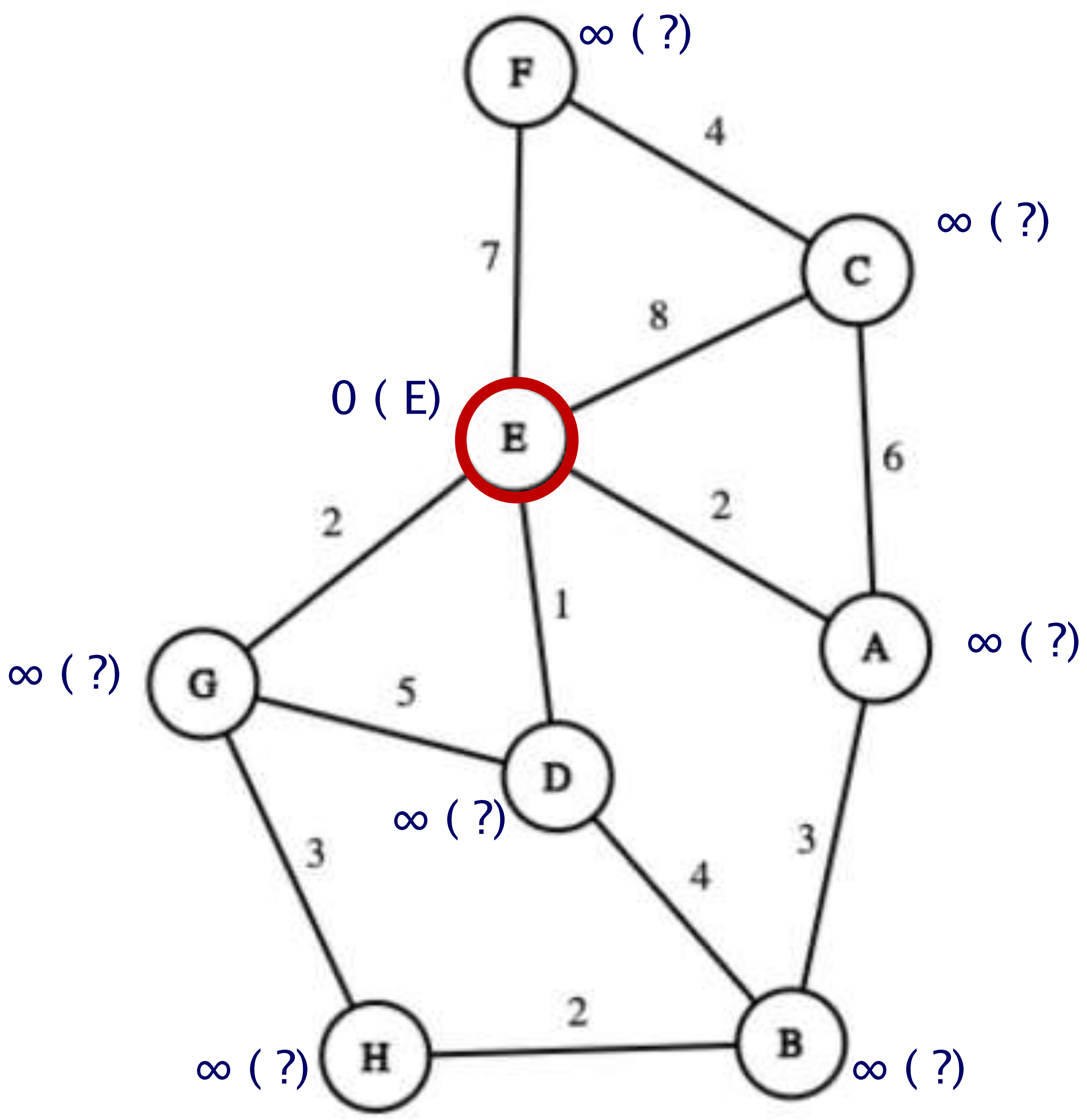
QUESTION-1

Initial condition is given below. We will start from vertex E.

For vertex E;

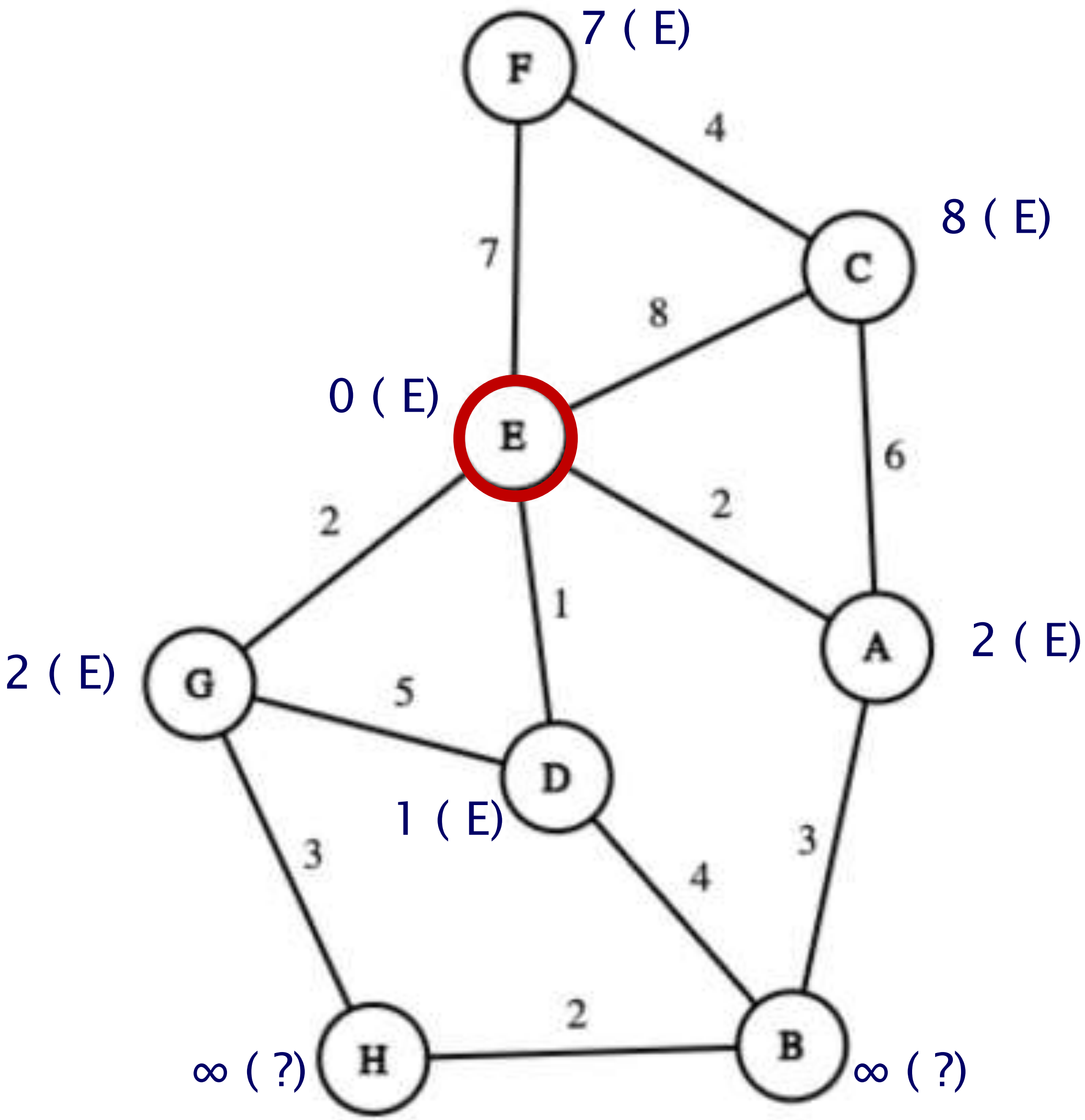
- Shortest path is 0
- Last visited vertex is E (itself, since it is the starting vertex)
- E is marked as known.



QUESTION-1

Check the neighbouring vertexes of E. Update their shortest path accordingly.

Next smallest distance value belongs to vertex D among all unknown vertices. So, we move to D.



QUESTION-1

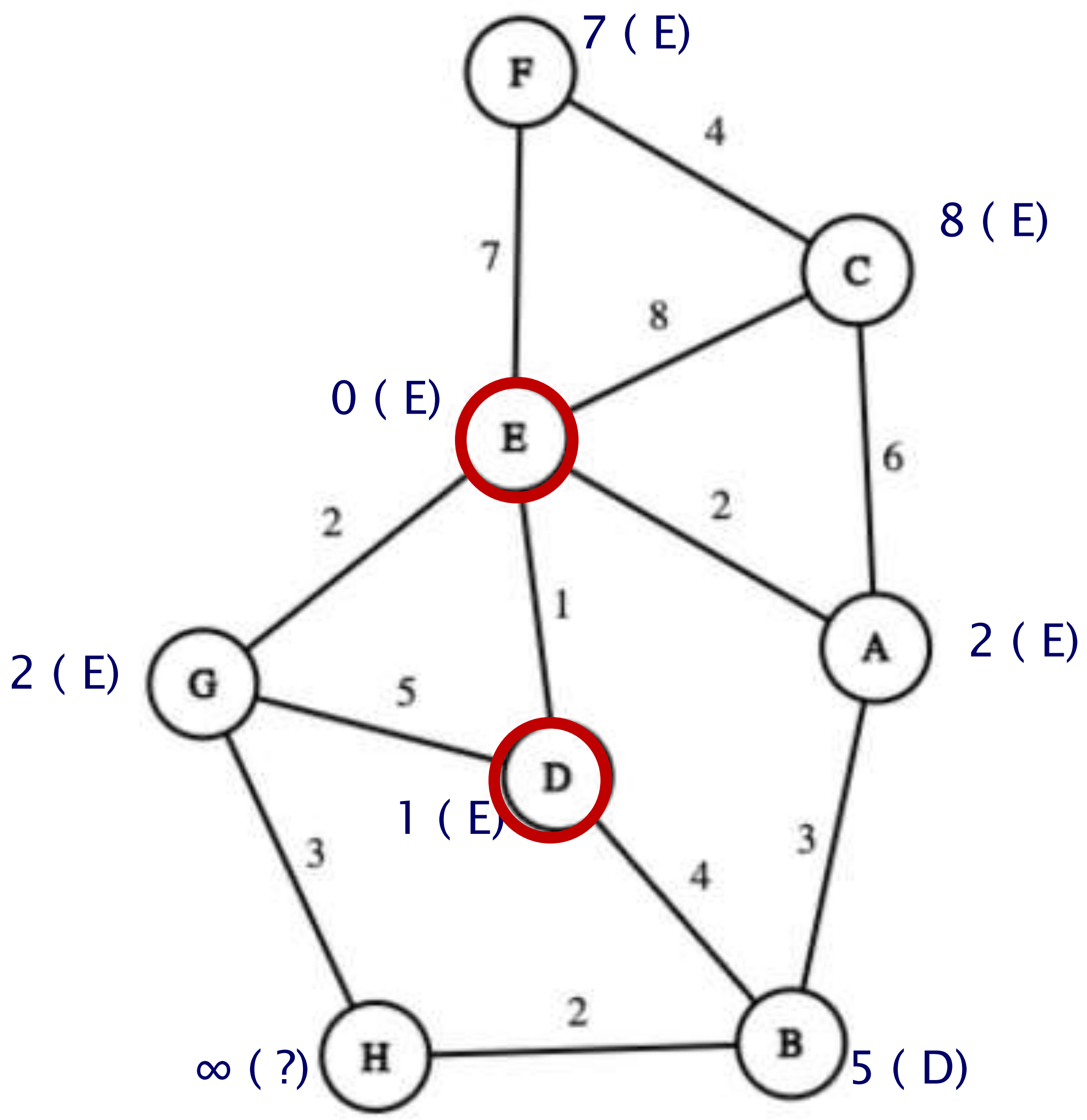
We are at D. D is marked as known. Check its neighbours, and update the shortest paths accordingly.

Shortest path to G is not changed since 2 < (1+5).

Shortest path to E is not changed since it is the initial vertex (dist=0).

B is updated since 5 < infinity. Last visited vertex through B has become D.

Next smallest distance value belongs to vertices G and A among all unknown vertices. Choose one. So, we move to A.



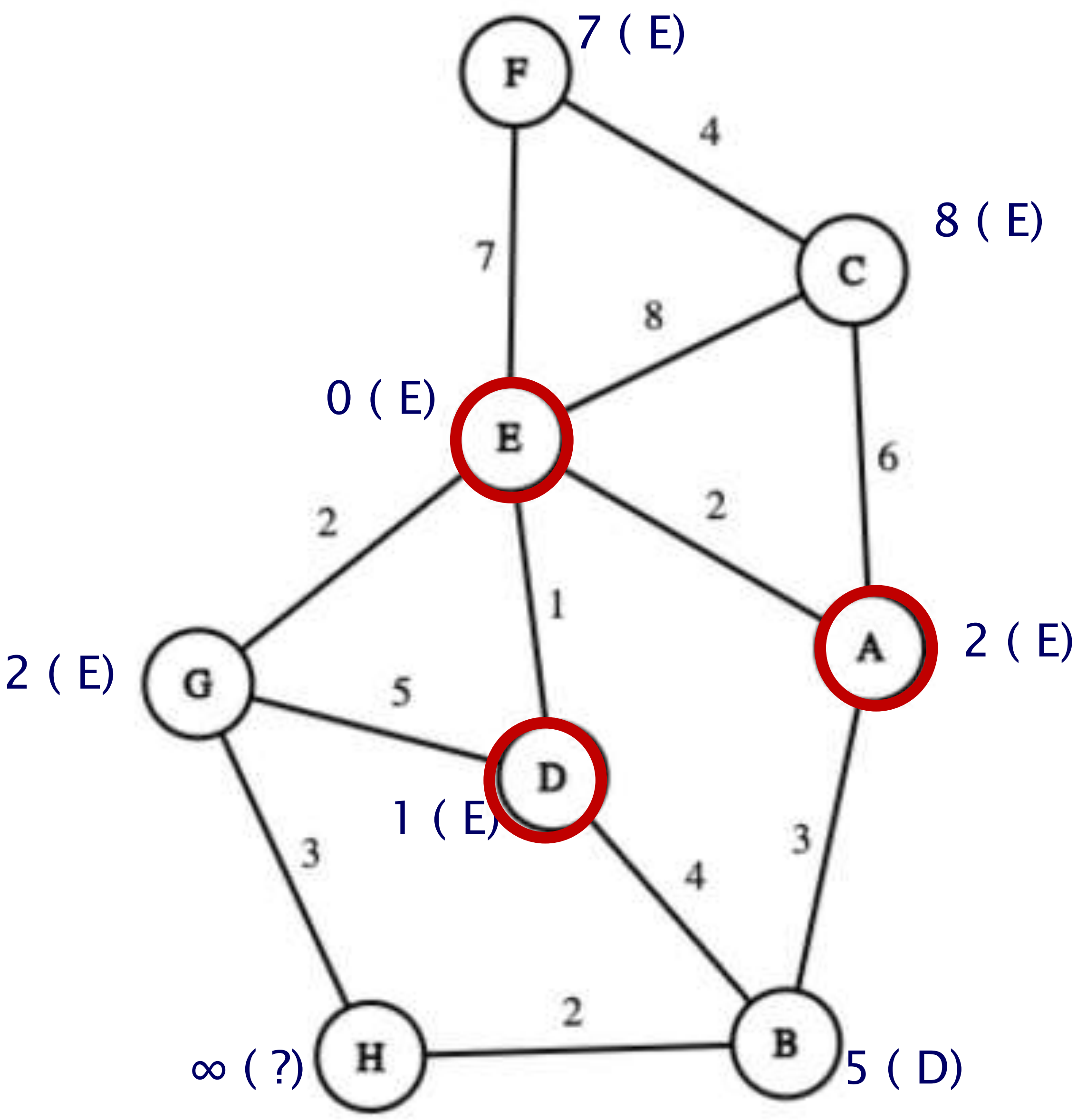
QUESTION-1

We are at A. A is marked as known. Check its neighbours, and update the shortest paths accordingly.

(2+6) = 8 for C, so C is not updated.

(2+3) = 5 for B, so B is not updated.

Next smallest distance value belongs to vertex G among all unknown vertices. So, we move to G.



QUESTION-1

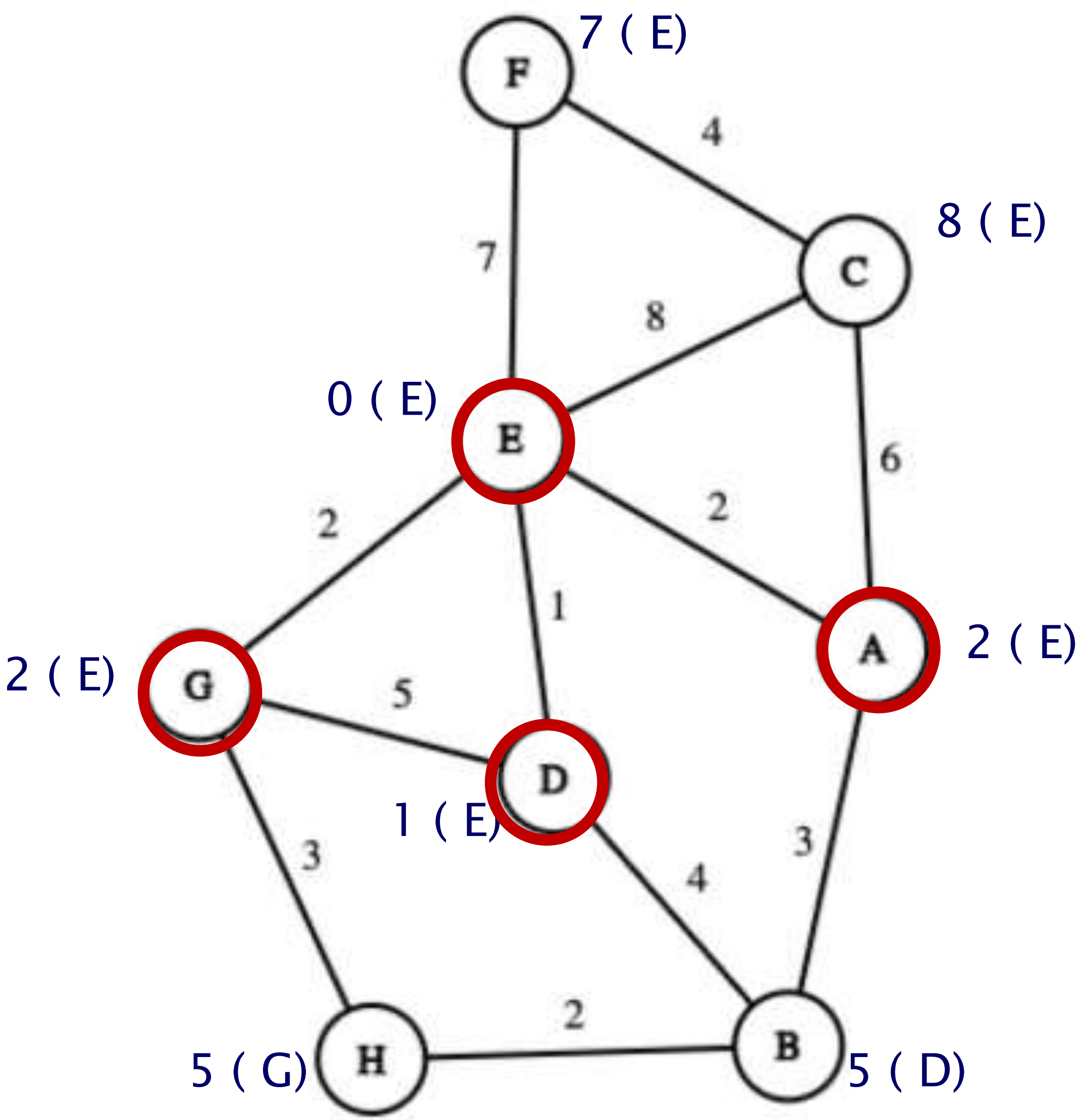
We are at G. G is marked as known. Check its neighbours, and update the shortest paths accordingly.

(2+3) < infinity, so we update H. Last visited vertex through H has become G.

Vertex D is known, skip.

Vertex E is known, skip.

Next smallest distance value belongs to vertices H and B among all unknown vertices. Choose one. So, we move to H.



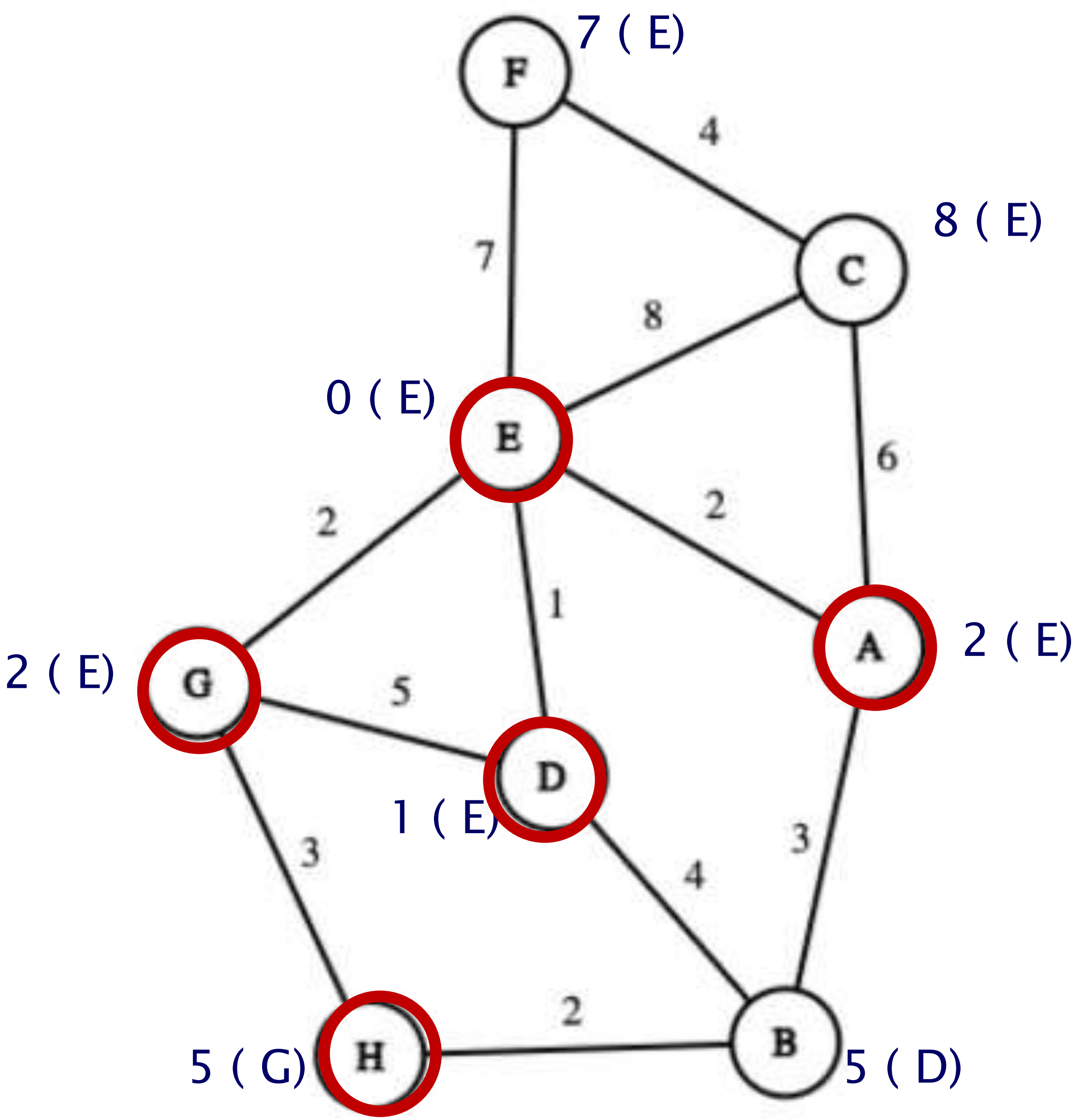
QUESTION-1

We are at H. H is marked as known. Check its neighbours, and update the shortest paths accordingly.

5 < (5+2), so we do not update B.

Vertex G is known, skip.

Next smallest distance value belongs to vertex B among all unknown vertices. So, we move to B.

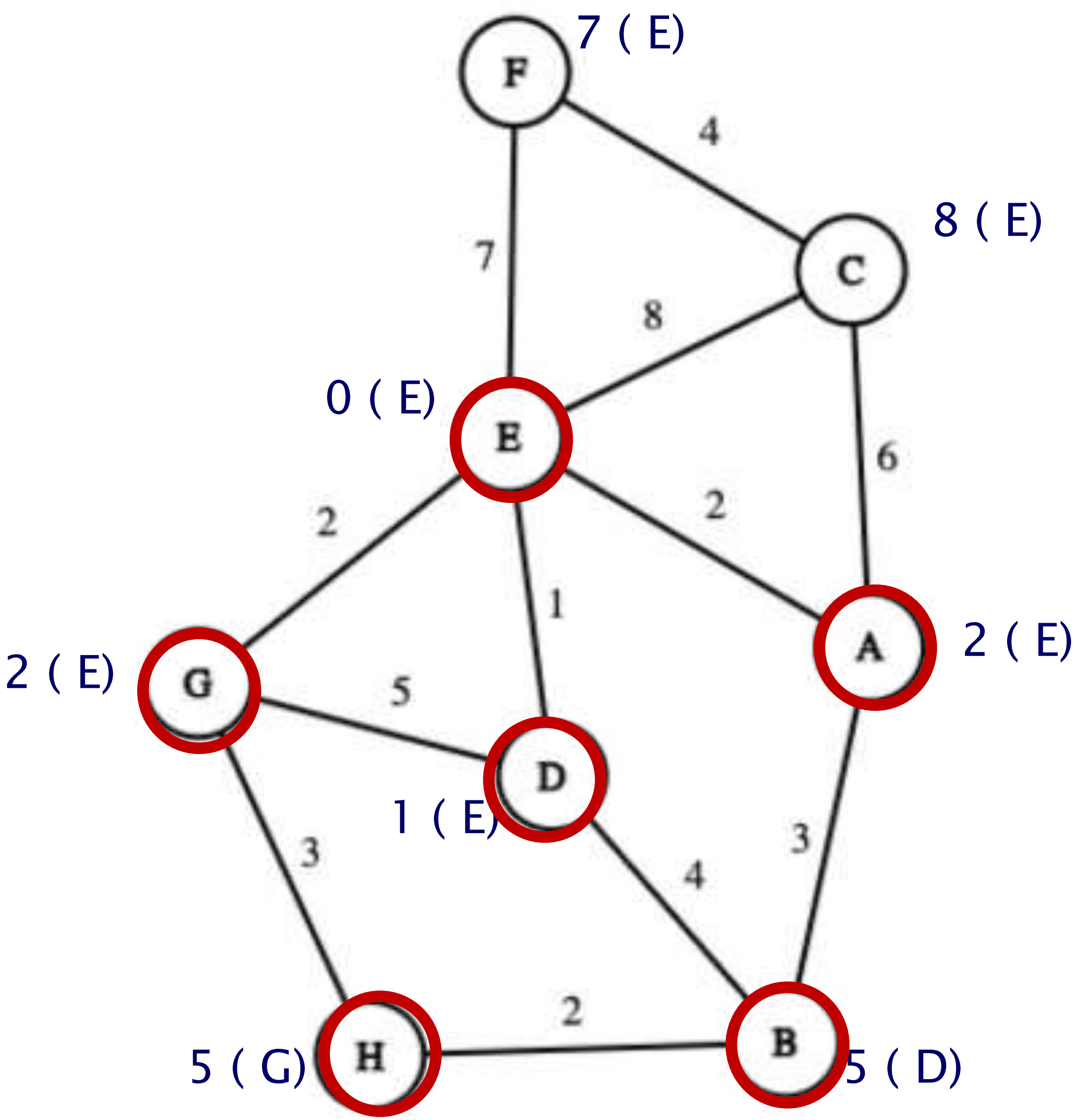


QUESTION-1

We are at B. B is marked as known. Check its neighbours, and update the shortest paths accordingly.

All neighbours are known, skip.

Next smallest distance value belongs to vertex F among all unknown vertices. So, we move to F.



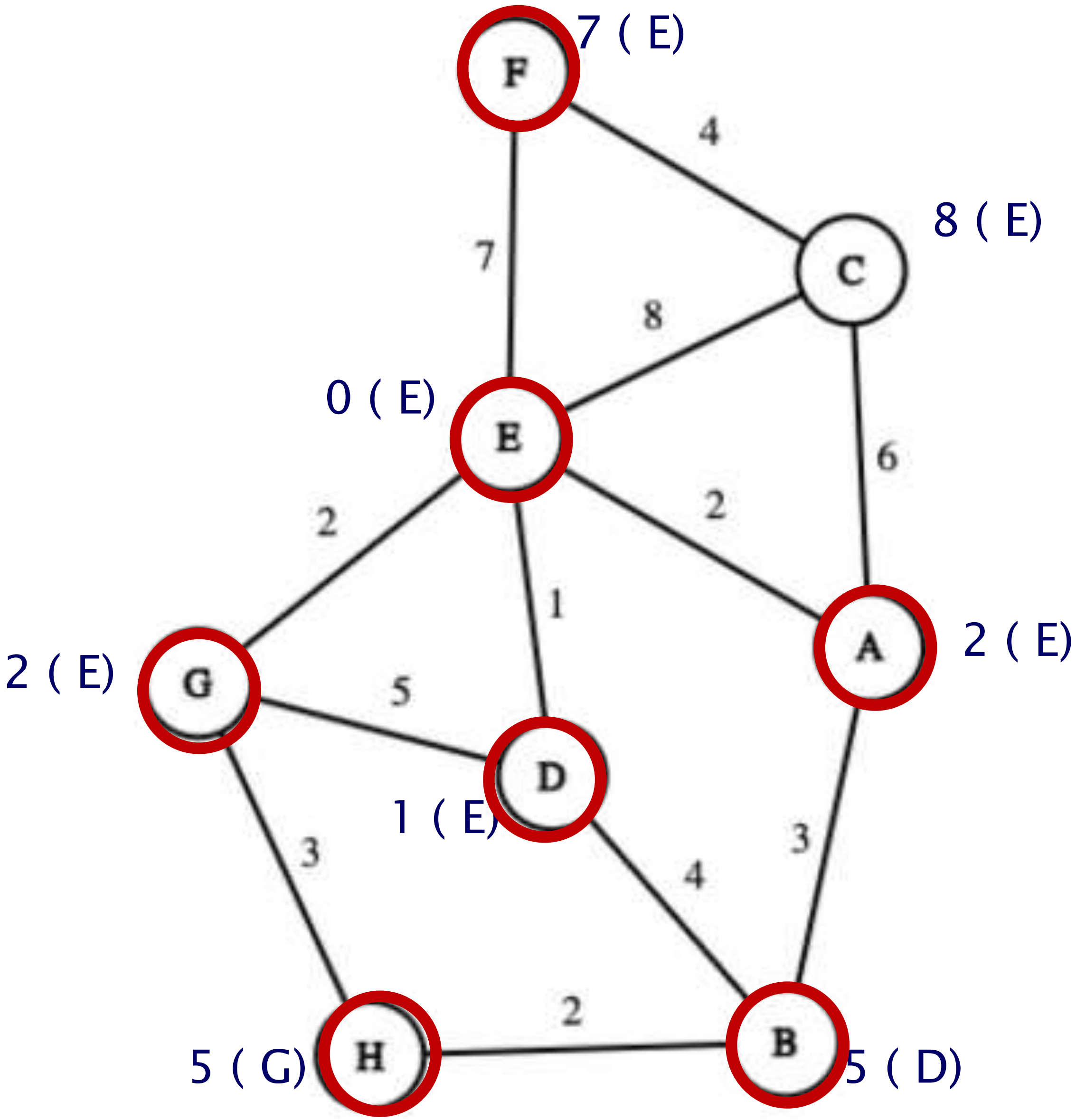
QUESTION-1

We are at F. F is marked as known. Check its neighbours, and update the shortest paths accordingly.

E is known, skip.

8 < (7+4), so we do not update C.

Next smallest distance value belongs to vertex C among all unknown vertices. So, we move to C.

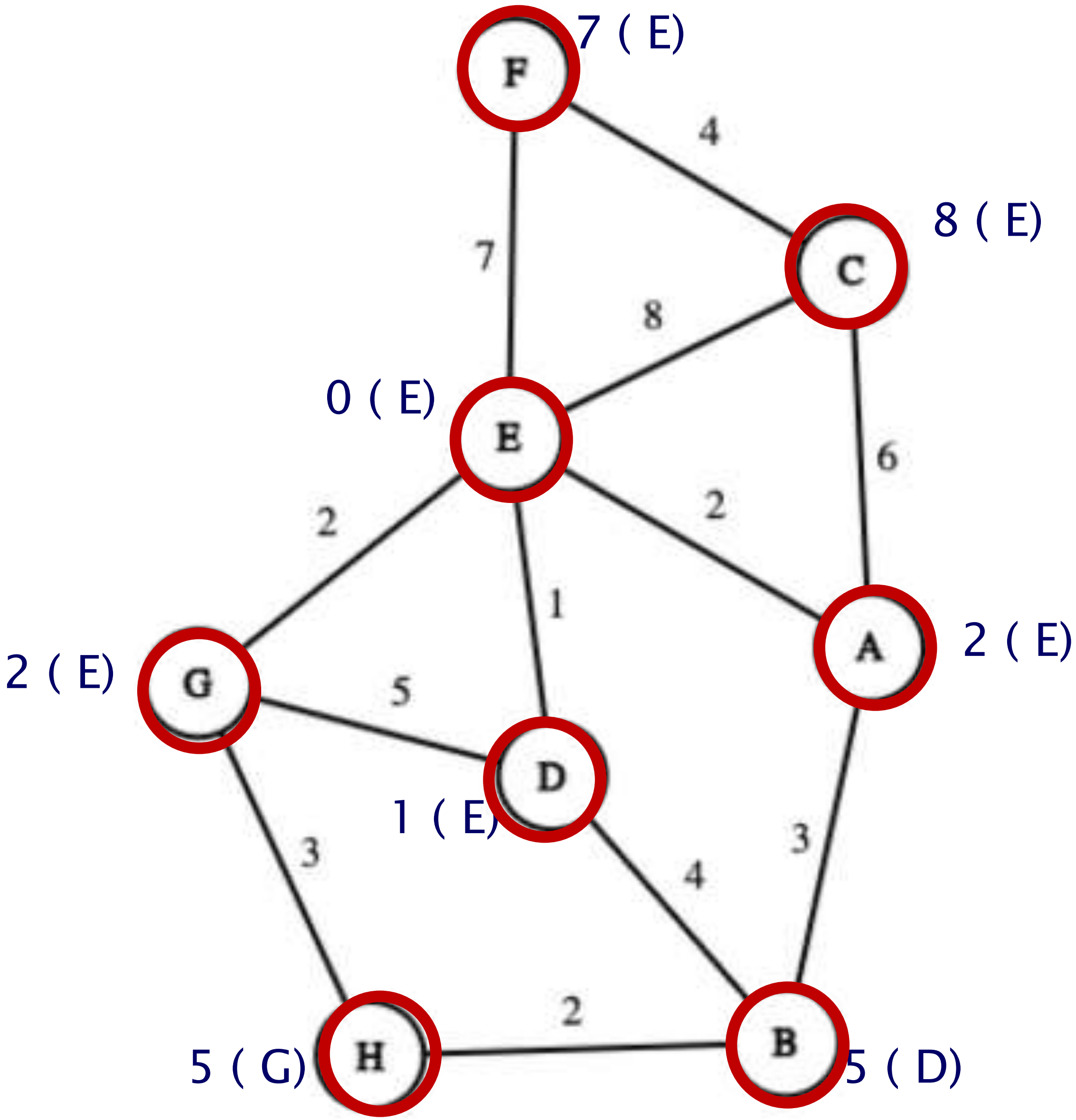


QUESTION-1

We are at C. C is marked as known. Check its neighbours, and update the shortest paths accordingly.

All neighbours are known, skip.

No unknown vertex left. Terminate.

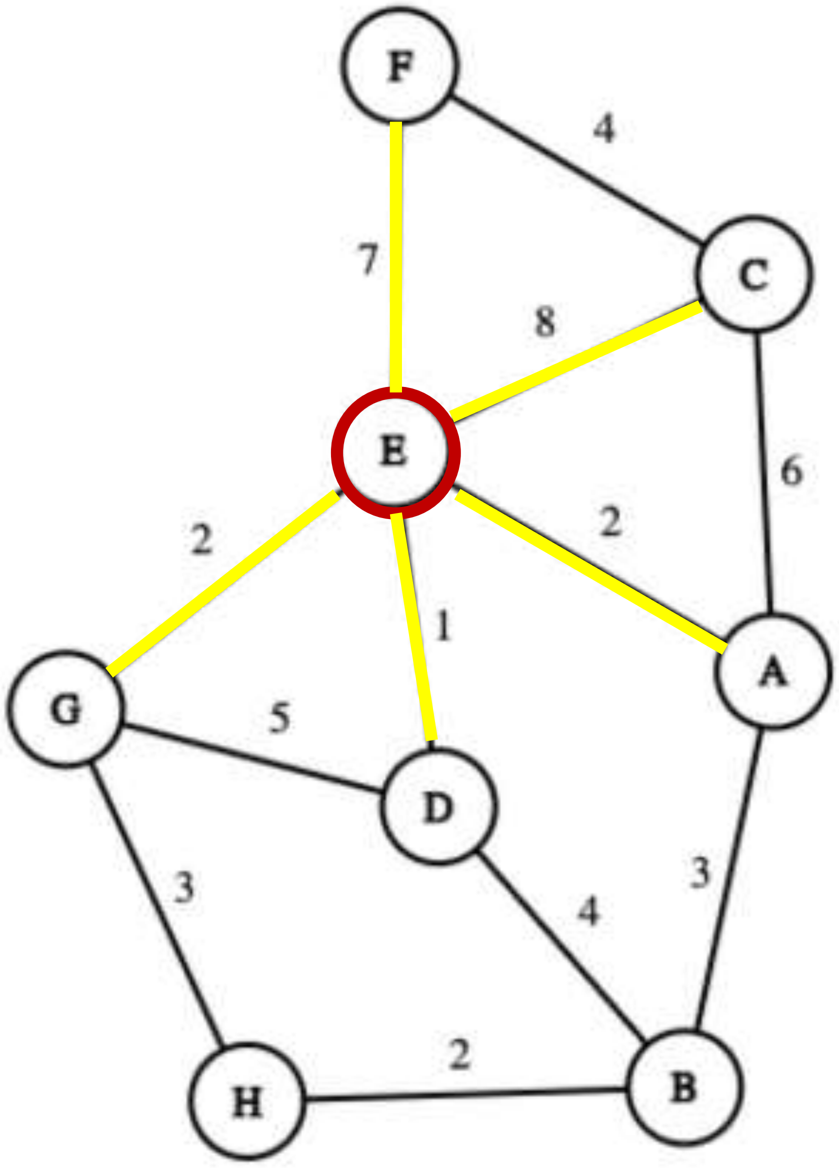


QUESTION-2

We start with vertex E. E is added to the tree. So, we mark E as known.

Check the edges (u,v) such that u is in the tree and v is not, and find which unknown vertex is the closest to the tree.

Smallest cost is the edge (E,D), so we move to D.

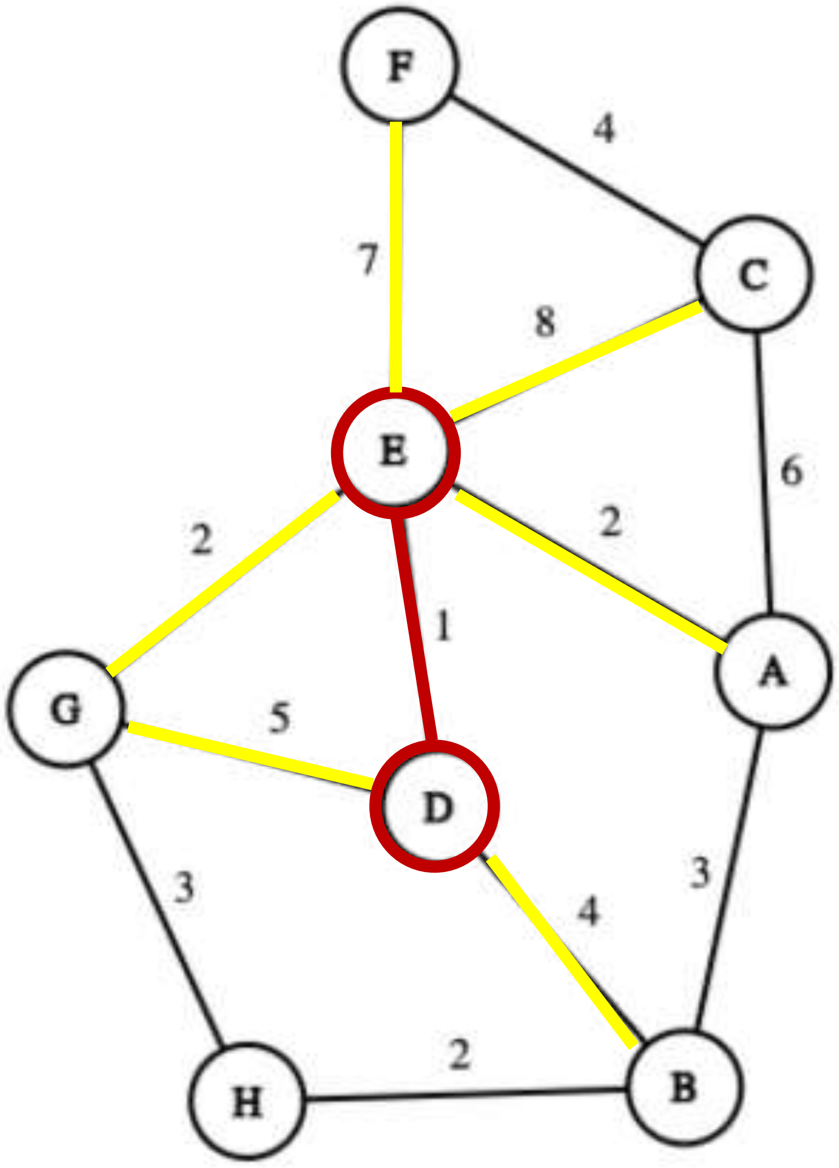


QUESTION-2

D is added to the tree. So, we mark D as known.

Check the edges (u,v) such that u is in the tree and v is not, and find which unknown vertex is the closest to the tree.

Smallest cost is the edges (E,G) and (E,A). Pick one. So we move to A.

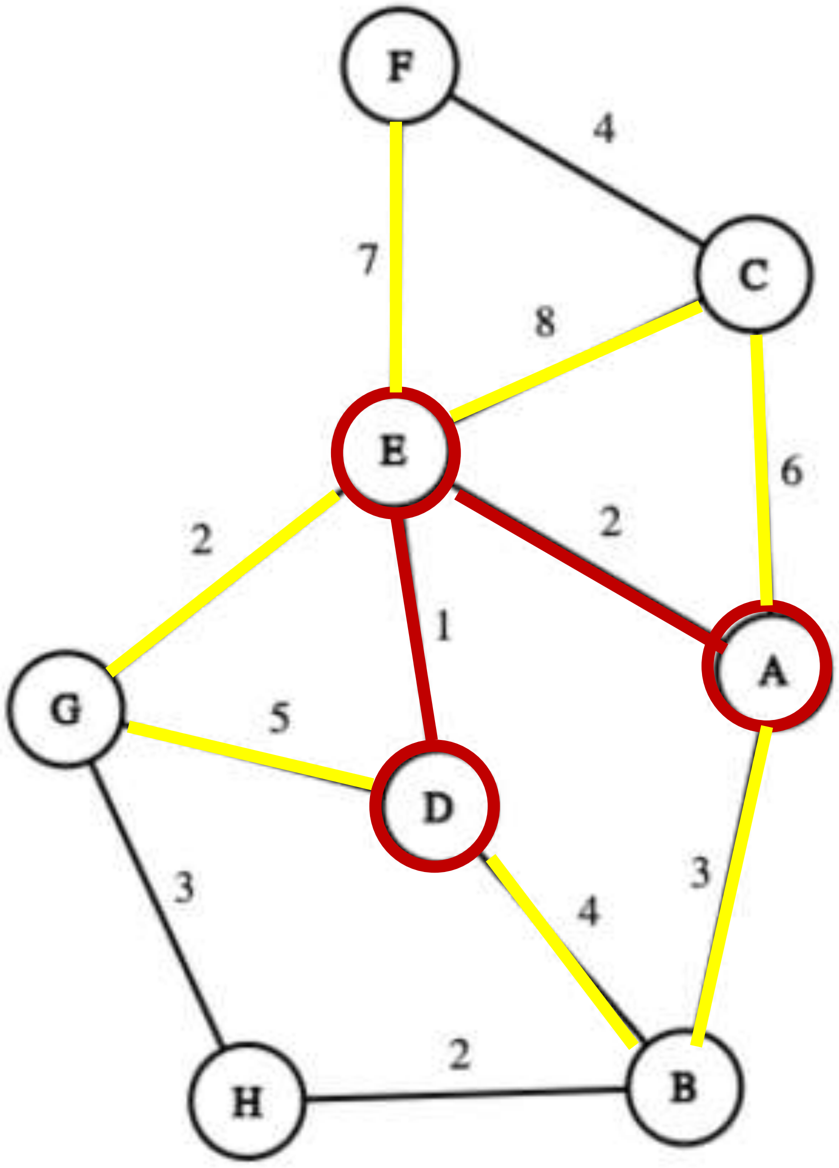


QUESTION-2

A is added to the tree. So, we mark A as known.

Check the edges (u,v) such that u is in the tree and v is not, and find which unknown vertex is the closest to the tree.

Smallest cost is the edge (E,G). So we move to G.

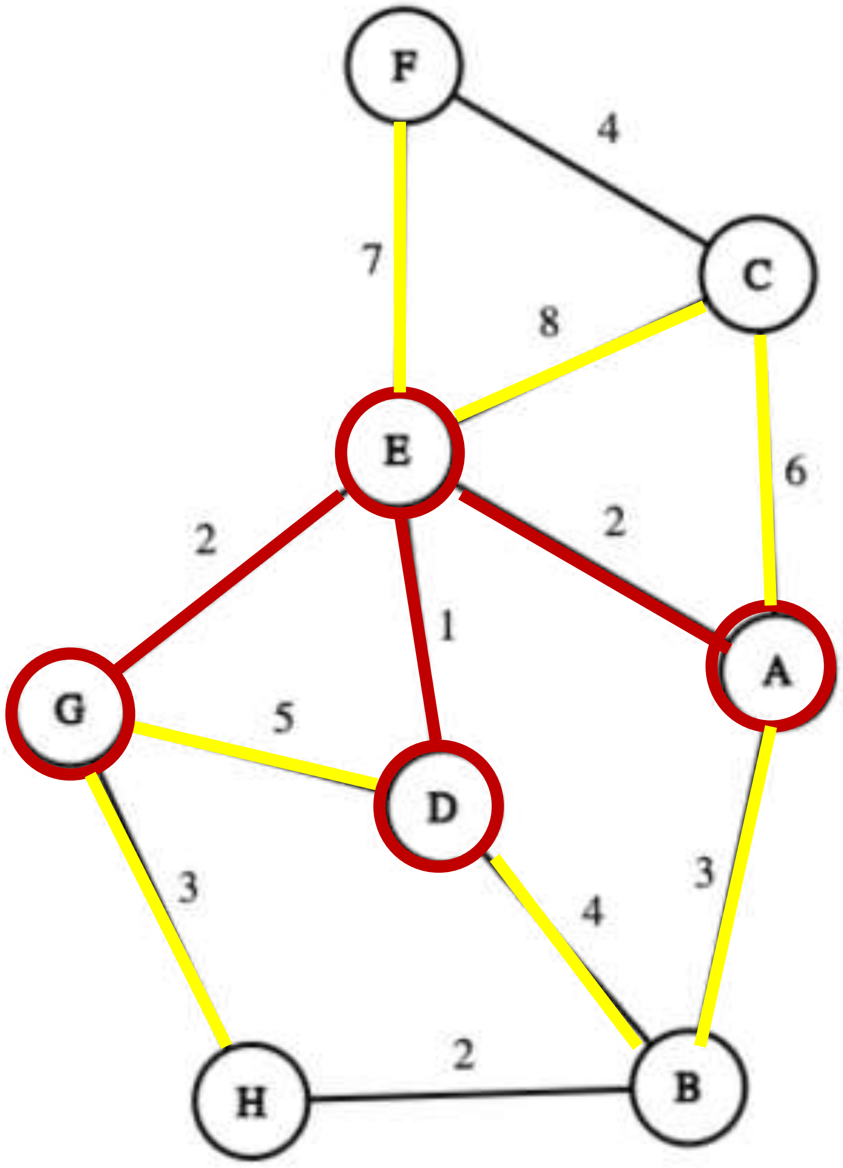


QUESTION-2

G is added to the tree. So, we mark G as known.

Check the edges (u,v) such that u is in the tree and v is not, and find which unknown vertex is the closest to the tree.

Smallest cost is the edges (A,B) and (G,H). Pick one. So we move to H.

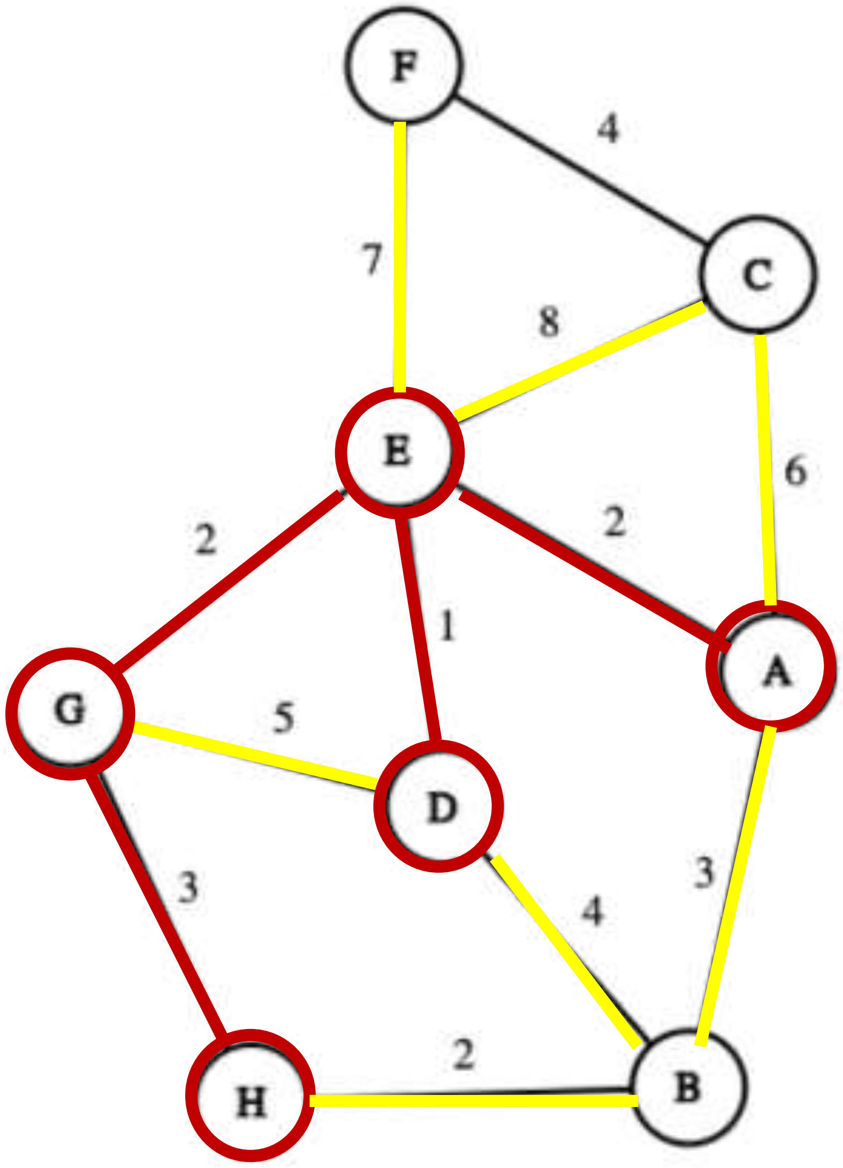


QUESTION-2

H is added to the tree. So, we mark H as known.

Check the edges (u,v) such that u is in the tree and v is not, and find which unknown vertex is the closest to the tree.

Smallest cost is the edge (H,B). So we move to B.

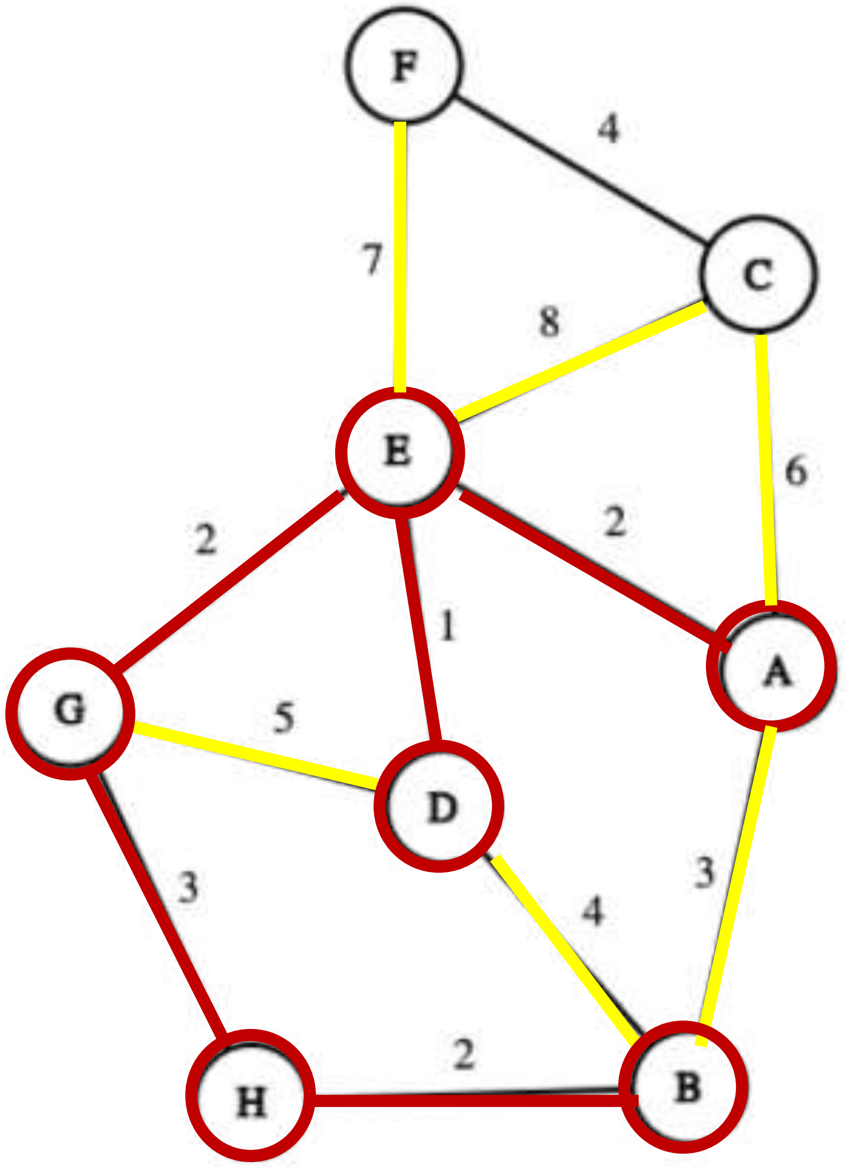


QUESTION-2

B is added to the tree. So, we mark B as known.

Check the edges (u,v) such that u is in the tree and v is not, and find which unknown vertex is the closest to the tree.

Smallest cost is the edge (A,C). So we move to C.

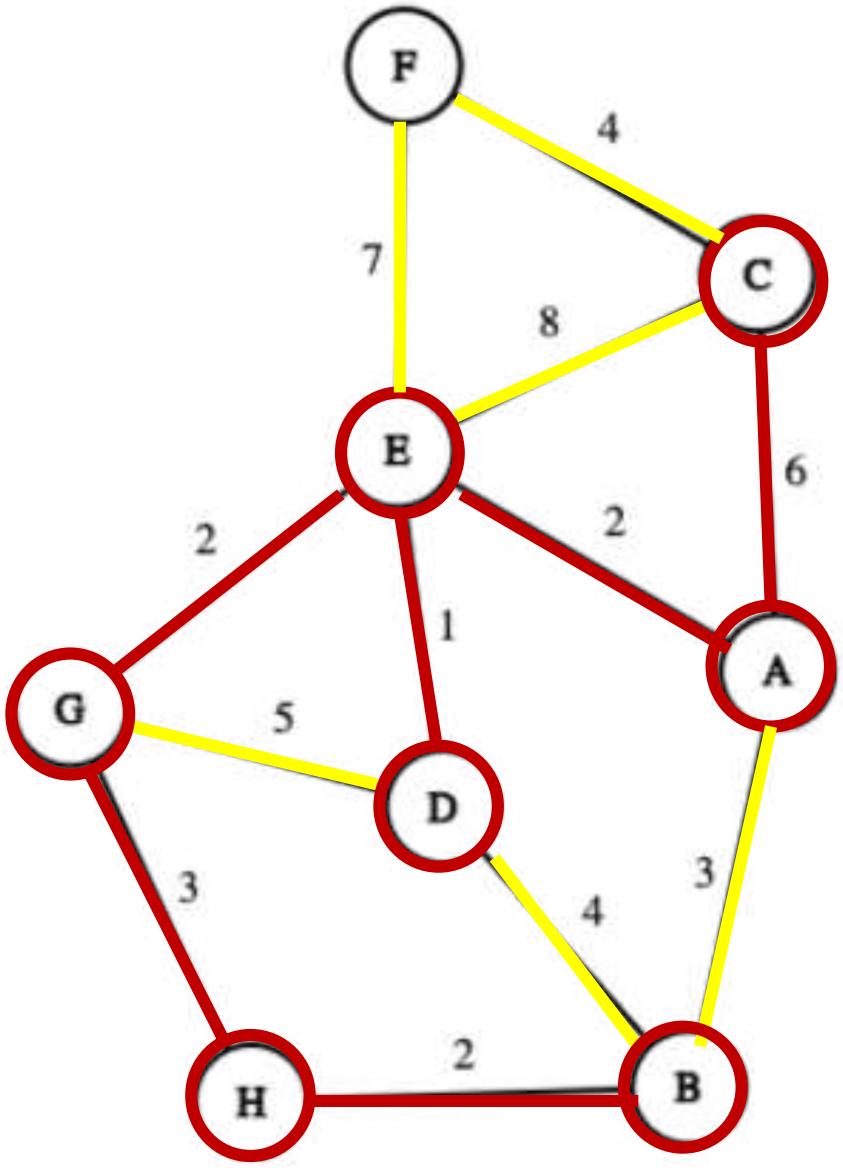


QUESTION-2

C is added to the tree. So, we mark C as known.

Check the edges (u,v) such that u is in the tree and v is not, and find which unknown vertex is the closest to the tree.

Smallest cost is the edge (C,F). So we move to F.

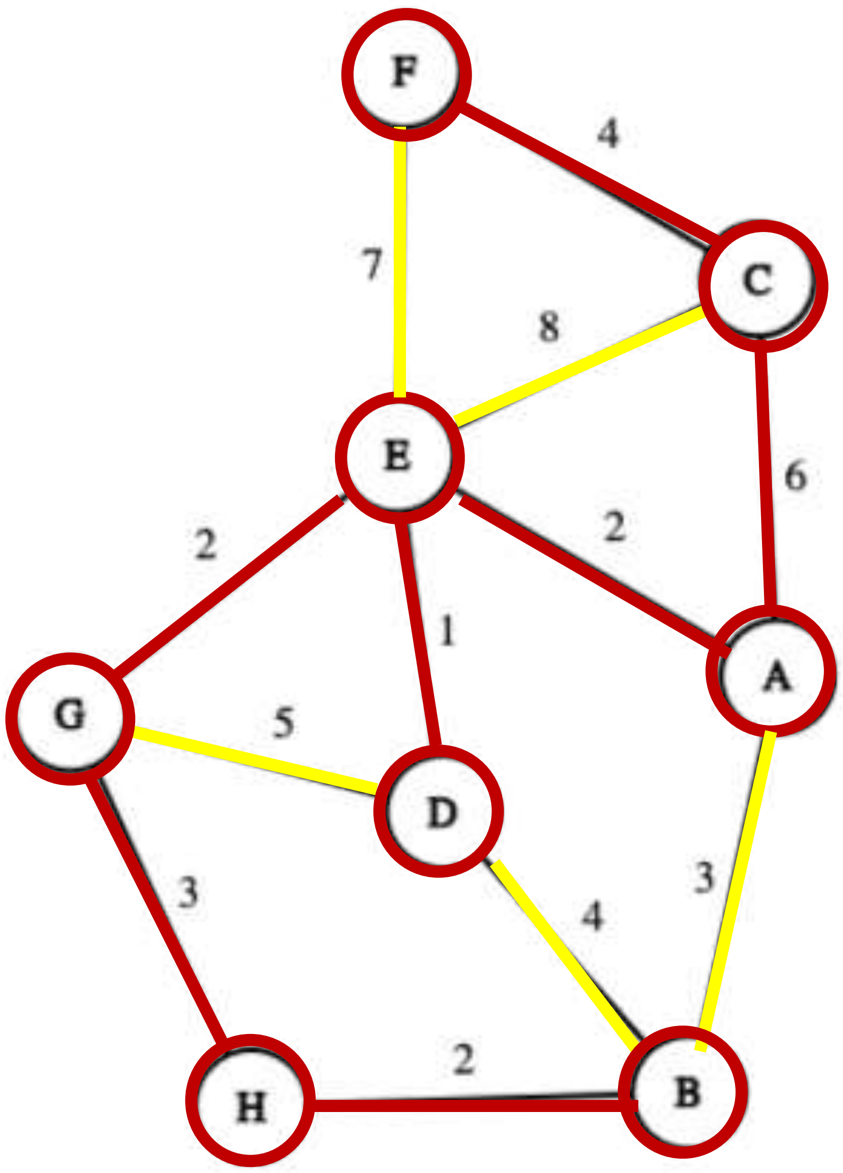


QUESTION-2

F is added to the tree. So, we mark F as known.

Check the edges (u,v) such that u is in the tree and v is not, and find which unknown vertex is the closest to the tree.

No more unknown vertices. The minimum spanning tree is shown with the red pathway. Terminate.



QUESTION-3

First, we sort the edges in ascending order with respect to their weights.

E to D = 1

E to G = 2

E to A = 2

H to B = 2

G to H = 3

A to B = 3

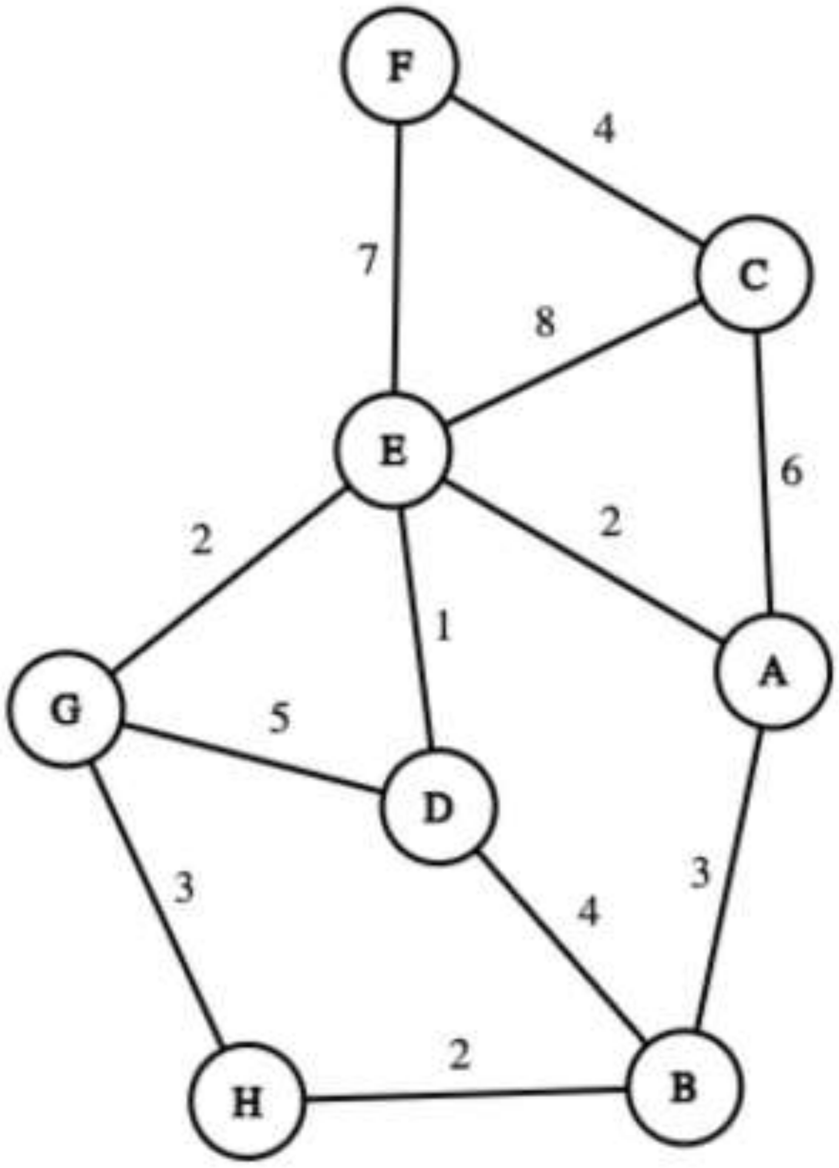
F to C = 4

G to D = 5

C to A = 6

E to F = 7

E to C = 8



Note that U to V = V to U since graph is undirected.

QUESTION-3

Start with the shortest edge. Union its vertices IF they are NOT already in the same tree.

E and D are connected.

E to D = 1

E to G = 2

E to A = 2

H to B = 2

G to H = 3

A to B = 3

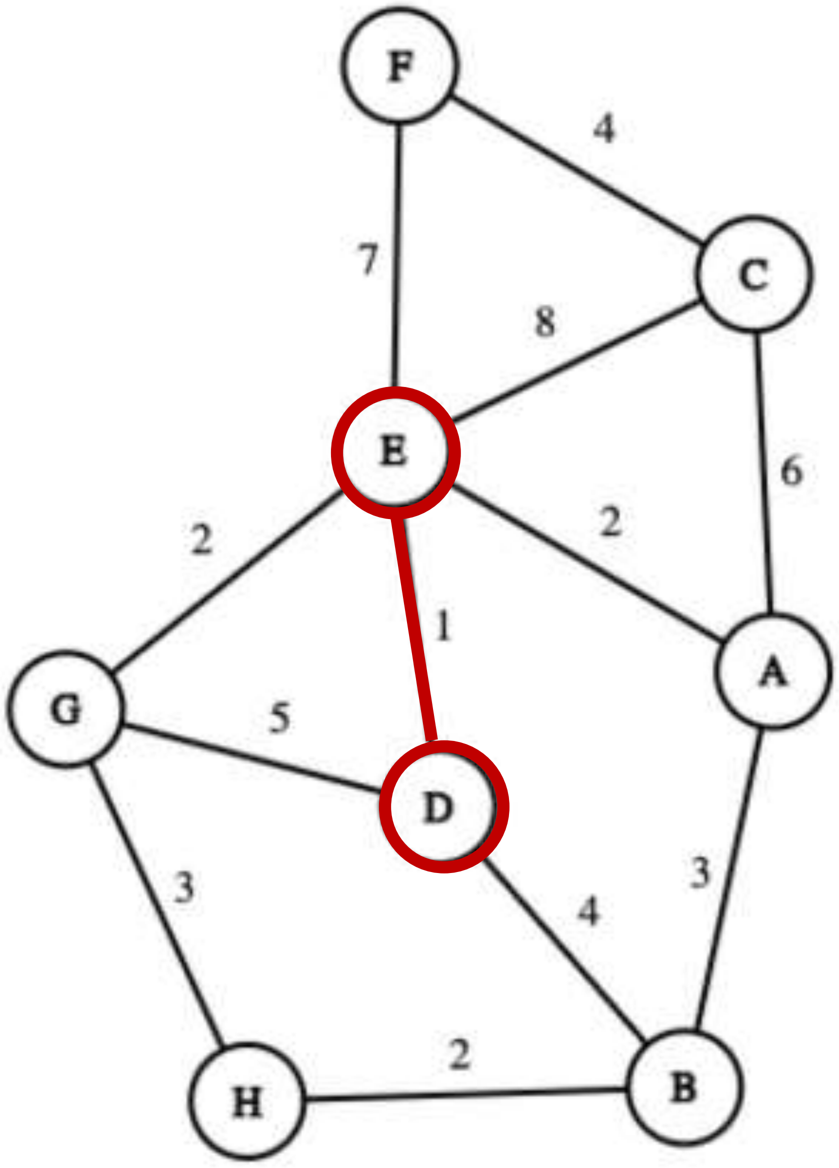
F to C = 4

G to D = 5

C to A = 6

E to F = 7

E to C = 8



QUESTION-3

Follow the next shortest edge. Union its vertices IF they are NOT already in the same tree.

E and G are connected.

E to D = 1

E to G = 2

E to A = 2

H to B = 2

G to H = 3

A to B = 3

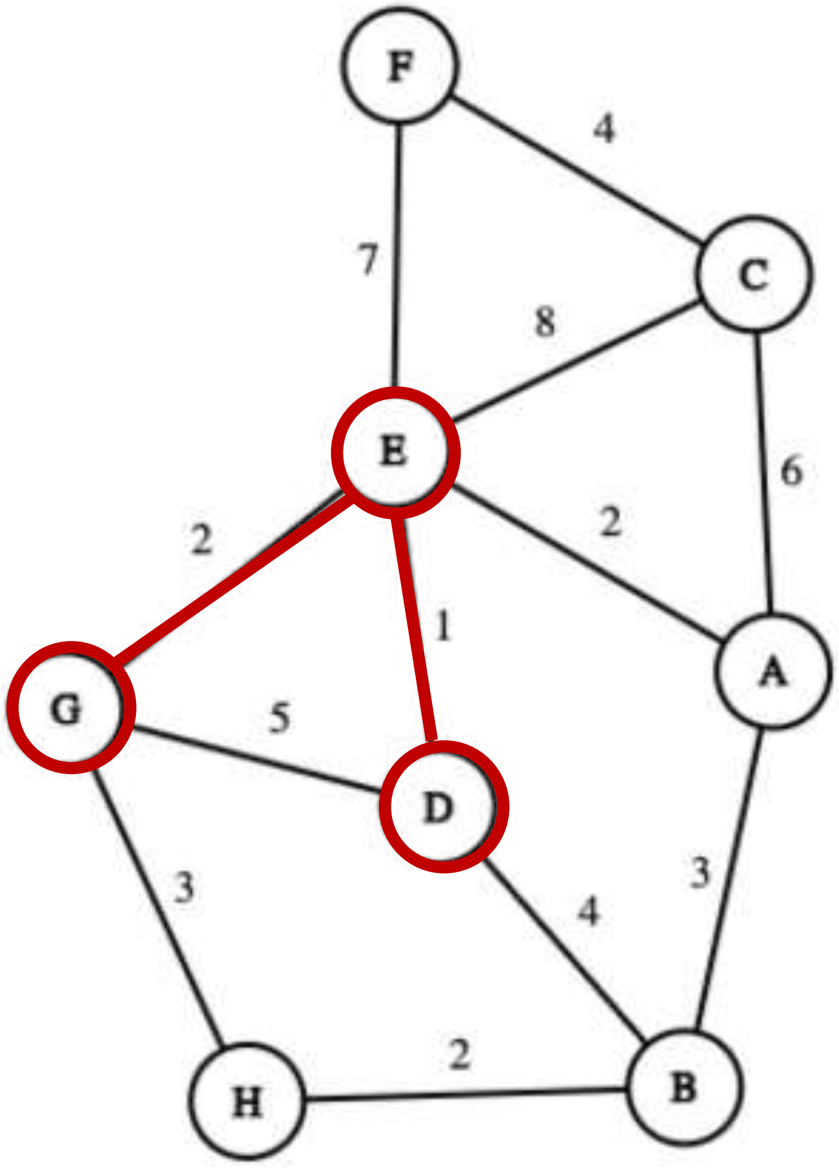
F to C = 4

G to D = 5

C to A = 6

E to F = 7

E to C = 8



QUESTION-3

Follow the next shortest edge. Union its vertices IF they are NOT already in the same tree.

E and A are connected.

E to D = 1

E to G = 2

E to A = 2

H to B = 2

G to H = 3

A to B = 3

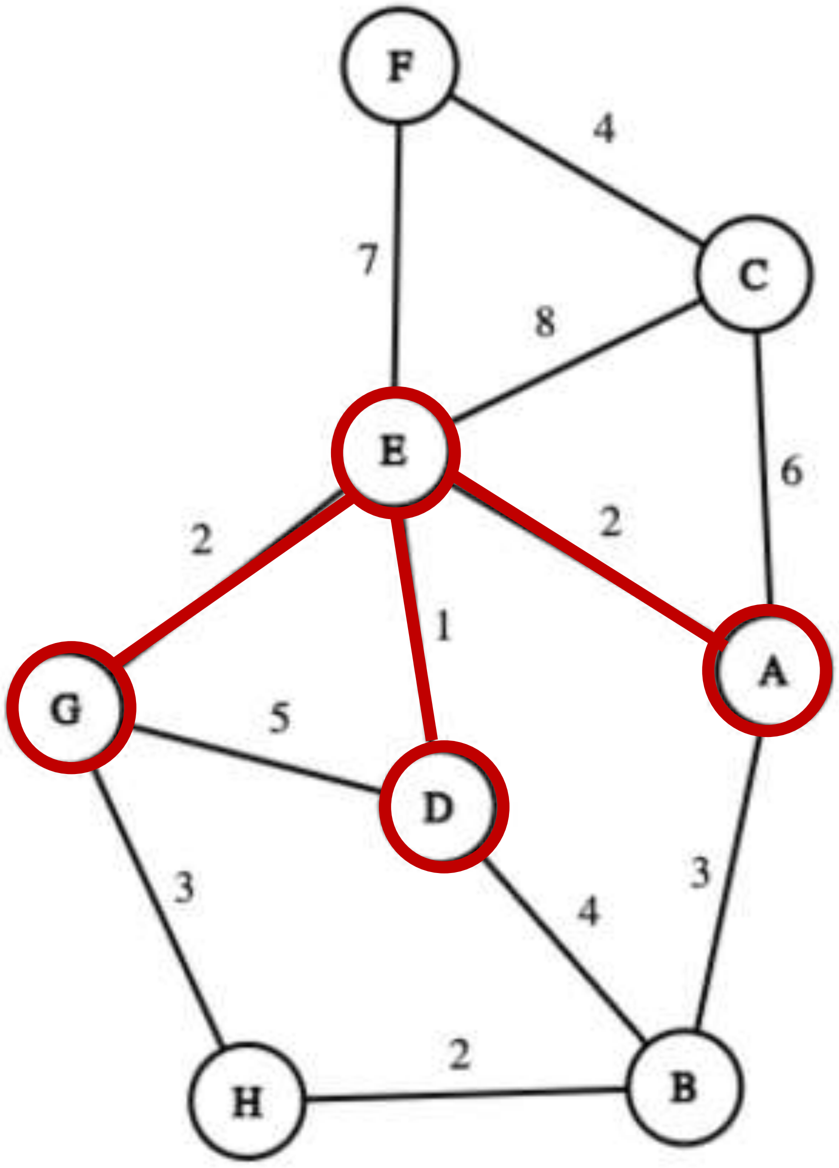
F to C = 4

G to D = 5

C to A = 6

E to F = 7

E to C = 8



QUESTION-3

Follow the next shortest edge. Union its vertices IF they are NOT already in the same tree.

H and B are connected.

E to D = 1

E to G = 2

E to A = 2

H to B = 2

G to H = 3

A to B = 3

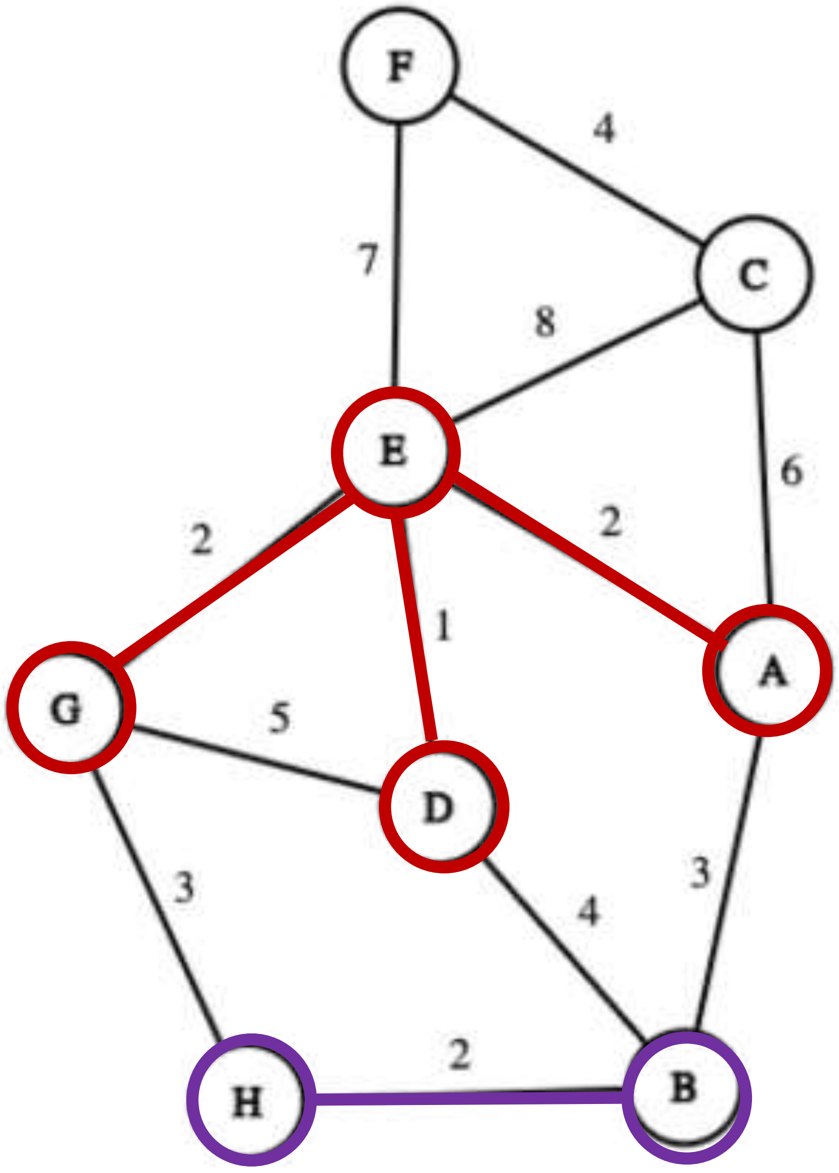
F to C = 4

G to D = 5

C to A = 6

E to F = 7

E to C = 8



QUESTION-3

Follow the next shortest edge. Union its vertices IF they are NOT already in the same tree.

G and H belong to different trees although they are connected to somewhere. We union them by merging two trees.

G and H are connected. Two trees merge into one.

E to D = 1

E to G = 2

E to A = 2

H to B = 2

G to H = 3

A to B = 3

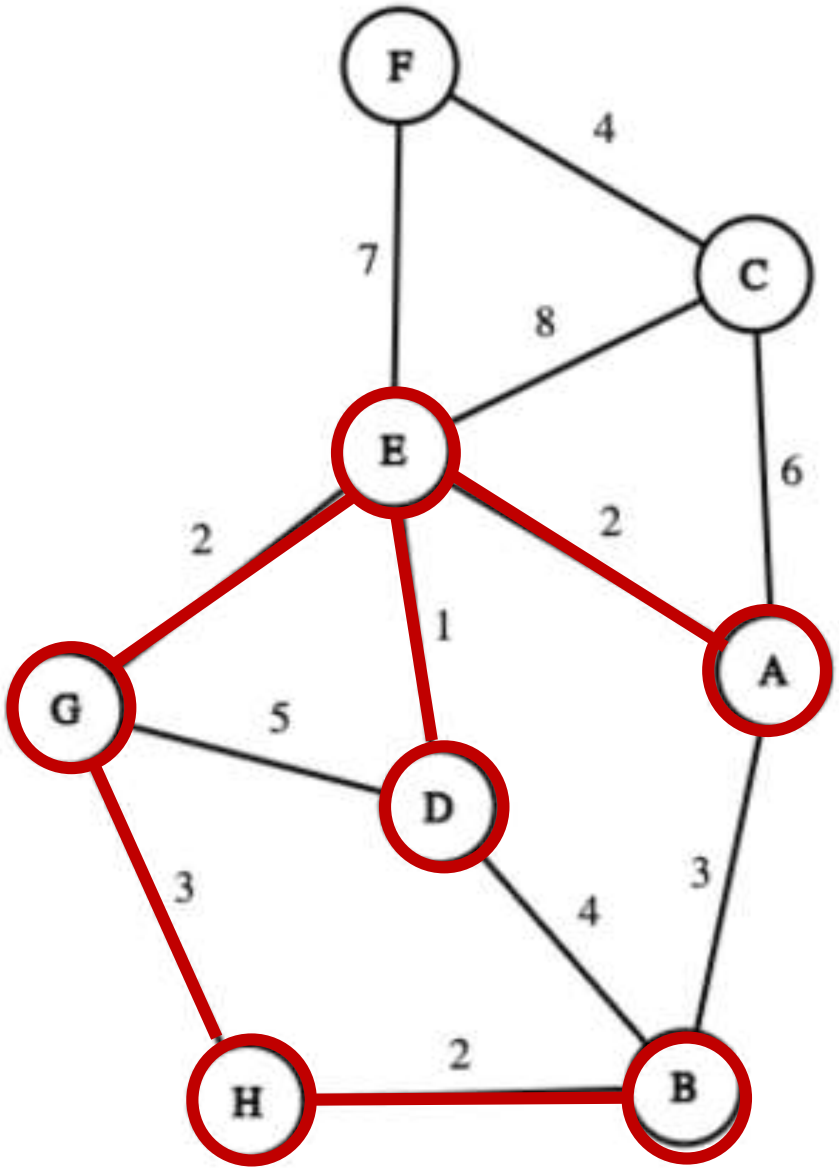
F to C = 4

G to D = 5

C to A = 6

E to F = 7

E to C = 8



QUESTION-3

Follow the next shortest edge. Union its vertices IF they are NOT already in the same tree.

A and B are already in the same tree. Skip.

E to D = 1

E to G = 2

E to A = 2

H to B = 2

G to H = 3

A to B = 3

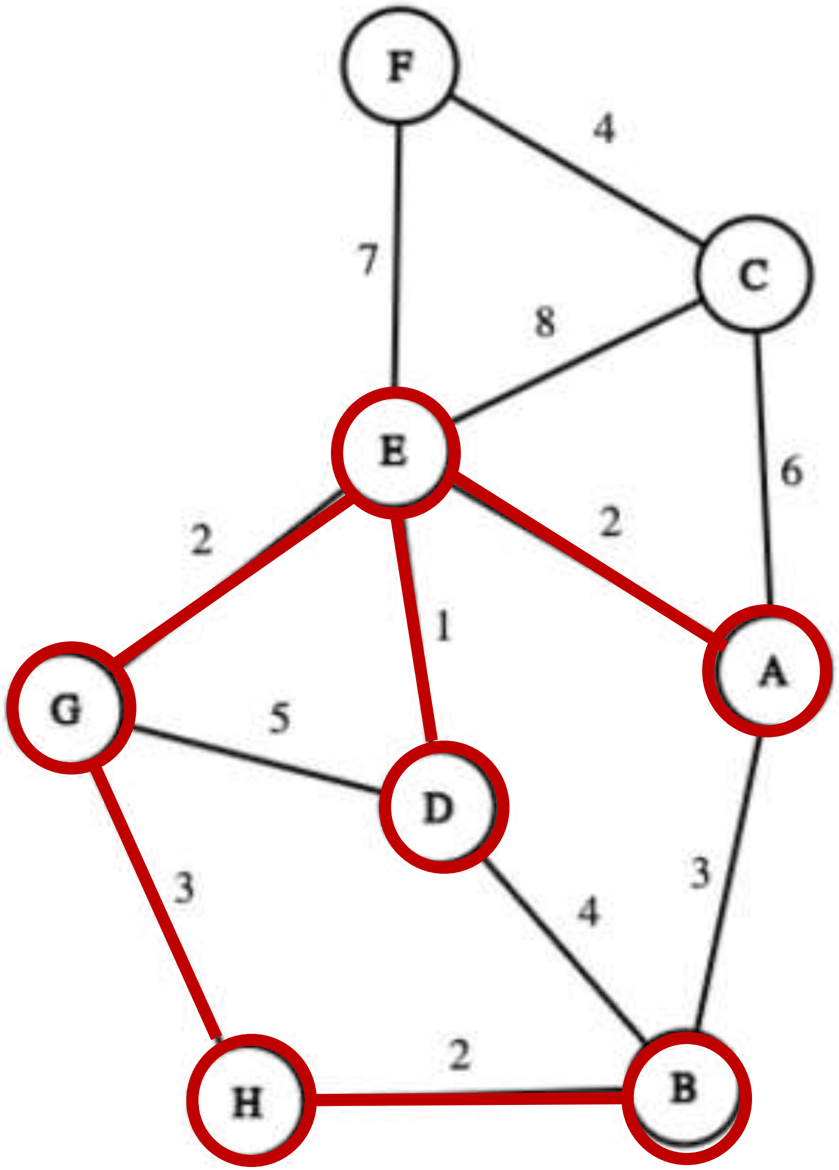
F to C = 4

G to D = 5

C to A = 6

E to F = 7

E to C = 8



QUESTION-3

Follow the next shortest edge. Union its vertices IF they are NOT already in the same tree.

F and C are connected.

E to D = 1

E to G = 2

E to A = 2

H to B = 2

G to H = 3

A to B = 3

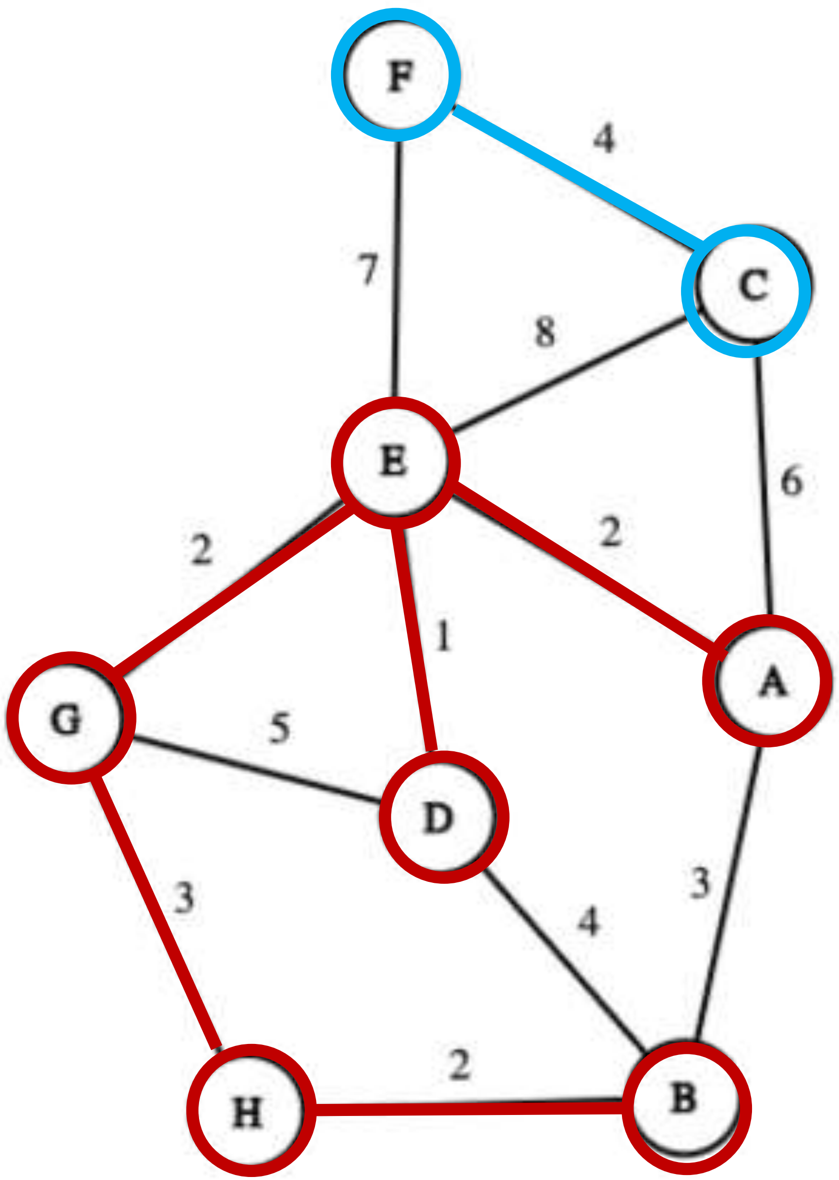
F to C = 4

G to D = 5

C to A = 6

E to F = 7

E to C = 8



QUESTION-3

Follow the next shortest edge. Union its vertices IF they are NOT already in the same tree.

F and C are connected.

E to D = 1

E to G = 2

E to A = 2

H to B = 2

G to H = 3

A to B = 3

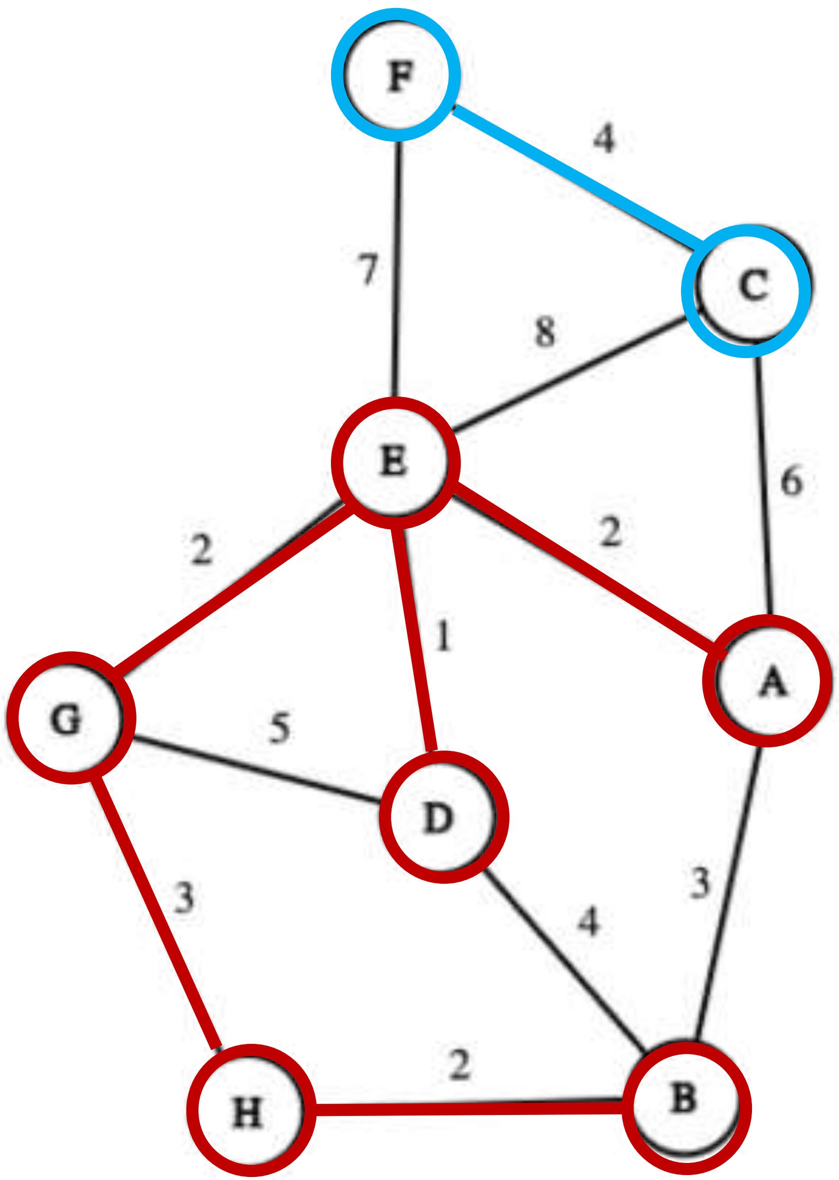
F to C = 4

G to D = 5

C to A = 6

E to F = 7

E to C = 8



QUESTION-3

Follow the next shortest edge. Union its vertices IF they are NOT already in the same tree.

A and C belong to different trees although they are connected to somewhere. We union them by merging two trees.

A and C are connected. Two trees merge into one.

E to D = 1

E to G = 2

E to A = 2

H to B = 2

G to H = 3

A to B = 3

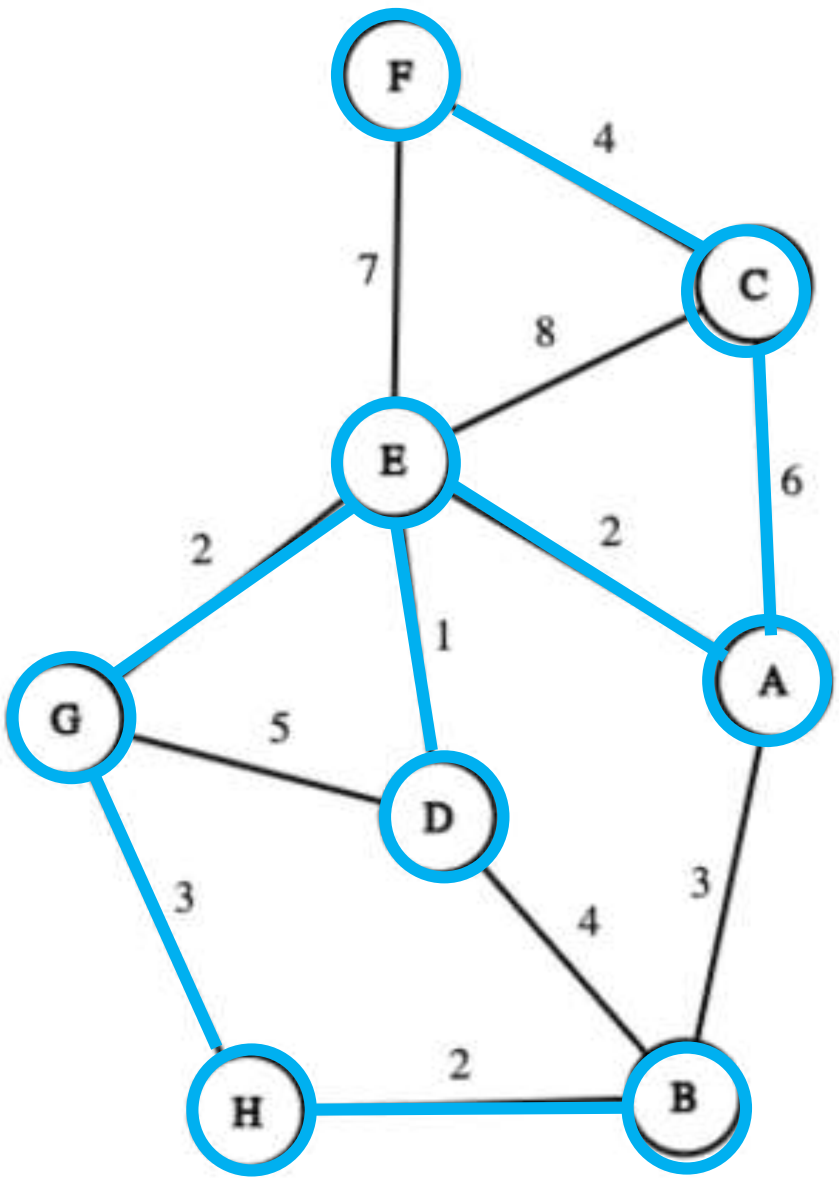
F to C = 4

G to D = 5

C to A = 6

E to F = 7

E to C = 8



QUESTION-3

Follow the next shortest edge. Union its vertices IF they are NOT already in the same tree.

E and F are already in the same tree. Skip.

E to D = 1

E to G = 2

E to A = 2

H to B = 2

G to H = 3

A to B = 3

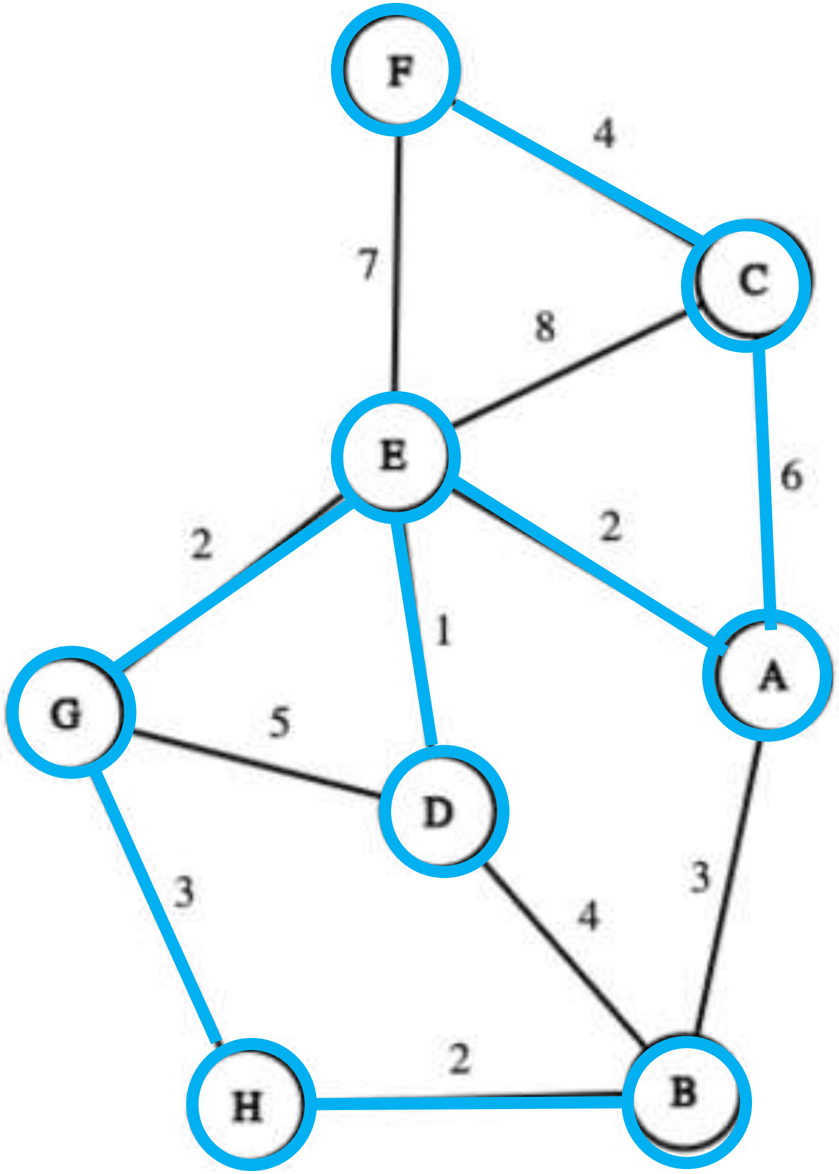
F to C = 4

G to D = 5

C to A = 6

E to F = 7

E to C = 8



QUESTION-3

Follow the next shortest edge. Union its vertices IF they are NOT already in the same tree.

E and C are already in the same tree. No unvisited edges left. Resulting minimum spaning tree is shown with blue. Terminate.

E to D = 1

E to G = 2

E to A = 2

H to B = 2

G to H = 3

A to B = 3

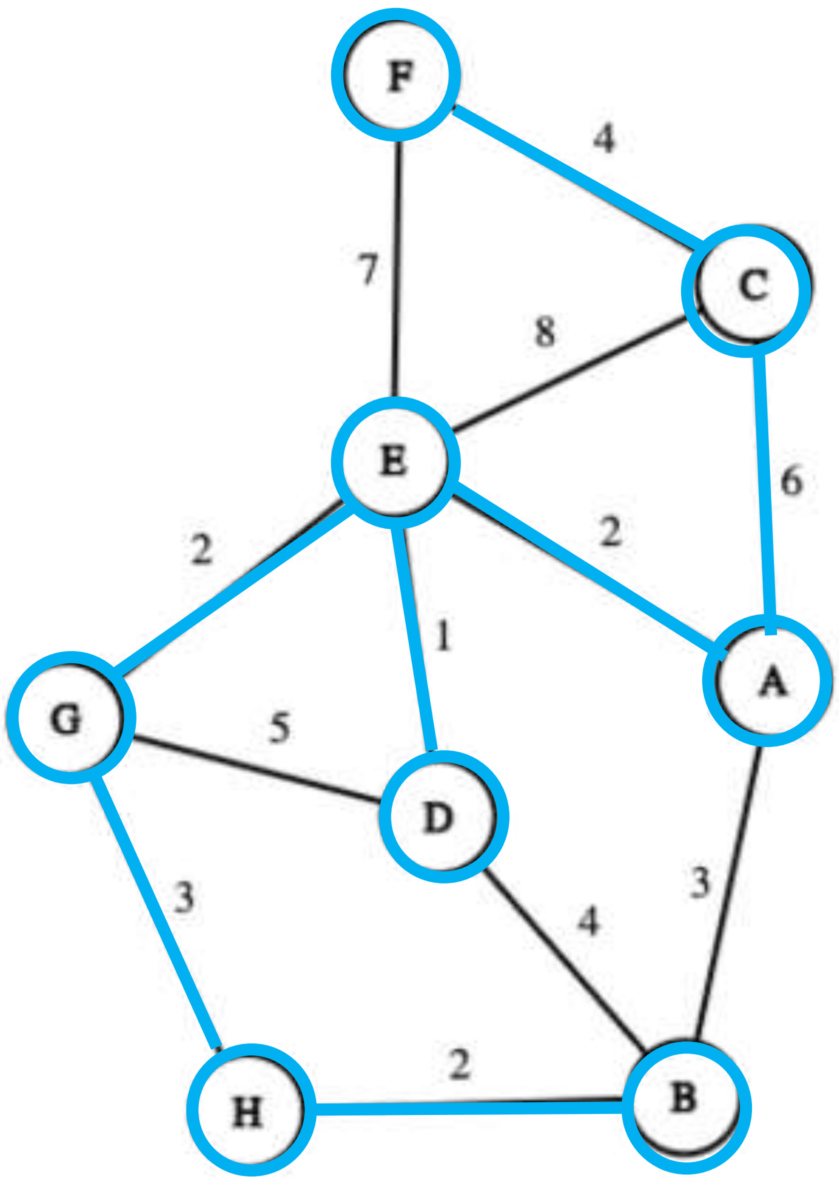
F to C = 4

G to D = 5

C to A = 6

E to F = 7

E to C = 8



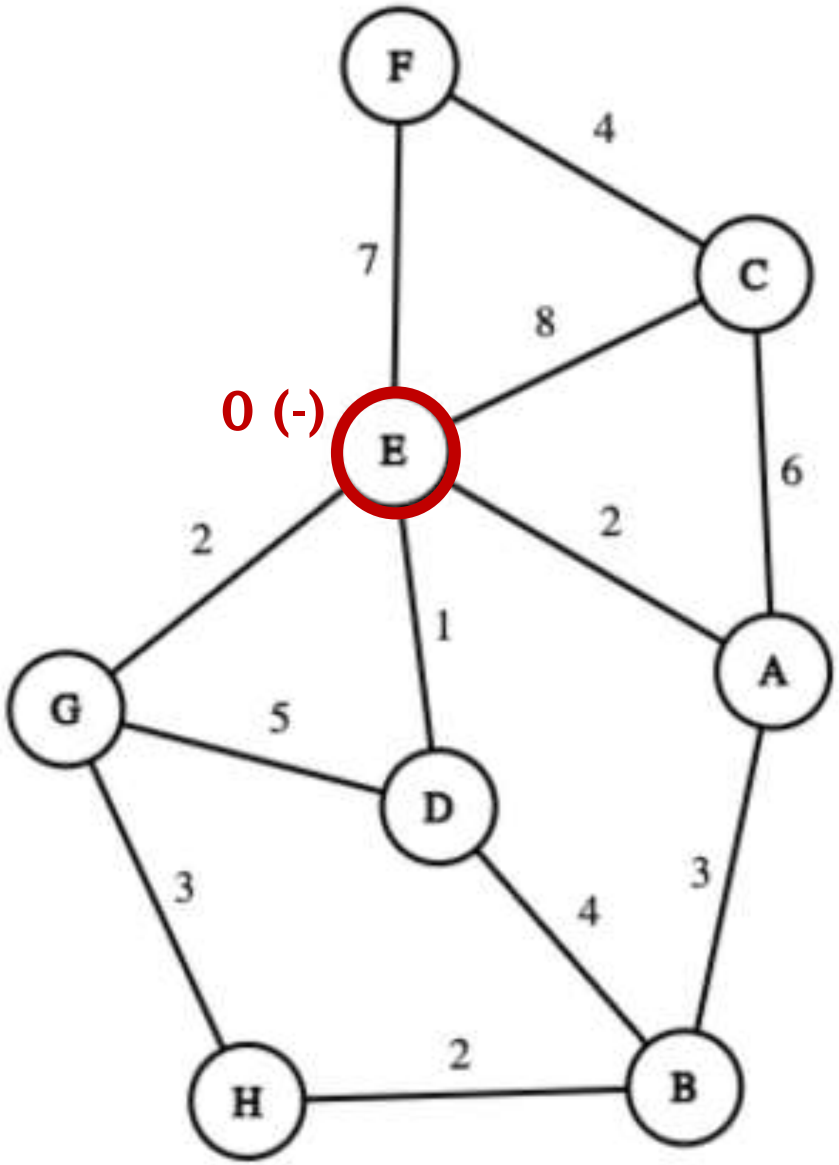
QUESTION-4

To keep track of BFS, we have a queue. On the graph, next to the vertices, the path and the distance through that vertex is shown in red.

We start with vertex E. Automatically, distance from E to E is 0. E is marked ‘‘known’’

QUEUE:

empty



QUESTION-4

To keep track of BFS, we have a queue. On the graph, next to the vertices, the path and the distance through that vertex is shown in red.

We enqueue the unknown vertices 1 away from E. So F, C, A, D, G are enqueued.

QUEUE:

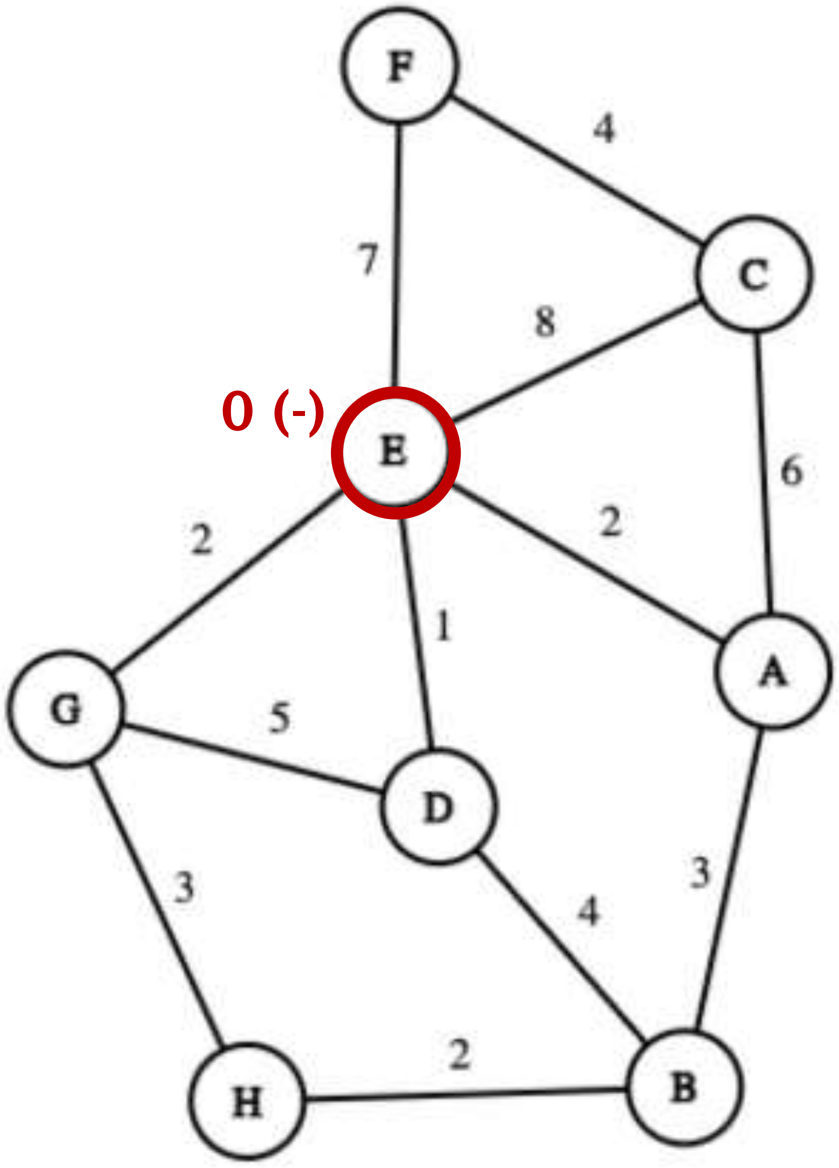
E to F, cost=7

E to C, cost=8

E to A, cost=2

E to D, cost=1

E to G, cost=2



QUESTION-4

To keep track of BFS, we have a queue. On the graph, next to the vertices, the path and the distance through that vertex is shown in red.

Go to F. F is known. Its cost and path are updated. Enqueue the unknown vertices 1 away from F. So, C is enqueued.

QUEUE:

E to F, cost=7

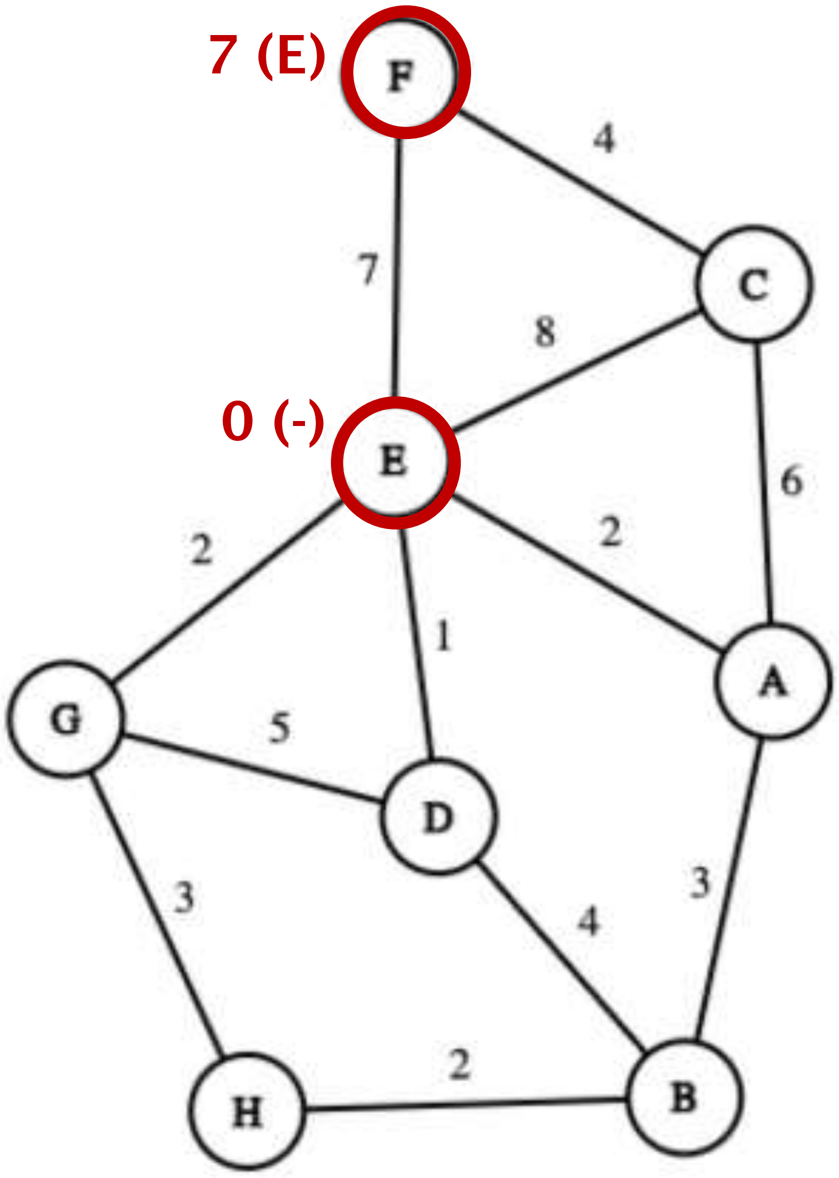
E to C, cost=8

E to A, cost=2

E to D, cost=1

E to G, cost=2

F to C, cost=7+4=11



QUESTION-4

To keep track of BFS, we have a queue. On the graph, next to the vertices, the path and the distance through that vertex is shown in red.

Go to C. C is known. Its cost and path are updated. Enqueue the unknown vertices 1 away from C. So, A is enqueued.

QUEUE:

E to C, cost=8

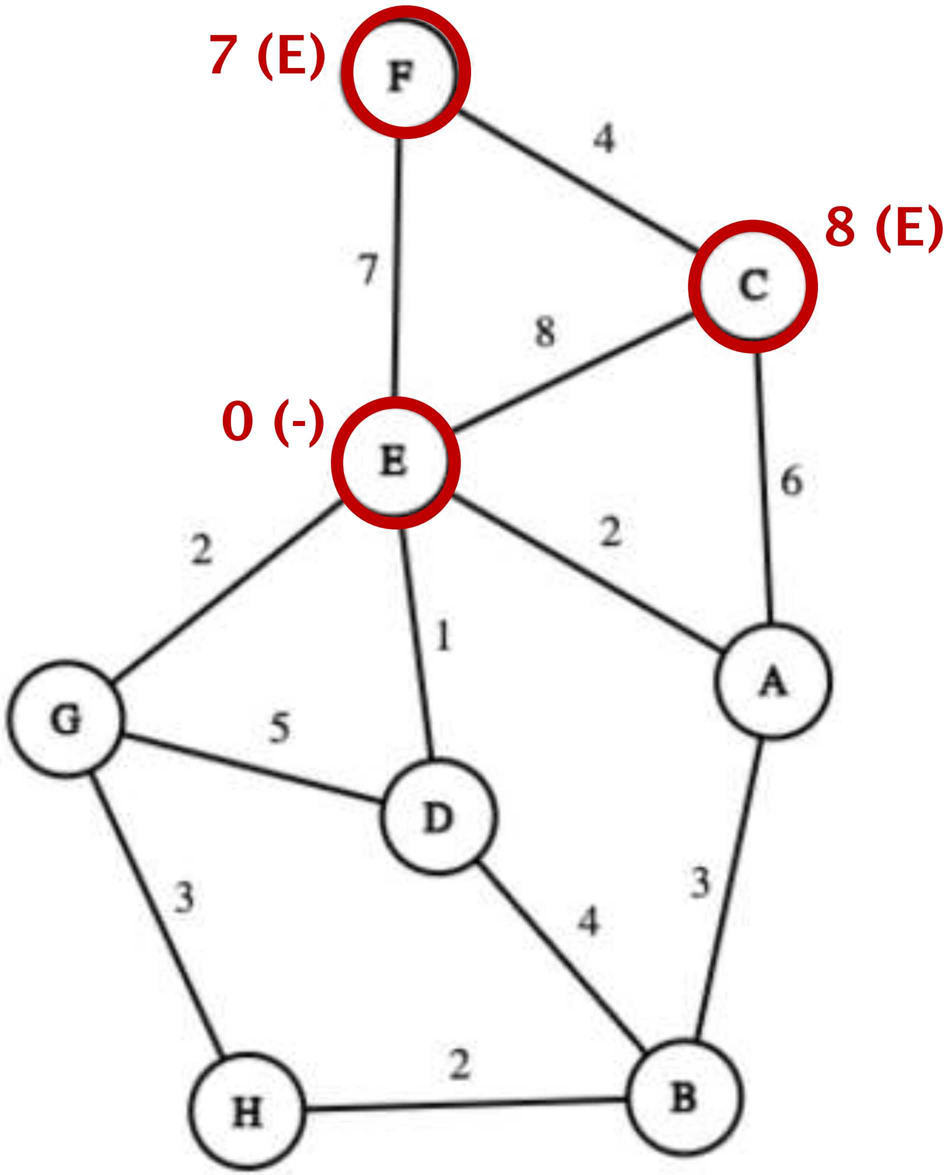
E to A, cost=2

E to D, cost=1

E to G, cost=2

F to C, cost=7+4=11

C to A, cost=8+6=14



QUESTION-4

To keep track of BFS, we have a queue. On the graph, next to the vertices, the path and the distance through that vertex is shown in red.

Go to A. A is known. Its cost and path are updated. Enqueue the unknown vertices 1 away from A. So, B is enqueued.

QUEUE:

E to A, cost=2

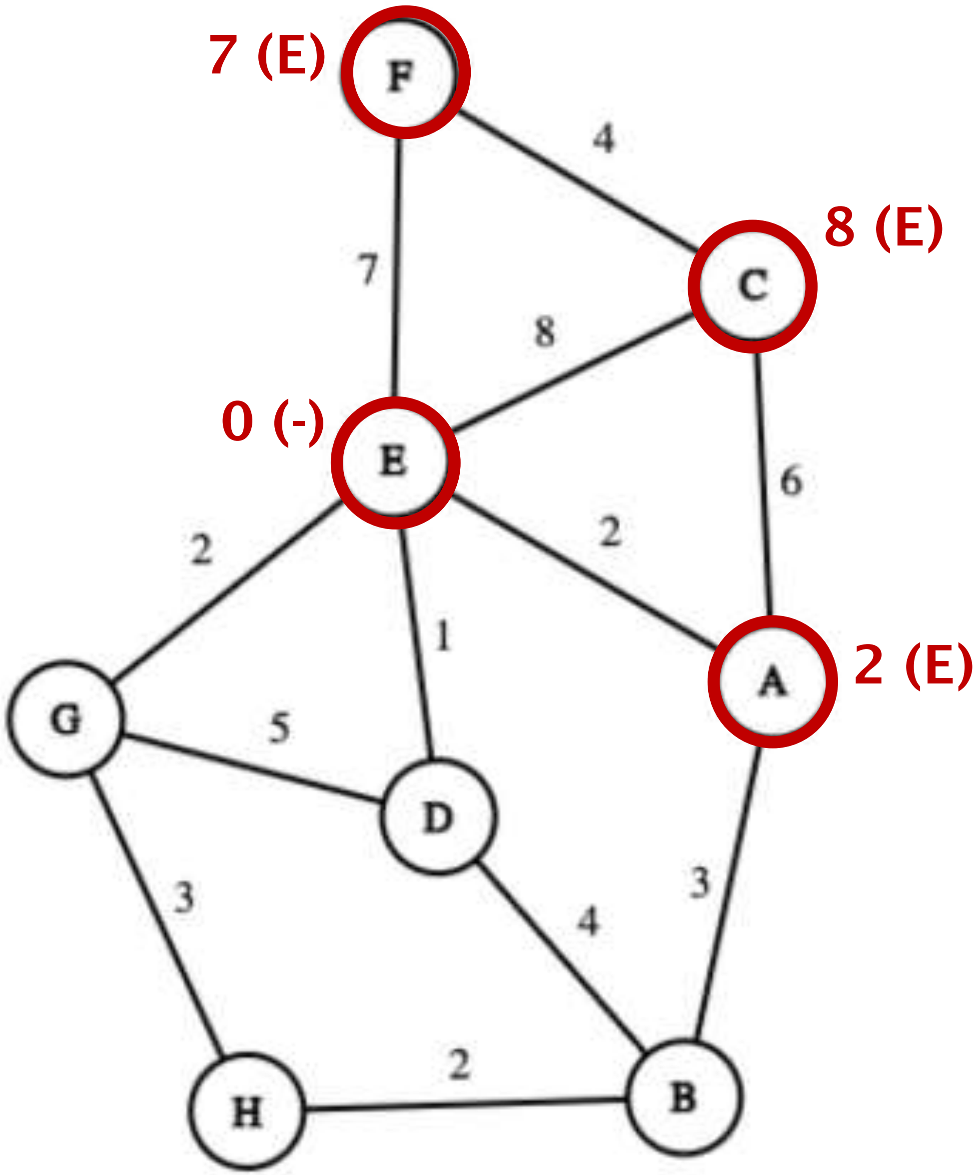
E to D, cost=1

E to G, cost=2

F to C, cost=7+4=11

C to A, cost=8+6=14

A to B, cost=2+3=5



QUESTION-4

To keep track of BFS, we have a queue. On the graph, next to the vertices, the path and the distance through that vertex is shown in red.

Go to D. D is known. Its cost and path are updated. Enqueue the unknown vertices 1 away from D. So, B and G are enqueued.

QUEUE:

E to D, cost=1

E to G, cost=2

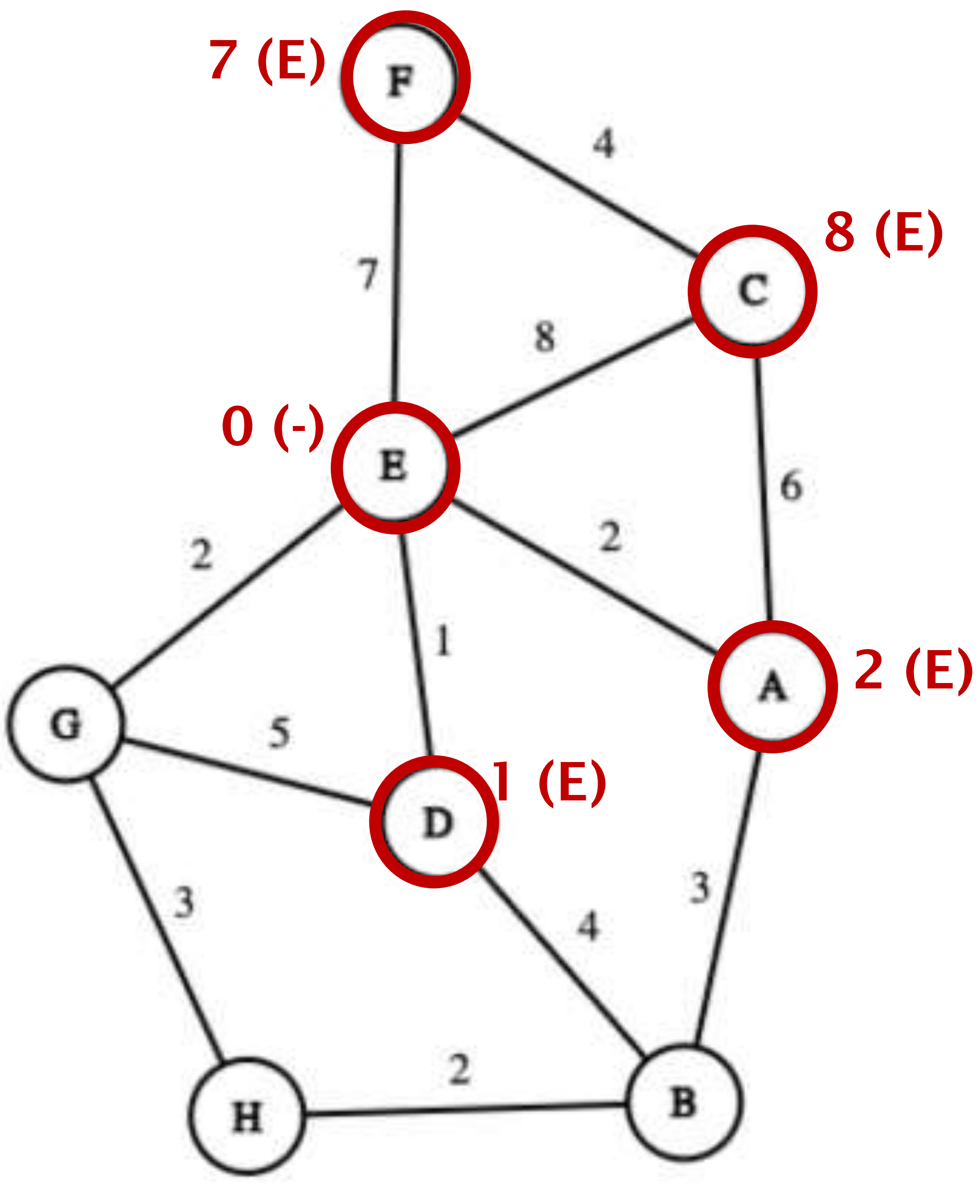
F to C, cost=7+4=11

C to A, cost=8+6=14

A to B, cost=2+3=5

D to G, cost=5+1=6

D to B, cost=1+4=5



QUESTION-4

To keep track of BFS, we have a queue. On the graph, next to the vertices, the path and the distance through that vertex is shown in red.

Go to G. G is known. Its cost and path are updated. Enqueue the unknown vertices 1 away from G. So, H is enqueued.

QUEUE:

E to G, cost=2

F to C, cost=7+4=11

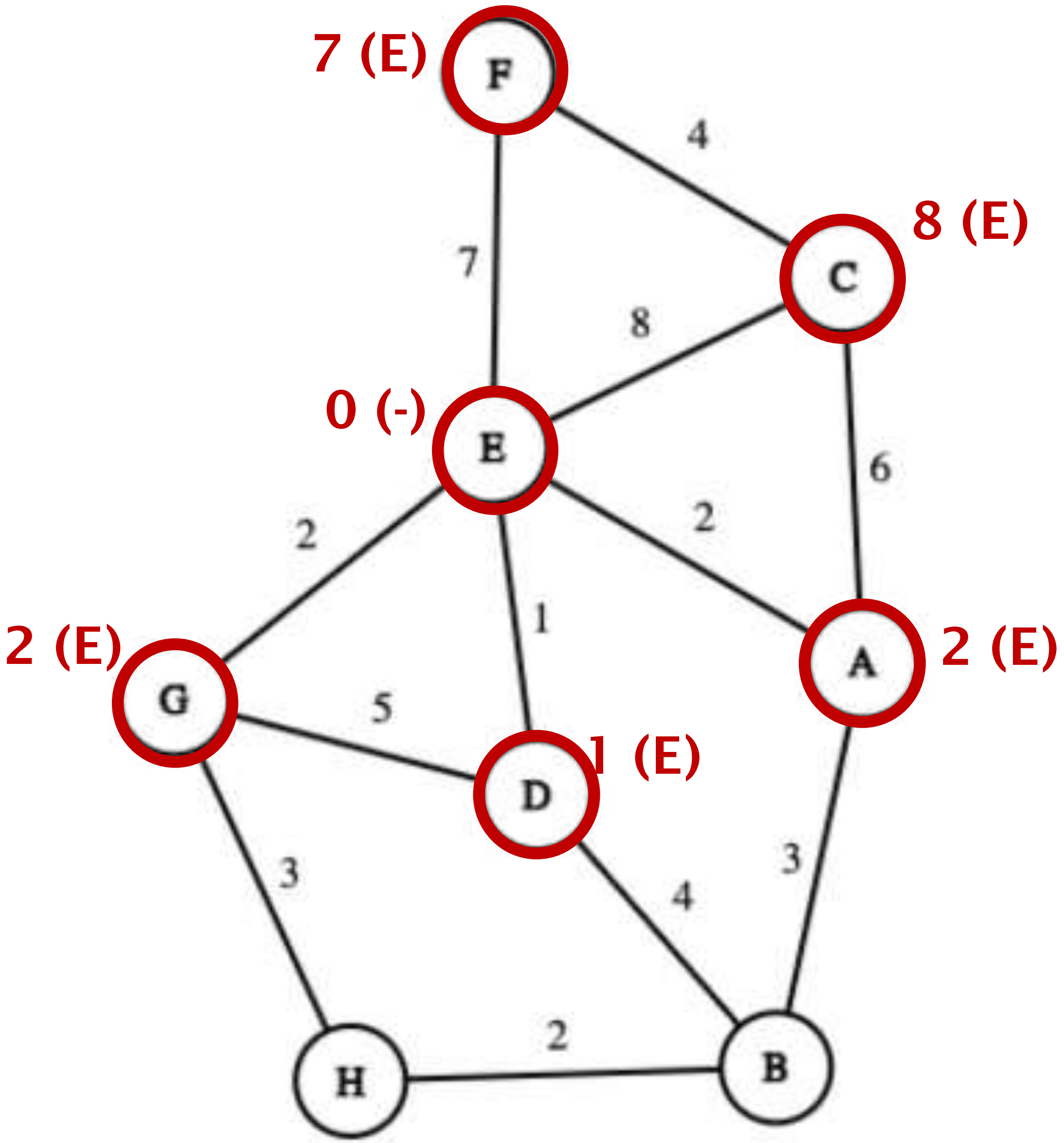
C to A, cost=8+6=14

A to B, cost=2+3=5

D to G, cost=5+1=6

D to B, cost=1+4=5

G to H, cost=2+3=5



QUESTION-4

To keep track of BFS, we have a queue. On the graph, next to the vertices, the path and the distance through that vertex is shown in red.

Both F and C are known. Cost: 11 > 8; therefore, path to C is NOT updated. Skip.

QUEUE:

F to C, cost=7+4=11

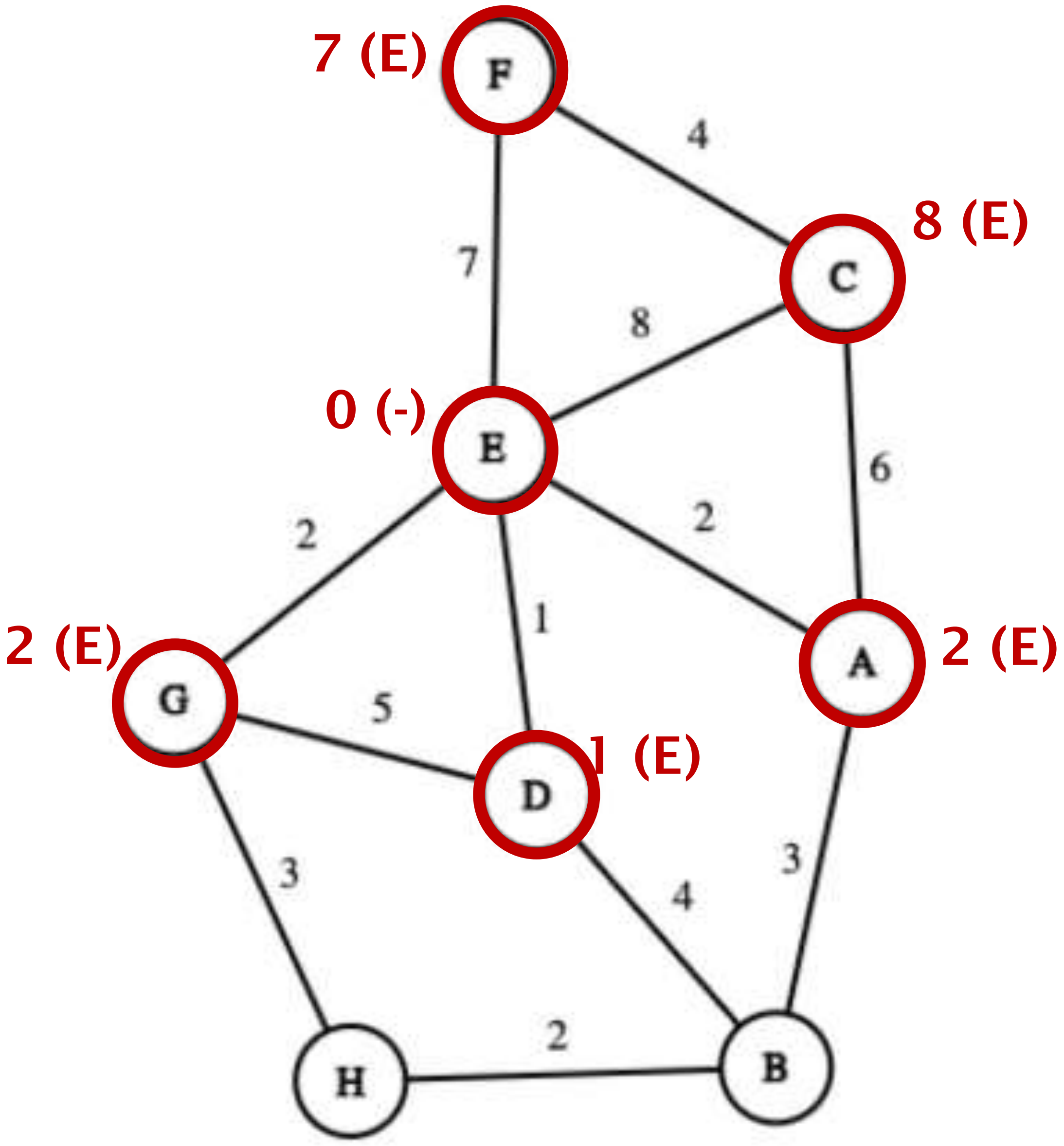
C to A, cost=8+6=14

A to B, cost=2+3=5

D to G, cost=5+1=6

D to B, cost=1+4=5

G to H, cost=2+3=5



QUESTION-4

To keep track of BFS, we have a queue. On the graph, next to the vertices, the path and the distance through that vertex is shown in red.

Both C and A are known. Cost: 14 > 2; therefore, path to A is NOT updated. Skip.

QUEUE:

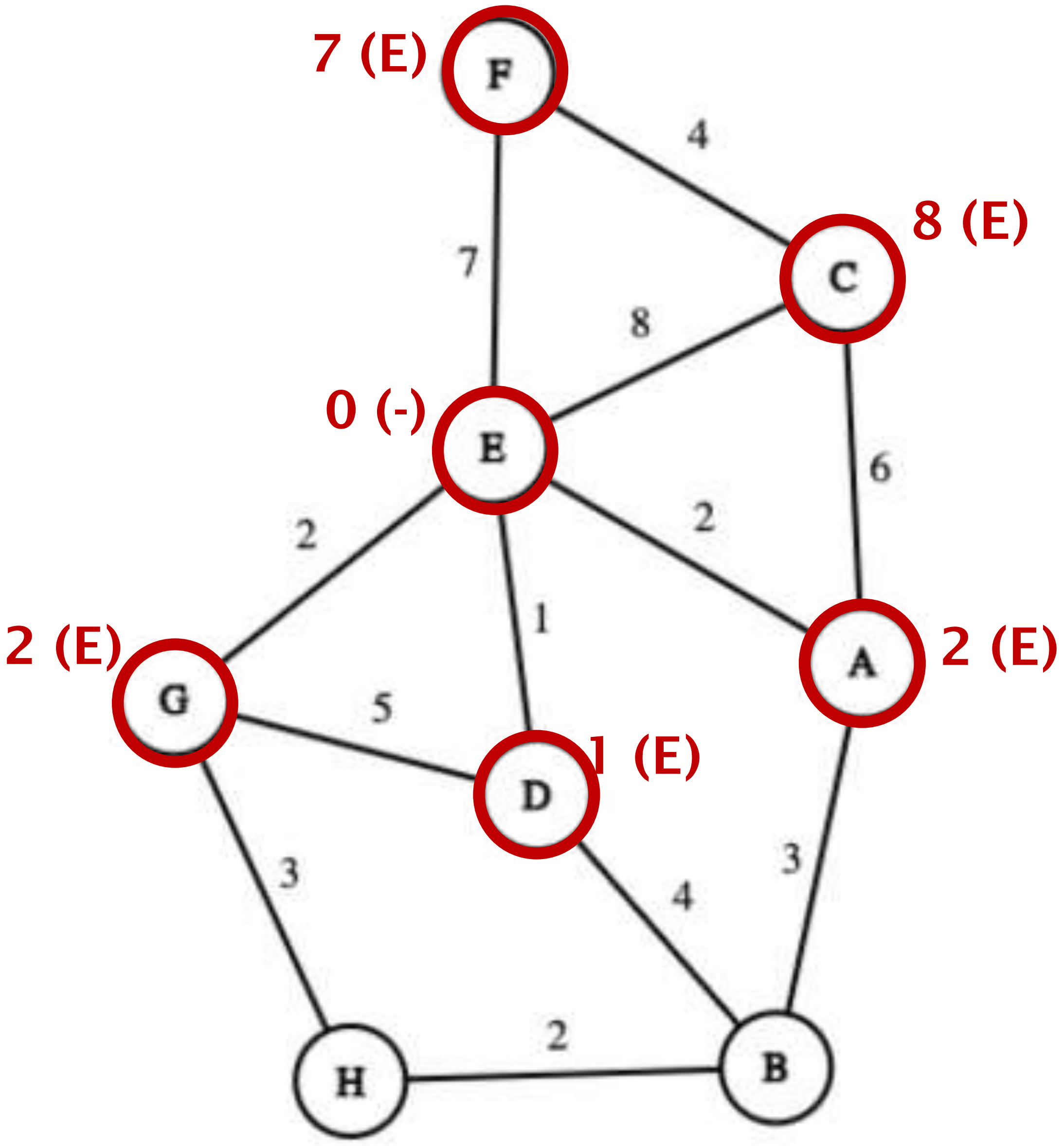
C to A, cost=8+6=14

A to B, cost=2+3=5

D to G, cost=5+1=6

D to B, cost=1+4=5

G to H, cost=2+3=5



QUESTION-4

To keep track of BFS, we have a queue. On the graph, next to the vertices, the path and the distance through that vertex is shown in red.

Go to B. B is known. Its cost and path are updated. Enqueue the unknown vertices 1 away from B. So, H is enqueued.

QUEUE:

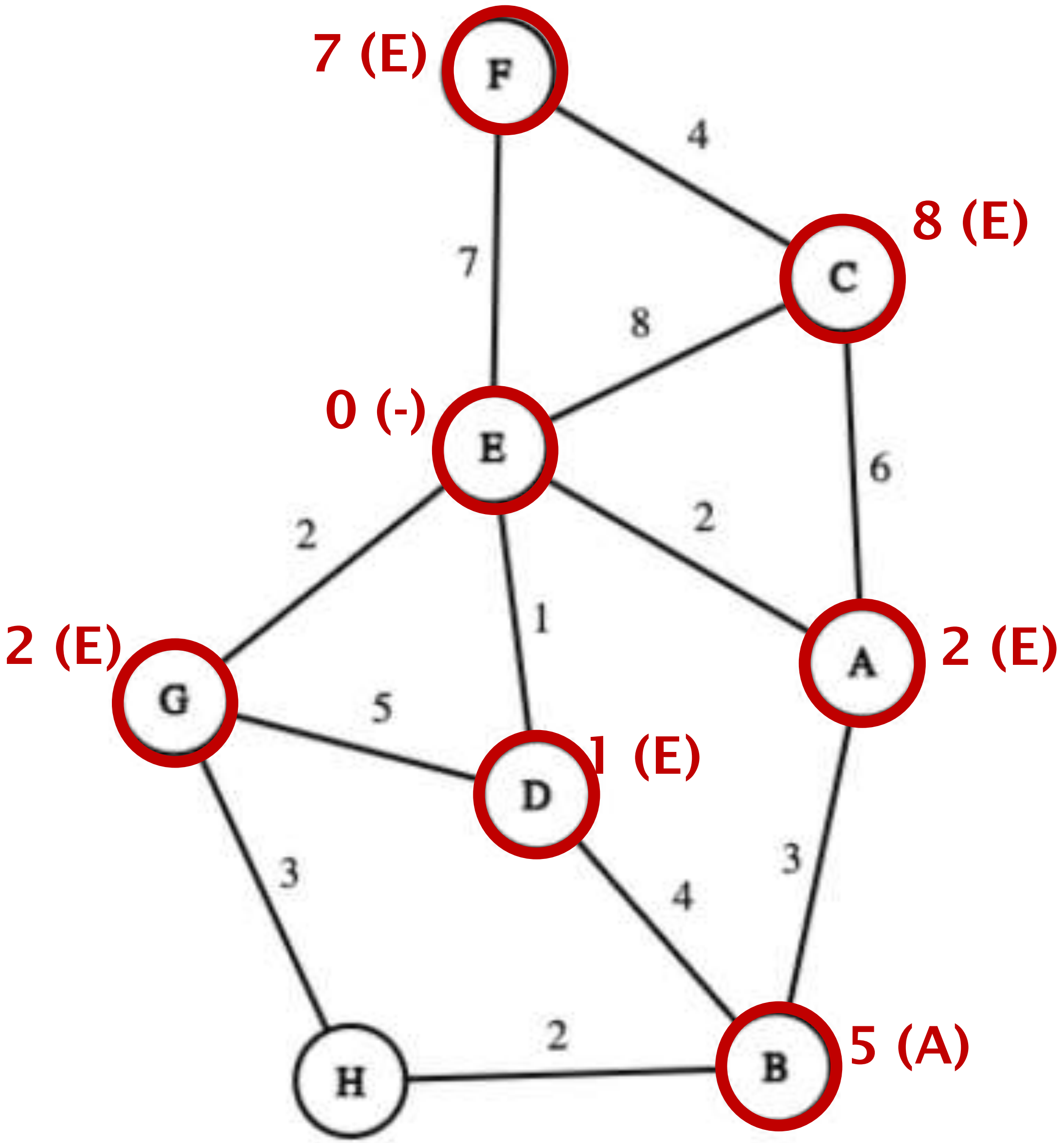
A to B, cost=2+3=5

D to G, cost=5+1=6

D to B, cost=1+4=5

G to H, cost=2+3=5

B to H, cost=5+2=7



QUESTION-4

To keep track of BFS, we have a queue. On the graph, next to the vertices, the path and the distance through that vertex is shown in red.

Both D and G are known. Cost: 6 > 2; therefore, path to G is NOT updated. Skip.

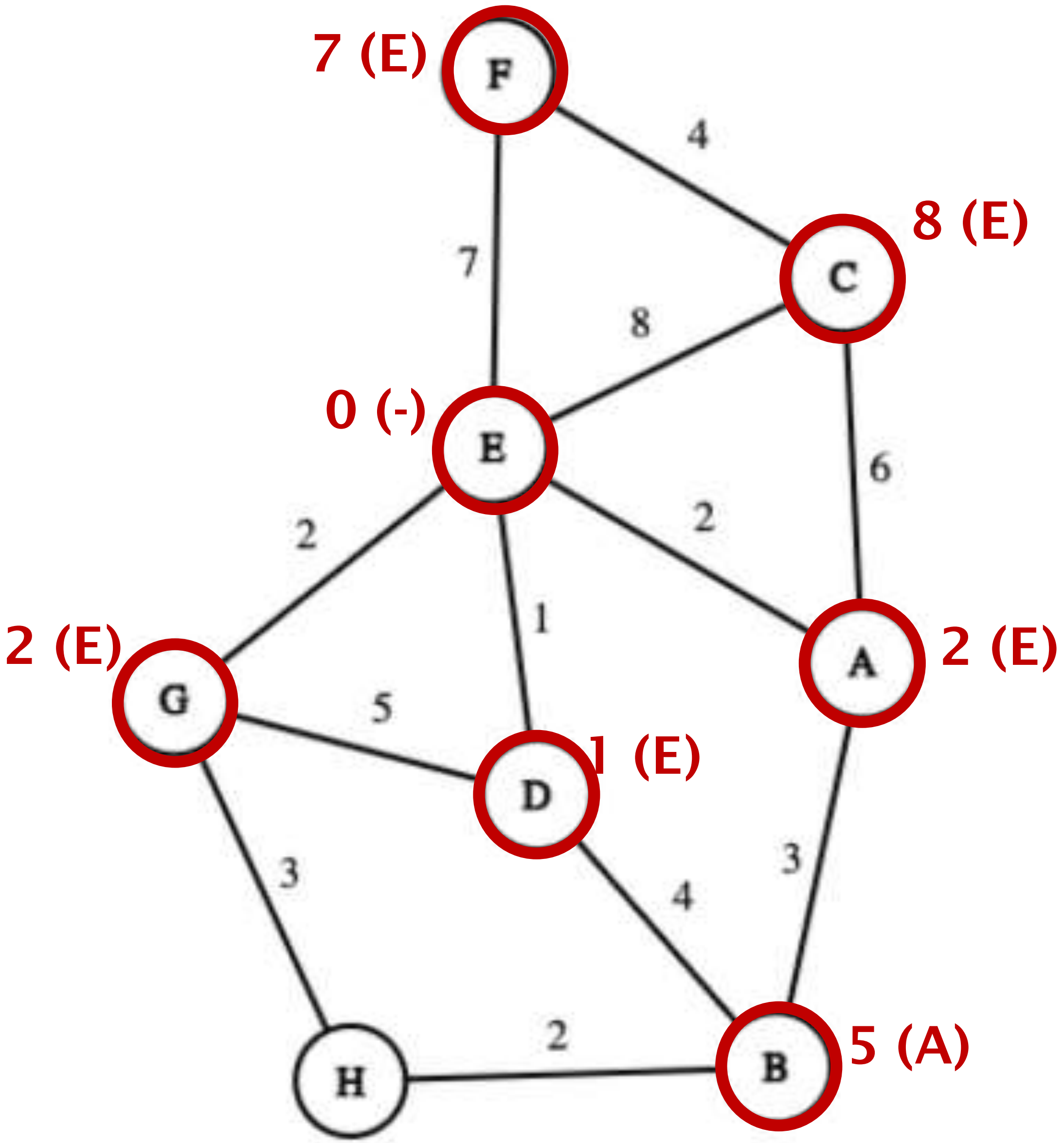
QUEUE:

D to G, cost=5+1=6

D to B, cost=1+4=5

G to H, cost=2+3=5

B to H, cost=5+2=7



QUESTION-4

To keep track of BFS, we have a queue. On the graph, next to the vertices, the path and the distance through that vertex is shown in red.

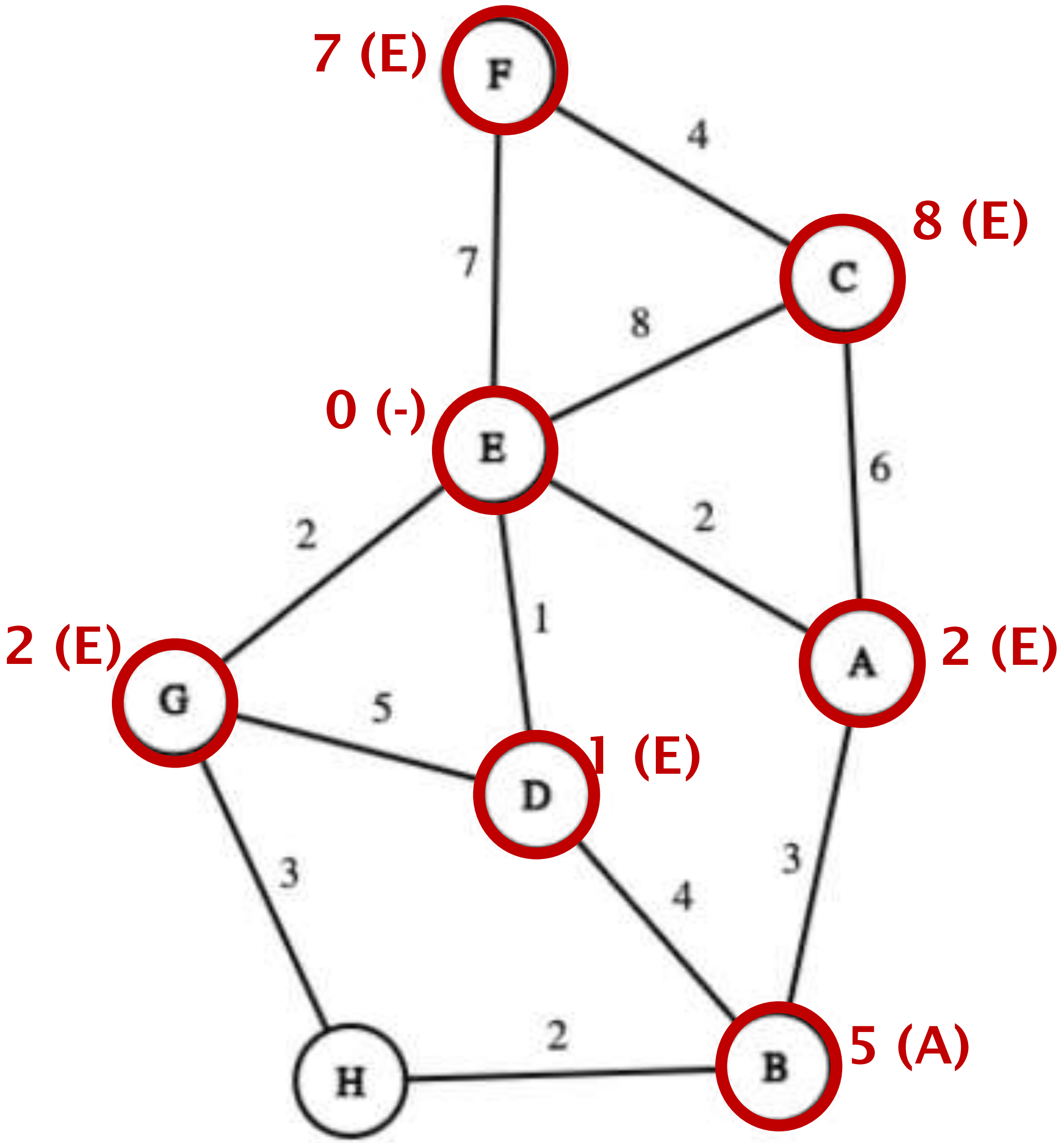
Both D and B are known. Cost: 5 = 5; therefore, path to B is NOT updated. Skip.

QUEUE:

D to B, cost=1+4=5

G to H, cost=2+3=5

B to H, cost=5+2=7



QUESTION-4

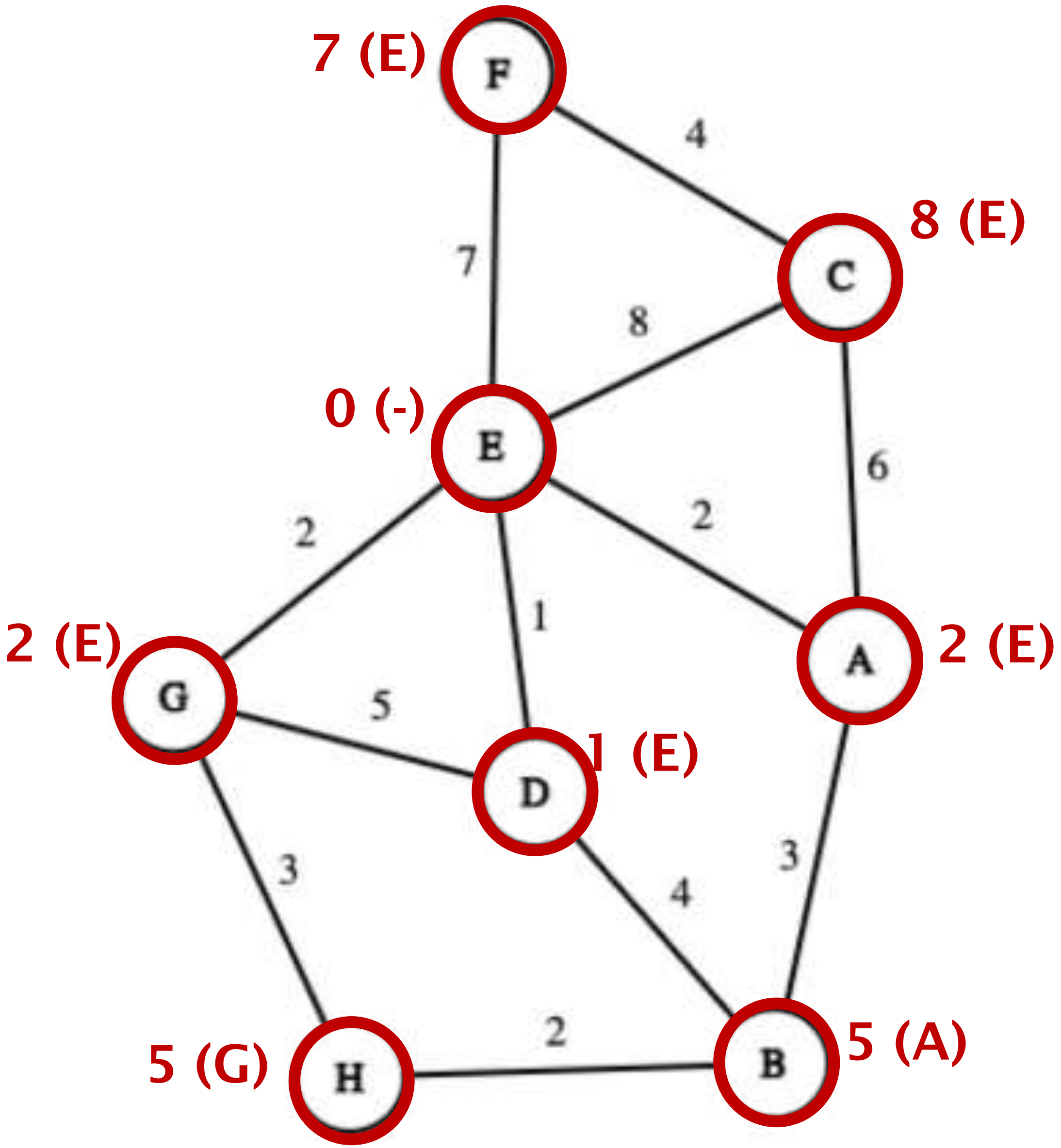
To keep track of BFS, we have a queue. On the graph, next to the vertices, the path and the distance through that vertex is shown in red.

Go to H. H is known. Its cost and path are updated. No unknown vertices from H. Skip.

QUEUE:

G to H, cost=2+3=5

B to H, cost=5+2=7



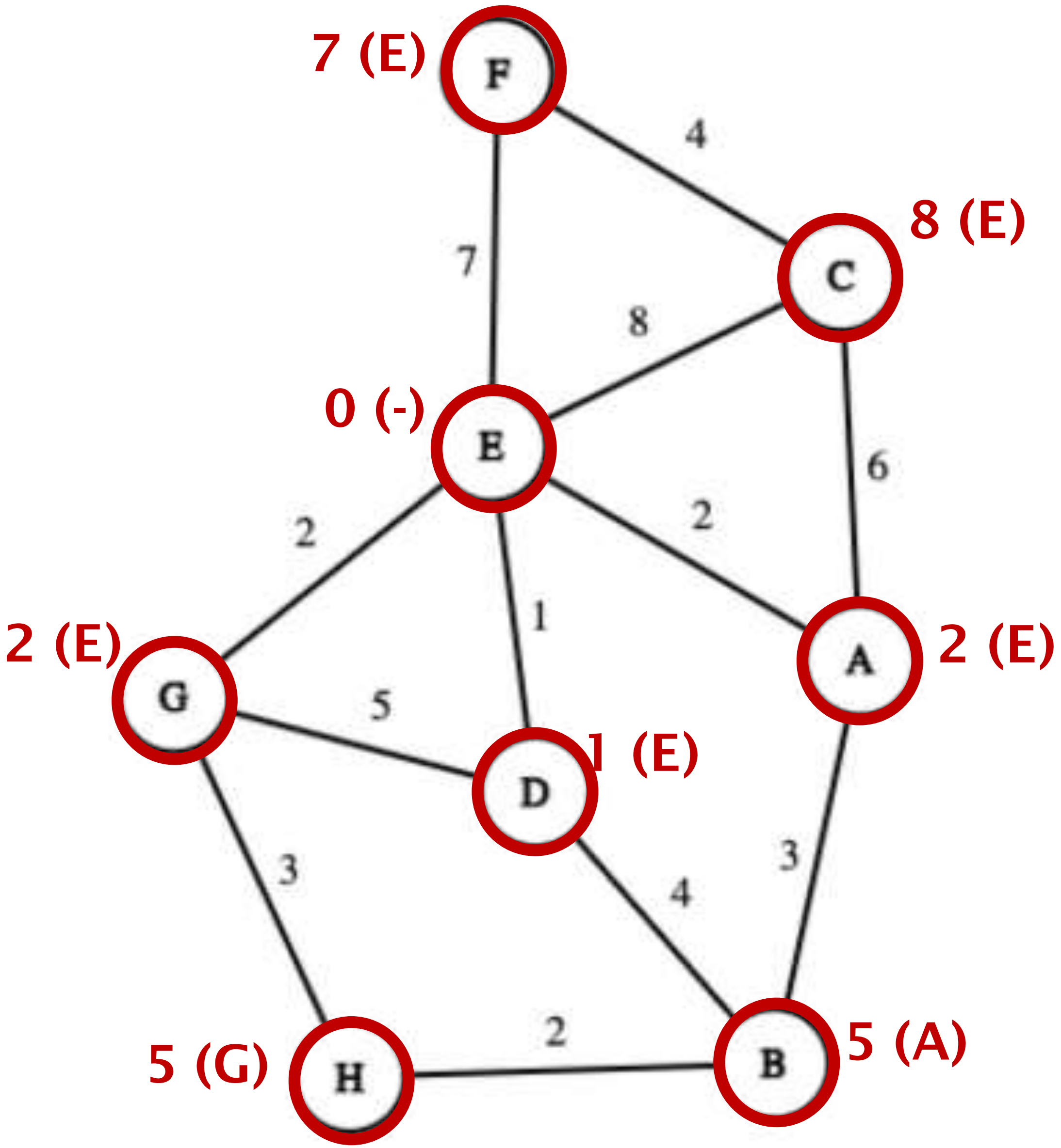
QUESTION-4

To keep track of BFS, we have a queue. On the graph, next to the vertices, the path and the distance through that vertex is shown in red.

Both B and H are known. Cost: 7 > 5; therefore, path to G is NOT updated. Skip.

QUEUE:

B to H, cost=5+2=7



QUESTION-4

To keep track of BFS, we have a queue. On the graph, next to the vertices, the path and the distance through that vertex is shown in red.

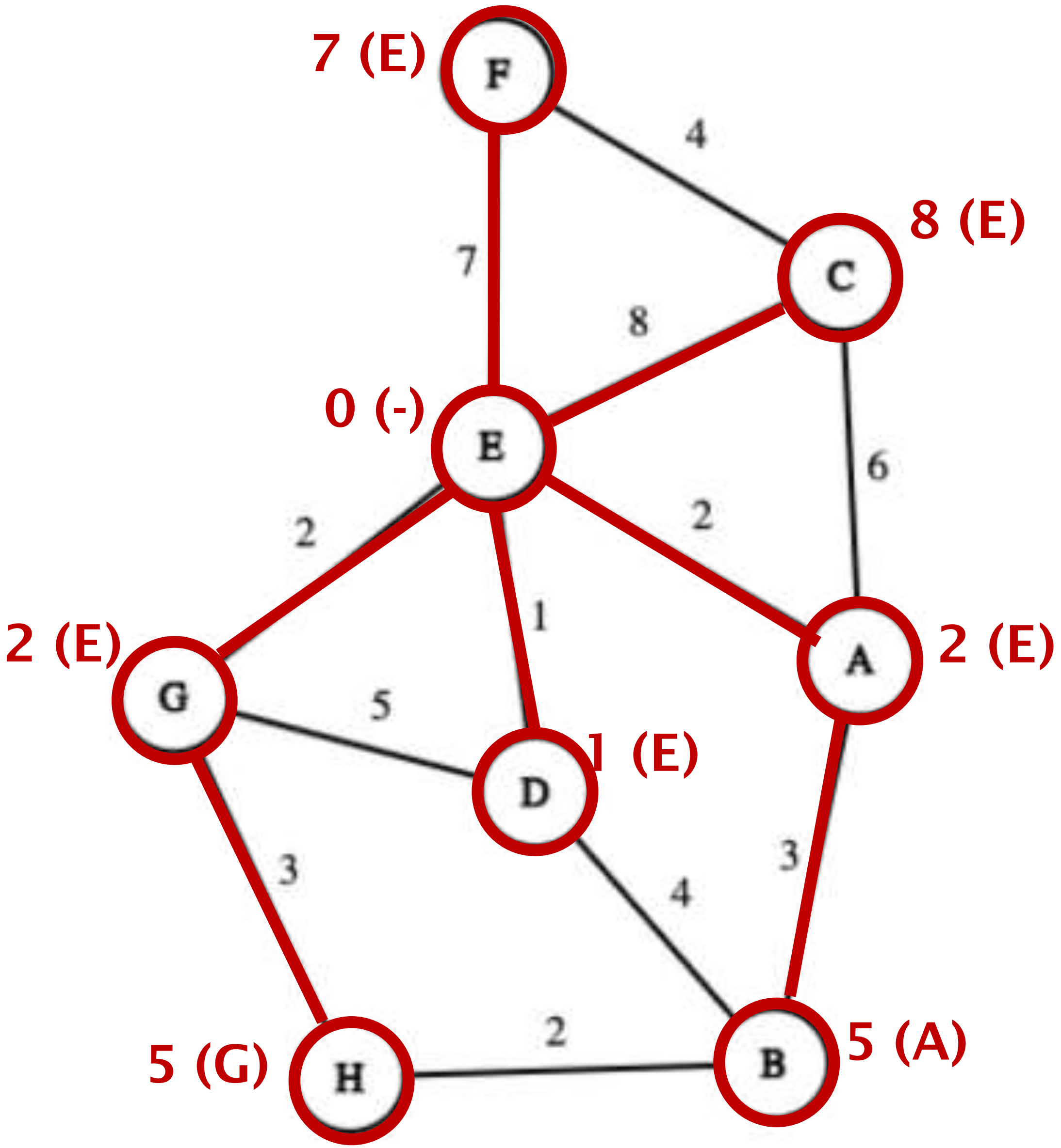
Both B and H are known. Cost: 7 > 5; therefore, path to G is NOT updated. Skip.

QUEUE:

empty

Queue is again empty, which means no further search left. BFS tree is shown on the graph.

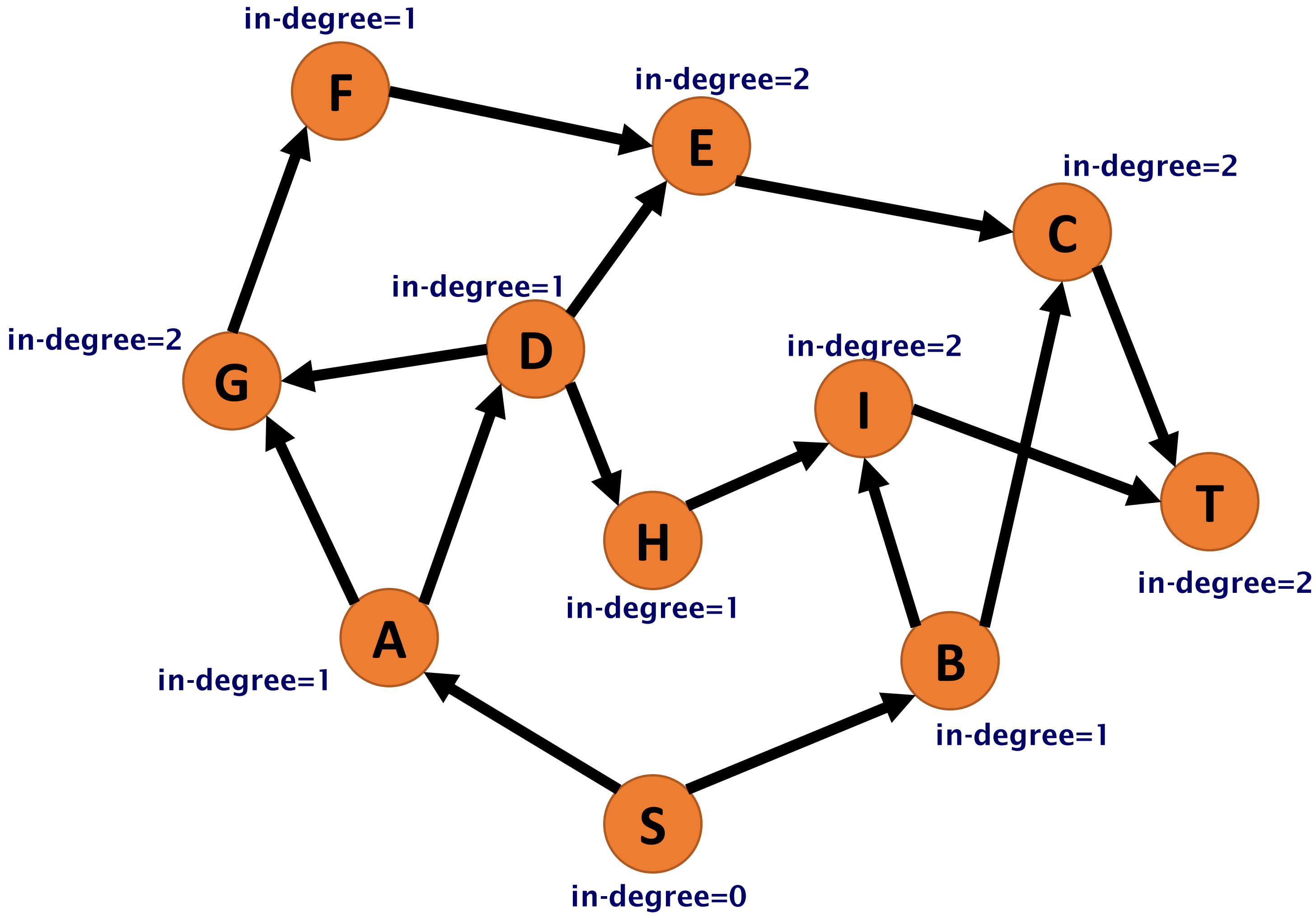
Terminate.



QUESTION-5

First of all, determine in-degree values for all vertices.

Values are shown on the graph. Start with a vertex having in-degree = 0.

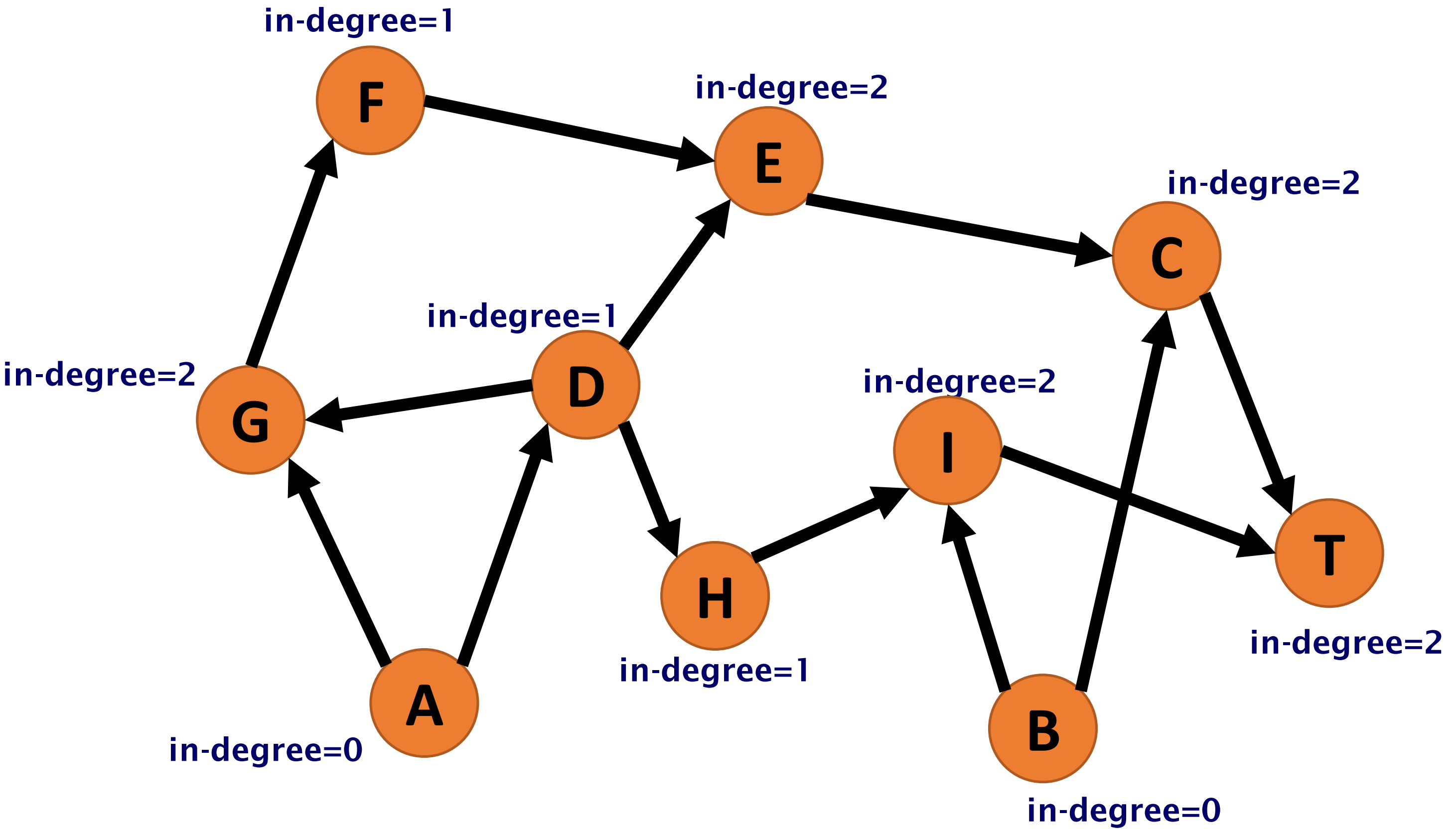


QUESTION-5

The only vertex having in-degree=0 is vertes S. So, start with S.

- Select S.
- Print S.
- Remove S.
- Update in-degree numbers of neighbouring vertices.
- Move into next vertex with in-degree=0

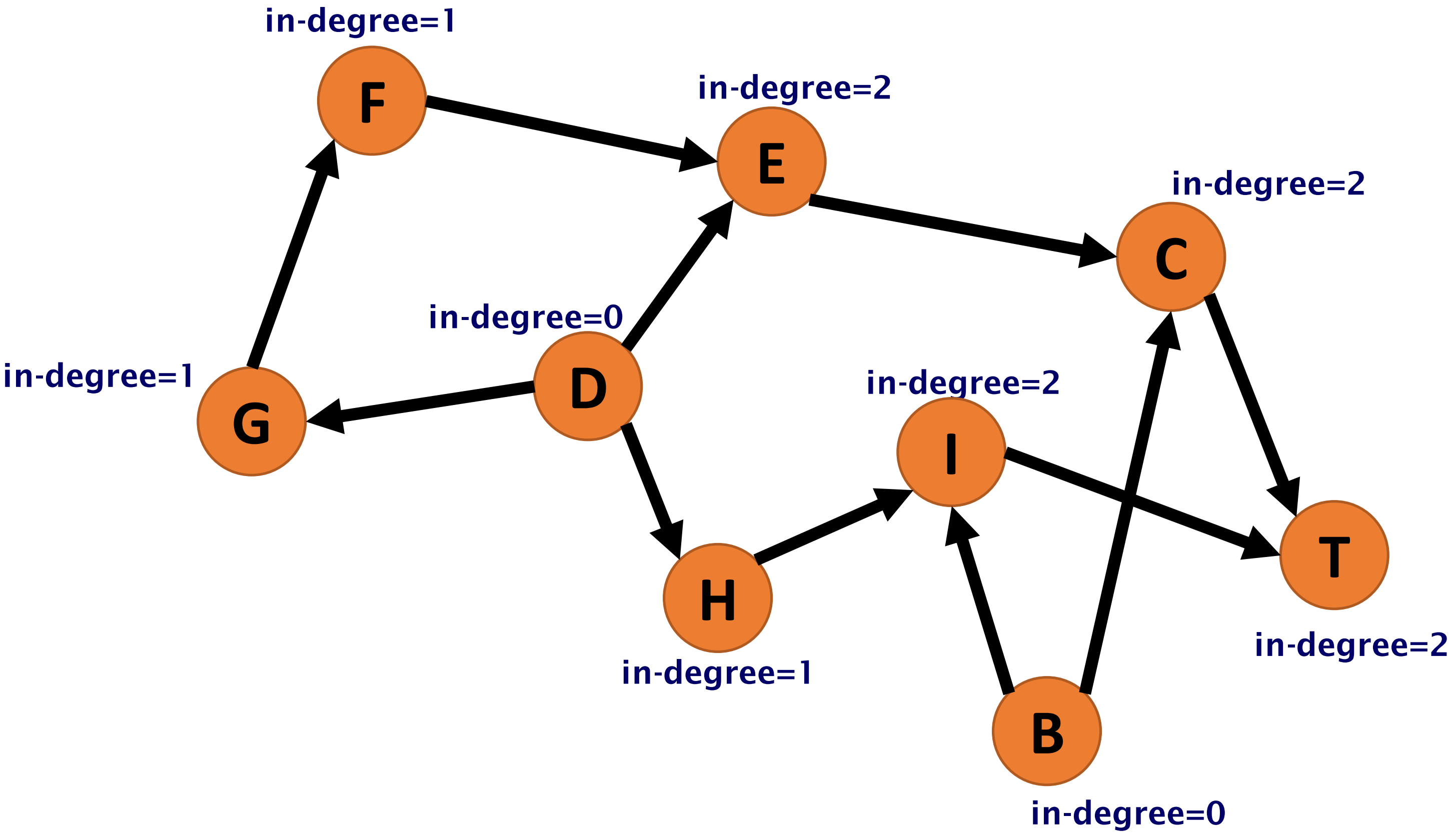
Output:



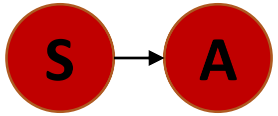
QUESTION-5

Vertices A and B have in-degree=0. Pick one randomly.

- Select A.
- Print A.
- Remove A.
- Update in-degree numbers of neighbouring vertices.
- Move into next vertex with in-degree=0



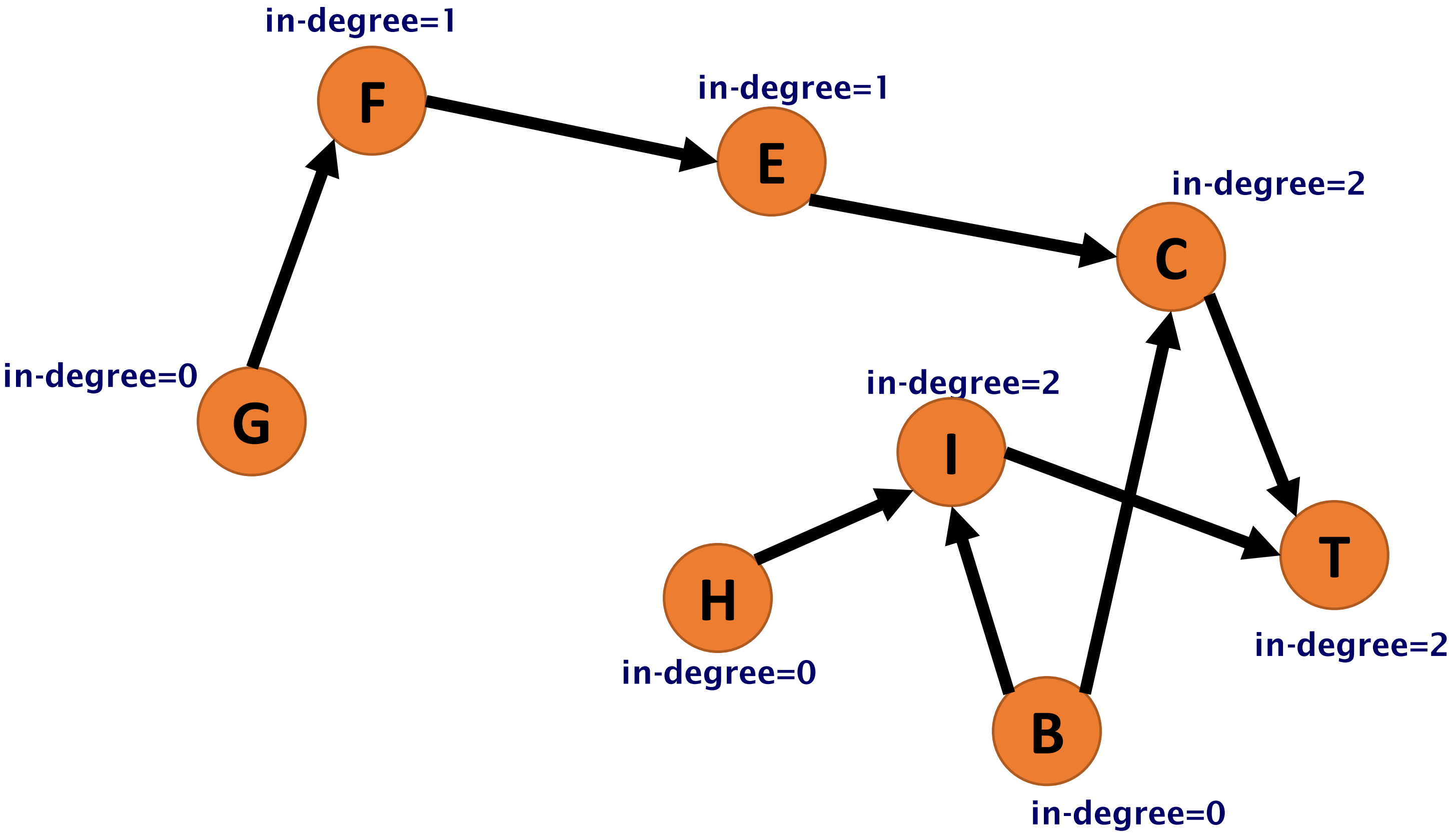
Output:



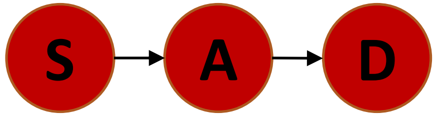
QUESTION-5

Vertices D and B have in-degree=0. Pick one randomly.

- Select D.
- Print D.
- Remove D.
- Update in-degree numbers of neighbouring vertices.
- Move into next vertex with in-degree=0



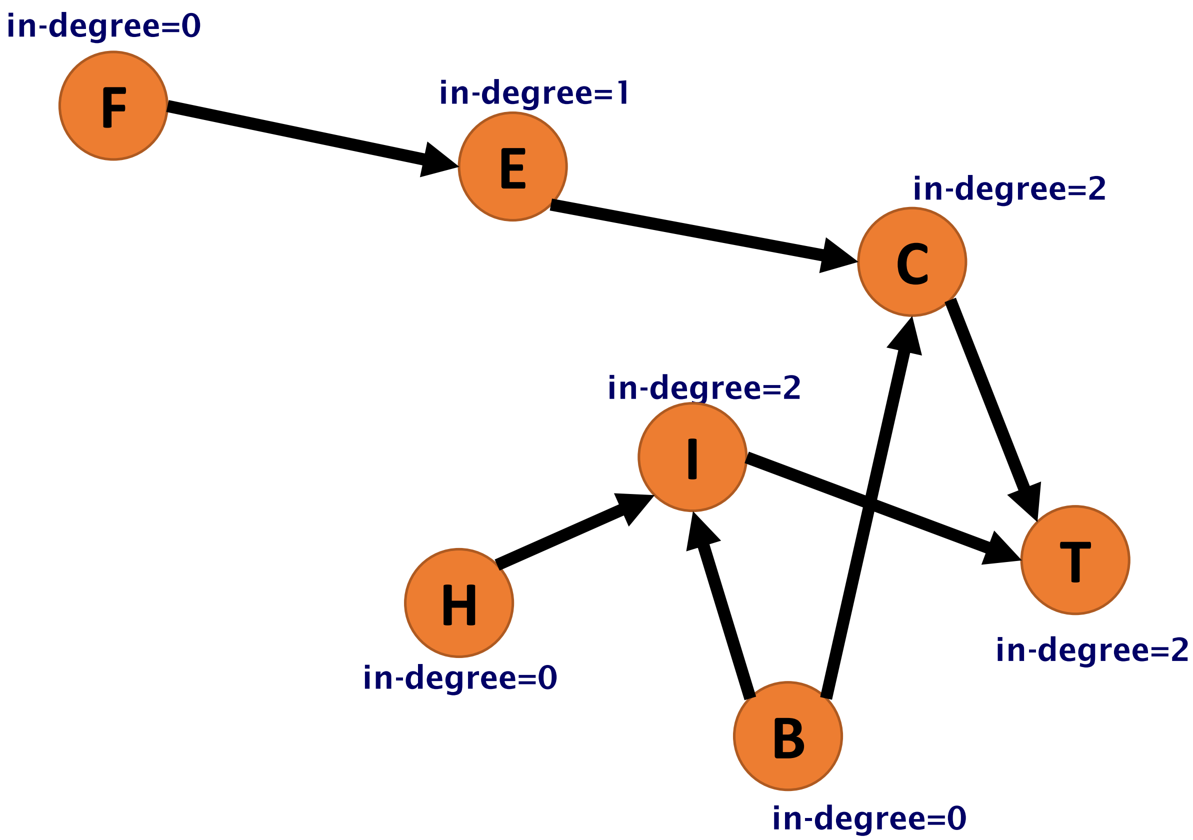
Output:



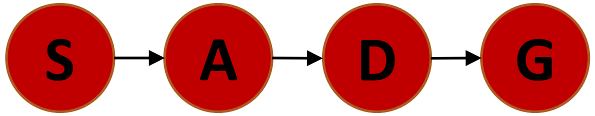
QUESTION-5

Vertices G, H and B have in-degree=0. Pick one randomly.

- Select G.
- Print G.
- Remove G.
- Update in-degree numbers of neighbouring vertices.
- Move into next vertex with in-degree=0



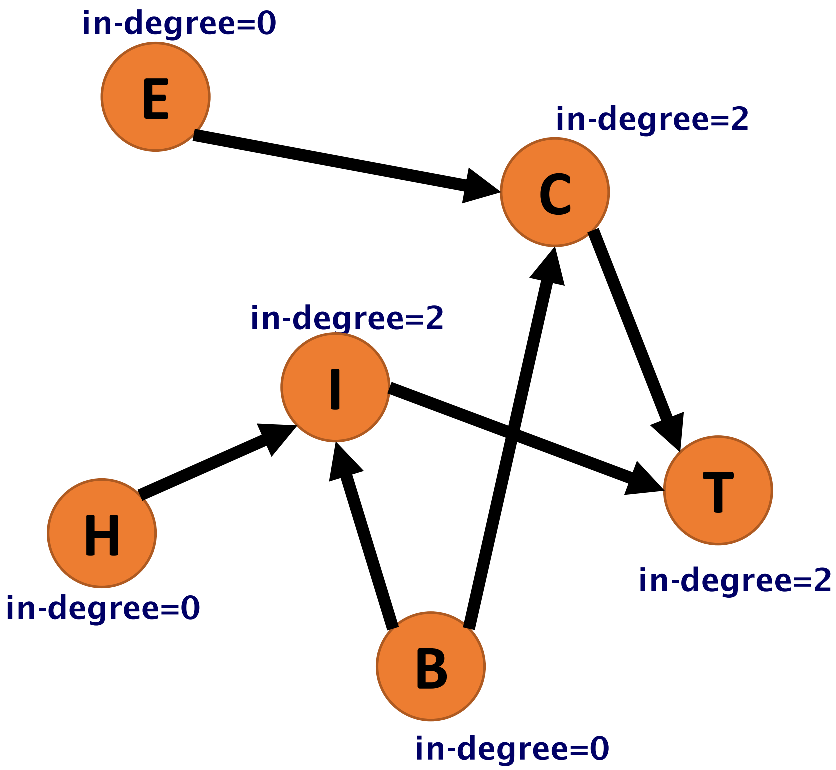
Output:



QUESTION-5

Vertices F, H and B have in-degree=0.
Pick one randomly.

- Select F.
- Print F.
- Remove F.
- Update in-degree numbers of neighbouring vertices.
- Move into next vertex with in-degree=0



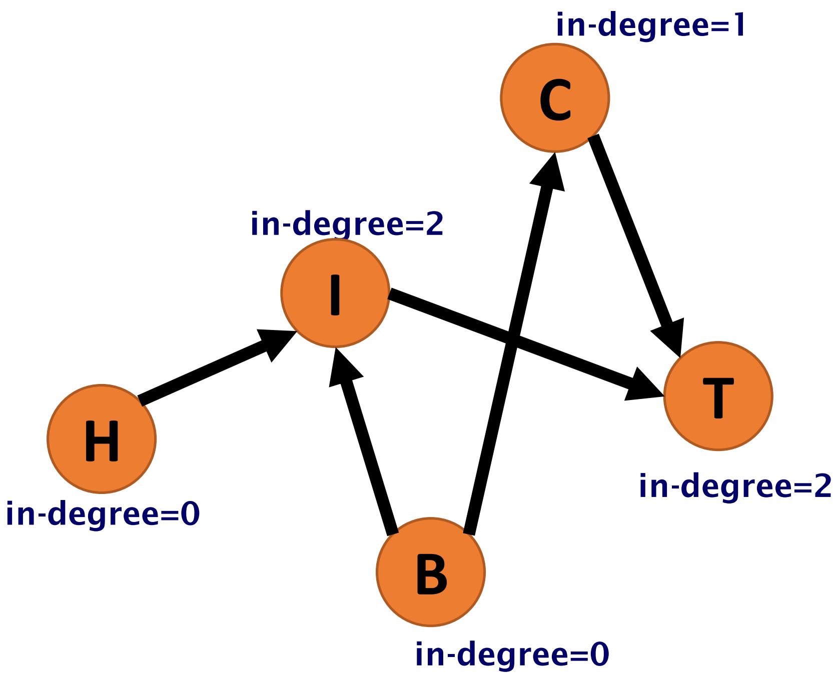
Output:

S → A → D → G → F

QUESTION-5

Vertices E, H and B have in-degree=0.
Pick one randomly.

- Select E.
- Print E.
- Remove E.
- Update in-degree numbers of neighbouring vertices.
- Move into next vertex with in-degree=0



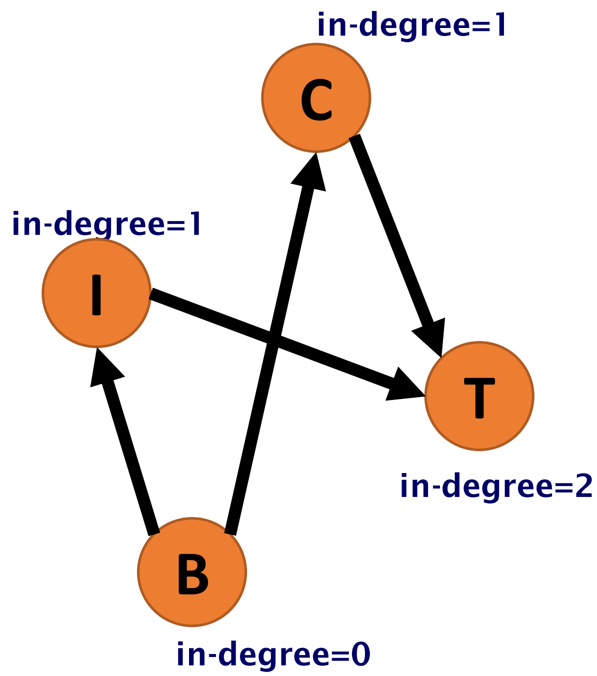
Output:

S → A → D → G → F → E

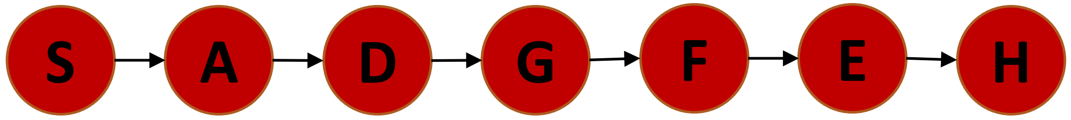
QUESTION-5

Vertices H and B have in-degree=0. Pick one randomly.

- Select H.
- Print H.
- Remove H.
- Update in-degree numbers of neighbouring vertices.
- Move into next vertex with in-degree=0



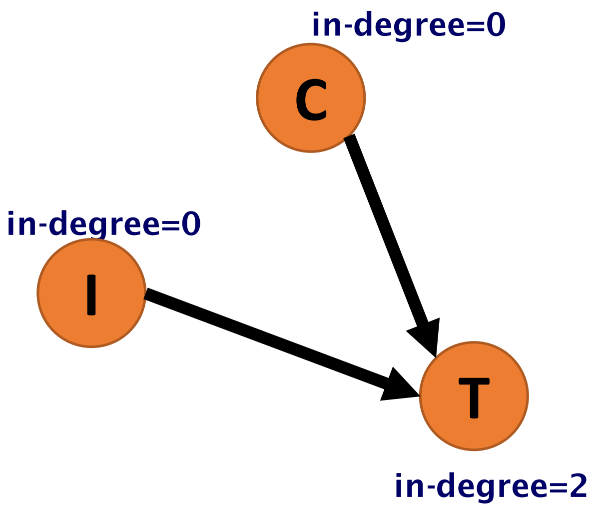
Output:



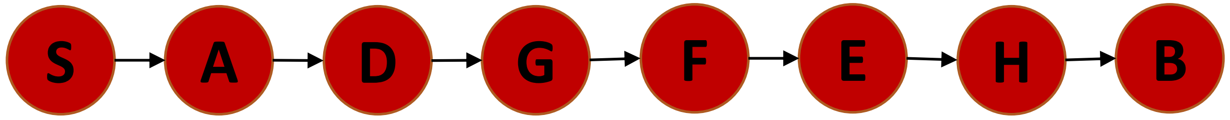
QUESTION-5

Vertex B have in-degree=0. Pick one randomly.

- Select B.
- Print B.
- Remove B.
- Update in-degree numbers of neighbouring vertices.
- Move into next vertex with in-degree=0



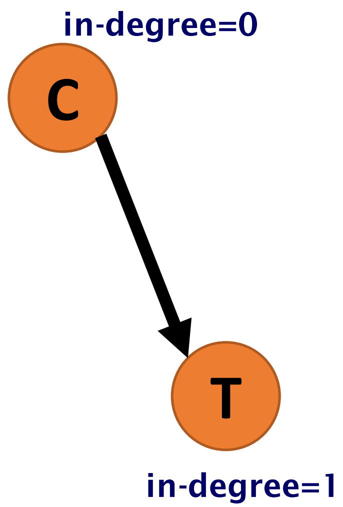
Output:



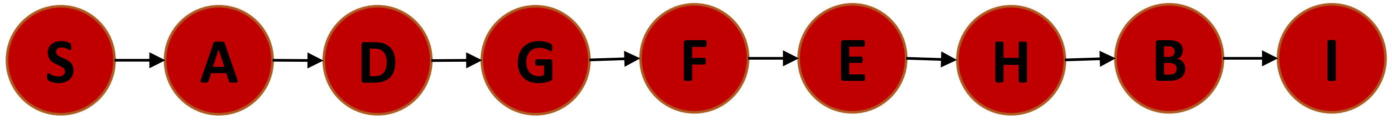
QUESTION-5

Vertices I and C have in-degree=0. Pick one randomly.

- Select I.
- Print I.
- Remove I.
- Update in-degree numbers of neighbouring vertices.
- Move into next vertex with in-degree=0



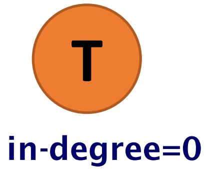
Output:



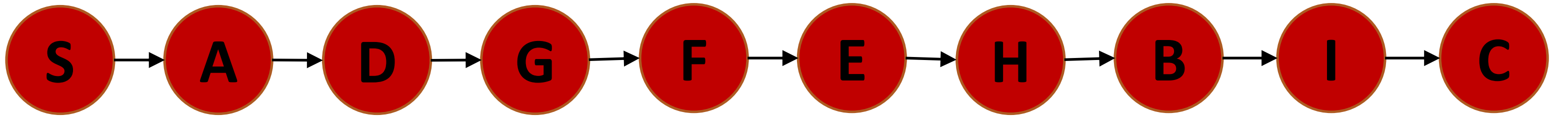
QUESTION-5

Vertex C have in-degree=0. Pick one randomly.

- Select C.
- Print C.
- Remove C.
- Update in-degree numbers of neighbouring vertices.
- Move into next vertex with in-degree=0



Output:



QUESTION-5

Vertex T have in-degree=0. Pick one randomly.

- Select T.
- Print T.
- Remove T.
- Update in-degree numbers of neighbouring vertices.
- Move into next vertex with in-degree=0

Output:

S → A → D → G → F → E → H → B → I → C → T

QUESTION-5

No vertices left.

Output of this topological sorting is as follows:

Output:

S → A → D → G → F → E → H → B → I → C → T

Note that this order is NOT unique. Random choices for vertices with in-degree=0 might change the order.