Written Assignment #3

Due Date: Sunday, Apr 05 @ 11:59PM
Total Points: 60

Instructions

Answer all questions in this written assignment, showing work when required. All work must be your own, created solely by you and using only the allowed resources for the course (as stated in the syllabus). Your solutions may either be handwritten and scanned, or typeset. Submit your work as a PDF via AutoLab to the WA3 submission target.

You may submit as many times as you like, but **only your last submission will be graded** and should include the entire submission. You should view your submission after you upload it to make sure that it is not corrupted or malformed. Submissions that are rotated, upside down, or that do not load will not receive credit. Illegible or incomplete submissions will lose credit depending on what can be read. Ensure that your final submission contains all pages.

You are responsible for making sure your submission went through successfully.

Written submissions may be turned in up to one day late for a 50% penalty.

No grace day usage is allowed.

Part 1: Graph Data Structures

For PA2, the StreetGraph class used to store the maps we were searching was implemented using an `EdgeList` data structure. The first function you had to implement created an external `AdjacencyList` that you could use for searching your graph.

Question 1 [*8 points*]: Derive the unqualified worst-case runtime (in terms of $|V|$ and $|E|$) to perform a BFS search on a Graph that is implemented using an `EdgeList`.
Justify your answer. Answers without justification will not receive credit.

Note: You may consult lecture materials on BFS and Graphs to help justify your answer.

Question 2 [*8 points*]: Derive the unqualified worst-case runtime (in terms of $|V|$ and $|E|$) to create an `AdjacencyList` representation of a graph from an `EdgeList` representation.

For this implementation you may assume that each `Vertex` object can hold a reference to a `List<Edge>` object, and that your algorithm must populate that list with ALL edges that are incident to the vertex (not just the outgoing edges).

You may assume that the `Edge` objects hold a reference to their origin and destination vertices just as they did in PA2.

Justify your answer. Answers without justification will not receive credit.

Question 3 [*4 points*]: Derive the unqualified worst-case runtime (in terms of $|V|$ and $|E|$) to convert a graph represented as `EdgeList` to one represented as an `AdjacencyList` AND THEN perform a BFS search on the `AdjacencyList` representation of that graph.

Given that runtime, if you have a graph that is implemented as an `EdgeList` that you want to perform BFS on, is it asymptotically faster to just directly do the search using the `EdgeList`, or should you convert to an `AdjacencyList` and then search the `AdjacencyList`?

This question requires two answers: a runtime, and whether or not you should convert an `EdgeList` before searching it. You must justify both of your answers. For the justification, you may refer to your answers to questions 1 and 2 as well as material discussed in class. Answers without justification will not receive credit.

Part 2: Binary Trees

Question 4 [*5 points*]: Draw a valid min heap (as a tree) containing the following values:

$$\{0,10,20,30,40,50,60,70,80\}$$

Question 5 [*4 points*]: Write the array representation of the heap **you drew** in Question 4

Question 6 [*5 points*]: Draw a valid BST **with a height of 4** containing the following values:

$$\{0,10,20,30,40,50,60,70,80\}$$

Question 7 [*4 points*]: Answer the following about the tree **you drew** in Question 6:

a) What value, if inserted into your tree, would increase its height?
b) What value, if inserted into your tree, would not change the height?

Question 8 [*2 points*]: An AVL tree is a BST where **every** node has the following property:

The height of its left child and the height of its right child differ by at most 1.

Note: By convention, the height of the empty tree node is −1.

Does the tree you drew in Question 6 meet the description of an AVL tree?
If your answer is no, state one node that does not meet the AVL property described above.

Part 3: Induction

Consider the following statement:

$$P(h) : \text{A complete binary tree of height } h \text{ has at least } 2^h \text{ nodes.}$$

For this part we will prove, using induction on h, that $P(h)$ is true for all $h \geq 0$.

Note: Heaps are complete binary trees. If we can prove $P(h)$ is true for all $h \geq 0$, then we'll have shown that for heaps, $n \geq 2^h$. Taking the log of both sides shows that $h \leq \log(n)$ which was a key property in determining the runtime of heap operations!

Question 9 [*5 points*]: Prove that $P(h)$ is true for an appropriate base case.

Question 10 [*5 points*]: State an appropriate inductive assumption.

Question 11 [*10 points*]: Prove the inductive step based on your given assumption.

Note: To complete this part of the proof, it will be useful to know that the number of nodes in a **perfect binary tree** of height h is exactly $2^{h+1} - 1$. You can use this fact as a given, and do not need to prove it (although it is fairly straightforward to prove by induction).

Hint: Just like in lecture 27, it will be helpful to figure out the possible ways you can have a complete binary tree of height $k + 1$. What do the possible children of a compelete binary tree look like?

Summation Rules

(S1) $\sum_{i=j}^{k} c = (k-j+1)c$

(S2) $\sum_{i=j}^{k} (cf(i)) = c\sum_{i=j}^{k} f(i)$

(S3) $\sum_{i=j}^{k} (f(i)+g(i)) = \sum_{i=j}^{k} f(i) + \sum_{i=j}^{k} g(i)$

(S4) $\sum_{i=j}^{k} f(i) = \sum_{i=l}^{k} f(i) - \sum_{i=l}^{j-1} f(i)$ for any $l < j$

(S5) $\sum_{i=j}^{k} f(i) = f(j) + f(j+1) + ... + f(k-1) + f(k)$

(S6) $\sum_{i=j}^{k} f(i) = f(j) + ... + f(l-1) + \sum_{i=l}^{k} f(i)$ for any $j < l \leq k$

(S7) $\sum_{i=j}^{k} f(i) = \left(\sum_{i=j}^{l} f(i)\right) + f(l+1) + ... + f(k)$ for any $j \leq l < k$

(S8) $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$

(S9) $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$

Log Rules

(L1) $\log(n^a) = a\log(n)$

(L2) $\log(an) = \log(a) + \log(n)$

(L3) $\log\left(\frac{n}{a}\right) = \log(n) - \log(a)$

(L4) $\log_b(n) = \frac{\log_c(n)}{\log_c(b)}$

(L5) $\log(2^n) = 2^{\log(n)} = n$