Assignment 14a-Shortest Path

R-7.1 Draw a simple, connected, undirected, weighted graph with 8 vertices and 16 edges, each with unique edge weights. Identify one vertex as a "start" vertex and illustrate a running of Dijkstra's shortest path algorithm on this graph.

Assignment 14b – Minimum Spanning Tree

R-7-8 Draw a simple, connected, undirected, weighted graph with 8 vertices and 16 edges, each with unique edge weights. Illustrate the execution of Prim-Jarvik's algorithm on this graph. (Note there is only one minimum spanning tree for this graph.)

R-7-9 Repeat the previous problem for Baruvka's algorithm.

1. Consider the following potential MST algorithms based on the generic MST algorithm. Which, if any, successfully computes a MST? **Hint**: to show that an algorithm does not compute an MST, all you need to do is find a counterexample. If it does, you need to argue why based on the cycle property and/or the partition property.

a.
```
Algorithm MST-a(G, w)
    T ← edges in E sorted in nonincreasing order of edge weights
    for each e ∈ T do {each e is taken in nonincreasing order by weight }
        if T – {e} is a connected graph then
            T ← T – {e}     {remove e from T}

    return T
```

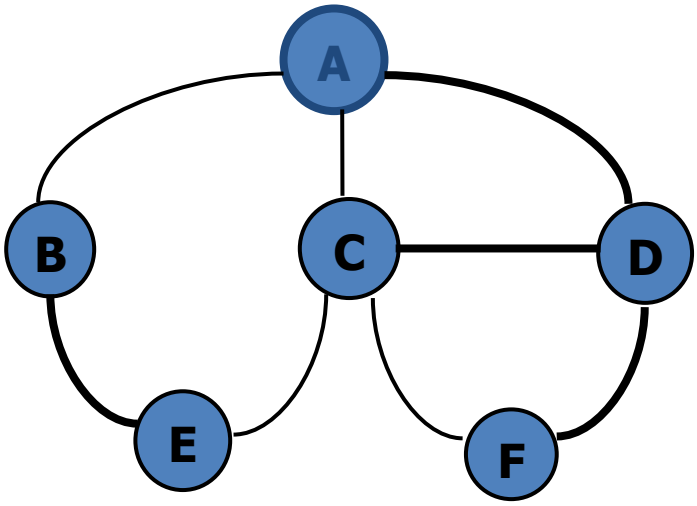
b.
```
Algorithm MST-b(G, w)
    T ← { }
    for each e ∈ E do { e is taken in arbitrary order }
        if T ∪ {e} has no cycles then
            T ← T ∪ {e}     {add e to T}

    return T
```

c.
```
Algorithm MST-c(G, w)
    T ← { }
    for each e ∈ E do { e is taken in arbitrary order }
        T ← T ∪ {e}     {add e to T}
        if T now has a cycle C then
            if e' is the edge of C with the maximum weight then
                T ← T − {e'}     {remove e' to T}
    return T
```

A. Using your BFS template from yesterday's assignment, override the hook methods so it calculates the number of edges in each path from the starting vertex to the other vertices in the tree calculated by BFS (you calculated paths in yesterday's assignment). For example, if a vertex *w* is 2 edges away from the starting vertex *v*, then 2 would be stored at *w*. This is similar to Dijkstra's algorithm for calculating distance except that distance is the number of edges (as if the edge weight is 1 for ever edge). Note also that this distance is the level number that we showed in the slides. See the diagram in the notes at the end of Dijkstra's shortest paths.

B. Given a graph **G=(V,E)** and a Sequence **T** where T⊆E, design a pseudo code algorithm **isSubtree(G, T)** that determines whether or not the edges in Sequence T form a tree (not necessarily a spanning tree). If the edges in T form a tree, then return true, otherwise return false. For example, in the graph below, if T={(A,C),(C,F),(D,F)}, then isSubtree would return true since these edges connect the vertices {A,C,D,F} and do not form a cycle. However, if T={(A,B),(C,F),(D,F)}, then **isSubtree** would return false since those edges do not form a tree (the vertices are not connected). Similarly, if T={(A,C),(C,D),(D,A)}, then **isSubtree** would also return false since T contains a cycle. To receive full credit, you must use the BFS Template with no unnecessary loops outside the Template. **Hint**: note that T is a **Sequence**, NOT a graph, i.e., use labels on edges and vertices like we did in other homework problems. Note also that the edges in T do not necessarily reference all vertices, so the subgraph may be a tree, but not be a spanning tree as in the first example.



C-5.1 A native Australian named Anatjari wishes to cross a desert carrying only a single water bottle. He has a map that marks all the watering holes along the way. Assuming he can walk *k* miles on one bottle of water, design an efficient algorithm for determining where Anatjari should refill his bottle in order to make as few stops possible. Argue why your algorithm is correct. Do this with two different assumptions:

1. Assume the watering holes are all located along the same road/path
2. Assume the watering holes are spread over the whole desert, i.e., there is no road.

Optional: If you have time.

R-7.7 Repeat the previous problem R-7.8 for Kruskal's algorithm.