9.1P Problem Solving: Graphs

1. Given a graph $G = \langle V, E \rangle$, what is to be the running time of the ***depth-first search*** algorithm, as a function of the number of nodes $n = |V|$ and edges $m = |E|$, if the input graph is represented by an ***adjacency matrix*** instead of an *adjacency list*?

The running time of the depth-first-search algorithm if the input of graph is represented by an adjacency matrix is V x V to traverse through, so the run time complexity is $O(N^2)$

V rows and V columns to find the edge for the node.

For the adjacency list it will be traversed through with the run time complexity of O(N+M)

N == sum of Vertices or Nodes

M == sum of Edges

2. After starting your first programming position your colleague asks you for advice on which algorithm they should choose. While your colleague studied programming until SIT232, they did not do SIT221 because they thought it would be too hard, therefore they have no idea how to decide on which algorithm to use. Because you are generous you decide to offer some advice.

   Their problem requires them to find the shortest path from where a user is currently located to each of the major tourist locations in the city. To solve this your colleague had googled for an algorithm and came across two: Dijkstra's algorithm, which will solve it in $O(V + E\ log\ V)$, and Floyd's algorithm, which will take $O(V^3)$, where $|E| \leq |V|^2$. While you recall studying Dijkstra's algorithm during your SIT221 unit you have never heard of Floyd's algorithm. Regardless because your colleague's Google search had also given the time complexity you feel equipped to provide advice.

   Discuss what factors your colleague should consider when deciding between the two algorithms and in which circumstances, they should choose one over the other.

Dijkstra's algorithm:

Pros:

1. If it was in a fixed position (single-source shortest), then Dijkstra's algorithm would be quicker then Floyd's algorithm
2. Faster initial runtime complexity (Depending on the data structure and required task at hand`)

Cons:

1. Although we can use Dijkstra's algorithm to find all the pairs of the shortest paths it would require running it for every vertex, this can result in a $O(V^3)$ which is the same as Floyd's $O(V^3)$. Since it is known that floyds algorithm is easier to implement floyds would be the best choice
2. Cannot be implemented in a distributed system
3. Doesn't work for negative weighted edge
4. Memory overhead is significantly higher then Floyd's

Floyd's algorithm:

Pros:

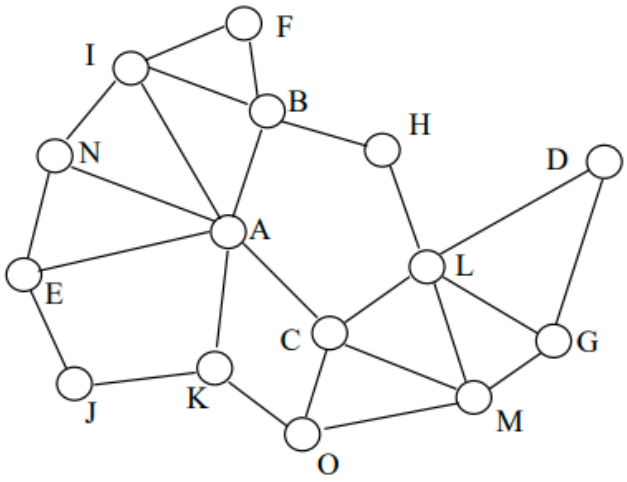
1. Can be used for distributed systems
2. Can be used graph of graphs such as maps, including negative edges
3. Easier to implement/code then Dijkstra's algorithm
4. Best to find all pair shortest path then Dijkstra's since it has the same run time complexity and is easier to implement

Cons:

1. When finding the single source shortest path, Floyd's algorithm is substantially more expensive then using Dijkstra's.

If the user is in a fixed position (single source) then Dijkstra's algorithm would be most appropriate, however if the user wanted to check the distance from different positions, then Floyd's algorithm would be best to implement.

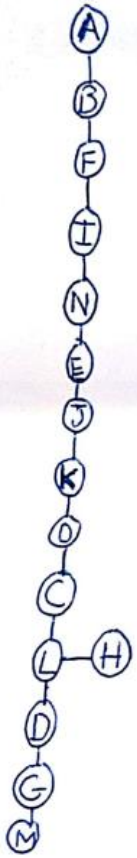
3. Given the following graph draw both the Depth-first and Breadth-first search trees that will result if you start at Node A. *Note: when selecting between nodes you should use alphabetical order.*



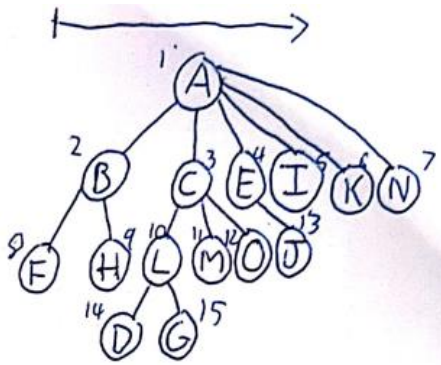
(ANSWERS FOR Q3 ON NEXT PAGE BELOW)

DFS)

A-B-F-I-N-E-J-K-O-C-L
↳ D-G-M-H



BFS)



A-B-C-E-I-K-N-F-H-L-M-O-J-D-G