Yuliang Jin 986381

Problem 1

A. Is the graph G connected? If not, what are the connected components for G? No, connected components are: (A, B, C, F, H, G),(D, E, I)

B. Draw a spanning tree/forest for G

C. Is G a Hamiltonian graph? No, because there is no Hamiltonian cycle in this graph.

D. Is there a Vertex Cover of size less than or equal to 5 for G? If so, what is the Vertex Cover? Yes, the Vertex cover is (D, E, F, A, G).

Problem 2

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

Problem 3

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): • computeEndpoints(edge) – returns the vertices that are at the endpoints of the input edge • belongsTo(vertex, set) – returns true if the input vertex is a member of the given set

Algorithm: compute Vertex Cover

Input: G=(V, E)

Output: a vertex cover with smallest size of vertices

initialize a vertext List,

while (E.size > 0). do

compute the vertex appears most times in all E,

add the vertext to the List,

remove all edges with the vertex in the edge.

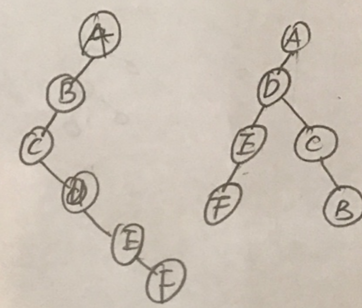
return List;

Problem 4

f spanning trees., they are not the same.

{(a,b), (b,c), (c,d), (d,e), (e,f)}

{(a,d), (d,e), (e,f), (d,c), (c,b)}



Problem 5

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

F

5. Algorithm: computeConnectedComponents
Input: a graph G=(V,E),
Output: a List of component
Define a List of Vertex,
While (there is vertex not visited)
SingleComponentLoop, (add all reachable vertex to the component

return List;

Problem 6

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

6.

Algorithm: findShortestPath

Input: Two vertices in a $G=(V,E)$,

Output: List of vertices from either v_1 to v_2 or v_2 to v_1.

```
choose v1 or v2 as root, a List, a isFound flag.
Initialize a queue Q, enqueue the root;
while (! isFound) {
      n = Q.size();
      Create a List of vertices as paths.

      for( i=1, to n)
         V. v = dequeue()
         mark v as visited;
         if (v equals v2) {
                      return list[i];

}
```