CSC265 F18: Assignment 3

Due: October 24, by midnight

Guidelines: (read fully!!)

- Your assignment solution must be submitted as a *typed* PDF document. Scanned handwritten solutions, solutions in any other format, or unreadable solutions will **not** be accepted or marked. You are encouraged to learn the LaTeX typesetting system and use it to type your solution. See the course website for LaTeX resources. Solutions typed using LaTeX receive 2 bonus marks.
- Your submission should be no more than 8 pages long, in a single-column US Letter or A4 page format, using at least 9 pt font and 1 inch margins.
- To submit this assignment, use the MarkUs system, at URL `https://markus.teach.cs.toronto.edu/csc265-2018-09`
- This is a *group assignment.* This means that you can work on this assignment with *at most one other* student. You are *strongly encouraged* to work with a partner. Both partners in the group should work on and arrive at the solution together. Both partners receive the same mark on this assignment.
- Work on all problems together. For each problem, one of you should write the solution, and one should proof-read and revise it. The first page of your submission must list the *name*, *student ID*, and *UTOR email address* of both group members. It should also list, for each problem, which group member wrote the problem, and which group member proof-read and revised it.
- You **may not** consult any other resources except: your partner; your class notes; your textbook and assigned readings. *Consulting any other resource, or collaborating with students other than your group partner, is a violation of the academic integrity policy!*
- You may use any data structure, algorithm, or theorem previously studied in class, or in one of the prerequisites of this course, by just referring to it, and without describing it. This includes any data structure, algorithm, or theorem we covered in lecture, in a tutorial, or in any of the assigned readings. Be sure to give a *precise reference* for the data structure/algorithm/result you are using.
- Unless stated otherwise, you should justify all your answers using rigorous arguments. Your solution will be marked based both on its completeness and correctness, and also on the clarity and precision of your explanation.

Question 1. (18 marks)

The Dominating-Points ADT supports a set P of n points in $\mathbb{Z}^2$ (i.e. points in the plane with integer coordinates) under the following operations:

- Insert(p) inserts a new point p into P, where p is specifed by its x- and y-coordinates $p.x$ and $p.y$.
- Dominating(q) returns the coordinates of a point $p \in P$ which is above and to the right of the point q, if such exists. Assume q is specifed by its x- and y-coordinates $q.x$ and $q.y$. Then the operation must return $p.x$ and $p.y$ for some $p \in P$ such that $p.x \geq q.x$ and $p.y \geq q.y$, if such exists, or return Fail otherwise.

For this question, you can assume that at any point in time there are no two points in P with the same x or y coordinate. Your goal will be to implement both operations in worst-case running time $O(\log n)$.

Part a.

Describe an augmented AVL tree which allows you to implement Insert and Dominating in the required worst-case time complexities. Specify what keys and auxiliary fields you are using for your AVL tree.

Part b.

Describe an algorithm implementing the procedure Dominating(q) in clear and precise English, and, optionally, using pseudocode. Justify why the procedure correctly returns a point $p \in P$ dominating q, and why it runs in time $O(\log n)$.

Part c.

Describe an algorithm implementing the procedure Insert(p) in clear and precise English. Justify why the procedure runs in time $O(\log n)$ and preserves the properties of the data structure, as you described them in the first subproblem.

Question 2. (15 marks)

Let T be a complete ternary tree, i.e. a tree in which every node, except the leaves, has exactly three children: a left, a middle, and a right child. Then T has 3^h leaves, where h is its height. Suppose that each leaf of T is given a value 0 or 1. Then we determine values for the internal nodes of T as follows: the value of an internal node u of T equals 0 if at least two of its children have value 0, and its value equals 1 otherwise.

Assume that the values of the leaves, numbered from left to right, are given in an array $A[1\,..\,3^h]$. Using an adversary argument, show that every deterministic algorithm which computes the value of the root of T must query *every* entry of A in the worst case, and, therefore, has worst-case complexity $\Omega(3^h)$.

Describe the adversary's strategy clearly and precisely. Justify why the strategy forces any algorithm which correctly determines the value of the root of T to query every entry of A.