Data Structure I: Tutorial 2

Definition 1 (Binary tree) *Binary tree is a tree data structure in which each node has at most two children, referred to as the left child and the right child. Each node except the root has a parent. A node without any child is called leaf. Every node contains some data of one element and every element is stored in one node. Every element is identified by a unique key and these keys are comparable.*

Definition 2 (Binary search tree (BST)) *Binary search tree is a binary tree data structure with the key of each internal node being greater than all the keys in the respective node's left subtree and less than the ones in its right subtree; see Figure 1.*

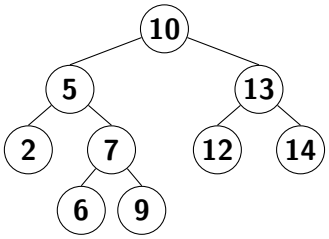

Figure 1: Example of a binary search tree.

Definition 3 (Basic operations in BST)

Find *a node containing a given key. Returns an invalid value if BST has no element with that key.*

Insert *a given element with its key. Returns False if the key is duplicate.*

Delete *the node containing a given key. Returns False if BST has no element with that key.*

Exercise 4 *Write algorithms for operations Find, Insert and Delete for BST. What is the worst-case complexity of these operations?*

Definition 5 (Single rotation in BST) *Given a node u different from the root, the single rotation of u with its parent p changes BST so that u becomes a parent of p as presented on Figure 2. Rotating a node u to the root means applying the single rotation on u until it becomes the root of BST.*

Exercise 6

1. *In Figure 2, rotate the node 7 to the root.*
2. *Consider an arbitrary BST containing elements with keys $1, \ldots, n$ and apply rotation to the root on all elements in increasing order. Describe the resulting BST.*
3. *Rotate all elements to the root once more. Estimate the number of single rotations needed in this procedure.*

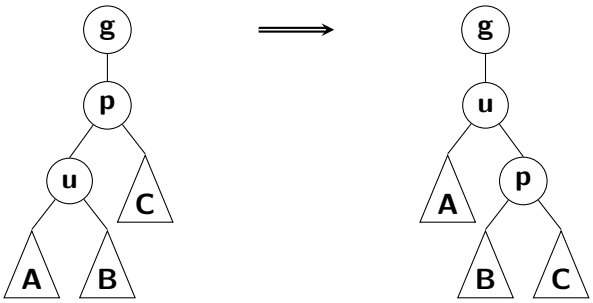

Figure 2: Single rotation of a vertex u.

Definition 7 (Operation Splay) *Zig-zag and zig-zig double rotations moves a given node u two level higher as presented on Figures 3 and 4. Operation Splay rotates a given node u to the root, but applies double rotations on u as long as possible and the last rotation can be single if needed.*

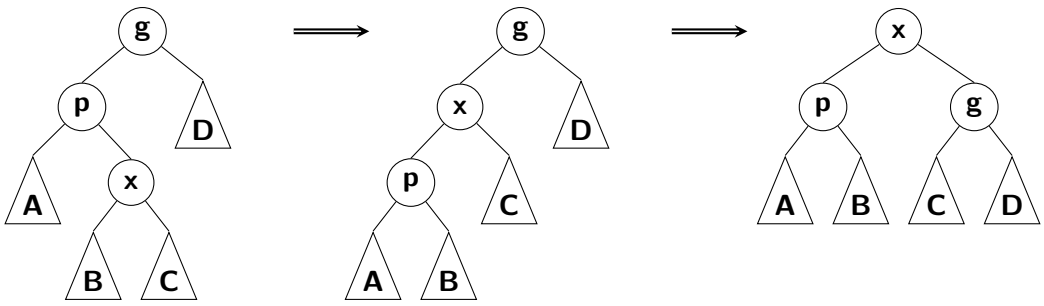

Figure 3: Zig-zag double rotation of a node u presented as two consecutive single rotations of u.

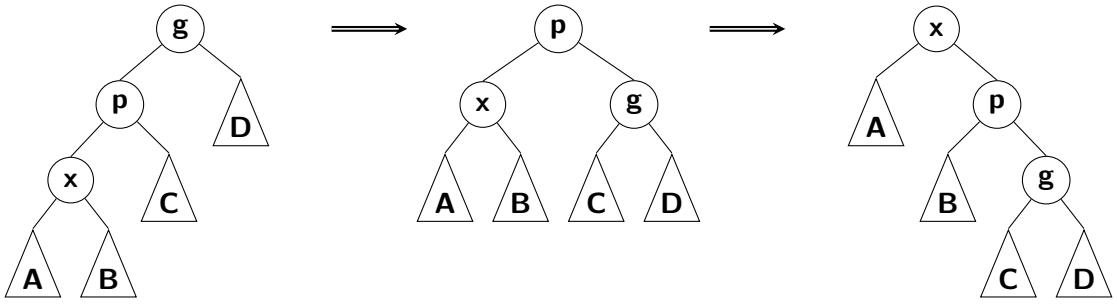

Figure 4: Zig-zig double rotation of a node u. The parent p of u is rotated and then u is rotated.

Emphasize that both double rotation can be easily implemented as two single rotations. Furthermore, zig-zig rotation of u cannot be implemented as two consecutive single rotations on u since this leads to an incorrect result; see Figure 5.

Definition 8 (Splay tree) *Splay tree is BST defined by the following operation.*

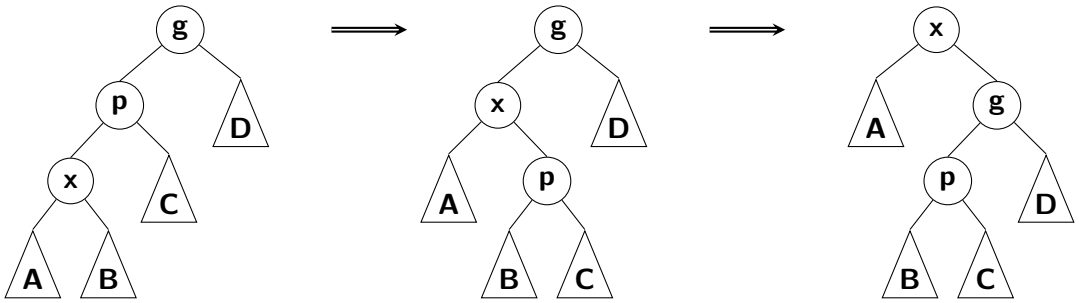

Figure 5: An incorrect implementation of Zig-zig double rotation using two consecutive single rotation of a node u.

Find *the node u containing a given key. If u exists, splay it; otherwise, splay the last visited node.*

Insert *the given element to a new leaf u as in BST. If the given key is not duplicate, splay u; otherwise, splay the original node containing the key.*

Delete *the node containing a given key. If the key exists, splay the deepest visited node including the replacement; otherwise, splay the last visited node.*

Emphasize that all operations always splay the deepest visited node even if failed.

Exercise 9 *Consider a BST on keys $1, \ldots, n$ forming the left path. Apply the operation Splay on keys 1, 2, and 3.*

Exercise 10 *The amortized complexity of operations Find, Insert and Delete in Splay tree is $O(\log n)$. This requires calling Splay on deepest visited node in every case. Given an example showing that this splaying is needed to achieve the desired complexity also in the following cases.*

1. *Splaying the last visited node in the operation Find when a given key is not present.*
2. *Splaying the inserted element.*
3. *Splaying the original element when inserting a duplicate key.*
4. *Splaying the last visited element when a given key to be deleted is not present.*
5. *When operation Delete requires replacement, show that it is not sufficient to Splay the node which contained the deleted key before replacement.*