COL 106 - Intro. to Data Structures and Algorithms

Major Exam (AGP only)

Exam Duration: 12:30 PM - 3:30 PM, July 8th 2020

All questions have to be answered.

If you find any ambiguity in any question, write down your assumption clearly and then answer the question.

All submissions have to be completed before 3:30PM IST on 8th July '20. If you complete your exam before that time, send a Whatsapp message with text "COMPLETED" to your assigned TA.

After every 1 hour, you should take a snapshot of your answer script (all pages you have used so far), and send it to your assigned TA by Whatsapp.

Instructions:

A. This is an open-book (any textbook) but ***not an open-Web exam***. You MUST provide the specific source(s) you have referred to arrive at an answer.
B. No collaborations allowed. Strict plagiarism checks will be conducted, and if found guilty, they will be referred to the institute disciplinary committee.

1. Design an ADT named **URLMapADT** and an efficient data structure that implements this ADT for storing and managing 1 Billion URLs. A URL --e.g., *http://www.cse.iitd.ac.in/~srikanta/course/col106-2020/* -- has the following components (a) protocol (`http://`), (b) hostname (`www.cse.iitd.ac.in`) and (c) filename (`~srikanta/course/col106-2020/`).
Two URLs are equal if and only if all their components are exactly equal. Further, we also define the *top-level-domain of a URL* as the last part of its hostname (i.e., the substring after the last "." -- in the above example it will be "in").
Assumptions:
   1. all URLs consist of only English lower-case alpha-numerics and do not have any special characters.
   2. the hostname has no more than 5 parts (i.e., not more than 4 "."s).
   3. filenames can be not more than 5 directories deep.
   4. Only protocols allowed are http://, ftp:// and https://

URLMapADT supports the following functions:

- **getURLwithDom**: Given a domain -- e.g., ".org",".in",".com", etc -- return a list of URLs that have this domain.
- **getURLwithCommonDom**: Given a URL, return URLs that have the same top-level domain.
- **isURLInCollection**: Given a URL, return true or false depending on if it is in the collection or not.

a) Fully define the **URLMapADT** with all its methods and state variables - if any. (3 marks)
b) Develop data structure named **URLMap** to implement all the methods of the URLMapADT in the most efficient manner possible. (6 marks)
c) Using the URLMap data-structure designed above write the pseudocode and derive the complexity for
   i) getURLwithDom (3 marks)
   ii) getURLwithCommonDom (3 marks)
   iii) isURLInCollection (3 marks)

Your solutions will be considered valid if you achieve at least the following efficiency

- **getURLwithDom** and **getURLwithCommonDom** takes time linear in the number of URLs in the collection that match the condition,
- **isURLInCollection** takes constant time (i.e., O(1)).

2. Consider the following algorithm operating on an array A of N numbers.

- i = 0.
- while (i < N)
  - If (i == 0 or A[i] >= A[i-1])
  - then
    - i++
  - else
    - swap A[i] and A[i-1]
    - i--

Does this sort the array? If yes, show how and prove its worst-case complexity (i.e., the total number of comparisons). If no, prove by providing at least one counter example. (4 marks)

3. Draw a **connected graph** (with at least 10 vertices) where all nodes have degree k or 0.
The value of k is determined as follows:

- Take the **last three** digits of **your entry number** and add them up to get an intermediate value X. Then k = (X%7 + 9) mod 10. Further, **if k = 0, then we will add 4 to it** *For example, if your entry number is **"20XXDD10213"** then k = (((2+1+3) % 7) + 9) % 10 = 5.* (4 marks)

4. Bachchan is a gambler, a great gambler. He gambles frequently and sometimes he wins and sometimes also loses. He has stored his winnings (and losses) meticulously in an array **WIN** of size **N** whose elements are (date, win/loss amount). For example, an element of the array could

be (4th June 2020, +20000), and the next element in the array could be (9th June 2020, -1000), and so on. Since this is the record of a great gambler, N is very large and spans many-many years. Array entries are maintained in the increasing order of dates.

Bachchan needs help in building a function named ***WinAverage*** that takes a start date, an end date, and computes the **average** of his winnings **between the start and end dates (including these two dates) on the WIN array**. Note that Bachchan may not have gambled on the dates that are given as arguments to the function.

a) Develop a data-structure and the function to support a large number of calls with different start and end dates in most efficient manner. You can write it as a pseudo-code but have to handle all the special cases.
b) What is the complexity (in terms of N) of the algorithm? For deriving the complexity, you may assume that the start date and end date are always separated by 1 year.

5. Show how Quicksort is efficient in the average case. (4 marks)

6. Consider a sequence of ‘n’ integers, where ‘n’ is a multiple of 10. Write a pseudo-code of the algorithm to find the average of the top 10% of the numbers. You may use any of the data structures studied in the class, but you need to specify the ADT of the data structure separately, before using it in your algorithm. Make sure that the ADT contains the function specifications which you are going to use.

7. We have seen Red Black (RB) trees in the class which are balanced by using various kinds of rotations. In this question you need to “dry-run” various insertions in the RB tree. Consider the left to right sequence of characters in your own entry number, e.g. for 2006CSZ8314, the sequence would be ‘2’, ‘0’, ‘0’, ‘0’, ‘6’, ‘C’ and so on. Consider the ordering between the characters as per their ASCII code (essentially implies that the digits come before the alphabets). Demonstrate the sequence of RB trees generated after inserting each character. Do show separately, the rotations required after each insertion to balance the tree. Also remember that you will need to think of an appropriate strategy to insert the duplicate keys in the tree.