COL106: Major Examination

11 January 2021 9:30am–11:30am
Maximum Marks 100

Instructions: Please read carefully before you start your examination.

- Make sure that you keep your videos on for the duration of the examination
- Answer each question and its subpart on a separate piece of paper
- Write your name, entry number and page number on each page
- At the end of the examination, you have 10 minutes to send images of your answers to your TAs using WhatApp/Telegram
- Once you have sent the pictures of your answers to your TAs, you need to convert them into a single PDF file and upload it on Gradescope by 12:00pm.
- Answers not uploaded on gradescope will not be evaluated
- In case you are unable to upload your answers on Gradescope by 12:00pm due to Internet or power issues, but you have sent the images of your answers to your TA by the deadline, please send your PDF to your TA as soon as possible.

Q1(a) (5 points) Describe the output for the following sequence of queue operations, assuming that the `dequeue()` function prints the element after removing it from the queue: `enqueue(15), enqueue(2), dequeue(), enqueue(23), enqueue(15), dequeue(), dequeue(), enqueue(7), enqueue(10), dequeue(), enqueue(17), enqueue(11), dequeue(), dequeue(), enqueue(24), dequeue(), dequeue()`.

Q1(b) (5 points) Consider a stack with an additional operation, `MULTIPOP`(S, k) which removes the top $k > 0$ objects of stack S, popping the entire stack if the stack contains fewer than k objects. The cost of the operation `MULTIPOP`(S, k) is the minimum of k and the number of elements remaining in the stack, while that of `PUSH`(S, x) and `POP`(S) is 1. Now consider a sequence of n stack operations on an initially empty stack, where each operation is either `PUSH`, `POP` or `MULTIPOP`. Give a short proof that the total cost of all these n operations is $\mathcal{O}(n)$.

Q1(c) (5 points) What does the following pseudocode compute when the parameter *node* is a pointer to a specific node in a binary search tree which contains only distinct keys ? Assume that the routine treeMaximum(node) returns the maximum key stored in the subtree rooted at node. Explain why the while loop is necessary.

```
Algorithm doSomething(node)
  if node.left ≠ NIL
    then return treeMaximum(node.left)
  y ← node.parent
  while y ≠ NIL and y.left = node
    node ← y
    y ← y.parent
  return y.key
```

Q1(d) (5 points) Consider red black trees of height at least two. Let b denote the number of black and r the number of red nodes in a red black tree. Then, what are the minimum and maximum values of the ratio r:b?

Q1(e) (5 points)

(a) Study the following pseudocode and describe what it is doing.

(b) Modify the code to print an in-order traversal of the tree.

```
function doSomething(Tree T)
        Stack S; Tree U;

        S ← new(Stack);
        S.push(T);
        while not empty(S) do
              U ← S.pop()
              if (U != null) then
                 print(U.value)
                 S.push(U.right)
                 S.push(U.left)
              endif
```

```
        end while
endFunction doSomething
```

Q2 (25 points) Tony Stark has created an underground labyrinth in the form of a maze in order to protect the Mindstone from Thanos. The stone lies at the point in the maze marked with **T**. He has further stationed d Iron Man droids from his Iron Legion at the point **S**. Each droid j has a size $size(j)$. If the droids sense some suspicious activity, one of them must fly to **T**. The only problem is that a droid j can pass a location l if and only if $size(j) \leq height(l)$. The problem is to pick the **biggest** droid that can reach **T** without getting stuck anywhere. Below is a sample maze (not drawn to scale) given in a PxQ matrix, where each cell represents a location in the maze. Non-zero values in the cells indicate heights at those locations and 0 indicates impenetrable gray walls.

S 3	3	3	3	3	5	5	5	5	5	5
0	0	3	0	0	5	0	0	0	0	1
0	0	3	2	2	5	0	0	0	0	1
0	0	0	0	0	5	0	0	2	0	1
0	0	0	0	0	5	0	0	2	0	1
2	1	1	1	1	5	2	2	2	3	3 T
2	0	0	0	0	0	0	0	3	0	0
2	0	0	0	0	0	0	0	3	0	0
3	3	3	3	3	3	3	3	3	0	0

1. Map the above description to a graph, to solve the problem of picking the biggest droid. What will be your nodes and edges? [*Hint: Think how to encode height information at different maze locations in your graph.*] [3]

2. Write pseudocode *CreateGraph* that outputs the above graph you describe as an adjacency matrix. Input to your pseudocode is a PxQ 2D matrix, as shown above. Entries in the matrix represent heights at those maze locations, with 0 indicating maze wall.[7]

3. Given the graph you create with say E edges and V vertices, write an $O((E+V)\log d)$ pseudocode *FindBiggestDroid* to pick the biggest droid that can reach T starting from S. You can assume to have an array of the d droids sorted in increasing order of sizes. [*Hint: The problem is not which droid reaches T in the least time, but the **biggest** droid that can reach T. A smaller droid with less flight time is not what we want.*] [8]

4. Using min-heaps, Dijkstra's shortest path algorithm can be made to run in $O((E+V)\log V)$ runtime. Can you modify Dijkstra's shortest path algorithm to solve this biggest droid selection problem, in $O((E+V)\log V + \log d)$ time? [*Hint: Metric to optimize will not the length of the path as in original Dijkstra.*] [7]

Q3 (25 points) Consider the following algorithm for sorting a set of numbers called Multipile Sorting. The algorithm works in two phases.
Phase 1: The numbers are put into a sequence of piles one-by-one, according to the following rules:

1. Initially, there are no piles. The first number forms a new pile consisting of the single number.

2. Each subsequent number is placed on the leftmost existing pile whose top number has a value greater than or equal to the new number's value; however, if the top number on all the piles is less than the new number, then it is placed to the right of all of the existing piles, thus forming a new pile.

Once all the numbers are placed in the piles, we proceed to phase 2.
Phase 2: The sorted sequence is obtained from the pile as follows:

1. Repeatedly remove and output the least of all the top numbers on the piles

Show that this algorithm can be implemented in O(n log n) time (without using any other sorting algorithm). More concretely:

(a) Under what conditions will the number of piles be minimum and under what conditions will they be maximum. Explain with examples. [5]

(b) Describe all the data structures you will use. [5]

(c) Show that in phase 1, each number can be inserted into the piles in O(log n) time. Give the pseudocode. [7]

(d) Show that in phase 2, each number can be extracted from the piles in O(log n) time. Give the pseudocode. [8]

Q4 (25 points) We wish to find the shortest paths between every pair of vertices in a directed graph, which may have negative edge weights, but no negative cycle. Input to the algorithm is a directed graph $G = (V, E)$ and the edge weights $W : E \to \mathcal{R}$. Let $d(i, j, k)$ be the length of the shortest path from vertex i to vertex j while using at most k edges. Design an algorithm to output the shortest paths between every pair of vertices. (Recall a variant of this algorithm was used in the class for finding the negative cycles).

(a) Write the rules to initialize $d(i, j, k)$ for $k = 1$ and update rules to iteratively compute $d(i, j, k)$ using previously computed distances $d(i', j', k')$ where $k' < k$. [6]

(b) Based on the above update rules, write the pseudo code of an efficient algorithm to print the shortest path (as well as its length) between every pair of vertices in the graph. Also write a simplified explanation of what your algorithm is trying to do. [7]

(c) Analyze the worst case time and space complexity of your algorithm. [6]

(d) Write a formal proof of correctness of your algorithm. [6]