AEL-3

Graph Traversal-1

Types of Graphs

1. **Undirected Graphs**: A graph in which edges have no direction, i.e., the edges do not have arrows indicating the direction of traversal. Example: A social network graph where friendships are not directional.
2. **Directed Graphs**: A graph in which edges have a direction, i.e., the edges have arrows indicating the direction of traversal. Example: A web page graph where links between pages are directional.
3. **Weighted Graphs:** A graph in which edges have weights or costs associated with them. Example: A road network graph where the weights can represent the distance between two cities.
4. **Unweighted Graph**s: A graph in which edges have no weights or costs associated with them. Example: A social network graph where the edges represent friendships.
5. **Complete Graphs:** A graph in which each vertex is connected to every other vertex. Example: A tournament graph where every player plays against every other player.
6. **Bipartite Graphs:** A graph in which the vertices can be divided into two disjoint sets such that every edge connects a vertex in one set to a vertex in the other set. Example: A job applicant graph where the vertices can be divided into job applicants and job openings.
7. **Trees**: A connected graph with no cycles. Example: A family tree where each person is connected to their parents.
8. **Cycles**: A graph with at least one cycle. Example: A bike-sharing graph where the cycles represent the routes that the bikes take.
9. **Sparse Graphs:** A graph with relatively few edges compared to the number of vertices. Example: A chemical reaction graph where each vertex represents a chemical compound and each edge represents a reaction between two compounds.
10. **Dense Graph**s: A graph with many edges compared to the number of vertices. Example: A social network graph where each vertex represents a person and each edge represents a friendship.

Advantages of graphs:

1. Graphs can be used to model and analyze complex systems and relationships.
2. They are useful for visualizing and understanding data.
3. Graph algorithms are widely used in computer science and other fields, such as social network analysis, logistics, and transportation.
4. Graphs can be used to represent a wide range of data types, including social networks, road networks, and the internet.

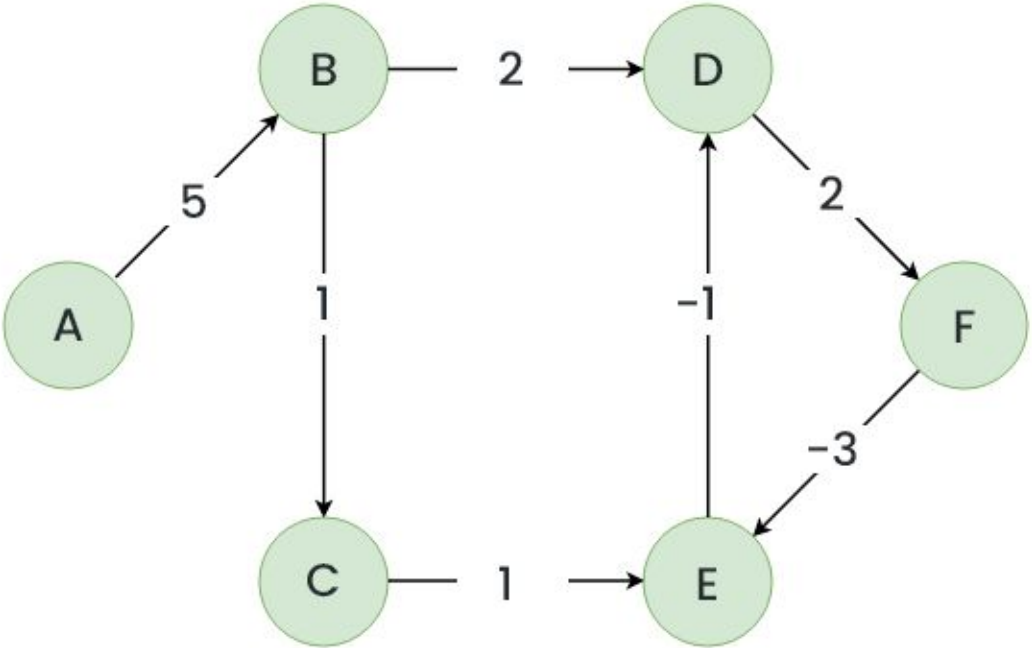
Disadvantages of graphs:

1. Large graphs can be difficult to visualize and analyze.
2. Graph algorithms can be computationally expensive, especially for large graphs.
3. The interpretation of graph results can be subjective and may require domain-specific knowledge.
4. Graphs can be susceptible to noise and outliers, which can impact the accuracy of analysis results.

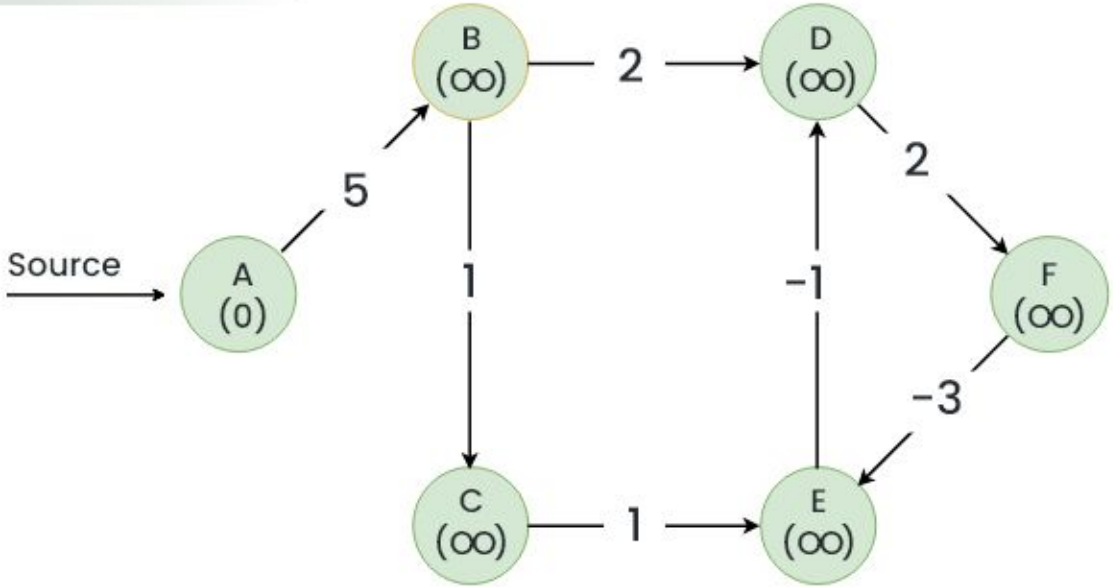
Difference between Graph and Tree

No.	Graph	Tree
1	It is a non-linear data structure.	It is also a non-linear data structure.
2	A graph is a set of vertices/ nodes and edges.	A tree is a set of nodes and edges.
3	In the graph, there is no unique node which is known as root.	In a tree, there is a unique node which is known as root.
4	Each node can have several edges.	Usually, a tree can have several child nodes, and in the case of binary trees, each node consists of two child nodes.
5	Graphs can form cycles.	Trees cannot form a cycle.

Bellman-Ford


B
2
D
5
2
A
1
-1
F
-3
C
1
E

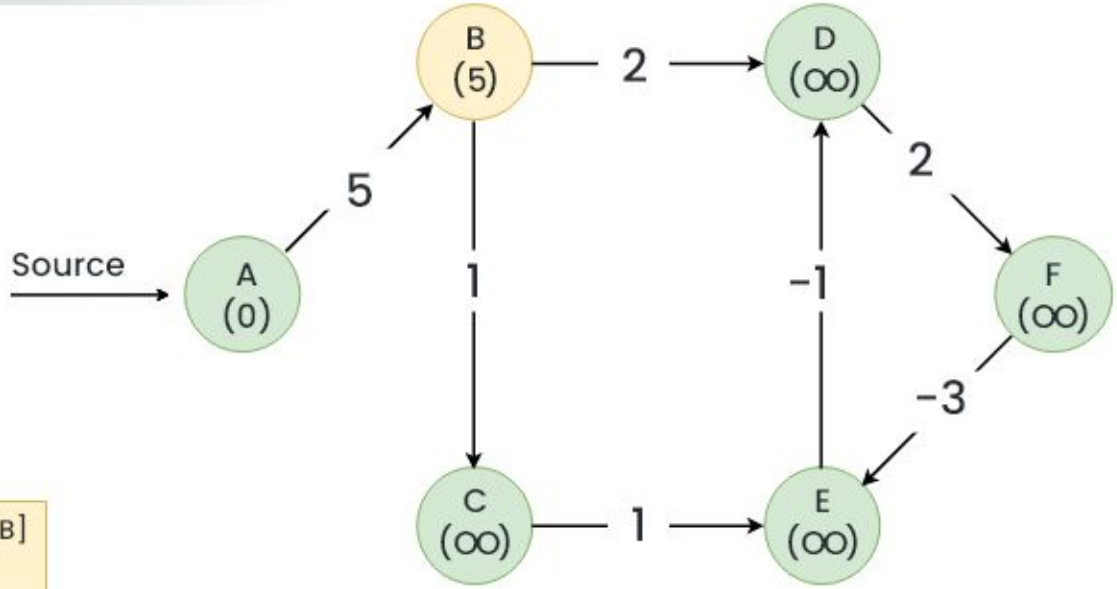
Initialize The Distance Array



Distance Array
Dist[ ]

A	B	C	D	E	F
0	∞	∞	∞	∞	∞

1st Relaxation Of Edges



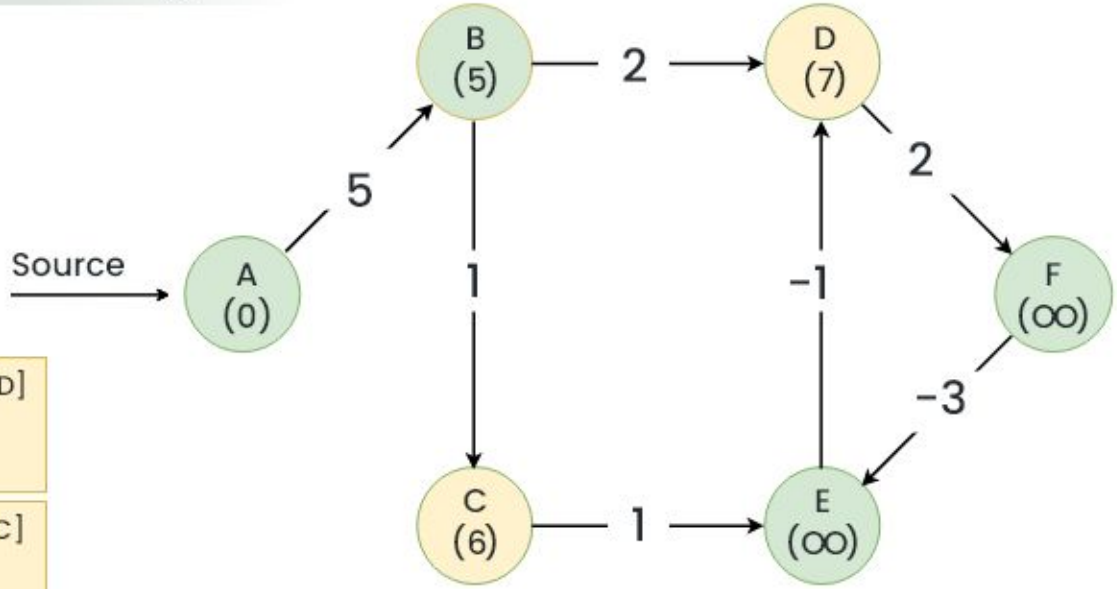
Dist [A] + 5 <Dist[B]
0+5<(∞)
Dist[B] = 5

Distance Array

A	B	C	D	E	F
0	∞	∞	∞	∞	∞

A	B	C	D	E	F
0	5	∞	∞	∞	∞

2nd Relaxation Of Edges



Dist [B] + 2 <Dist[D]
5+2<(∞)
Dist[D] = 7

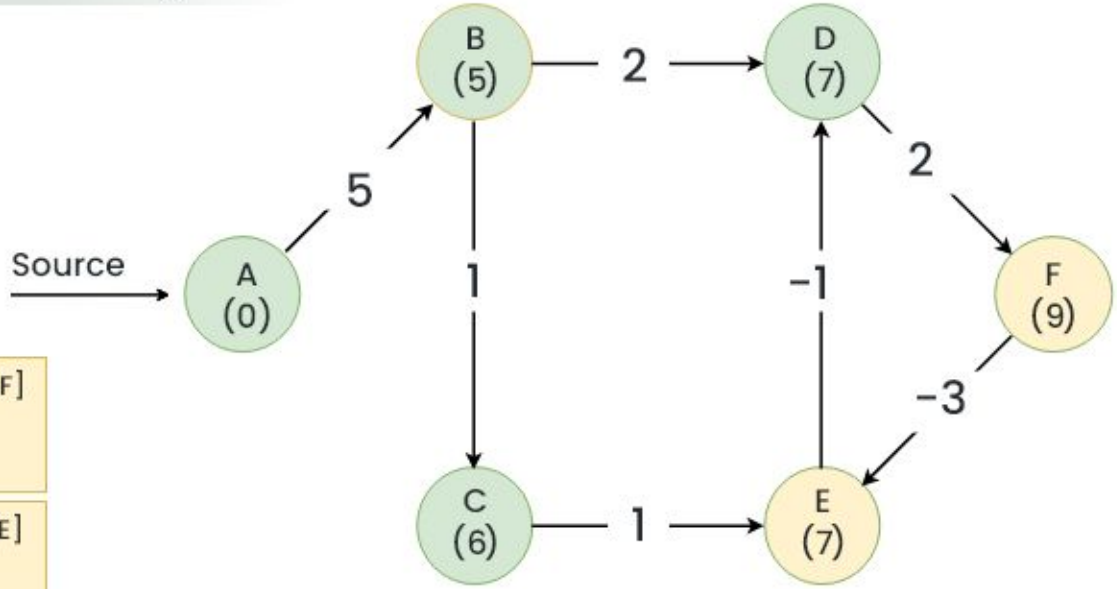
Dist [B] + 1 <Dist[C]
5+1 <(∞)
Dist[C] = 6

Distance Array

A	B	C	D	E	F
0	5	∞	∞	∞	∞

A	B	C	D	E	F
0	5	6	7	∞	∞

3rd Relaxation Of Edges



Dist [D] + 2 <Dist[F]
7+2<(∞)
Dist[F] = 9

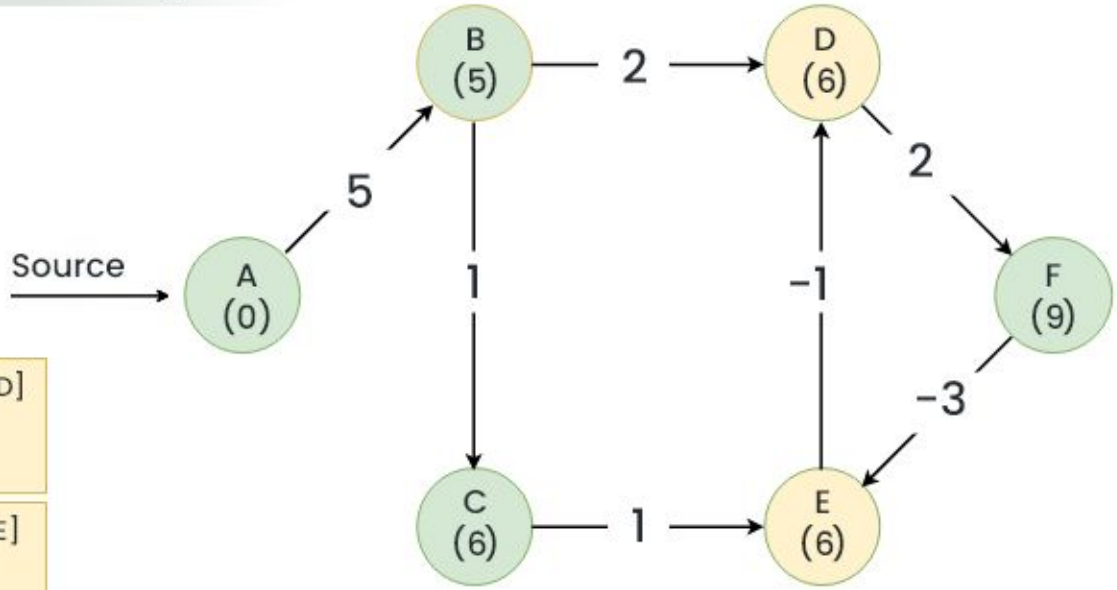
Dist [C] + 1 <Dist[E]
6+1 <(∞)
Dist[E] = 7

Distance Array

A	B	C	D	E	F
0	5	6	7	∞	∞

A	B	C	D	E	F
0	5	6	7	7	9

4th Relaxation Of Edges



Dist [E] + 2 <Dist[D]
7+(-1)< 7
Dist[D] = 6

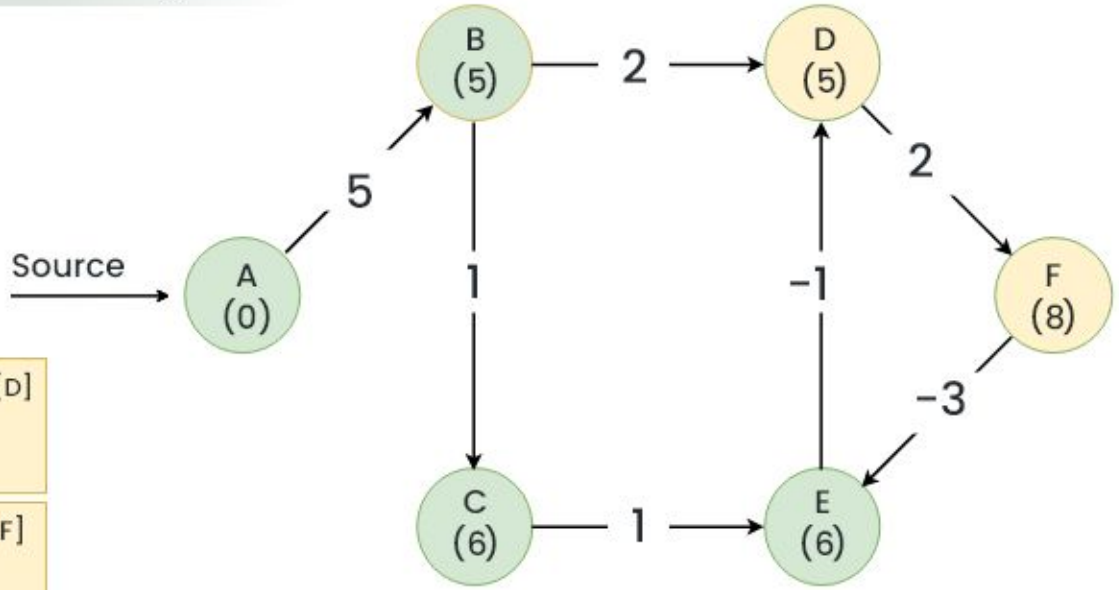
Dist [F] + 1 <Dist[E]
9+(-3) < 6
Dist[E] = 6

Distance Array

A	B	C	D	E	F
0	5	6	7	7	9

A	B	C	D	E	F
0	5	6	6	6	9

5th Relaxation Of Edges



Dist [E] + (-1) < Dist[D]
6+(-1)< 6
Dist[D] = 5

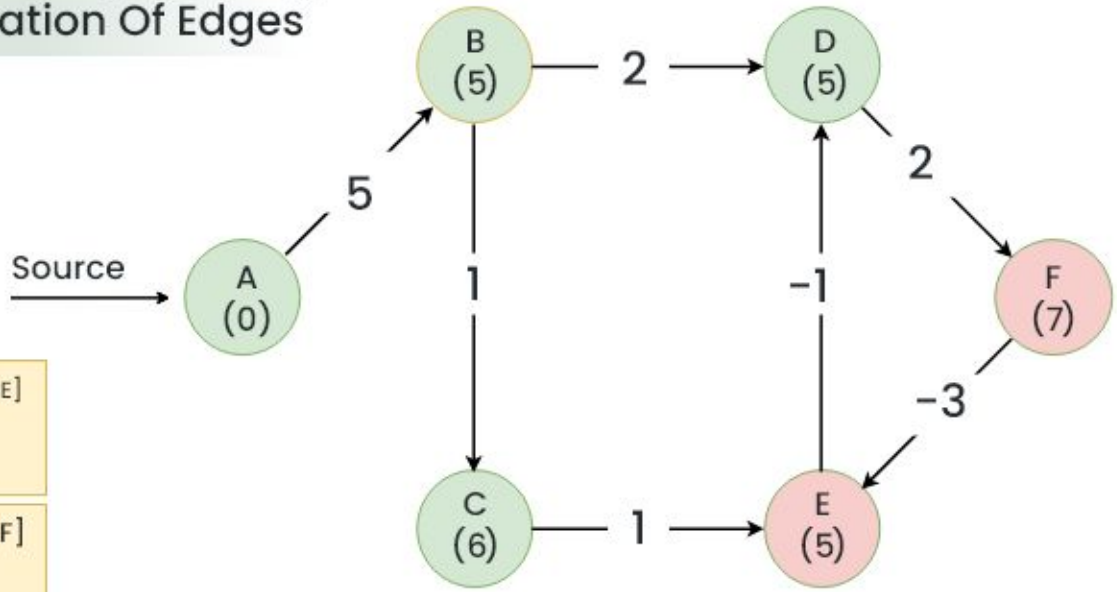
Dist [D] + 2 < Dist[F]
6+2< 9
Dist[F] = 8

Distance Array

A	B	C	D	E	F
0	5	6	6	6	9

A	B	C	D	E	F
0	5	6	5	6	8

Detecting The Negative Edge By 6Th Relaxation Of Edges



Dist [F] + (-3) <Dist[E]
8+(-3)<6
Dist[E] = 5

Dist [D] + 2 <Dist[F]
6+2< 8
Dist[F] = 7

Distance Array

A	B	C	D	E	F
0	5	6	5	6	8

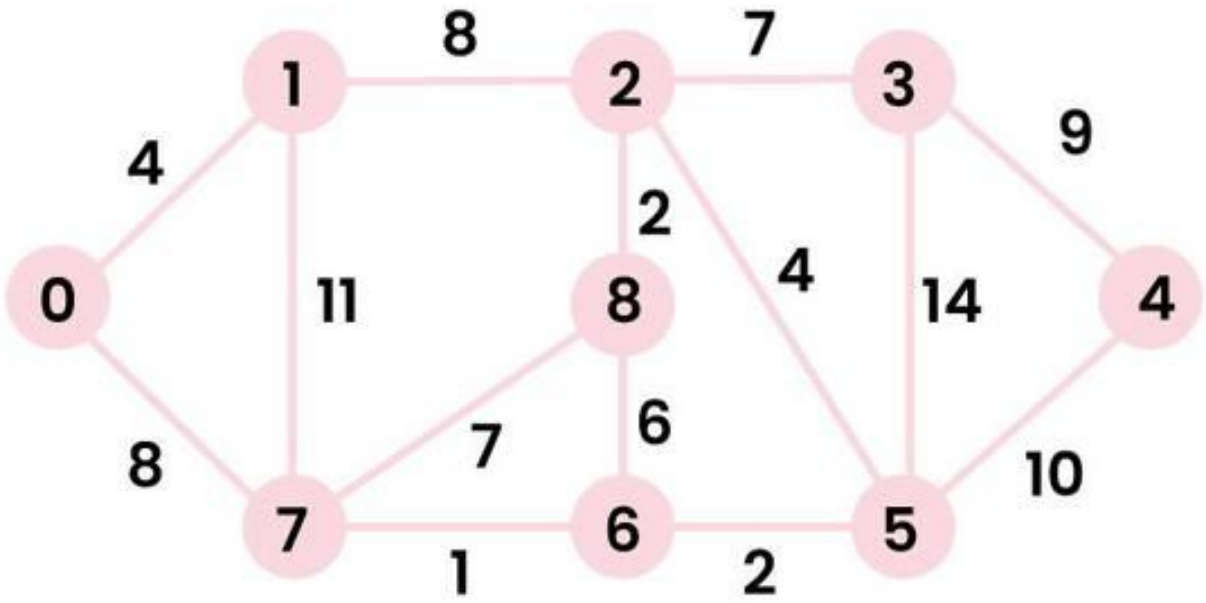
A	B	C	D	E	F
0	5	6	4	4	6

Why Relaxing Edges N-1 times, gives us Single Source Shortest Path?

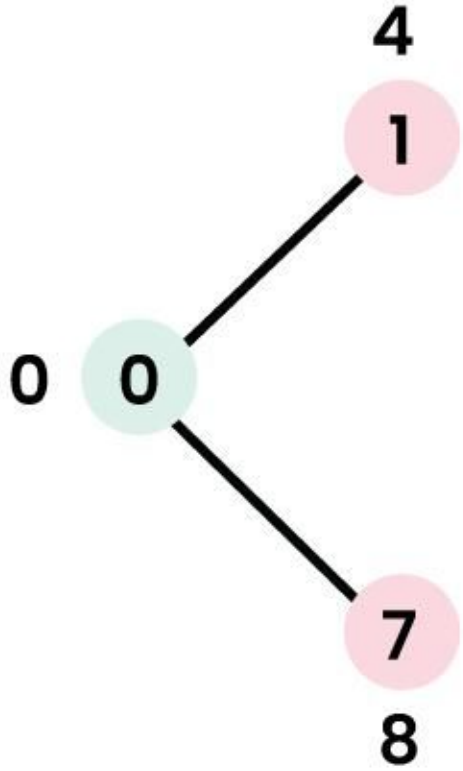
In the worst-case scenario, a shortest path between two vertices can have at most **N-1** edges, where **N** is the number of vertices. This is because a simple path in a graph with **N** vertices can have at most **N-1** edges, as it's impossible to form a closed loop without revisiting a vertex.

By relaxing edges **N-1** times, the Bellman-Ford algorithm ensures that the distance estimates for all vertices have been updated to their optimal values, assuming the graph doesn't contain any negative-weight cycles reachable from the source vertex. If a graph contains a negative-weight cycle reachable from the source vertex, the algorithm can detect it after **N-1** iterations, since the negative cycle disrupts the shortest path lengths.

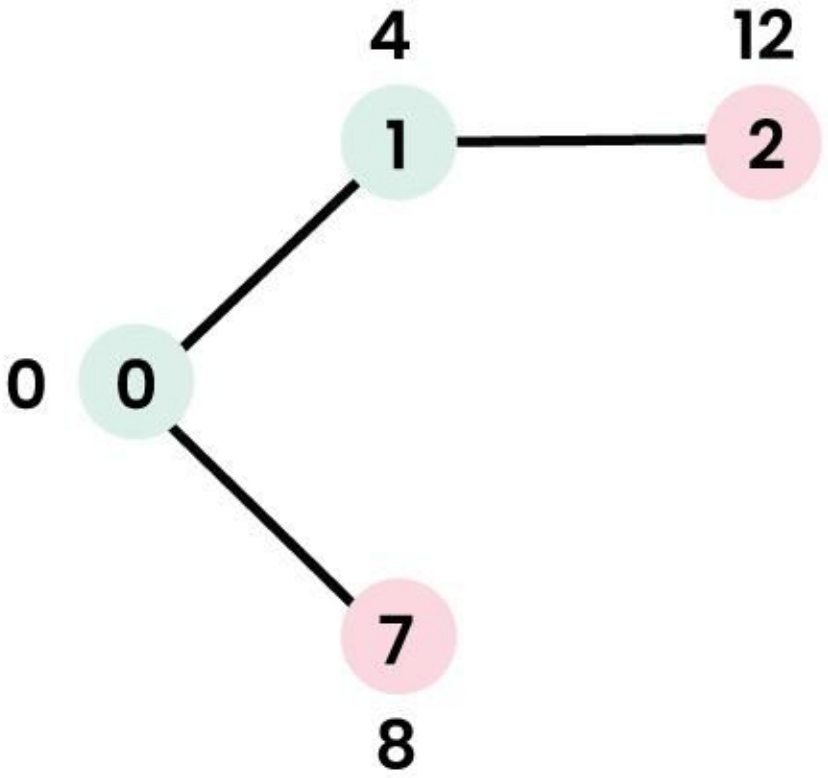
Dijkstra



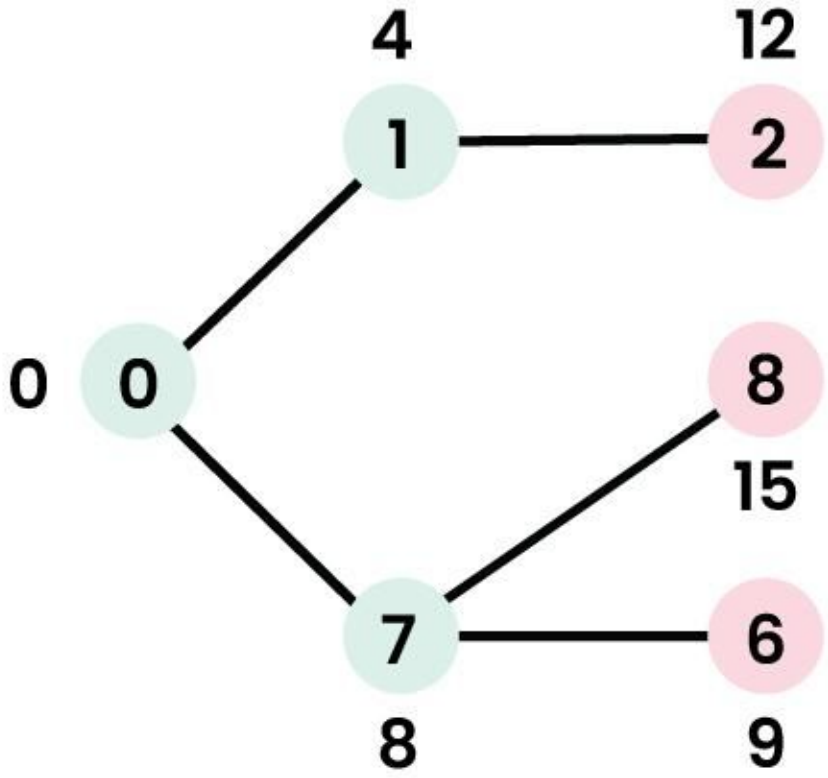
Working of Dijkstra's Algorithm



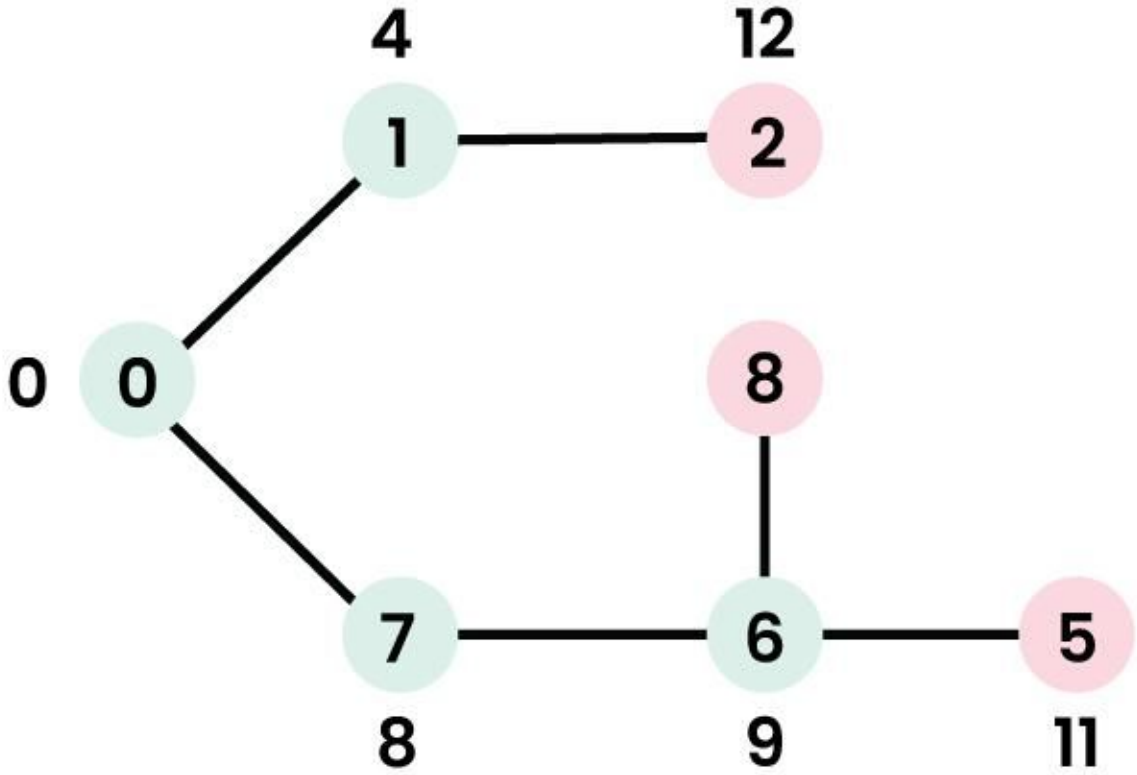
Working of Dijkstra's Algorithm



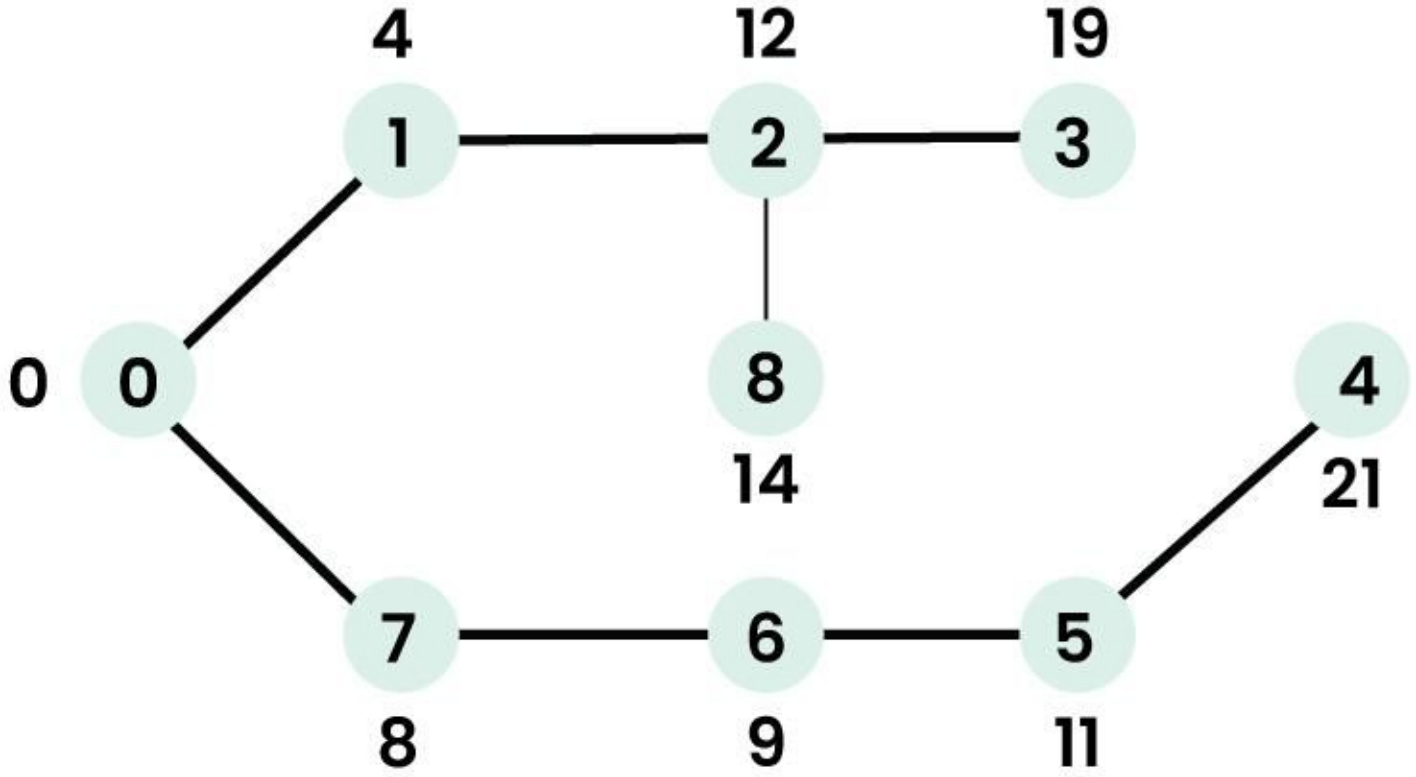
Working of Dijkstra's Algorithm



Working of Dijkstra's Algorithm



Working of Dijkstra's Algorithm



Working of Dijkstra's Algorithm

Why does Dijkstra's Algorithm fail on negative weights?

So no one can handle negative cycle but bellman-ford or floyd warshall can handle negative weight unlike dijkstra.