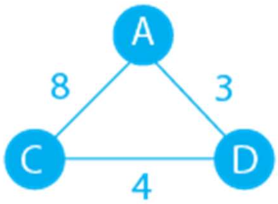
Example of Distance Vector Routing

In the network shown below, there are three routers, A, C, and D, with the following weights − AC =8, CD =4, and DA =3.



Step 1 − Each router in this DVR(Distance Vector Routing) network shares its routing table with every neighbor. For example, A will share its routing table with neighbors C and D, and neighbors C and D will share their routing table with A.

Form A	A	C	D
A	0	8	3
C			
D			

Form C	A	C	D
A			
C	8	0	4
D			

Form D	A	C	D
A			
C			
D	3	4	0

Step 2 − If the path via a neighbor is less expensive, the router adjusts its local table to send packets to the neighbor. In this table, the router updates the lower cost for A and C by updating the new weight from 8 to 7 in router A and from 8 to 7 in router C.

Form A	A	C	D
A	0	7	3
C			
D			

Form C	A	C	D
A			
C	7	0	4
D			

Form D	A	C	D
A			
C			
D	3	4	0

Step 3 − The final revised routing table with the reduced cost distance vector routing protocol for all routers A, C, and D is shown below-

Router A

Form A	A	C	D
A	0	7	3
C	7	0	4
D	3	4	0

Router C

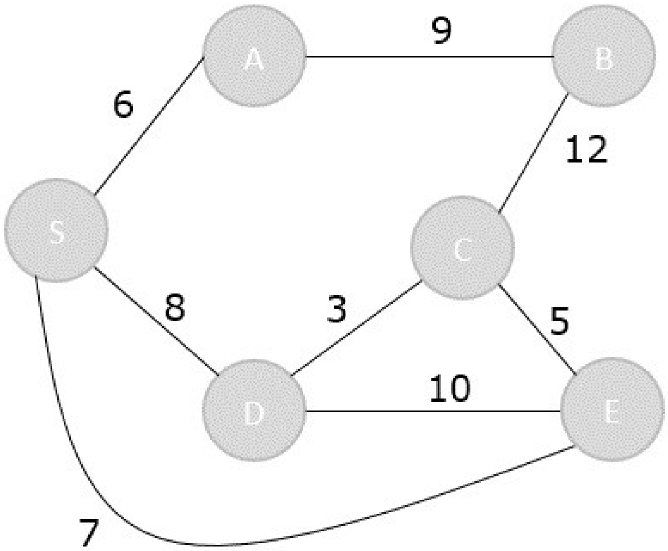
Form C	A	C	D
A	0	7	3
C	7	0	4
D	3	4	0

Router D

Form D	A	C	D
A	0	7	3
C	7	0	4
D	3	4	0

DIJKSTRA EXAMPLE

To understand the dijkstra’s concept better, let us analyze the algorithm with the help of an example graph −



Step 1

Initialize the distances of all the vertices as ∞, except the source node S.

Vertex	S	A	B	C
Distance	0	∞	∞	∞

Now that the source vertex S is visited, add it into the visited array.

visited = {S}

Step 2

The vertex S has three adjacent vertices with various distances and the vertex with minimum distance among them all is A. Hence, A is visited and the dist[A] is changed from ∞ to 6.

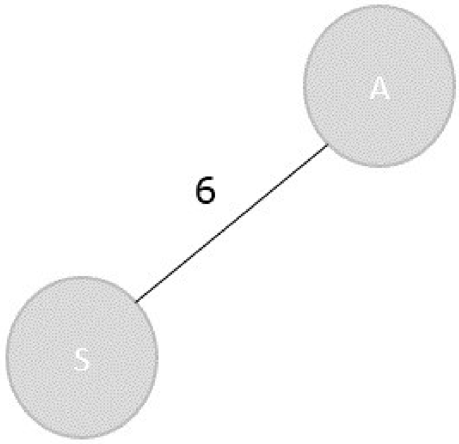
S → A = 6

S → D = 8

S → E = 7

Vertex	S	A	B	C
Distance	0	6	∞	∞

Visited = {S, A}



Step 3

There are two vertices visited in the visited array, therefore, the adjacent vertices must be checked for both the visited vertices.

Vertex S has two more adjacent vertices to be visited yet: D and E. Vertex A has one adjacent vertex B.

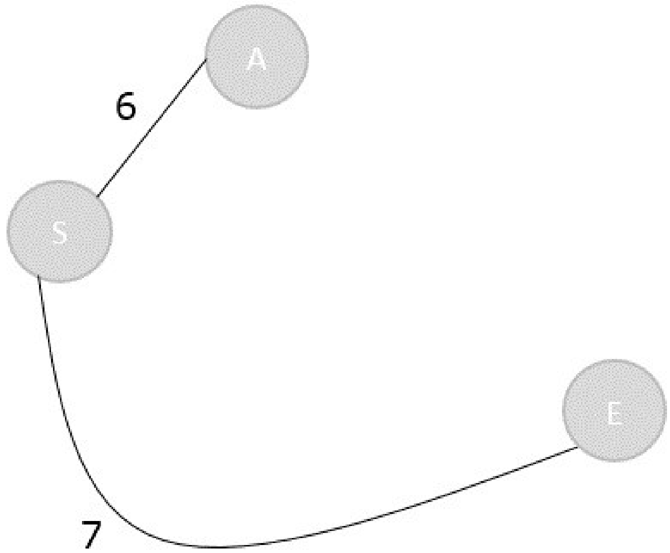
Calculate the distances from S to D, E, B and select the minimum distance −

S → D = 8 and S → E = 7.

S → B = S → A + A → B = 6 + 9 = 15

Vertex	S	A	B	C
Distance	0	6	15	∞

Visited = {S, A, E}



Step 4

Calculate the distances of the adjacent vertices – S, A, E – of all the visited arrays and select the vertex with minimum distance.

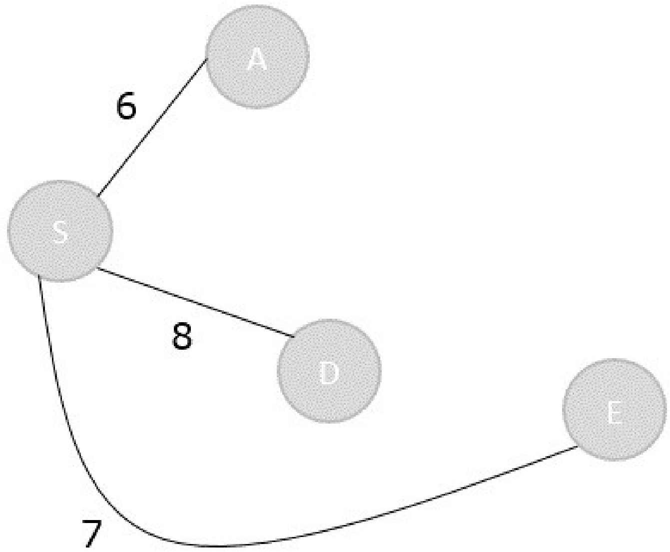
S → D = 8

S → B = 15

S → C = S → E + E → C = 7 + 5 = 12

Vertex	S	A	B	C
Distance	0	6	15	12

Visited = {S, A, E, D}



Step 5

Recalculate the distances of unvisited vertices and if the distances minimum than existing distance is found, replace the value in the distance array.

S → C = S → E + E → C = 7 + 5 = 12

S → C = S → D + D → C = 8 + 3 = 11

dist[C] = minimum (12, 11) = 11

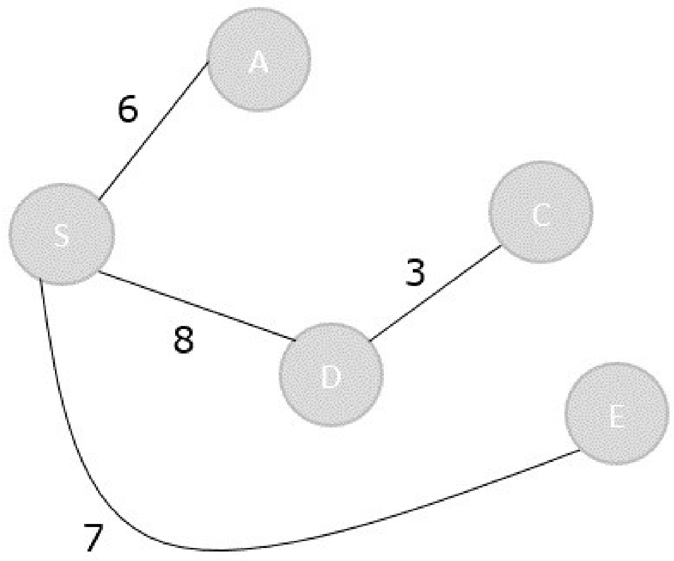
S → B = S → A + A → B = 6 + 9 = 15

S → B = S → D + D → C + C → B = 8 + 3 + 12 = 23

dist[B] = minimum (15,23) = 15

Vertex	S	A	B	C
Distance	0	6	15	11

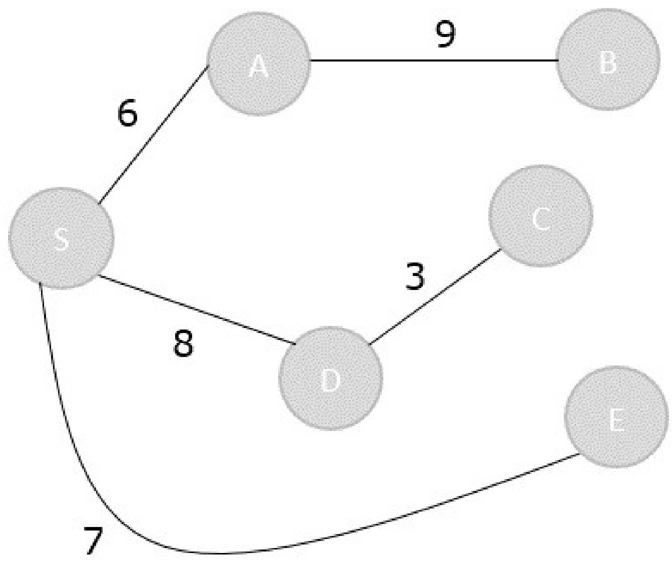
Visited = { S, A, E, D, C}



Step 6

The remaining unvisited vertex in the graph is B with the minimum distance 15, is added to the output spanning tree.

Visited = {S, A, E, D, C, B}



The shortest path spanning tree is obtained as an output using the dijkstra's algorithm.