Assignment 4

This assignment is due **at 10:00pm** on **Thursday, 9th November, 2023**.

This assignment is primarily coding-based. Please follow the usual instructions to submit on GitHub Classroom and Gradescope.

- For Question 1c, fill in your function calls in main.c replacing "your code here". You shouldn't need to make any changes in main for 1a or 1b. Note that you should edit q1.h to enter your preferred struct definition.
- For Question 2b, fill in your function calls in main.c replacing "your code here". You shouldn't need to make any changes in main for 2a.
- Please consult the header files for the functions expected of you.
- Please remove ALL .o files from the repo before running make. This includes .o files in the helpers and random generator directory.
- Accept the assignment on GitHub using this link. Submit your solutions using this template.
- This is a short and easy pset, don't stress!

Problems

Problem 4-1. Caves of Steel [40 points]

(a) [10 points] In Mumbai 2060, every line of the local trains has two tracks in parallel: fast and slow. The slow trains travel along the route stopping at every station on the way. The fast trains travel the same route but only stop at every n^{th} station where n is a term in the Fibonacci sequence (For example, the first 6 stops of the fast train will be the 0^{th}, 1^{st}, 2^{nd}, 3^{rd}, 5^{th}, and 8^{th} stations on the line.)
The railway corporation has asked you to implement a data structure in C describing this model of train lines and stations. Each station should be represented as (value, pointers) where value is the numeric identifier of the station and pointers is an array of pointers to the next station(s). Print the stops of each train.

(b) [10 points] The railway corporation has realized that having the fast trains stop along Fibonacci stops makes no sense, and want to transition to a different system. They want to lay down m lines, such that the 0^{th} line stops at every station, the 1^{st} line stops at every other station, and so on. Re-implement your structure using this system. Print the stops of every train.

(c) [10 points] In fact, the Fibonacci indexed stations have so little traffic that they will soon be demolished. Update your structure from (b) to delete those stations, then print the new stops for each line.

(d) [10 points] Write an algorithm (pseudocode is fine) to convert the first structure (of two levels) to the second (of m levels). Count the total number of link reassignments (i.e., the number of times you change a value in node $->$ pointers to point to a different node). Argue the time complexity.

Problem 4-2. Runaround [40 points]

(a) [20 points] You are given a set of 15 integers (randomly generated using the seed). Write a C program to accept them into an AVL tree. At every stage, print the tree (in the same format as before, Inorder: w x y z), and each rotation you perform to keep the tree balanced ("L" for left rotation, "R" for right rotation). Note that your tree doesn't need to handle duplicates (if a number occurs twice, you only need to include it in the tree once).

(b) [10 points] Ensure you have the final tree. You are now required to print and extract the C^{th}, $C+1^{th}$, and $C+2^{th}$ smallest elements from the tree (recall a similar question from the mid-semester exam). After every extraction, print the element and the tree, and each rotation you perform to keep the tree balanced.

(c) [10 points] Provide a sequence of 15 integers that maximizes the number of rotations required by the AVL tree during the insertion process. What is the worst-case time complexity of insertion in an AVL tree? Argue why your sequence meets this bound.

Problem 4-3. The Last Question [20 points]

In class, we've spoken about B-trees and B+ trees.

(a) [5 points] Using the 15 integer values from the autograder for question 2, draw a B tree and a B+ tree.

(b) [5 points] Argue the worst-case time complexity for insertion and search in a B tree and a B+ tree.

(c) [10 points] For both the B tree and B+ tree, indicate the steps required to construct the tree, including balancing, after every insertion.