CS240: Data Structures
Winter Semester – Academic Year 2017-18
Panagiota Fatourou

3rd Set of Theoretical Exercises

Submission deadline: Monday, 4 December 2017

How to submit: The assignments must be submitted either in electronic format using the turnin program (see instructions on the website of the course), or to the teaching assistants of the course in the laboratory of postgraduates, on Monday, 4 December2017, time 15:00-16:00. Assignments submitted after 16:00 of Monday, 4/12/2017 are considered out of date. Out of date assignments are accepted only in electronic format using the turnin program.

Exercise 1 [40 Points]

a. Consider a simple-linked binary tree T (without guard node). Present pseudocode for a function that will take as argument a pointer to the root of T and it will creates a copy of T. The only difference of the T-copy from the T itself is that the copy must be a tree with a node guard. The copy (as well as the pointers to the guard node) should be created with a single trace of T. [20%]

b. Consider a binary tree T that implements a (not necessarily binary) ordered tree D for which the following are applying: the key of each node is different from the keys of all the other nodes of the tree (that is, the nodes' keys are unique). Present pseudocode for a recursive function that will take as arguments a pointer to the root of T and a pointer to a node v of T and it will return the depth of v to the ordered tree D (note, this depth is not necessarily the same with the depth of v in the binary tree T). [20%]

Note: In the example of slide 20 in Section 3, the depth that the algorithm should print e.g. for key G is 2 that is its depth in tree (A) and not 4 that is its depth in tree (B). The function is allowed to have additional arguments.

Exercise 2 [30 Points]

Consider two sorted binary trees (i.e two binary search trees), let T_1 and T_2. An algorithm is required to merge that two trees into one at time $O(n_1 + n_2)$, where n_1 and n_2 are the number of nodes of T_1 and T_2, respectively. The final tree should have a height $O(\log (n_1+n_2))$.

Suggestion: Place the elements of T_1 on an array, which will be sorted in ascending order, at time $O(n_1)$. Place the elements of T2 on a second array, which will also be sorted in ascending order, at time $O(n_2)$. Merge the elements of the two arrays on a third array that will also be sorted in ascending order. The merge should run at time $O(n_1+n_2)$. Base on the third array, make the tree T containing the elements of T_1 and those of T_2. T should have a height $O(\log(n_1+n_2))$ and its creation should occur at time $O(n_1+n_2)$.

Exercise 3 [40 Points]

a. Consider a orderd, double-linked binary tree T (i.e T is a binary search tree). Consider also a single-linked list L that is sorted in ascending order. Let **the set of keys of L being a subset of all T_1 keys**. Present pseudocode for a function which for each node v of L will delete from T_1 the node that has the same key as v. The total time for all deletions from T_1 to be made should be O(n), where n is the number of nodes of T_1. [20%]

Note: You have to select the appropriate trace for T1 so that you can properly perform all node deletions with a single trace of T1 (as well as a single trace of L).

b. Consider a sorted binary tree T (i.e a binary search tree). Suppose that each node v of T contains a cnt field which is an integer that expresses the number of nodes present in the left sub-tree of v.

Present pseudocode for a function that will take as an argument a pointer to the root of T and a array B of bits and it will do the following. It will examine the bits in sequence and, starting from the root of T, whenever the current bit is 0, it will follow the lc pointer of the current node in the tree, and if the bit is 1, it will follow the rc pointer of the node. The algorithm will be terminated when it will reach the last node, let v, of the path defined by the bit sequence of B. The function should return how many nodes contain smaller keys than the key of node v. The algorithm should run at time O (h), where h is the height of T. [20%]