A

BRAC UNIVERSITY
Department of Computer Science and Engineering

Examination: Semester Final Exam — Semester: Spring 2025
Duration: 1 Hour 45 Minutes — Full Marks: 40

CSE 221: Algorithms

Answer the following questions.
Figures in the right margin indicate marks.

Name:	ID:	Section:

1 **a. CO1** For a directed graph, does the number of back edges equal the number of cycles present? **Explain** briefly. **01**

b. CO1 Given a directed graph G = (V, E) represented with an adjacency list, **explain** how you can find whether the graph is a directed acyclic graph (DAG) or not. Your method must be as asymptotically efficient as possible. **01**

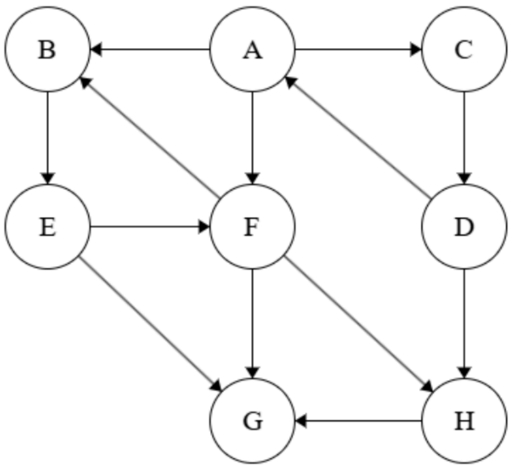

Figure 1: Q1(b)

c. CO1 **Simulate** Kosaraju's algorithm on the given graph Q1(b) to find the strongly connected components (SCCs). Label each SCC with the set of nodes it contains. While performing graph traversal, use node A as the starting node and break ties in alphabetical order. **04**

d. CO1 Now, treat each SCC identified in Q1(c) as a single "super node." In this new graph, every edge from the original graph either becomes internal (and thus disappears, since it stays within an SCC) or becomes an edge connecting two different SCCs. This transformation results in a Directed Acyclic Graph (DAG). **Draw** this DAG. **01**

e. CO1 Given a set of SCCs, what is the maximum number of new SCCs that can appear if we delete one edge from the DAG of these SCCs? **01**

Consider the following algorithm:

- Step 1: Perform DFS on the given graph. For each node, keep track of the finishing time (the time when the algorithm finishes exploring all descendants of the node).
- Step 2: Sort the nodes in ascending order of the finish time.
- Step 3: Perform DFS in the sorted order of the nodes. For each DFS traversal, include each reachable node as one SCC.

f. **CO2** **Prove** that the given algorithm is incorrect by providing a graph as a counterexample, on which the algorithm will fail to find the correct set of SCCs. **02**

2 The country of Technovia is setting up an emergency grid that connects 8 major cities: A (Capital), B, C, D, E, F, G, H. These cities are connected by one-way roads. Each road has a traffic weight which is used to estimate the delivery delay due to congestion. Medical aid and personnel must be delivered from the capital (A) to all other cities. The map of the country is given below:

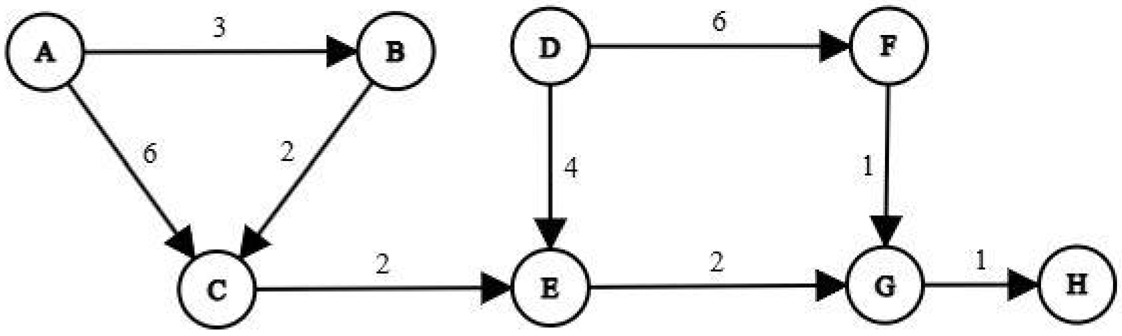

Figure 2: Q2.

a. **CO1** Suppose you, an algorithm enthusiast, have been assigned to determine the minimum delivery time from capital A to all other cities. **04**

1. Which SSSP (single source shortest path) algorithm would you prefer to solve the problem?
2. **Simulate** your mentioned algorithm with the necessary data structures to find the minimum delivery times from the capital to all other cities. Travel times are equal to the traffic weights of the roads.

b. **CO3** Continuing from the given scenario, the Ministry of Transport enforces that while delivering medical resources, after using any high-traffic road (traffic weight > 4), the next road used in the delivery path must have a traffic weight ≤ 4. This is to prevent back-to-back gridlock. **02**

Propose a modification to your algorithm of Q2(a) to determine the minimum delivery time from capital A to all other cities while strictly following the additional constraint of traffic weights. **Present** your idea using programmable code/pseudocode/step-by-step instructions.

c. **CO1** Now the Ministry wants to design a minimum-traffic infrastructure plan that connects all the cities (an MST of the graph). Will Kruskal's or Prim's MST finding algorithm work properly for the given one-way roads? If yes, **justify** your answer. Otherwise, **propose** necessary modifications to the graph so that the algorithm works. **02**

d. **CO3** While finding the MST of the graph, we want to enforce a rule that '*No city can have more than two (02) roads*'. **02**

Propose a modification to the MST algorithm (either Kruskal's or Prim's) to accommodate this requirement. **Present** your idea using programmable code/pseudocode/step-by-step instructions.

3 The final exam has just ended, and you are on a well-deserved trip to a remote area. During your journey, you have collected different types of digital content that you want to share with your friends and family. Each *Content Type* has a *Total Duration* and a *Value*. Total Duration measures the content's pre-compression size, while Value reflects its importance or meaning to the audience.

Due to bandwidth constraints, one of your friends suggested using Huffman encoding, a lossless compression technique that assigns shorter binary codes to more voluminous (i.e., longer duration) items.

Content Type	Total Duration (Seconds)	Value
A	13	20
B	11	14
C	9	6
D	8	17
E	6	11
F	5	7
G	4	8

a. **CO1** Construct a Huffman Tree based on the given **Total Duration** to assign optimal binary codes to each content type. **04**

b. **CO1** Calculate the Total Number of Bits *(duration * code length)* required to transmit the full content of each type. **03**

c. **CO1** Again, due to bandwidth constraints, you are now trying to select which content to send and how much of each. You can shorten any content, but its value drops proportionally if you do so. Your goal is to choose the types and lengths of the contents that give you the highest total value without going over *30 seconds*. **Determine** which content to send and how long each part should be. **03**

Please Turn Over

4 a. String1: <Your ID>
CO1 String2: __________

i. Create String2 in a way that the following constraints match. **Write** the string. **03**
   - Length of String2 = 8 (eight)
   - Length of LCS(String1,String2) = 5 (five)
   - There are exactly three distinct LCS (of length = 5)

ii. **Write** all three LCS strings (just write, no need to show any work). **01**

b. String1: <Your ID>
CO1 String2: “20251905”

iii. **Find** the length of the LCS of these two strings. You can use a memory table or a recursion tree. Show your work. **04**

iv. Using your work from the previous question, **find** the LCS string. **02**

B

BRAC UNIVERSITY
Department of Computer Science and Engineering

Examination: Semester Final Exam — Semester: Spring 2025
Duration: 1 Hour 45 Minutes — Full Marks: 40

CSE 221: Algorithms

Answer the following questions.
Figures in the right margin indicate marks.

Name:	ID:	Section:

1 **a. CO1** For a directed graph, does the number of back edges equal the number of cycles present? **Explain** briefly. **01**

b. CO1 Given a directed graph G = (V, E) represented with an adjacency list, **explain** how you can find whether the graph is a directed acyclic graph (DAG) or not. Your method must be as asymptotically efficient as possible. **01**

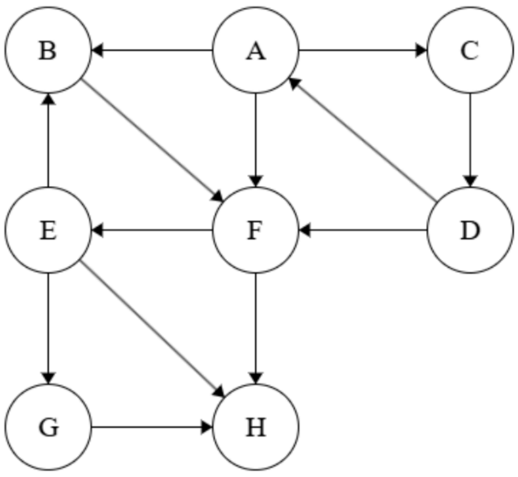

Figure 1: Q1(b)

c. CO1 **Simulate** Kosaraju's algorithm on the given graph Q1(b) to find the strongly connected components (SCCs). Label each SCC with the set of nodes it contains. While performing graph traversal, use node A as the starting node and break ties in alphabetical order. **04**

d. CO1 Now, treat each SCC identified in Q1(c) as a single "super node." In this new graph, every edge from the original graph either becomes internal (and thus disappears, since it stays within an SCC) or becomes an edge connecting two different SCCs. This transformation results in a Directed Acyclic Graph (DAG). **Draw** this DAG. **01**

e. CO1 Given a set of SCCs, what is the maximum number of new SCCs that can appear if we delete one edge from the DAG of these SCCs? **01**

Consider the following algorithm:

- Step 1: Perform DFS on the given graph. For each node, keep track of the finishing time (the time when the algorithm finishes exploring all descendants of the node).
- Step 2: Arrange the nodes in ascending order based on the finish time.
- Step 3: Perform DFS in the sorted order of the nodes. For each DFS traversal, include each reachable node as one SCC.

f. **CO2** **Prove** that the given algorithm is incorrect by providing a graph as a counterexample, on which the algorithm will fail to find the correct set of SCCs. **02**

2 The country of Technovia is setting up an emergency grid that connects 8 major cities: A (Capital), B, C, D, E, F, G, H. These cities are connected by one-way roads. Each road has a traffic weight which is used to estimate the delivery delay due to congestion. Medical aid and personnel must be delivered from the capital (A) to all other cities. The map of the country is given below:

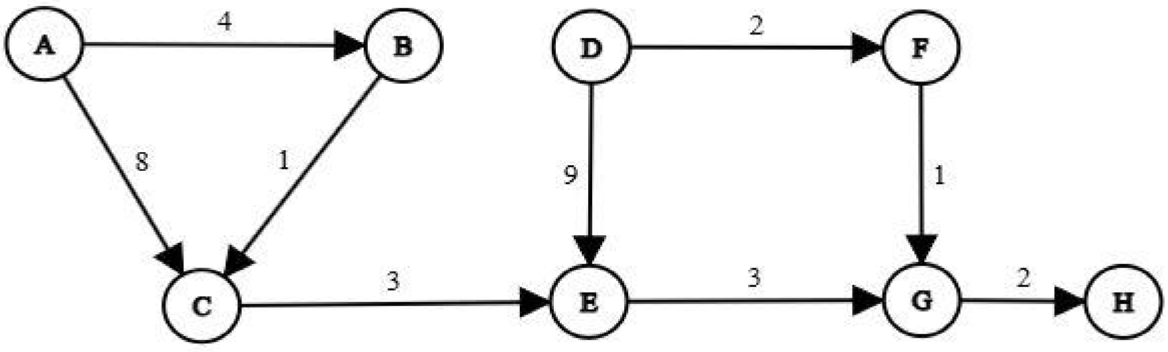

Figure 2: Q2

a. **CO1** Suppose you, an algorithm enthusiast, have been assigned to determine the minimum delivery time from capital A to all other cities. **04**

1. Which SSSP (single source shortest path) algorithm would you prefer to solve the problem?
2. **Simulate** your mentioned algorithm with the necessary data structures to find the minimum delivery times from the capital to all other cities. Travel times are equal to the traffic weights of the roads.

b. **CO3** Continuing from the given scenario, the Ministry of Transport enforces that while delivering medical resources, after using any high-traffic road (traffic weight > 5), the next road used in the delivery path must have a traffic weight ≤ 5. This is to prevent back-to-back gridlock. **02**

Propose a modification to your algorithm of Q2(a) to determine the minimum delivery time from capital A to all other cities while strictly following the additional constraint of traffic weights. **Present** your idea using programmable code/pseudocode/step-by-step instructions.

c. **CO1** Now the Ministry wants to design a minimum-traffic infrastructure plan that connects all the cities (an MST of the graph). Will Kruskal's or Prim's MST finding algorithm work properly for the given one-way roads? If yes, **justify** your answer. Otherwise, **propose** necessary modifications to the graph so that the algorithm works. **02**

d. **CO3** While finding the MST of the graph, we want to enforce a rule that '*No city can have more than two (02) roads*'. **02**

Propose a modification to the MST algorithm (either Kruskal's or Prim's) to accommodate this requirement. **Present** your idea using programmable code/pseudocode/step-by-step instructions.

3 The final exam has just ended, and you are on a well-deserved trip to a remote area. During your journey, you have collected different types of digital content that you want to share with your friends and family. Each *Content Type* has a *Total Duration* and a *Value*. Total Duration measures the content's pre-compression size, while Value reflects its importance or meaning to the audience.

Due to bandwidth constraints, one of your friends suggested using Huffman encoding, a lossless compression technique that assigns shorter binary codes to more voluminous (i.e., longer duration) items.

Content Type	Total Duration (Seconds)	Value
A	14	21
B	12	15
C	10	7
D	9	18
E	7	12
F	6	8
G	5	7

a. **CO1** Construct a Huffman Tree based on the given **Total Duration** to assign optimal binary codes to each content type. **04**

b. **CO1** Calculate the Total Number of Bits *(duration * code length)* required to transmit the full content of each type. **03**

c. **CO1** Again, due to bandwidth constraints, you are now trying to select which content to send and how much of each. You can shorten any content, but its value drops proportionally if you do so. Your goal is to choose the types and lengths of the contents that give you the highest total value without going over *35 seconds*. **Determine** which content to send and how long each part should be. **03**

Please Turn Over

4 **a.** String1: <Your ID>
CO1 String2: __________

i. Create String2 in a way that the following constraints match. **Write** the string. **03**
   - Length of String2 = 8 (eight)
   - Length of LCS(String1,String2) = 5 (five)
   - There are exactly three distinct LCS (of length = 5)

ii. **Write** all three LCS strings (just write, no need to show any work). **01**

b. String1: <Your ID>
CO1 String2: "20251905"

iii. **Find** the length of the LCS of these two strings. You can use a memory table or a recursion tree. Show your work. **04**

iv. Using your work from the previous question, **find** the LCS string. **02**