CS 2124: DATA STRUCTURES
Spring 2024

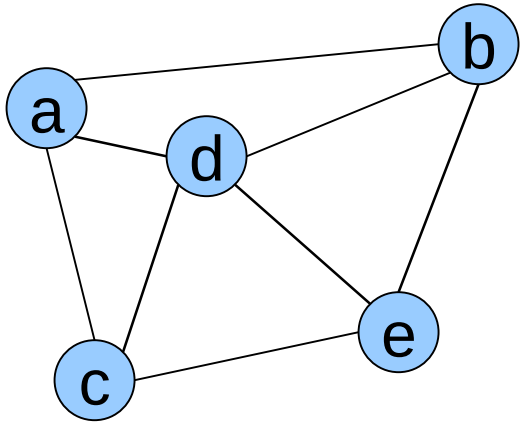
Lecture 12.2

Topics: Breadth First Search (BFS), Depth First Search (DFS), and Dijkstra's Algorithm

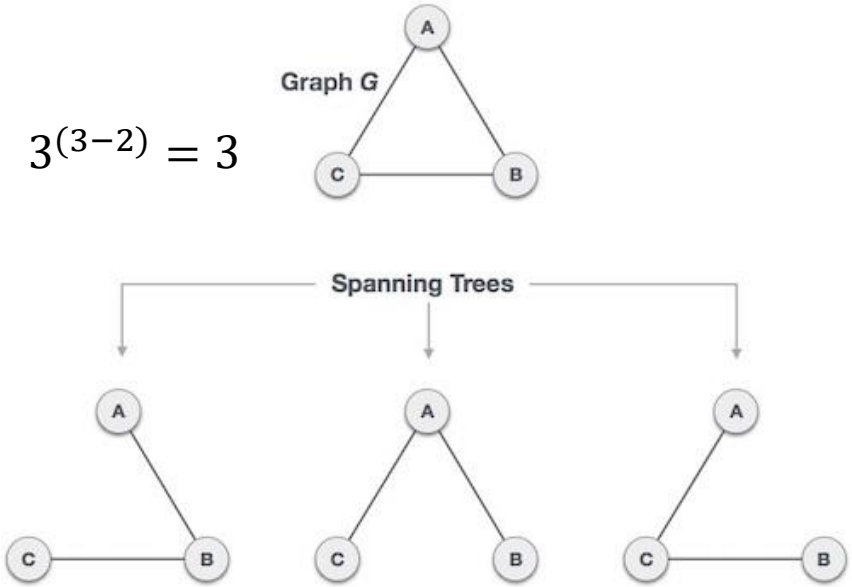
Spanning Trees

- If a graph is **a complete graph** with n vertices, then total number of spanning trees is $n^{(n-2)}$ where n is the number of nodes in the graph.

$n^{(n-2)}\ (n = 5) \Rightarrow 5^{(5-2)} \Rightarrow 5^{(3)}$

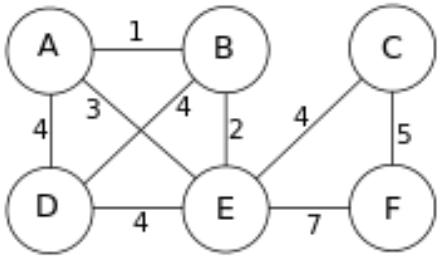


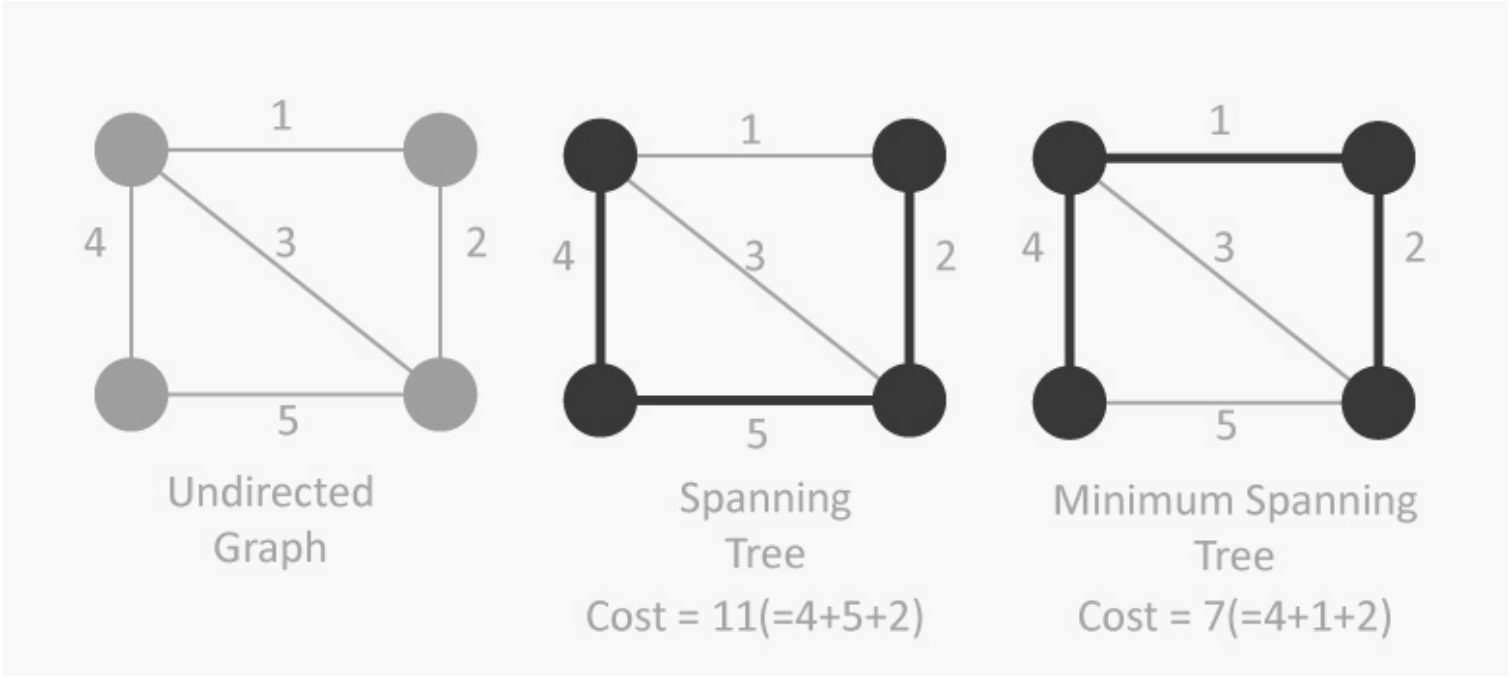
$3^{(3-2)} = 3$

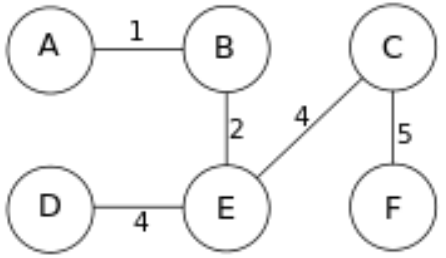


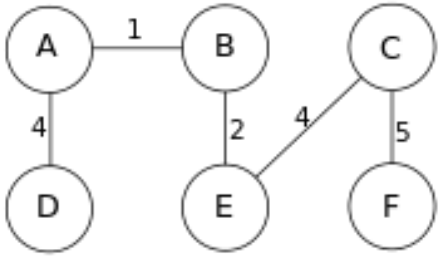
Minimum Spanning Tree (MST)

- A minimum spanning tree (MST) is a subset of the edges of a connected, edge-weighted graph that connects all the vertices together without any cycles and with the minimum possible total edge weight.
- It is a way of finding the **most economical way to connect a set of vertices**.



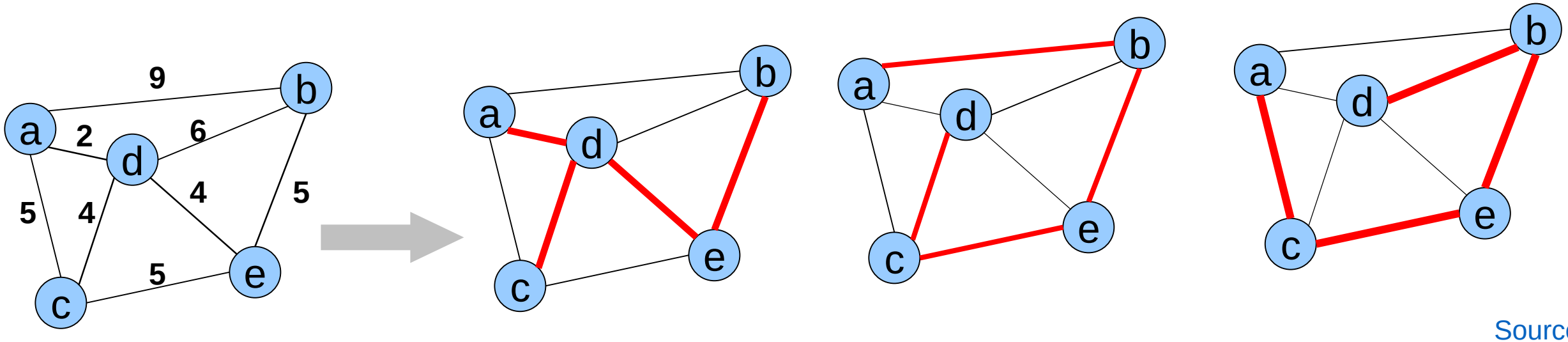






Minimum Spanning Tree (MST)

- A **minimum spanning tree** is a subgraph of an undirected weighted graph ***G***, such that
- It is a tree (i.e., It is acyclic)
- It covers all the vertices ***V***
  - Contains |***V***| - ***1*** edges
- The total cost associated with tree edges is the minimum among all possible spanning trees
- Not necessarily unique

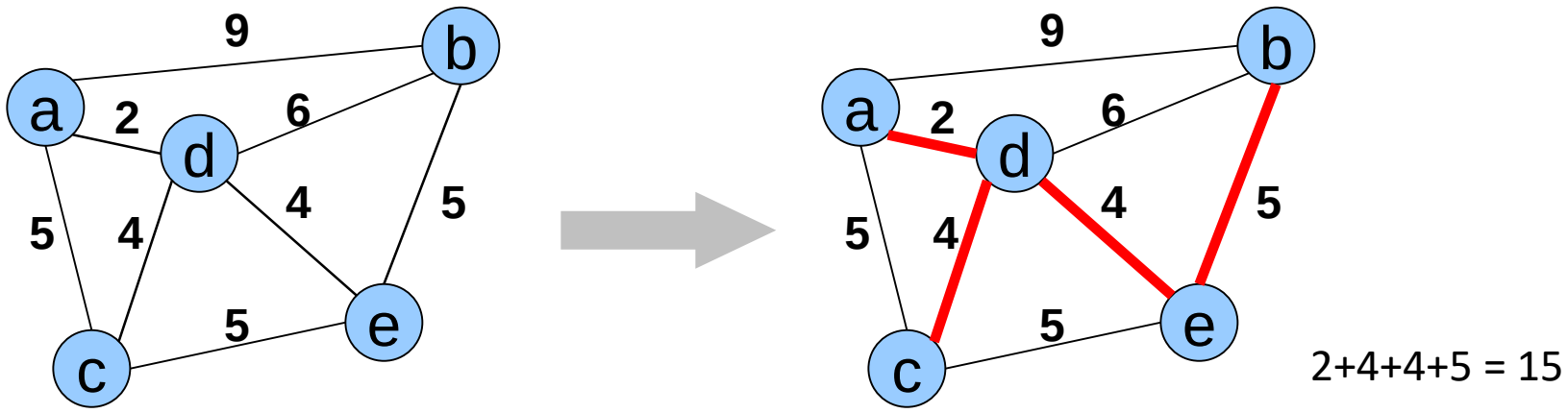


Source

Minimum Spanning Tree (MST)

A **minimum spanning tree** is a subgraph of an undirected weighted graph ***G***, such that

- it is a tree (i.e., it is acyclic)
- it covers all the vertices V
  - contains $|V| - 1$ edges
- the total cost associated with tree edges is the minimum among all possible spanning trees
- not necessarily unique

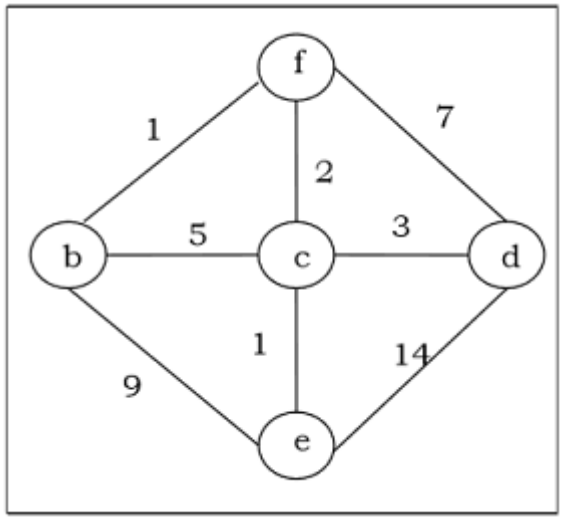


2+4+4+5 = 15

Pair of E	Weight
a,b / b,a	9
b,d / d,b	6
b,e / e,b	5
e,d / d,e	4
c,d / d,c	4
c,a / a,c	5
a,d / d,a	2

Source

Spanning Trees



- MST (Minimum Spanning Tree) ?
- It is a way of finding the **most economical way to connect a set of vertices**

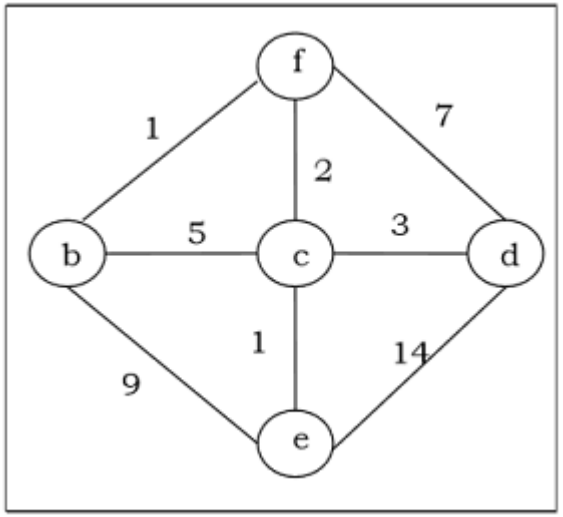
	B-0	C-1	D-2	E-3	F-4
B-0	0	5	0	9	1
C-1	5	0	3	1	2
D-2	0	3	0	14	7
E-3	9	1	14	0	0
F-4	1	2	7	0	0

Kruskal Algorithm

- Kruskal algorithm is used to generate a minimum spanning tree for a given graph.
- Kruskal's algorithm sorts all the edges in increasing order of their edge weights and keeps adding nodes to the tree only if the chosen edge does not form any cycle.
- Also, it picks the edge with a minimum cost at first and the edge with a maximum cost at last.
- Hence, you can say that the Kruskal algorithm makes a locally optimal choice, intending to find the global optimal solution.
- That is why it is called a Greedy Algorithm.

Kruskal Algorithm

- Kruskal Algorithm:
  - Step 1: Sort all edges in increasing order of their edge weights.
  - Step 2: Pick the smallest edge.
  - Step 3: Check if the new edge creates a cycle or loop in a spanning tree.
  - Step 4: If it doesn't form the cycle, then include that edge in MST. Otherwise, discard it.
  - Step 5: Repeat from step 2 until it includes |V| - 1 edges in MST.

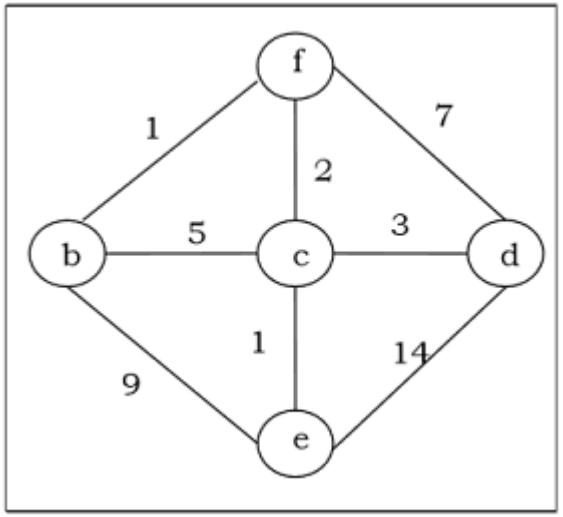


Source Vertex	Destination Vertex	Weight
b	f	1
b	c	5
b	e	9
c	f	2
c	e	1
c	d	3
e	d	14
f	d	7

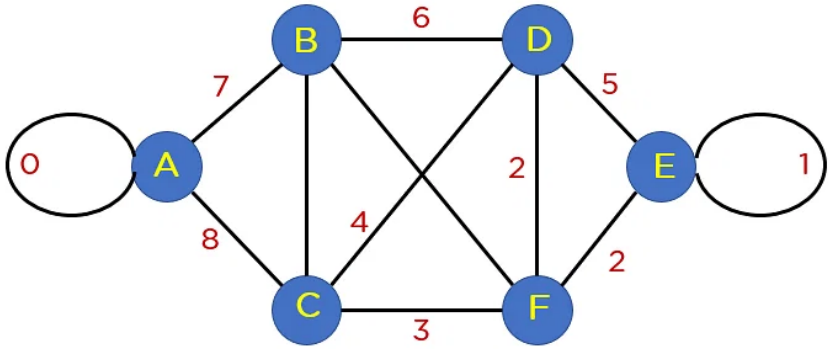
Source Vertex	Destination Vertex	Weight
b	f	1
c	e	1
c	f	2
c	d	3
b	c	5
f	d	7
b	e	9
e	d	14

Kruskal Algorithm

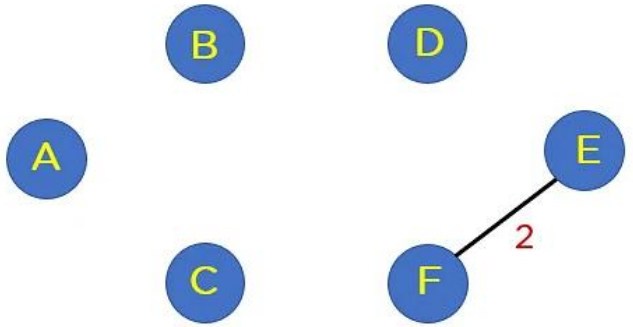
- Kruskal Algorithm:
  - Step 1: Sort all edges in increasing order of their edge weights.
  - Step 2: Pick the smallest edge.
  - Step 3: Check if the new edge creates a cycle or loop in a spanning tree.
  - Step 4: If it doesn't form the cycle, then include that edge in MST. Otherwise, discard it.
  - Step 5: Repeat from step 2 until it includes |V| - 1 edges in MST.

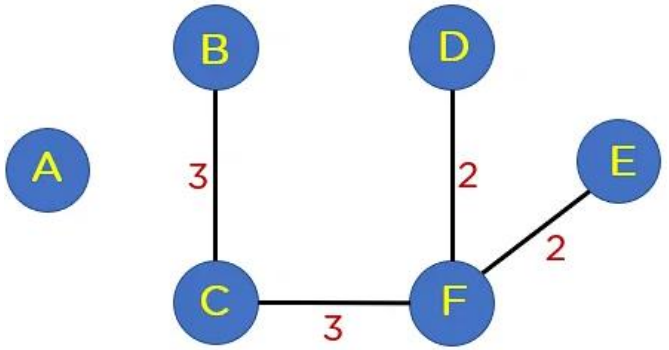


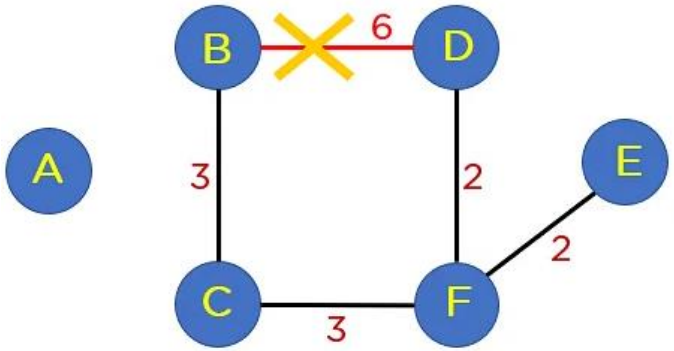
Kruskal Algorithm



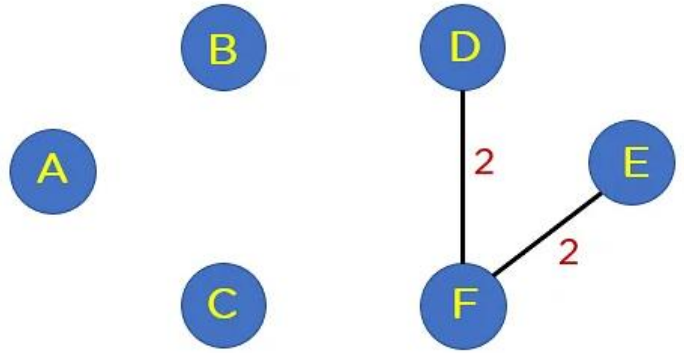
The Edges of the Graph	Edge Weight	
Source Vertex	Destination Vertex	
E	F	2
F	D	2
B	C	3
C	F	3
C	D	4
B	F	5
B	D	6
A	B	7
A	C	8

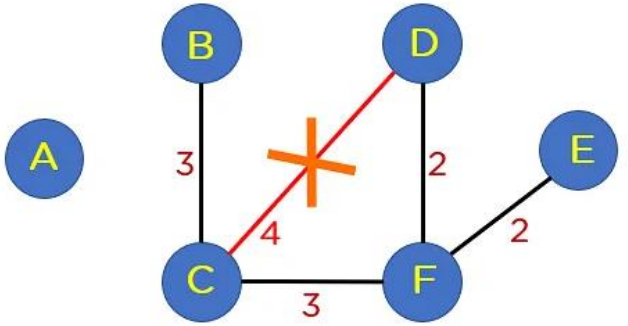




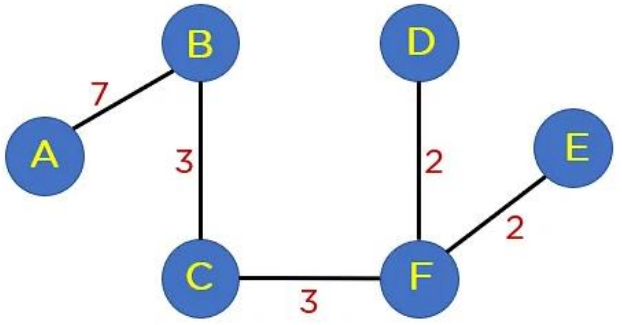


Edge BD should be discarded.





Edge CD should be discarded, as it creates loop.



Minimum Spanning Tree.

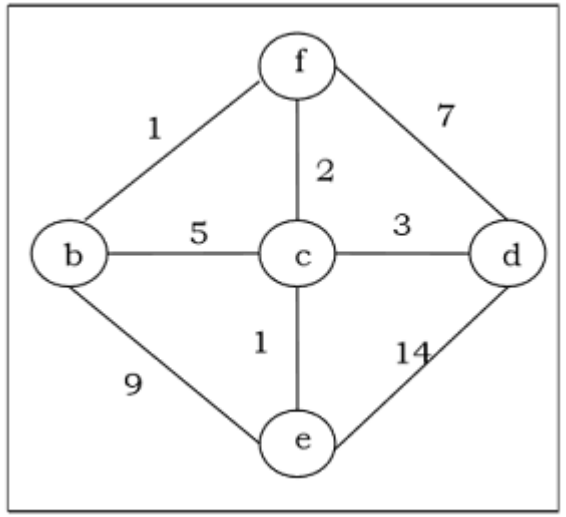
Prim’s Algorithm for MST

- We have discussed Kruskal’s algorithm for Minimum Spanning Tree.
- Like Kruskal’s algorithm, Prim’s algorithm is also a Greedy algorithm.
- This algorithm always starts with a single node and moves through several adjacent nodes, in order to explore all of the connected edges along the way.

Step 1: Determine an arbitrary vertex as the starting vertex of the MST.
Step 2: Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).
Step 3: Find edges connecting any tree vertex with the fringe vertices.
Step 4: Find the minimum among these edges.
Step 5: Add the chosen edge to the MST if it does not form any cycle.
Step 6: Return the MST and exit

Prim's Algorithm for MST

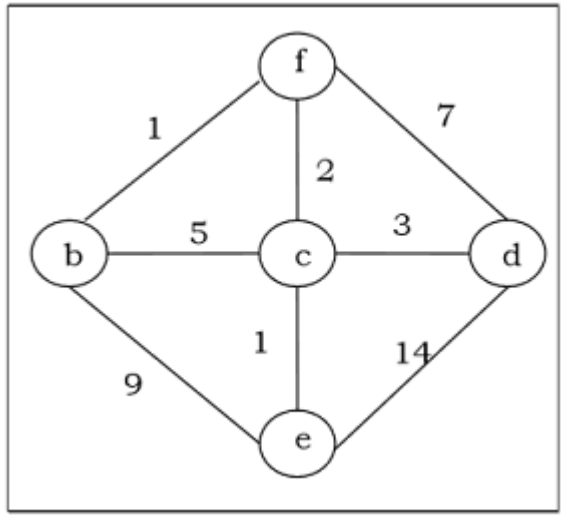
Step 1: Determine an arbitrary vertex as the starting vertex of the MST.
Step 2: Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).
Step 3: Find edges connecting any tree vertex with the fringe vertices.
Step 4: Find the minimum among these edges.
Step 5: Add the chosen edge to the MST if it does not form any cycle.
Step 6: Return the MST and exit



	B-0	C-1	D-2	E-3	F-4
B-0	0	5	0	9	1
C-1	5	0	3	1	2
D-2	0	3	0	14	7
E-3	9	1	14	0	0
F-4	1	2	7	0	0

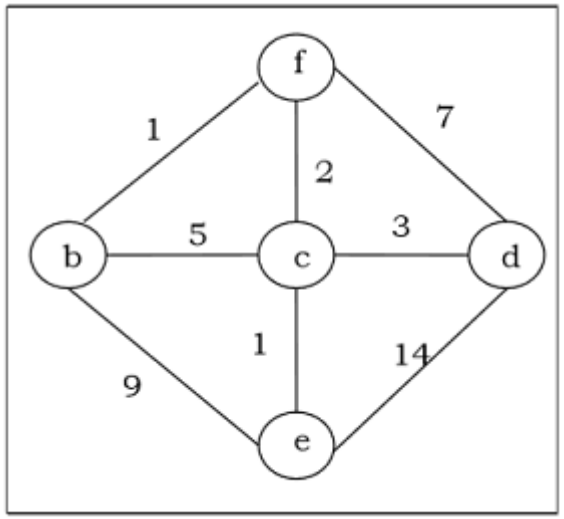
Prim's Algorithm for MST

Step 1: Determine an arbitrary vertex as the starting vertex of the MST.
Step 2: Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).
Step 3: Find edges connecting any tree vertex with the fringe vertices.
Step 4: Find the minimum among these edges.
Step 5: Add the chosen edge to the MST if it does not form any cycle.
Step 6: Return the MST and exit



	B-0	C-1	D-2	E-3	F-4
B-0					
C-1					
D-2					
E-3					
F-4					

Prim’s Algorithm for MST

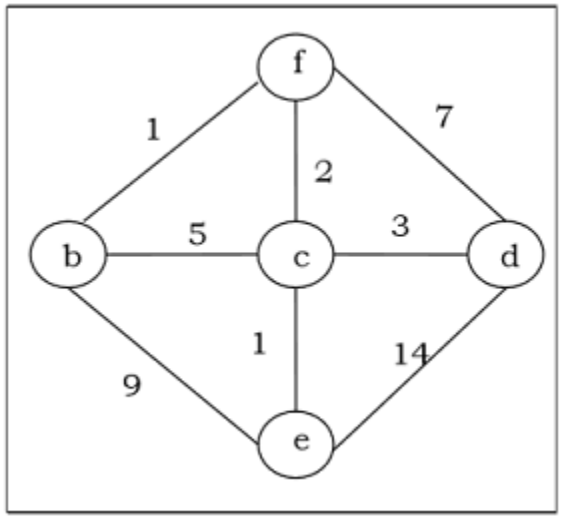


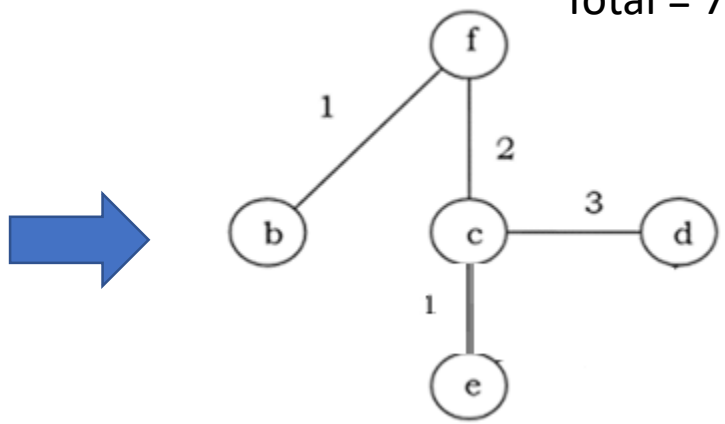
MST (Minimum Spanning Tree) ?

```
90  int main()
91  {
92      int graph[V][V] = { { 0, 5, 0, 9, 1 },
93                          { 5, 0, 3, 1, 2 },
94                          { 0, 3, 0, 14, 7 },
95                          { 9, 1, 14, 0, 0 },
96                          { 1, 2, 7, 0, 0 } };
97
98      primMST(graph);
99      return 0;
```

	B-0	C-1	D-2	E-3	F-4
B-0	0	5	0	9	1
C-1	5	0	3	1	2
D-2	0	3	0	14	7
E-3	9	1	14	0	0
F-4	1	2	7	0	0

Prim's Algorithm for MST





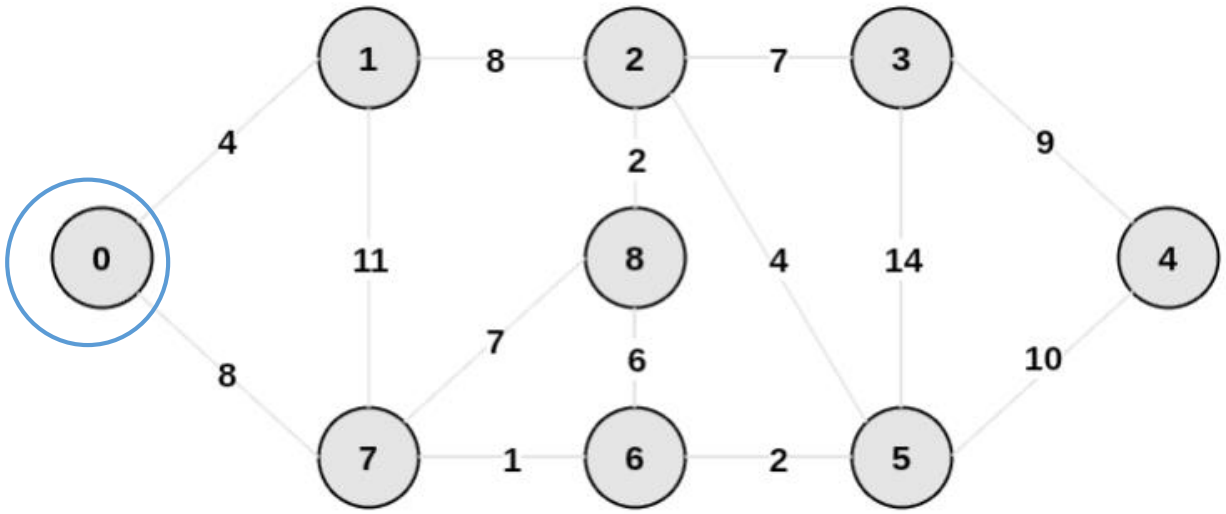

```
90  int main()
91  {
92      int graph[V][V] = { { 0, 5, 0, 9, 1 },
93                          { 5, 0, 3, 1, 2 },
94                          { 0, 3, 0, 14, 7 },
95                          { 9, 1, 14, 0, 0 },
96                          { 1, 2, 7, 0, 0 } };
97
98      primMST(graph);
99      return 0;
```

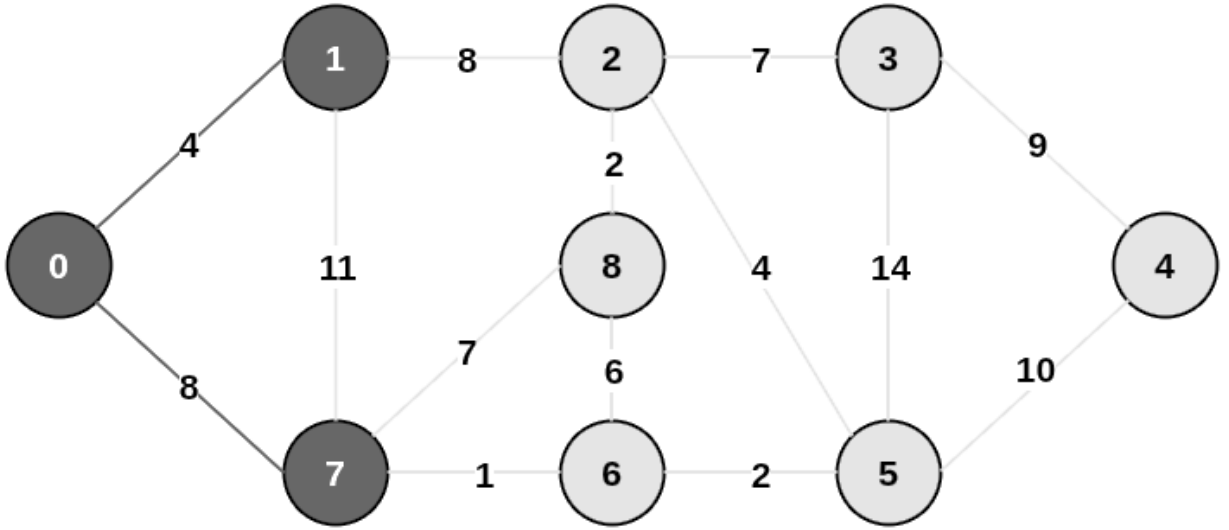
```
Edge      Weight
4 - 1     2
1 - 2     3
1 - 3     1
0 - 4     1
```

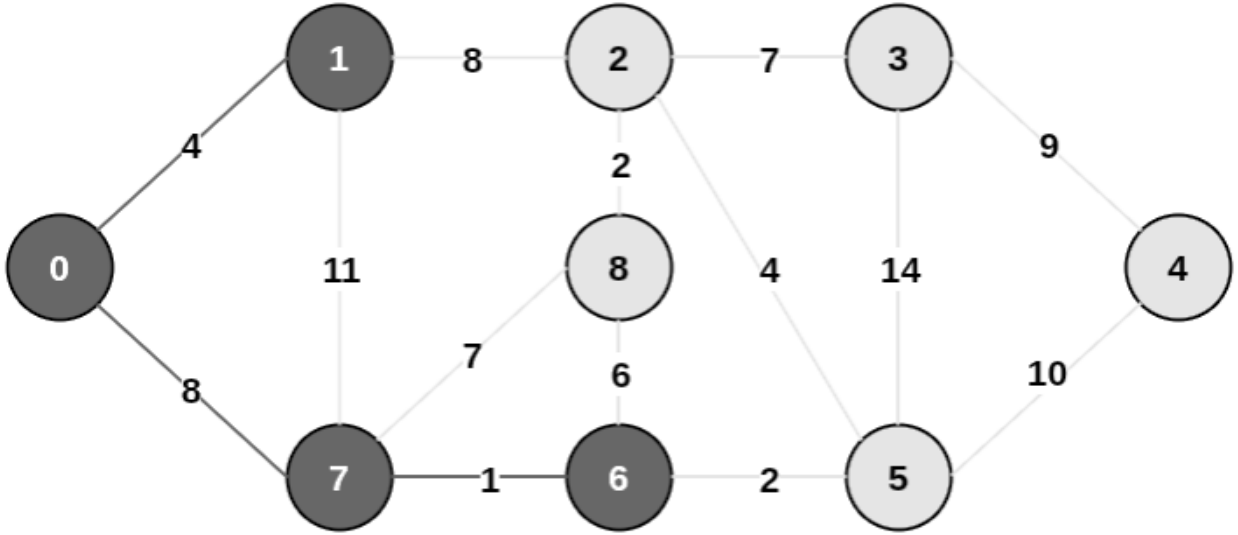
Nodes	
F <-> C	2
C <-> D	3
C <-> E	1
B <-> F	1

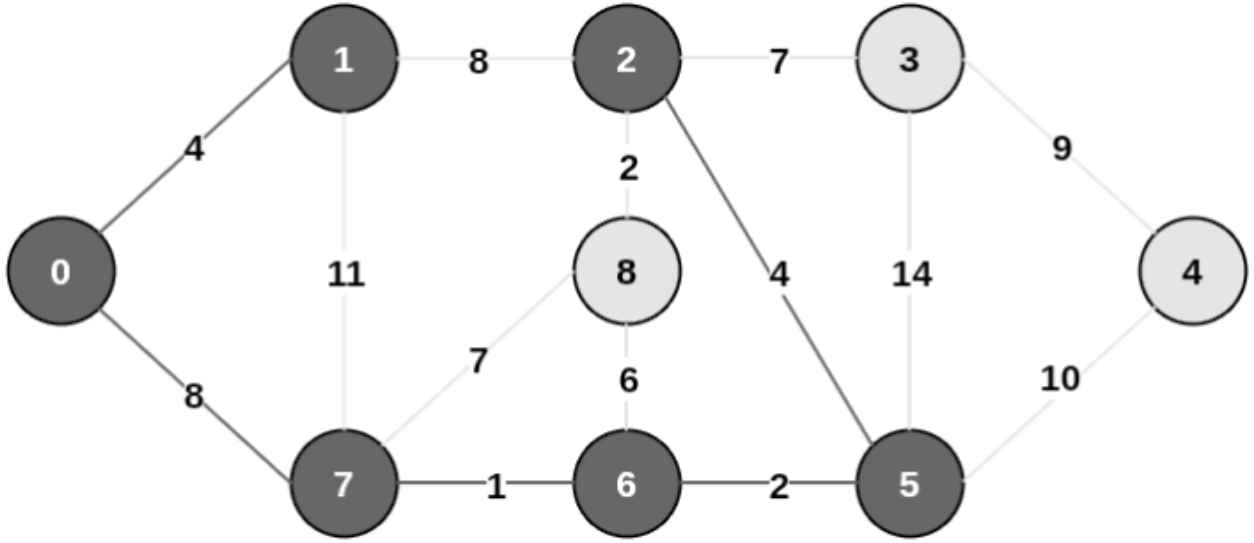
	B-0	C-1	D-2	E-3	F-4
B-0	0	5	0	9	1
C-1	5	0	3	1	2
D-2	0	3	0	14	7
E-3	9	1	14	0	0
F-4	1	2	7	0	0

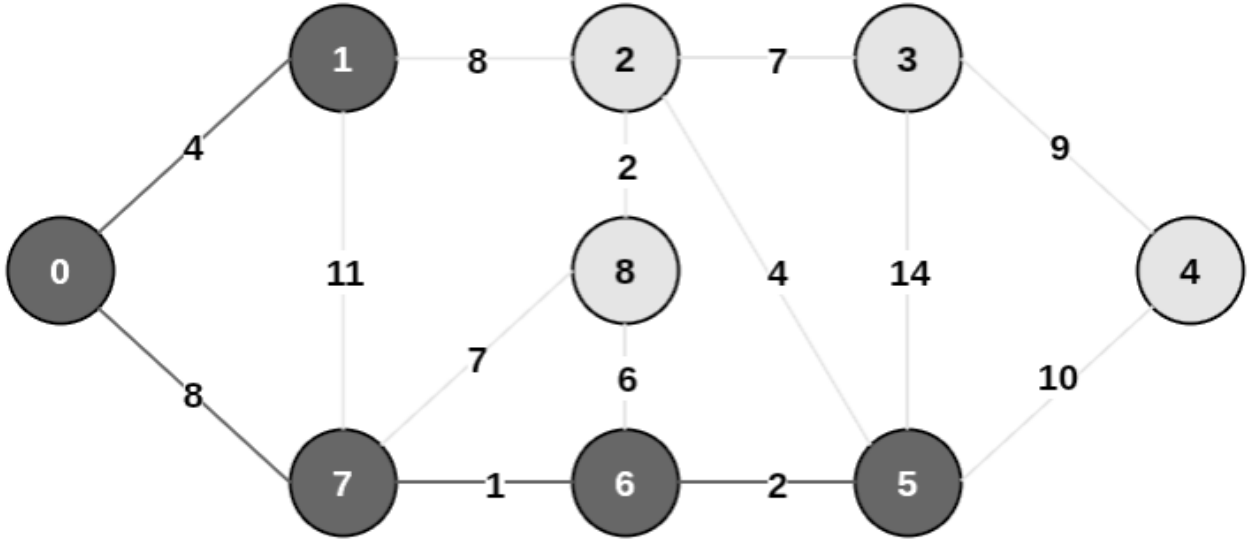
Prim’s Algorithm

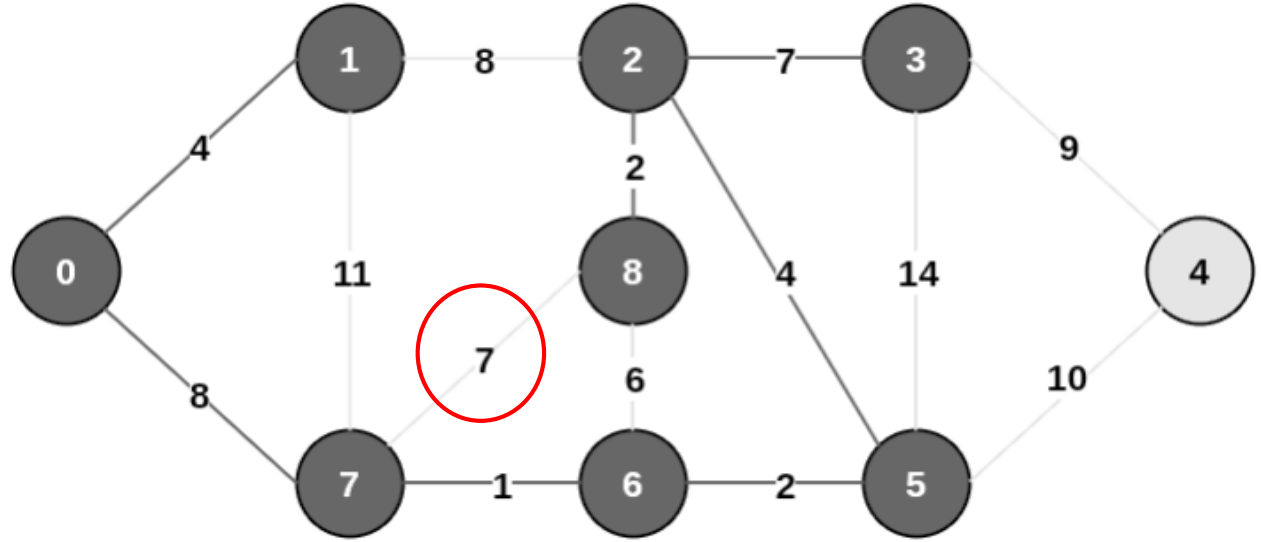


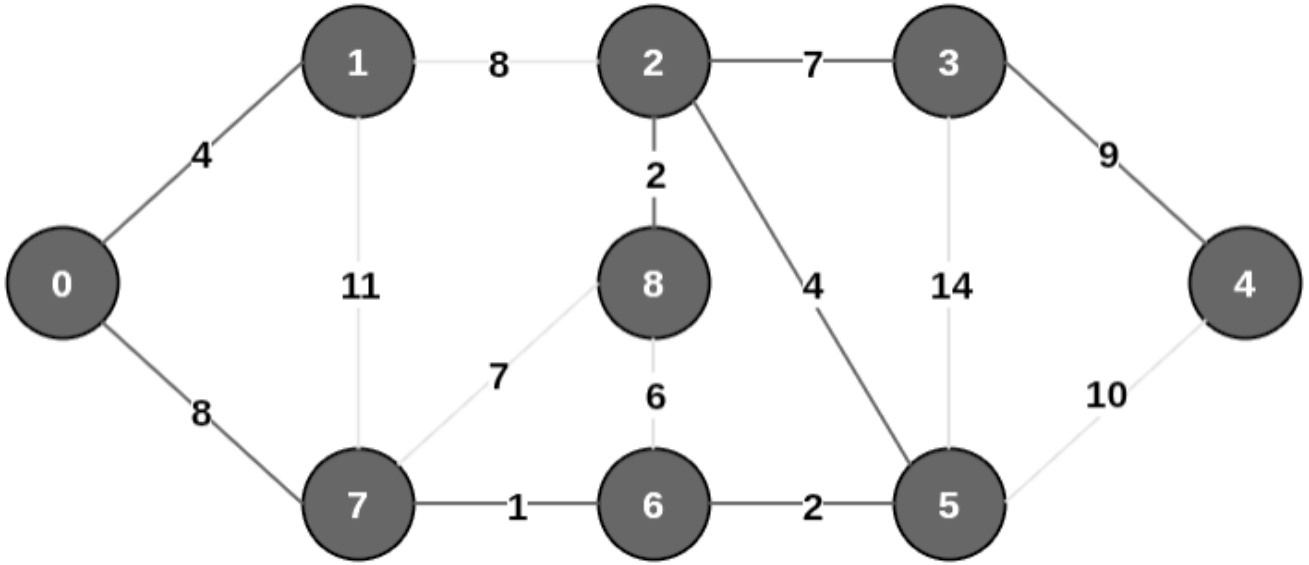


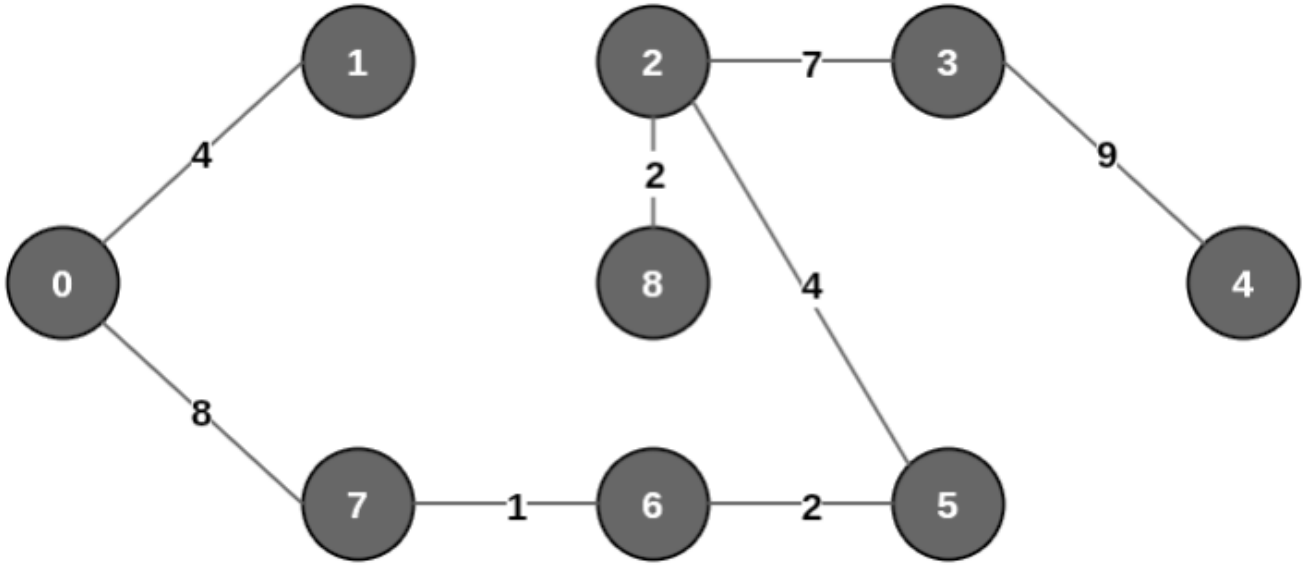

0
1
2
3
4
5
6
7
8
4
8
8
7
11
2
7
4
14
9
10
6
1
2

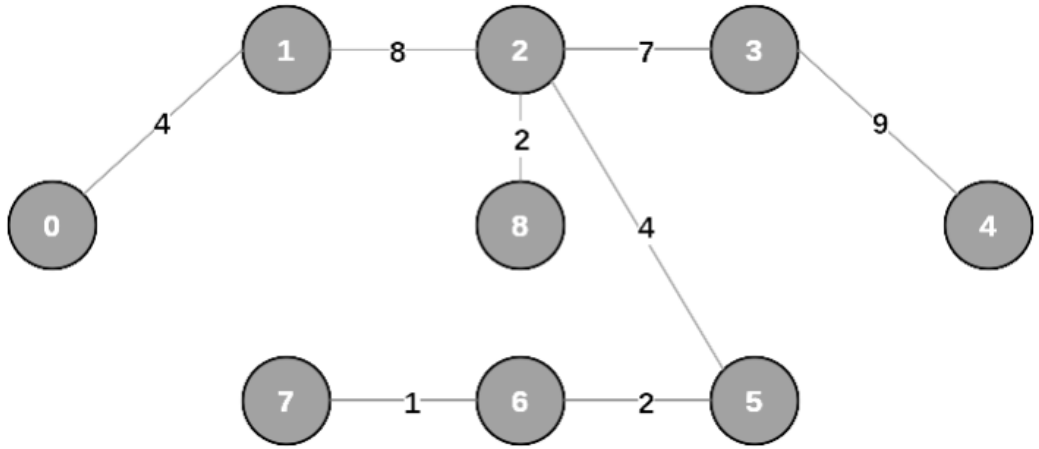

0
1
2
3
4
5
6
7
8
4
8
8
7
11
2
7
4
14
9
10
6
1
2


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

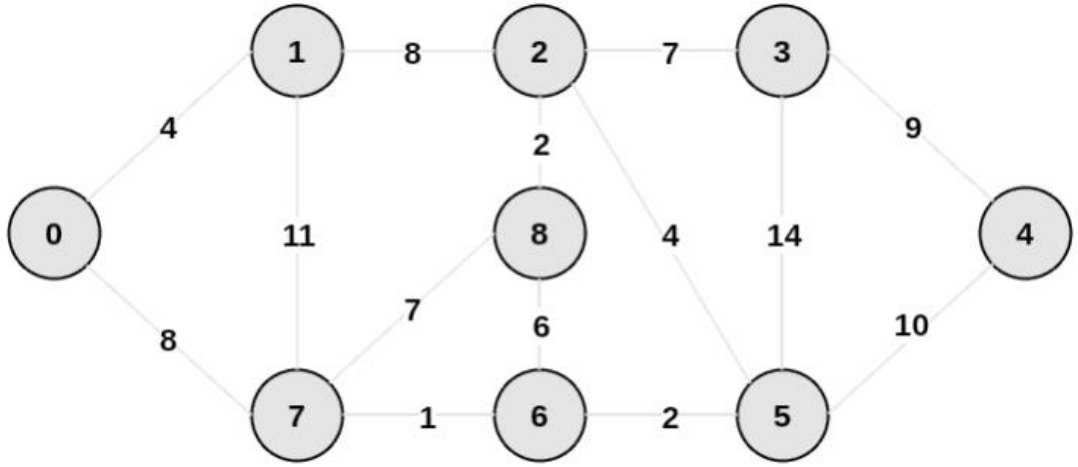

0
1
2
3
4
5
6
7
8
4
8
8
7
11
2
7
4
14
9
10
6
1
2


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


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


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

Prim’s Algorithm (Steps 1/3)

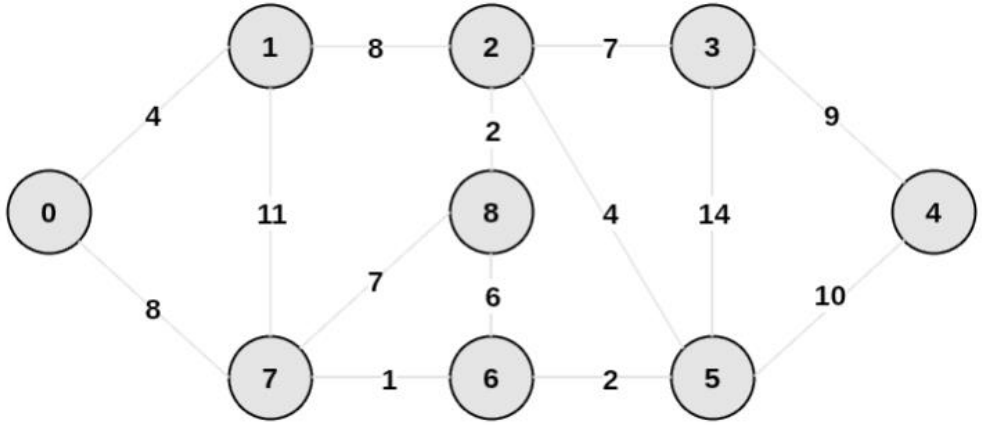


Step 1: *Firstly, we select an arbitrary vertex that acts as the starting vertex of the Minimum Spanning Tree. Here we have selected vertex* ***0*** *as the starting vertex.*

Step 2: *All the edges connecting the incomplete MST and other vertices are the edges {0, 1} and {0, 7}. Between these two the edge with minimum weight is {0, 1}. So include the edge and vertex 1 in the MST.*

Step 3: *The edges connecting the incomplete MST to other vertices are {0, 7}, {1, 7} and {1, 2}. Among these edges the minimum weight is 8 which is of the edges {0, 7} and {1, 2}. Let us here include the edge {0, 7} and the vertex 7 in the MST. [We could have also included edge {1, 2} and vertex 2 in the MST].*

Prim’s Algorithm (Steps 2/3)



Step 4: *The edges that connect the incomplete MST with the fringe vertices are {1, 2}, {7, 6} and {7, 8}. Add the edge {7, 6} and the vertex 6 in the MST as it has the least weight (i.e., 1).*

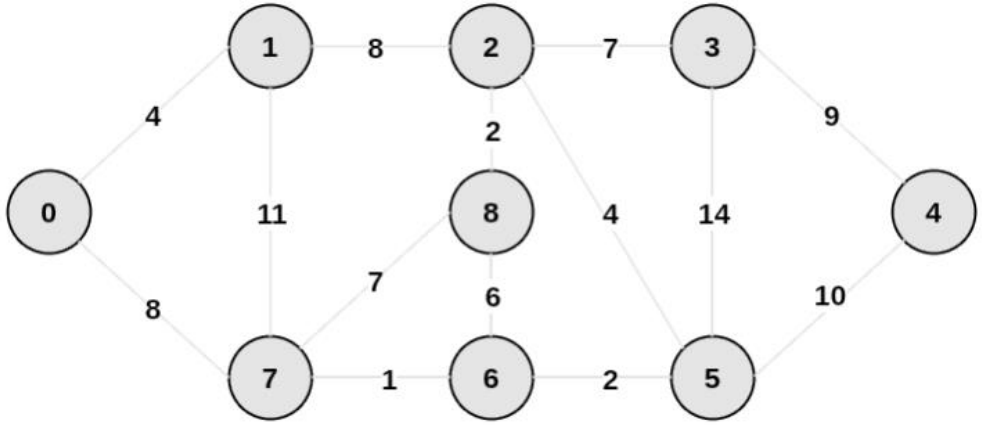
Step 5: *The connecting edges now are {7, 8}, {1, 2}, {6, 8} and {6, 5}. Include edge {6, 5} and vertex 5 in the MST as the edge has the minimum weight (i.e., 2) among them.*

Step 6: *Among the current connecting edges, the edge {5, 2} has the minimum weight. So include that edge and the vertex 2 in the MST.*

Step 7: *The connecting edges between the incomplete MST and the other edges are {2, 8}, {2, 3}, {5, 3} and {5, 4}. The edge with minimum weight is edge {2, 8} which has weight 2. So include this edge and the vertex 8 in the MST.*

Step 8: *See here that the edges {7, 8} and {2, 3} both have same weight which are minimum. But 7 is already part of MST. So we will consider the edge {2, 3} and include that edge and vertex 3 in the MST.*

Prim’s Algorithm (Steps 3/3)



Step 9: *Only the vertex 4 remains to be included. The minimum weighted edge from the incomplete MST to 4 is {3, 4}.*
The final structure of the MST is as follows and the weight of the edges of the MST is (4 + 8 + 1 + 2 + 4 + 2 + 7 + 9) = ***37****.*

- The prim's algorithm selects the root vertex in the beginning and then traverses from vertex to vertex adjacently.
- On the other hand, Krushal's algorithm helps in generating the minimum spanning tree, initiating from the smallest weighted edge.

Minimum Spanning Tree (MST) - Applications

- Minimum spanning tree analysis of brain networks: A systematic review of network size effects, sensitivity for neuropsychiatric pathology, and disorder specificity (link)

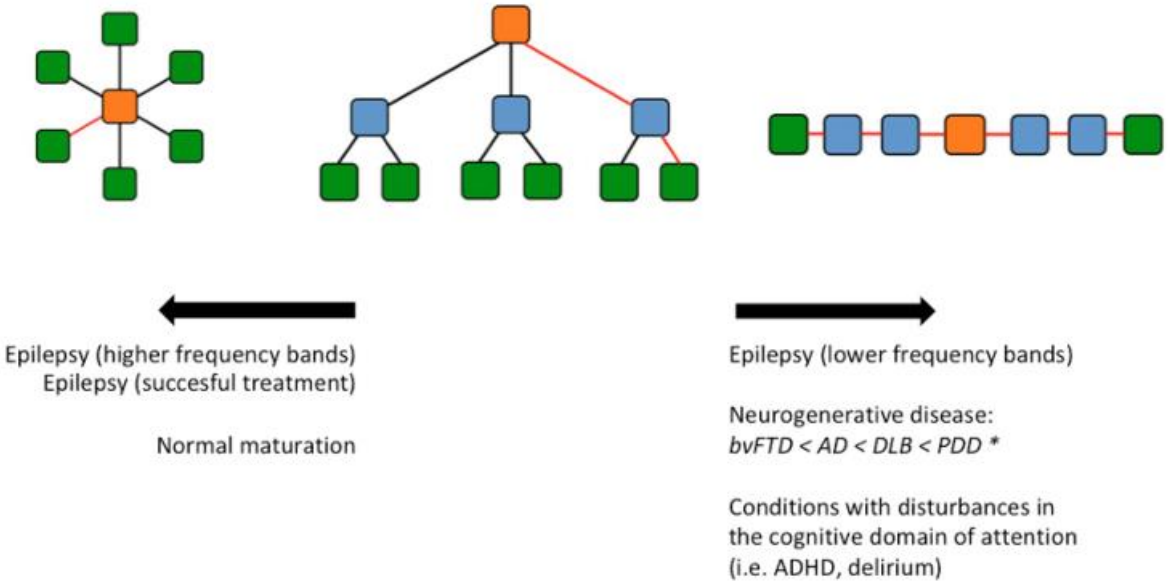
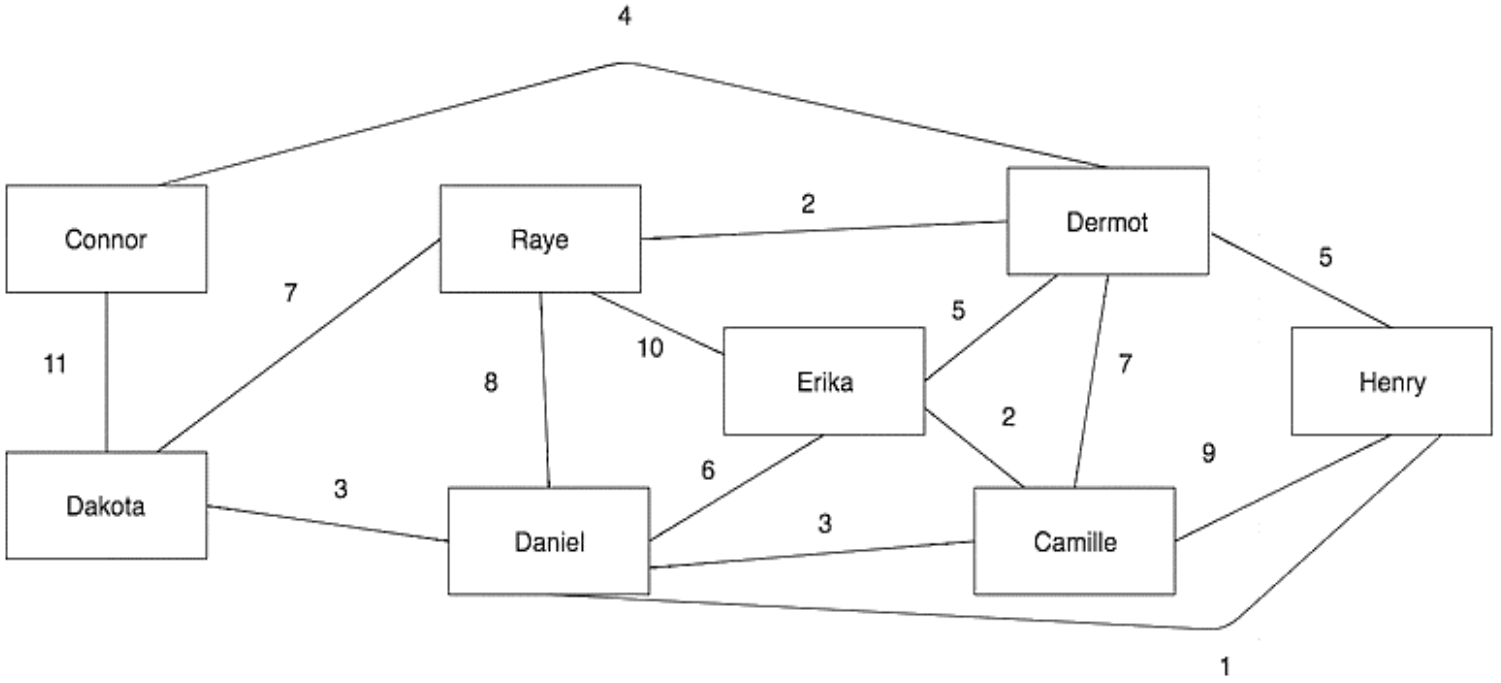


Figure 1. Schematic depiction of three different minimum spanning trees, with a starlike, intermediate and linelike configuration from left to right. The green nodes represent leaf nodes. Central nodes are depicted in orange. Diameter is depicted in red. Individual conditions and the correlated changes in network topology as described in the discussion section are displayed, with an arrow depicting the direction of the change. For neurodegenerative diseases conditions are displayed left to right from having the least shift toward a more linelike topology (bvFTD) to the most (PDD). AD, Alzheimer's disease; bvFTD, behavioral variant of frontotemporal dementia; DLB, dementia with Lewy bodies; PDD, Parkinson's disease dementia.

Minimum Spanning Tree (MST) - Applications

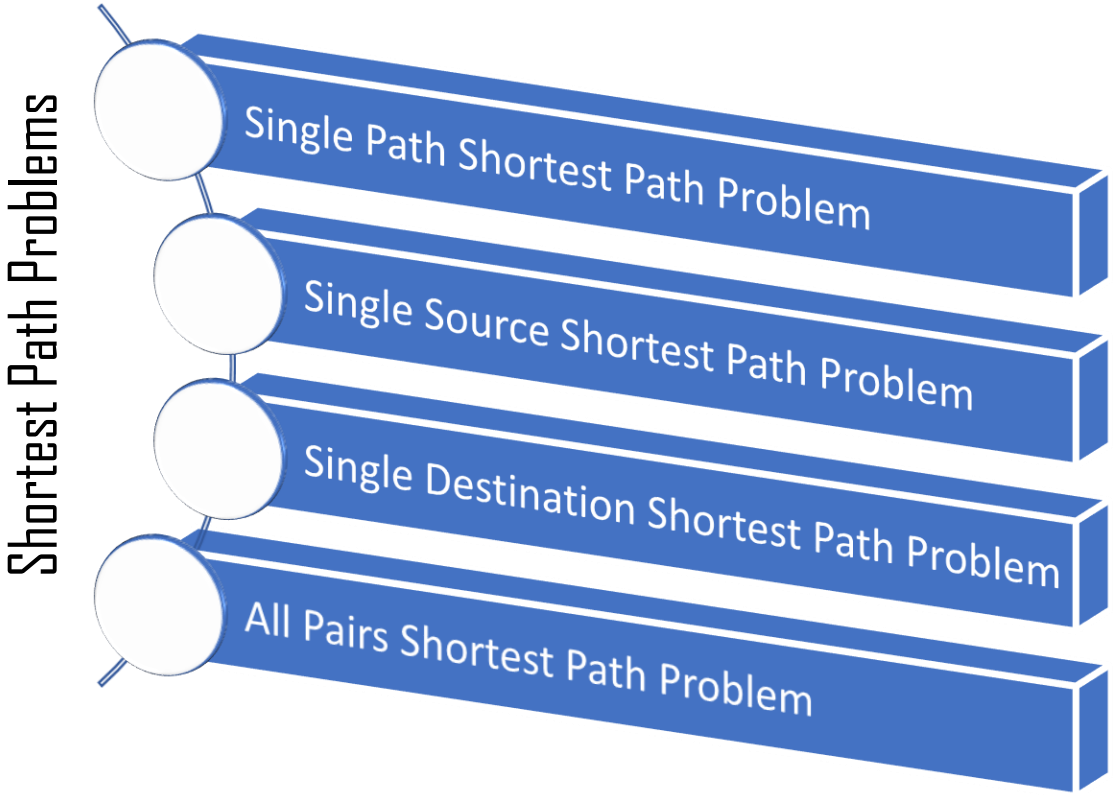
- Graph Algorithms: Minimum Spanning Trees for Social Network Analysis (link)

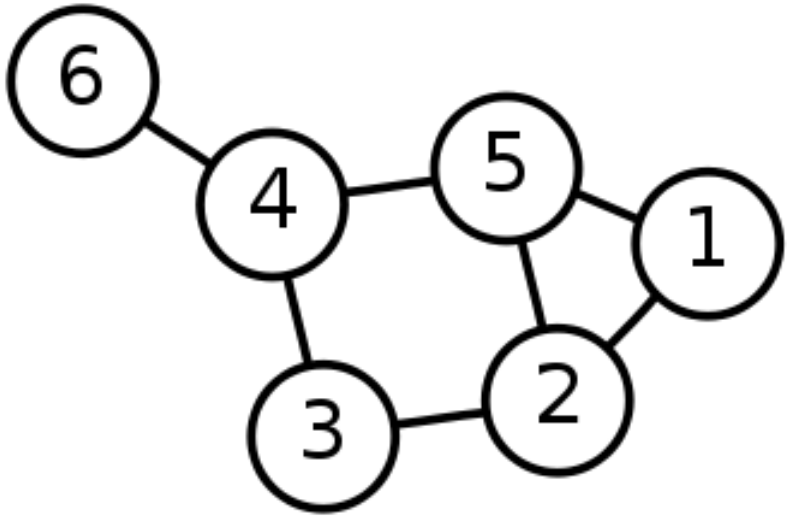
These edges quantify how much Users are interacting with one another. Dermot and Henry exchange 5 likes or comments every 24 hours, Daniel and Henry exchange 1 like or comment every 24 hours, Connor and Dakota exchange 11 likes every 24 hours



Single-Source Shortest Path Problem (SSSP)

- The problem of finding shortest paths from a source vertex *v* to all other vertices in the graph.
- Algorithms such as Breadth-First-Search (BFS) for unweighted graphs or Dijkstra solve this problem.





Dijkstra's algorithm

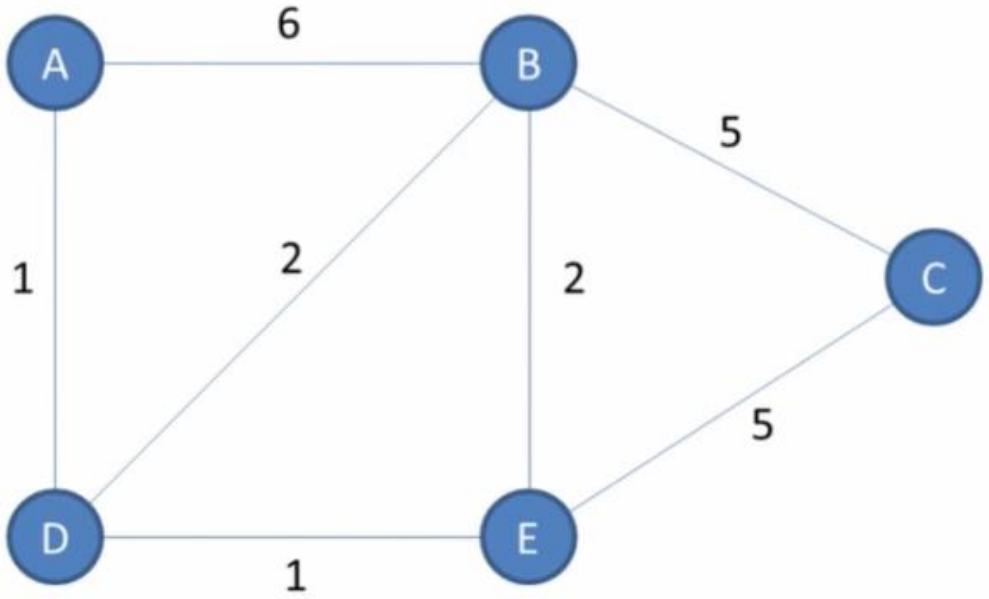
- **Dijkstra's algorithm** - is a solution to the single-source shortest path problem in graph theory.
  - With Dijkstra's Algorithm, you can find the shortest path between nodes in a graph. Particularly, you can find the shortest path from a node (called the "source node") to all other nodes in the graph, producing a shortest-path tree.
  - This algorithm is used in GPS devices to find the shortest path between the current location and the destination. It has broad applications in industry, specially in domains that require modeling networks.

- Works on both directed and undirected graphs. However, all edges must have nonnegative weights.

- Approach: Greedy
- Input: Weighted graph G={E,V} and source vertex, such that all edge weights are nonnegative
- Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex to all other vertices

Dijkstra’s Shortest Path Algorithm

- **Objective:** Find the shorted path from the given starting vertex (A) to every other graph

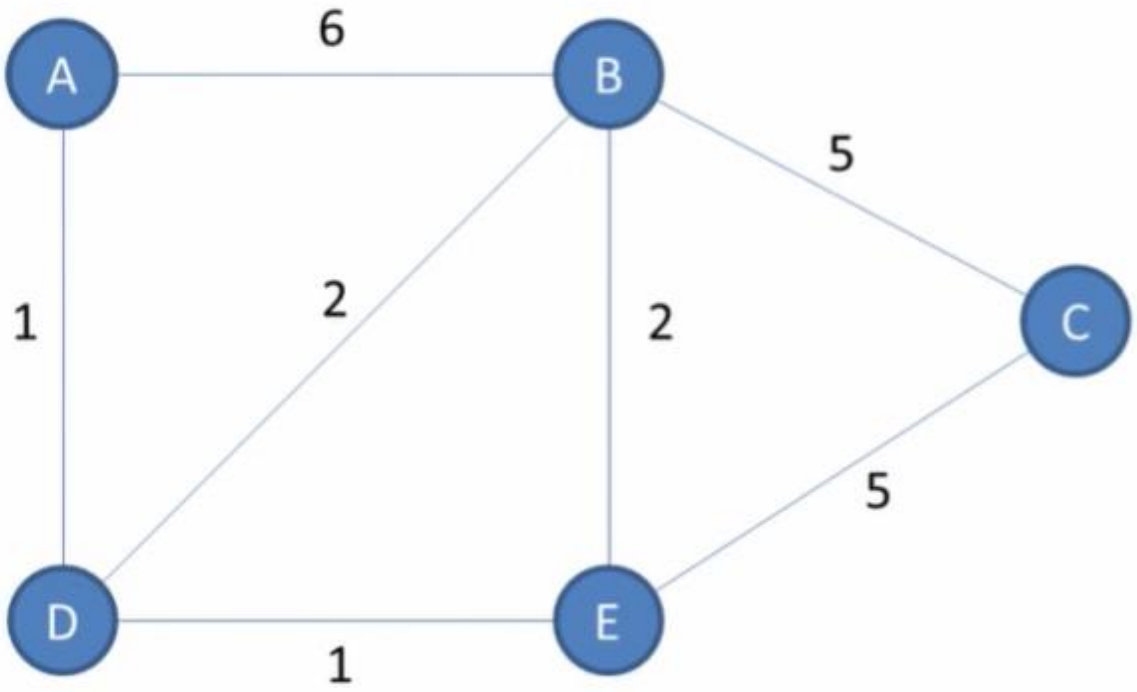


- **Dijkstra's Algorithm**

1. Dijkstra's Algorithm basically starts at the node that you choose (the source node) and it analyzes the graph to find the shortest path between that node and all the other nodes in the graph.
2. The algorithm keeps track of the currently known shortest distance from each node to the source node and it updates these values if it finds a shorter path.
3. Once the algorithm has found the shortest path between the source node and another node, that node is marked as "visited" and added to the path.
4. The process continues until all the nodes in the graph have been added to the path. This way, we have a path that connects the source node to all other nodes following the shortest path possible to reach each node

Dijkstra's Shortest Path Algorithm

- **Objective:** Find the shorted path from the given starting vertex (A) to every other graph



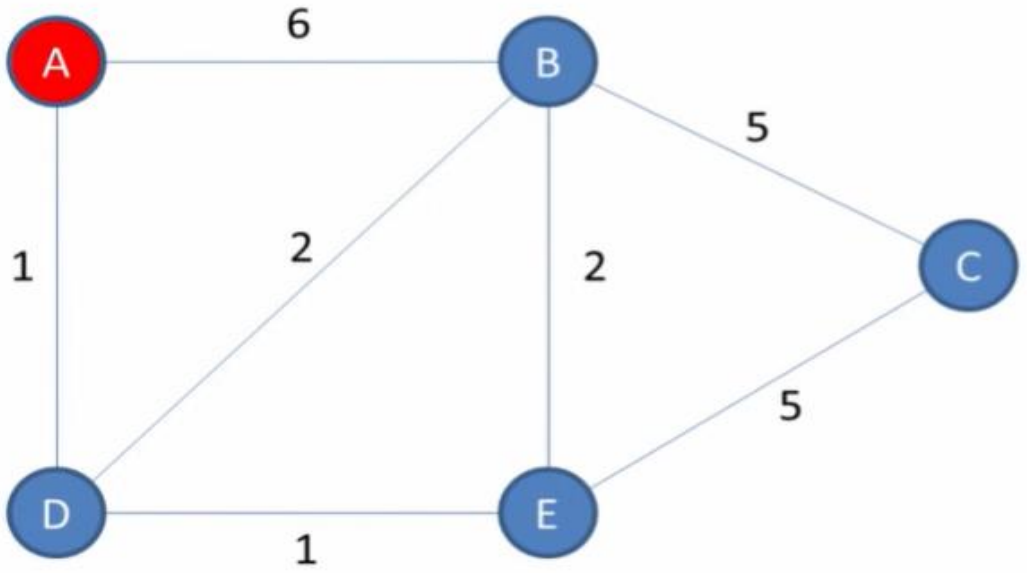
Vertex	Shortest distance from A	Previous vertex
A		
B		
C		
D		
E		

Visited = [ ] Unvisited = [A, B, C, D, E]

Dijkstra's Shortest Path Algorithm

- **Objective:** Find the shorted path from the given starting vertex (A) to every other graph

Consider the start vertex, A
Distance to A from A = 0
Distances to all other vertices from A are unknown, therefore ∞ (infinity)



Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

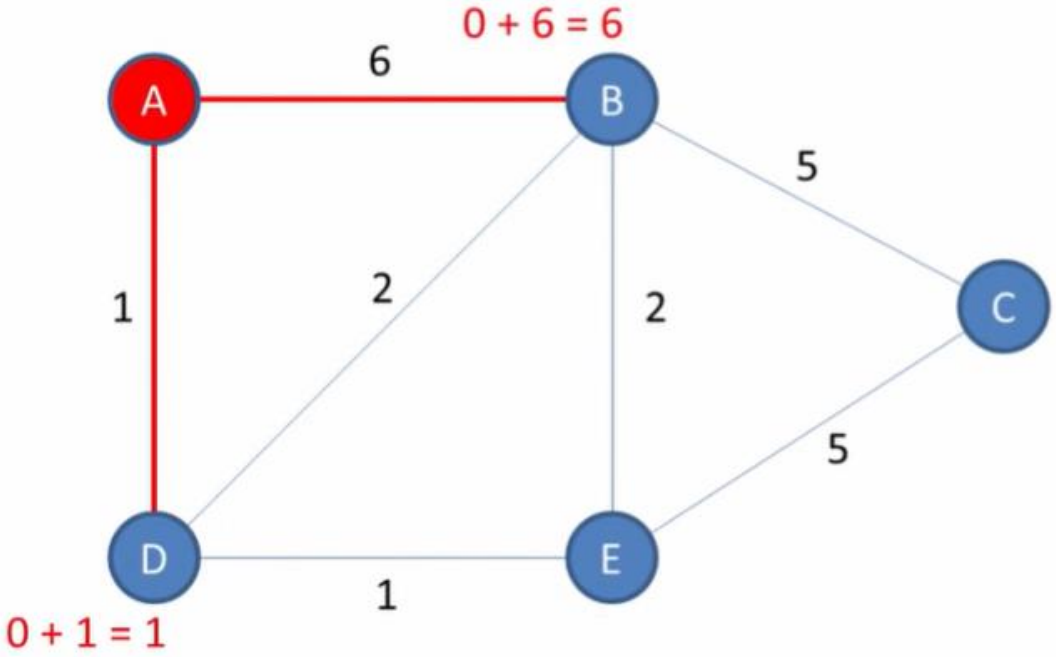
Visited = [ ]

Unvisited = [A, B, C, D, E]

Dijkstra's Shortest Path Algorithm

- **Objective:** Find the shorted path from the given starting vertex (A) to every other graph

If the calculated distance of a vertex is less than the known distance, update the shortest distance



Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	
C	∞	
D	1	
E	∞	

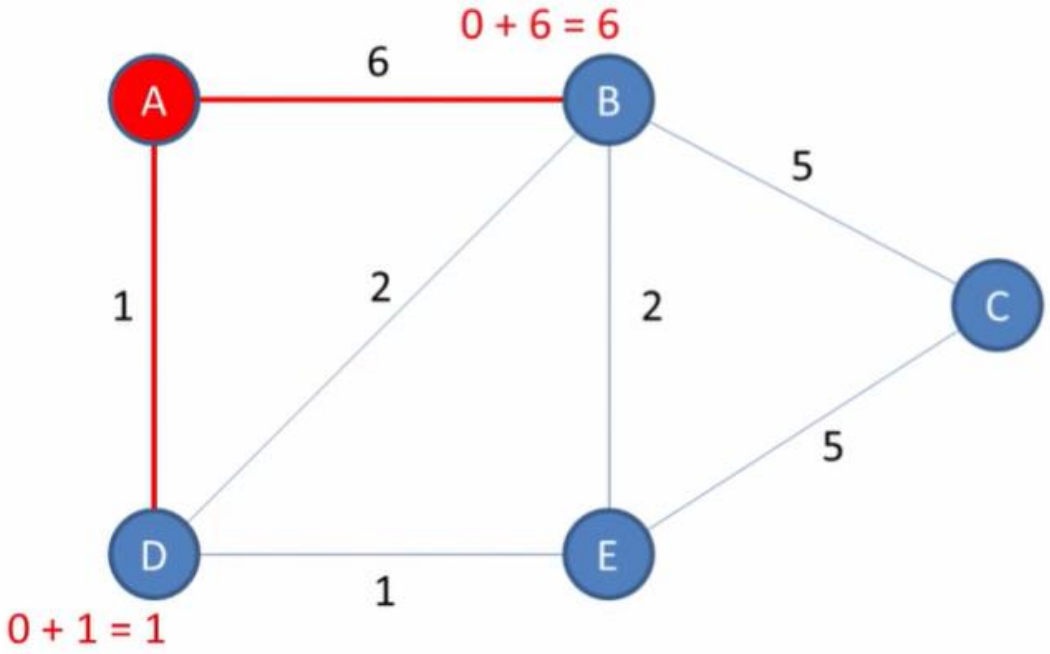
Visited = [ ]

Unvisited = [A, B, C, D, E]

Dijkstra's Shortest Path Algorithm

- **Objective:** Find the shorted path from the given starting vertex (A) to every other graph

Update the previous vertex for each of the updated distances
In this case we visited B and D via A



Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

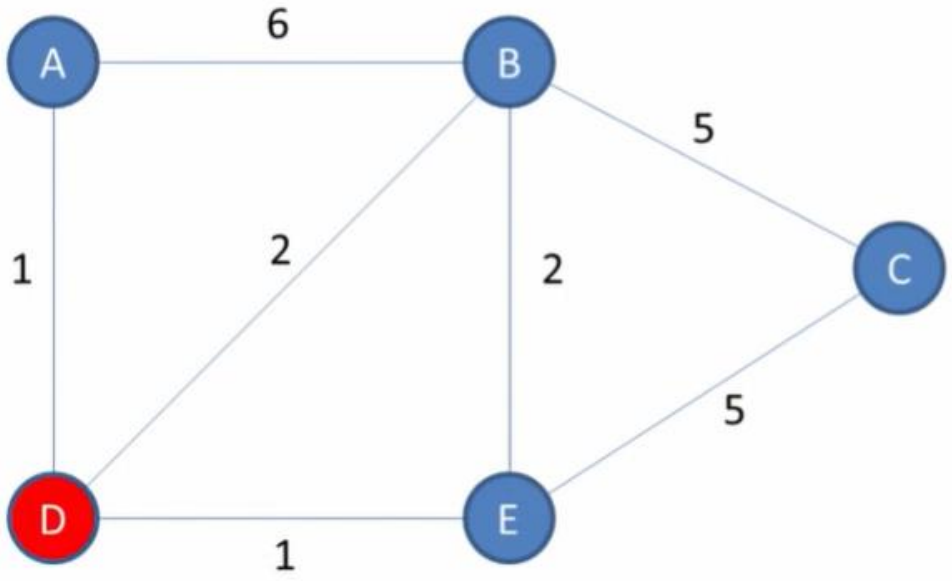
Visited = [ ]

Unvisited = [A, B, C, D, E]

Dijkstra's Shortest Path Algorithm

- **Objective:** Find the shorted path from the given starting vertex (A) to every other graph

Visit the unvisited vertex with the smallest known distance from the start vertex
This time around, it is vertex D



Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

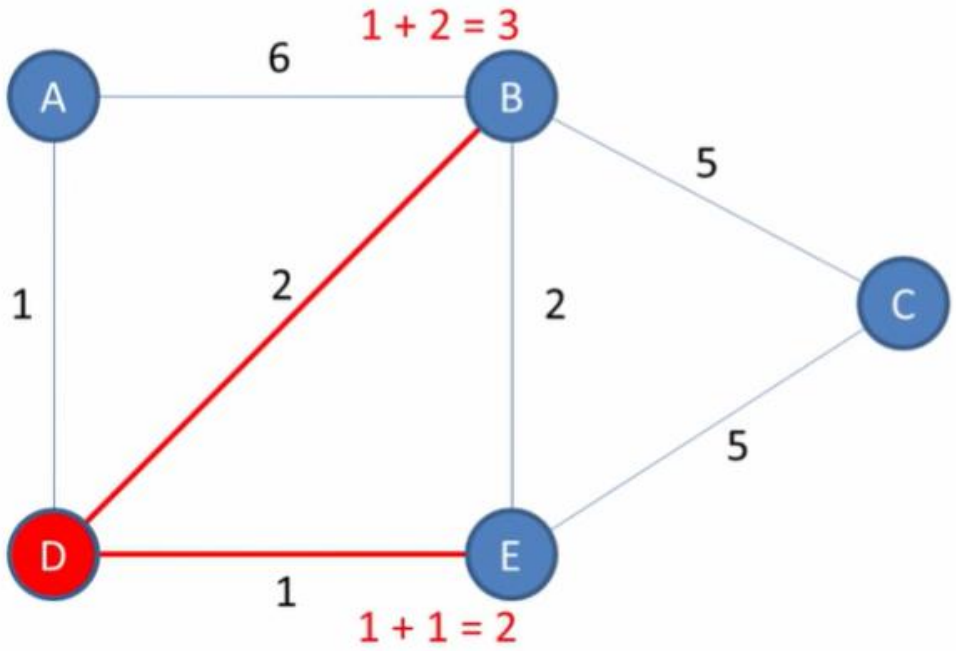
Visited = [A]

Unvisited = [B, C, D, E]

Dijkstra's Shortest Path Algorithm

- **Objective:** Find the shorted path from the given starting vertex (A) to every other graph

If the calculated distance of a vertex is less than the known distance, update the shortest distance



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	A
C	∞	
D	1	A
E	2	

Update new weight

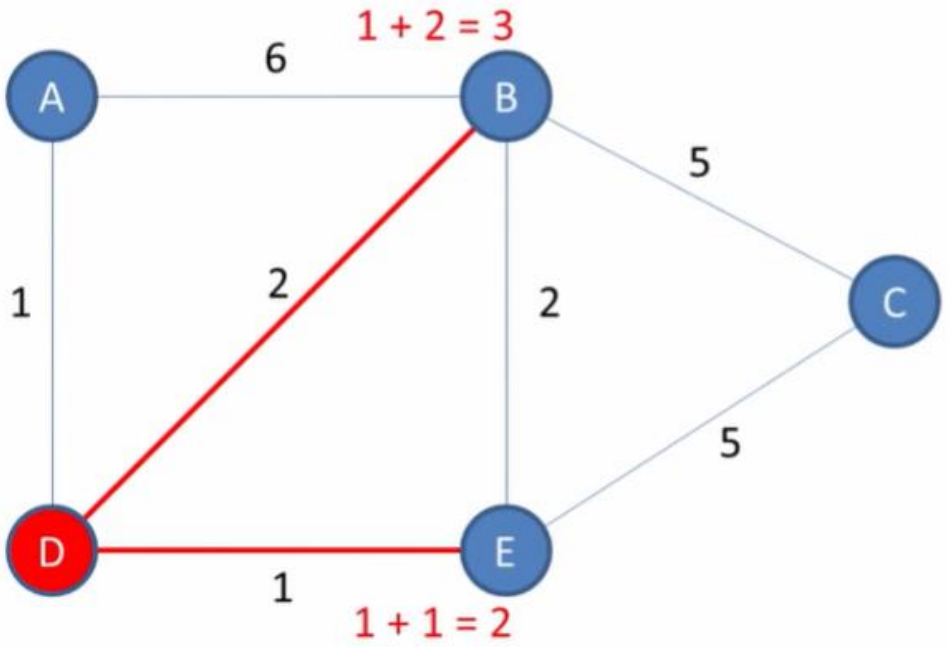
Visited = [A]

Unvisited = [B, C, D, E]

Dijkstra's Shortest Path Algorithm

- **Objective:** Find the shorted path from the given starting vertex (A) to every other graph

Update the previous vertex for each of the updated distances
In this case we visited B and E via D



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Update new path

Visited = [A]

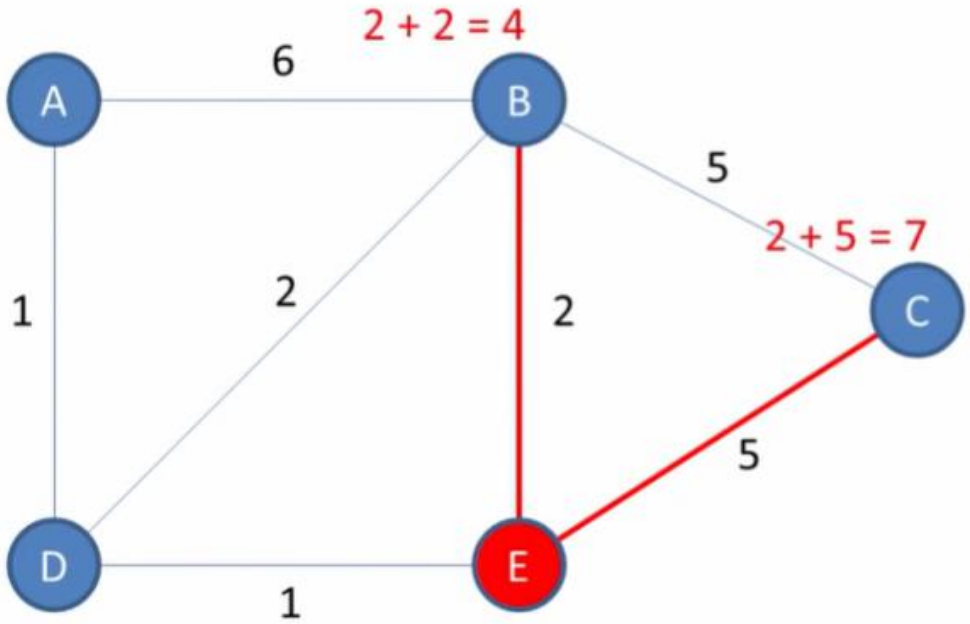
Unvisited = [B, C, D, E]

Dijkstra's Shortest Path Algorithm

- **Objective:** Find the shorted path from the given starting vertex (A) to every other graph

If the calculated distance of a vertex is less than the known distance, update the shortest distance

We do not need to update the distance to B



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	
D	1	A
E	2	D

Visited = [A, D]

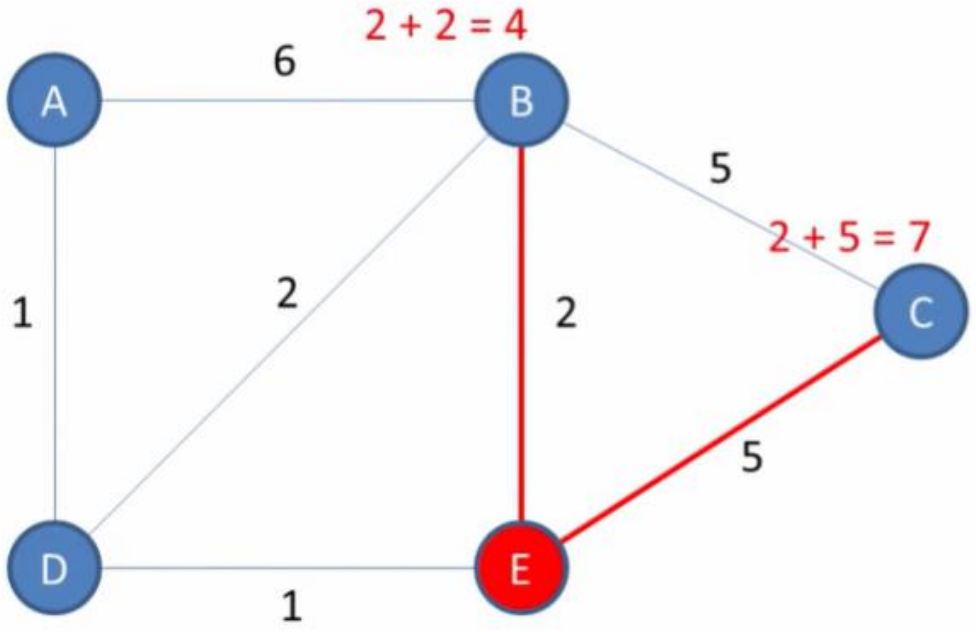
Unvisited = [B, C, E]

Dijkstra's Shortest Path Algorithm

- **Objective:** Find the shorted path from the given starting vertex (A) to every other graph

Update the previous vertex for each of the updated distances

In this case we visited C via E



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D]

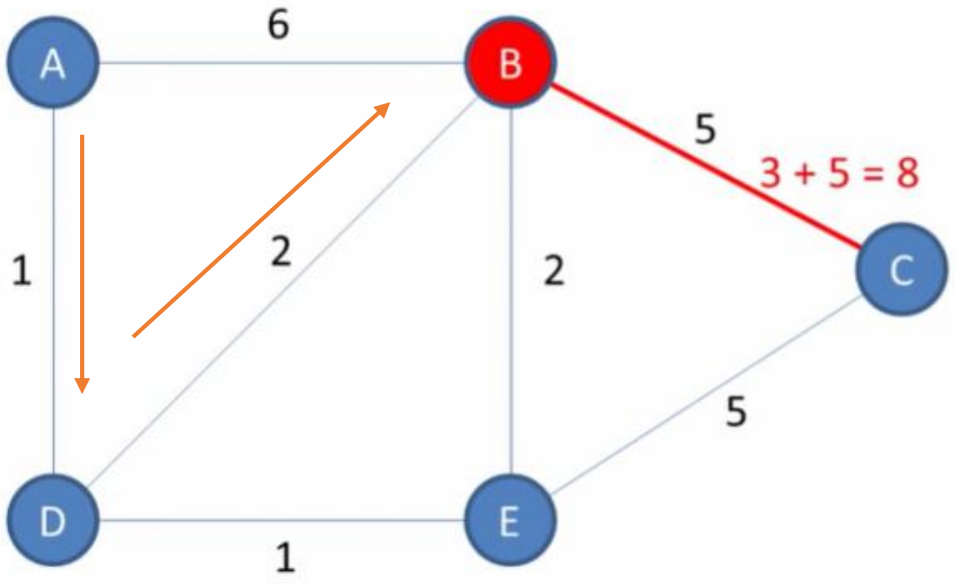
Unvisited = [B, C, E]

Dijkstra's Shortest Path Algorithm

- **Objective:** Find the shorted path from the given starting vertex (A) to every other graph

If the calculated distance of a vertex is less than the known distance, update the shortest distance

We do not need to update the distance to C



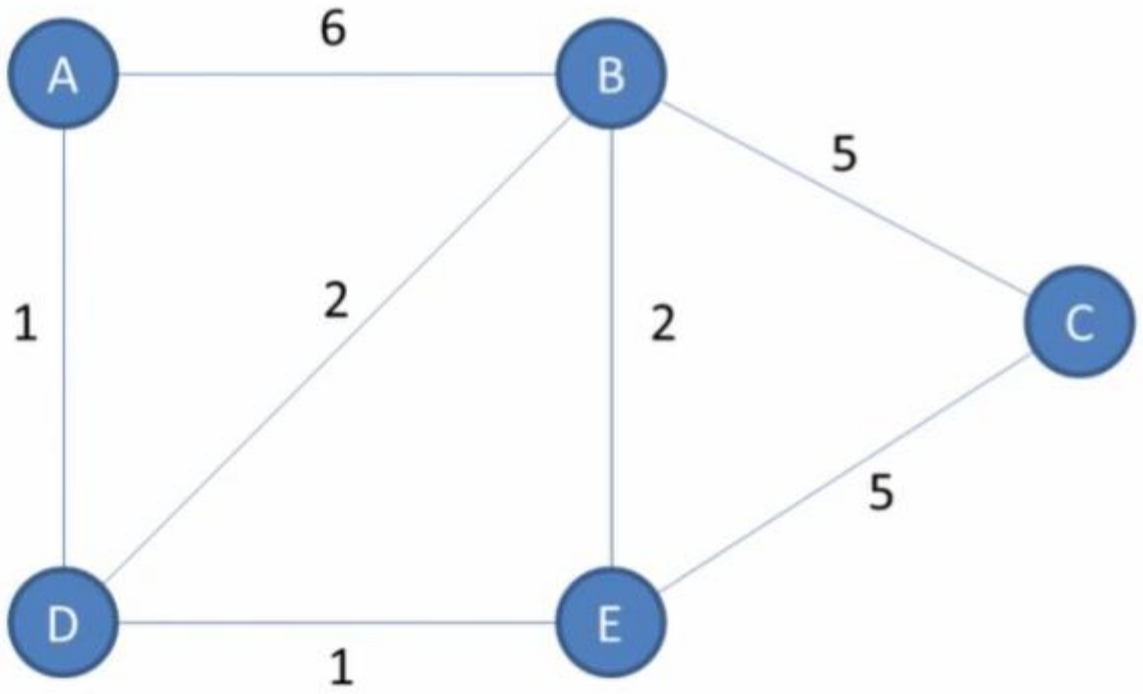
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

We do not need to update it

Visited = [A, D, E] Unvisited = [B, C]

Dijkstra's Shortest Path Algorithm

- **Objective:** Find the shorted path from the given starting vertex (A) to every other graph



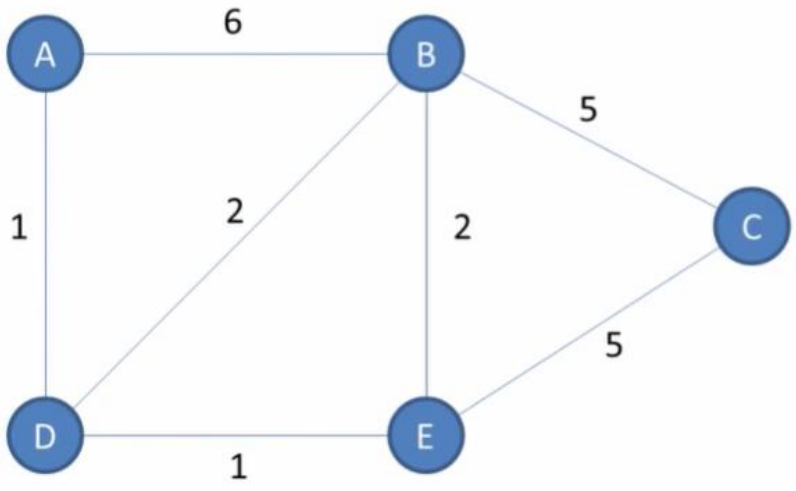
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B, C] Unvisited = []

This shows the shortest path when we have A as the starting vector

Dijkstra's Shortest Path Algorithm

- **Objective:** Find the shorted path from the given starting vertex (A) to every other graph



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

```
87  int main()
88  {
89      int graph[V][V] = { { 0, 6, 0, 1, 0 },
90                          { 6, 0, 5, 2, 2 },
91                          { 0, 5, 0, 0, 5 },
92                          { 1, 2, 0, 0, 1 },
93                          { 0, 2, 5, 1, 0 }
94                           };
95
96      dijkstra(graph, 0);
```

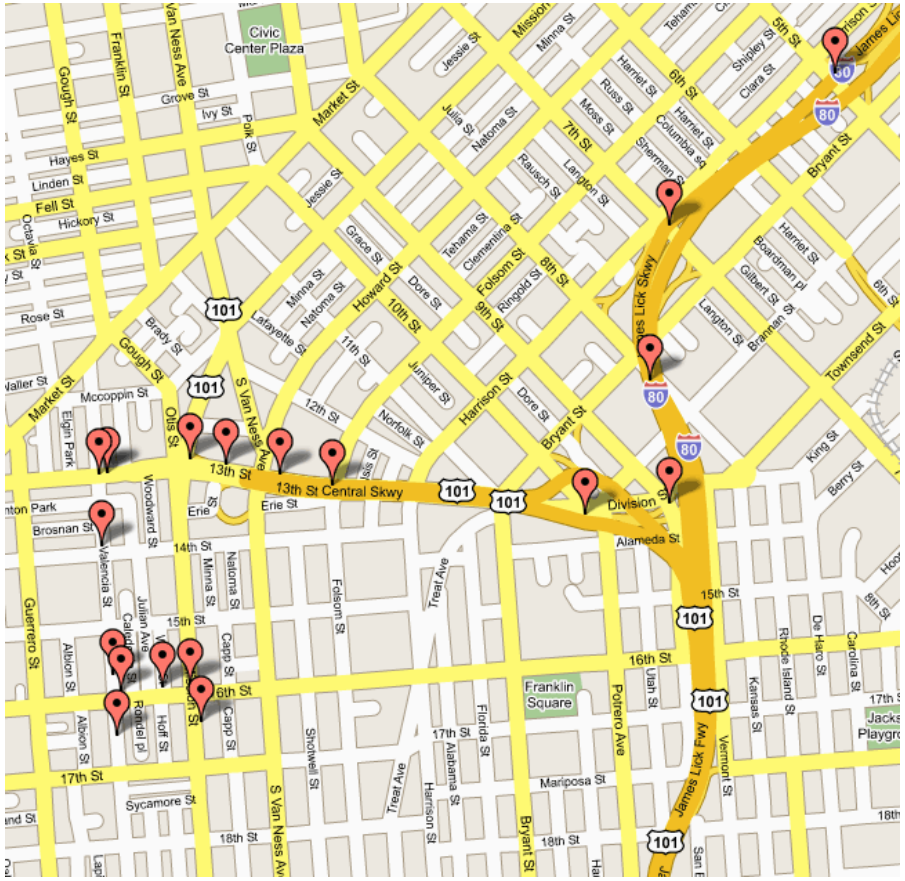
```
Vertex                  Distance from Source
0                                       0
1                                       3
2                                       7
3                                       1
4                                       2
```

Visited = [A, D, E, B, C] Unvisited = []

This shows the shortest path when we have A as the starting vector

Applications of Dijkstra's Algorithm

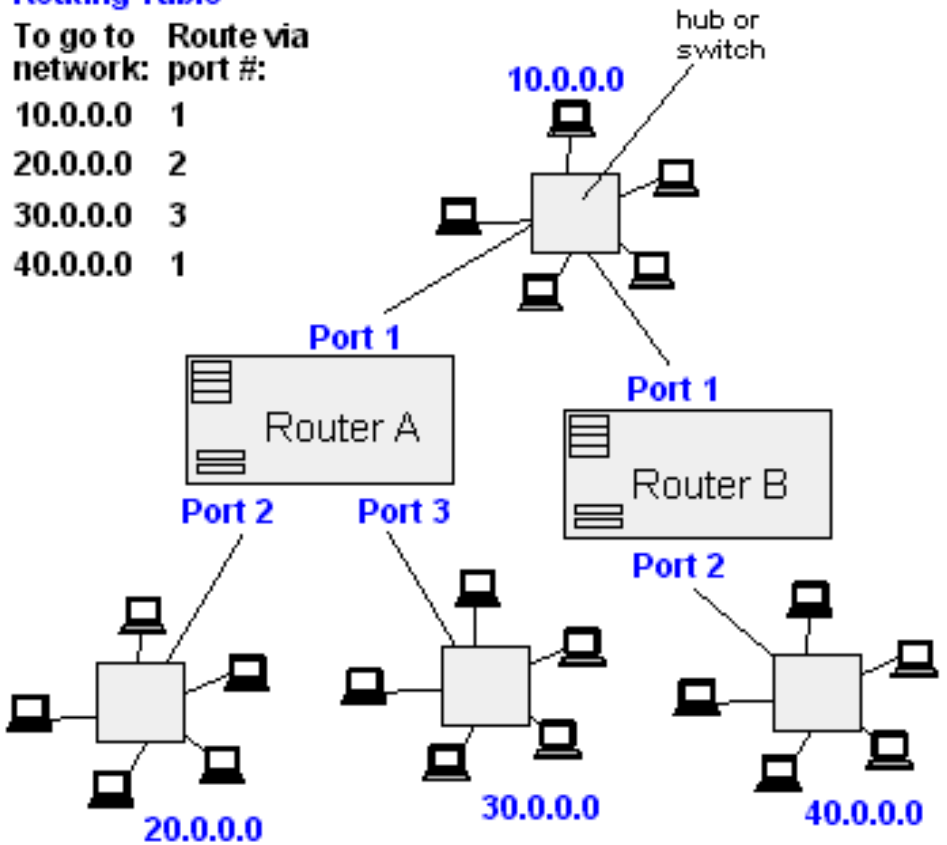
- Traffic Information Systems are most prominent use
- Mapping (Map Quest, Google Maps)
- Routing Systems



From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.

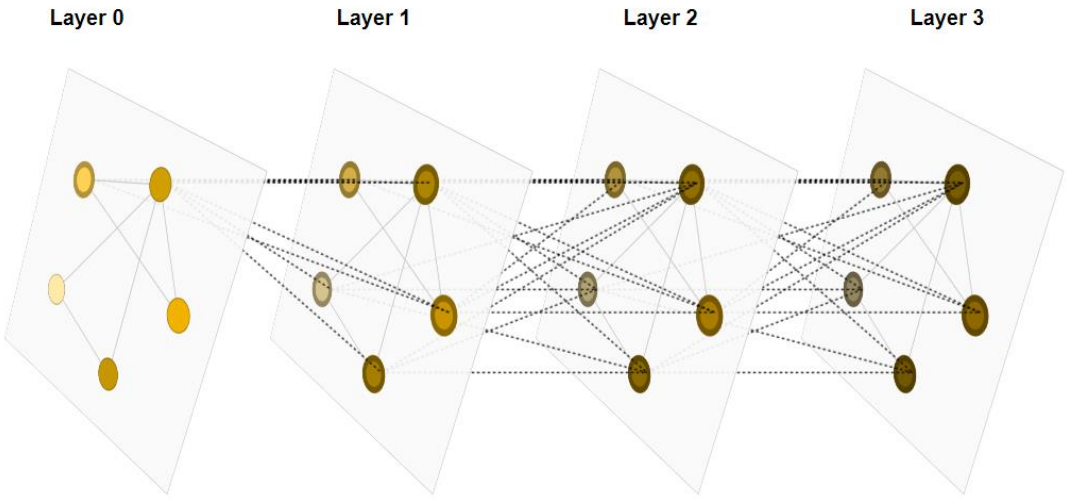
Router A
Routing Table

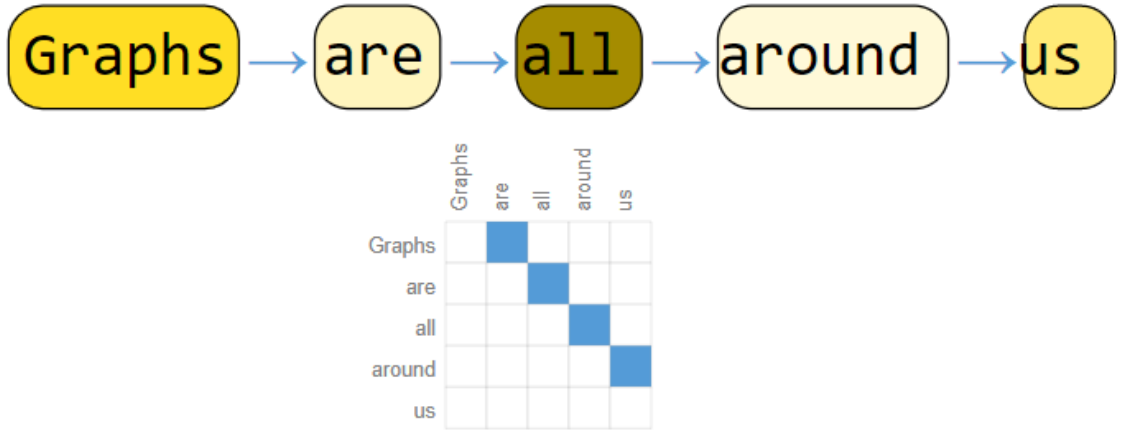
To go to network:	Route via port #:
10.0.0.0	1
20.0.0.0	2
30.0.0.0	3
40.0.0.0	1

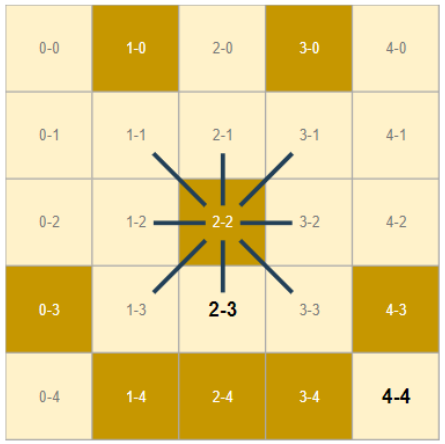


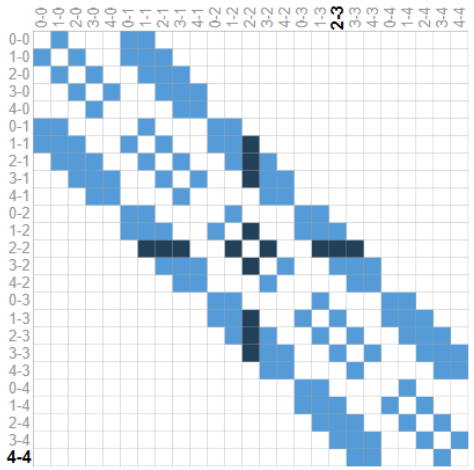
Source

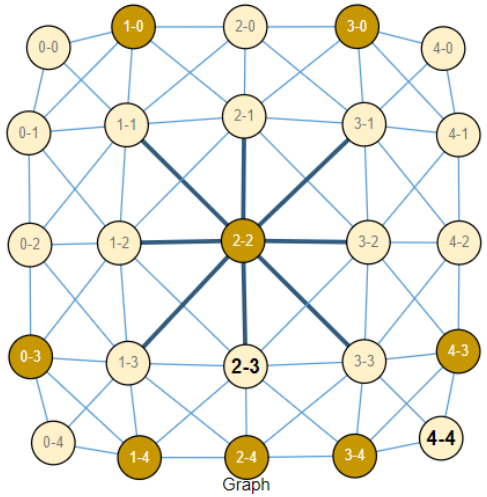
A Gentle Introduction to Graph Neural Networks











Source