Computer Science B63
University of Toronto Scarborough

Homework Exercise #1

Handing in and marking

For this exercise, you need to submit your solutions to the pencil-and-paper exercises on crowdmark and your solutions to the programming question on MarkUs. Your pencil-and-paper solutions will be marked with respect to correctness, clarity, brevity, and readability. Your code will be marked with respect to correctness, efficiency, program design and coding style, clarity, and readability. This exercise counts for 10% of the course grade.

Question 1. Asymptotic Bounds Basics [10 MARKS]

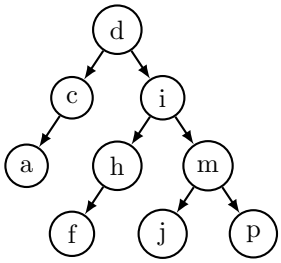
Prove each of the following *using the definitions of* Big-Oh / Big-Omega / Big-Theta.

- If $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(h)$ then $f \in \mathcal{O}(h)$, for all functions f, g, h in $\mathbb{N} \to \mathbb{R}^+$.
- If $f \in \Omega(g)$ and $g \in \Omega(h)$ then $f \in \Omega(h)$, for all functions f, g, h in $\mathbb{N} \to \mathbb{R}^+$.
- $\log_\phi(\sqrt{5}(n+2)) - 2 \in \mathcal{O}(\log_2 n)$ where ϕ is the golden ratio.

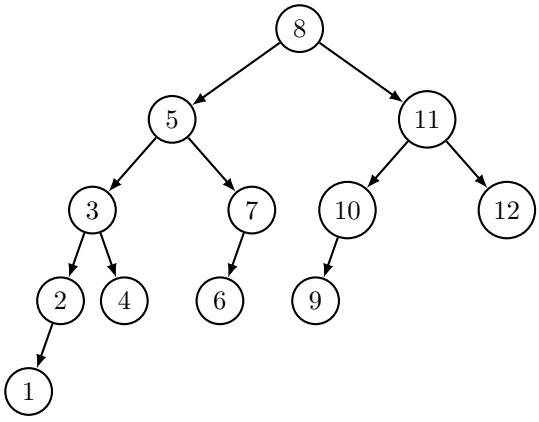
Question 2. AVL Trees Basic Operations [20 MARKS]

Show the AVL trees resulting from performing the requested sequence of operations. Whenever a rotation is required, indicate the type of rotation used and what node it applies to. Whenever a double rotation is required, show the intermediate step (i.e., show the result after each of the single rotations).

- On an initially empty tree, show each step of inserting the keys `9, 10, 12, 14, 3, 34, 19, 37, 20`.
- On the tree shown below, show each step of deleting the keys `p`, `d`, `h`.



- Carefully consider the tree below.



– If we start with an empty AVL tree, what sequence of insertions would result in this tree?

– Show each step of deleting key `11`.

Question 3. Implementing AVL Trees [50 MARKS]

We provided you with the starter code for implementing AVL Trees. Your task is to implement the functions declared in `AVL_tree.h` that are not already implemented in `AVL_tree.c`. **Each function must be implemented in $\mathcal{O}(\log n)$ running time**, as we learned in class. Note that `AVL_tree.c` is the only file you will submit, so make sure your implementation works with the original provided `AVL_tree.h`. Make sure to **carefully study the starter code** before you begin to add your own.

The marking scheme for this question is as follows:

- Correctness:
  - `AVL_Node* search(AVL_Node* node, int key)`: 5 marks
  - `AVL_Node* insert(AVL_Node* node, int key, void* value)`: 10 marks
  - `AVL_Node* delete(AVL_Node* node, int key)`: 20 marks
- Program design (modular implementation, self-explanatory code, clear logic, no repeated code, no unnecessary code): 10 marks
- Readability and coding style (good use of white space, good naming, etc.): 5 marks
  - You are encouraged (but not required) to use a `linter` to help check and/or auto-format your code.

Question 4. Augmenting AVL Trees [20 MARKS]

Consider an ADT consisting of a set S of distinct integers and the following operations:

- `search(S, x)`: Return true if x is in S and false otherwise.
- `insert(S, x)`: Insert the element x into the set S. This operation has no effect if x is already in S.
- `delete(S, x)`: Delete the element x from the set S. This operation has no effect if x is not in S.
- `min_difference(S)`: Given a set S with size of at least 2, return a pair of distinct integers (x, y), with $x \in S, y \in S$, with the minimum absolute difference, i.e. $\forall x', \forall y', (x' \neq y' \rightarrow |x - y| \leq |x' - y'|)$

Your task is to design a data structure to implement this ADT, such that all operations are performed in $\mathcal{O}(\log n)$ time, where $n = |S|$. You will do so by augmenting our familiar AVL tree.

1. Describe all information that will be stored in the nodes.
2. Provide pseudo-code for each required operation.
3. Justify why your algorithms are correct and why they achieve the required time bound.