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

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

COMP 261 : Test 2

6 April 2023, ** WITH SOLUTIONS **

Instructions

- Time allowed: **50 minutes**
- Attempt **all** the questions. There are 50 marks in total.
- **In-person**: Write your answers in this test paper and hand in all sheets.
  Remote: Type your answers in the template file and submit to "Test 1 Remote" on the COMP 261 submission system.
- If you think a question is unclear, ask for clarification.
- This test contributes 10% 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. Regular Expressions	[10]	
2. Adjacency Matrix Data Structures	[6]	
3. Adjacency List Data Structures	[9]	
4. Shortest Paths.	[10]	
5. Connected Components.	[10]	
6. A∗ Search	[5]	
	TOTAL:	

Question 1. Regular Expressions. **[10 marks]**

(See the reference table below)

(a) **[3 marks]** Carefully circle or underline all the sections of the following text that are matched by the regular expression:

```
t[a-z]*t
```

```
this is text with twenty words with lots of tee.
the pattern to the question is often totally different.
```

(b) **[3 marks]** Carefully circle or underline all the sections of the following text that are matched by the regular expression:

```
catch\s((this|that)\s(cat|dog)\s)*
```

```
don't catch this cat that dog this dog for me.
catch this that cat dog cat cat.
this cat catch that cat this dog; now catch thisthat catdog for you.
```

Reference table for Regular expression symbols:

|	or
*	0 or more times
+	1 or more times
?	optional (0 or 1 time)
[. . .]	set of characters (can use - for a range)
(. . .)	grouping (for | or for repetition)
\s	space
\d	any digit
\w	any word character (letter or digit)
\b	a word boundary

(Question 1 continued on next page)

(Question 1 continued)

(c) **[4 marks]**

Write a regular expression that would match simple variable declarations that have an initialisation in Java programs: a type, a variable, and an = sign.

For example, it should match the underlined sections in the following lines of code, but not any other parts.

```
int size = 15;
size = area * height;
Edge edge=fringe.poll();
Stop firstStop;
(int param1, int param2)
Stop   stop1 =stop.neighbours().get(0);
Cat6Cable cable = cables.pop();
Train lastTrain   = station.getLastTrain();
```

Note:

- The regexp does **not** have to match variables whose type is an array or a collection type with type parameters.
- Take note of spaces.
- Assume that identifiers (types and variables) may only contain letters and digits and must not start with a digit.

```
\b[a-zA-Z]\w*\s+[a-zA-Z]\w*\s*=
```

Question 2. Adjacency Matrix Data Structures for Graphs. **[6 marks]**

Consider the following adjacency matrix data structure to represent an **undirected** graph with a length associated with each edge.

The value in edges[j][k] represents the length of the edge from node j to node k, or -1 to indicate there is no edge from node j to node k.

```
public class Graph {
    private String[ ] nodeNames;        // names of the nodes
    private double[ ][ ] edges;         // adjacency matrix of edge lengths
```

(a) **[2 marks]** Assume there are N nodes in the graph and at most Δ edges out of each node. What is the Big-O cost of finding the closest neighbour of node j? (ie, the node with the shortest edge from node j.)

$O(\quad N \quad)$

(b) **[4 marks]** Complete the following method that would count and return the number of neighbours of node fromIndex.

Hint: the number of columns in the fromIndex row of edges is edges[fromIndex].length.

```
public int countOutEdges (int fromIndex){
    int count = 0;
    for ( int toIndex=0; toIndex< edges[fromIndex].length; toIndex++){
        if (edges[fromIndex][toIndex] > -1){count++;}
    }
//OR
    for (double length : edges[fromIndex]){
        if (length != -1){count++;}
    }
    return count;
}
```

Question 3. Adjacency List Data Structures for Graphs. **[9 marks]**

Consider the following adjacency list data structure to represent a simple, **directed** graph with a length associated with each edge.

```
public class Graph{
   private String[ ] nodeNames;
   private List<Edge>[ ] outEdges;    // outEdges[indx] is a list of the edges from node indx
```

Edge objects have three methods:

```
   public int from();          // returns the index of the node at the start of the edge
   public int to();            // returns the index of the node at the end of the edge
   public double length();     // returns the length of the edge
```

(a) **[2 marks]** Assume there are N nodes in the graph and at most Δ edges out of each node. What is the Big-O cost of finding the shortest edge out of a given node?

O(Δ)

(b) **[3 marks]** Complete countOutEdges(...) to return the number of out-edges from node fromIndex

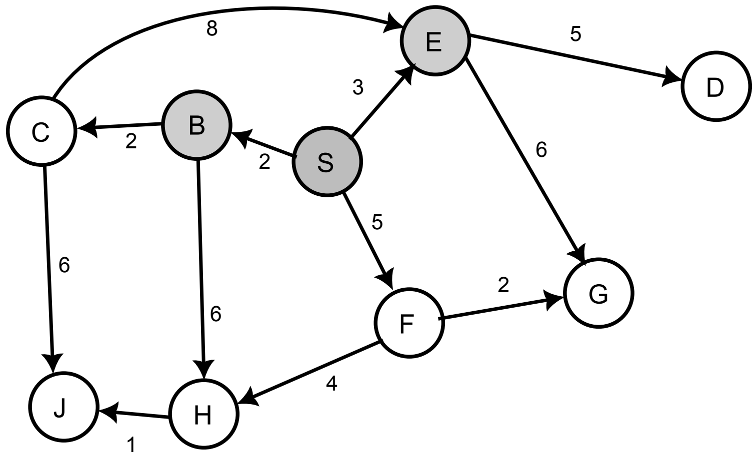
```
public int countOutEdges (int fromIndex){
    return outEdges[fromIndex].size();


}
```

(c) **[4 marks]** Complete countInEdges(...) to return the number of edges from other nodes to node toIndex.

```
public int countInEdges (int toIndex){
    int count = 0;
    for (int i=0; i<outEdges.length; i++){
        for (int j=0; j<outEdges[i].size(); j++){
            if (outEdges[i].get(j).to()==toIndex){count++;}
    }}
//OR
    for (List<Edge> edges : outEdges){
        for (Edge edge : edges){
            if (edge.to()==toIndex){count++;}
    }}
    return count;
}
```

Question 4. Shortest Paths. **[10 marks]**

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



After three iterations of the **while** loop:

- The algorithm will have visited three nodes **S**, **B** and **E**;
- The fringe will contain five items:
  ⟨F, S-F, 5⟩, ⟨H, B-H, 8⟩, ⟨C, B-C, 4⟩, ⟨D, E-D, 8⟩, ⟨G, E-G, 9⟩
- The backpointers Map will contain two items
  ⟨B → S-B⟩, ⟨E → S-E⟩

```
DjikstrasShortestPath ( start , goal ):
   fringe ← PriorityQueue of ⟨node, edge, length−to−node⟩
   backpointers ← Map of nodes to edges
   put ⟨ start , null , 0⟩ on the fringe .
   while fringe is not empty:
      ⟨node, edge, length−to−node⟩ ← remove from fringe
      if node is not visited :
         visit node
         put ⟨node, edge⟩ into backpointers
         if node=goal:
            return ReconstructPath( start , goal , backpointers )
         for each edge out of node to a neighbour:
            if neighbour is not visited :
               length−to−neighbour ← length−to−node + edge.length
               add ⟨neighbour, edge, length−to−neighbour⟩ to fringe
```

(Question 4 continued on next page)

(Question 4 continued)

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

- which node it will visit
- what will be added to the Backpointers
- what items will be added to the fringe

Show the shortest path that it finds.

Iteration 4:

Node visited: C

Additions to Backpointers Map: add $\langle$C $\rightarrow$ B-C$\rangle$

Additions to fringe: $\langle$J, C-J, 10$\rangle$ *(E is already visited)*

Iteration 5:

Node visited: F

Additions to Backpointers Map: add $\langle$F $\rightarrow$ S-F$\rangle$

Additions to fringe: $\langle$G, F-G, 7$\rangle$, $\langle$H, F-H, 9$\rangle$

Iteration 6:

Node visited: G

Additions to Backpointers Map: add $\langle$G $\rightarrow$ F-G$\rangle$

Additions to fringe: *None: G is the goal, so it returns the path (S-F, F-G)*

Shortest Path:

(S-F, F-G) or (S, F, G)

Question 5. Connected Components. **[10 marks]**

The following findComponents() method finds all the connected components in an **undirected** graph, labeling each node with the number of its component. It uses the visitComponent(...) method to label all the nodes connected to a given node.

Complete visitComponent(...) to do a recursive depth first traversal from the given node, labeling the connected nodes with the given component number.

Note:

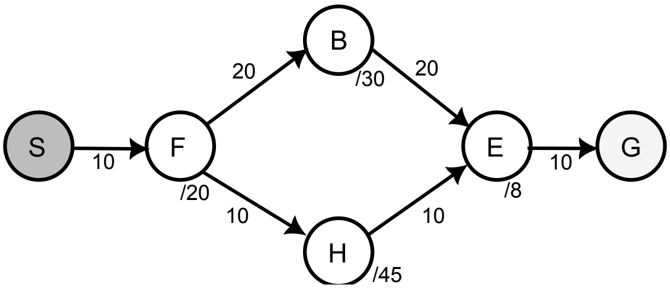
- You may assume that all nodes initially have a component number of -1;
- The relevant fields and methods of the Graph and Node classes are given at the bottom.
- You may (but do not have to) use the component number of the nodes to record whether a node has been visited.

```
public void findComponents(){
    int comptNum = 0;
    for (Node node : nodes){
        if (node.getCompt()== -1){
            visitComponent(node, comptNum);
            comptNum++;
        }
    }
}
public void visitComponent(Node node, int comptNum){
    if (node.getCompt()==-1){
        node.setCompt(comptNum);
        for (Node neighbour : node.getNeighbours()){
            visitComponent(neighbour, comptNum);
    }}
//OR      (checking visited _before_ calling visitComponent)
    node.setCompt(comptNum);
    for (Node neighbour : node.getNeighbours()){
        if (neighbour.getCompt()==-1){
}           visitComponent(neighbour, comptNum);
    }}
```

```
public class Graph {
    private Collection<Node> nodes;
    :
    public void findComponents(){...}
}

public class Node {
    private String name;
    private List<Node> neighbours; // neighour nodes, connected to this node by an edge
    private int component = -1;    // the number of the component it belongs to

    public List<Node> getNeighbours(){ return Collections.unmodifiableList(neighbours); }
    public int getCompt(){ return component; }
    public void setCompt(int c){ component = c; }
}
```

Question 6. A∗ Search. **[5 marks]**

To be admissable, an estimate of the remaining path length must not be an overestimate. Explain why admissability is important for A* search using the following example graph. Note that an estimate of the remaining distance to the goal node is shown beside each node.



If a node has an overestimate of the remaining distance, then it may not be removed from the fringe and visited, even if it is actually on the shortest path.
This can block A* from finding the shortest path, and make it return the wrong path.

In the example, the overestimate on the remaining distance from H blocks the search from finding the path through H:
When H and B are added to the fringe, H will have an estimated total path of 65, which is more than the estimated total path for B (60), so that B will be visited before H.
Visiting B will put E on the fringe (from B). with an estimated path of 58 so E will be visited before H.
Visiting E will put G on the fringe (from E) with a total path length of 60.
Since this is also a lower cost than the estimate for H, the search will visit G (from E, from B...).
Since G is the goal, it will return the path S-F-B-E-G (cost 60) instead of the shortest path S-F-H-E-G (cost 40).

The overestimate on H blocked the shorter path from being explored.
If the estimate on H had been an underestimate, then it would have visited H Before visiting the goal, even if the estimate on B had initially made B look more promising than H.

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

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.