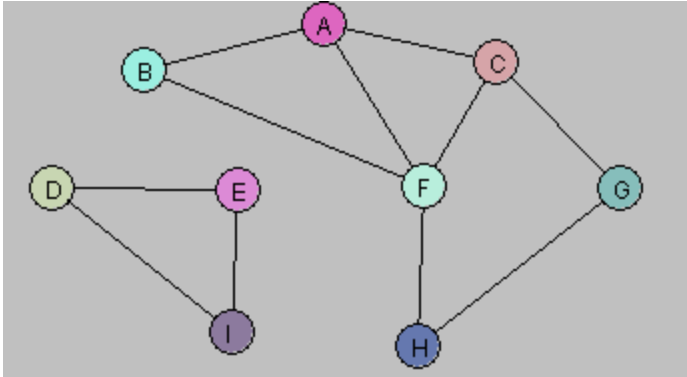


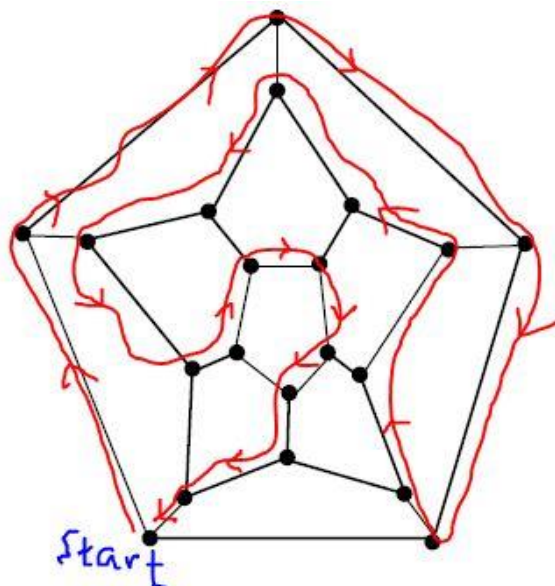
Lab 11 Solutions

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



- A. Is the graph G connected? If not, what are the connected components for G ?
 Solution: G is not connected. It has two connected components...
- B. Draw a spanning tree/forest for G .
 Solution: $T = \{DE, EI, FB, FA, FC, FH, GH\}$
- C. Is G a Hamiltonian graph?
 Solution: No, it has no Hamiltonian Cycle.
- D. Is there a Vertex Cover of size less than or equal to 5 for G ? If so, what is the Vertex Cover?
 Solution: Yes. $C = \{D, E, F, A, G\}$

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



3. Express in pseudo-code an algorithm which accepts as input a graph G and which outputs a vertex cover for G of smallest possible size. You may make use of the PowerSet algorithm without showing any pseudo-code details indicating how it works. Also, you may assume that your algorithm can make use of these operations freely:

`computeEndpoints(e)` //returns the two endpoints of the edge e

`belongsTo(x, U)` // returns true if vertex x belongs to set U ; false otherwise

Follow the rules for the pseudo-code language as completely as possible.

Solution:

Algorithm: SmallestVertexCover

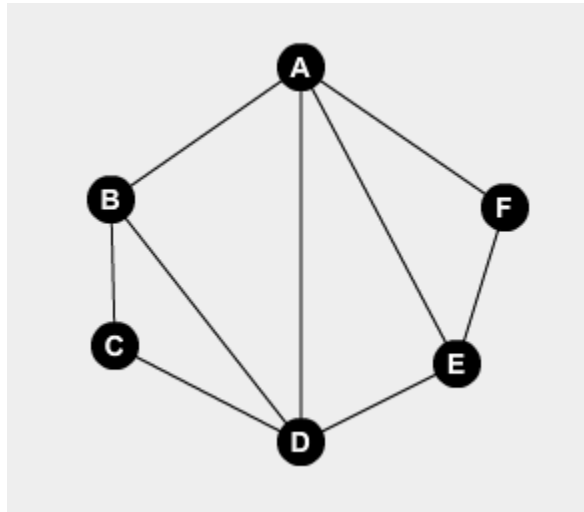
Input: A graph G whose set of vertices is denoted V and set of edges is denoted E

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
    //verify U is a vertex cover
    for each e in E do
        (u,v) ← computeEndpoints(e)
        if ( !belongsTo(u,U) and !belongsTo(v,U) )
            isCover ← false

    if(isCover and U.size() < minCover.size()) then
        minCover ← U
        minVal ← |U|
return minVal
```

4. Compute two spanning trees for the graphs below using algorithms we discuss in class. (You can start with vertex A) Are the two spanning trees same?



Solution:

Using DFS starting with A: AB, BC, CD, DE, EF

Using BFS starting with A: AB, AD, AE, AF, BC

The two spanning trees are not same.

5. Write the pseudo-code for compute connected components algorithm discussed in class. Your algorithm can be built on top of DFS discussed in the slides.

Solution:

Make a ConnectedComponentSearch subclass of DFS

Initialize ArrayList<List<Vertex>> componentMap;

Initialize HashMap<Vertex,Integer> vertexComponentMap;

CurrentComponentNumber \leftarrow 0

Algorithm: additionalProcessing
currentComponentNumber++

Algorithm: processVertex(v)
vertexComponentMap.put(v, currentComponentNumber)
componentMap.get(currentComponentNumber).add(v)

Algorithm: computeConnectedComponents

```
//start DFS
start()
Graph[] components ← new Graph[currentComponentNumber];
for i ← 0 to currentComponentNumber do
    // For each component i, we get a list of vertices for that component
    List<Vertex> vlist ← componentMap.get(i)
    List<Edge> elist ← new ArrayList<Edge>()
    foreach Vertex v in vlist
        List<Vertex> adjList ← adjacencyList.get(v);
        foreach Vertex u in adjList
            Edge e ← new Edge(v, u))
            if e not yet in elist then
                elist.add(e)
        components[i] ← new Graph(elist)
return components
```

6. Write the pseudo-code for the algorithm, discussed in class, that computes the shortest path length between two vertices in a graph. You can assume that:
- The graph is connected.
 - A version of BFS is provided that accepts a specified starting vertex.

Solution:

Make ShortestPath a subclass of BFS

Initialize HashMap<Vertex, Integer> levelsMap

Initialize HashMap<Vertex, Vertex> parentMap

Algorithm processEdge(Vertex v, Vertex w)

```
//v is the parent, and w is the child
parentMap.put(w, v);
```

Algorithm processVertex(Vertex v)

```
parentVertex ← parentMap.get(v)
if parentVertex is null then //v is the starting vertex
    parentMap.put(v, null)
    levelsMap.put(v, 0)
else
    plevel ← levelsMap.get(parentVertex)
    levelsMap.put(v, plevel + 1)
```

Algorithm computeShortestPathLength(Vertex s, Vertex v)

```
//start BFS with starting vertex s
start(s)
//now levels and parents have been computed
return levelsMap.get(v)
```