Family Name:..... **First Name:**.....

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

COMP261: Algorithms and Data Structures

Term Test 3

9 May 2024 ** WITH SOLUTIONS **

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 write notes and working on this paper, but make sure your answers are clear.
- Only silent non-programmable calculators or silent programmable calculators with their memories cleared are permitted in this examination.
- No electronic dictionaries are allowed.
- Paper foreign to English language dictionaries are allowed.

Sections	**Marks**	
1. Network Flows	[20]	
2. Centrality	[10]	
3. Cycles and Spanning Trees	[20]	
	TOTAL:	

SECTION A Network Flows

1. In a network flow problem, what constitutes the 'source'? **[1 mark]**

a) Vertex with the maximum capacity

b) Vertex with no outgoing edges

c) Vertex with no incoming edges

d) Vertex with the least capacity

c

2. How many constraints must a flow satisfy in a network flow problem? **[1 mark]**

a) One

b) Two

c) Three

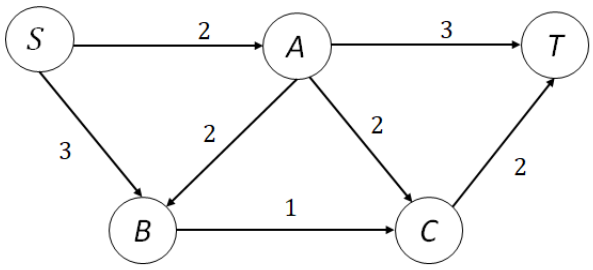
d) four

b

3. State True or False. In a standard network flow problem, under certain circumstances, a vertex `v` that is not a source or a sink can have a total in-flow that has a different value than its total outflow. **[1 mark]**

False

4. Consider the following graph **[3 marks]**



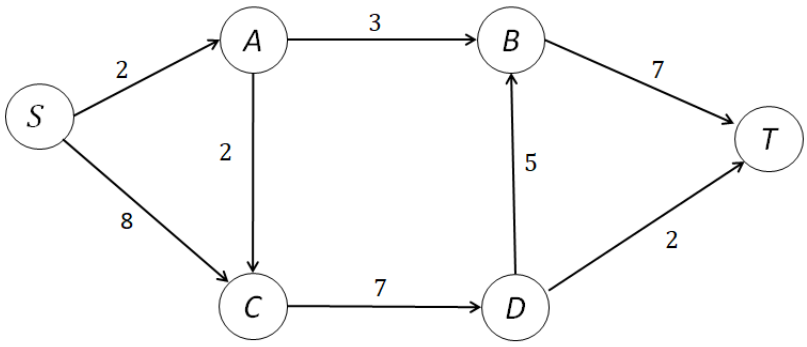
If the Ford-Fulkerson method identifies `S->A->C->T` as the first augmentation path, are there any remaining augmentation paths within the residual graph? If so, list the vertices in a valid augmentation path in the box below along with the value of the corresponding flow. If there is no other augmentation path that can be found write 'none '.

S-B - C - A - T with flow 1

5. Suppose that you are asked to implement the Edmond-Karp algorithm to find maximum flow through a network. Using the following pseudo-code of the algorithm answer the following questions. **[12 marks]**

```
EdmondKarp(Graph, start, goal):
 // Build Residual Graph to include flow values and reverse edges
1 for each edge e in G where e is of the form ⟨fromVertex,toVertex, capacity⟩
2    add e ← ⟨fromVertex,toVertex, capacity,0⟩ to the Residual graph //flow value = 0 initially
3    add the corresponding reverse edge e' ← ⟨toVertex,fromVertex,0,0⟩ to the Residual graph
4 maxflow ← 0
5 Repeat
6    Use BFS to find a path P from start to goal in the Residual Graph such that
     all edges e in P have capacity(e)>0
7    if P exists
       //Find bottleneck:the capacity of the minimum capacity edge in P
8       pathFlow ← Bottleneck(P)
9       Output (P, pathFlow) as an augmentation path
10      maxflow ←maxflow + pathflow
11      Update the Residual graph to reflect the changes in the capacities of edges
        that constitue P and the corresponding reverse edges
13   else
14     output Maxflow
15     return
```

(a) **[3 marks]** In the following graph, if the first path discovered by BFS is `S->A->B->T`, which edges in the residual graph will be updated at step 11 of the pseudo-code? Also list the updated `capacity` and `flow` for each updated edge.



S to A: Capacity 0, Flow 2
A to S: Capacity 2, Flow 0
A to B: Capacity 1 Flow 2
B to A: Capacity 2 Flow 0
B to T: capacity 5 Flow 2
T to B: Capacity 2 Flow 0

(b) **[8 marks]** For the path `S->A->B->T` identified above, trace out what the `BFS` will do at each iteration in terms of:

- contents of the Queue
- contents of the Backpointers Map

Note: If at a given iteration more than one node is to be added to the queue, add them in alphabetical order. The first iteration has been done for you.

Queue: S

Iteration 1:

Queue: A C

Backpointers: ⟨A → Edge S-to-A⟩, ⟨C → Edge S-to-C⟩

Iteration 2:

Queue: C B

Backpointers: ⟨A → Edge S-to-A⟩, ⟨C → Edge S-to-C⟩, ⟨B → Edge A-to-B⟩

Iteration 3:

Queue: B D

Backpointers: ⟨A → Edge S-to-A⟩, ⟨C → Edge S-to-C⟩, ⟨B → Edge A-to-B⟩, ⟨D → Edge C-to-D⟩

Iteration 4:

Queue: D

Backpointers: ⟨A → Edge S-to-A⟩, ⟨C → Edge S-to-C⟩, ⟨B → Edge A-to-B⟩,⟨D → Edge C-to-D⟩, ⟨T → Edge B-to-T⟩

(c) **[1 mark]** When running Ford-Fulkerson as per the implementation, how does the algorithm know when to terminate?

The program terminates when the BFS does not return any path from start to goal in the residual graph.

6. Name the constraint that the flow identified in figure 2, for the network depicted in figure 1, fails to satisfy. **[2 marks]**

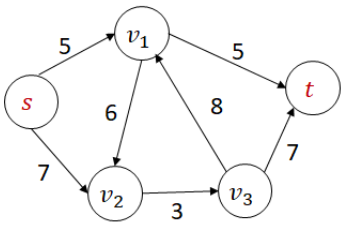


Figure 1

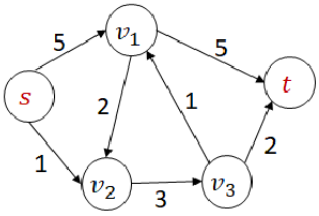


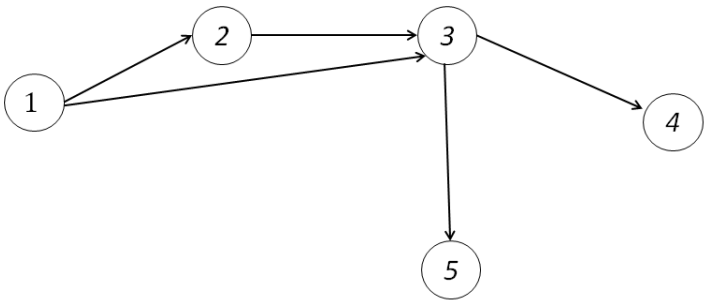
Figure 2

balance constraint / fails to conserve flow at vertex v1

SECTION B Centrality

7. Trace out the steps of the PageRank algorithm for the given web graph using the provided pseudo-code with the parameter `iter` set to 1 and `dampingfactor` set to .85. **[6 marks]**

```
ComputePageRank(Graph, iter, dampingfactor):
  nNodes ← get count of nodes in the graph
  for each node n in Graph
    set pageRank(n) ← 1.0/nNodes
  count ← 1
  Repeat
    randomjumpShare ← (1−dampingfactor)/nNodes
    sinkShare ← 0
    for each node n in the Graph that is a sink node (has no outlinks)
        sinkShare ← sinkShare + dampingfactor x (pageRank(n)/nNodes)
    for each node n in the Graph
        neighbourShare ← 0
        for each neighbour b of n
            neighbourShare ← neighbourShare + pageRank(b)/outdegree(b)
        newPageRank(n) ← sinkShare + randomjumpShare + d x neighbourShare
    Update pageRank with newpageRank
    count++
  Until count > iter
```



PR(1) = PR(2) = PR(3) = PR(4) = PR(5) = 1/5 = .2

Iter 1:

sinkShare = .85 x (1/5 + 1/5)/5= .85 x .08 = .068

RandomJumps = .15/5 = .03

PR(1) = .068 + .03 = .098

PR(2) = .068 + .03 + .85 x (1/5) x (1/2)= .183

PR(3) = .068 + .03 + .85 x ((1/5) x(1/2) + (1/5)) = 353

PR(4) = .068 + .03 + .85 x (1/5) x (1/2) = .183

PR(5) = .068 + .03 + .85 x (1/5) x (1/2) = .183

8. Which of the following will you use to assess the vulnerability of critical infrastructure networks, such as water distribution systems or power grids, by identifying nodes that, if disrupted, could lead to cascading failures? **[1 mark]**

a) Degree centrality

b) Closeness centrality

c) Betweenness centrality

d) Network flows

c

9. What kind of centrality would you want to analyze in a graph if you wanted to inject information that flows through the shortest path in a network and have it spread quickly? **[1 mark]**

a) Degree

b) Group

c) Closeness

d) Betweenness

c

10. Select **all** of the following problems where `closeness` would be a valuable centrality measure. **[2 marks]**

a) Optimize Bus stops locations

b) Optimize Bus routes

c) Optimize Traffic signal timings

d) Optimize Bus stop timings along a route

a, b

SECTION C Cycles and Spanning Trees

11. When using DFS for cycle detection, what does the presence of a back edge indicate? **[1 mark]**

a) The graph is acyclic

b) A cycle is present in the graph

c) The graph is connected

d) The graph is disconnected

b

12. Which of the following data structures is typically used in Prim's algorithm to efficiently select the next edge to add to the spanning tree? **[1 mark]**

a) Priority queue

b) Stack

c) Queue

d) Array

a

13. In Kruskal's algorithm, which of the following operations is performed to efficiently determine whether adding an edge creates a cycle in the spanning tree? **[1 mark]**

a) Breadth-first search

b) Depth-first search

c) Find operation with disjoint data structure

d) Dijkstra's algorithm

c

14. If a graph has `n` vertices, what is the maximum number of edges it can have without forming a cycle? **[1 mark]**

a) `n-1`

b) `n`

c) `n-2`

d) `2n`

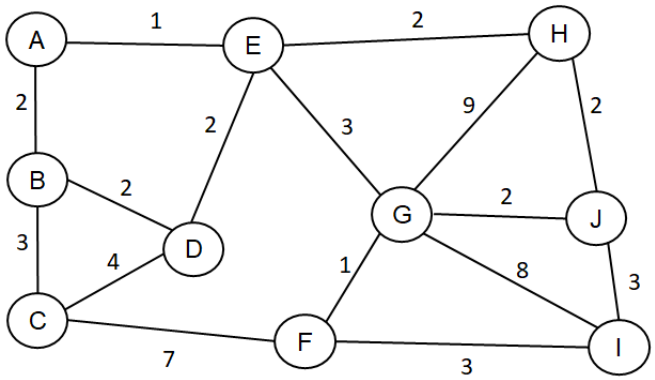
a

15. State True or False: If a graph has multiple edges with the same weight, Prim's algorithm may produce different minimum spanning trees depending on the order in which edges are processed. **[1 mark]**

True

16. Using Prim's algorithm find the cost of the minimum cost spanning tree of the following graph. You should show the edges and their weights (e.g., AC 6) in the order of being added into the tree by the algorithm considering A as the starting node. **[6 marks]**

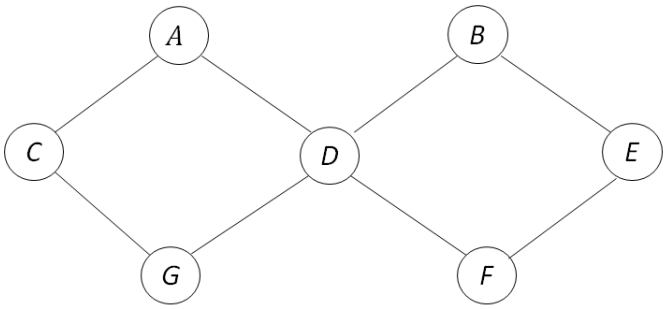
Note: The cost of a spanning tree equals the sum of the costs of all its edges.



Minimum cost remains the same - 18

AE - 1
EH - 2 (at this step someone might have chosen to pick AB or ED since AB, ED and EH all have the same weight and are connected to the two nodes already visited: A and E)
HJ - 2
JG - 2
GF - 1
AB -2
BD - 2
JI -3
BC -3
Total cost: 18

17. Consider the following undirected graph g: **[5 marks]**



Using the provided pseudo-code, answer the following questions related to the call to `CycleDetection(g)`:
a) List the sequence of the calls made to `isCyclicDFS()` starting with the initial call `isCyclicDFS(g, A, visited, -1)`.

b) Identify the call when the algorithm detects the presence of a cycle. Provide an explanation for your answer.

c)For each call also identify which node will be marked as visited.

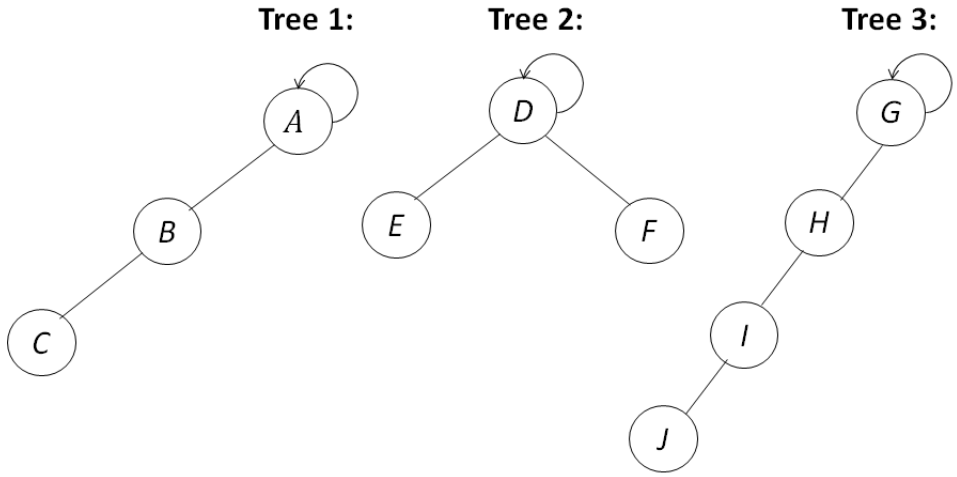
Note: If at a given iteration a node has more than one neighbours, traverse them in alphabatic order.

```
function CycleDetection(Graph graph):
  nNodes ← get count of nodes in the graph
  boolean visited [nNodes] // all initialized to false as none of the nodes have been visited
  for each node v in the graph
    if (! visited [v])
        if (isCyclicDFS(graph, v, visited , −1))
          return true
  return false
end function

function isCyclicDFS(Graph, start , visited [], parent)
  set visited [ start ] to true
  for each neighbour n of node start
    if (! visited [n])
        if (isCyclicDFS(graph, n, visited , start ))
        return true
    else if (n != parent)
        return true
  return false
end function
```

isCyclicDFS(graph, A, visited, -1),visited (A)=true →
isCyclicDFS(graph, C, visited, A), visited(C) = true →
isCyclicDFS(graph, G, visited, C), visited(G) = true →
isCyclicDFS(graph, D, visited, G), visited(D) = true. Cycle as A is neighbour of which is already visited and A is not eqaul to G

18. Consider the following forest of trees. **[4 marks]**

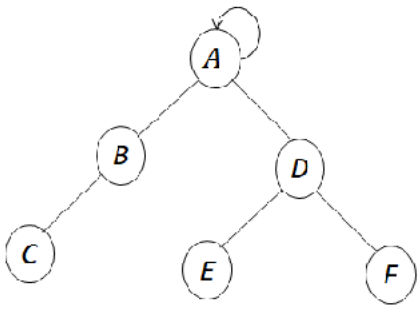


Using the pseudo-code below, present the forest graphs obtained after each of the following two merging operations performed in a sequence: `Union(B,E)` followed by `Union(I,D)`.

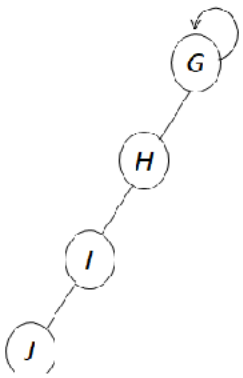
```
Union(x, y):
    xroot ← Find(x)
    yroot ← Find(y)
    if (xroot == yroot)
        return
    else
        if (xroot.depth < yroot.depth)
            xroot.parent ← yroot
            remove xroot from forest
        else
            yroot.parent ← xroot
            remove yroot from forest
            if (xroot.depth == yroot.depth)
                xroot.depth ++
```

Union(B,E)

Tree 1

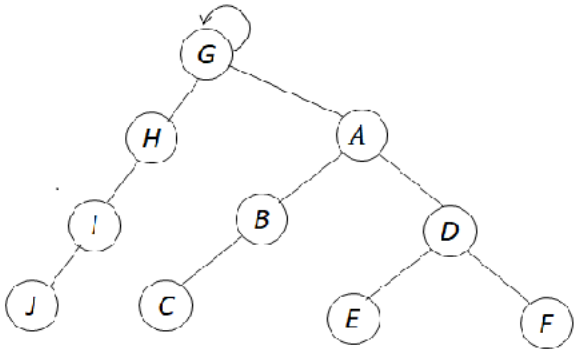


Tree 3:



Union(I,D)

Tree 3



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

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.