THE UNIVERSITY OF SYDNEY

CONFIDENTIAL EXAM PAPER

This paper is not to be removed from the exam venue.

Computer Science

EXAMINATION

Semester 1- Main, 2022

COMP2123 Data Structures and Algorithms

EXAM WRITING TIME: 2 hours
READING TIME: 10 minutes

EXAM CONDITIONS:
This is a OPEN book examination.
All submitted work must be **done individually** without consulting someone else's help, in accordance with the University's "Academic Dishonesty and Plagiarism" policies.

MATERIALS PERMITTED IN THE EXAM VENUE:

MATERIALS TO BE SUPPLIED TO STUDENTS:

INSTRUCTIONS TO STUDENTS:
Type your answers in your text editor (Latex, Word, etc.) and convert it into a pdf file.

Submit this pdf file via Canvas. No other file format will be accepted. Hand-written responses will **not** be accepted.

Start by typing you student ID at the top of the first page of your submission. Do **not** type your name.

Submit only your answers to the questions. Do **not** copy the questions.

Do **not** copy any text from the permitted materials. Always write your answers in your own words.

For examiner use only:

Problem	1	2	3	4	Total
Marks					
Out of	10	10	20	20	60

Problem 1.

a) Suppose we have a binary search tree T containing n keys and some integer x. [5 marks]

```
1: def FOO(T, x)
2:     result ← 0
3:     current ← T.root
4:     while not current.isExternal() and current.key > x^2 do
5:         result ← result + current.key
6:         current ← current.left_child
7:     return result
```

Analyze the time complexity of running FOO.

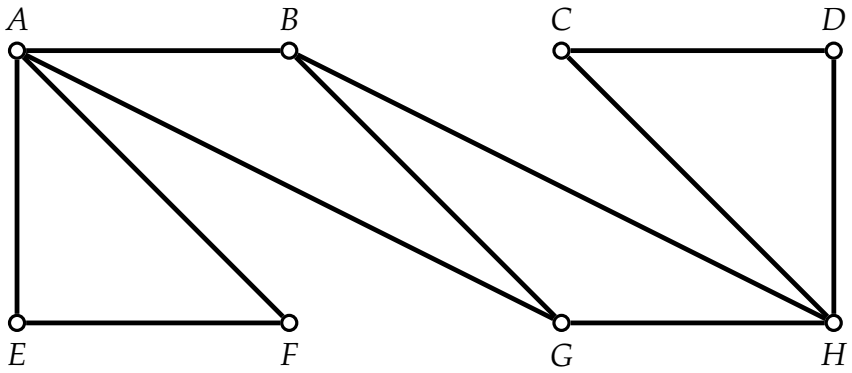
b) We are given a set W of positive integer weights $w_1, ..., w_n$ and we have an (infinite) set of identical boxes, each able to store a certain weight w_{max}. All weights in W are at most w_{max}. We want to determine the minimum number of boxes needed to store all n weights. [5 marks]

Example:
$W = \{3, 5, 3, 1, 2\}$
$w_{max} = 7$
In the example above, we need only two boxes, as we can store weights 3, 3, and 1 in one box and weights 5 and 2 together in a second box.

Construct a counterexample to show that the following algorithm doesn't compute the minimum number of boxes: Sort the weights in non-increasing order. For every weight w_i, check if there exists a box that can still store w_i. If there is, add w_i to the fullest box that can still take w_i (i.e., the one with least weight left out of all boxes having at least w_i weight left). If no such box exists, open a new box and put w_i in that.

```
1: def FEWEST BOXES(w_1, ..., w_n, w_max)
2:     boxCount ← 0
3:     Sort W in non-increasing order and renumber the weights such that
   |w_1| ≥ |w_2| ≥ ... ≥ |w_n|
4:     for each w_i (in the above order) do
5:         Let B be the set of boxes for which w_max minus the sum of the
   weights stored in the box is at least w_i
6:         if B = ∅ then
7:             Start a new box and add w_i to it
8:             boxCount ← boxCount + 1
9:         else
10:            Let b be the box in B storing the largest total weight
11:            Add w_i to box b
12:    return boxCount
```

Problem 2. Consider the following undirected graph:



Assuming that ties are broken lexicographically, **your task** is to:

a) Compute the breadth first search tree T of the graph starting from A. List the edges in T. [5 marks]

b) Compute the depth first search tree T of the graph starting from H. List the edges in T. [5 marks]

(You do **not** have to explain your answer.)

Problem 3. Computer scientists have decided to play the Hunger Games, but were unhappy about the small number of players per district and the rules being too simple. So they came up with a new version of the game, and asked you to come up with an efficient data structure to handle the Games.

Rules: There are n districts, numbered 1 to n. Each district has its own number of players: initially, each district has $m > 0$ players. Each player has a score (non-negative integer), initially set to 1. Each district also has an integer score, initially set to 0.

- When two districts play against each other, their best players compete (one from each district). The winning player and their district both have their score incremented; the losing player and their district both have their score decremented.
- When a player reaches score 0, they are disqualified (removed from their district and the game).
- When a district reaches 0 players, it is disqualified (removed from the game).
- The last district in play wins.

Your task: Your data structure should use $O(nm)$ space, and implement the following operations:

- GETKTHDISTRICT(k): Given a positive integer k, return a pointer to the district currently ranked k^{th} (i.e., the district with the highest score is ranked first), or null if $k > n$. Ties are broken by district number, so if district 10 and district 98 have the same score, district 98 is considered "better" because $98 > 10$. This operation should run in $O(\log n)$ time.
- COMPETE(k, j, k_wins): Given two positive integers $1 \leq k, j \leq n$ and a Boolean *k_wins*, record the result of a game between the districts currently ranked k^{th} and j^{th} and update the data structure according to the rules (or do nothing if $k = j$ or if either k or j is not between 1 and n). If *k_wins* is True, then the k^{th} district wins; otherwise, the j^{th} district wins. This operation should run in $O(\log n + \log m)$ time.
- GETSTRONGEST(): Returns a pointer to the district whose best player has the largest score. This operation should run in $O(n)$ time.

Note that the number of (currently playing) districts n can change during the Hunger Games, as districts get eliminated.

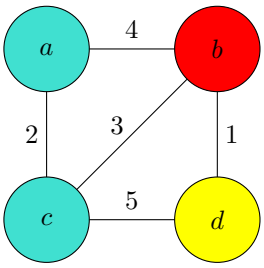
a) Describe your data structure implementation in plain English. [8 marks]

b) Prove the correctness of your data structure *(pay special attention to the rules of the game!)*. [7 marks]

c) Analyze the time and space complexity of your data structure. [5 marks]

Problem 4. We're given a connected graph with positive weights on the edges stored as an adjacency list. Additionally, every vertex of the graph has a colour assigned to it. These colors come from a set of size k (k is **not** a constant). We call a path between two vertices u and v a rainbow path if no two consecutive vertices along this path have the same color. For a given pair of vertices u and v, you need to determine the length of the shortest rainbow path between them, if one exists (output ∞ (i.e., infinity) otherwise).

Example:



In the above example we have three colors: turquoise (vertex a and c), red (vertex b), and yellow (vertex d). The shortest rainbow path between a and c is path a, b, c with length 7. The path a, c is shorter with length 2, but it isn't a rainbow path, since a and c have the same color. The path a, b, d, c would also be a rainbow path, but with length 10 it isn't the shortest one between a and c.

Your task is to give an algorithm for this problem that runs in $O(m + n \log n)$ time, i.e., it shouldn't depend on k in any way. Remember to:

a) Describe your algorithm in plain English. [8 marks]

b) Prove the correctness of your algorithm. [7 marks]

c) Analyze the time complexity of your algorithm. [5 marks]