Graph Theory

Chapter 8

Shortest-Path Problems

Shortest Path

- Generalize distance to weighted setting
- $G = (V,E)$ with weight function $W\!: E \rightarrow R$ (assigning real values to edges)
- Weight of path $p = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ is

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

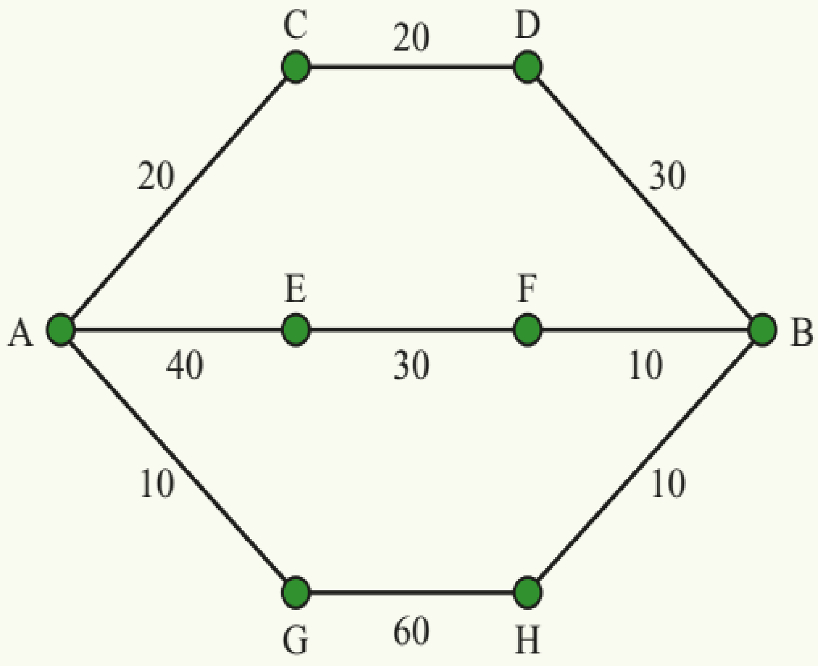
- Shortest path = a path of the minimum weight
- Applications
  - static/dynamic network routing
  - robot motion planning
  - map/route generation in traffic

Shortest-Path Problems

- Shortest-Path problems
  - **Single-source (single-destination).** Find a shortest path from a given source (vertex *s*) to each of the vertices. The topic of this lecture.
  - **Single-pair.** Given two vertices, find a shortest path between them. Solution to single-source problem solves this problem efficiently, too.
  - **All-pairs.** Find shortest-paths for every pair of vertices. Dynamic programming algorithm.
  - Unweighted shortest-paths – BFS.

Finding the shortest path by inspection

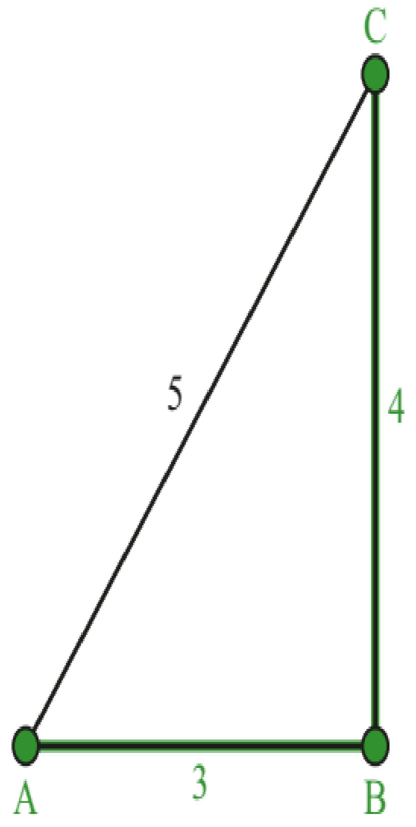
The network below shows cities connected by highways. The length of each highway is indicated by an edge weight. What is the shortest path between cities A and B?



Solve/Think	Apply
There are three paths from A to B. 1 Path ACDB has a length of 20 + 20 + 30 = 70. 2 Path AEFB has a length of 40 + 30 + 10 = 80. 3 Path AGHB has a length of 10 + 60 + 10 = 80. The path with the shortest length is ACDB.	One way to find the shortest path is to calculate the length of every path, and then select the path with the shortest length. This method only works in practice when there is a small number of possible paths to consider.

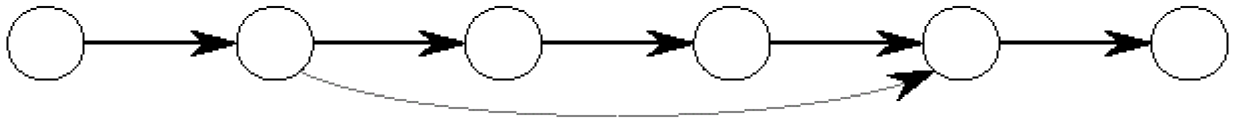
Minimum spanning trees

The minimum spanning tree does not always provide the shortest path between two points. For example, in the network on the right the minimum spanning tree contains the path ABC (highlighted in green), whereas the shortest path between vertices A and C is AC.



Optimal Substructure

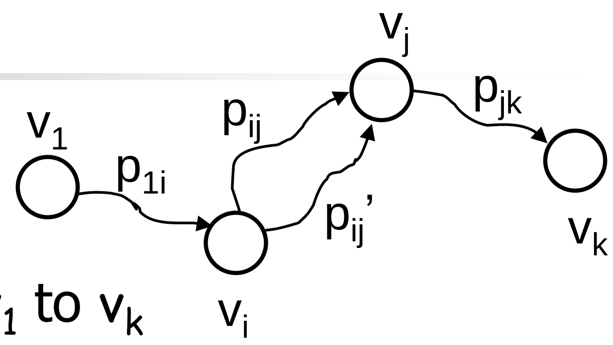
- *Theorem*: subpaths of shortest paths are shortest paths
- Proof ("cut and paste")
  - if some subpath were not the shortest path, one could substitute the shorter subpath and create a shorter total path



Optimal Substructure Theorem

Given:

- A weighted graph $G = (V, E)$
- A weight function $w: E \to \mathbf{R}$,
- A shortest path $p = \langle v_1, v_2, \ldots, v_k \rangle$ from v_1 to v_k
- A subpath of p: $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$, with $1 \le i \le j \le k$



Then: p_{ij} is a shortest path from v_i to v_j

Proof: $p = v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$

$$w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$$

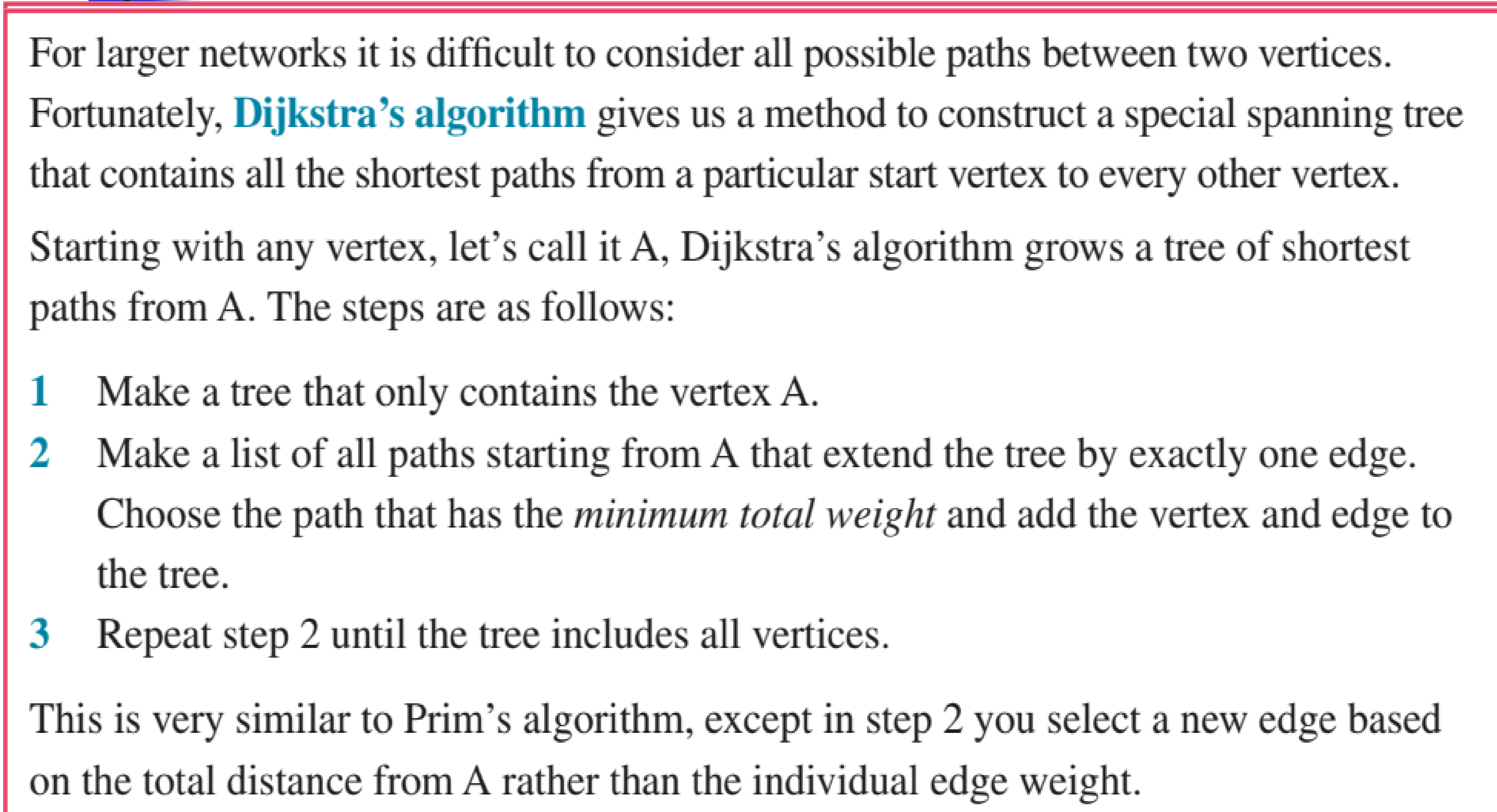
Assume $\exists$ p_{ij}' from v_i to v_j with $w(p_{ij}') < w(p_{ij})$

$\Rightarrow w(p') = w(p_{1i}) + w(p_{ij}') + w(p_{jk}) < w(p)$ contradiction!

Dijkstra's algorithm only works for positive or zero edge weights

Dijkstra's ALgorithm

Solution to **Single-source (single-destination).**



For larger networks it is difficult to consider all possible paths between two vertices. Fortunately, **Dijkstra's algorithm** gives us a method to construct a special spanning tree that contains all the shortest paths from a particular start vertex to every other vertex.

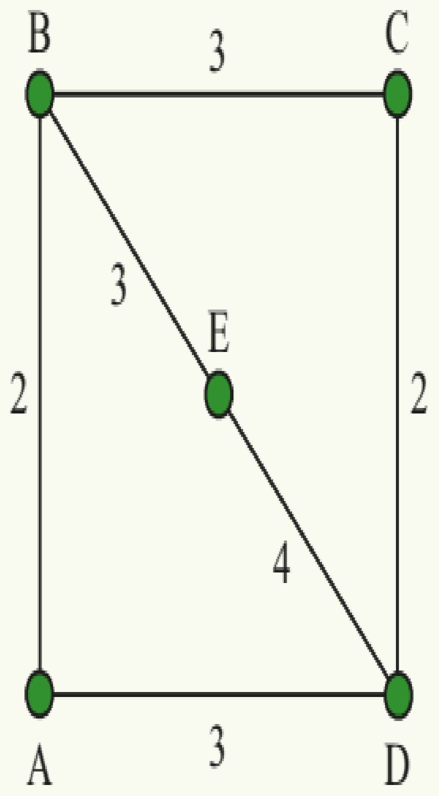
Starting with any vertex, let's call it A, Dijkstra's algorithm grows a tree of shortest paths from A. The steps are as follows:

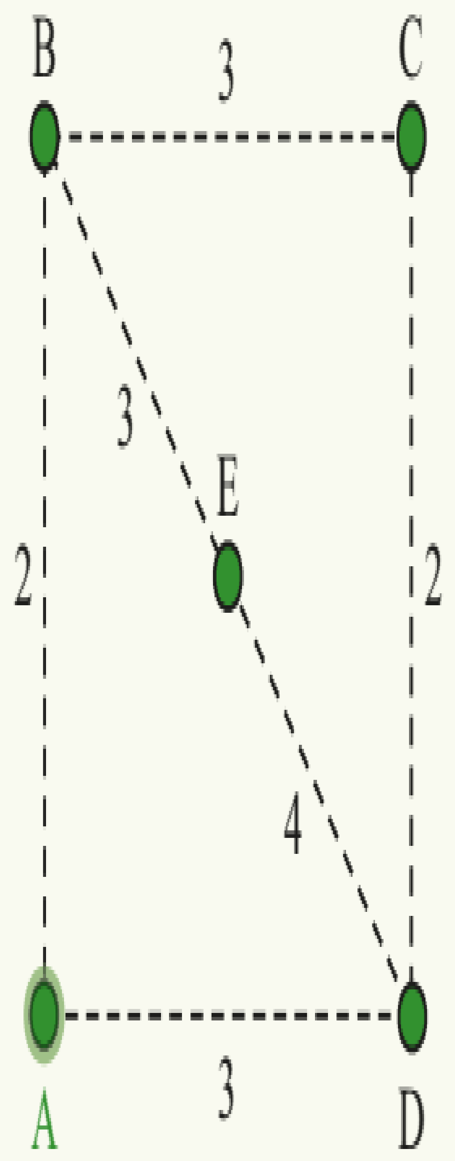
1. Make a tree that only contains the vertex A.
2. Make a list of all paths starting from A that extend the tree by exactly one edge. Choose the path that has the *minimum total weight* and add the vertex and edge to the tree.
3. Repeat step 2 until the tree includes all vertices.

This is very similar to Prim's algorithm, except in step 2 you select a new edge based on the total distance from A rather than the individual edge weight.

Using Dijkstra's algorithm

Use Dijkstra's algorithm to find the shortest path from A to E in the following network.

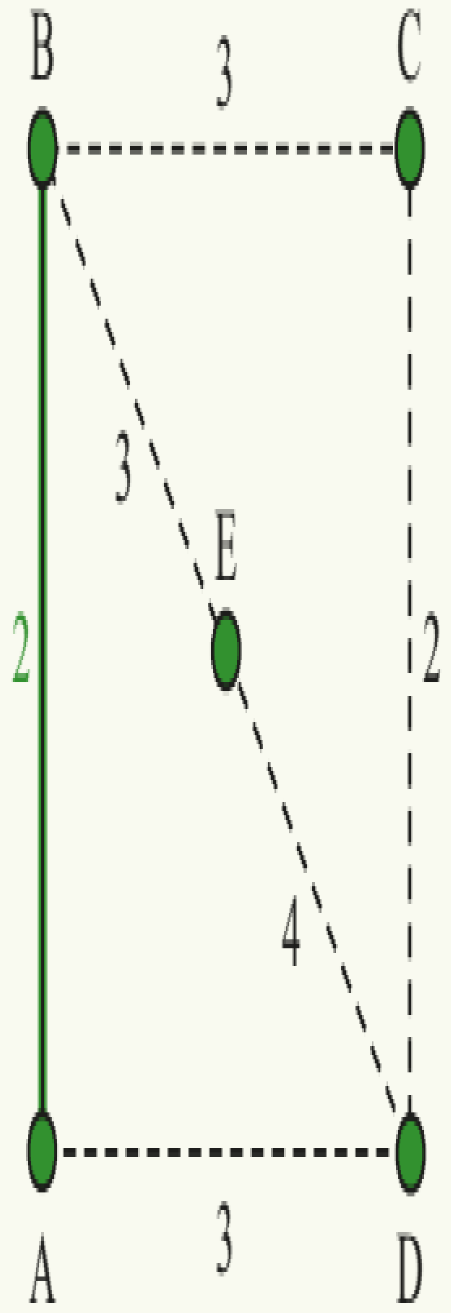


Step	Solve/Think	
1	Make a tree with only vertex A.	

2.1 The tree contains only the vertex A. The list of all paths from A with exactly one vertex outside this tree is:

Path	AB	AD
Total weight	2	3

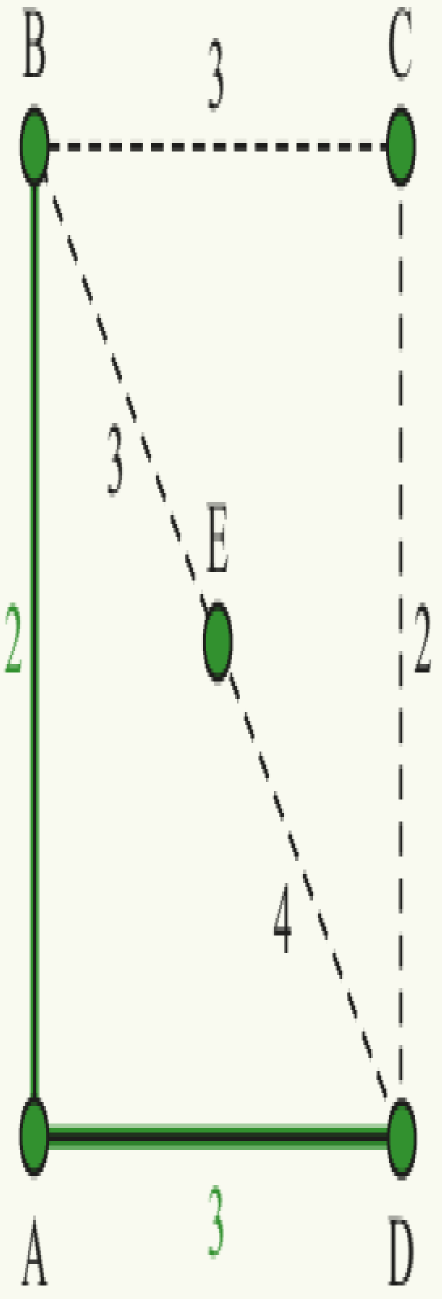
Path AB has minimum total weight, so add it to the tree.



2.2 The tree contains vertices A and B. The list of all paths starting at A with exactly one vertex outside this tree is:

Path	AD	ABE	ABC
Total weight	3	5	5

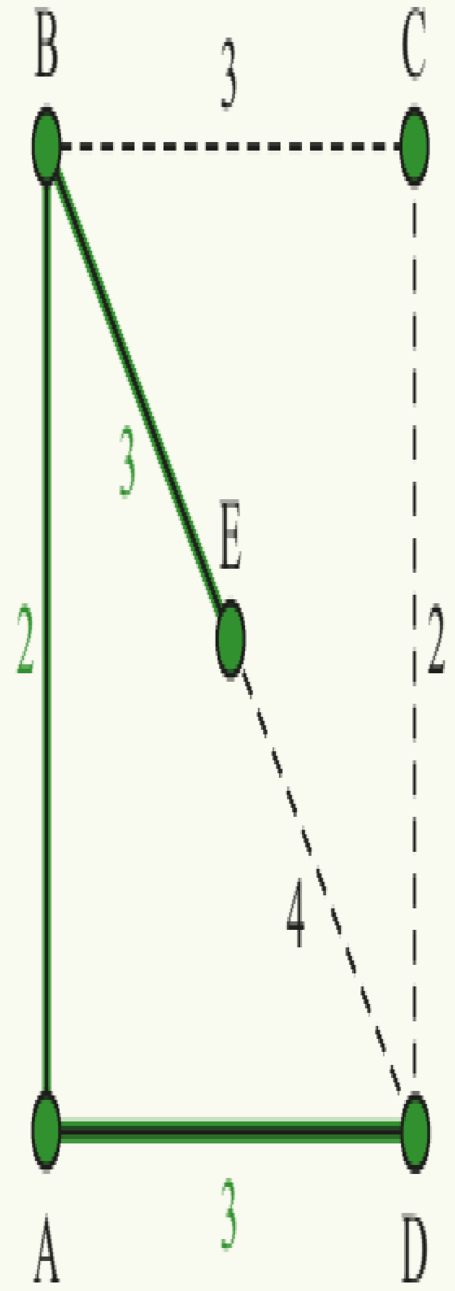
Path AD has minimum total weight, so add it to the tree.



2.3 The tree contains vertices A, B and D. The list of all paths from A with exactly one vertex outside this tree is:

Path	ABE	ABC	ADE	ADC
Total weight	5	5	7	5

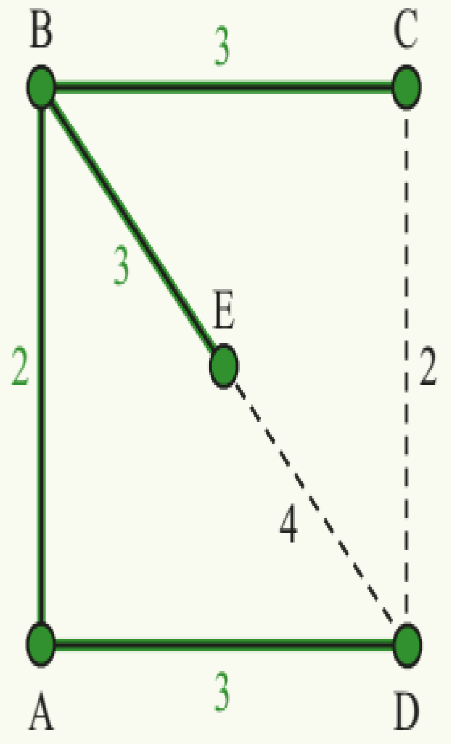
Paths ABE, ABC and ADC all have the same minimum total weight, so we are free to choose any one and add it to the tree. In this example we have chosen to add ABE.



2.4 The tree contains vertices A, B, D and E. Only vertex C remains outside the tree. The paths from A with exactly one vertex outside the tree are:

Path	ABC	ADC
Total weight	5	5

All paths have the same weight, so we are again free to choose any one and add it to the tree. In this example we have chosen to add ABC.



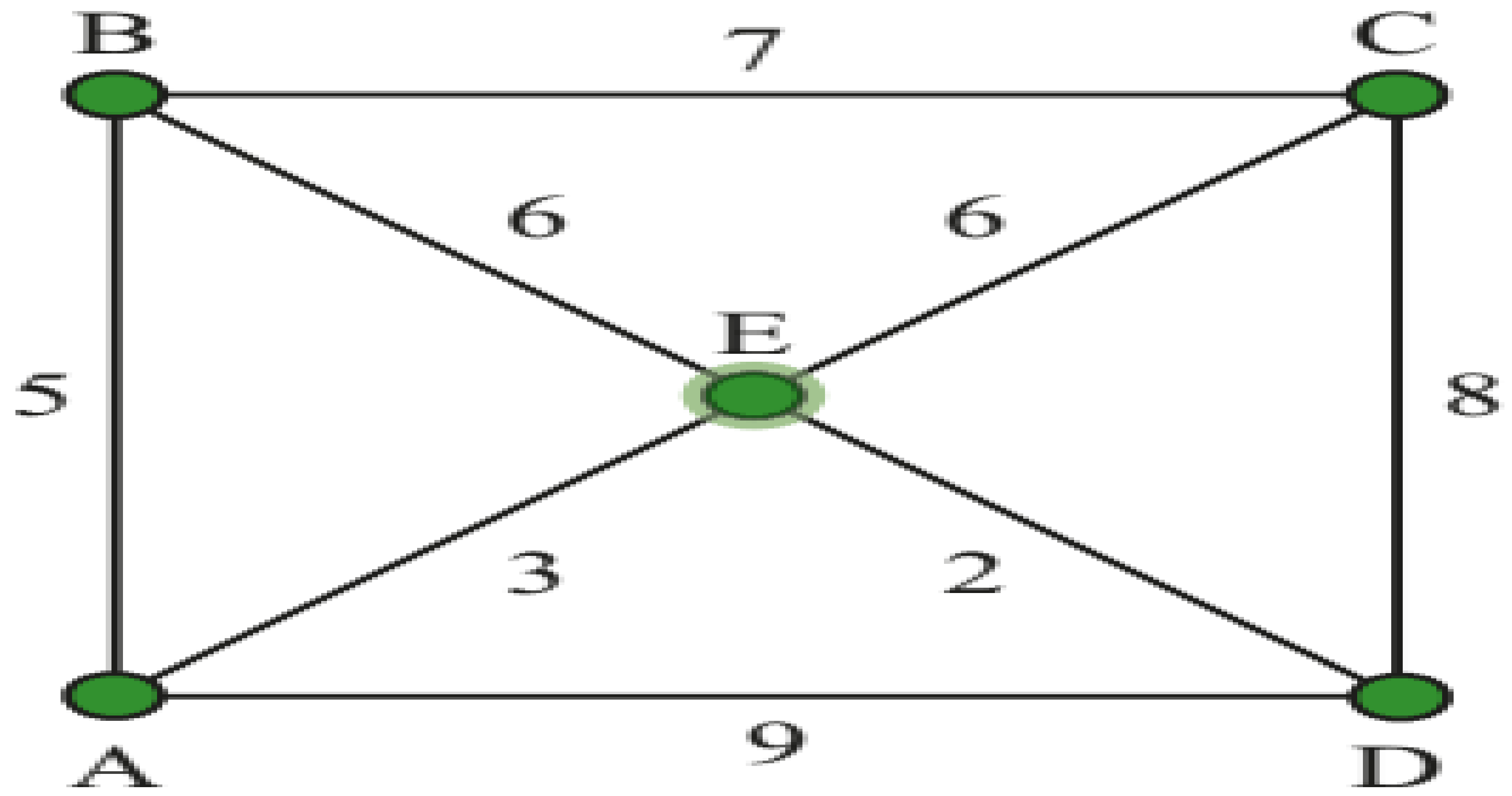
2.5 The resulting tree is a spanning tree that contains the shortest path from A to any other vertex. In particular, the shortest path from A to E in this tree is ABE, which has total weight 2 + 3 = 5.

Apply

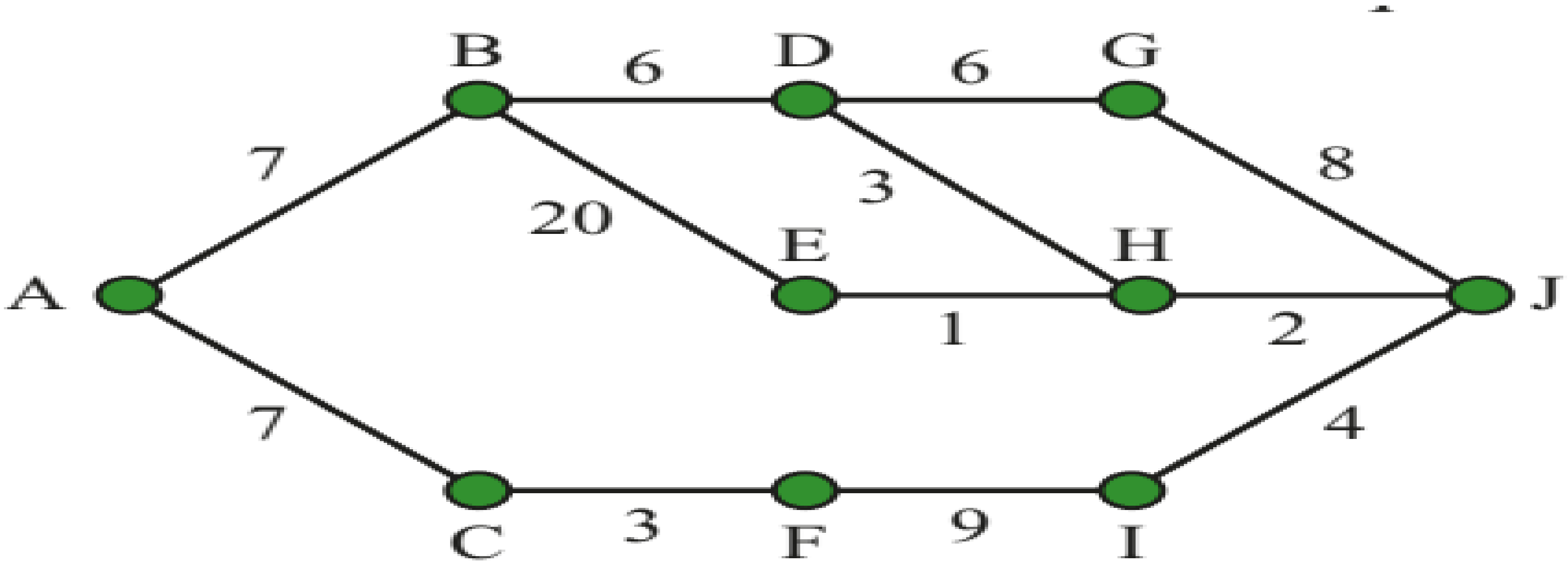
Follow the steps in Dijkstra's algorithm to create a spanning tree that contains all of the shortest paths in the network.

Note: we could have answered the question in step 2.3 when the path from A to E, ABE, was added to the tree of shortest paths.

Use Dijkstra's algorithm to find the shortest path from E to B in the network below.

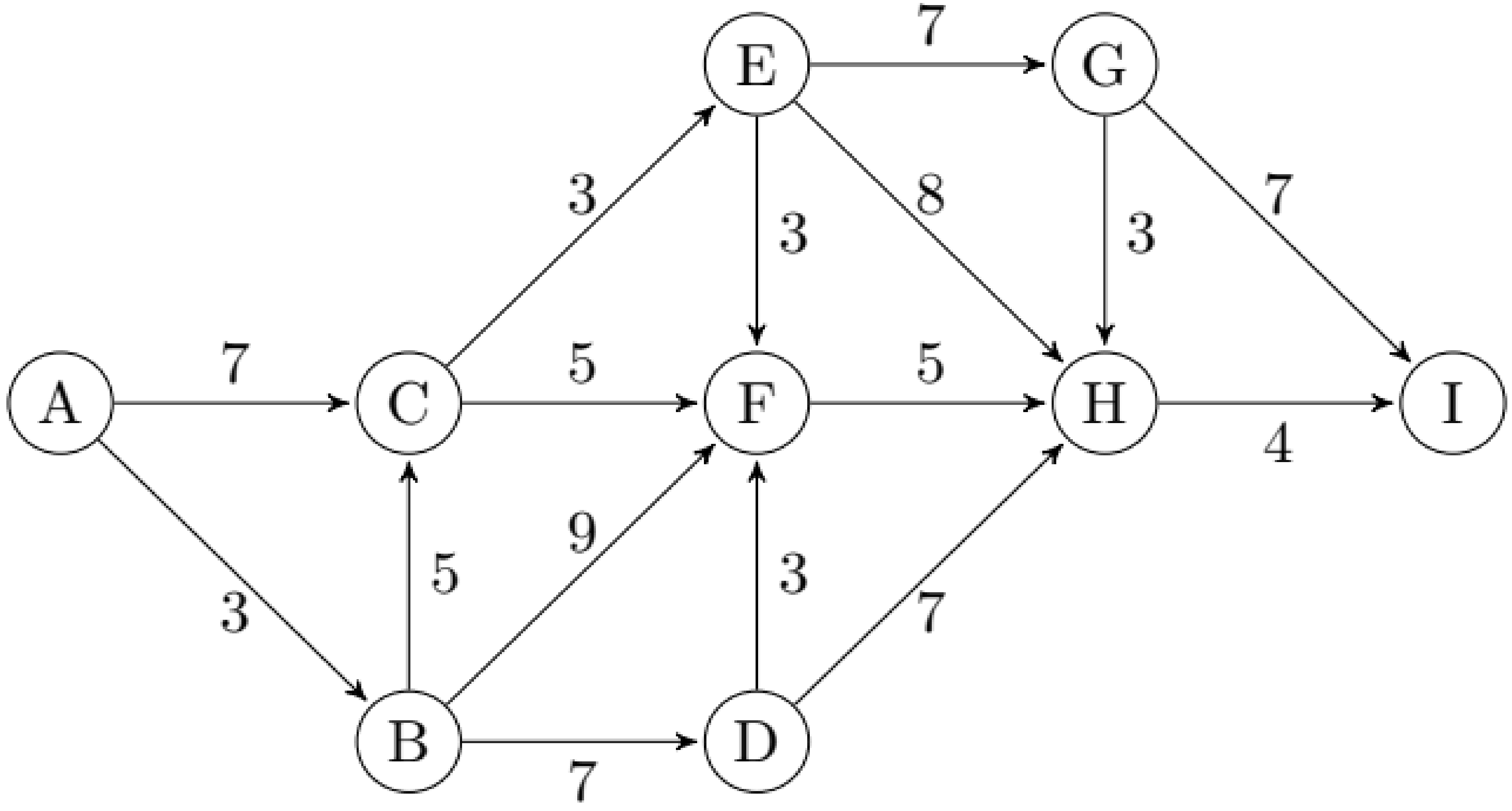


Use Dijkstra's algorithm to find the shortest path between A and J in this network.

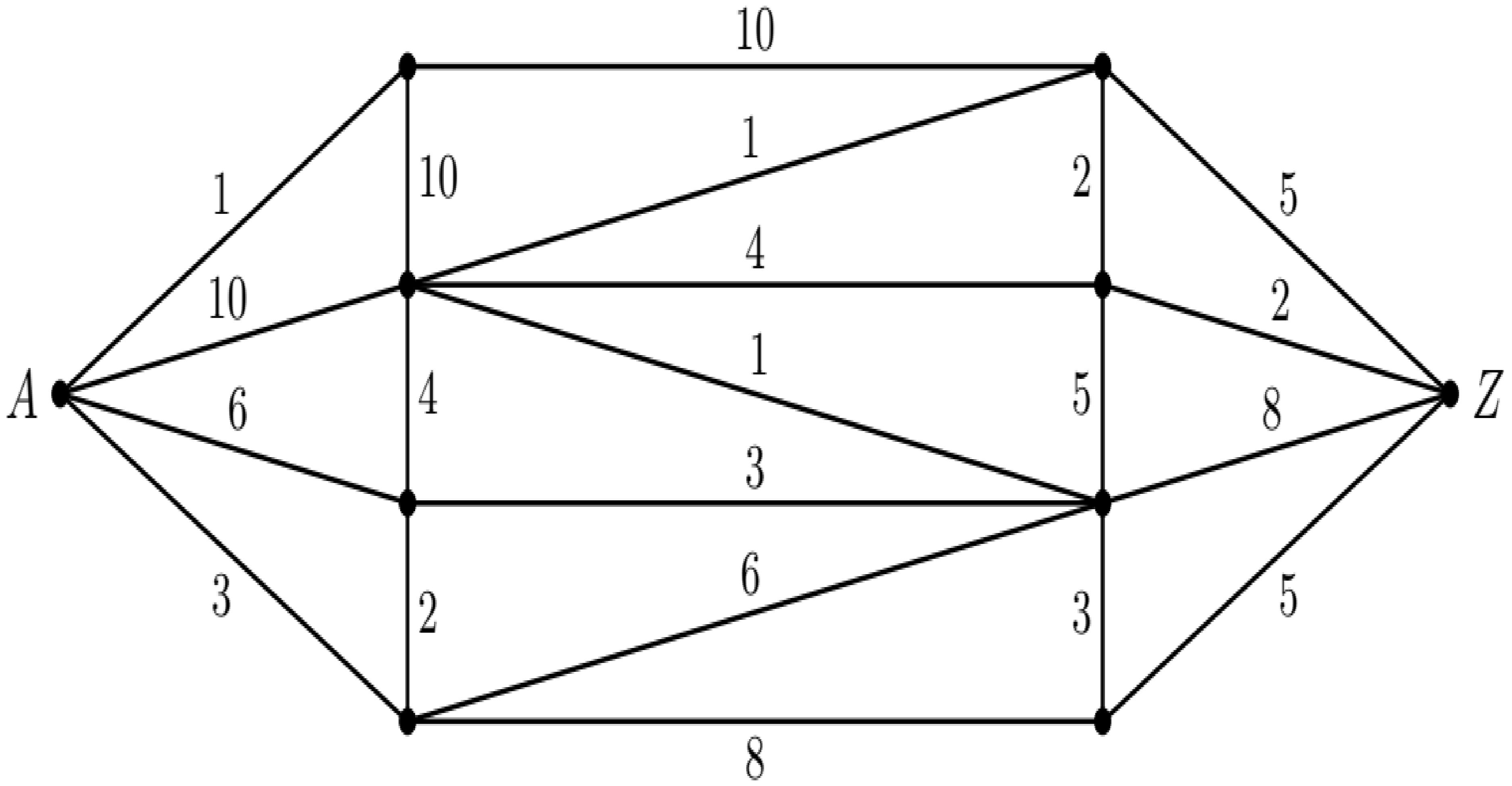


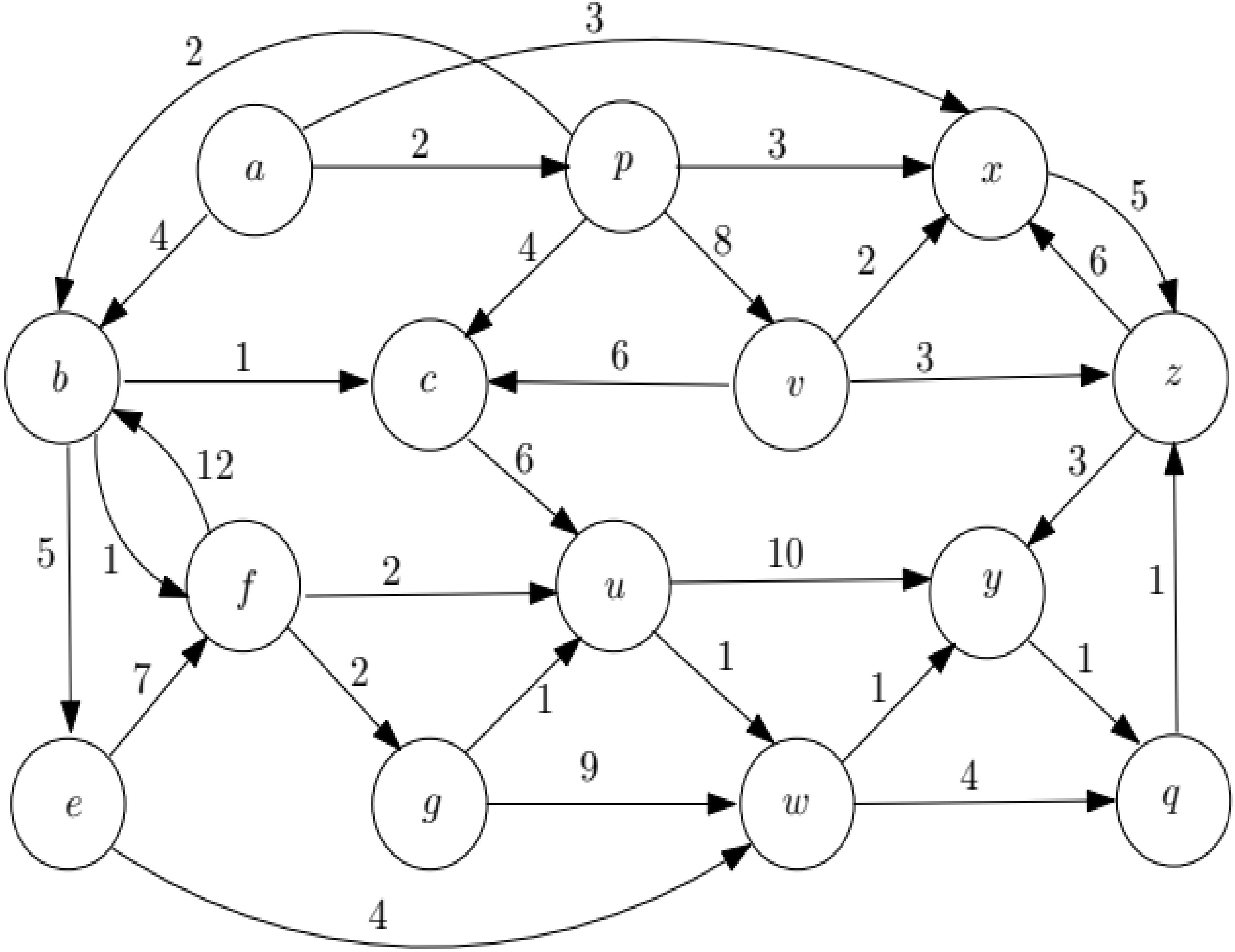
What is the shortest path between B and E in this network?

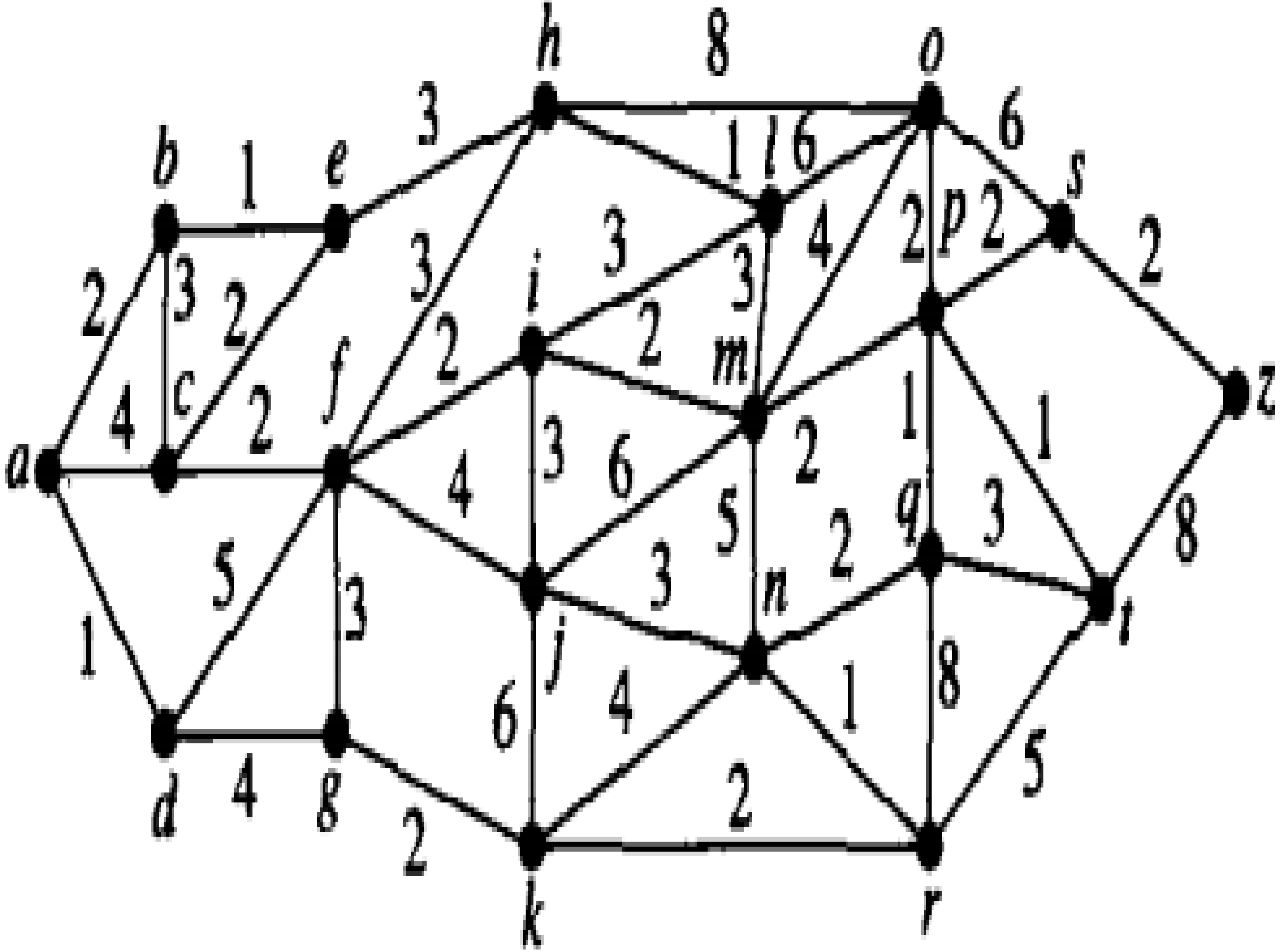
The shortest path from A to any other point



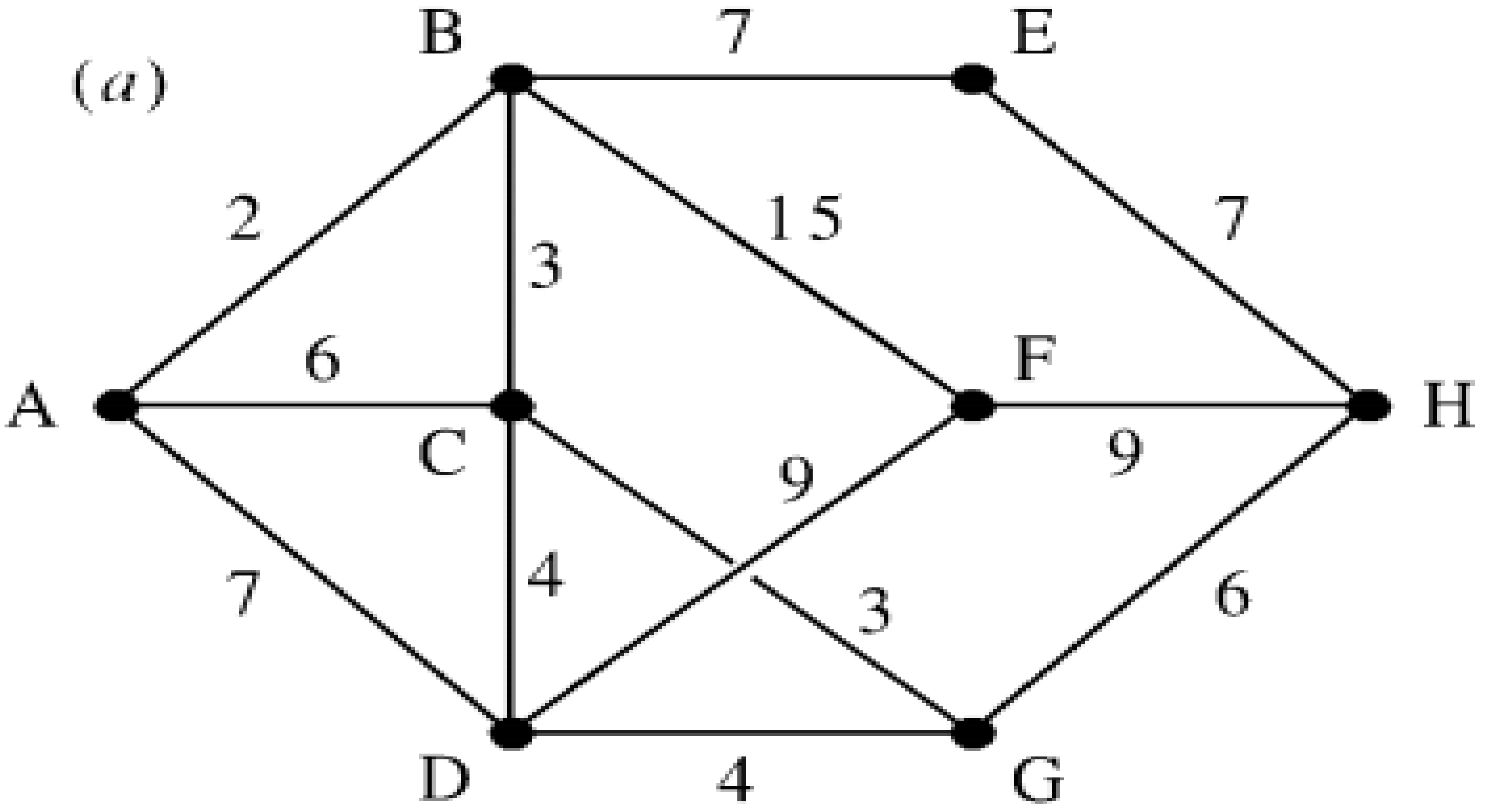
In the following graph, find all shortest paths from A to Z.

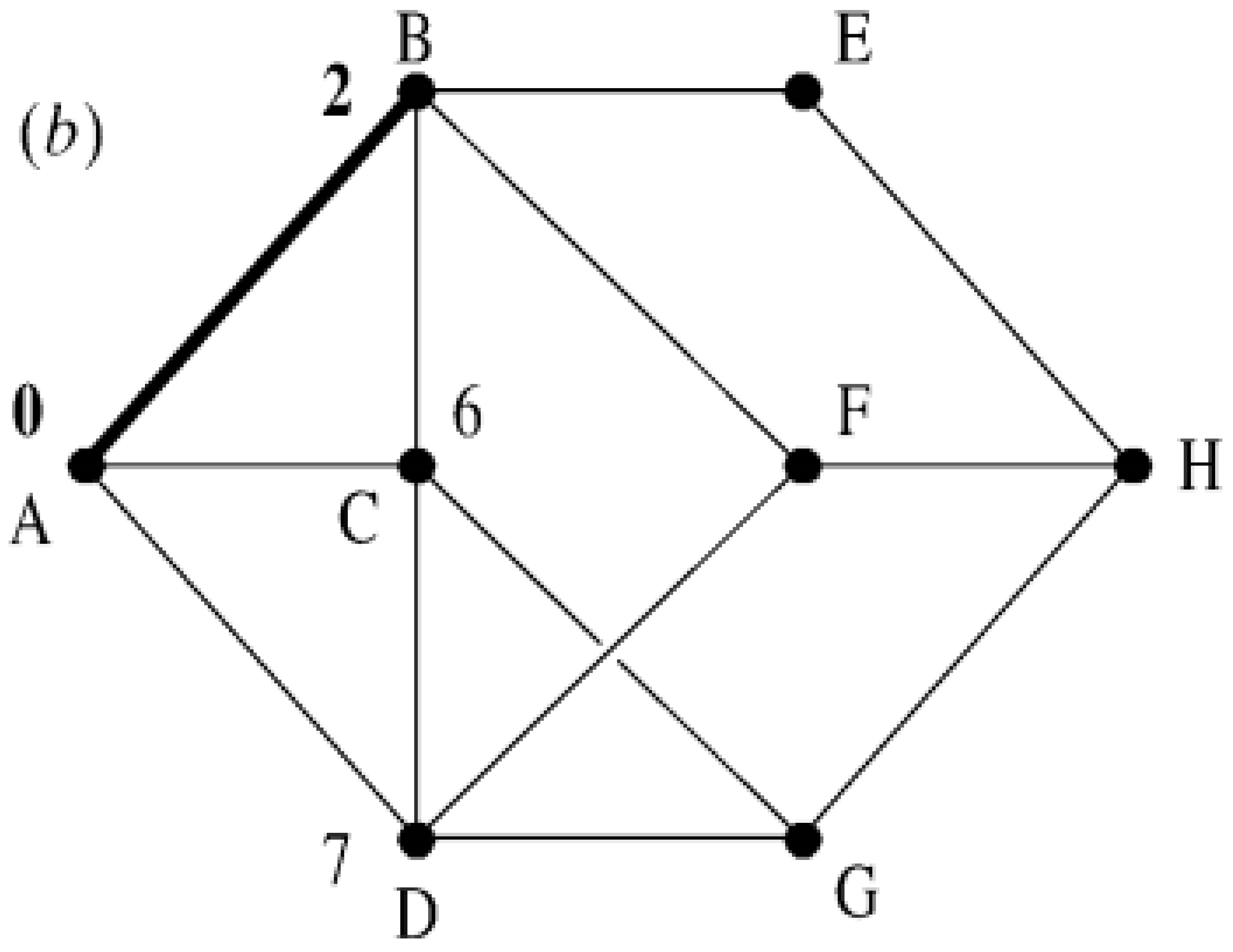


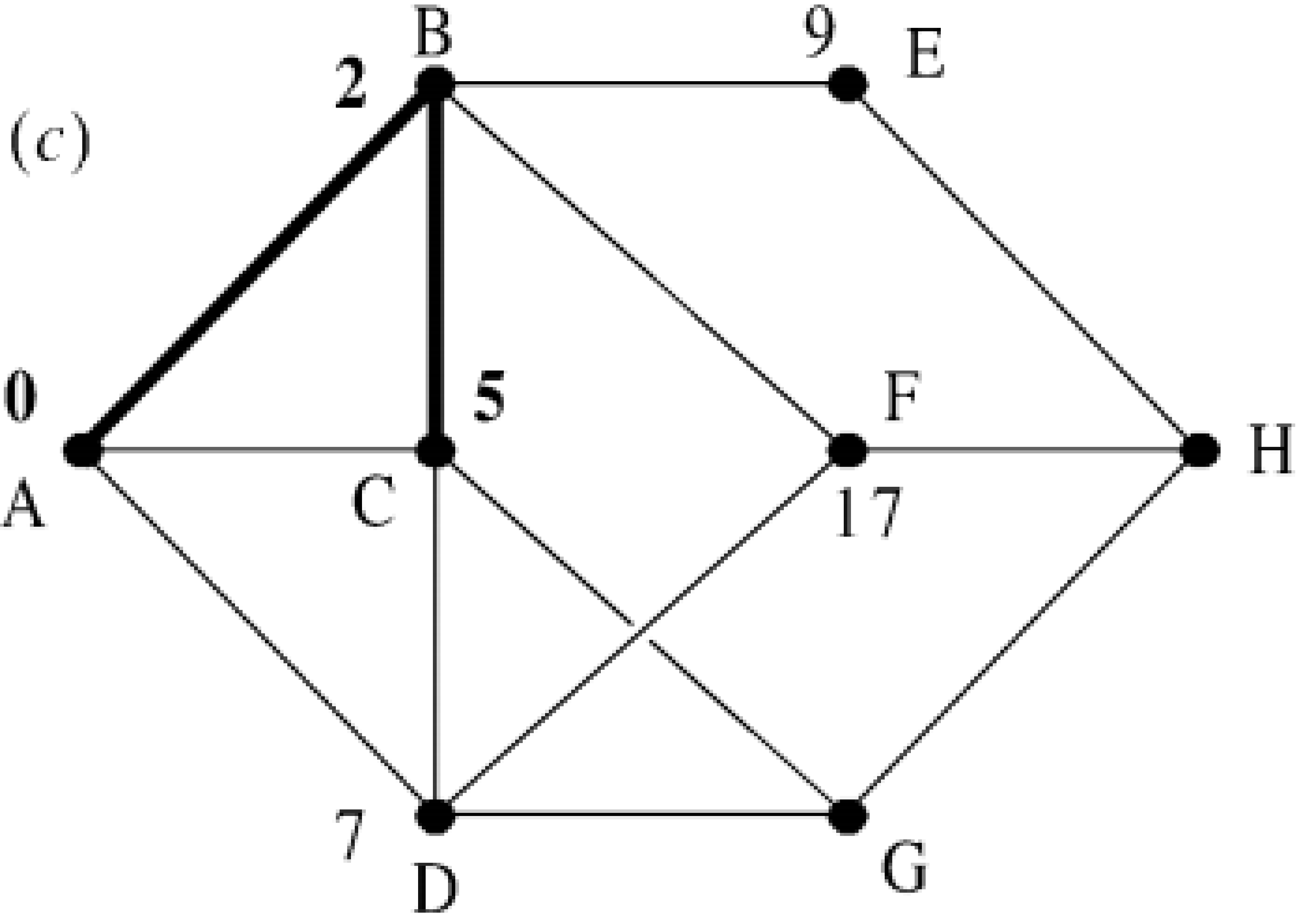

a
p
x
b
c
v
z
f
u
y
e
g
w
q
2
3
2
3
5
4
4
8
2
6
1
6
3
12
6
3
5
1
2
10
1
7
2
1
1
1
9
4
4

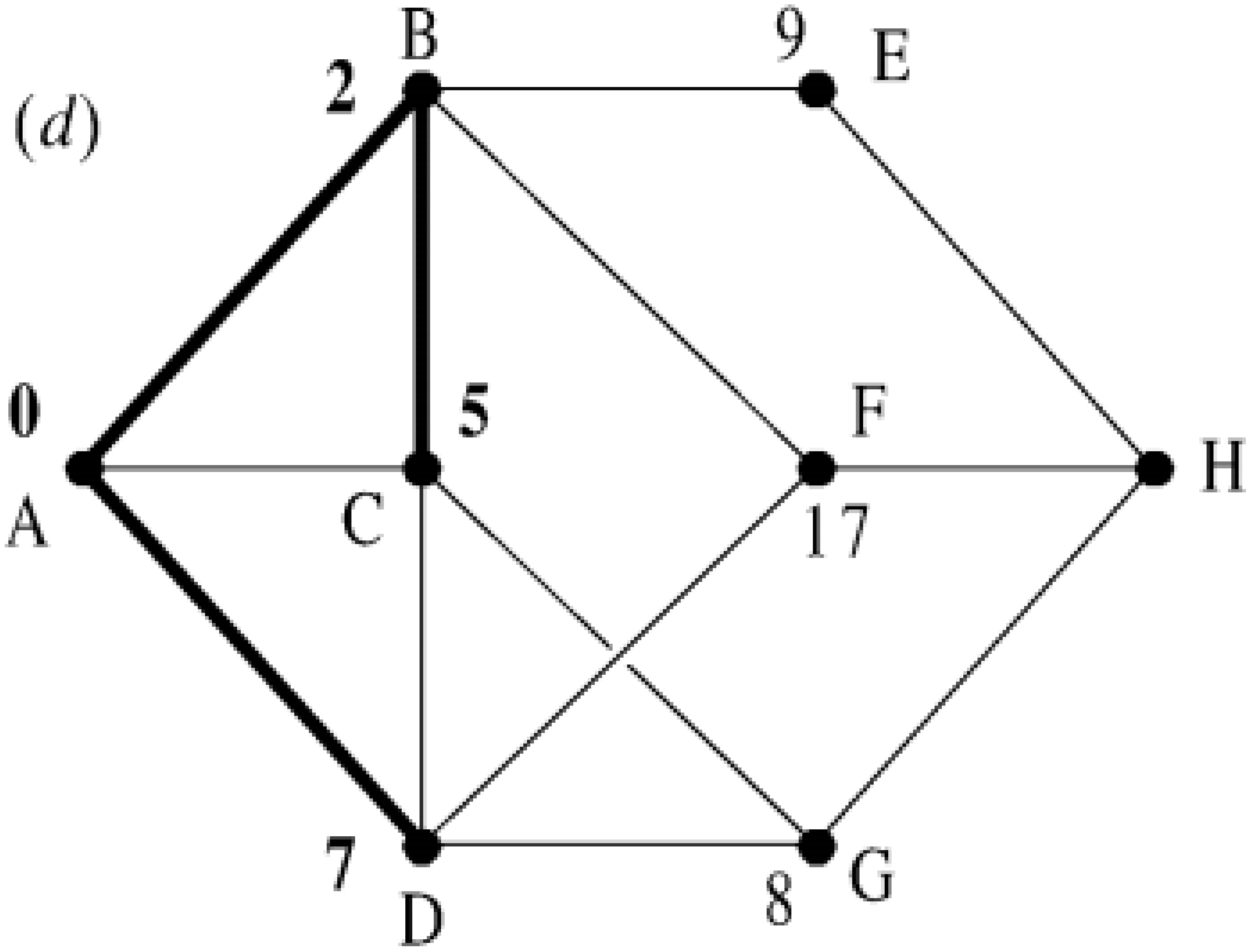

h
8
o
3
1
l
6
6
b
1
e
s
2
3
4
2
p
2
2
i
3
3
2
3
2
2
m
f
c
4
2
z
1
3
6
1
2
a
4
q
3
5
5
2
3
8
n
3
t
j
1
6
4
1
8
d
4
g
2
2
5
k
r

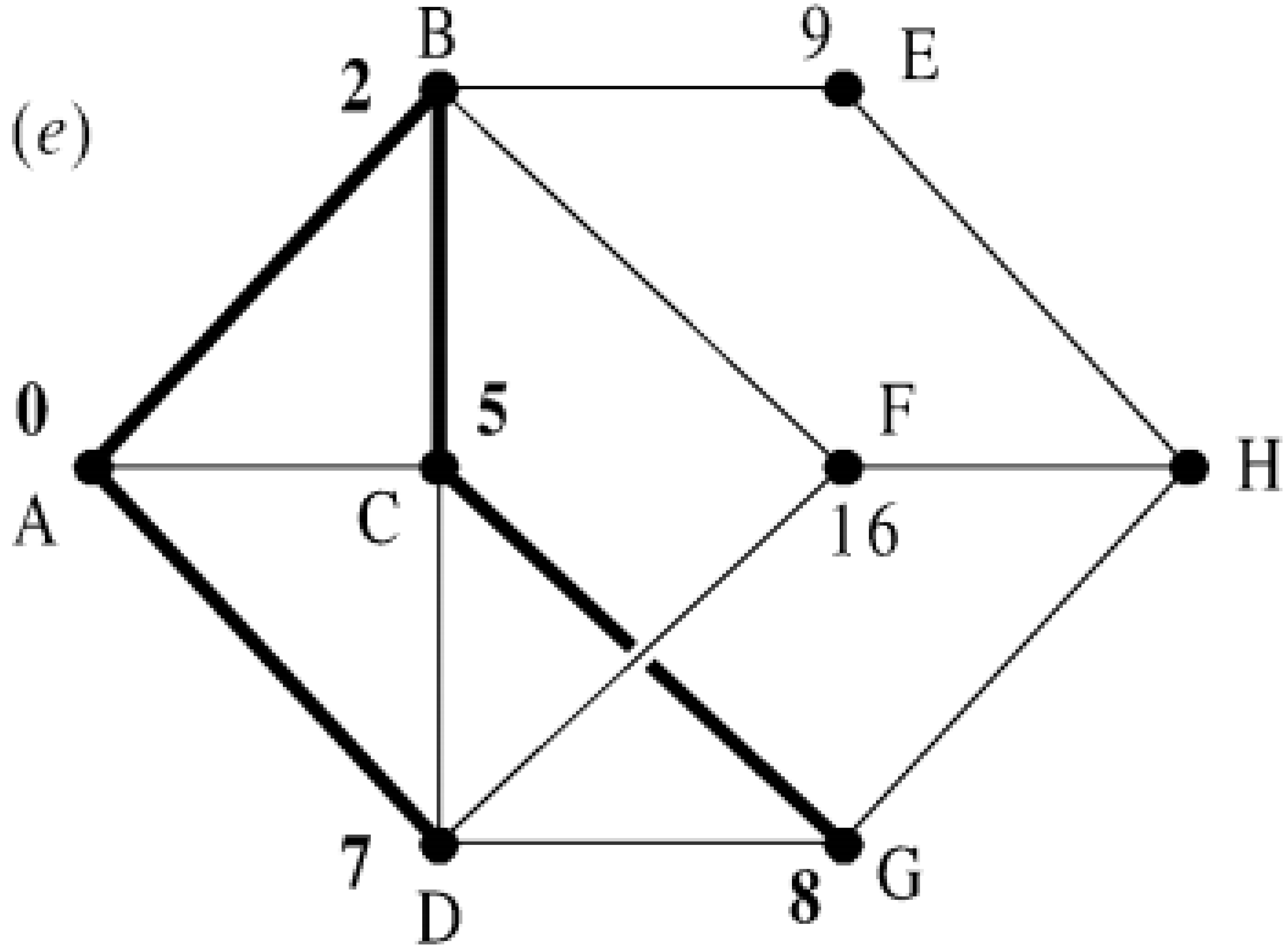
The shortest path from A to any other point

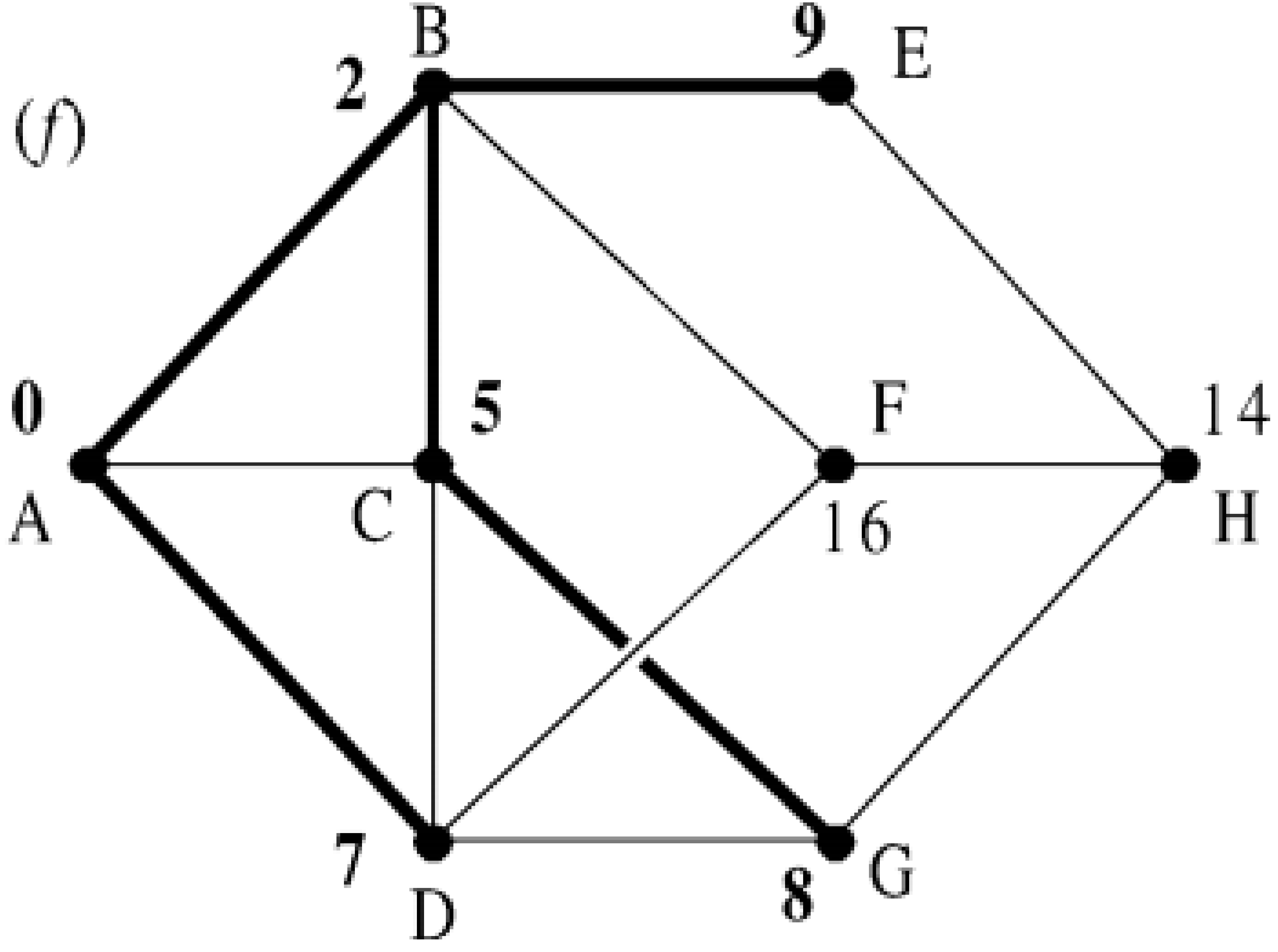



(b)
B
E
2
0
6
F
H
A
C
7
G
D

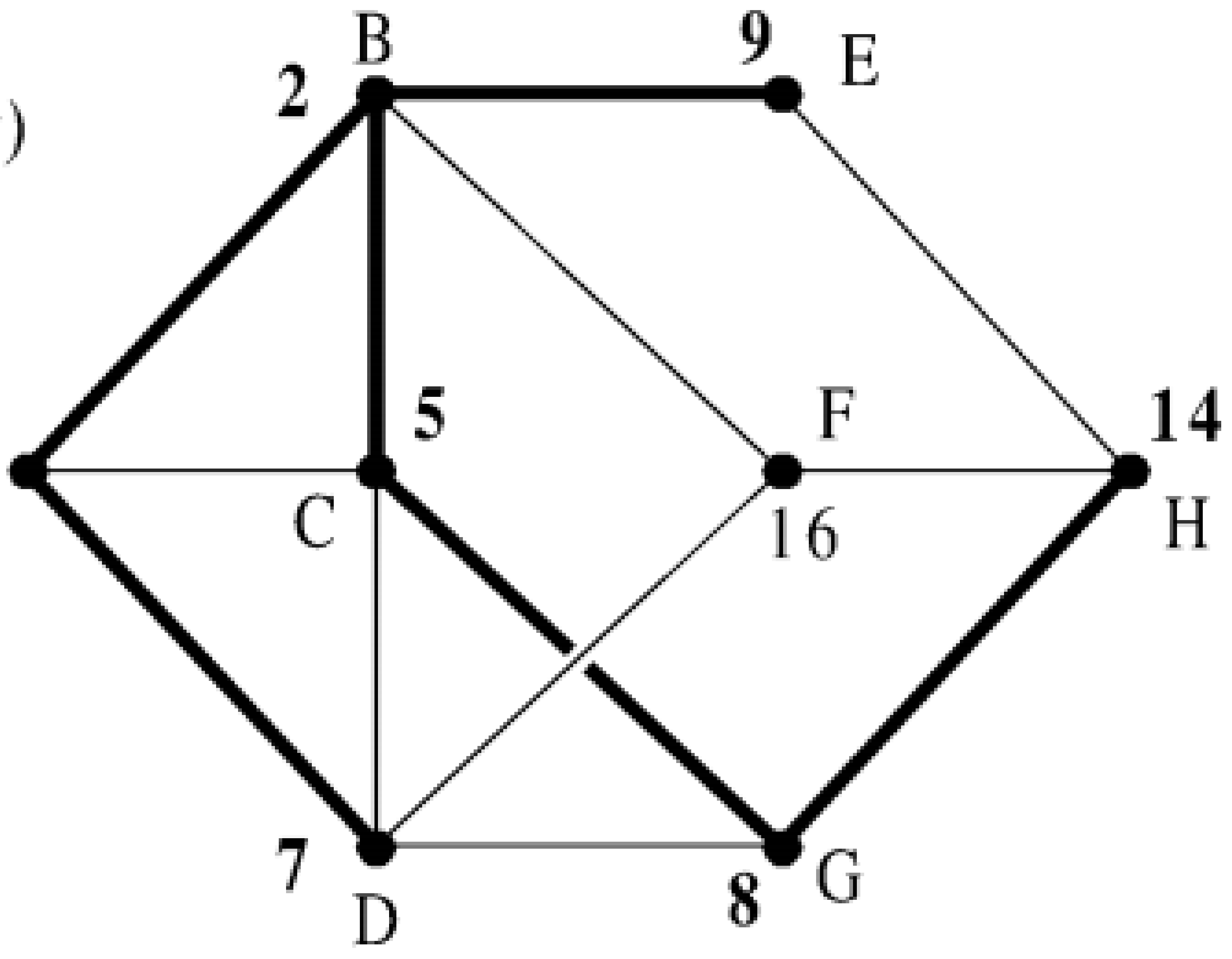

(c)
B
2
9
E
0
A
5
C
F
17
H
7
D
G


(d)
B
2
9
E
0
5
F
H
A
C
17
7
8
G
D

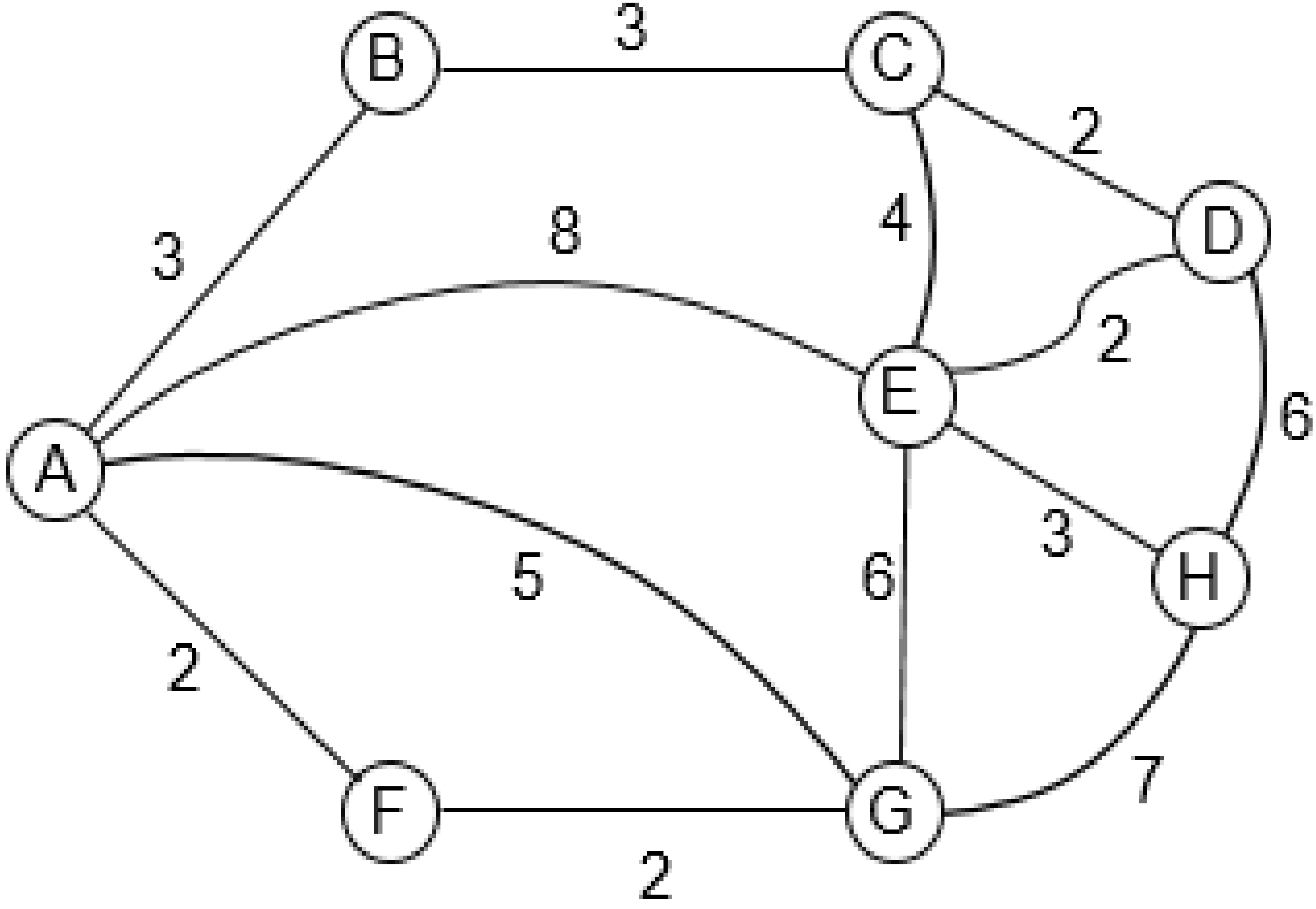

(e)
B
2
9
E
0
5
F
A
C
16
H
7
D
8
G


(f)
B
2
9
E
0
5
F
14
A
C
16
H
7
D
8
G

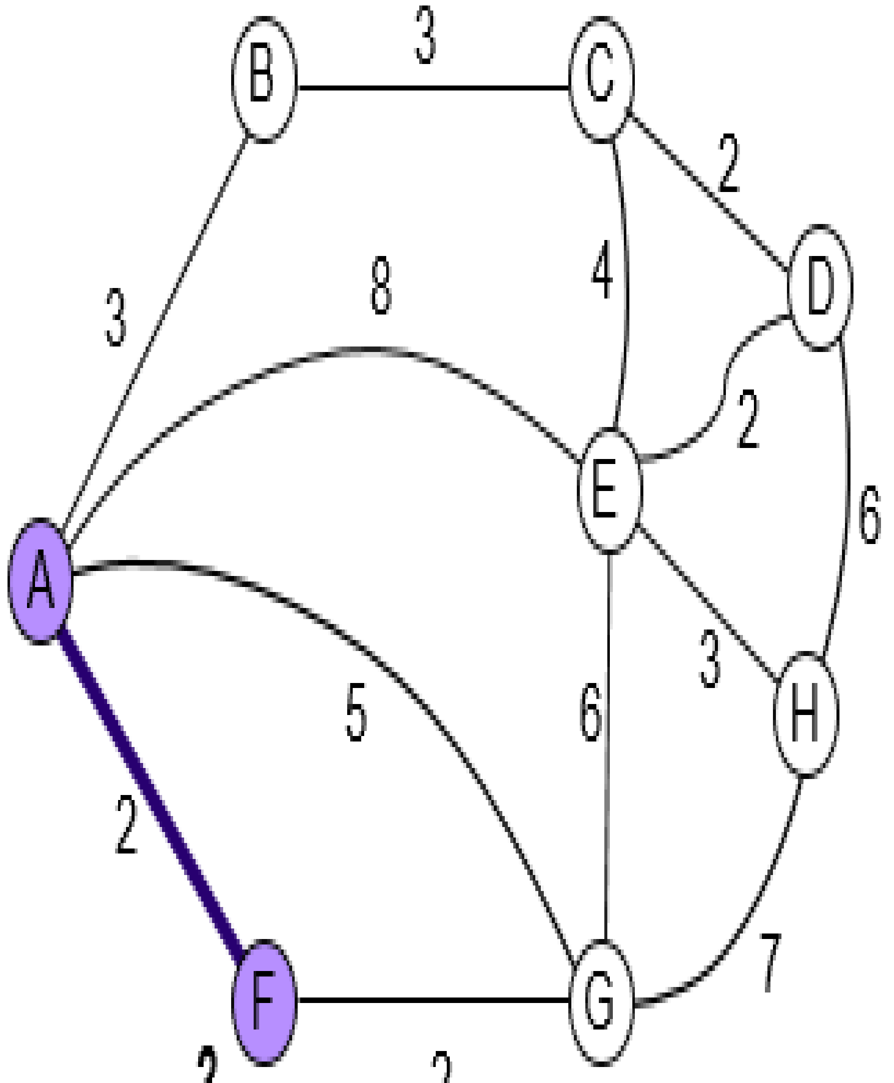
(*g*)



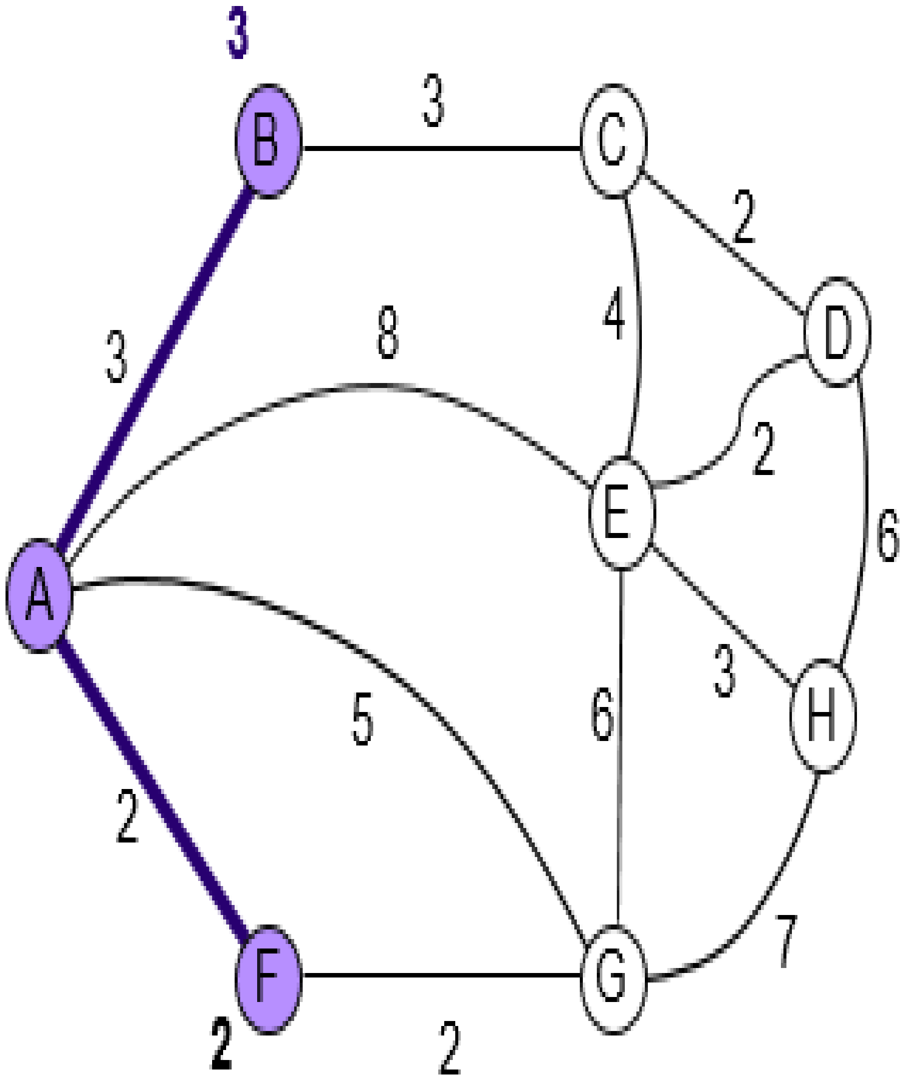
The shortest path from A to any other point



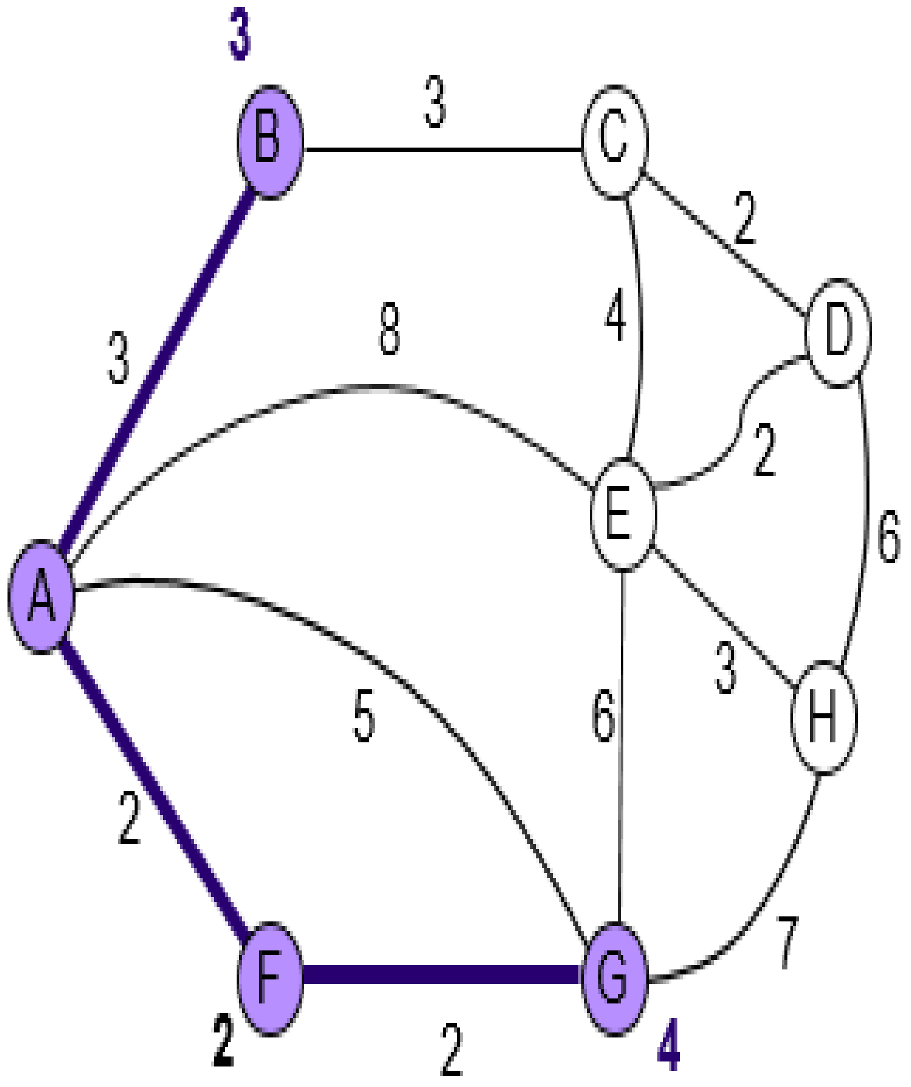
The path starts at A so colour A. F is closer to A than B so colour F and the edge AF. Place a 2 beside F.



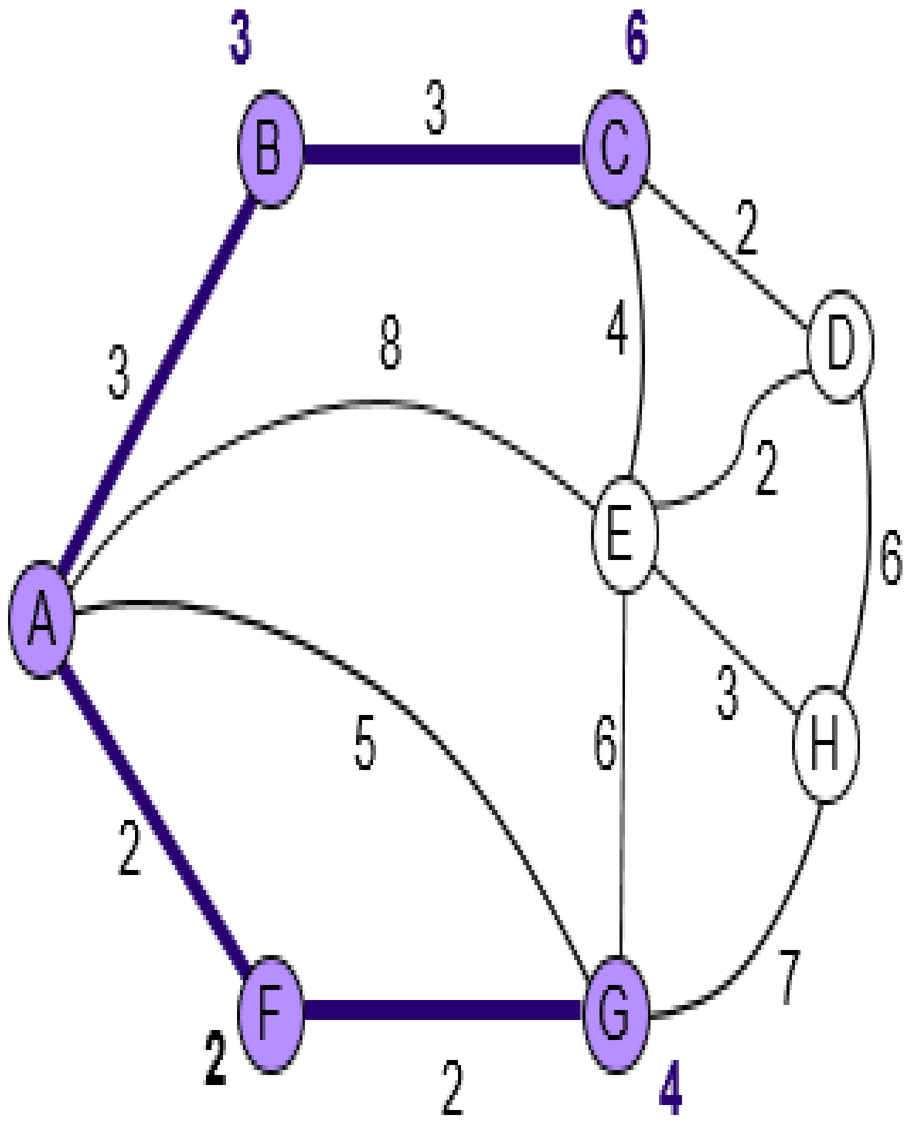
Check B, E and G to see how far they are from A. B is closest to A so colour B and the edge AB. Place a 3 beside B.



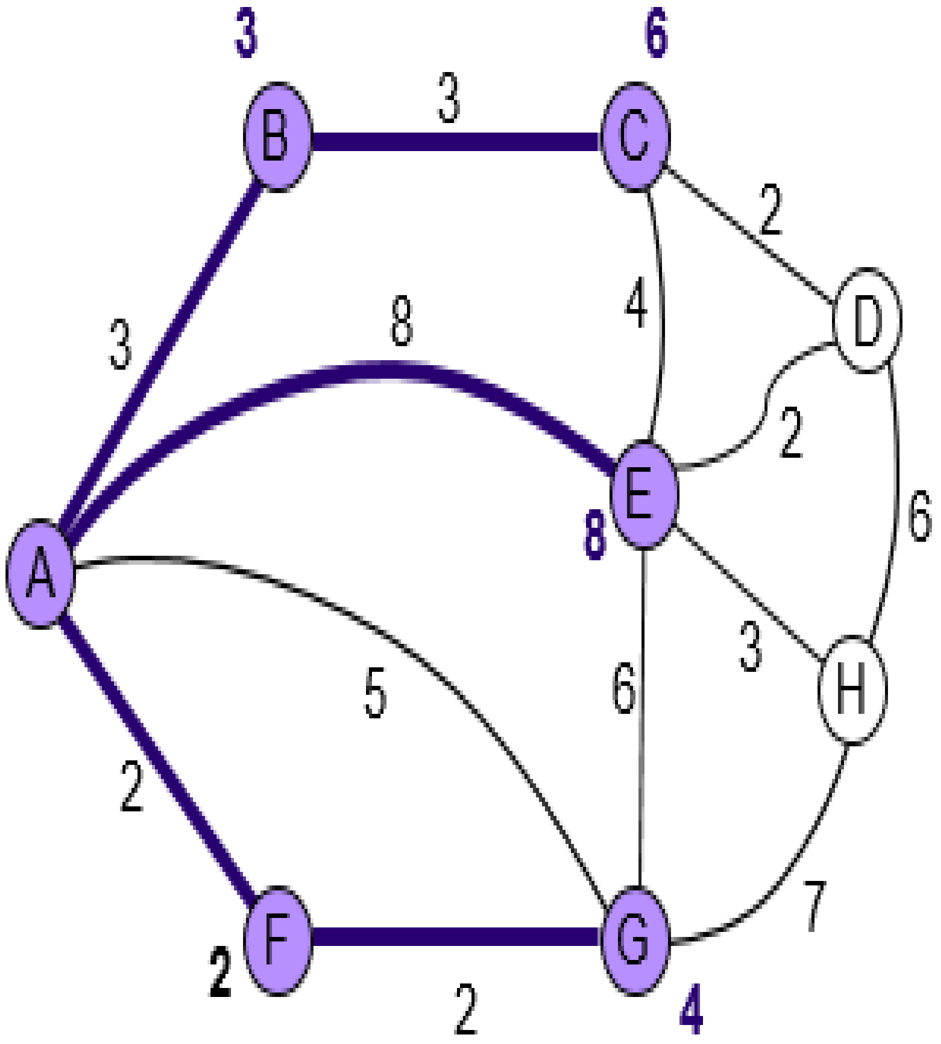
Check C, E and G to see how far they are from A. G is closest to A so colour G and the edge FG. Place a 4 beside G.



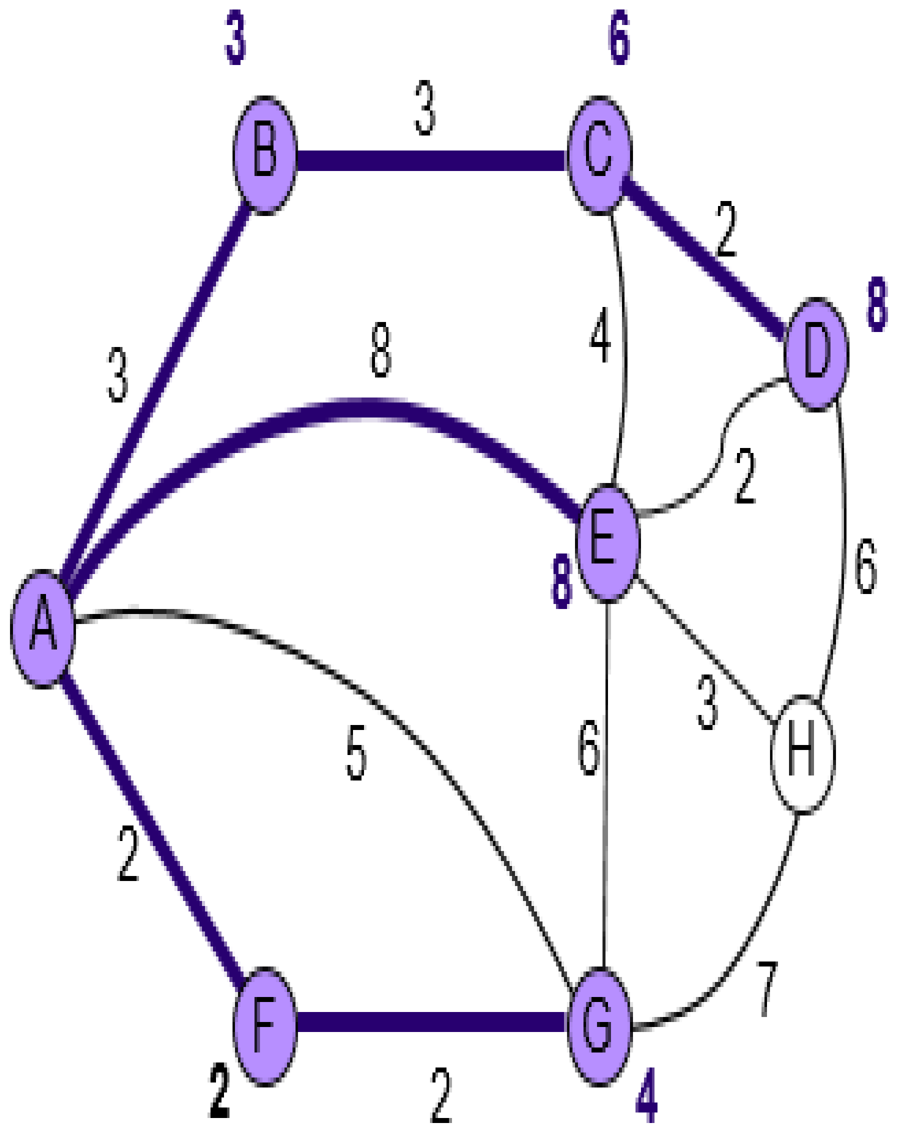
Check C, E and H to see how far they are from A. C is closest to A, so colour C and the edge BC. Place a 6 beside C.



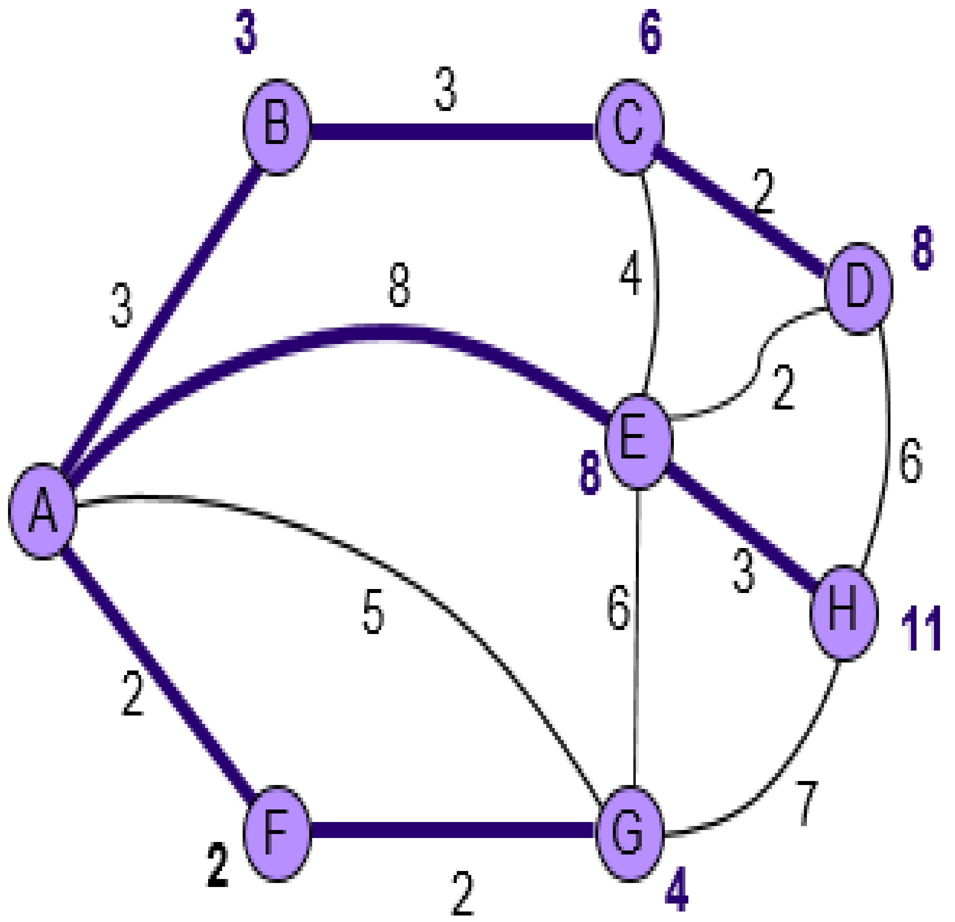
Check D, E and H to see how far they are from A. D and E are both the same distance from A so choose either one of these, say E. Colour E and the edge AE. Place an 8 beside E.



Check D and H to see how far they are from A. D is closest, so colour D and the edge CD. Place an 8 beside D.



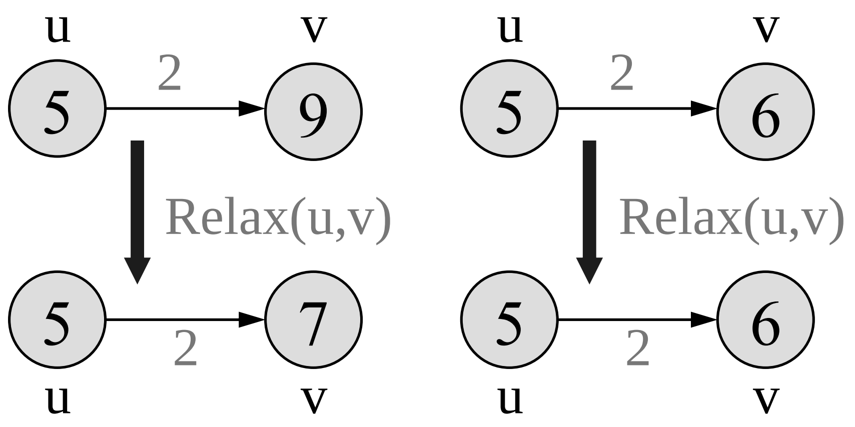
Check H to see how far it is from A. The distance is 11 either along EH or GH so choose one of these, say EH. Colour both H and EH and place 11 beside H.



We have arrived at H and so a shortest distance from A to H is 11 along path AEH or equivalently AFGH.

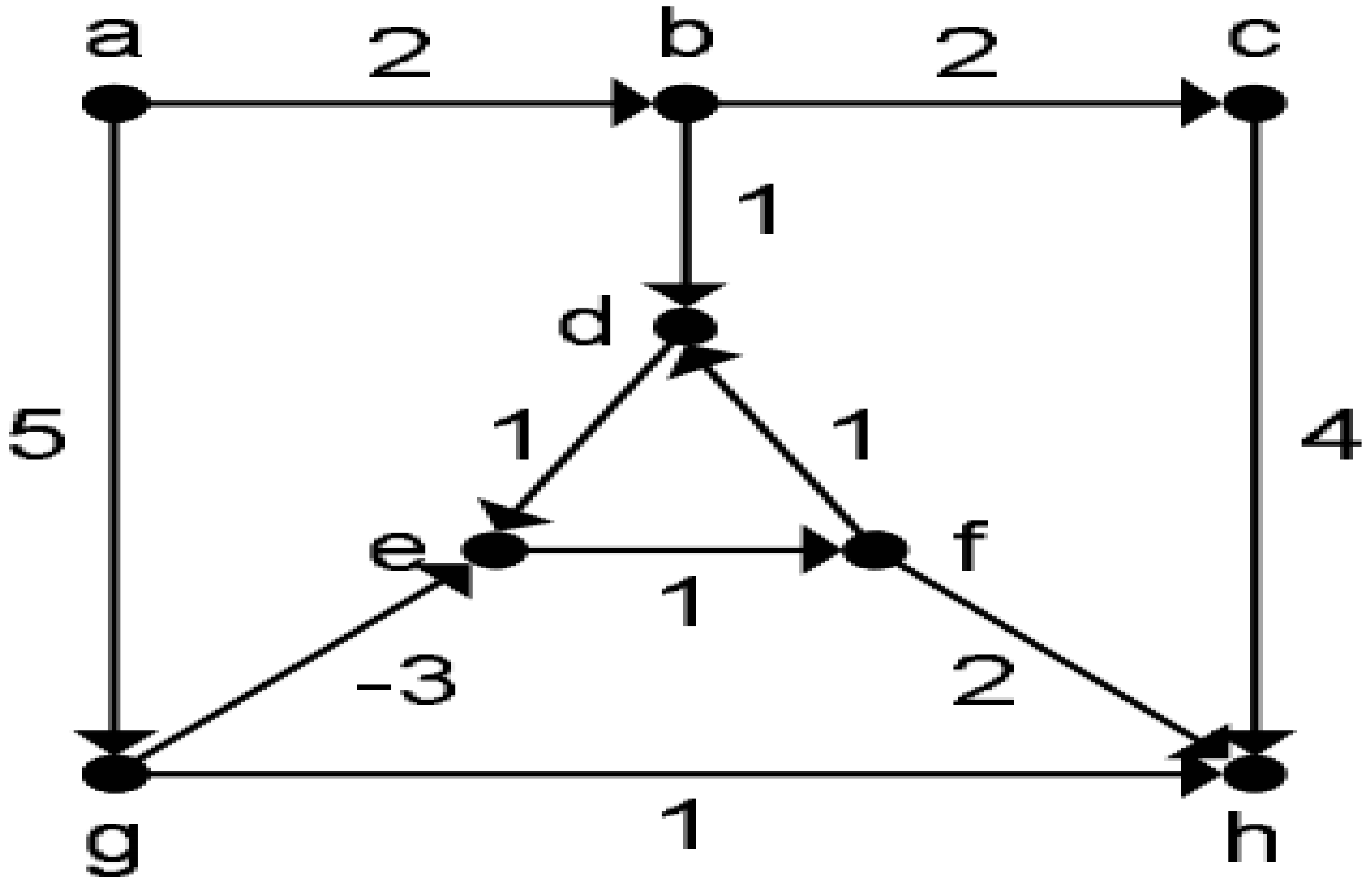
Relaxation

- For each vertex *v* in the graph, we maintain *v*.**d**(), the estimate of the shortest path from *s*, initialized to ∞ at the start
- Relaxing an edge (*u*,*v*) means testing whether we can improve the shortest path to *v* found so far by going through *u*

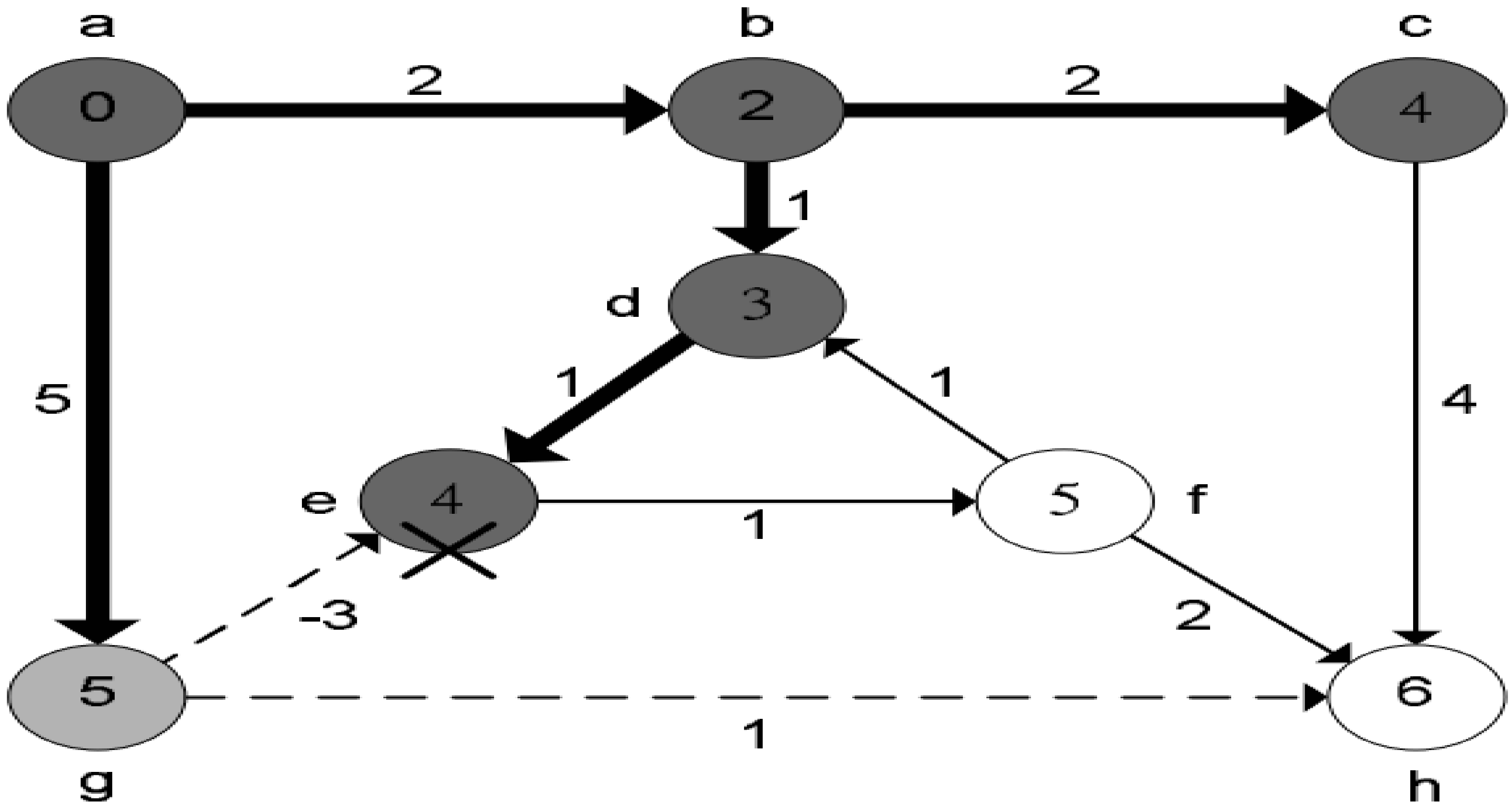


```
Relax (u,v,G)
if v.d() > u.d()+G.w(u,v) then
    v.setd(u.d()+G.w(u,v))
    v.setparent(u)
```

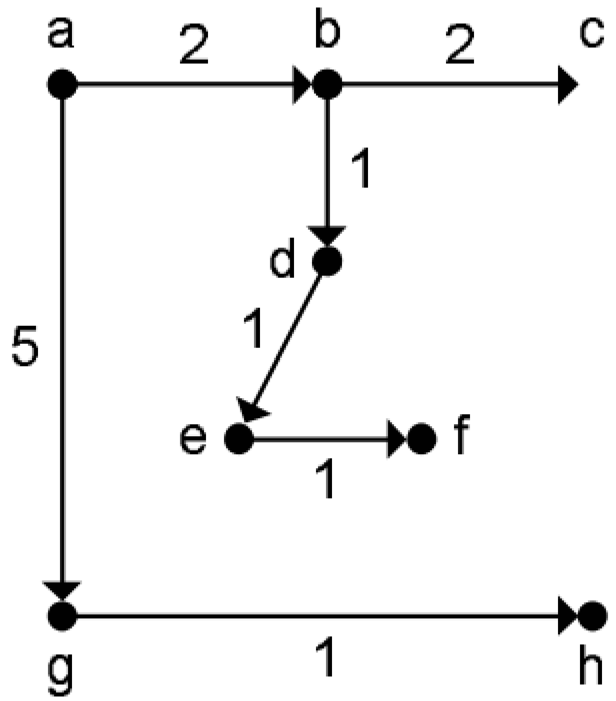
Apply **Dijkstra algorithm**, for the following **graph** and find the shortest path from the source vertex a to the other vertices.

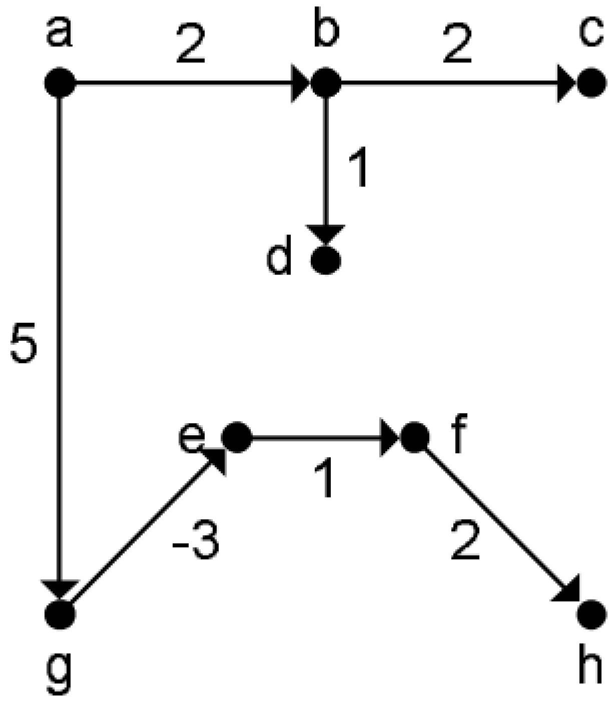


A failure of Dijkstra's algorithm.



A failure of Dijkstra's algorithm.





Bellman–Ford algorithm

Algorithm: *Initialize-Single-Source (G, S)*

```
{
    for all vertices q∈V do
            D[q] := ∞;
             π [q] : = NIL; //Predecessor
    D[S]:= 0;
}
```

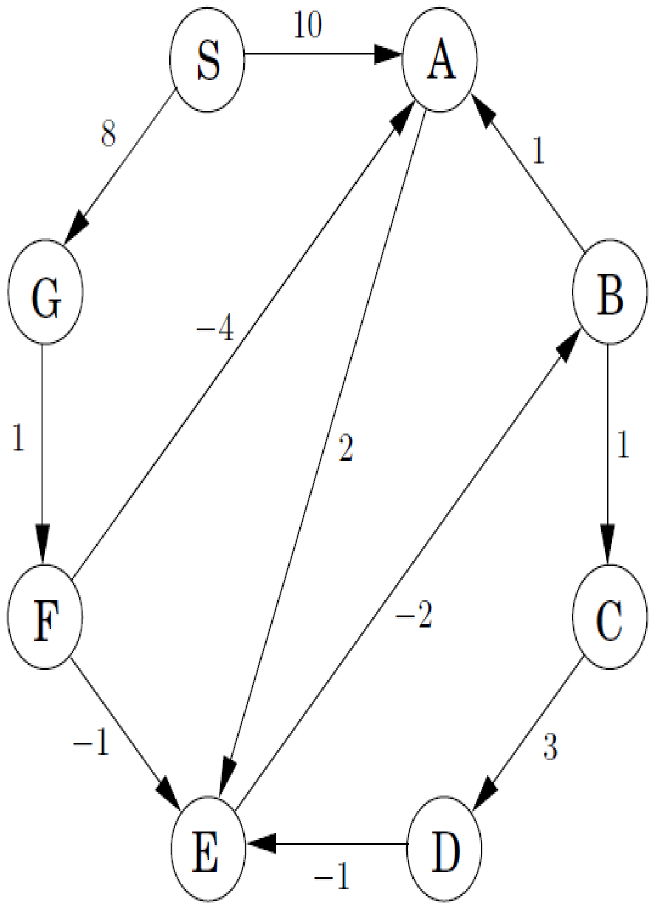
Algorithm: *Relax (p, q, w)*

```
{
    if (D[q] > D[p]+ w[p,q])
            Then D[q]: = D[p]+ w[p,q];
             π [q]: = p;
}
```

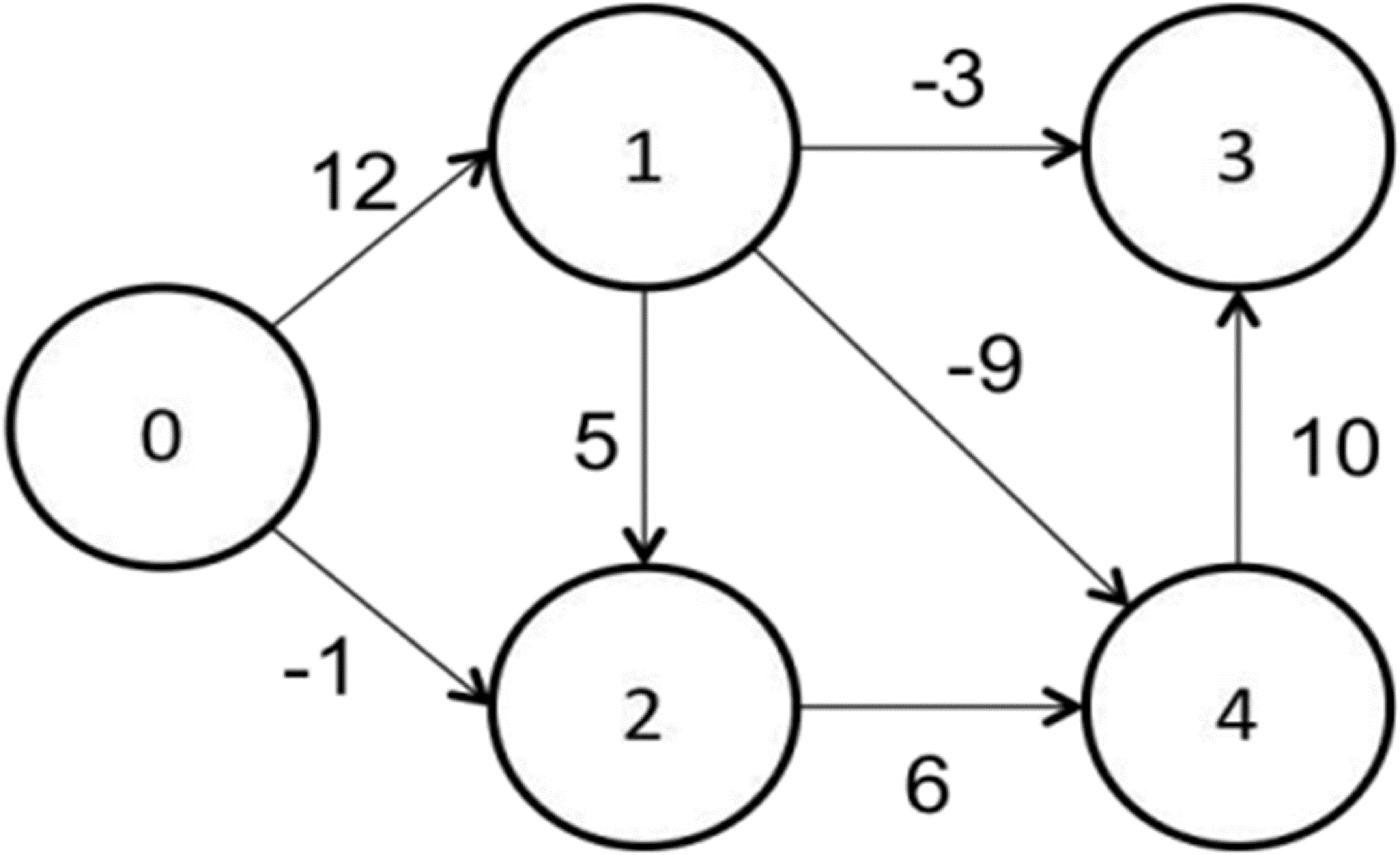
Algorithm: *BELLMAN-FORD (G,W,S)*

```
{
    Initialize-Single-Source(G,S)
    for j: = 1 to |V|-1
            do for every edge (p, q)∈E
                    do RELAX (p, q, W);
    for every edge (p, q)∈E
            do if D[q] > D[p] + w (p, q)
                    then return FALSE
    return TRUE                    // no negative weight cycle
}
```

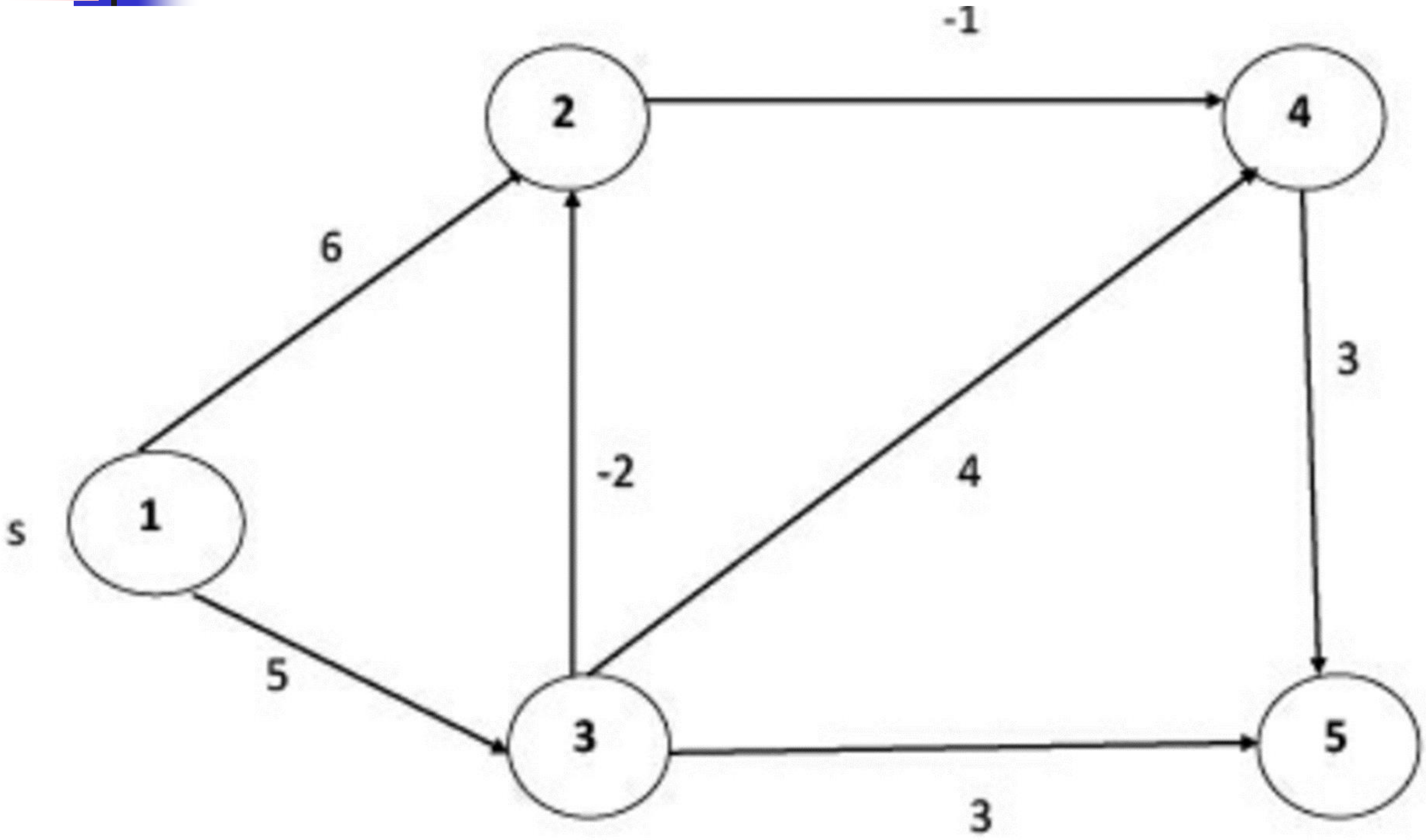
Run Bellman-Ford algorithm for the following graph, provided by E = {(S,A),(S,G),(A,E),(B,A),(B,C),(C,D),(D,E),(E,B),(F,A),(F,E),(G,F)}



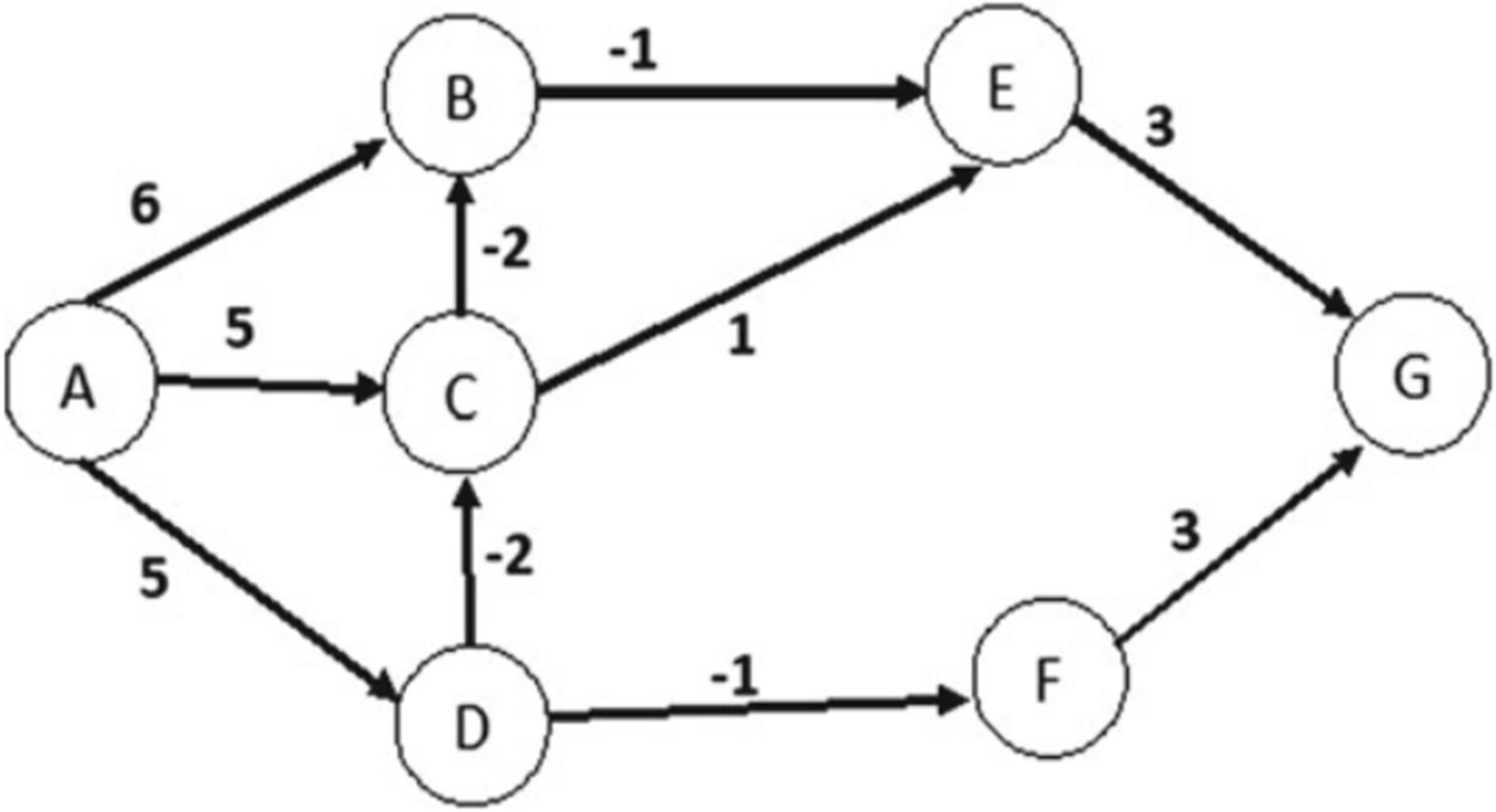
Run Bellman-Ford algorithm for the following graph



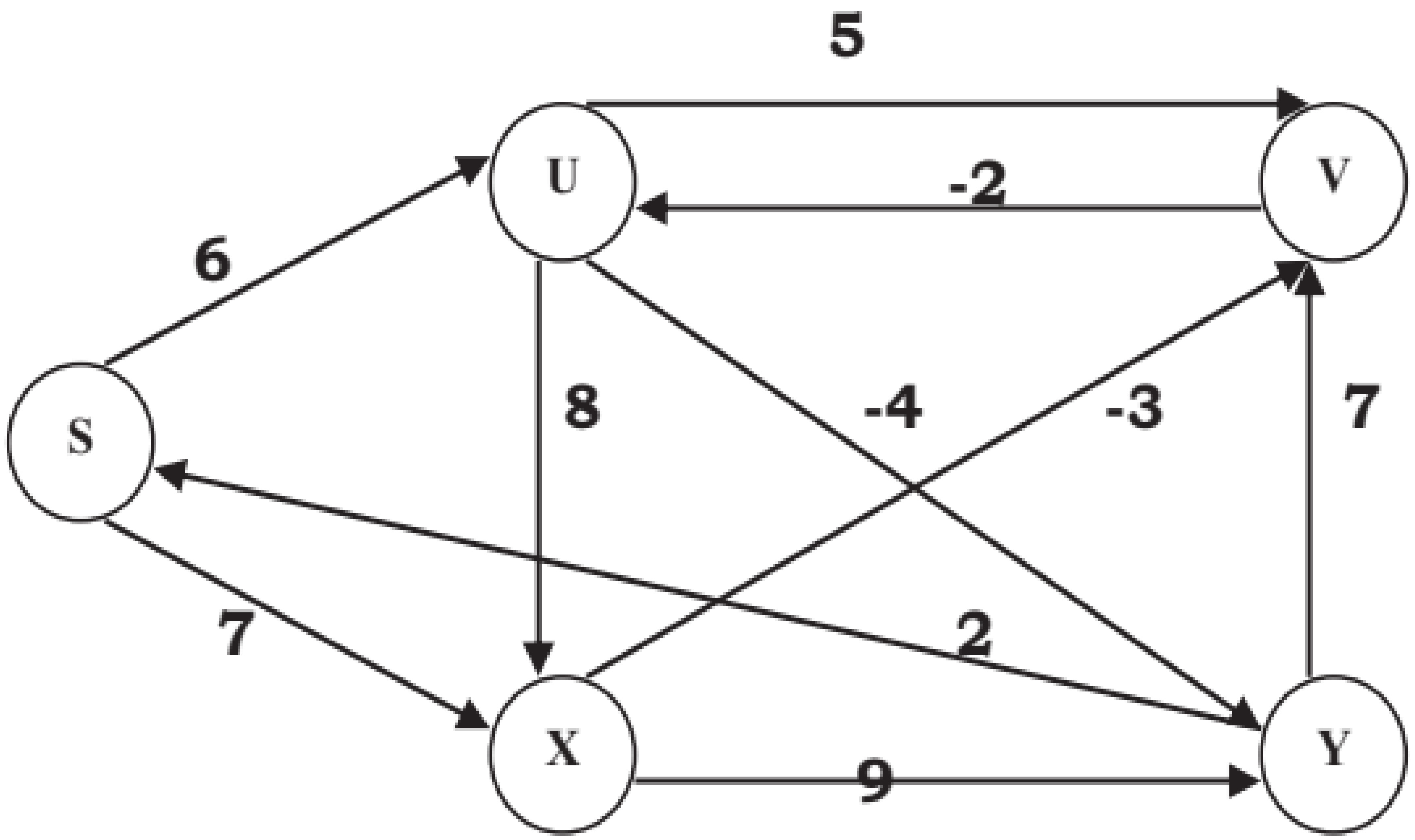
Run Bellman-Ford algorithm for the following graph



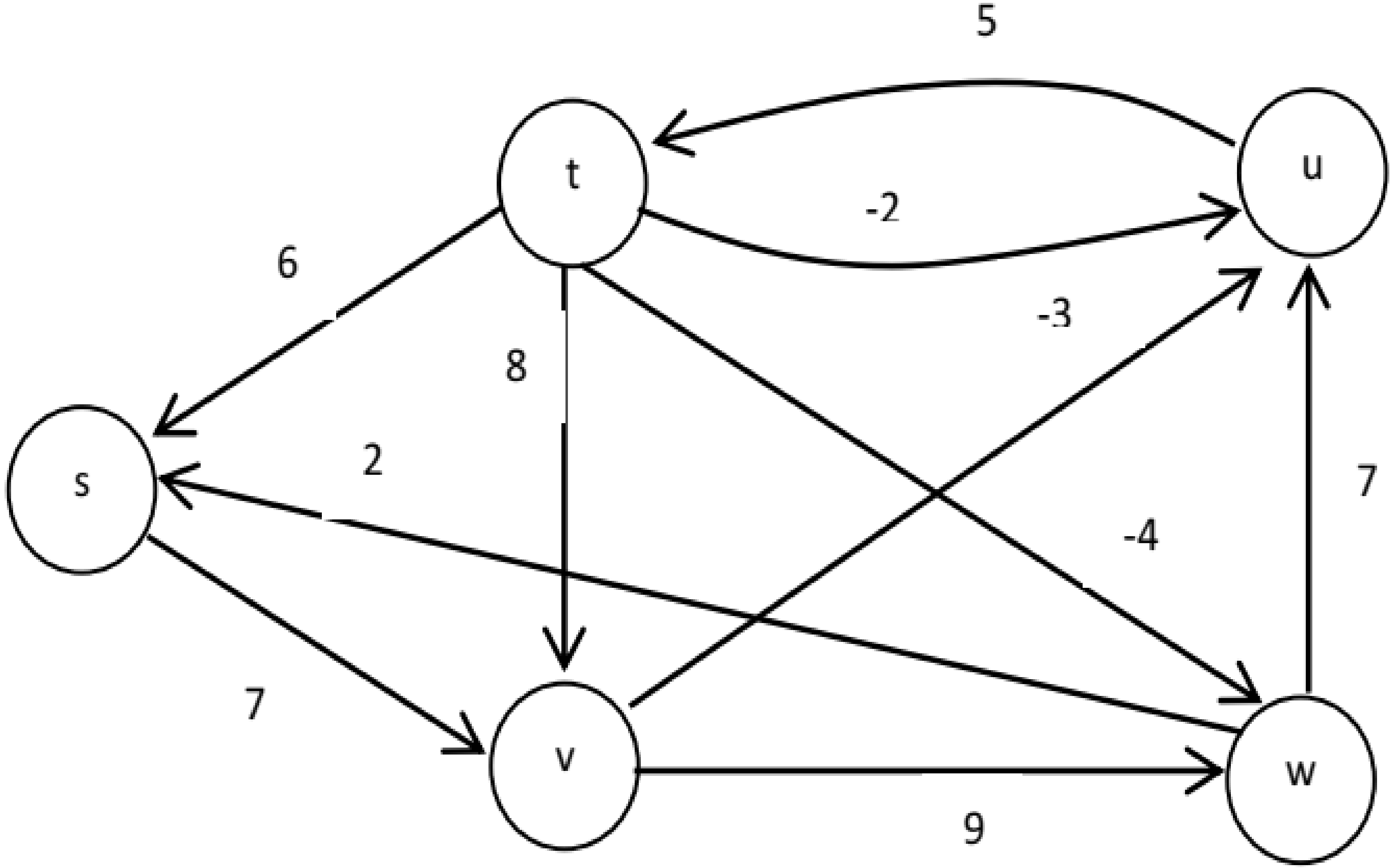
Run Bellman-Ford algorithm for the following graph



Run Bellman-Ford algorithm for the following graph



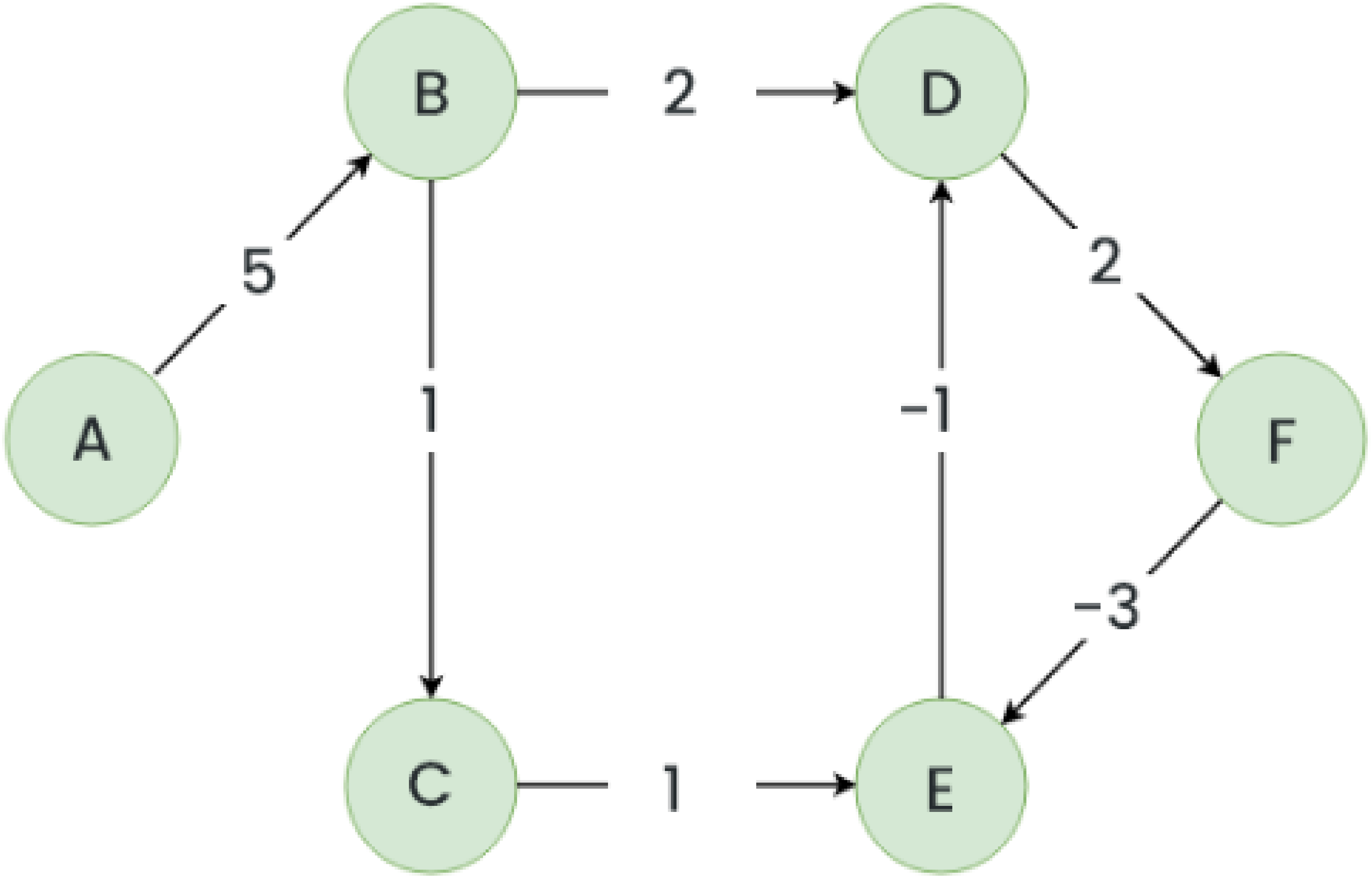
Run Bellman-Ford algorithm for the following graph

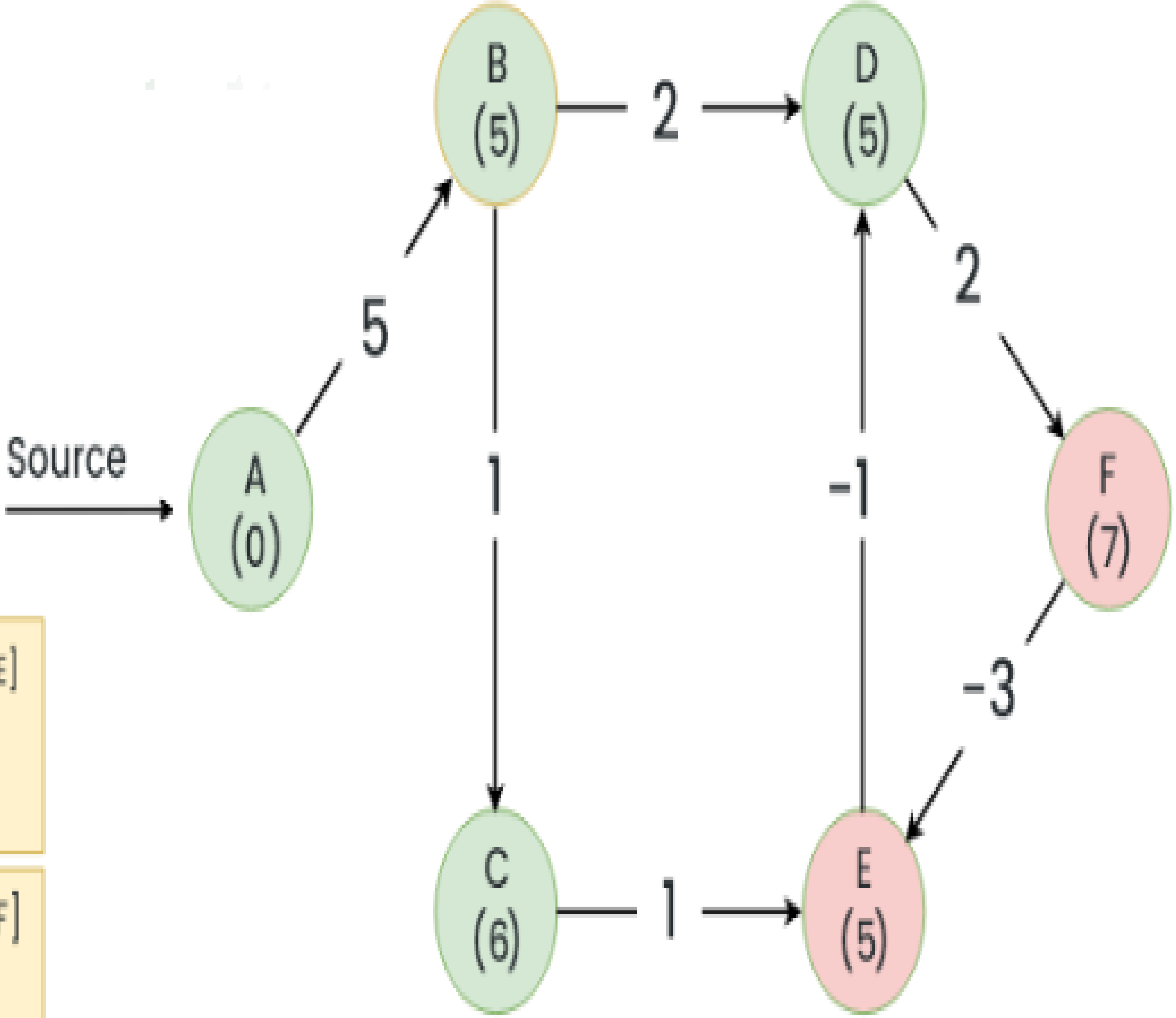


Discussion

T F Given a graph $G = (V, E)$ with positive edge weights, the Bellman-Ford algorithm and Dijkstra's algorithm can produce different shortest-path trees despite always producing the same shortest-path weights.

Solution: True. Both algorithms are guaranteed to produce the same shortest-path weight, but if there are multiple shortest paths, Dijkstra's will choose the shortest path according to the greedy strategy, and Bellman-Ford will choose the shortest path depending on the order of relaxations, and the two shortest path trees may be different.


B
2
D
5
2
A
1
-1
F
-3
C
1
E



Dist [F] + (-3) < Dist[E]

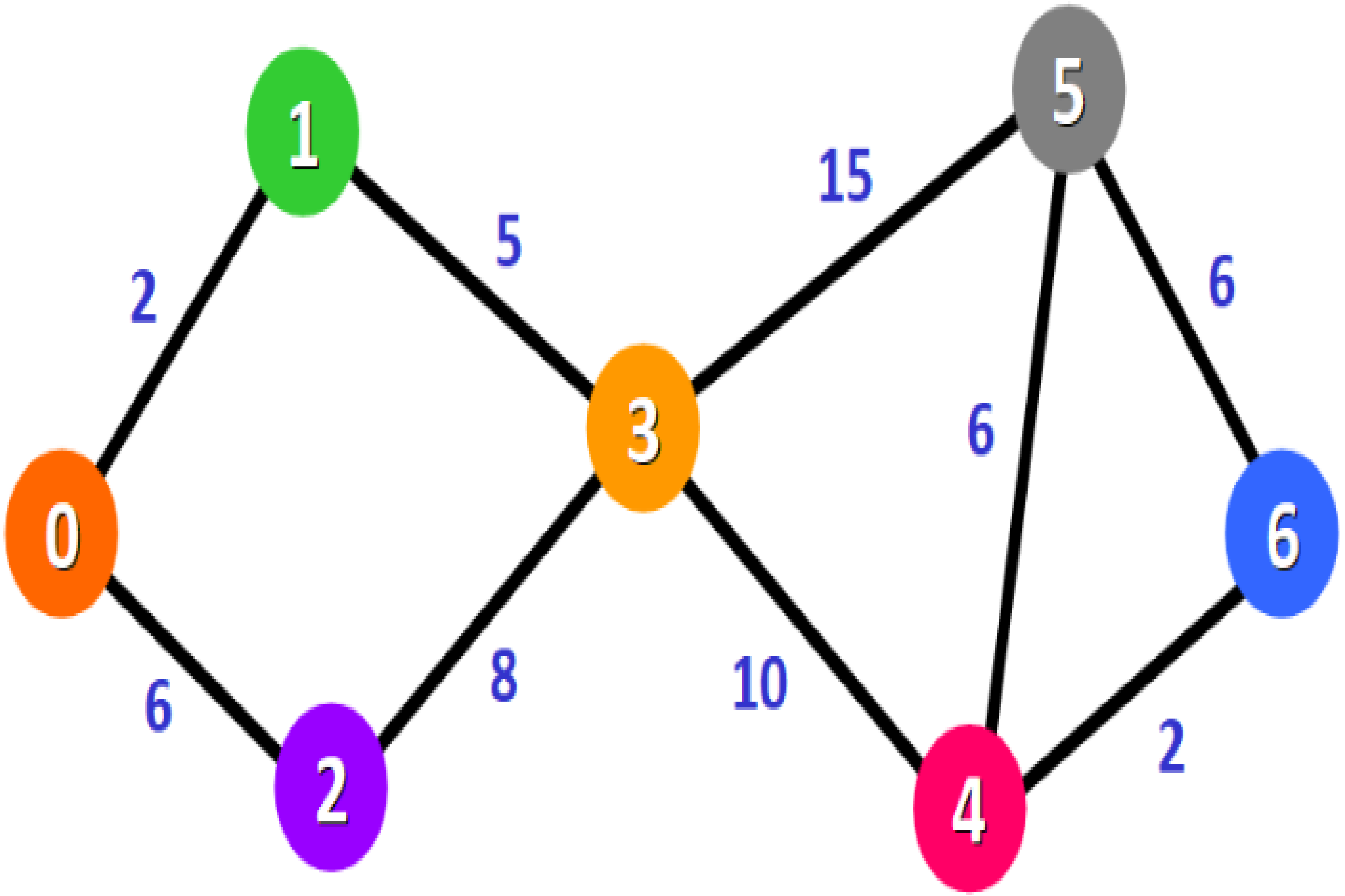
8+(-3)<6

Dist[E] = 5

Dist [D] + 2 < Dist[F]

6+2<8

Dist[F] = 7


1
5
2
15
6
3
6
0
6
8
10
2
2
4
6
5