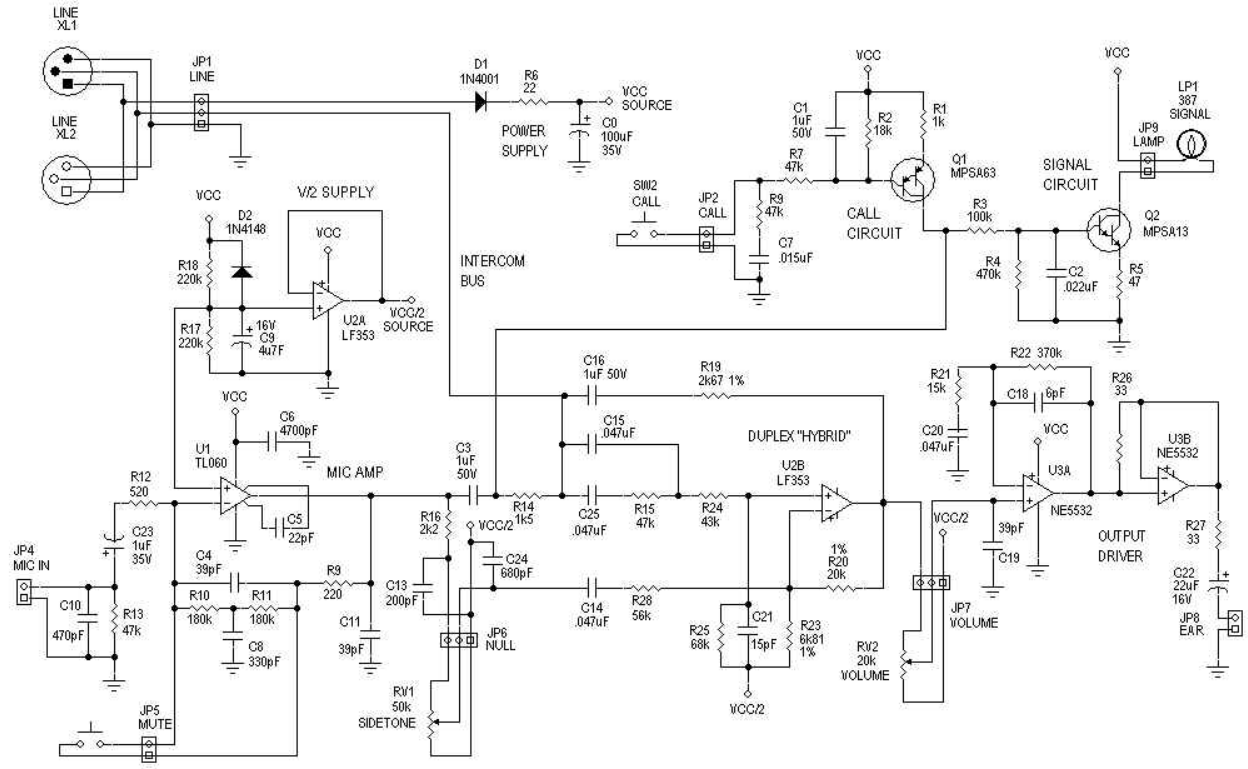
Shortest Paths

Shortest Path

- Given a weighted directed graph, one common problem is finding the shortest path between two given vertices
- In a weighted graph, the length of a path is, the sum of the weights of each of the edge in that path

Applications

- One application is circuit design: the time it takes for a change in input to affect an output depends on the shortest path



Versatility and importance of shortest path algorithms across different domains

- **Shortest path algorithms are used in various fields and applications, including:**
  - **Navigation and Routing: Used in GPS systems to find the shortest route between locations.**
  - **Network Routing: Helps in optimizing data transmission across networks, such as in telecommunications and computer networks.**
  - **Transportation Planning: Used in logistics and supply chain management to minimize travel costs and times.**

Versatility and importance of shortest path algorithms across different domains

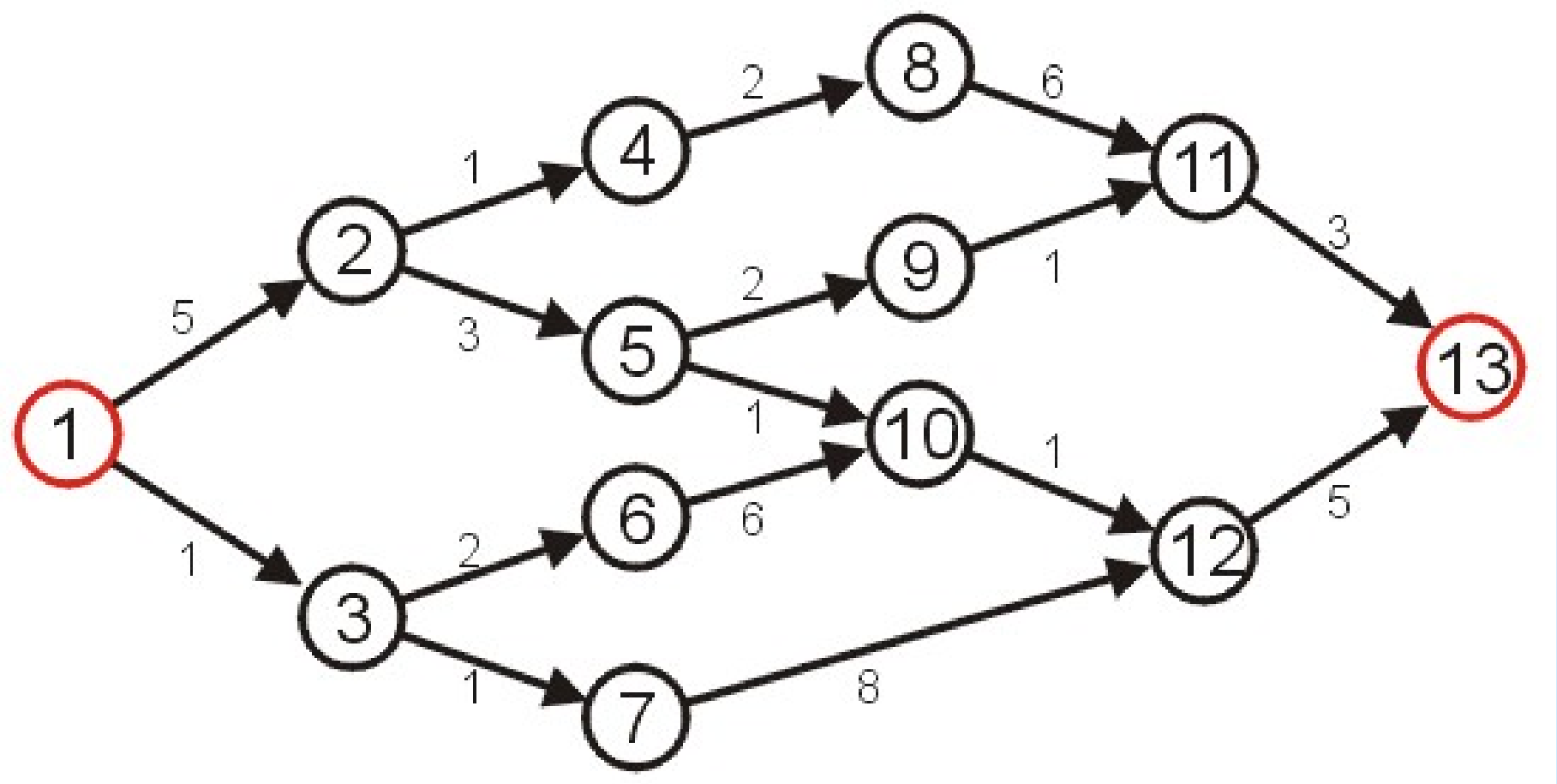
- **Urban Planning**: Aids in designing efficient road networks and public transportation systems.
- **Game Development**: Helps in character pathfinding and AI navigation in video games.
- **Robotics**: Used for route optimization in robotic navigation and automated vehicles.
- **Social Network Analysis**: Analyzes shortest paths in social graphs to understand connections and influence.

Versatility and importance of shortest path algorithms across different domains

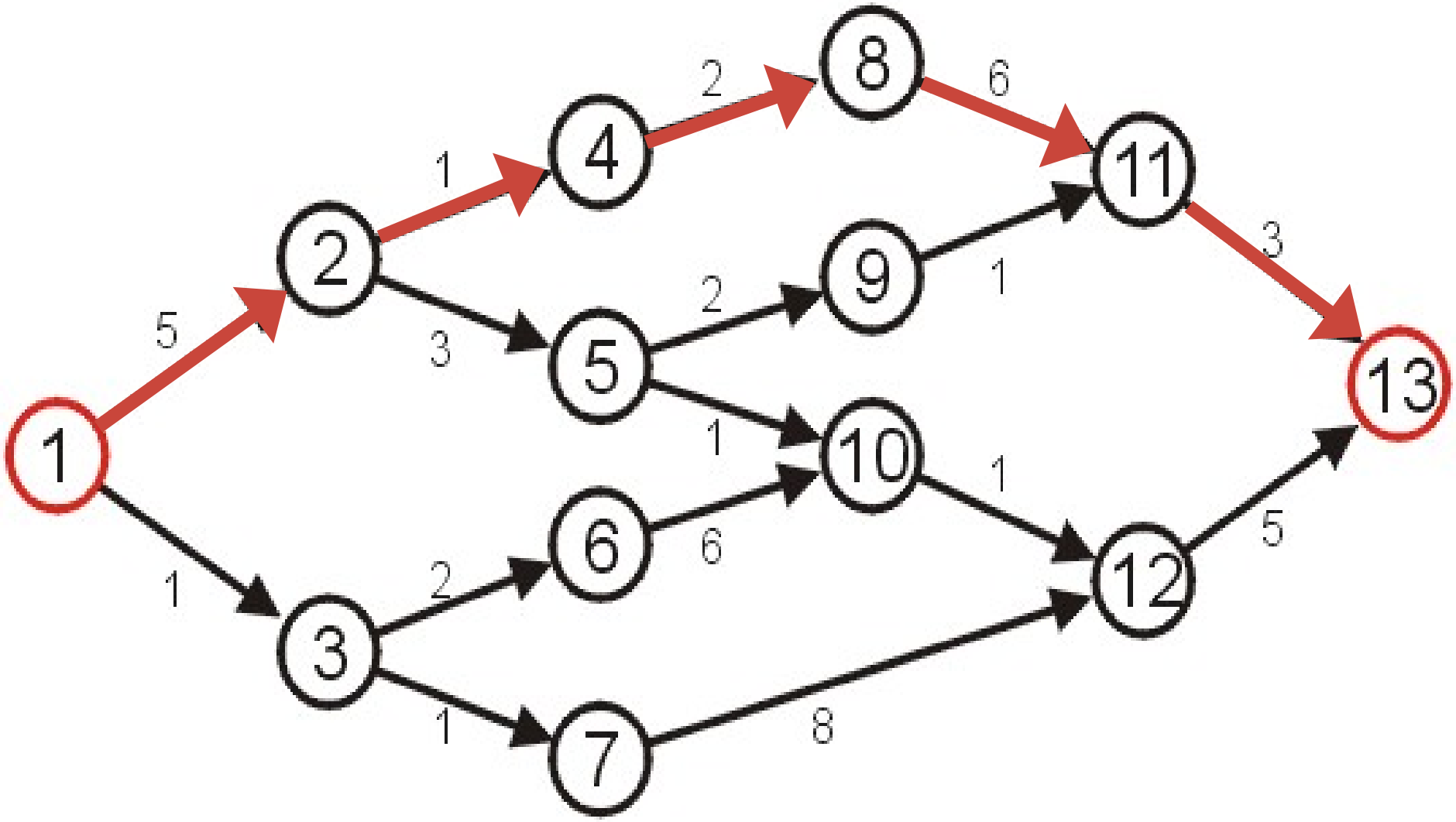
- **Geographical Information Systems (GIS): Assists in spatial analysis for mapping and environmental studies.**
- **Telecommunications: Optimizes signal routing in network design.**
- **Operations Research: Used in various optimization problems involving logistics, resource allocation, and scheduling.**

Shortest Path

- Given the graph below, suppose we wish to find the shortest path from vertex 1 to vertex 13

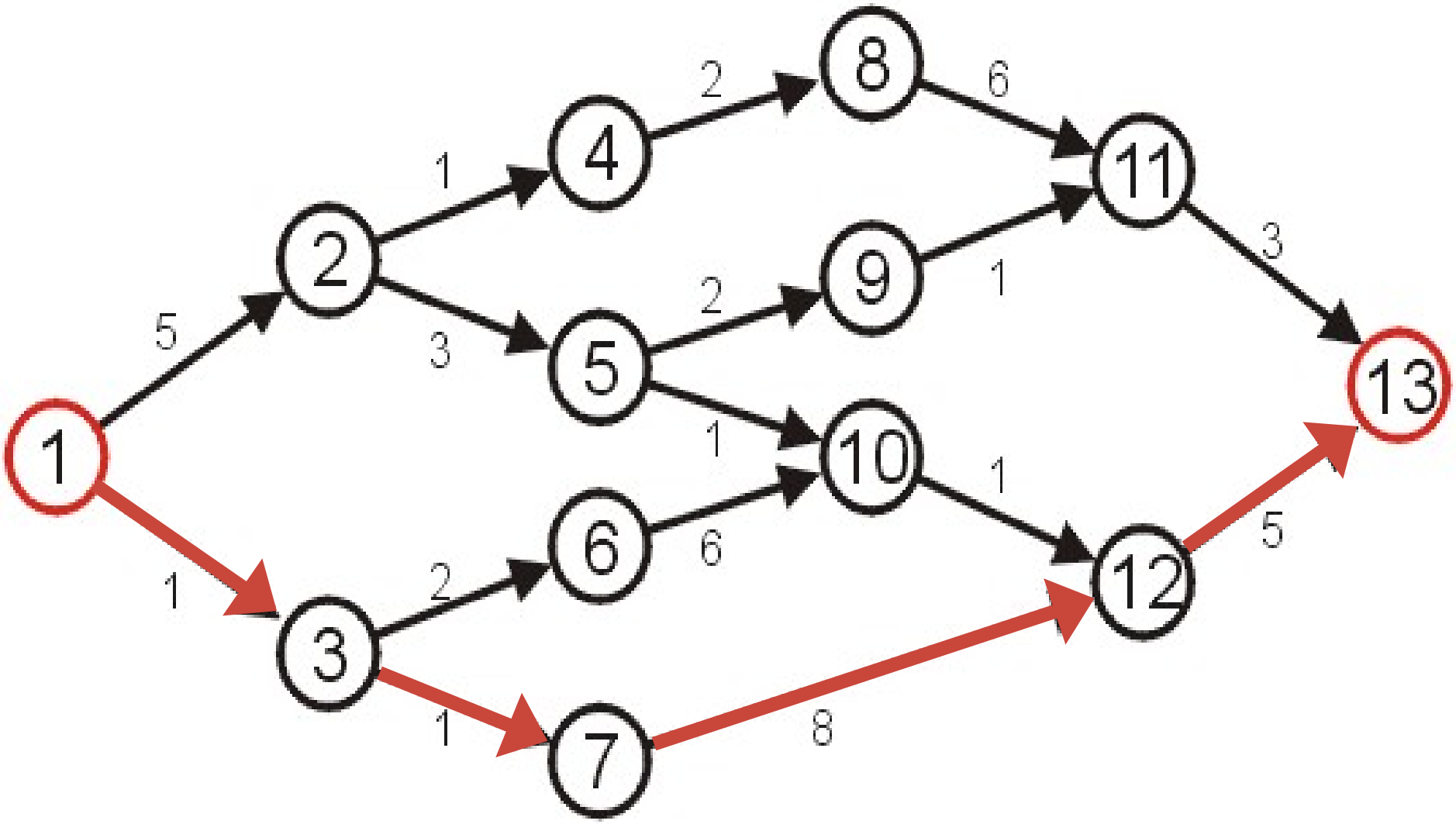


What is the cost of shortest path from 1 to 13?



Is it 5+1+2+6+3 = 17 ?

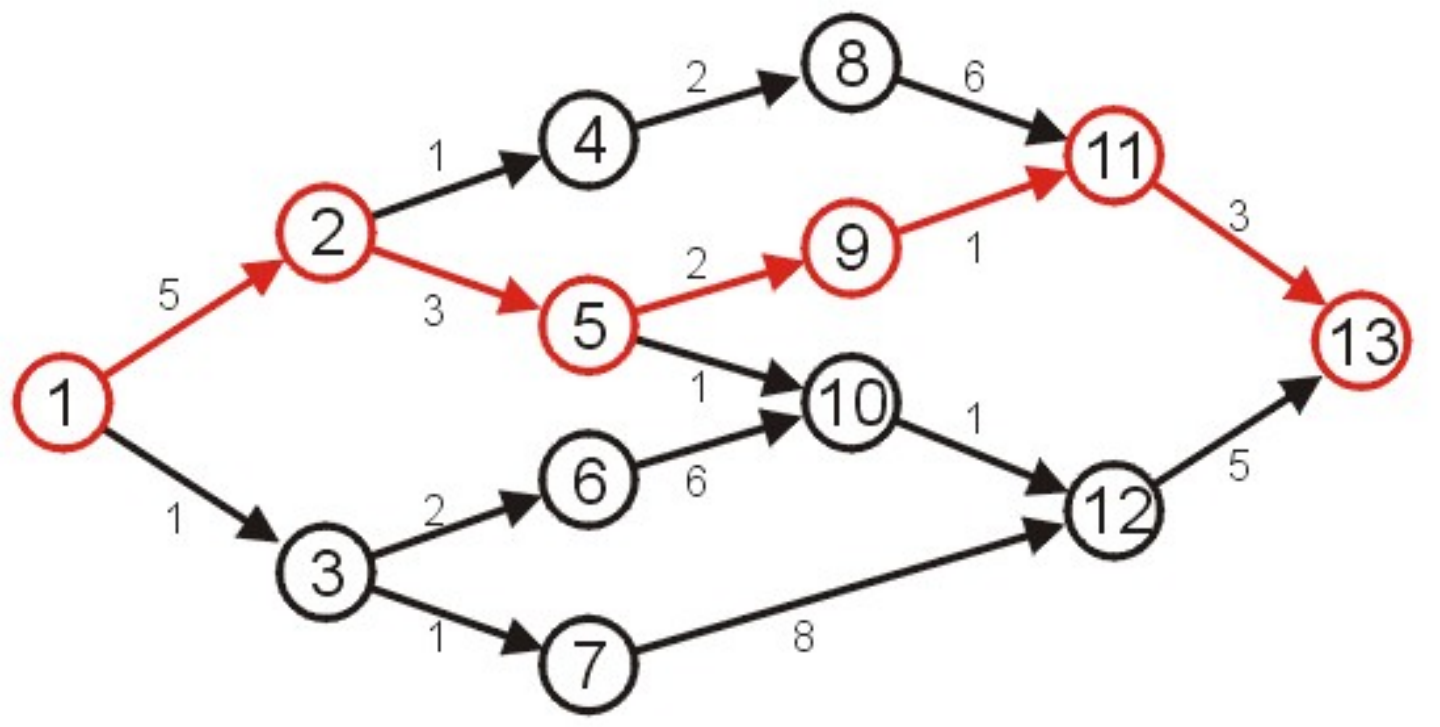
What is the cost of shortest path from 1 to 13



Is it 1+1+8+5 = 15 ?

Shortest Path

- After some consideration, we may determine that the shortest path is as follows, with length 14

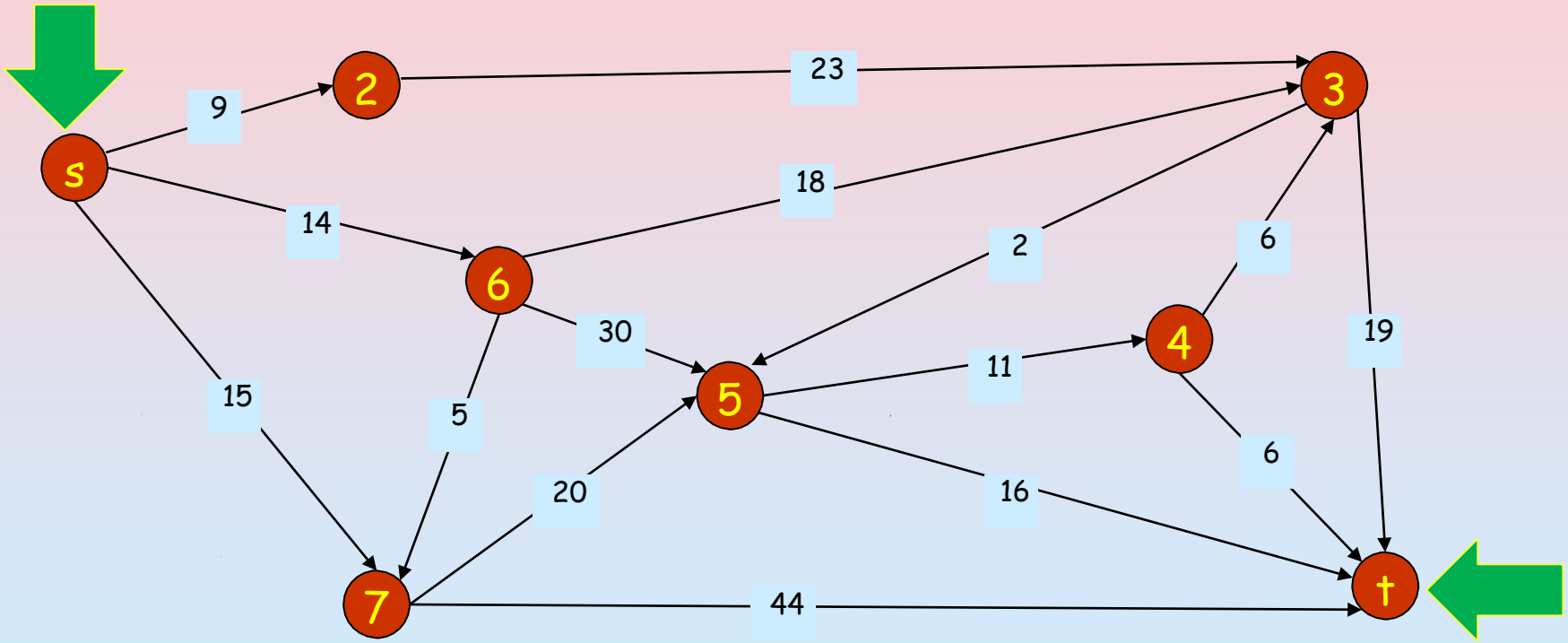


5+3+2+1+3 = 14

- Other paths exists, but they are longer

Shortest Path Example

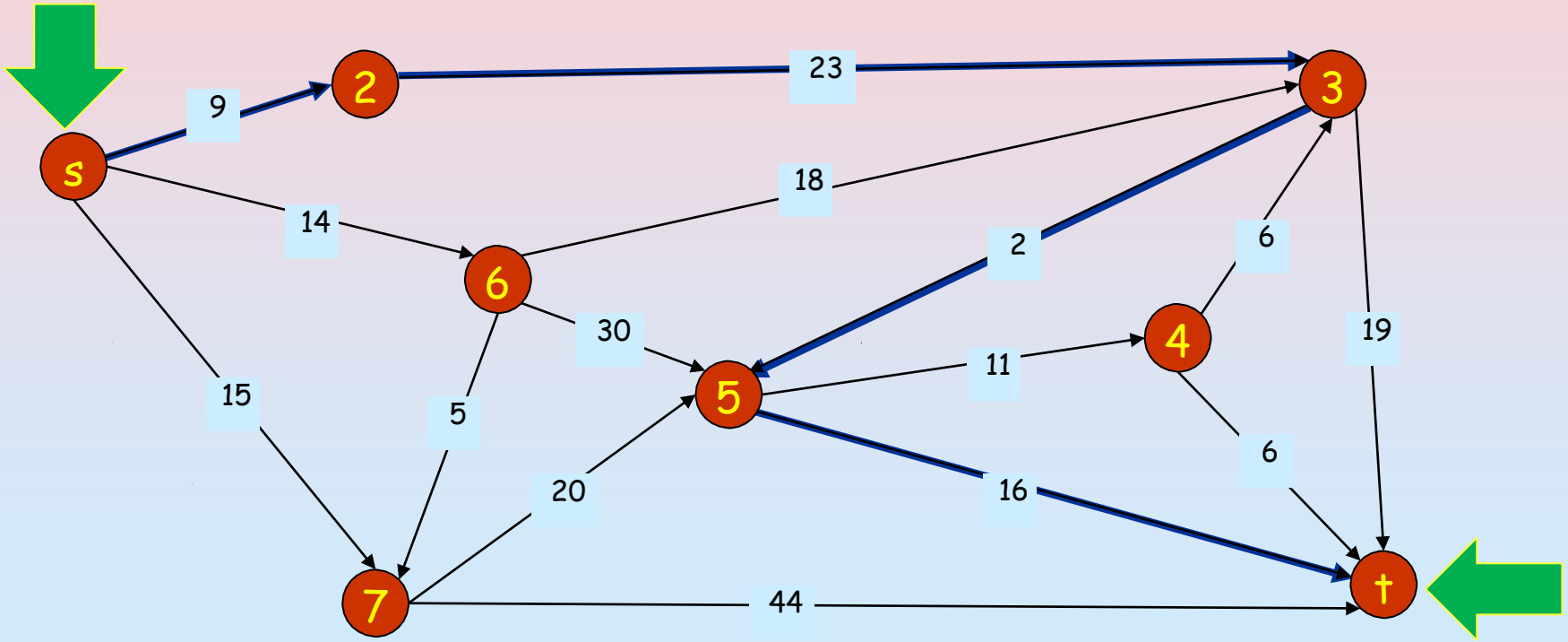
- Given:
  - Weighted Directed graph G = (V, E).
  - Source s, destination t.



Find shortest directed path from *s* to *t*.

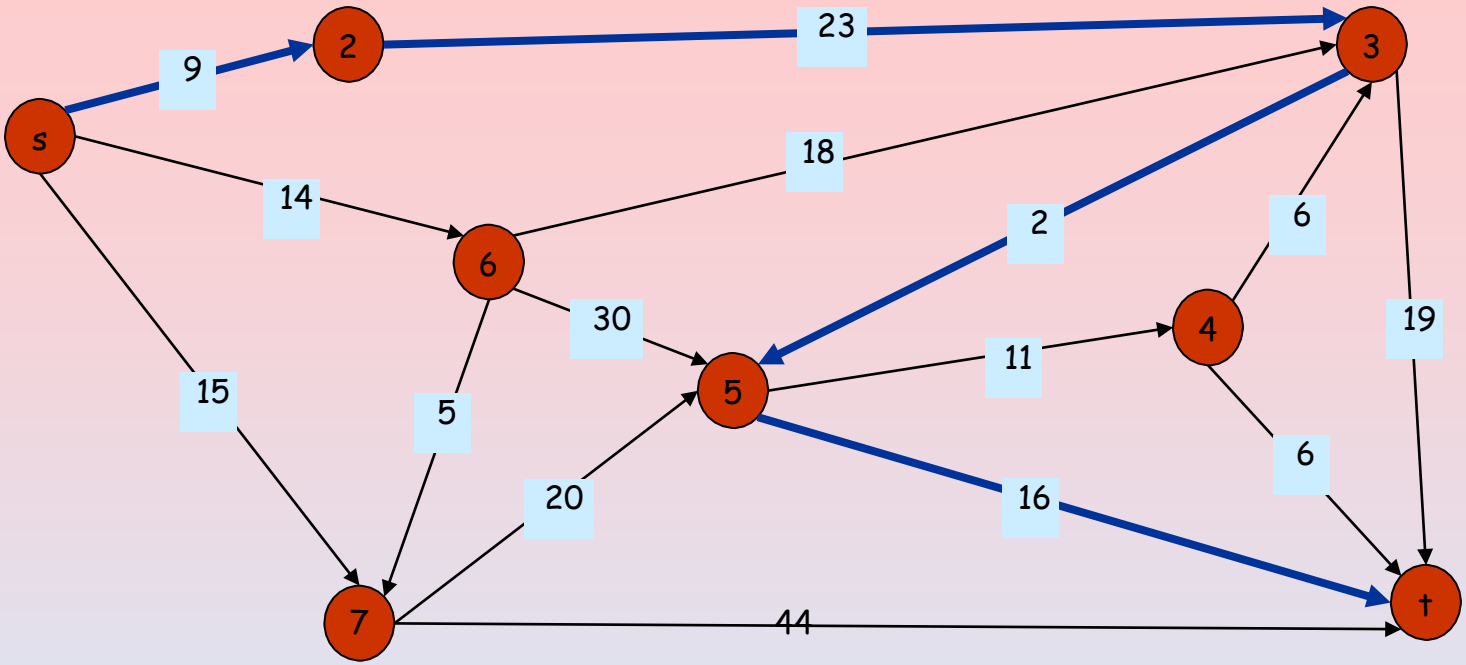
Shortest Path Example

- Given:
  - Weighted Directed graph G = (V, E).
  - Source s, destination t.
- Shortest directed path from *s* to *t*.



Cost of path s-2-3-5-t = 9+23+2+16=50

Discussion Items



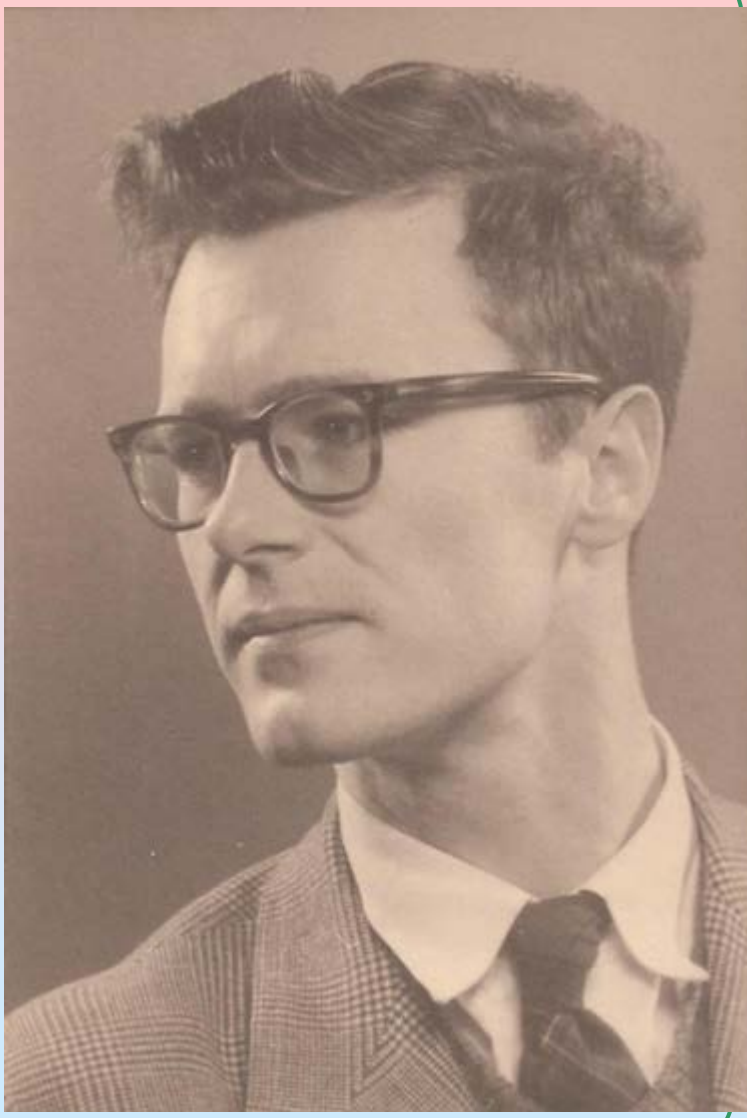
- How many possible paths are there from *s* to *t*?
- Can we safely ignore cycles? If so, how?
- Any suggestions on how to reduce the set of possibilities?

Key Observation

- A key observation is that if the shortest path contains the node *v*, then:
  - It will only contain v once, as any cycles will only add to the length.
  - The path from s to v must be the shortest path to *v* from s.
  - The path from v to t must be the shortest path to t from v.
- Thus, if we can determine the shortest path to all other vertices that are incident to the target vertex we can easily compute the shortest path.
  - Implies a set of sub-problems on the graph with the target vertex removed.

Dijkstra's Algorithm

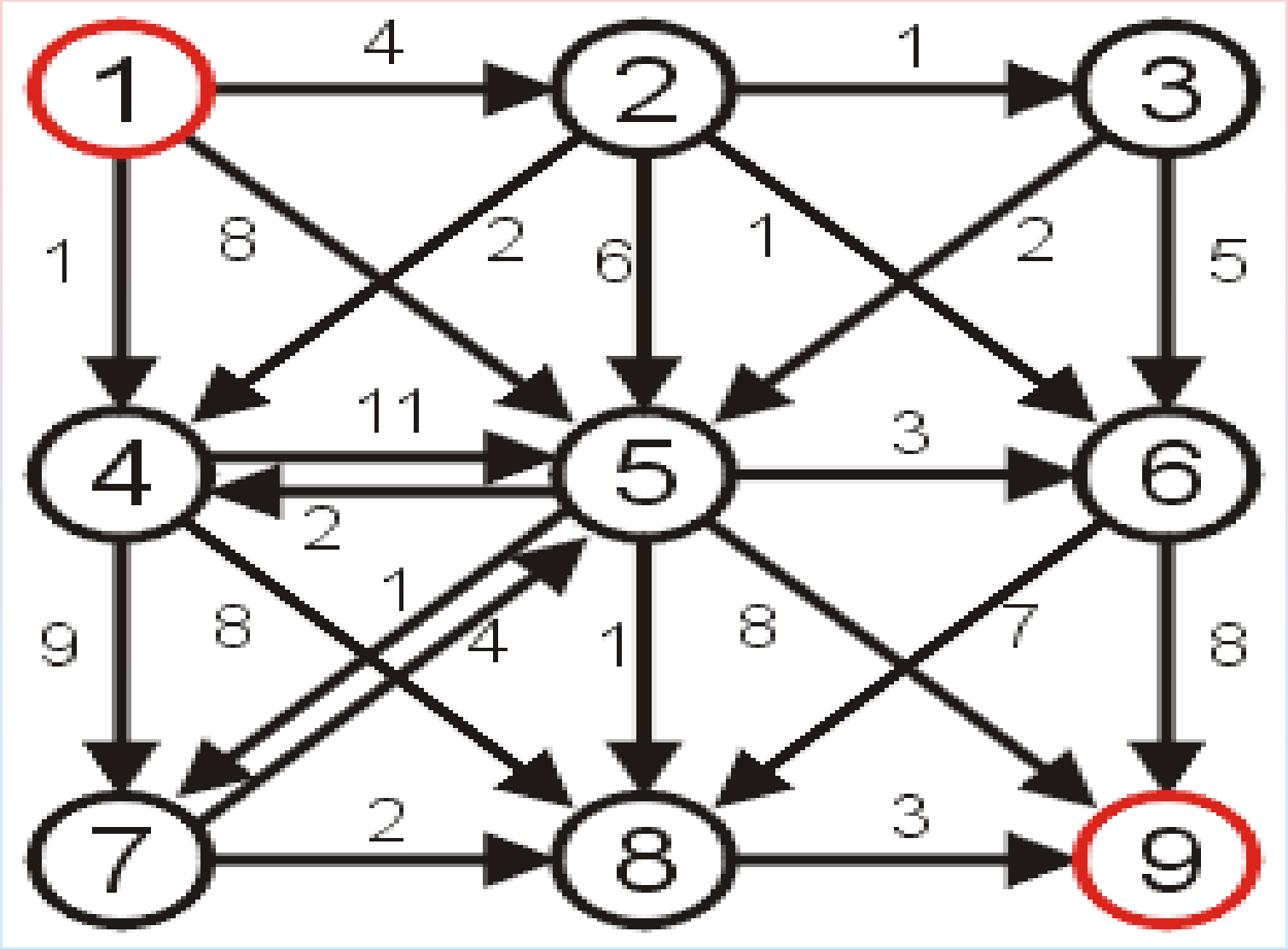
Edsger W. Dijkstra



- Works when all the weights are positive.
- Provides the shortest paths from a source to all other vertices in the graph.
  - Can be terminated early once the shortest path to *t* is found if desired.

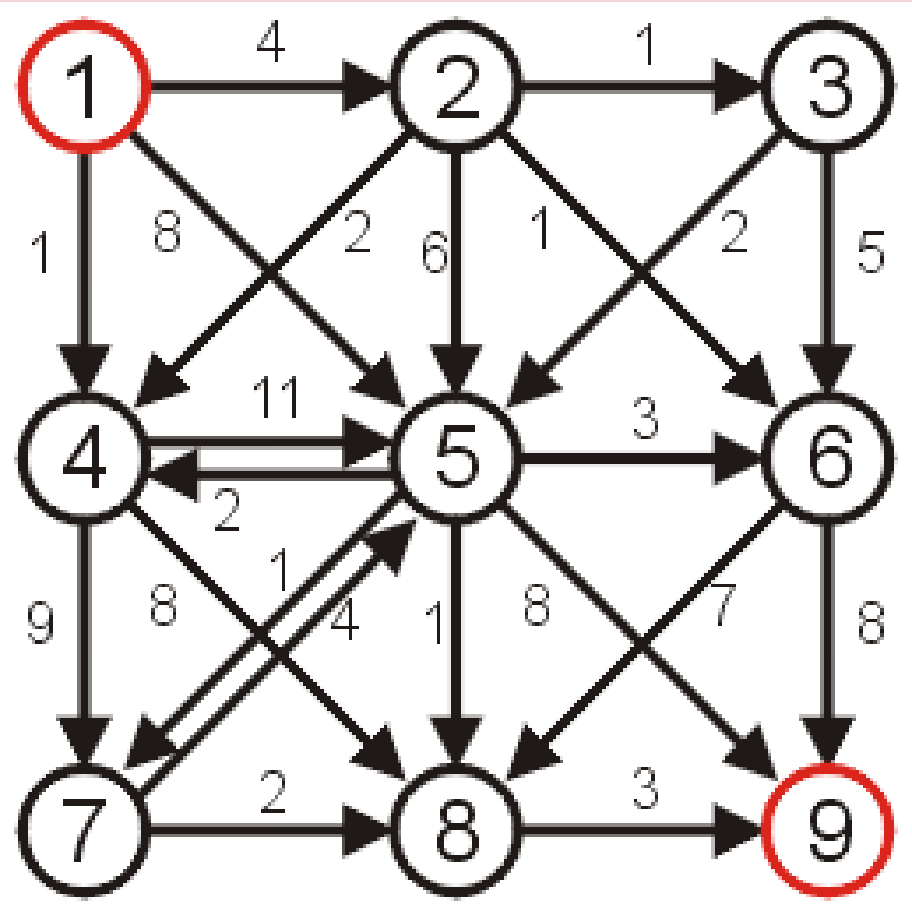
Shortest Path

- Consider the following graph with positive weights and cycles.



Dijkstra's Algorithm

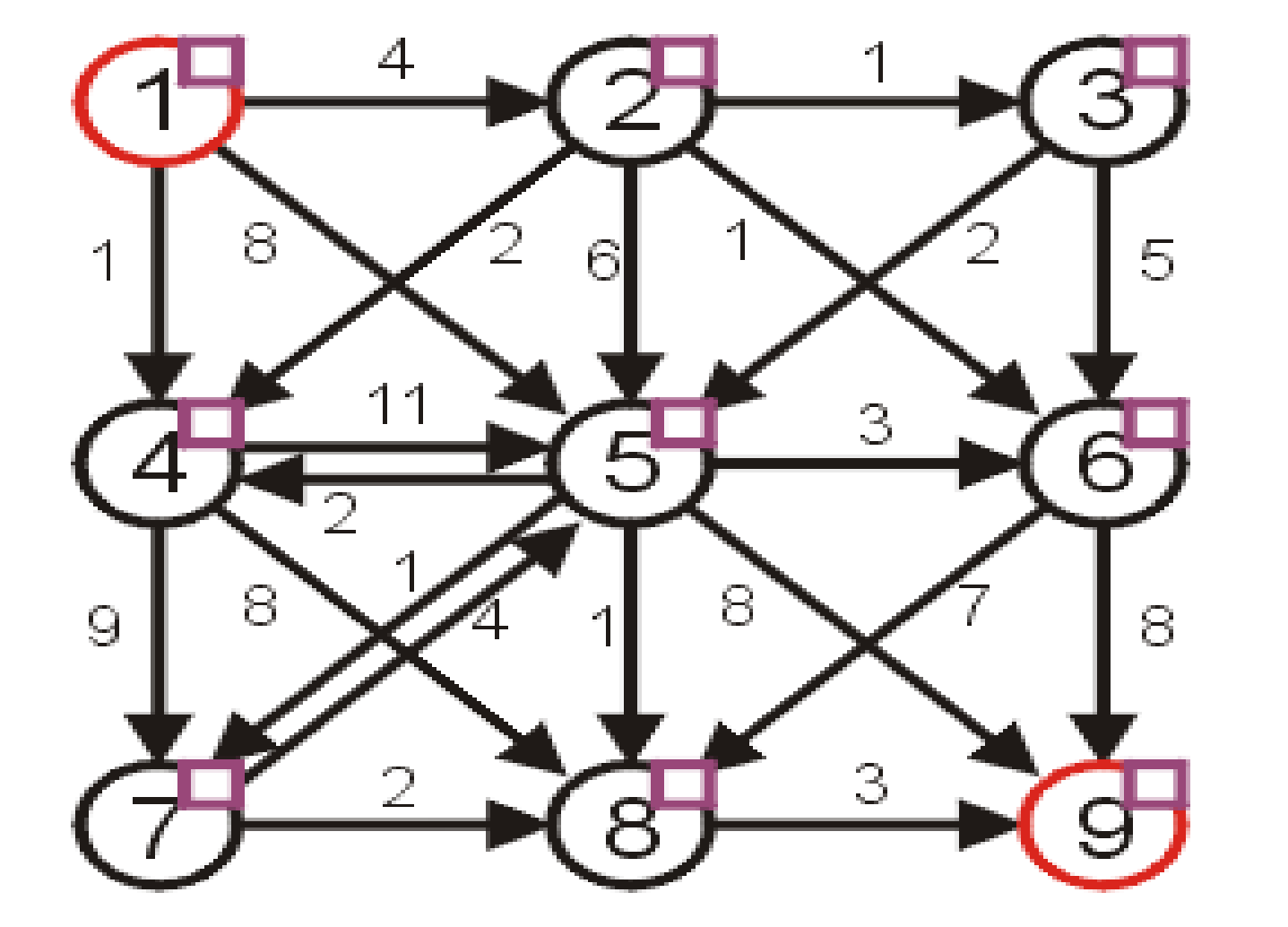
- A first attempt at solving this problem might require an array of Boolean values, all initially false, that indicate whether we have found a path from the source.



1	F
2	F
3	F
4	F
5	F
6	F
7	F
8	F
9	F

Dijkstra's Algorithm

- Graphically, we will denote this with **check boxes** next to each of the vertices (**initially unchecked**)

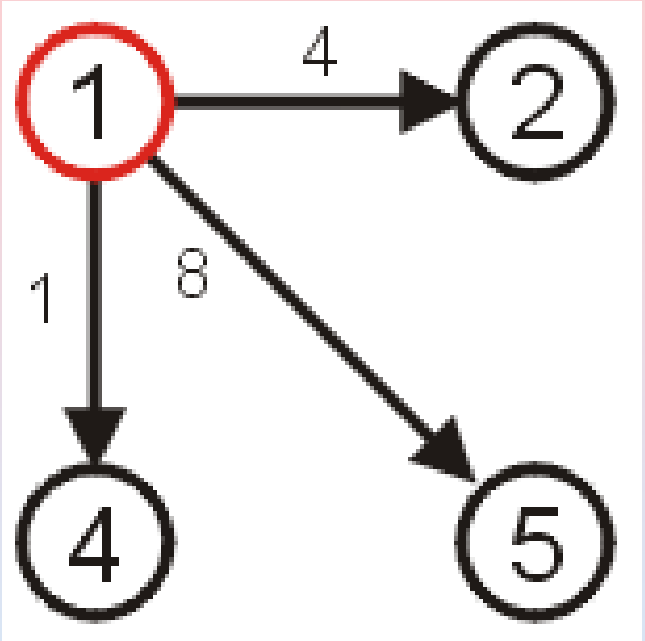


Dijkstra's Algorithm

- We will work **bottom up**.
  - Note that if the starting vertex has any adjacent edges, then there will be one vertex that is the shortest distance from the starting vertex. This is the shortest reachable vertex of the graph.
- We will then try to **extend any *existing* paths** to **new vertices**.
- Initially, we will start with the path of length 0
  - this is the trivial **path from vertex 1 to itself**

Dijkstra's Algorithm

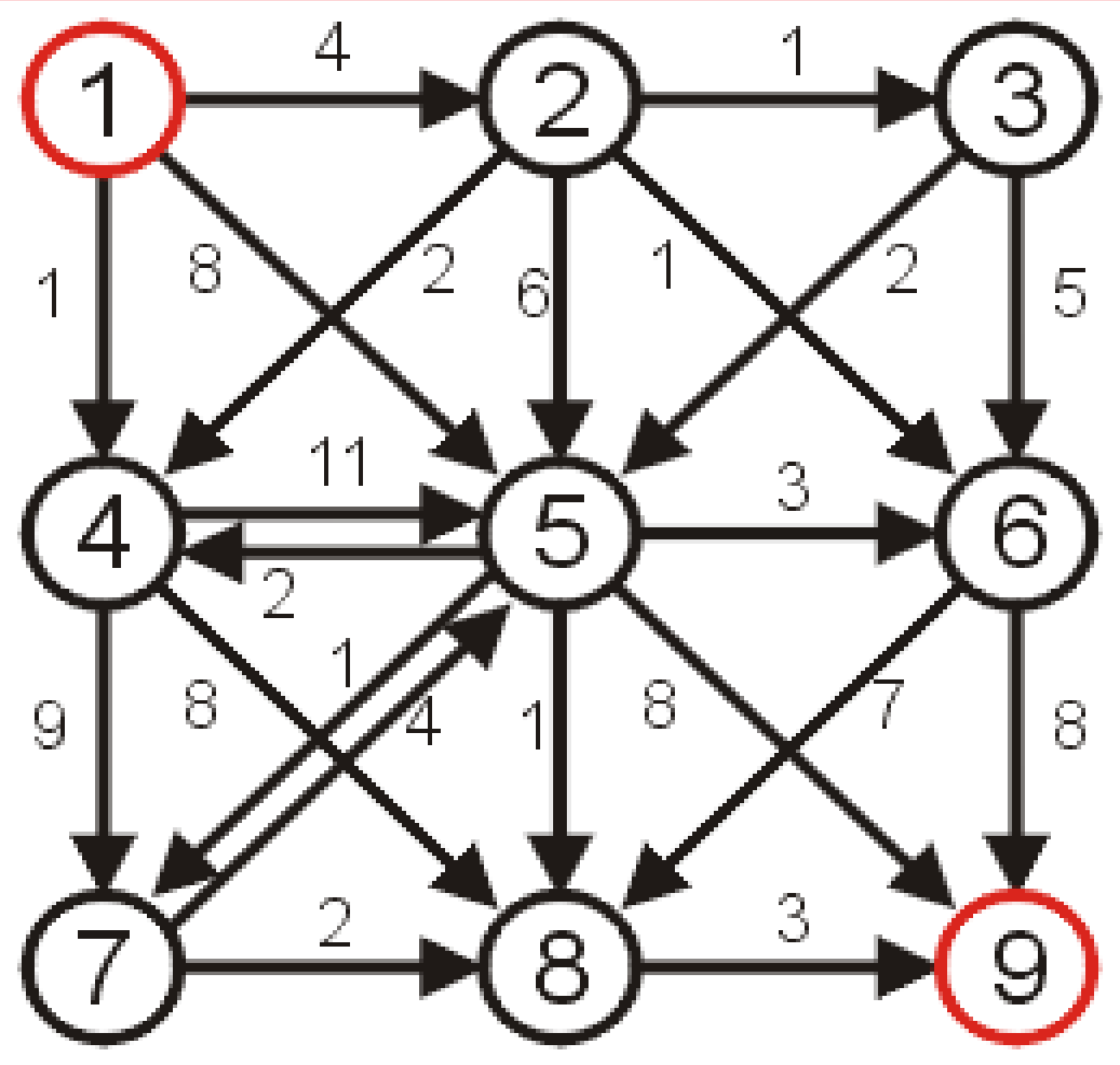
- If we now extend this path, we should consider the paths



  - (1, 2) length 4
  - (1, 5) length 8
  - (1, 4) length 1

The *shortest* path so far is (1, 4) which is of length 1.

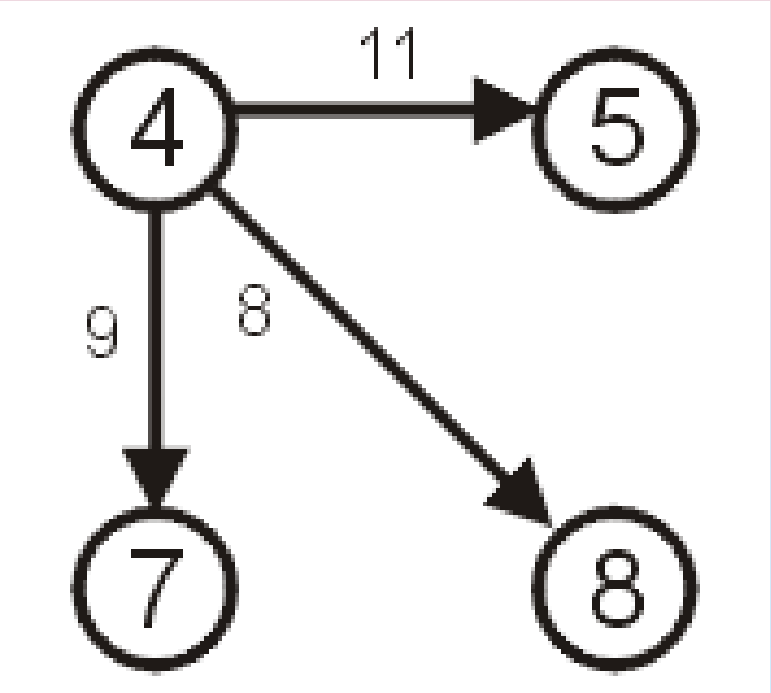
Dijkstra's Algorithm



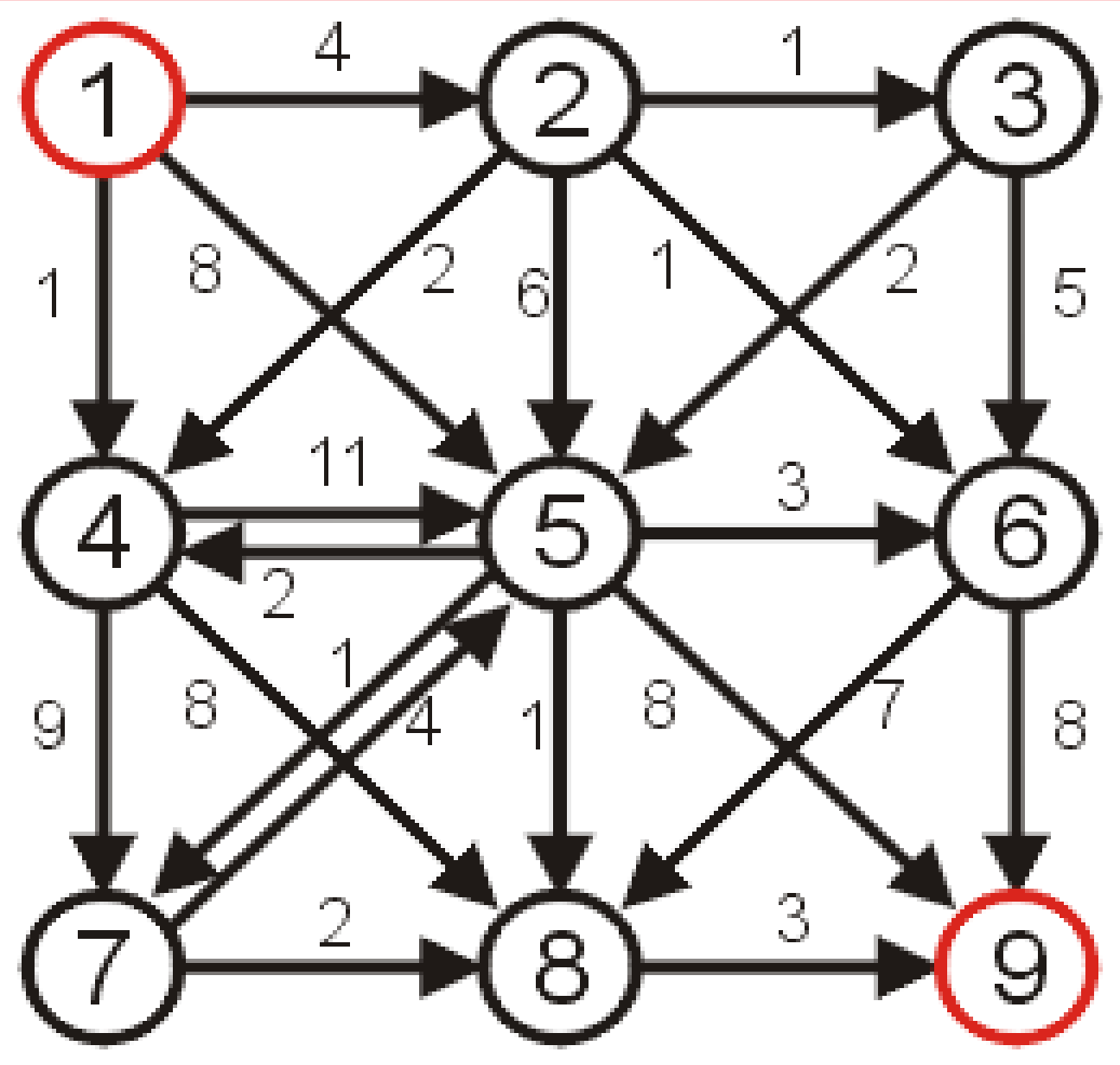
Dijkstra's Algorithm

- **Thus, if we now examine vertex 4, we may deduce that there exist the following paths:**

  - **(1, 4, 5) length 12**
  - **(1, 4, 8) length 9**
  - **(1, 4, 7) length 10**



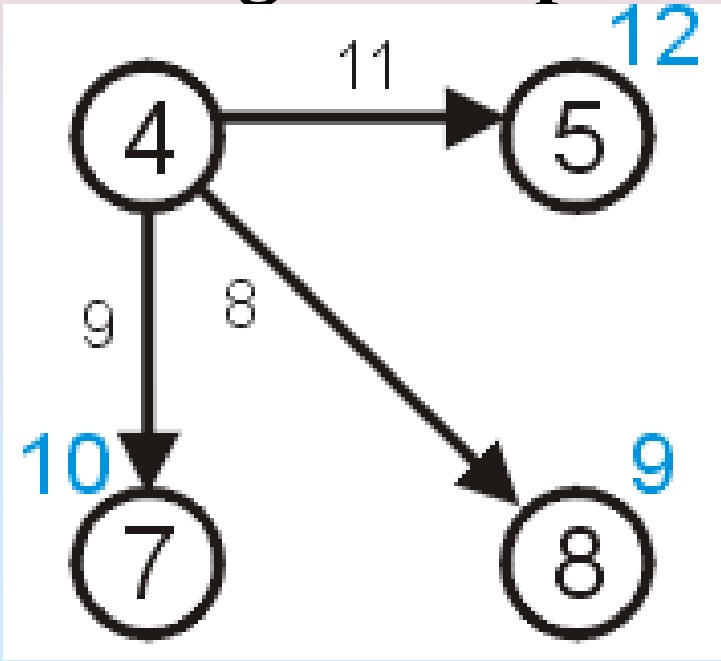
Dijkstra's Algorithm



Dijkstra's Algorithm

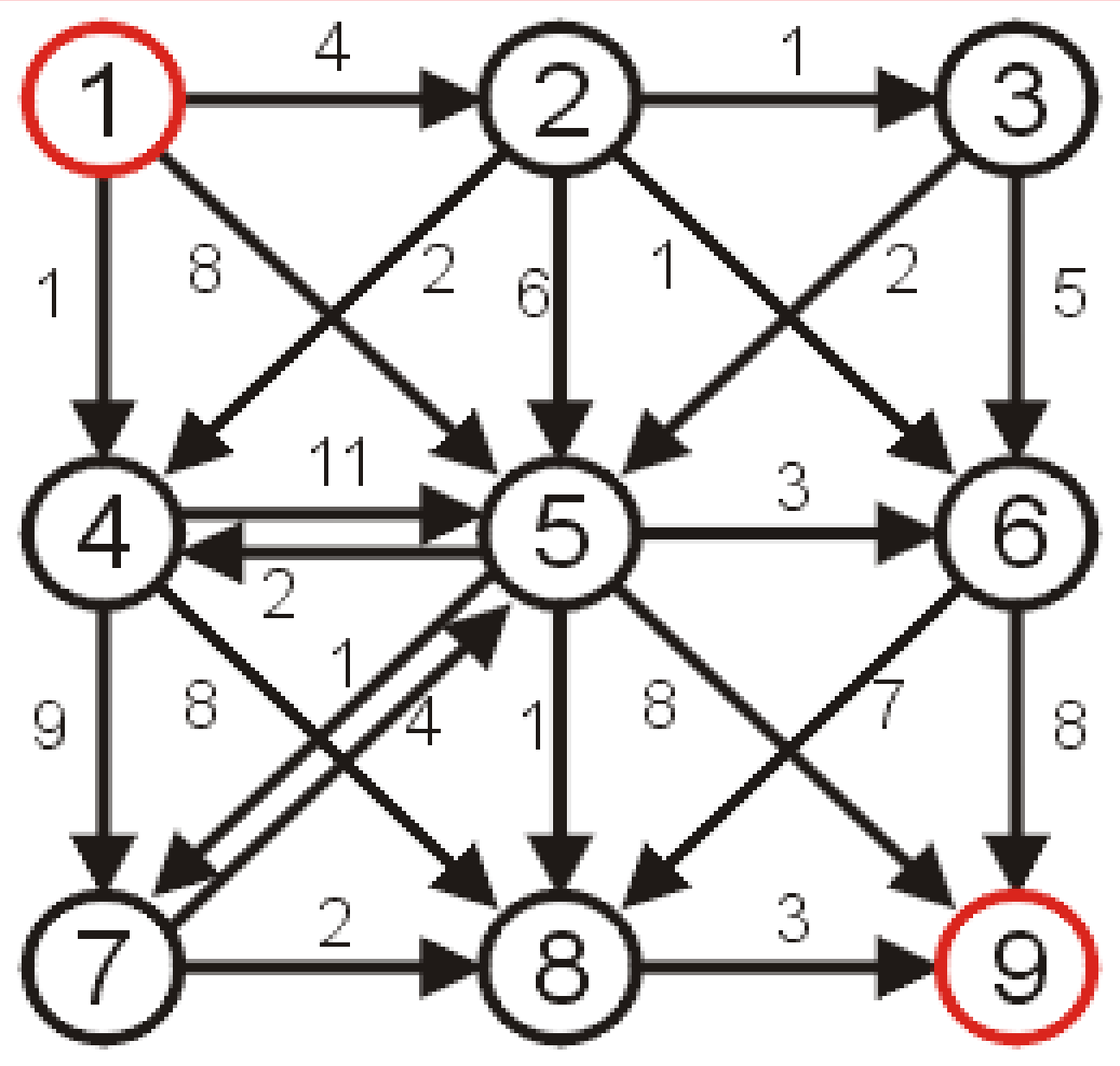
- We need to remember that the length of that path from node 1 to node 4 is 1

- Thus, we need to store the length of a path that goes through node 4:



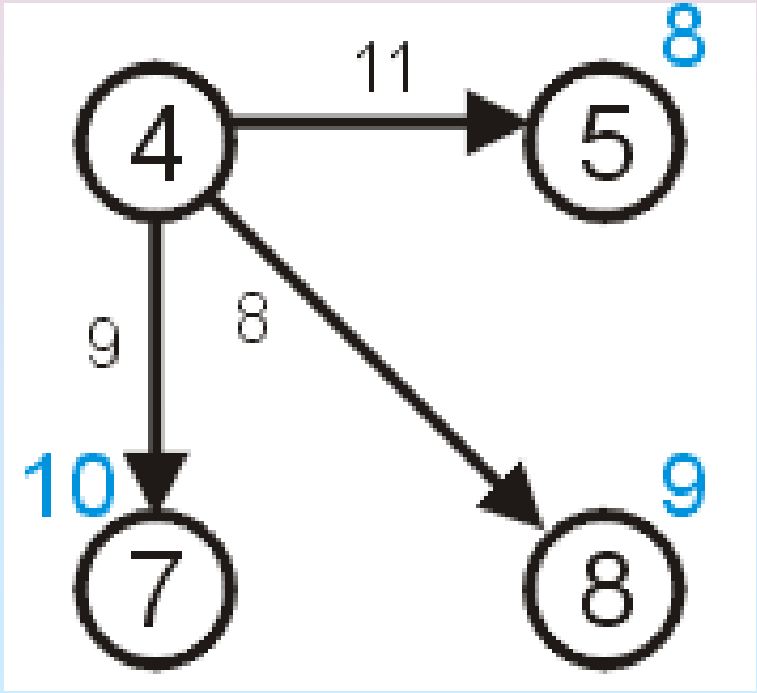
  - 5 of length 12
  - 8 of length 9
  - 7 of length 10

Dijkstra's Algorithm

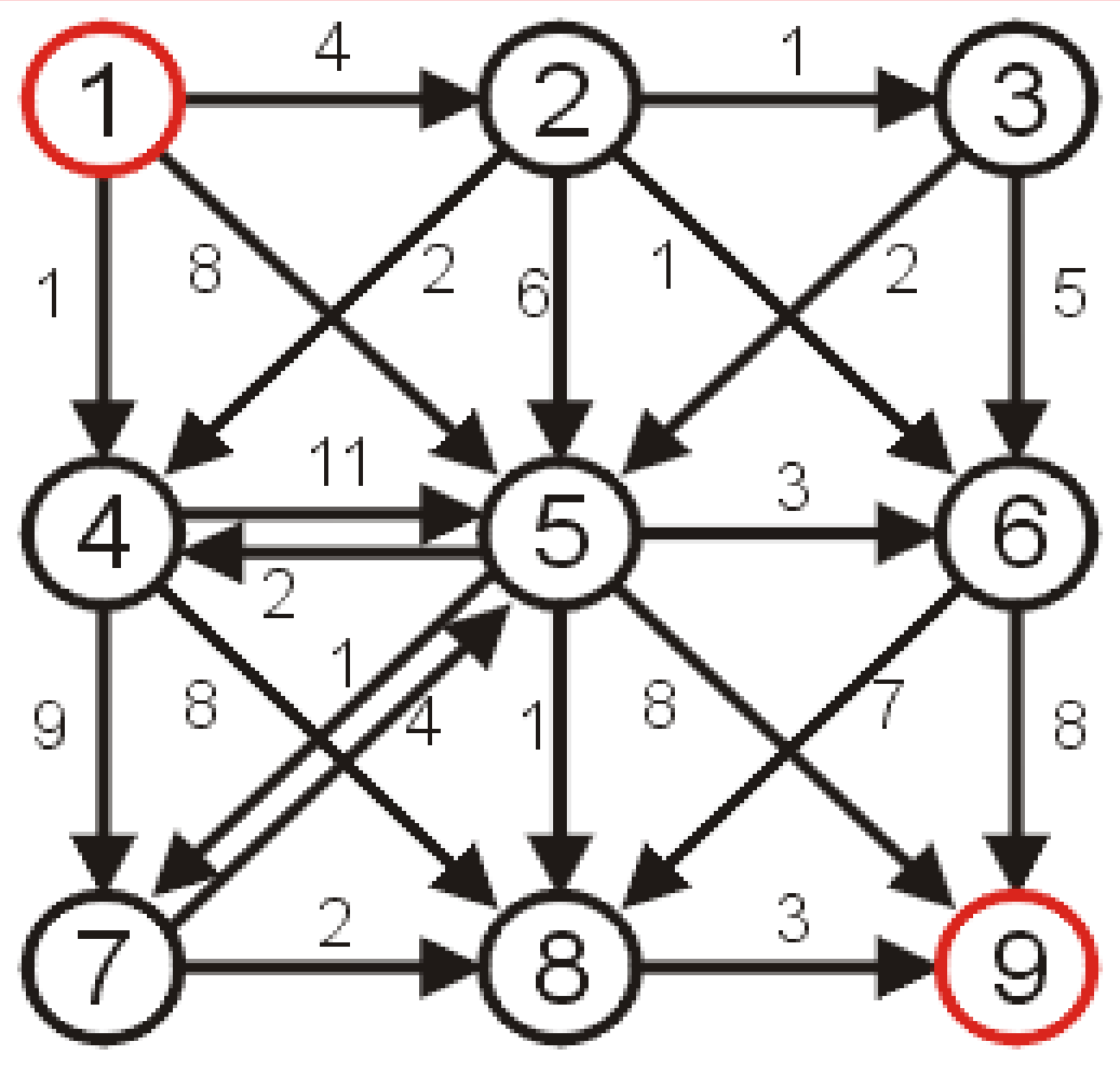


Dijkstra's Algorithm

- We have already discovered that there is a path of length 8 to vertex 5 with the path (1, 5).
- Thus, we can safely ignore this longer path.



Dijkstra's Algorithm



Dijkstra's Algorithm

- **We now know that:**
  - There exist paths from vertex 1 to vertices { 2, 4, 5, 7, 8 }.
  - We know that the shortest path from vertex 1 to vertex 4 is of length 1.
  - We know that the shortest path to the other vertices { 2, 5, 7, 8 } is at least the length listed in the table to the right.

Vertex	Length
1	0
2	4
4	1
5	8
7	9
8	9

Dijkstra's Algorithm

- There cannot exist a shorter path to either of the vertices 1 or 4, since the distances can only increase at each iteration.

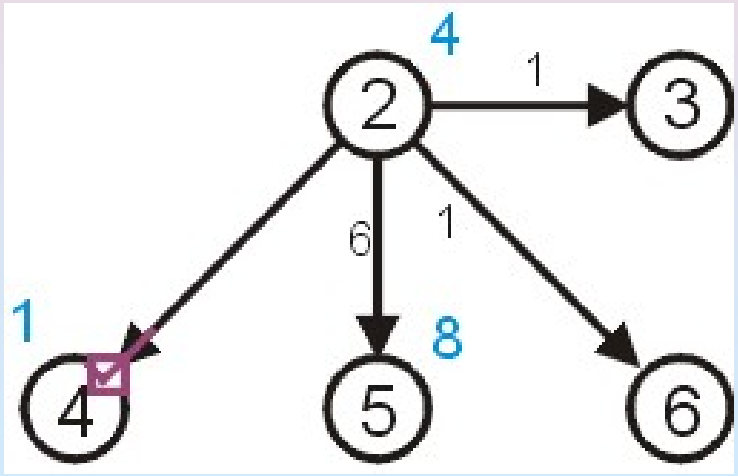
Vertex	Length
1	0
2	4
4	1
5	8
7	9
8	9

- We consider these vertices to be visited

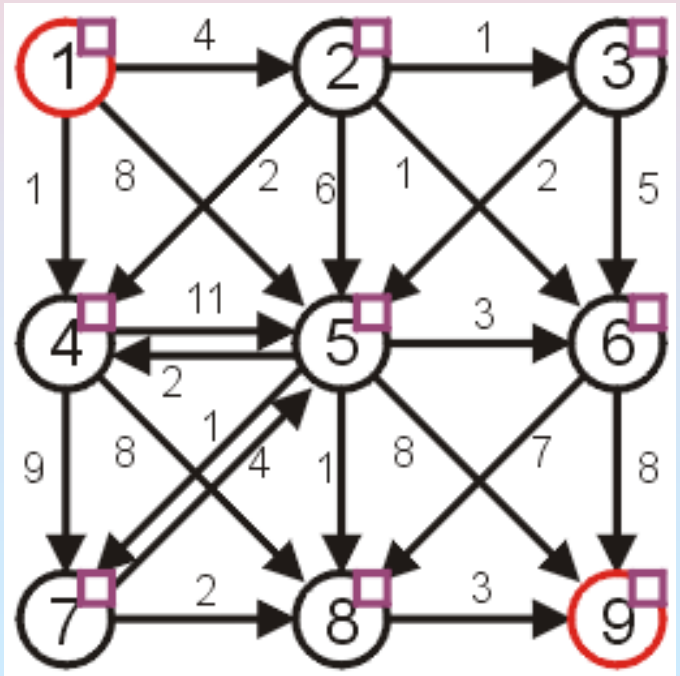
Dijkstra's Algorithm

- In Dijkstra's algorithm, we always take the next unvisited vertex which has the current shortest path from the starting vertex in the table.

- This is vertex 2

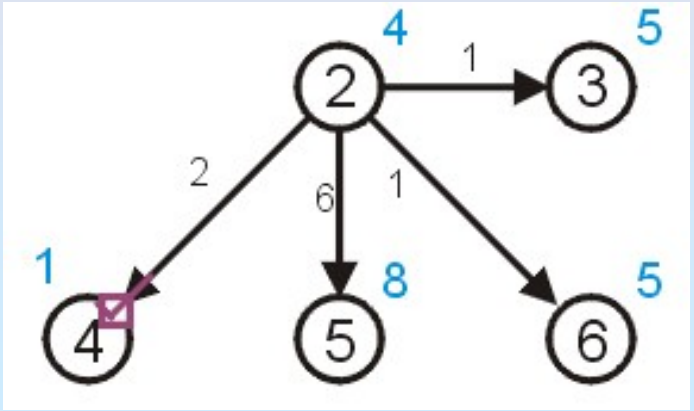


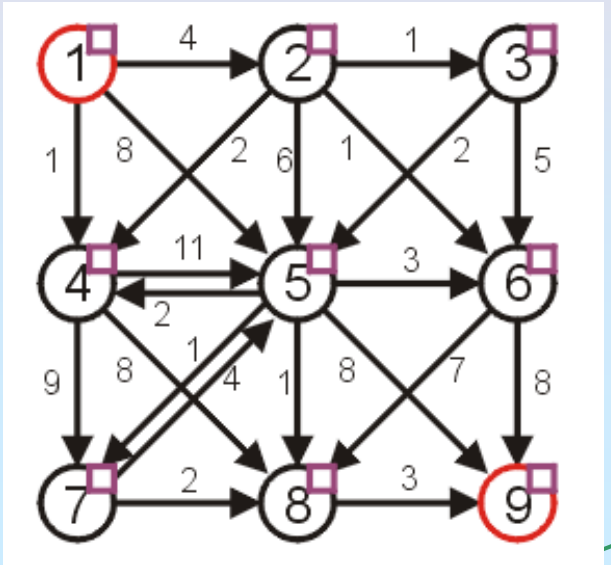
Vertex	Length
1	0
2	**4**
4	1
5	8
7	9
8	9



Dijkstra's Algorithm

- We can try to update the shortest paths to vertices 3 and 6 (both of length 5) however:
  - there already exists a path of length 8 < 10 to vertex 5 (10 = 4 + 6)
  - we already know the shortest path to 4 is 1



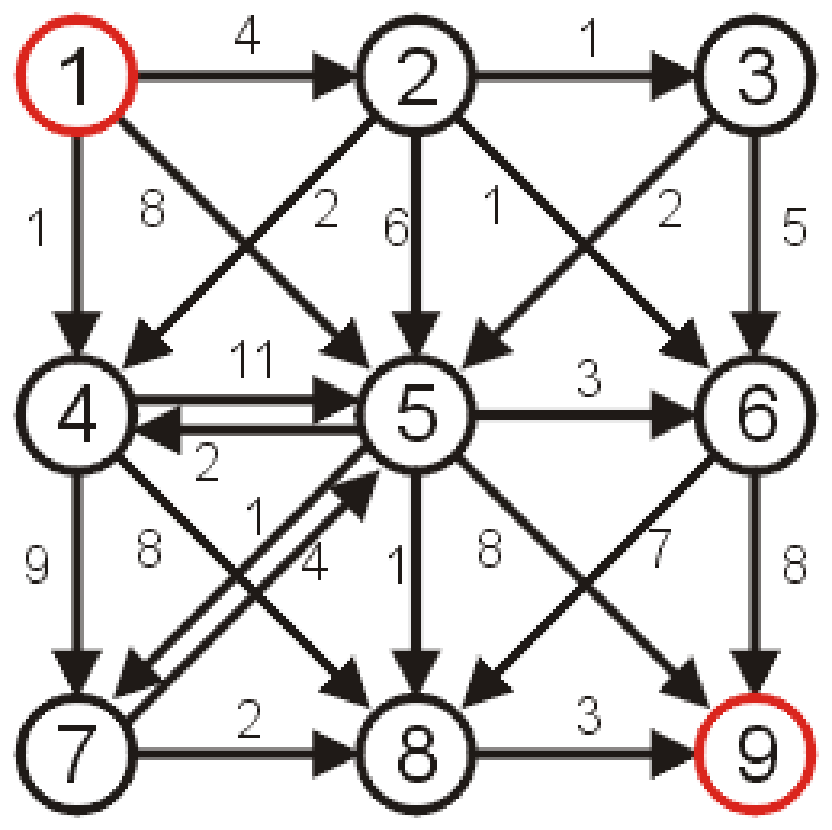


Dijkstra's Algorithm

- To keep track of those vertices to which no path has reached, we can assign those vertices an initial distance of either
  - infinity (∞),
  - a number larger than any possible path, or
  - a negative number
- For demonstration purposes, we will use ∞
- Each time the shortest distance to a particular vertex is updated, keep track of the predecessor used to reach this vertex on the shortest path.

Dijkstra's Algorithm

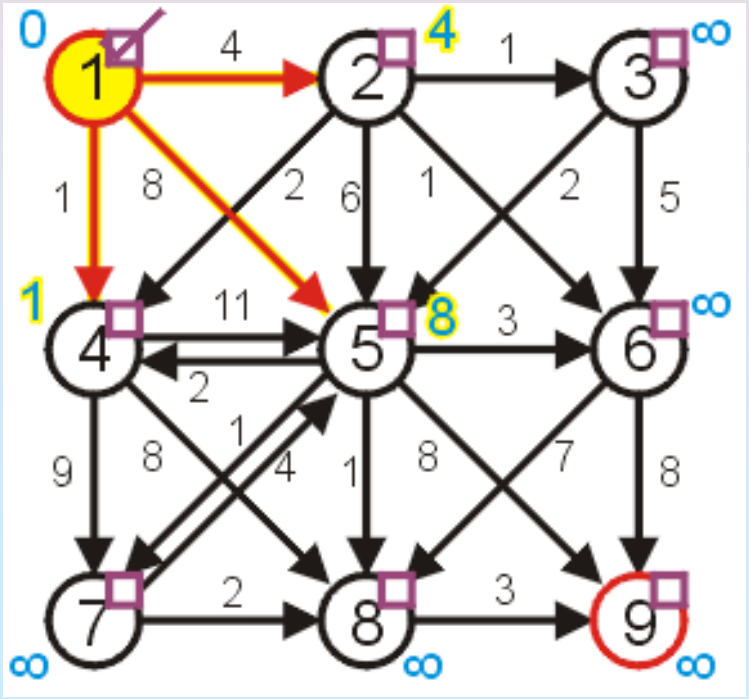
- Store a table of pointers, each initially 0
- This table will be updated each time a distance is updated



1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

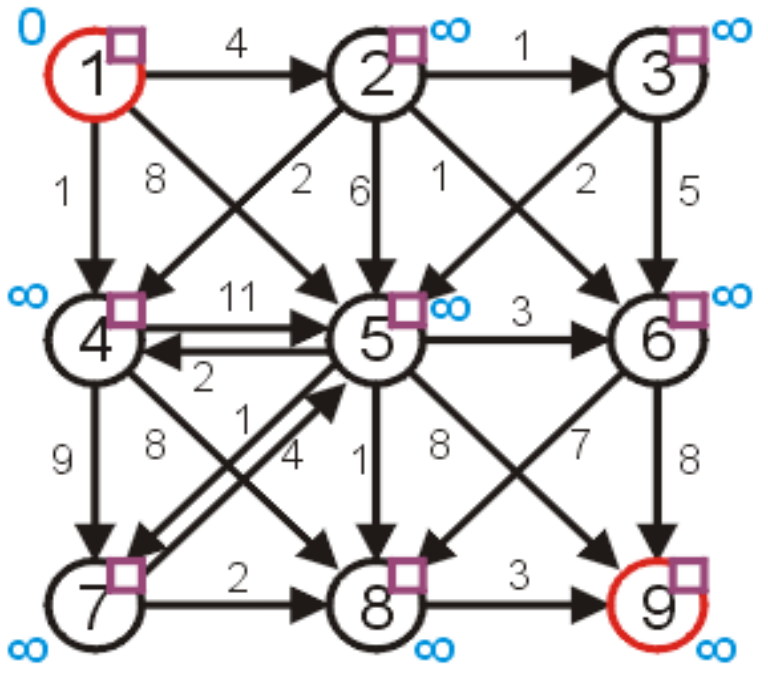
Dijkstra's Algorithm

- Graphically, display the reference to the preceding vertex by a red arrow.
  - If the distance to a vertex is ∞, there will be no preceding vertex
  - otherwise, there will be at least one preceding vertex



Dijkstra's Algorithm

- **Thus, for initialization:**
  - **set the current distance to the initial vertex as 0**
  - **for all other vertices, set the current distance to ∞**
  - **all vertices are initially marked as unvisited**
  - **set the previous pointer for all vertices to null**



Dijkstra's Algorithm

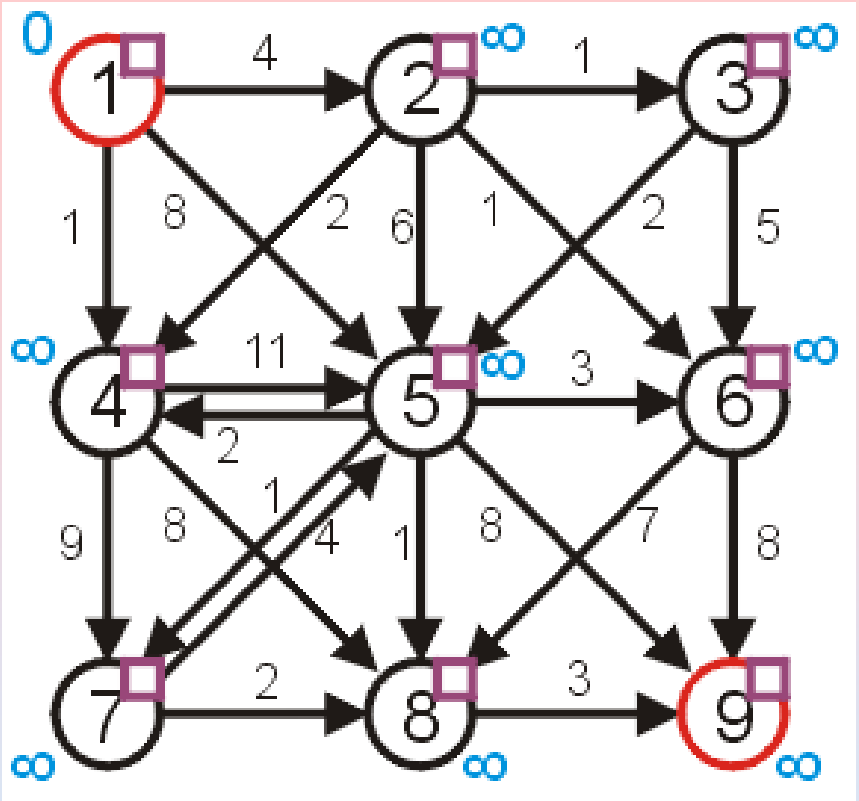
- **Thus, iterate:**
  - **find an unvisited vertex which has the shortest distance to it**
  - **mark it as visited**
  - **for each unvisited vertex which is adjacent to the current vertex:**
    - **add the distance to the current vertex to the weight of the connecting edge**
    - **if this is less than the current distance to that vertex, update the distance and set the parent vertex of the adjacent vertex to be the current vertex**

Dijkstra's Algorithm

- **Halting condition:**
  - successfully halt when the vertex being visited is the target vertex
  - if at some point, all remaining unvisited vertices have distance ∞, then no path from the starting vertex to the end vertex exits
- **Note:** Do not halt just because the distance to the end vertex is updated, as the target vertex has not been visited so far.

Example

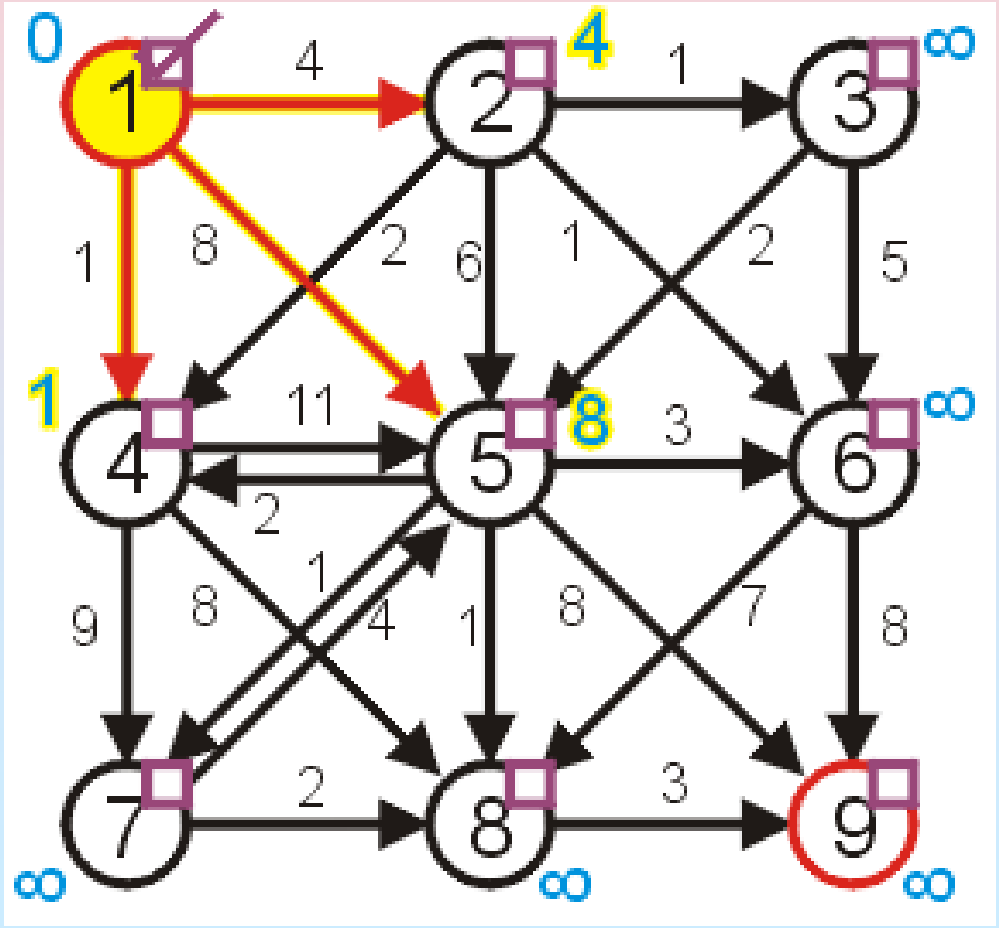
- **Consider the graph:**



- **the distances are appropriately initialized**
- **all vertices are marked as being unvisited**

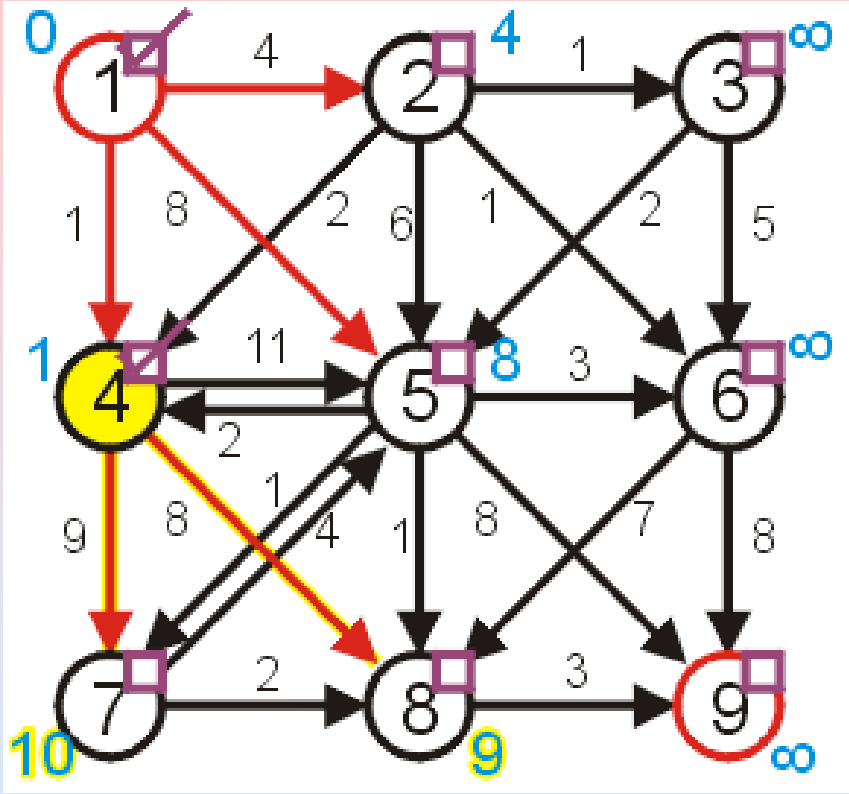
Example

- Visit vertex 1 and update its neighbours, marking it as visited
  - the shortest paths to 2, 4, and 5 are updated



Example

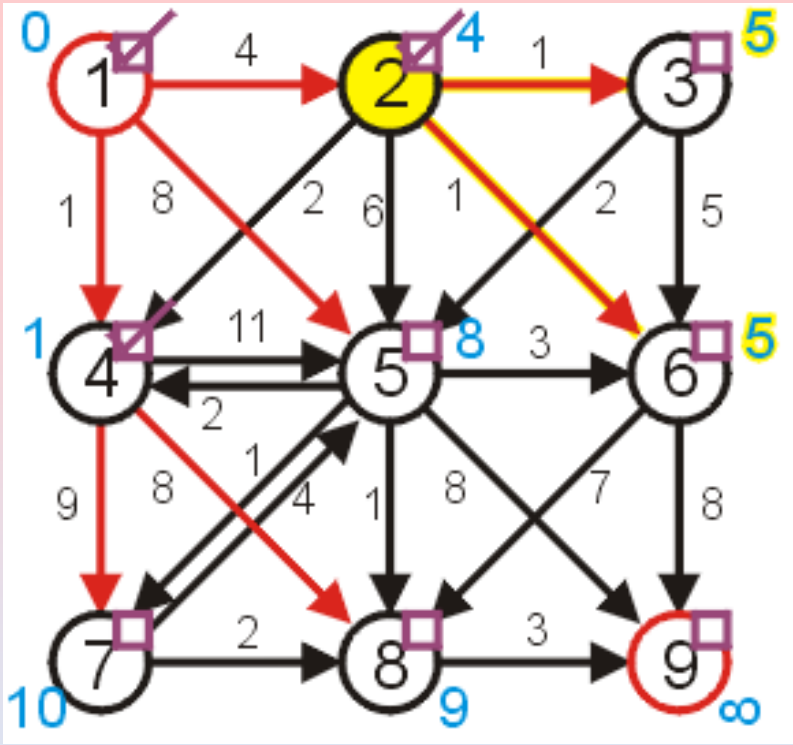
- The next vertex to visit is vertex 4



- vertex 5 $1 + 11 \geq 8$ don't update
- vertex 7 $1 + 9 < \infty$ update
- vertex 8 $1 + 8 < \infty$ update

Example

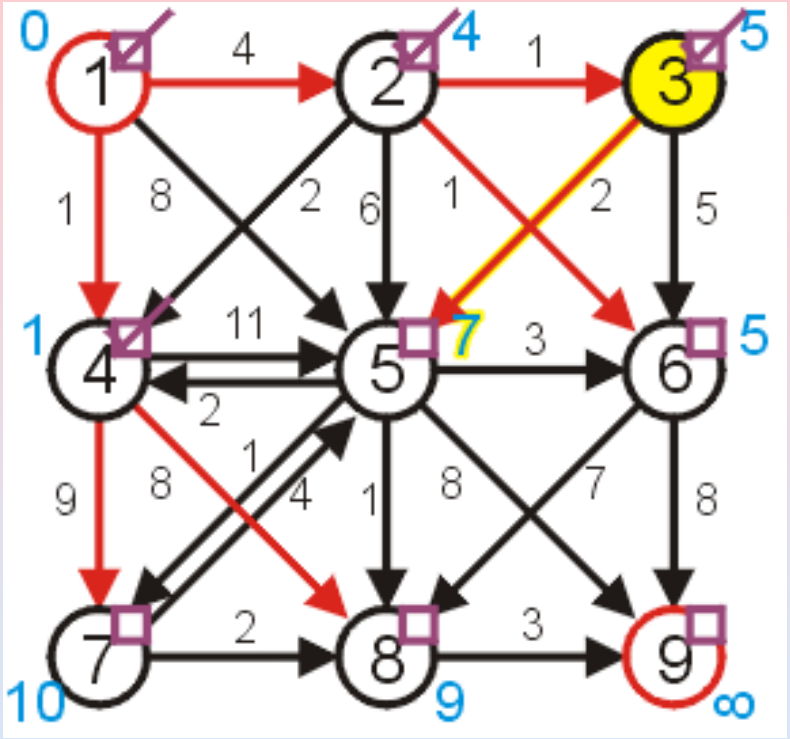
- **Next, visit vertex 2**



➤ **vertex 3**	**4 + 1 < ∞**	**update**
➤ **vertex 4**		**already visited**
➤ **vertex 5**	**4 + 6 ≥ 8**	**don't update**
➤ **vertex 6**	**4 + 1 < ∞**	**update**

Example

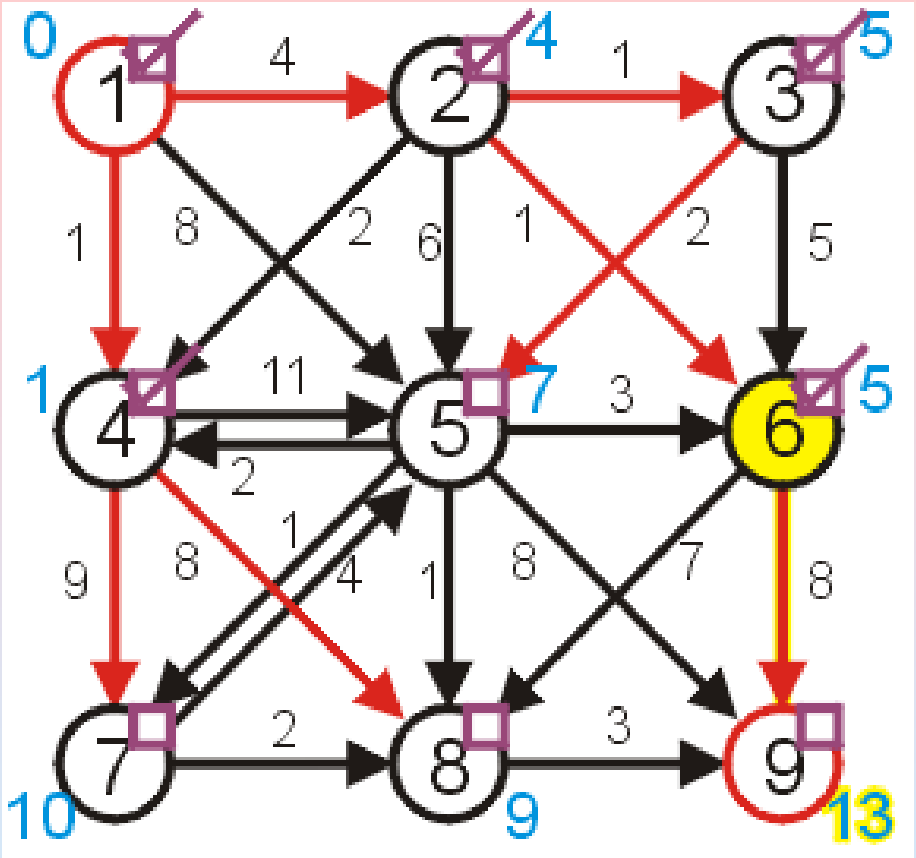
- Next, there is a choice of either 3 or 6



- Choose to visit 3
  - vertex 5 — 5 + 2 < 8 — update
  - vertex 6 — 5 + 5 ≥ 5 — don't update

Example

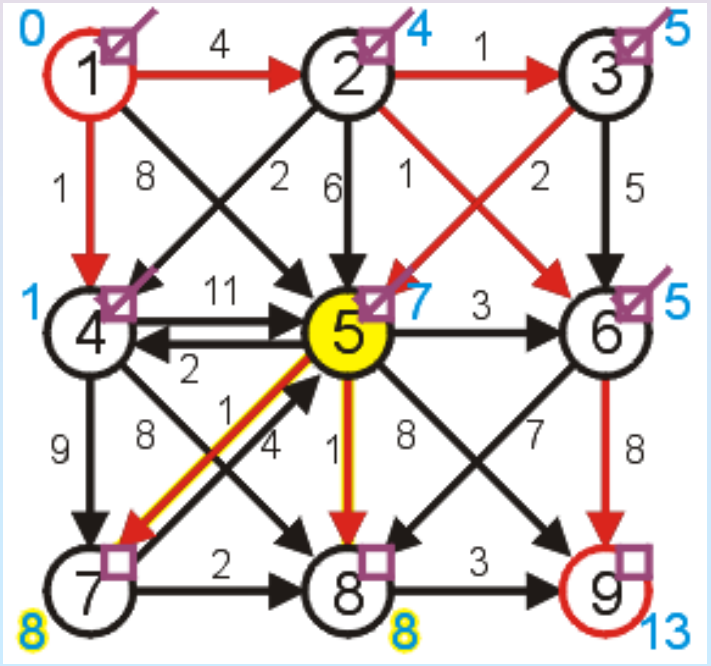
- Then visit 6



- vertex 8 $5 + 7 \geq 9$ don't update
- vertex 9 $5 + 8 < \infty$ update

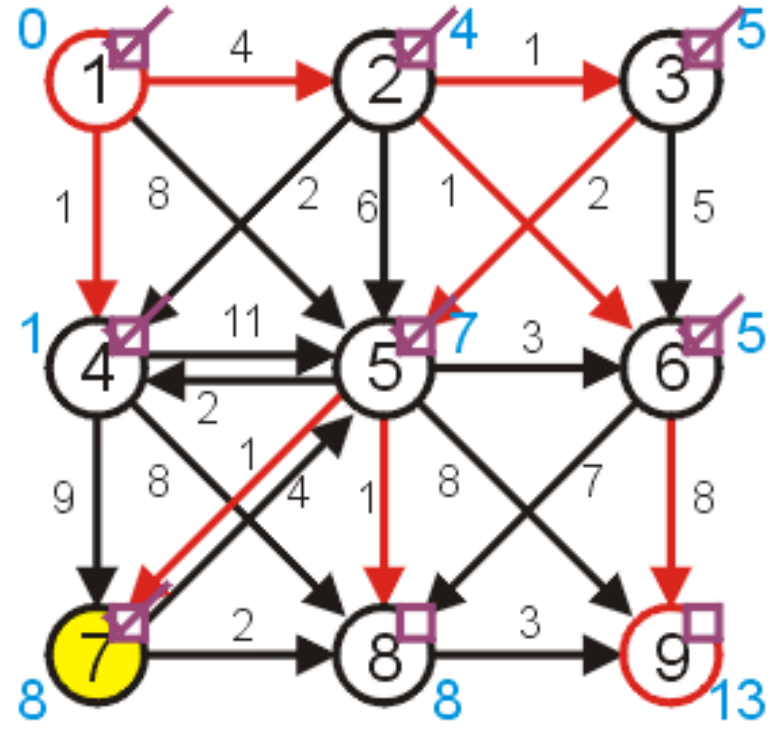
Example

- Next, finally visit vertex 5:
  - vertices 4 and 6 have already been visited
  - vertex 7 — $7 + 1 < 10$ — update
  - vertex 8 — $7 + 1 < 9$ — update
  - vertex 9 — $7 + 8 \geq 13$ — don't update



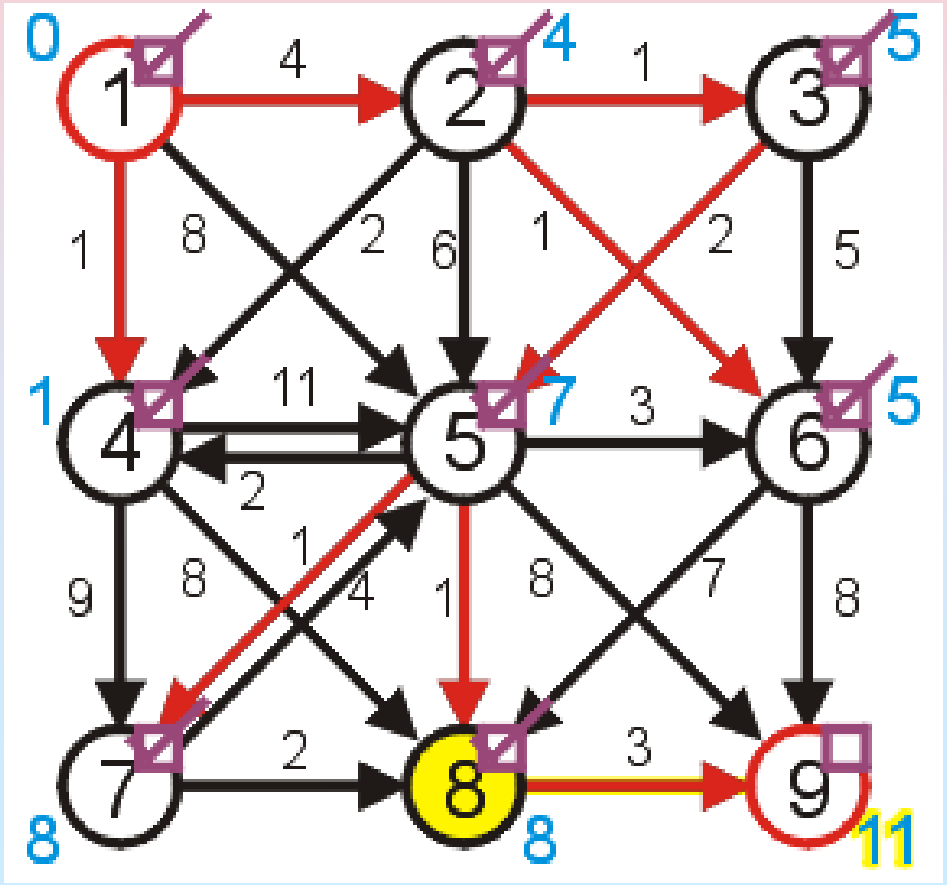
Example

- Given a choice between vertices 7 and 8, choose vertex 7
  - vertices 5 has already been visited
  - vertex 8 $8 + 2 \geq 8$ don't update



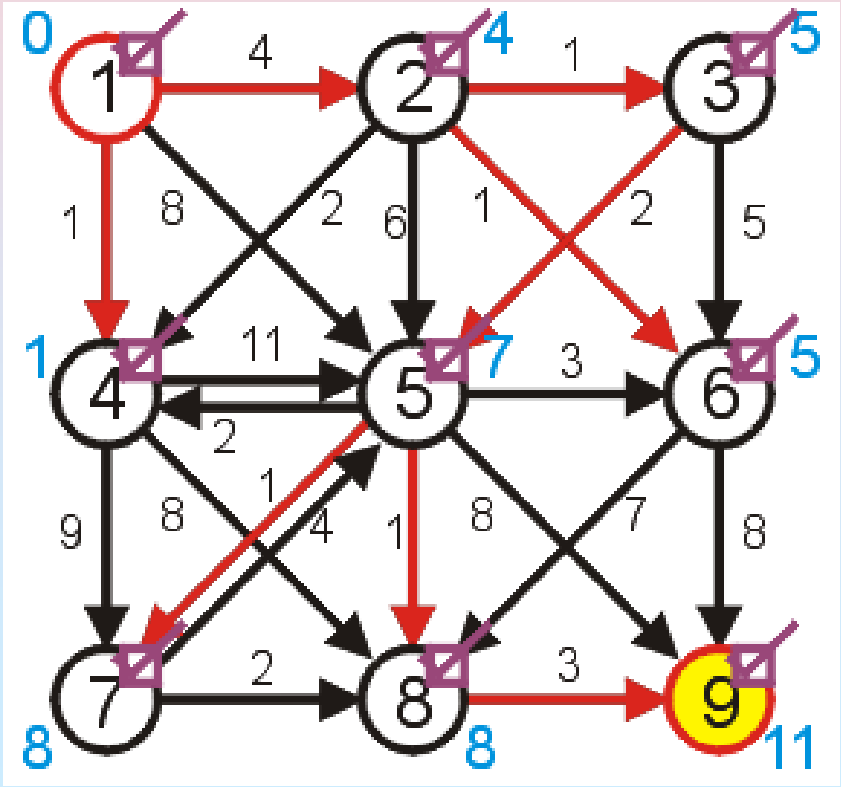
Example

- **Next, visit vertex 8:**
  - **vertex 9** **8 + 3 < 13** **update**



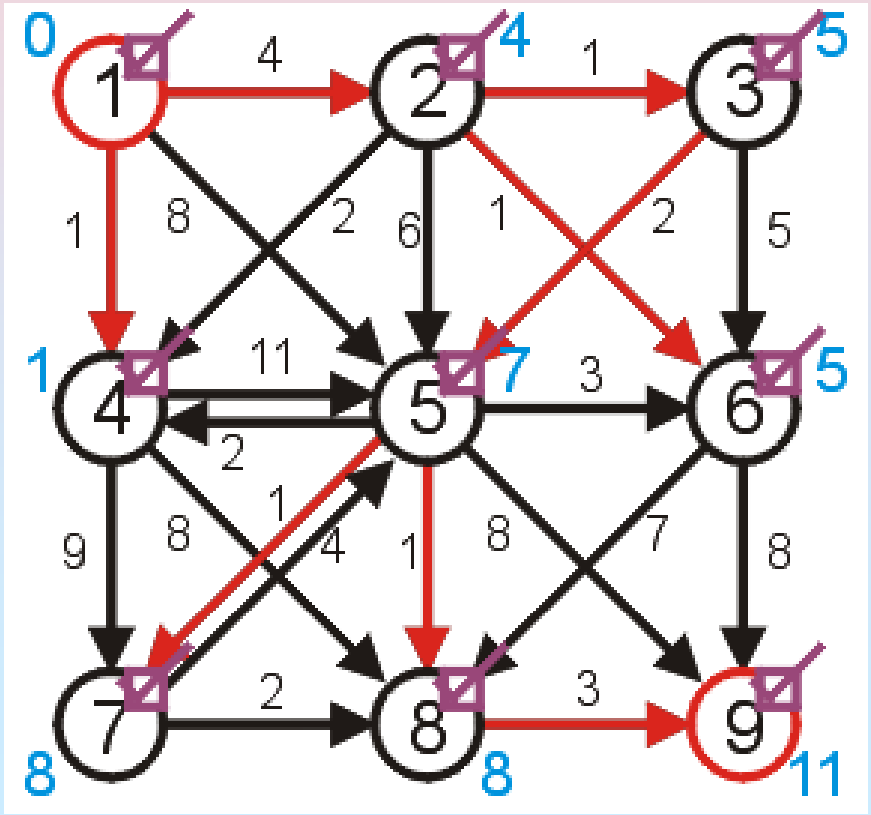
Example

- Finally, visit the end vertex
- Therefore, the shortest path from 1 to 9 has length 11



Example

- **Find the shortest path by working back from the final vertex:**
  - **9, 8, 5, 3, 2, 1**
- **Thus, the shortest path is (1, 2, 3, 5, 8, 9)**

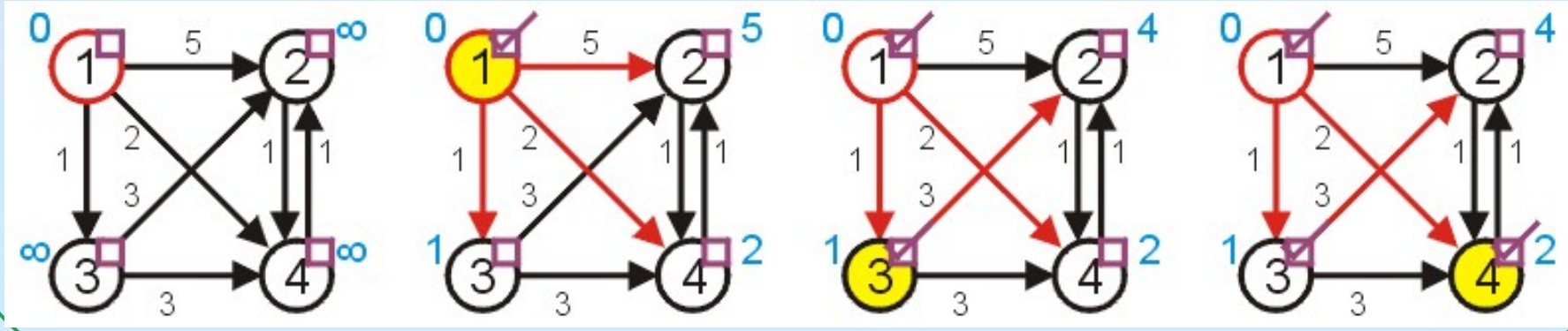


Example

- In the example, we visited all vertices in the graph before we finished
- This is not always the case, consider the next example

Example

- Find the shortest path from 1 to 4:
  - the shortest path is found after only three vertices are visited
  - we terminated the algorithm as soon as we reached vertex 4
  - we only have useful information about 1, 3, 4
  - we don't have the shortest path to vertex 2



Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
    do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ ▷ Q is a priority queue maintaining $V - S$
while $Q \neq \varnothing$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        $S \leftarrow S \cup \{u\}$
        for each $v \in Adj[u]$
            do if $d[v] > d[u] + w(u, v)$
                then $d[v] \leftarrow d[u] + w(u, v)$
                $p[v] \leftarrow u$

Dijkstra's algorithm

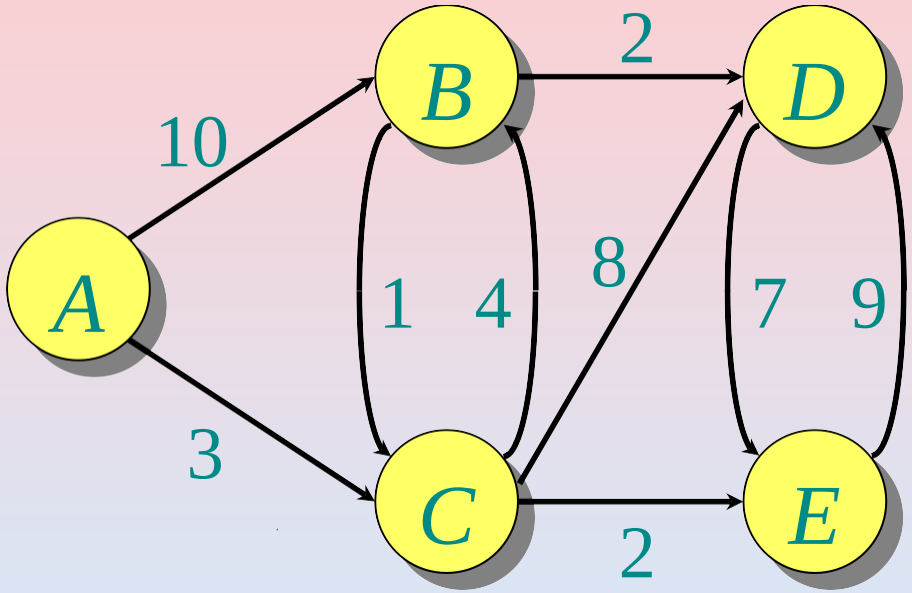
$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \emptyset$
$Q \leftarrow V$ ▷ Q is a priority queue maintaining $V - S$
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$
      do if $d[v] > d[u] + w(u, v)$
        then $d[v] \leftarrow d[u] + w(u, v)$
          $p[v] \leftarrow u$

relaxation step

Implicit DECREASE-KEY

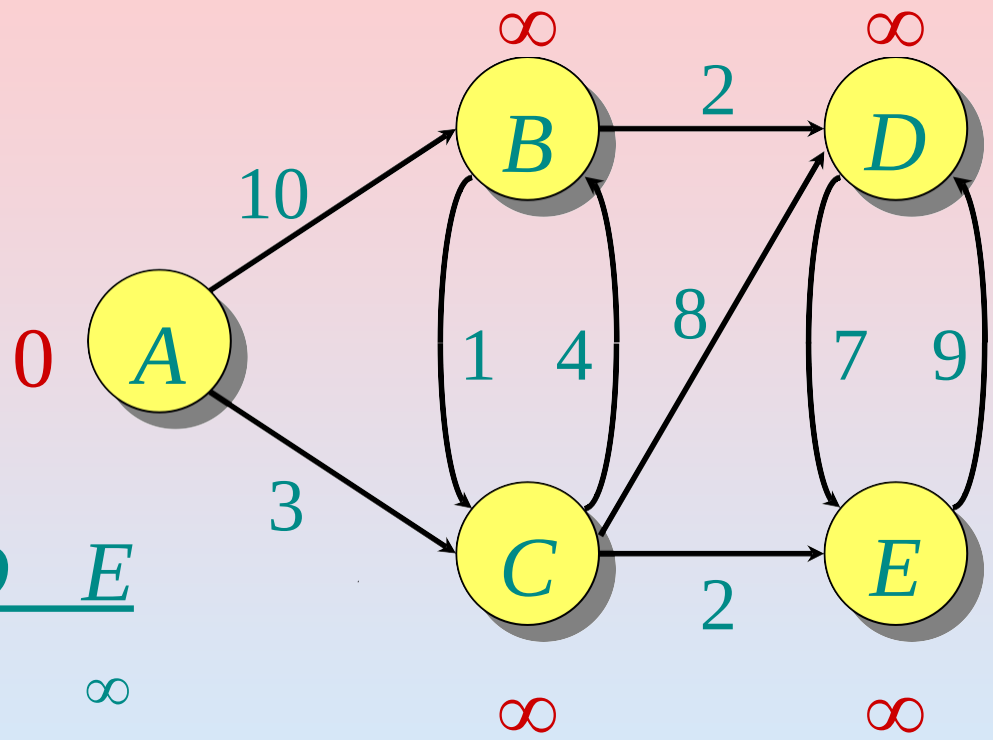
Example of Dijkstra's algorithm

Graph with nonnegative edge weights:



Example of Dijkstra's algorithm

Initialize:

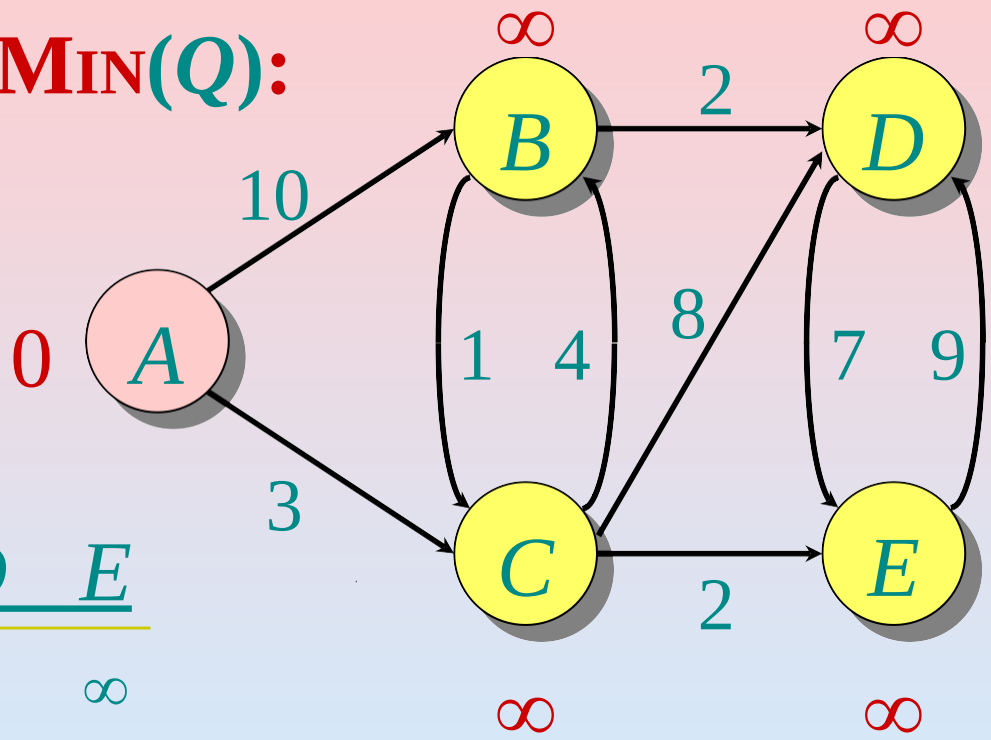


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

Example of Dijkstra's algorithm

"*A*" ← EXTRACT-MIN(*Q*):

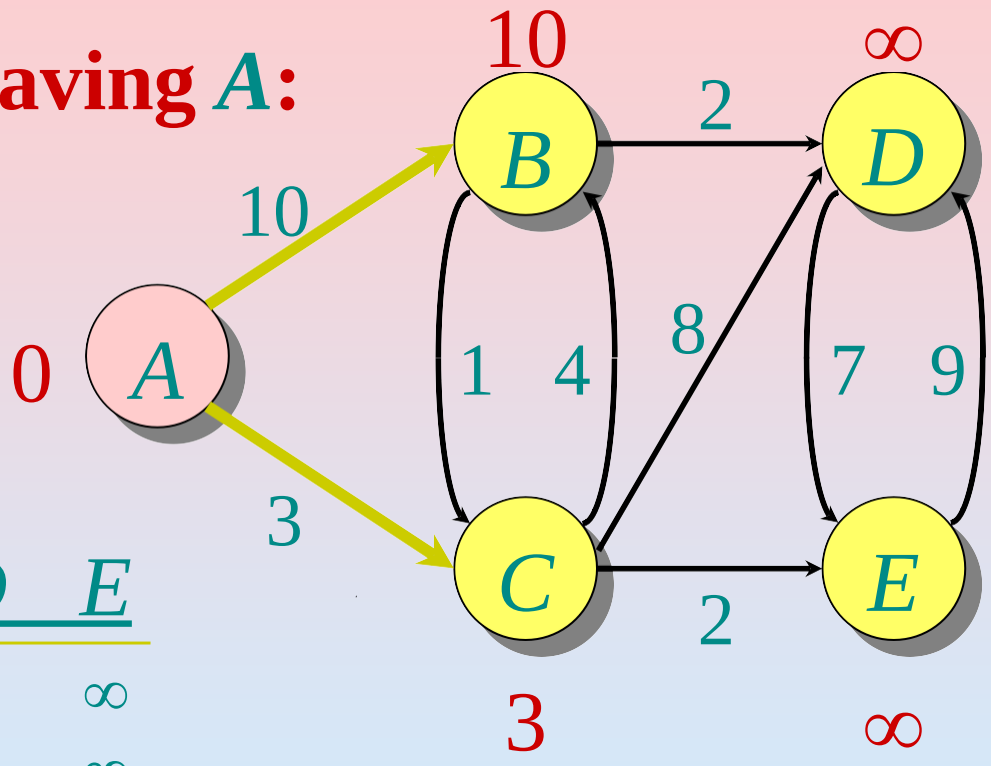


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞

S: { *A* }

Example of Dijkstra's algorithm

Relax all edges leaving *A*:

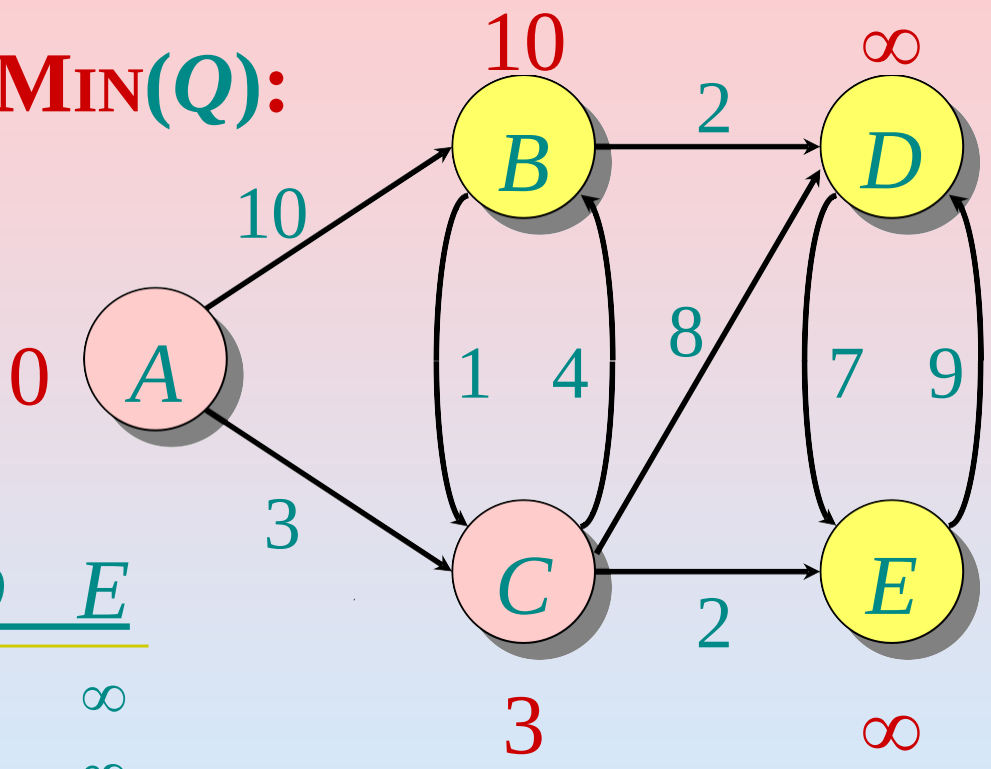


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A* }

Example of Dijkstra's algorithm

"*C*" ← EXTRACT-MIN(*Q*):



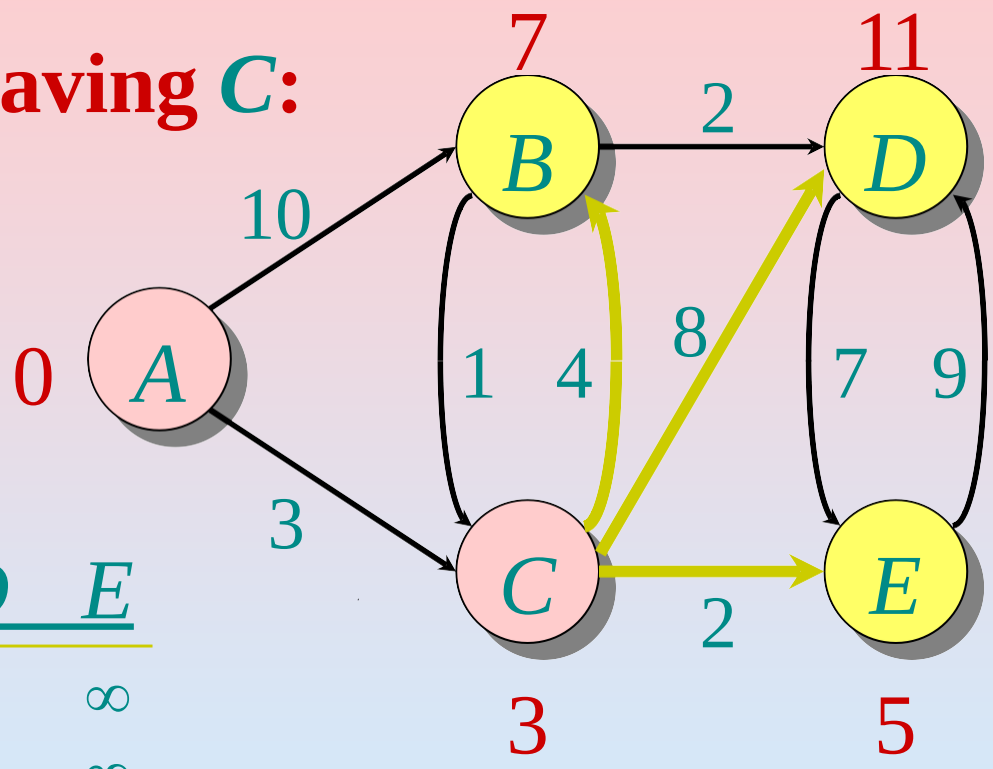
Q:

A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
	10	3	∞	∞

S: { *A*, *C* }

Example of Dijkstra's algorithm

Relax all edges leaving *C*:

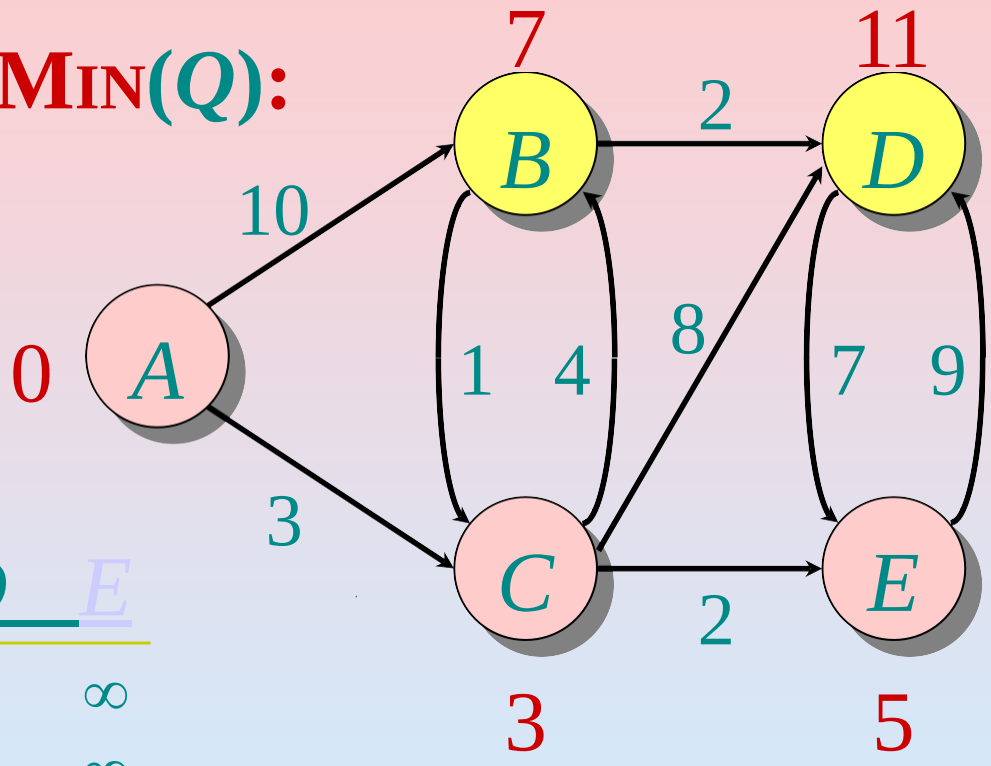


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C* }

Example of Dijkstra's algorithm

"*E*" ← EXTRACT-MIN(*Q*):



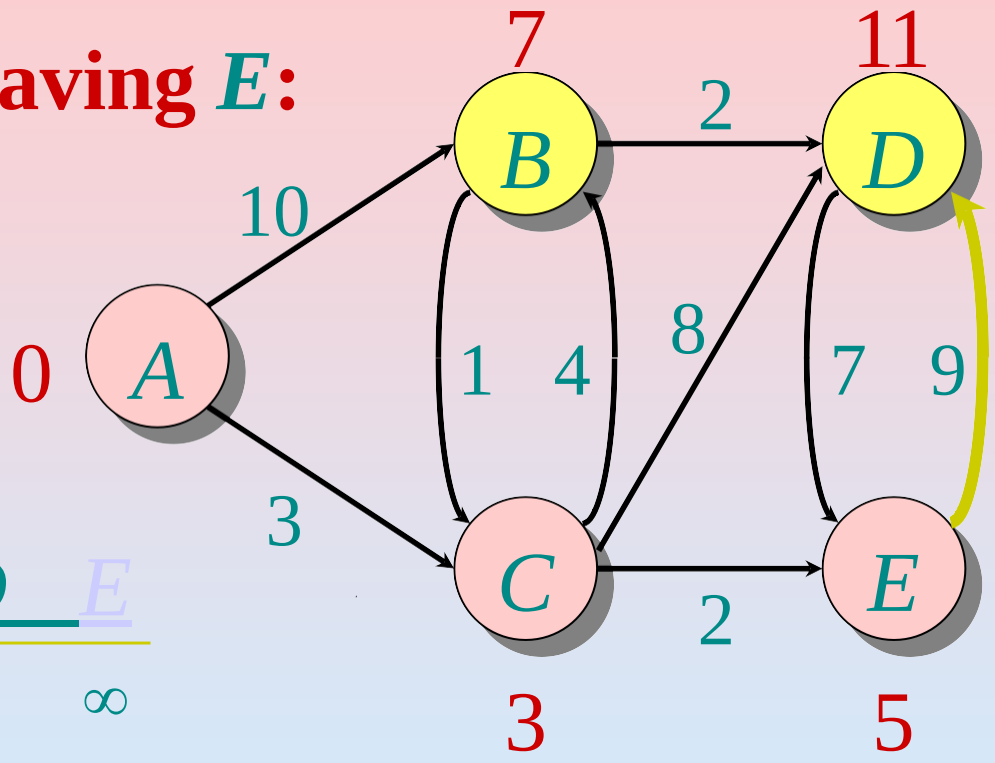
Q:

A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5

S: { *A*, *C*, *E* }

Example of Dijkstra's algorithm

Relax all edges leaving *E*:

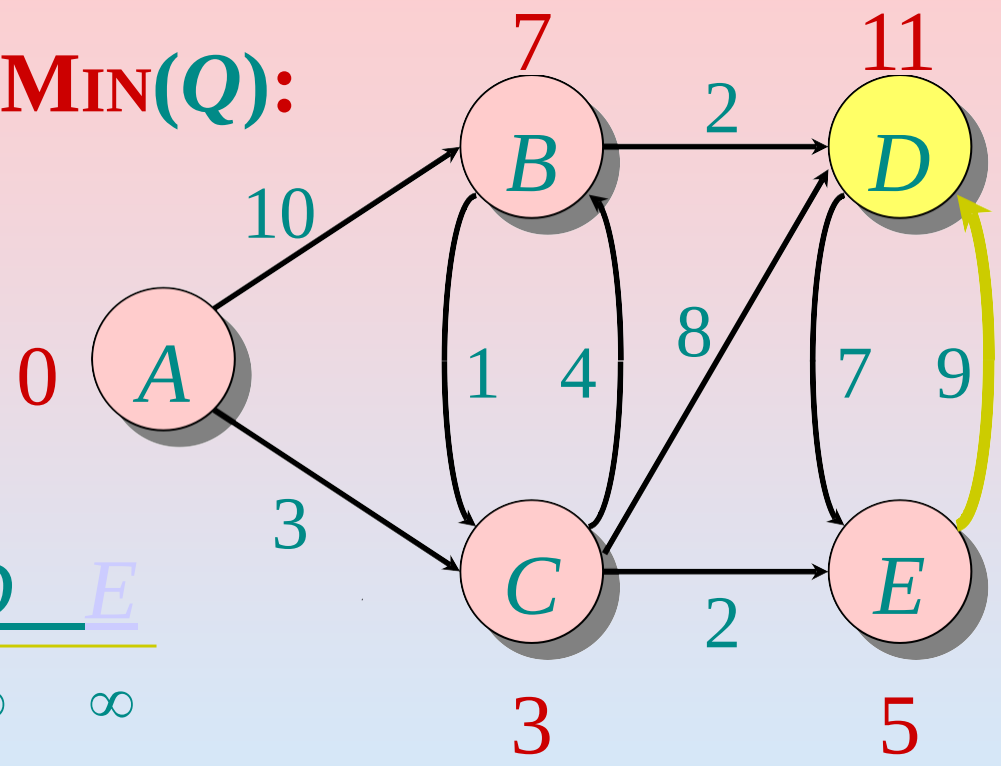


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { *A*, *C*, *E* }

Example of Dijkstra's algorithm

"*B*" ← EXTRACT-MIN(*Q*):



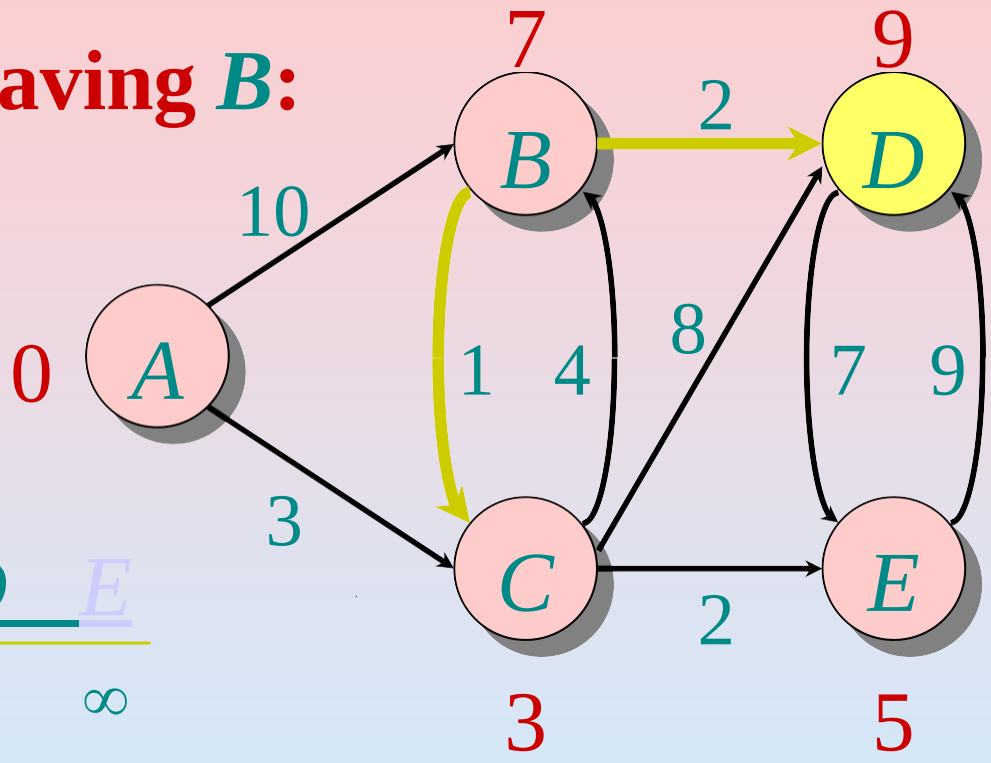
Q:

A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	

S: { *A*, *C*, *E*, *B* }

Example of Dijkstra's algorithm

Relax all edges leaving *B*:

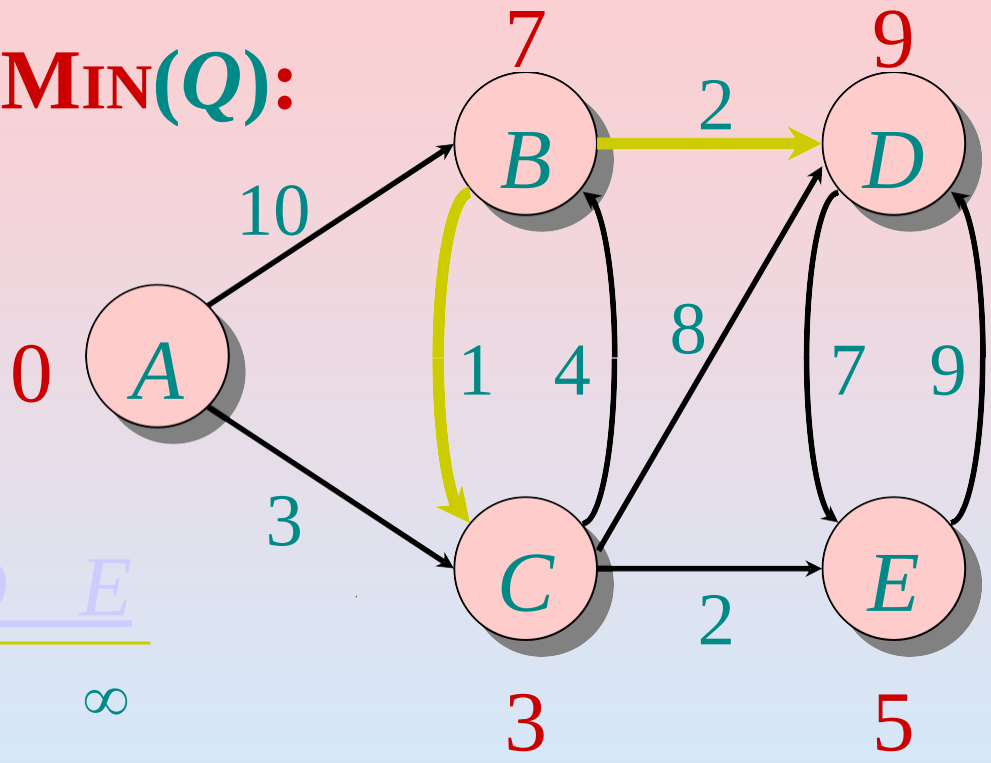


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { *A*, *C*, *E*, *B* }

Example of Dijkstra's algorithm

"*D*" ← EXTRACT-MIN(*Q*):



Q: *A* *B* *C* *D* *E*

0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	
			9	

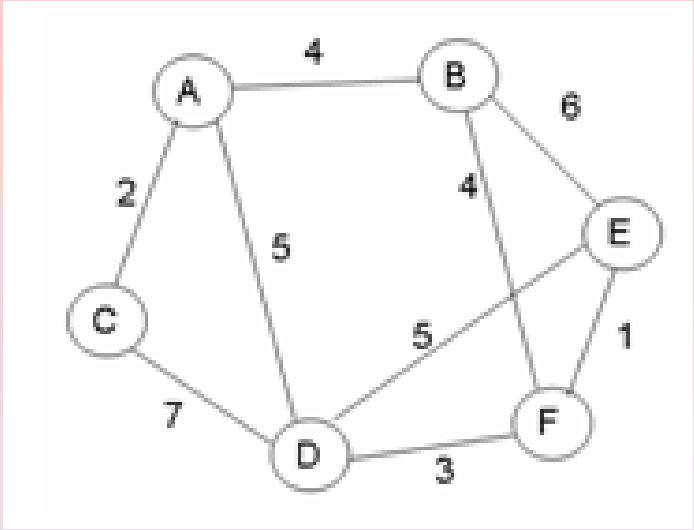
S: { *A*, *C*, *E*, *B*, *D* }

Summary

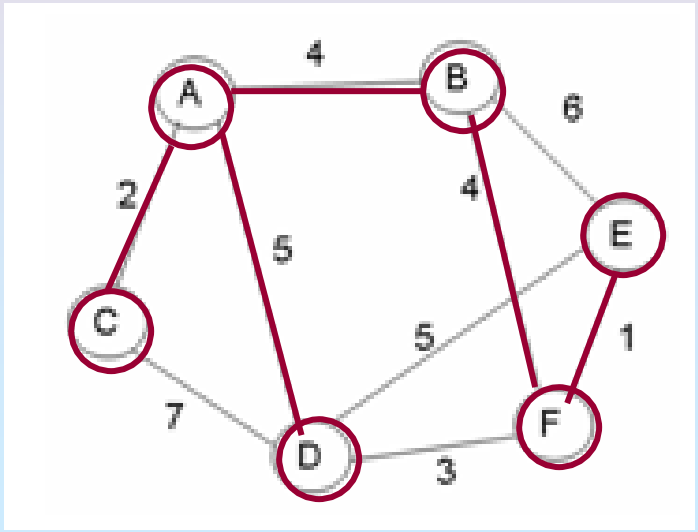
- Given a weighted directed graph, we can find the shortest distance between two vertices by:
  - starting with a trivial path containing the initial vertex
  - growing this path by always going to the next vertex which has the shortest current path

Practice

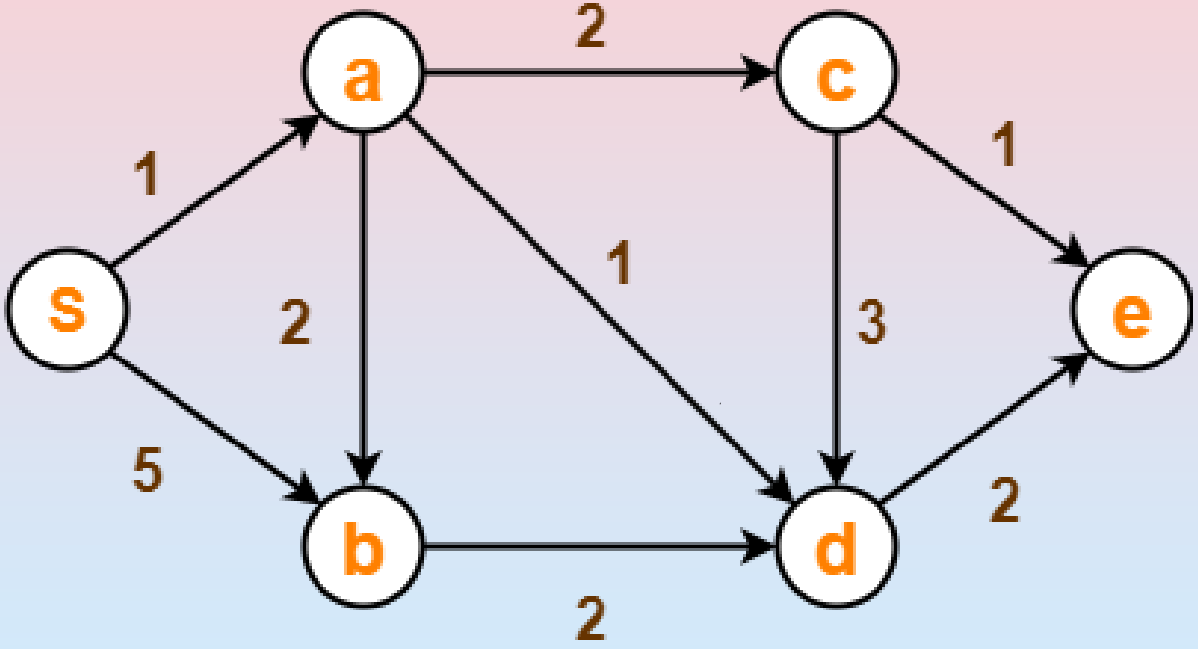
Node	Included	Distance	Path
A	t	-	-
B	f t	4	A
C	f t	2	A
D	f t	5	A
E	f t	~~∞~~ 10 9	- ~~B~~ F
F	f t	~~∞~~ 8	- B



Give the shortest path tree for node A for this graph using Dijkstra's shortest path algorithm. Show your work with the 3 arrays given and draw the resultant shortest path tree with edge weights included.



Using Dijkstra's Algorithm, find the shortest distance from source vertex 'S' to remaining vertices in the following graph



Write the order in which the vertices are visited.

1. Unvisited set : {S , a , b , c , d , e}
   Visited set : { }

2. Two variables Π and d are created for each vertex and initialized as-

$$\Pi[S] = \Pi[a] = \Pi[b] = \Pi[c] = \Pi[d] = \Pi[e] = NIL$$

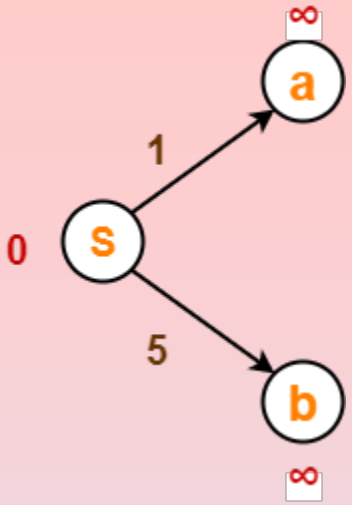
$$d[S] = 0$$

$$d[a] = d[b] = d[c] = d[d] = d[e] = \infty$$

3. Vertex ‘S’ is chosen.

The outgoing edges of vertex ‘S’ are relaxed.

Before Edge Relaxation-



$d[S] + 1 = 0 + 1 = 1 < \infty$

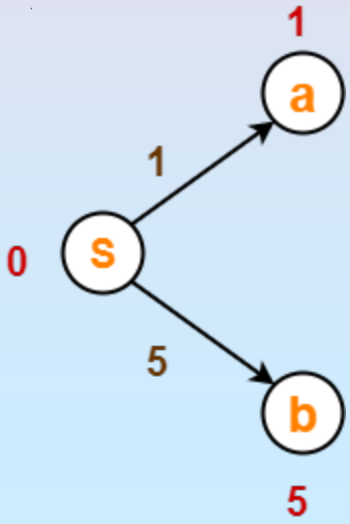
$\therefore d[a] = 1$ and $\Pi[a] = S$

$d[S] + 5 = 0 + 5 = 5 < \infty$

$\therefore d[b] = 5$ and $\Pi[b] = S$

After edge relaxation,

shortest path tree is -



Now, the sets are updated as-

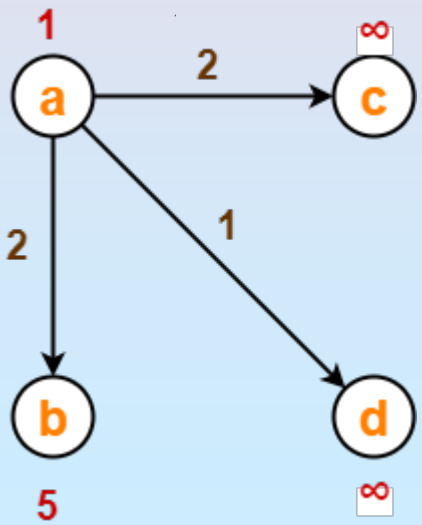
Unvisited set : {a , b , c , d , e}

Visited set : {S}

4. Vertex ‘a’ is chosen.

The outgoing edges of vertex ‘a’ are relaxed.

Before Edge Relaxation-



The outgoing edges of vertex ‘a’ are relaxed.

Now, $d[a] + 2 = 1 + 2 = 3 < \infty$

$\therefore d[c] = 3$ and $\Pi[c] = a$

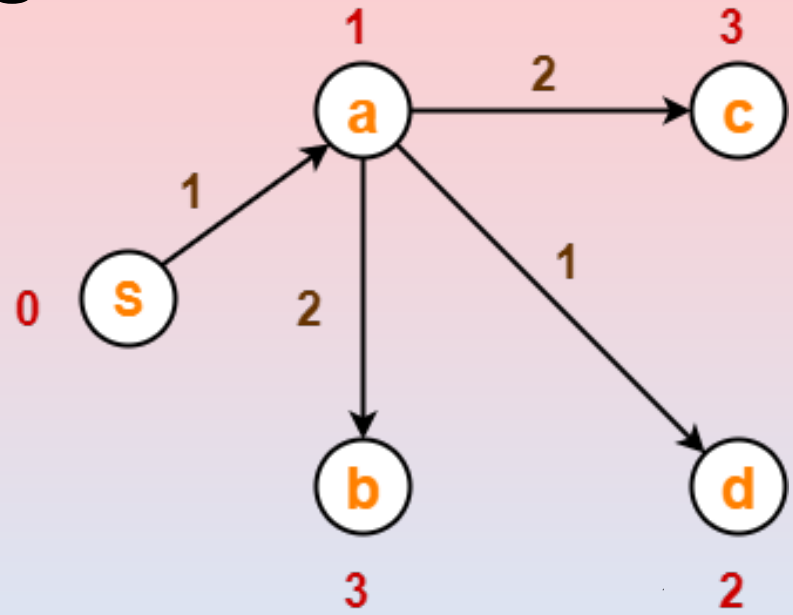
$d[a] + 1 = 1 + 1 = 2 < \infty$

$\therefore d[d] = 2$ and $\Pi[d] = a$

$d[b] + 2 = 1 + 2 = 3 < 5$

$\therefore d[b] = 3$ and $\Pi[b] = a$

After edge relaxation, our shortest path tree is-



Now, the sets are updated as-

Unvisited set : {b , c , d , e}

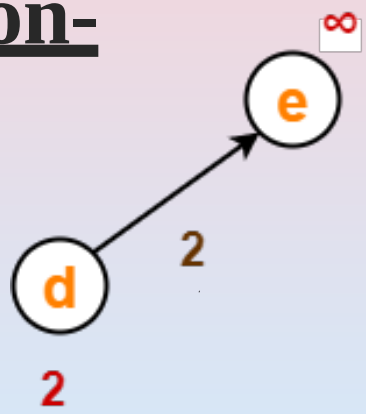
Visited set : {S , a}

5. **Vertex ‘d’ is chosen.**

This is because shortest path estimate for vertex ‘d’ is least.

The outgoing edges of vertex ‘d’ are relaxed.

<u>Before Edge Relaxation-</u>

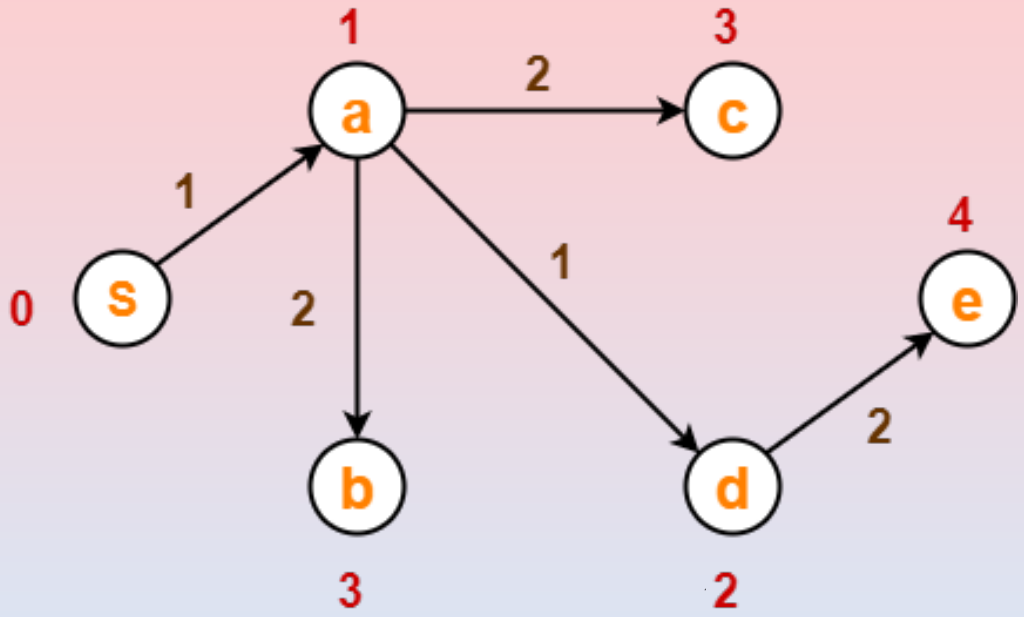


Now,

$d[d] + 2 = 2 + 2 = 4 < \infty$

$\therefore d[e] = 4$ and $\Pi[e] = d$

After edge relaxation, shortest path tree is-



Now, the sets are updated as-

Unvisited set : {b , c , e}

Visited set : {S , a , d}

6. **Vertex ‘b’ is chosen.**

This is because shortest path estimate for vertex ‘b’ is least.

Vertex ‘c’ may also be chosen since for both the vertices, shortest path estimate is least.

The outgoing edges of vertex ‘b’ are relaxed.

Before Edge Relaxation-

Now, d[b] + 2 = 3 + 2 = 5 > 2

∴ No change

After edge relaxation, our shortest path tree remains the same as in Step-05.

Now, the sets are updated as-

Unvisited set : {c , e}

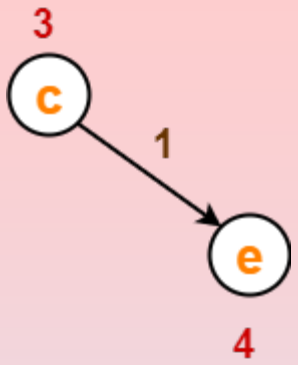
Visited set : {S , a , d , b}

Vertex ‘c’ is chosen.

This is because shortest path estimate for vertex ‘c’ is least.

The outgoing edges of vertex ‘c’ are relaxed.

Before Edge Relaxation-



Now, d[c] + 1 = 3 + 1 = 4 = 4

∴ No change

After edge relaxation, our shortest path tree remains the same as in Step-05.

Now, the sets are updated as-

Unvisited set : {e}

Visited set : {S , a , d , b , c}

8. Vertex ‘e’ is chosen.

This is because shortest path estimate for vertex ‘e’ is least.

The outgoing edges of vertex ‘e’ are relaxed.

There are no outgoing edges for vertex ‘e’.

So, our shortest path tree remains the same as in Step-05.

Now, the sets are updated as-

Unvisited set : { }

Visited set : {S , a , d , b , c , e}

8. Vertex ‘e’ is chosen.

This is because shortest path estimate for vertex ‘e’ is least.

The outgoing edges of vertex ‘e’ are relaxed.

There are no outgoing edges for vertex ‘e’.

So, our shortest path tree remains the same as in Step-05.

Now, the sets are updated as-

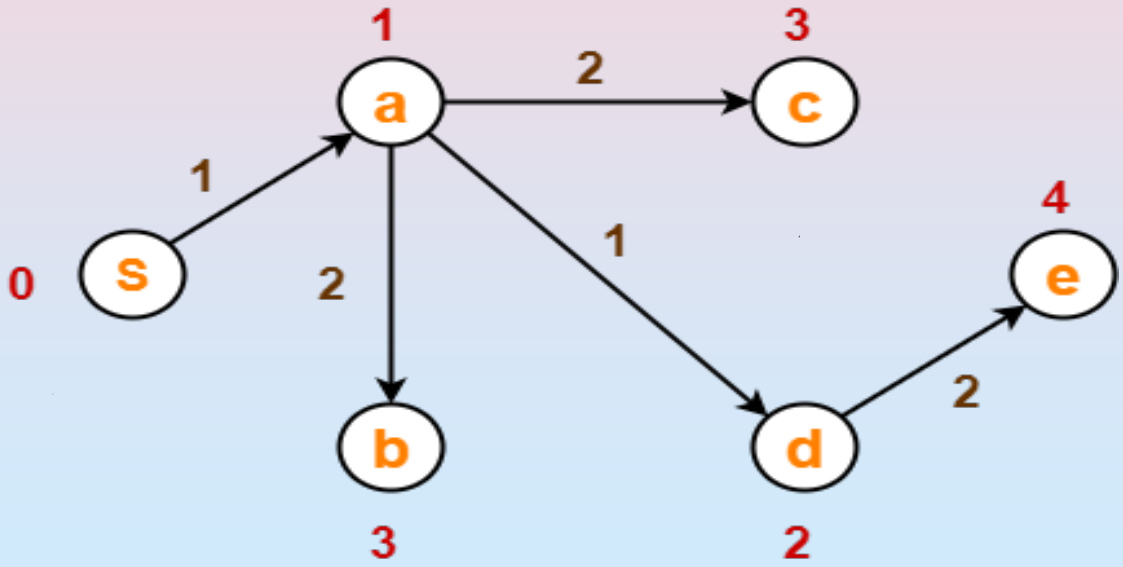
Unvisited set : { }

Visited set : {S , a , d , b , c , e}

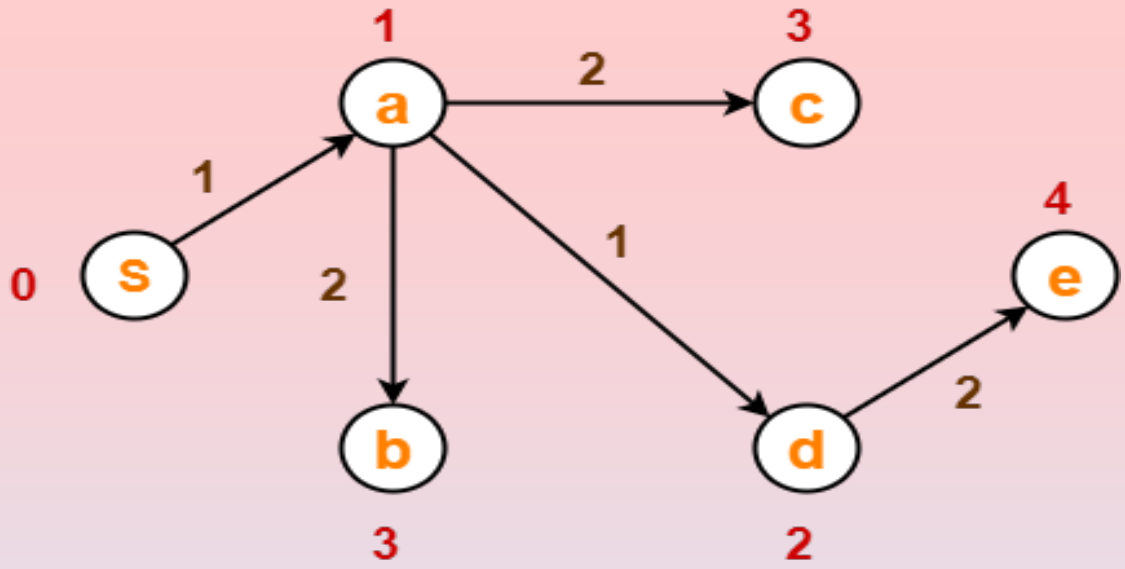
Now, All vertices of the graph are processed.

Our final shortest path tree is as shown below.

It represents the shortest path from source vertex ‘S’ to all other remaining vertices.



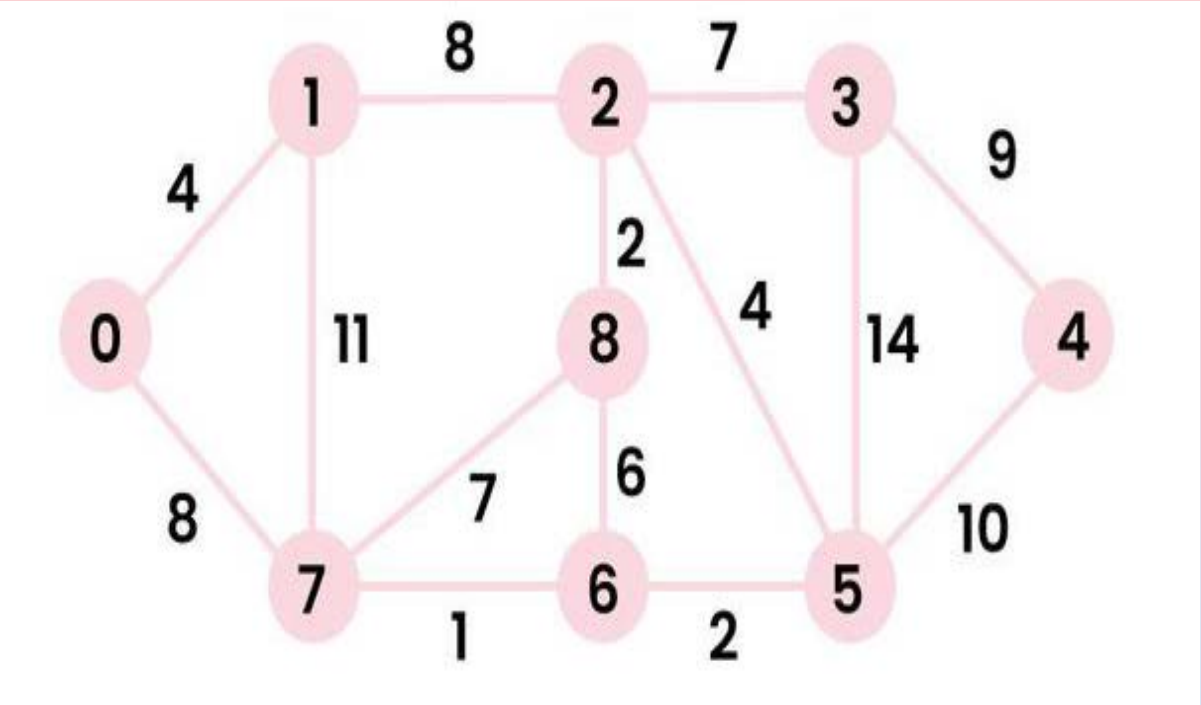
Shortest Path Tree

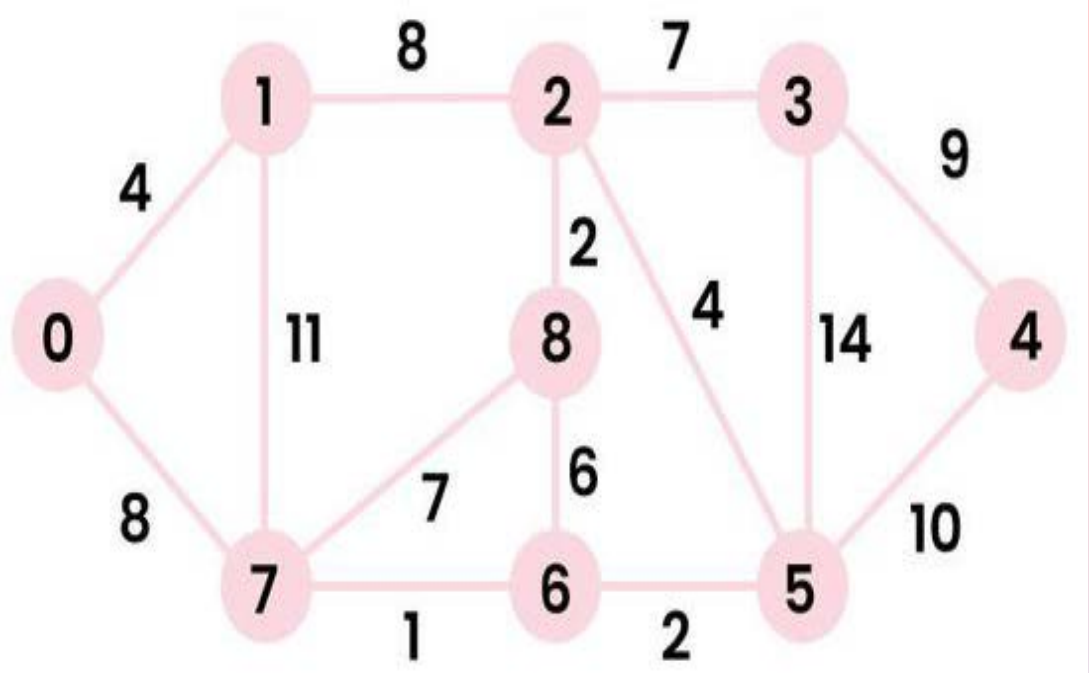


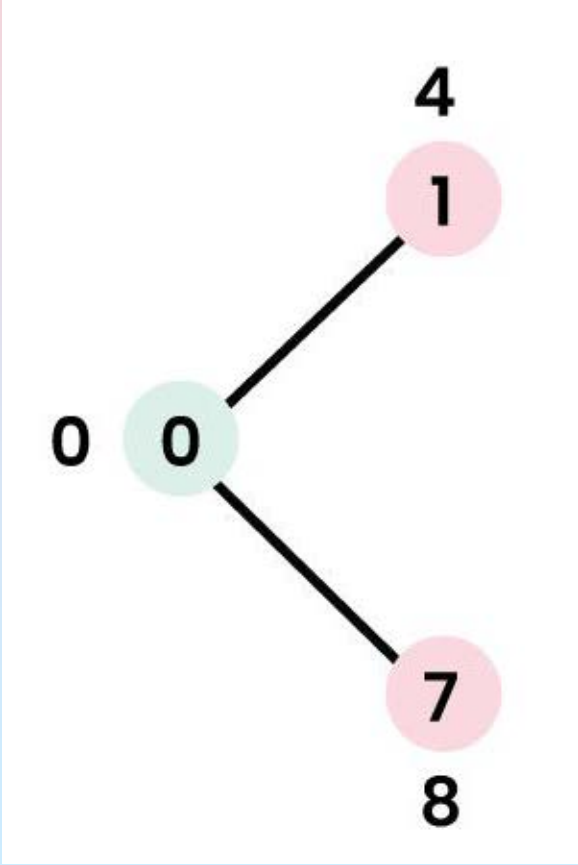
Shortest Path Tree

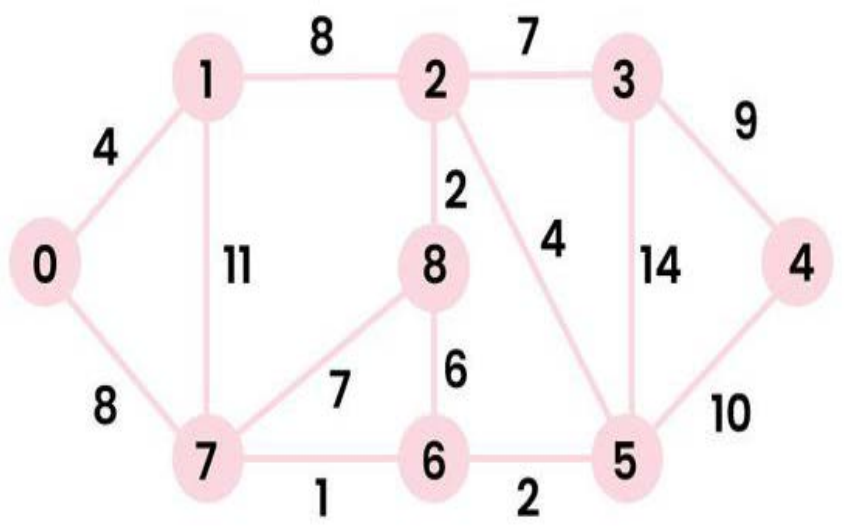
The order in which all the vertices are processed is :

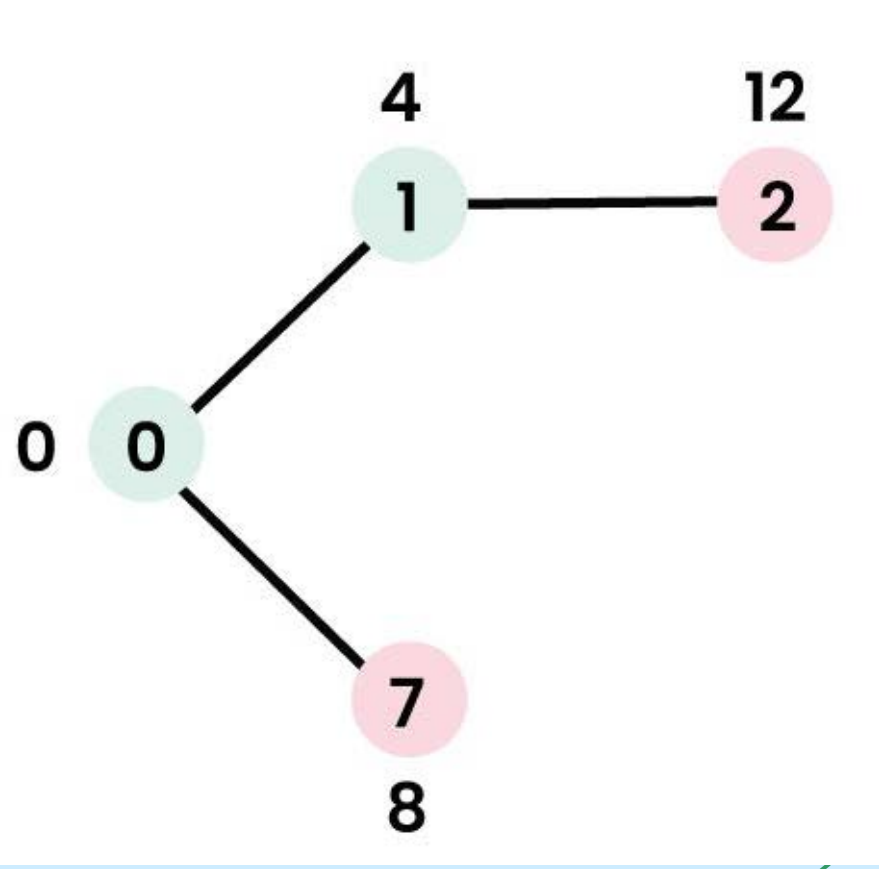
S , a , d , b , c , e.

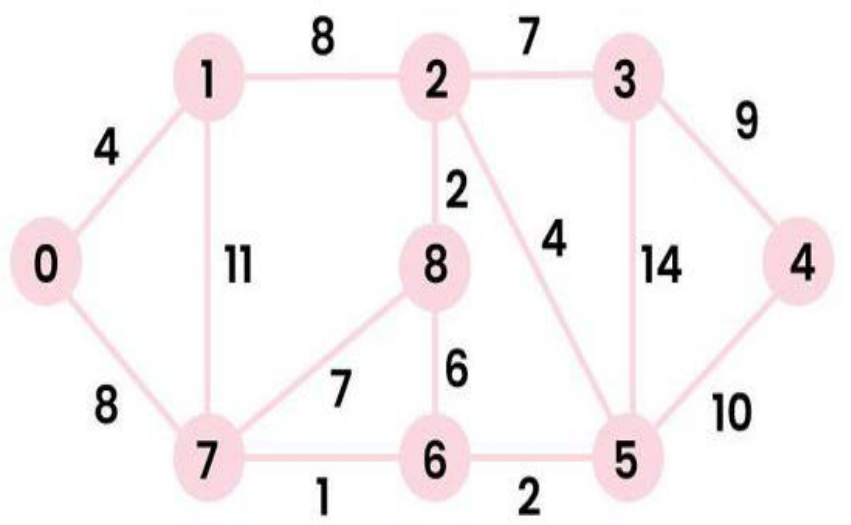

0
1
2
3
4
5
6
7
8
4
8
8
7
11
2
4
7
6
9
14
10
1
2

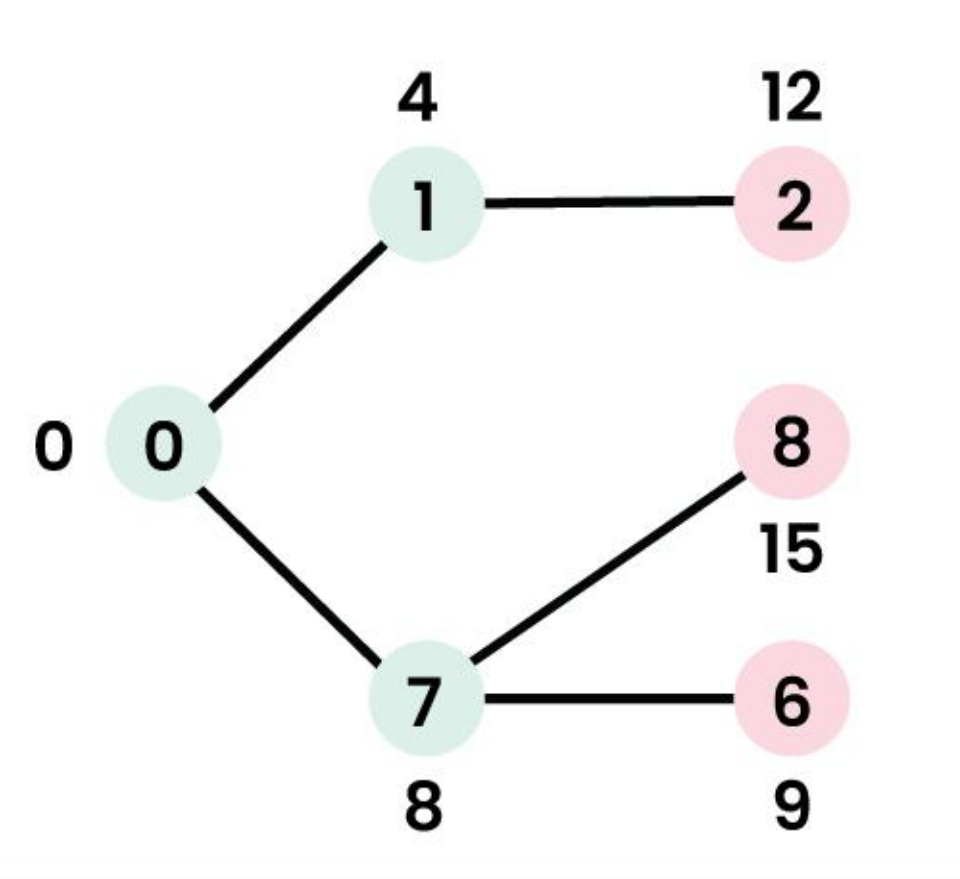

8
7
1
2
3
4
9
2
0
11
8
4
14
4
8
7
6
10
7
6
5
1
2

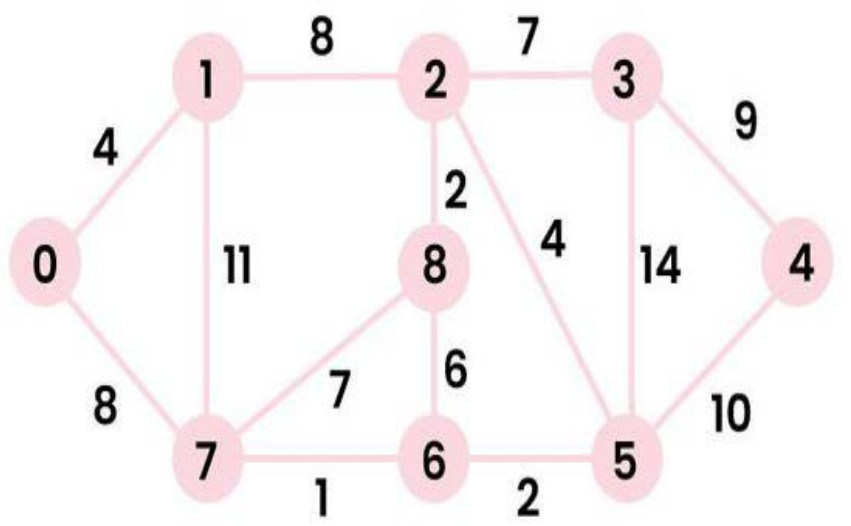

4
1
0
0
7
8

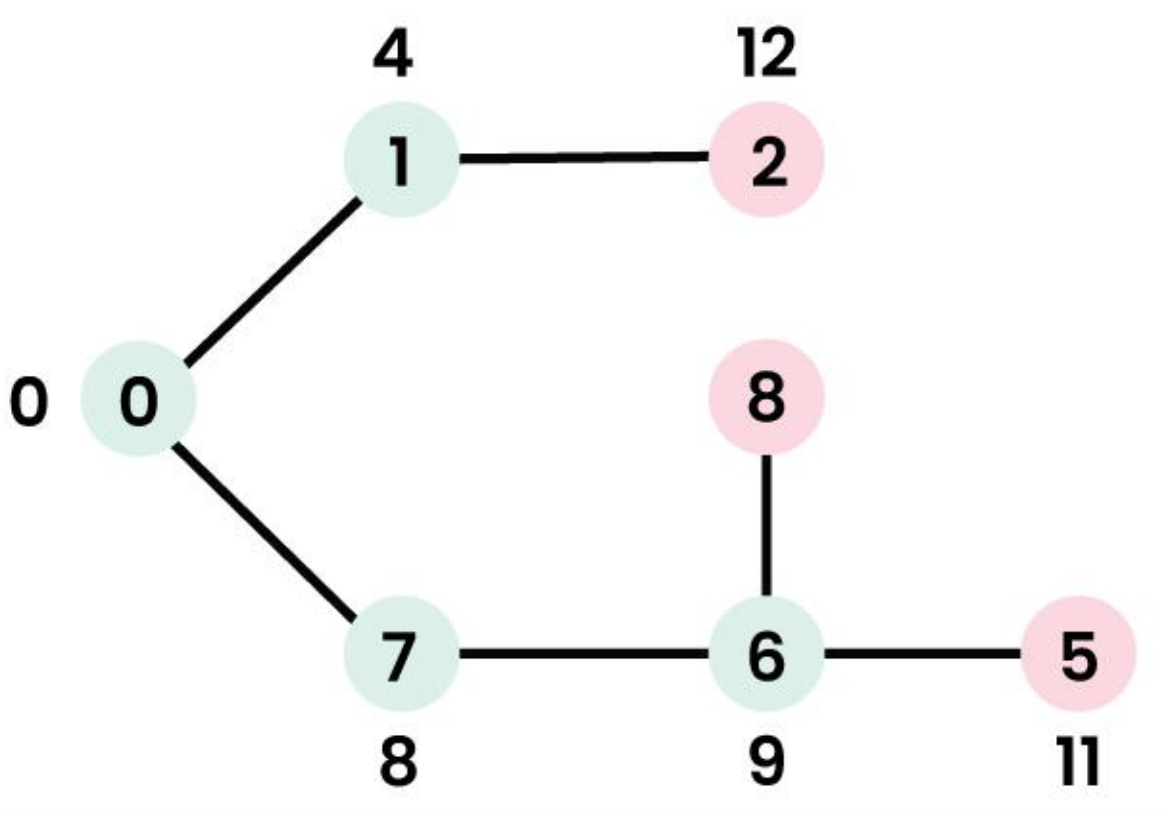

8
7
1
2
3
4
9
2
0
11
8
4
14
4
8
7
6
10
7
6
5
1
2

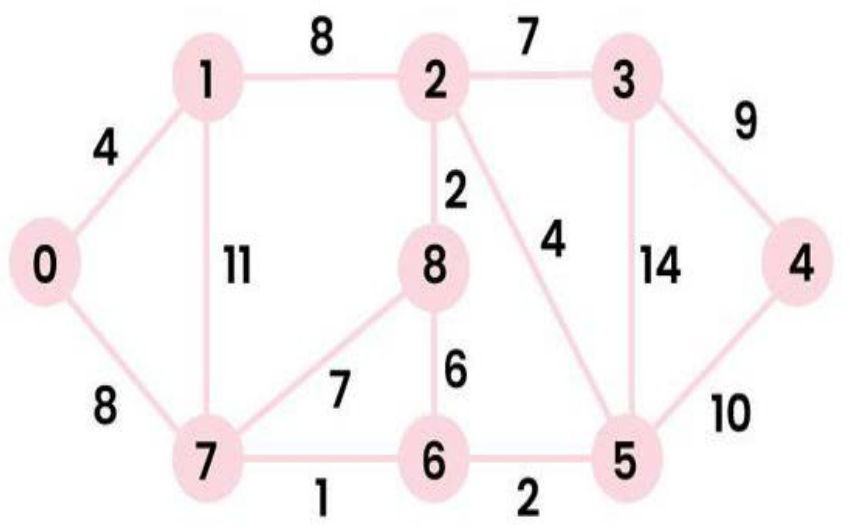

4
12
1
2
0
0
7
8

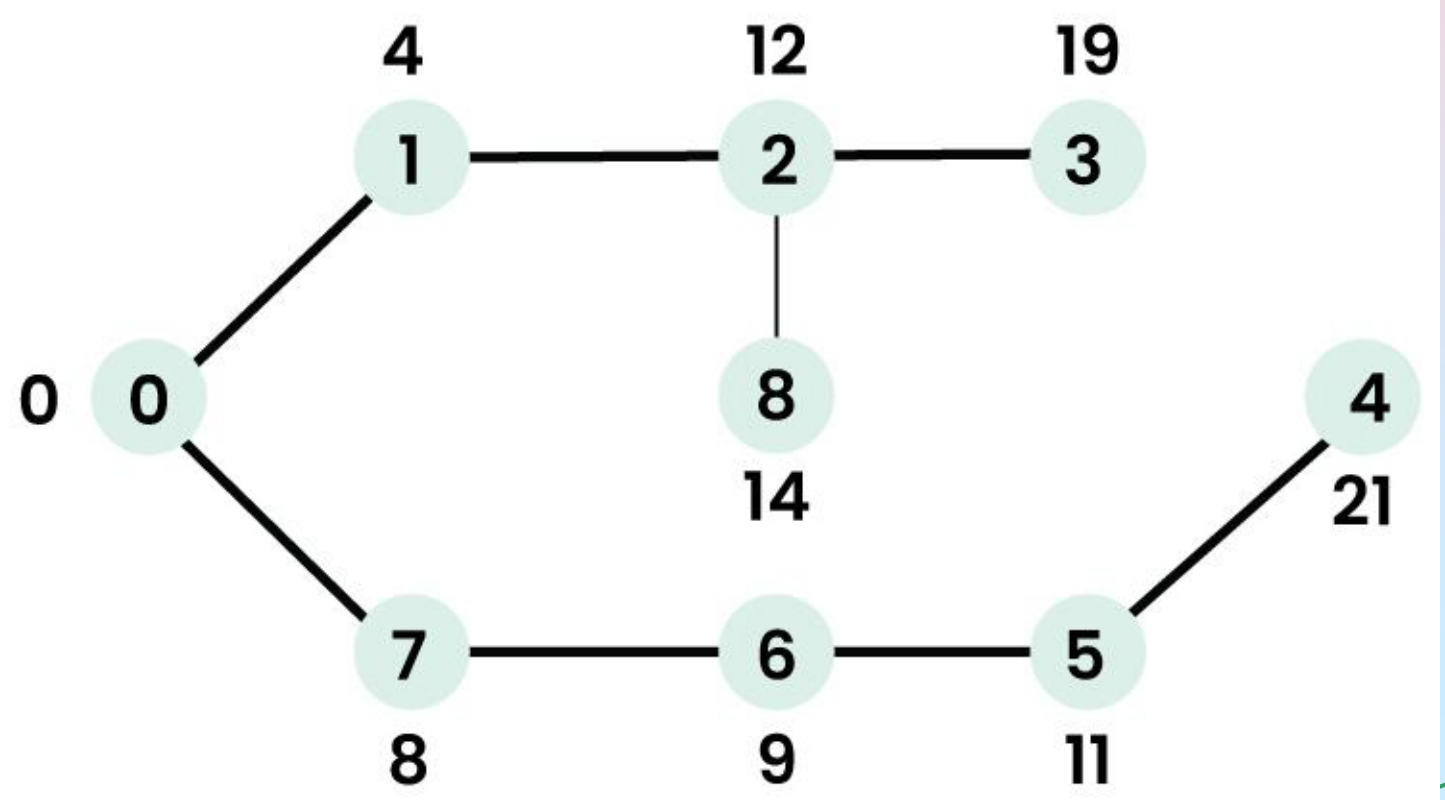

0
1
2
3
4
5
6
7
8
8
7
4
11
8
2
4
14
9
4
6
7
10
1
2


0
0
4
1
12
2
8
15
7
6
8
9


8
7
1
2
3
4
9
2
0
11
8
4
14
4
6
8
7
10
7
6
5
1
2


4
12
1
2
0
0
8
7
6
5
8
9
11


8
7
1
2
3
4
9
2
4
0
11
8
14
4
6
7
8
10
7
6
5
1
2


4
12
19
1
2
3
0
0
8
4
14
21
7
6
5
8
9
11