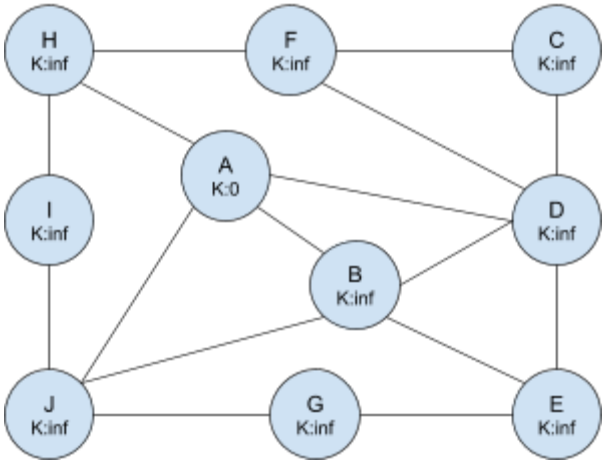
Brandon Swanson
CS 325 HW 3
11/8/15

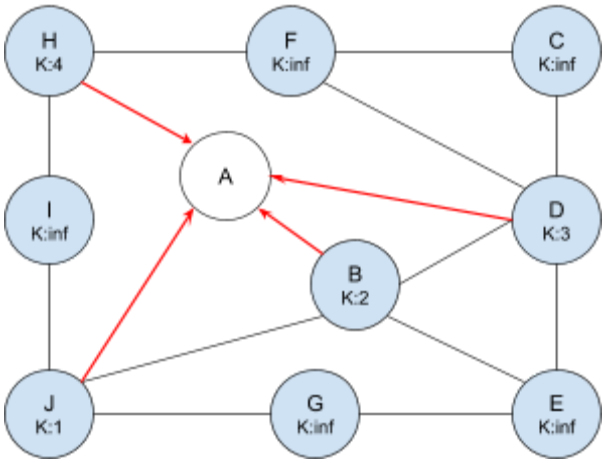
1) Prim's algorithm

Vertices colored blue are in the Que. v.key labeled inside circle with k: # or inf for ∞.
Red arrowed lines point towards a vertex's predecessor with the red arrow pointing to v.π
Each image shows the graph at the end of the While Q not empty loop

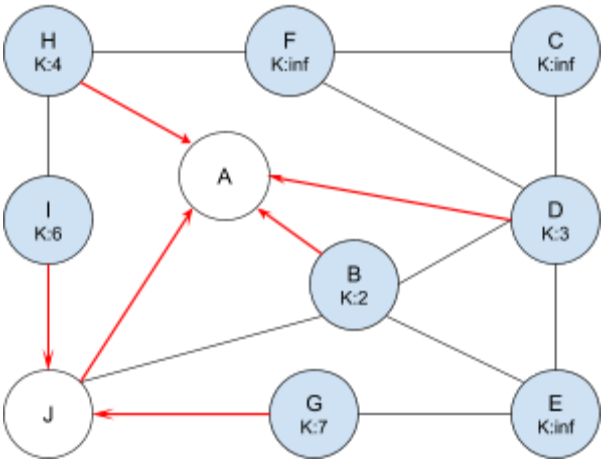
Initialization step



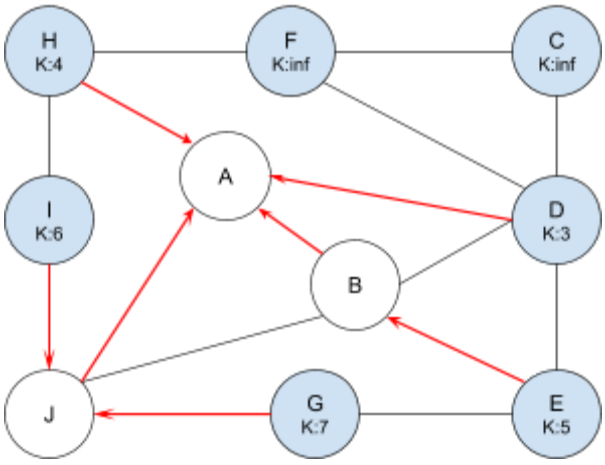
Iteration #1



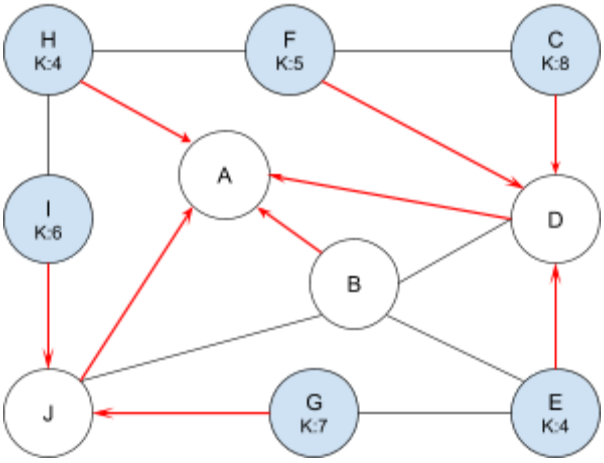
Iteration #2



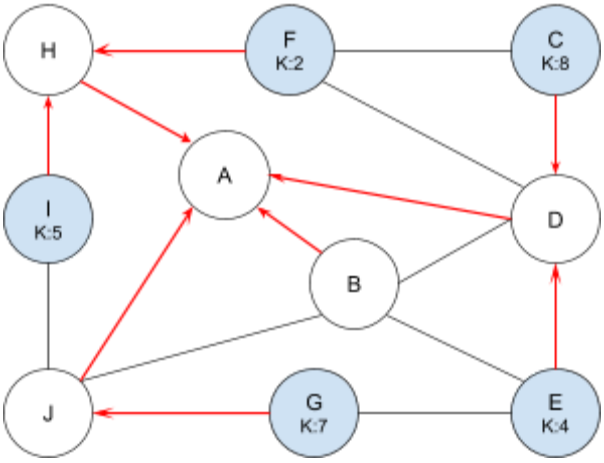
Iteration #3



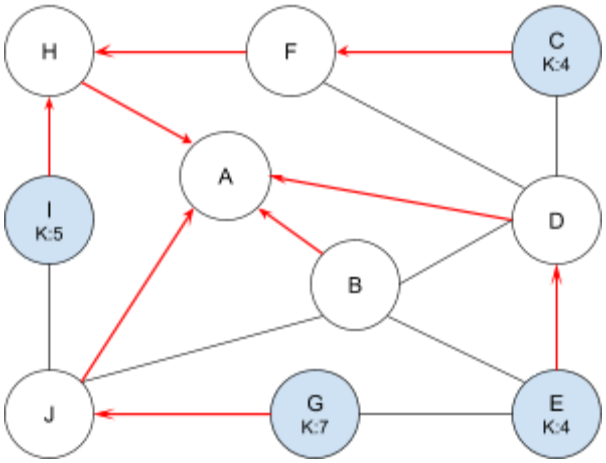
Iteration #4



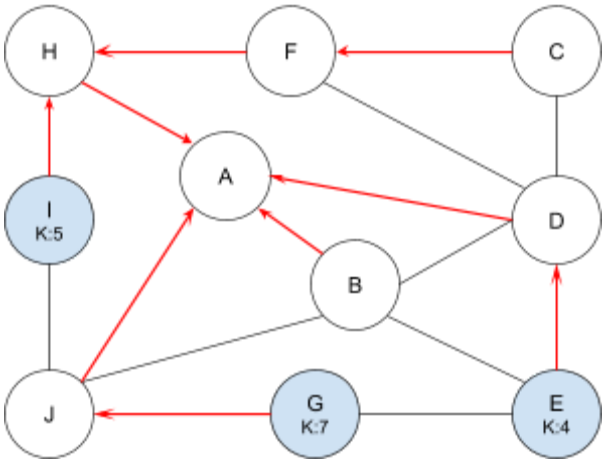
Iteration #5



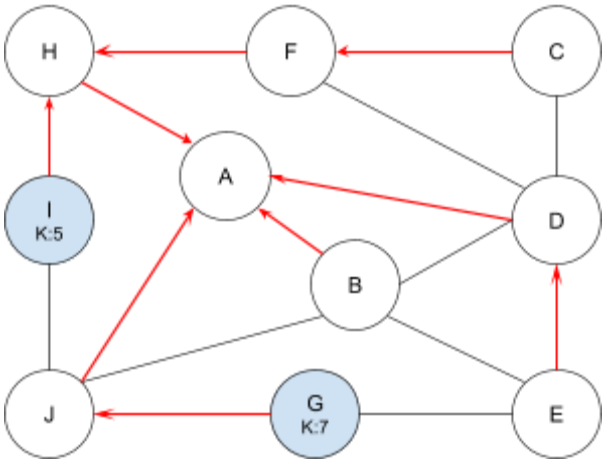
Iteration #6



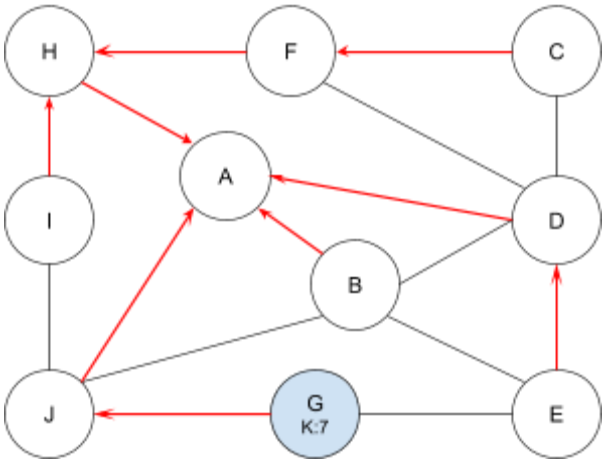
Iteration #7

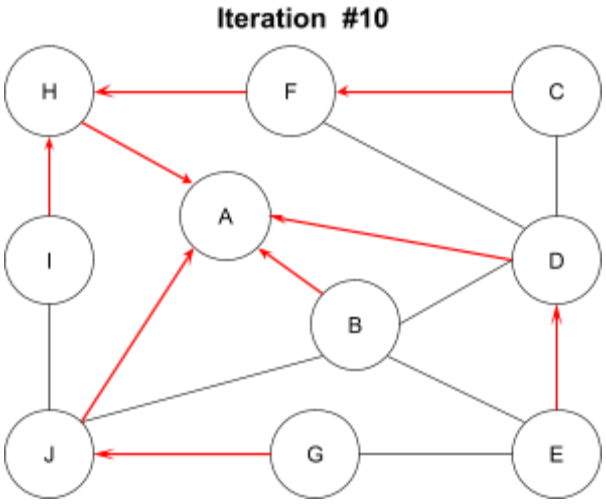


Iteration #8



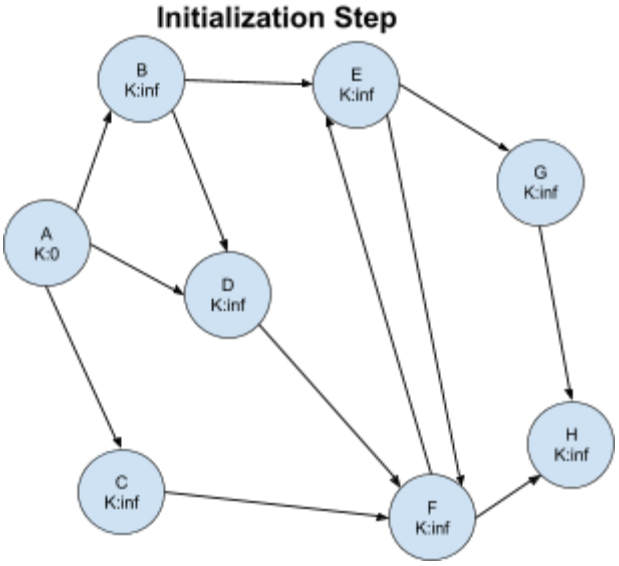
Iteration #9

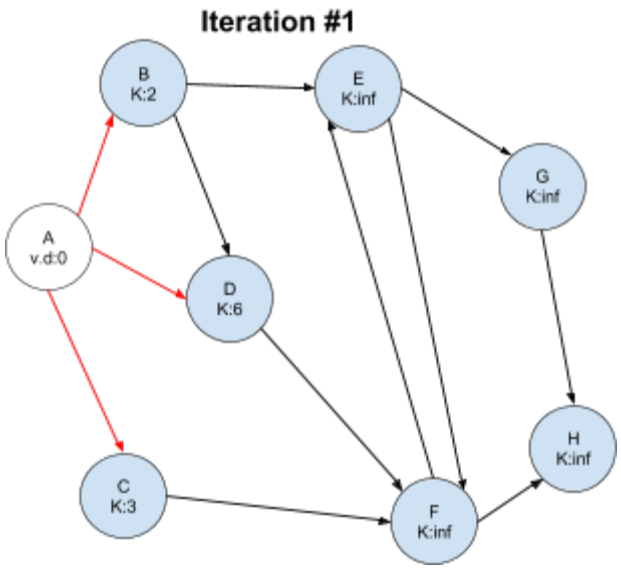




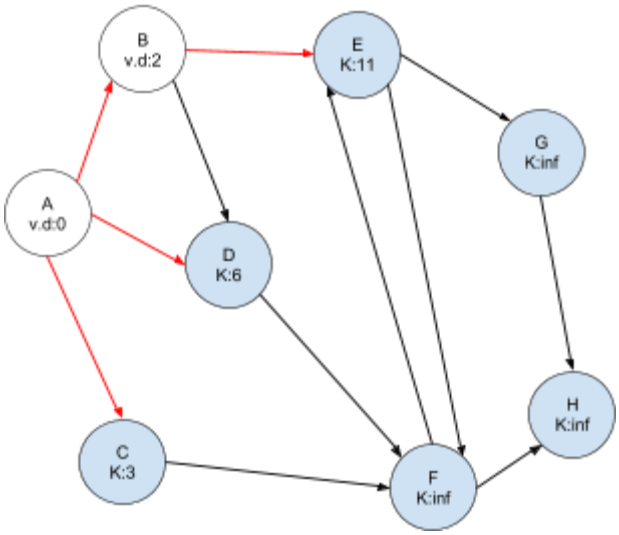
2) Dijkstra's algorithm demonstration

Vertices colored blue are in the Que, White Vertices are in the set S of computed vertices. v.key labeled inside circle with k: # or inf for ∞. Red lines represent a vertex's predecessor with the red arrow coming from a predecessor. Each image shows the graph at the end of the While Q not empty loop

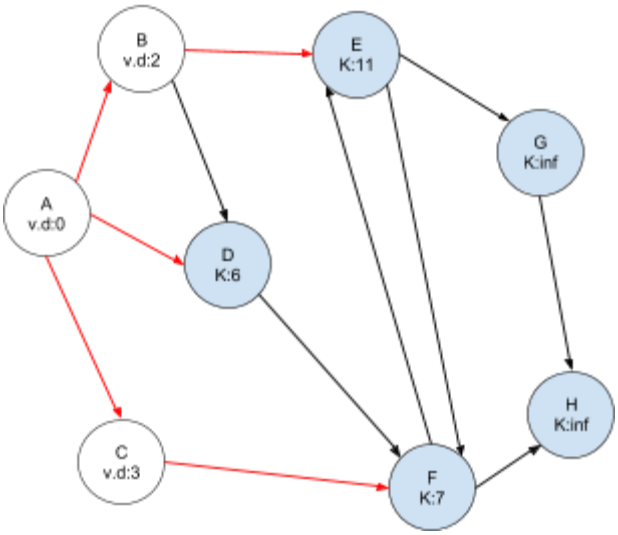




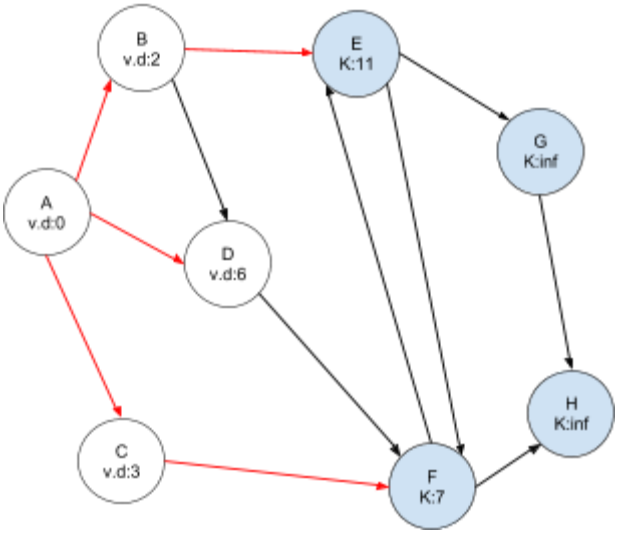
Iteration #2



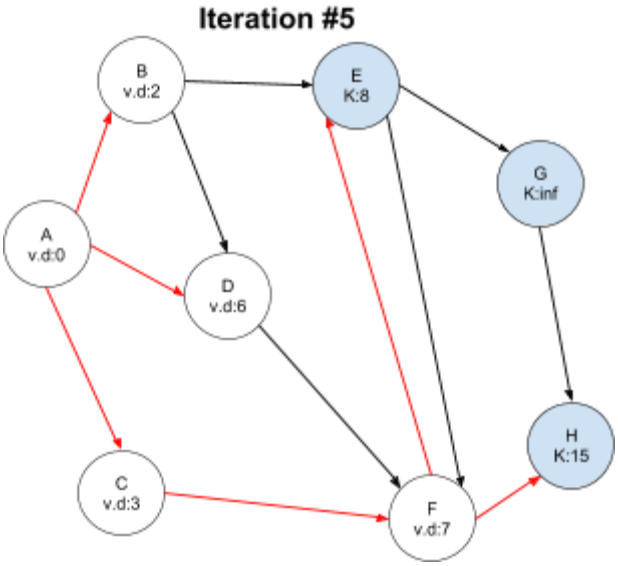
Iteration #3



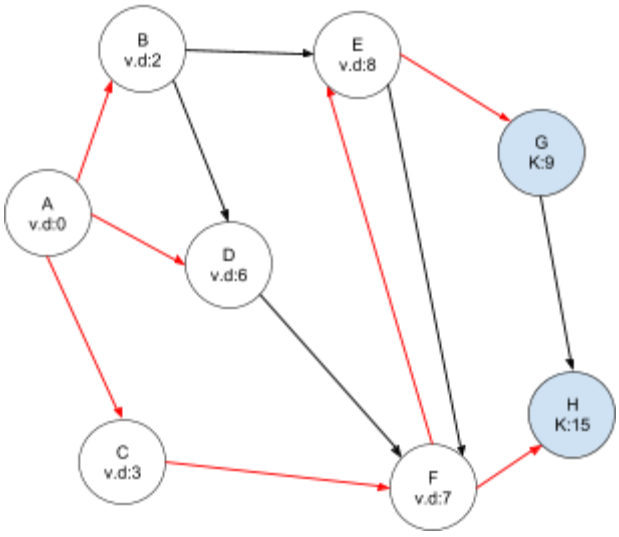
Iteration #4



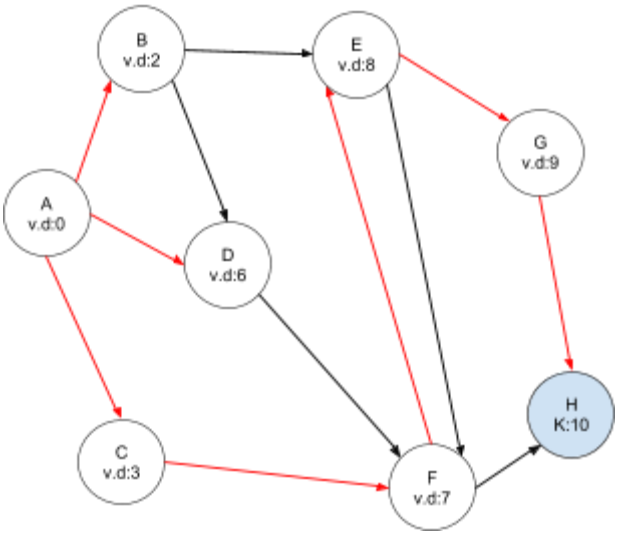
Iteration #5

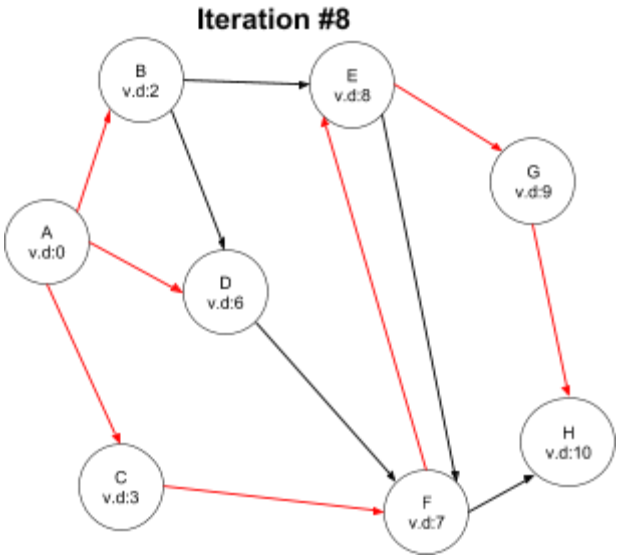


Iteration #6



Iteration #7

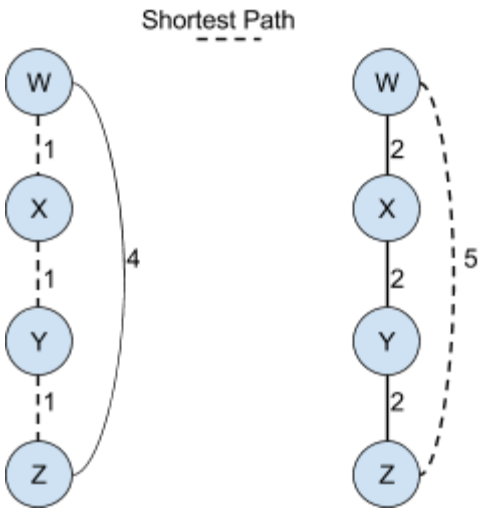




3)

a) No the minimum spanning tree of the graph would not ever need to change, a minimum spanning tree of the graph will always still be a minimum spanning tree of this new graph with each edge weight incremented by 1. If it were no longer an MST of the graph there would need to be a new edge added to the minimum spanning tree that crosses some cut of the graph that replaces an edge from the old minimum spanning tree that crosses the same cut and thies replacing edge would therefore need to have a smaller weight than the replaced edge.
There cannot exist such a pair of edges where W(x,y) < W(a,b) and W(x,y)+1 >= W(a,b) + 1

b) The shortest paths computed from source vertex could change, We will prove this by counter example.
Let us consider a shortest path S(w,z), this could be the path from source to a vertex or a sub path along a shortest path, If it can be shown that it is possible for this path to no longer be the shortest path from w to z after the weights of all edges have increased by one then the shortest paths of this new graph may change.
If the path S(w,z) is made up of 3 edges (w,x), (x,y), and (y,z) each with a weight of 1, and there is only one other path from w to z and edge (w,z) with a weight of 4. In this case the shortest path from w,z is the edges (w,x), (x,y) and (y,z) totalling a weight of 3 after having the weights increased this path has a total weight of 6, whereas the edge (w,z) has a weight of 5 and is now the shortest path from w to z



4)

a)

Vertex	Discovery Time	Finish Time
A	1	12
B	8	11
C	2	7
D	9	10
E	3	6
F	4	5

b) Based on this depth first search rooted at vertex A a valid order for taking the classes would be :

A → B → D → C → E → F

This is found by producing a topological sort by inserting these vertexes at the head of a list as they finish, or in other words sorting them by their finish times in descending order.

5) If the graph is a DAG graph then a topological sort can be obtained of this graph by running a Depth First Search and then inserting the vertexes to the head of a list as they finish producing an order of vertices in descending order of finish time. This can be performed in O(V+E) time. After performing a topological sort if there is an edge in the graph from each vertex to the vertex following it then there is a Hamiltonian path through the graph.

This will correctly find an answer to the question of whether or not there is a Hamiltonian path. If this condition is satisfied, that each vertex has an edge to the next vertex in the topological ordering then that is a Hamiltonian path. To complete this proof of correctness we must show that if this is not the case (there is a vertex with no edge to the next vertex in the topological ordering, naturally excluding the final vertex) there does not exist a Hamiltonian path through this graph.

If there were such a vertex in the topological sort that does not contain an edge to the next vertex in the topological ordering there can be two cases. one case is It has no other forward edges and therefore it and all vertexes preceding it are not connected to the other vertexes. There cannot be a Hamiltonian path if the graph is not connected. The other case is it has a forward edge to some other vertex:v . There must then be some other vertexes that have edges directed towards vertex:v otherwise it would be the next vertex in the topological order and the desired property would be maintained.

let us call this vertex not connected to its successor, vertex T and Its successor vertex U

If the hamiltonian path were to go through U it must start at U in which case it will never be able to go through T as there are no back edges in a DAG or it must go through U from some forward edge connecting to U which T does not posses so again U will not be reachable in the path. Proof Complete

Pseudo Code for the algorithm:
This algorithm instead of tracking time finished will insert the vertexes on the top of a stack, which will then be popped off in topological order.

```
DFS(g)
    stack = []
    for u in g{
        if u.color == WHITE:
            DFSVisit(u, stack)
    }
    return stack

DFSVisit(u, stack)
    u.color = GRAY

    for v in u.adj{
        if v.color == WHITE:
            v.predecesor = u
            DFSVisit(v, stack)
    }

    u.color = BLACK
    stack.push(u)

Hamiltion(g)
    toposort = DFS(g)

    //is there an HPath through an empty graph?
    if toposort is empty: return False

    u = toposort.pop()
    while toposort not empty{
        v = topsort.pop()

        //check for edge from u to v
        if v not in u.adj: return false

        u = v
    }
    return True
```

Running Time of Hamilton(g)

It first makes a call to the DFS method which uses the DFS Visit. This is a modified version of the the DFS search from the CLRS book with some constant time operations of assigning times removed and a constant time operation of adding the vertex to a stack added. This call to in line 22 is therefore Θ(V+E)

next we can see that the while loop on line 28 will run exactly |V|-1 times and on each iteration it needs to check the adjacency list of each vertex for a specific vertex, line 32.

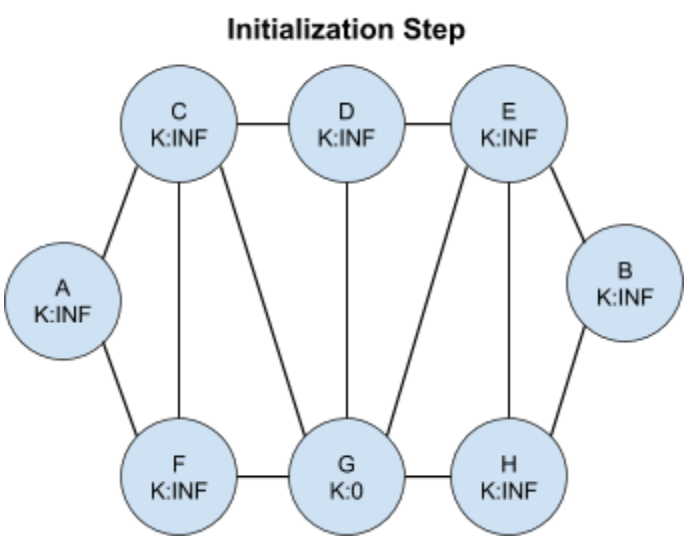
for each call this check will be O(|v.adj|) but as this is called on but one vertex only once the total cost of the line 28 while loop will be less than checking all of the edges in the graph or O(E) so we have Θ(V+E) + O(E) or O(V + 2E) or

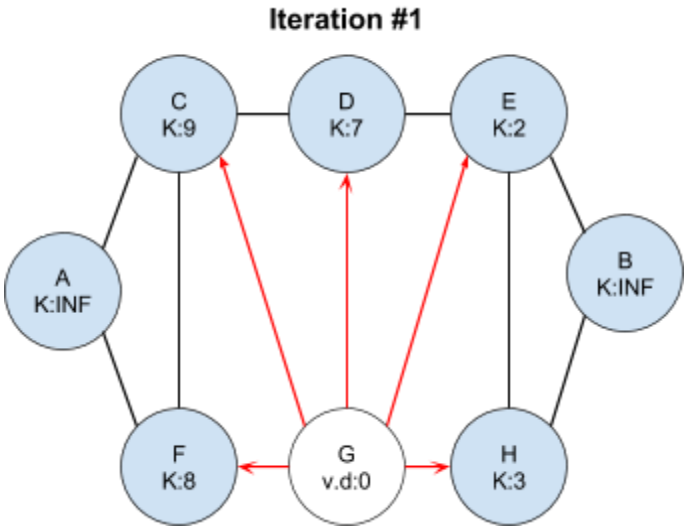
O(V + E) running time

6)

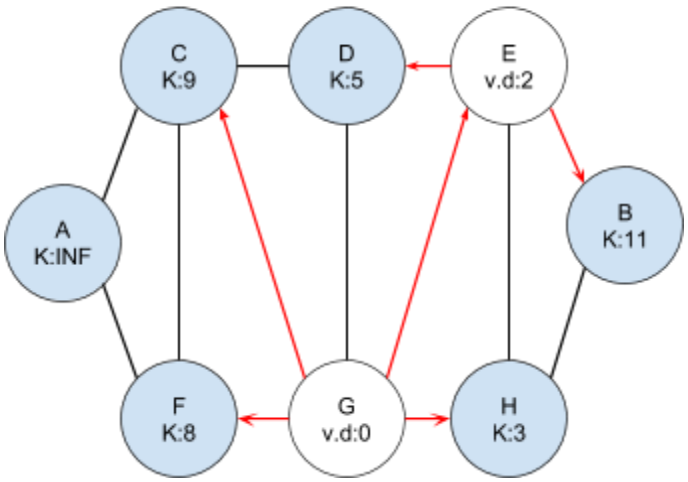
a) I would use Dijkstra's Algorithm because it can find the shortest path from a single source (the fire station) to all other points on the graph and because the roads will not have negative distance this algorithm is safe to use and has a better run time than Bellman-Ford.

Vertices colored blue are in the Que, White Vertices are in the set S of computed vertices. v.key labeled inside circle with k: # or inf for ∞. Red lines represent a vertex's predecessor with the red arrow coming from a predecessor. Each image shows the graph at the end of the While Q not empty loop

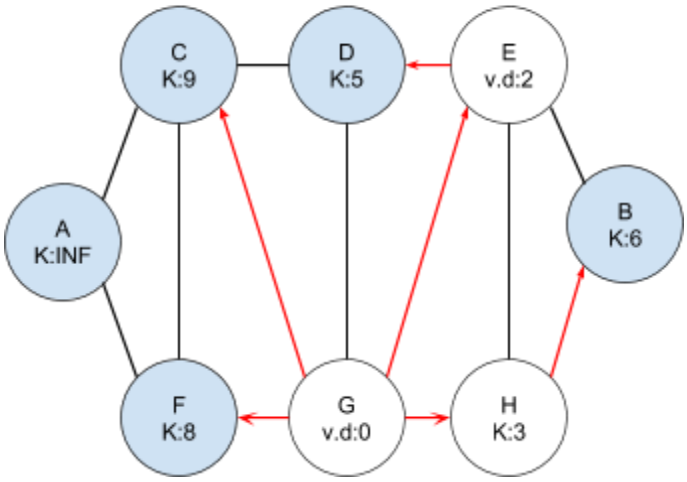




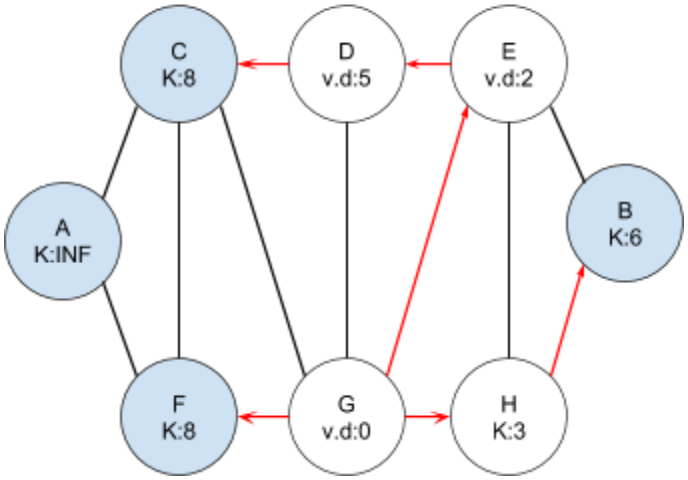
Iteration #2



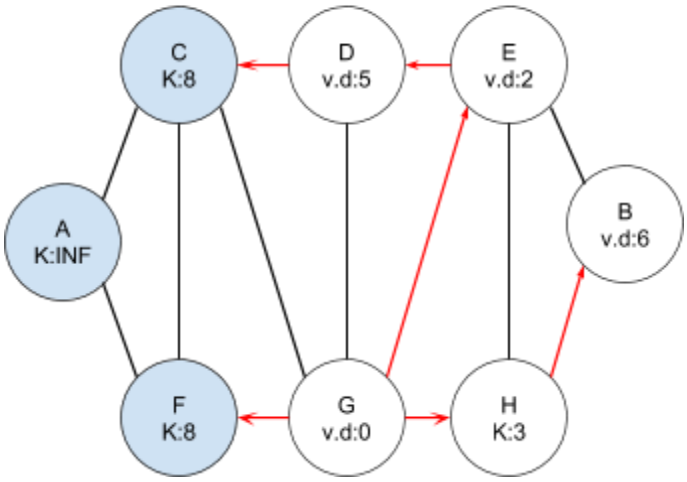
Iteration #3



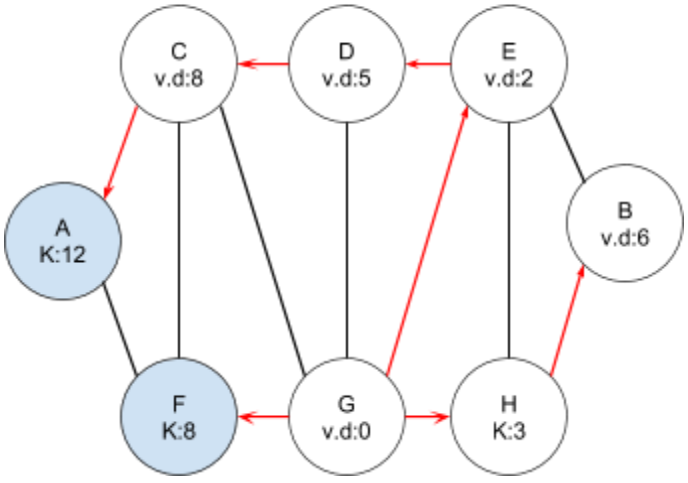
Iteration #4



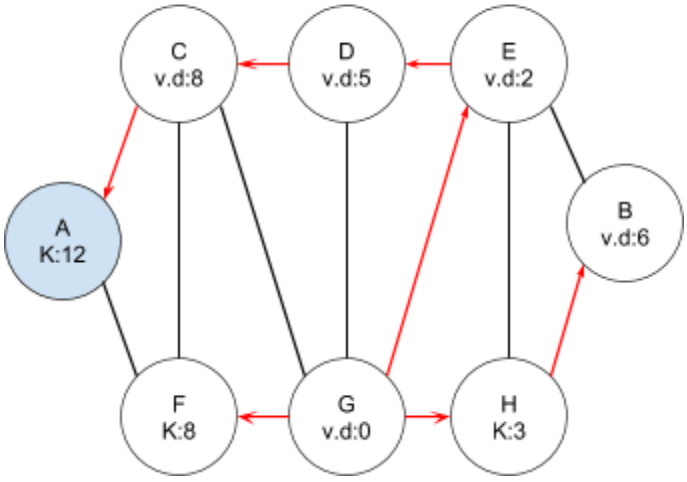
Iteration #5

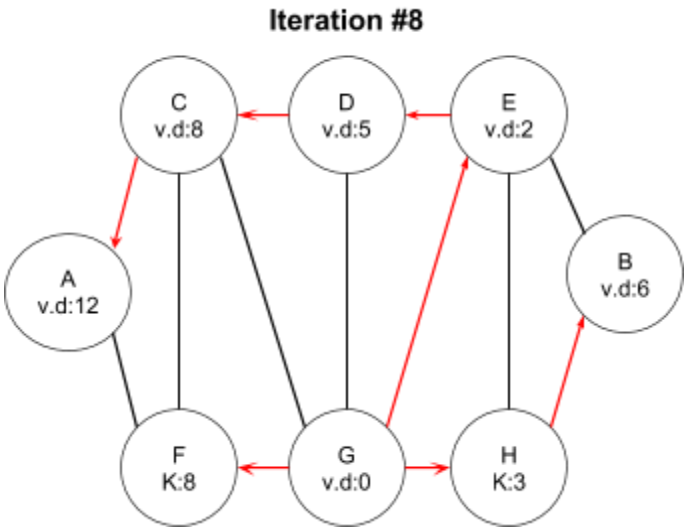


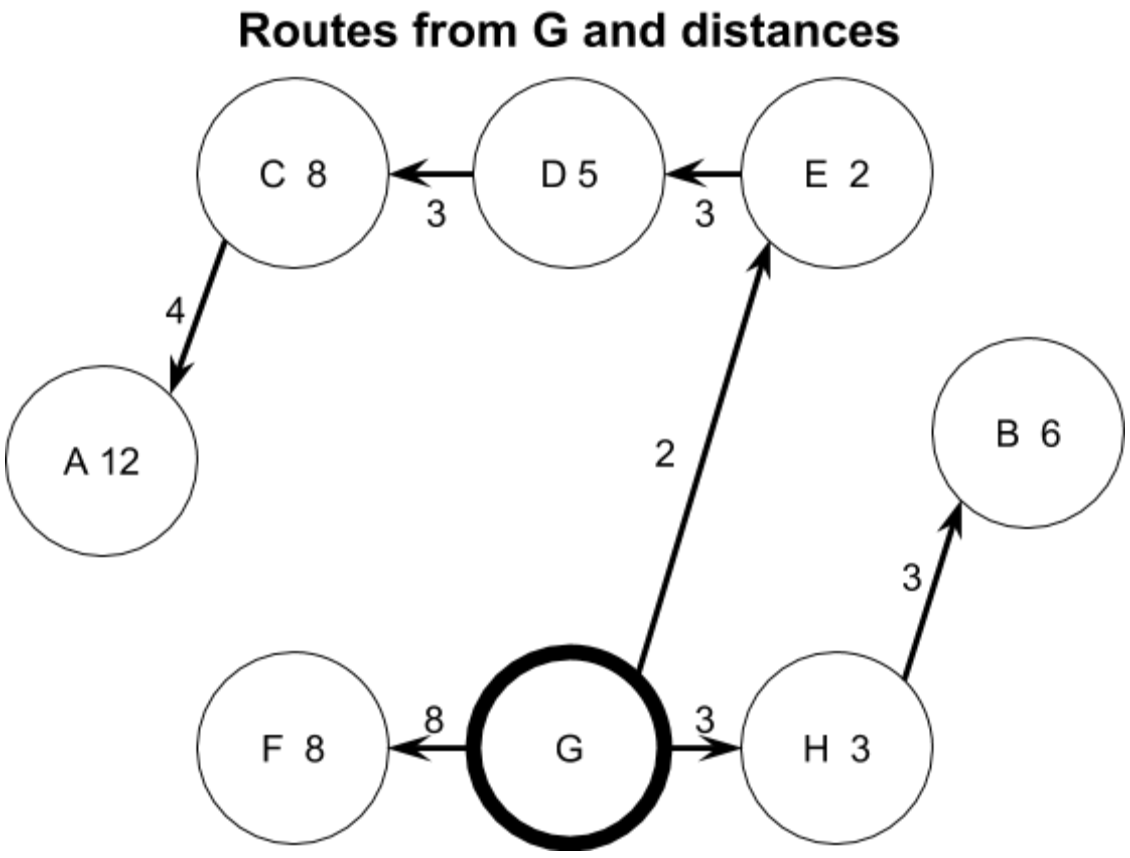
Iteration #6



Iteration #7







b) To find the optimal location to place the fire station on an arbitrary road map, and that is to reduce the length of the longest shortest path from the source, we could devise an algorithm that performs Dijkstra's Algorithm from each vertex, and then compute the maximum distance from source each time, keeping track of from which vertex was this minimized. To avoid linear

time for finding the maximum distanced vertex from a source and for finding which one of those are the minimum they will be compared with a running value as they are computed. The largest distance compared as a vertex is added to the S set in Dijkstra's algorithm and that largest distance compared to a minimum after Dijkstra's is completed

Pseudo Code

In the algorithm below Dijkstra not only modifies the distance value for each vertex in G but also returns the longest shortest-path to a vertex in the graph

```
1 //makes use of CLRS INITIALIZE-SINGLE-SOURE and RELAX
2 //DIJKSTRA from CLRS shown with modifications for this algorithm
3
4 OptimalFire(G)
5
6     best_location = nill
7     best_distance = INFINITY
8
9     for s in G.V{
10        longest = Dijkstra(G,w,s, best_distance)
11        if longest < best_distance:
12            best_distance = longest
13            best_location = s
14
15    reutrn best_location
16
17 Dijkstra(G,w,s, best)
18
19    INITIALIZE-SINGLE(G,s)
20    Q = G.V   //fibinocci que
21
22    max_distance = -INFINITY
23
24    while Q not empty{
25        u = EXTRACT-MIN(Q)
26
27        if u.d > max_distance:
28            max_distance = u.d
29            if max_distance >= best: return max_distance
30
31        for each vertex v in G.adj(u)
32            RELAX (u,v,w)
33    }
34
35    return maxdistance
```

The Dijkstra algorithm halts before completion if it finds that the shortest path to one of the vertexes in the graph is longer than or equal to the the longest shortest path of one of the

previously computed source vertexes, as if this is the case then there is no way for this source vertex to be a more optimal solution. See line 29. But this does not change the running time asymptotically.

If the classic Dijkstra function makes use of a fibonacci heap then its running time will be O(VlgV + E), and this algorithm has only added constant time steps to each iteration (comparing the extracted vertex's distance with a maximum) so this new Dijkstra function is still O(VlgV + E) time. It will be called exactly V times for a running time of $O(V^2lgV + VE)$ time.
In our roadmap example the vertexes represent the possible fire locations at intersection and the edges represent the roads so for our fire station optimization we have a running time of. $O(F^2lgF + FR)$

c) The optimal location to place the fires station is at E, it has the minimal longest shortest-path of 10 to vertex A.
The shortest possible distances from the first station from E are as follows
a = 10 from fires station
b = 8 from fires station
c = 6 from fires station
d = 3 from fires station
e = 0 from fires station
f = 8 from fires station
g = 2 from fires station
h = 5 from fires station

like so