MET's Institute of Engineering

Bhujbal Knowledge City, Adgaon, Nashik.

Department of Computer Engineering

"GRAPH"

Prepared By

Prof. Anand N. Gharu

(Assistant Professor)

Computer Dept.

CLASS : SE COMPUTER 2019

15 February 2024

SUBJECT : DSA (SEM-II)

UNIT : III

Note: The material to prepare this presentation has been taken from internet and are generated only for students reference and not for commercial use.

SYLLABUS

Savitribai Phule Pune University Second Year of Engineering (2019 Course) 210252: Data Structures and Algorithms		
Teaching Scheme	**Credit Scheme**	**Examination Scheme and Marks**
Lecture: 03 Hours/Week	03	**Mid_Semester(TH):** 30 Marks **End_Semester(TH):** 70 Marks
Prerequisite Courses:	110005: Programming and Problem Solving 210242: Fundamentals of Data Structures	
Companion Course:	210257: Data Structures and Algorithms Laboratory	

SYLLABUS

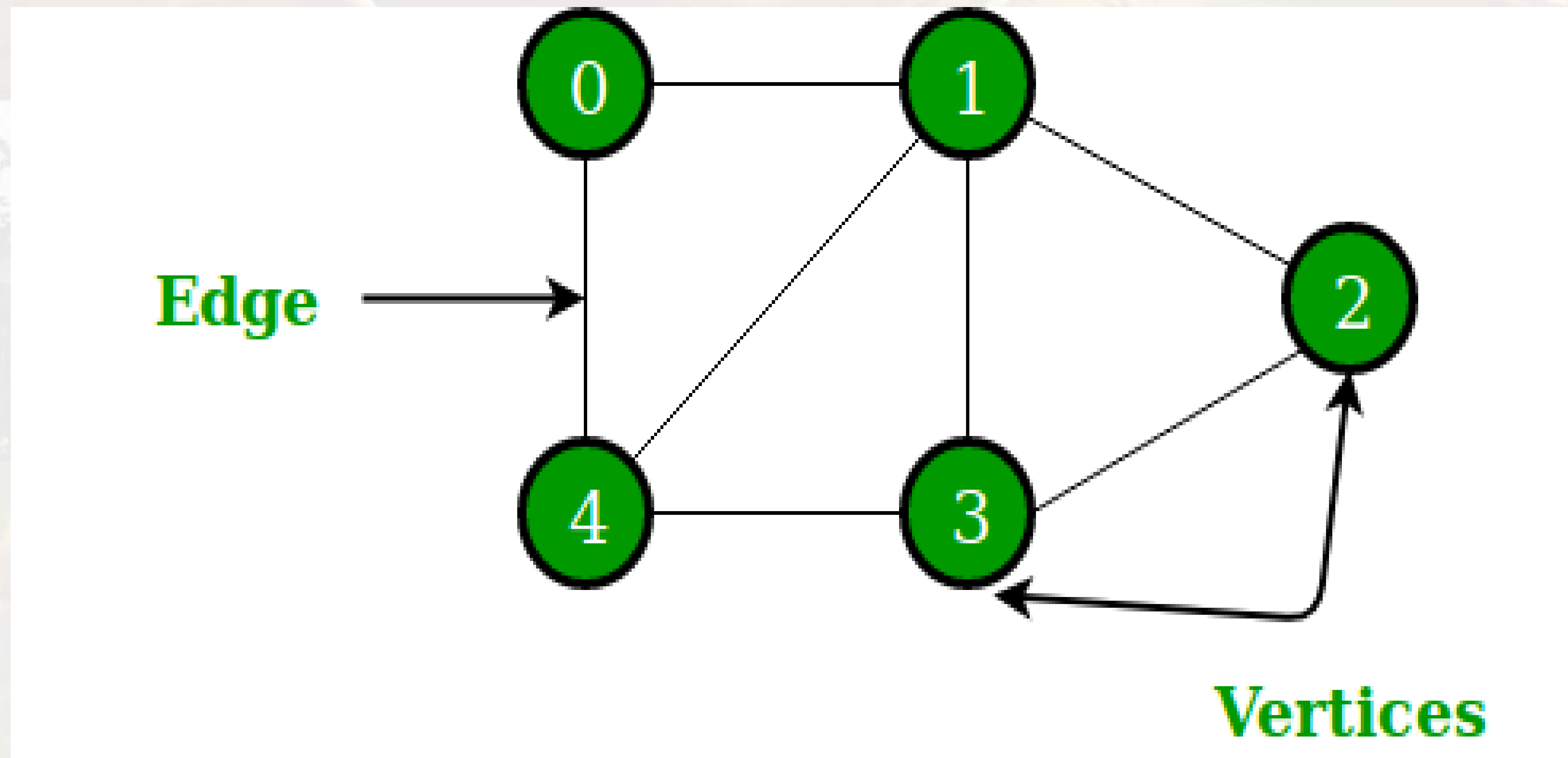
Basic Concepts, Storage representation, Adjacency matrix, adjacency list, adjacency multi list, inverse adjacency list. **Traversals** - depth first and breadth first, Minimum spanning Tree, Greedy algorithms for computing minimum spanning tree- Prims and Kruskal Algorithms, Dikjtra's Single source shortest path, All pairs shortest paths- Flyod-Warshall Algorithm Topological ordering.

UNIT-III

GRAPH

Introduction of Graph

A graph is a non-linear data structure, which consists of vertices(or nodes) connected by edges(or arcs) where edges may be directed or undirected.



Introduction of Graph

A graph G can be defined as an ordered set G(V, E) where V(G) represents the set of vertices and E(G) represents the set of edges which are used to connect these vertices.

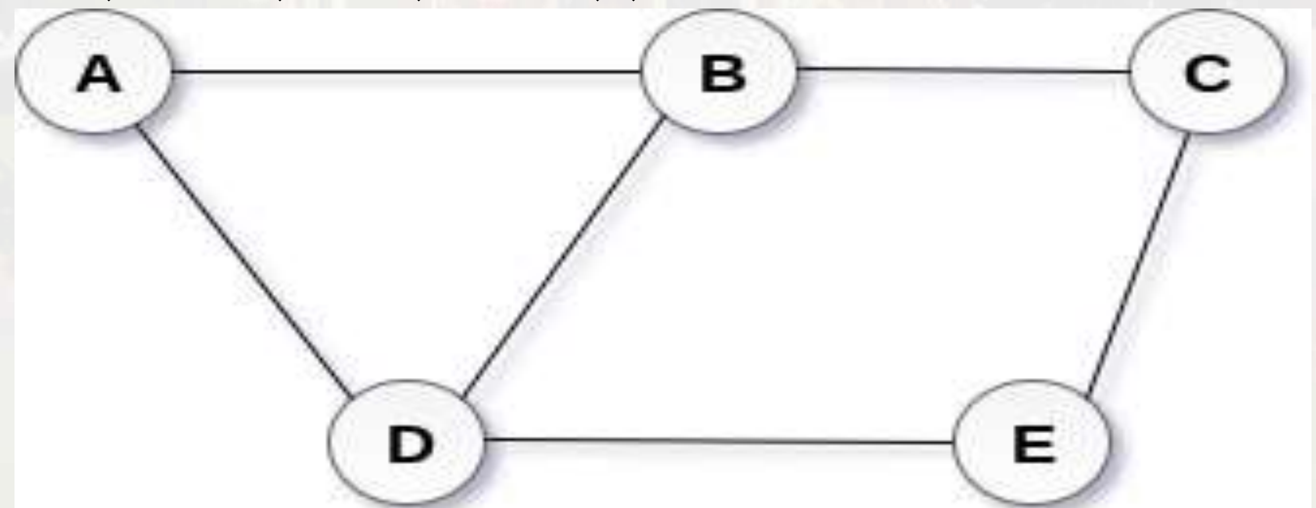
Types of Graph :

1. Directed Graph
2. Undirected Graph
3. Weighted Graph

Undirecred Graph

A graph G can be defined as an ordered set G(V, E) where V(G) represents the set of vertices and E(G) represents the set of edges which are used to connect these vertices.

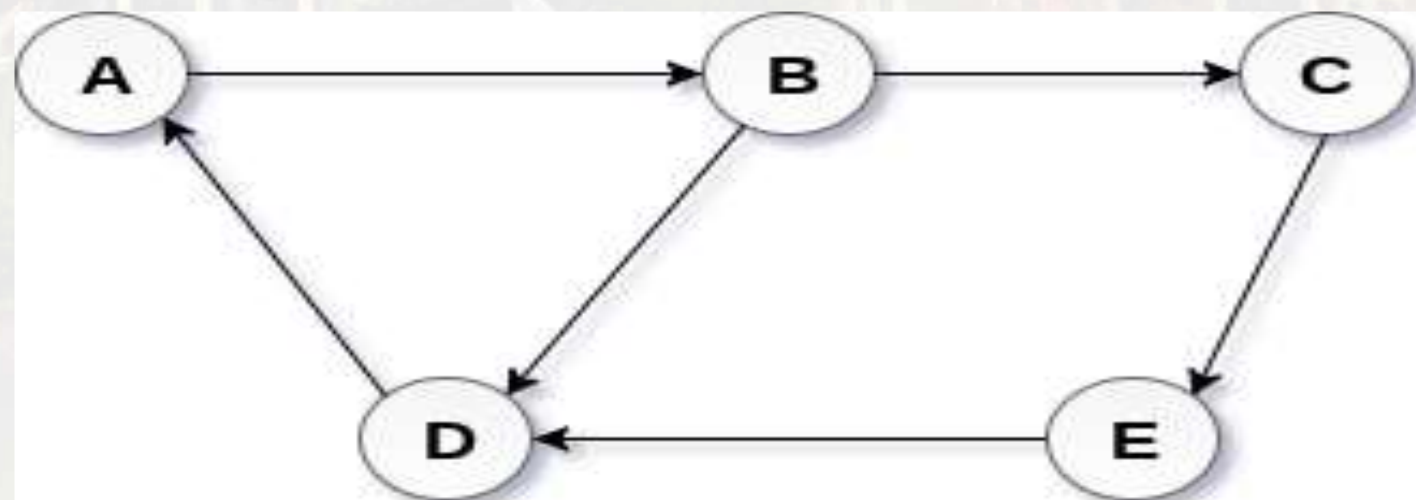
A Graph G(V, E) with 5 vertices (A, B, C, D, E) and six edges ((A,B), (B,C), (C,E), (E,D), (D,B), (D,A)) is shown in the following figure.



Undirected Graph

Directed Graph

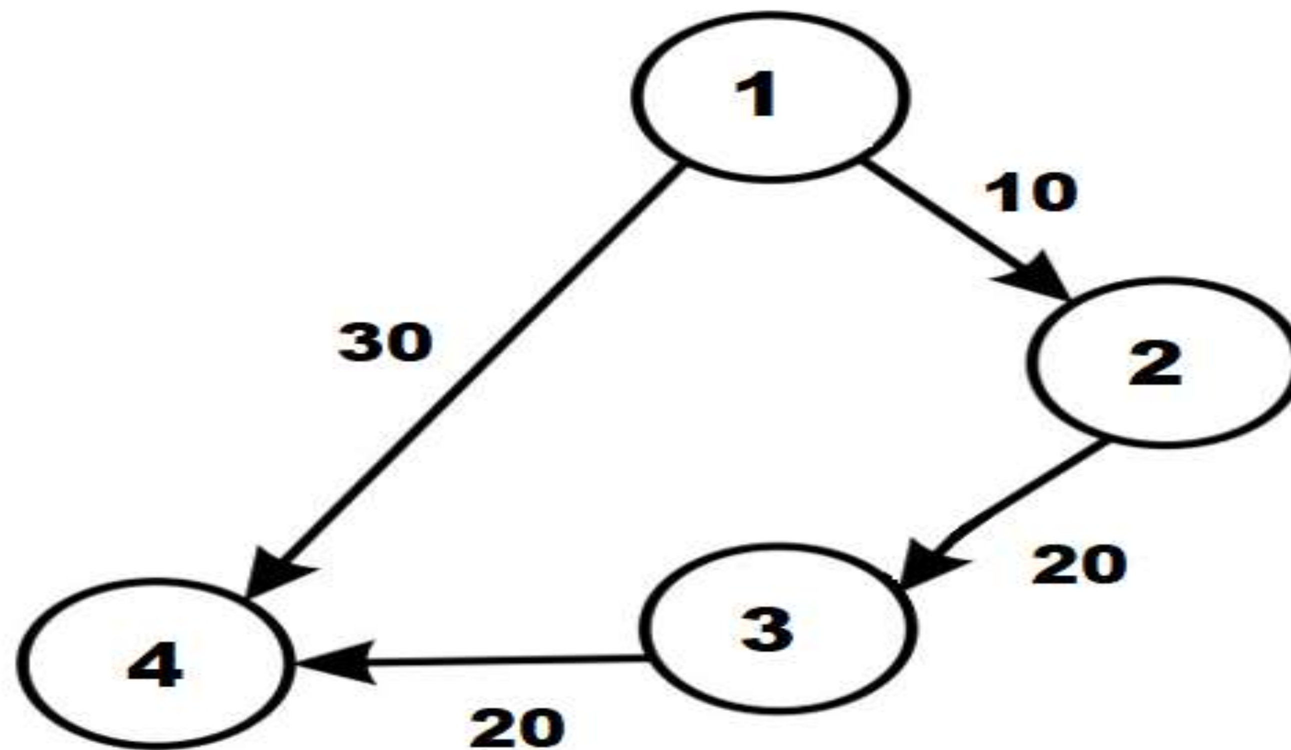
A graph can be directed or undirected. However, in an undirected graph, edges are not associated with the directions with them. An undirected graph is shown in the above figure since its edges are not attached with any of the directions. If an edge exists between vertex A and B then the vertices can be traversed from B to A as well as A to B.



Directed Graph

Wighted Graph

A graph that has a value associated with every edge. The values corresponding to the edges are called weights. A value in a weighted graph can represent quantities such as cost, distance, and time, depending on the graph. Weighted graphs are typically used in modeling computer networks.



Weighted Graph

Terminologies of Graph

Path

A path can be defined as the sequence of nodes that are followed in order to reach some terminal node V from the initial node U.

Closed Path

A path will be called as closed path if the initial node is same as terminal node. A path will be closed path if Vo=VN.

Simple Path

If all the nodes of the graph are distinct with an exception Vo=VN, then such path P is called as closed simple path.

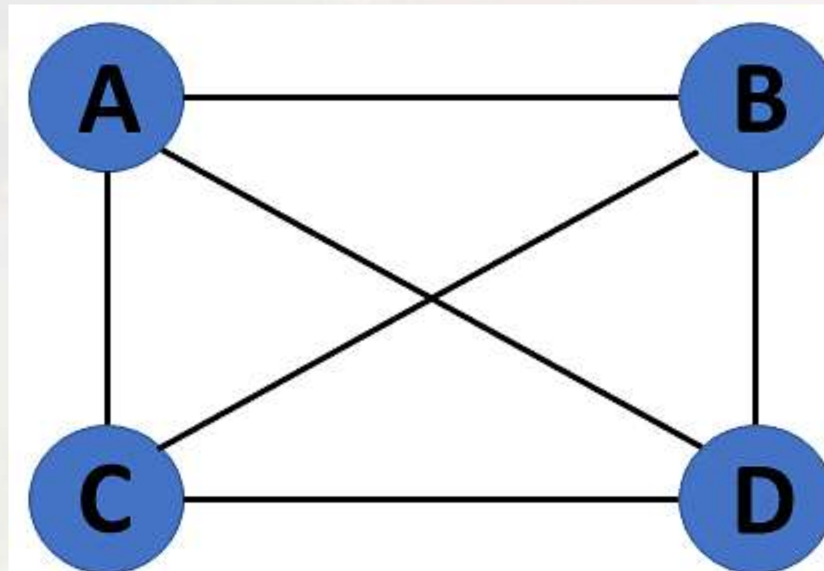
Terminologies of Graph

Cycle

A cycle can be defined as the path which has no repeated edges or vertices except the first and last vertices.

Connected Graph

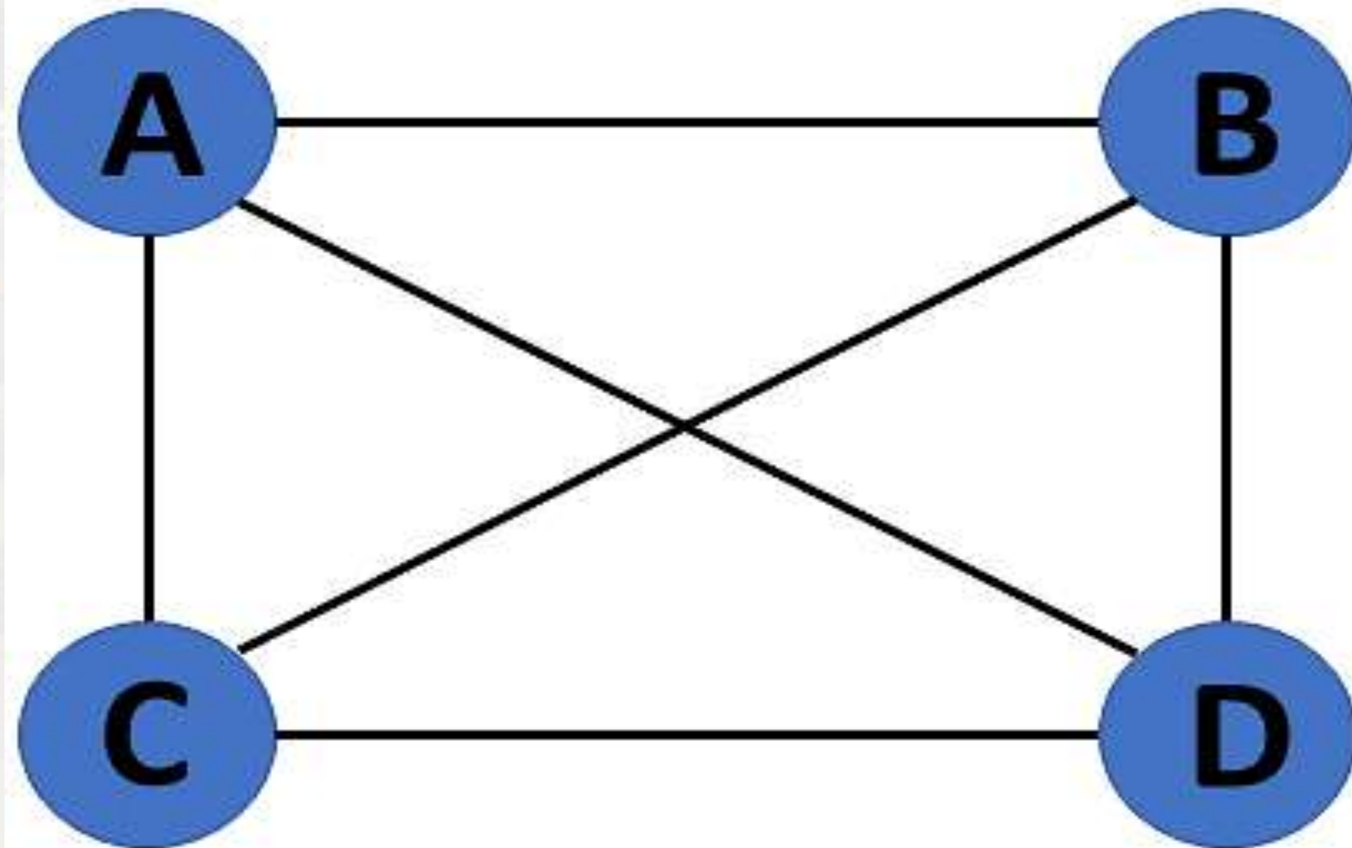
A connected graph is the one in which some path exists between every two vertices (u, v) in V. There are no isolated nodes in connected graph.



Terminologies of Graph

Complete Graph

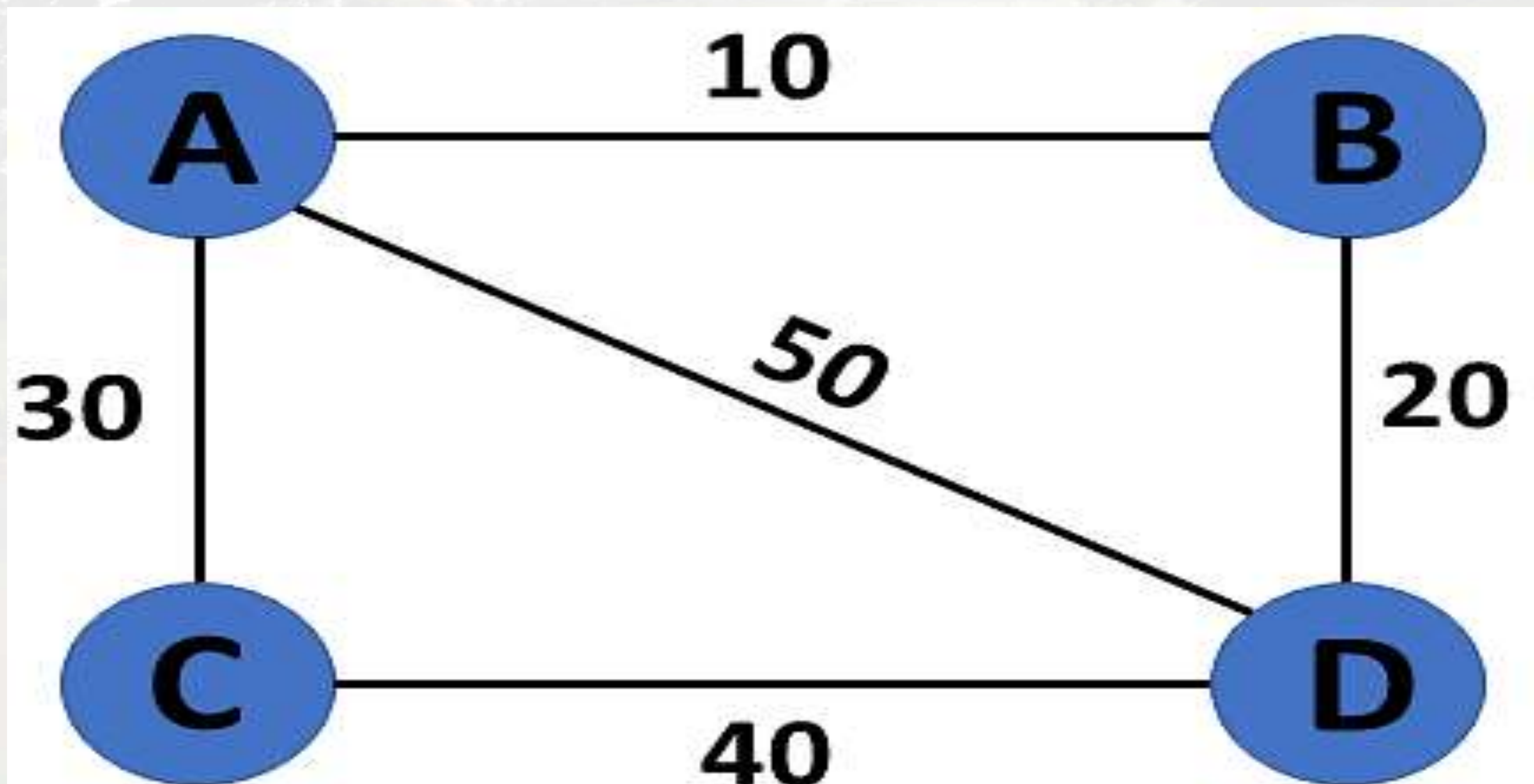
A complete graph is the one in which every node is connected with all other nodes. A complete graph contain n(n-1)/2 edges where n is the number of nodes in the graph.



Terminologies of Graph

Weighted Graph

In a weighted graph, each edge is assigned with some data such as length or weight. The weight of an edge e can be given as w(e) which must be a positive (+) value indicating the cost of traversing the edge.



Terminologies of Graph

Weighted Graph

In a weighted graph, each edge is assigned with some data such as length or weight. The weight of an edge e can be given as w(e) which must be a positive (+) value indicating the cost of traversing the edge.

Digraph

A digraph is a directed graph in which each edge of the graph is associated with some direction and the traversing can be done only in the specified direction.

Terminologies of Graph

Loop

An edge that is associated with the similar end points can be called as Loop.

Adjacent Nodes

If two nodes u and v are connected via an edge e, then the nodes u and v are called as neighbours or adjacent nodes.

Degree of the Node

A degree of a node is the number of edges that are connected with that node. A node with degree 0 is called as isolated node.

Terminologies of Graph

Loop

An edge that is associated with the similar end points can be called as Loop.

Adjacent Nodes

If two nodes u and v are connected via an edge e, then the nodes u and v are called as neighbours or adjacent nodes.

Degree of the Node

A degree of a node is the number of edges that are connected with that node. A node with degree 0 is called as isolated node.

Terminologies of Graph

- **Outgoing edges of a vertex** are directed edges that the vertex is the origin.
- **Incoming edges of a vertex** are directed edges that the vertex is the destination.
- **The degree of a vertex** in a graph is the total number of edges incident to it.
- In a directed graph, **the out-degree of a vertex** is the total number of outgoing edges
- the **in-degree** is the total number of incoming edges.

Terminologies of Graph

Pendant Vertex

A vertex with degree one is called a pendant vertex.

Isolated Vertex

A vertex with degree zero is called an isolated vertex.

Terminologies of Graph

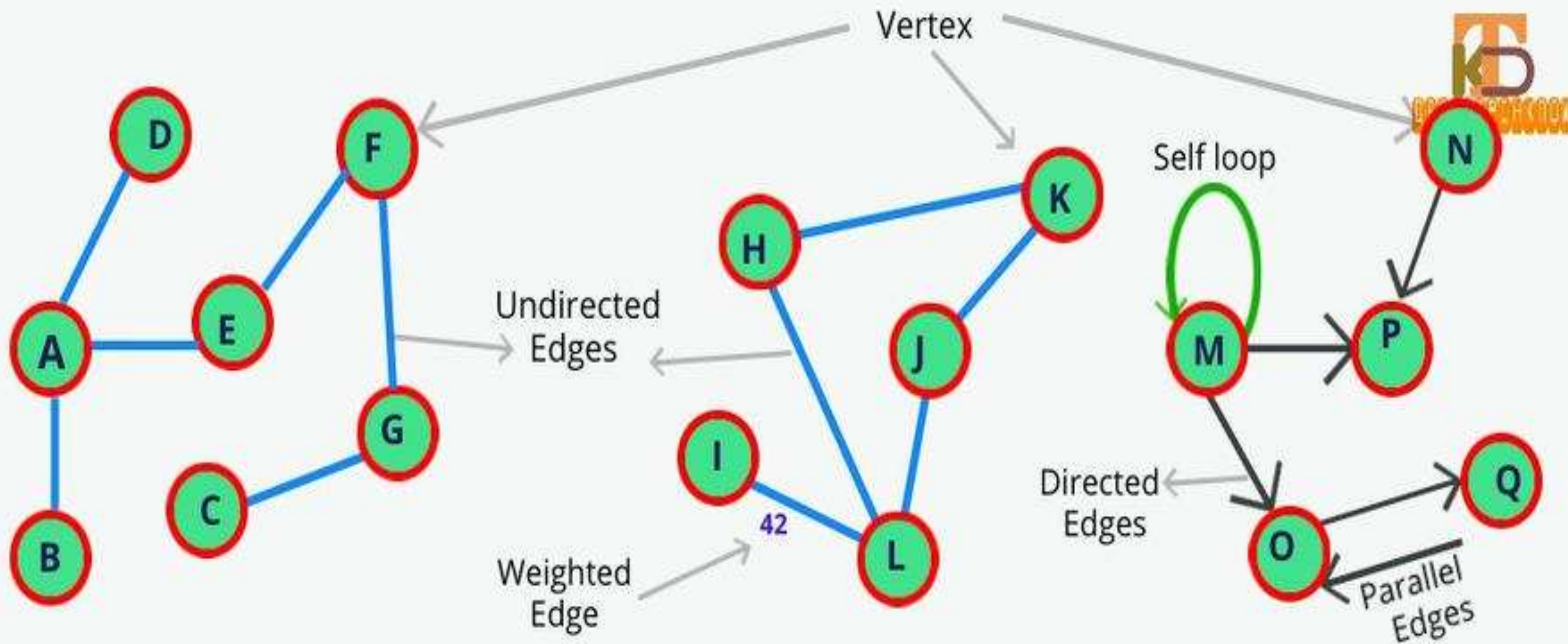


Figure 1.1

Graph Abstract Data Type

1. *create() :Graph*
2. *insert vertex(Graph, v) :Graph*
3. *delete vertex(Graph, v) :Graph*
4. *insert edge(Graph, u, v) :Graph*
5. *delete edge(Graph, u, v) :Graph*
6. *is empty(Graph) :Boolean;*
7. *end graph*

Graph Representation

Adjacency Matrix Representation of Graph

We can easily represent the graphs using the following ways,

1. Adjacency matrix (sequential representation)
2. Adjacency list (linked representation)
3. Adjacency Multilist
4. Inverse Adjacency List

Graph Representation

- By Graph representation, we simply mean the technique which is to be used in order to store some graph into the computer's memory.

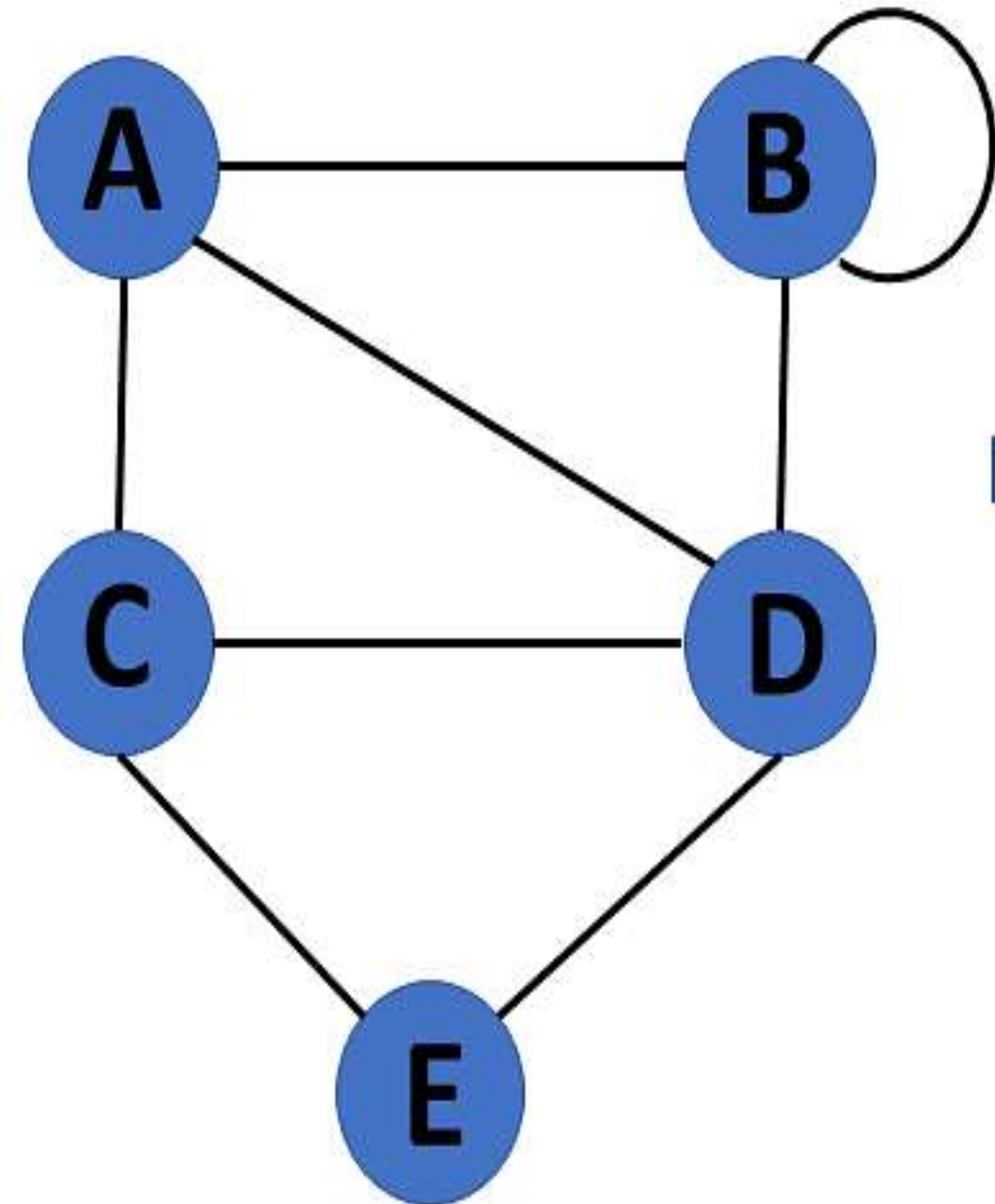
- There are two ways to store Graph into the computer's memory. In this part of this tutorial, we discuss each one of them in detail.

Adjacency Matrix

1. A sequential representation is an adjacency matrix.
2. It's used to show which nodes are next to one another. I.e., is there any connection between nodes in a graph?
3. You create an MXM matrix G for this representation. If an edge exists between vertex a and vertex b, the corresponding element of G, $g_{i,j} = 1$, otherwise $g_{i,j} = 0$.
4. If there is a weighted graph, you can record the edge's weight instead of 1s and 0s.

Adjacency Matrix

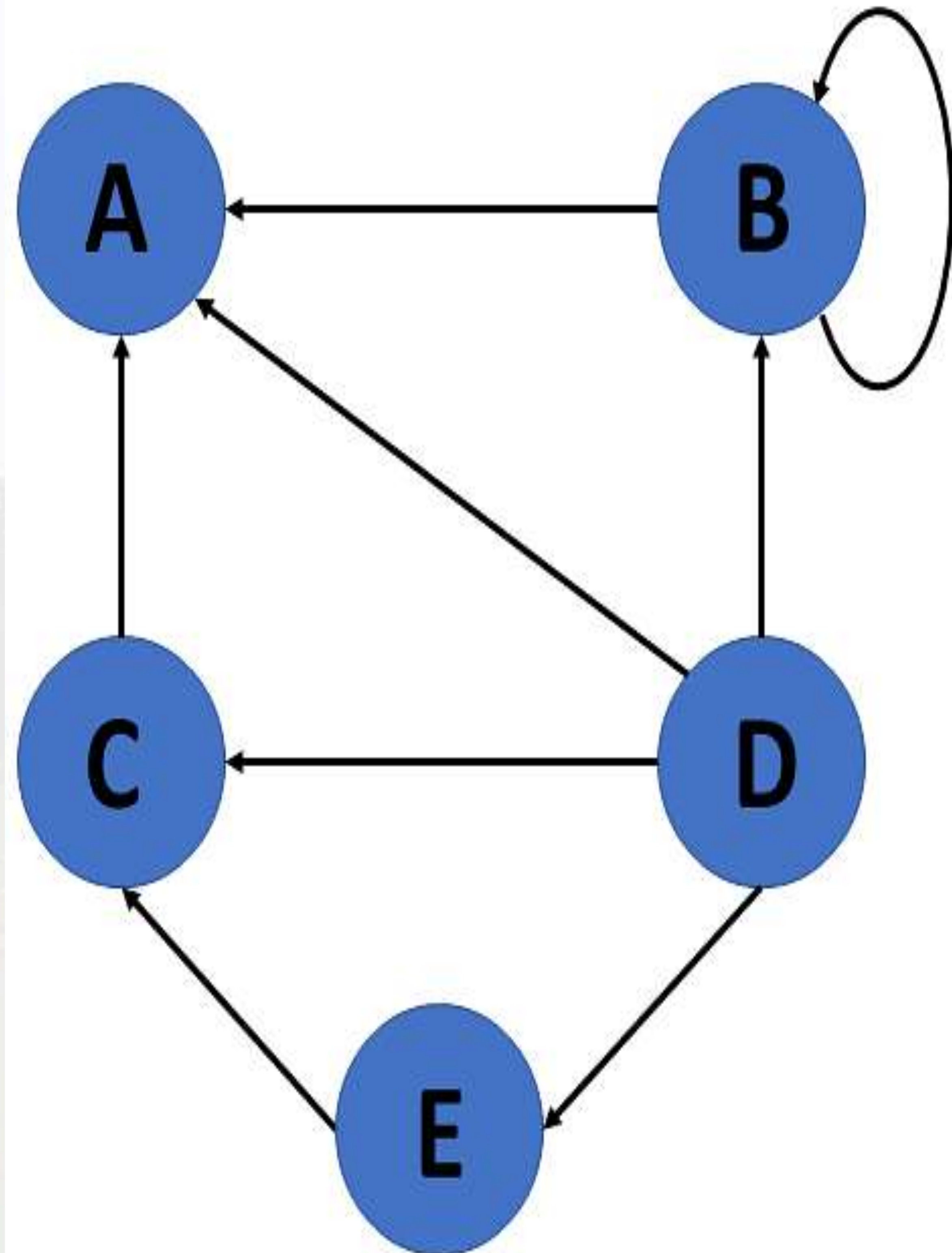
Undirected Graph Representation :



	A	B	C	D	E
A	0	1	1	1	0
B	1	1	0	1	0
C	1	0	0	1	1
D	1	1	1	0	1
E	0	0	1	1	0

Adjacency Matrix

Directed Graph Representation :

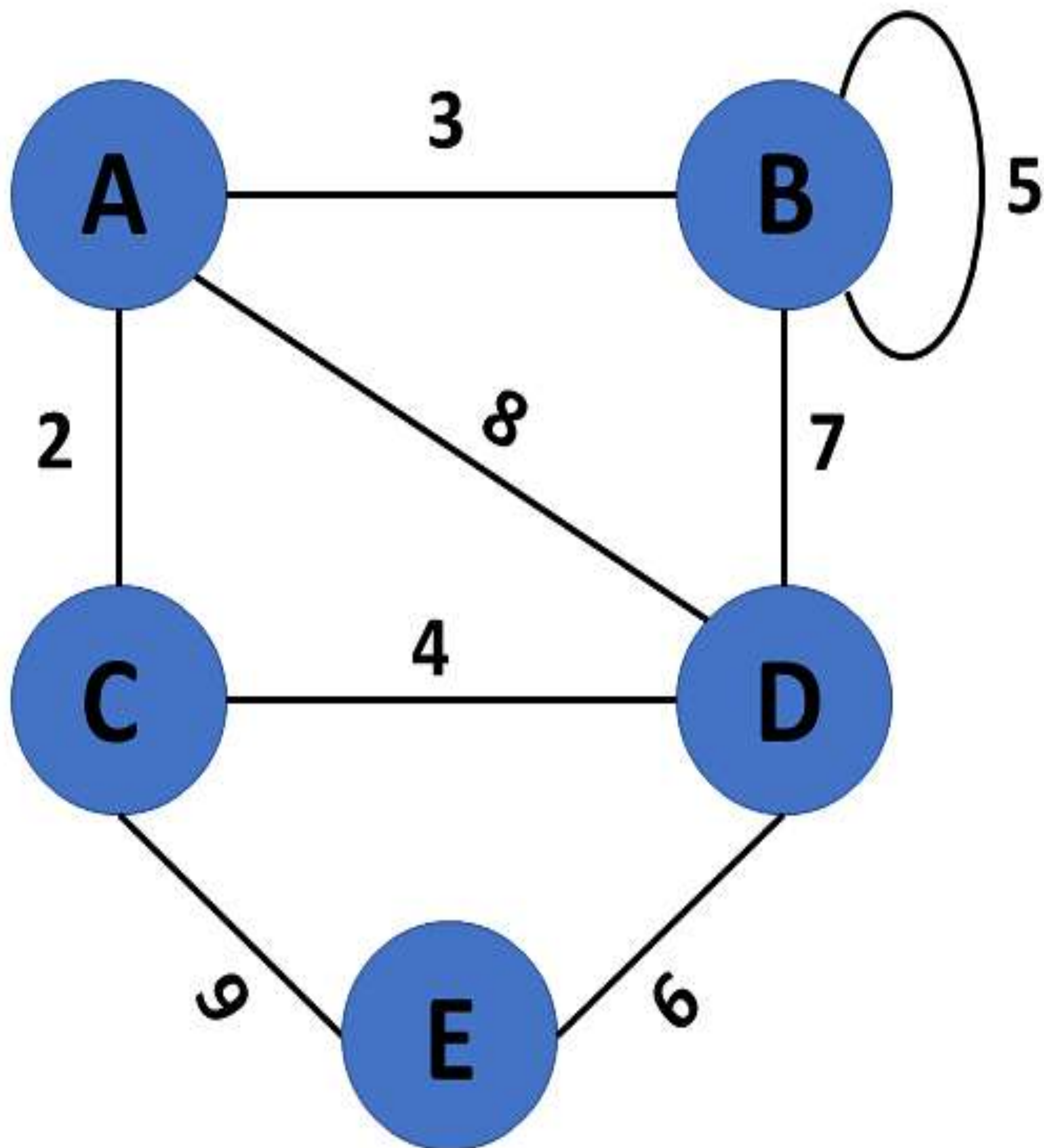


	A	B	C	D	E
A	0	0	0	0	0
B	1	1	0	0	0
C	1	0	0	0	0
D	1	1	1	0	1
E	0	0	1	0	0

Adjacency Matrix

Weighted Undirected Graph Representation

Weight or cost is indicated at the graph's edge, a weighted graph representing these values in the matrix



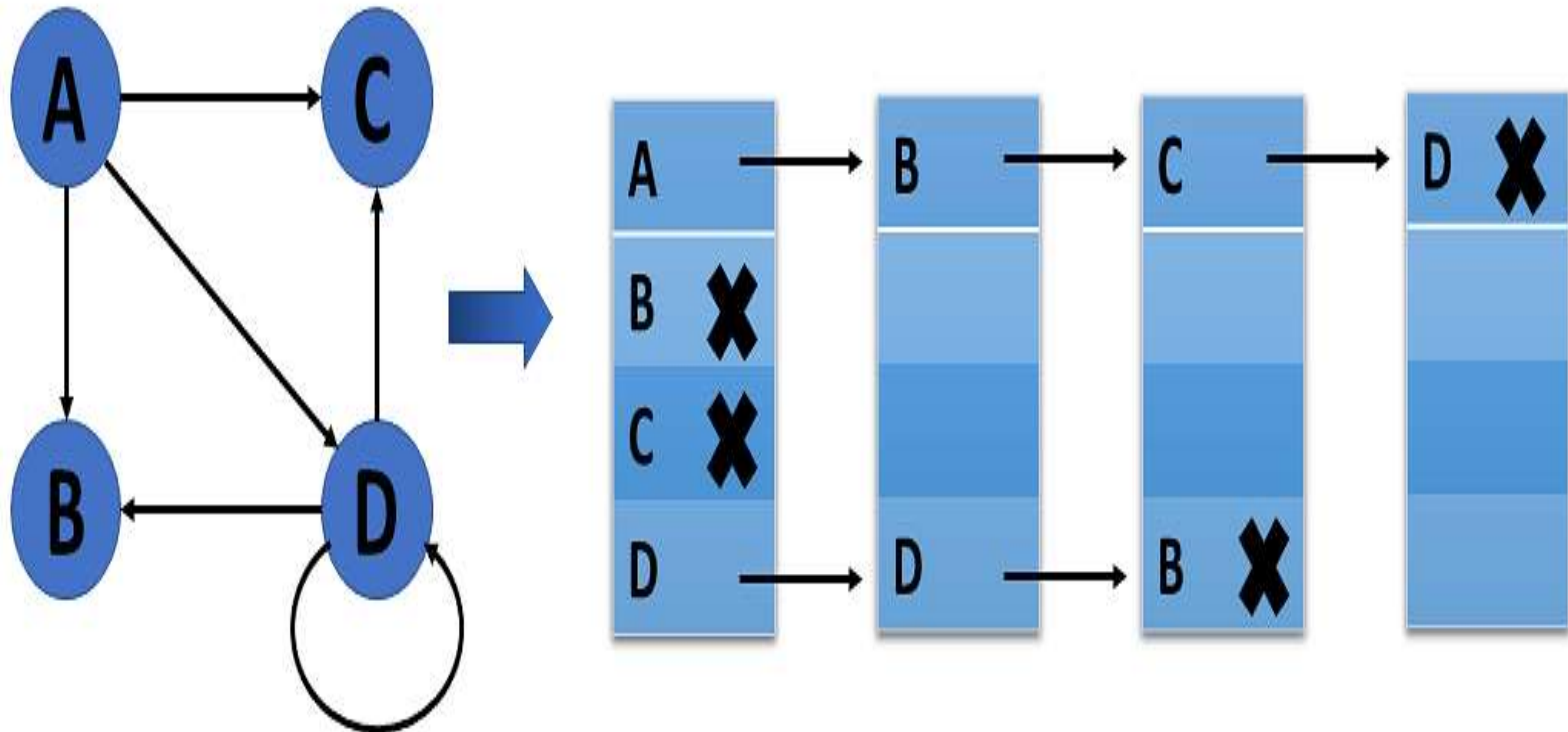
	A	B	C	D	E
A	0	3	2	8	0
B	3	5	0	7	0
C	2	0	0	4	9
D	8	7	4	0	6
E	0	0	9	6	0

Adjacency List

1. A linked representation is an adjacency list.
2. You keep a list of neighbors for each vertex in the graph in this representation. It means that each vertex in the graph has a list of its neighboring vertices.
3. You have an array of vertices indexed by the vertex number, and the corresponding array member for each vertex x points to a singly linked list of x's neighbors

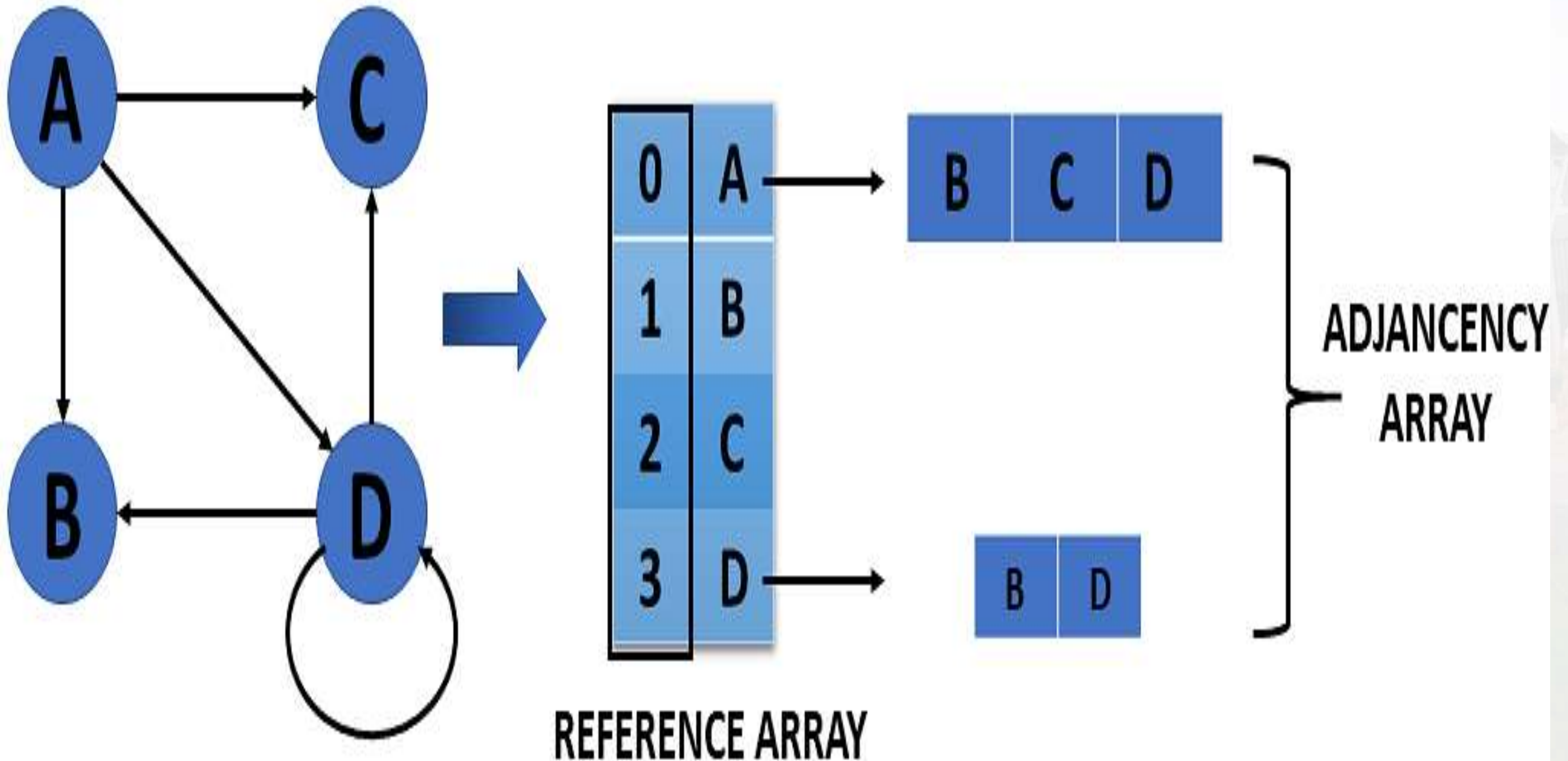
Adjacency List

Weighted Undirected Graph Representation Using Linked-List



Adjacency List

Weighted Undirected Graph Representation Using an Array



Graph Representation

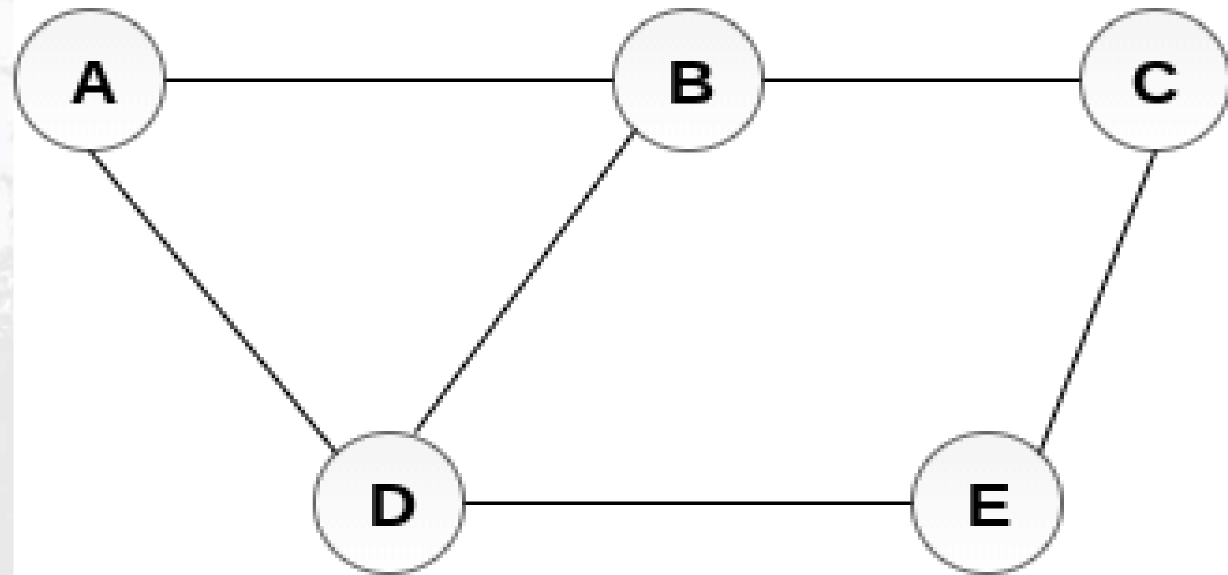
1. Sequential Representation

- In sequential representation, we use adjacency matrix to store the mapping represented by vertices and edges. In adjacency matrix, the rows and columns are represented by the graph vertices. A graph having n vertices, will have a dimension n x n.

- An entry Mij in the adjacency matrix representation of an undirected graph G will be 1 if there exists an edge between Vi and Vj.

Graph Representation

An undirected graph and its adjacency matrix representation is shown in the following figure.



Undirected Graph

	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	1	0
C	0	1	0	0	1
D	1	1	0	0	1
E	0	0	1	1	0

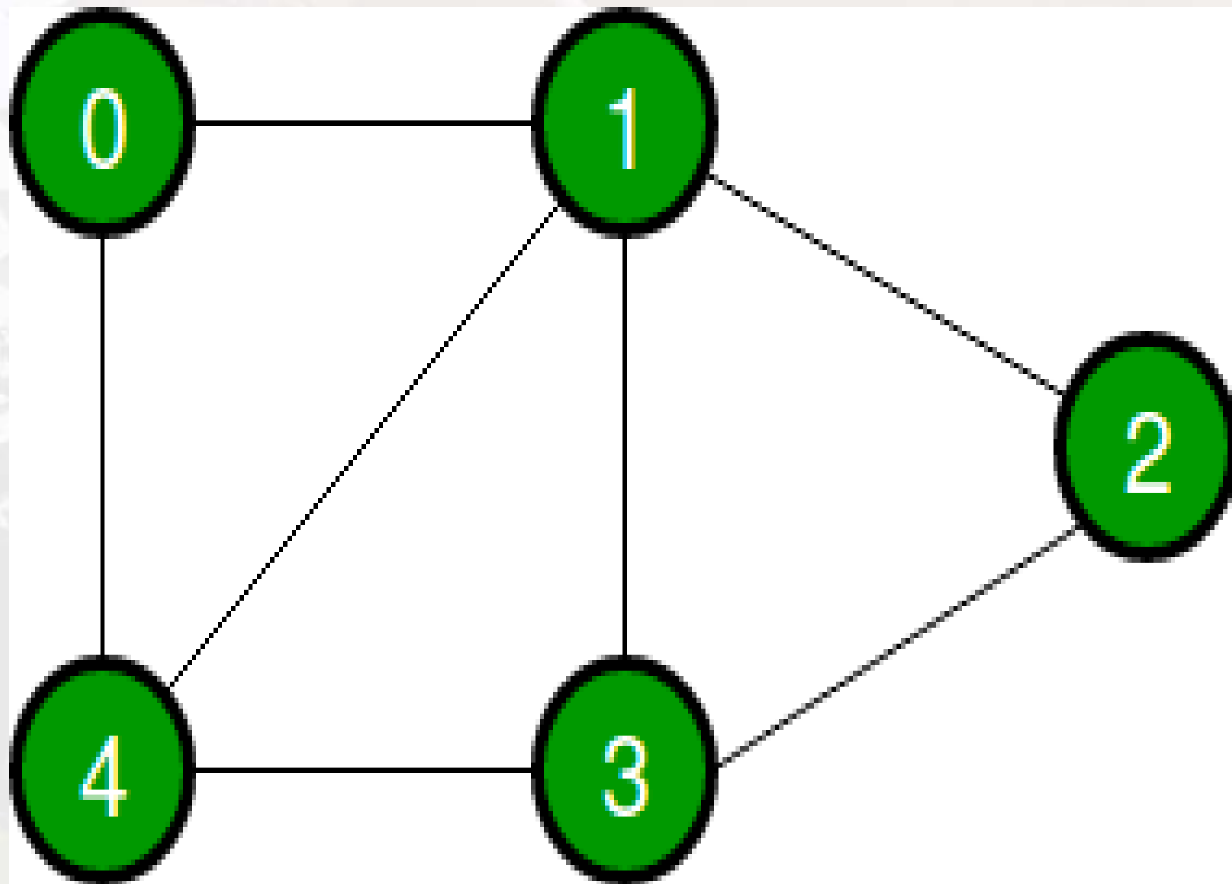
Adjacency Matrix

in the above figure, we can see the mapping among the vertices (A, B, C, D, E) is represented by using the adjacency matrix which is also shown in the figure.

There exists different adjacency matrices for the directed and undirected graph. In directed graph, an entry A_{ij} will be 1 only when there is an edge directed from V_i to V_j.

Graph Representation

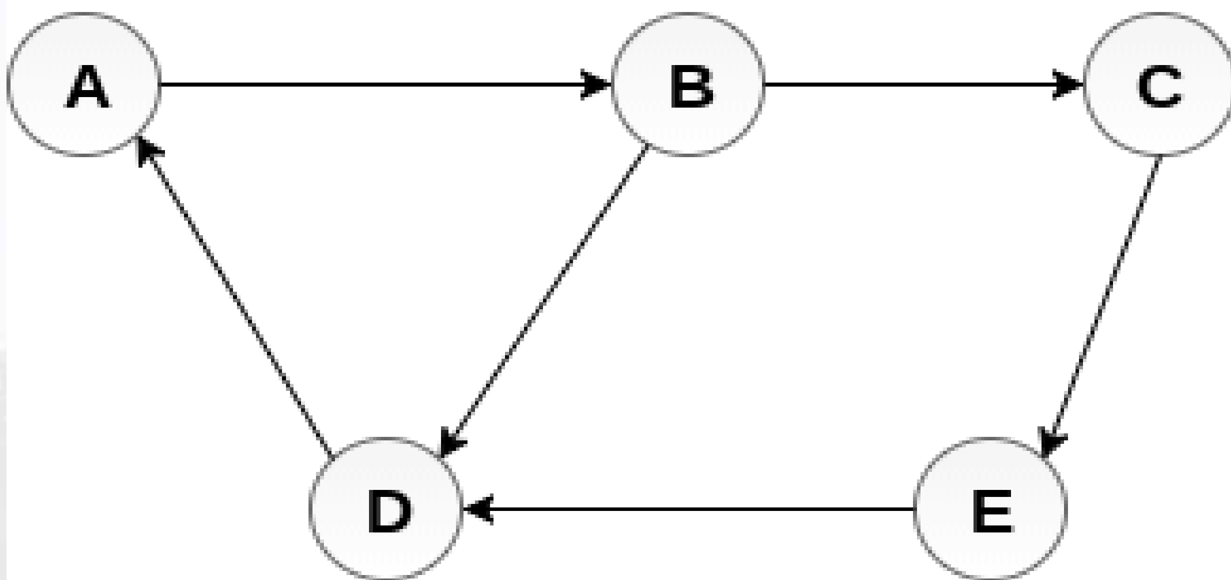
An undirected graph and its adjacency matrix representation is shown in the following figure.



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

Graph Representation

A directed graph and its adjacency matrix representation is shown in the following figure.



Directed Graph

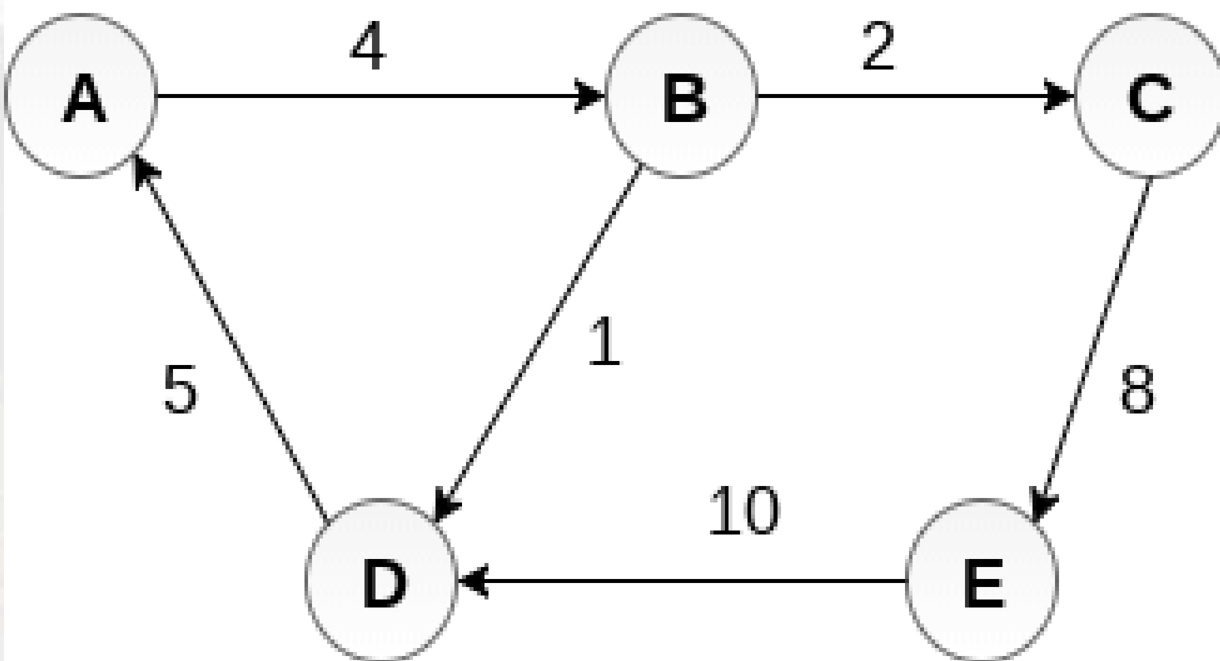
	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	1	0
C	0	0	0	0	1
D	1	0	0	0	0
E	0	0	0	1	0

Adjacency Matrix

Representation of weighted directed graph is different. Instead of filling the entry by 1, the Non- zero entries of the adjacency matrix are represented by the weight of respective edges.

Graph Representation

The weighted directed graph along with the adjacency matrix representation is shown in the following figure.



Weighted Directed Graph

	A	B	C	D	E
A	0	4	0	0	0
B	0	0	2	1	0
C	0	0	0	0	8
D	5	0	0	0	0
E	0	0	0	10	0

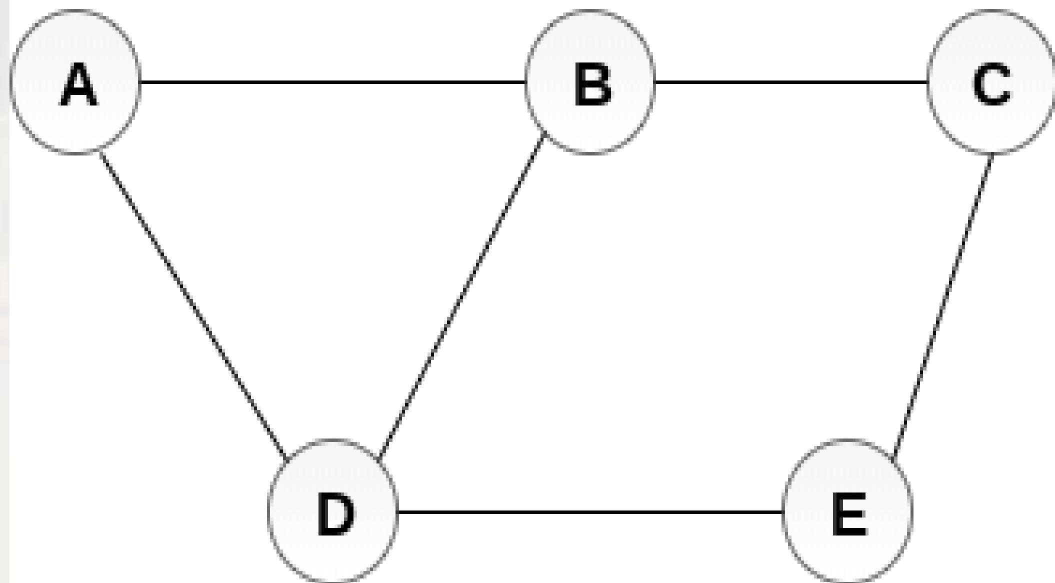
Adjacency Matrix

Graph Representation

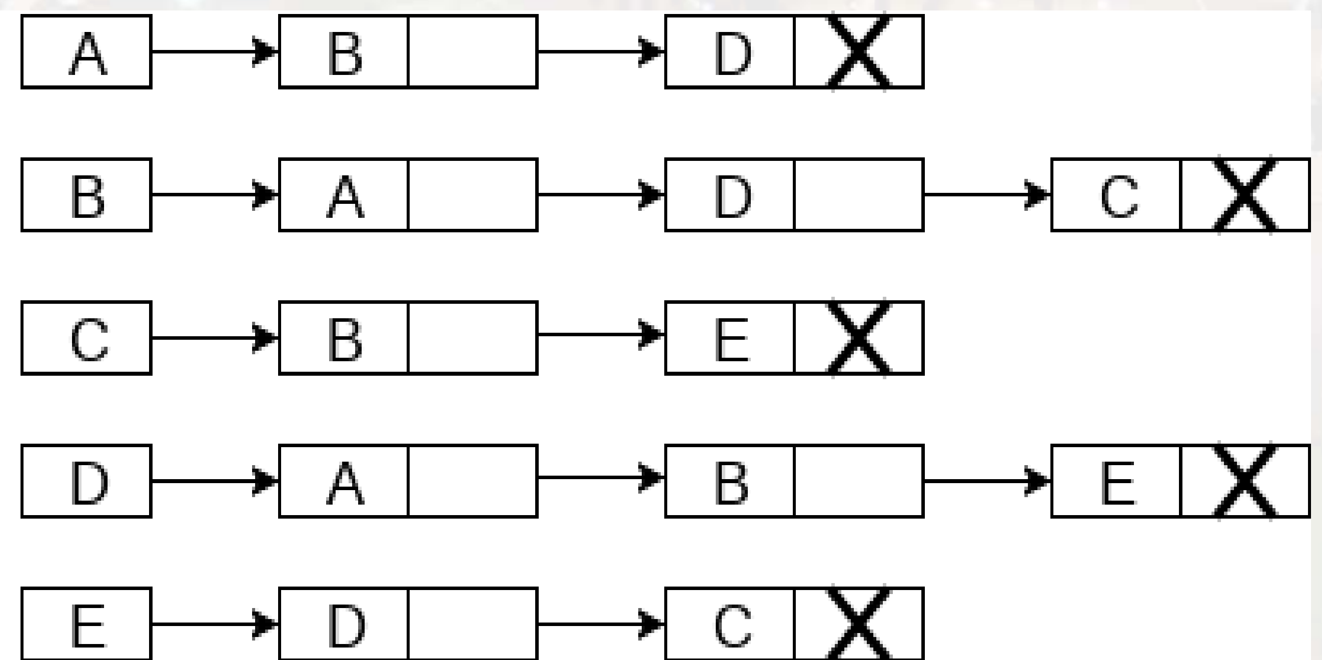
2. Linked Representation :

In the linked representation, an adjacency list is used to store the Graph into the computer's memory.

Consider the undirected graph shown in the following figure and check the adjacency list representation.



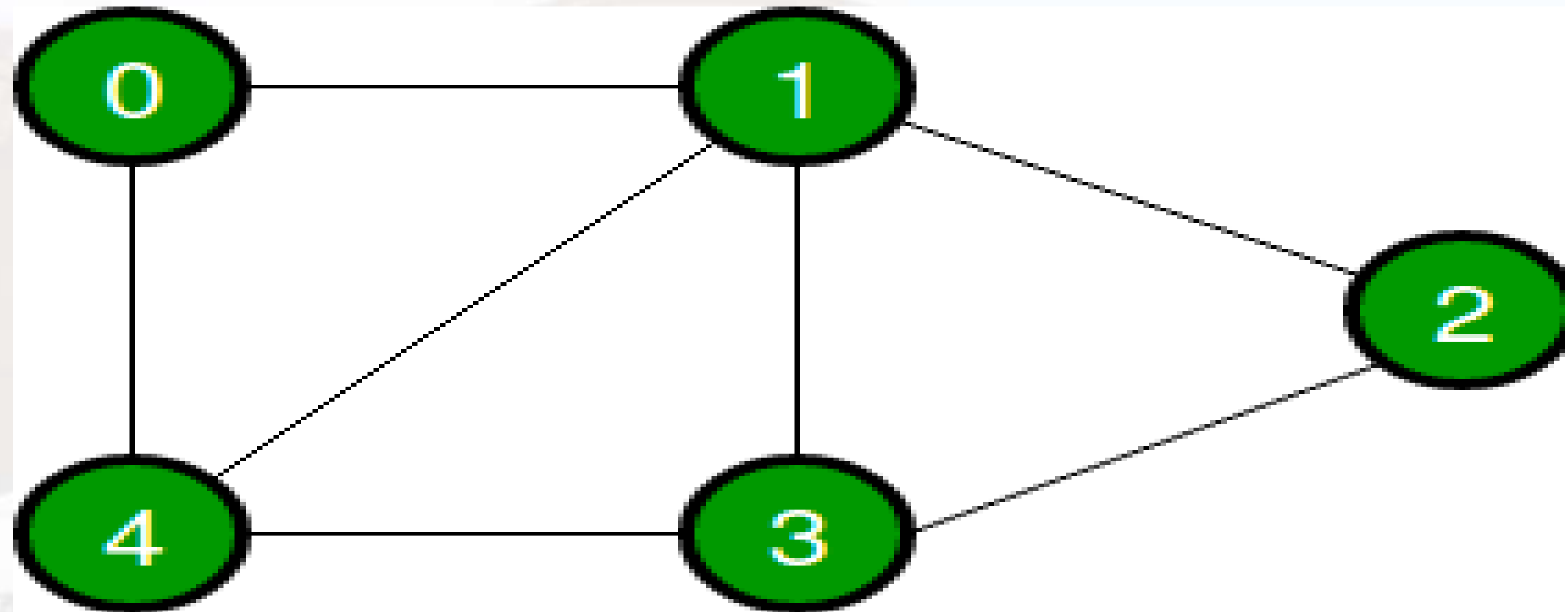
Undirected Graph



Adjacency List

Graph Representation

2. Linked Representation :



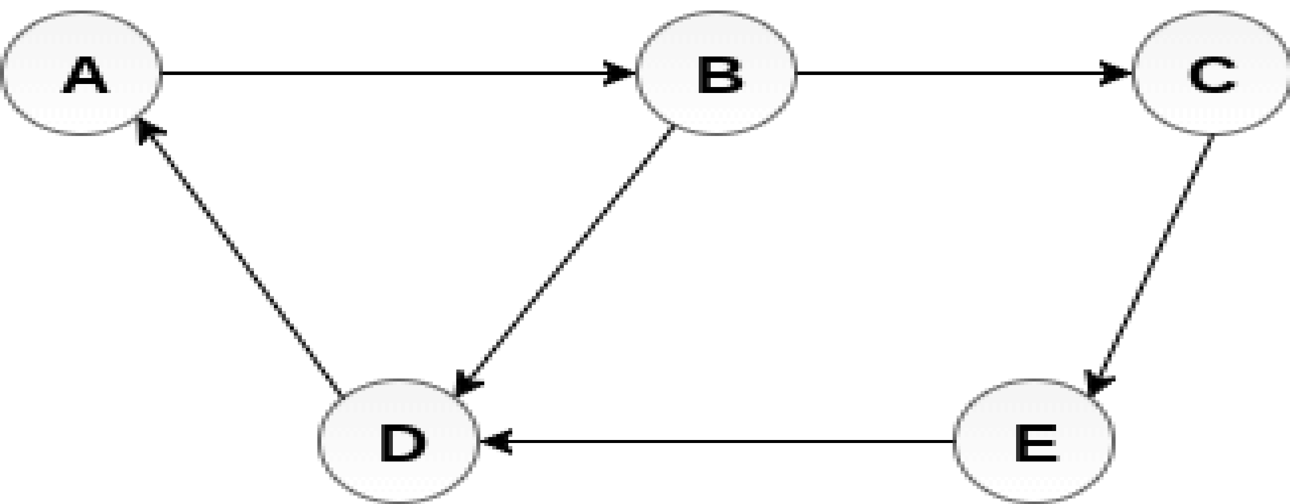
0 → 1 → 4 /
1 → 0 → 4 → 2 → 3 /
2 → 1 → 3 /
3 → 1 → 4 → 2 /
4 → 3 → 0 → 1 /

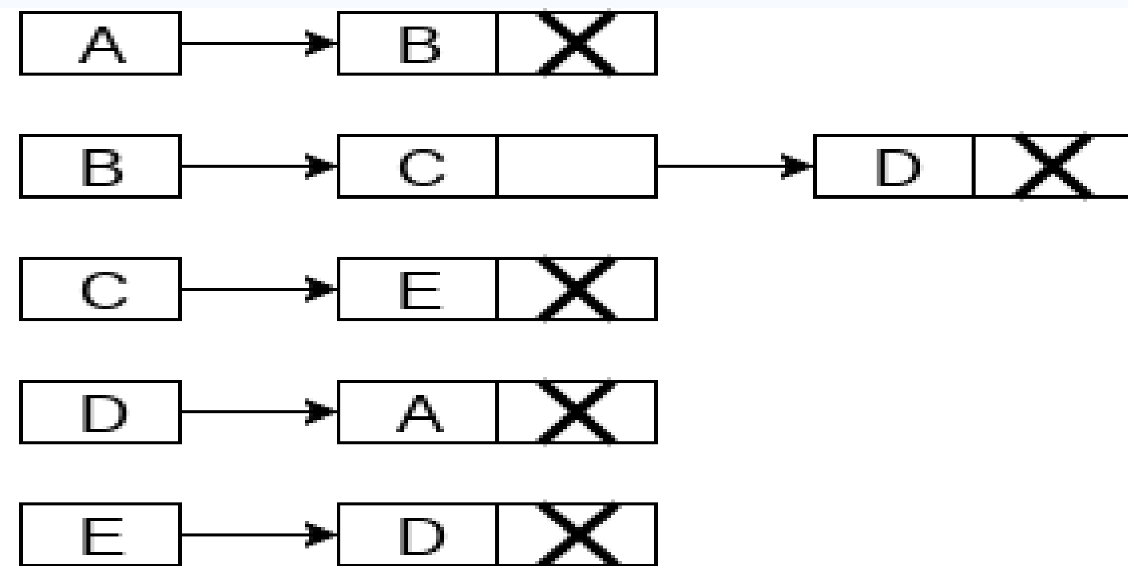
Graph Representation

An adjacency list is maintained for each node present in the graph which stores the node value and a pointer to the next adjacent node to the respective node. If all the adjacent nodes are traversed then store the NULL in the pointer field of last node of the list. The sum of the lengths of adjacency lists is equal to the twice of the number of edges present in an undirected graph.

Consider the directed graph shown in the following figure and check the adjacency list representation of the graph.

Graph Representation

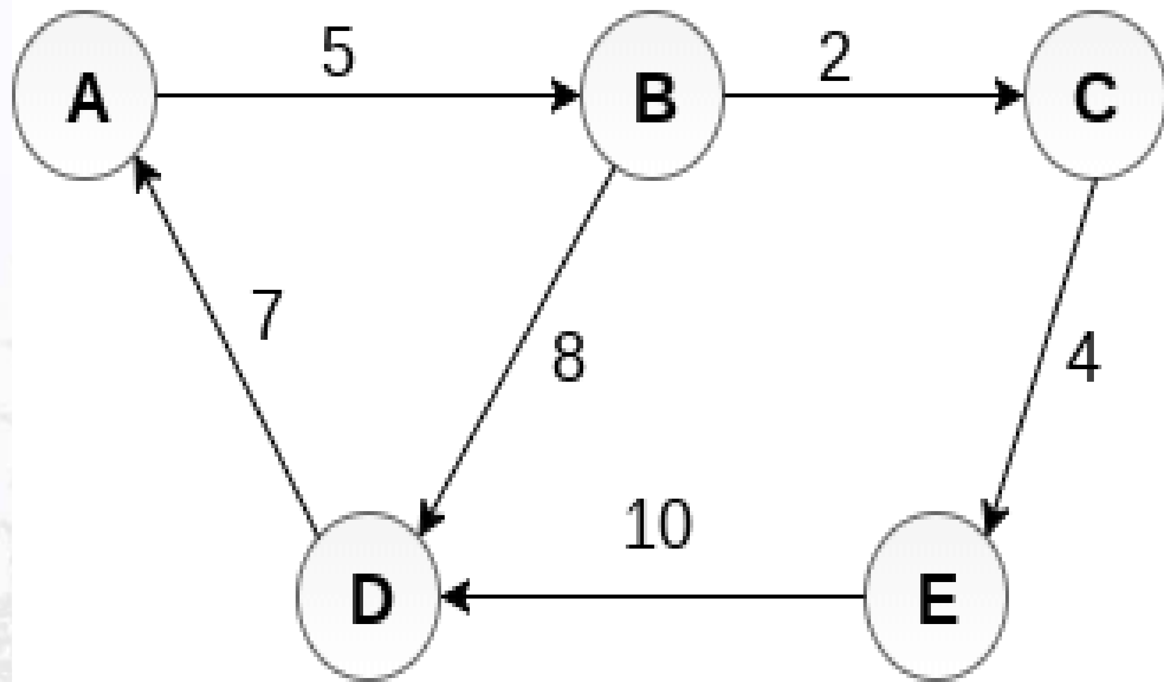




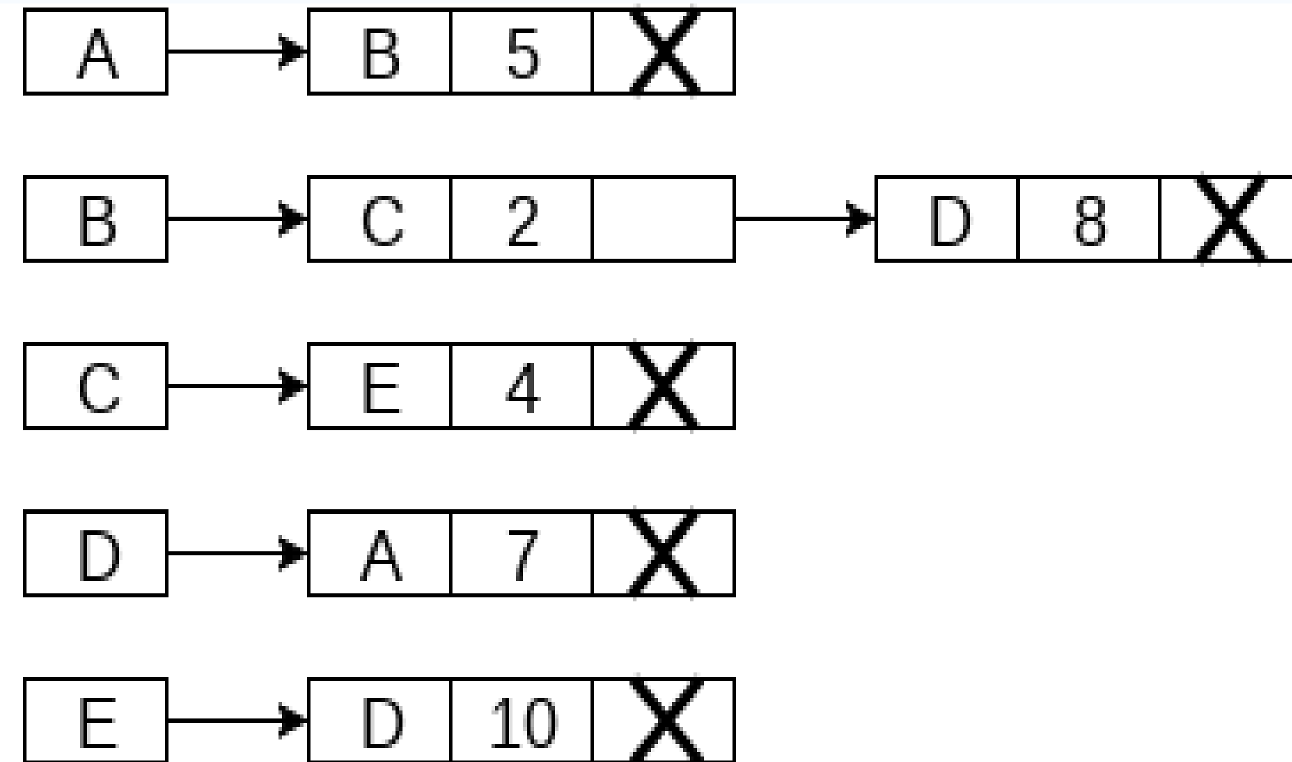
In a directed graph, the sum of lengths of all the adjacency lists is equal to the number of edges present in the graph.

In the case of weighted directed graph, each node contains an extra field that is called the weight of the node. The adjacency list representation of a directed graph is shown in the following figure.

Graph Representation



Weighted Directed Graph



Adjacency List

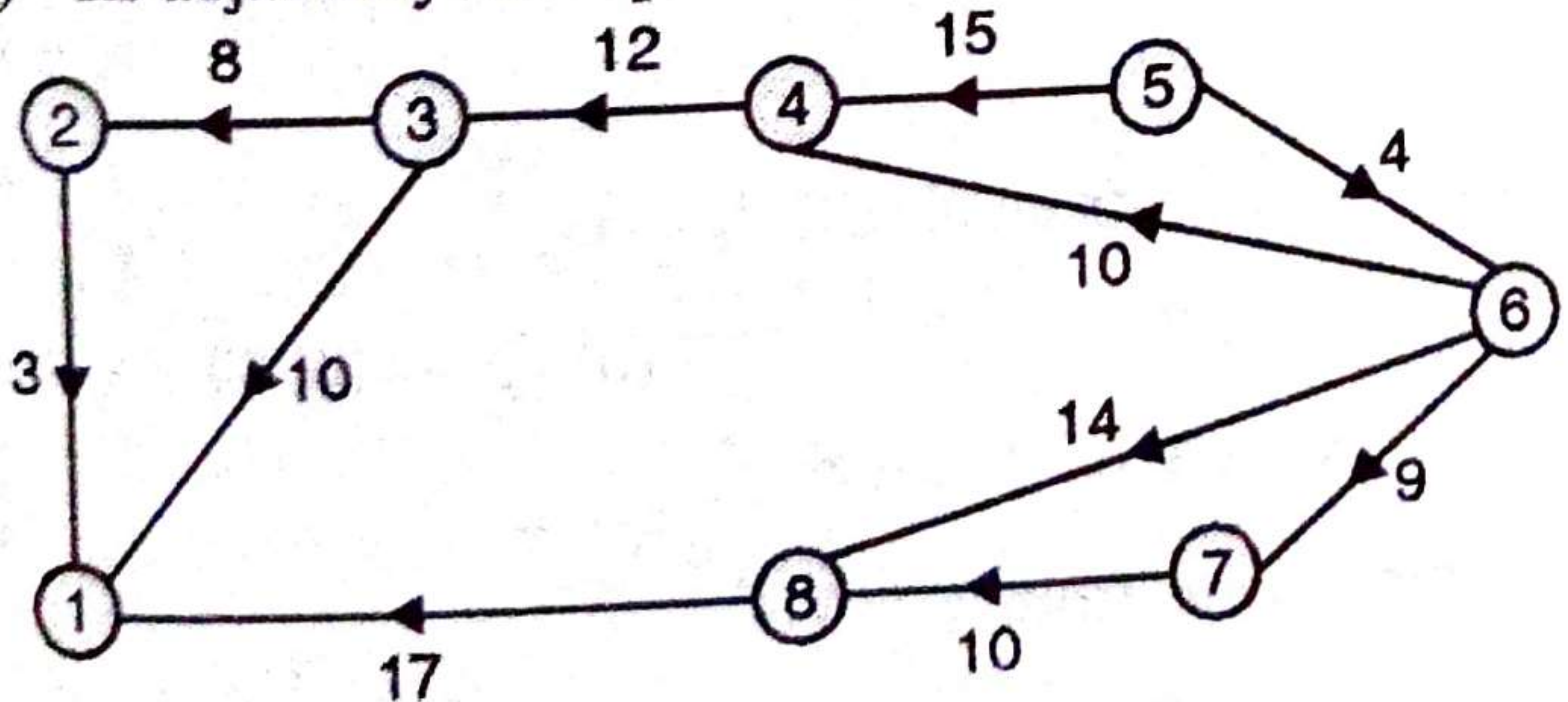
Graph Representation Examples

For the graph shown in Fig. P.3.3.1 obtain :

(i) The in degree and out degree of each vertex,

(ii) Its adjacency matrix

(iii) Its adjacency list representation.

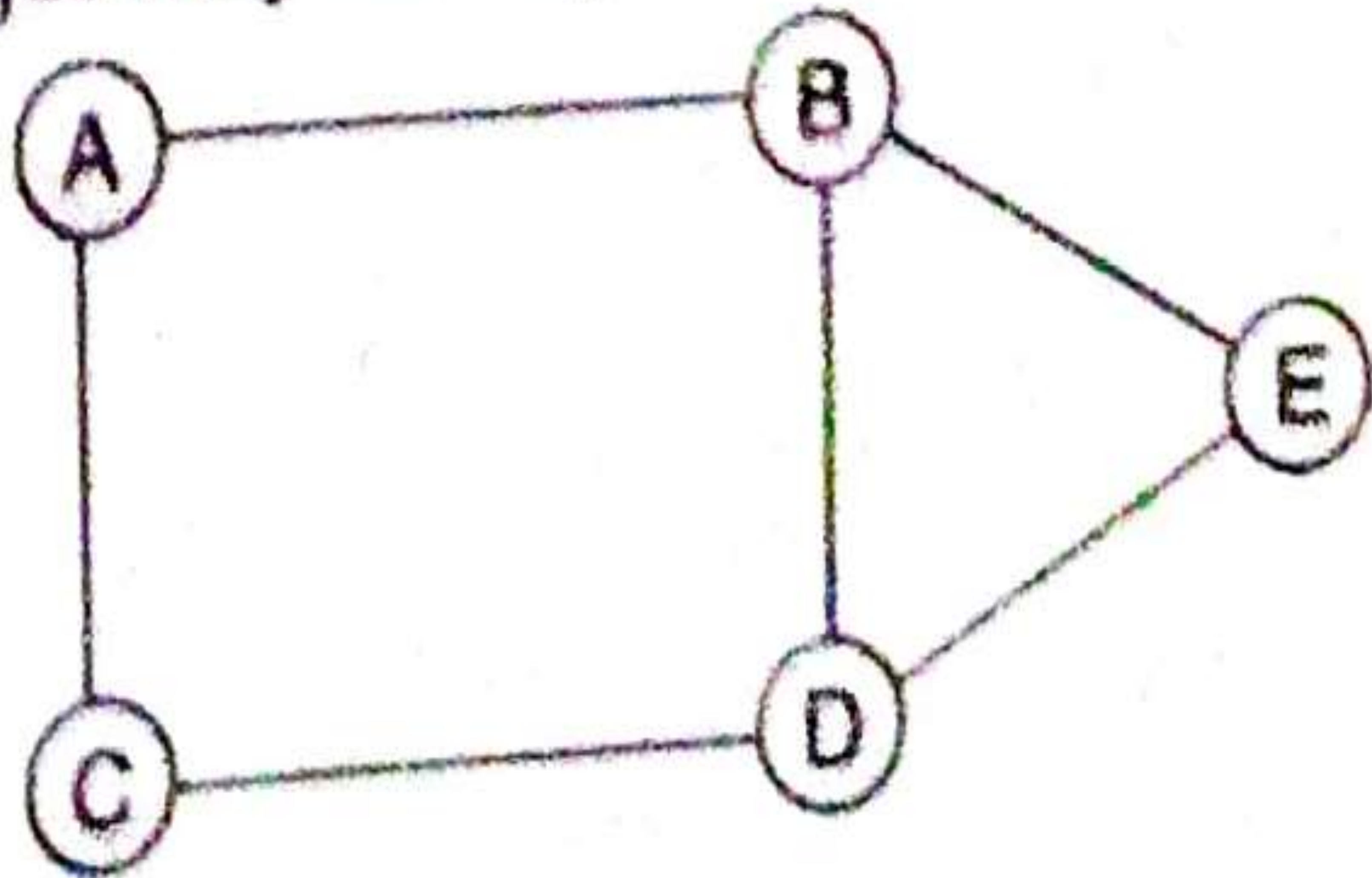


Graph Representation Examples

Consider the graph shown in Fig. P. 3.3.2.

(i) Give adjacency matrix representation.

(ii) Give adjacency list representation of the graph.

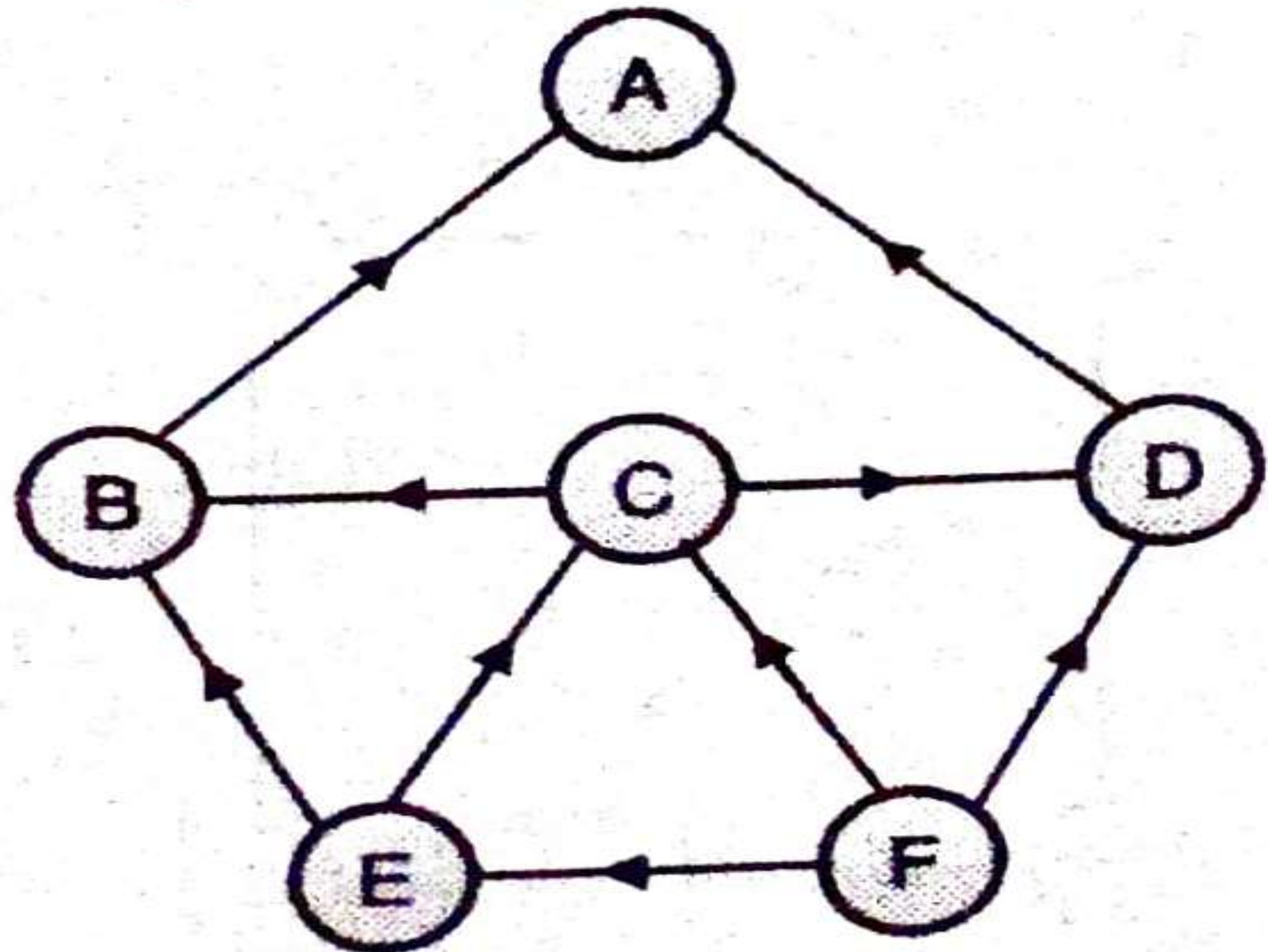


Graph Representation Examples

Consider the graph shown in Fig. P.3.3.5 :

(i) Give adjacency matrix representation.

(ii) Give adjacency list representation of graph.



Adjacency Multi-list

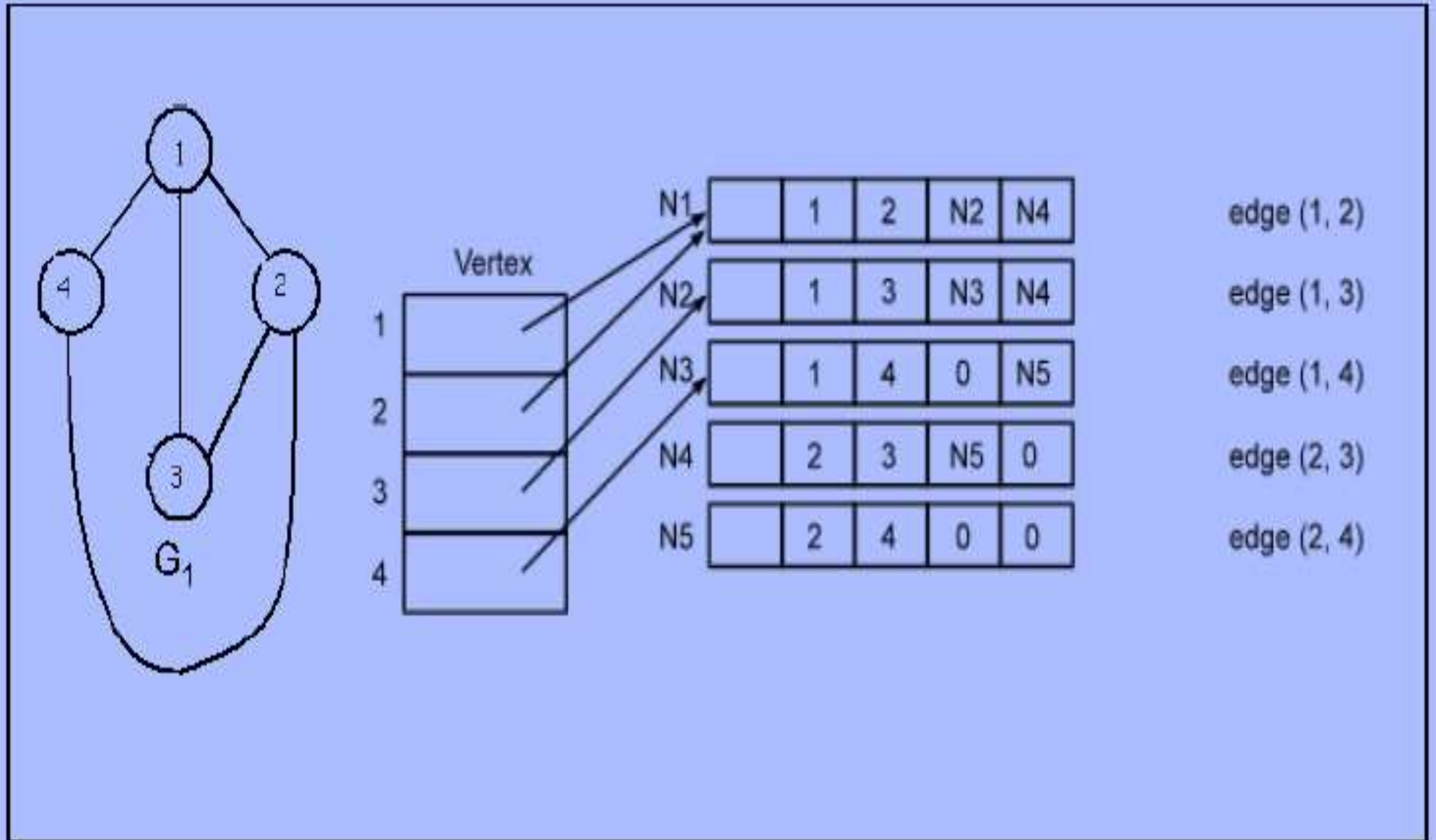
- Multiclass are lists where nodes may be shared among several other lists

Adjacency Multi-list

- The node structure of such a list can be represented as follows :

Visited tag	V_1	V_2	Link1 for V_1	Link2 for V_2

Adjacency Multi-list for Graph

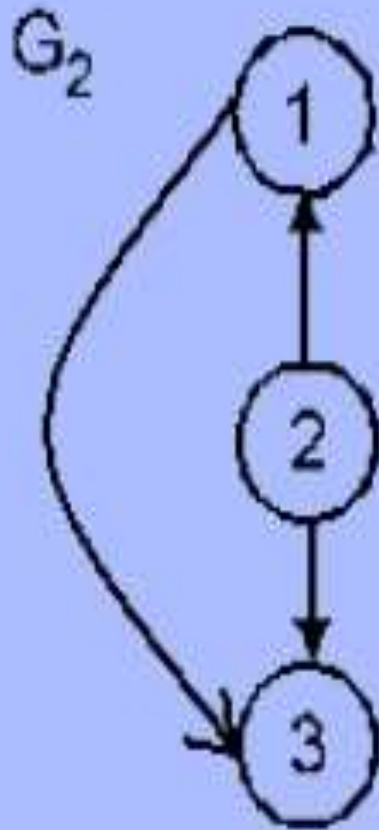


Inverse Adjacency List

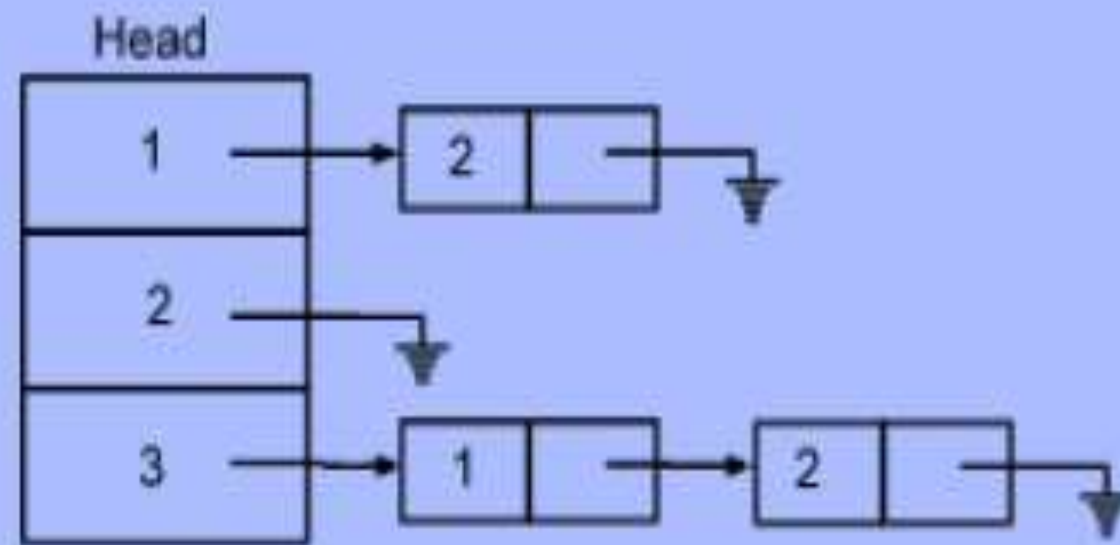
- Inverse adjacency lists is a set of lists that contain one list for vertex

- Each list contains a node per vertex adjacent to the vertex it represents

Graph and its Inverse adjacecy list

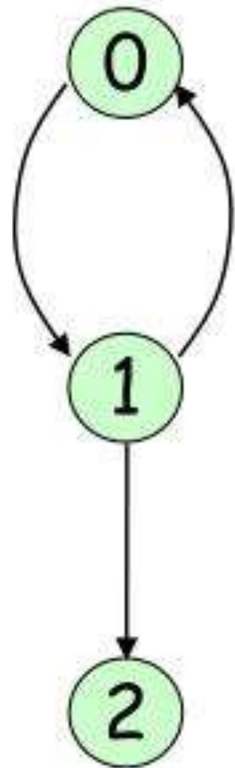


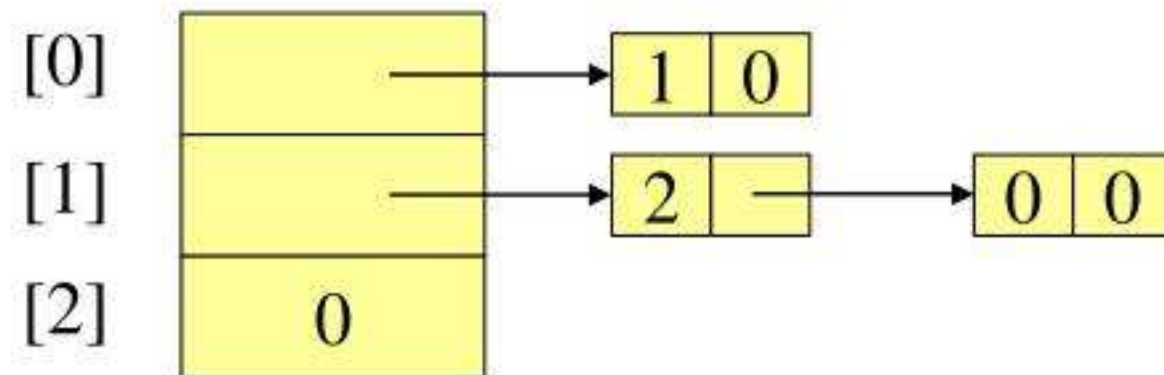
(a)

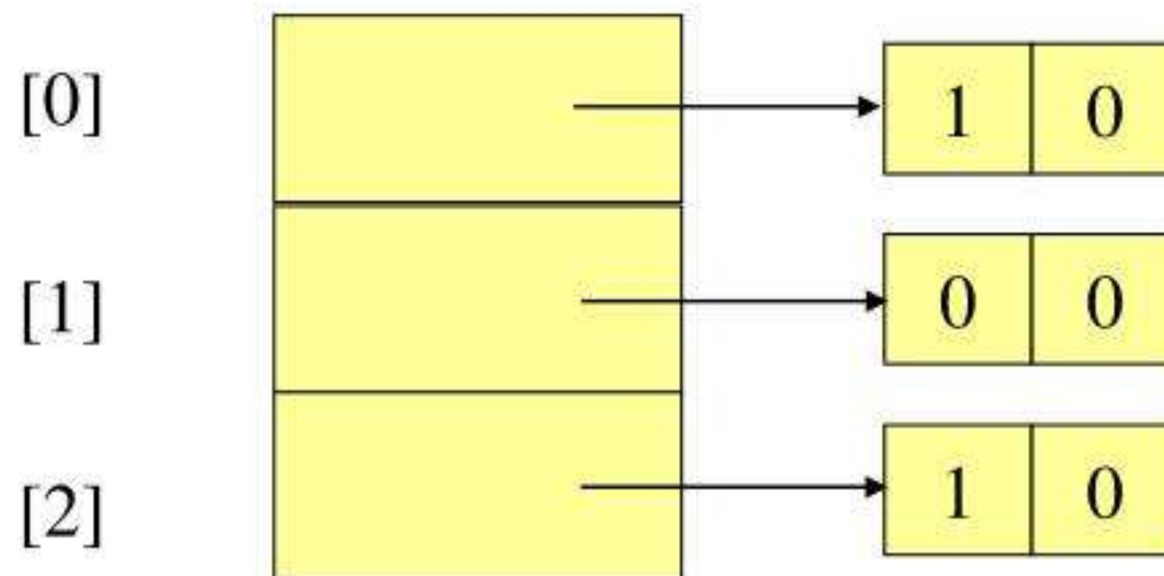


(b)

Inverse Adjacency Lists







Degree of Vertex in Graph

Degree of Vertex in a Directed Graph

In a directed graph, each vertex has an indegree and an outdegree.

Indegree of a Graph

Indegree of vertex V is the number of edges which are coming into the vertex V.

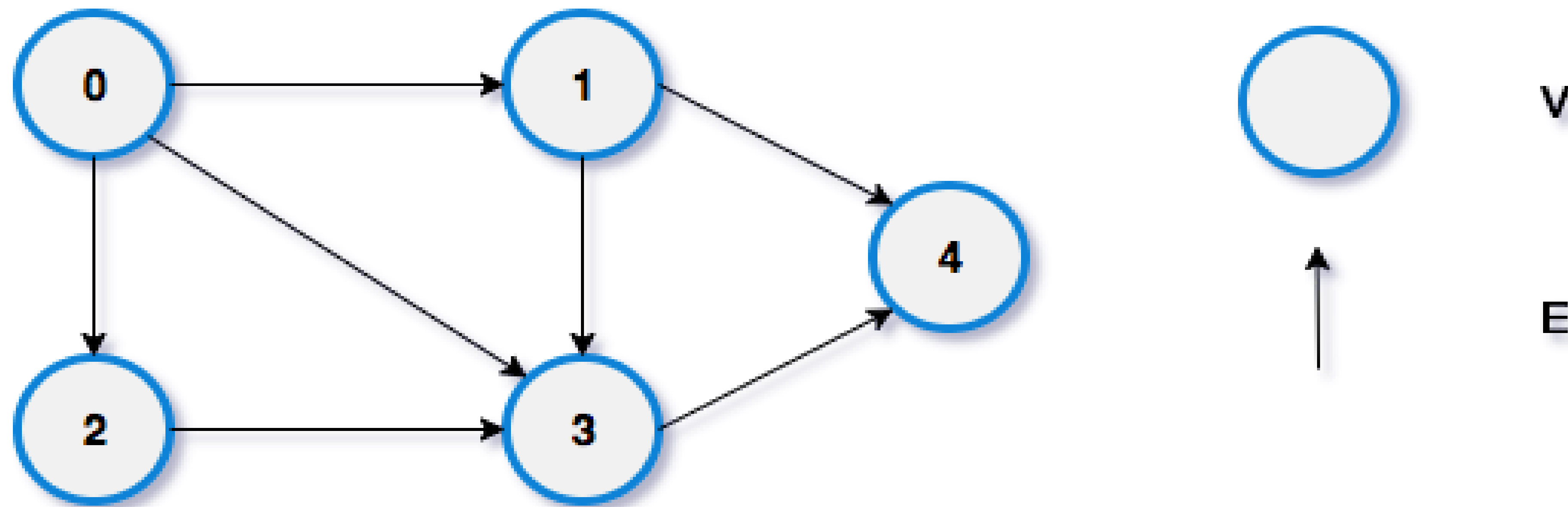
Notation − $deg^{-}(V)$.

Outdegree of a Graph

Outdegree of vertex V is the number of edges which are going out from the vertex V.

Notation − $deg^{+}(V)$.

Degree of Vertex in Graph



In-degree :

In-degree of a vertex is the number of edges coming to the vertex.

In-degree of vertex 0 = 0

In-degree of vertex 1 = 1

In-degree of vertex 2 = 1

In-degree of vertex 3 = 3

In-degree of vertex 4 = 2

Out-degree

Out-degree of a vertex is the number edges which are coming out from the vertex.

Out-degree of vertex 0 = 3

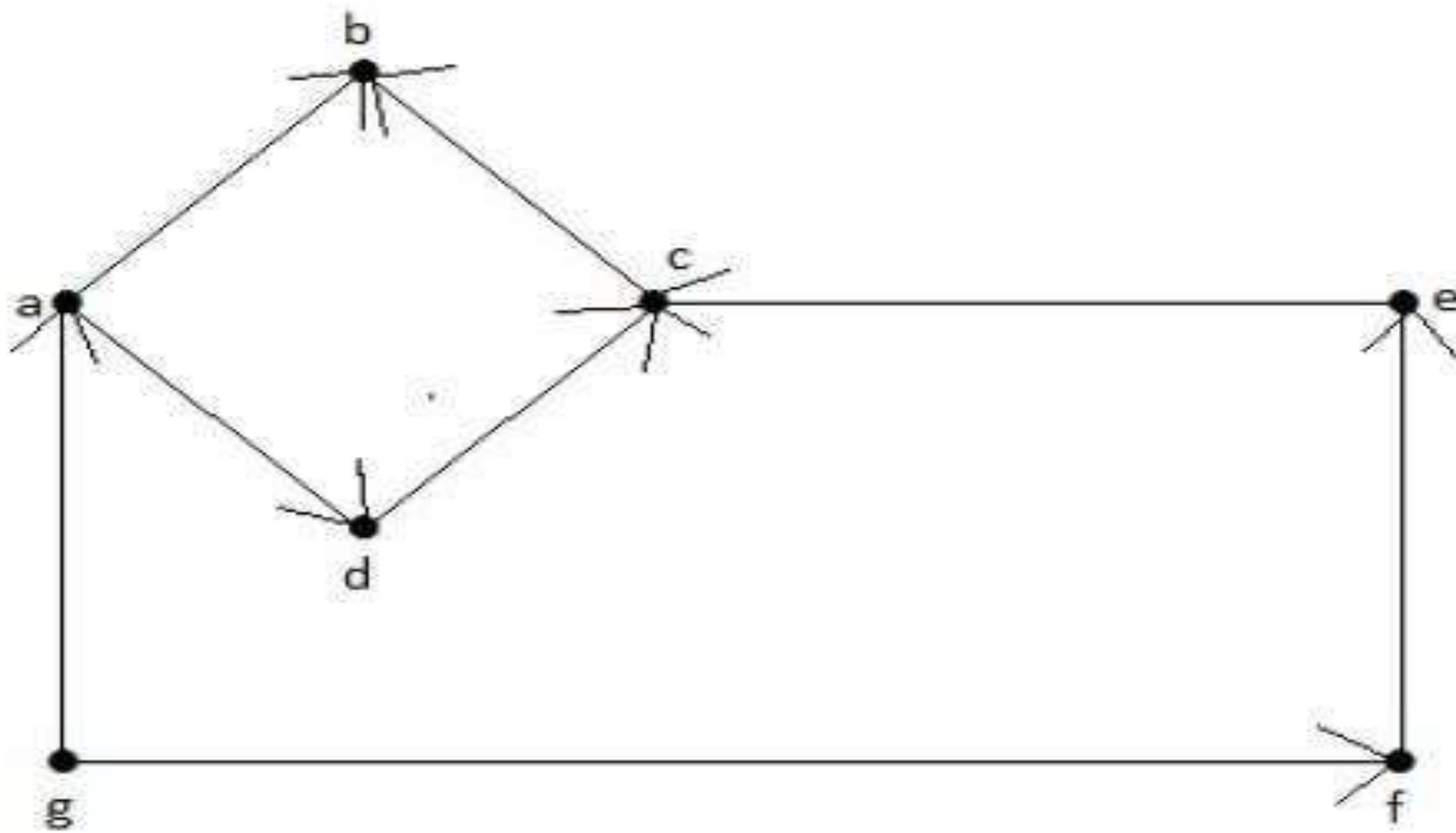
Out-degree of vertex 1 = 2

Out-degree of vertex 2 = 1

Out-degree of vertex 3 = 1

Out-degree of vertex 4 = 0

Degree of Vertex in Graph



Vertex	Indegree	Outdegree
a	1	2
b	2	0
c	2	1
d	1	1
e	1	1
f	1	1
g	0	2

Graph Traversal Algorithms

Traversing the graph means examining all the nodes and vertices of the graph. There are two standard methods by using which, we can traverse the graphs. Lets discuss each one of them in detail.

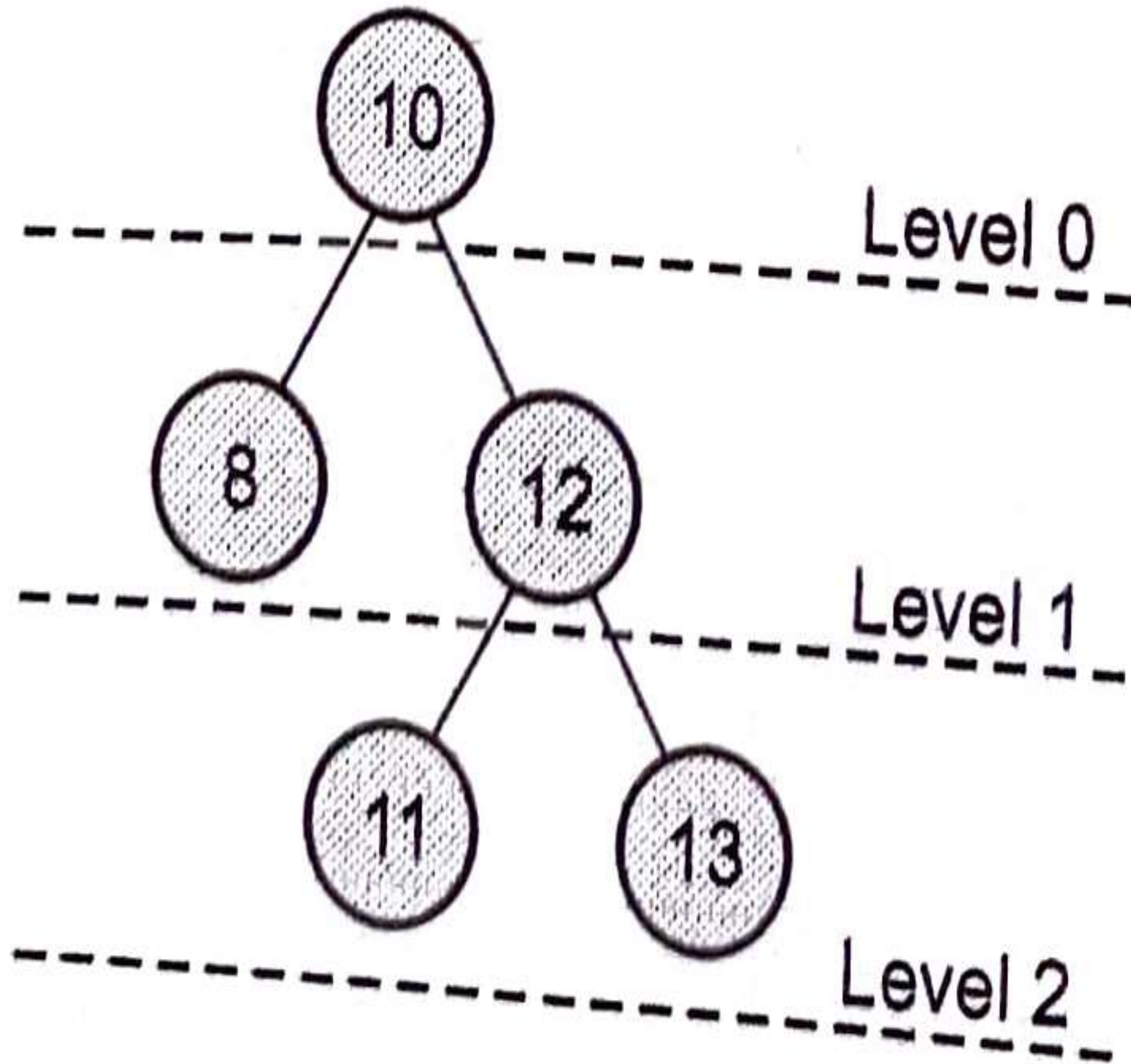
1. Breadth First Search
2. Depth First Search

BFS Algorithms

1. Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unvisited nodes. While using BFS for traversal, any node in the graph can be considered as the root node.

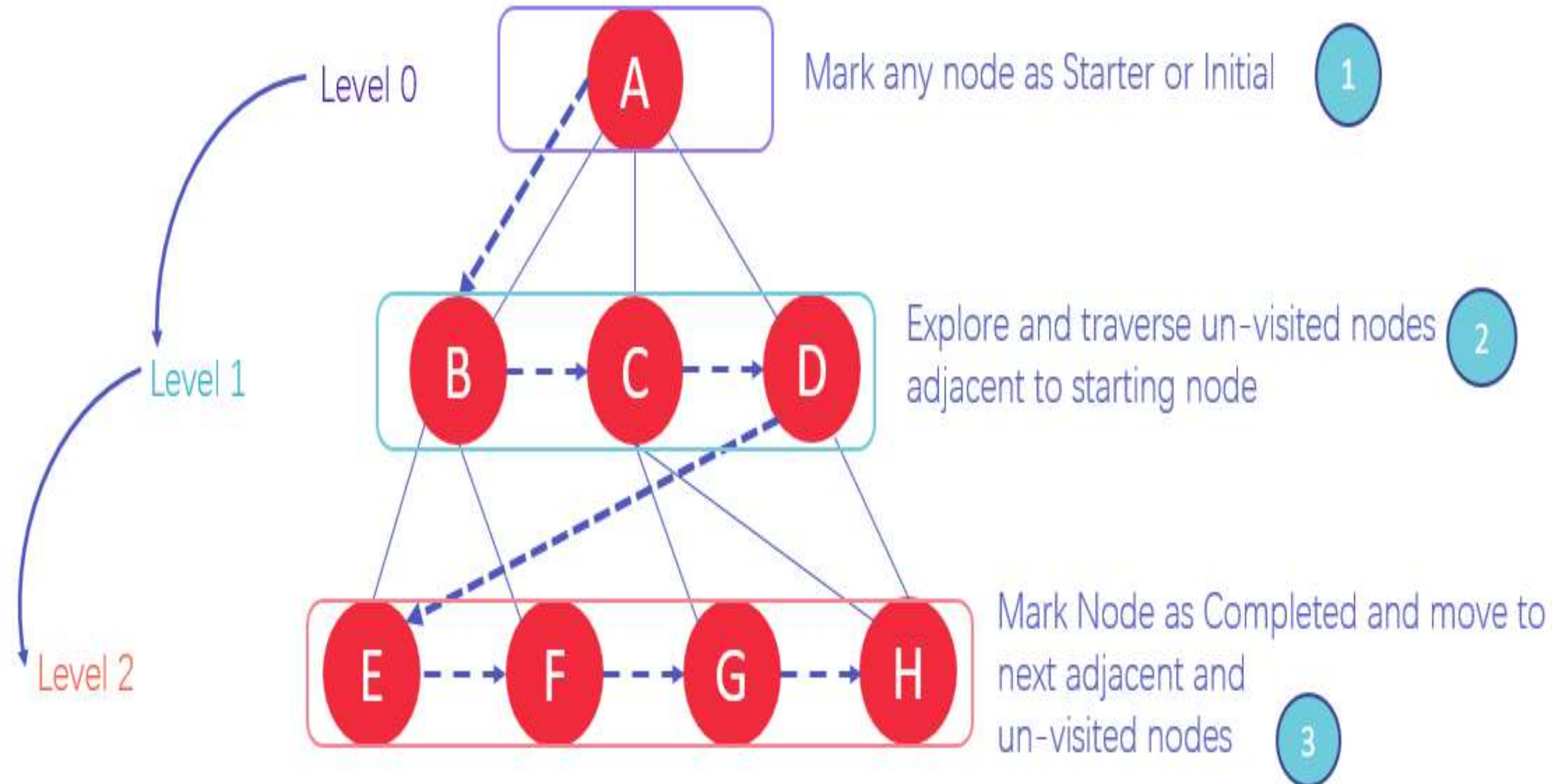
2. BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories - visited and non-visited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node

BFS Algorithms



visit 10,
visit 8, 12
visit 11, 13
∴ The BFS sequence will be **10, 8, 12, 11, 13**

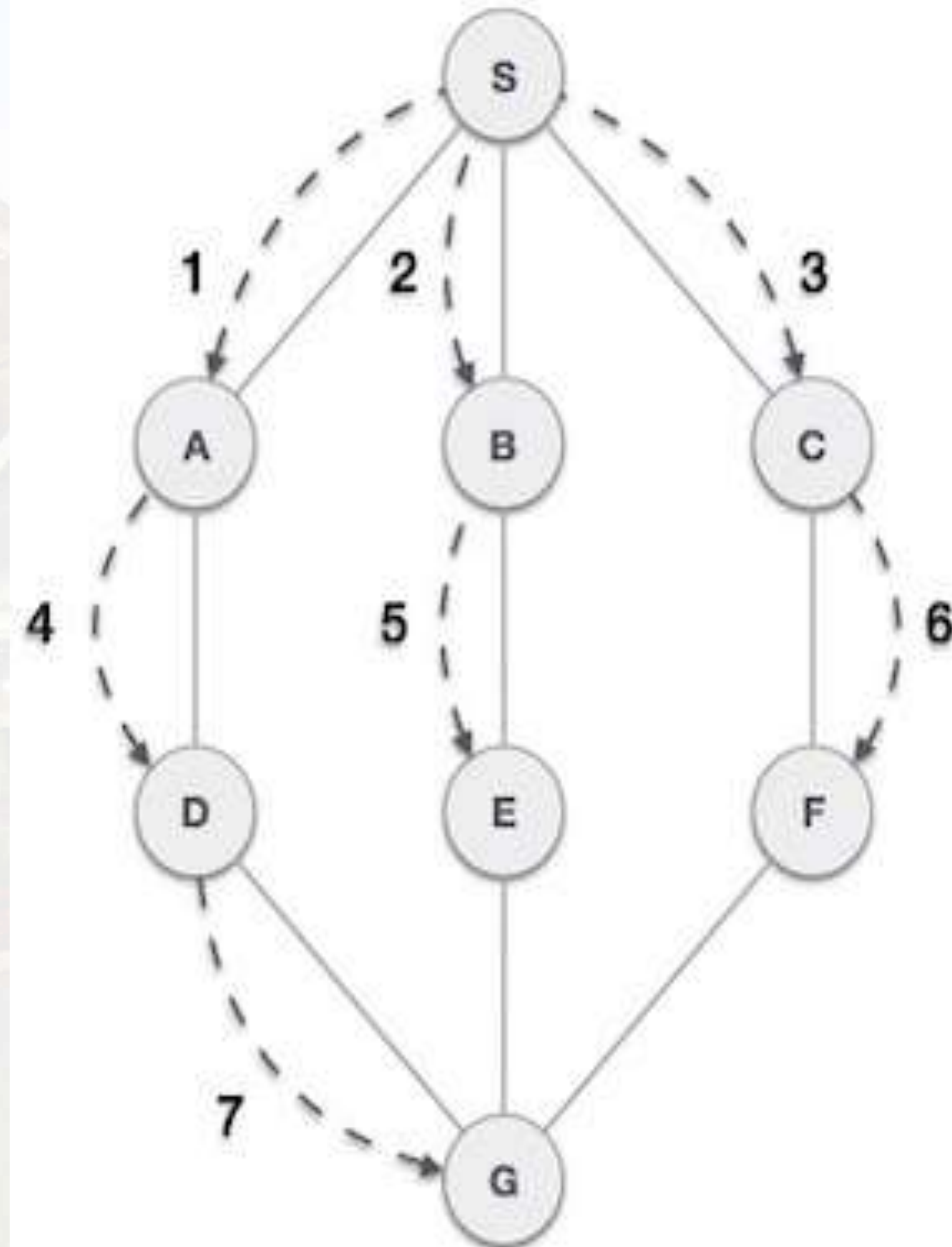
BFS Algorithms



BFS Algorithms

As in the example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs the following rules.

- **Rule 1** − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
- **Rule 2** − If no adjacent vertex is found, remove the first vertex from the queue.
- **Rule 3** − Repeat Rule 1 and Rule 2 until the queue is empty.



BFS Algorithms

Step 1: SET STATUS = 1 (ready state)

for each node in G

Step 2: Enqueue the starting node A

and set its STATUS = 2

(waiting state)

Step 3: Repeat Steps 4 and 5 until

QUEUE is empty

Step 4: Dequeue a node N. Process it

and set its STATUS = 3

(processed state).

Step 5: Enqueue all the neighbours of

N that are in the ready state

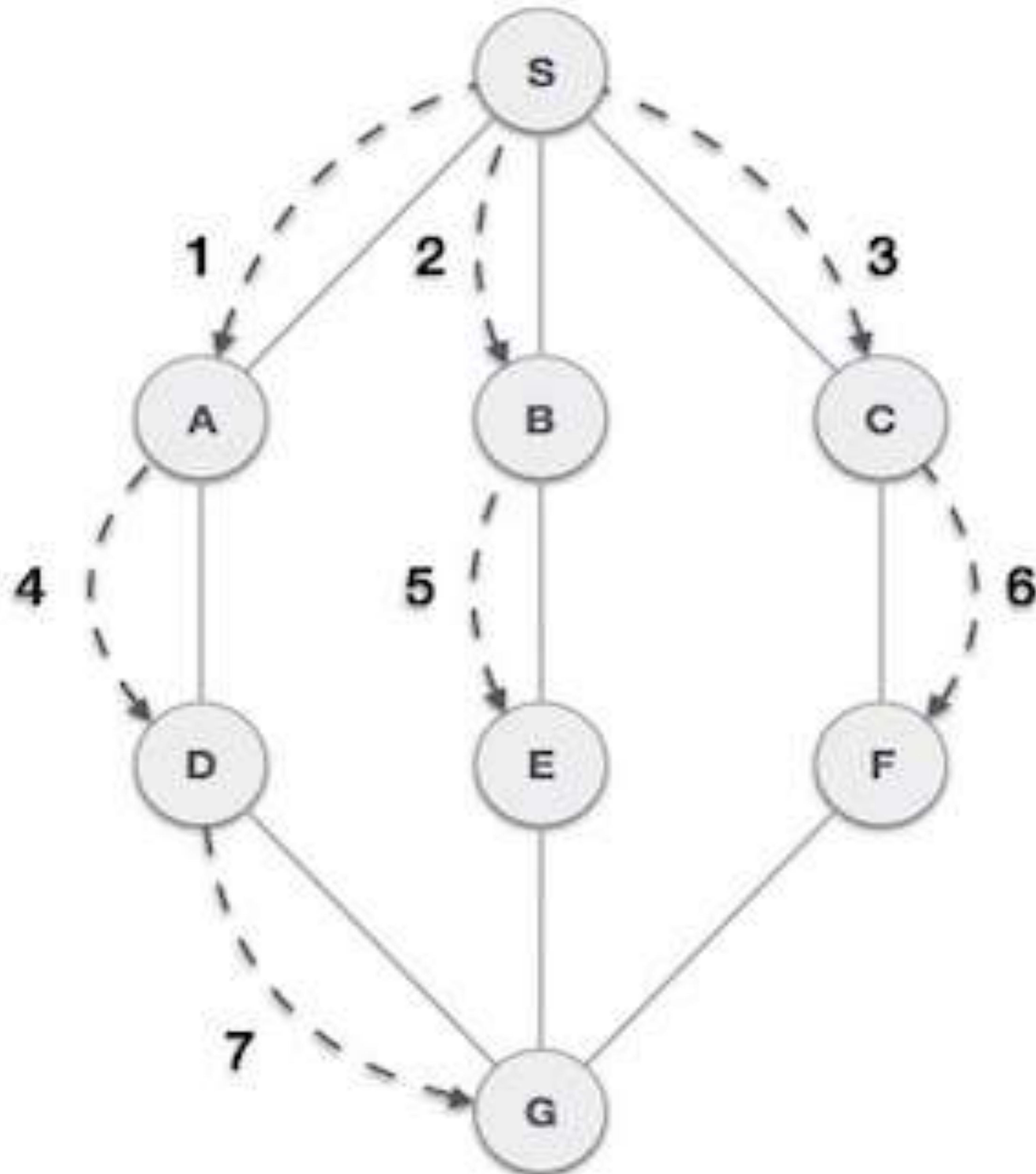
(whose STATUS = 1) and set

their STATUS = 2

(waiting state)

[END OF LOOP]

Step 6: EXIT



BFS Algorithms

Rule 1 – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.

Rule 2 – If no adjacent vertex is found, remove the first vertex from the queue.

Rule 3 – Repeat Rule 1 and Rule 2 until the queue is empty.

BFS Algorithms

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

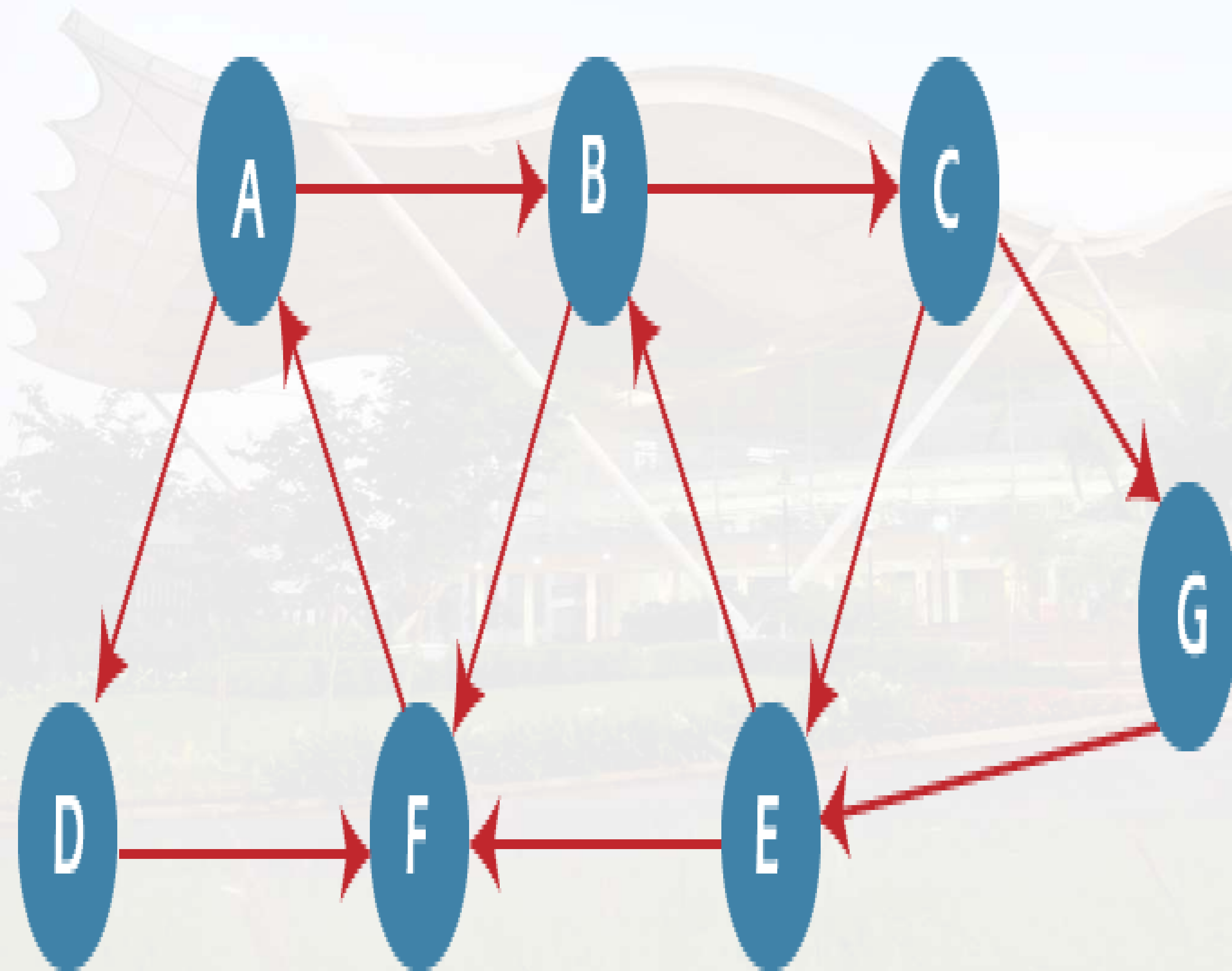
Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set

their STATUS = 2

(waiting state)

[END OF LOOP]

Example of BFS Algorithms



Adjacency Lists

A : B, D

B : C, F

C : E, G

G : E

E : B, F

F : A

D : F

Complexity of BFS Algorithms

The time complexity of the BFS algorithm is represented in the form of O(V + E), where V is the number of nodes and E is the number of edges.

The space complexity of the algorithm is O(V).

Advantages and Disadvantages of BFS

- **Advantages :**

1. A BFS will find the shortest path between the starting point and any other reachable node.
2. A depth-first search will not necessarily find the shortest path.

- **Disadvantages :**

A BFS on a binary tree generally requires more memory than a DFS.

Application of BFS Algorithms

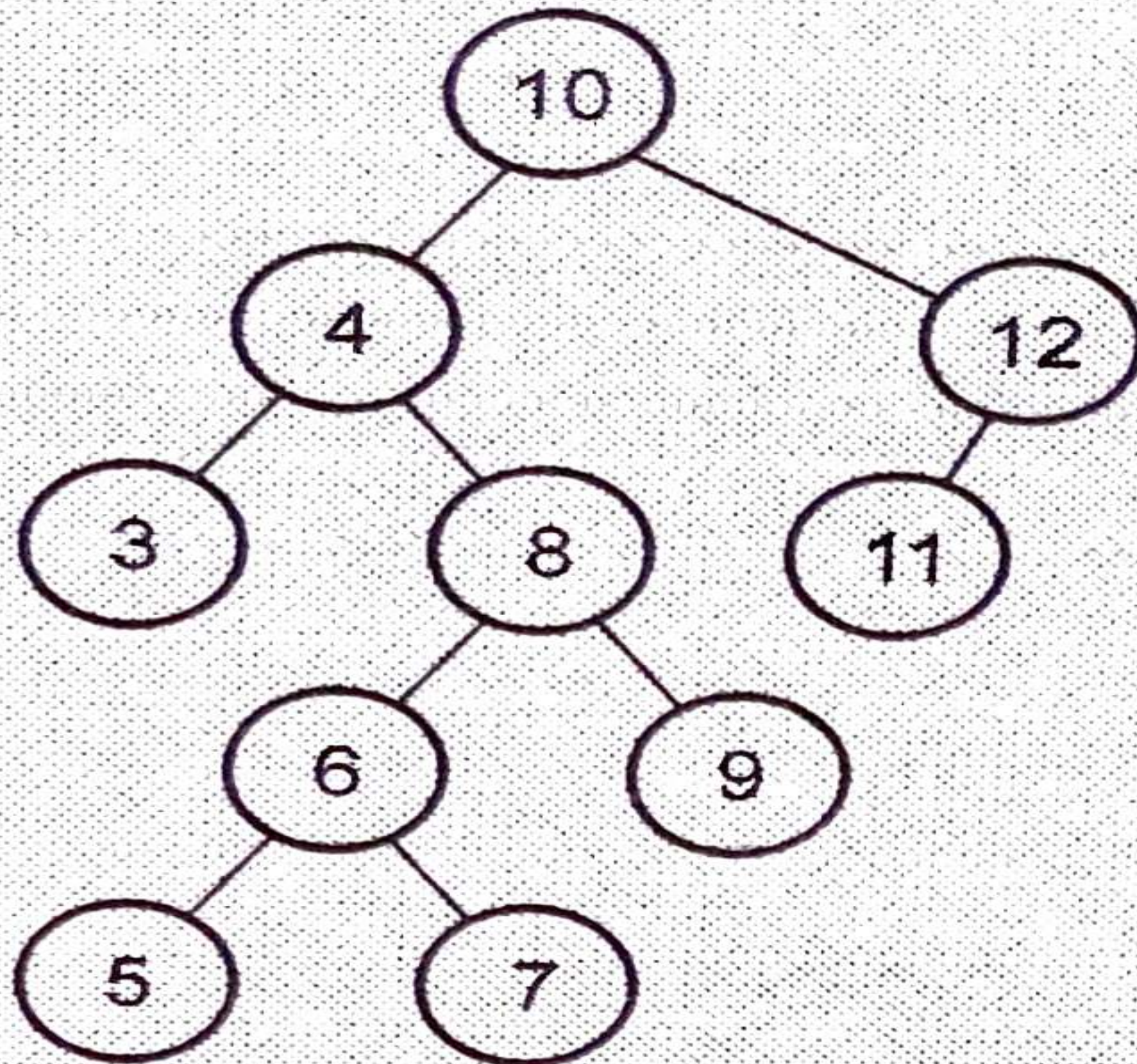
1. BFS can be used to find the neighboring locations from a given source location.
2. In a peer-to-peer network, BFS algorithm can be used as a traversal method to find all the neighboring nodes. Most torrent clients, such as BitTorrent, uTorrent, etc. employ this process to find "seeds" and "peers" in the network.
3. BFS is used to determine the shortest path and minimum spanning tree.
4. BFS is also used in Cheney's technique to duplicate the garbage collection.
5. It can be used in ford-Fulkerson method to compute the maximum flow in a flow network.

Application of BFS Algorithms

1. To build index by search index
2. For GPS navigation
3. Path finding algorithms
4. Cycle detection in an undirected graph
5. In minimum spanning tree

Application of BFS Algorithms

Consider following tree and find its BFS sequence.



7, display it.

BFS sequence : 10, 4, 12, 3, 8, 11, 6, 9, 5, 7

Depth First Search (DFS)

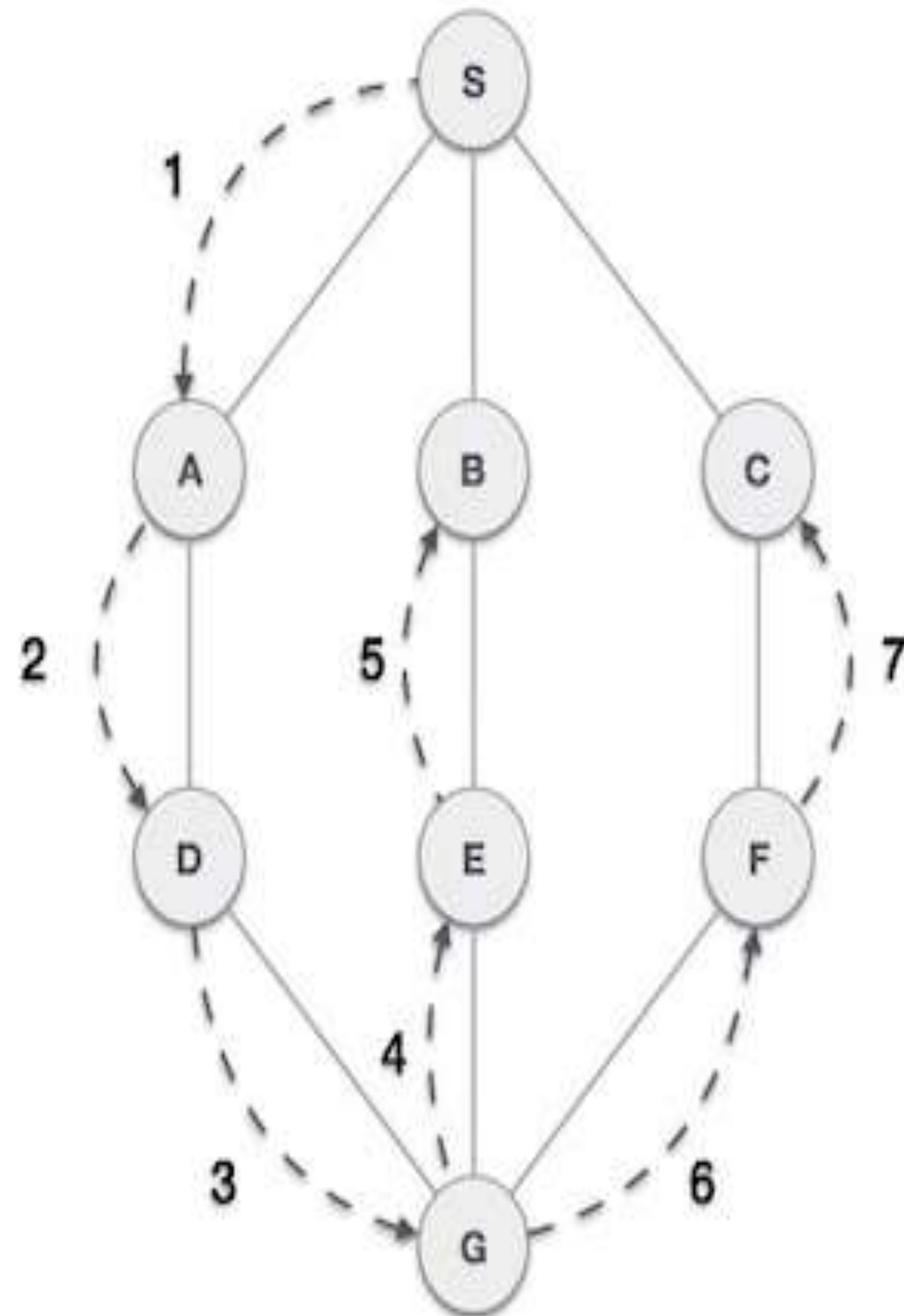
Depth first search (DFS) algorithm starts with the initial node of the graph G, and then goes to deeper and deeper until we find the goal node or the node which has no children. The algorithm, then backtracks from the dead end towards the most recent node that is yet to be completely unexplored.

The data structure which is being used in DFS is stack. The process is similar to BFS algorithm. In DFS, the edges that leads to an unvisited node are called discovery edges while the edges that leads to an already visited node are called block edges.

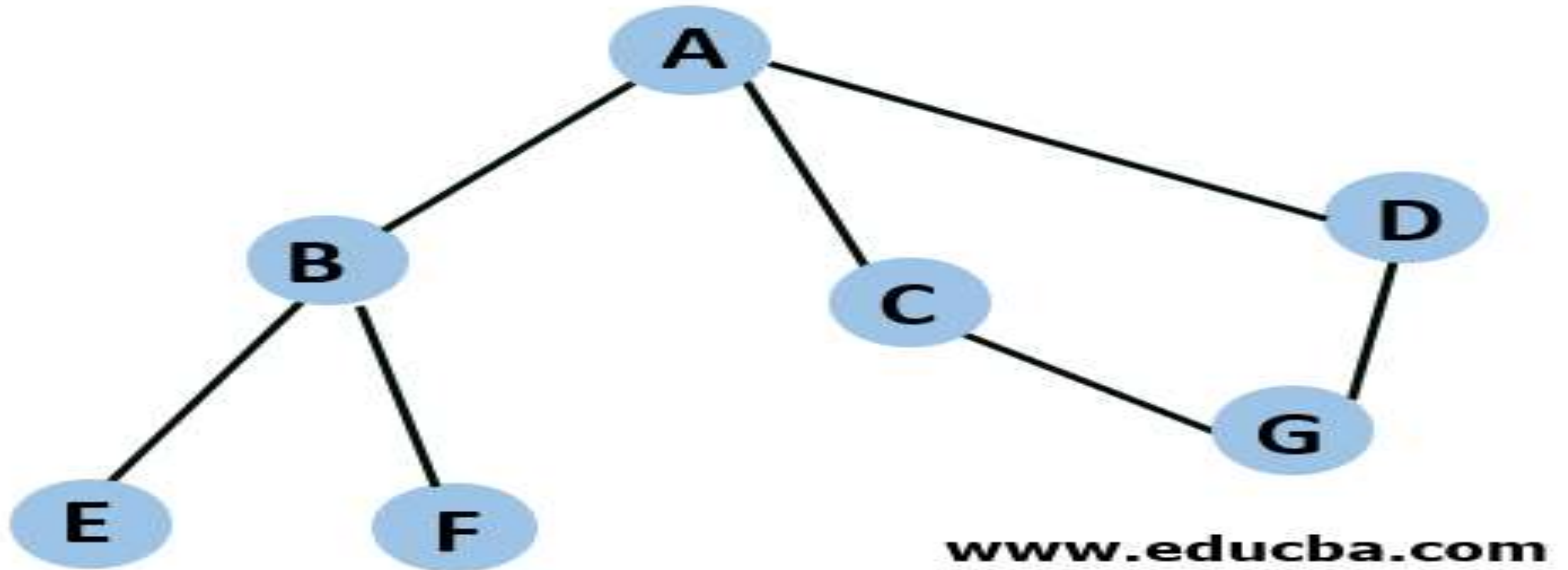
DFS Algorithms

Depth First Search (DFS) algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

- As in the example given above, DFS algorithm traverses from A to B to C to D first then to E, then to F and lastly to G. It employs the following rules.
- **Rule 1** − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.
- **Rule 2** − If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
- **Rule 3** − Repeat Rule 1 and Rule 2 until the stack is empty



DFS Algorithms



Traversing the above graph in BFS fashion would result from A -> B -> E -> F -> C -> G -> D. The algorithm starts from node A and traverses all its child nodes. As soon as it encounters B, it seems that it has further child nodes. So, the child nodes of B are traversed before proceeding to the next child node of A

DFS Algorithms

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbours of N that are in the ready state (whose STATUS = 1) and set their

STATUS = 2 (waiting state)

[END OF LOOP]

Step 6: EXIT

DFS Algorithms

Depth first search (DFS) algorithm starts with the initial node of the graph G, and then goes to deeper and deeper until we find the goal node or the node which has no children. The algorithm, then backtracks from the dead end towards the most recent node that is yet to be completely unexplored.

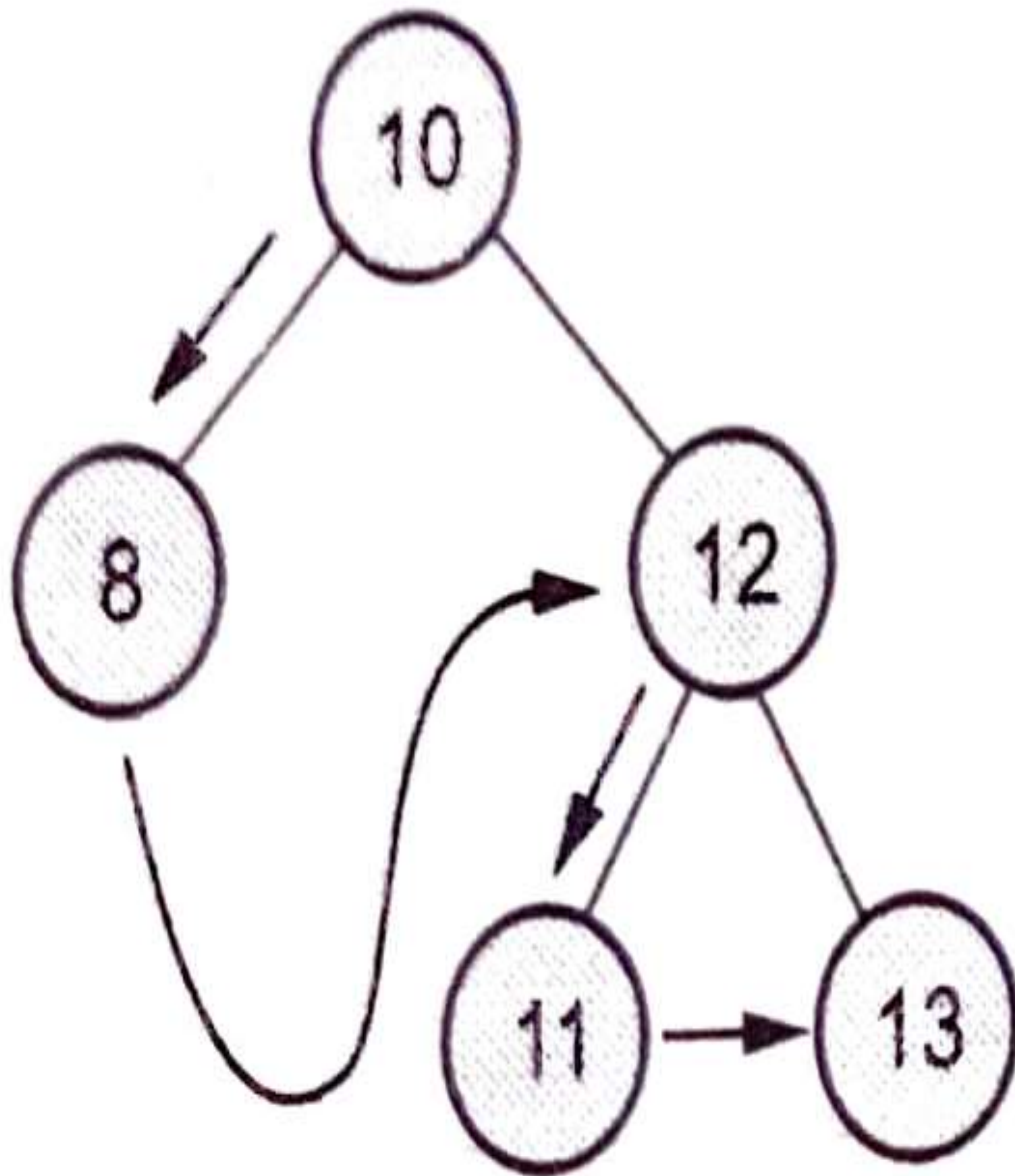
The data structure which is being used in DFS is stack. The process is similar to BFS algorithm. In DFS, the edges that leads to an unvisited node are called discovery edges while the edges that leads to an already visited node are called block edges.

DFS Algorithms

DFS is an algorithm for finding or traversing graphs or trees in depth-ward direction. The execution of the algorithm begins at the root node and explores each branch before backtracking. It uses a stack data structure to remember, to get the subsequent vertex, and to start a search, whenever a dead-end appears in any iteration.

The full form of DFS is Depth-first search.

DFS Algorithms



The DFS sequence is **10, 8, 12, 11, 13**

DFS Algorithms

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

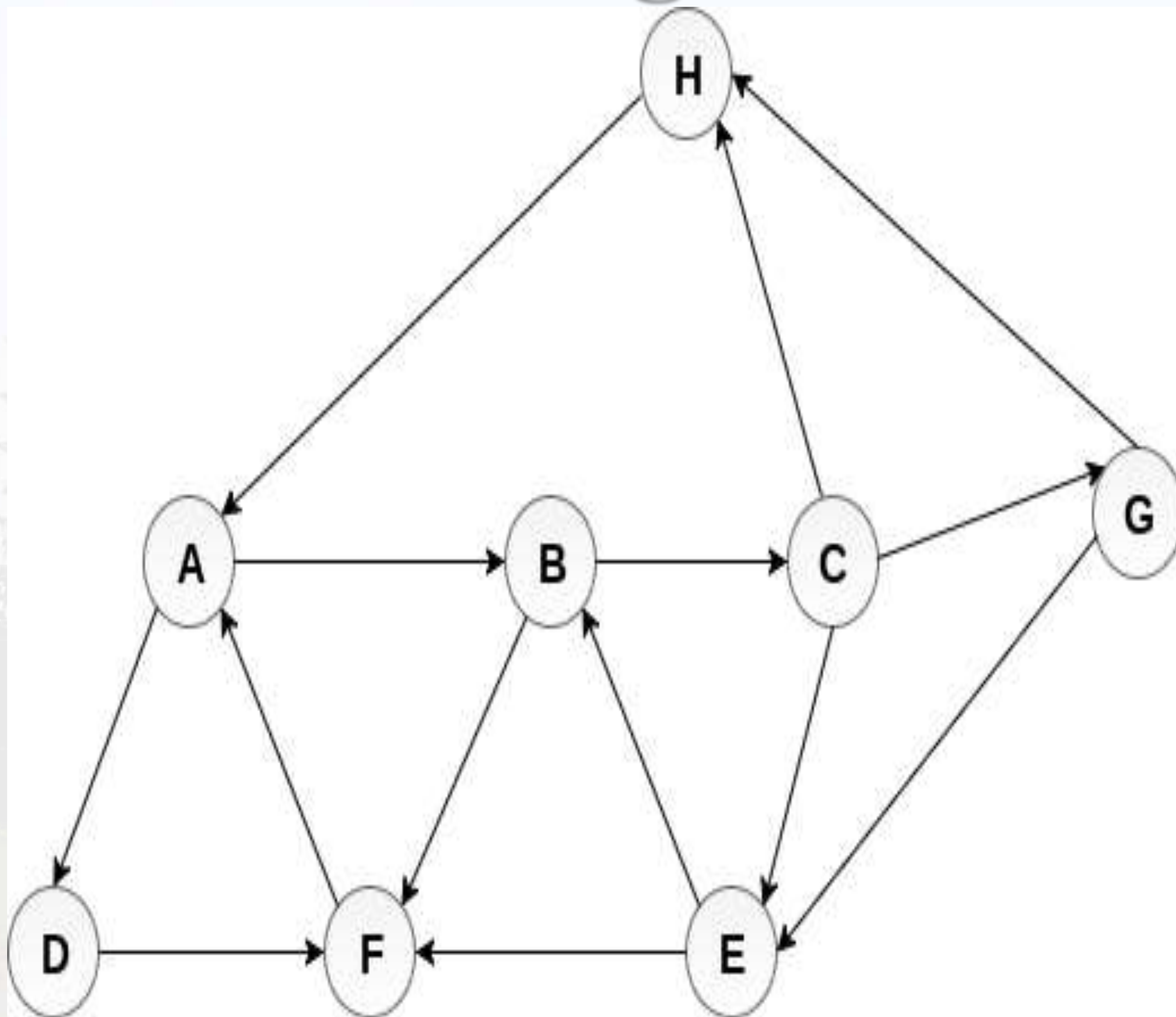
Step 3: Repeat Steps 4 and 5 until STACK is empty

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbours of N that are in the ready state (whose STATUS = 1) and set their
STATUS = 2 (waiting state)
[END OF LOOP]

Step 6: EXIT

DFS Algorithms Example



Adjacency Lists

A : B, D

B : C, F

C : E, G, H

G : E, H

E : B, F

F : A

D : F

H : A

The printing sequence of the graph will be :

H → A → D → F → B → C → G → E

Complexity of DFS Algorithms

Complexity of Depth First Search

The time complexity of the DFS algorithm is

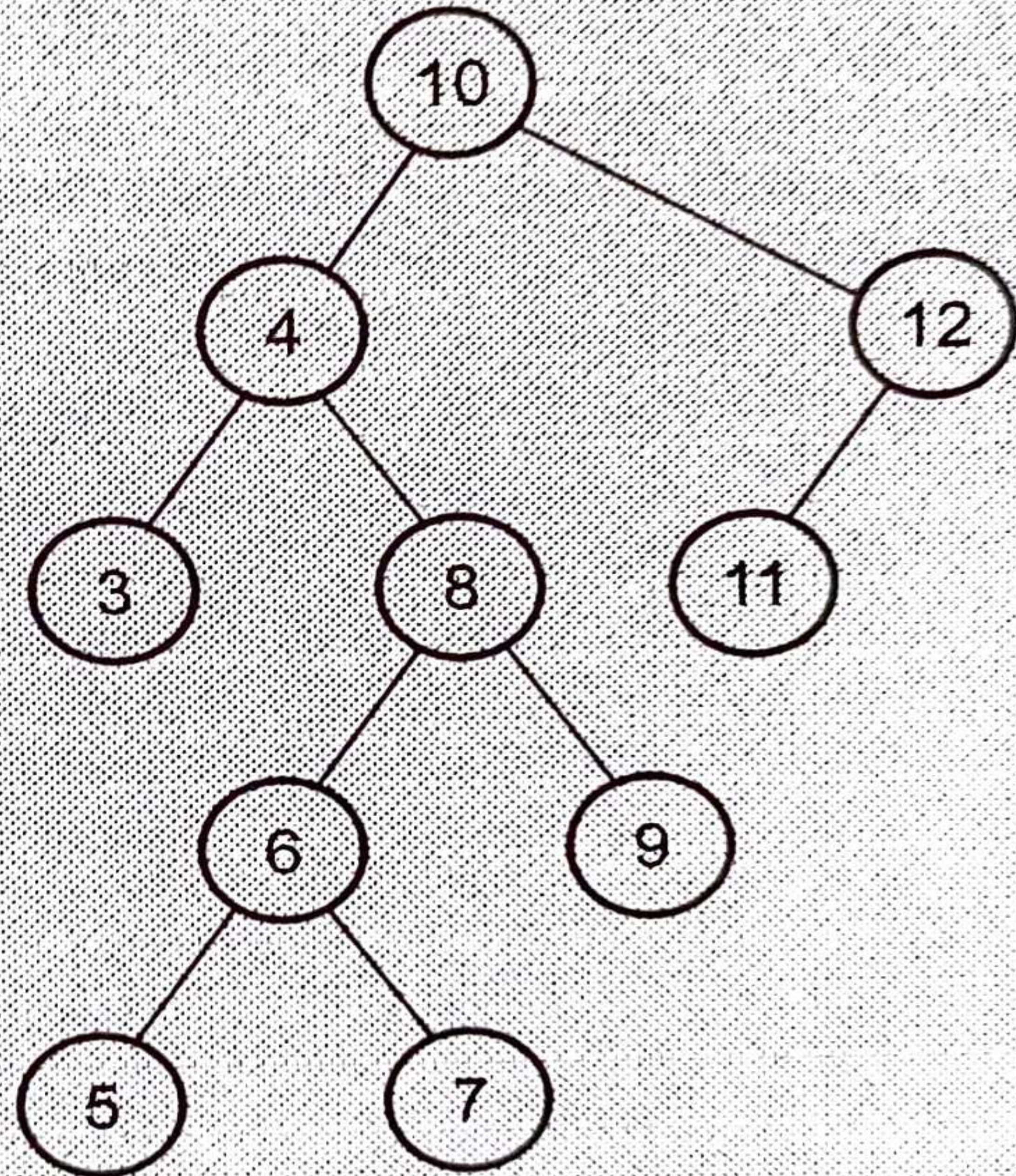
represented in the form of O(V + E), where V is the

number of nodes and E is the number of edges.

The space complexity of the algorithm is O(V).

Example of DFS Algorithms

Example 2.7.1 *Consider following tree and obtain its DFS sequence.*



10, 4, 3, 8, 6, 5, 7, 9, 12, 11

Advantages and Disadvantages of DFS

- **Advantages :**

1. Depth-first search on a binary tree generally requires less memory than breadth-first.
2. Depth-first search can be easily implemented with recursion..

- **Disadvantages :**

A DFS doesn't necessarily find the shortest path to a node, while breadth-first search does.

Application of DFS Algorithms

1. For finding the path
2. To test if the graph is bipartite
3. For finding the strongly connected components of a graph
4. For detecting cycles in a graph

BFS Vs DFS

sn	BFS	DFS
1	BFS finds the shortest path to the destination.	DFS goes to the bottom of a subtree, then backtracks.
2	The full form of BFS is Breadth-First Search.	The full form of DFS is Depth First Search.
3	It uses a **queue** to keep track of the next location to visit.	It uses a **stack** to keep track of the next location to visit.
4	BFS traverses according to **tree level.**	DFS traverses according to **tree depth.**
5	It is implemented using **FIFO list**.	It is implemented using **LIFO list.**
6	It requires **more memory** as compare to DFS.	It requires **less memory** as compare to BFS.

BFS Vs DFS

sn	BFS	DFS
7	There is no need of backtracking in BFS.	There is a need of backtracking in DFS.
8	You can never be trapped into finite loops.	You can be trapped into infinite loops.
9	If you do not find any goal, you may need to expand many nodes before the solution is found.	If you do not find any goal, the leaf node backtracking may occur.

Applications of Graph

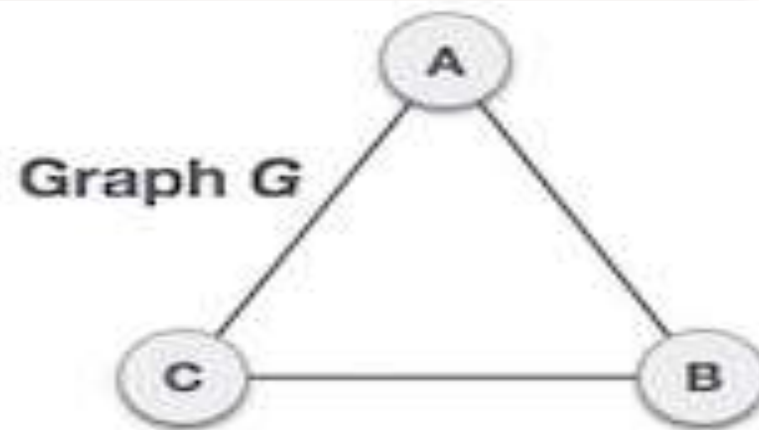
1. In **Computer science** graphs are used to represent the flow of computation.
2. **Google maps** uses graphs for building transportation systems.
3. **In Facebook,** users are considered to be the vertices and if they are friends then there is an edge running between them.
4. Graphs are used to define the flow of computation.
5. Graphs are used to represent networks of communication.
6. Graphs are used to represent data organization.
7. Graph theory is used to find shortest path in road or a network.

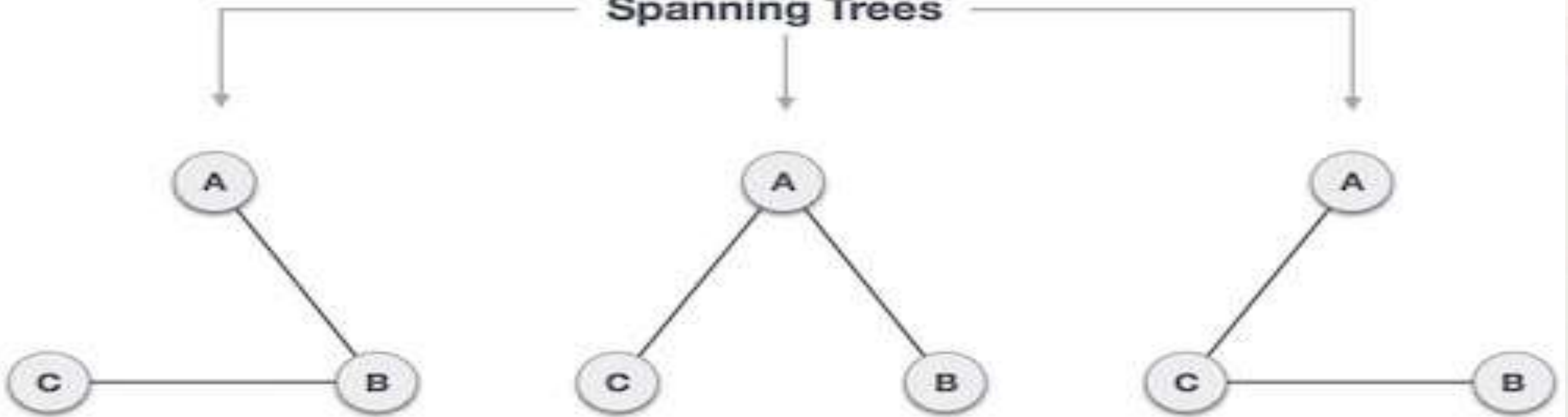
Tree Vs Graph

Basis For Comparision	TREE	GRAPH
Path	Only one between Two Vertices	More than one path allowed.
Root node	It Has Exactly one root node	Graph may or may not have root node
Loops	No Loops are Permited	No Loop are Permited if loops exist then we call it Multigraph.
Complexity	Less Complex	More Complex comparetively
Traversal Technique	Pre-order, in-order, post-order, level-order	Breadth First Search, Depth First Search
Number of Edges	N-1	Not Define
Model Type	Hierarchical	Network

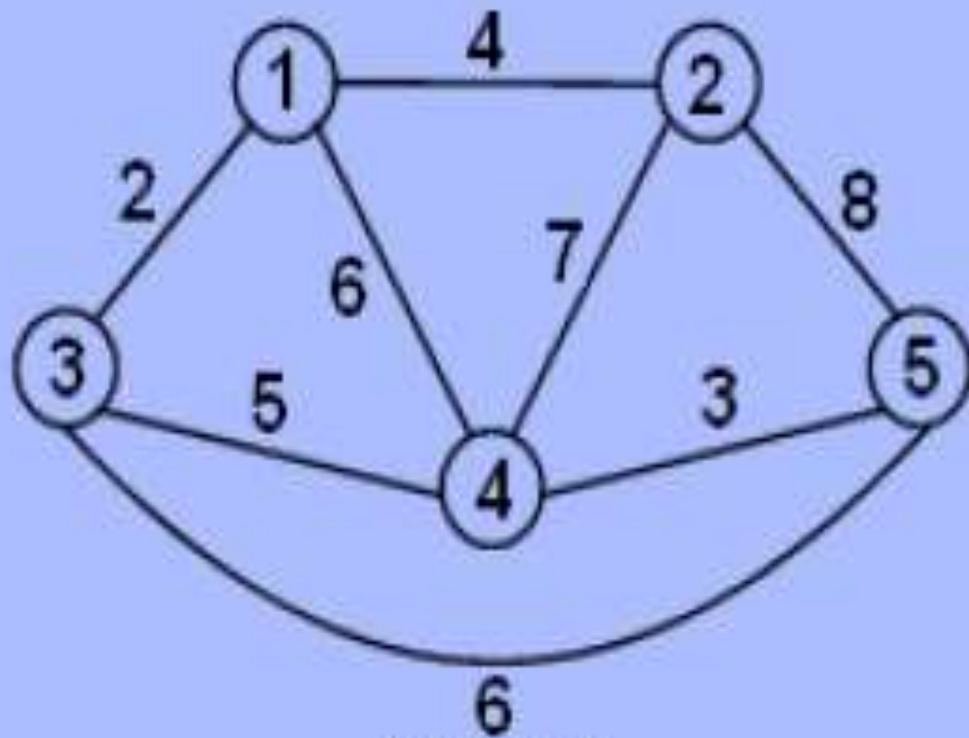
Spanning Tree of Graph

A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected..

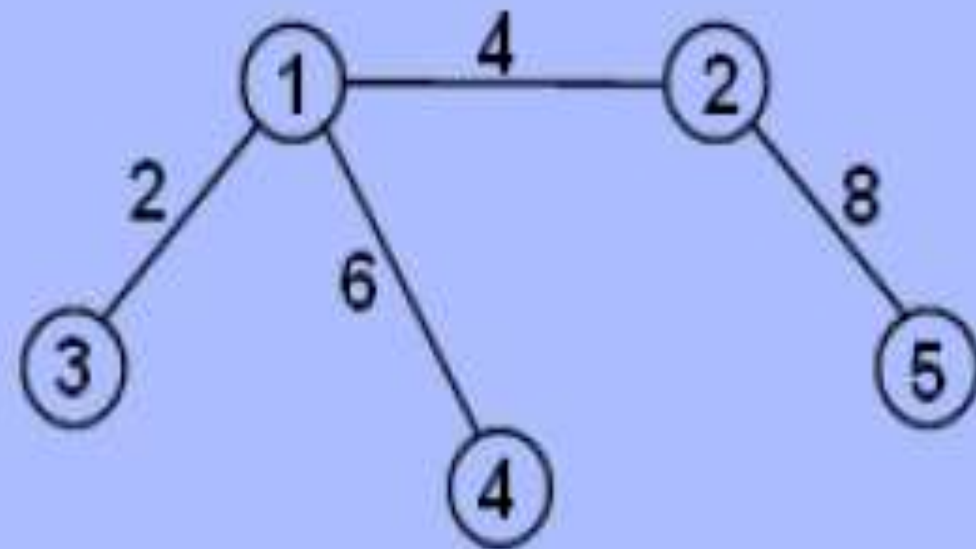




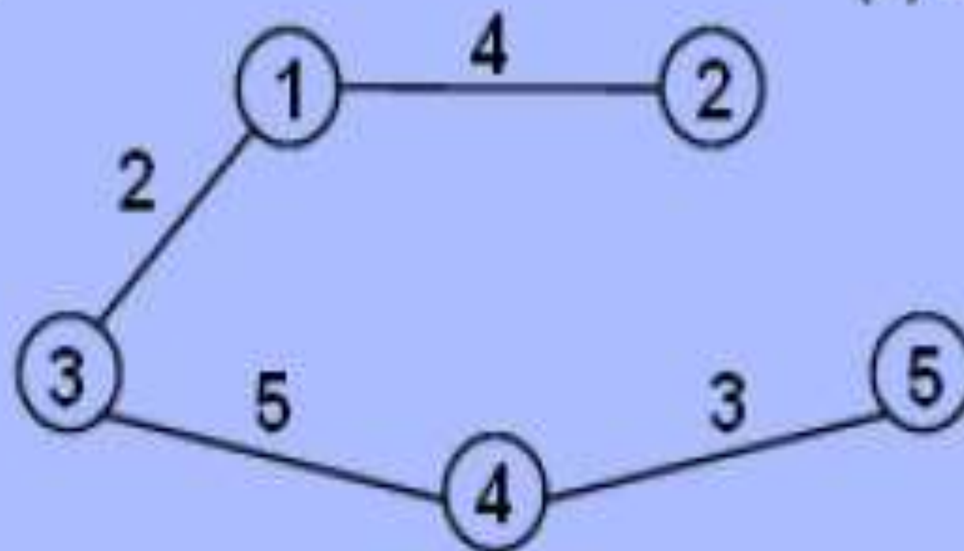
Minimum Spanning Tree



(a) Graph



(b) Spanning Tree



(c) Minimum Spanning Tree

Spanning Tree of Graph

- **Spanning Tree Applications**

1. Computer Network Routing Protocol
2. Cluster Analysis
3. Civil Network Planning

- **Minimum Spanning tree Applications**

1. To find paths in the map
2. To design networks like telecommunication networks, water supply networks, and electrical grids.

GREEDY STRATEGIES

Greedy algorithm :

An algorithm is designed to achieve **optimum solution** for a given problem. In greedy algorithm approach, decisions are made from the given solution domain. As being greedy, **the closest solution that seems to provide an optimum solution is chosen.**

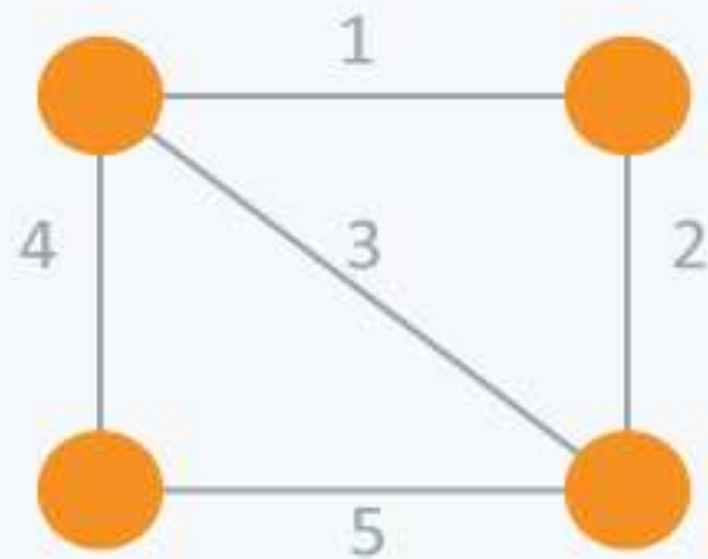
Example of greedy strategy :

1. Travelling Salesman Problem
2. Prim's Minimal Spanning Tree Algorithm
3. Kruskal's Minimal Spanning Tree Algorithm
4. Dijkstra's Minimal Spanning Tree Algorithm
5. Knapsack Problem
6. Job Scheduling Problem

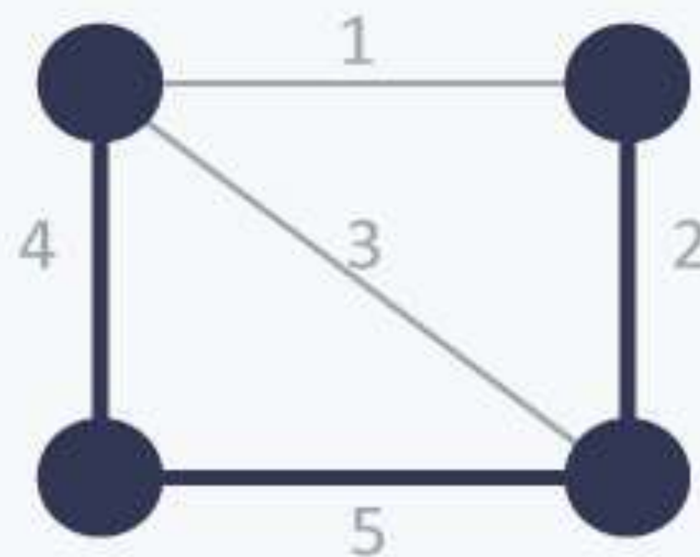
GREEDY STRATEGIES

1. Minimum Spanning tree (Prims or Kruskal's algorithms)

The cost of the spanning tree is the sum of the weights of all the edges in the tree. There can be many spanning trees. Minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees. There also can be many minimum spanning trees.

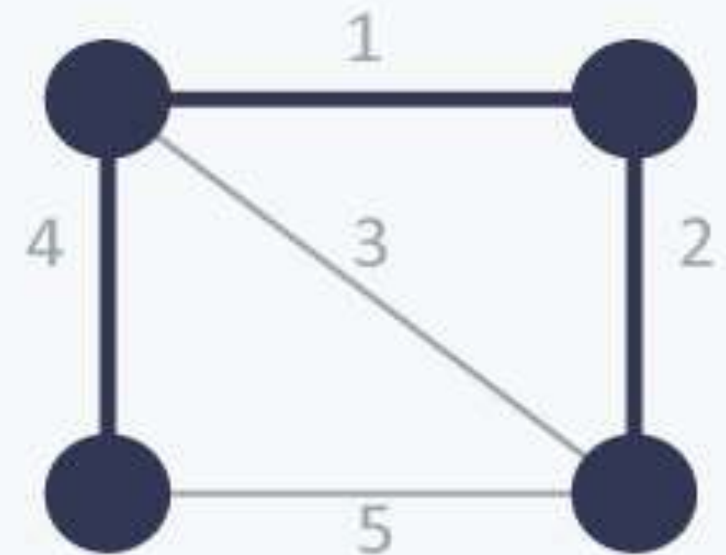


Undirected Graph



Spanning Tree

Cost = 11(=4+5+2)



Minimum Spanning Tree

Cost = 7(=4+1+2)

GREEDY STRATEGIES

1. **Kruskal's algorithms :**

Kruskal's Algorithm builds the spanning tree by adding edges one by one into a growing spanning tree. Kruskal's algorithm follows greedy approach as in each iteration it finds an edge which has least weight and add it to the growing spanning tree.

Algorithm Steps :

1. Sort the graph edges with respect to their weights.
2. Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
3. Only add edges which doesn't form a cycle , edges which connect only disconnected components.

GREEDY STRATEGIES

1. **Kruskal's algorithms :**

- Kruskal's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which
- form a tree that includes every vertex
- has the minimum sum of weights among all the trees that can be formed from the graph
- **Kruskal's Algorithm Complexity**
- The time complexity Of Kruskal's Algorithm is: O(E log E).

GREEDY STRATEGIES

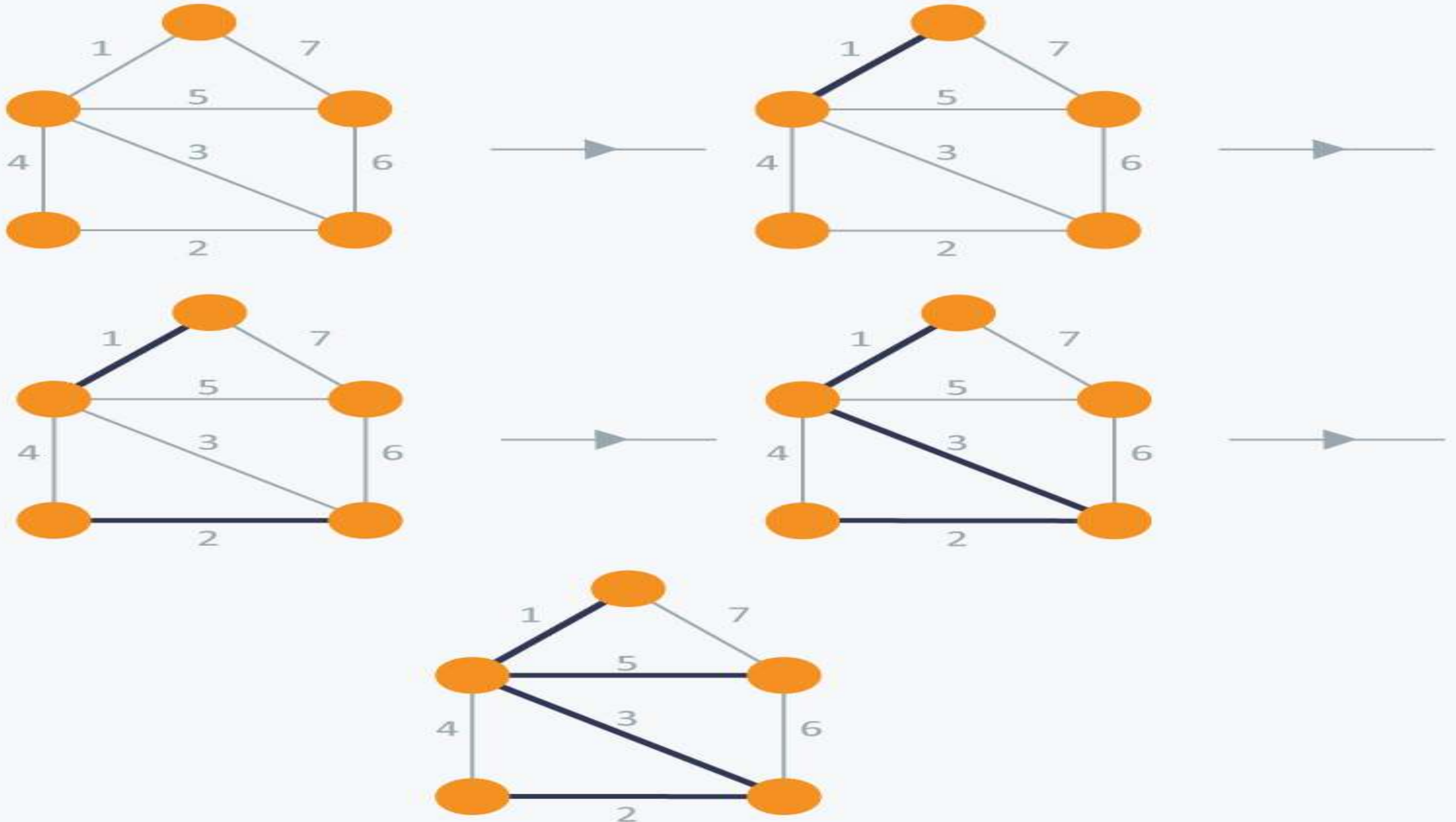
1. Kruskal’s algorithms :

The steps for implementing Kruskal's algorithm are as follows:

1. Sort all the edges from low weight to high
2. Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
3. Keep adding edges until we reach all vertices.

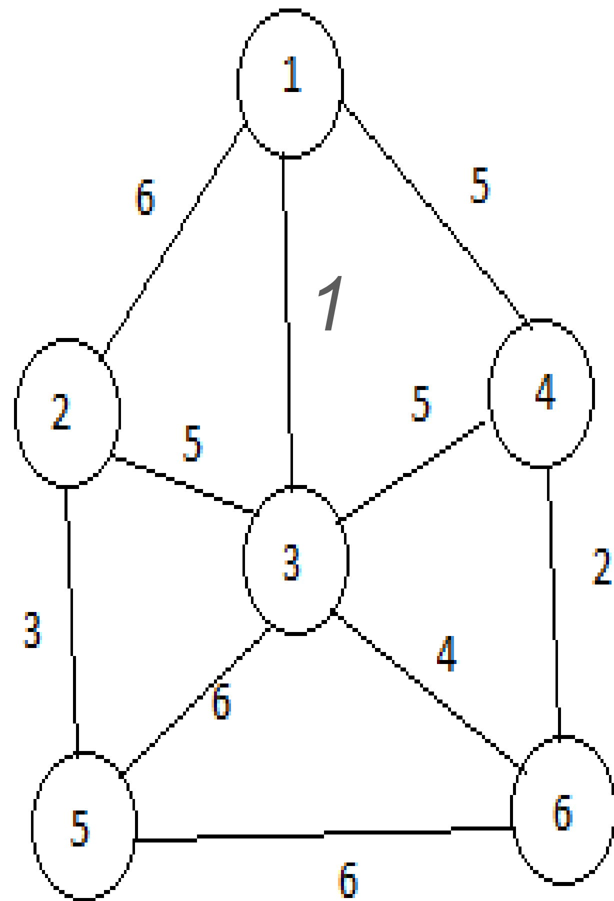
GREEDY STRATEGIES

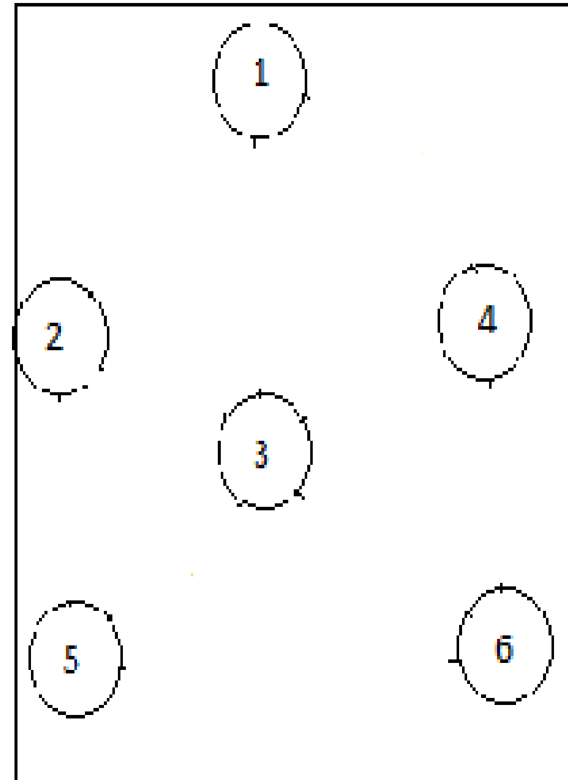
2. **Kruskal's algorithms : Example**

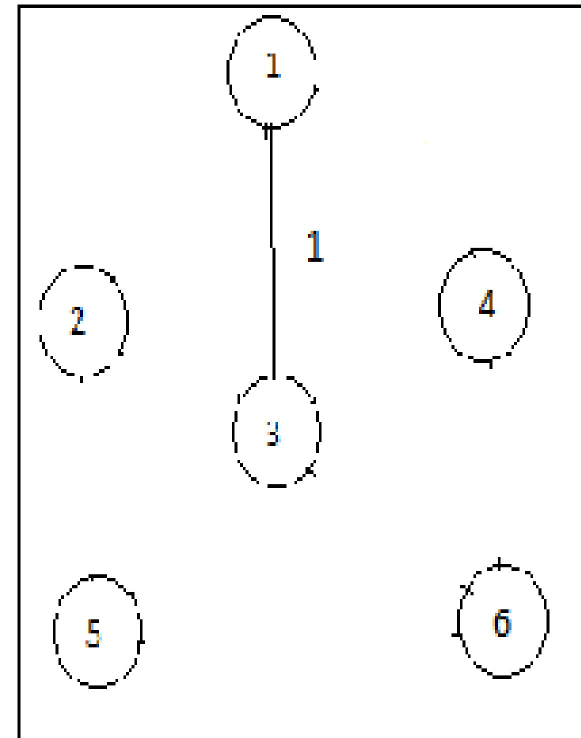


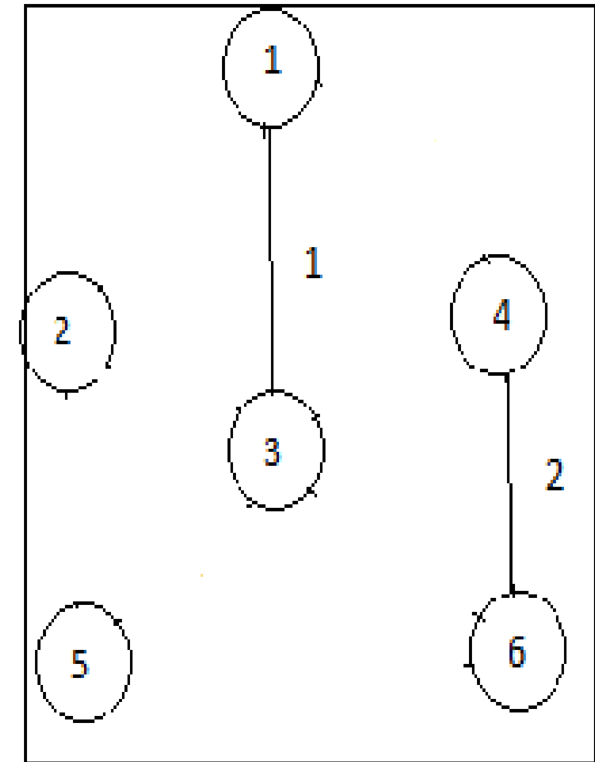
GREEDY STRATEGIES

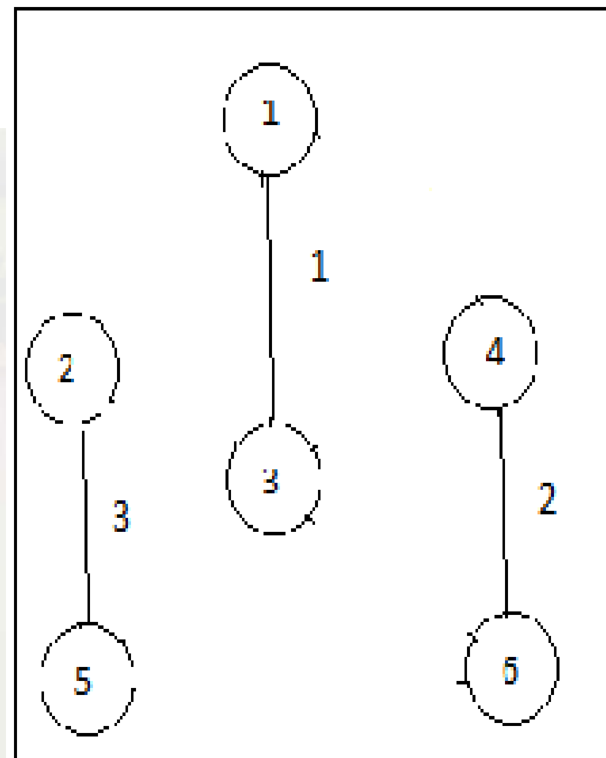
2. Kruskal's algorithms : Example

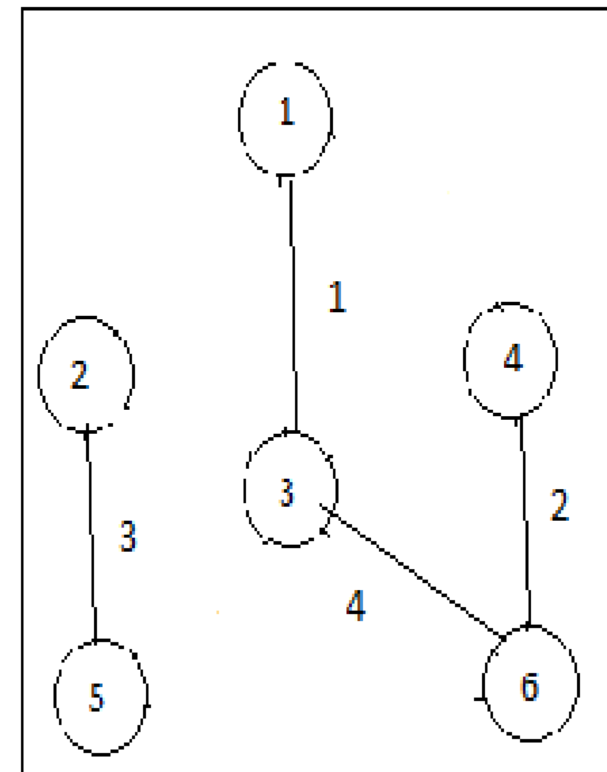


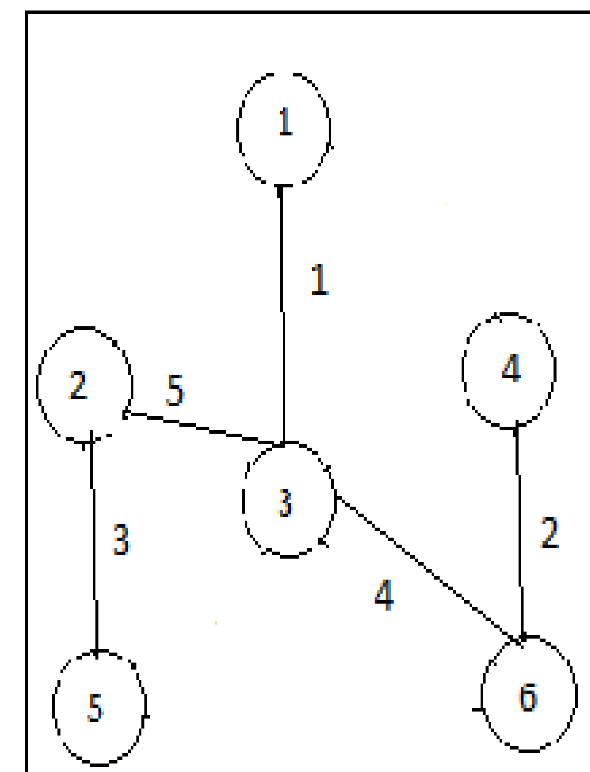






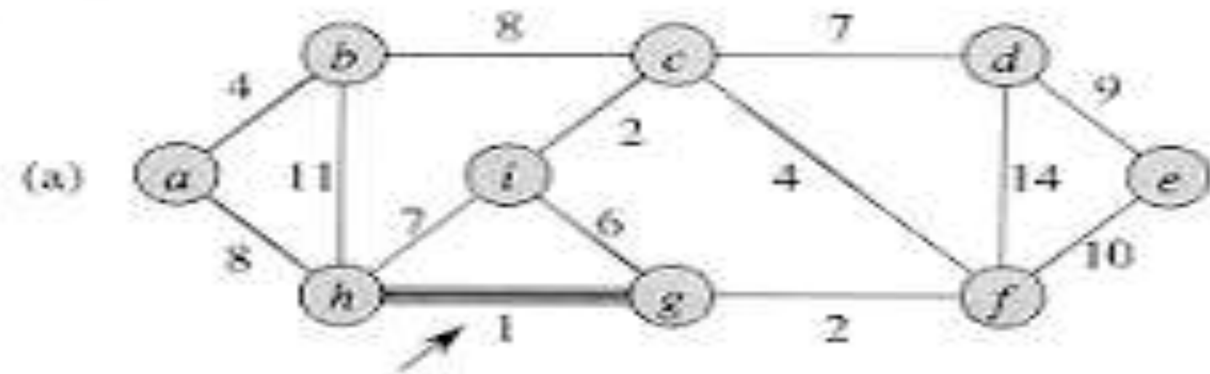


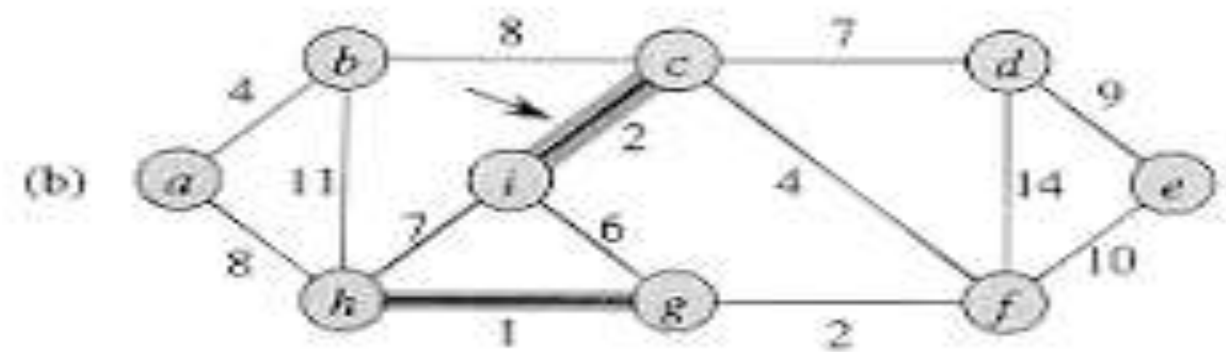


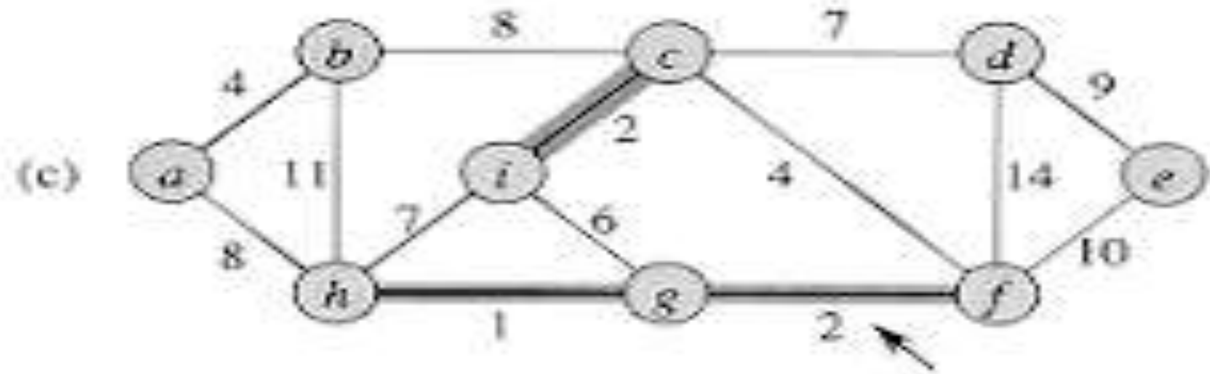


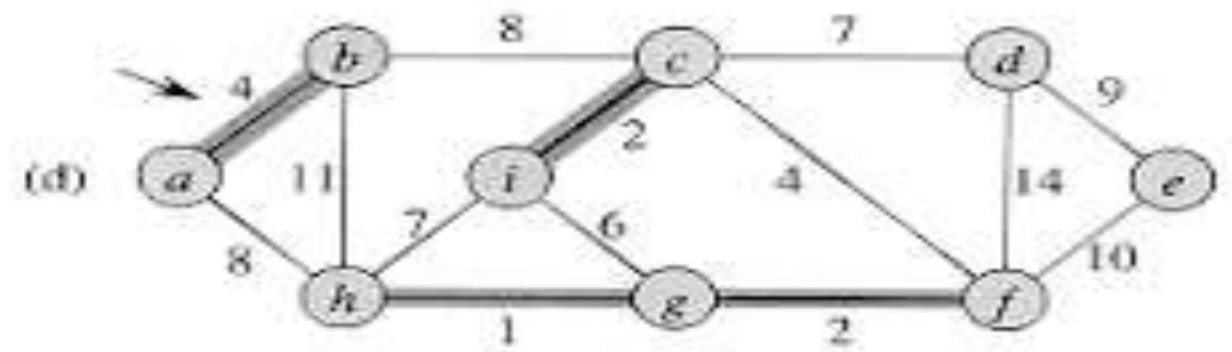
GREEDY STRATEGIES

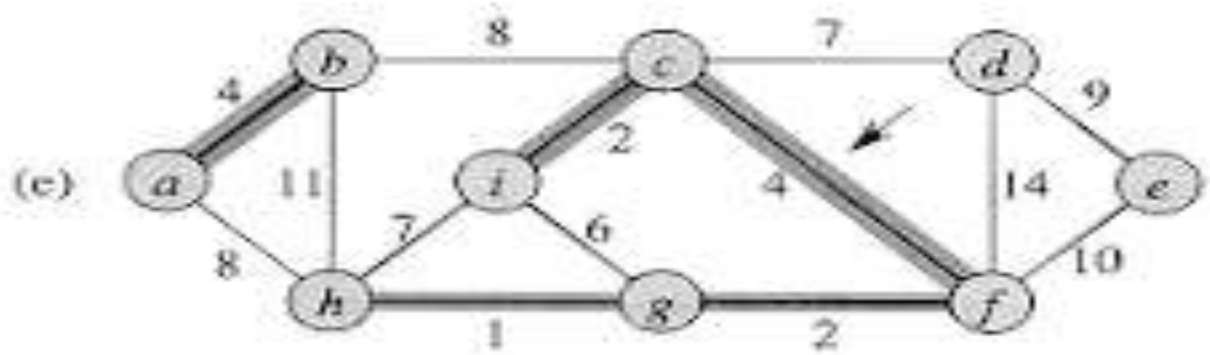
2. **Kruskal's algorithms : Example**

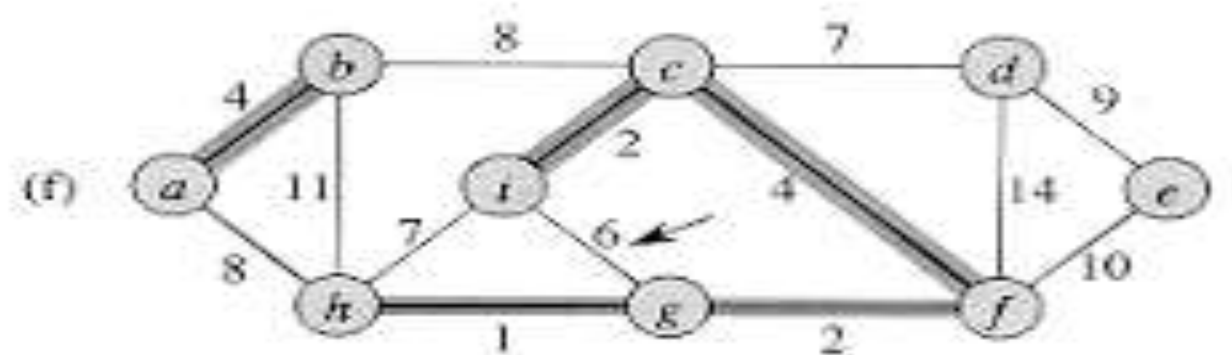


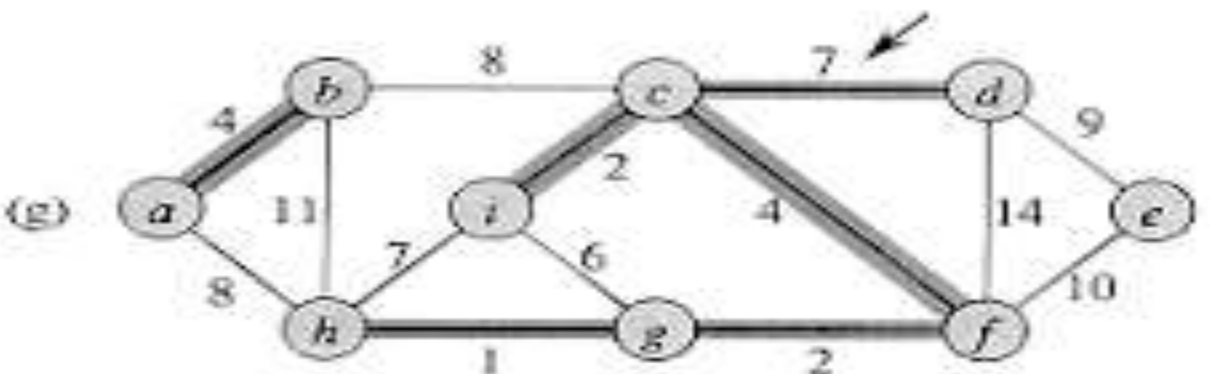


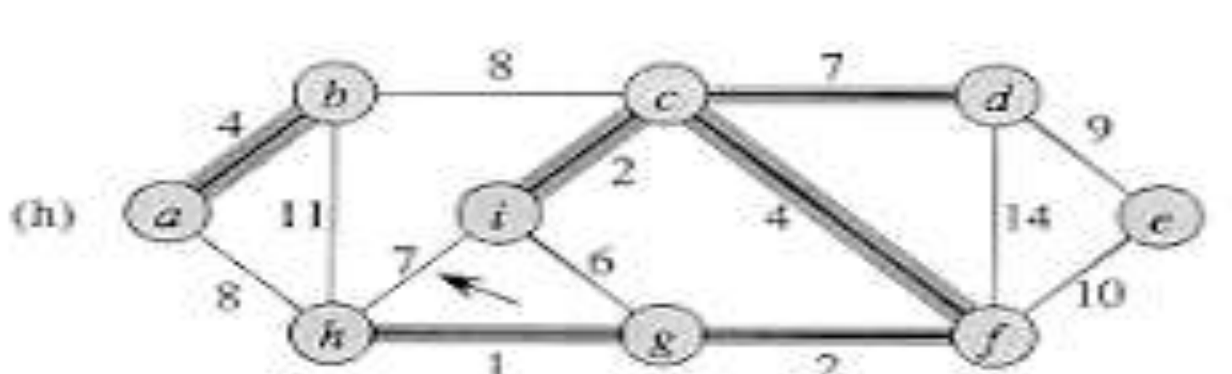






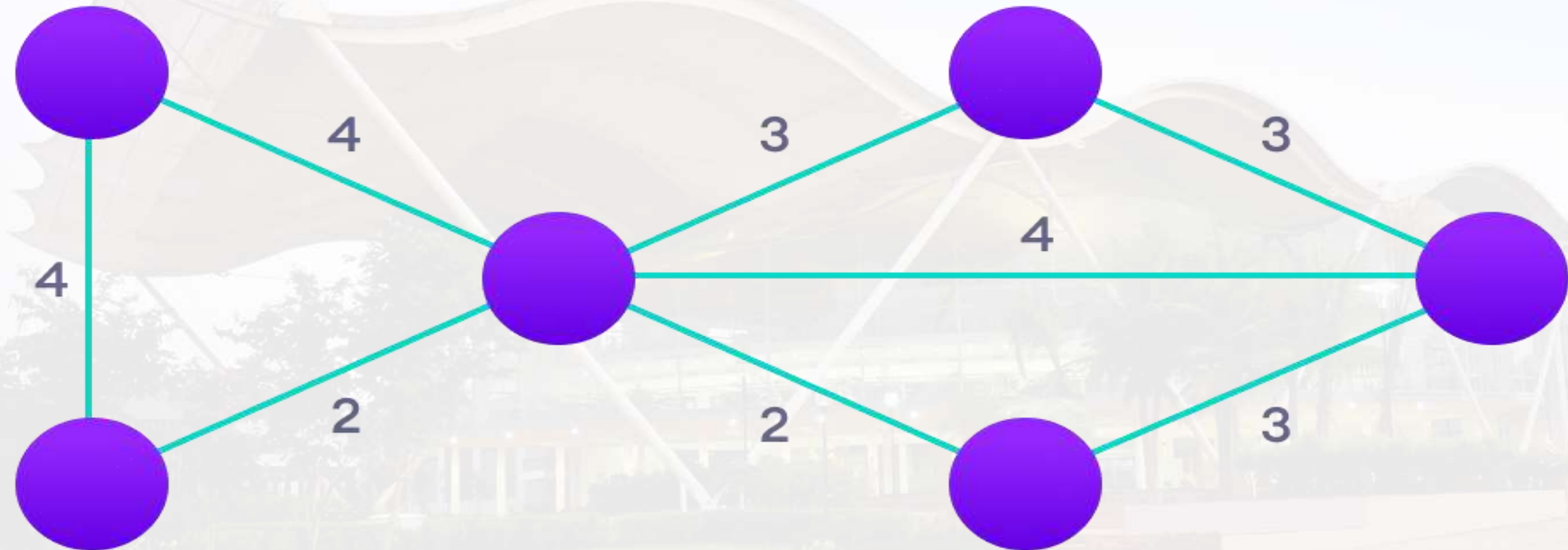






GREEDY STRATEGIES

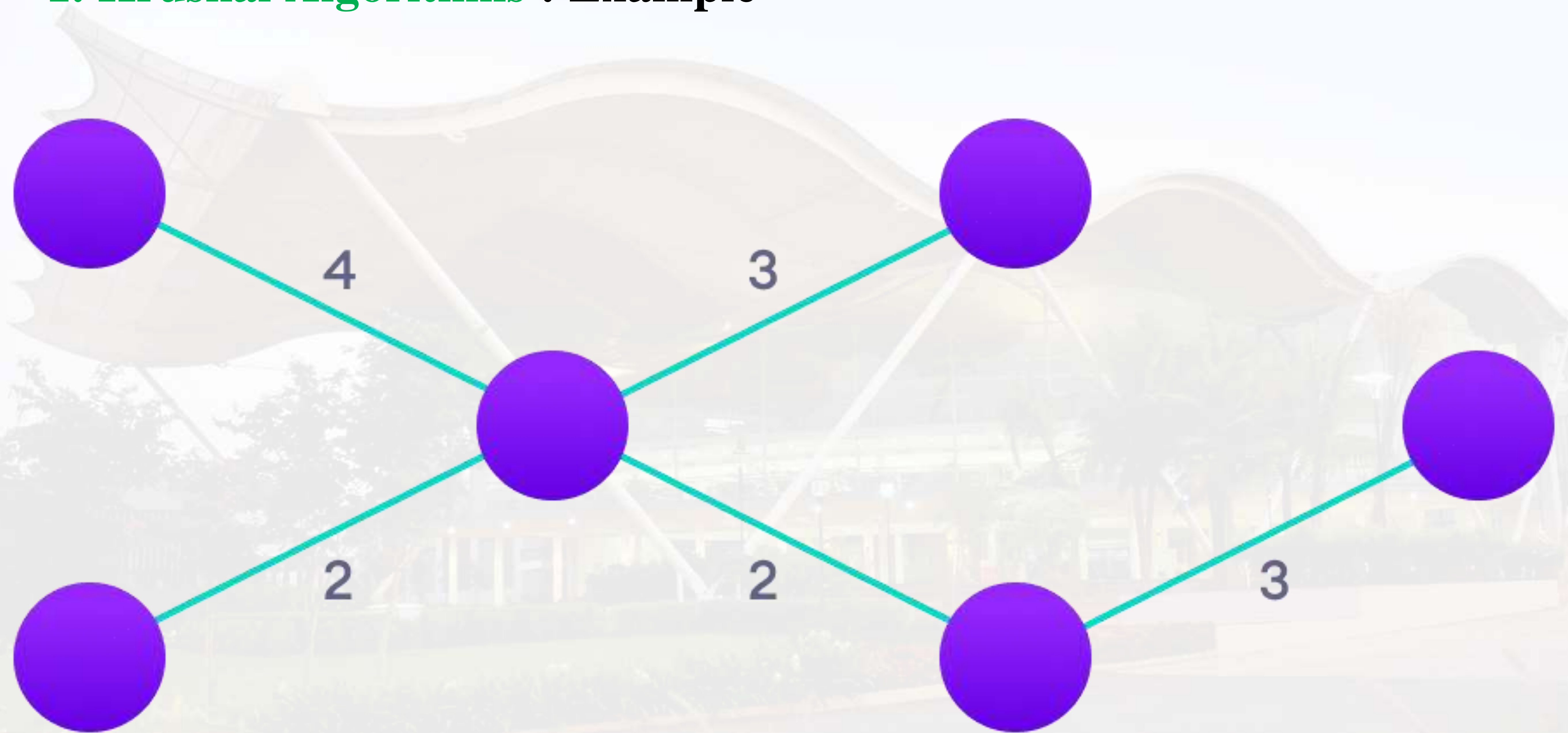
2. **Kruskal Algorithms : Example**



Step: 1

GREEDY STRATEGIES

1. Kruskal Algorithms : Example



Step: 6

GREEDY STRATEGIES

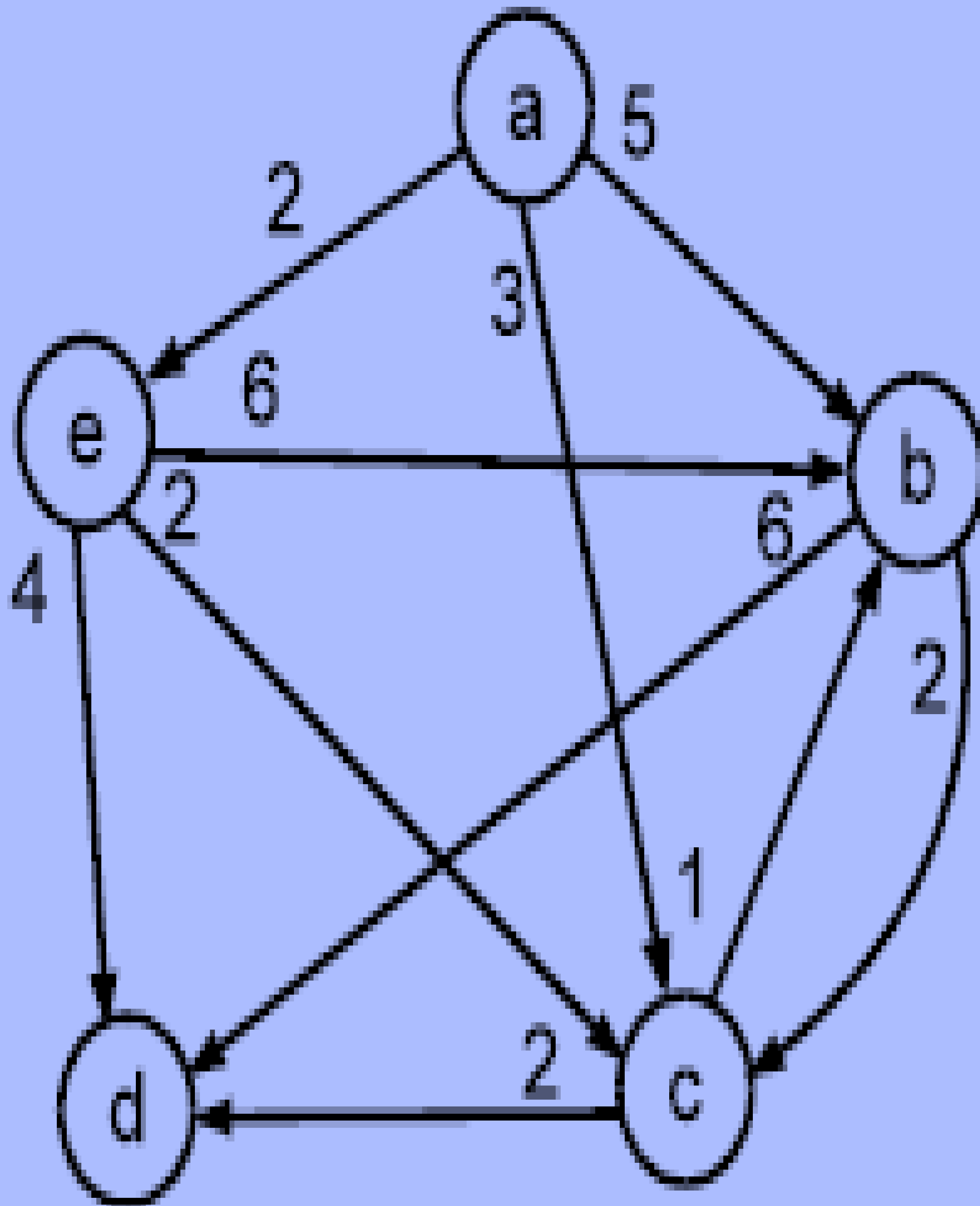
1. Kruskal Algorithms :

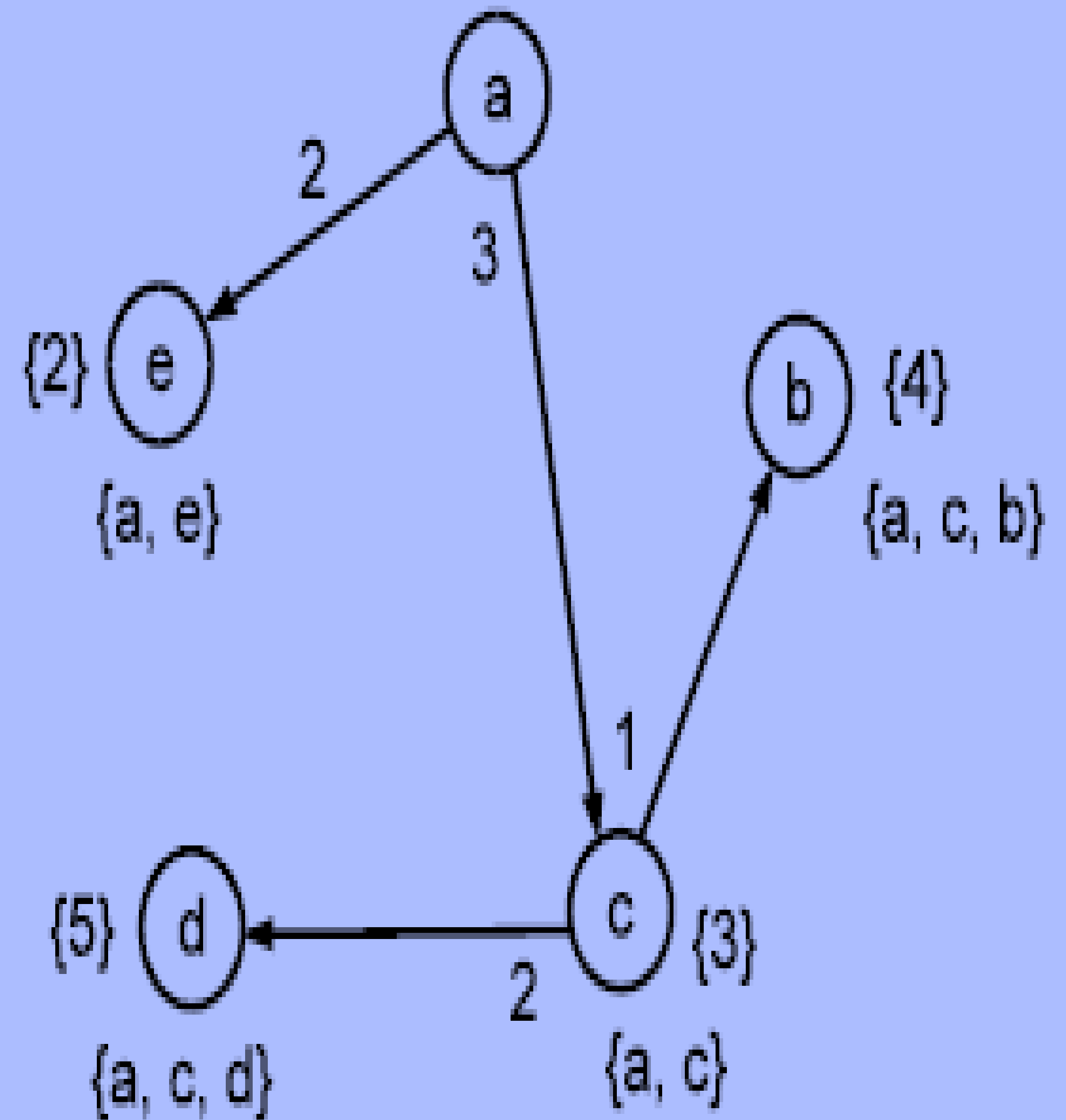
Kruskal's Algorithm Applications

1. In order to layout electrical wiring
2. In computer network (LAN connection)

GREEDY STRATEGIES

1. Kruskal Algorithms :

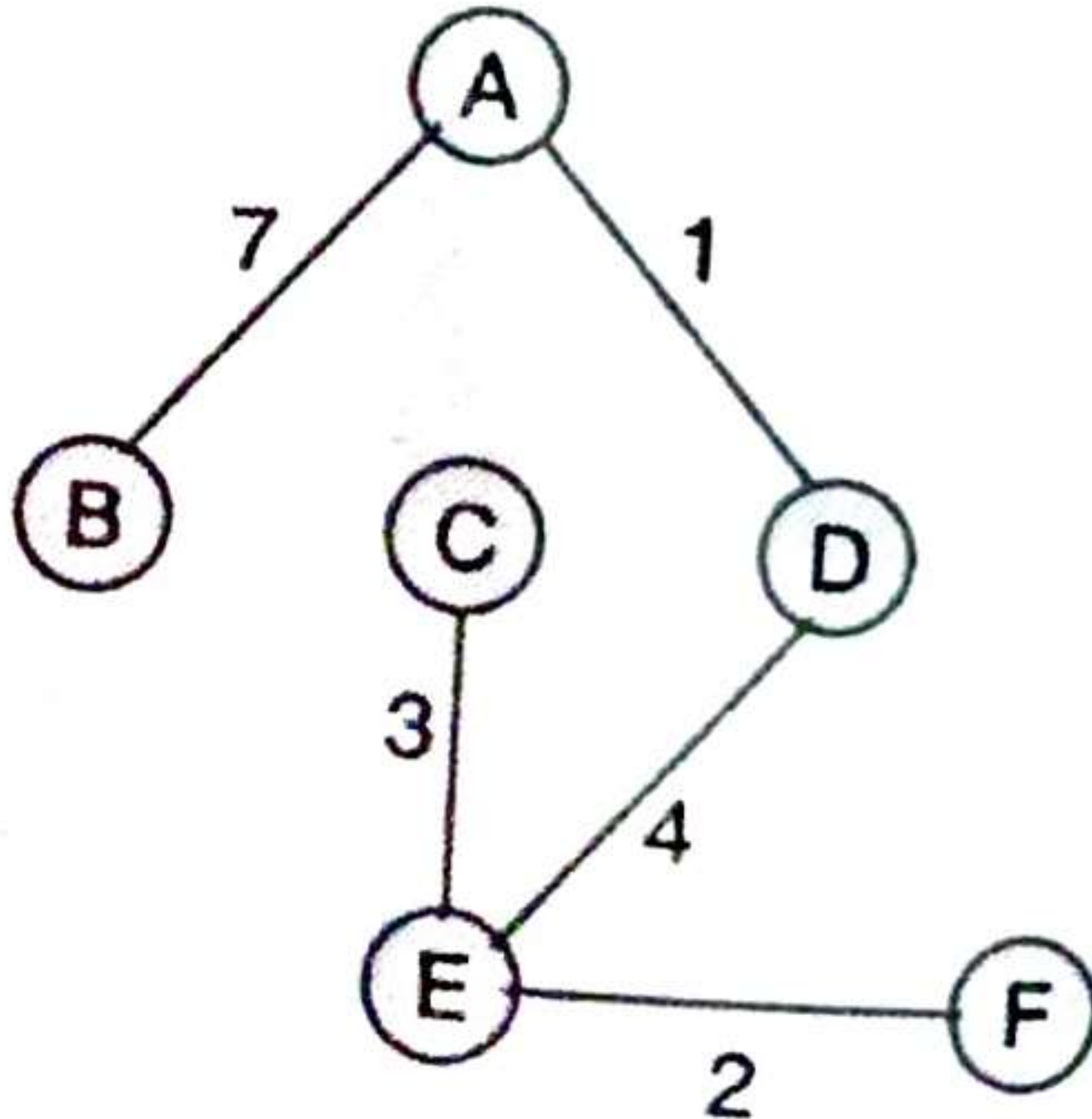




GREEDY STRATEGIES

1. Kruskal Algorithms :

Find the minimum spanning tree for the ... graph using
Kruskal's alg...



GREEDY STRATEGIES

2. **Prims algorithm:** Prim's Algorithm also use Greedy approach to find the minimum spanning tree. In Prim's Algorithm we grow the spanning tree from a starting position. Unlike an **edge** in Kruskal's, we add **vertex** to the growing spanning tree in Prim's.

Algorithm Steps:

1. Initialize the minimum spanning tree with a vertex chosen at random.

2. Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree.

3. Keep repeating **step** 2 until we get a minimum spanning tree.

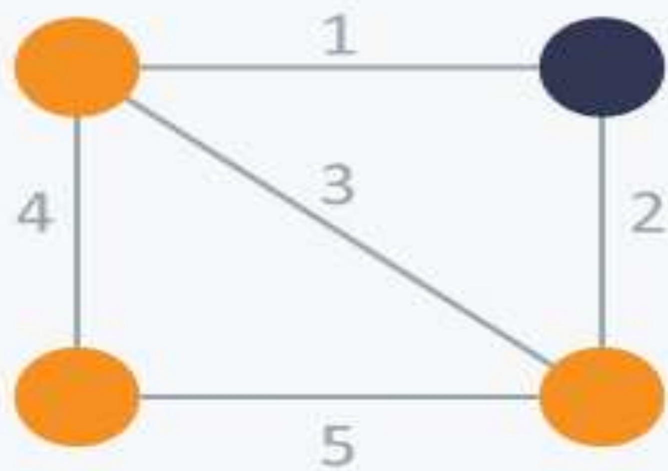
Prim's Algorithm Complexity

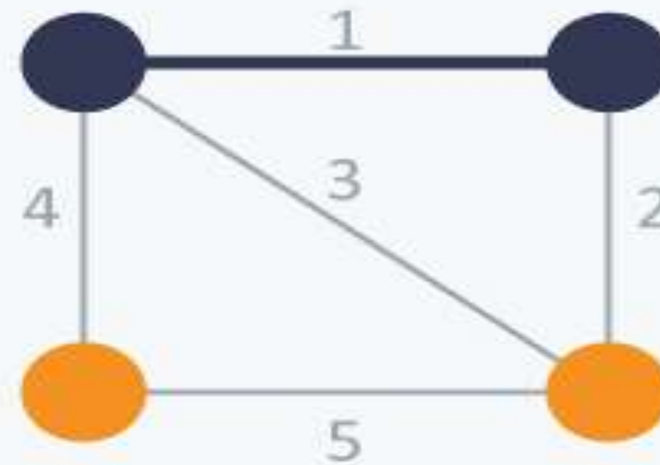
The time complexity of Prim's algorithm is **O(E log V).**

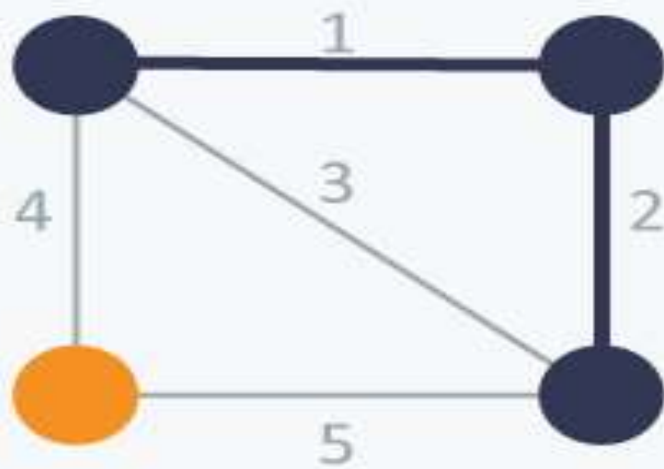
GREEDY STRATEGIES

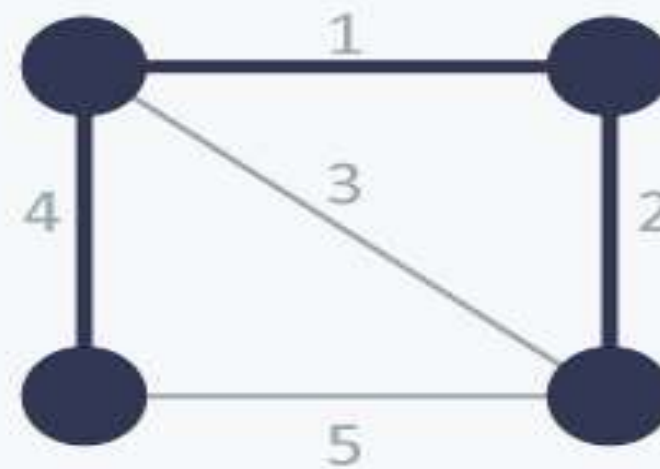
2. **Prims algorithm: Example**

Prim's Algorithm



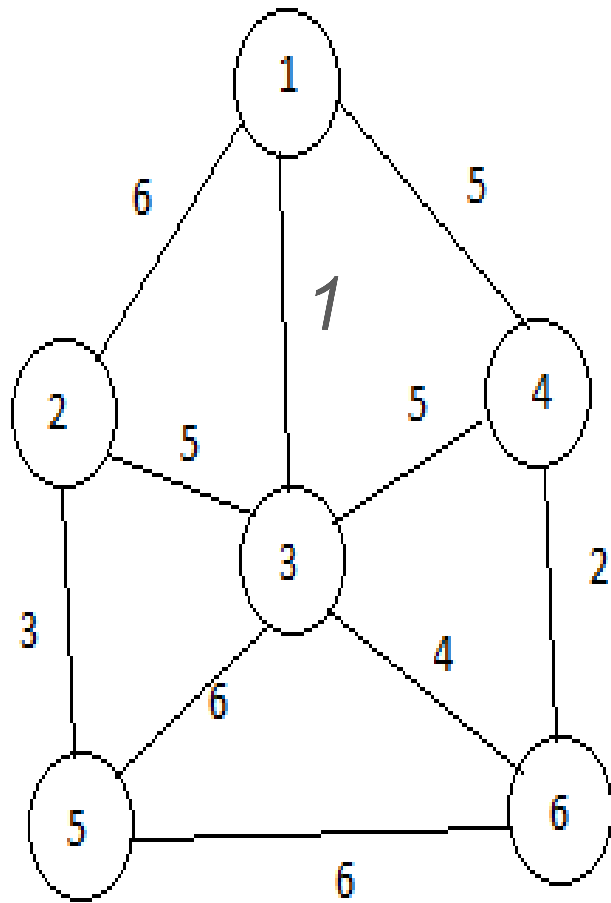


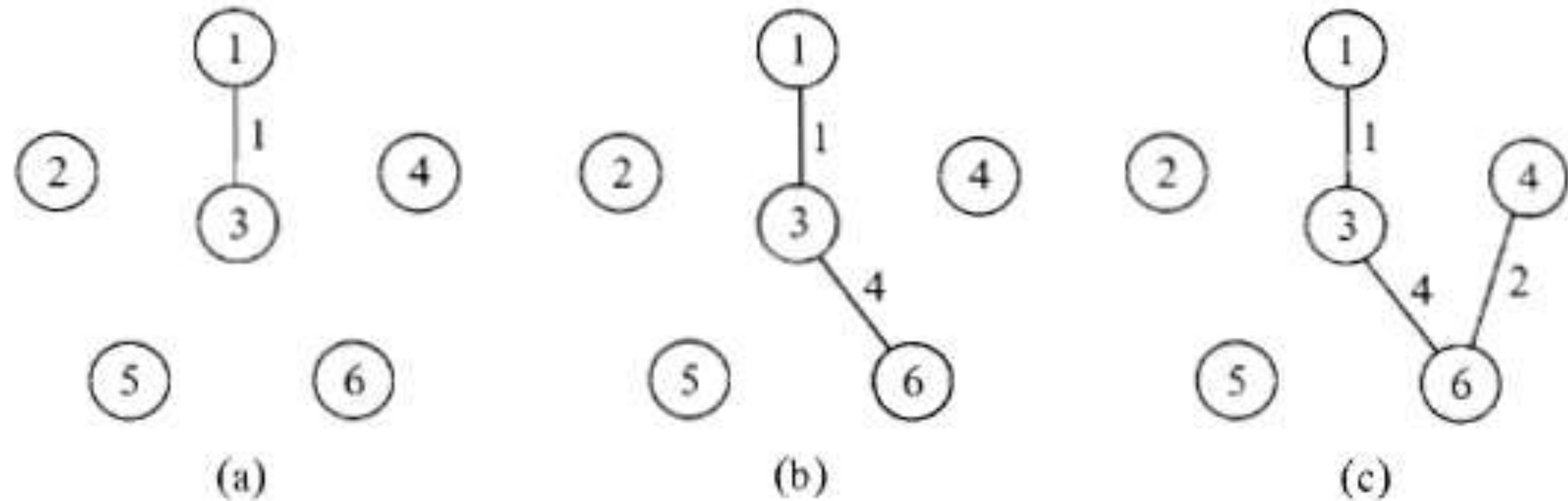


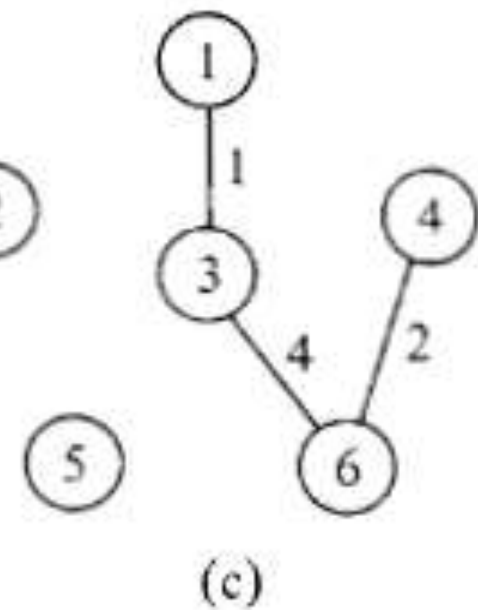


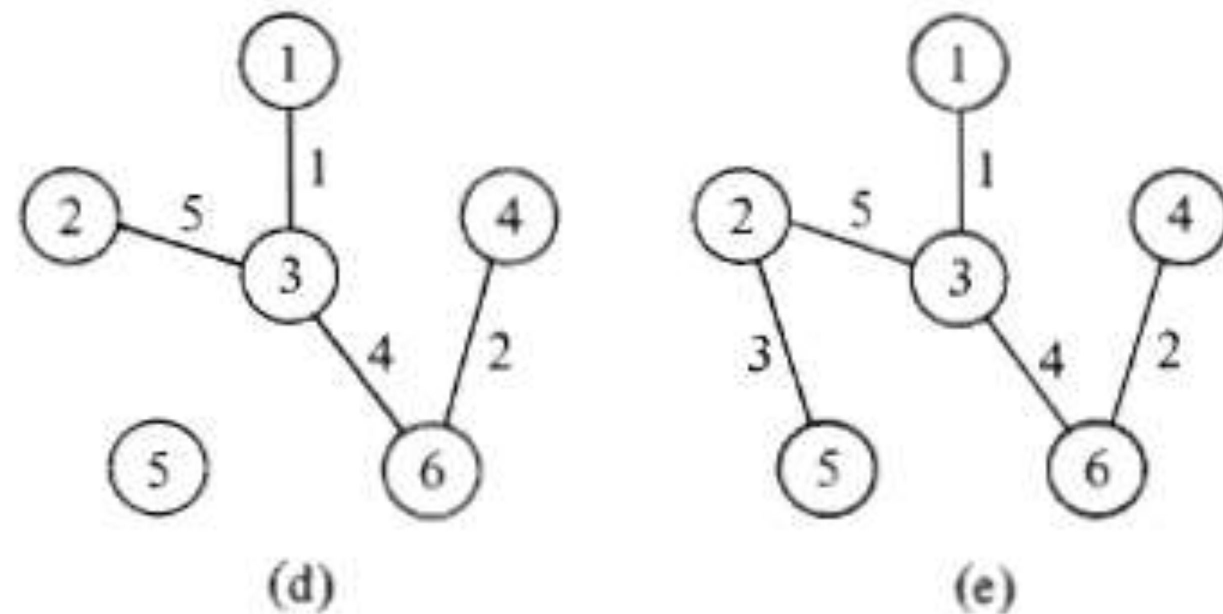
GREEDY STRATEGIES

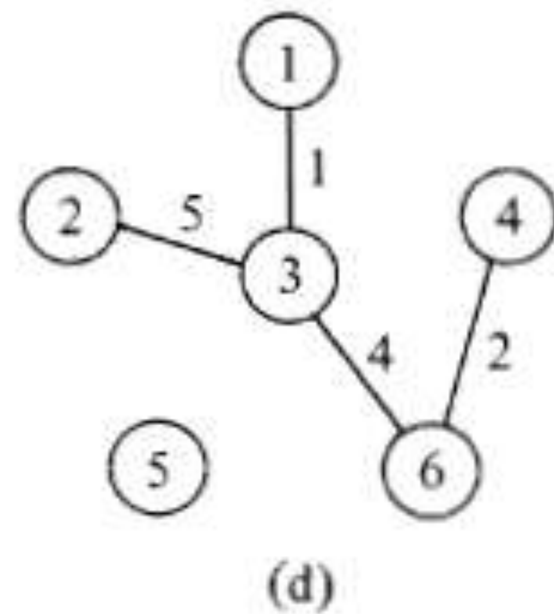
1. Prims' algorithms : Example











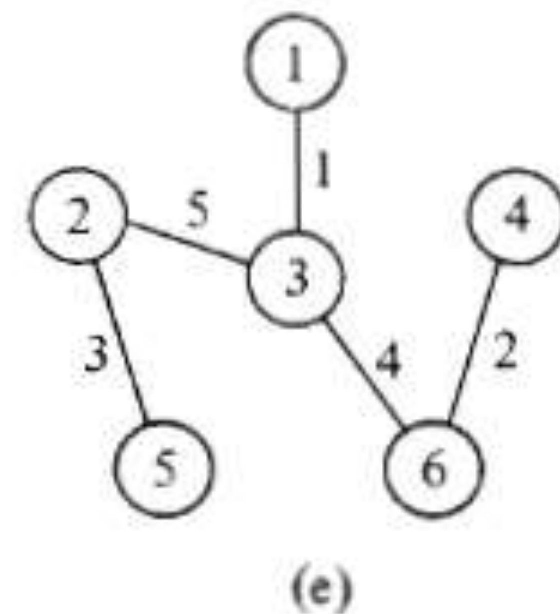
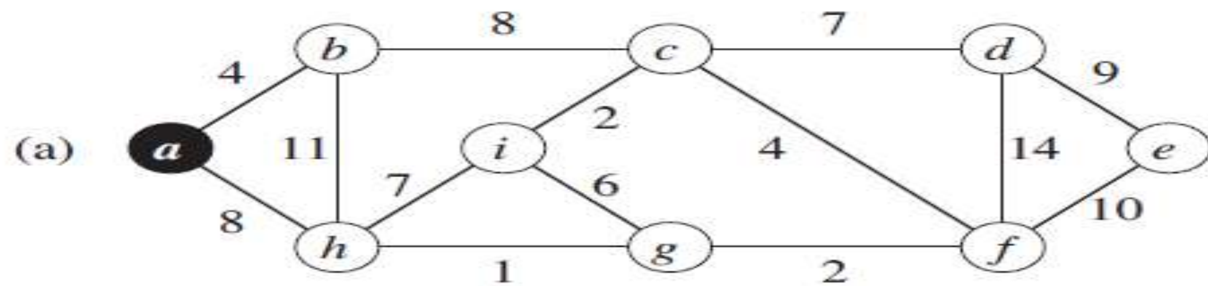
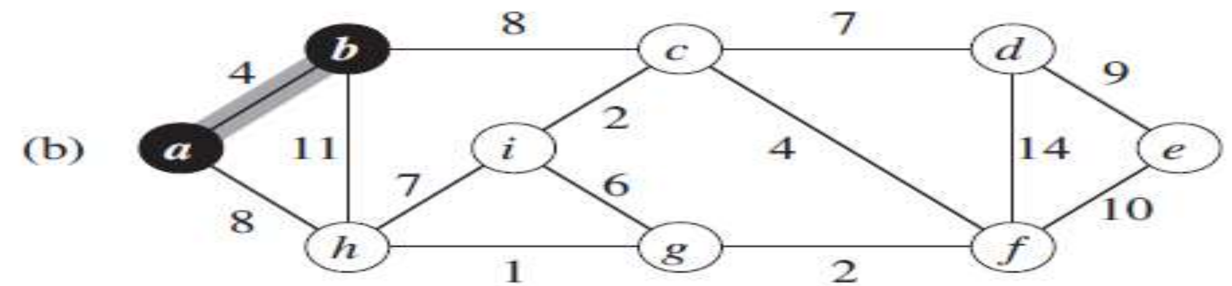


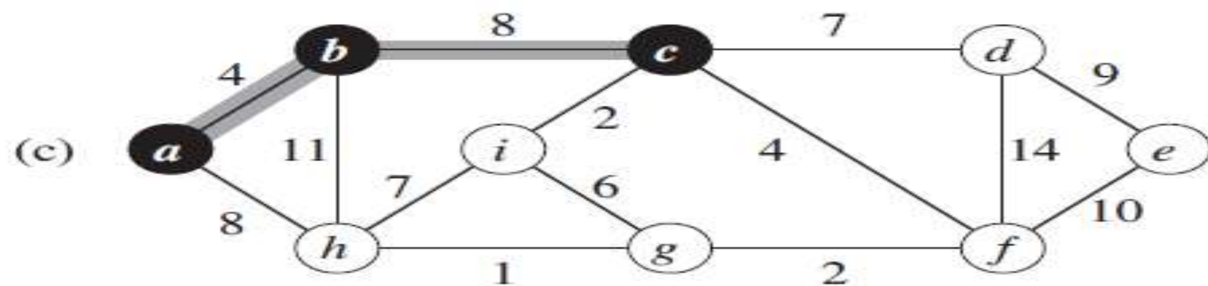
Fig. 7.7. Sequences of edges added by Prim's algorithm.

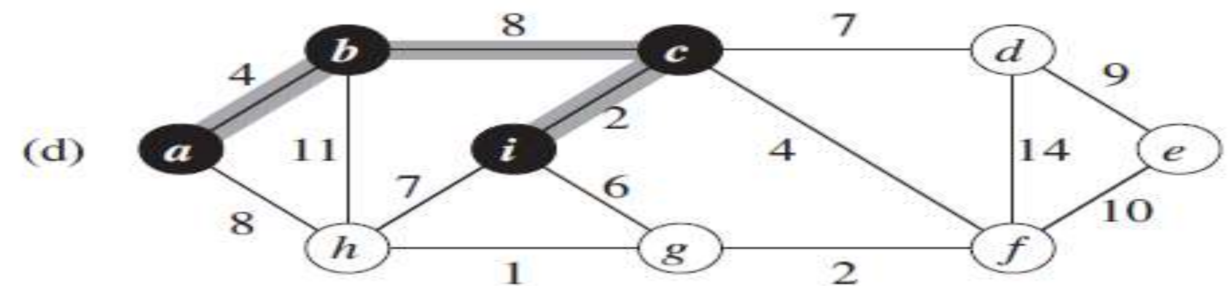
GREEDY STRATEGIES

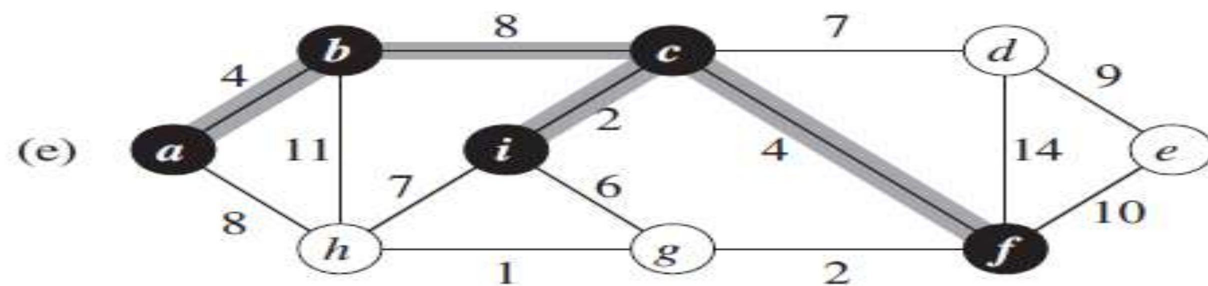
2. Prims algorithm: Example

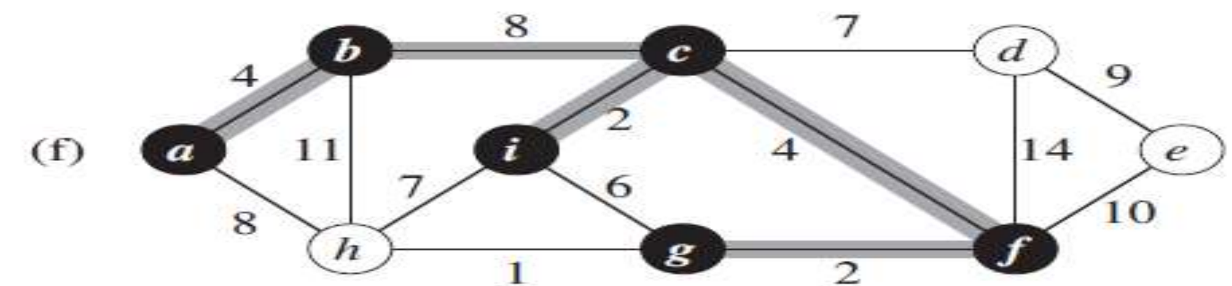


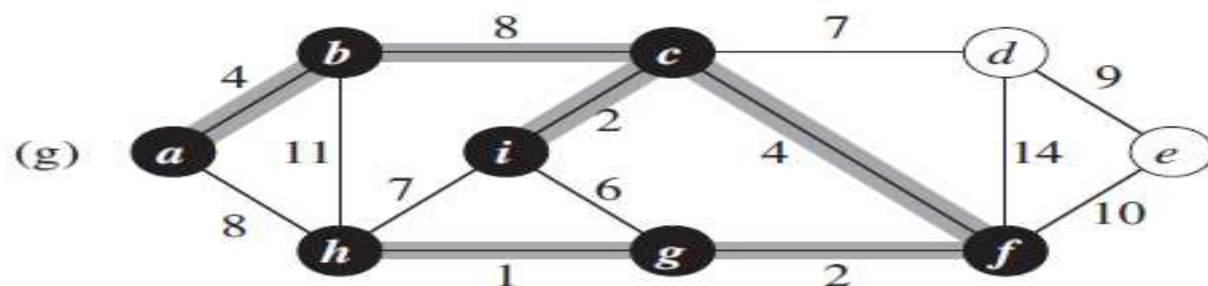


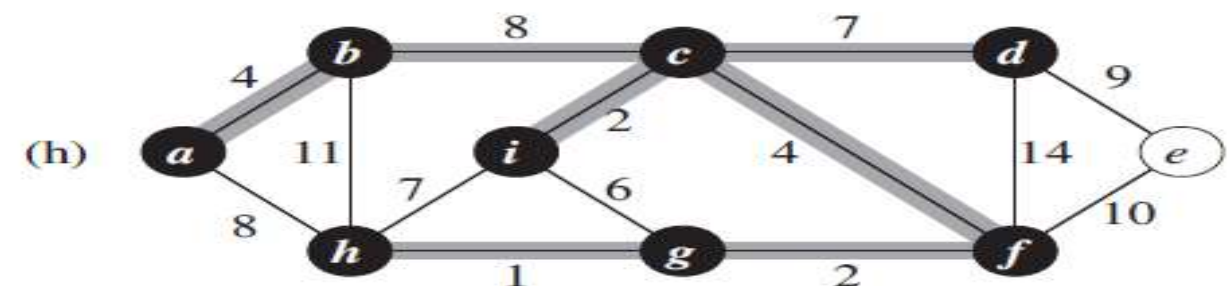


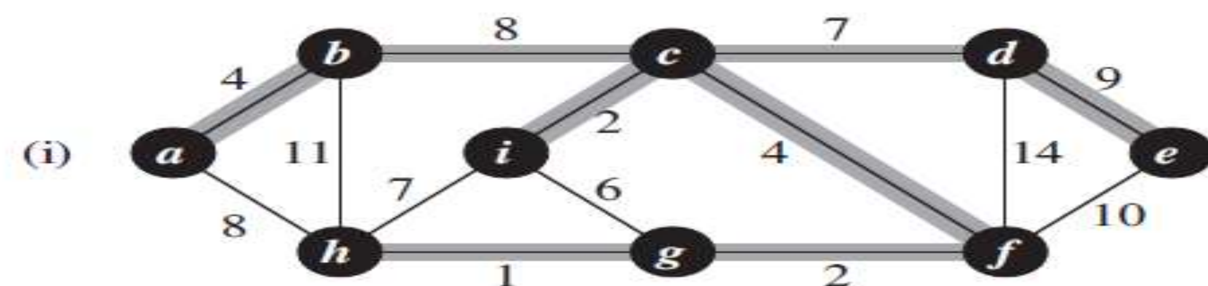






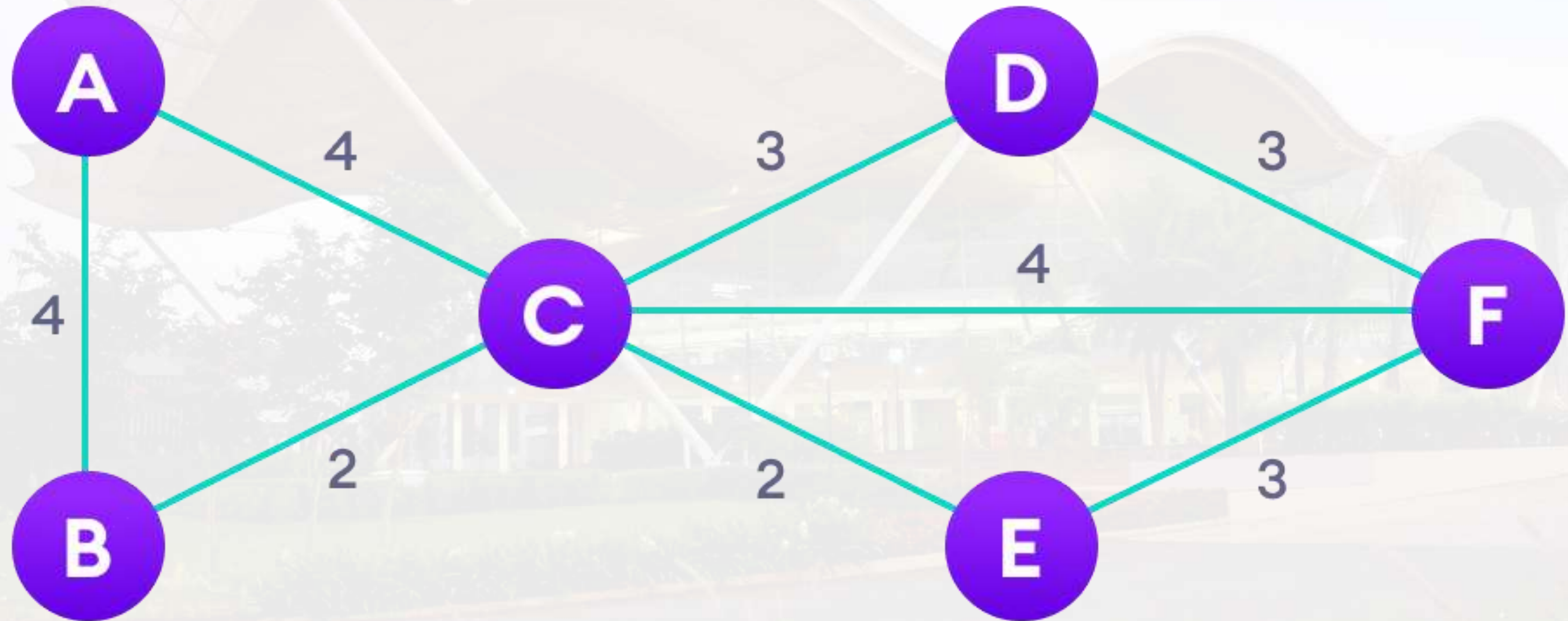






GREEDY STRATEGIES

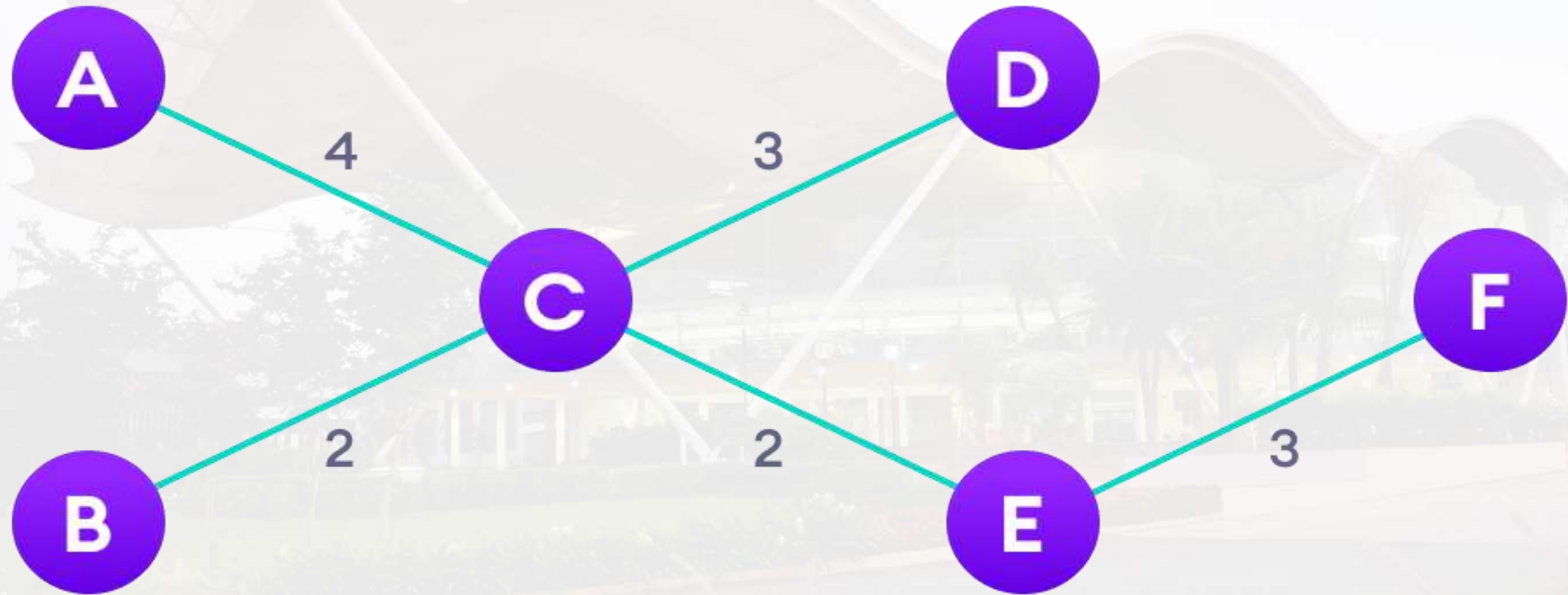
2. **Prims algorithm: Example**



Step: 1

GREEDY STRATEGIES

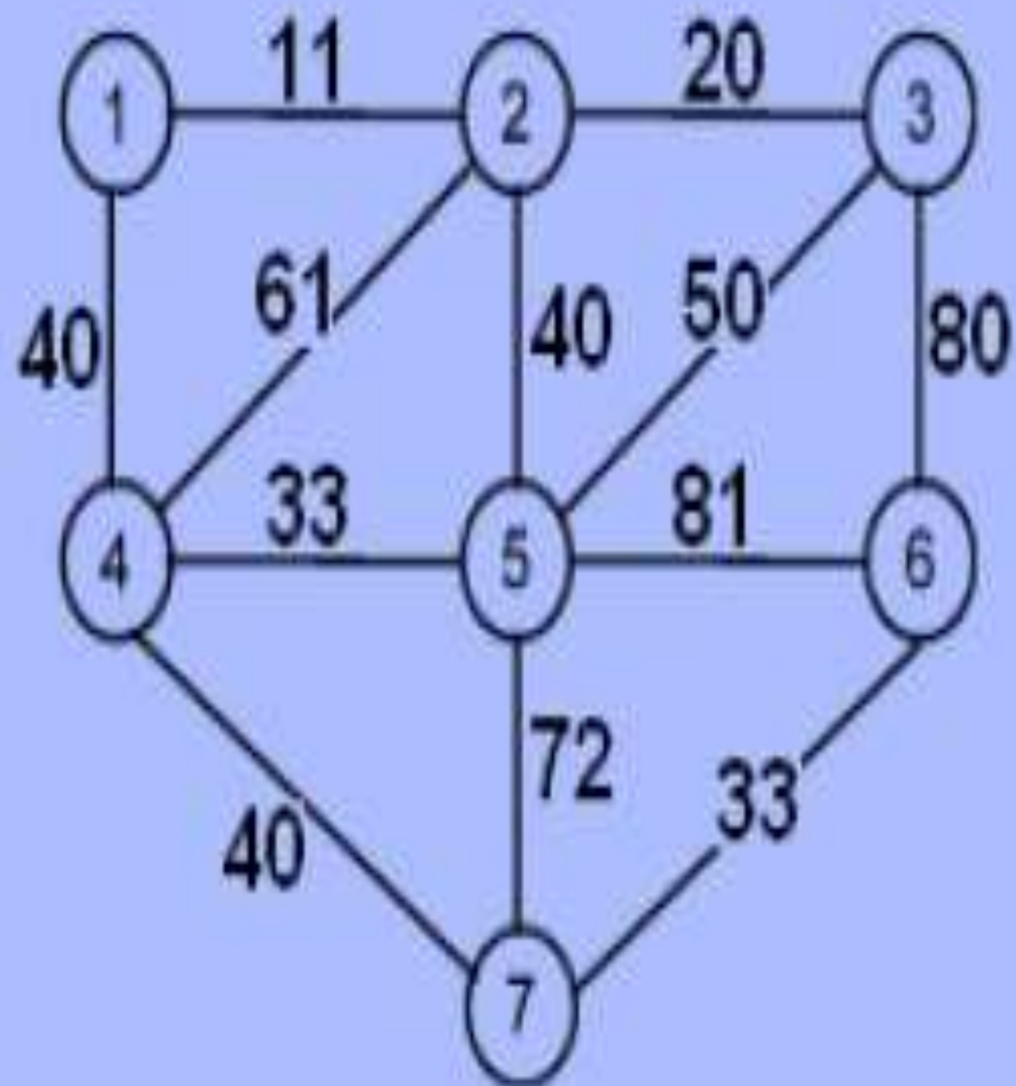
2. **Prims algorithm: Example**

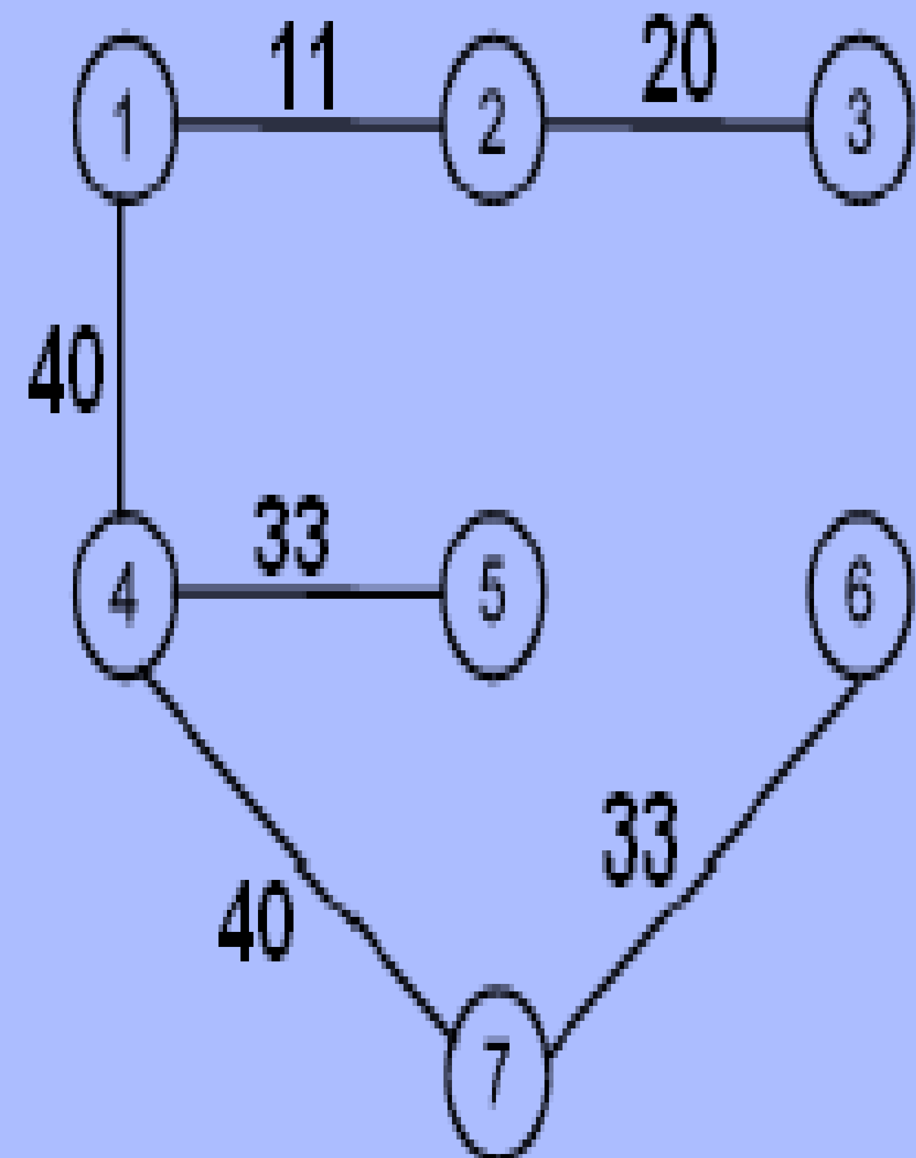


Step: 6

GREEDY STRATEGIES

2. **Prims algorithm: Example**



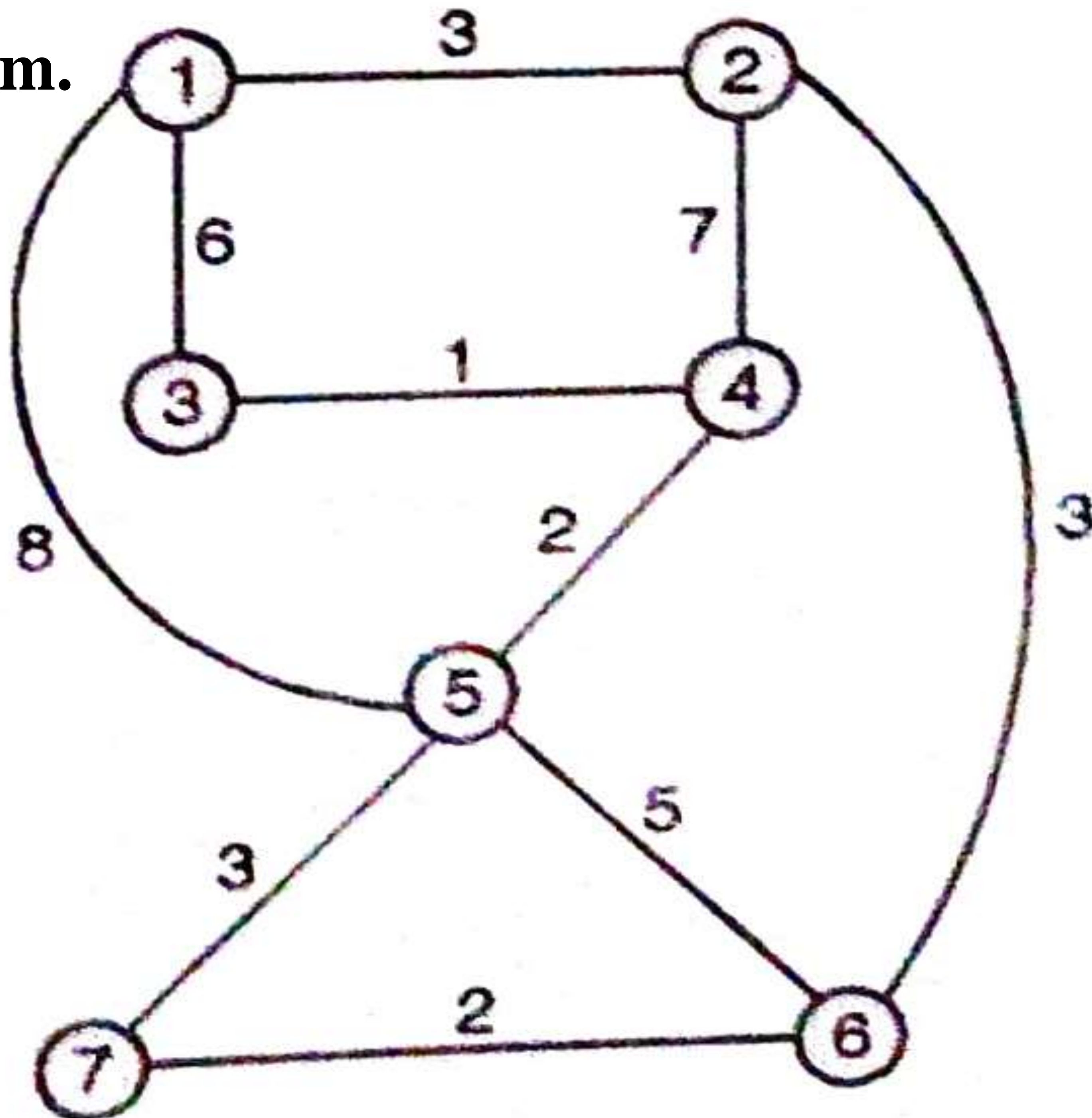


GREEDY STRATEGIES

2. **Prims algorithm: Example**

Find minimum spanning tree for following graph using Prim's algorithm. Show various steps.

Kruska's algorithm.



GREEDY STRATEGIES

2. **Prims algorithm**:

Prim's Algorithm Application

1. Laying cables of electrical wiring
2. In network designed
3. To make protocols in network cycles

Advantages of Disadvantages of Greedy approach

Advantages :

- The algorithm is easier to describe.
- This algorithm can perform better than other algorithms (but, not in all cases).

Disadvantages :

- The greedy algorithm doesn't always produce the optimal solution.

Applications of Greedy Algorithms

- It is used in finding the shortest path.
- It is used to find the minimum spanning tree using the prim's algorithm or the Kruskal's algorithm.
- It is used in a job sequencing with a deadline.
- This algorithm is also used to solve the fractional knapsack problem.

Prim's Vs Kruskal Algorithms

Prims Algorithm	Kruskal Algorithm
It start to build the MST from any of the Node.	It start to build the MST from Minimum weighted vertex in the graph.
Adjencary Matrix , Binary Heap or Fibonacci Heap is used in Prims algorithm	Disjoint Set is used in Kruskal Algorithm.
Prims Algorithm run faster in dense graphs	Kruskal Algorithm run faster in sparse graphs
Time Complexity is O(EV log V) with binay heap and O(E+V log V) with fibonacci heap.	Time Complexity is **O(E log V)**
The next Node included must be connected with the node we traverse	The next edge include may or may not be connected but should not form the cycle.
It traverse the one node saveral time in order to get it minimum distance	It travese the edge only once and based on cycle it will either reject it or accept it,
Greedy Algorithm	Greedy Algorithm

Single Source Shorstest Path

Dijkstra’s Algorithms

- Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph.
- It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph.

How Dijkstra's Algorithms works?

- Dijkstra's Algorithm works on the basis that any subpath B -> D of the shortest path A -> D between vertices A and D is also the shortest path between vertices B and D.

the shortest path between the source and destination

a subpath which is also the shortest path between its source and destination

How Dijkstra's Algorithms works?

- Djikstra used this property in the opposite direction i.e we overestimate the distance of each vertex from the starting vertex. Then we visit each node and its neighbors to find the shortest subpath to those neighbors.

- The algorithm uses a greedy approach in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

Algorithms of Dijkstra's Algorithms

```
Begin
   create a status list to hold the current status of the selected node
   for all vertices u in V do
      status[u] := unconsidered
      dist[u] := distance from source using cost matrix
      next[u] := start
   done
   status[start] := considered, dist[start] := 0 and next[start] := φ
   while take unconsidered vertex u as distance is minimum do
      status[u] := considered
      for all vertex v in V do
         if status[v] = unconsidered then
         if dist[v] > dist[u] + cost[u,v] then
            dist[v] := dist[u] + cost[u,v]
            next[v] := u
      done
   done
End
```

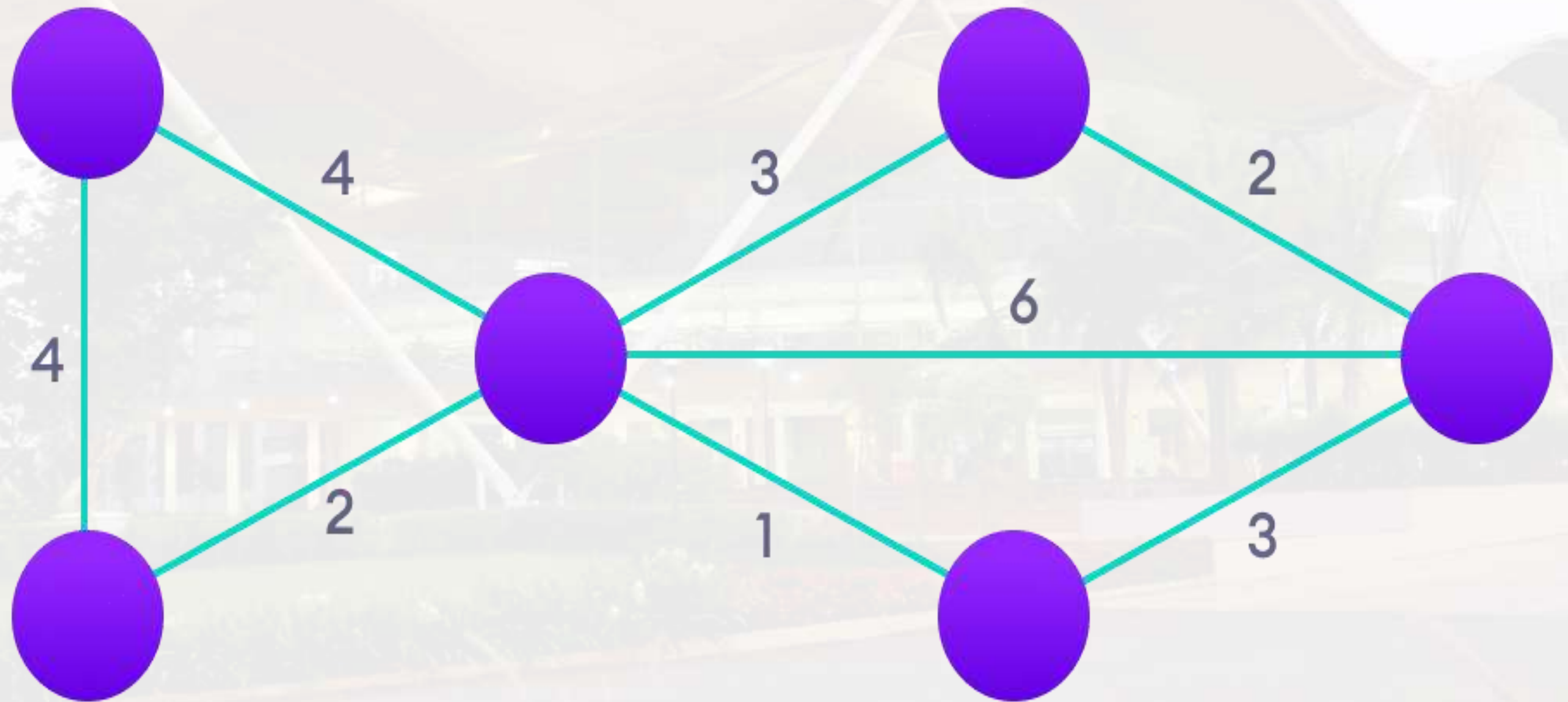
Algorithms of Dijkstra's Algorithms

```
function dijkstra(G, S)
    for each vertex V in G
        distance[V] <- infinite
        previous[V] <- NULL
        If V != S, add V to Priority Queue Q
    distance[S] <- 0

    while Q IS NOT EMPTY
        U <- Extract MIN from Q
        for each unvisited neighbour V of U
            tempDistance <- distance[U] + edge_weight(U, V)
            if tempDistance < distance[V]
                distance[V] <- tempDistance
                previous[V] <- U
    return distance[], previous[]
```

Example of Dijkstra's Algorithms

- **Example of Dijkstra's algorithm**

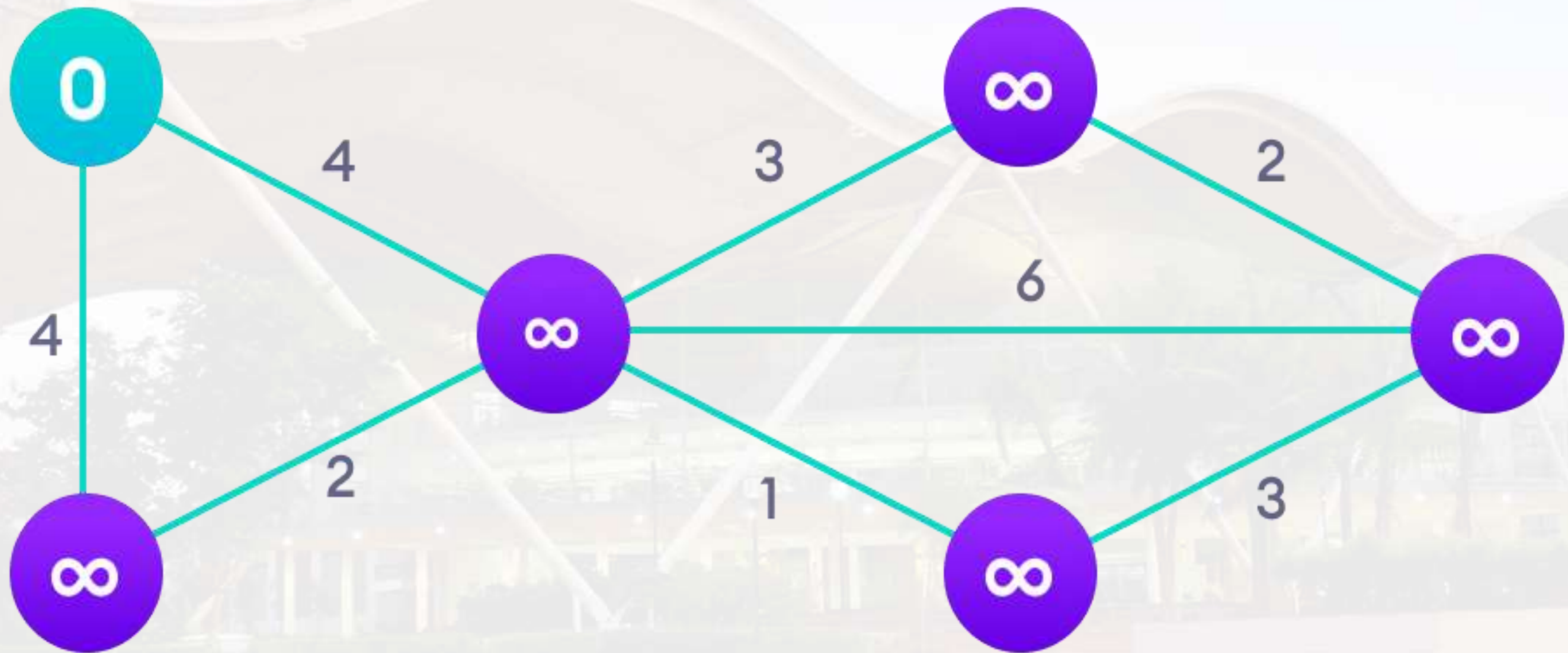
It is easier to start with an example and then think about the algorithm.



1. Start with a weighted graph

Example of Dijkstra's Algorithms

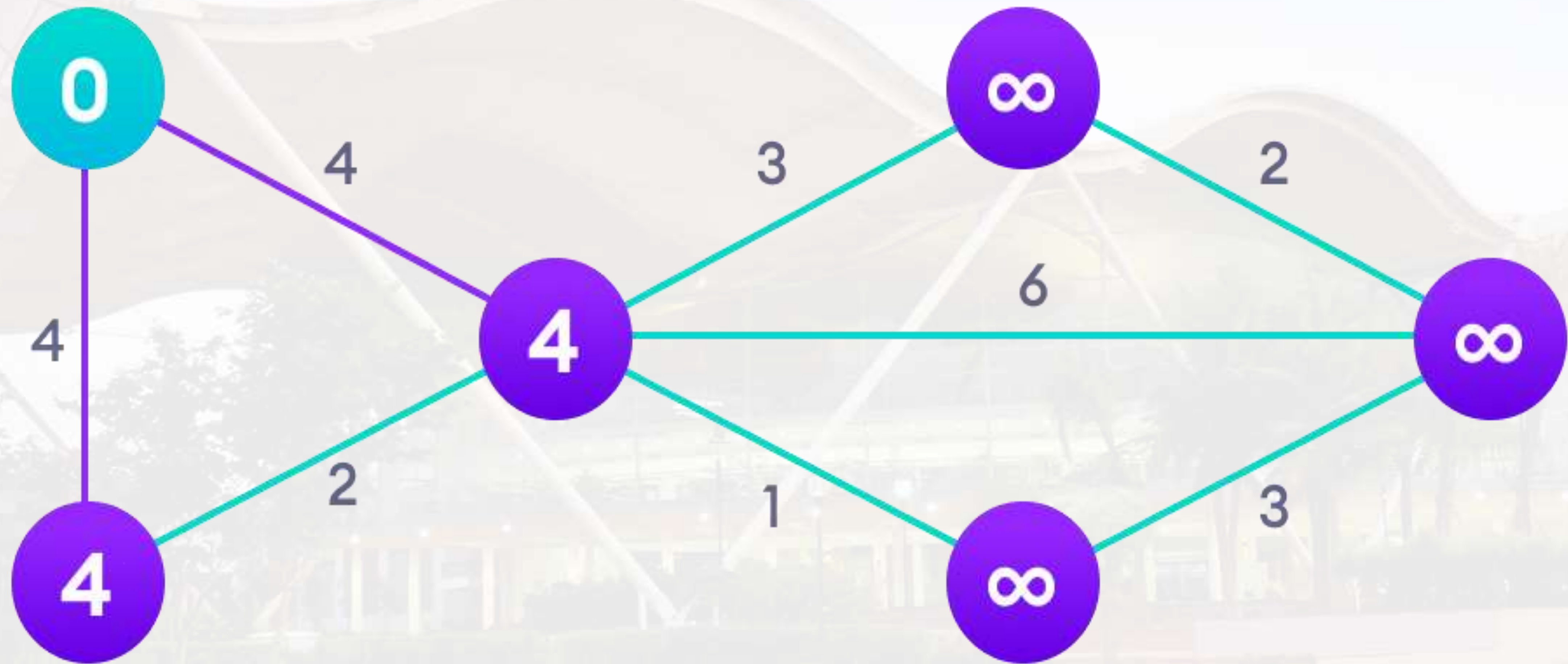
- **Example of Dijkstra's algorithm**



Choose a starting vertex and assign infinity path values to all other devices

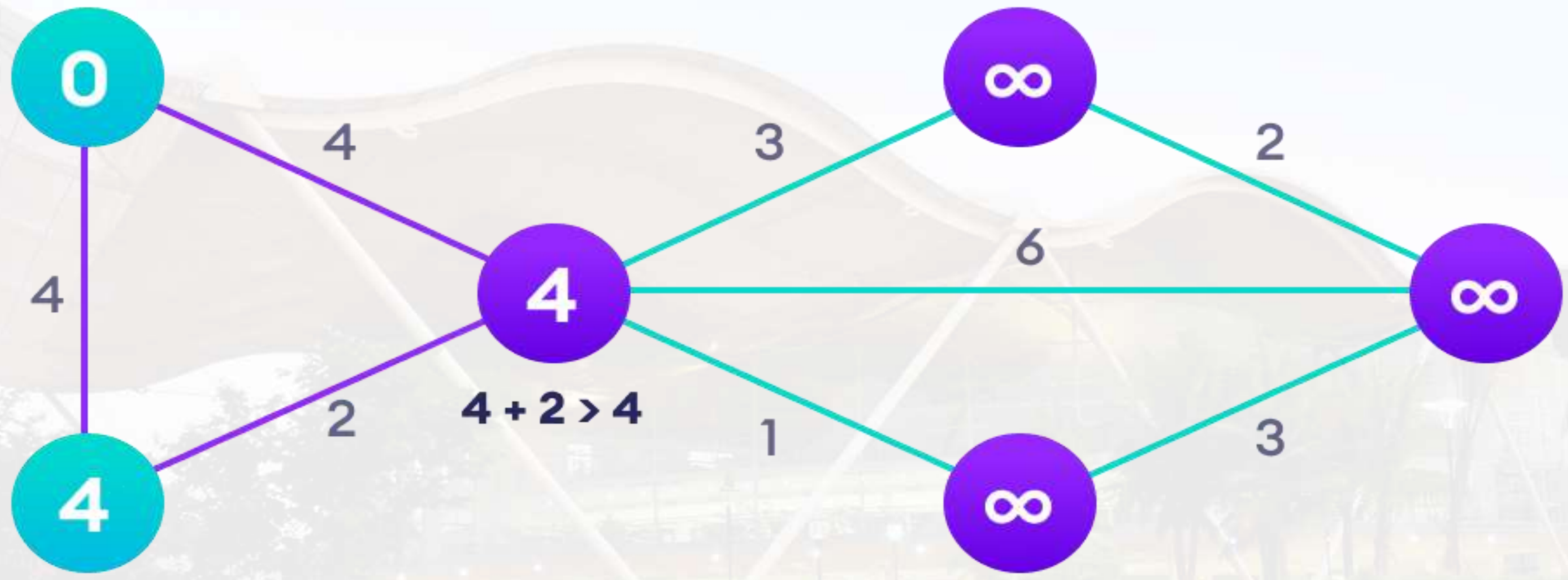
Example of Dijkstra's Algorithms

- **Example of Dijkstra's algorithm**



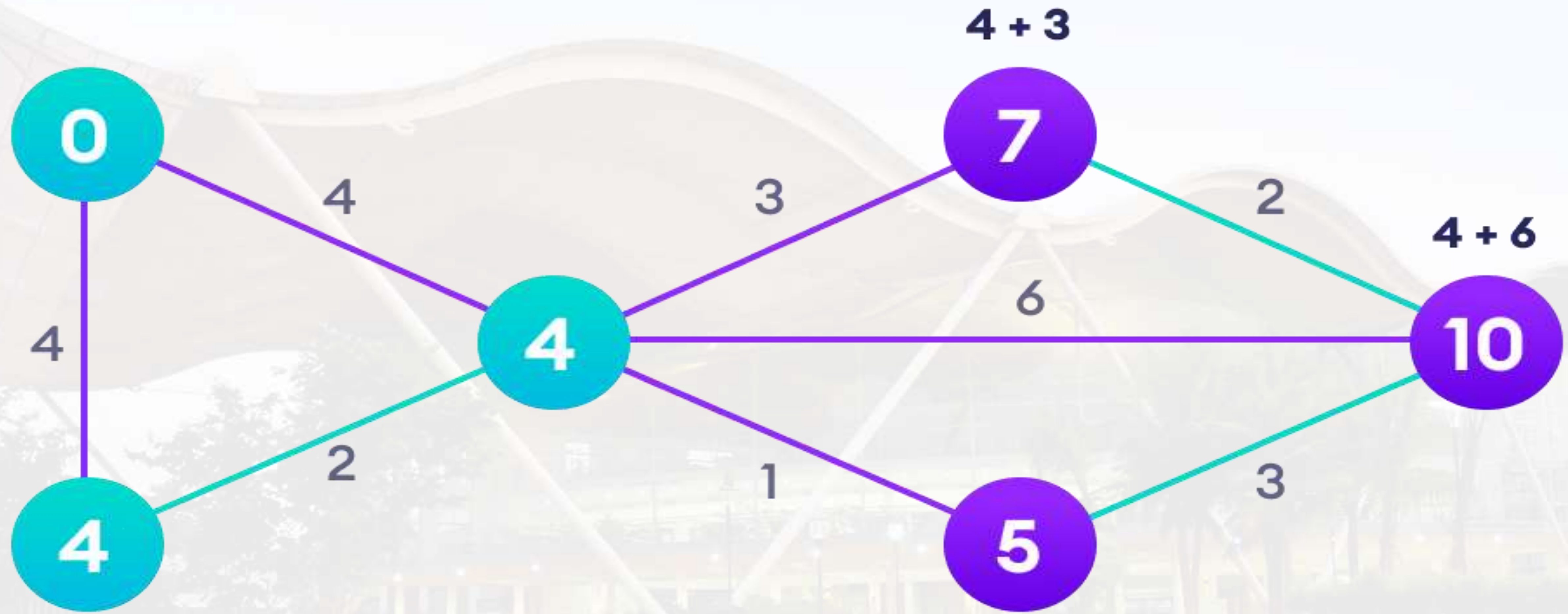
Go to each vertex and update its path length

Example of Dijkstra's Algorithms



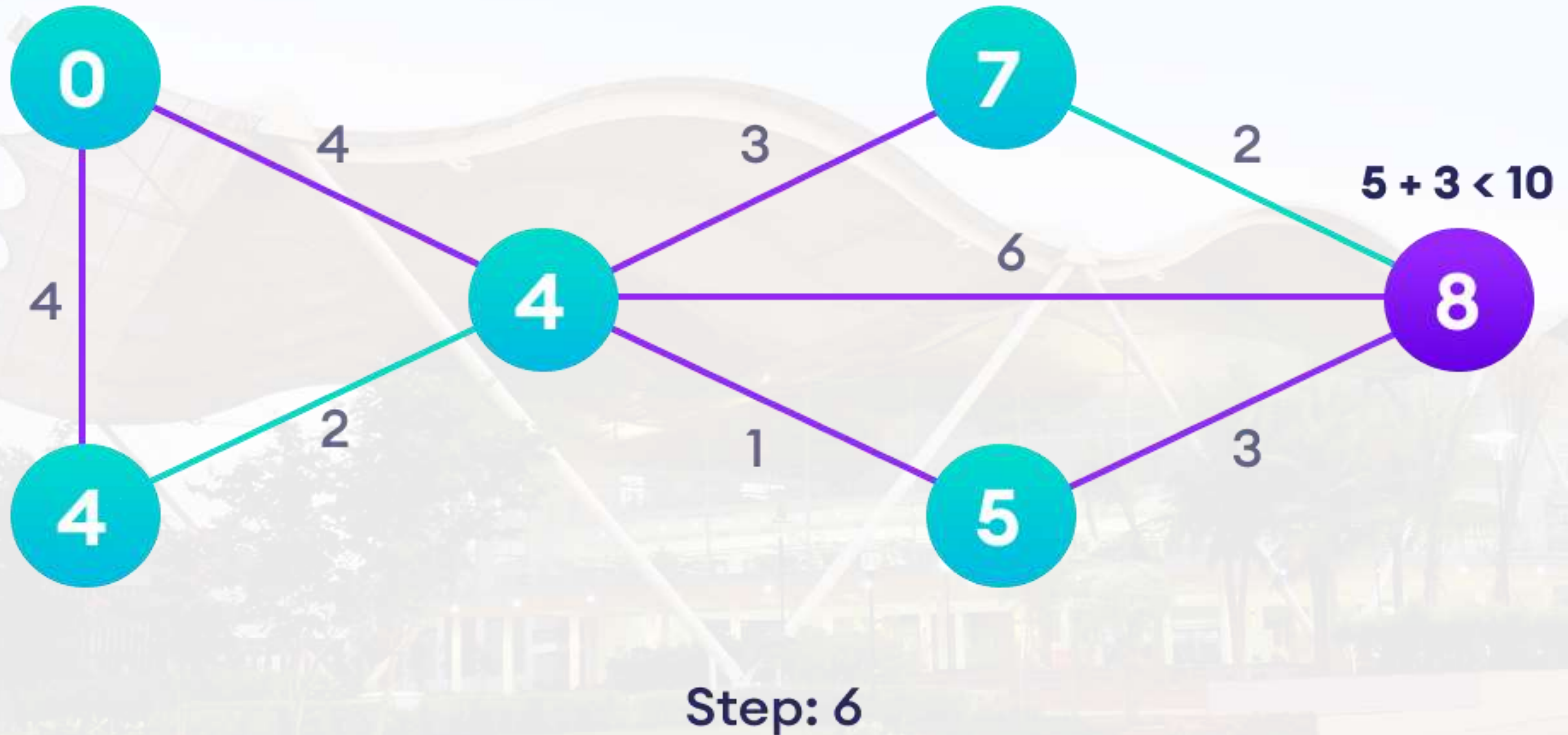
If the path length of the adjacent vertex is lesser than new path length, don't update it

Example of Dijkstra's Algorithms



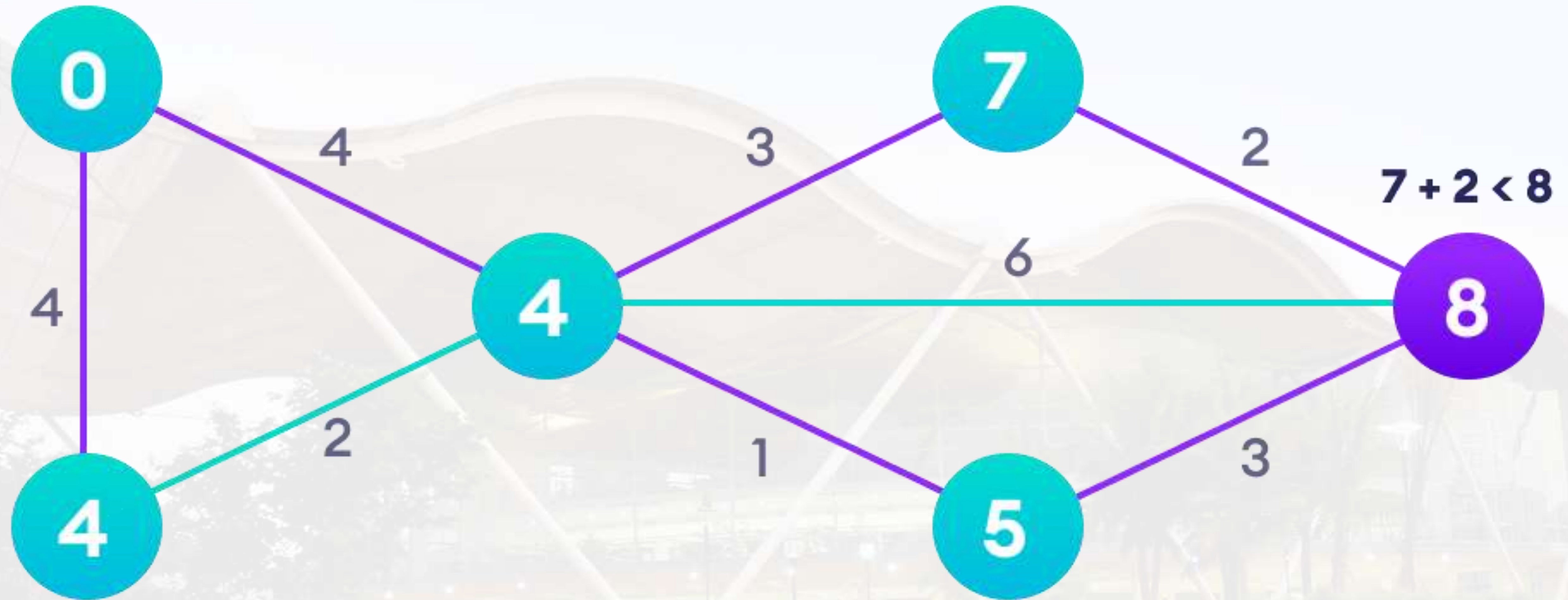
Avoid updating path lengths of already visited vertices

Example of Dijkstra's Algorithms



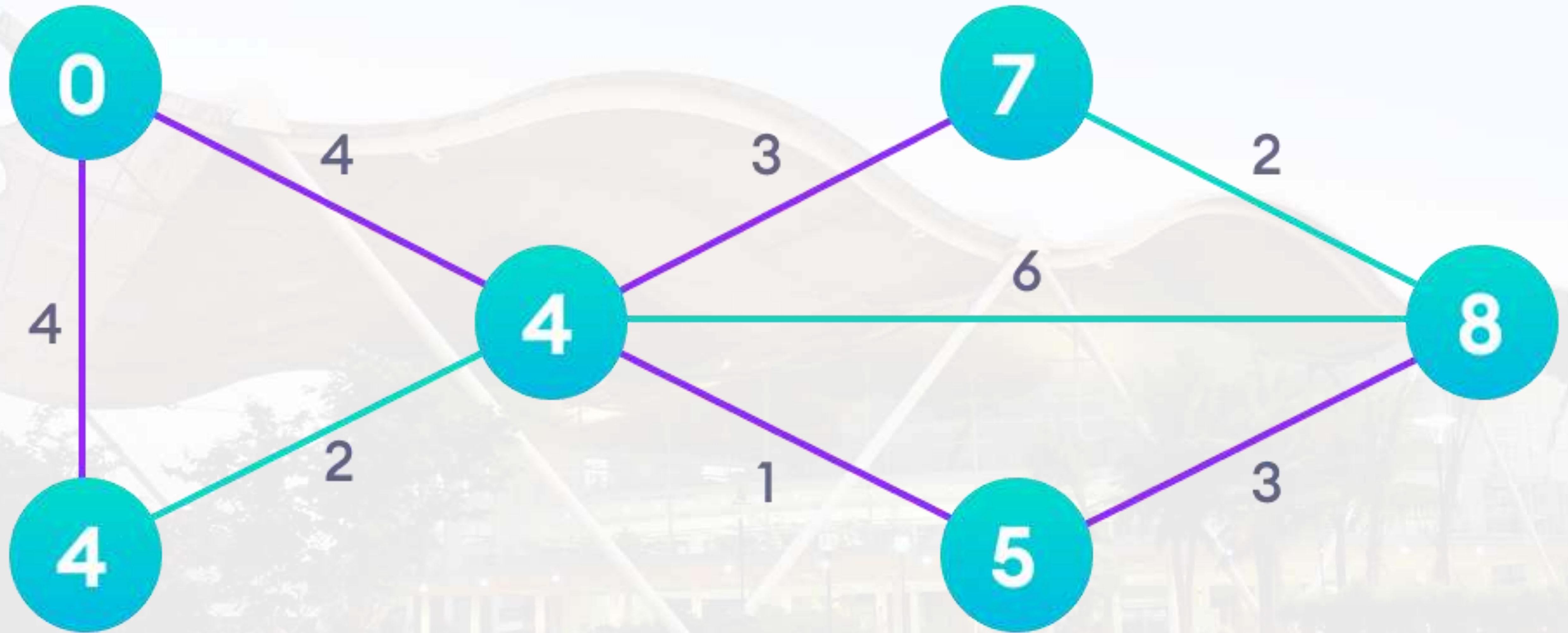
After each iteration, we pick the unvisited vertex with the least path length. So we choose 5 before 7

Example of Dijkstra’s Algorithms



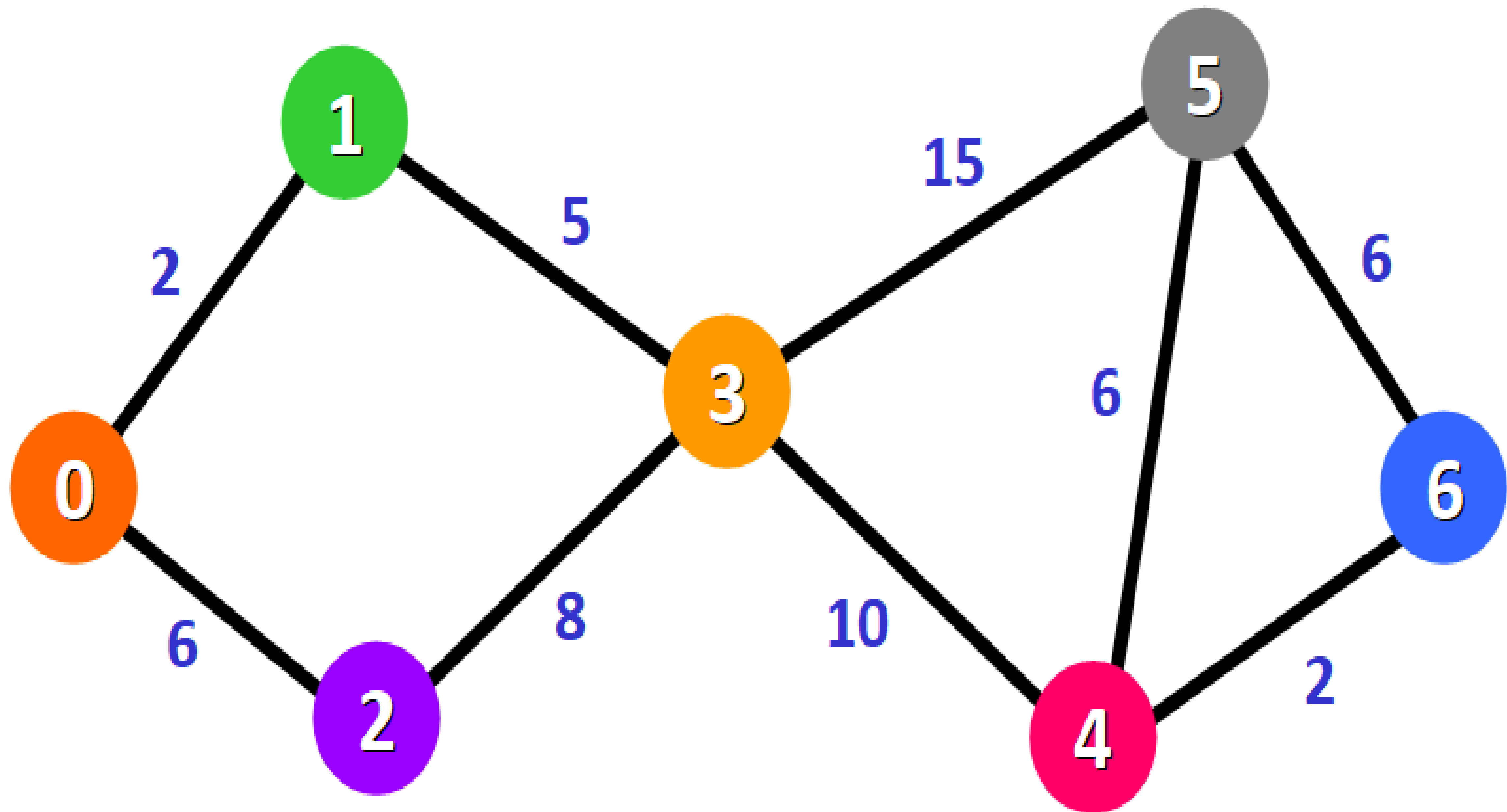
Notice how the rightmost vertex has its path length updated twice

Example of Dijkstra's Algorithms



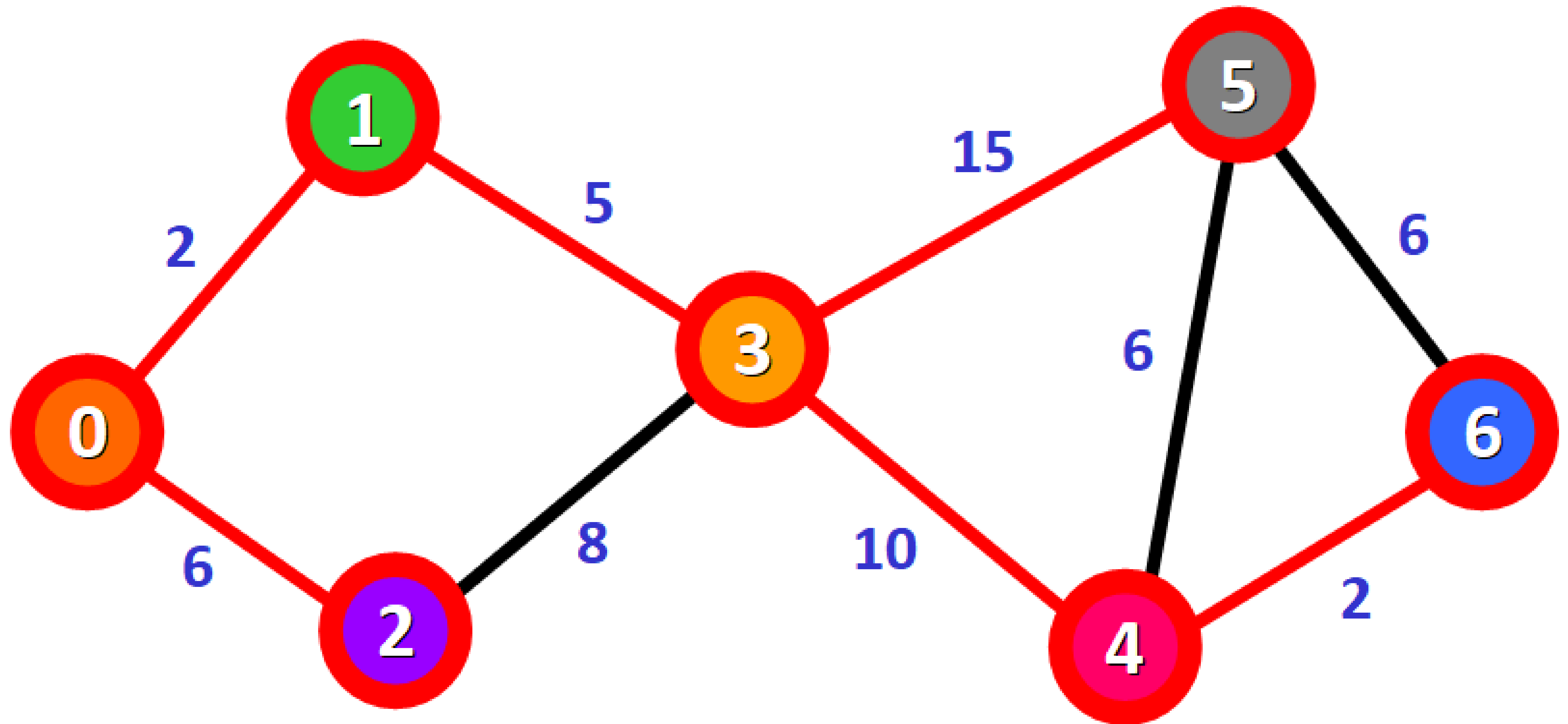
Repeat until all the vertices have been visited

Example of Dijkstra's Algorithms



https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/

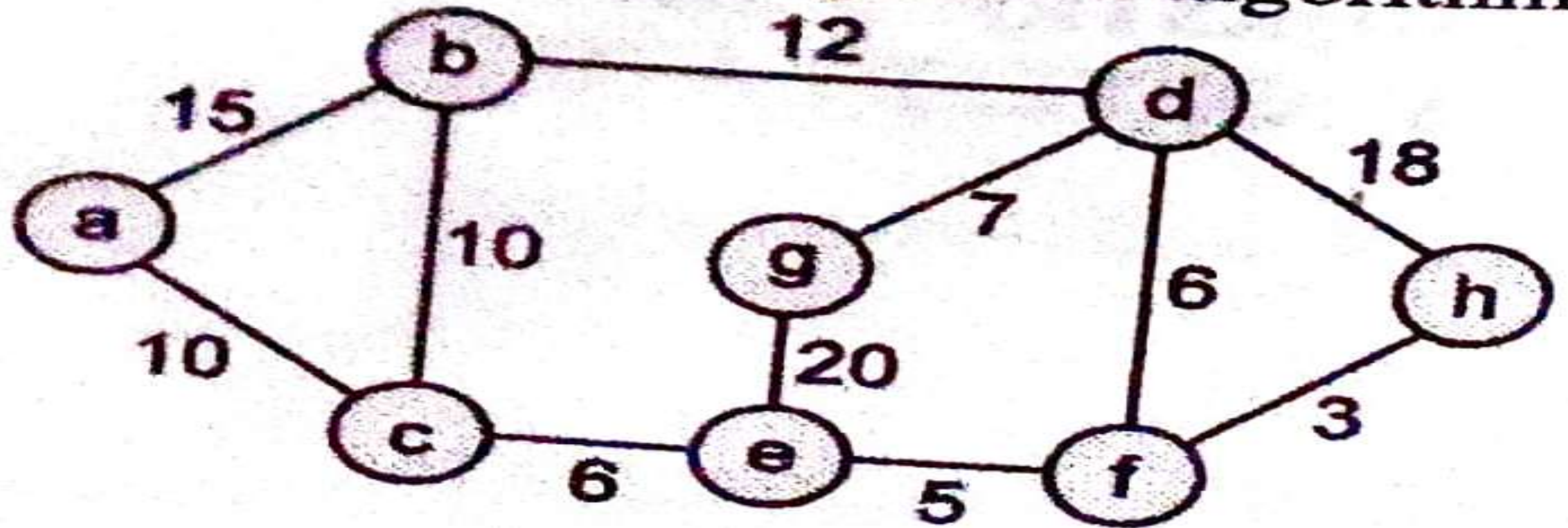
Example of Dijkstra's Algorithms



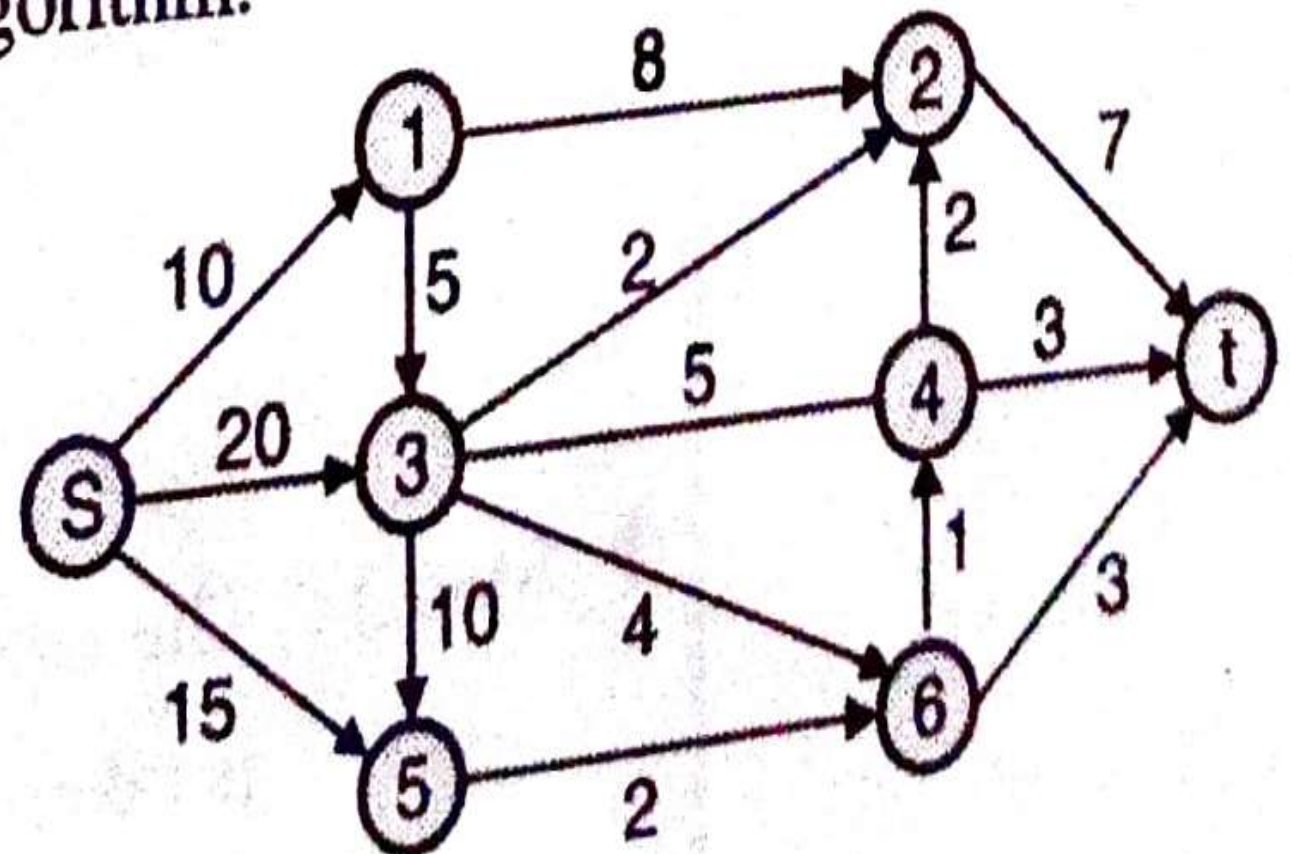
ANSWERs : https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/

Example of Dijkstra’s Algorithms

Find the shortest path using Dijkstra's algorithm :



Find the shortest path using Dijkstra's Algorithm.



Time Complexity Dijkstra’s Algorithms

- **Time Complexity**: O(E Log V)

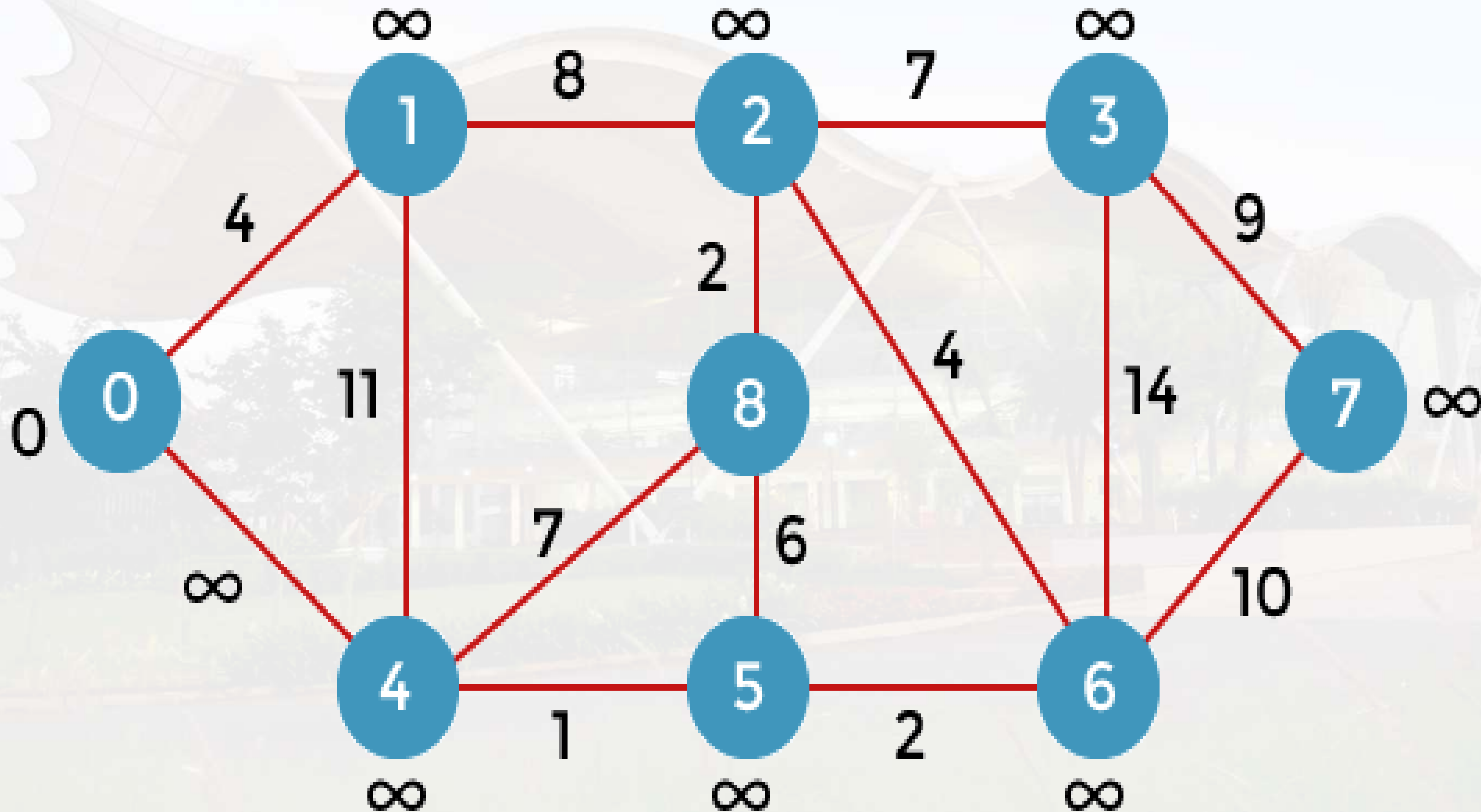
where, E is the number of edges and V is the number of vertices.

- **Space Complexity**: O(V)

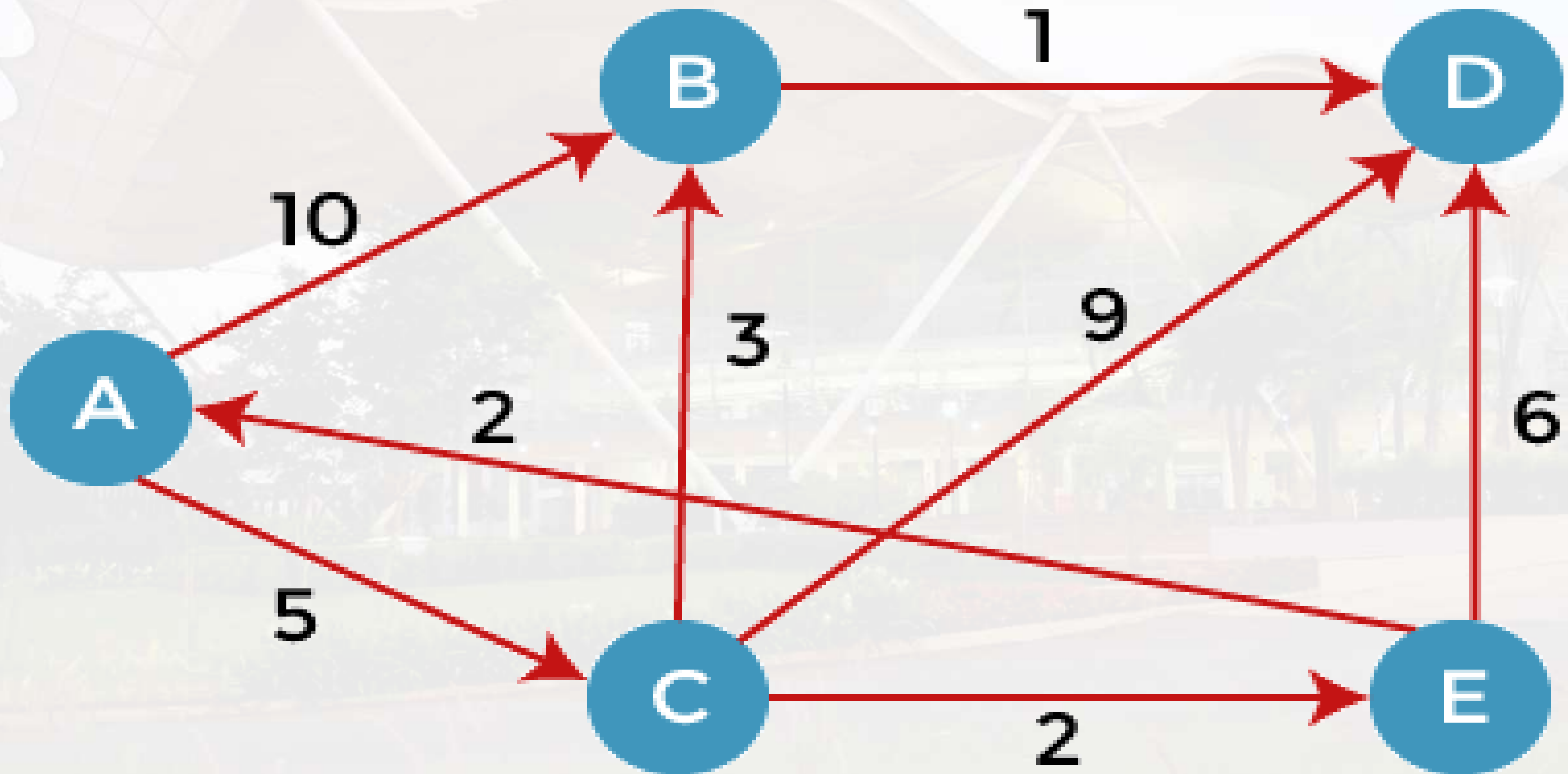
Applications of Dijkstra's Algorithms

- To find the shortest path
- In social networking applications
- In a telephone network
- To find the locations in the map

Example of Dijkstra's Algorithms



Example of Dijkstra's Algorithms



All Pair Shorstest Path

Floyd-Warshall Algorithms

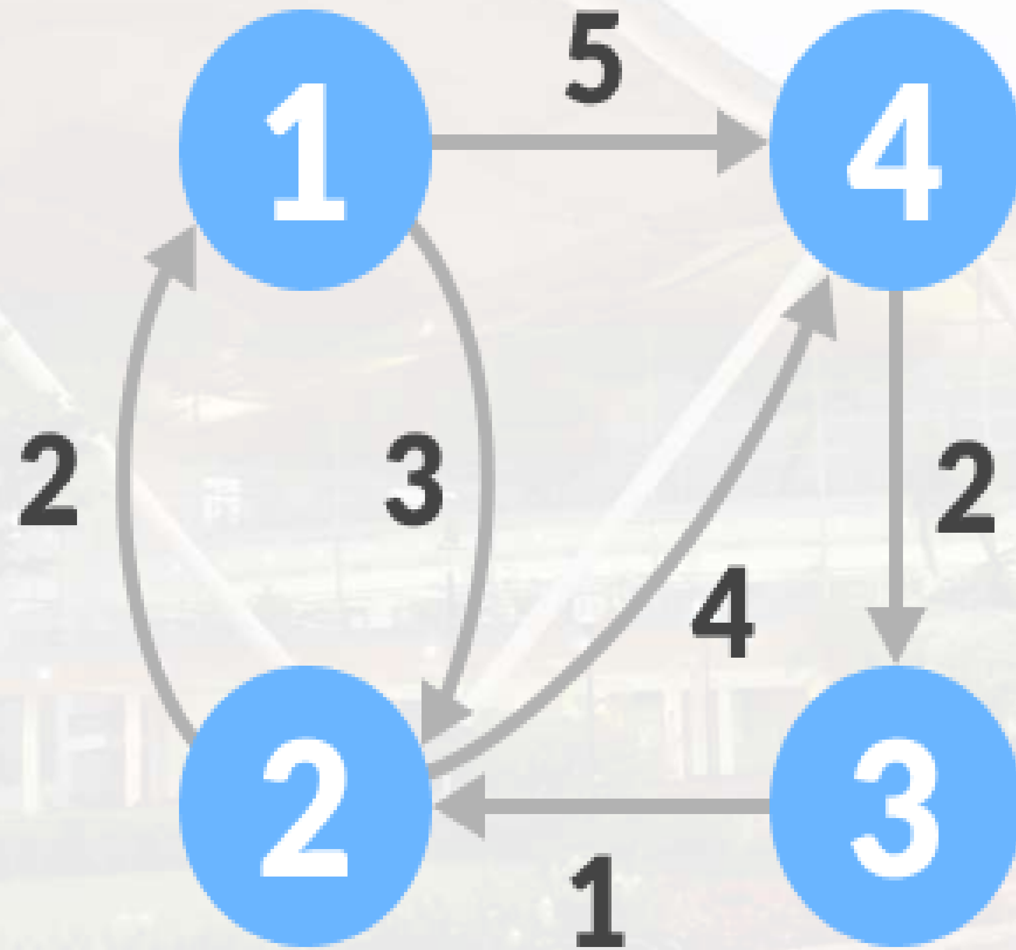
- Floyd Warshall Algorithm is a famous algorithm.
- It is used to solve All Pairs Shortest Path Problem.
- It computes the shortest path between every pair of vertices of the given graph.
- Floyd Warshall Algorithm is an example of dynamic programming approach.

Floyd-Warshall Algorithms

```
n = no of vertices

A = matrix of dimension n*n

for k = 1 to n

    for i = 1 to n

        for j = 1 to n

            Ak[i, j] = min (Ak-1[i, j], Ak-1[i, k] + Ak-1[k, j])

return A
```

Sample e.g. : https://www.gatevidyalay.com/floyd-warshall-algorithm-shortest-path-algorithm/

Example of Floyd-Warshall Algorithms



Initial graph

Example : https://www.programiz.com/dsa/floyd-warshall-algorithm

Example of Floyd-Warshall Algorithms

$$A^0 = \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 5 \\ 2 & 2 & 0 & \infty & 4 \\ 3 & \infty & 1 & 0 & \infty \\ 4 & \infty & \infty & 2 & 0 \end{array}$$

Fill each cell with the distance between ith and jth vertex

Example of Floyd-Warshall Algorithms

$$A^1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 5 \\ 2 & 2 & 0 & & \\ 3 & \infty & & 0 & \\ 4 & \infty & & & 0 \end{array} \rightarrow \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 5 \\ 2 & 2 & 0 & 9 & 4 \\ 3 & \infty & 1 & 0 & 8 \\ 4 & \infty & \infty & 2 & 0 \end{array}$$

Calculate the distance from the source vertex to destination vertex through this vertex k

Example of Floyd-Warshall Algorithms

A^2 =

	1	2	3	4
1	0	3		
2	2	0	9	4
3		1	0	
4		∞		0

→

	1	2	3	4
1	0	3	9	5
2	2	0	9	4
3	3	1	0	5
4	∞	∞	2	0

Calculate the distance from the source vertex to destination vertex through this vertex 2

Example of Floyd-Warshall Algorithms

$A^3 =$

	1	2	3	4
1	0		∞	
2		0	9	
3	∞	1	0	8
4			2	0

→

	1	2	3	4
1	0	3	9	5
2	2	0	9	4
3	3	1	0	5
4	5	3	2	0

Calculate the distance from the source vertex to destination vertex through this vertex 3

Example of Floyd-Warshall Algorithms

$$A^4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & & & 5 \\ 2 & & 0 & & 4 \\ 3 & & & 0 & 5 \\ 4 & 5 & 3 & 2 & 0 \end{array} \rightarrow \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 7 & 5 \\ 2 & 2 & 0 & 6 & 4 \\ 3 & 3 & 1 & 0 & 5 \\ 4 & 5 & 3 & 2 & 0 \end{array}$$

Calculate the distance from the source vertex to destination vertex through this vertex 4

A4 gives the shortest path between each pair of vertices.

Floyd-Warshall Algorithms

- **Advantages :**

1. It is extremely simple.
2. It is easy to implement.

Floyd-Warshall Algorithms

- **Time Complexity**

There are three loops. Each loop has constant complexities. So, the time complexity of the Floyd-Warshall algorithm is $O(n^3)$.

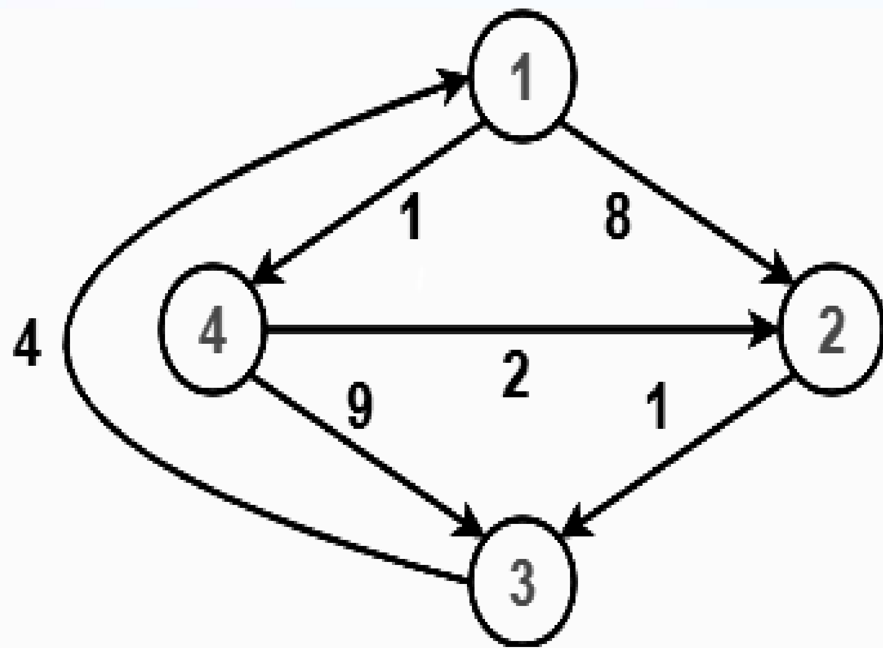
- **Space Complexity**

The space complexity of the Floyd-Warshall algorithm is $O(n^2)$.

Applications of Floyd-Warshall Algorithms

1. To find the shortest path is a directed graph
2. To find the transitive closure of directed graphs
3. To find the Inversion of real matrices
4. For testing whether an undirected graph is bipartite
5. https://youtu.be/NdBHw5mqIZE https://youtu.be/Gc4mWrmJBsw

Example of Floyd-Warshall Algorithms

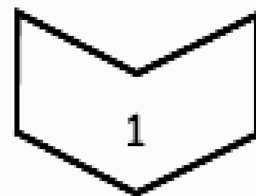


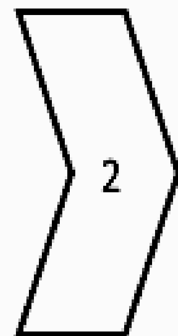
D_1 =

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	9	0

D_2 =

	1	2	3	4
1	0	8	9	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	3	0





1+2=3

$3 < 8 \rightarrow 3$

D_0 =

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	∞	0	∞
4	∞	2	9	0

D_3 =

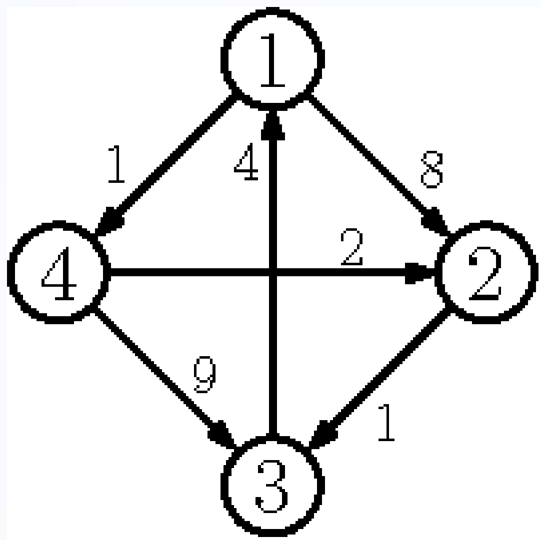
	1	2	3	4
1	0	8	9	1
2	5	0	1	6
3	4	12	0	5
4	7	2	3	0

D_4 =

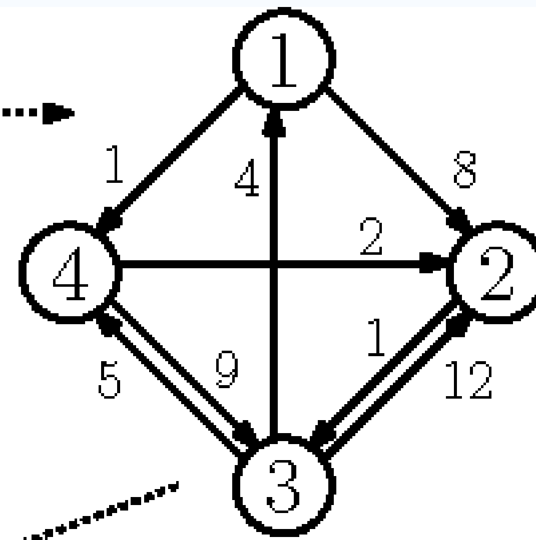
	1	2	3	4
1	0	3	4	1
2	5	0	1	6
3	4	7	0	5
4	7	2	3	0

by: Masoud Azizi

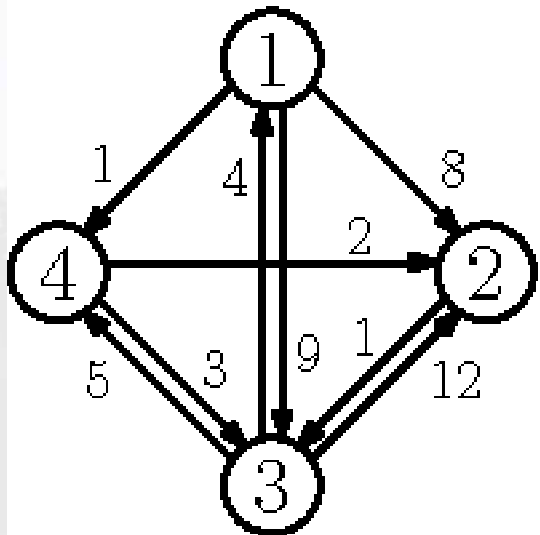
Example of Floyd-Warshall Algorithms



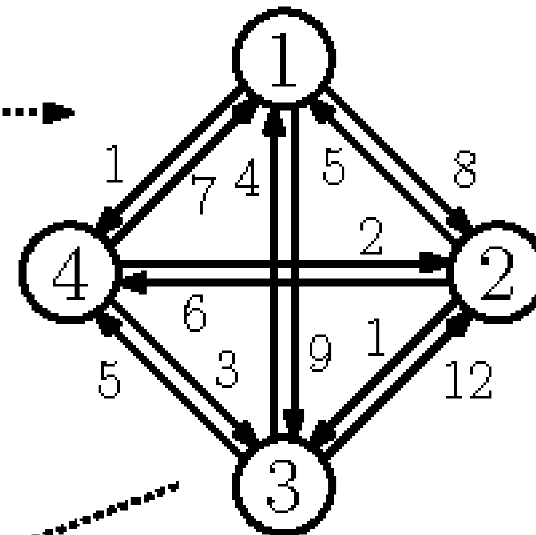
$$d^{(0)} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix}$$



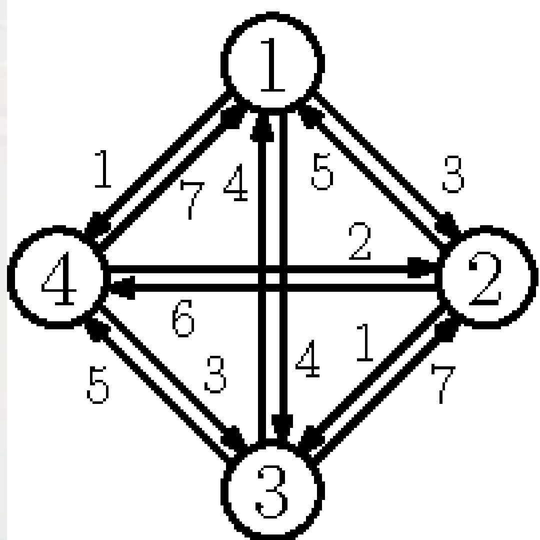
$$d^{(1)} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix}$$



$$d^{(2)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{bmatrix}$$



$$d^{(3)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$$



$$d^{(4)} = \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$$

$$\text{final} = \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$$

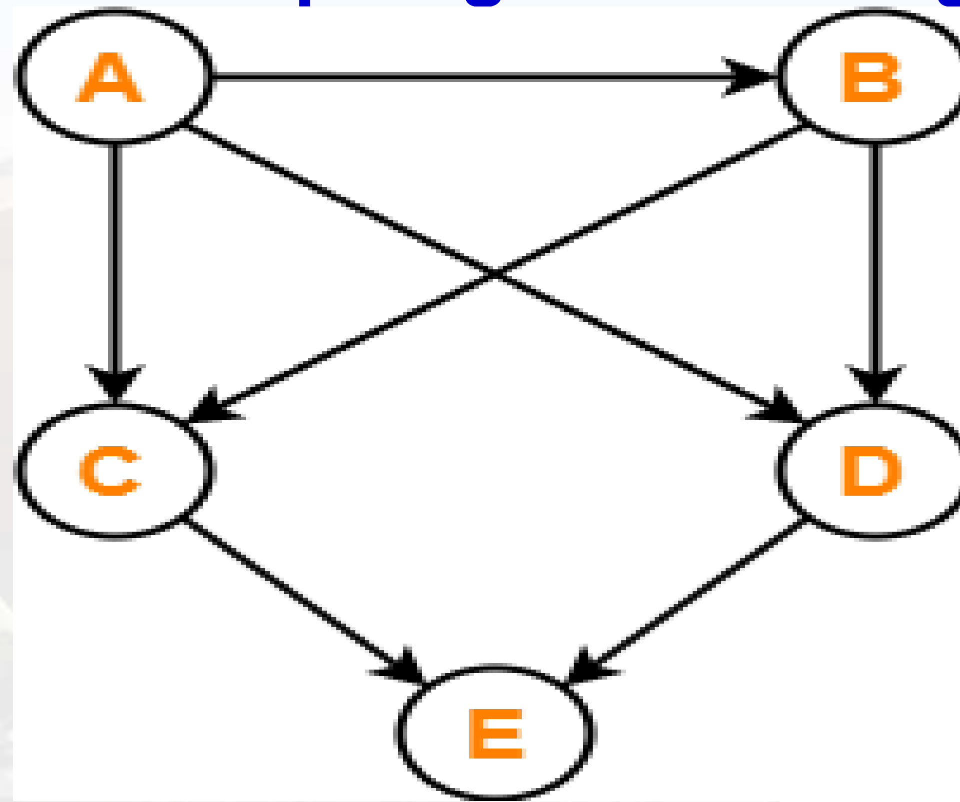
Topological Ordering / Sort

Topological Sort/Ordering

- Topological Sorting is possible if and only if the graph is a Directed Acyclic Graph.
- There may exist multiple different topological orderings for a given directed acyclic graph.

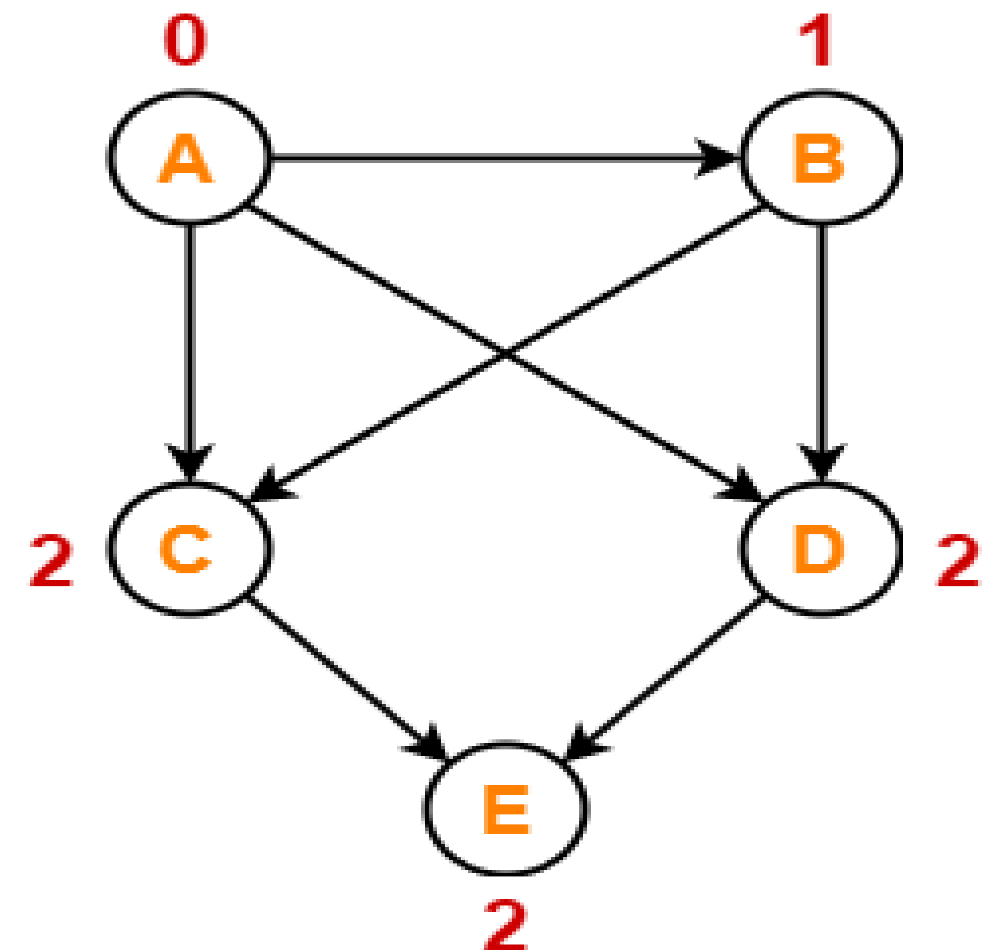
Topological Sort/Ordering

Find the number of different topological orderings possible for the given graph-



Step-01:

Write in-degree of each vertex-

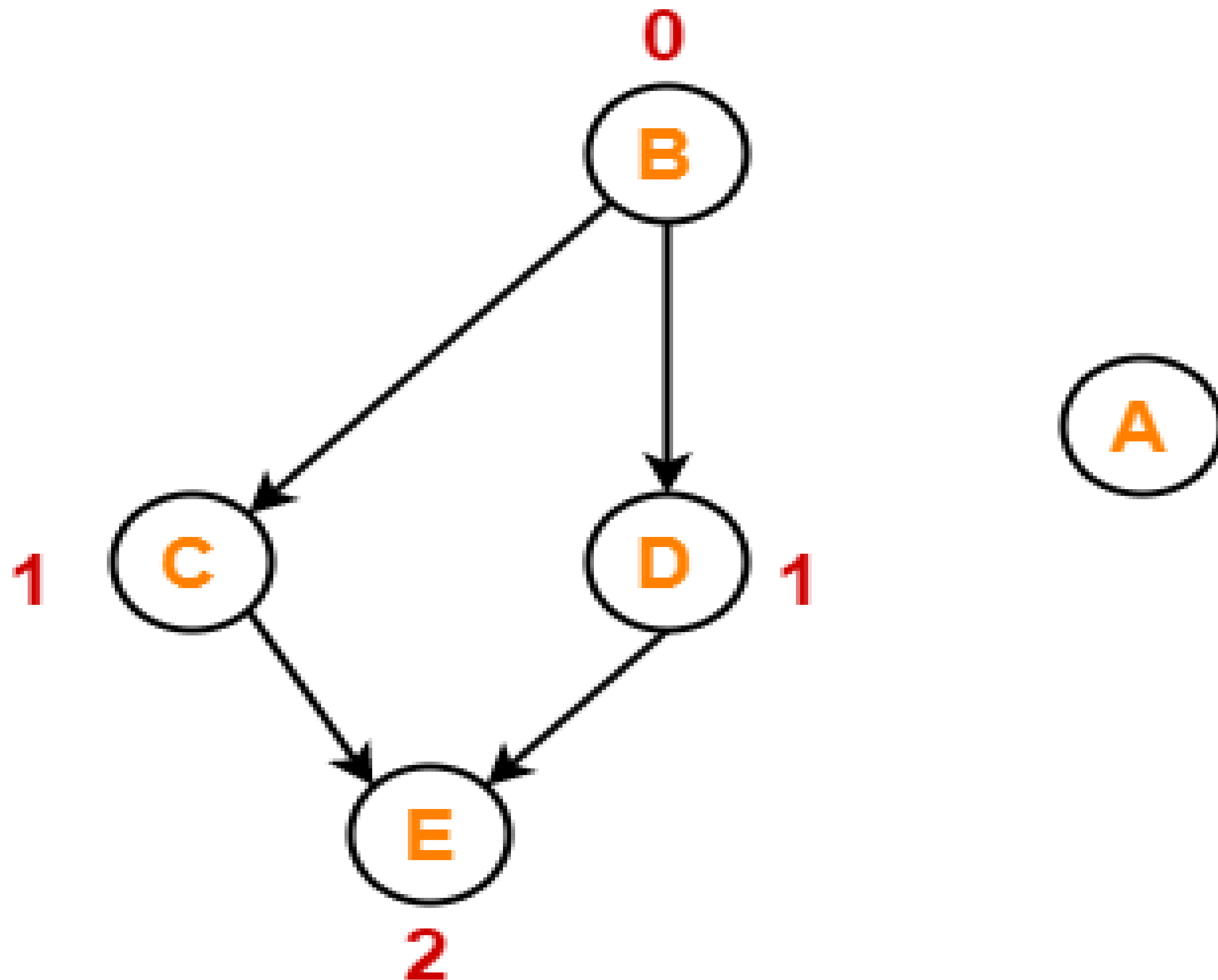


•

Topological Sort/Ordering

Step-02:

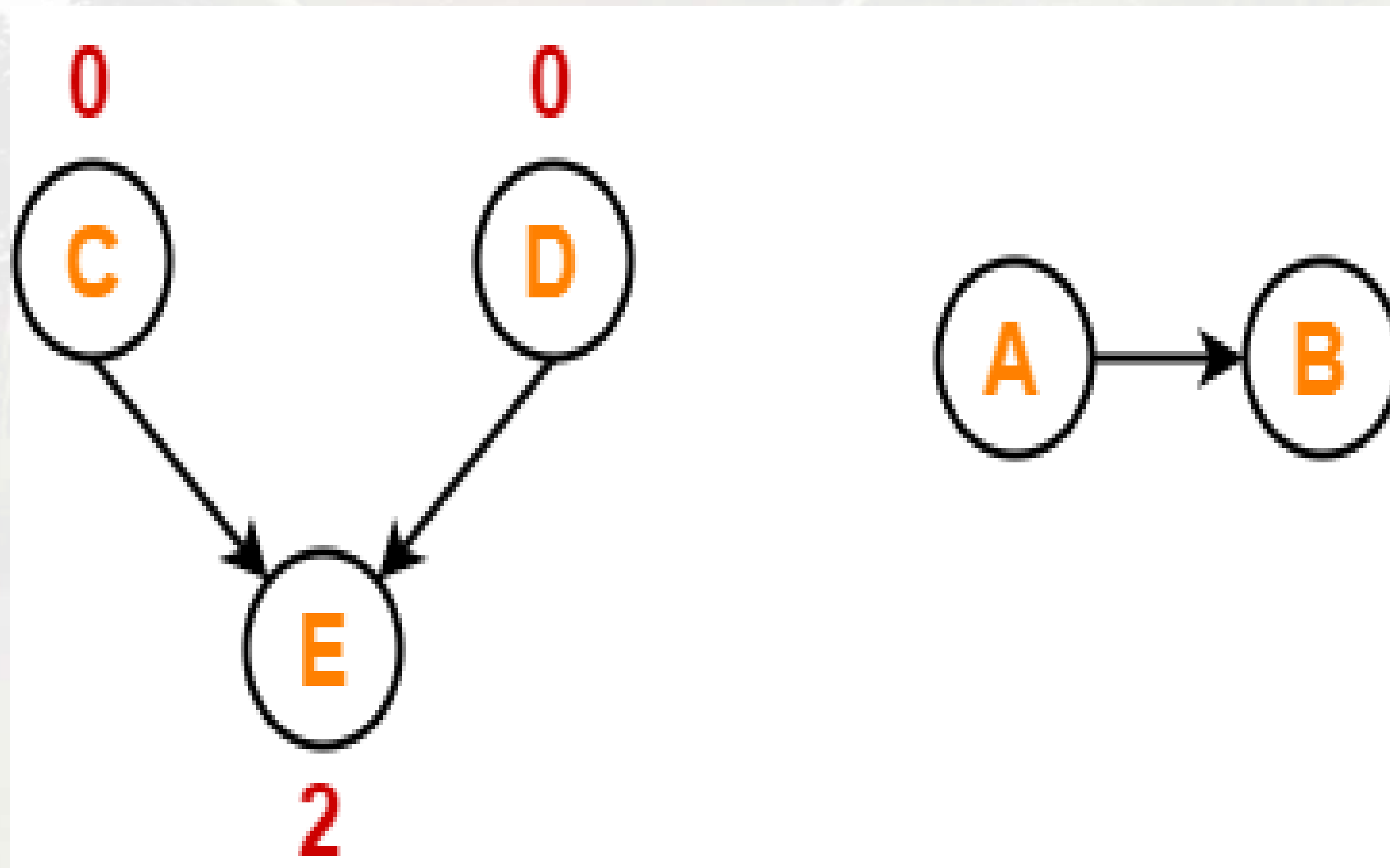
- Vertex-A has the least in-degree.
- So, remove vertex-A and its associated edges.
- Now, update the in-degree of other vertices.



Topological Sort/Ordering

Step-03:

- Vertex-B has the least in-degree.
- So, remove vertex-B and its associated edges.
- Now, update the in-degree of other vertices.



Topological Sort/Ordering

Step-04:

- There are two vertices with the least in-degree. So, following 2 cases are possible-

In case-01,

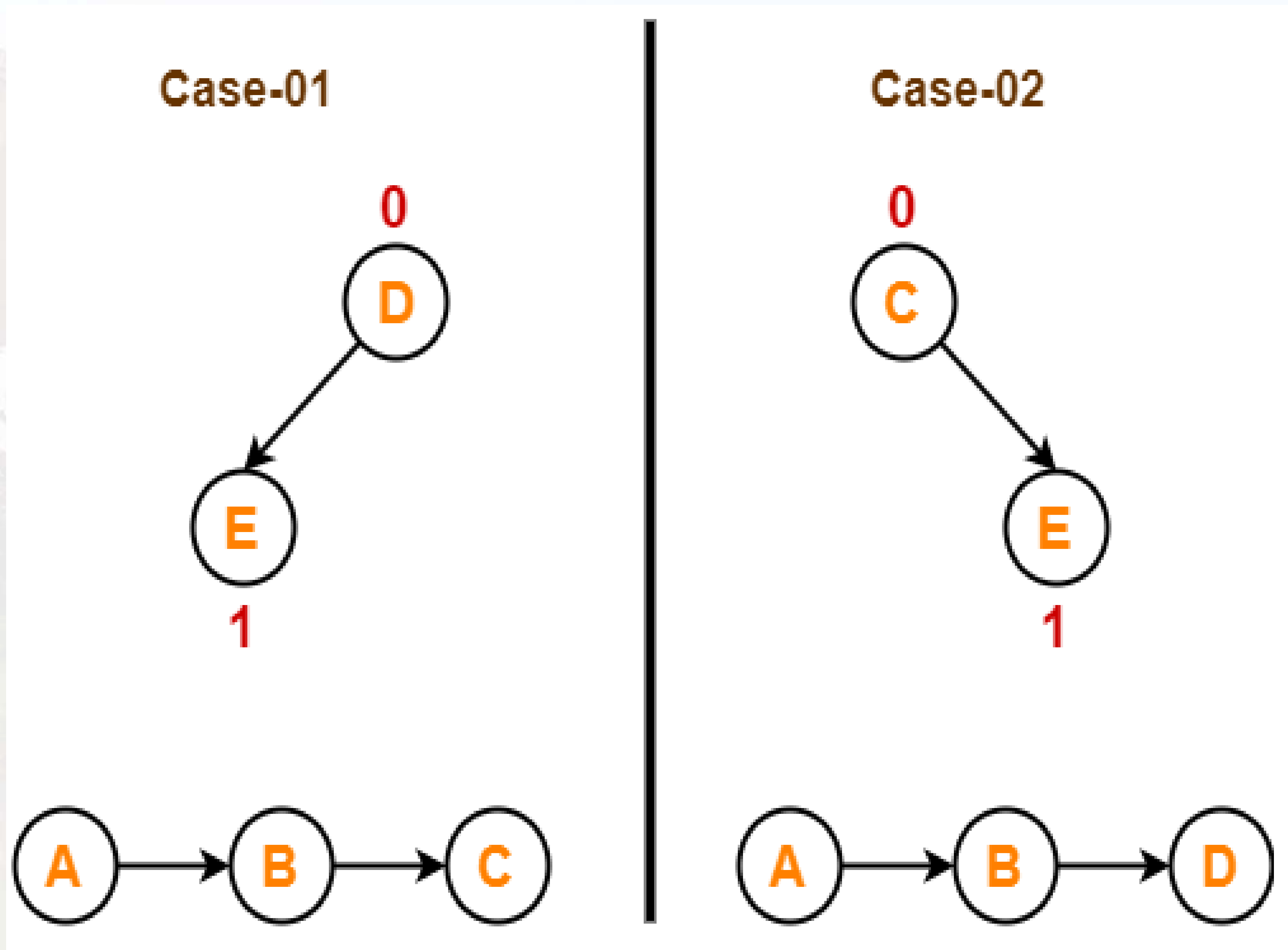
1. Remove vertex-C and its associated edges.
2. Then, update the in-degree of other vertices.

In case-02,

1. Remove vertex-D and its associated edges.
2. Then, update the in-degree of other vertices.

Topological Sort/Ordering

Step-04:



Topological Sort/Ordering

Step-05:

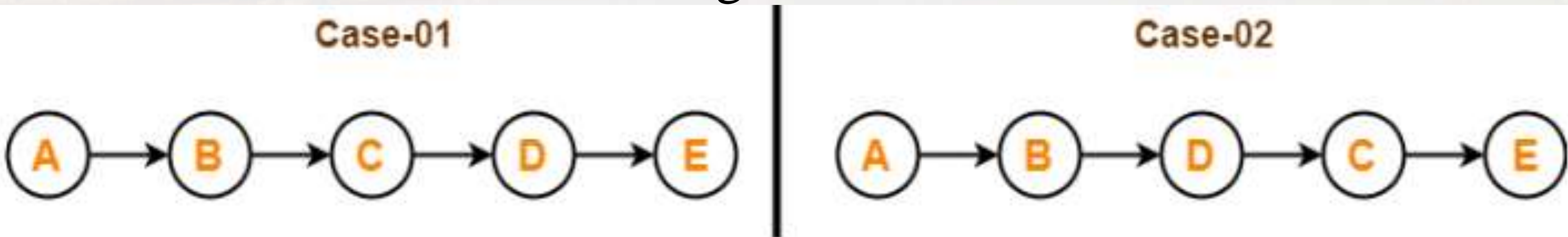
- Now, the above two cases are continued separately in the similar manner.

In case-01,

1. Remove vertex-D since it has the least in-degree.
2. Then, remove the remaining vertex-E.

- **In case-02,**

1. Remove vertex-C since it has the least in-degree.
2. Then, remove the remaining vertex-E.



Topological Sort/Ordering

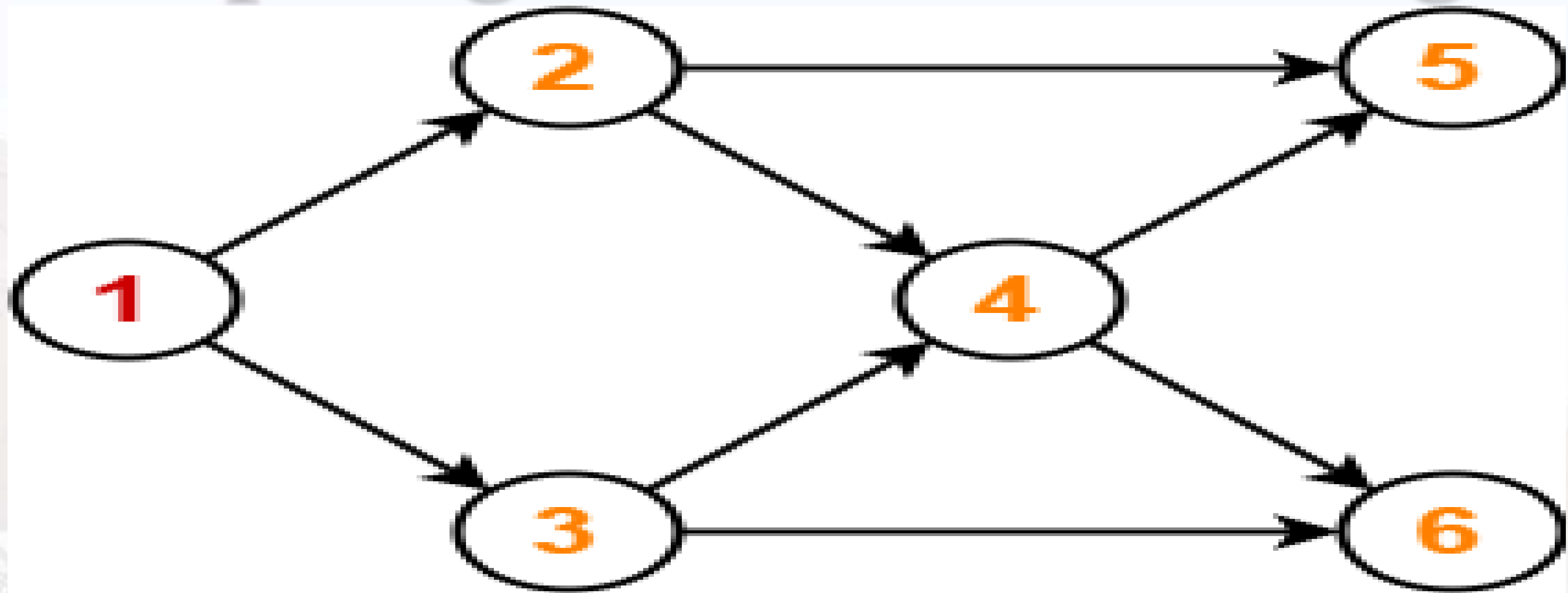
Conclusion-

For the given graph, following 2 different topological orderings are possible-

A B C D E

A B D C E

Topological Sort/Ordering



Folowing are the 4 different Topological order :

1 2 3 4 5 6

1 2 3 4 6 5

1 3 2 4 5 6

1 3 2 4 6 5

https://www.gatevidyalay.com/topological-sort-topological-sorting/

Applications of Topological Sort/Ordering

- Few important applications of topological sort are-

1. Scheduling jobs from the given dependencies among jobs
2. Instruction Scheduling
3. Determining the order of compilation tasks to perform in makefiles
4. Data Serialization

AOE Network and Critical Path

- **AOE(Activity on edge network):** in a weighted directed graph representing the project, the vertex represents the event, the directed edge represents the activity, and the weight on the edge represents the duration of the activity. Such a directed graph is called the edge represents the activity network, or AOE network for short
- The vertices in the AOE network without edges are called starting points (or source points); Points without edges are called endpoints (or sinks)

AOE Network and Critical Path

- **Properties of AOE networks**

1. Only after the event represented by a vertex occurs, the activity starting from the vertex can start;
2. The event represented by a vertex can occur only when all activities entering a vertex are completed.

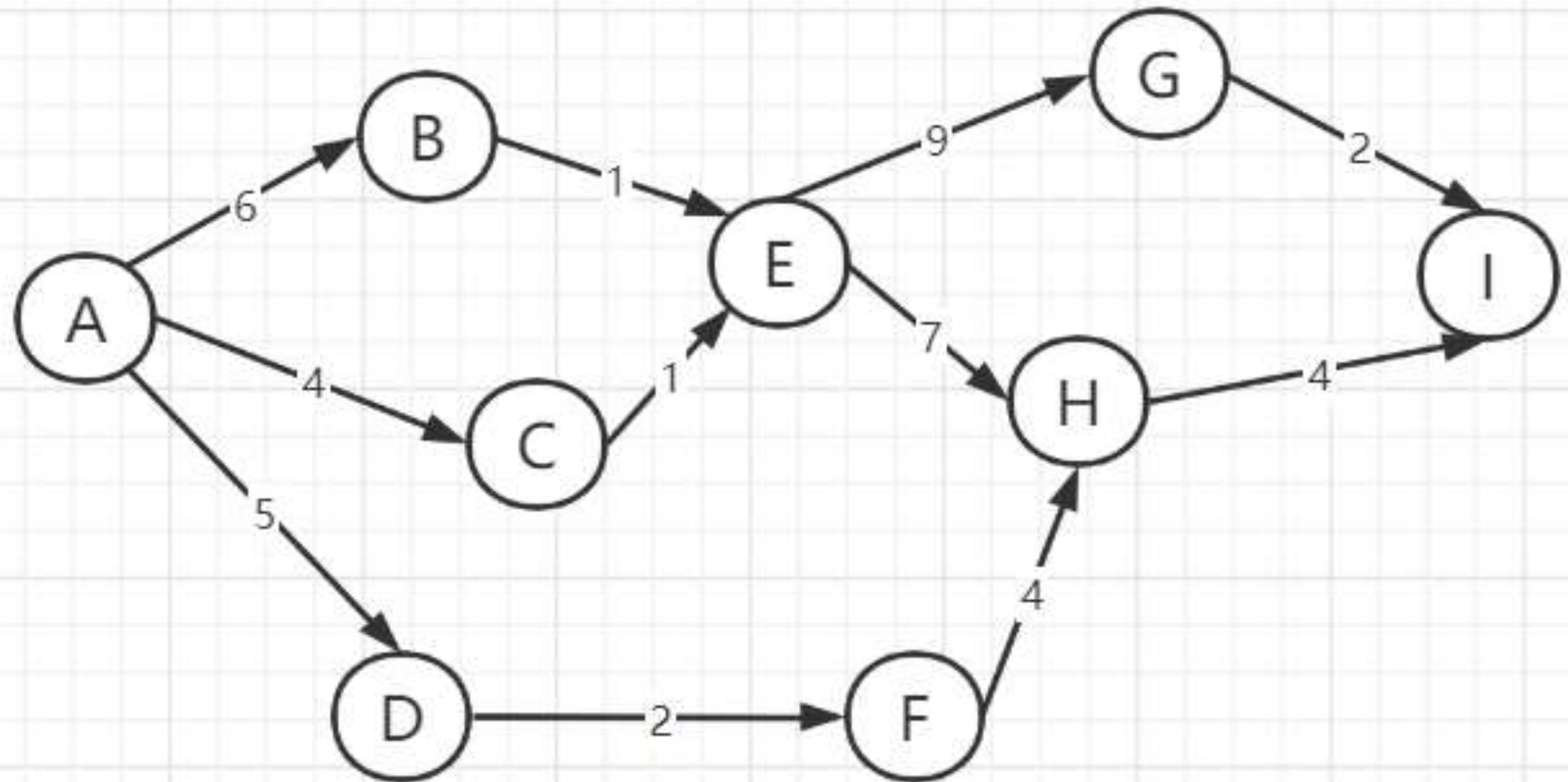
- **AOE network can solve the following problems**

1. At least how long will it take to complete the whole project
2. What activities should be accelerated to shorten the time required to complete the project

AOE Network and Critical Path

- **Key activities**
- **Critical path:** there may be more than one activity that takes the longest time, so the most important thing is to find the activity that cannot be delayed is called critical activity
- If the time of an activity is shortened and the overall end time cannot be changed, the activity is not a key activity; If you shorten the time of an activity, the activity is the key activity.

AOE Network and Critical Path



THANK YOU!!!

My Blog : https://anandgharu.wordpress.com/

Email : gharu.anand@gmail.com