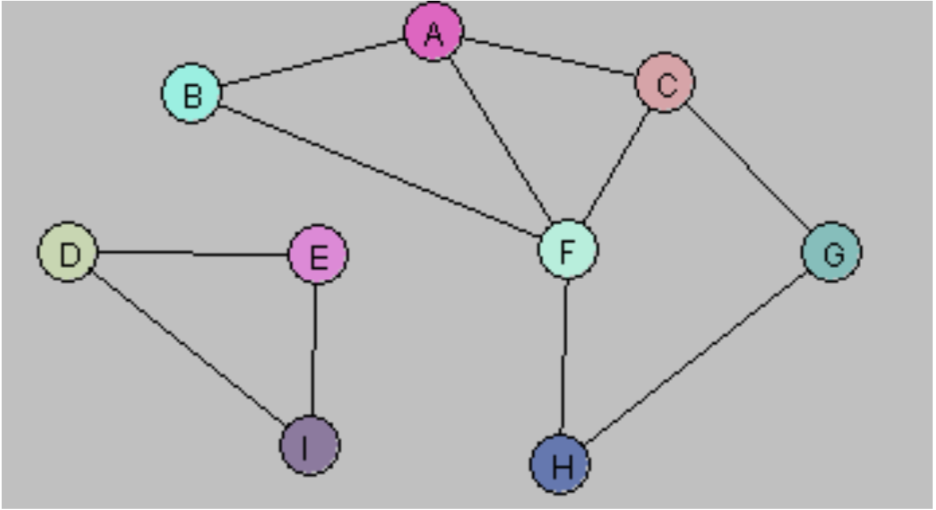
**LAB 11**

**Jimmy Palma**

**610756**

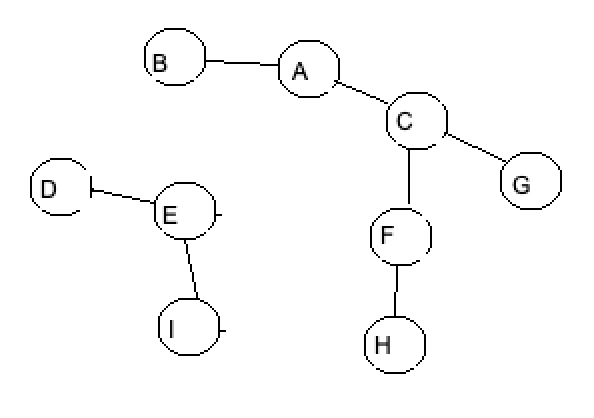
1. **Answer questions about the graph G = (V,E) displayed below.**



1. **Is the graph G connected? If not, what are the connected components for G?**

It is not connected. It has two components

1. **Draw a spanning tree/forest for G.**



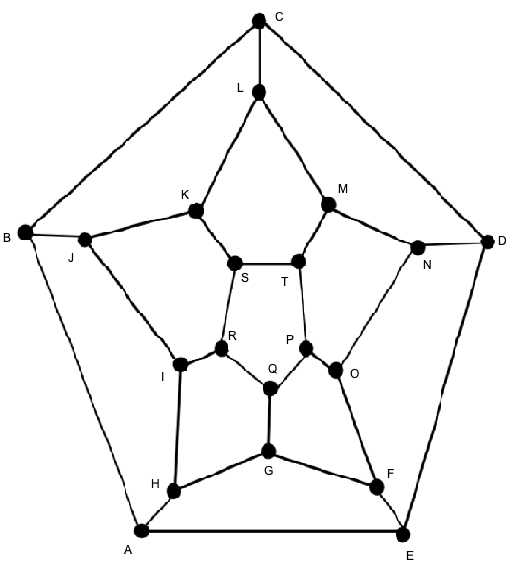
1. **Is G a Hamiltonian graph?**

No it is not

1. **Is there a Vertex Cover of size less than or equal to 5 for G? If so, what is the Vertex Cover?**

Yes, C = {D, E, F, A, G}

1. ***Hamiltonian Graphs.* The following graph has a Hamiltonian cycle. Find it.**



A B C D E F O N M L K J I R S T P Q G H A

1. ***Vertex Covers.* Create an algorithm for computing the smallest size of a vertex cover for a graph. The input of your algorithm is a set V of vertices along with a set E of edges. Assume you have the following functions available (no need to implement these):** 
   1. **computeEndpoints(edge) – returns the vertices that are at the endpoints of the input edge**
   2. **belongsTo(vertex, set) – returns true if the input vertex is a member of the given set**

***Hint:* Loop through all subsets of V. For each subset W, check to see if W is a vertex cover. Do this by looping through all edges; for each edge e, check to see if at least one of its endpoints lies in W.**

Algorithm: SmallestVertexCover  
 Input: set V of vertices along with a set E of edges

Output: Smallest size of a vertex cover U for G

pow ← PowerSet(V) minCover ← V  
 minVal ← |V|  
 for each U in pow do

isCover ← true  
 for each e in E do

(u,v) ← computeEndpoints(e)  
 if( !(belongsTo(u,U) and !belongsTo(v,U))

isCover ← false

End for

if(isCover and U.size() < minCover.size()) then

minCover ← U

minVal ← |U|

End for

return minVal

1. ***Graph Implementation.* Use the BFS class to solve the following problems. Implement by implementing the unimplemented methods in the Graph class.**
   * **Given two vertices, is there a path that joins them?**
   * **Is the graph conected? If not, how many connected components does it have?**
   * **Does the graph contains a cycle?**

The main graph:

Vertices:

A B C D F E

Edges:

A-B A-C B-C C-D F-E

Are B and C adjacent? true

Are A and C adjacent? true

The spanning tree/forest:

Vertices:

B A C D E F

Edges:

B-A A-C C-D E-F

Is the graph connected?

false

Does the graph have cycle?

false

Is there a path?

false

true

1. **Implement a subclass ShortestPathLength of BreadthFirstSearch that will provide, for any two vertices x, y in a graph G, the length of the shortest path from x to y in G. (You can assume G is connected.) Use the ideas mentioned in the slides for your implementation. Be sure to add a method of the Graph class having the following signature:**

**int shortestPathLength(Vertex u, Vertex v)**

**which will make use of your new subclass.**