Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Markus Püschel, David Steurer

12. November 2018

Algorithms & Data Structures — Homework 8 — HS 18

Exercise Class (Room & TA): _____

Submitted by: _____

Peer Feedback by: _____

Points: _____

Exercise 8.1 *Search Trees.*

1. Draw the resulting tree when the keys 1,3,7,4,5,8,6,2 in this order are inserted into an initially empty natural search tree.
2. Delete key 4 in the above tree, and afterwards key 7 in the resulting tree.
3. Draw the resulting tree when the keys are inserted into an initially empty AVL tree.
4. Delete key 7 in the above tree, and afterwards key 8 in the resulting tree.

Exercise 8.2 *Tree Traversals.*

There are three essential ways to traverse binary trees. The first one is Preorder(T), which at first visits the root v, then $T_l(v)$ and then $T_r(v)$, where $T_l(v)$ is the left subtree of v and $T_r(v)$ is the right subtree of v. The second one is Postorder(T), which at first visits $T_l(v)$, then $T_r(v)$ and then v. The third one is Inorder(T), which at first visits $T_l(v)$, then v and then $T_r(v)$.

In each case the left and right subtrees are visited recursively in the same order.

1. Consider this pseudocode for the Preorder procedure:

Algorithm 1: Preorder(T)

```
1 if T is non-empty then
2 |   v ← Root(T);
3 |   Visit(v);
4 |   Preorder(T_l(v));
5 |   Preorder(T_r(v));
6 end
```

   Write pseudocodes for Postorder and Inorder procedures.

2. For the above search trees in 8.1.1 and 8.1.3 give the Preorder, the Postorder, the Inorder of the nodes.
3. Draw the binary tree with keys $1, 2, 3, 4, 5, 6, 7, 8$ such that the Preorder starts with $3, 1, 2, 7$ and the Postorder ends with $5, 8, 7, 3$.

Exercise 8.3 *Advanced Search Trees **(1 Point)**.*

In this exercise, we wish to extend the functionality of a search tree. We consider a binary search tree over integers. In addition to finding a number, we want to be able to answer the following questions:

1. How many elements in the tree are *multiples of 3* and greater than a given number k?
2. How many elements in the tree are *multiples of 3* and (strictly) between two given numbers k_1 and k_2, with $k_1 < k_2$?

Discuss how you can modify the tree so that you can answer these questions efficiently. Describe how the insertion and the removal operation must be changed accordingly. Include a discussion of the running times of the modified algorithms.

Exercise 8.4 *Maximum Depth Difference of two Leaves.*

Consider an AVL tree of height h. What is the maximum possible difference of the depths of two leaves? Imaging which structure such trees need to have, and draw examples of corresponding trees for every $h \in \{2, 3, 4\}$. Derive a recursive formula (depending on h), solve it and use induction to prove the correctness of your solution. Provide a detailed explanation of your considerations.

Note: For the proof the principle of *complete induction* can be used. Let $\mathcal{A}(n)$ be a statement for a number $n \in \mathbb{N}$. If, for every $n \in \mathbb{N}$, the validity of *all* statements $\mathcal{A}(m)$ for $m \in \{1, \ldots, n-1\}$ implies the validity of $\mathcal{A}(n)$, then $\mathcal{A}(n)$ is true for every $n \in \mathbb{N}$.

$$\Big(\forall n \in \mathbb{N} : \big(\forall m \in \{1, \ldots, n-1\} : \mathcal{A}(m)\big) \Rightarrow \mathcal{A}(n)\Big) \Rightarrow \forall n \in \mathbb{N} : \mathcal{A}(n). \tag{1}$$

Thus, complete induction allows multiple base cases and inductive hypotheses.

Exercise 8.5 *One-Column Candy Crush **(2 Points.)**.*

Consider a column of n candies, and assume that if three or more *adjacent* candies in this column are equal, then these candies can be removed, and the column shrinks. Removing the candies is done with the following tie breaking rule: The candies are removed from top to bottom, i.e., when there is more than one group that can be removed at the same time, the upper group is removed first. We are interested in the number of candies that can not be removed.

You get this column of candies stored on a stack S, such that the first (topmost) candy of the column is on the top of this stack. Recall that the operations on a stack are `top()`, `pop()`, and `push(`v`)` where v is a candy. Moreover, you can access the number of candies in the stack. Assume that all these operations require constant time.

Your task is to design a linear time algorithm that returns the number of the remaining candies. Your algorithm is allowed to use stack S plus one additional stack T, for which you can assume that it is initially empty. On top of that, you are allowed to use only $O(1)$ extra memory space. Provide an analysis of the actual running time of your algorithm.

Submission: On Monday, 19.11.2018, hand in your solution to your TA *before* the exercise class starts.