Family Name: ..... **Other Names:** .....

Student ID: ..... **Signature** .....

COMP 261 : Test 2

18 April 2024,

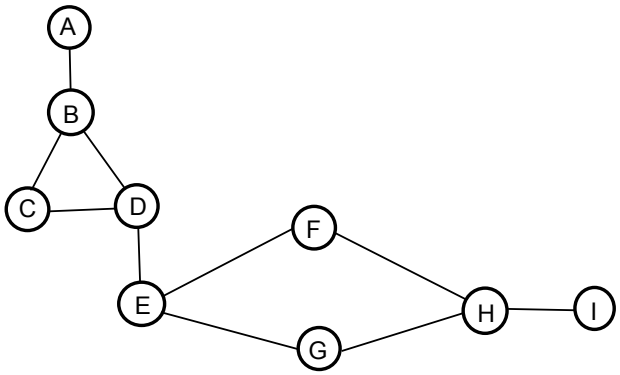
Instructions

- Time allowed: **50 minutes**
- Attempt **all** the questions. There are 50 marks in total.
- Write your answers in this test paper and hand in all sheets.
- If you think a question is unclear, ask for clarification.
- This test contributes 15% of your final grade.
- You may use dictionaries and calculators.
- You may write notes and working on this paper, but make sure your answers are clear.

Questions	**Marks**	
1. Adjacency Matrix and Adjacency List	[13]	
2. Shortest Paths	[15]	
3. Strongly Connected Components.	[10]	
4. Articulation Points	[12]	
	TOTAL:	

Question 1. Adjacency Matrix and Adjacency List **[13 marks]**

In the graph below, the nodes are labelled as A, B, ..., I. The edges are all undirected.



Please note that we use the labels (A-I), instead of integers(0-8), to represent the nodes in the following questions.

(a) **[5 marks]** Fill in the *adjacency matrix* of the above graph.

	A	B	C	D	E	F	G	H	I
A									
B									
C									
D									
E									
F									
G									
H									
I									

(Question 1 continued on next page)

(Question 1 continued)

(b) **[5 marks]** Write the *adjacency list* of the above graph.

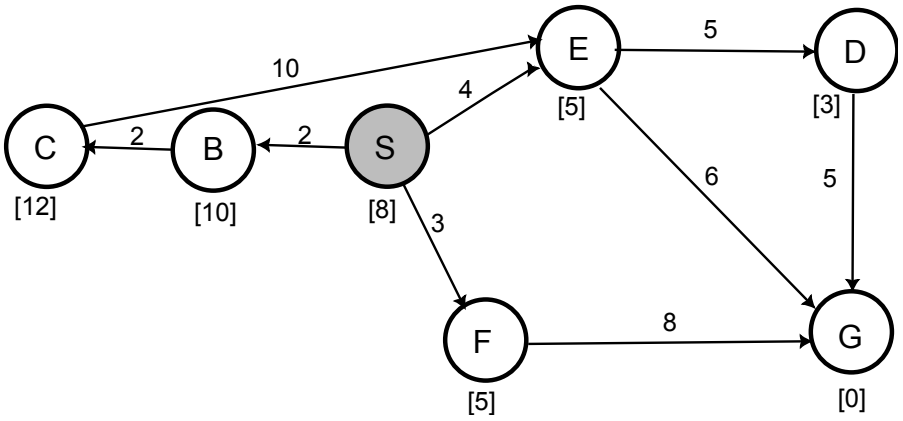
A:
B:
C:
D:
E:
F:
G:
H:
I:

(c) **[3 marks]** Suppose we want to find all the articulation points in the graph using the algorithm described in lectures, which data structure between *Adjacency Matrix* and *Adjacency List* has lower cost? Briefly justify your answer.

Question 2. Shortest Paths **[15 marks]**

Suppose we are using A$*$ Search algorithm to search for the shortest path from node S to node G in the graph below. The fringe is a priority queue of $\langle node, edge, CostSoFar, EstimatedTotalCost \rangle$ items, ordered by *EstimatedTotalCost*.

The number on the edge is the edge length. The number in square brackets [ ] is the estimated cost from the node to the goal.



(Question 2 continued on next page)

(Question 2 continued)

Show what the algorithm will do on each of the next four iterations:

- which node it will visit
- what will be added to the Backpointers. If nothing is added, briefly explain why.
- what items will be added to the fringe. If nothing is added, briefly explain why.

Write the items and nodes in the format: $\langle node, edge, CostSoFar, EstimatedTotalCost \rangle$.

Show the shortest path that it finds.

Iteration 1:

Node visited: $\langle S, null, 0, 8 \rangle$

Additions to Backpointers Map:

Additions to fringe:

Iteration 2:

Node visited:

Additions to Backpointers Map:

Additions to fringe:

Iteration 3:

Node visited:

Additions to Backpointers Map:

Additions to fringe:

Iteration 4:

Node visited:

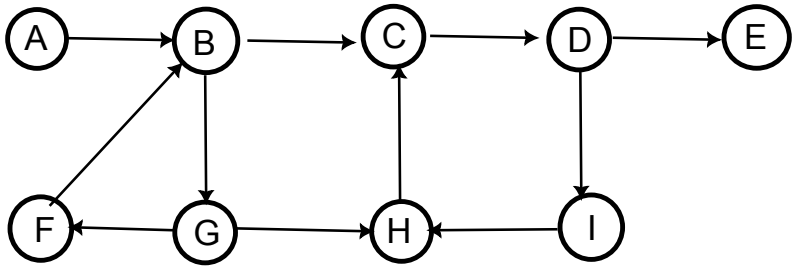
Additions to Backpointers Map:

Additions to fringe:

Shortest Path:

Question 3. Strongly Connected Components. **[10 marks]**

Suppose we are using Kosuraja's algorithm described in lectures to search for the strongly connected components in the graph below.



The algorithm is given on the facing page for your reference.

The start node is A. The neighbours of a node are enumerated in *alphabetic order*.

Show how it works by the following:

List the nodes in the order they are visited. (If a node has many neighbours, they must be visited in alphabetic order.)

List the nodes in the nodeList in the order they are added

Label the nodes with their component number.

Rewrite each component as a list of nodes. The nodes in each component must be listed in the order their component numbers are assigned.

Alphabet: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

```
Kosuraja(graph):
        for each node in graph:
                node.component ← −1   // initialize nodes to not be in a component
        componentNum ← 0
        nodeList ← empty list;
        visited ← empty set

        for each node in graph:
           if node is not visited then
                   // traverse graph from node forward along edges, adding nodes to nodeList in post−order
                   ForwardVisit(node, nodeList, visited )



        for each node in nodeList in reverse order:
           if node.component = −1 then
                   // traverse graph from node backward along edges, marking nodes with the component number
                   BackwardVisit(node, componentNum)
                   componentNum++

// Search forward from node, putting node on nodeList after visiting everything it can get to.
ForwardVisit(node, nodeList, visited ):
        if node is not in visited then
                add node to visited .
                for each neighbour in node.outNeighbours:
                        ForwardVisit(neighbour, nodeList, visited )
                add node to nodeList

// Search backwards from node, marking all the nodes that can get to it as the same component
BackwardVisit(node, componentNum):
        if node.component = −1 then
                node.component ← componentNum
                for each backNeighbour in node.inNeighbours:
                        BackwardVisit(backNeighbour, componentNum).
```

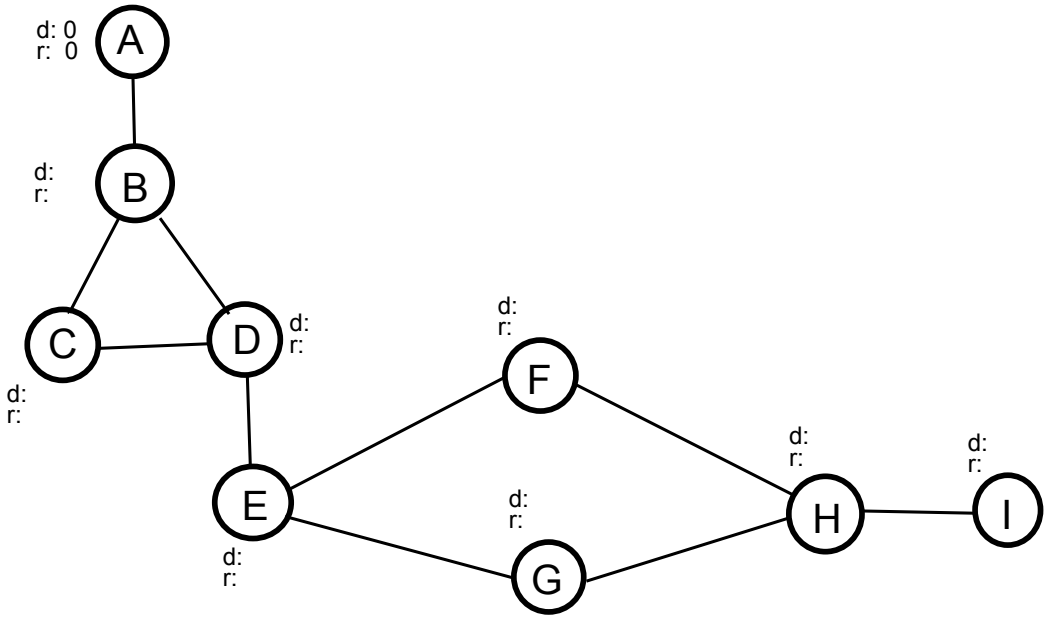
Question 4. Articulation Points **[12 marks]**

Find the articulation points using the Recursive Articulation Point Algorithm described in lectures. The start node is A, whose depth and reachBack are set to 0.

- Calculate the depth (d) of each node.
- Calculate the reachBack (r) value of each node.
- List the articulation points in the order they are identified.

The algorithm is given on the facing page for your reference.

The neighbours of a node are enumerated in *alphabetic order*.



List the Articulation Points in the order they are identified:

Alphabet:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

```
FindArticulationPoints (graph, start ):
    for each node:
        node.depth ← −1
    articulationPoints ← emptyset
    start .depth ← 0
    numSubtrees ← 0
    for each neighbour of start
        if neighbour.depth = −1 then
            RecursiveArtPts(neighbour, 1, start )
            numSubtrees ++
    if numSubtrees > 1 then add start to articulationPoints

RecursiveArtPts(node, depth, fromNode):
    node.depth ← depth
    reachBack ← depth
    for each neighbour of node other than fromNode
        if neighbour.depth ≠ −1 then
            reachBack ← min(neighbour.depth, reachBack)
        else
            childReach ← RecursiveArtPts(neighbour, depth + 1, node)
            if childReach ≥ depth then add node to articulationPoints
            reachBack ← min(childReach, reachBack )
    return reachBack
```

* * * * * * * * * * * * * * *

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.