Problem 1. [10 points]
Consider the different graph representations discussed in class, and fill out the table below. In the following
table, show the worst-case running times for the best algorithms for implementing the given operations.
Justify your answer by writing a 2 sentence (maximum) explanation of how the operation would be performed.

Note: Be careful what data structures you will be using for each implementation.

Solution.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Adjacency Matrix</th>
<th>Adjacency List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a vertex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove a vertex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find a vertex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add an edge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove an edge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find an edge</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Problem 2. [5 points] - Proof problem
Some friends of yours work on wireless networks, and they’re currently studying the properties of a network
of n mobile devices. As the devices move around (actually, as their human owners move around), they define
a graph at any point in time as follows: there is a node representing each of the n devices, and there is an
edge between device i and device j if the physical locations of i and j are no more than 500 meters apart.
(If so, we say that i and j are “in range” of each other.)

They’d like it to be the case that the network of devices is connected at all times, and so they’ve constrained
the motion of the devices to satisfy the following property: at all times, each device i is within 500 meters
of at least n/2 of the other devices. (We’ll assume n is an even number.)

What they’d like to know is: Does this property by itself guarantee that the network will remain connected?

Here’s a concrete way to formulate the question as a claim about graphs.

Claim: Let G be a graph on n nodes, where n is an even number. If every node of G has degree at least
n/2, then G is connected.

(a) Decide whether you think the claim is true or false.
(b) Give a counter-example if it is false, or write a clear proof on why it is true.

Problem 3. [10 points]
Consider a version of the stable matching problem where there are n men and n women as before. Assume
each man ranks the women (and vice versa), but now we allow ties in the ranking. In other words, we could
have a man that is indifferent to two women \(w_1\) and \(w_2\), but prefers either of them over some other woman \(w_3\) (and vice versa). We say a woman \(w\) prefers man \(m_1\) to \(m_2\) if \(m_1\) is ranked higher on the \(w\)’s preference list and \(m_1\) and \(m_2\) are not tied.

With indifferences in the rankings, there could be two natural notions for stability. And for each, we can ask about the existence of stable matchings, as follows.

(a) **Strong Instability.** A strong instability in a perfect matching, \(S\), consists of a man \(m\) and a woman \(w\), such that each of \(m\) and \(w\) prefers the other to their partner in \(S\). Does there always exist a perfect matching with no strong instability? Either give an example of a set of men and women with preference lists for which every perfect matching has a strong instability; or give an algorithm that is guaranteed to find a perfect matching with no strong instability.

(b) **Weak Instability.** A weak instability in a perfect matching, \(S\), consists of a man \(m\) and a woman \(w\), such that their partners in \(S\) are \(w’\) and \(m’\), respectively, and one of the following holds:

* \(m\) prefers \(w\) to \(w’\), and \(w\) either prefers \(m\) to \(m’\) or is indifferent between these two choices; or
* \(w\) prefers \(m\) to \(m’\), and \(m\) either prefers \(w\) to \(w’\) or is indifferent between these two choices.

In other words, the pairing between \(m\) and \(w\) is either preferred by both, or preferred by one while the other is indifferent. Does there always exist a perfect matching with no weak instability? Either give an example of a set of men and women with preference lists for which every perfect matching has a weak instability; or give an algorithm that is guaranteed to find a perfect matching with no weak instability.

Problem 4. [10 points]
Consider the BFS implementation on pp. 90-91. When creating the list \(L[i + 1]\), the algorithm has the following line:

\[
\text{For each node } u \text{ from 2 to } L[i]
\]

This line does not specify the order in which the nodes of \(L[i]\) should be considered.

(a) Argue that the layer assignments are independent of the specific order chosen in this line of the algorithm. In other words, the set \(L[0]; L[1]; \ldots\) do not depend on the order.

(b) Show by example that the BFS tree \(T\) returned by the algorithm can depend on the particular order chosen in this line.

Problem 5. [5 points]
Remember problem 6 from the last assignment, in which you used arrays to implement the different high-level operations in the Gale-Shapley algorithm presented in Chapter 1.

(a) Rewrite the algorithm, in pseudocode form. In other words, you should remove any details about types.

(b) Show that the worst case running time complexity of the algorithm is indeed \(O(n^2)\).

Problem 6. [10 points - Coding problem]
There’s a natural intuition that two nodes that are far apart in a communication network—separated by many hops—have a more tenuous connection than two nodes that are close together. There are a number of algorithmic results that are based to some extent on different ways of making this notion precise. Here’s one that involves the susceptibility of paths to the deletion of nodes.

Suppose that an \(n\)-node undirected graph \(G = (V, E)\) contains two nodes \(s\) and \(t\) such that the distance between \(s\) and \(t\) is strictly greater than \(n/2\).

(a) - Submit this with the rest of the assignment. Prove that there must exist some node \(v\), not equal to either \(s\) or \(t\), such that deleting \(v\) from \(G\) destroys all \(s – t\) paths. (In other words, explain that the graph obtained from \(G\) by deleting \(v\) contains no path from \(s\) to \(t\).)
(b) - Coding part to be submitted through the Google form. Write a Java function to implement an algorithm with running time $O(m + n)$ to find such a node $v$.

Your function should have the following definition:

```java
public int FindConnectorNode(String fileName, int s, int t)
```

**Input Types:**
Your input is a String, which represents the name of a txt file that contains the input graph, represented as an adjacency list. Also, the indices of $s$, and $t$ are given as part of the input.

The graph file contains the input graph, in the form of an adjacency list. The first line in that file is the number of vertices in the graph, and then each line represents the adjacency list of each node. An example graph is as follows:

```
6
1,2
0,3
0,3
1,2,4
3,5
4
```

This means that vertex 0 is adjacent to vertices 1 and 2, and that vertex 1 is adjacent to vertices 0 and 3, etc. Note that we are using a 0-base because you’ll be using Java. Also, assume all inputs are valid, and that the two given nodes $s$ and $t$ are at a distance that is strictly greater than $n/2$.

**Output Types:**
The output format is a single int value representing the index of that connector node $v$, as defined in the problem above.

**Example:**
For the graph shown in the file above, the method invocation,

```java
public int FindConnectorNode("graphExample.txt", 0, 5)
```

will produce an output of 3 or 4.