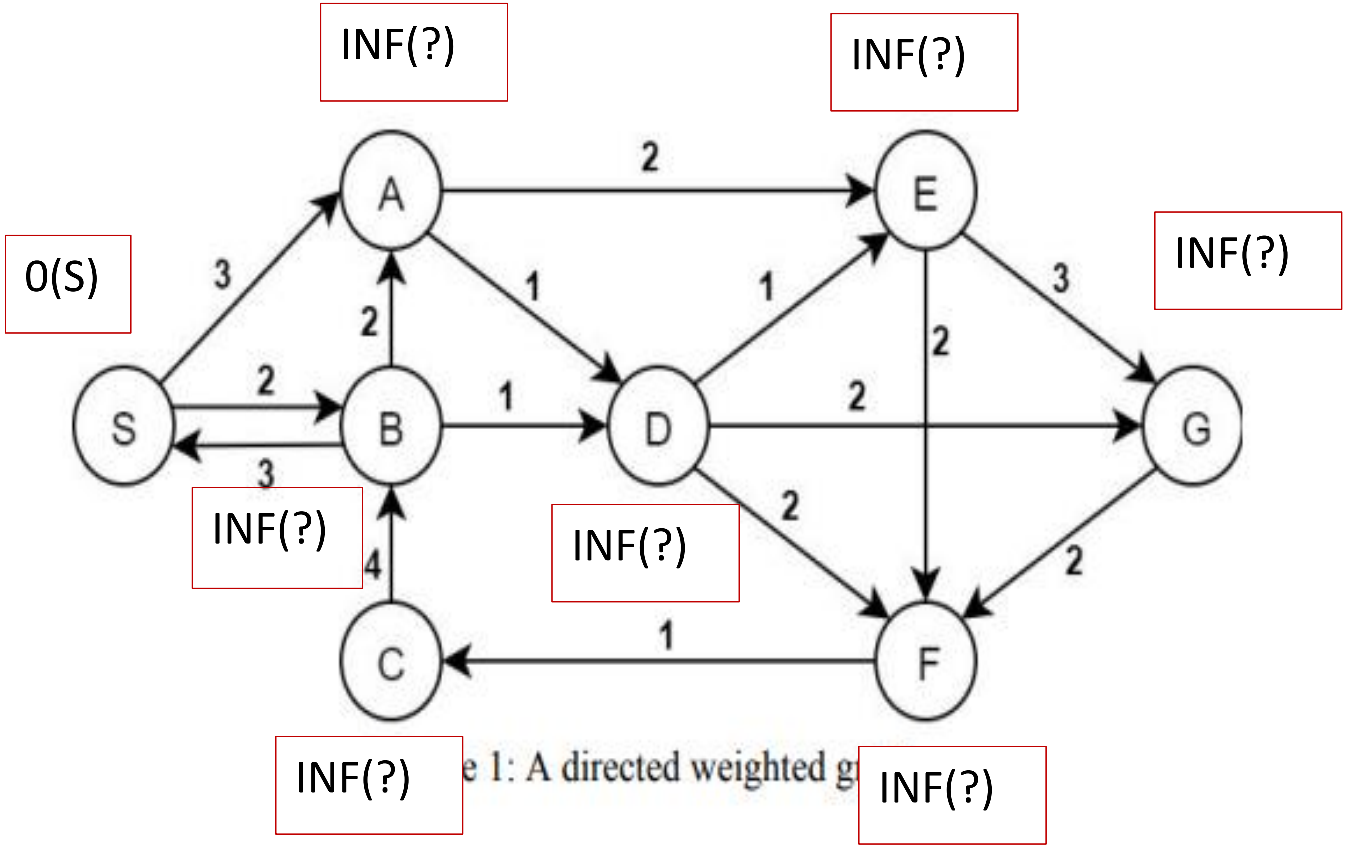
CS300 HW5

Dilara Nur Memiş

27868

Q1: Dijkstra



First, we will initialize all nodes as unknown.

S is at distance 0 from itself.

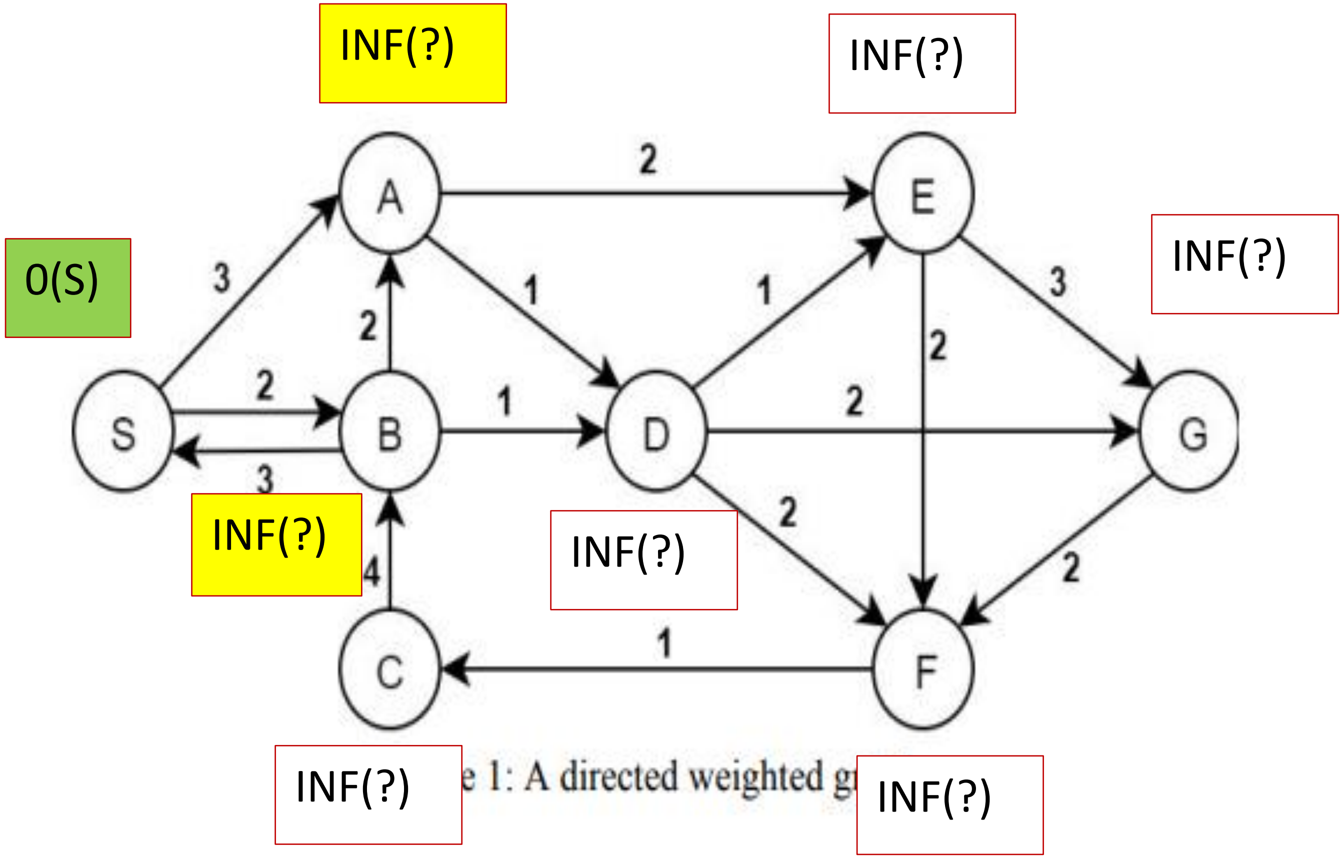
All others are at distance INF from S.

: Known

: Unknown

: Adjacent to min.distance node.

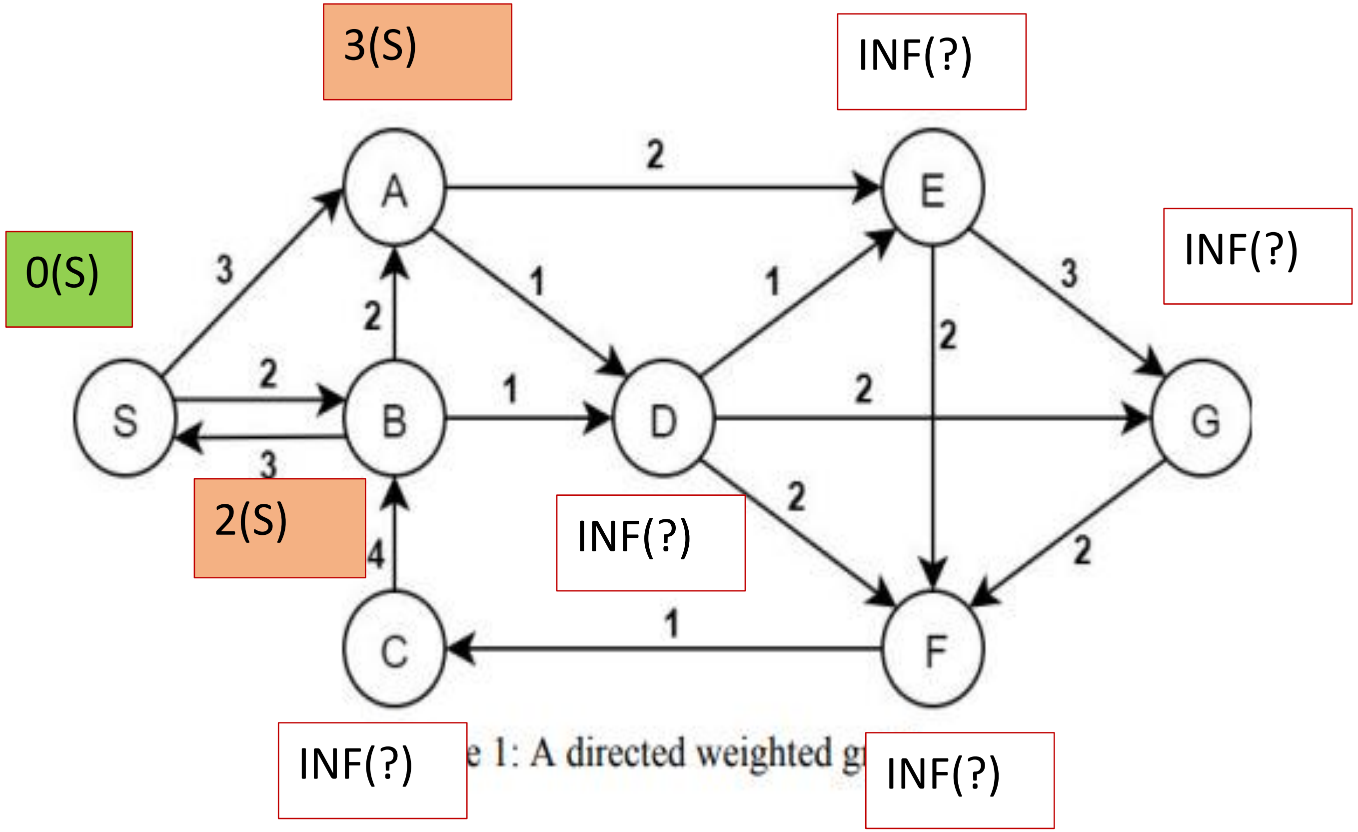
Q1- Dijkstra



Among all unknown nodes, S has the minimum distance. Choose S. Marked it as known.

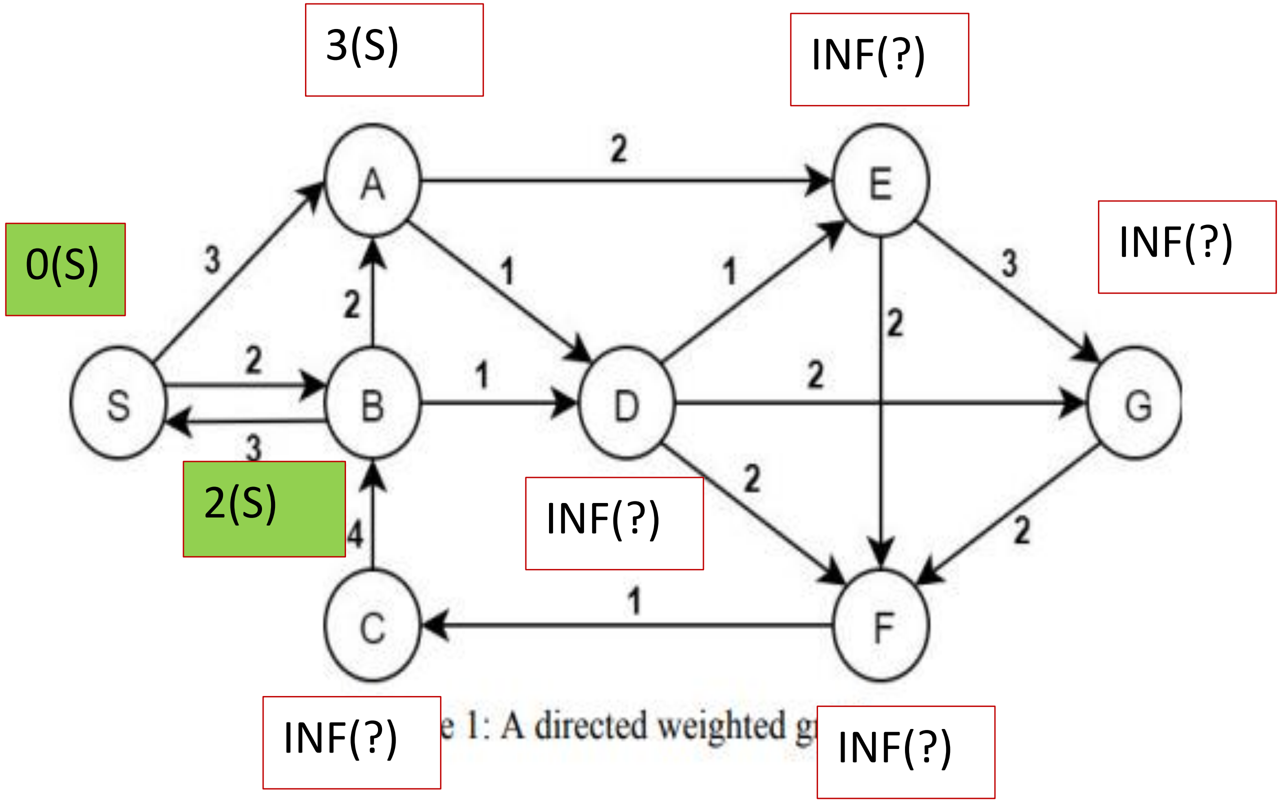
Nodes adjacent to S are A and B.

Q1- Dijkstra



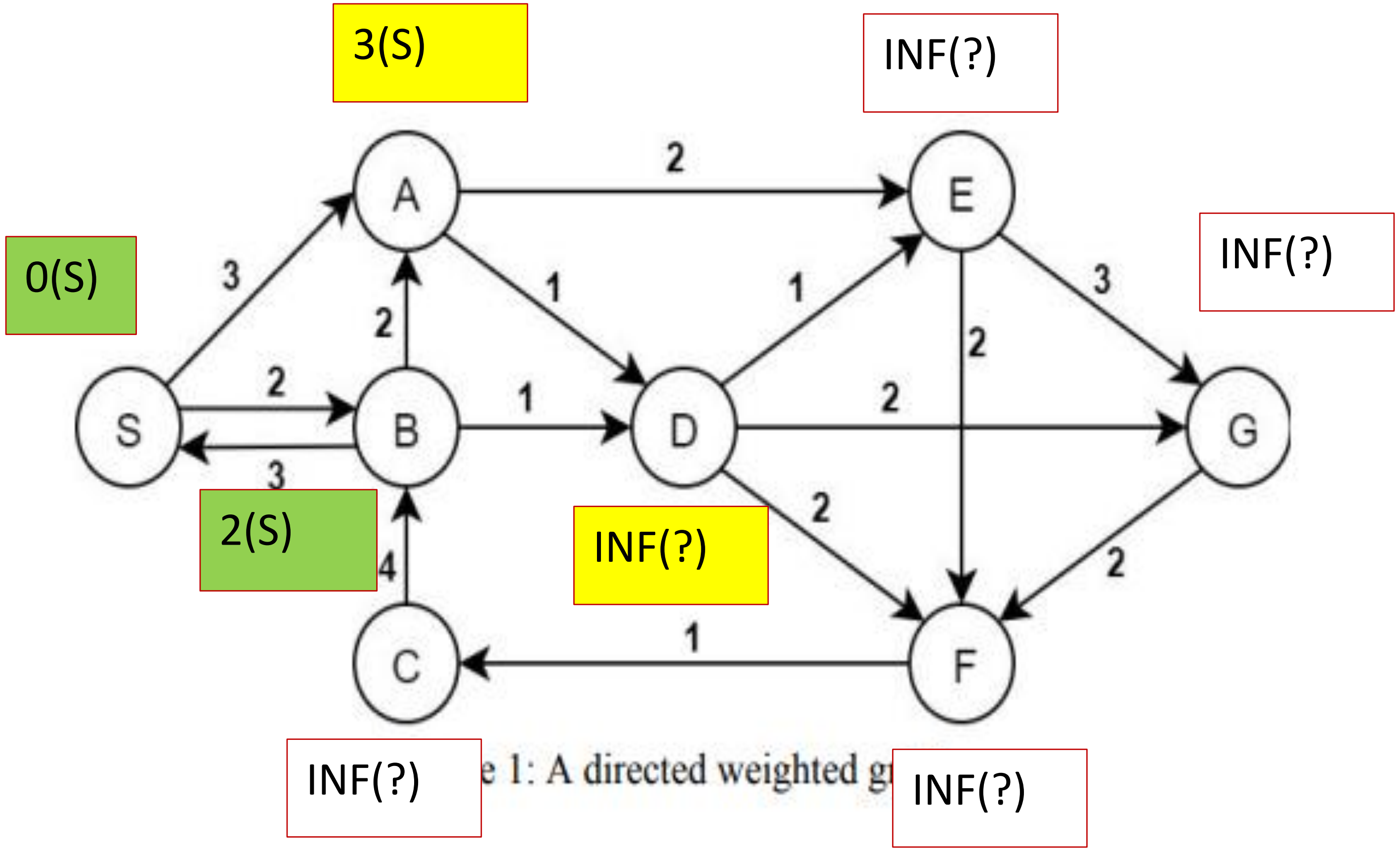
Update the distances and paths of A and B since there is a shorter path.

Q1- Dijkstra



Among all unknown nodes, B has the minimum distance. Choose B. Marked it as known.

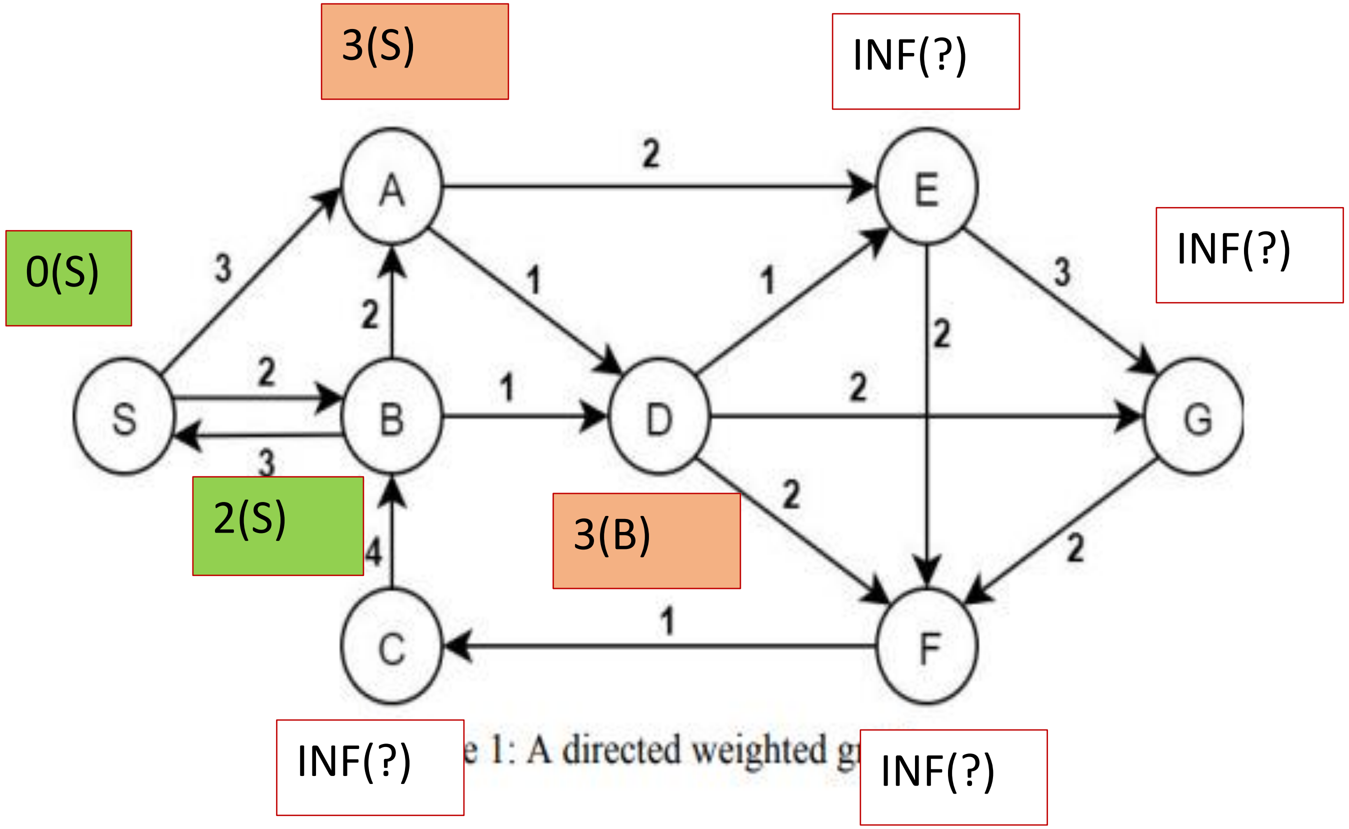
Q1- Dijkstra



Nodes adjacent to B are S, A and D.

Since S is known, we dont process it again.

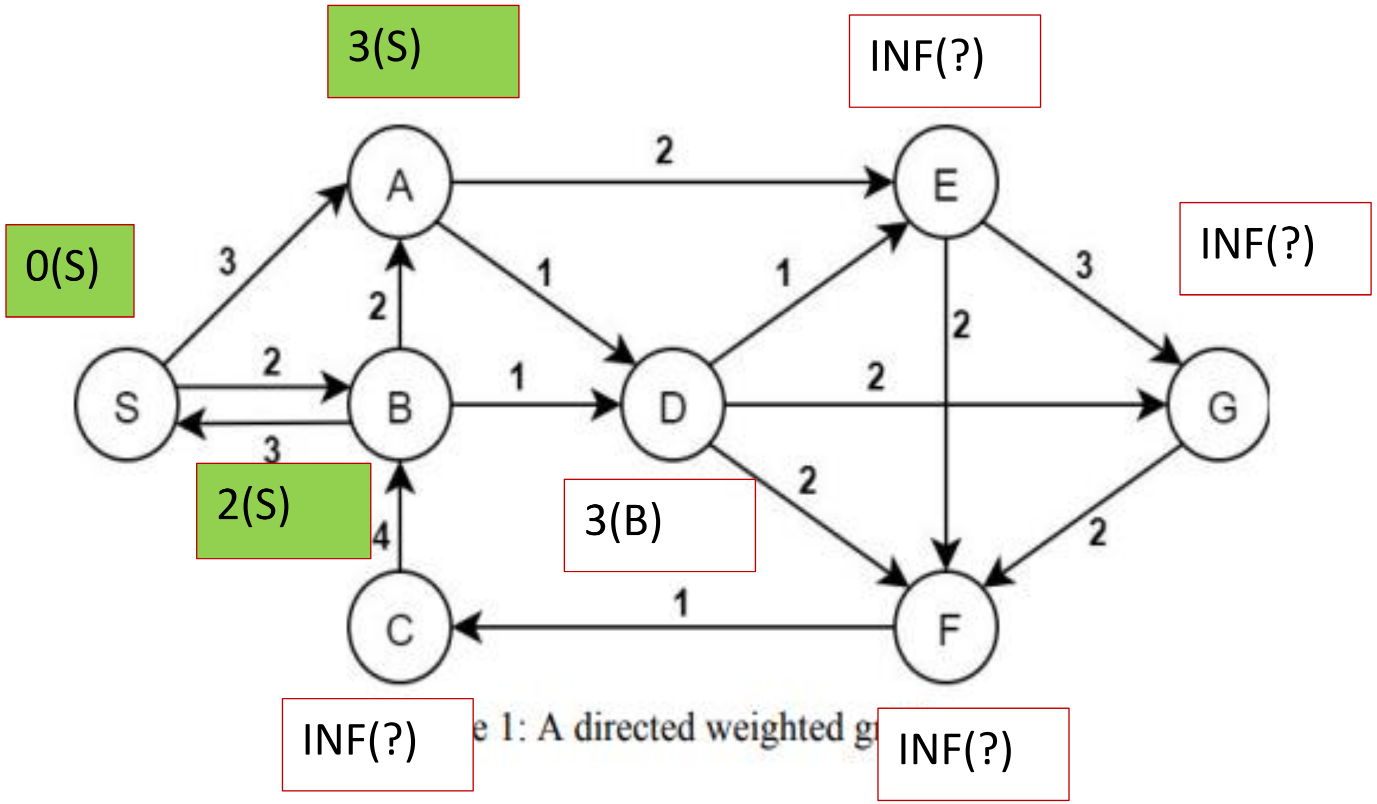
Q1- Dijkstra



We updated distance of D since there is a shorter path to D now.

We didnt change distance of A since B does not give a shorter path to A.

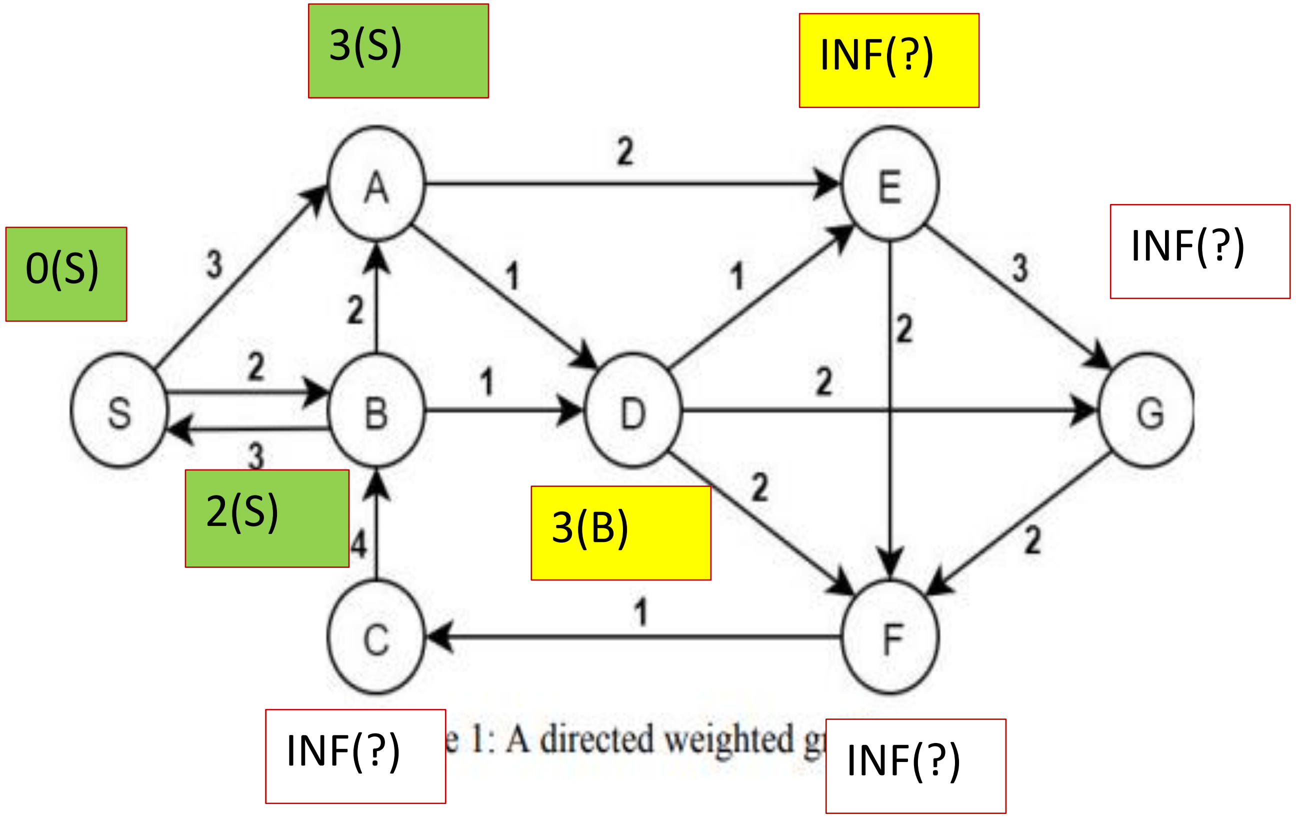
Q1- Dijkstra



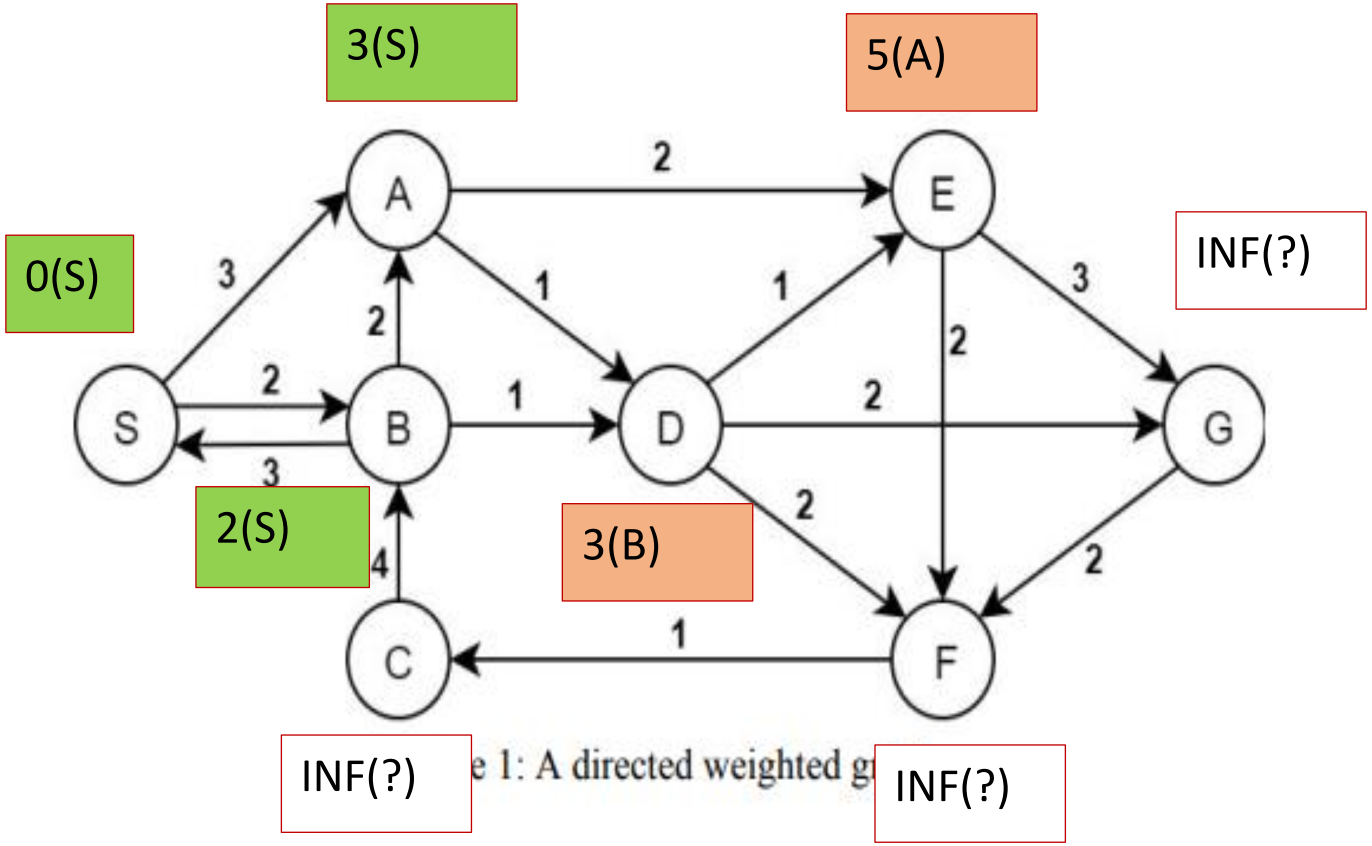
Among all unknown nodes, A and D have the minimum distances. Choice is arbitrary. Choose A. Marked it as known.

Q1- Dijkstra

Nodes adjacent to A are E and D.



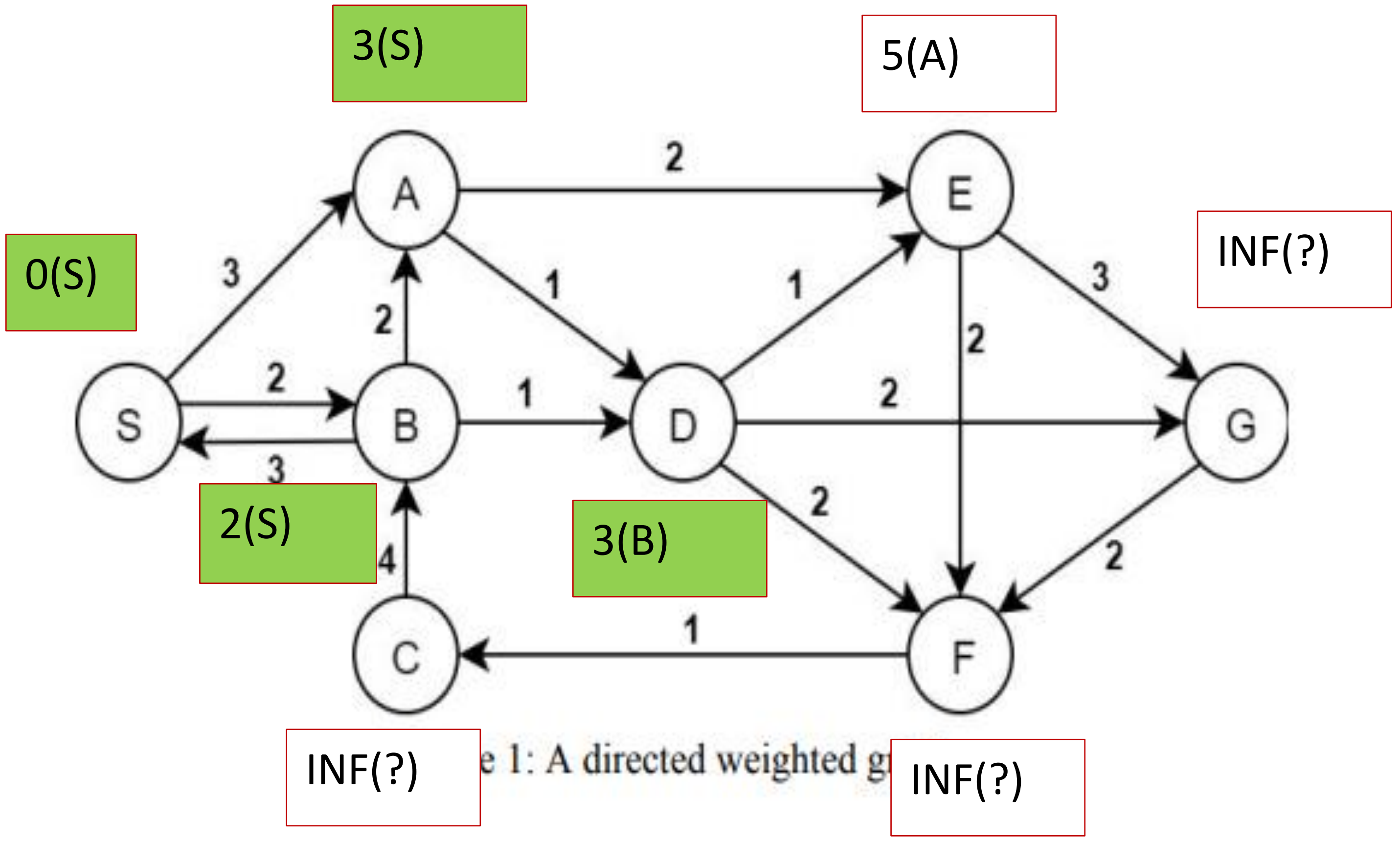
Q1- Dijkstra



We updated distance of D since A gives a shorter path to D.

Distance of D was not updated since A does not give a shorter path to this node.

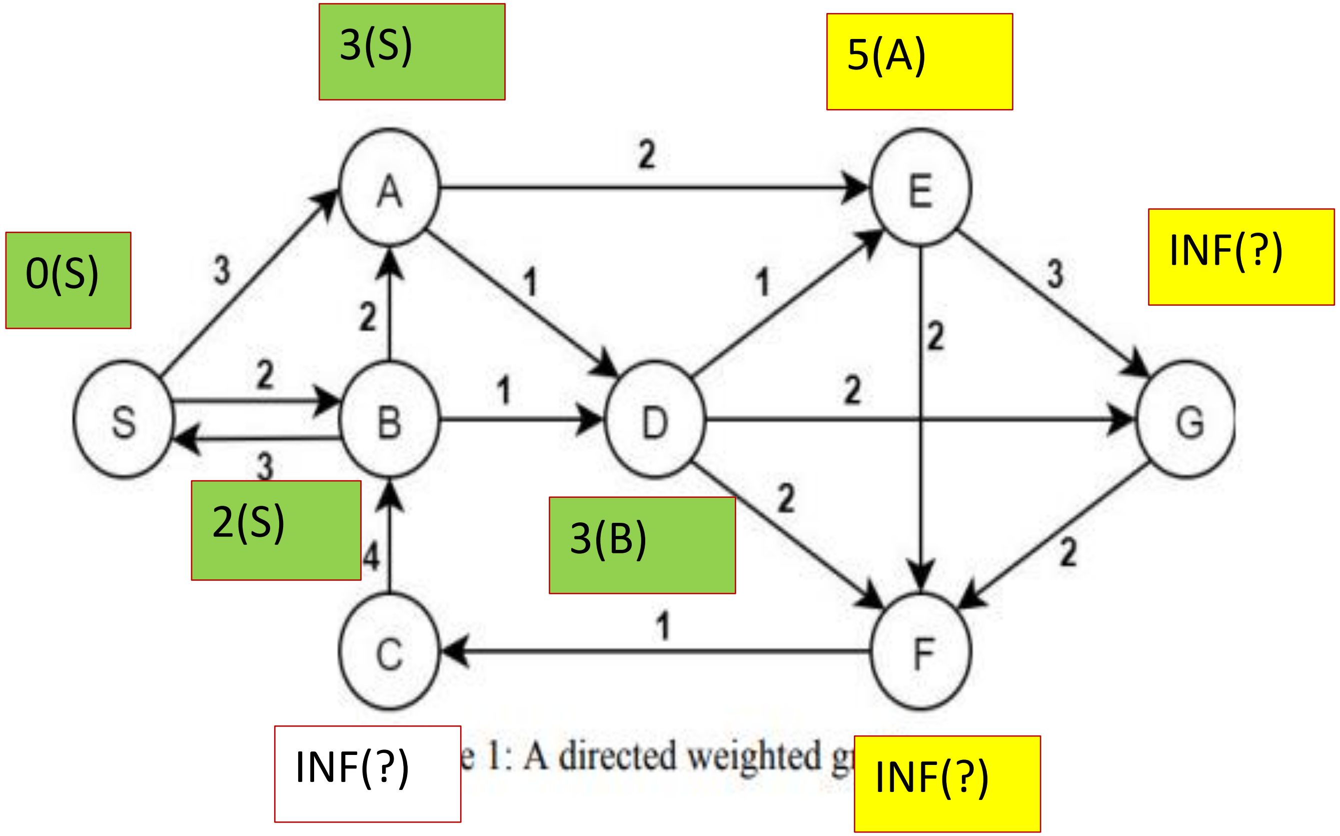
Q1- Dijkstra



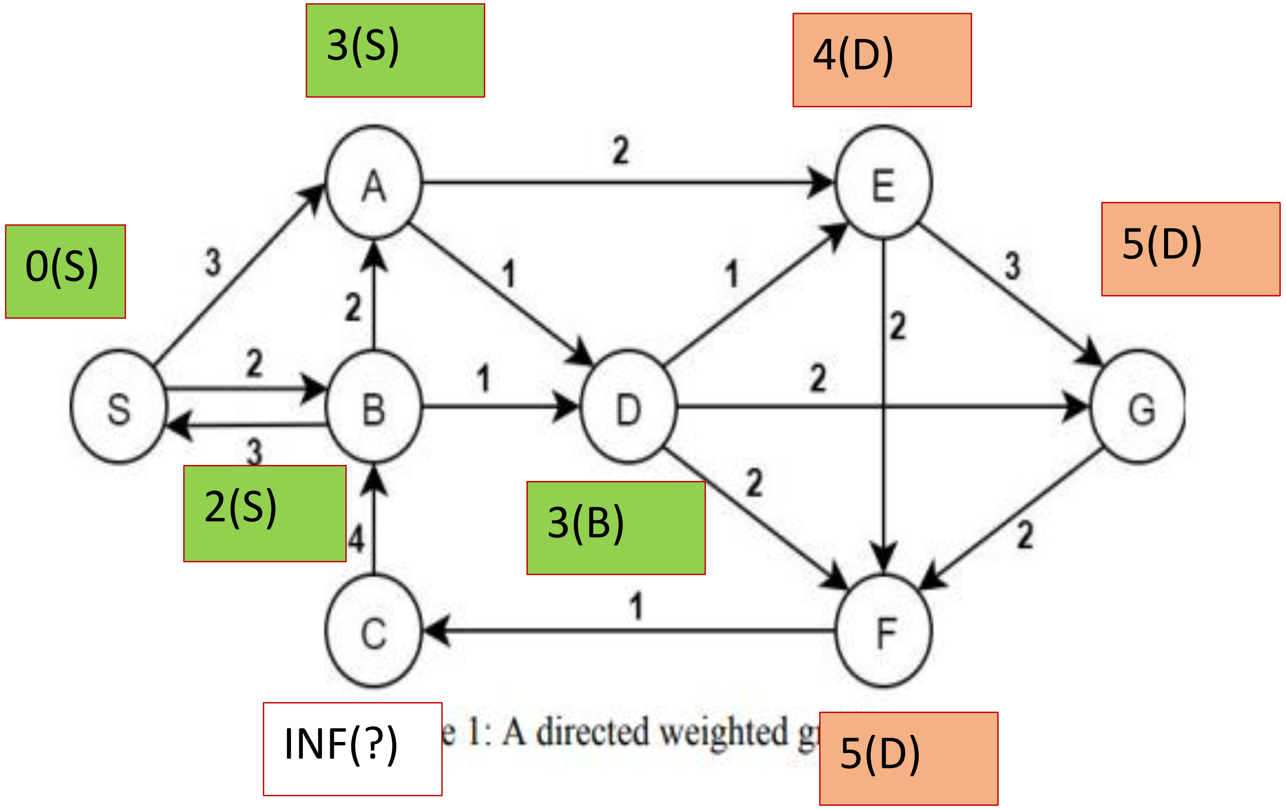
Among all unknown nodes, D has the minimum distance. Choose D. Marked it as known.

Q1- Dijkstra

Nodes adjacent to D are E, G and F.



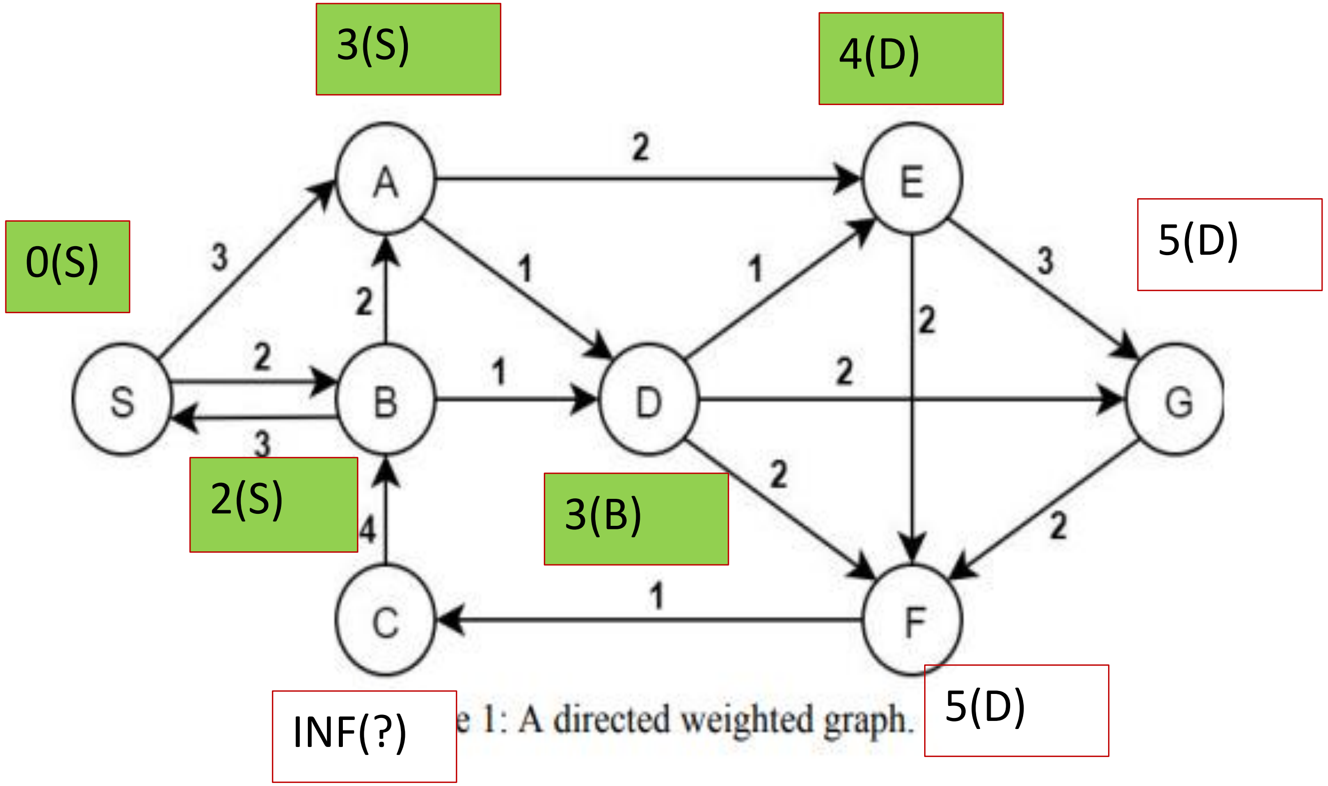
Q1- Dijkstra



We updated distances of E, G and F since D gives all three nodes shorther paths.

Q1- Dijkstra

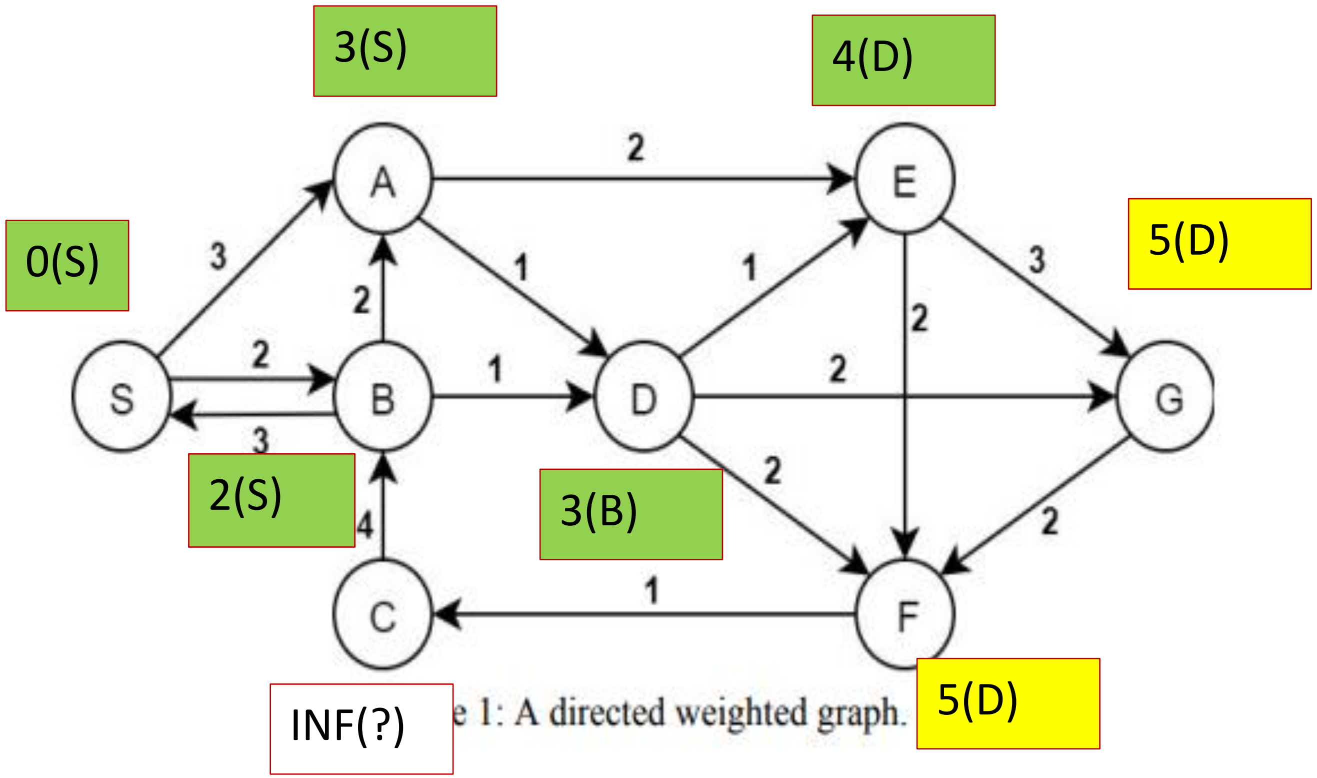
Among all unknown nodes, E has the minimum distance. Choose E. Marked it as known.



e 1: A directed weighted graph.

Q1- Dijkstra

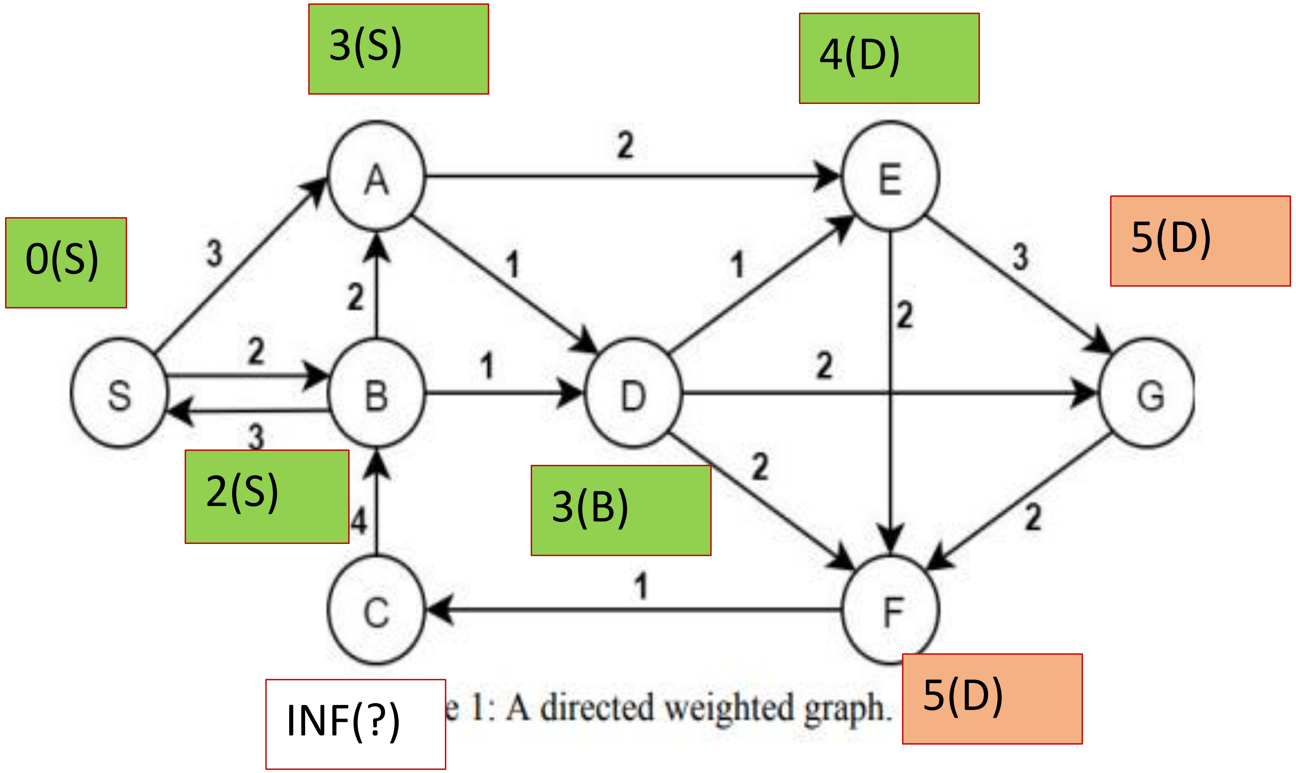
Nodes adjacent to E are G and F.



e 1: A directed weighted graph.

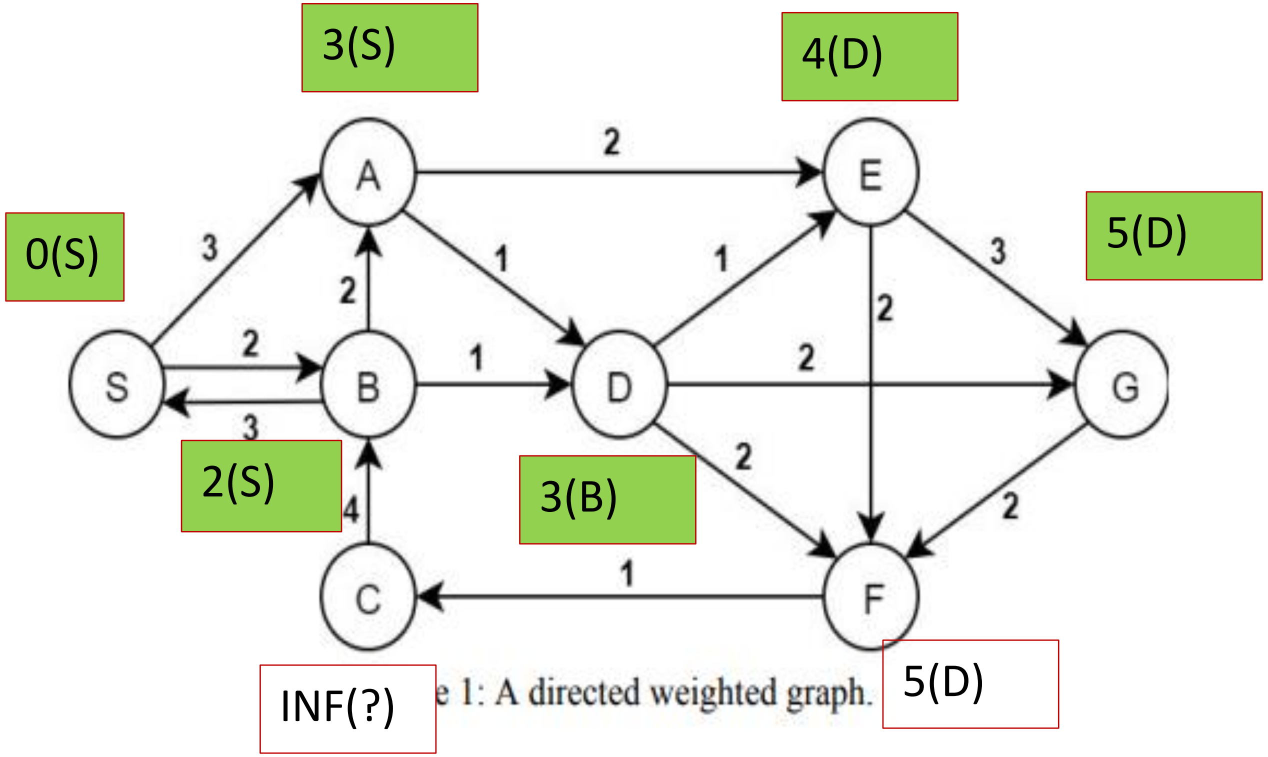
Q1- Dijkstra

We did not updated distances of them since E does not give shorter paths.



e 1: A directed weighted graph.

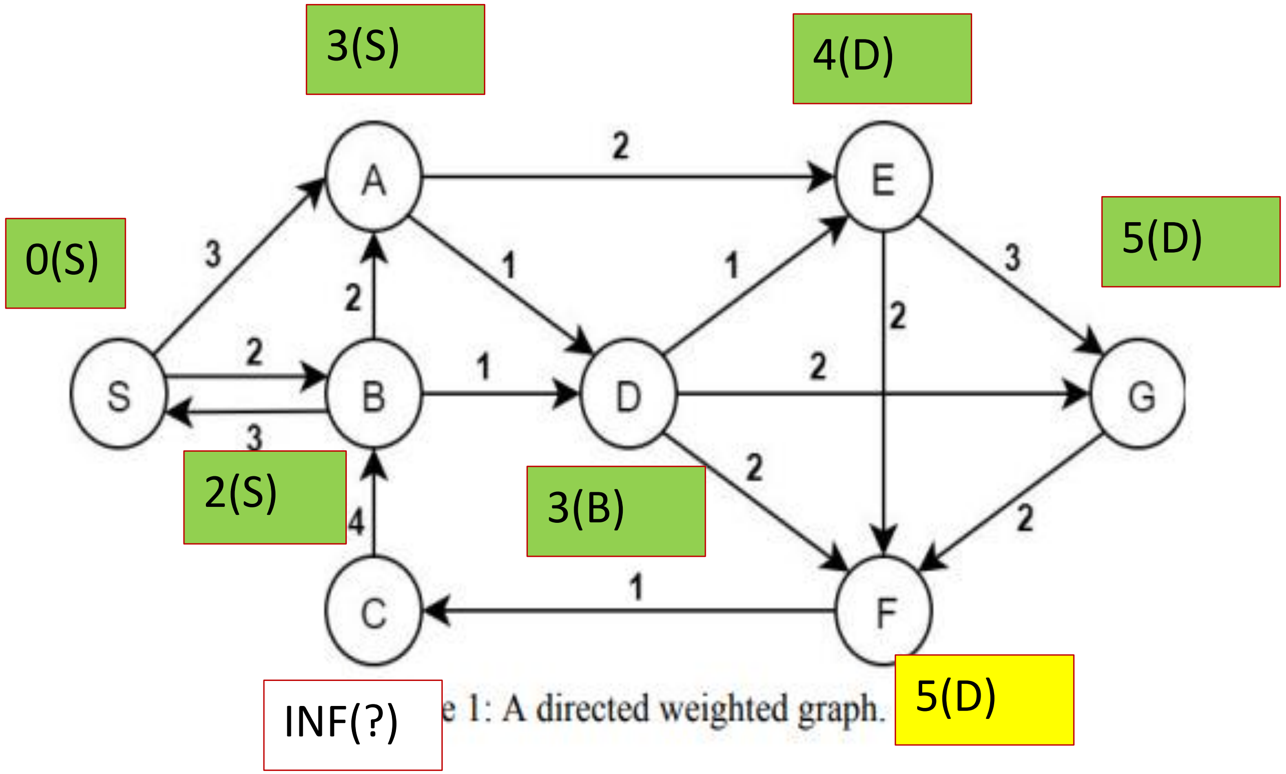
Q1- Dijkstra



e 1: A directed weighted graph.

Among all unknown nodes, G and F have the minimum distances. Choice is arbitrary. Choose G. Marked it as known.

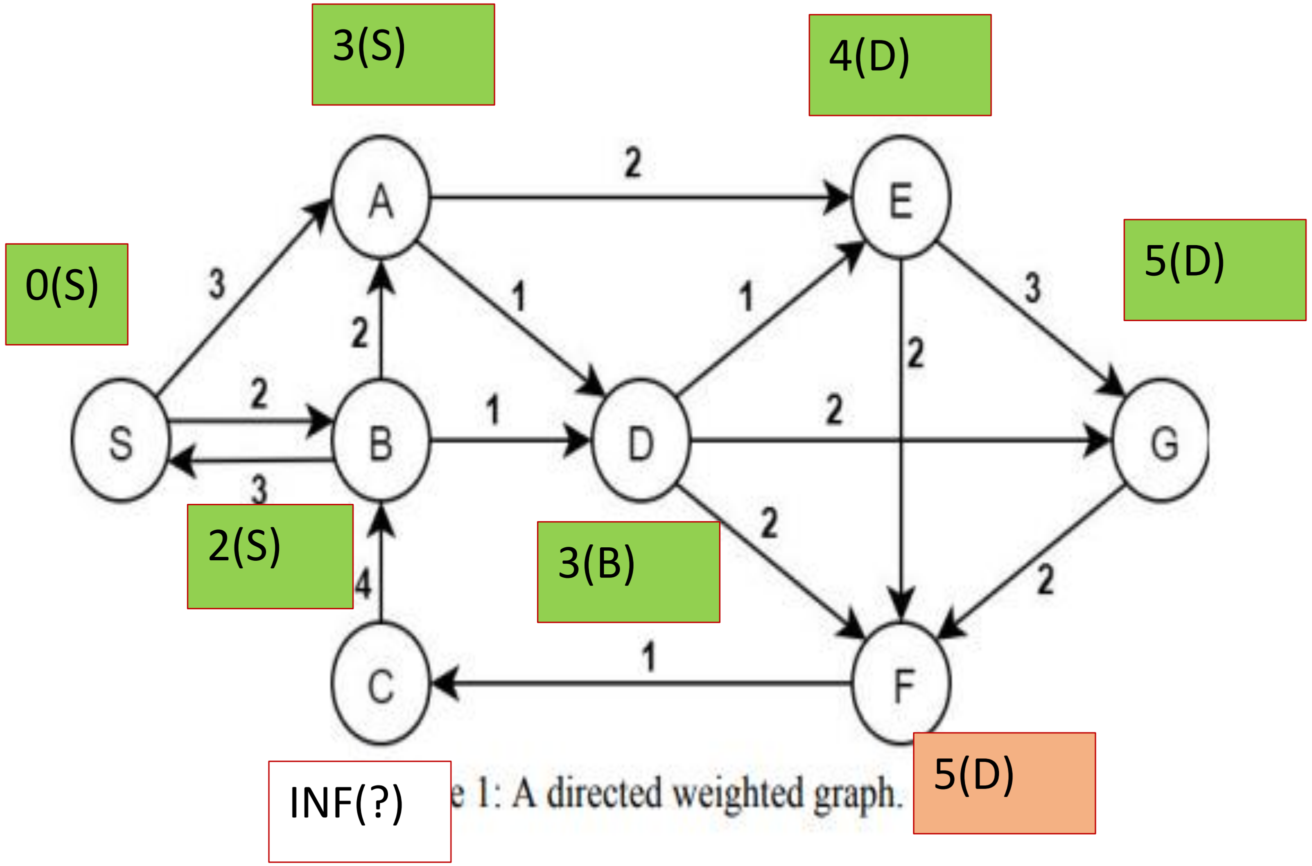
Q1- Dijkstra



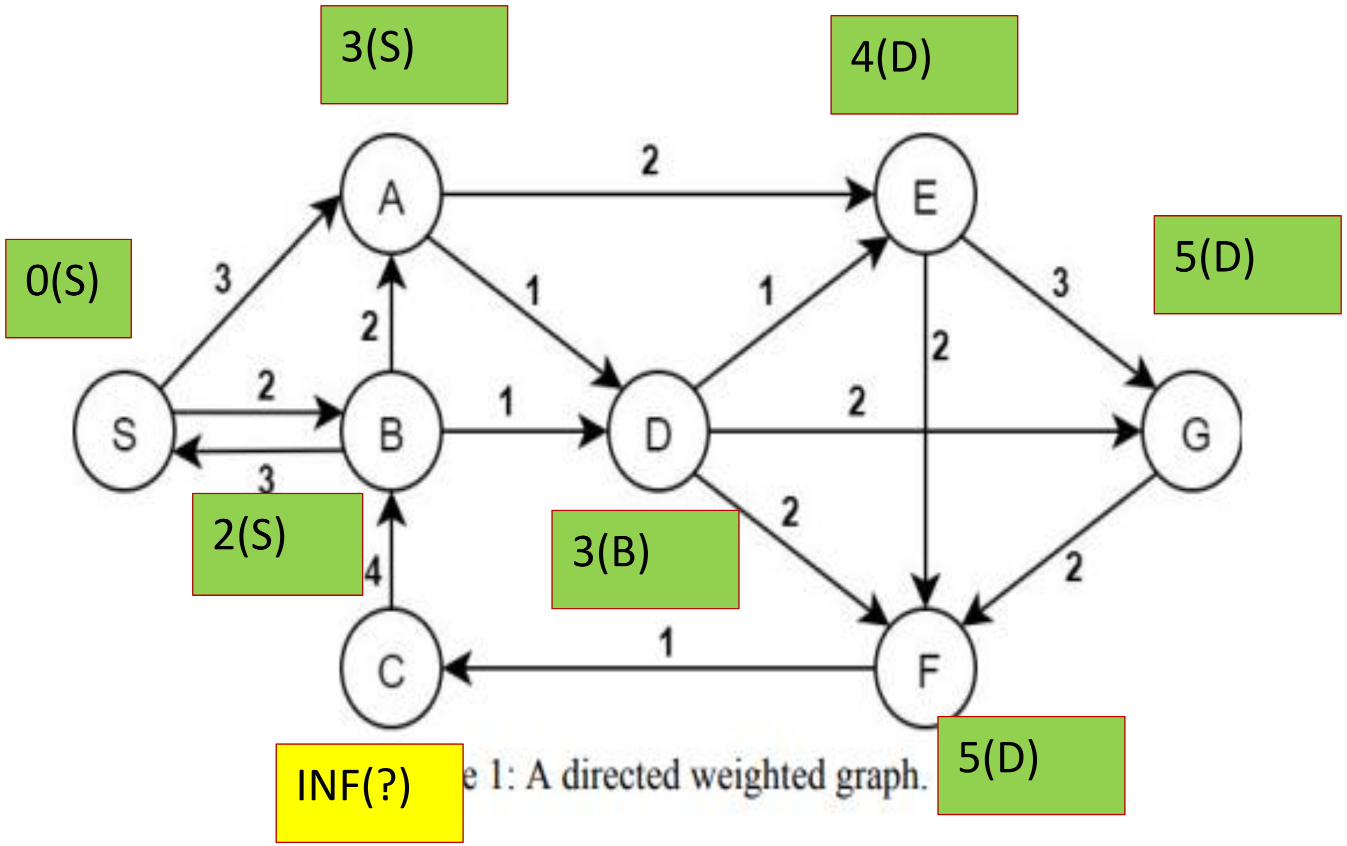
Node adjacent to G is F.

Q1- Dijkstra

We did not updated distances of F since G does not give a shorter path.



Q1- Dijkstra



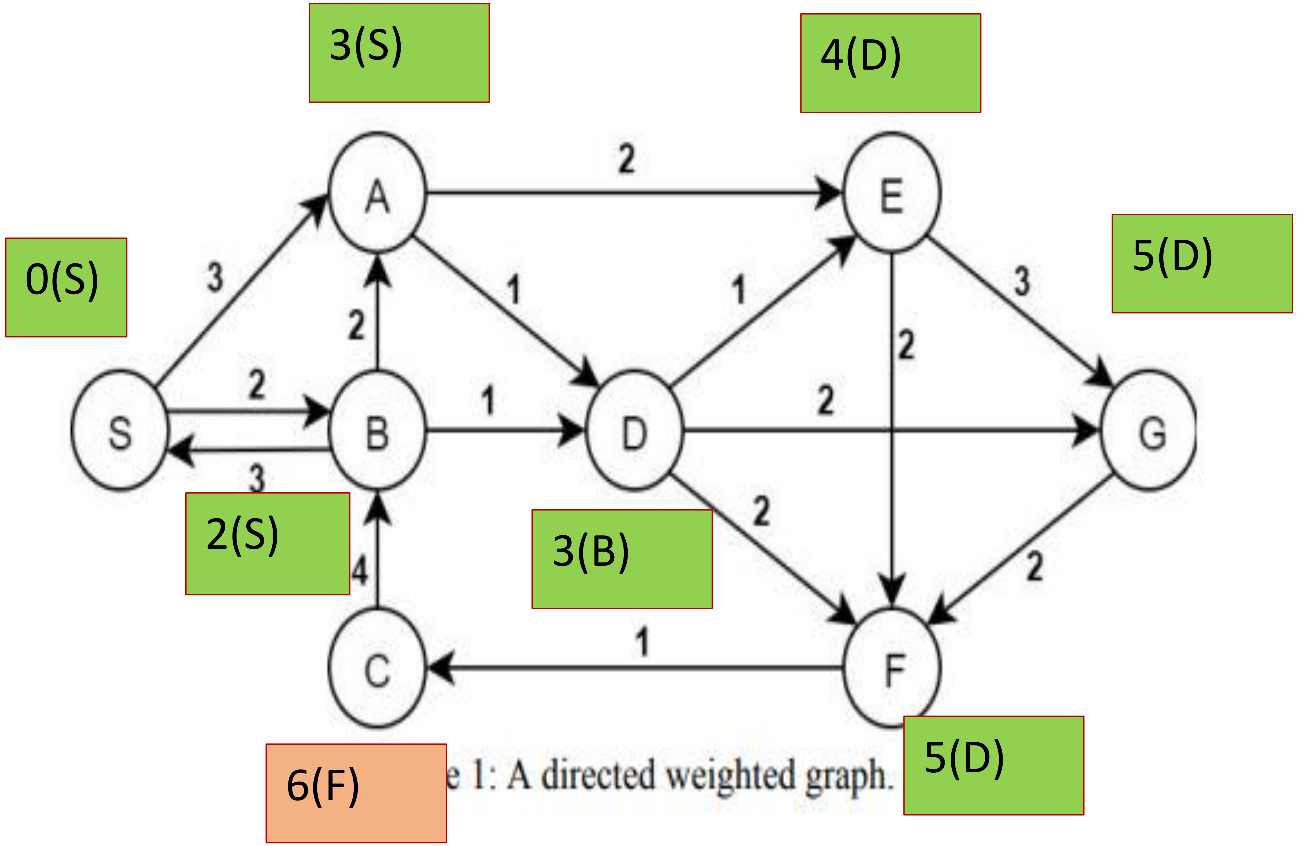
e 1: A directed weighted graph.

Among all unknown nodes, F has the minimum distance. Choose F. Marked it as known.

Node adjacent to F is C.

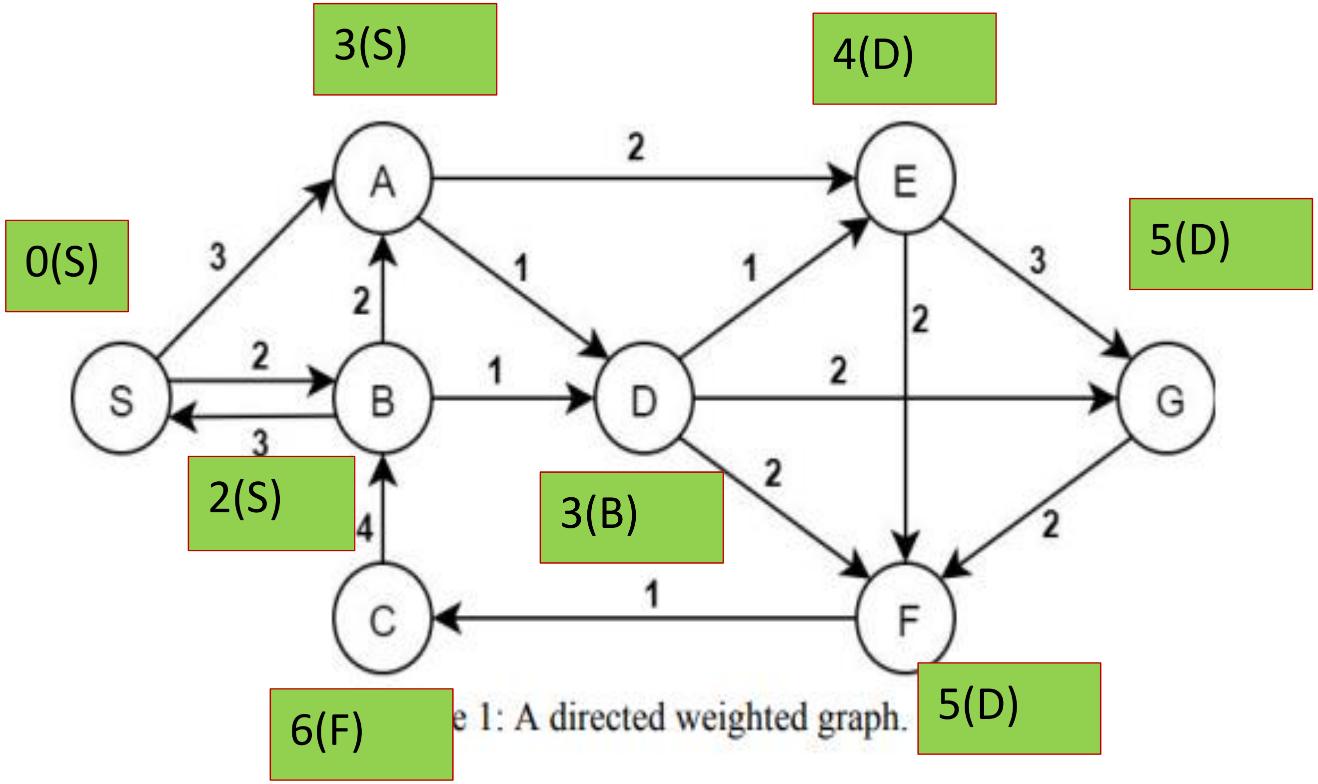
Q1- Dijkstra

We updated distance of C since F gives a shorter path.



e 1: A directed weighted graph.

Q1- Dijkstra

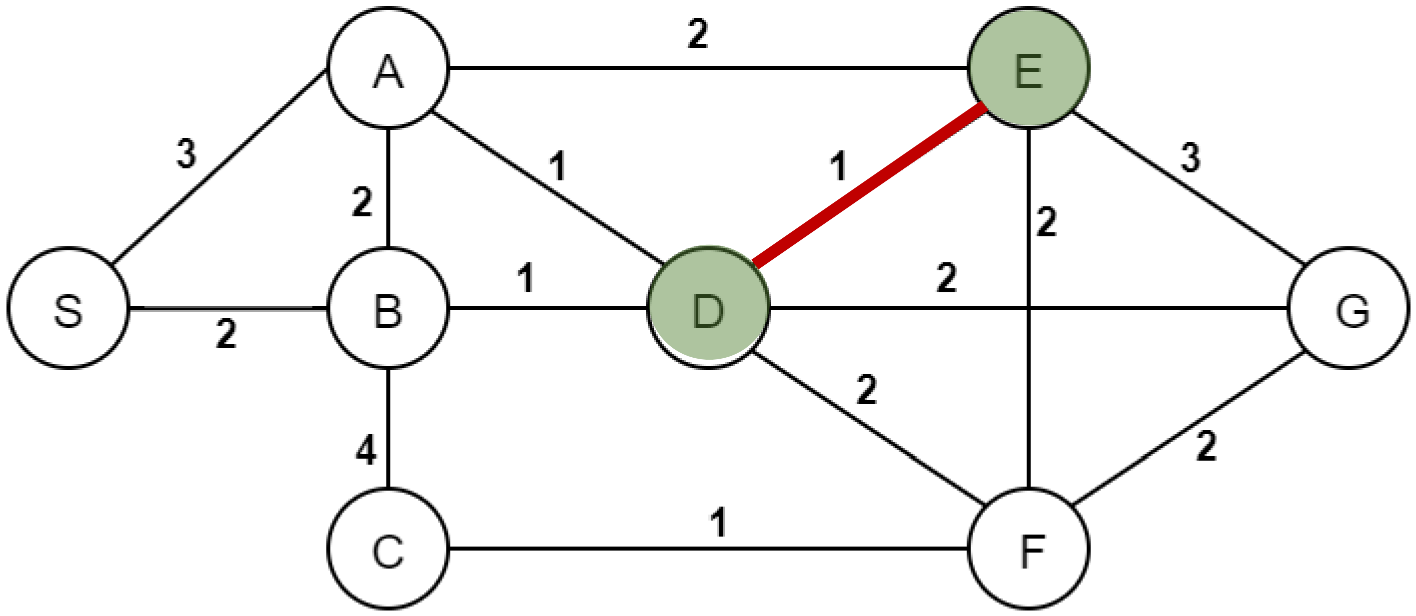


e 1: A directed weighted graph.

Among all unknown nodes, C has the minimum distance. Choose C. Marked it as known.

All vertices and distances are known now!

Q2: Prim’s MST



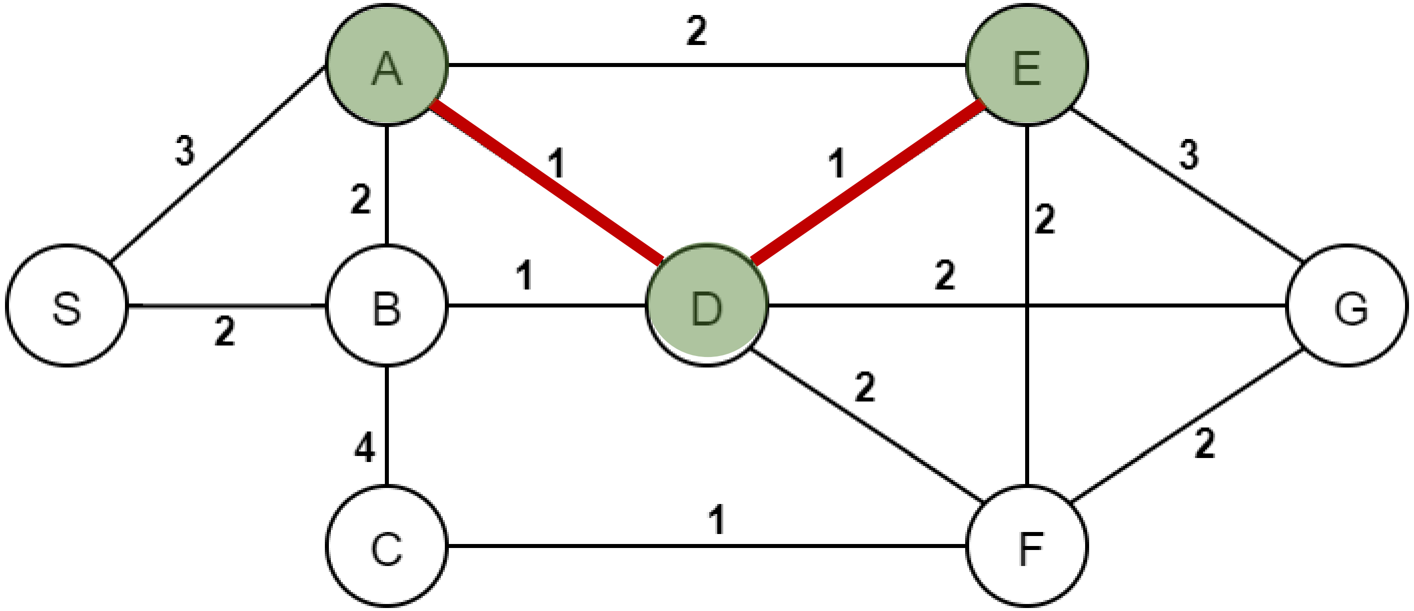
Select an edge with the mininum weight.

(Choice is arbitrary since there are multiple edges with weight 1.)

Select edge between E,D and add E and D to the tree.

Min distances of vertices to the tree
A: 1
B: 1
C: 3
S: 3
F: 2
G: 2

Q2: Prim’s MST



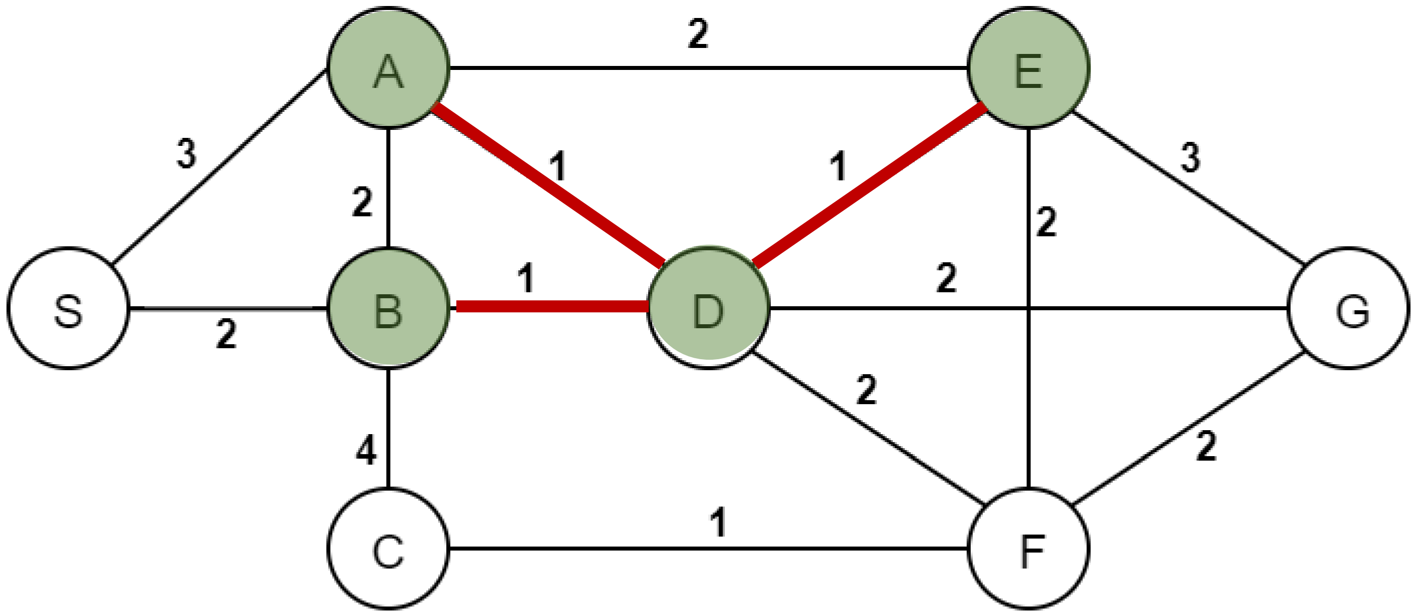
Choose the vertex that is not in the tree but closest to the tree.

A,B are such vertices.
Choice is arbitrary.

Choose A.

Distances of vertices to the tree
B: 1
C: 3
S: 3
F: 2
G: 2

Q2: Prim’s MST

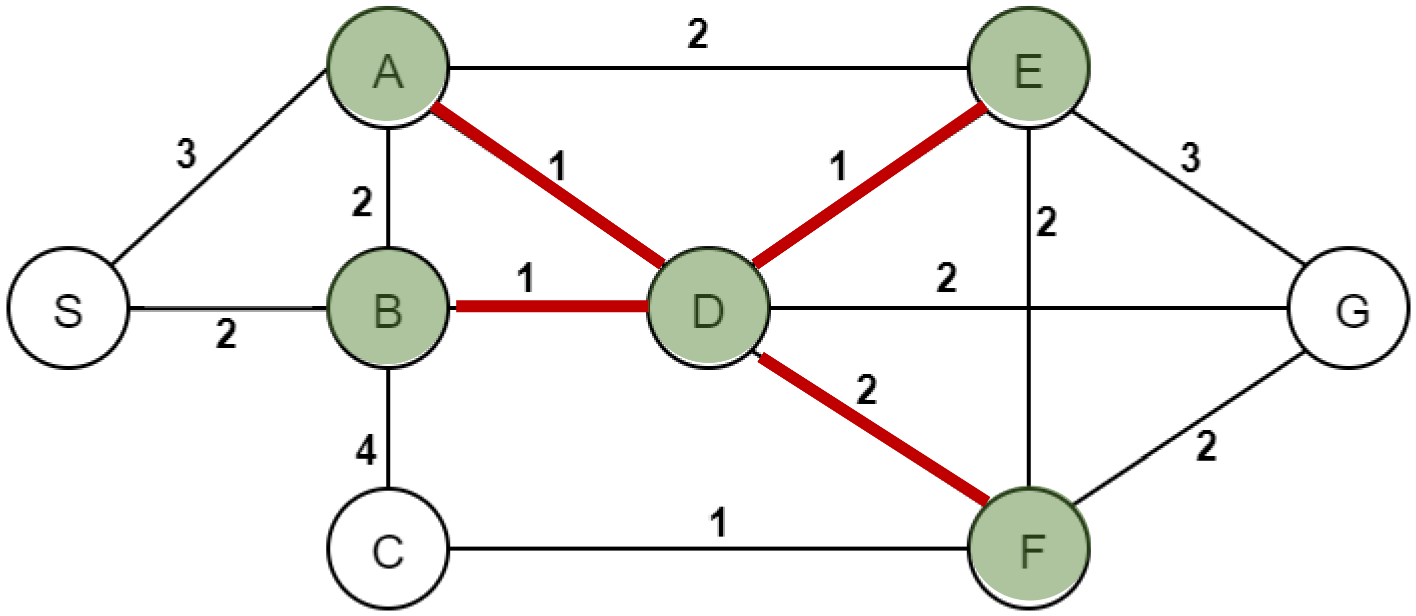


Choose the vertex that is not in the tree but closest to the tree.

B is that vertex. Add B to the tree.

Distances of vertices to the tree
C: 3
S: 2
F: 2
G: 2

Q2: Prim's MST

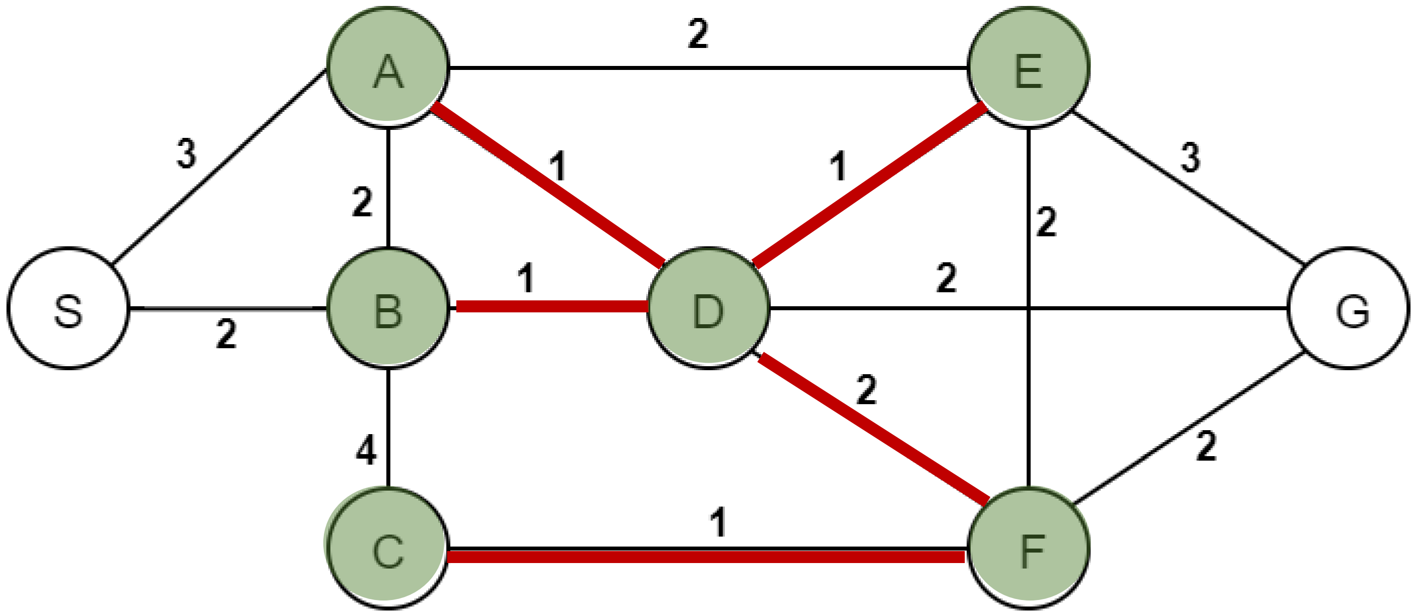


Choose the vertex that is not in the tree but closest to the tree.

F,G,S are such vertices. Choice is arbitrary. Choose F and add it to the tree.

Distances of vertices to the tree
C: 1
S: 2
G: 2

Q2: Prim's MST

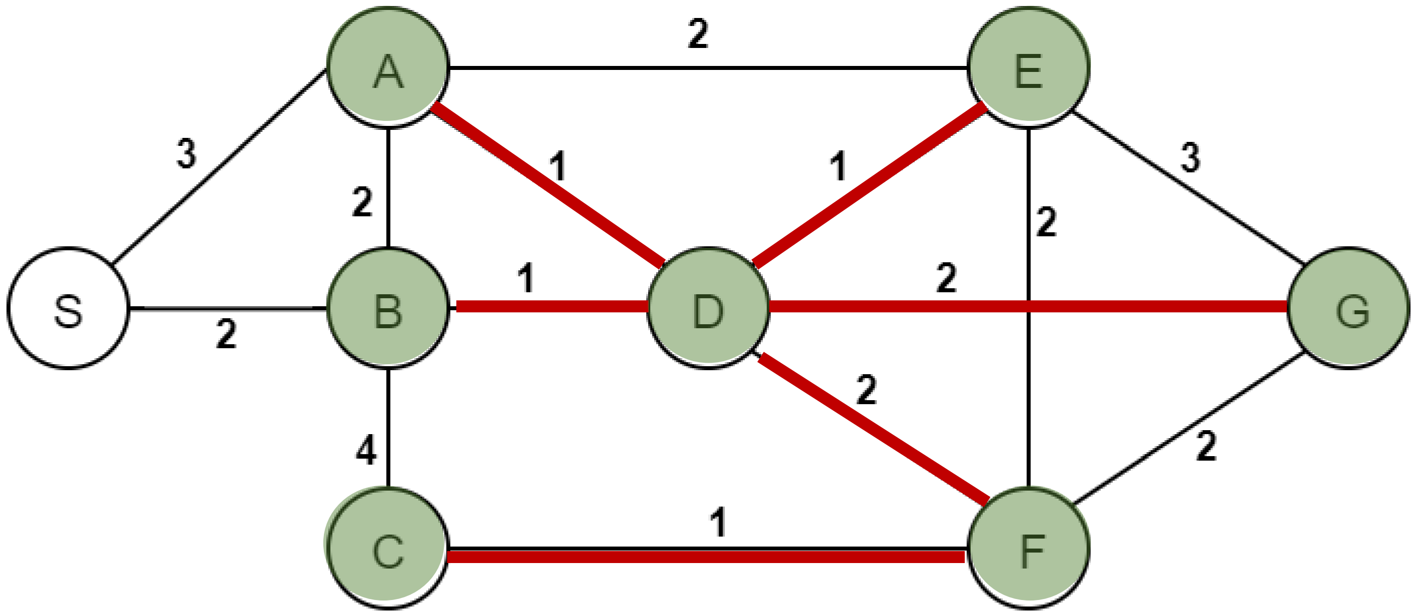


Choose the vertex that is not in the tree but closest to the tree.

Choose C and add it to the tree.

Distances of vertices to the tree
S: 2
G: 2

Q2: Prim’s MST

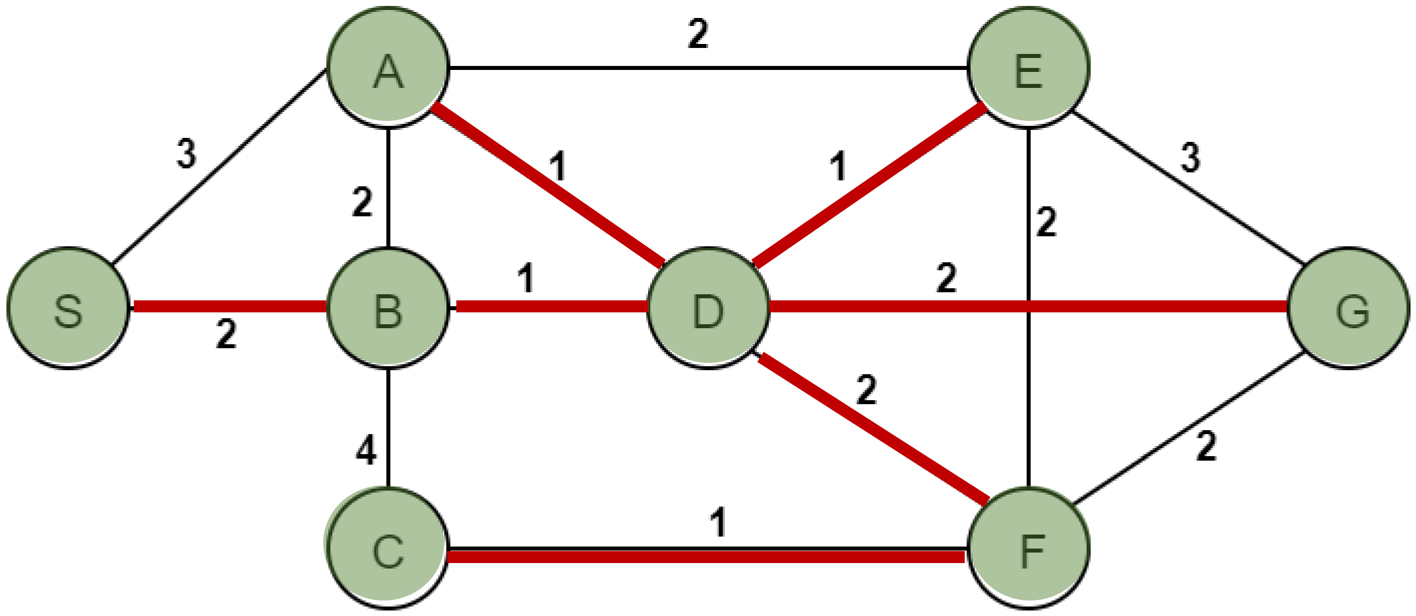


Choose the vertex that is not in the tree but closest to the tree.

Choose G and add it to the tree.

Distances of vertices to the tree
S: 2

Q2: Prim’s MST

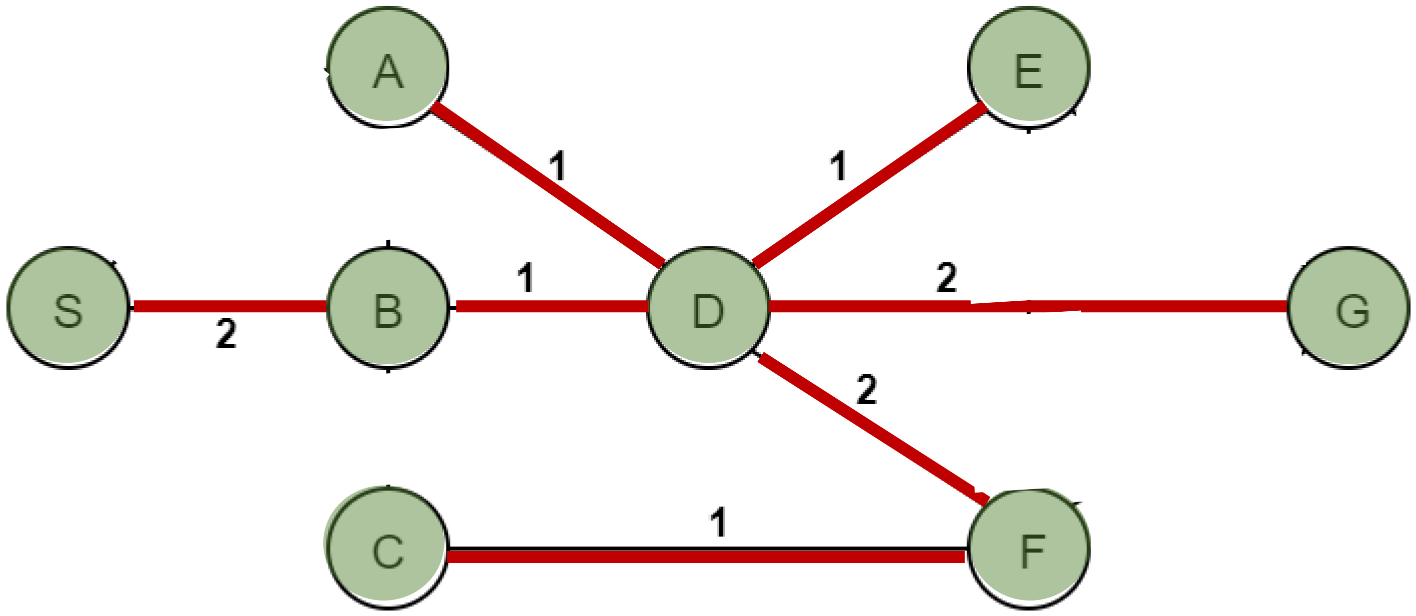


Choose the vertex that is not in the tree but closest to the tree.

Choose S and add it to the tree.

All vertices are added.

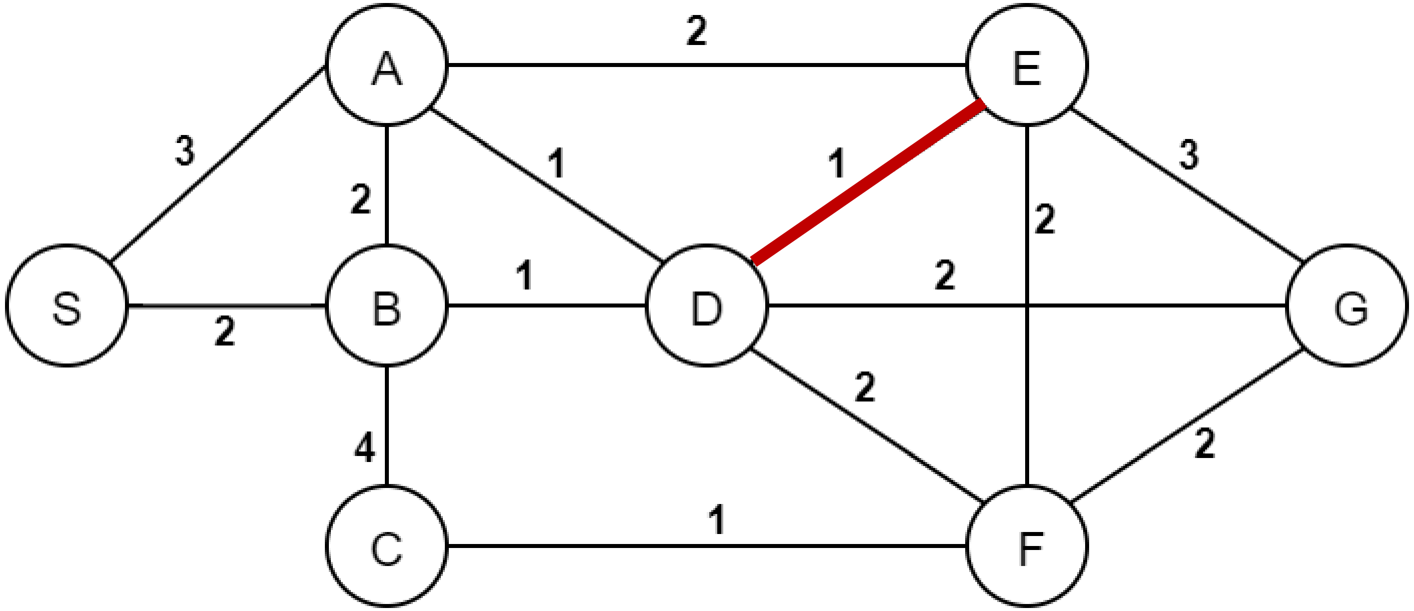
Q2: Prim's MST



This is our MST.

Total Cost: 10

Q3: Kruskal's MST



-1	**-1**	**-1**	**-1**	**-2**	**4**	**-1**	**-1**
S	**A**	**B**	**C**	**D**	**E**	**F**	**G**

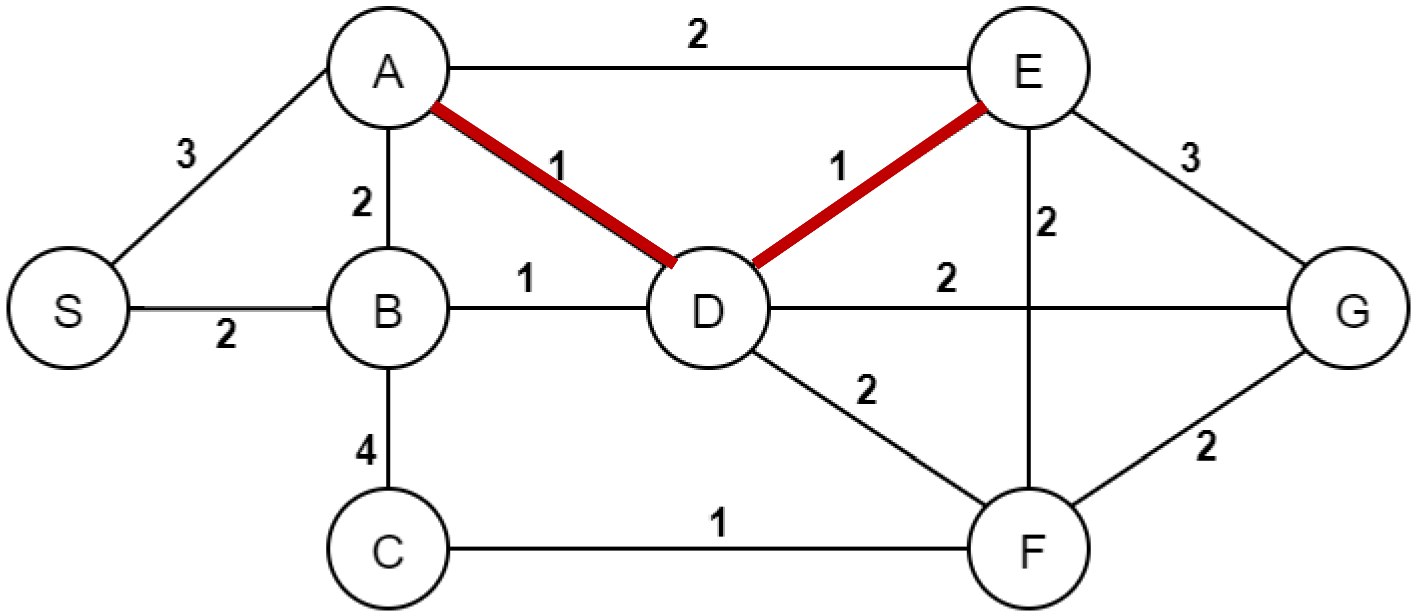
Equivalence classes of vertices

Select an edge with the mininum weight.

(Choice is arbitrary since there are multiple edges with weight 1.)

Select edge between E,D. Union E and D in disjoint set.

Q3: Kruskal



-1	4	-1	-1	-3	4	-1	-1
S	A	B	C	D	E	F	G

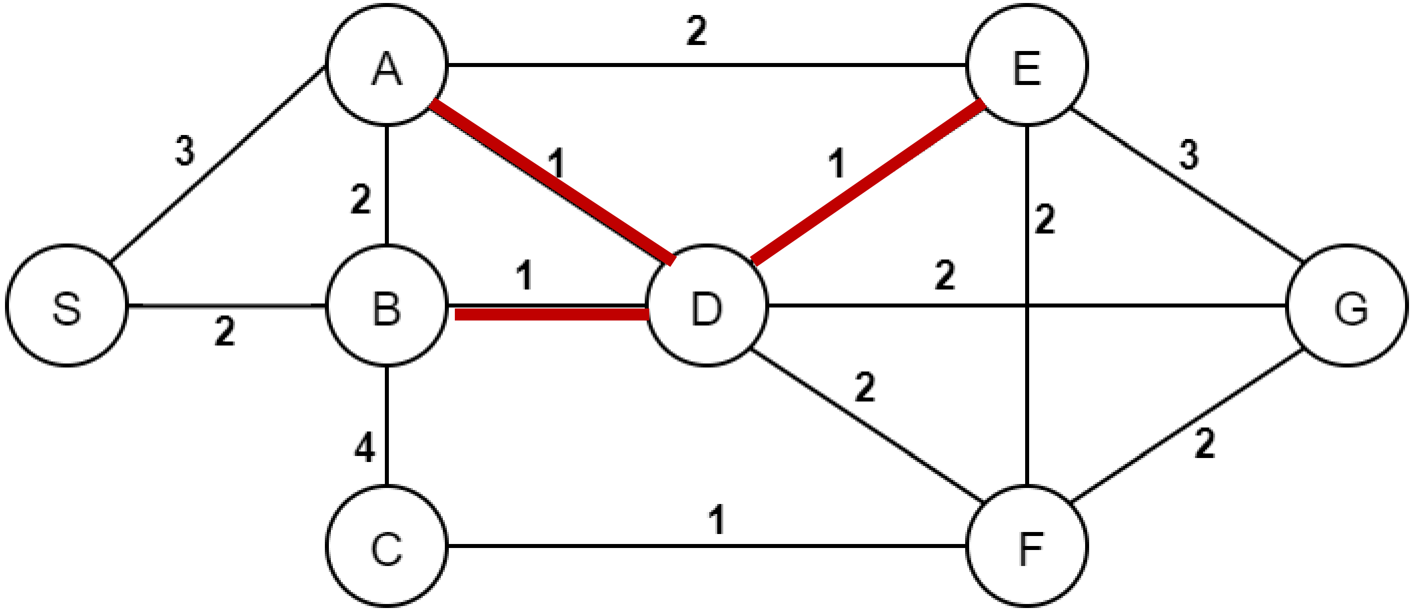
Equivalence classes of vertices

Select an edge with the mininum weight.

(Choice is arbitrary since there are multiple edges with weight 1.)

Select edge between A,D. Union A and D in disjoint set.

Q3: Kruskal



Select an edge with the mininum weight.

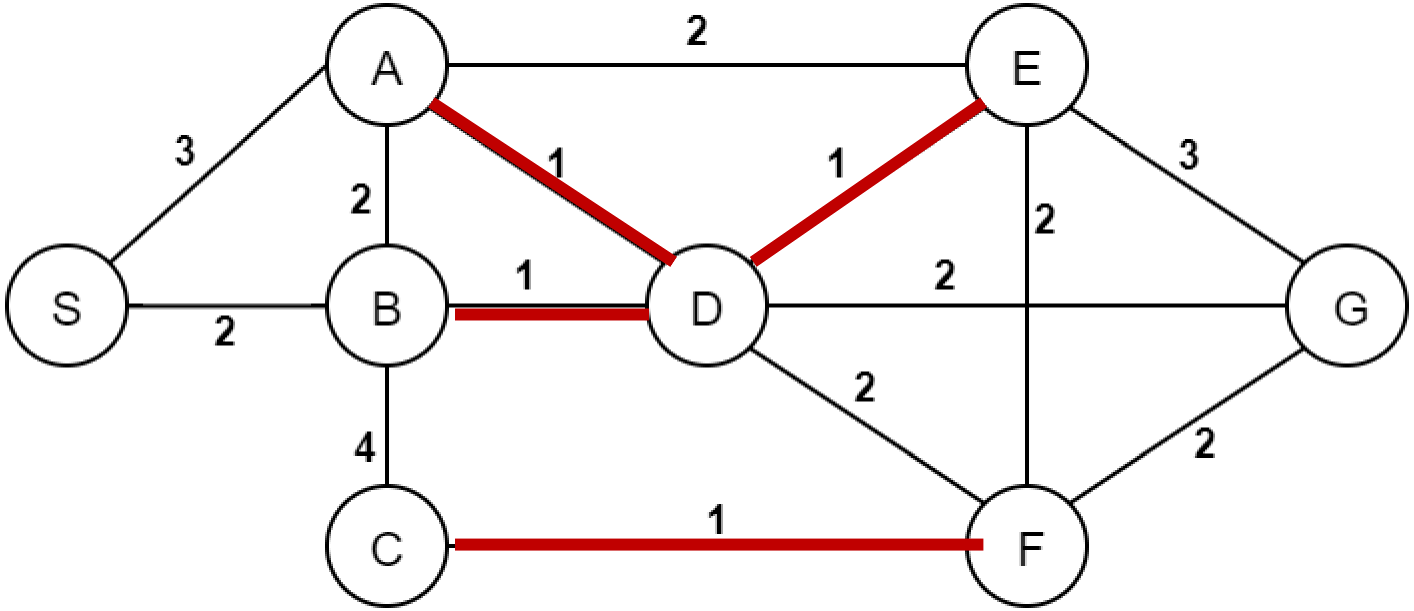
(Choice is arbitrary since there are multiple edges with weight 1.)

Select edge between B,D. Union B and D in disjoint set.

-1	**4**	**4**	**-1**	**-4**	**4**	**-1**	**-1**
S	**A**	**B**	**C**	**D**	**E**	**F**	**G**

Equivalence classes of vertices

Q3: Kruskal



-1	**4**	**4**	**-2**	**-4**	**4**	**3**	**-1**
S	**A**	**B**	**C**	**D**	**E**	**F**	**G**

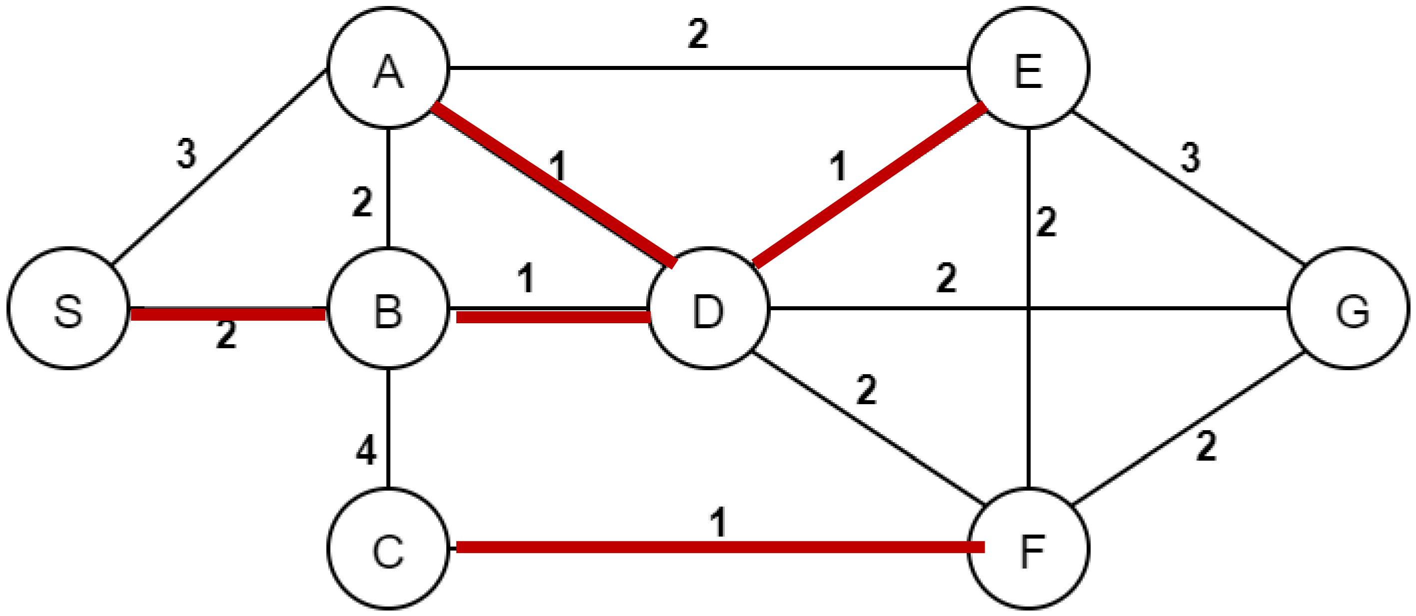
Equivalence classes of vertices

Select an edge with the mininum weight.

(Min weighted edge is between C and F.)

Select edge between C,F. Union them.

Q3: Kruskal



Select an edge with the mininum weight.

(choice is arbitrary since there are multiple edges with weight 2.)

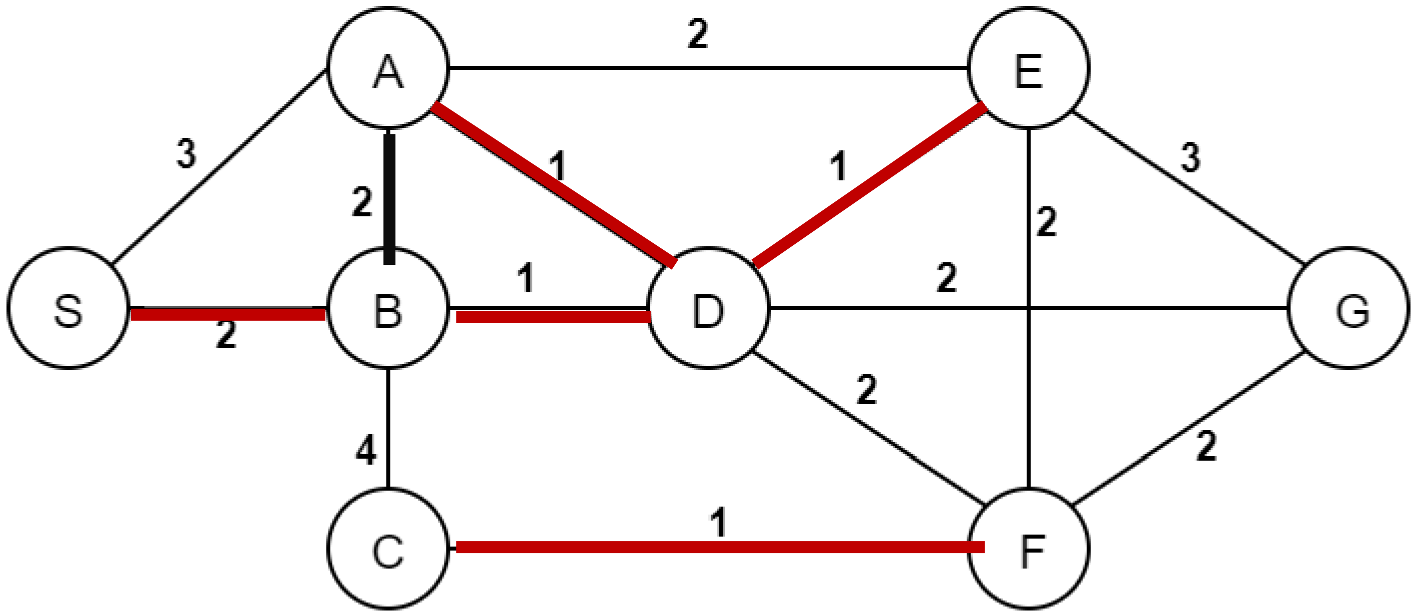
Select edge between S,B. Union them.

4	**4**	**4**	**-2**	**-5**	**4**	**3**	**-1**

S	**A**	**B**	**C**	**D**	**E**	**F**	**G**

Equivalence classes of vertices

Q3: Kruskal



Select an edge with the mininum weight.

(choice is arbitrary since there are multiple edges with weight 2.)

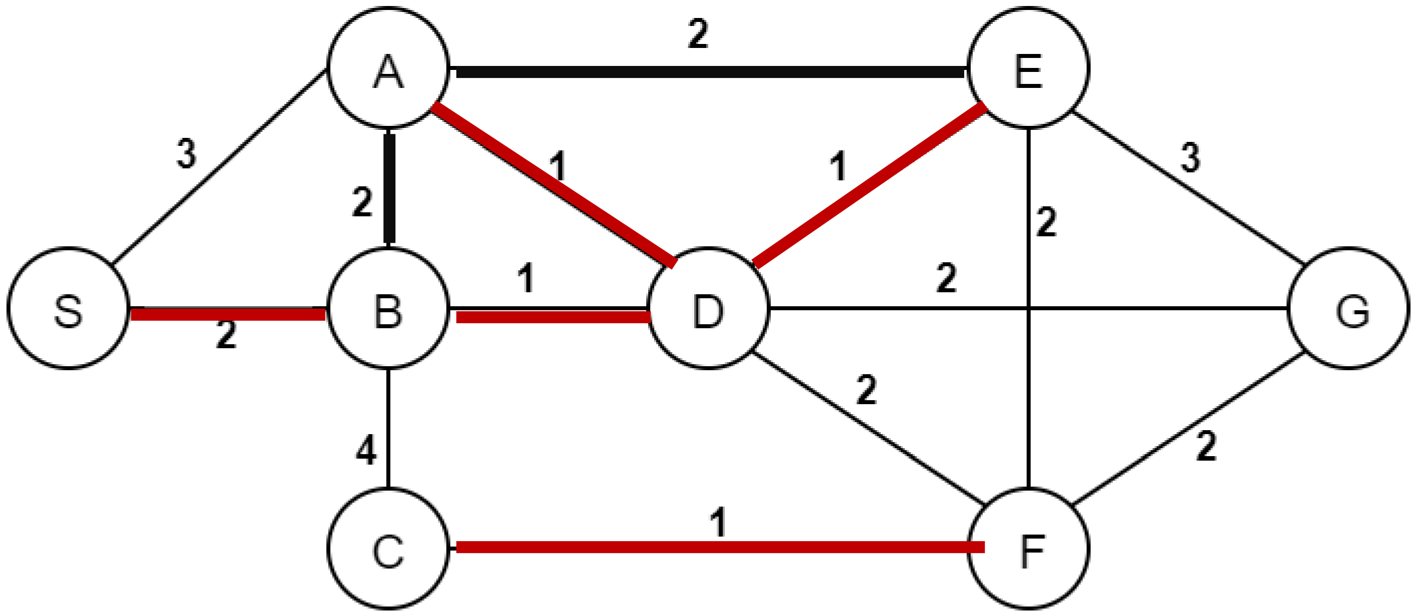
WE CANNOT SELECT edge between A and B because A and B are already in the same equivalence class. This edge causes cycle.

Mark this edge thick. Dont add to tree.

4	**4**	**4**	**-2**	**-5**	**4**	**3**	**-1**
S	**A**	**B**	**C**	**D**	**E**	**F**	**G**

Equivalence classes of vertices

Q3: Kruskal



Select an edge with the mininum weight.

(choice is arbitrary since there are multiple edges with weight 2.)

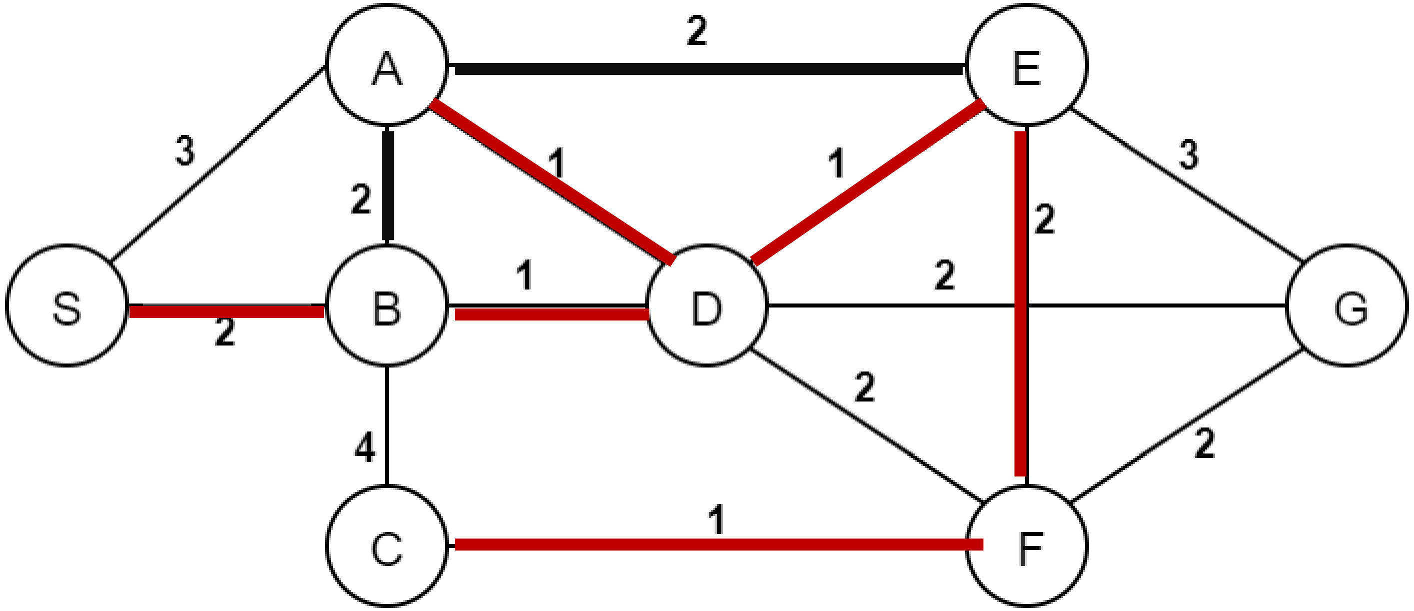
WE CANNOT SELECT edge between A and E because A and E are already in the same equivalence class. This edge causes cycle.

Mark this edge thick. Dont add to tree.

4	**4**	**4**	**-2**	**-5**	**4**	**3**	**-1**
S	**A**	**B**	**C**	**D**	**E**	**F**	**G**

Equivalence classes of vertices

Q3: Kruskal



4	**4**	**4**	**4**	**-7**	**4**	**4**	**-1**
S	**A**	**B**	**C**	**D**	**E**	**F**	**G**

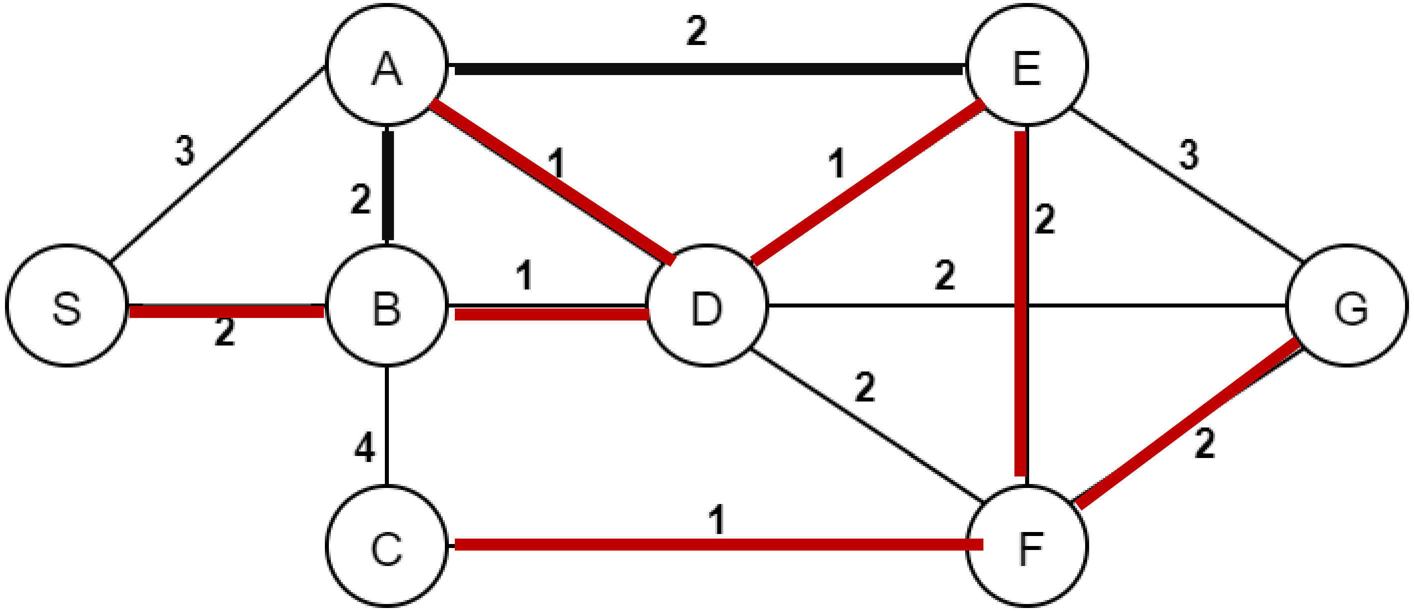
Equivalence classes of vertices

Select an edge with the mininum weight.

(choice is arbitrary since there are multiple edges with weight 2.)

Select edge between E and F. Union them.

Q3: Kruskal



Select an edge with the mininum weight.

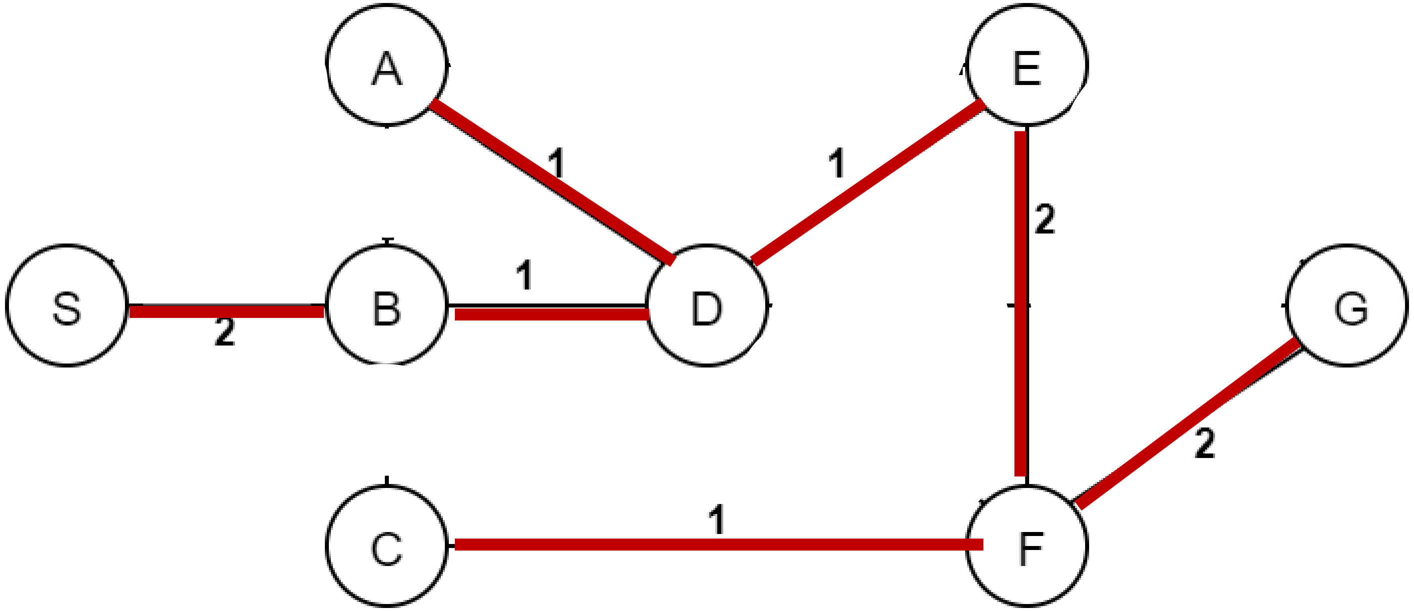
(choice is arbitrary since there are multiple edges with weight 2.)

Select edge between G and F. Union them.

4	**4**	**4**	**4**	**-8**	**4**	**4**	**4**
S	**A**	**B**	**C**	**D**	**E**	**F**	**G**

Equivalence classes of vertices

Q3: Kruskal



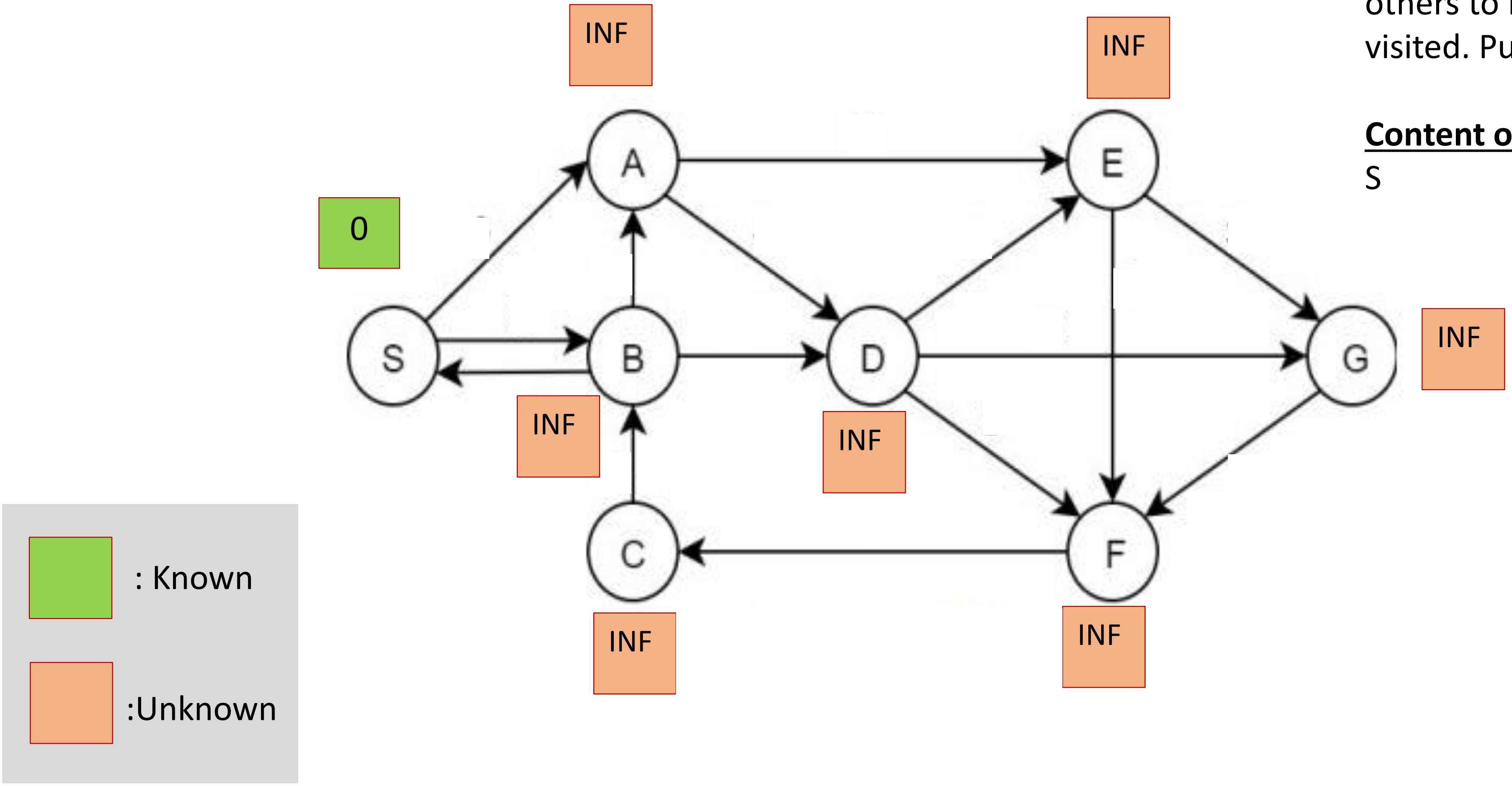
We acceptted 7 edges so far. Since we have 8 vertices in total and all vertices are now in the same equivalence class, the algorithm completed.

Total weight of edges in our MST: 10.

4	**4**	**4**	**4**	**-8**	**4**	**4**	**4**
S	**A**	**B**	**C**	**D**	**E**	**F**	**G**

Equivalence classes of vertices

Q4- BFS



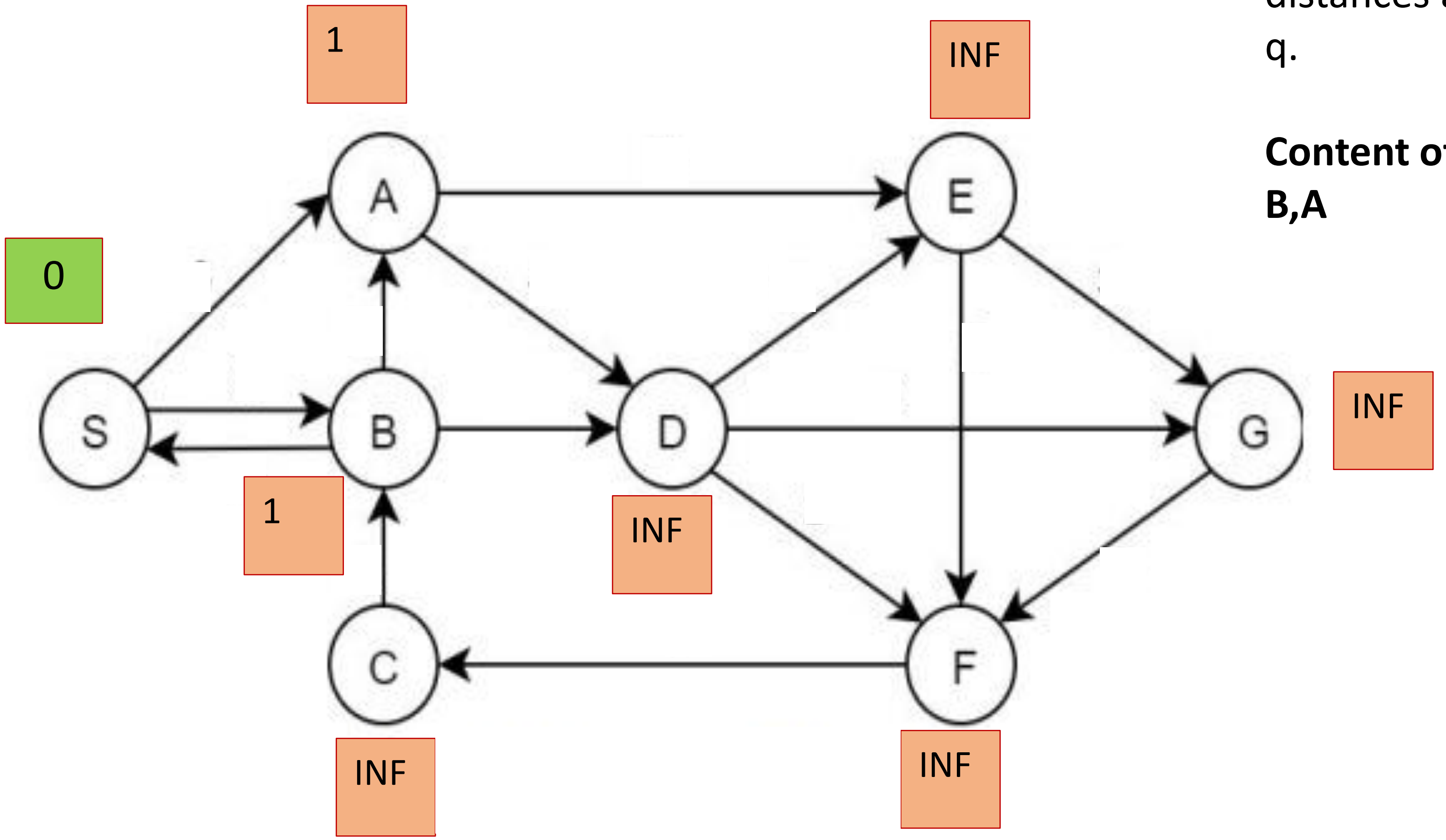
Queue containing vertices that are visited for the first time: q

Initialize distance of S to 0 and others to INF. Mark S as visited. Push it to q.

Content of q:

S

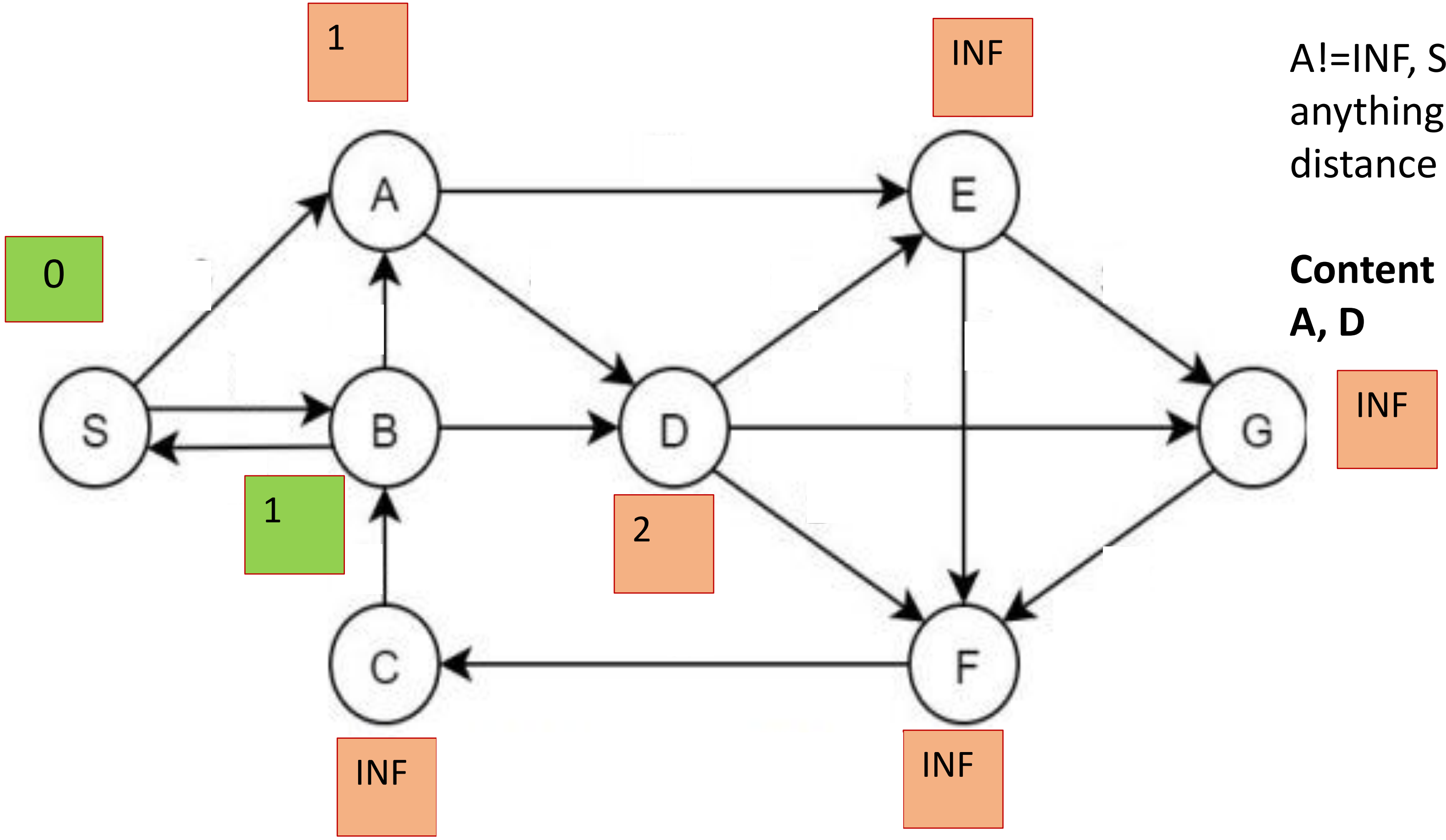
Q4- BFS



Dequeue S. Visit vertices adjacent to S (A,B). Since their distance is INF update distances and add them to q.

Content of q:
B,A

Q4- BFS

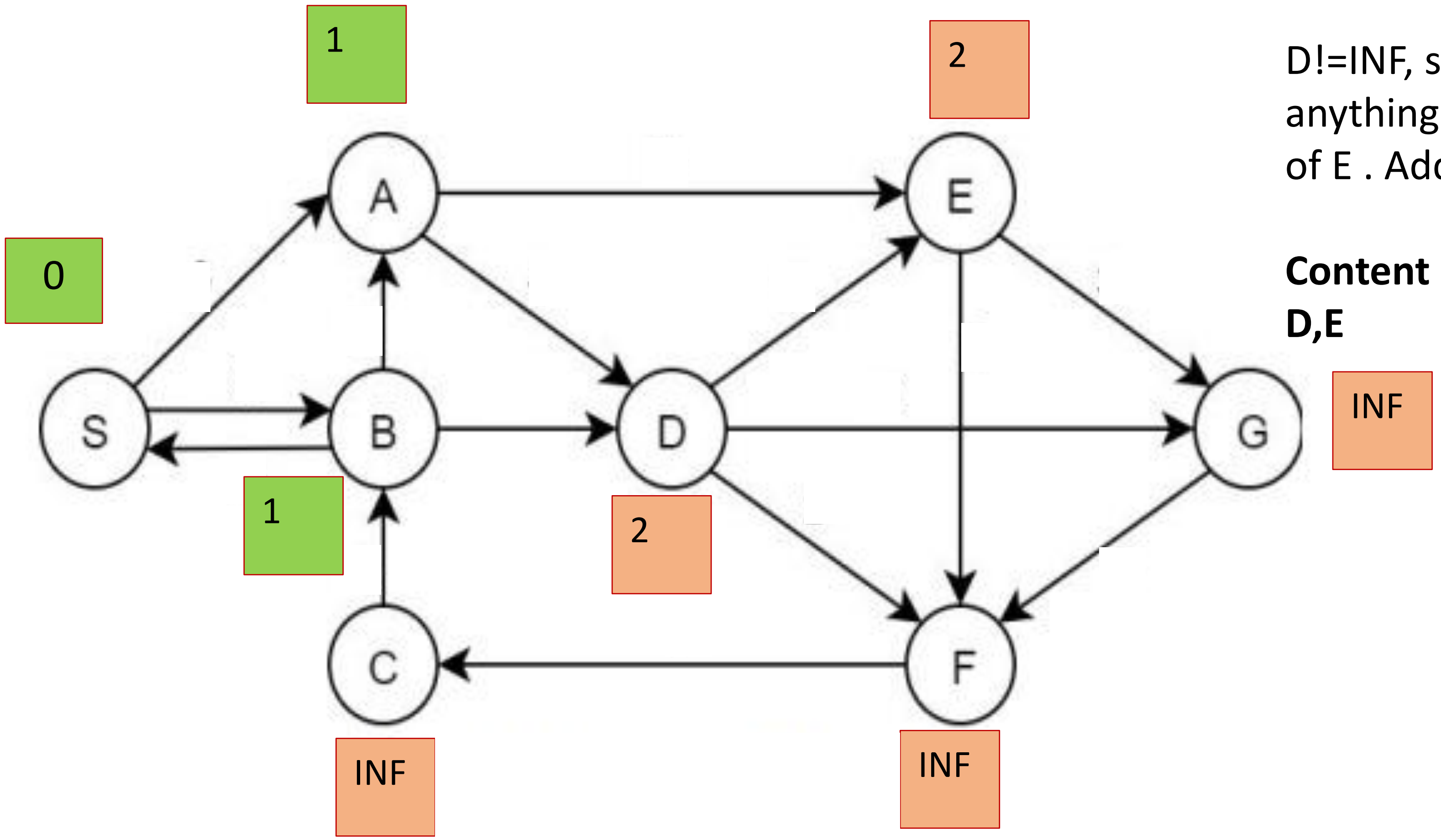


Deque B. Marked it as visited. Visit vertices adjacent to (A,D,S).

A!=INF, S!=INF. So don't do anything for them. Update distance of D . Add it to q.

Content of q:
A, D

Q4- BFS

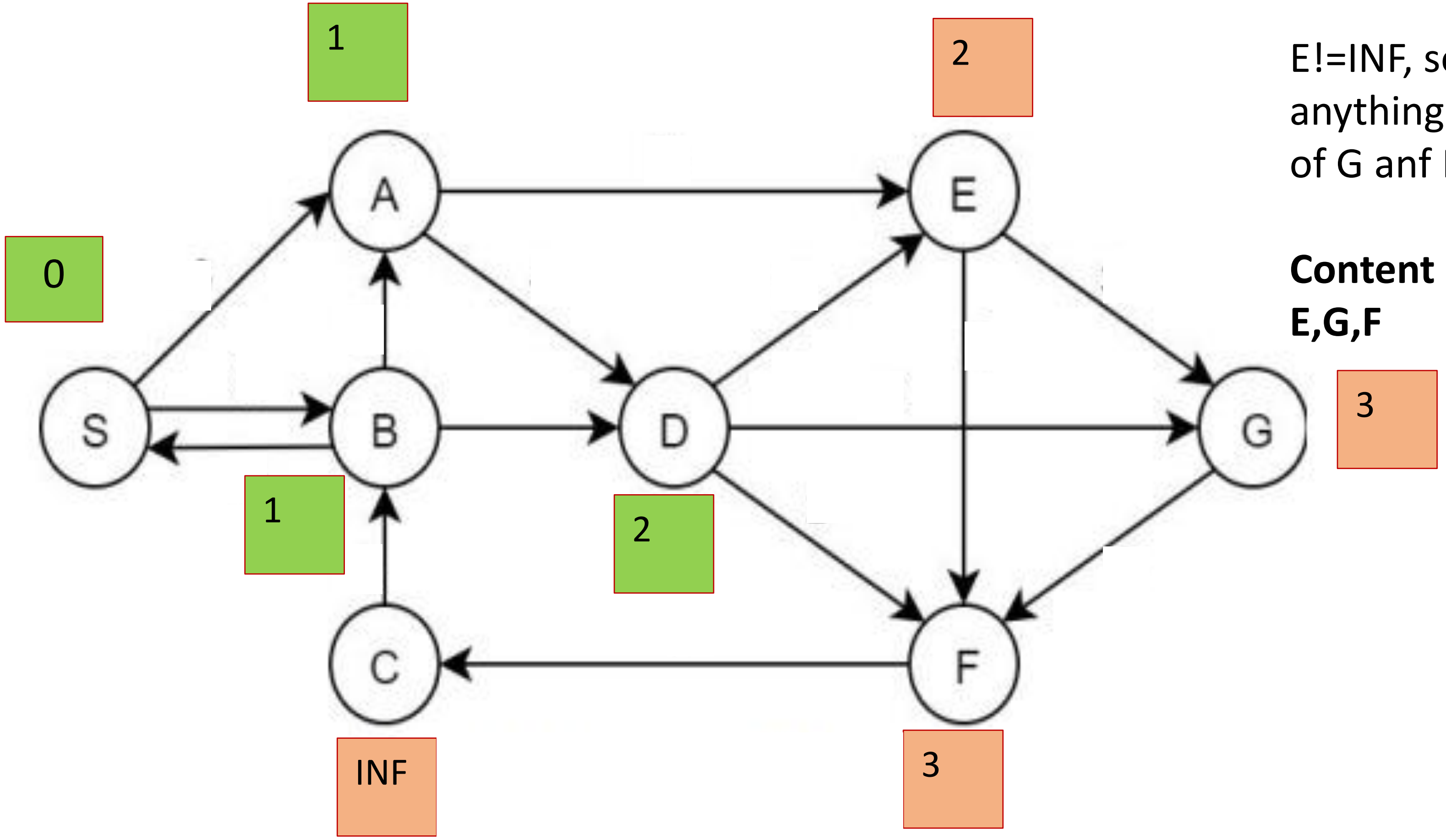


Dequeue A. Mark it as visited. Visit vertices adjacent to (E,D).

D!=INF, so don't do anything. Update distance of E . Add it to q.

Content of q:
D,E

Q4- BFS

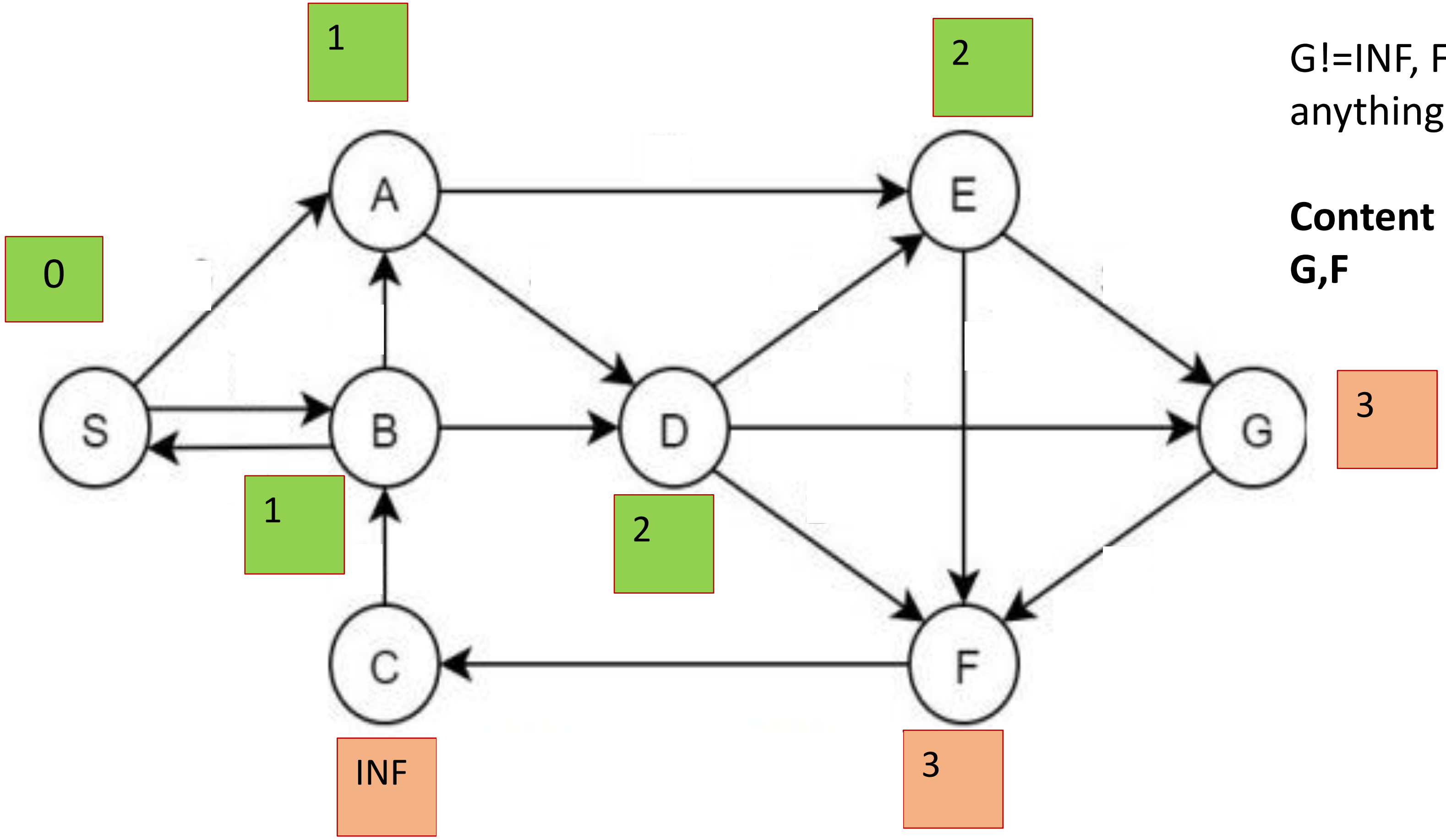


Dequeue D. Mark it as visited. Visit vertices adjacent to (E,G,F).

E!=INF, so don’t do anything. Update distances of G anf F. Add them to q.

Content of q:
E,G,F

Q4- BFS

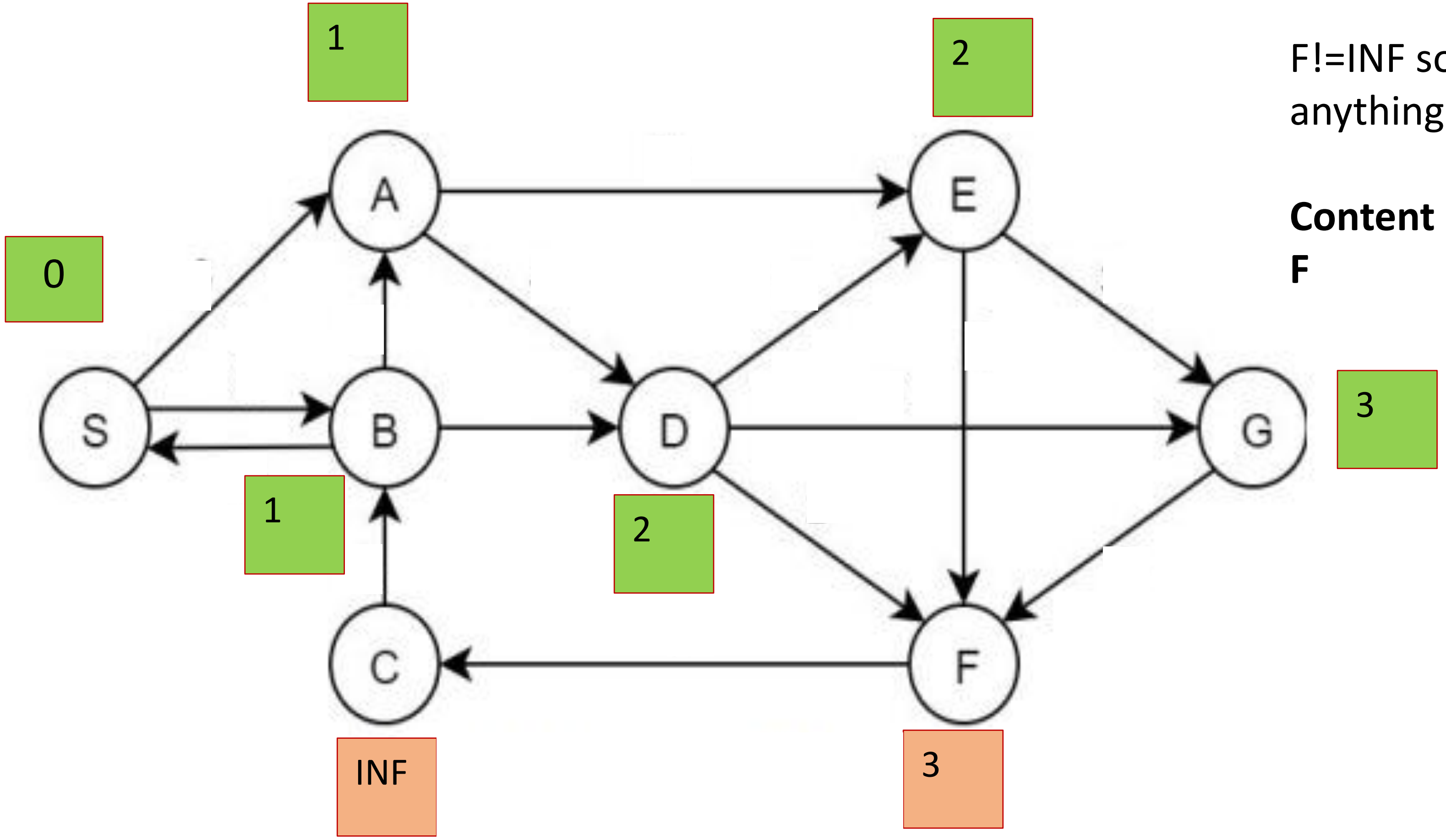


Dequeue E. Mark it as visited. Visit vertices adjacent to (G,F).

G!=INF, F!=INF so don't do anything.

Content of q:
G,F

Q4- BFS

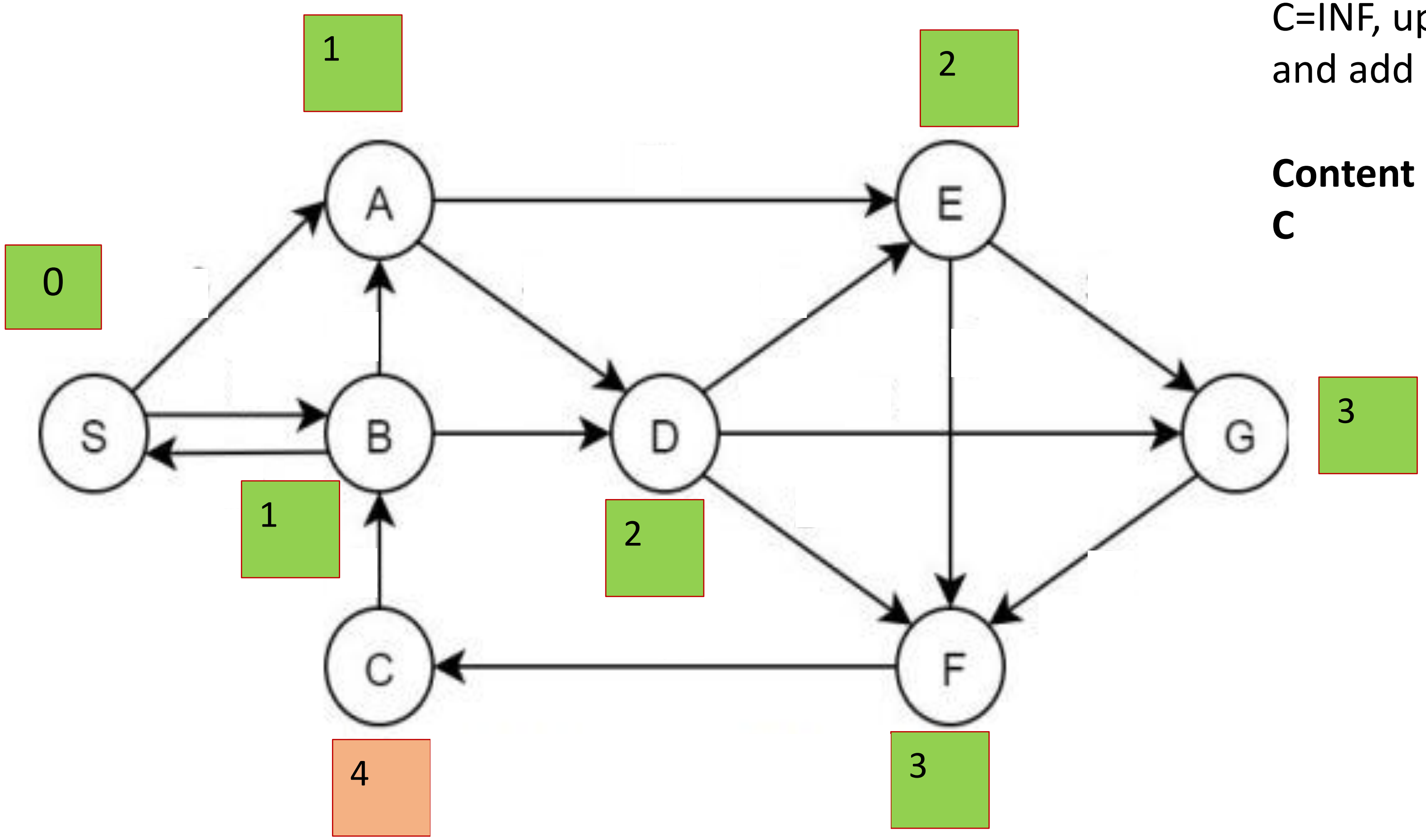


Dequeue G. Mark it as visited. Visit vertices adjacent to it. (F).

F!=INF so don't do anything.

Content of q:
F

Q4- BFS

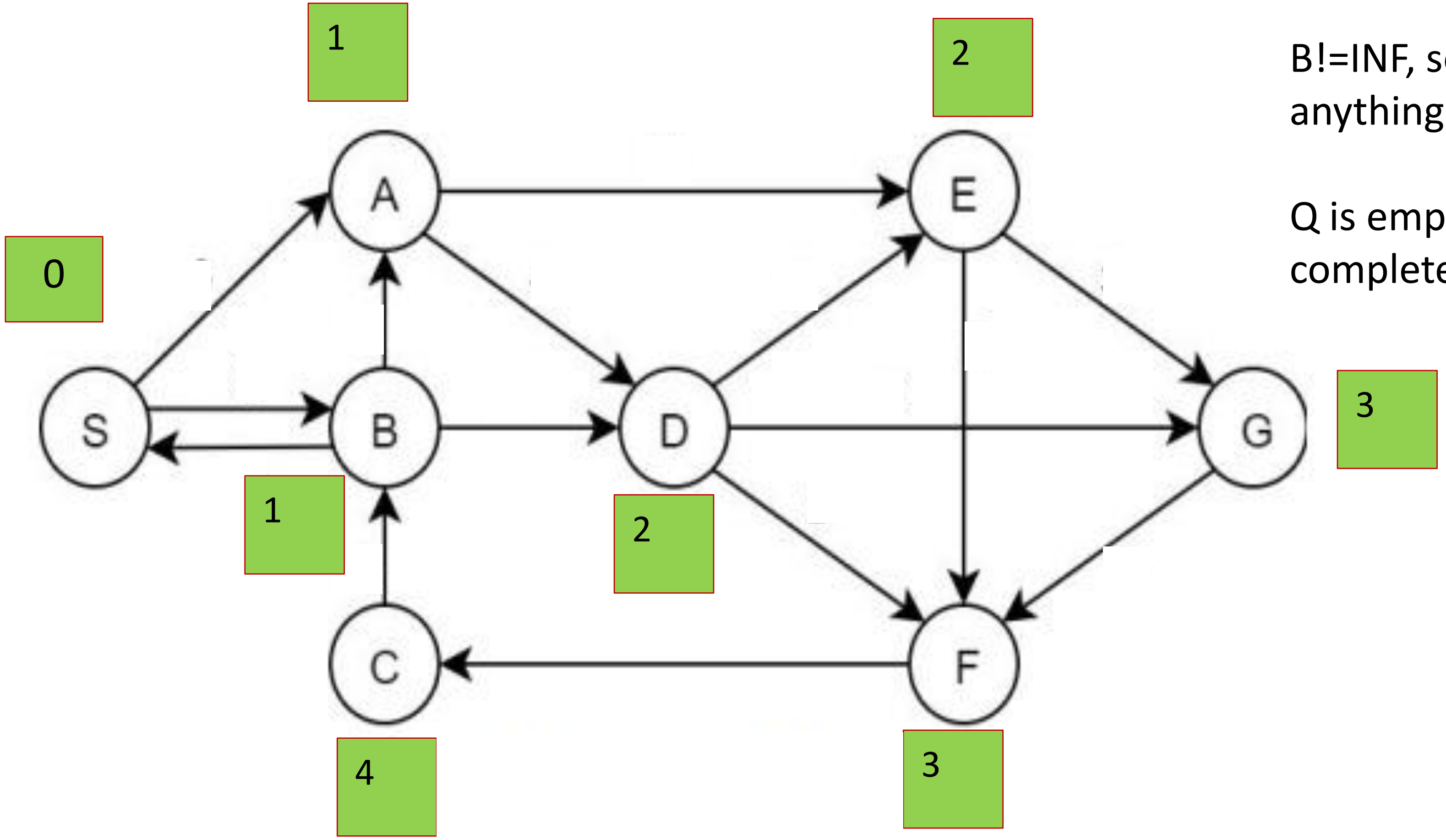


Dequeue F. Visit vertices adjacent to it (C).

C=INF, update its distance and add it to q.

Content of q:
C

Q4- BFS

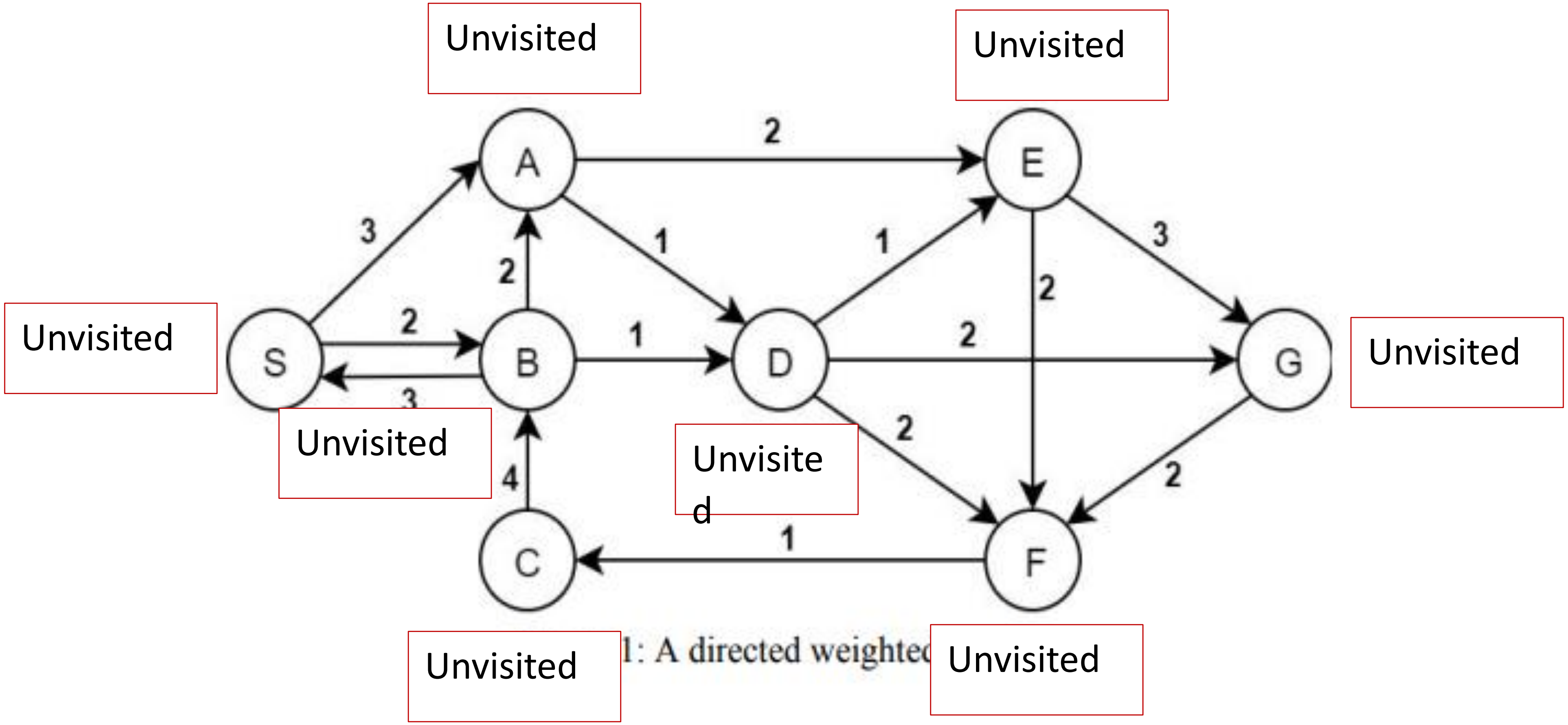


Dequeue C. Mark it as visited. Visit vertices adjacent to it (B).

B!=INF, so dont do anything.

Q is empty. BFS is complete!

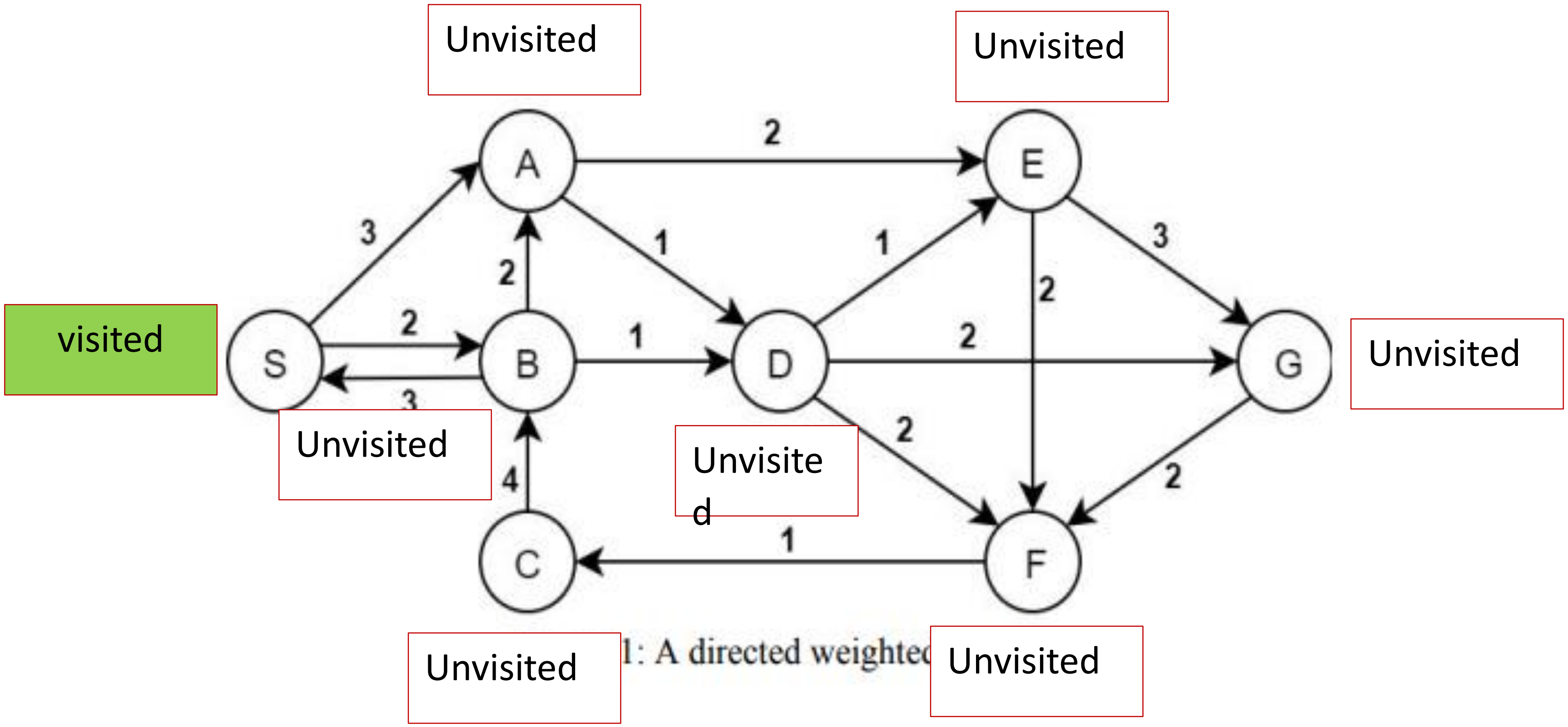
Q5: DFS



Unvisited
Unvisited
Unvisited
Unvisited
Unvisite d
Unvisited
Unvisited
Unvisited
S
A
B
C
D
E
F
G
3
2
3
2
4
1
1
2
1
1
2
2
3
2
2
1
1: A directed weighted

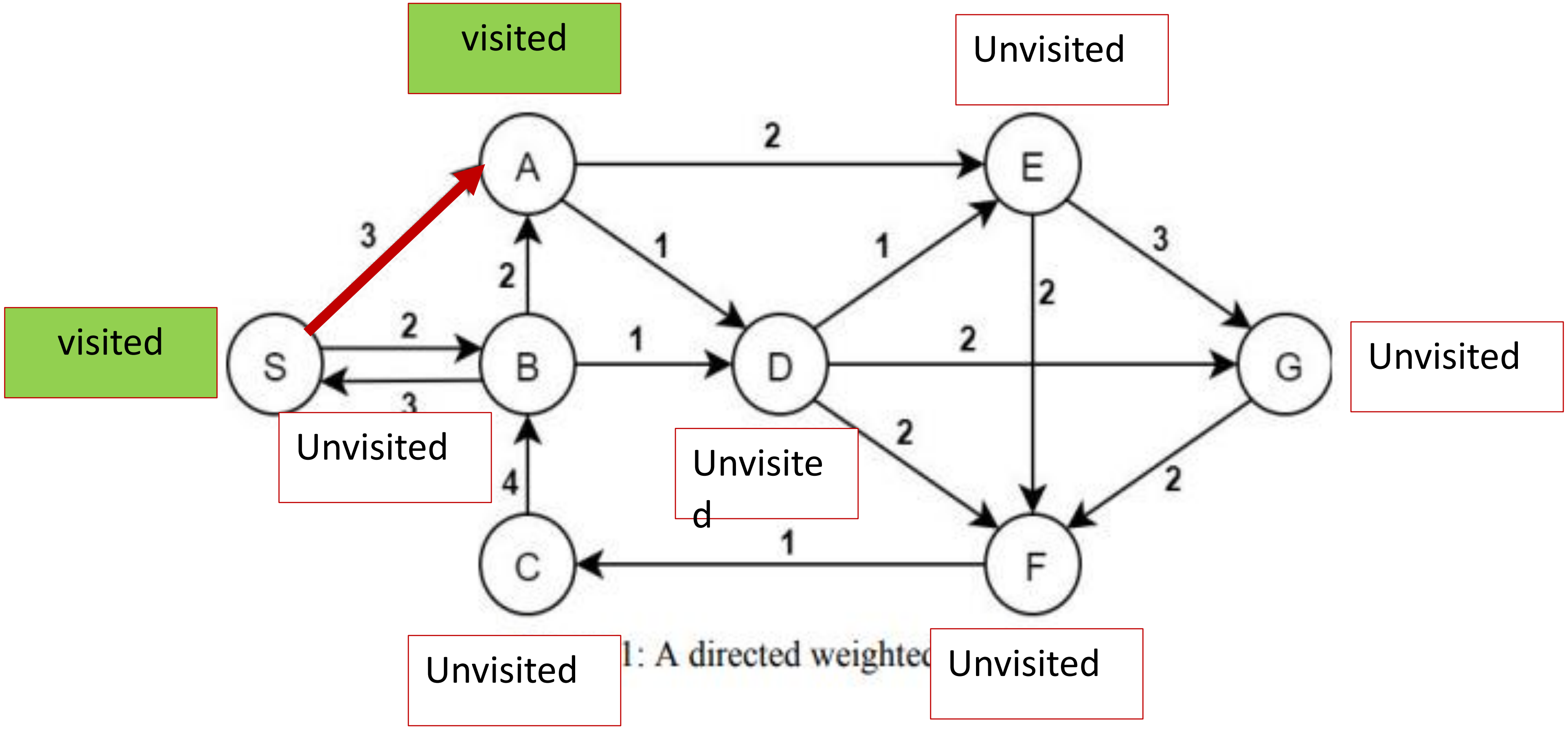
Q5: DFS

DFS Calls:
Dfs(s)



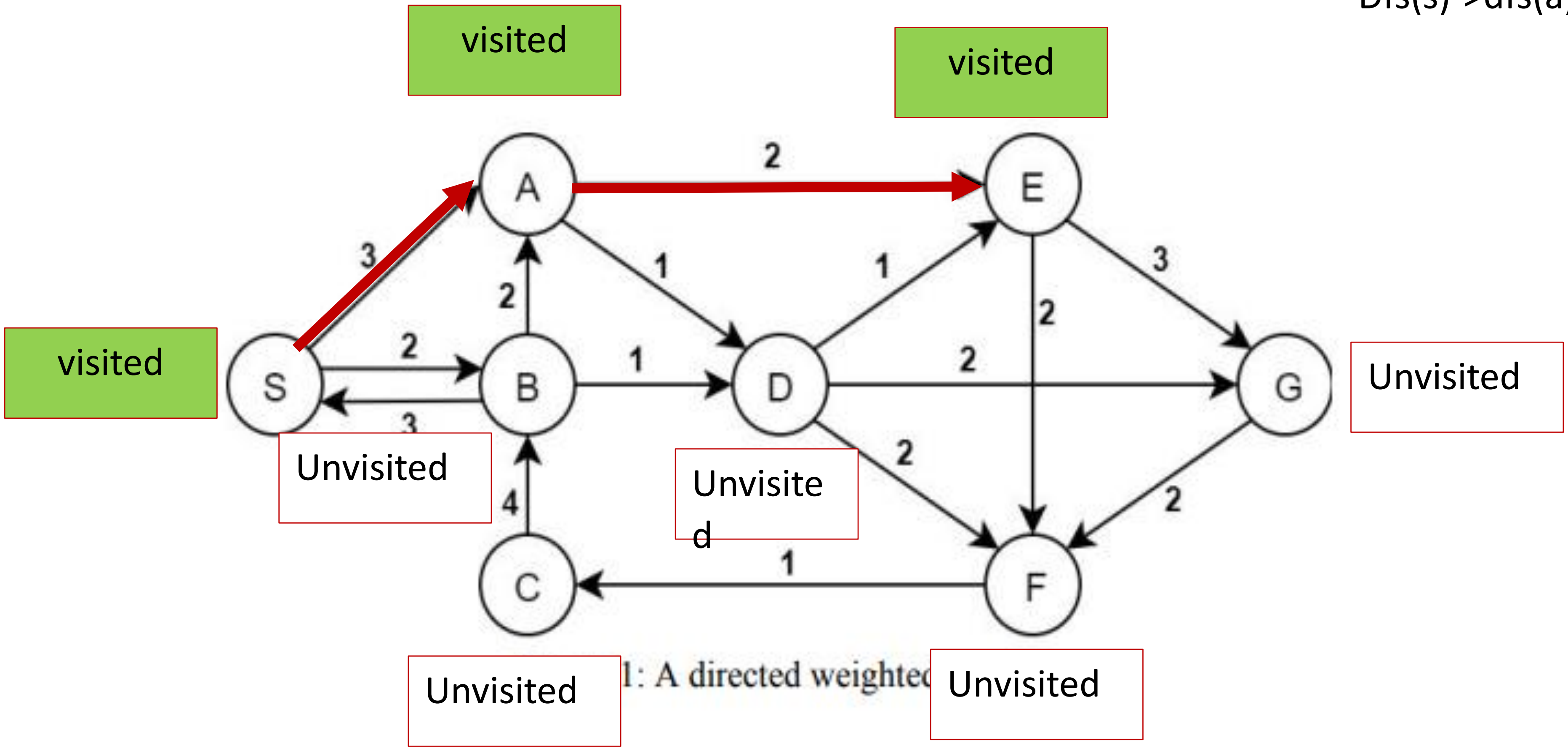
Q5: DFS

DFS Calls:
Dfs(s)->dfs(a)



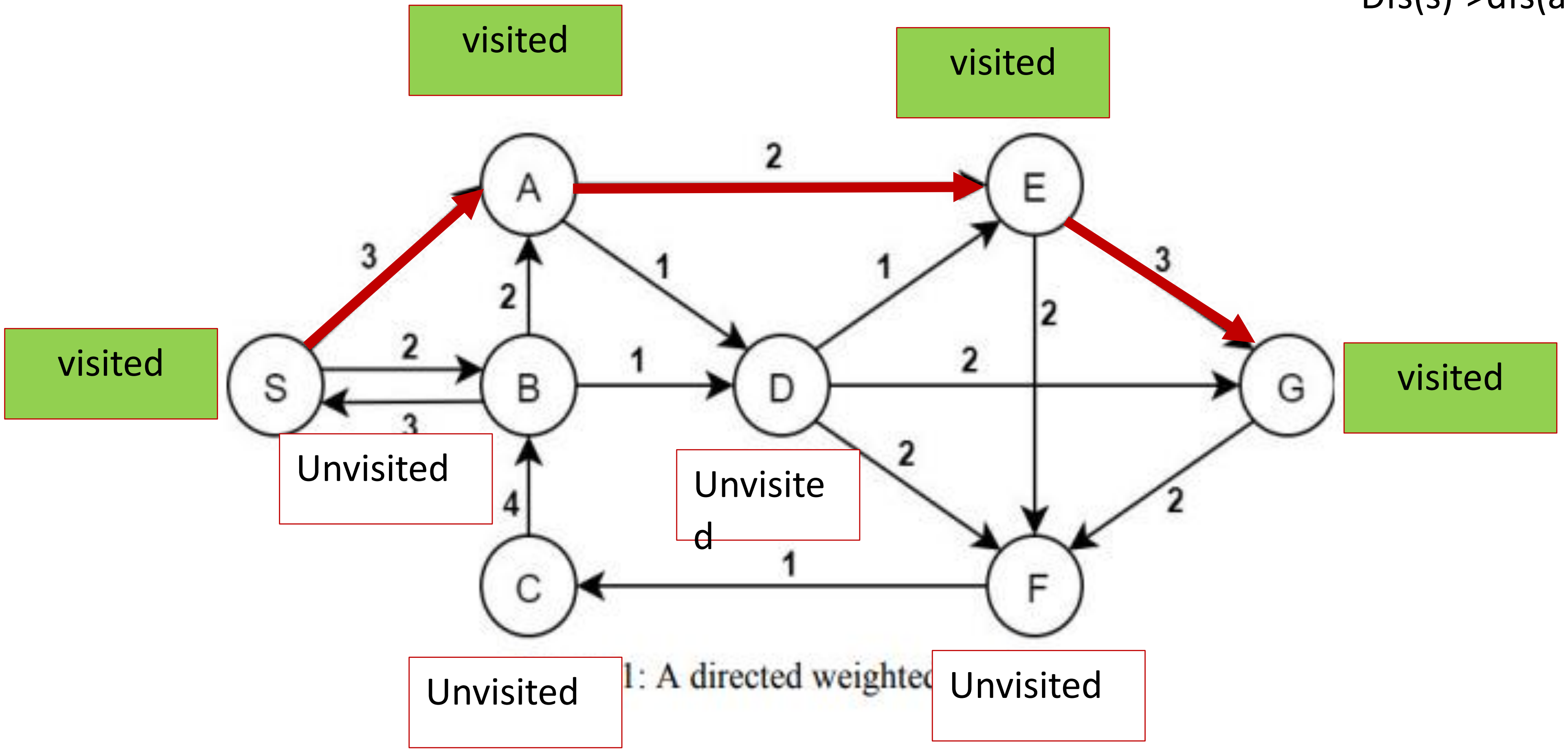
Q5: DFS

DFS Calls:
Dfs(s)->dfs(a)->dfs(e)

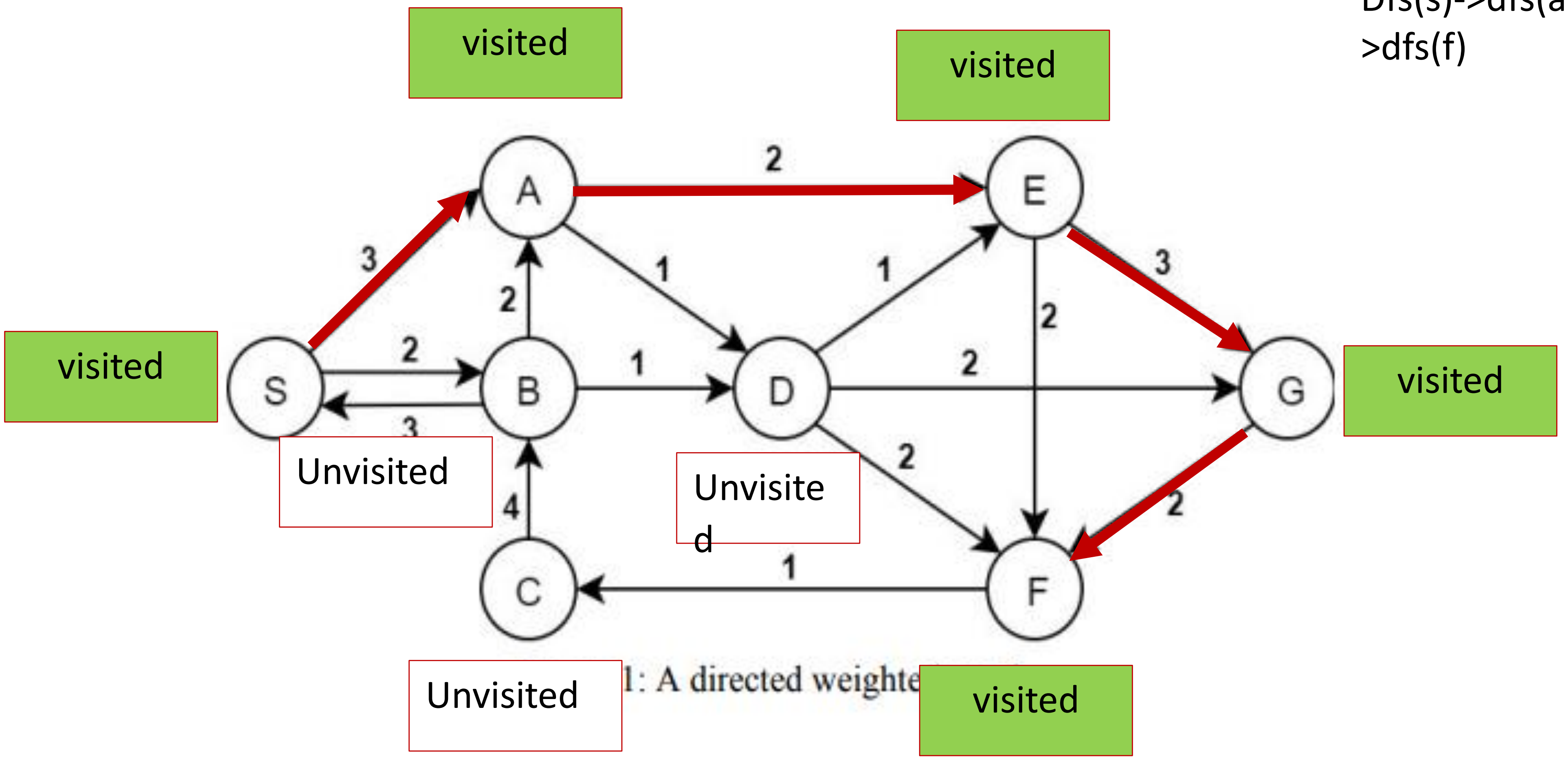


Q5: DFS

DFS Calls:
Dfs(s)->dfs(a)->dfs(e)->dfs(g)



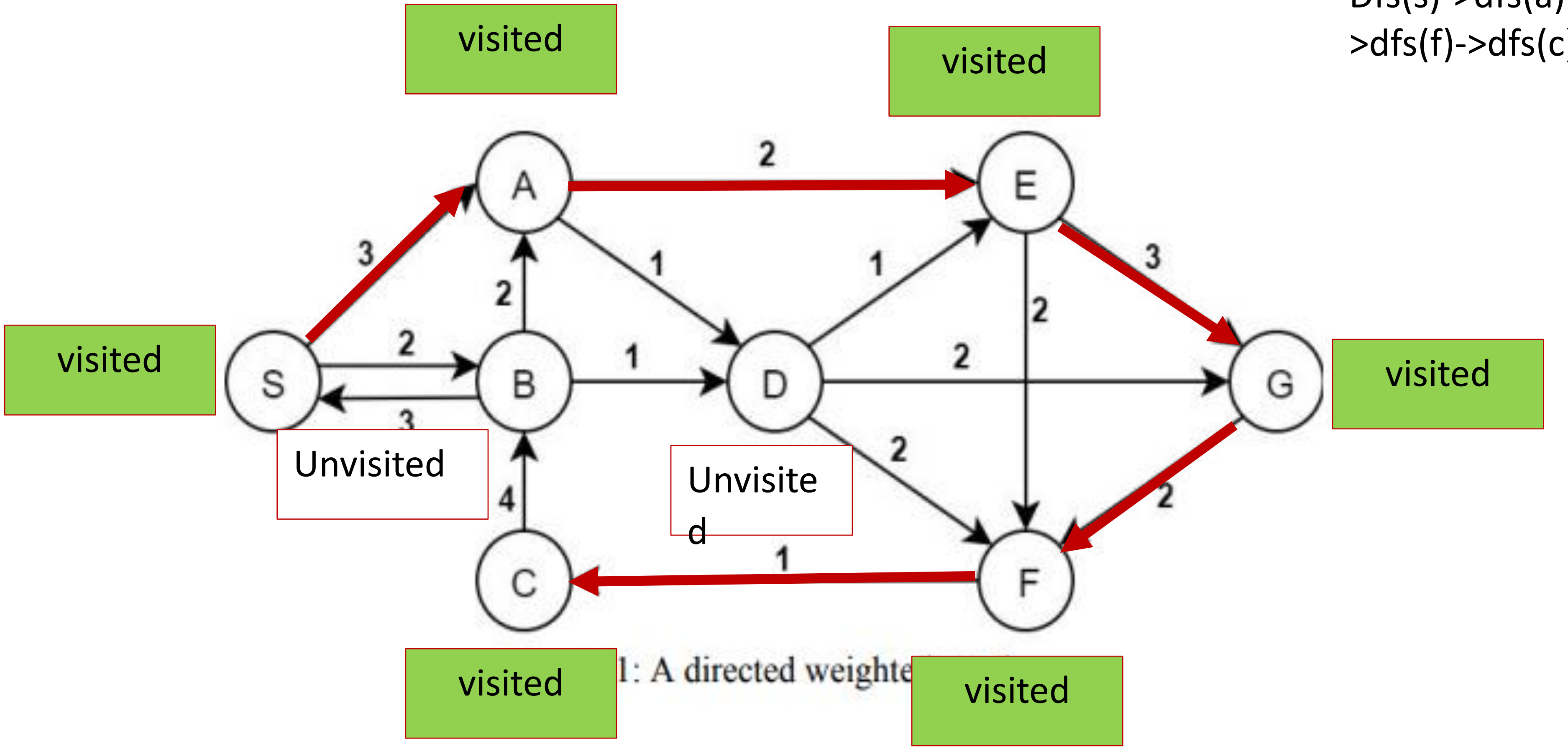
Q5: DFS



DFS Calls:
Dfs(s)->dfs(a)->dfs(e)->dfs(g)->dfs(f)

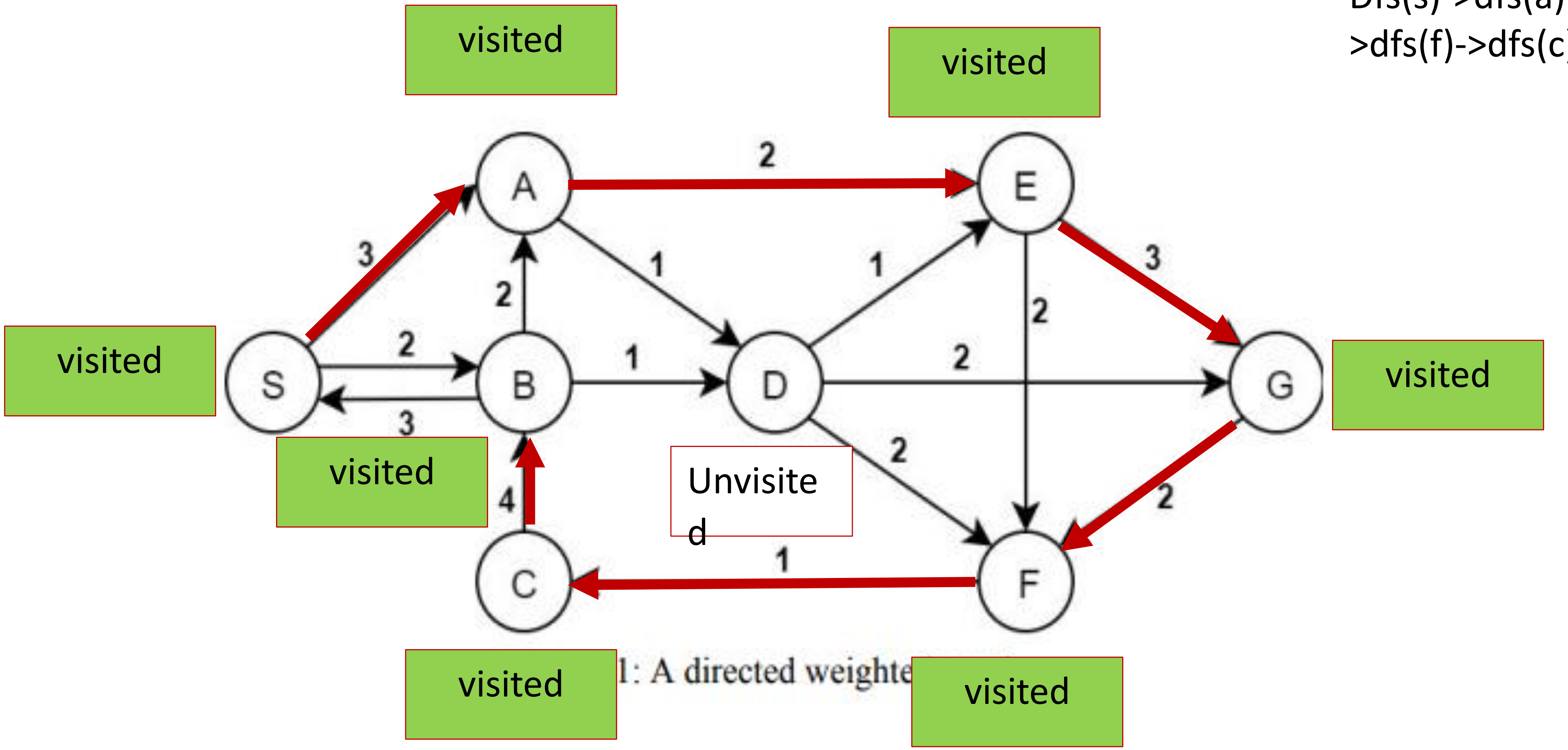
Q5: DFS

DFS Calls:
Dfs(s)->dfs(a)->dfs(e)->dfs(g)->dfs(f)->dfs(c)

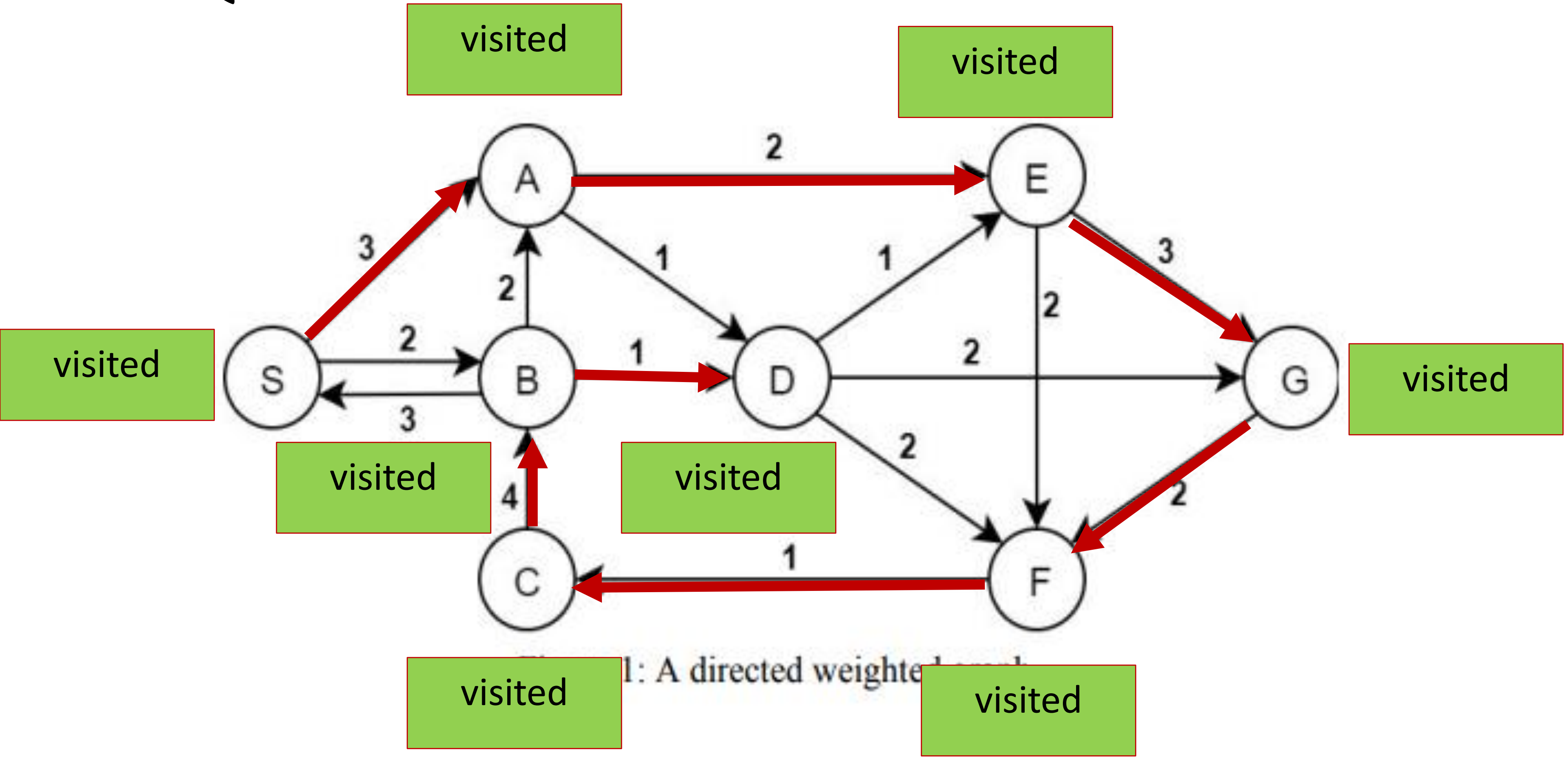


Q5: DFS

DFS Calls:
Dfs(s)->dfs(a)->dfs(e)->dfs(g)->dfs(f)->dfs(c)->dfs(b)



Q5: DFS



DFS Calls:
Dfs(s)->dfs(a)->dfs(e)->dfs(g)->dfs(f)->dfs(c)->dfs(b)->dfs(d)

Calls For other adj vertices of nodes

Dfs(d) cant call dfs(e), dfs(f), dfs(g) because they are already visited.

Dfs(b) cant call dfs(s),dfs(a) because they are already visited.

Dfs(e) cant call dfs(f) because f is already visited.

Dfs(a) cant call dfs(d) because d is already visited.

Dfs(s) cant call dfs(b) because b is already visited.

Q5: DFS

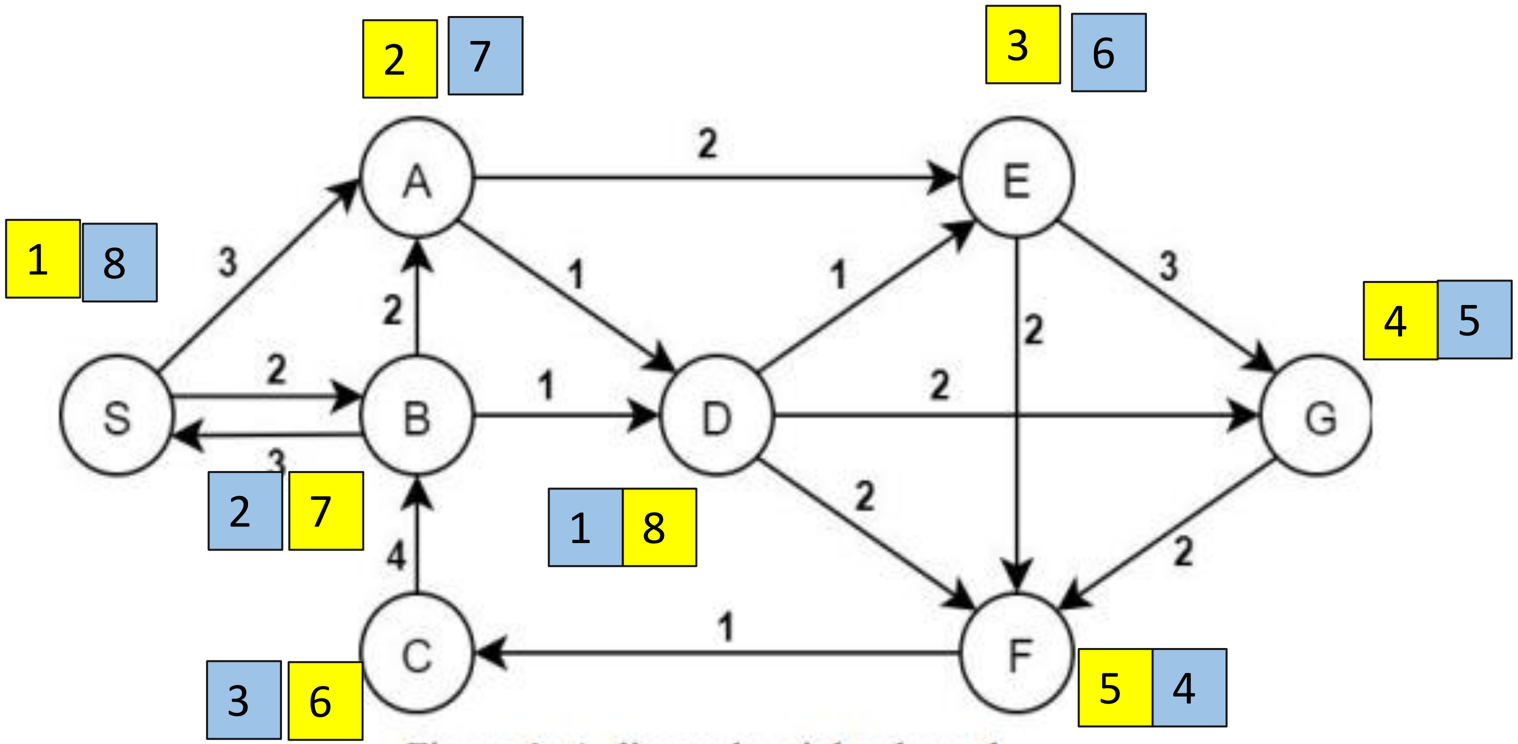


Figure 1: A directed weighted graph.

DFS Calls Order:
Dfs(s)->dfs(a)->dfs(e)->dfs(g)->dfs(f)->dfs(c)->dfs(b)->dfs(d)

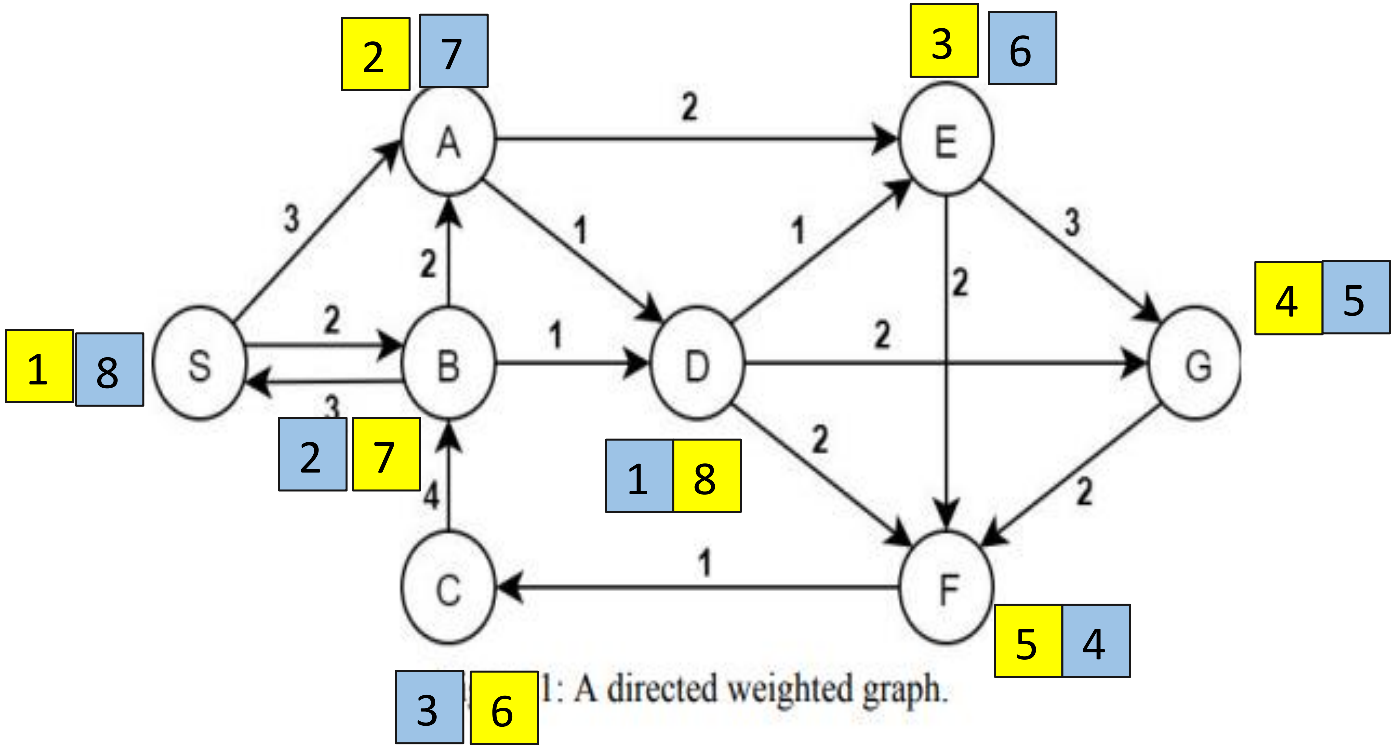
Preorder numbers increments in the same order with DFS calls order.

x : Pre order number

Postorder numbers increments in the reverse order with DFS calls order.

x : Post order number

Q5: DFS

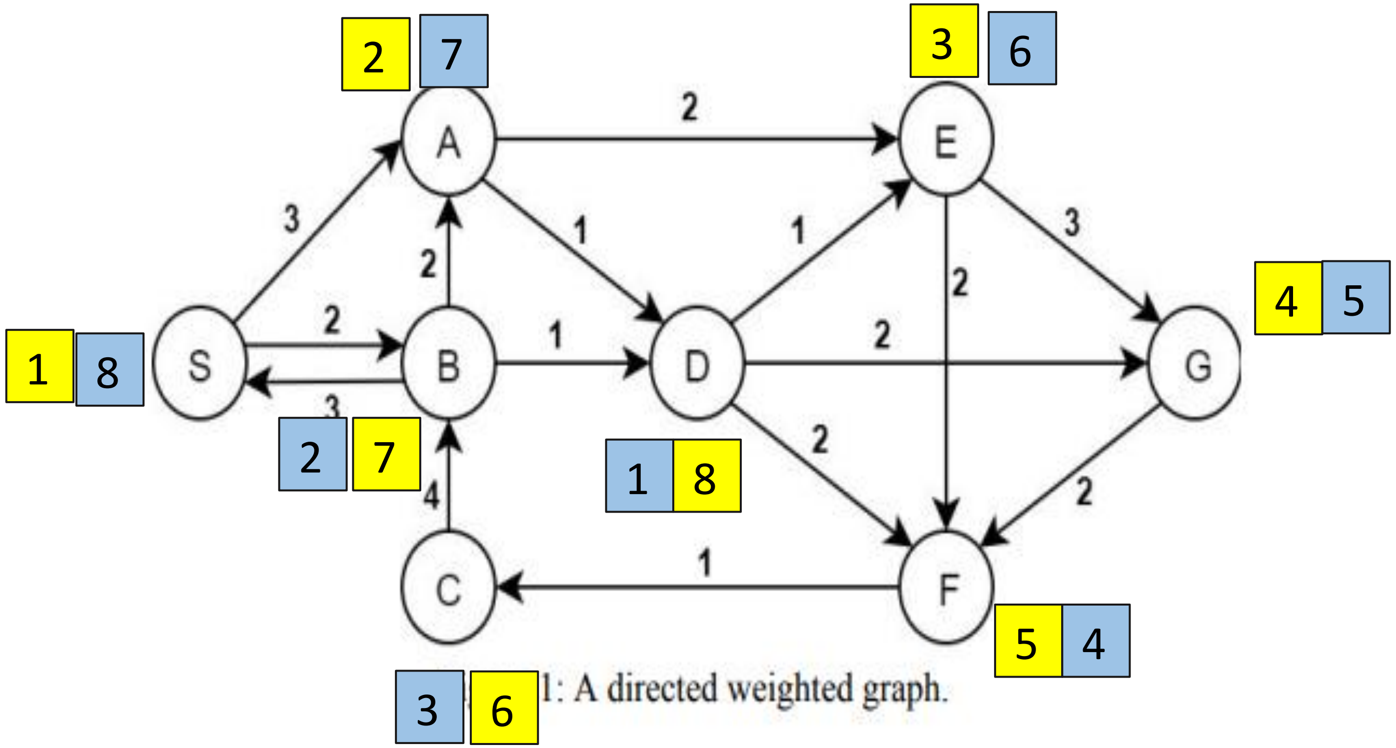


1: A directed weighted graph.

CLASSIFICATION OF ARCS:

1. **Arc S->A: TREE ARC** (An edge in DFS tree)
2. **Arc S->B: FORWARD ARC**
   PRE(S)<PRE(B) &&PON(S)>PON(B)
3. **Arc B->S: BACKWARD ARC**
   PON(S)>PON(B)
4. **Arc B->A: BACKWARD ARC**
   PON(A)>PON(B)
5. **Arc B->D: TREE ARC** (An edge in DFS tree)
6. **Arc A->D: FORWARD ARC**
   PON(D)<PON(A)&&PRE(D)>PRE(A)
7. **Arc A->E: TREE ARC** (An edge in DFS tree)
8. **Arc C->B: TREE ARC** (An edge in DFS tree)

Q5: DFS



1: A directed weighted graph.

CLASSIFICATION OF ARCS:

9 **Arc D->E: BACKWARD ARC**
PON(E)>PON(D)

10. Arc E->G: TREE ARC (An edge in DFS Tree)

11. Arc E->F: FORWARD ARC
PRE(F)>PRE(E)&&PON(F)<PON(E)

12. Arc D->F: BACKWARD ARC
PON(F)>PON(D)

13. Arc D->G: BACKWARD ARC
PON(G)>PON(D)

14. Arc G->F: TREE ARC (An edge in DFS Tree)

15. Arc F->C: TREE ARC (An edge in DFS Tree)

Q6- Topological Sort

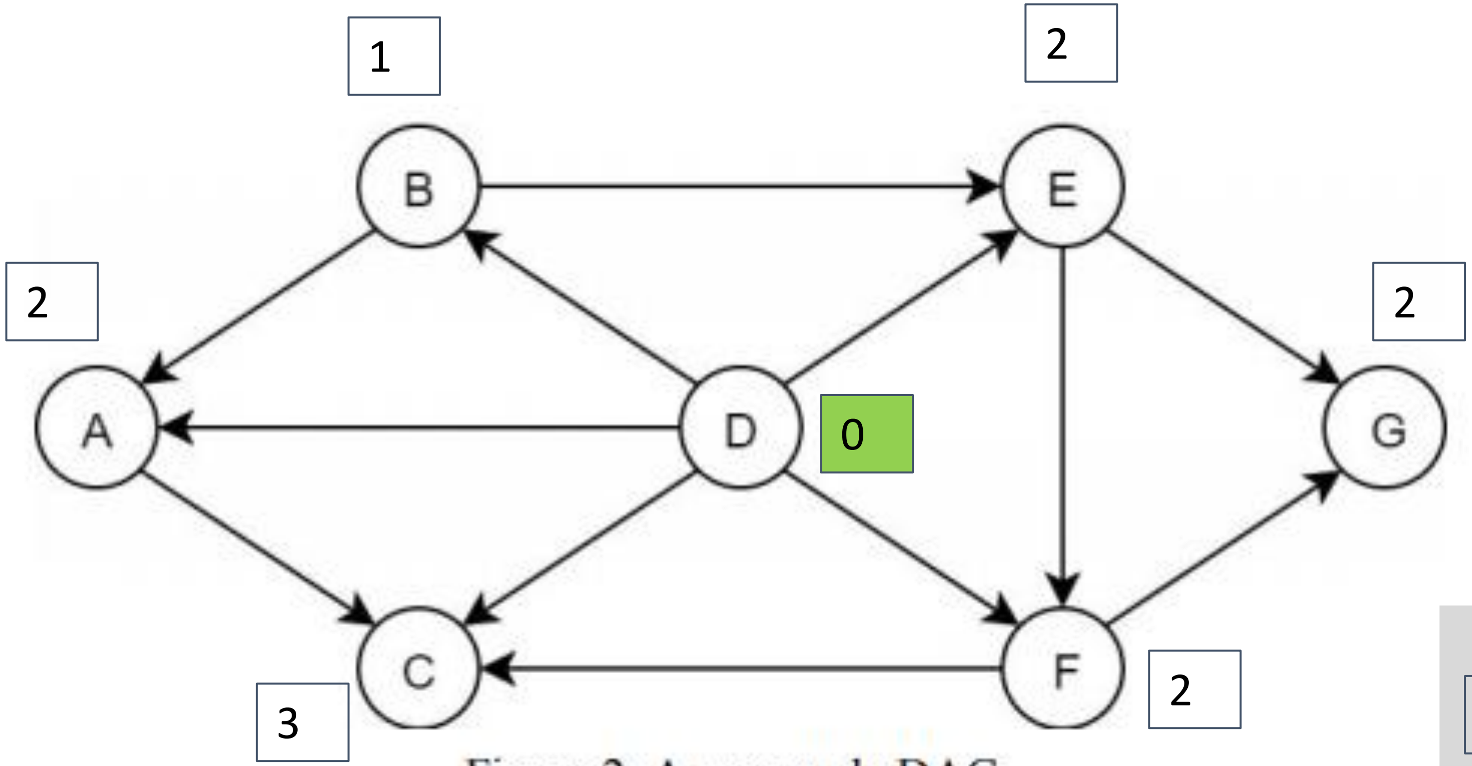


Figure 2: An example DAG.

Select D
Print D
Remove D
Update indegrees

X : Vertex with indegree X.

0 : Vertex with indegree 0.

Q6- Topological Sort

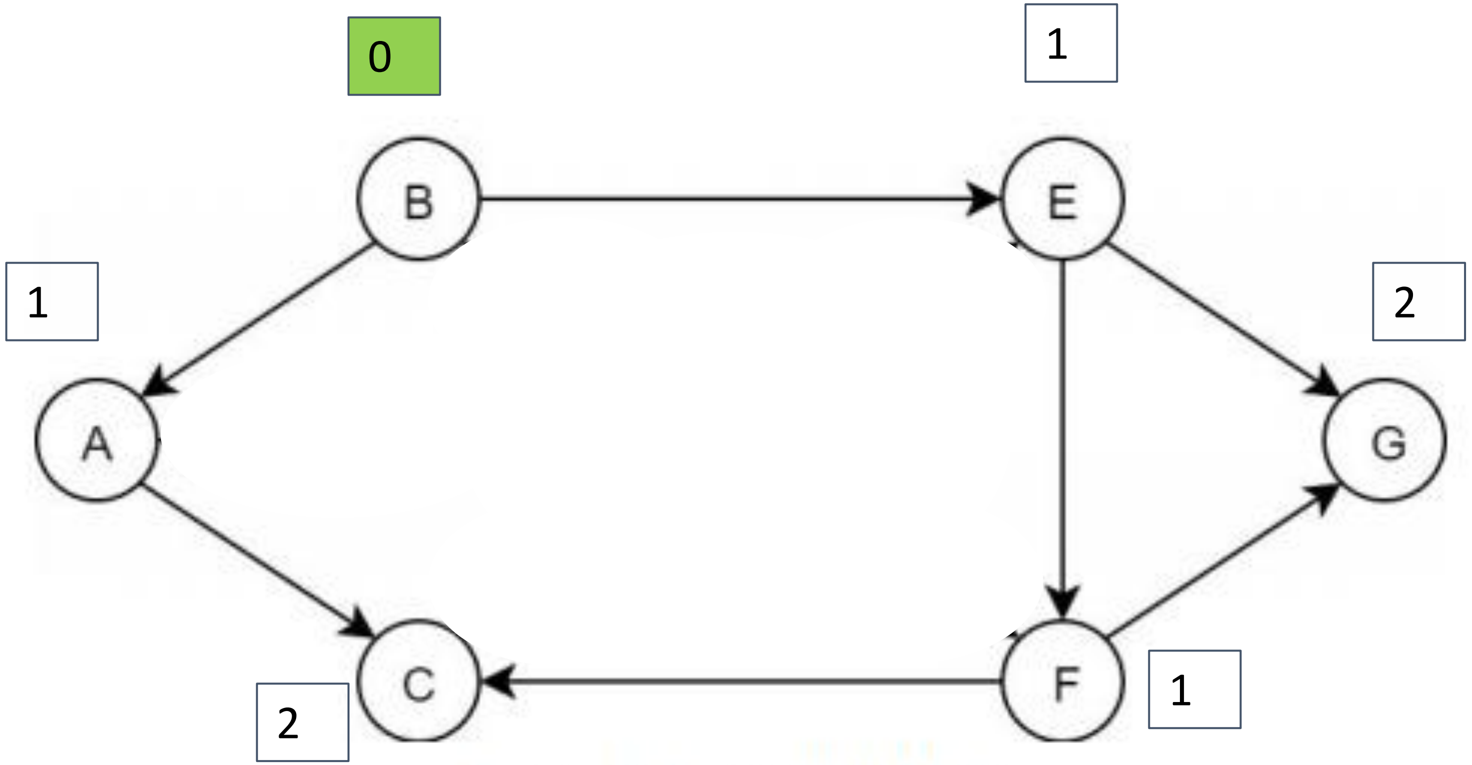


Figure 2: An example DAG.

Select B
Print B
Remove B
Update indegrees

SORTED NODES: D

Q6- Topological Sort

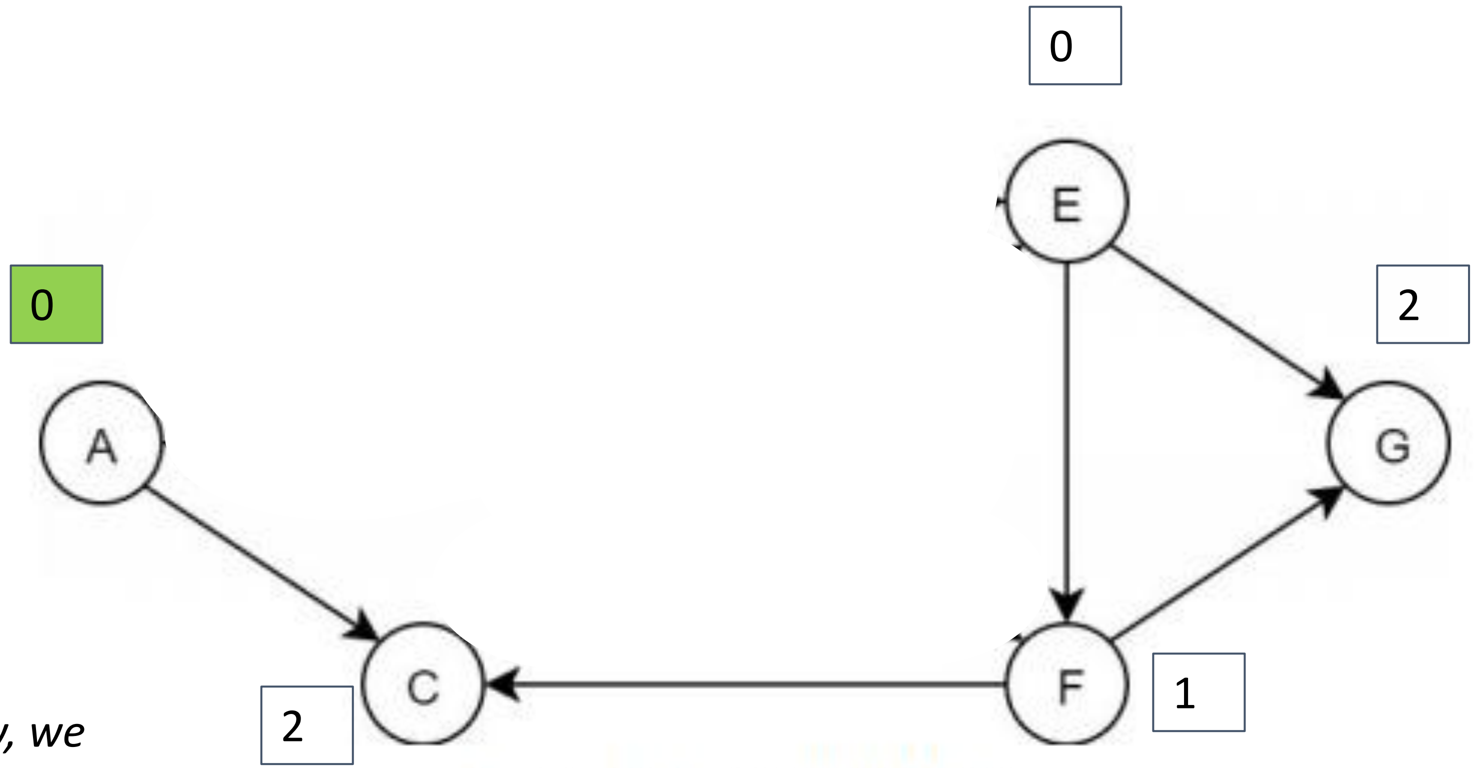


Figure 2: An example DAG.

Select A (*choice is arbitrary, we could select E.)*
Print A
Remove A
Update indegrees

SORTED NODES: D,B

Q6- Topological Sort

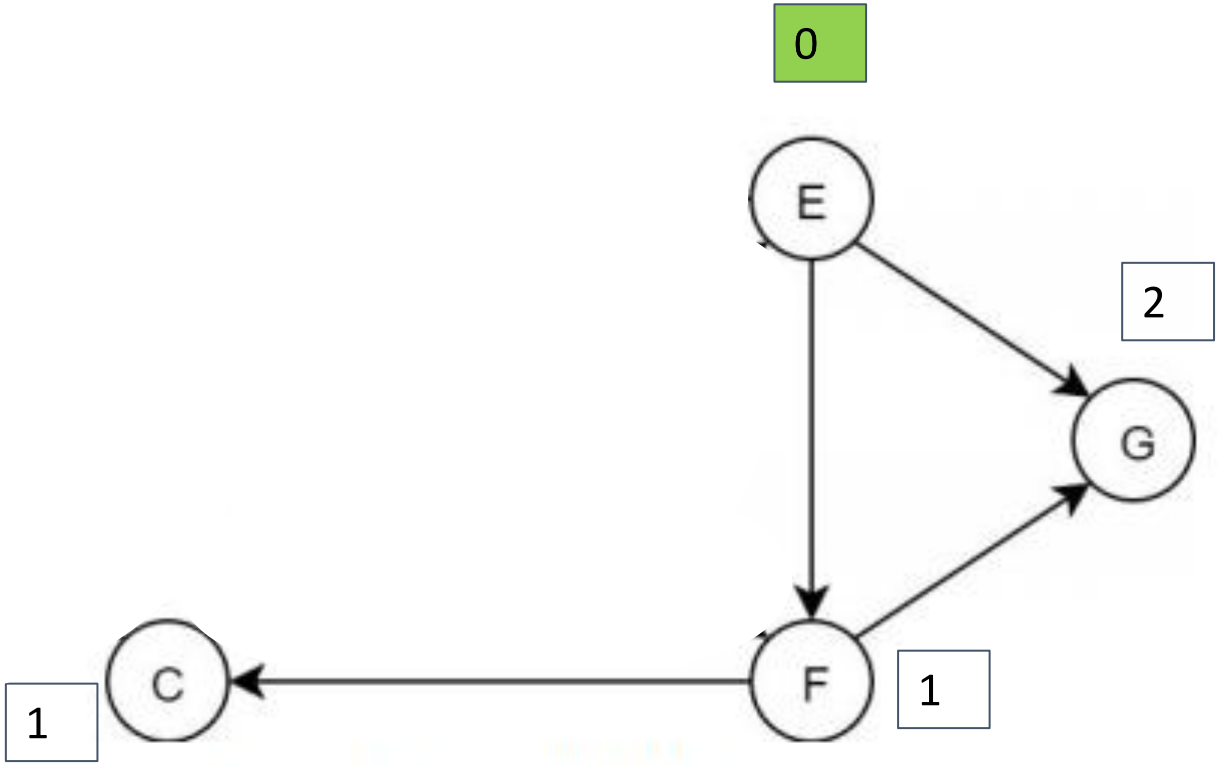


Figure 2: An example DAG.

Select E
Print E
Remove E
Update indegrees

SORTED NODES: D,B,A

Q6- Topological Sort

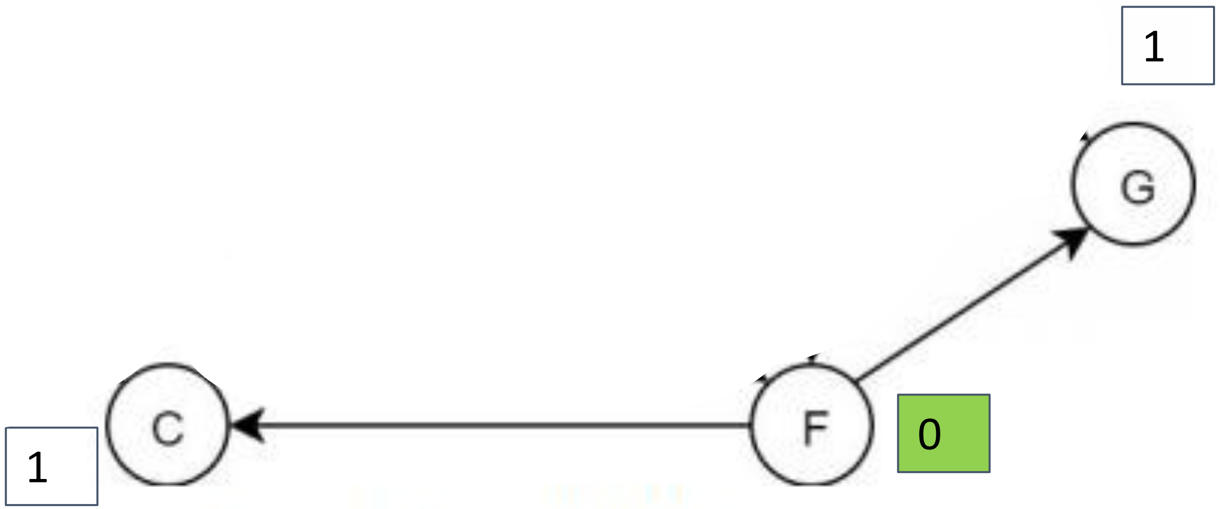


Figure 2: An example DAG.

Select F
Print F
Remove F
Update indegrees

SORTED NODES: D,B,A,E

Q6- Topological Sort

Select C
Print C
Remove C

Select G
Print G
Remove G

Order of choice is arbitrary.

SORTED NODES: D,B,A,E,F

Q6- Topological Sort

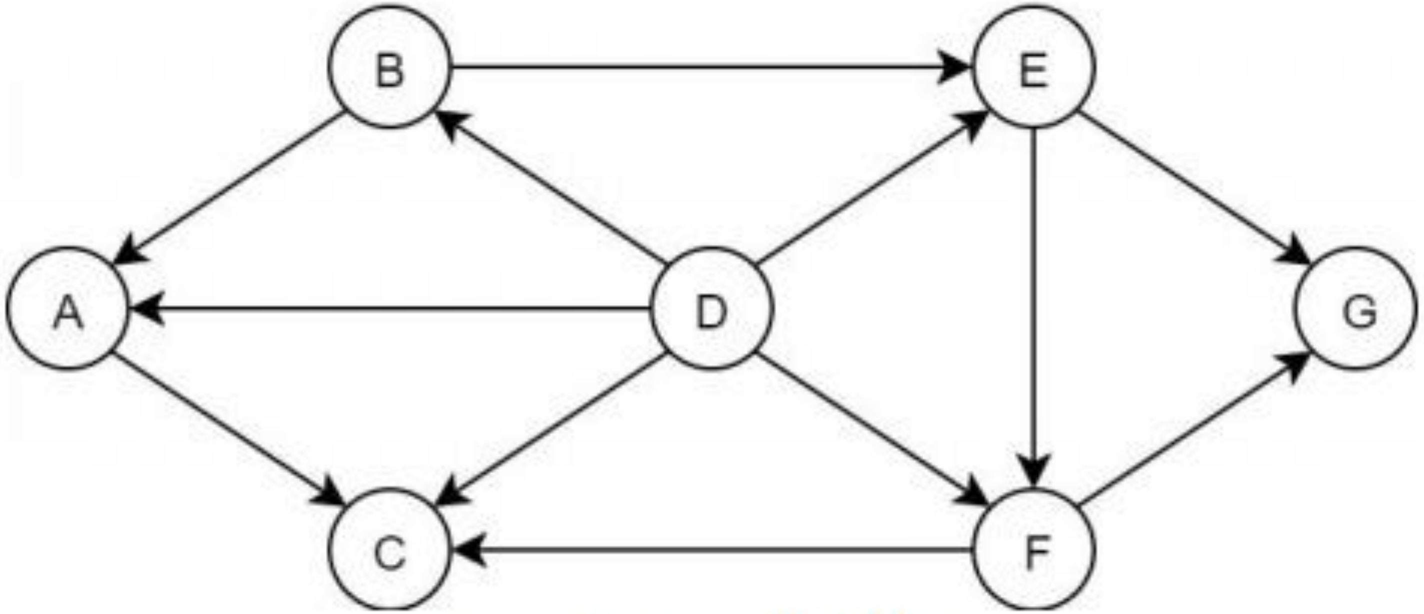


Figure 2: An example DAG.

TOPOLOGICAL ORDER: D,B,A,E,F,C,G