CS5050 Advanced Algorithms

Spring Semester, 2018

Assignment 4: Data Structure Design

Due Date: 3:00 p.m., Tuesday, Mar. 20, 2018 (**at the beginning of CS5050 class**)

1. **(20 points)** Suppose we have a min-heap with n distinct keys that are stored in an array $A[1 \dots n]$ (a min-heap is one that stores the smallest key at its root). Given a value x and an integer k with $1 \leq k \leq n$, design an algorithm to determine whether the k-th smallest key in the heap is smaller than x (so your answer should be "yes" or "no"). The running time of your algorithm should be $O(k)$, independent of the size of the heap.

   Remark. If we were to find the k-th smallest key of the heap, denoted by y, then the best way would be to perform k times *deleteMin* operations, which would take $O(k \log n)$ time (or using the selection algorithm, which would take $O(n)$ time). Our above problem, however, is actually a *decision problem*. Namely, you only need to decide whether y is smaller than x, and you do not have to know what the exact value of y is. Hence, the problem is easier and we are able to solve it in a faster way, i.e., $O(k)$ time.

2. **(20 points)** Suppose you are given a binary search tree T of n nodes (as discussed in class, each node v has $v.left$, $v.right$, and $v.key$). We assume that no two keys in T are equal. Given a value x, the *rank* operation $rank(x)$ is to return the *rank* of x in T, which is defined to be one plus the number of keys of T smaller than x. For example, if T has 3 keys smaller than x, then $rank(x) = 4$. Note that x may or may not be a key in T. Refer to Figure 1 for more examples.

   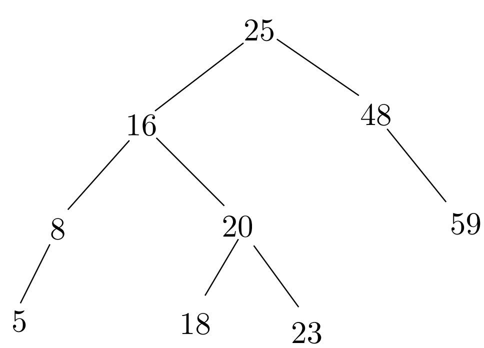

   Figure 1: $rank(16) = 3$, $rank(21) = 6$, $rank(25) = 7$, $rank(26) = 8$.

   Let h be the height of T. We know that T can support the ordinary *search, insert*, and *delete* operations, each in $O(h)$ time. You are asked to augment T, such that the *rank* operation, as well as the normal *search, insert*, and *delete* operations, all take $O(h)$ time each.

   Please explain clearly how you augment T and give your algorithm for performing the *rank* operations (please give the pseudocode). You do not need to give the details for other operations (search, insert, delete), but only need to briefly explain why they still take $O(h)$ time after you augment T.

3. **(20 points)** This problem is concerned with **range queries** (we have discussed a similar problem in class) on a binary search tree T whose keys are real numbers (no two keys in T are equal). Let h denote the height of T. The range query is a generalization of the ordinary *search* operation. The **range** of a range query on T is defined by a pair $[x_l, x_r]$, where x_l and x_r are real numbers and $x_l \leq x_r$. Note that x_l and x_r may not be the keys in T.

   You already know that T can support the ordinary *search, insert*, and *delete* operations, each in $O(h)$ time. You are asked to design an algorithm to efficiently perform the *range queries*. That is, in each range query, you are given a range $[x_l, x_r]$, and your algorithm should report all keys x stored in T such that $x_l \leq x \leq x_r$. Your algorithm should run in $O(h + k)$ time, where k is the number of keys of T in the range $[x_l, x_r]$. In addition, it is required that all keys in $[x_l, x_r]$ be reported in a *sorted order*. Please give the pseudocode for your algorithm.

   Remark. Such an algorithm of $O(h + k)$ time is an *output-sensitive* algorithm because the running time (i.e., $O(h + k)$) is also a function of the output size k. As an application of the range queries, suppose the keys of T are the student scores of an exam. A range query like $[70, 80]$ would report all scores in the range in sorted order.

4. **(20 points)** Consider one more operation on the above binary search tree T in Problem 3: *range-sum*(x_l, x_r). Given any range $[x_l, x_r]$ with $x_l \leq x_r$, the operation *range-sum*(x_l, x_r) reports the *sum* of the keys in T that are in the range $[x_l, x_r]$.

   You are asked to augment the binary search tree T, such that the *range-sum*(x_l, x_r) operations, as well as the normal *search, insert*, and *delete* operations, all take $O(h)$ time each, where h is the height of T.

   You must present: (1) the design of your data structure (i.e., how you augment T); (2) the algorithm for implementing the *range-sum*(x_l, x_r) operation (please give the pseudocode).

Total Points: 80