CS2040C Semester 2 2018/2019

Data Structures and Algorithms

Tutorial 07 - Table ADT 2, Balanced BST

For Week 09

Document is last modified on: March 1, 2019

1 Introduction and Objective

The purpose of this tutorial is to reinforce the concepts of Binary Search Tree (BST) and the importance of having a balanced BST. In CS2040/C, we learn Adelson-Velskii Landis (AVL) Tree as one such possible balanced BST implementation.

In this tutorial, we will discuss two extra bBST operations: Select and Rank.

We then show the versatility of balanced BST data structure as an alternative implementation of ADT Priority Queue that we have learned earlier.

We will also discuss balanced BST versus Hash Table (discussed in Tut06) as implementation for Table ADT.

2 Tutorial 07 Questions

Basic Operations of (balanced) Binary Search Tree: AVL Tree

Q1). (Optional, only when majority are still not comfortable with basic bBST operations): We will start this tutorial with a quick review of basic BST operations, but on a balanced BST: AVL Tree. Tutor will first open `https://visualgo.net/en/avl`, click Create → Random. Then, the tutor will ask students to Search for some integers, find Successor of existing integers, perform Inorder Traversal, Insert a few random integers, and also Remove existing integers (details in Q2).

Q2). Draw a valid AVL Tree and nominate a vertex to be deleted such that if that vertex is deleted:

a). No rotation happens

b). Exactly one of the four rotation cases happens

c). Exactly two of the four rotation cases happens (you can**not** use the sample given in VisuAlgo which is `https://visualgo.net/en/bst?mode=AVL&create=8,6,16,3,7,13,19,2,11,15,18,10`, delete vertex 7; think of your own test case)

Extra BST Operations

Q3). There are two important BST operations: Select and Rank that are not included in VisuAlgo yet (overview at `https://visualgo.net/en/bst?slide=5-1`) but can be quite useful for some **order-statistics** problems. Please discuss on how to implement these two operations efficiently.

Binary Heap... or Not?

Q4). We know that Binary (Max) Heap can be used as Priority Queue and can do `ExtractMax()` in $O(\log n)$ time. What modifications/additions/alterations are required so that *both* `ExtractMax()` and `ExtractMin()` can be done in $O(\log n)$ time for the set of n elements and every other Priority Queue related-operations, especially Insert/Enqueue retains the same $O(\log n)$ running time? Hint: What is the topic of this tutorial?

Q5). Quick follow up from Q4). above:
Now revisit Q3). of Tut05. Would you answer that question differently?

Hash Table or Balanced BST?

Q6). As of now, you have been exposed with both possible implementations of Table ADT: Hash Table (and its variations) and BST (including Balanced BST like AVL Tree). Now write down four potential usage scenarios of Table ADT. Two scenarios should favor the usage of Hash Table whereas for the other two scenarios, using Balanced BST is better.