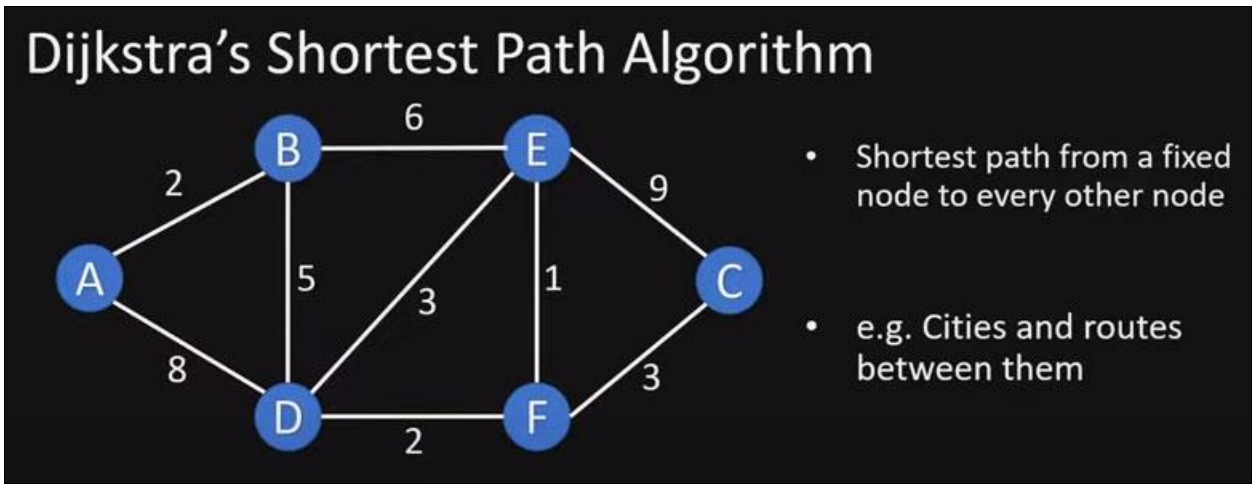

Dijkstra's Shortest Path Algorithm
A
B
C
D
E
F
2
6
9
5
3
1
8
3
2
Shortest path from a fixed node to every other node
e.g. Cities and routes between them

- Calculate the shortest path from A to C

Step-1: Keep track of two list

- List of vertices unvisited
- List of vertices visited

Right now we haven't visited any vertex so

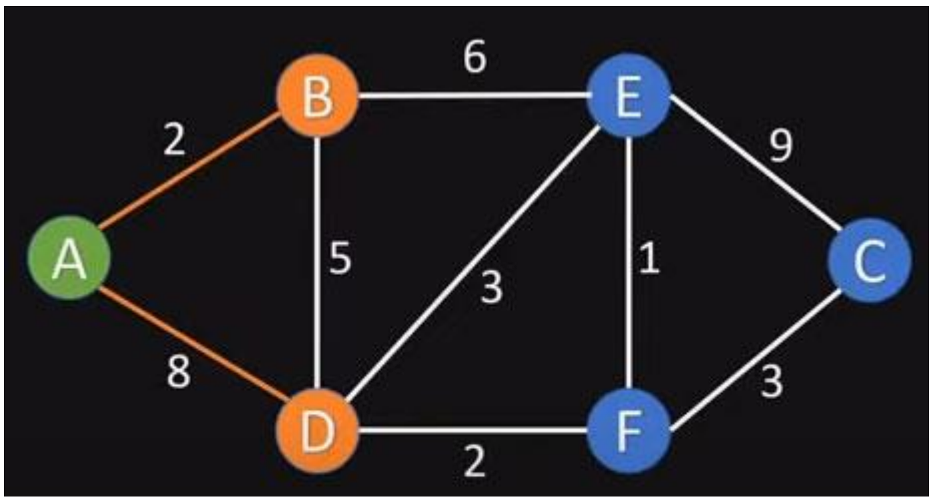
Visited Vertex [ ] **Unvisited Vertex [A, B, C, D, E, F]**

Step-2: Assign to all vertices a tentative distance value, for that create a table and initialize the shortest distance with the ∞**.** The shortest distance of source vertex i.e. A would be zero (0).

Vertex	Shortest Distance	Previous Vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	

Now start calculating from the source vertex i.e. A

Check the distance of neighbor vertex of source A, in the above graph A's unvisited neighbor vertices are B and D.



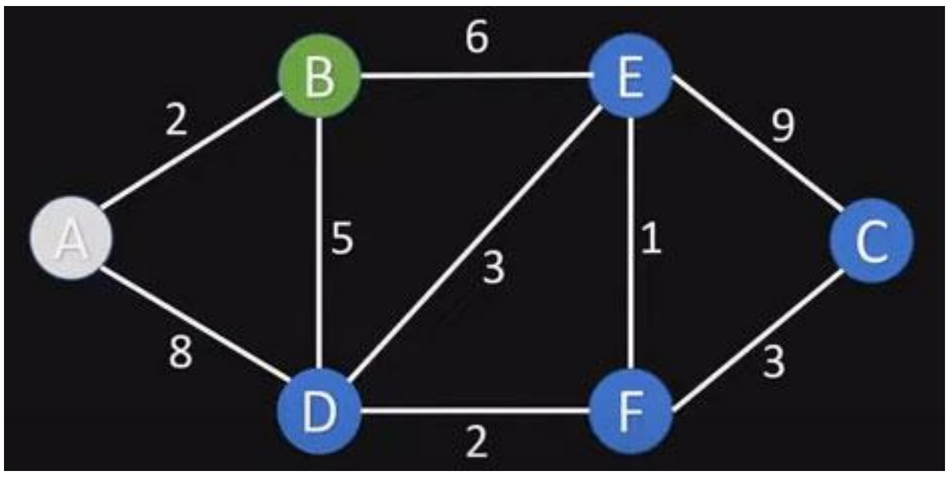
Update the shortest distance, if new distance is shorter than old distance

Vertex	Shortest Distance	Previous Vertex
A	0	
B	**2**	**A**
C	∞	
D	**8**	**A**
E	∞	
F	∞	

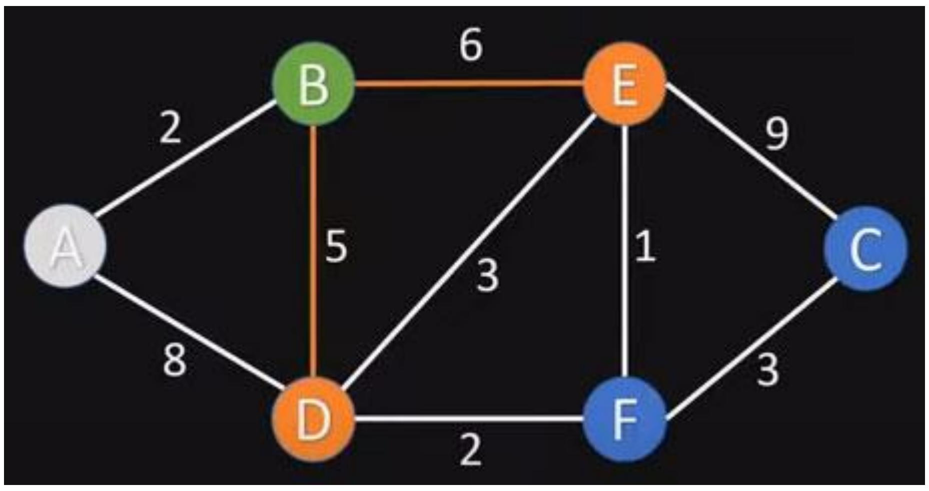
Now we have visited the vertex A, so update in the tracking lists

Visited Vertex [**A**] **Unvisited Vertex** [**B, C, D, E, F**]

Step-3: To choose the next vertex from the unvisited having a minimum cost/weightage, according to above table vertex B has '2' weightage, so next visited vertex would be 'B'.

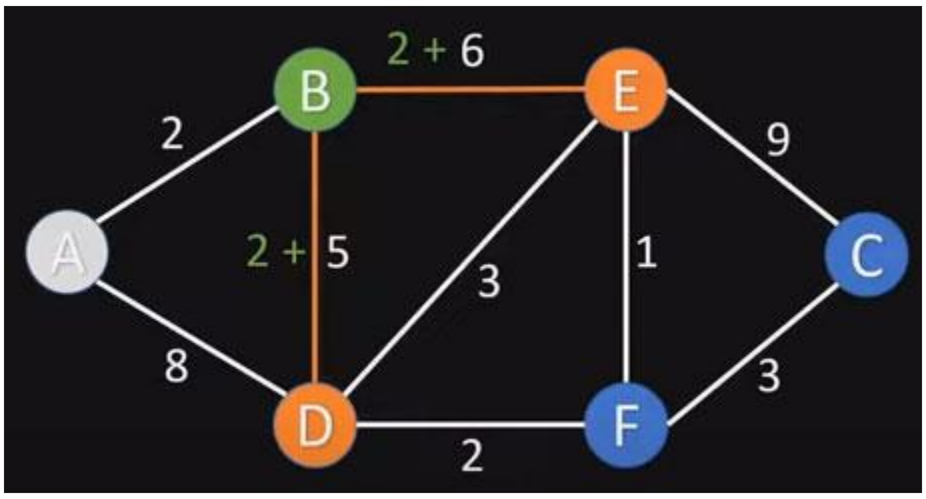


Current visited vertex is 'B' and its unvisited neighbors are 'E' and 'D'.



Step-4: Now calculate the distance for the above neighbors' i.e. 'E' and 'D', remember we add the weightage of current visited vertex with each neighbor vertices to get the total shortage distance between two vertices.

Our current vertex is 'B' and according to the above table 'B' has shortage distance value of '2', now add it to each neighbor's weightage.



Update the table and tracking list accordingly, we get.

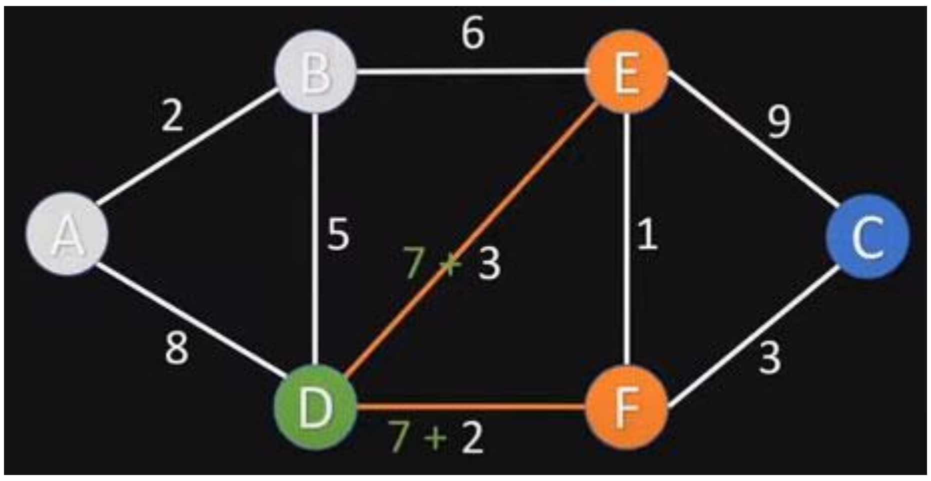
Vertex	Shortest Distance	Previous Vertex
A	0	
B	2	A
C	∞	
D	7	B
E	8	B
F	∞	

If you observe that previously vertex 'D' had value of '8' now we updated with the new value i.e. '7', which is minimum from the previous value and also updated previous vertex with 'B'.

Visited Vertex [A, B] **Unvisited Vertex [C, D, E, F]**

Repeat the step-3 to step-4, for the next vertex

Step-3: To choose the next vertex from the unvisited having a minimum cost/weightage, according to above table vertex D has ‘7’ weightage, so next visited vertex would be ‘D’.



Step-4: Now calculate the distance for the above neighbors’ i.e. ‘E’ and ‘F’, remember we add the weightage of current visited vertex with each neighbor vertices to get the total shortage distance between two vertices.

Our current vertex is ‘D’ and according to the above table ‘D’ has shortage distance value of ‘7’, now add it to each neighbor’s weightage.

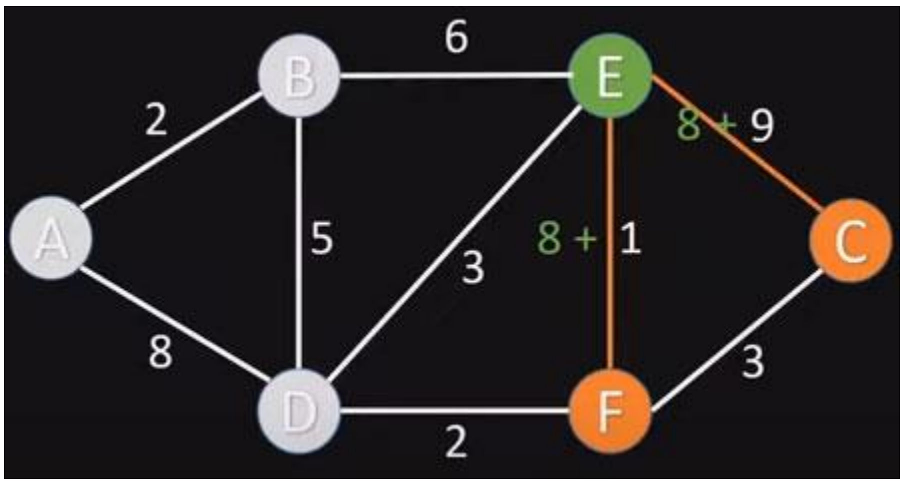
Update the table and tracking list accordingly, we get.

Vertex	Shortest Distance	Previous Vertex
A	0	
B	2	A
C	∞	
D	7	B
E	8	B
F	9	D

If you observe that vertex ‘E’ has value of ‘8’ and current calculated distance is ‘10’ which is greater than the previous value so we not update the value of ‘E’.

Visited Vertex [A, B, D] **Unvisited Vertex [C, E, F]**

Repeat the step-3 to step-4, for the next vertex



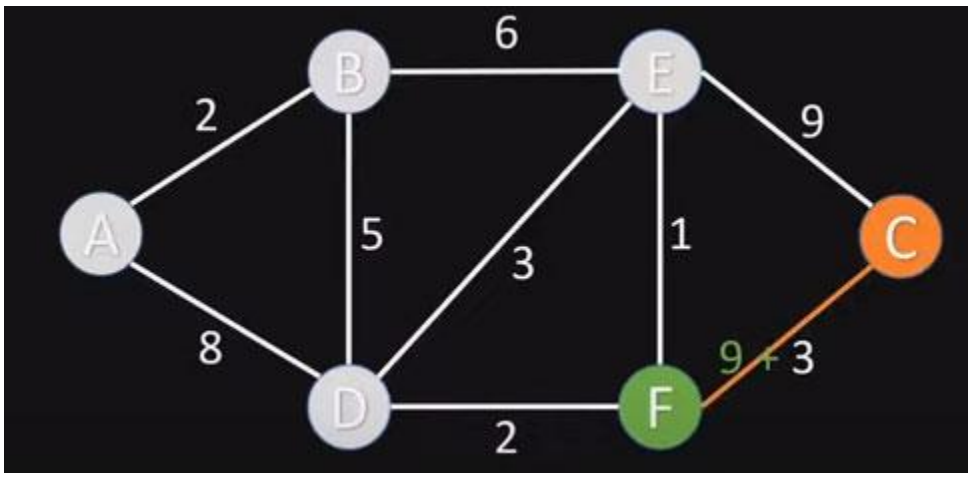
Update the table and tracking list accordingly, we get.

Vertex	Shortest Distance	Previous Vertex
A	0	
B	2	A
C	17	E
D	7	B
E	8	B
F	9	D

If you observe that vertex 'F' has the same value of '8' already so we don't update the value of 'F'.

Visited Vertex [A, B, D, E] Unvisited Vertex [C, F]

Repeat the step-3 to step-4, for the next vertex



We have only one distance vertex i.e. ‘C’ and it has value of 17, but from vertex ‘F’ to ‘C’ its distance calculation give us value of ‘12’, so we update the value of vertex ‘C’ with ‘12’.

Vertex	Shortest Distance	Previous Vertex
A	0	
B	2	A
C	12	F
D	7	B
E	8	B
F	9	D

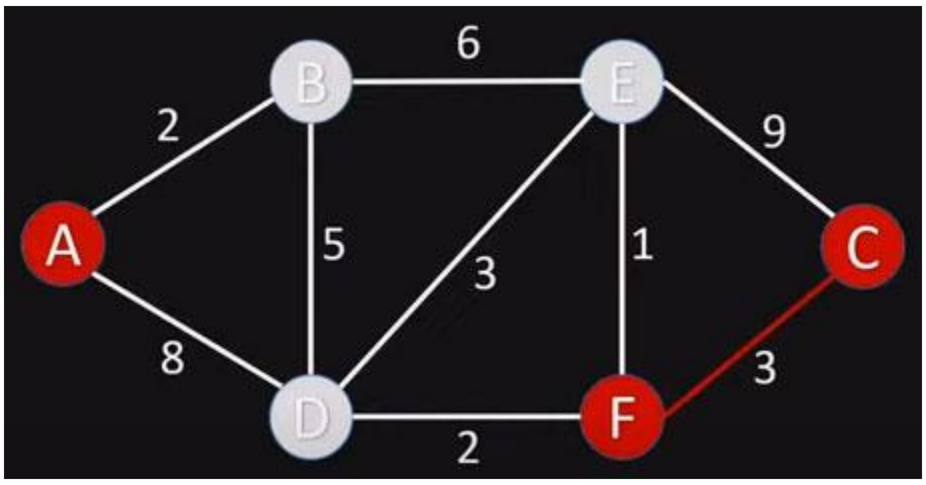
Visited Vertex [A, B, D, E, F] **Unvisited Vertex [C]**

If the final vertex left in the unvisited vertex list, then the algorithm stops calculating and include the final vertex i.e. ‘C’ in the list of visited vertices.

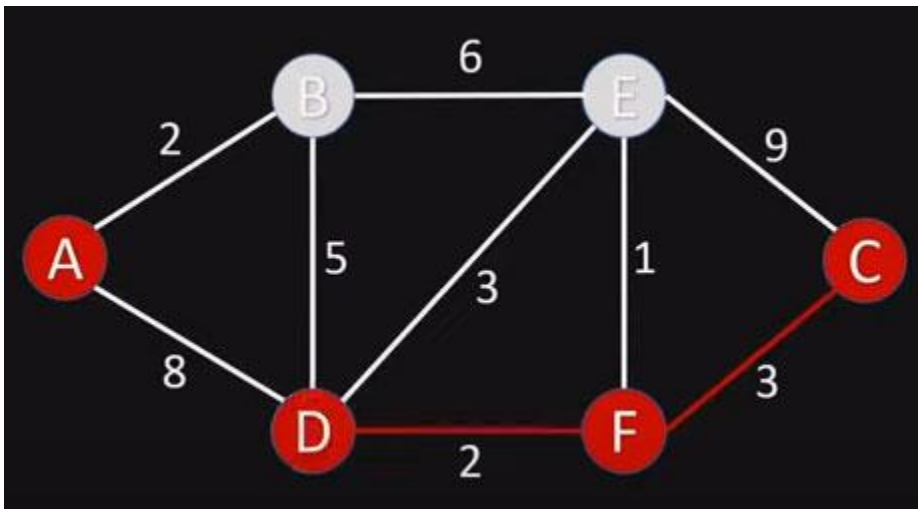
Visited Vertex [A, B, D, E, F, C] **Unvisited Vertex [ ]**

The final step is to re-construct the path according to the values given in the above table. We were finding the shortest path between vertex ‘A’ and ‘C’, now move backward from vertex ‘C’ to ‘A’

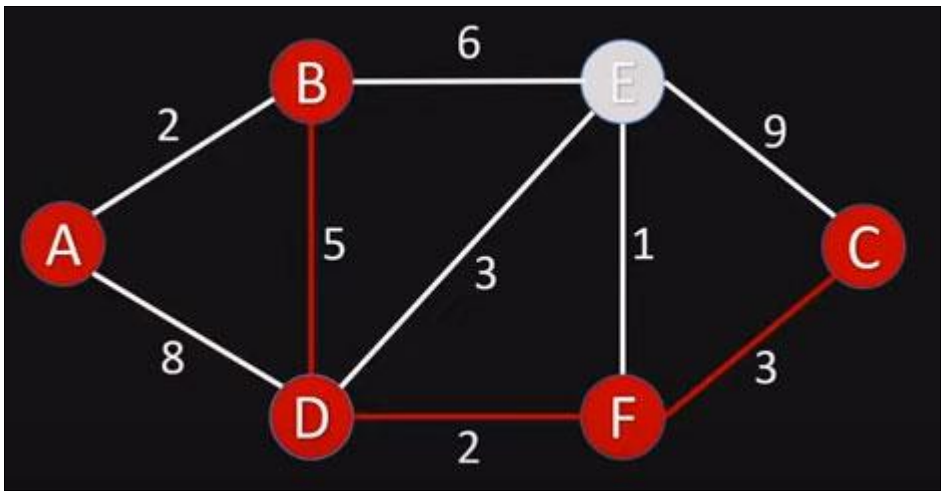
So starting from vertex ‘C’ and in the table its pervious vertex is ‘F’



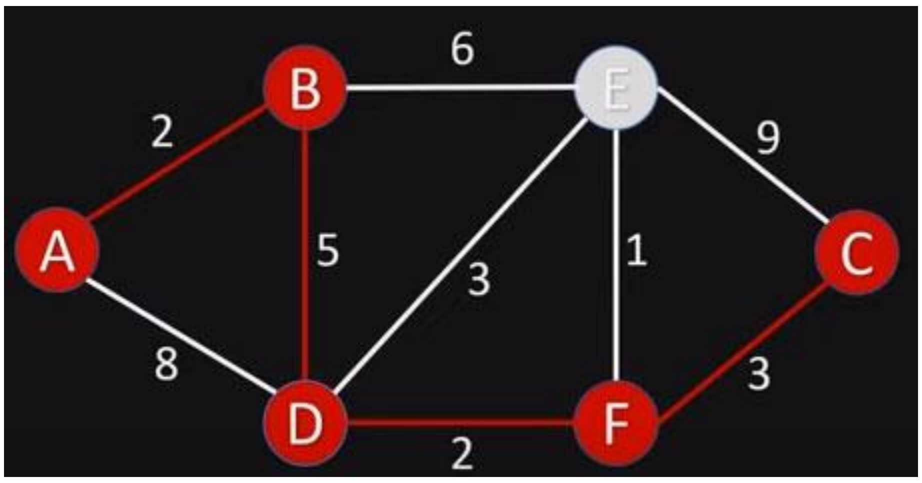
Now in the table vertex ‘F’ has previous vertex ‘D’



Vertex ‘D’ has previous vertex ‘B’



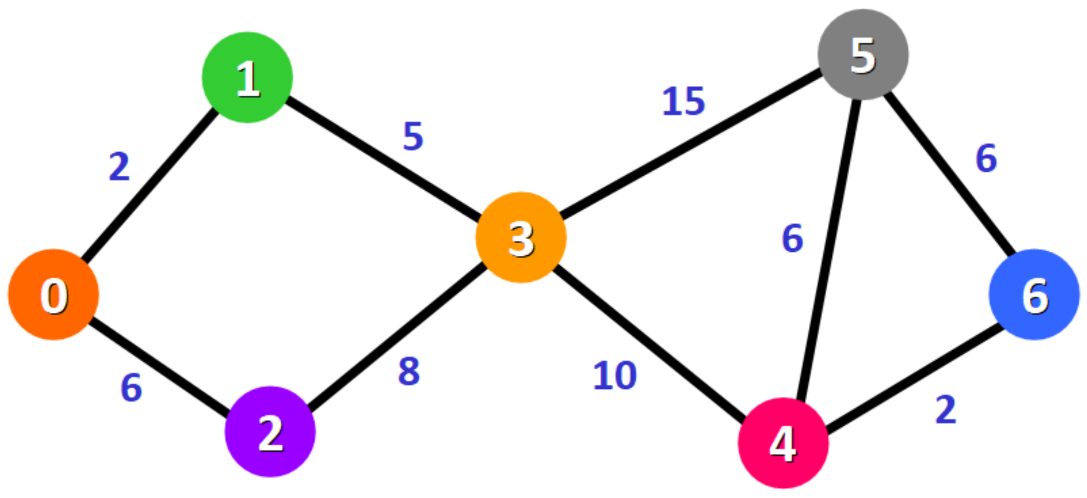
Vertex ‘B’ has previous vertex ‘A’



Hence, the shortest path weightage/cost from ‘A’ to ‘C’ is

$$\{2 + 5 + 2 + 3\} = 12$$

Let’s take another example:



Calculate the shortest path from vertex ‘0’ to vertex ‘6’ using Dijkstra’s Algorithm (CLASS TASK)