Prof. Dr. Thomas Schultz
Olivier Morelle (morelle@uni-bonn.de)

B-IT
Bonn-Aachen International
Center for Information Technology

December 02, 2024

Winter term 2024/25

Computer Science for Life Scientists

Assignment Sheet 9

Solution has to be uploaded by December 11, 2024, 10:00 a.m., via eCampus

- This exercise can be submitted in **small groups** of 2-3 students. Submit each solution only once, but clearly indicate who contributed to it by forming a team in eCampus. Remember that all team members have to be able to explain all answers.
- Remember to include proper **documentation** in all your code, in particular, docstrings for functions.
- Please submit your answers as a single file in .ipynb or .pdf format.

If you have questions concerning the exercises, please use the forum on eCampus.

Exercise 1 (Doubly Linked Lists, *3 Points*)

You have learned in the lecture that a node in a doubly linked list stores pointers to its previous and next elements in the list. In addition, we store `head` and `tail` pointers to the first and last node, respectively. In the provided Notebook file, complete the *insert_item* function to insert a new item at index position `i` in the doubly linked list. Make sure your function works correctly for any value i. Your code should go into the part that is marked by ToDo as comment.

Exercise 2 (Implementing a Binary Search Tree, *11 Points*)

In this exercise, you will create, edit, and print a Binary Search Tree (BST).

a) Define a TreeNode class that has all data attributes that are required for a node in a binary search tree. (1P)

b) Define a BinaryTree class with all required data attributes (1P), as well as with the following methods:
   - *insert(n)*: to insert a node with integer value n into the tree. (1P)
   - *delete(n)*: to delete the node with value n from the tree. (3P)
   - *search(n)*: to check if n is an element in the tree. (1P)

c) Trees in general can be traversed in different ways. Add a *traverse(order)* method to your class that recursively traverses the tree as specified by *order* and prints node values accordingly. Your method must support the following orders:
   - *preorder*: First print the value of the *root*, then the *leftChild* and then the *rightChild* (both recursively). (1P)
   - *inorder*: First print the value of the *leftChild*, then *root* and then the *rightChild* (again, both recursively). (1P)

- *postorder*: First print the value of the *leftChild*, then the *rightChild* (again, both recursively) and then the *root*. (1P)

d) Generate a tree and insert values $\{10, 5, 7, 1, 15, 3, 6, 9, 8, 11\}$ into it. Use a suitable variant of *traverse* to output the values in sorted order. Delete the node with value 7 and output the tree again, to double-check that deletion worked correctly. (1P)

Exercise 3 (AVL Trees and Heaps, *4 Points*)

Perform the following tasks manually and submit a sketch showing every step. You can make the figures on your computer or on paper. Make sure to include all figures in your final notebook.

- Delete the items 6 and 13 from the AVL tree given in Figure 1. Does it make a difference in which order you remove them? (2P)
- Given the integer input array $[6, 10, 3, 2, 7, 11]$, convert it into an array representation of a binary max heap, as one would do it in the initial phase of the heap sort algorithm. *Hint:* You might find it helpful to sketch the corresponding binary tree. (2P)

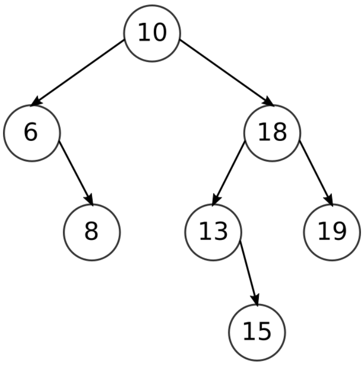

Figure 1: AVL Tree for the exercise above.

Exercise 4 (Priority Queues, *7 Points*)

In the lecture, heaps were presented as a possible implementation for priority queues. Another option is to use AVL trees.

a) Assume that the operations *insert*, *delete*, *search* for an AVL tree are implemented. How would you implement the operations *insert*, *maximum*, *extract-max*, *increase-key*, *decrease-key* that a priority queue should support? You only need to provide the pseudocode. (4P)

b) What are the asymptotic running times for these operations? How do they compare to those with a heap implementation of priority queues? (3P)

Note: In this exercise, you do not have to worry about complications that might arise from multiple items with identical priorities. If you wish, you can assume that all priorities are different.