CS240: Data Structures
Winter Semester – Academic Year 2023-24
Panagiota Fatourou

3rd Set of Exercises

Submission Deadline: Monday, December 4, 2023
Submission Method: The exercises can be delivered until Monday, December 4, 2023, **14:59**, to the teaching assistant who will have office hours on the day of the submission to receive the exercises. Exercises that are delivered after **14:59** on Monday, 4/12/2023, are considered overdue. Late assignments are accepted only in electronic form and must be sent using the "turnin" program. The electronically delivered exercises should be in pdf format, be legible and include on all pages the full name and number of the student to whom they belong.

Exercise 1 [20 Points]

1. **Binary Tree Completeness Check Algorithm [10P]**

   Present pseudocode for a recursive algorithm that will check whether a binary tree is complete. What is the time complexity of your algorithm and what is the spatial overhead it imposes?

2. **Algorithm in a (not necessarily binary) ordered tree [10P]**

   Present an algorithm that takes as an argument a pointer to the root of a binary tree that implements a (non-necessarily binary) ordered tree and an integer k. The algorithm should check whether the kth child (if any) of each node in the non-dual ordered tree has an even key.

   After your algorithm has finished running, the binary tree representing the ordered tree should remain unchanged.

   a. Provide pseudocode for your algorithm.

   b. What is the time complexity of your algorithm and what is the spatial overhead it imposes?

Exercise 2 [20 Points]

Consider an ordered binary tree T (i.e., a binary search tree) of height h. An algorithm is required that creates a new tree T' containing the same keys as T and having height O(log n), where n is the number of nodes in T. The original tree remains unchanged during the execution of the algorithm. What is the time complexity of the algorithm of your algorithm?

Hint: Place the elements of T in a table, which will be sorted in ascending order, in Time O(n). Based on this table, create the tree T' which will have height O(log n).

Exercise 3 [20 Points]

Let the extension of the abstract data type Dictionary support in addition to Insert(), Delete() and Search(), the function Split(D, x), which performs a split of the dictionary D into two new dictionaries D1 and D2 so that the former consists of keys less than or equal to x and the latter has keys greater than x, where x is some key that may or may not belong to D. Consider that the Dictionary is implemented with ordered binary trees. Present an algorithm which implements split(D, x) (i.e. an algorithm that takes as arguments a pointer to the root of the sorted tree representing the dictionary D and the key x and returns two pointers to the roots of the sorted trees representing dictionaries D1 and D2). The time complexity of of your algorithm should be O(h), where h is the height of T and the spatial overhead should be O(1).

Exercise 4 [40 Points]

Leaf-oriented binary search trees are an alternative implementation of the abstract dictionary data type. They are defined as follows: (a) all dictionary keys are stored in the leaves of the tree, from left to right by non-decreasing key value, and (b) the internal nodes store keys (not necessarily corresponding to dictionary keys), such that the following invariant condition holds at each node v:

The key of the left child of v is less than or equal to that of v,

while the right child of v has a key greater than that of v

Notice that by definition, the interior nodes have both indexes non-empty, while both leaf indexes are empty. An example of a leaf-oriented tree is shown in Figure 1.

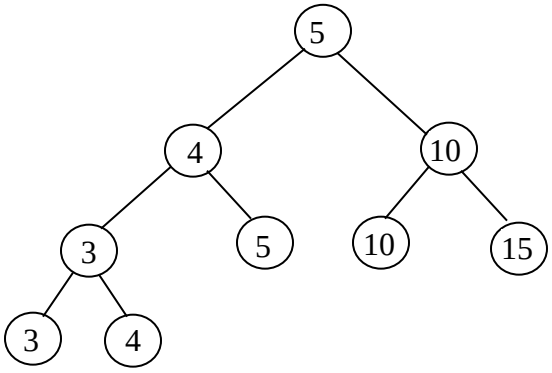

Figure 1

a. Present insertion, deletion and search algorithms in leaf-oriented binary search trees. Your algorithms should have time complexity O(h), where h is the height of the tree and spatial overhead O(1). [Insert: 10P, Delete: 10P, Search: 5P]

b. Describe an algorithm which, given a leaf v in a doubly-linked leaf-oriented tree, will find the leaf containing the next highest key from that of v among the keys of all leaves in the tree. Your algorithm should have time complexity O(h), where h is the height of the tree and spatial overhead O(1). [15P]

Clarifications on the functions of Insertion and Deletion:

1. To insert a new node, v, with key K into a leaf-oriented binary search tree, we search to find the leaf, v', which should be the parent node of v in the tree. However, the key, K', of v' must still appear in a leaf of the tree. To achieve this, we replace v' by a three-node tree, which consists of an interior node with two children, both of which are leaves. The left of these two leaves as well as v have key min{K, K'}, while the right leaf has key max{K, K'}. For example, the tree in Figure 2 is the tree that results after inserting key 1 into the tree in Figure 1.

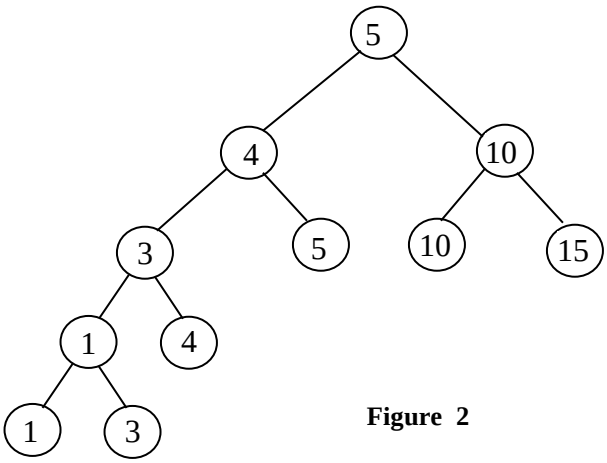

Figure 2

2. To delete a node, v, from a leaf-oriented binary search tree, I find its parent node, v', as well as the parent node of v', let v''. For the deletion, I replace the pointer of v'' pointing to v' so that it points to the sibling node of v. For example, the tree in Figure 3 is the tree resulting after deleting key 10 from the tree in Figure 1.

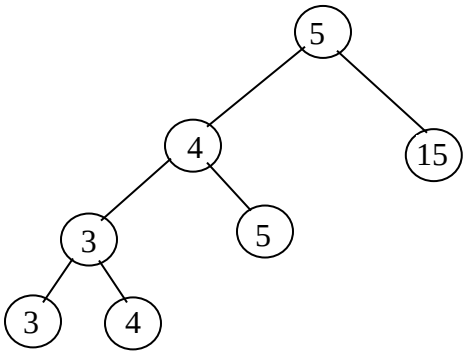

Figure 3