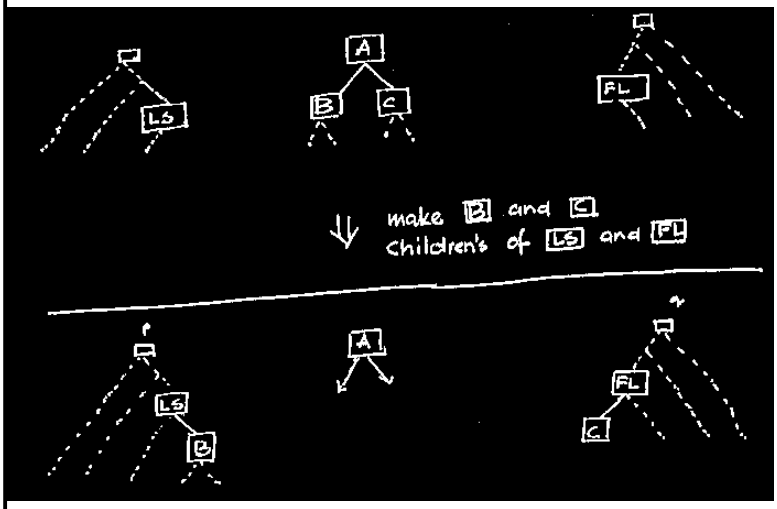


Algorithm

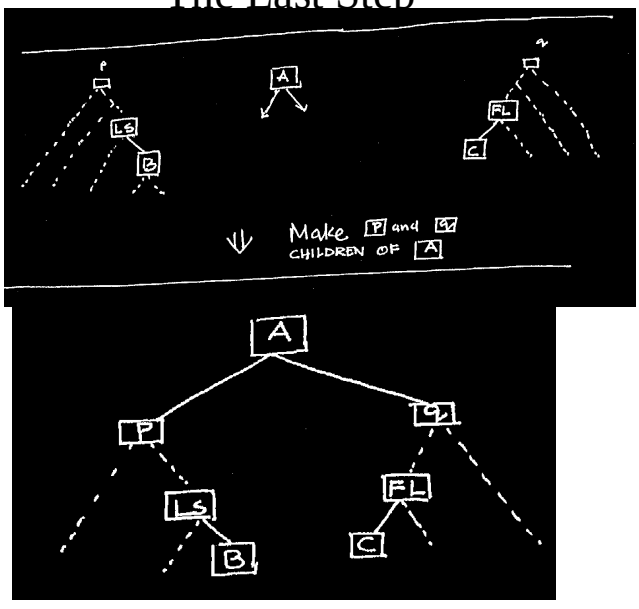


Case 1 < C

Final Two Steps..

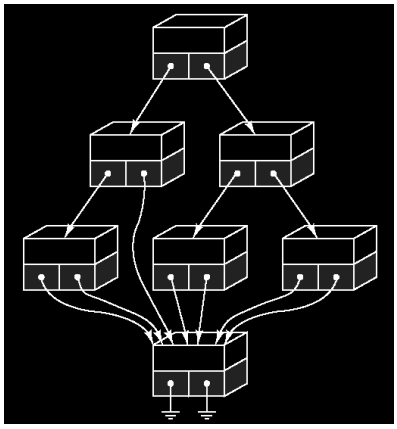


The Last Step



Sentinel Binary Tree

- No Need to Check for NULL Pointer. When we are at "Sentinel" node, we know we are at the leaf.
- In Splay Tree we start by copying the "target" at the sentinel node..
- The two pointers of the sentinel keeps track of the low and high sub-trees.




```
TreeNode *TreeSplay(TreeNode *root, KeyType target)
{
   TreeNode *current;        /* the current position in the tree
   TreeNode *child;          /* one of the children of current
   TreeNode *lastsmall;      /* largest key known to be less than the target
   TreeNode *firstlarge;     /* smallest key known to be greater than the target
   extern TreeNode *sentinel;

   sentinel->entry.key = target;   /* Establish sentinel for searching.
   lastsmall = firstlarge = sentinel;

   for (current = root; NE(current->entry.key, target); ) {
      if (LT(current->entry.key, target)) {
         child = current->right;
         if (EQ(target, child->entry.key)) {
            current = LinkLeft(current, &lastsmall);
         } else if (GT(target, child->entry.key)) {
            current = RotateLeft(current);
            current = LinkLeft(current, &lastsmall);
         } else {
            current = LinkLeft(current, &lastsmall);
            current = LinkRight(current, &firstlarge);
         }
      } else {
         child = current->left;
         if (EQ(target, child->entry.key)) {
            current = LinkRight(current, &firstlarge);
         } else if (LT(target, child->entry.key)) {
            current = RotateRight(current);
            current = LinkRight(current, &firstlarge);
         } else {
            current = LinkRight(current, &firstlarge);
            current = LinkLeft(current, &lastsmall);
         }
      }
   if (current == sentinel) {
      printf("Target has been inserted as root of the tree.");
      root = current = MakeNode(target, sentinel);
```