Prim’s Algorithm for MST

Prim’s Algorithm

- Prim’s Algorithm also uses Greedy approach to find the MST.
- In Prim’s Algorithm we grow the spanning tree from a starting vertex.
- Unlike an **edge** in Kruskal's, we add a **vertex** to the growing spanning tree.

Algorithm

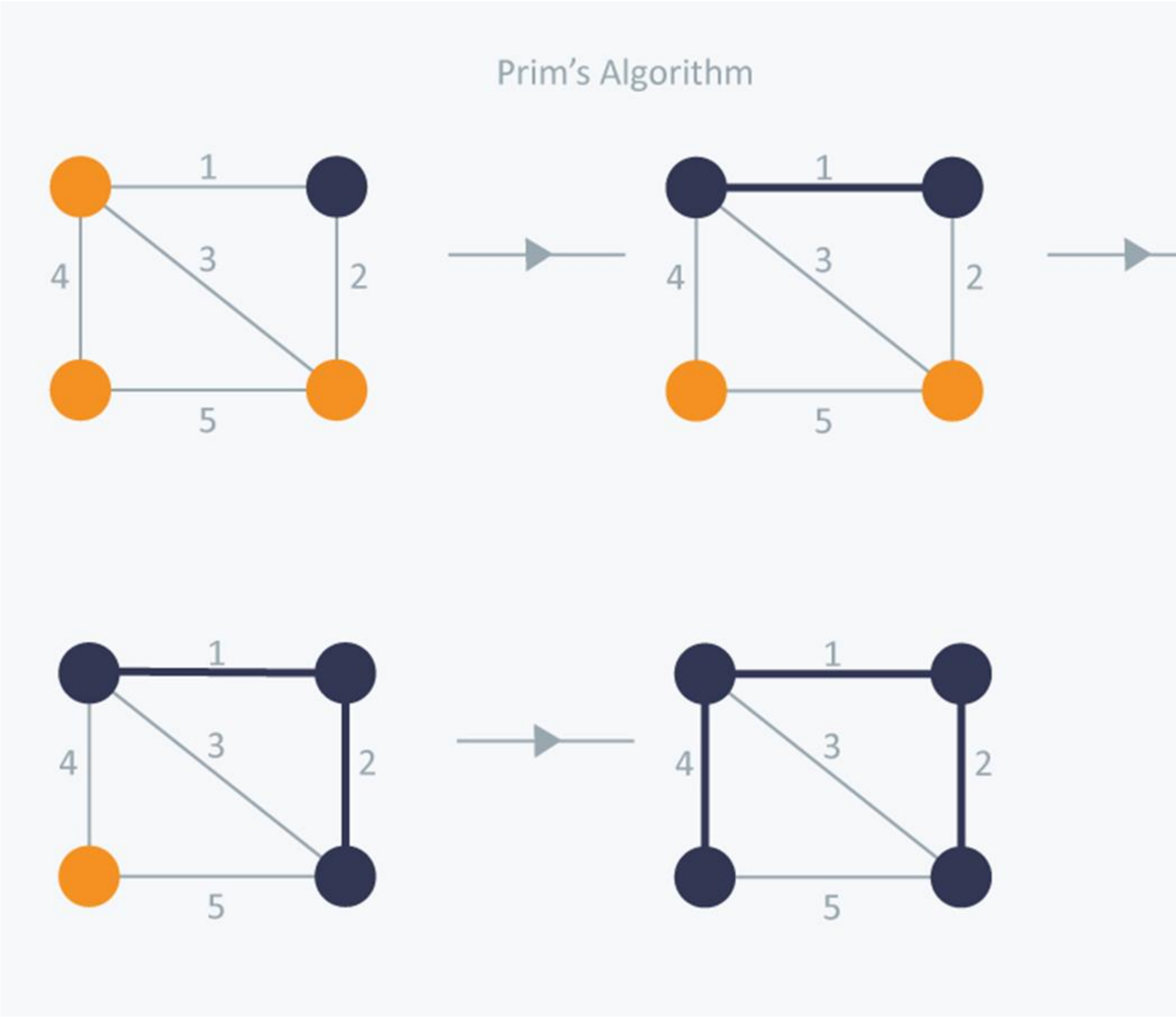
- Maintain two disjoint sets of vertices.
- One set contains vertices V1 that are on the tree
- other set are vertices V2 that are not yet inserted
- Examine vertices V2 which connect to vertices V1 on the spanning tree.
- V2 are inserted into a Priority Queue.
- Select the cheapest path from V2 to vertex in V1 and add it into the growing spanning tree.
- Do not include a vertex if it is going to create a cycle.

Psuedo Code for Prim’s Algorithm

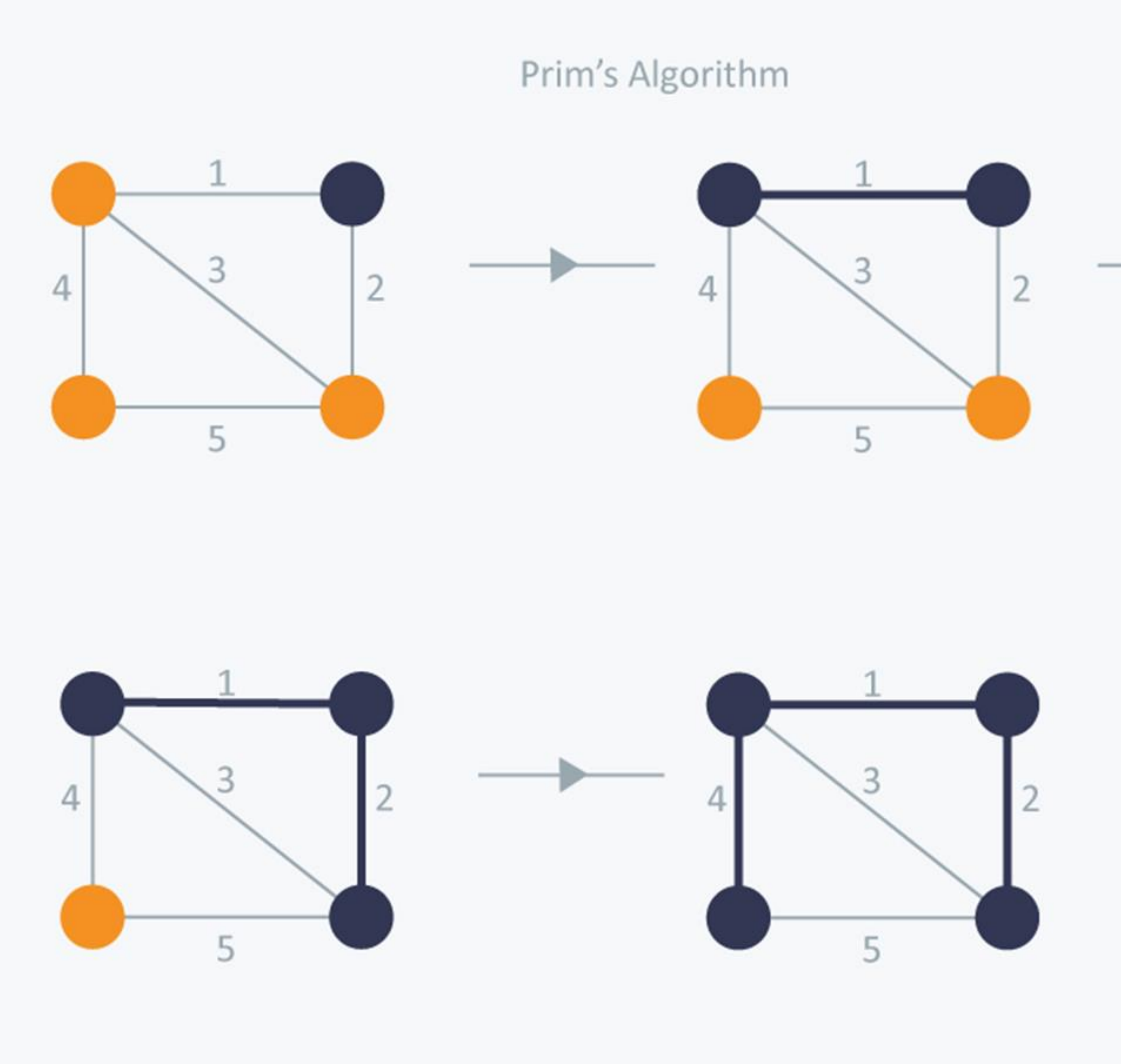
G : graph, Q: Priority Queue, w: weight function,   r: root node
Adj[u]: list containing neighbors of u

```
PRIM(G, w, r):
  for each u in G:
     u.key = INF
     u.p = NIL
  r.key = 0
  Q = G
  while Q is not empty:
    u = EXTRACT-MIN(Q)
    for each v in Adj[u]:
      if v in Q and w(u, v) < v.key:
         v.p = u
         v.key = w(u, v)
  return G
```

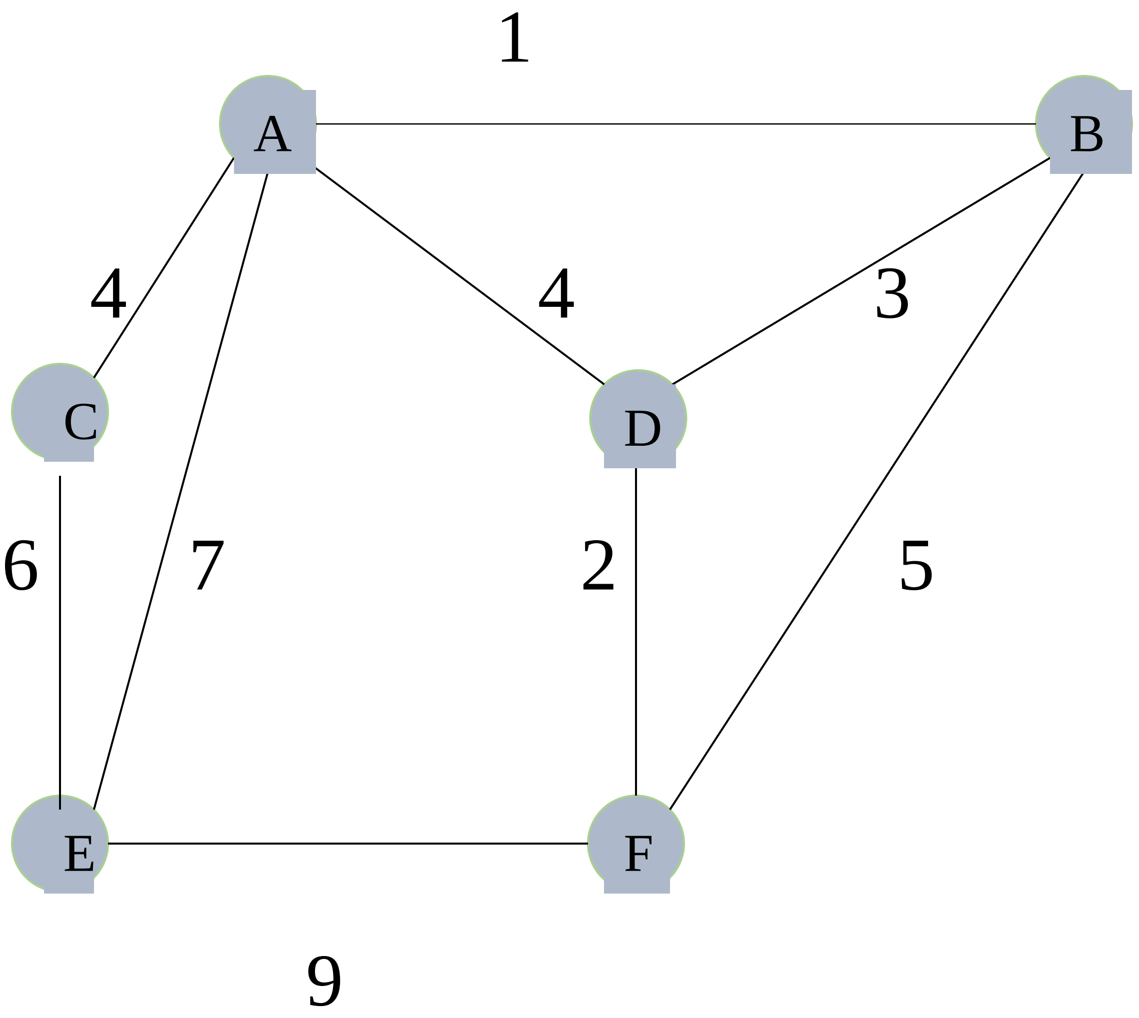
- Since all vertices have to be on the spanning tree, we arbitrarily choose a vertex. See left top graph.
- 2 vertices are connected to it, one with weight 1, one with weight 2.
- So we will simply choose the edge with weight 1.
- Now there are 2 vertices on MST.
- We have three options, edges with weight 2, 3 and 4 which connect to these 2 vertices.
- So, we will select the edge with weight 2 and mark the vertex.



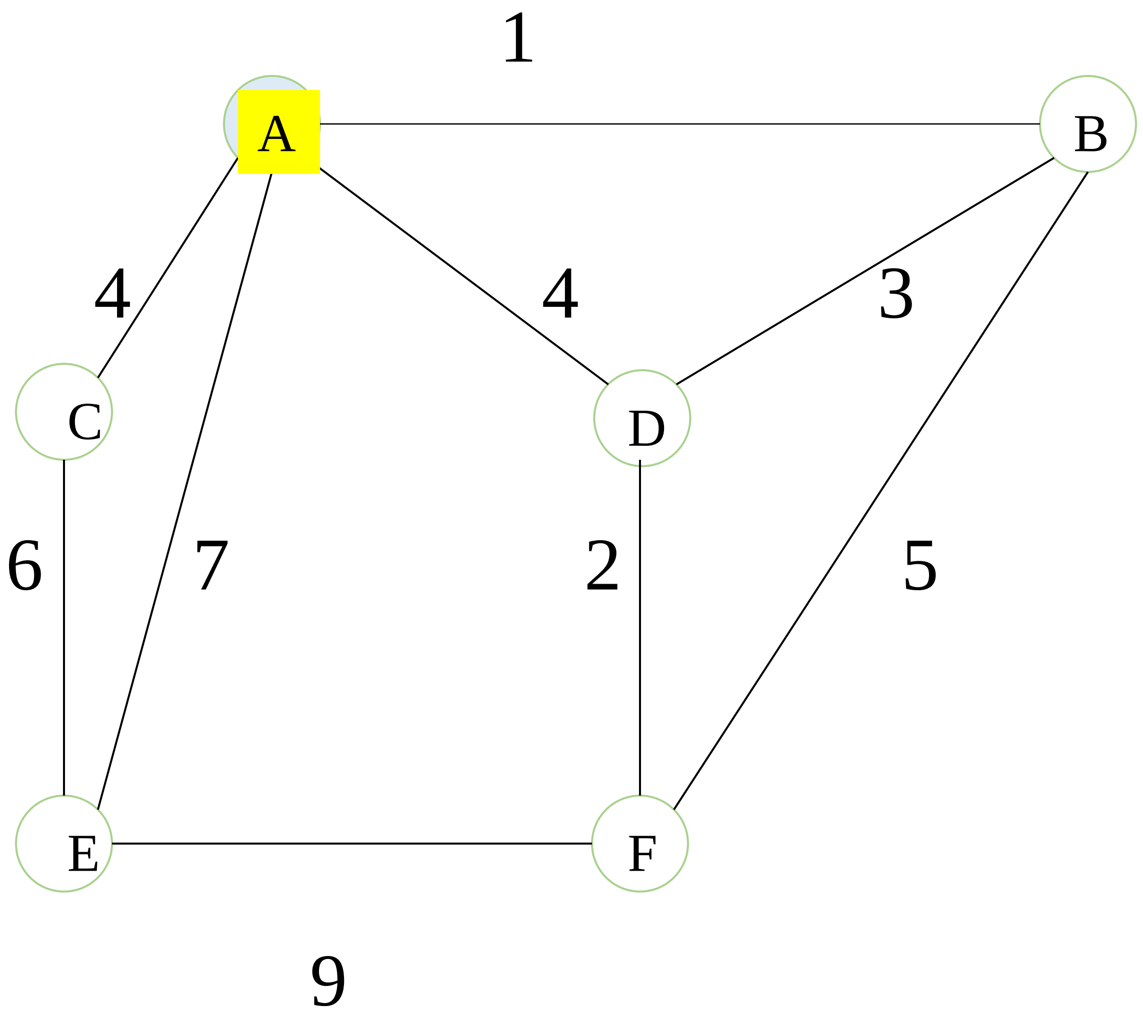
- Now the options are edges with weight 3, 4 and 5.
- But we can't choose edge with weight 3 as it is creating a cycle.
- So we will select the edge with weight 4.
- Edge with weight 5 is rejected.
- We end up with the minimum spanning tree of total cost 7 (= 1 + 2 +4).



Use Prim's algorithm to create MST for the following graph.



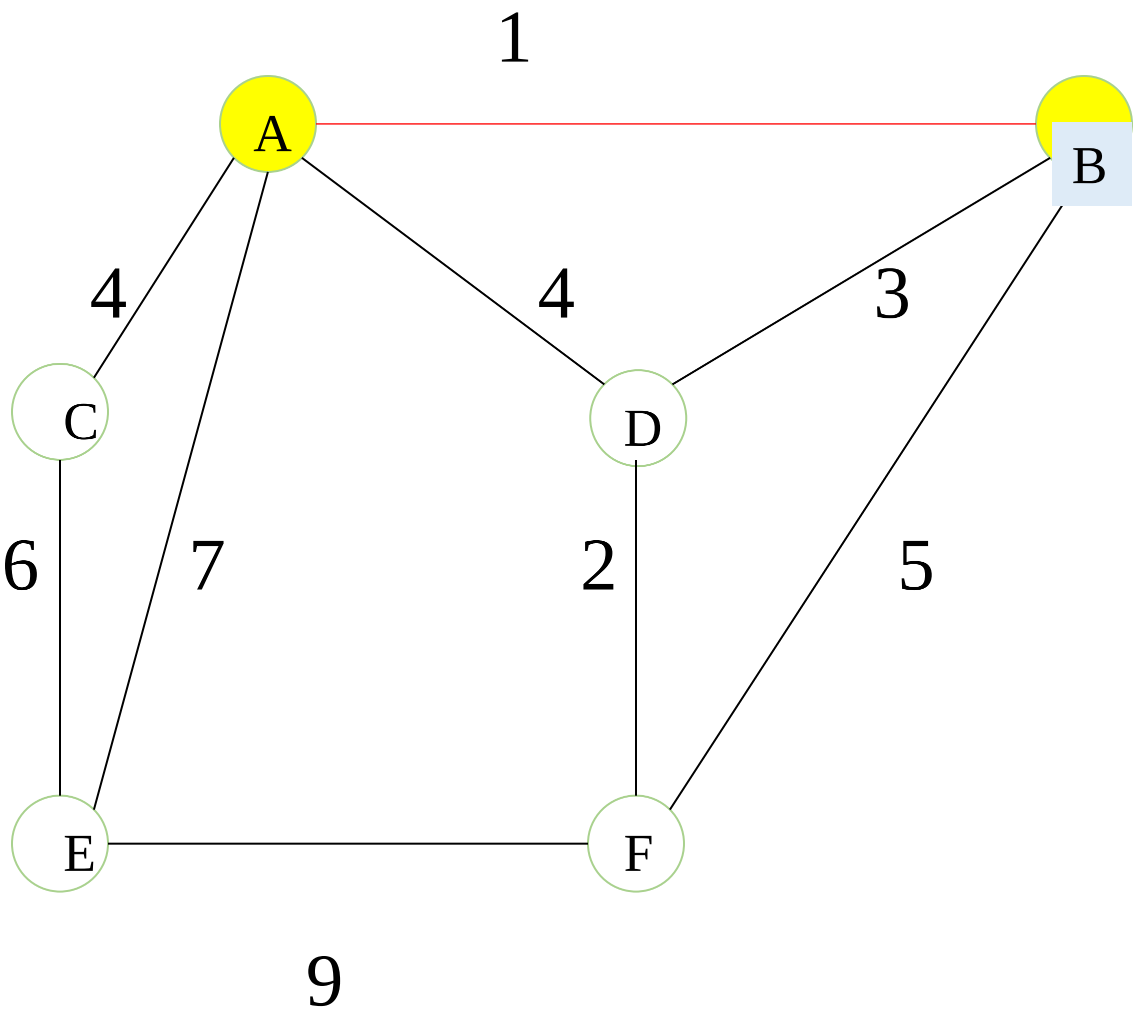
Put vertex A on spanning Tree



Node A is connected to
B, C, D, E

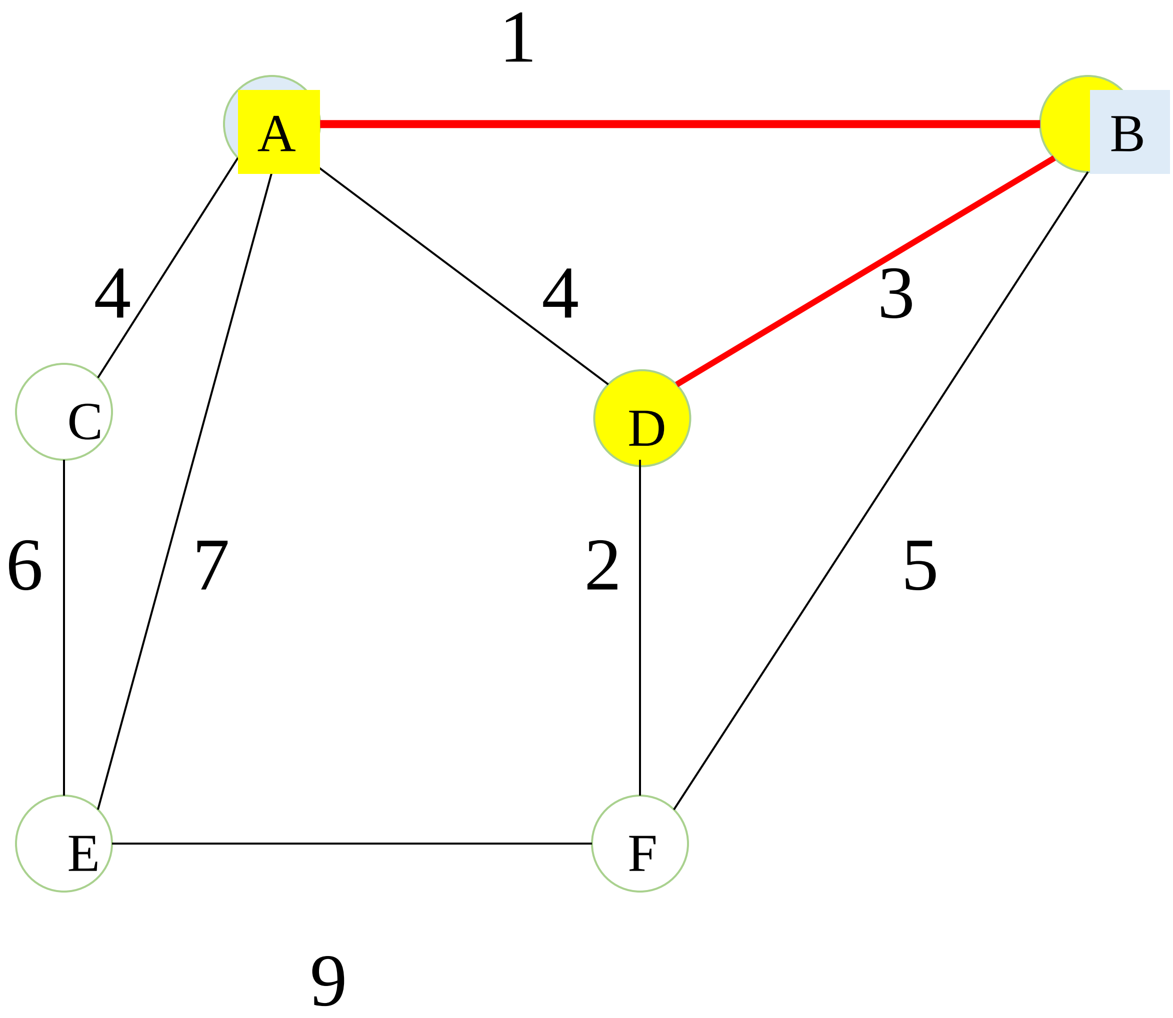
Given BA, CA, DA, EA.
BA is shortest
Min d[B] = 1

select B



Node A is connected to C and D.
AC=AD= 4

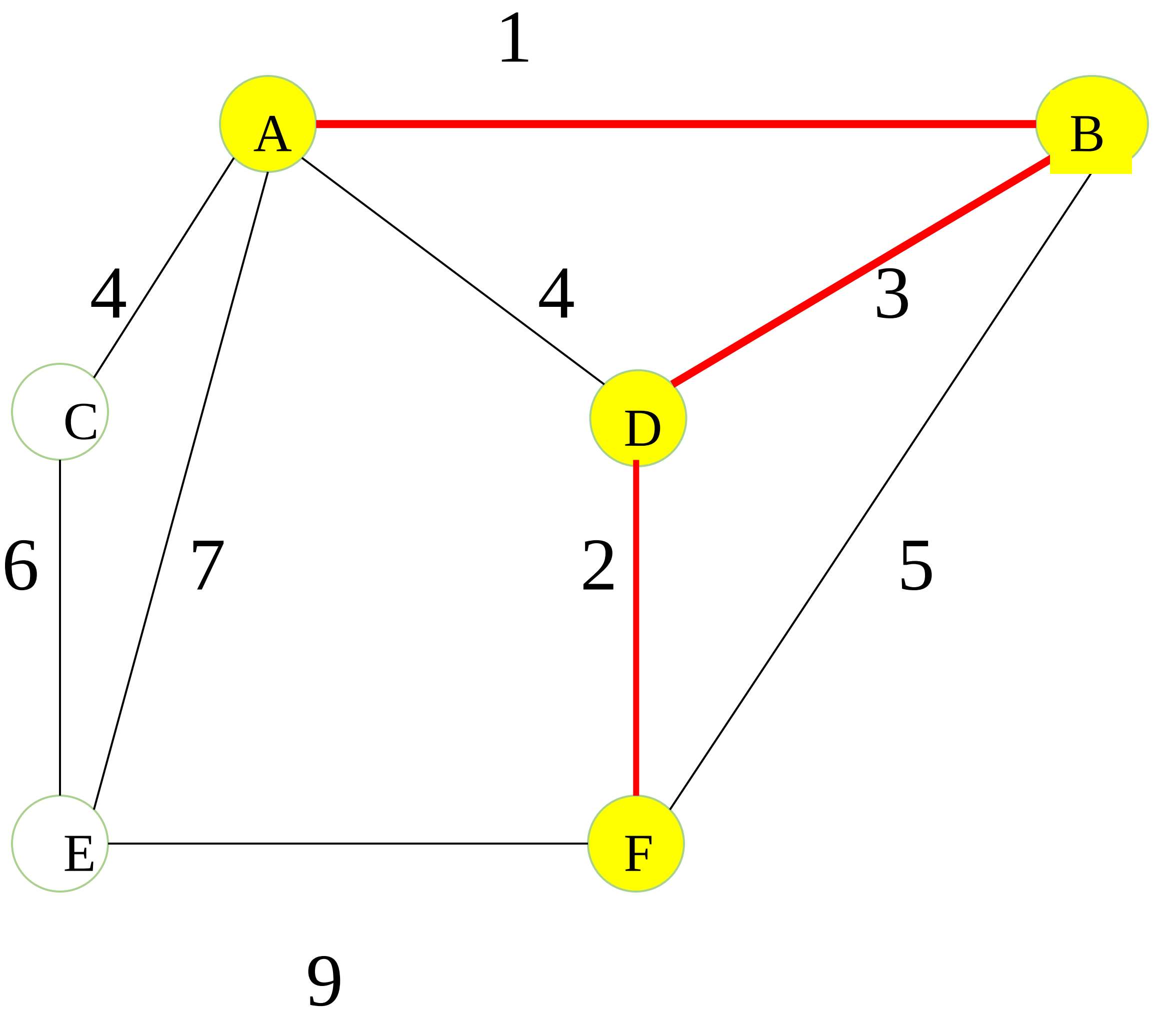
Node B is connected to D and F
DB = 3
FB = 5
select D (closest to B)



F connected to D
F connected to B

FD = 2
FB = 5
EA = 7
CA=4

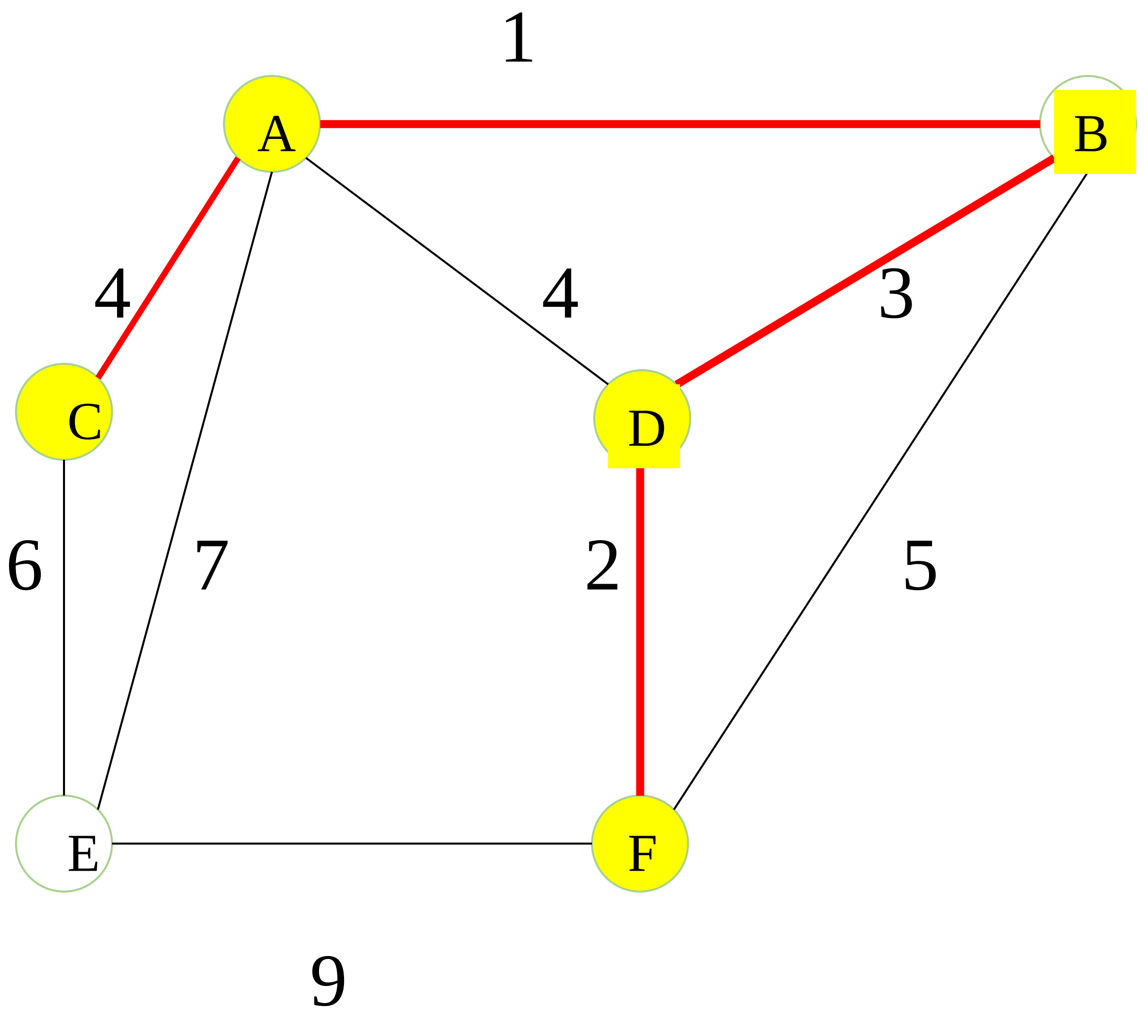
F is closest to D
Select F



CA : 4
EA : 7
EF = 9

C is closest to node A of MST

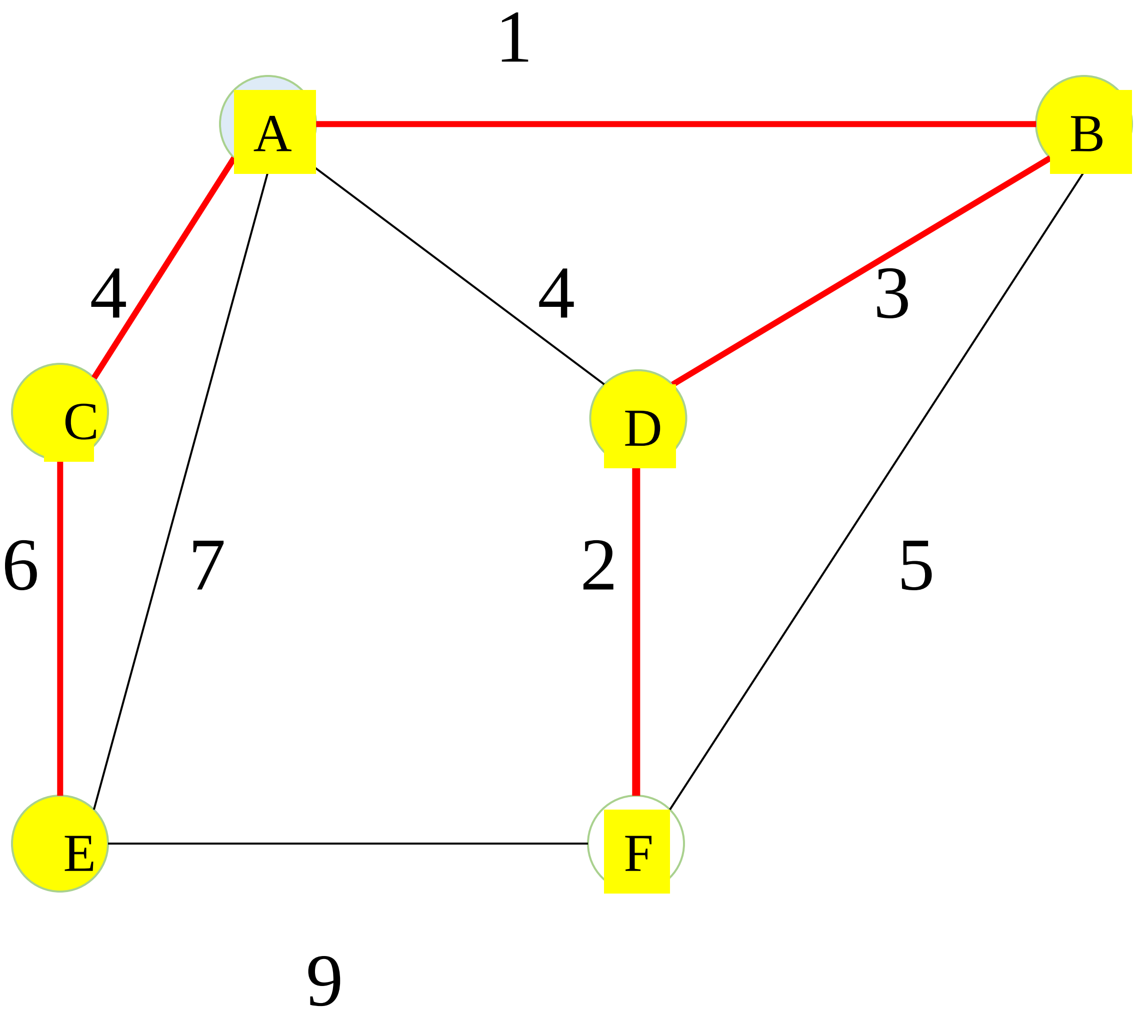
Select C



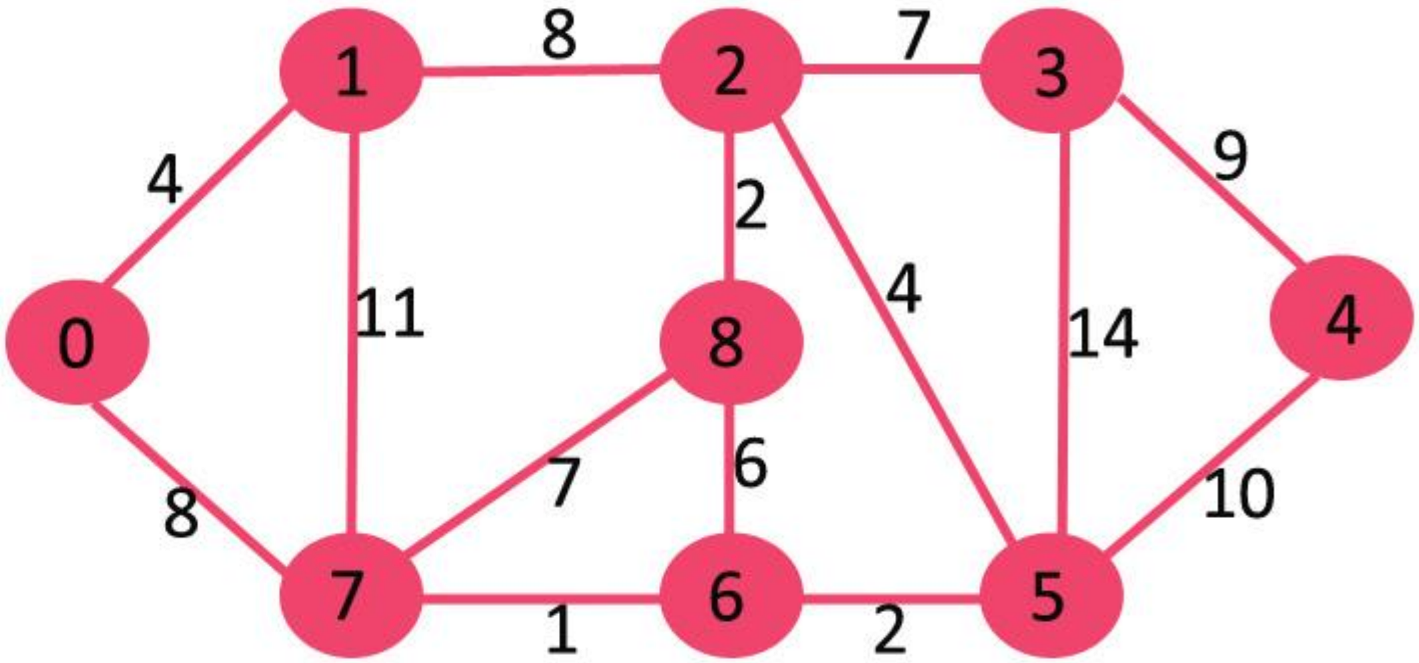
EA : 7
EC= 6
EF=9

E is closest to C of MST

Select E

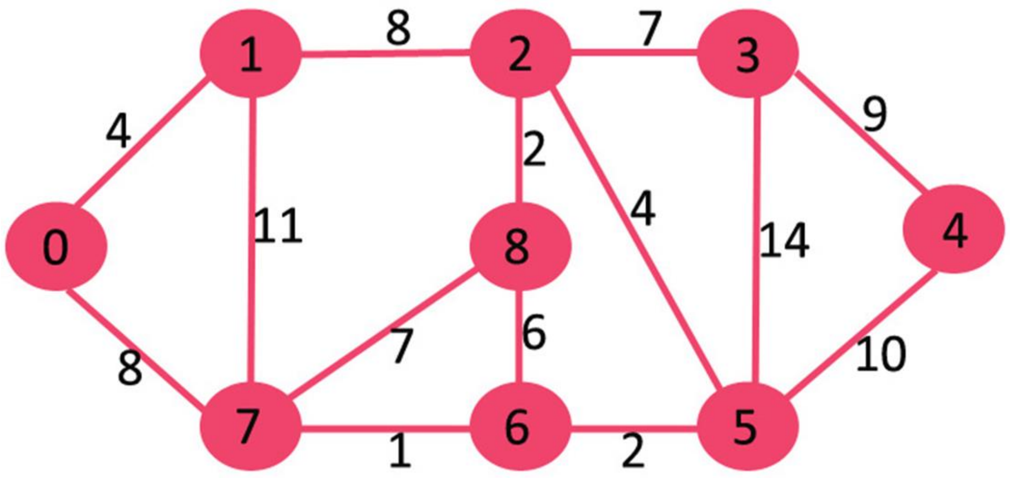


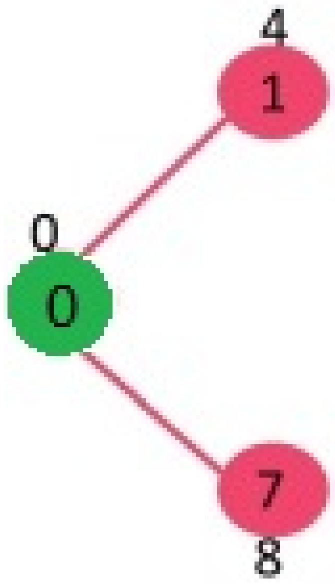
MST



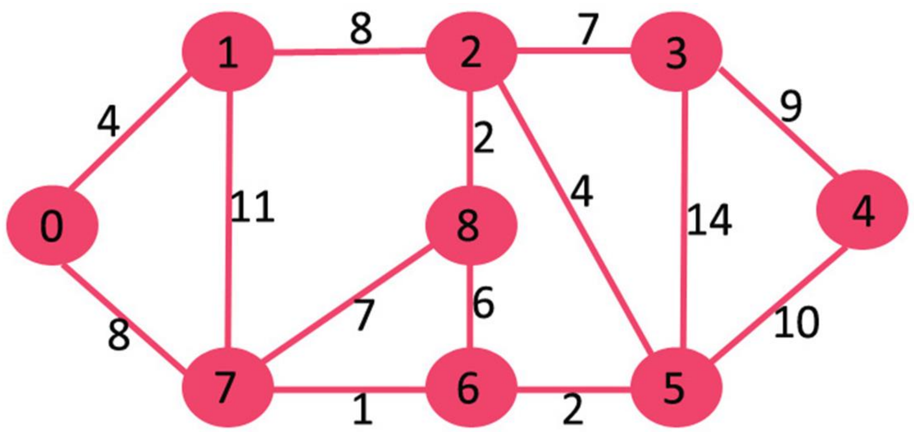
0
1
2
3
4
5
6
7
8
4
8
11
8
7
2
7
6
1
2
4
14
9
10

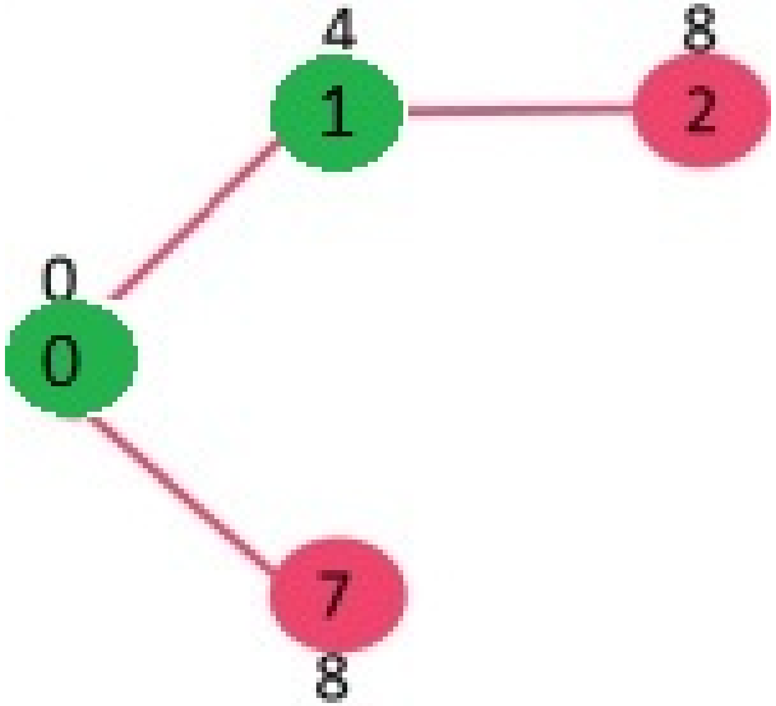
• Adjacent nodes of 0 are 1 and 7



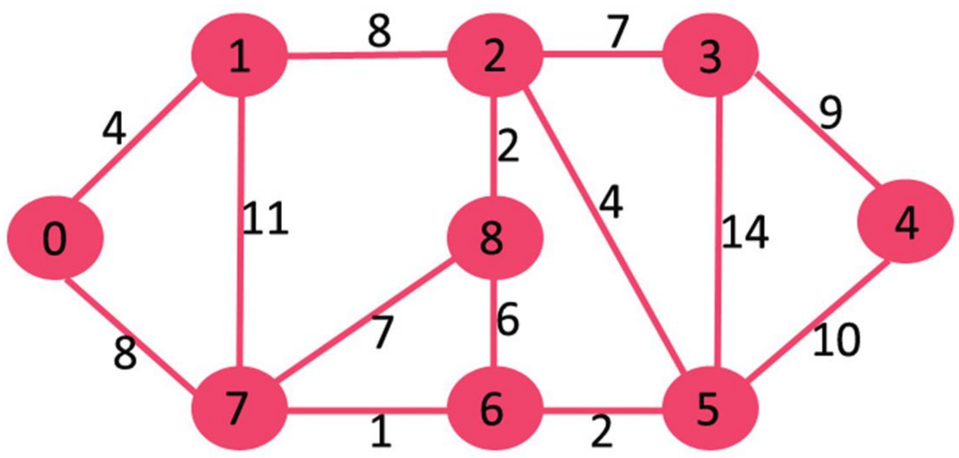


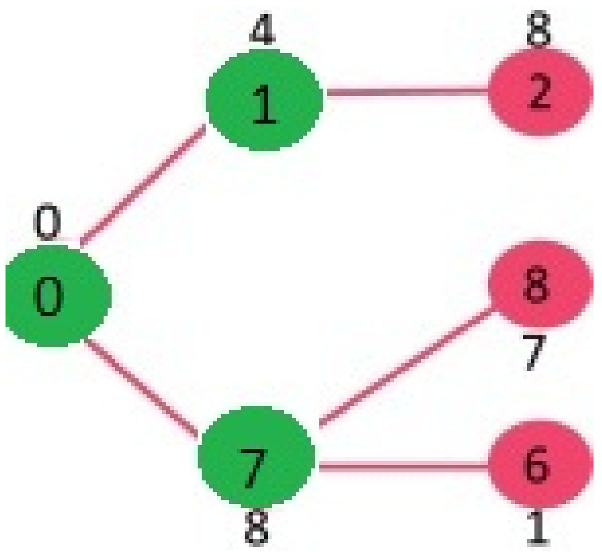
- 1 is selected as it is minimum.
- Then either 2 or 7 can be selected.



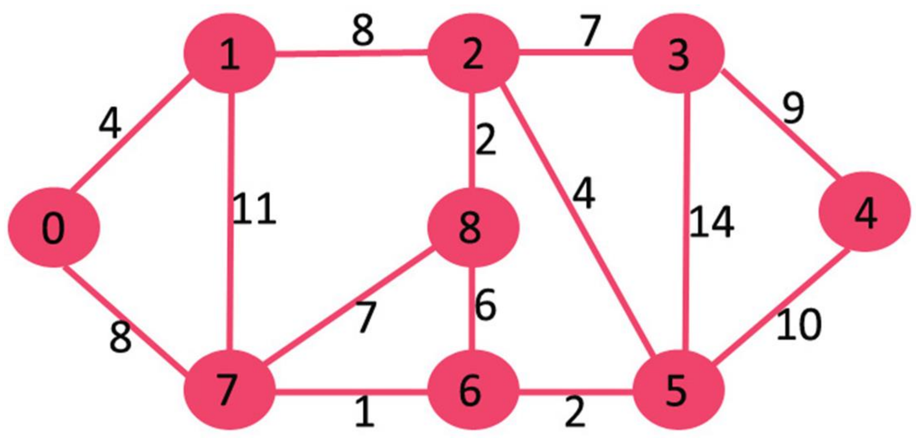


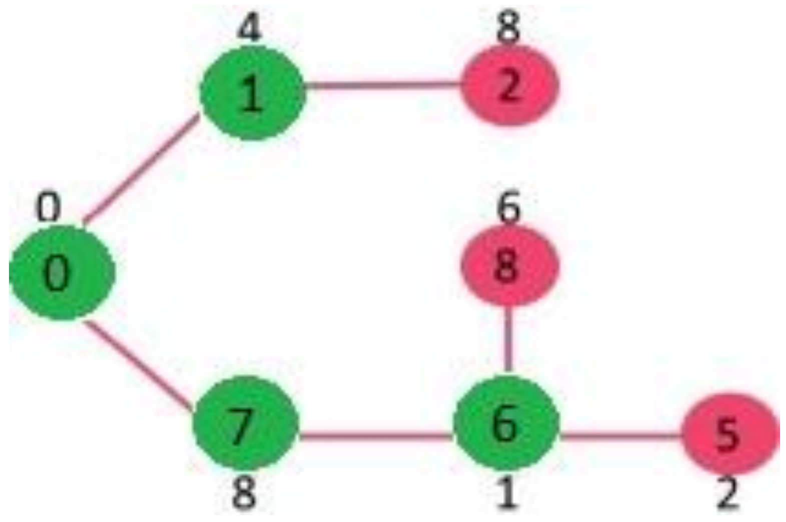
- We choose 7.
- Now we have choice between vertices 2,6 and 8

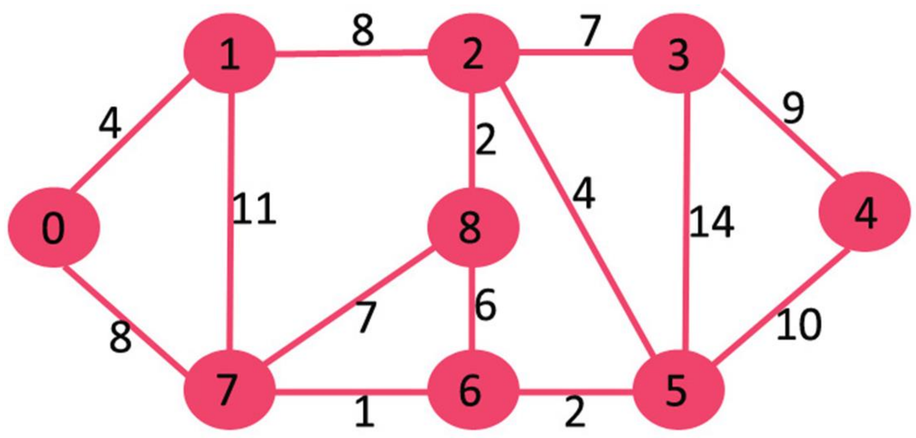




- 6 is picked up as it connects with minimum edge weight
- 5 and 8 are updated.
- 5 is picked, as it has min. edge weight.

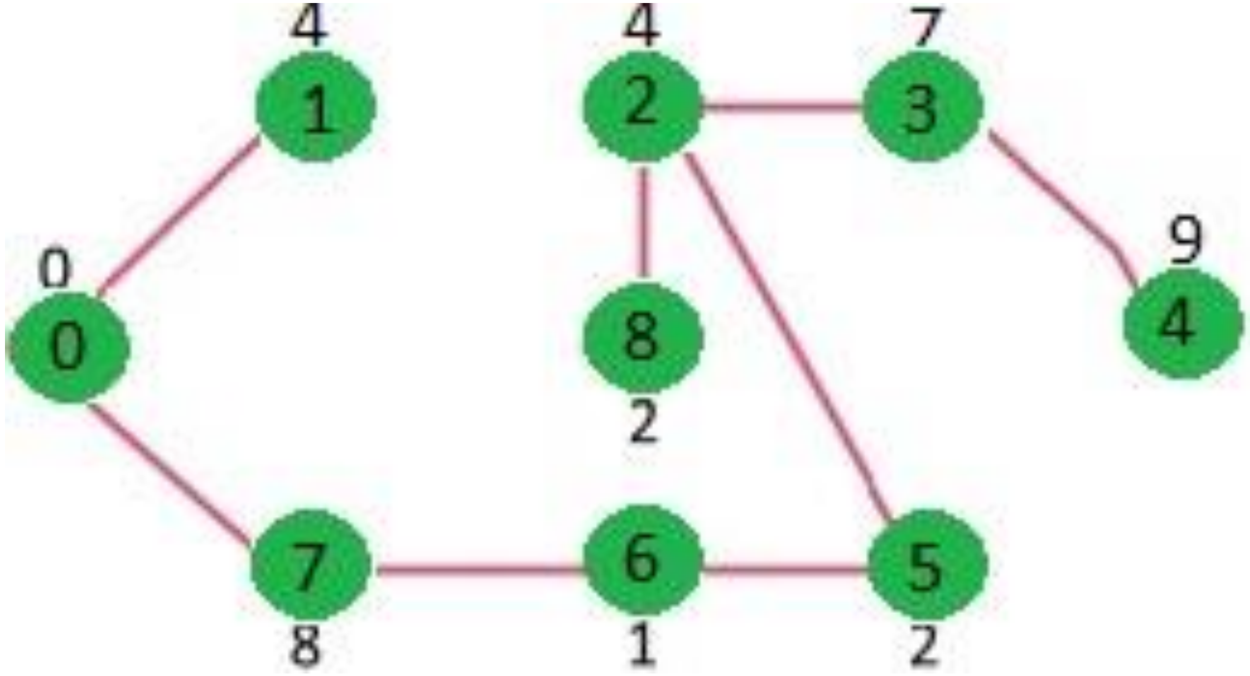






- 6 is picked up as it connects with minimum edge weight
- 5 and 8 are updated.
- 5 is picked, as it has min. edge weight.
- 2 is put on tree, with edge to 5 chosen.
- next 8 connects to 2.
- followed by 3 and then 4.

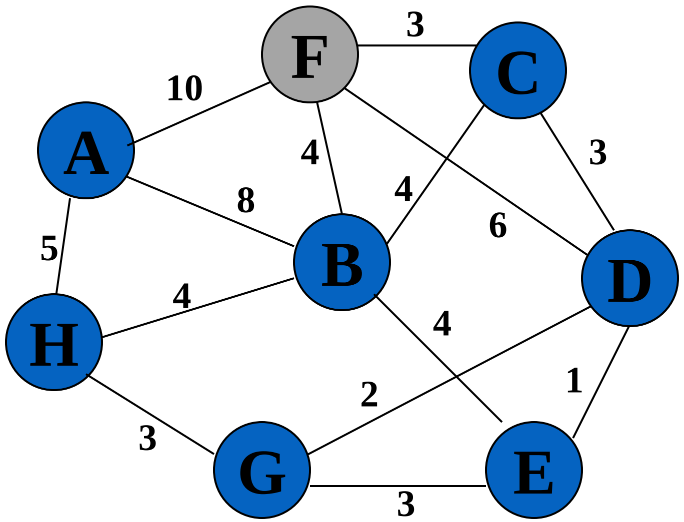
- Finally the MST.



Time Complexity

- Heap tree operation takes log V.
- If the input graph is represented using an adjacency list, then the time complexity of Prim's algorithm is O(E log V) with the help of a binary heap.
- In this implementation, we are always considering the spanning tree to start from the root of the graph

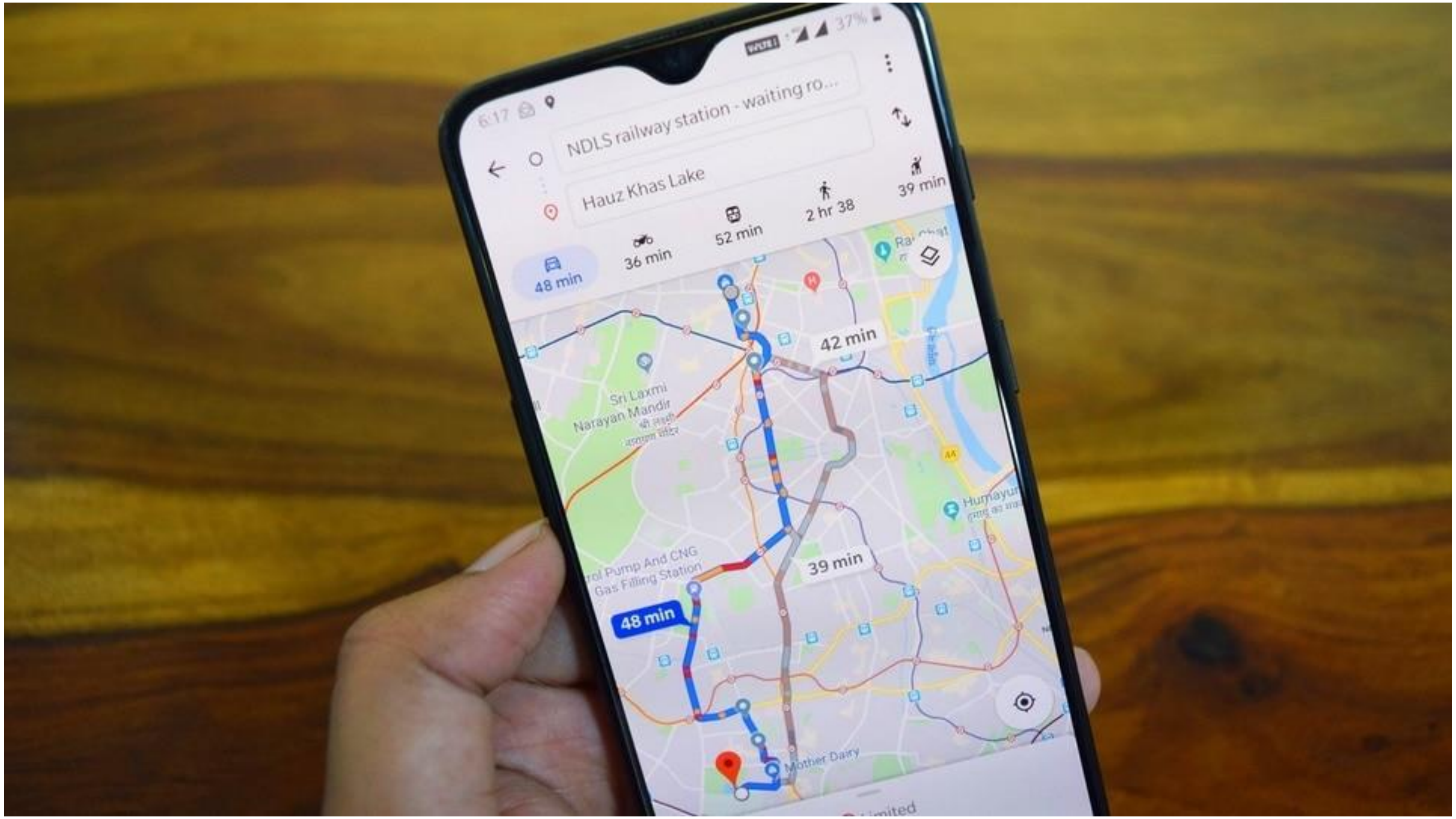
- If some of the edge weights in a graph are same, there will be alternatives to choose from
- and it is possible that the MST will not be unique.



Consider an undirected, weight graph

Graphs- III
Shortest Path Problem

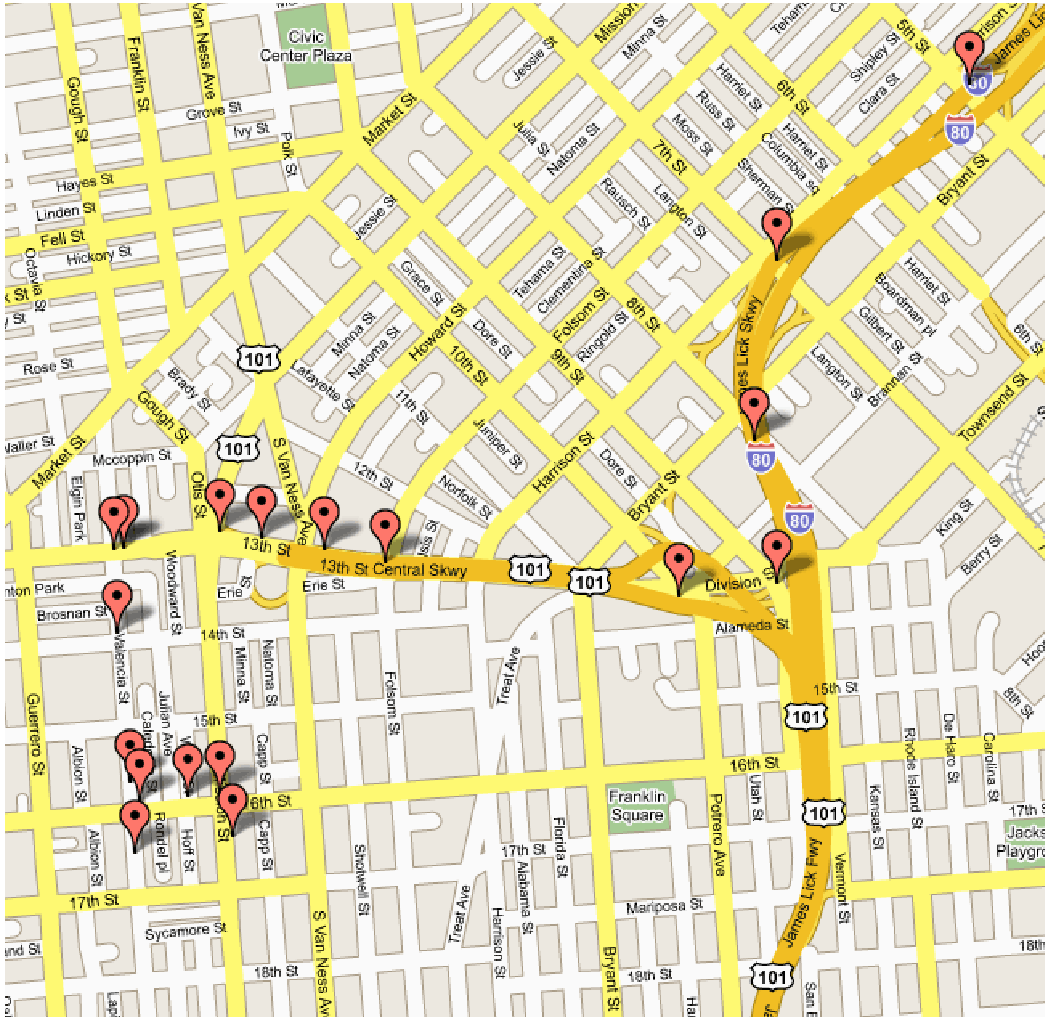
google maps

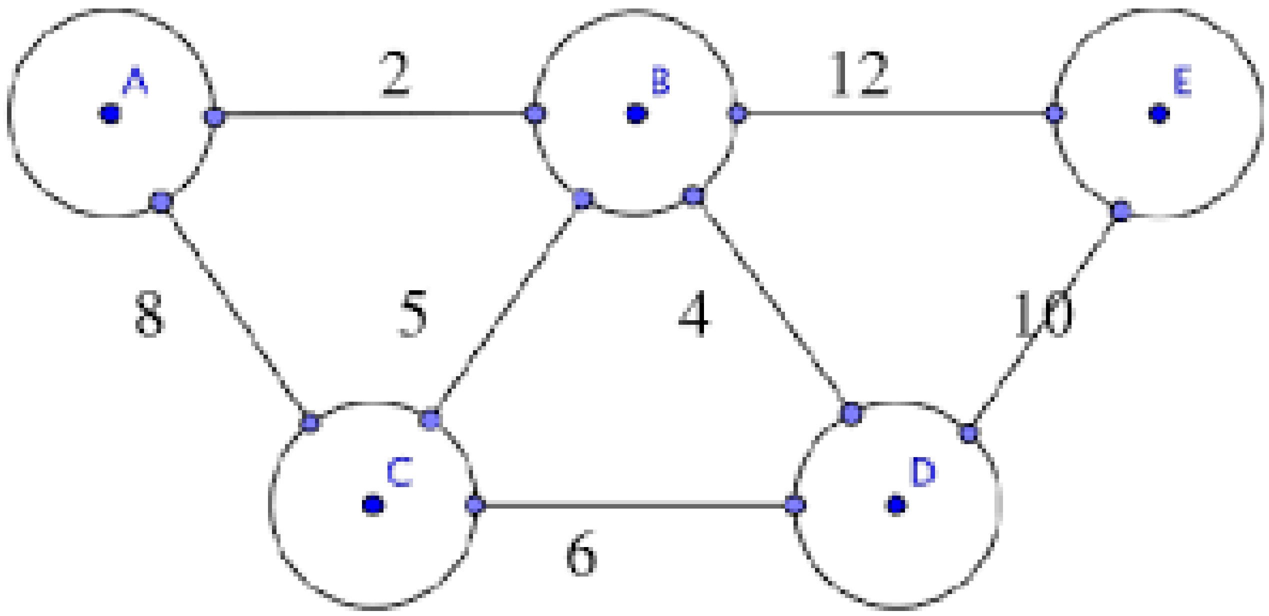

NDLS railway station - waiting ro...
Hauz Khas Lake
48 min
36 min
52 min
2 hr 38
39 min
42 min
Sri Laxmi Narayan Mandir
39 min
Gas Filling Station
48 min
Mother Dairy

- weights between nodes are the estimated time required to travel a particular distance, by considering the live traffic situation.

Applications

- Maps
- Routing Systems, Networks




A
B
E
C
D
2
12
8
5
4
10
6

Route finding problem

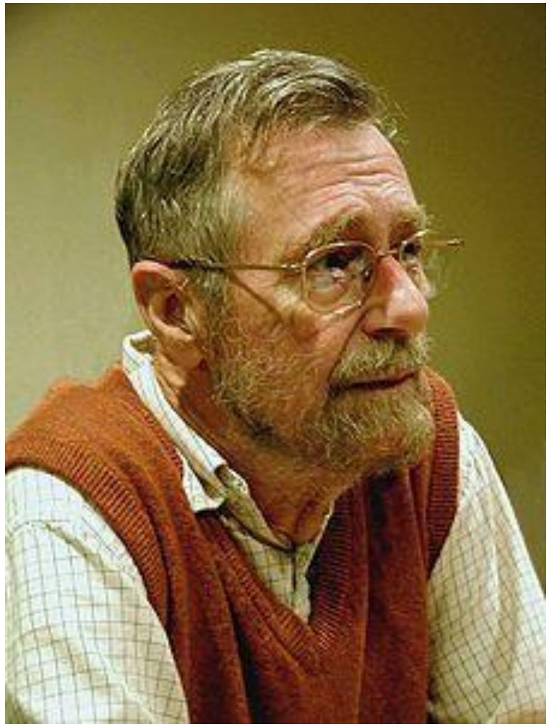
- There may be number of paths between vertex A and vertex B.
- Distances between any two directly connected vertices is specified on the graph
- The problem is to trace the path from A to B whose cost (distance) is least compared to all other possible paths.

Single-Source Shortest Path Problem

The problem of finding shortest paths
from a specified source vertex *v*
to all other vertices in the graph.

Dijkstra’s Algorithm
for
shortest path from single vertex

Edsger Wybe Dijkstra
(1930-2002)



Dutch computer scientist,
received the 1972 A. M. Turing Award, widely considered the most prestigious award in computer science.

Dijkstra's algorithm

Dijkstra's algorithm - is a solution to the single-source shortest path problem in graph theory.

Input:
Weighted graph G={E,V} and source vertex,

Output:
Lengths of shortest paths (or the shortest paths themselves) from the source vertex to all other vertices

Approach

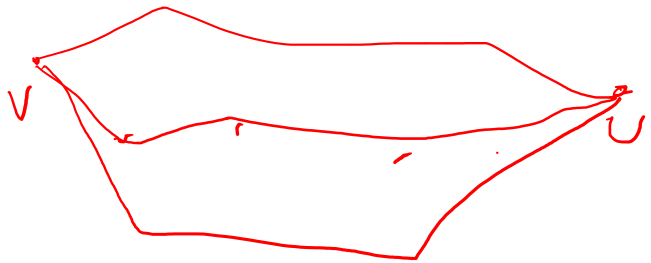
- The algorithm computes for each vertex u
  - the distance to u *from the start vertex v,*
  - that is, the weight of a shortest path between v and u.

- the algorithm keeps track of the set of vertices for which the distance has been computed, called P[v]

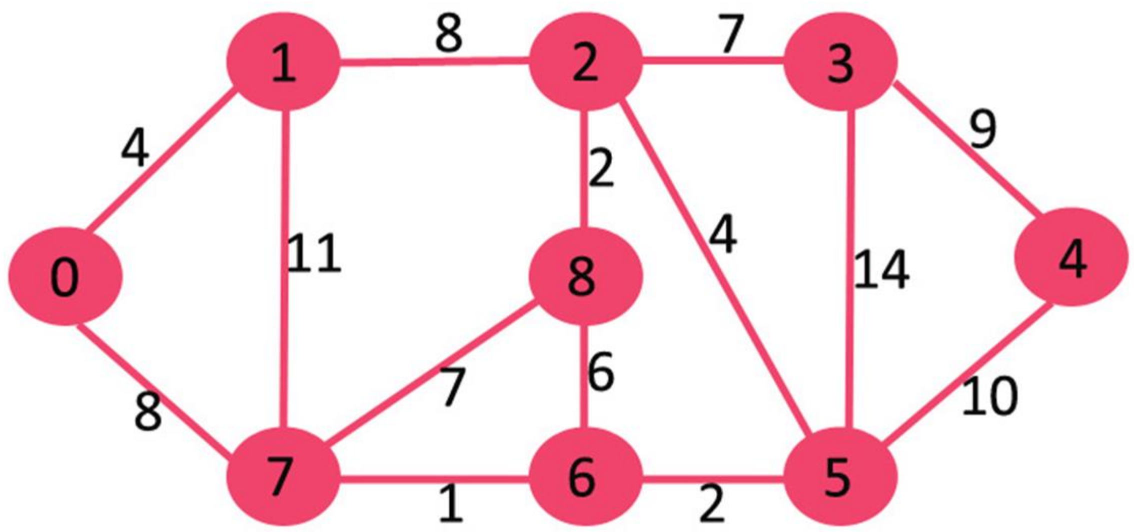
How does it work?

- Every vertex has a label D associated with it.
- For any vertex u, D[u] stores so far best distance between start vertex v and u.
- The algorithm examines *all paths from start v to vertex u*.
- It will update a D[u] value when it finds a shorter path from v to u.

- When a vertex u is finally selected, its label D[u] stores the actual (final) distance between the starting vertex v and vertex u.



- Just to illustrate the basic idea
- Let 0 be the source vertex in the same graph we had used earlier.
- Consider the path from vertex 0 to 8
- Suppose we have figured out a path 0 – 7 – 8 with cost 15
- We shall explore other longer paths as well
- like 0 – 1 – 2 – 8 . The cost of this pathis 4+8+2 = 14
- We shall prefer this path over the earlier one.



- The algorithm uses the greedy strategy
- At each stage it tries to figure out the shortest possible path to reach a vertex
- Then uses this path to explore paths to other vertices.

Dijkstra’s shortest Path method

- Create a null set SP (shortest path tree set), to keep track of vertices for shortest-path tree

- Assign *distance value* to all vertices in the input graph.
  - Assign 0 as distance value for the source vertex.
  - Initialize all other distances to INFINITY.

- Keep track of vertices included in shortest-path tree, by calculating minimum distances from the source as newer paths are discovered.

- While set SP doesn’t include all vertices
  - Pick a vertex u (not yet in SP) which has a minimum distance value.
  - Include u in set SP.
  - Update distance value of all adjacent vertices of u.
  - To update the values, iterate through all adjacent vertices.
  - For every adjacent vertex v, if

    distance to u + weight of edge u-v < the distance value of v

    , then update the distance value of v.

Dijkstra’s greedy strategy

Let current distance from source to v be d[v].

Distance to u + new distance from u to v = $d[u] + w(u,v)$

If $d[v] > d[u] + w(u, v)$ then a better path has been found.

Replace each d[v] by this new distance’

Also make a note that best path to v is through vertex u.

Suppose old $d[v] = 18$, for new u, $d[u] = 12$, $w(u,v) = 3$

then since $12+3 < 18$,

better $d[v] = 15$

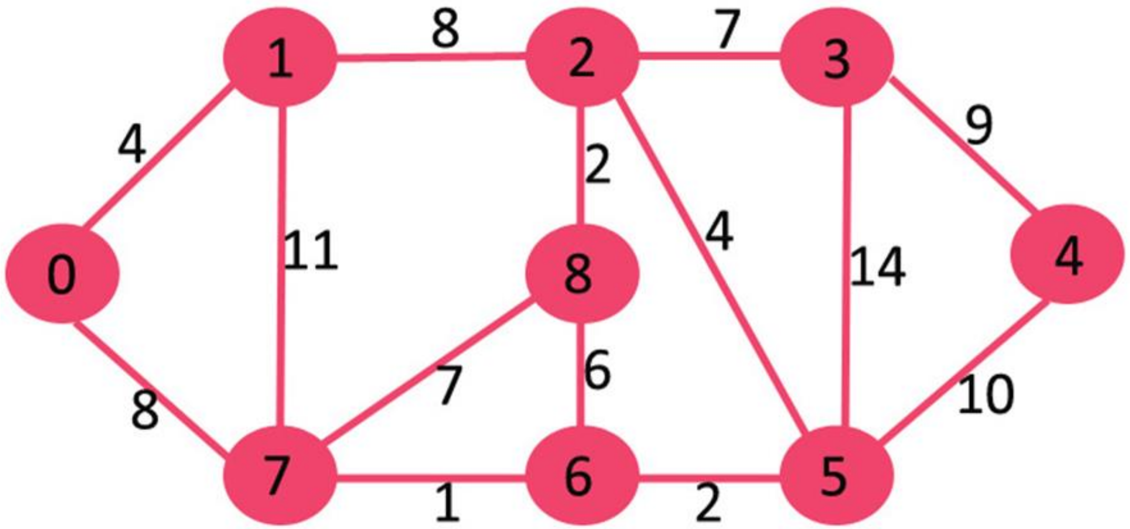
Pick up a vertex v. Let us say so far best distance is d[v].
There will be some neighboring vertices

Consider a neighboring vertex u.
Minimum distance from source to u is known as dist[u]

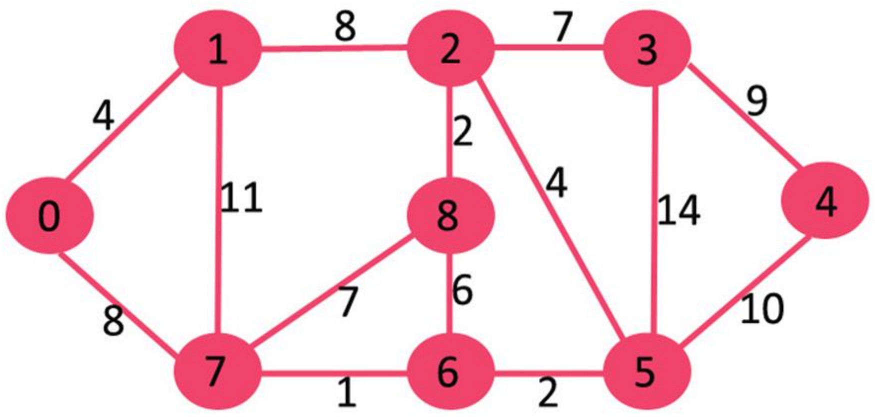
Let distance from u to v be w(u,v).

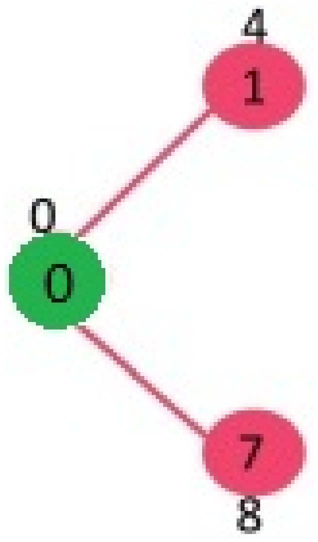
for all $v \in$ neighbors[u]
do if dist[v] > dist[u] + w(u, v)
(if new shortest path found)
then d[v] = d[u] + w(u, v)
P[v] = u
(set new value of shortest path)
return dist

- Source vertex is ‘0’.
- Distance of source from source is zero, and all other distances are assumed to be infinity (unknown)
- SP is [0, ∞, ∞, ∞, ... , ∞]

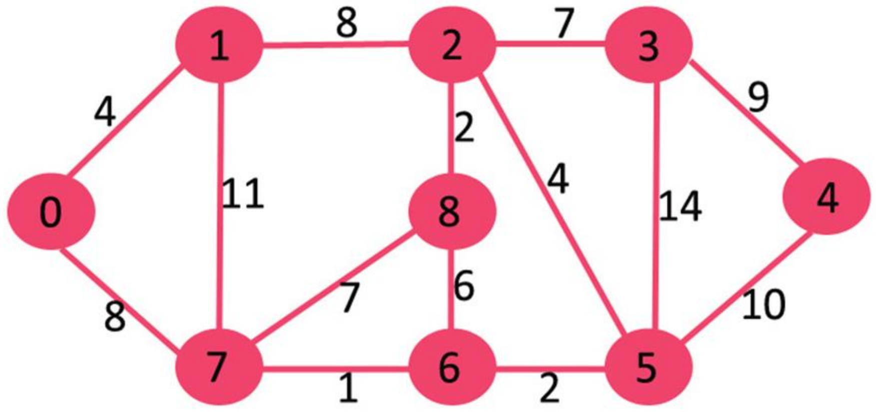


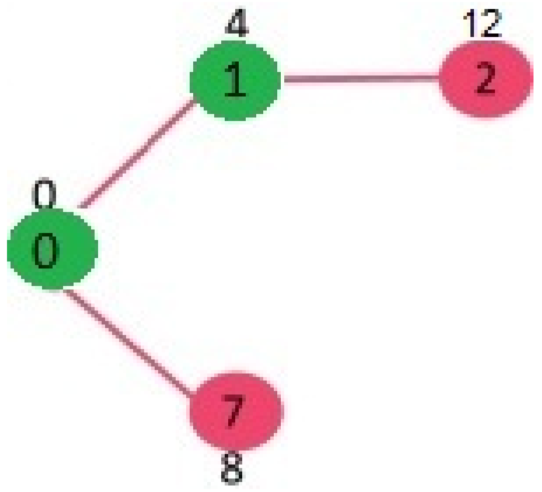
- sptSet shortest path set now becomes {0}.
- Adjacent vertices of 0 are 1 and 7.
- The distance values of 1 and 7 are updated as 4 and 8



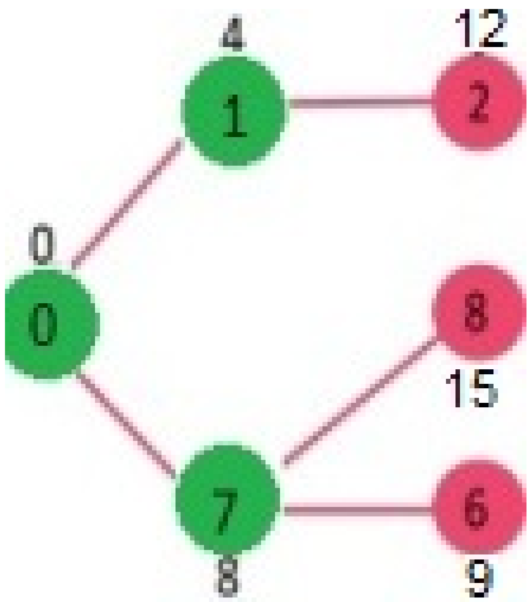


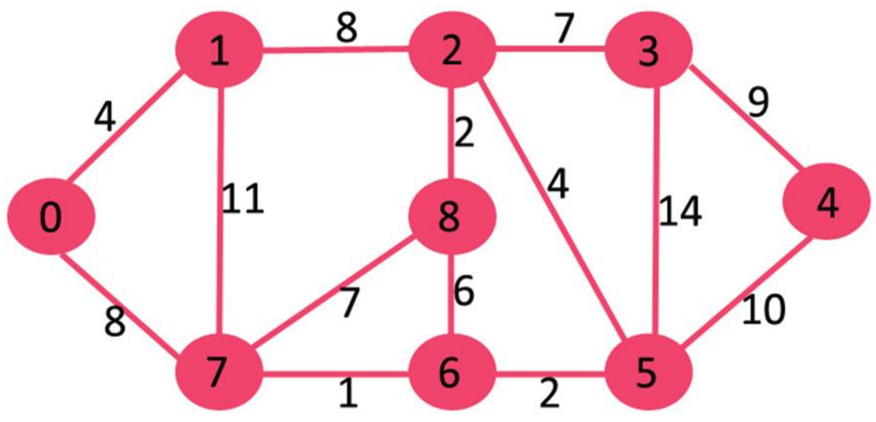
- Pick the vertex with minimum distance value and not already included in SPTree.
- vertex 1 is picked and added to sptSet.
- So sptSet now becomes {0, 1}.
- adjacent vertices of 1 are 7 and 2.
- Distance of vertex 7 stored as **8**.
- Distance of vertex 2 stored as **12**.



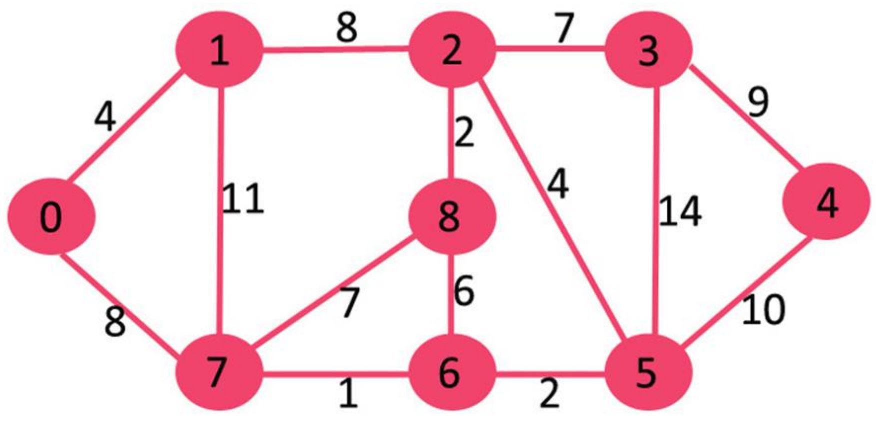


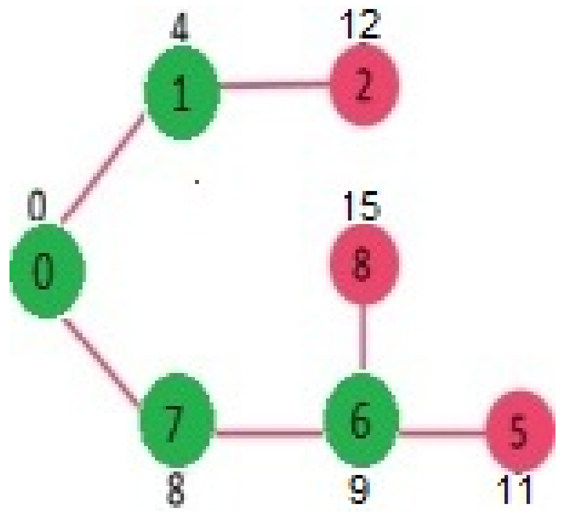
- Pick the vertex with minimum distance value and not already included in SPTtree. Vertex 7 is picked. So sptSet now becomes **{0, 1, 7}.**
- Update the distance values of adjacent vertices of 7.
- The distance value of vertex 6 is 9
- The distance value of vertex 8 is 15
-



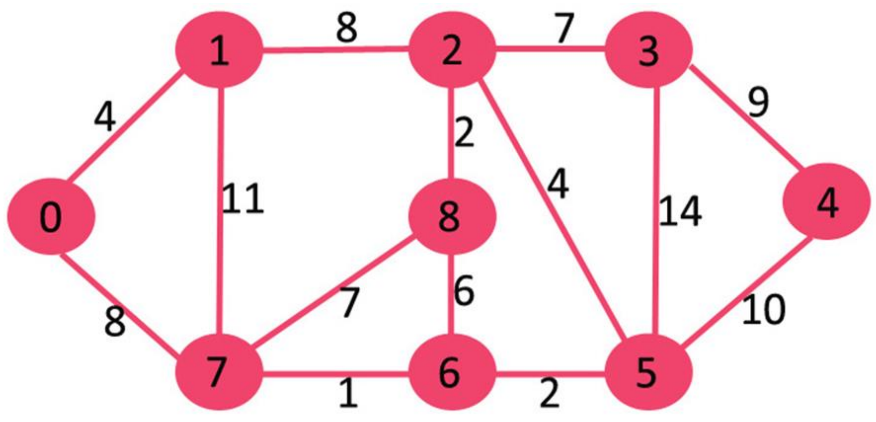


- Pick the vertex with minimum distance value and not already included in SPTtree. Vertex 6 is picked. So sptSet now becomes **{0, 1, 7,6}.**
- Update the distance values of adjacent vertices of 6.
- The distance value of vertex 5 is 11
- The distance value of vertex 8 is 15
-

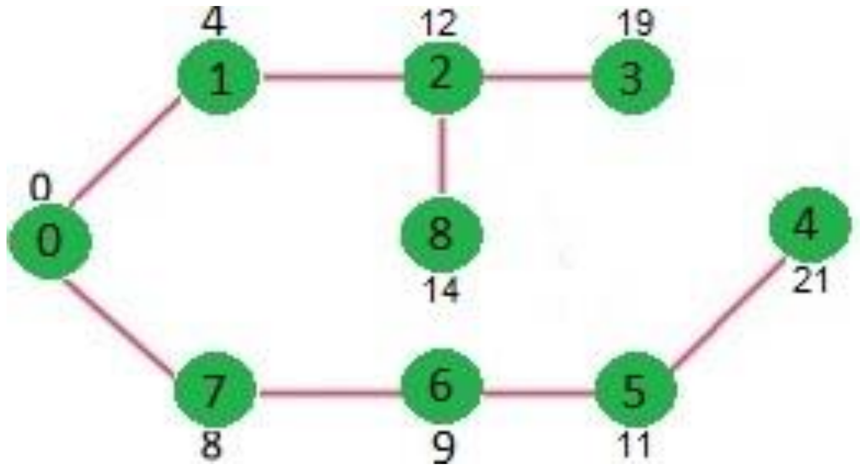




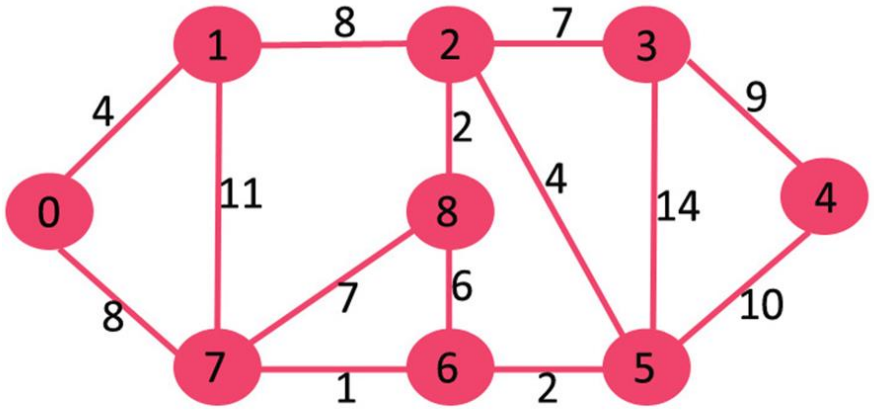
- Pick the vertex with minimum distance value and not already included in SPTtree. Between 5, 8 and 11 ,Vertex 5 is picked. So sptSet now becomes **{0, 1, 7, 6, 5}.**
- Update the distance values of adjacent vertices of 5.
- *The distance value of vertex 2 using vertex 5 is 11+4*
- *but we already have value to 2 as 12,*
- *so we do not update that.*
- The distance value of vertex 3 is 25
- The distance value of vertex 4 is 21
- So we pick up vertex 2 in SPTree



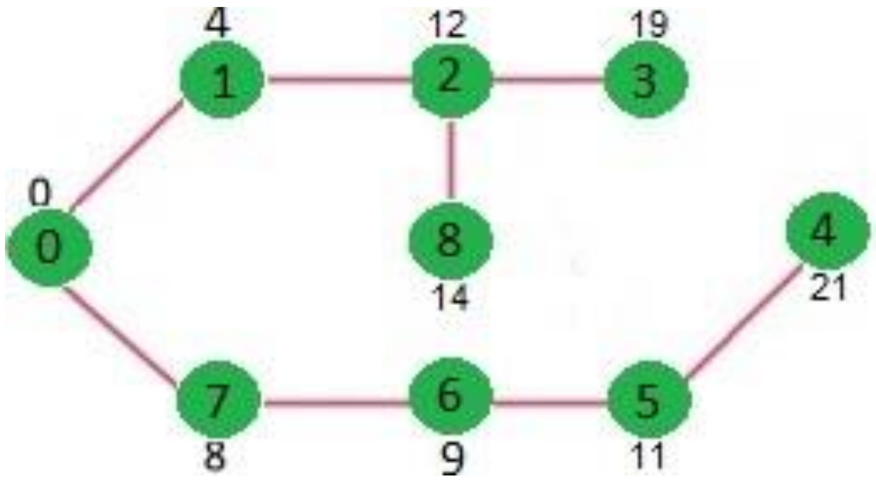
-



- Pick the vertex with minimum distance value and not already included in SPTtree. Between 2, 3 and 4 ,Vertex 2 is picked. So sptSet now becomes **{0, 1, 7, 6, 5, 2}.**
- Update the distance values of adjacent vertices of 2.
- The distance value of vertex 5 using 2 is 16
- But better value of 5 is old value 11
- The distance value of vertex 3 is 19
- The distance value of vertex 8 is 14
- So we pick up vertex 3 in SPTree



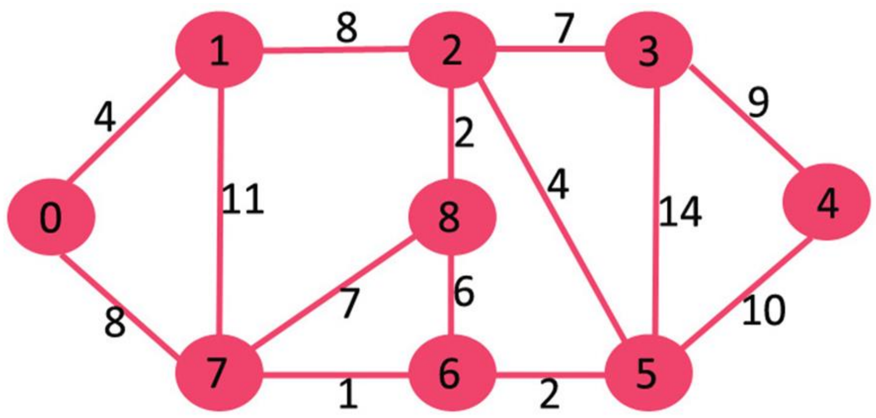
-



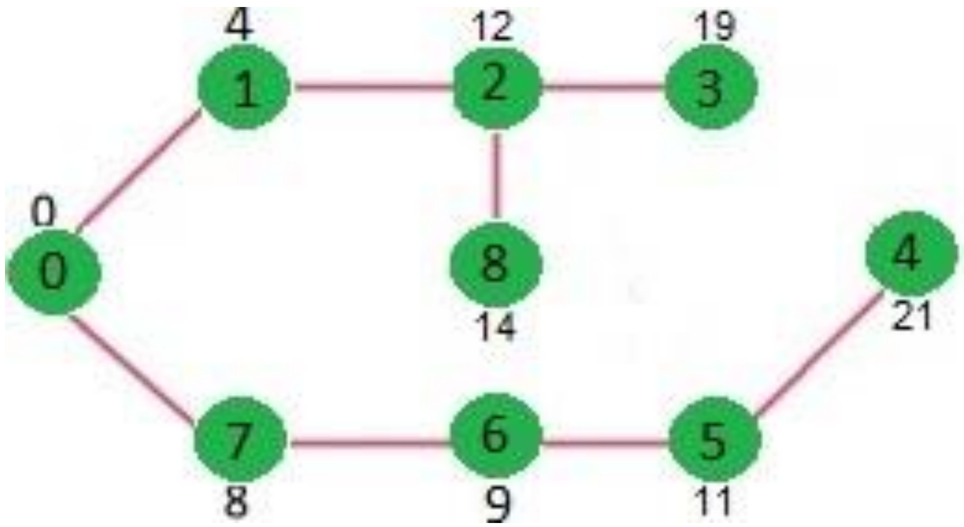
- Pick up remaining vertices, it is noticed that no more distances can be updated.

- The green graph shows
- the shortest paths
- and shortest distances to all vertices from vertex 0.

- **Time Complexity:** O(V^2)



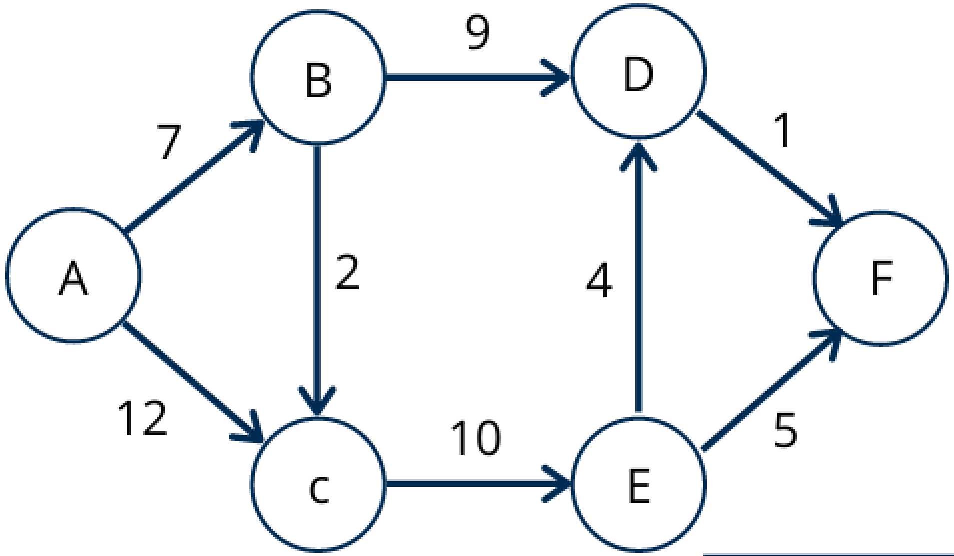
-



- The same example is illustrated very nicely in following link, the names of vertices are slightly different.

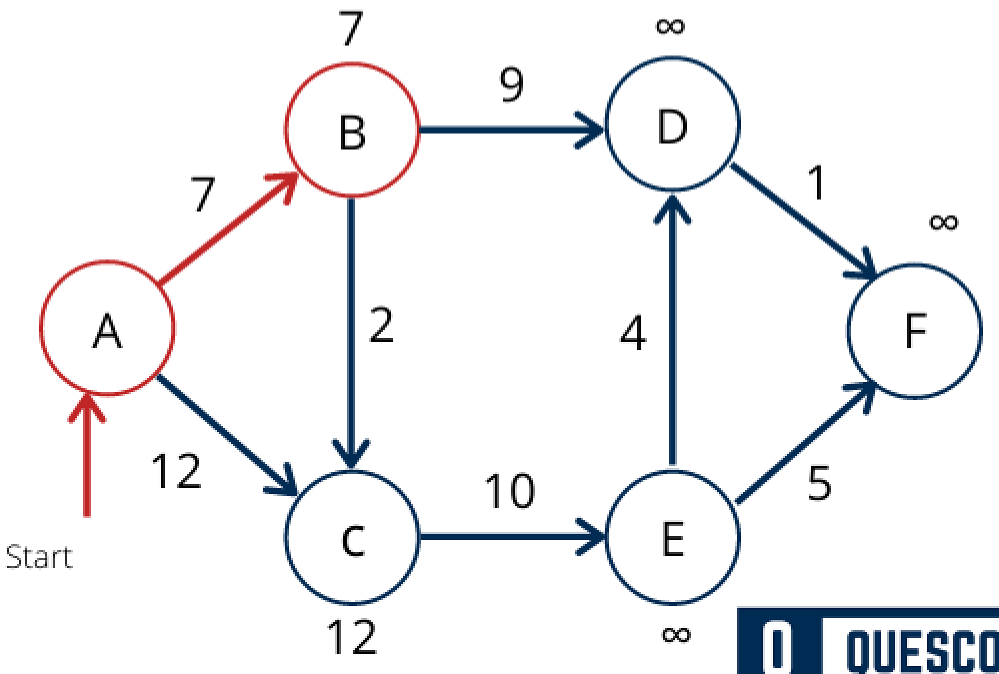
- https://www.javatpoint.com/dijkstras-algorithm
-

Dijkastra’s method on a directed graph



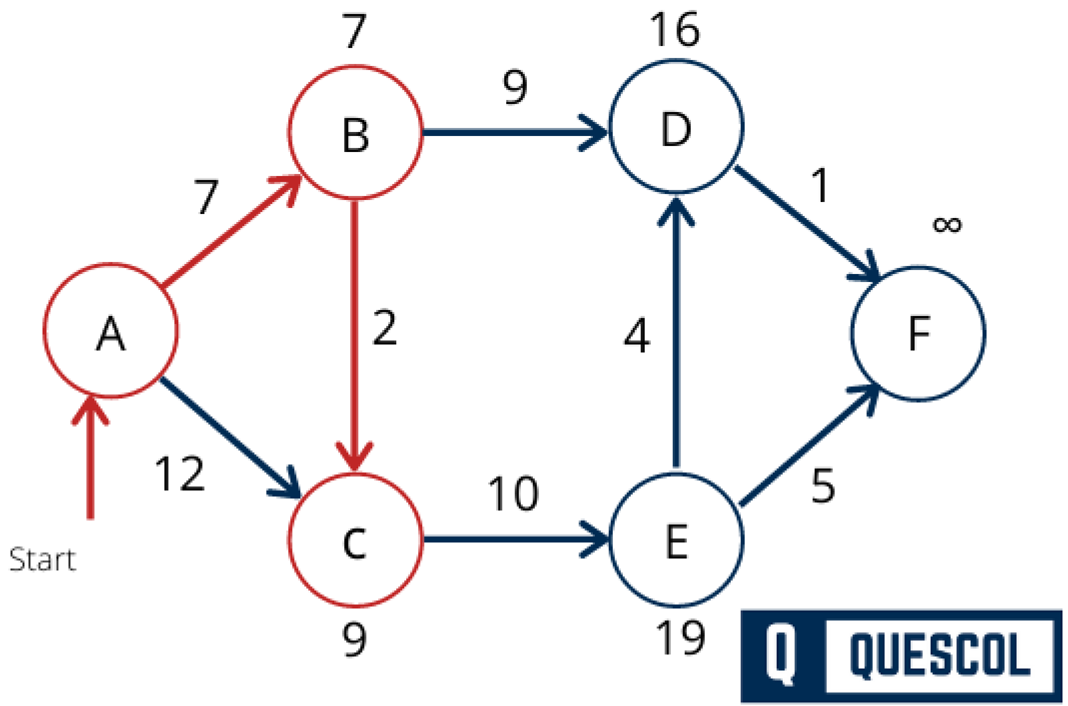
Let A be source vertex.
We want to find shortest paths to all vertices starting from A.
Set all distances to infinity.

	K	d_v	p_v
A	T	0	
B	F	∞	
C	F	∞	
D	F	∞	
E	F	∞	
F	F	∞	
			–



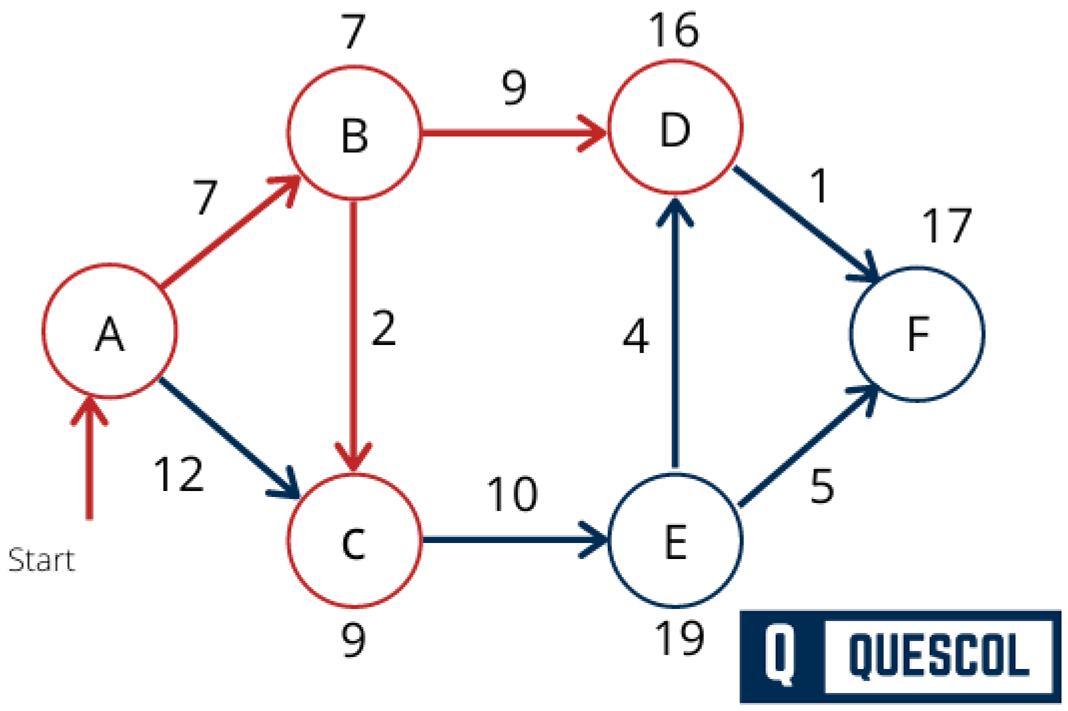
	K	d_v	p_v
A	T	0	
B	T	7	A
C	T	12	
D	F	∞	
E	F	∞	
F	F	∞	
			–

Path AB is 7, AC is12…



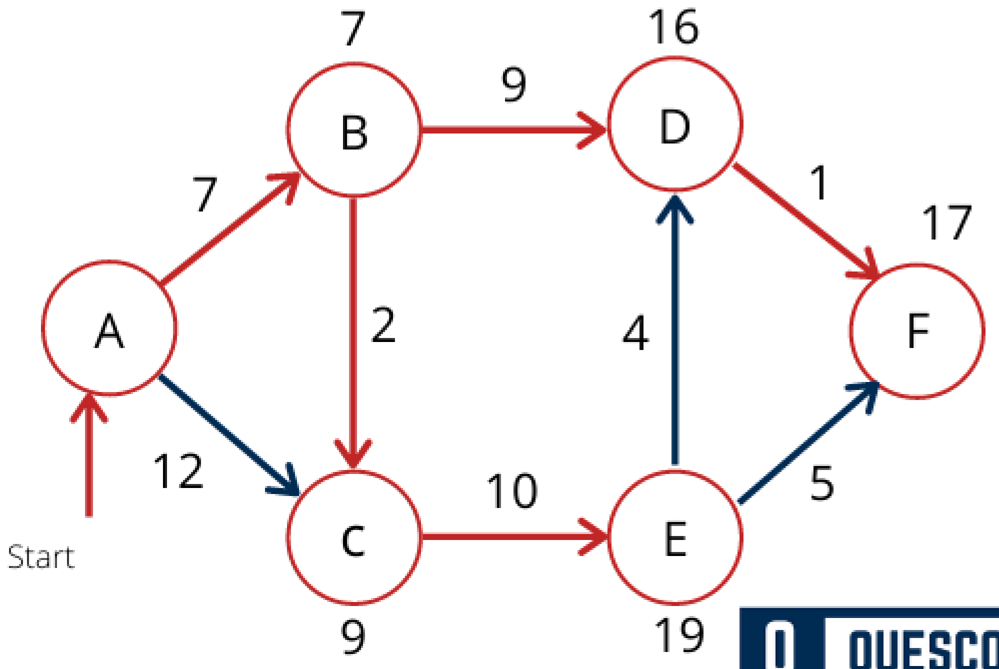
	K	d_v	p_v
A	T	**0**	
B	T	7	A
C	T	9	B
D	F	∞	
E	F	∞	
F	F	∞	
			–

Path AC is 12, path ABC is 9, so we select ABC…



	K	d_v	p_v
A	T	0	
B	T	7	A
C	T	9	B
D	T	16	B
E	F	∞	
F	F	∞	
			–

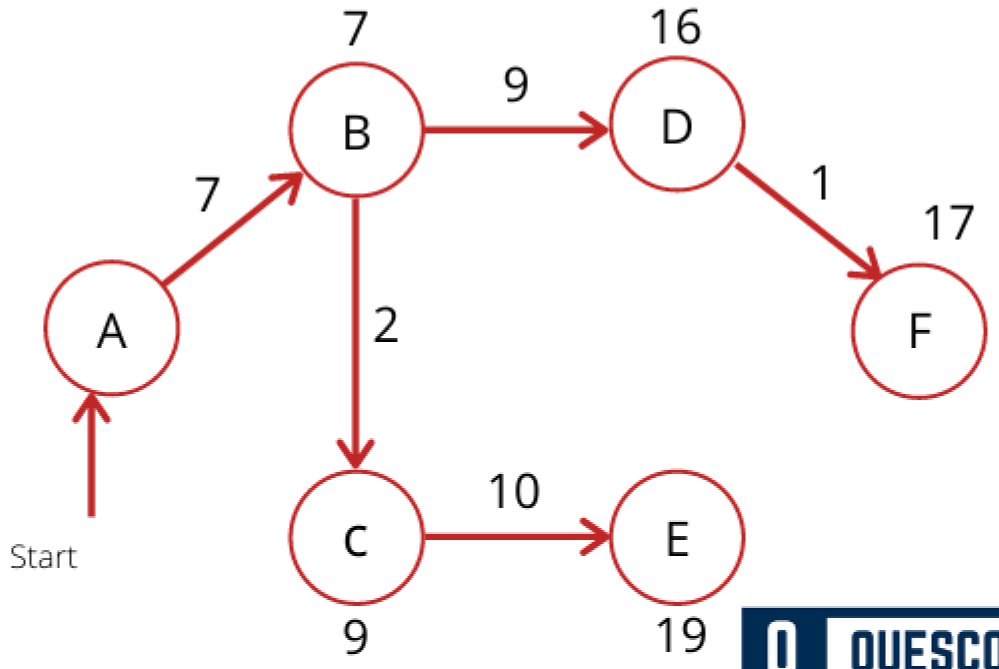
ABCE is 19, ABD is 16, so now we select BD…



	K	d_v	p_v
A	T	**0**	
B	T	7	A
C	T	9	B
D	T	16	B
E	T	19	C
F	T	17	D
			–

Having reached D, only one path to F, also only single path to E…

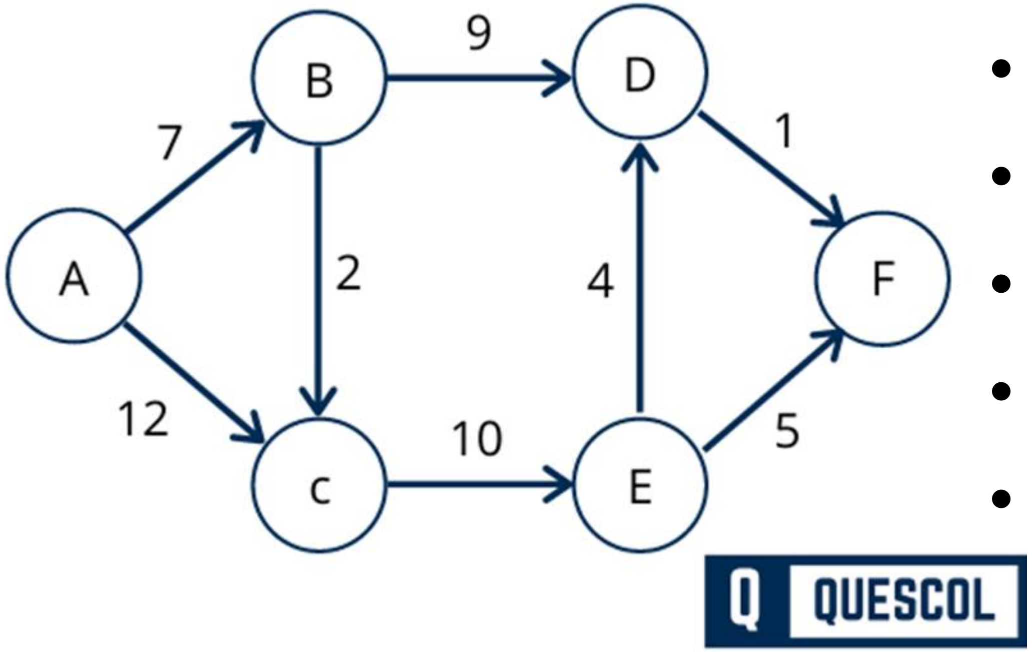
Shortest Path Tree



	K	d_v	p_v
A	T	0	
B	T	7	A
C	T	9	B
D	T	16	B
E	T	19	C
F	T	17	D
			–

All shortest paths are shown here
Paths can be traced from adjoining table…

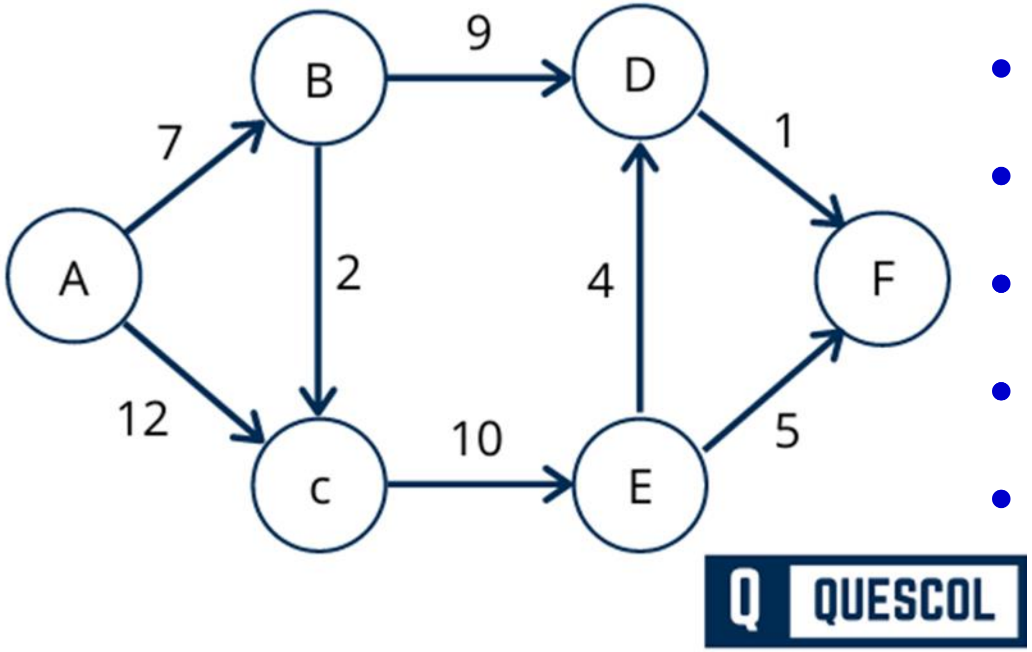
Single table solution



- A 0
- B ∞
- C ∞
- D ∞
- E ∞
- F ∞

- START WITH A

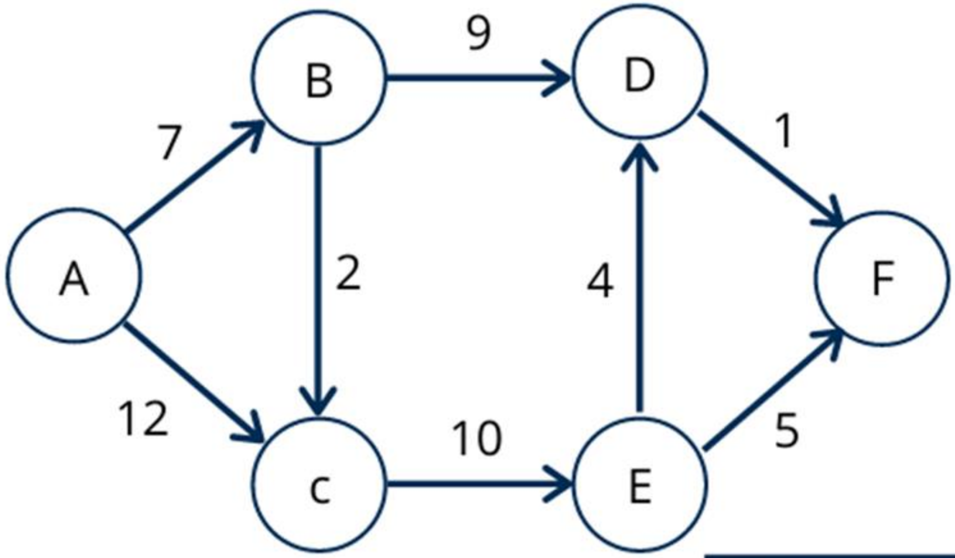
Single table solution



- A 0
- B ∞ 7
- C ∞ 12
- D ∞
- E ∞
- F ∞

- Select B

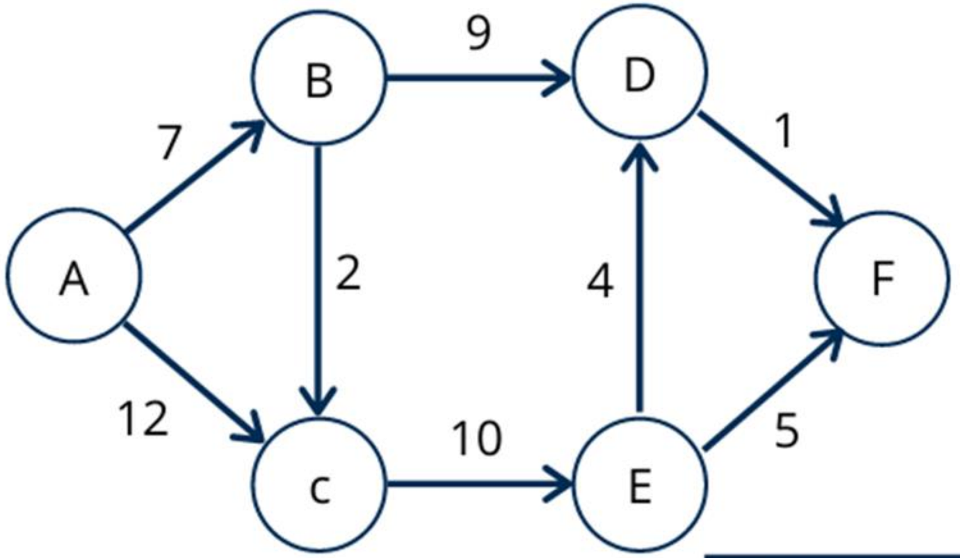
Single table solution



- A 0 A B
- B ∞ 7
- C ∞ 12 9
- D ∞ 16
- E ∞
- F ∞

- Select C

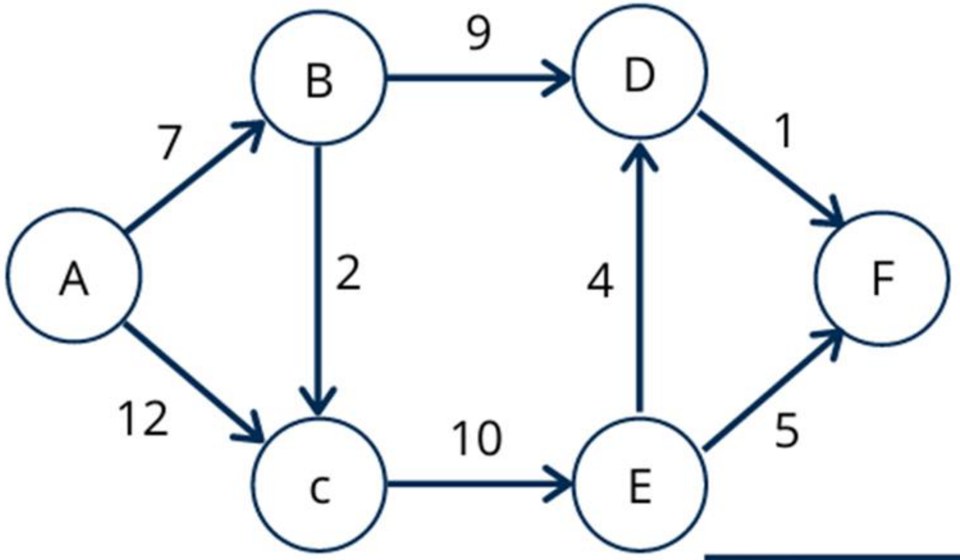
Single table solution



- A 0 A B C
- B ∞ 7
- C ∞ 12 9
- D ∞ 16
- E ∞ 19
- F ∞

- Select D

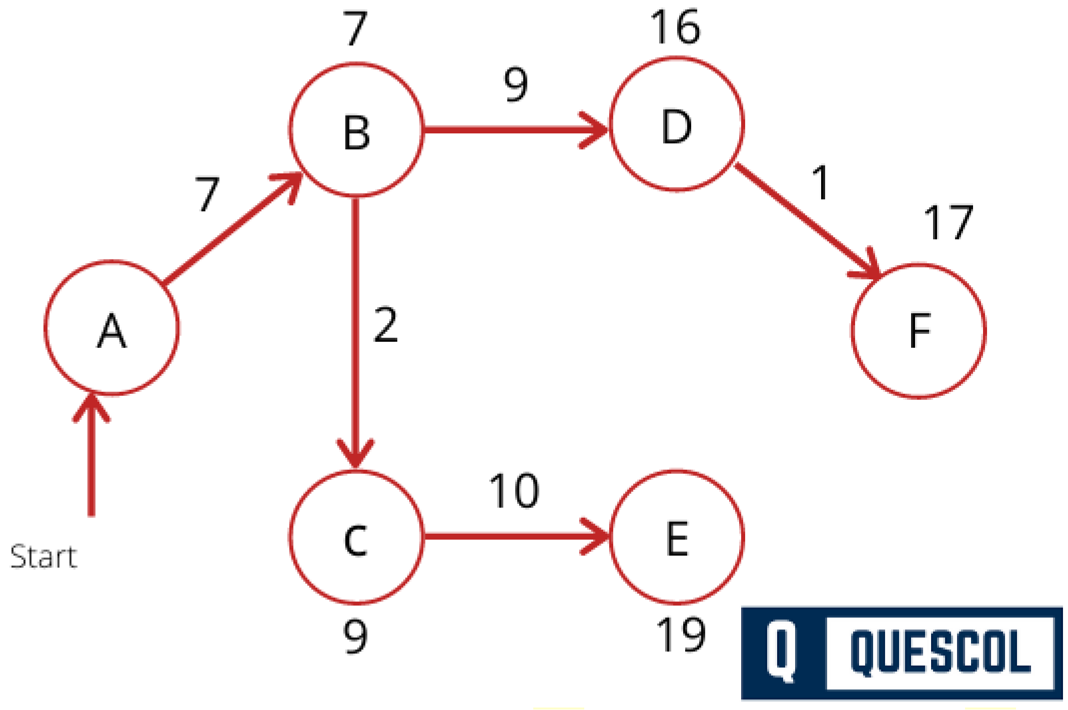
Single table solution



- A 0 A B C D
- B ∞ 7
- C ∞ 12 9
- D ∞ 16
- E ∞ 19
- F ∞ 17

- Select D

Shortest Path Tree

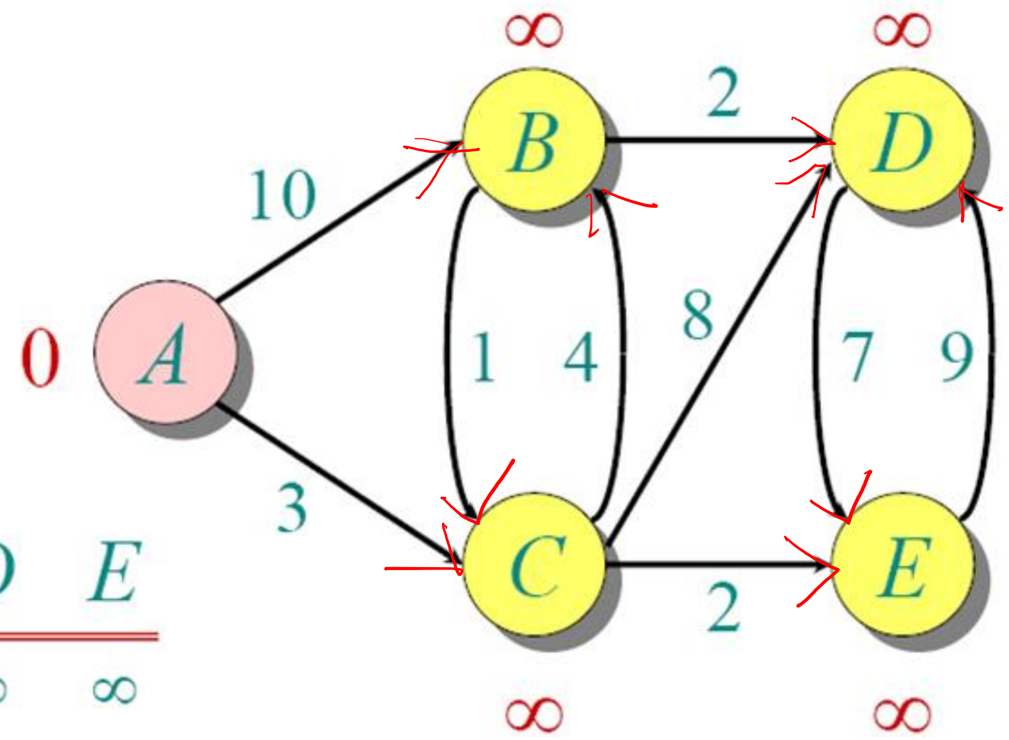


	K	d_v	p_v
A	T	0	
B	T	7	A
C	T	9	B
D	T	16	B
E	T	19	C
F	T	17	D
			–

All shortest paths are shown here
Paths can be traced from adjoining table…

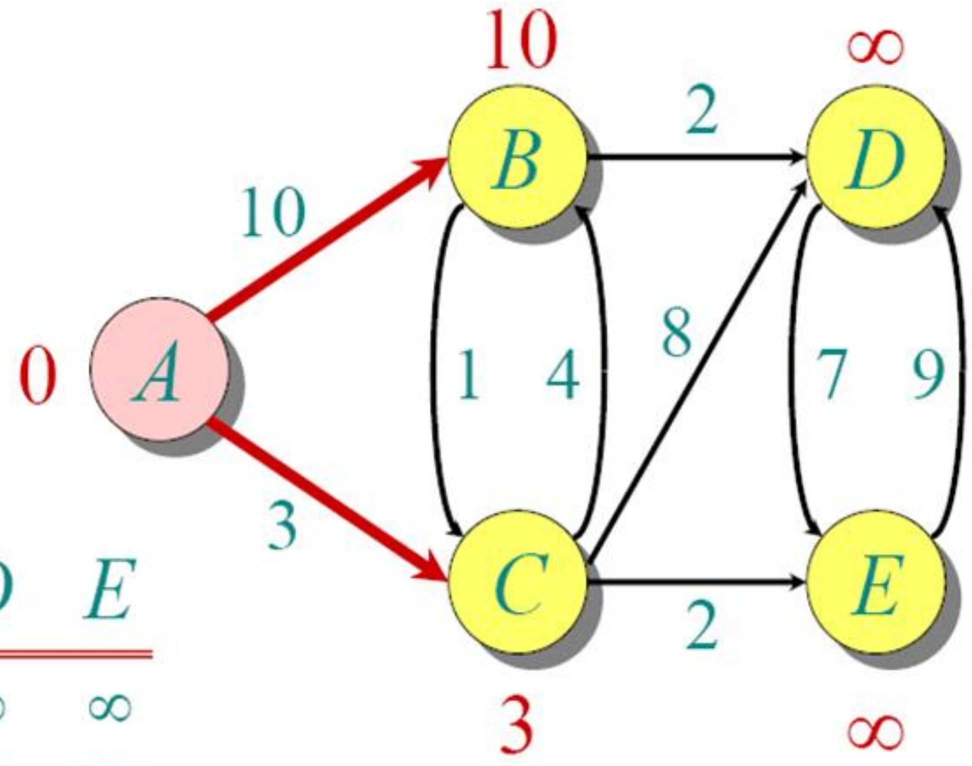
Example 3

Node A specified



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

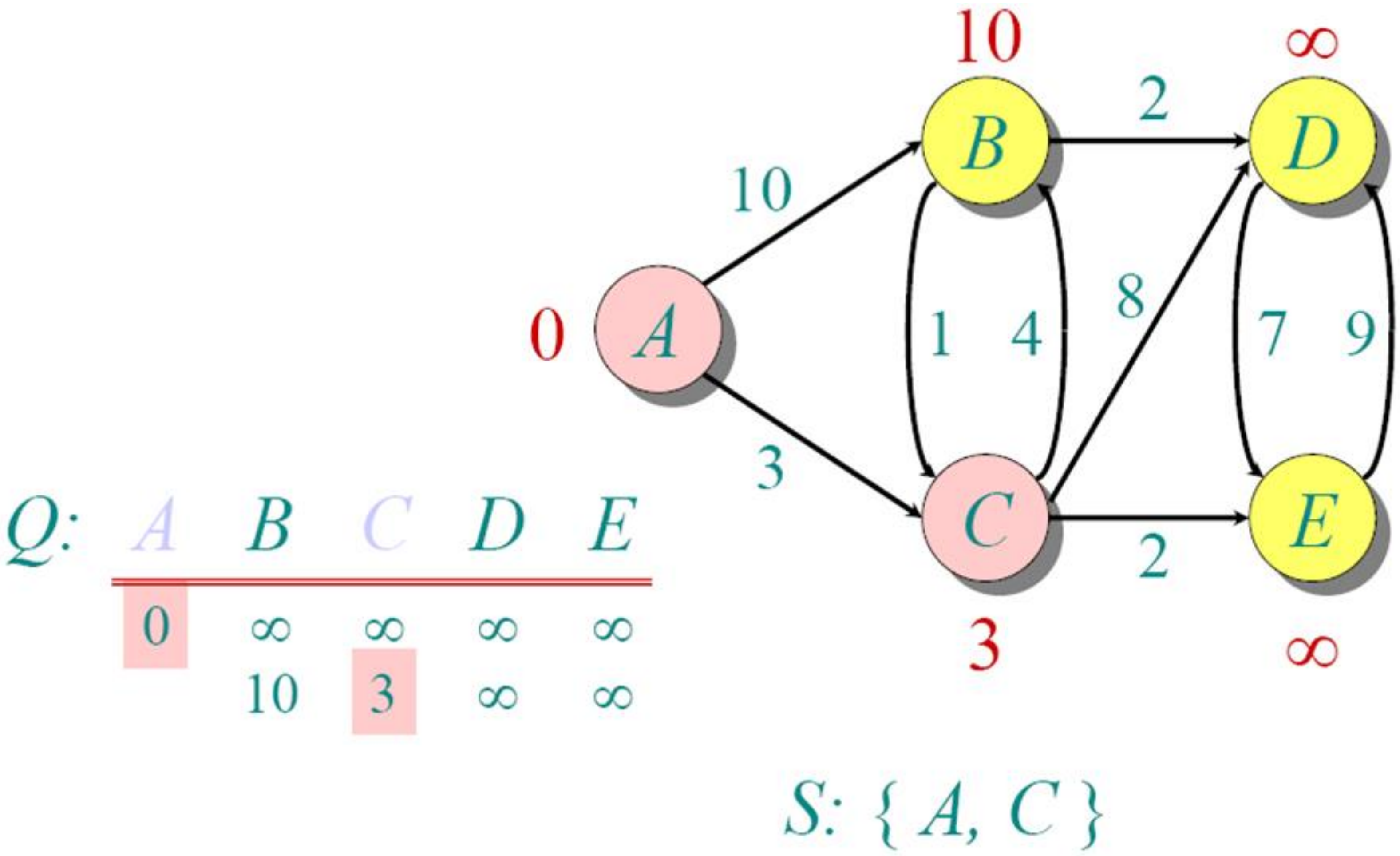
Direct distance B is 10, C is 3



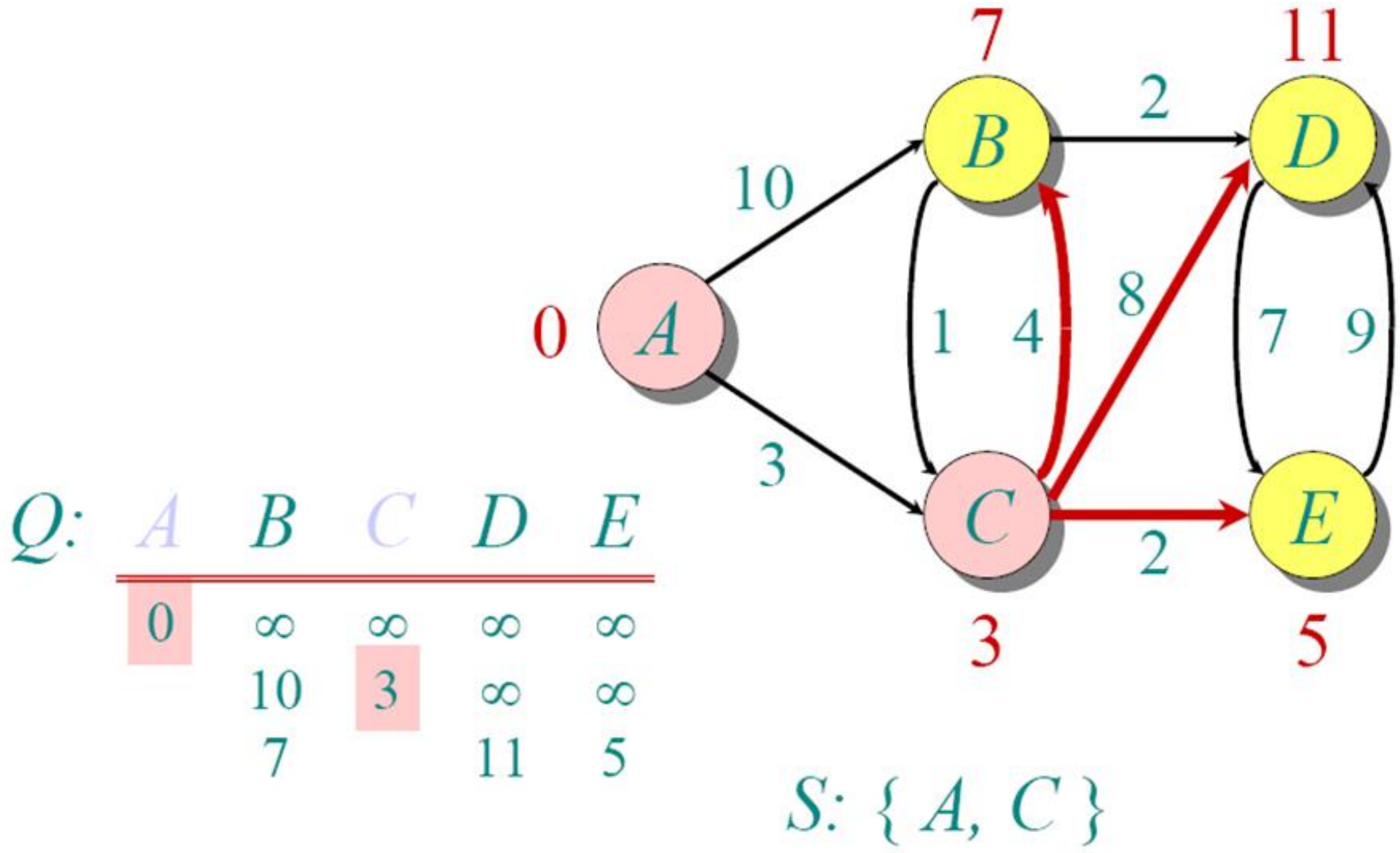
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

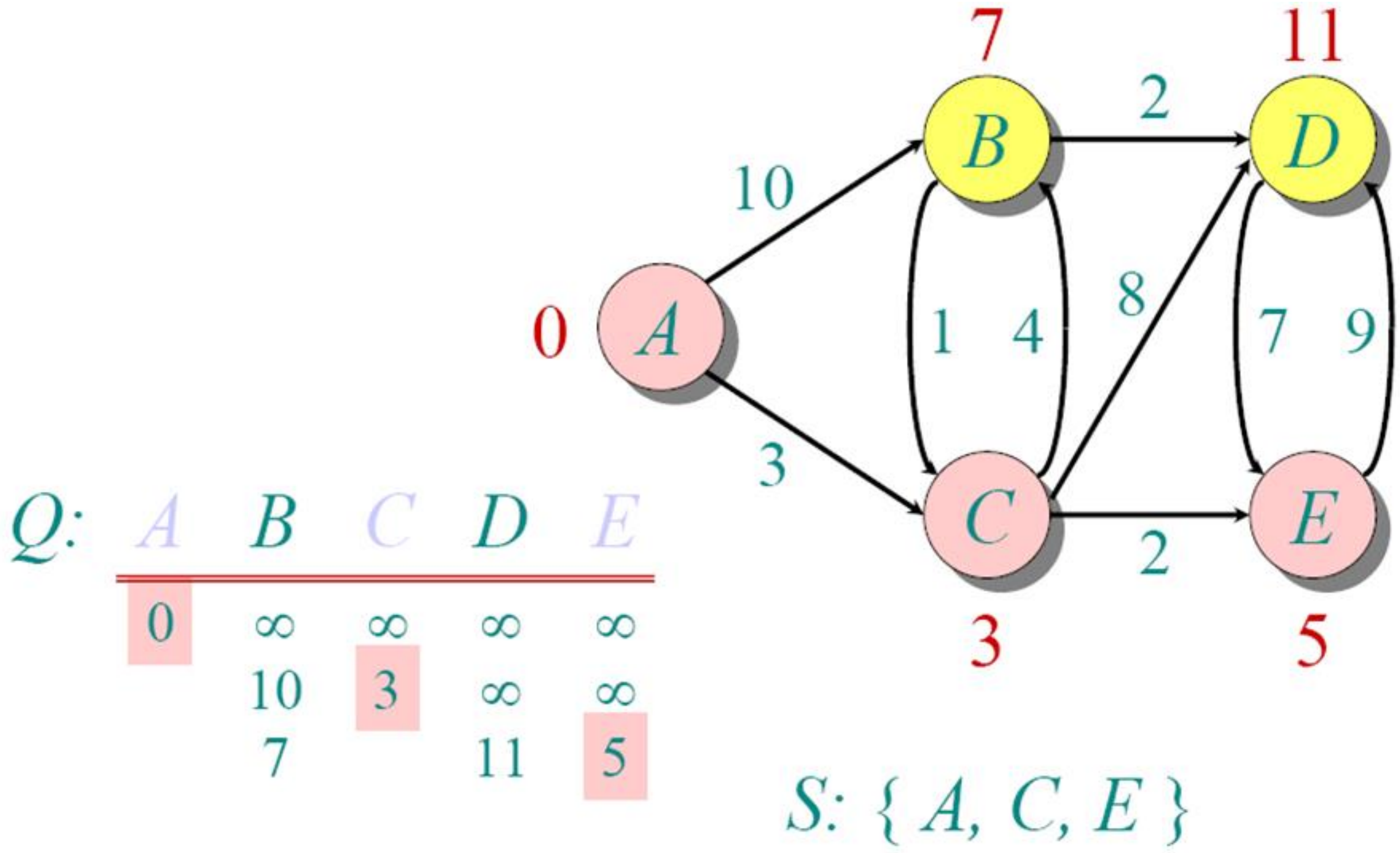
C chosen



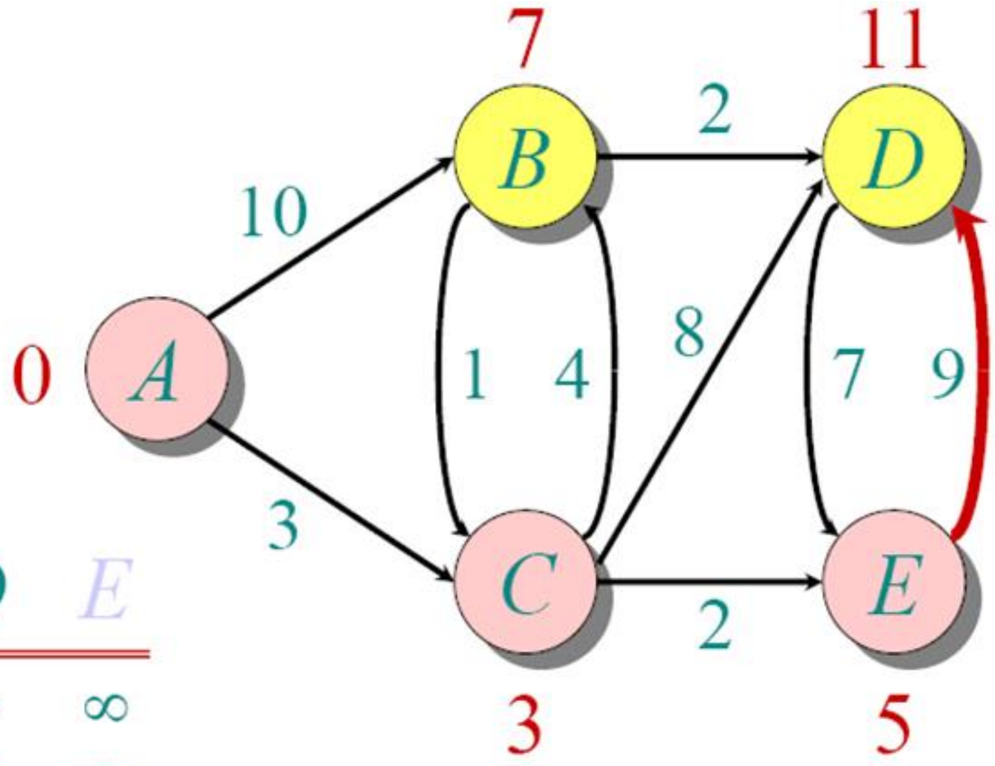
Through C, D is 3+8, and E is 3+2



Choose E as it is smallest dist.



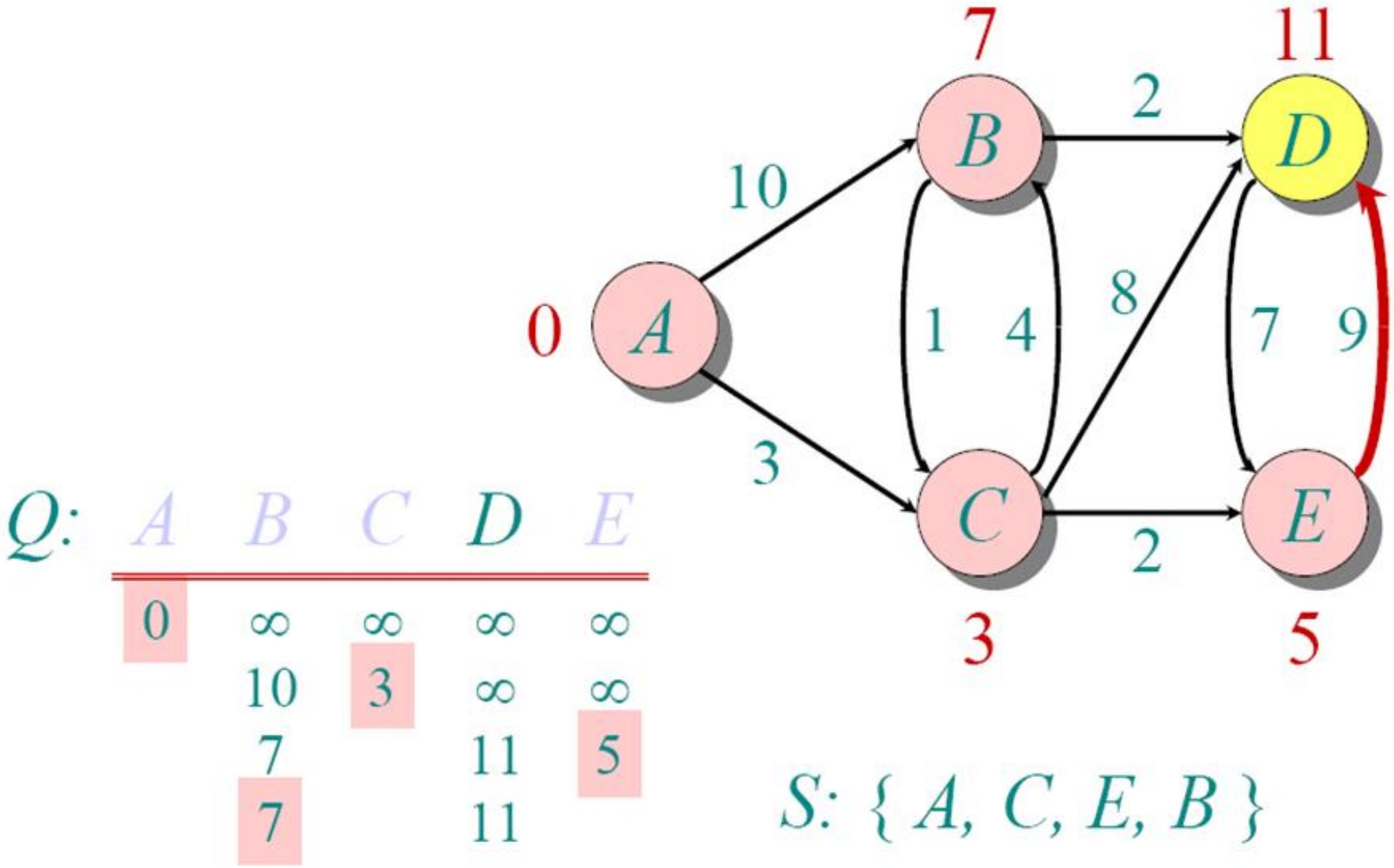
No new updating



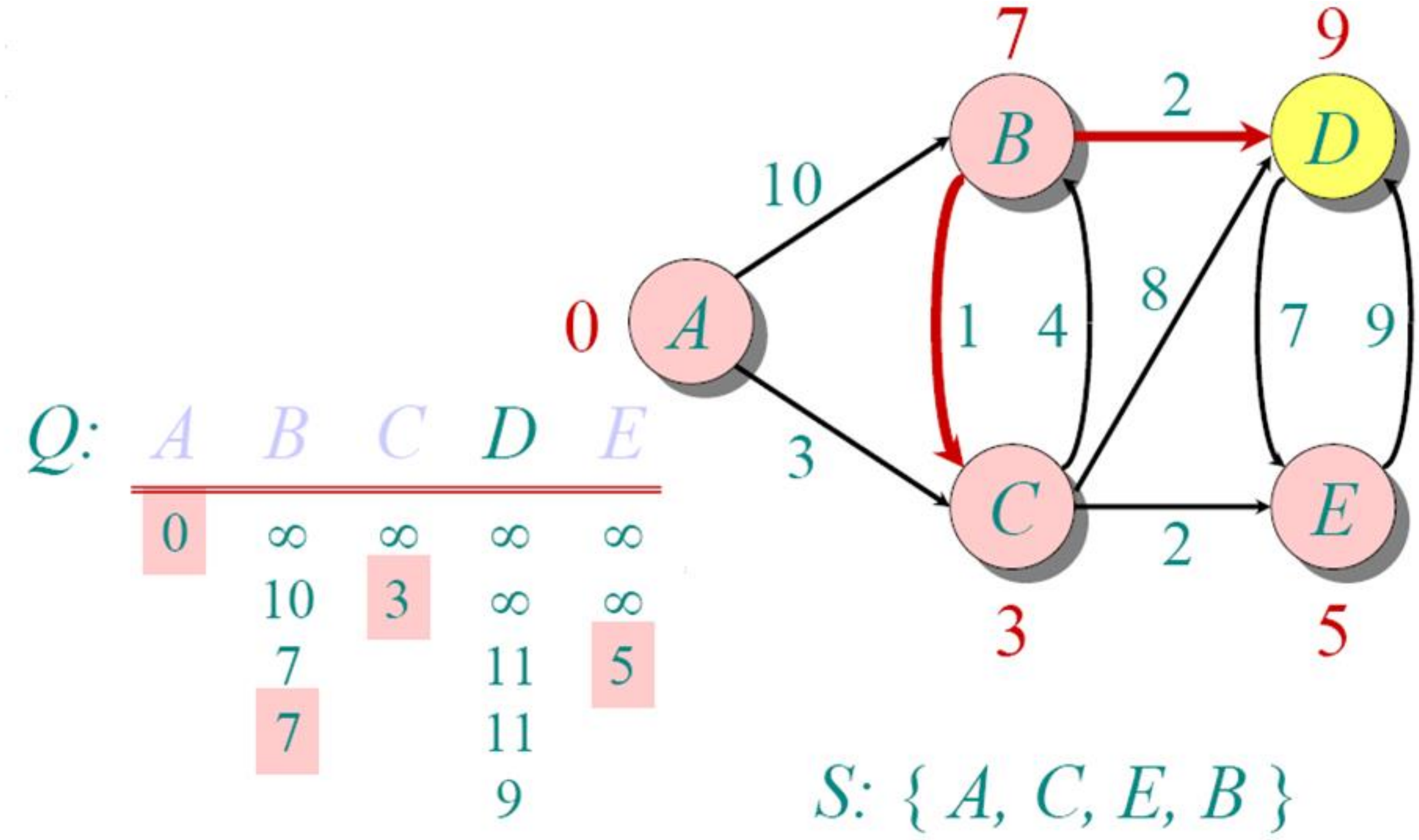
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }

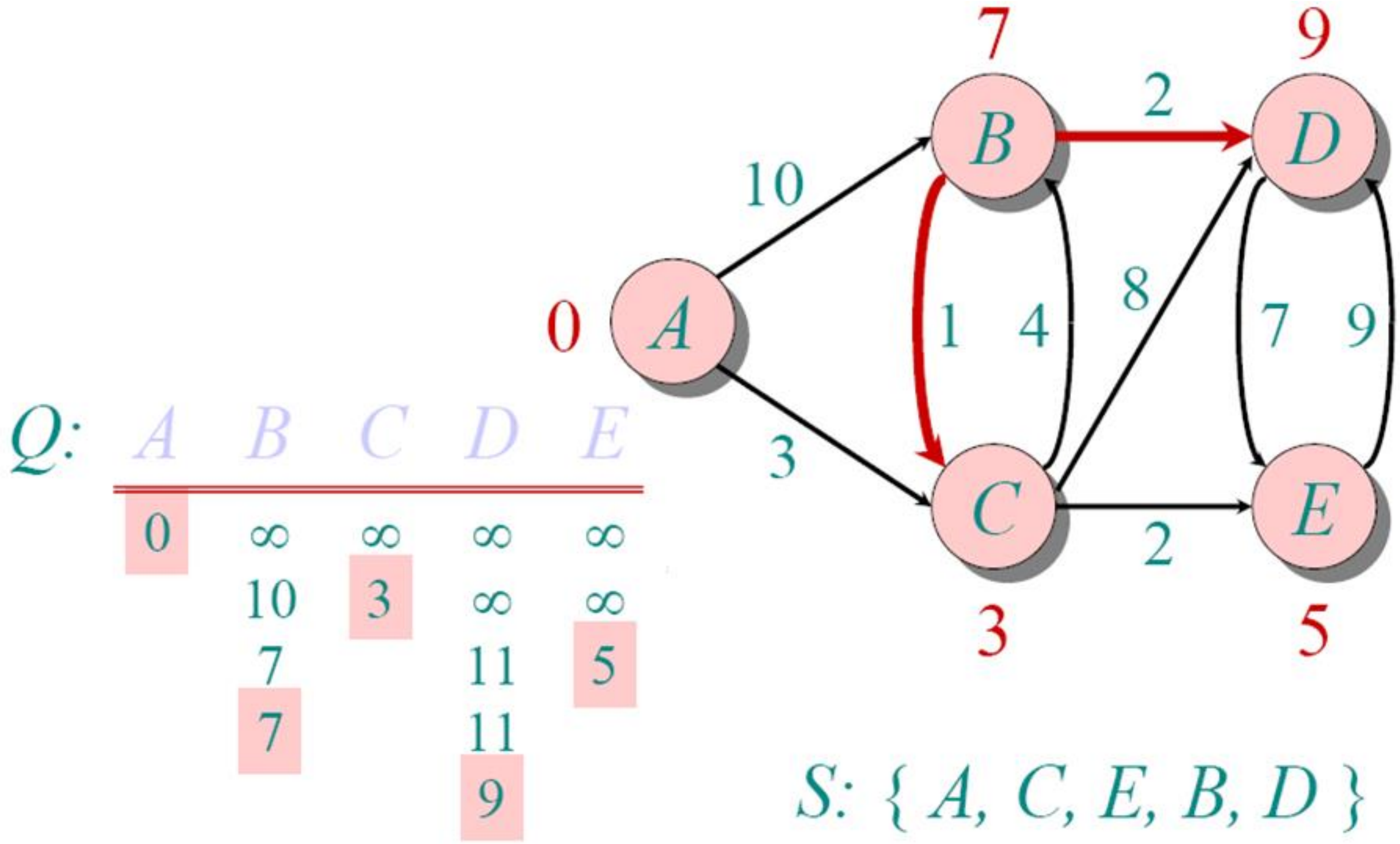
between B and D, B chosen



By choosing B, distance to D is 9



Shortest distance to all nodes computed



Single Table Solution

- A C E B D
- A 0 0 0 0 0
- B ∞ 10 7 7 7 7
- C ∞ 3 3 3 3 3
- D ∞ ∞ 11 11 9 9
- E ∞ ∞ 5 5 5 5