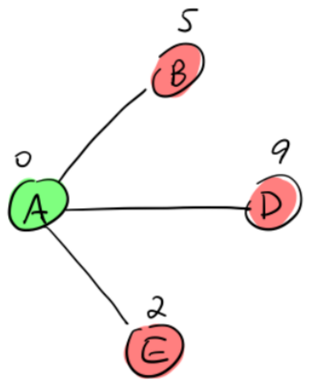
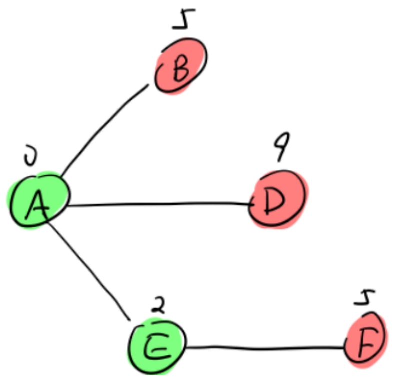
1.

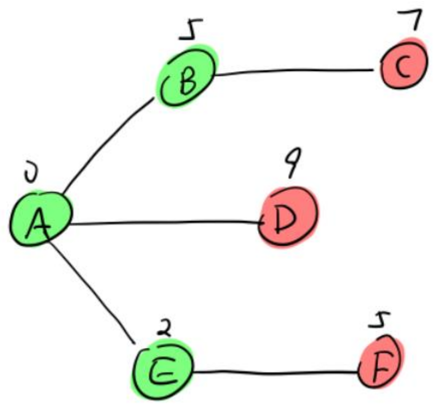
The set sptSet is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum distance value. The vertex "A" is picked, include it in sptSet. So sptSet becomes {"A"}. After including "A" to sptSet, update distance values of its adjacent vertices. Adjacent vertices of "A" are "B", "D" and "E". The distance values of "B", "D" and "E" are updated as 5, 9 and 2. Following subgraph shows vertices and their distance values, only the vertices with finite distance values are shown. The vertices included in SPT are shown in green color.



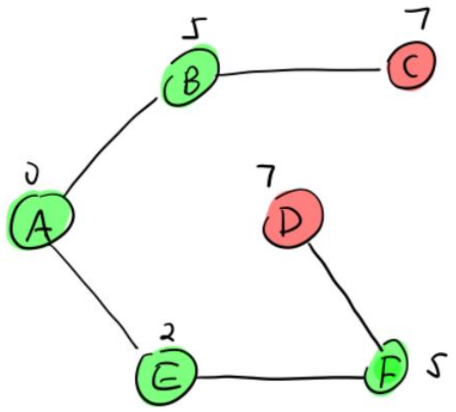
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex "E" is picked and added to sptSet. So sptSet now becomes {"A", "E"}. Update the distance values of adjacent vertices of "E". The distance value of vertex "F" becomes 5.



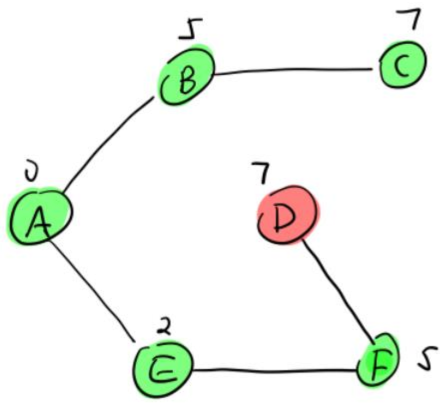
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Either "B" or "F" could be picked. In this case, vertex "B" is picked and added to sptSet. So sptSet now becomes {"A", "E", "B"}. Update the distance values of adjacent vertices of "B". The distance value of vertex "C" becomes 7.



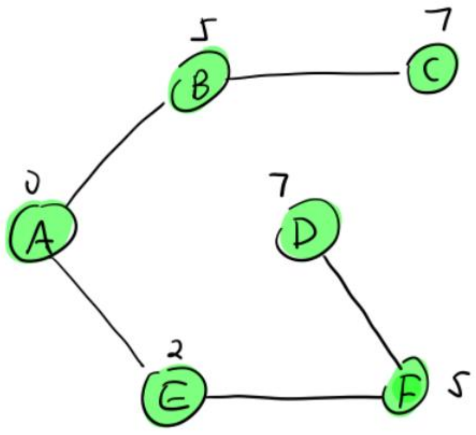
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex "F" is picked and added to sptSet. So sptSet now becomes {"A", "E", "B", "F"}. Update the distance values of adjacent vertices of "F". The distance value of vertex "D" becomes 7.



Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Either "C" or "D" could be picked. In this case, vertex "C" is picked and added to sptSet. So sptSet now becomes {"A", "E", "B", "F", "C"}. Update the distance values of adjacent vertices of "C".



Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex "D" is picked and added to sptSet. So sptSet now becomes {"A", "E", "B", "F", "C", "D"}. Finally, we get the following Shortest Path Tree (SPT).

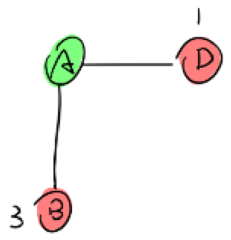


Path map:

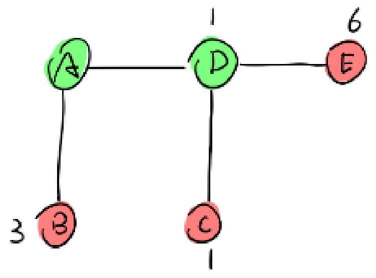
V	Parent	Distance
A	Null	0
B	A	5
C	B	7
D	F	7
E	A	2
F	E	5

2.

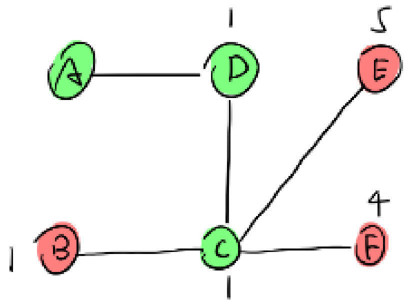
The set mstSet is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with the minimum key value. The vertex "A" is picked, include it in mstSet. So mstSet becomes {"A"}. After including to mstSet, update key values of adjacent vertices. Adjacent vertices of "A" are "B" and "D". The key values of "B" and "D" are updated as 3 and 1. Following subgraph shows vertices and their key values, only the vertices with finite key values are shown. The vertices included in MST are shown in green color.



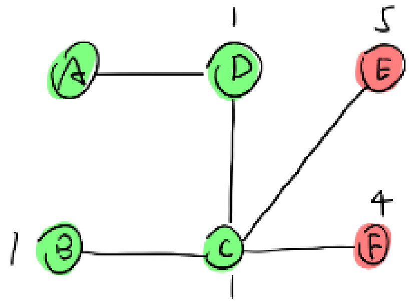
Pick the vertex with minimum key value and not already included in MST (not in mstSET). The vertex "D" is picked and added to mstSet. So mstSet now becomes {"A", "D"}. Update the key values of adjacent vertices of "D". The key value of vertex "C" becomes 1, and the key value of vertex "E" becomes 6.



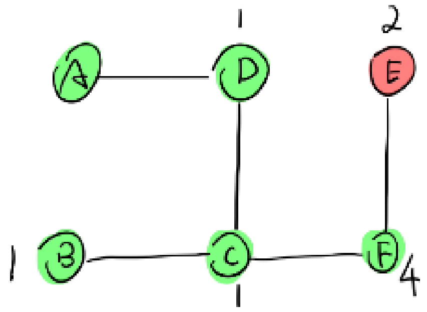
Pick the vertex with minimum key value and not already included in MST (not in mstSET). The vertex "C" is picked and added to mstSet. So mstSet now becomes {"A", "D", "C"}. Update the key values of adjacent vertices of "C". The key value of vertex "F" becomes 4, the key value of vertex "B" becomes 1, and the key value of vertex "E" becomes 5.



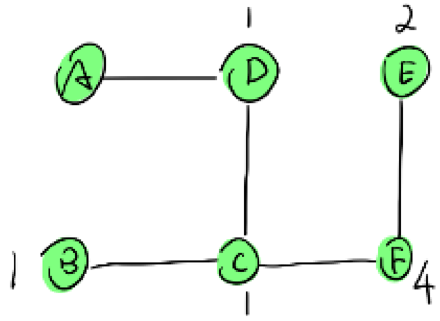
Pick the vertex with minimum key value and not already included in MST (not in mstSET). The vertex “B” is picked and added to mstSet. So mstSet now becomes {“A”, “D”, “C”, “B”}. Update the key values of adjacent vertices of “B”.



Pick the vertex with minimum key value and not already included in MST (not in mstSET). The vertex “F” is picked and added to mstSet. So mstSet now becomes {“A”, “D”, “C”, “B”, “F”}. Update the key values of adjacent vertices of “F”. The key value of vertex “E” becomes 2.



Pick the vertex with minimum key value and not already included in MST (not in mstSET). The vertex “E” is picked and added to mstSet. So mstSet now becomes {“A”, “D”, “C”, “B”, “F”, “E”}. Finally, we get the following graph.

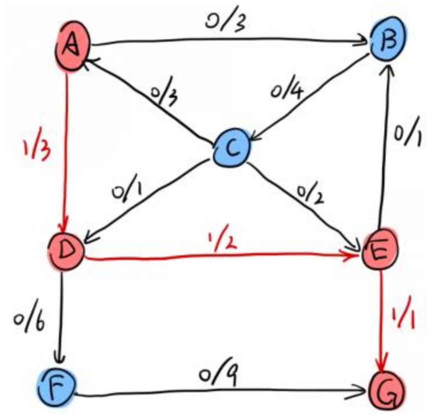


edge table:

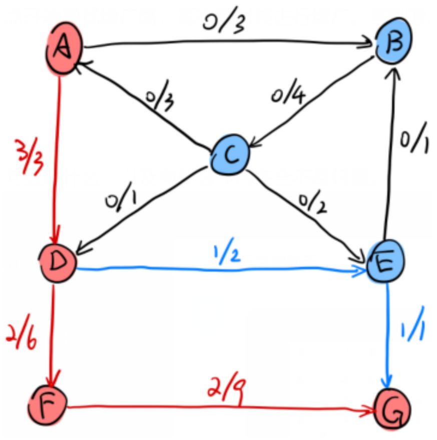
V	Edge
B	BC
C	CD
D	AD
E	EF
F	CF

3.

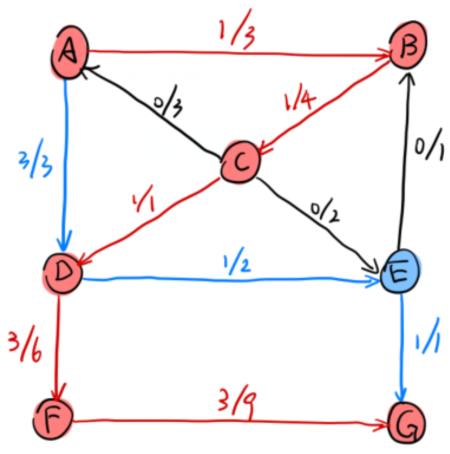
Step 1: A->D->E->G (flow: 1)



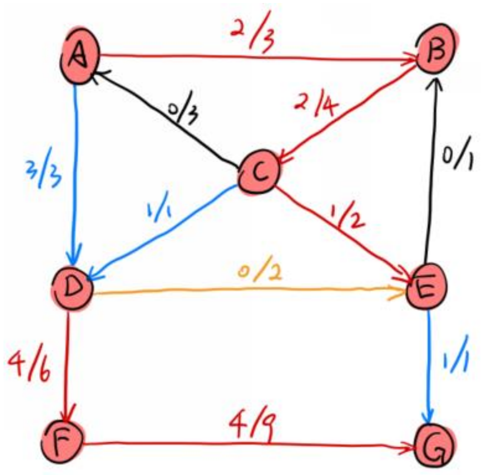
Step 2: A->D->F->G (flow: 2)



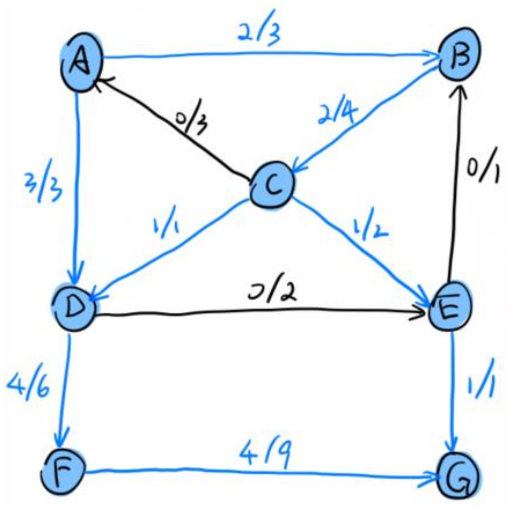
Step 3: A->B->C->D->F->G (flow: 1)



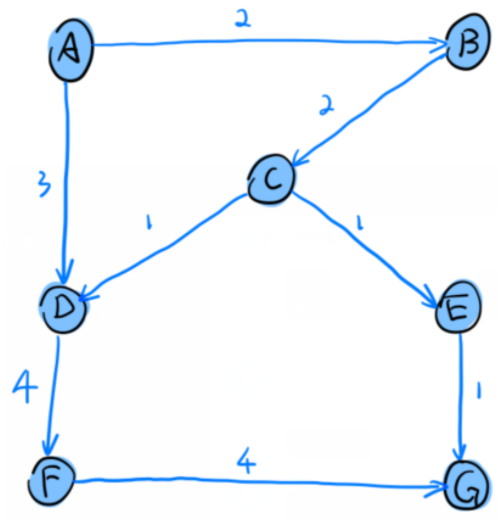
Step 4: A->B->C->E->D->F->G (flow: 1)



Then we get the final maximum flow from source A to destination G:



which could be simplified to:



The maximum flow is 5.

4.

bfs(vert, start, sink)

Input: The vertices list, the start node, and the sink node.

Output – True when the sink is visited.

```
Begin
    initially mark all nodes as unvisited
    state of start as visited
    predecessor of start node is φ
    insert start into the queue qu
    while qu is not empty, do
        delete element from queue and set to vertex u
        for all vertices i, in the residual graph, do
            if u and i are connected, and i is unvisited, then
                add vertex i into the queue
                predecessor of i is u
                mark i as visited
        done
    done
    return true if state of sink vertex is visited
End
```

fordFulkerson(vert, source, sink)

Input: The vertices list, the source vertex, and the sink vertex.

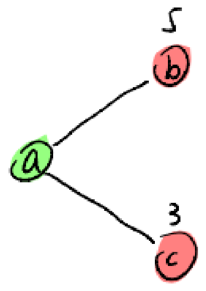
Output – The maximum flow from start to sink.

```
Begin
    create a residual graph and copy given graph into it
    while bfs(vert, source, sink) is true, do
        pathFlow := ∞
        v := sink vertex
        while v ≠ start vertex, do
            u := predecessor of v
            pathFlow := minimum of pathFlow and residualGraph[u, v]
            v := predecessor of v
        done
        v := sink vertex
        while v ≠ start vertex, do
            u := predecessor of v
            residualGraph[u,v] := residualGraph[u,v] – pathFlow
            residualGraph[v,u] := residualGraph[v,u] – pathFlow
            v := predecessor of v
        done
        maxFlow := maxFlow + pathFlow
    done
    return maxFlow
End
```

The method “bfs” is used to detect whether there is still an augmented-path in the residual network. It starts from the source and marks every vertex that is connected to the visited ones to see whether the sink vertex could be visited. If the sink vertex could be visited, there should be at least one more augmented-path.
The method “fordFulkerson” is used to find out the augmented-path and update the residual network. The first loop is implemented to find out the pathFlow of the augmented-path. The second loop is implemented to update the residual network.
There is no recursive nature to the algorithm. The termination point of this algorithm is when there is no more augmented-path in the residual network, which means the return value of method “bfs” is false.

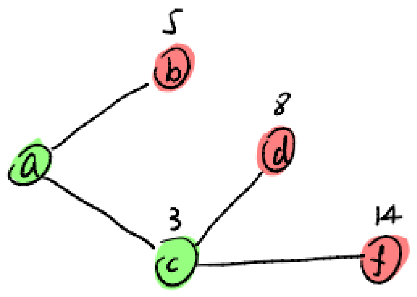
5.
A) Prim’s algorithm. In each loop, the minimum priority edge that is connected to the vertices in the set is chosen. And the priority of each vertex is the weight of the edge instead of the simulation of the edges from the start point.
B) Kruskal’s algorithm. In each loop, the edge with the smallest priority in the whole graph is chosen.
C) Dijktra’s algorithm. In each loop, the minimum priority edge that is connected to the vertices in the set is chosen. And the priority of each vertex is the simulation of the edges from the start point.

6.
a)
The set sptSet is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum distance value. The vertex “A” is picked, include it in sptSet. So sptSet becomes {“A”}. After including “A” to sptSet, update distance values of its adjacent vertices. Adjacent vertices of “A” are “B” and “C”. The distance values of “B” and “C” are updated as 5 and 3. Following subgraph shows vertices and their distance values, only the vertices with finite distance values are shown. The vertices included in SPT are shown in green color.

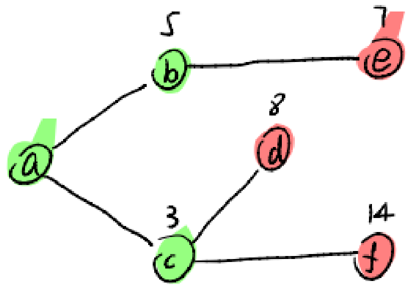


Pick the vertex with minimum distance value and not already included in SPT (not in sptSET).

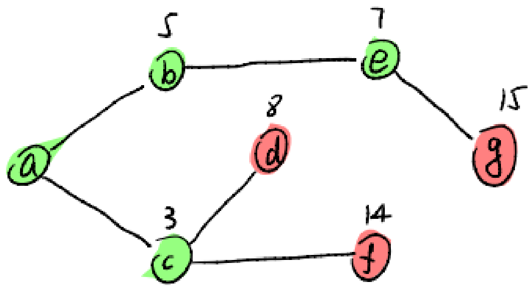
The vertex “C” is picked and added to sptSet. So sptSet now becomes {“A”, “C”}. Update the distance values of adjacent vertices of “C”. The distance value of vertex “D” becomes 8 and the distance value of vertex “F” becomes 14.

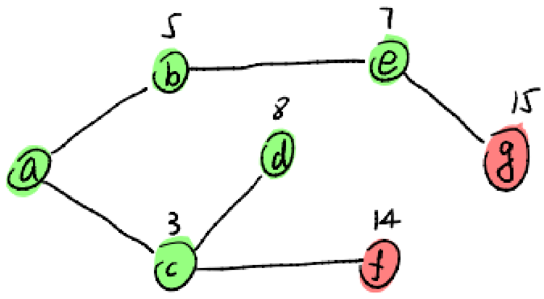


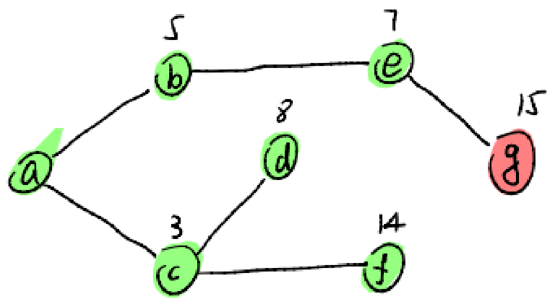
Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex “B” is picked and added to sptSet. So sptSet now becomes {“A”, “C”, “B”}. Update the distance values of adjacent vertices of “B”. The distance value of vertex “E” becomes 7.



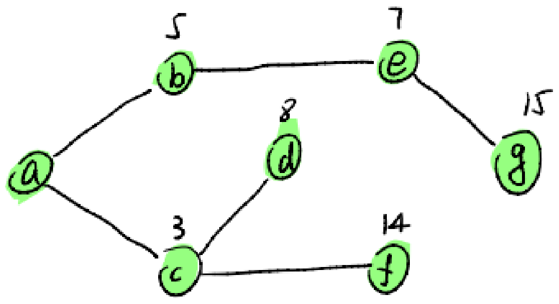
Repeat the steps:





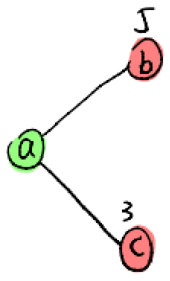


Finally, we get the following Shortest Path Tree (SPT).

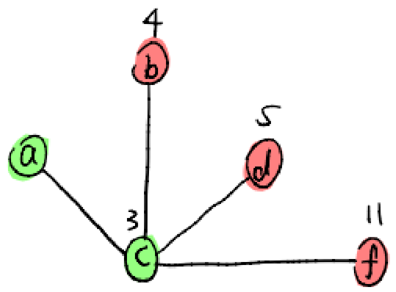


b)

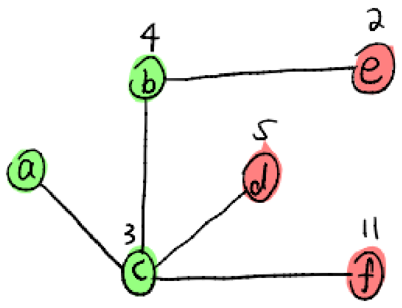
The set mstSet is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with the minimum key value. The vertex "A" is picked, include it in mstSet. So mstSet becomes {"A"}. After including to mstSet, update key values of adjacent vertices. Adjacent vertices of "A" are "B" and "C". The key values of "B" and "C" are updated as 5 and 3. Following subgraph shows vertices and their key values, only the vertices with finite key values are shown. The vertices included in MST are shown in green color.



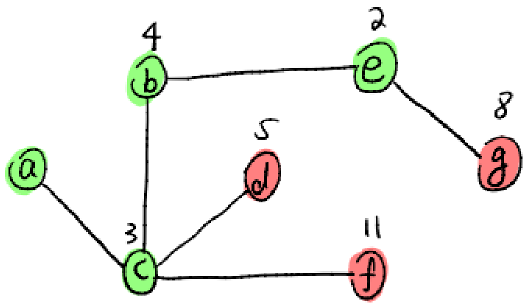
Pick the vertex with minimum key value and not already included in MST (not in mstSET). The vertex "C" is picked and added to mstSet. So mstSet now becomes {"A", "C"}. Update the key values of adjacent vertices of "C". The key value of vertex "B" becomes 4, The key value of vertex "D" becomes 5 and the key value of vertex "F" becomes 11.

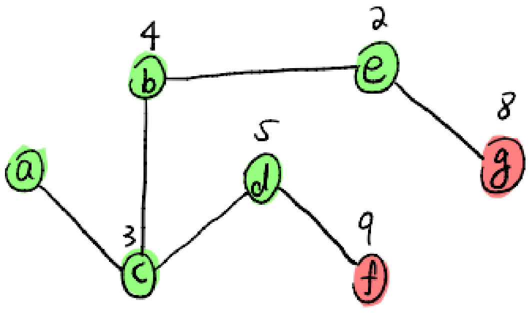


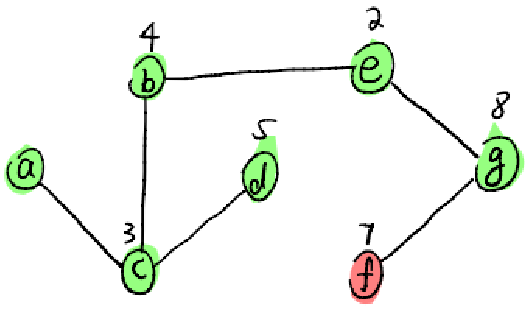
Pick the vertex with minimum key value and not already included in MST (not in mstSET). The vertex “B” is picked and added to mstSet. So mstSet now becomes {“A”, “C”, “B”}. Update the key values of adjacent vertices of “B”. The key value of vertex “E” becomes 2.



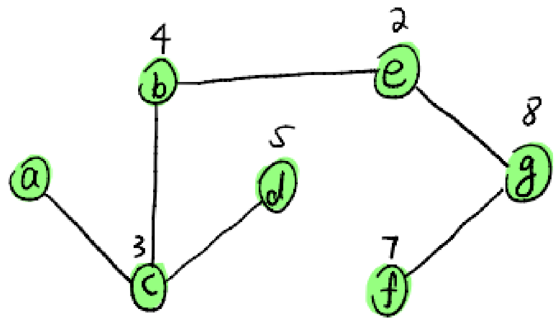
Repeat the steps:







Finally, we get the following graph.



d)

Algorithm	Time complexity	Space complexity
Dijkstra Shortest-Path	$O(V^2)$	$O(E+V)$
Prim's	$O(E * \log(V))$	$O(E+V)$