•BCA

SEMESTER – III

DATA STRUCTURES

UNIT– IV - Graph

Prof. Nirav Suthar

Prof. Ankita Shah

• GRAPH

- **Introduction**
- **Terms**
- **Sequential Representation of Graphs**
- **Linked Representation of Graphs**
- **Traversal Of Graphs**

  DFS

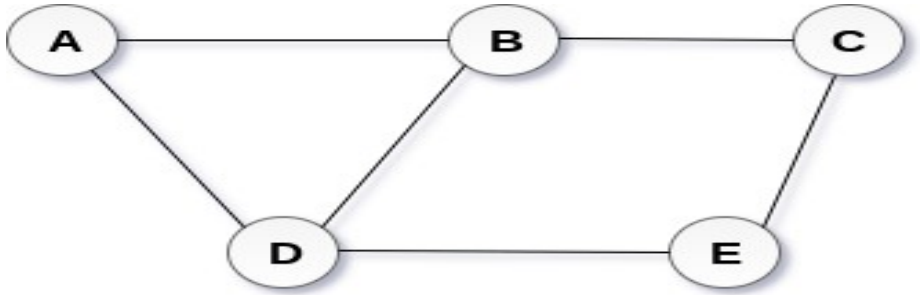
  BFS
- **Spanning Tree**
- **Minimal Spanning Tree**
- **Shortest Path**

• Introduction to Graph

- A graph can be defined as group of vertices and edges that are used to connect these vertices.
- A graph can be seen as a cyclic tree, where the vertices (Nodes) maintain any complex relationship among them instead of having parent child relationship.

• Introduction to Graph

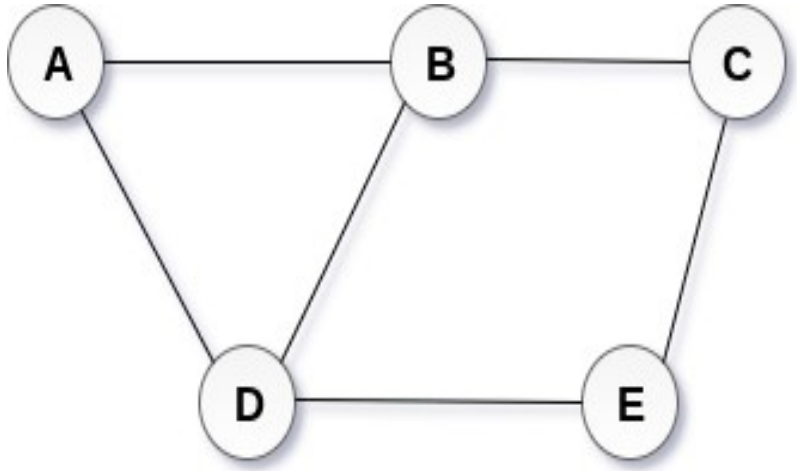
- **Definition**:
- A graph G can be defined as an ordered set G(V, E) where V(G) represents the set of vertices and E(G) represents the set of edges which are used to connect these vertices.
- A Graph G(V, E) with 5 vertices (A, B, C, D, E) and six edges ((A,B), (B,C), (C,E), (E,D), (D,B), (D,A)) is shown in the following figure.



Undirected Graph

• Types of Graph

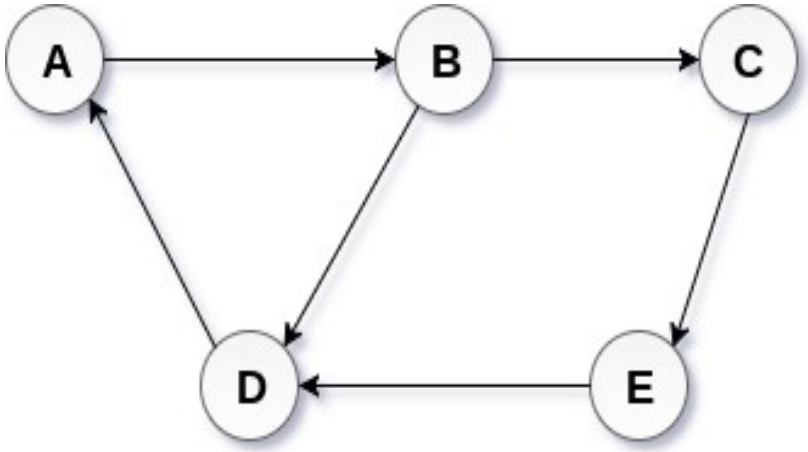
- A graph can be **directed or undirected.**
- **Undirected Graph:**
- In an undirected graph, edges are not associated with the directions with them.
- An undirected graph is shown in the figure since its edges are not attached with any of the directions. If an edge exists between vertex A and B then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



Undirected Graph

• Types of Graph

- **Directed Graph:**
- In a directed graph, edges form an ordered pair.
- Edges represent a specific path from some vertex A to another vertex B. Node A is called initial node while node B is called terminal node.



Directed Graph

• Graph Terminology

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.
- A path P can be written as P = {V0,V1,….,Vn) of length n from node U to V is defined as sequence of (n+1) nodes.

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 V0=Vn.

• Graph Terminology

Simple Path:

- A path P is known as a simple path if all the nodes in the path are distinct with an exception that V0 may be equal to Vn.
- If V0=Vn then the path is called a Closed Simple Path.

Cycle:

- A cycle can be defined as the path which has no repeated edges or vertices except the first and last vertices.
- A Closed Simple Path with length 3 or more is knows as Cycle. A cycle of length k is called a k-Cycle.

• Graph Terminology

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.

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.**

• Graph Terminology

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.

• Graph Terminology

Loop:

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

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.

• Graph Terminology

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.

Regular Graph:

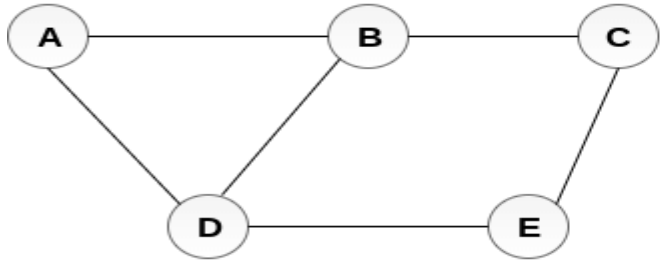
- It is a graph where **each vertex has the same number of neighbours**. That is every node has the same degree.
- A Regular graph with vertices of degree k is called a **k-regular graph or regular graph of degree k.**

• Sequential Representation of Graphs

- 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.

• Sequential Representation of Graphs

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



Undirected Graph

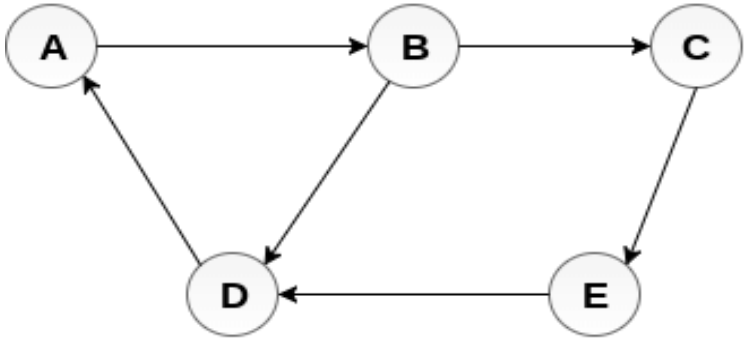
	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	1	0
C	0	1	0	0	1
D	1	1	0	0	1
E	0	0	1	1	0

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 Aij will be 1 only when there is an edge directed from Vi to Vj.

• Sequential Representation of Graphs

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



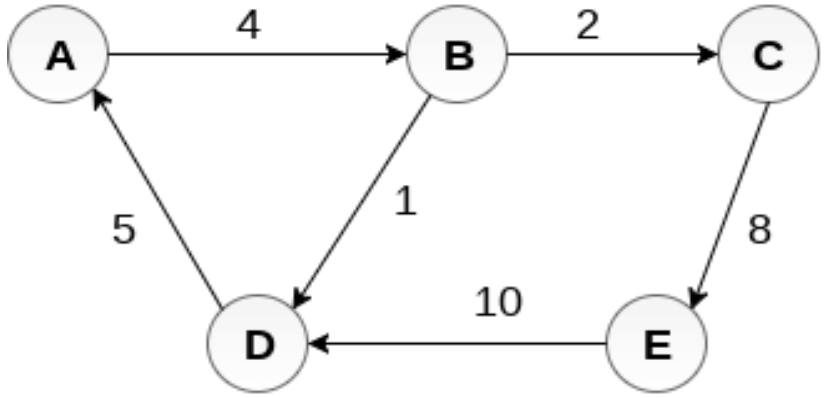
Directed Graph

	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

• Sequential Representation of Graphs

- 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.
- 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

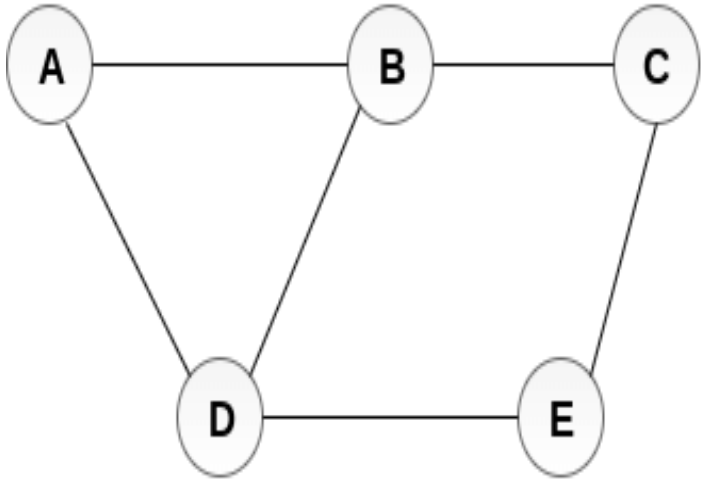
Adjacency Matrix

• Linked Representation of Graphs

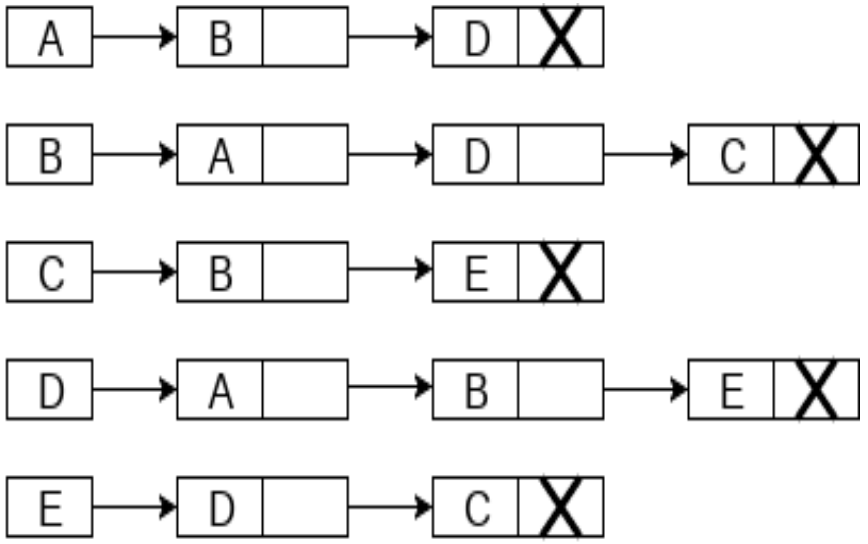
- In the linked representation, an adjacency list is used to store the Graph into the computer's memory.
- An adjacency list is maintained for each node present in the graph which stores the node value and a pointer to the next adjacent node to the respective node.
- If all the adjacent nodes are traversed 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.

• Linked Representation of Graphs

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



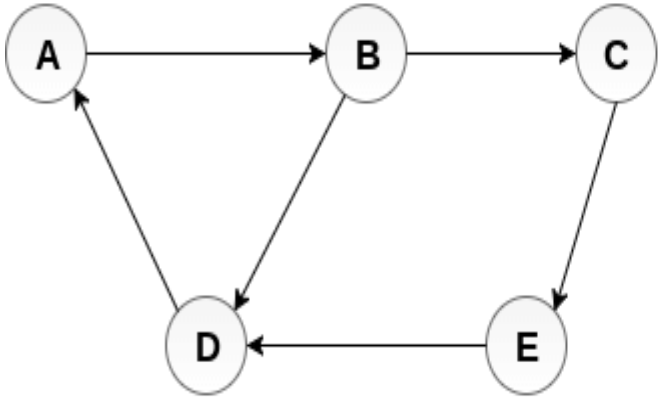
Undirected Graph



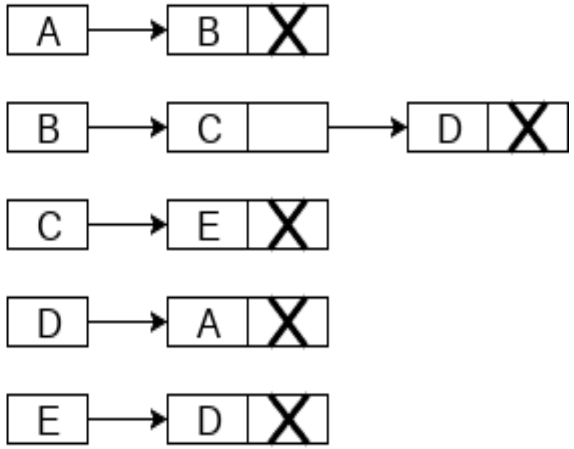
Adjacency List

• Linked Representation of Graphs

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



Directed Graph

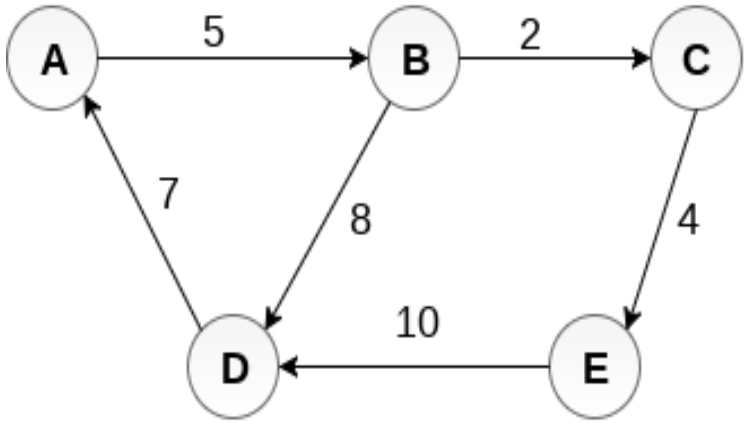


Adjacency List

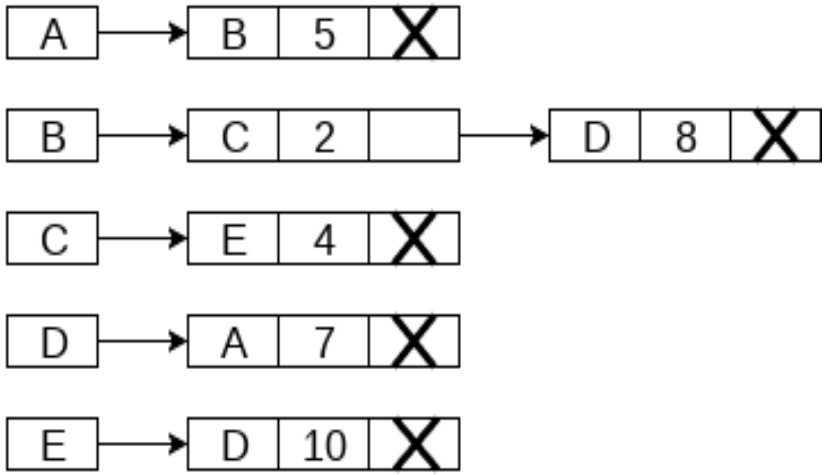
- In a directed graph, the sum of lengths of all the adjacency lists is equal to the number of edges present in the graph.

• Linked Representation of Graphs

- 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.



Weighted Directed Graph



Adjacency List

• Traversal of Graphs

- 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

- **Breadth First Search (BFS)**
- **Depth First Search (DFS)**

- **BFS** uses a queue as an auxiliary data structure to store nodes for further processing.
- **DFS** uses a stack as an auxiliary data structure.
- Both of these algorithms make use of a variable **STATUS.**

• Traversal of Graphs

- During the execution of the algorithm, every node in the graph will have the variable **STATUS** set to 1 or 2 or 3, depending on its current state.
- Following table shows the value of STATUS and its significance.

STATUS	STATE OF THE NODE	DESCRIPTION
1	**Ready**	The initial state of the node N
2	**Waiting**	Node N is placed on the queue or stack and waiting to be processed
3	**Processed**	Node N has been completely processed.

• Breadth First Search (BFS)

- Breadth first search is a graph traversal algorithm that **starts traversing the graph from root node and explores all the neighbouring nodes.**
- Then, it selects the nearest node and explore all the unexplored nodes.
- The algorithm follows the same process for each of the nearest node until it finds the goal.
- The algorithm starts with examining the node A and all of its neighbours.
- In the next step, the neighbours of the nearest node of A are explored and process continues in the further steps.
- **The algorithm explores all neighbours of all the nodes and ensures that each node is visited exactly once and no node is visited twice.**

• Breadth First Search (BFS)

Algorithm:

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)

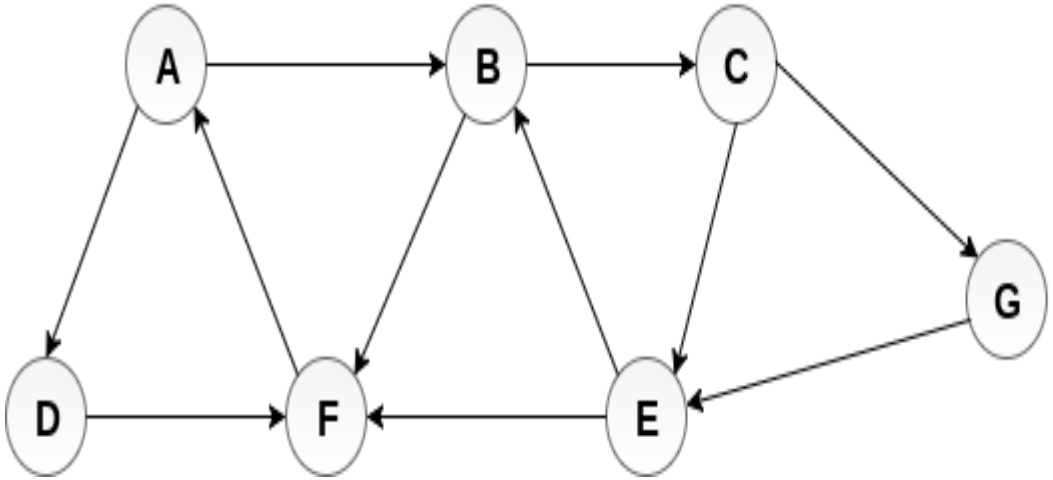
and set their STATUS = 2 (waiting state)

[END OF LOOP]

Step 6: END

• Breadth First Search (BFS)

Example: Consider the graph G shown in the following image, calculate the minimum path p from node A to node E. Given that each edge has a length of 1.



Adjacency Lists

A : B, D

B : C, F

C : E, G

G : E

E : B, F

F : A

D : F

• Breadth First Search (BFS)

Solution:

Minimum Path P can be found by applying breadth first search algorithm that will begin at node A and will end at E. the algorithm uses two queues, namely QUEUE1 and QUEUE2.

QUEUE1 holds all the nodes that are to be processed

QUEUE2 holds all the nodes that are processed and deleted from QUEUE1.

Lets start examining the graph from Node A.

• Breadth First Search (BFS)

1. Add A to QUEUE1 and NULL to QUEUE2.

```
QUEUE1 = {A}
QUEUE2 = {NULL}
```

2. Delete the Node A from QUEUE1 and insert all its neighbours. Insert Node A into QUEUE2

```
QUEUE1 = {B, D}
QUEUE2 = {A}
```

3. Delete the node B from QUEUE1 and insert all its neighbours. Insert node B into QUEUE2.

```
QUEUE1 = {D, C, F}
QUEUE2 = {A, B}
```

4. Delete the node D from QUEUE1 and insert all its neighbours. Since F is the only neighbour of it which has been inserted, we will not insert it again. Insert node D into QUEUE2.

```
QUEUE1 = {C, F}
QUEUE2 = { A, B, D}
```

• Breadth First Search (BFS)

5. Delete the node C from QUEUE1 and insert all its neighbours. Add node C to QUEUE2.

> QUEUE1 = {F, E, G}
> QUEUE2 = {A, B, D, C}

6. Remove F from QUEUE1 and add all its neighbours. Since all of its neighbours has already been added, we will not add them again. Add node F to QUEUE2.

> QUEUE1 = {E, G}
> QUEUE2 = {A, B, D, C, F}

7. Remove E from QUEUE1, all of E's neighbours has already been added to QUEUE1 therefore we will not add them again. All the nodes are visited and the target node i.e. E is encountered into QUEUE2.

> QUEUE1 = {G}
> QUEUE2 = {A, B, D, C, F, E}

Now, backtrack from E to A, using the nodes available in QUEUE2.

The minimum path will be **A** → **B** → **C** → **E**.

• Breadth First Search (BFS)

Applications of BFS:

- Finding all connected components in a graph G.
- Finding all nodes within an individual connected component.
- Finding the shortest path between two nodes, u and v of an unweighted graph.
- Finding the shortest path between two nodes, u and v of an weighted graph.

• 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.**

• Depth First Search (DFS)

Algorithm:

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)

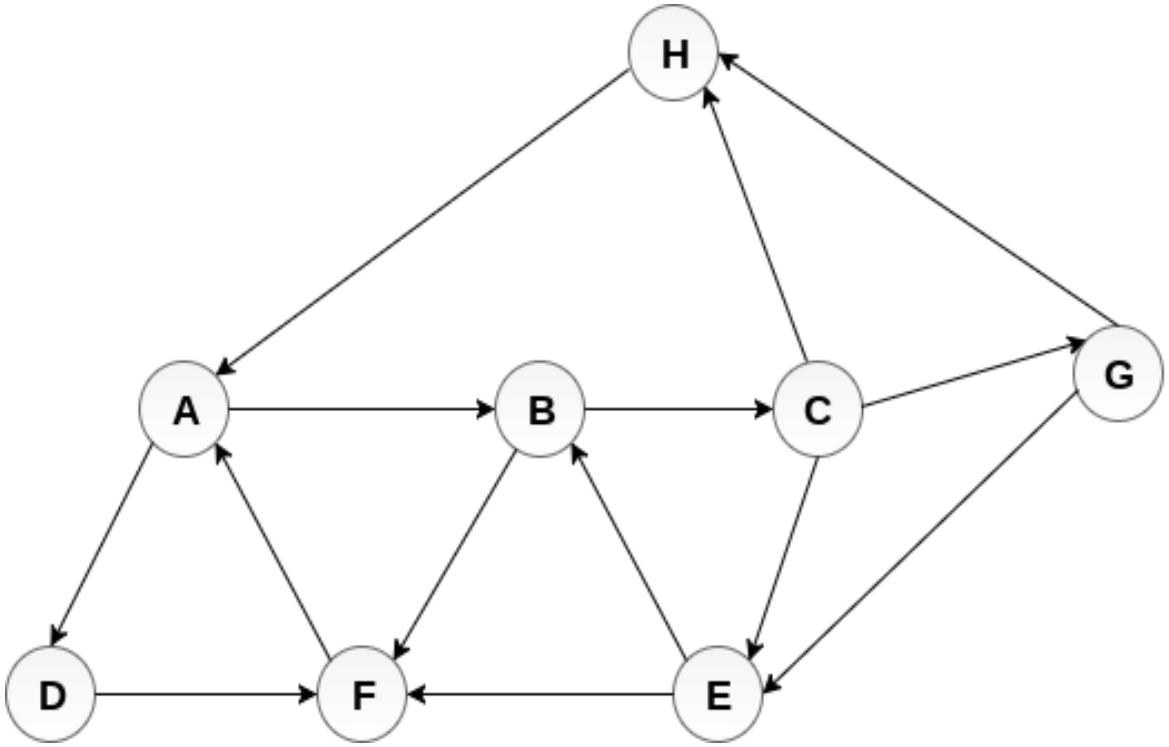
Step 5: Push on the stack all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

[END OF LOOP]

Step 6: END

• Depth First Search (DFS)

Example: Consider the graph G along with its adjacency list, given in the figure below. Calculate the order to print all the nodes of the graph starting from node H, by using depth first search (DFS) algorithm.



Adjacency Lists

A : B, D
B : C, F
C : E, G, H
G : E, H
E : B, F
F : A
D : F
H : A

• Depth First Search (DFS)

Push H onto the stack

> STACK : H

POP the top element of the stack i.e. H, print it and push all the neighbours of H onto the stack that are is ready state.

> Print H
> STACK : A

Pop the top element of the stack i.e. A, print it and push all the neighbours of A onto the stack that are in ready state.

> Print A
> Stack : B, D

Pop the top element of the stack i.e. D, print it and push all the neighbours of D onto the stack that are in ready state.

> Print D
> Stack : B, F

• Depth First Search (DFS)

Pop the top element of the stack i.e. F, print it and push all the neighbours of F onto the stack that are in ready state.

> Print F
> Stack : B

Pop the top of the stack i.e. B and push all the neighbours

> Print B
> Stack : C

Pop the top of the stack i.e. C and push all the neighbours.

> Print C
> Stack : E, G

Pop the top of the stack i.e. G and push all its neighbours.

> Print G
> Stack : E

• Depth First Search (DFS)

Pop the top of the stack i.e. E and push all its neighbours.

> Print E
>
> Stack :

Hence, the stack now becomes empty and all the nodes of the graph have been traversed.

The printing sequence of the graph will be :

> H → A → D → F → B → C → G → E

• Spanning Tree

- Spanning tree can be defined as a sub-graph of connected, undirected graph G that is a tree produced by removing the desired number of edges from a graph.
- In other words, Spanning tree is a non-cyclic sub-graph of a connected and undirected graph G that connects all the vertices together.
- A graph G can have multiple spanning trees.

• Minimal Spanning Tree

- There can be weights assigned to every edge in a weighted graph.
- However, A minimum spanning tree is a spanning tree which has minimal total weight.
- In other words, minimum spanning tree is the one which contains the least weight among all other spanning tree of some particular graph.

• Shortest path algorithms

- There are two algorithms which are being used for this purpose.

- **Prim's Algorithm**

- **Kruskal's Algorithm**

• Prim’s algorithm

- Prim's Algorithm is used to find the minimum spanning tree from a graph.
- Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.
- Prim's algorithm starts with the single node and explore all the adjacent nodes with all the connecting edges at every step.
- The edges with the minimal weights causing no cycles in the graph got selected.
- **Tree vertices:** Vertices that are a part of the minimum spanning tree T.
- **Fringe vertices:** Vertices that are currently not a part of T, but are adjacent to some tree vertex.
- **Unseen vertices:** Vertices that are neither tree vertices nor fringe vertices fall under this category.

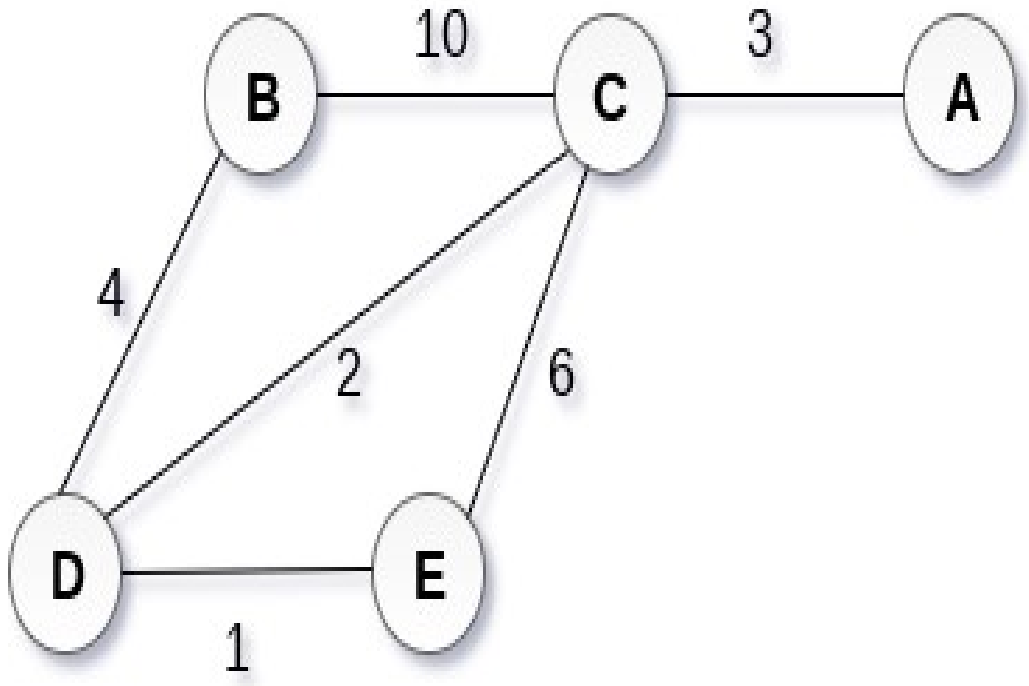
• Prim’s algorithm

- **Algorithm:**

Step 1: Select a starting vertex
Step 2: Repeat Steps 3 and 4 until there are fringe vertices
Step 3: Select an edge 'e' connecting the tree vertex and fringe vertex that has minimum weight
Step 4: Add the selected edge and the vertex to the minimum spanning tree T
[END OF LOOP]
Step 5: EXIT

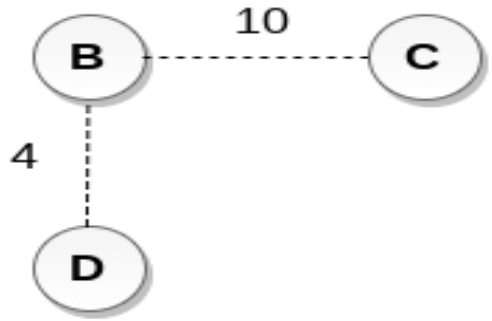
• Prim’s algorithm

- **Example:** Construct a minimum spanning tree of the graph given in the following figure by using prim's algorithm.

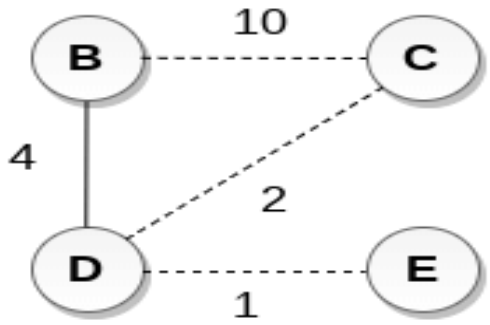


• Prim’s algorithm

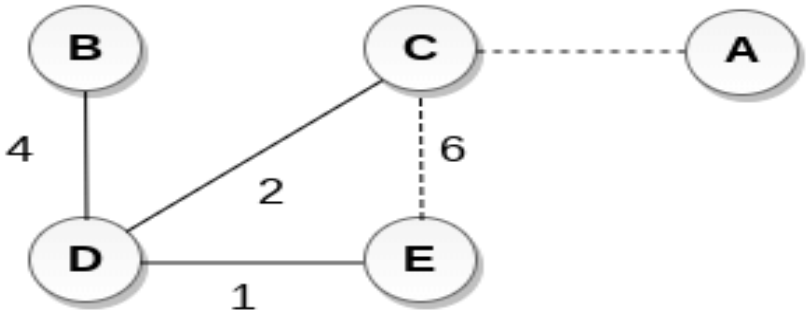
Step 1



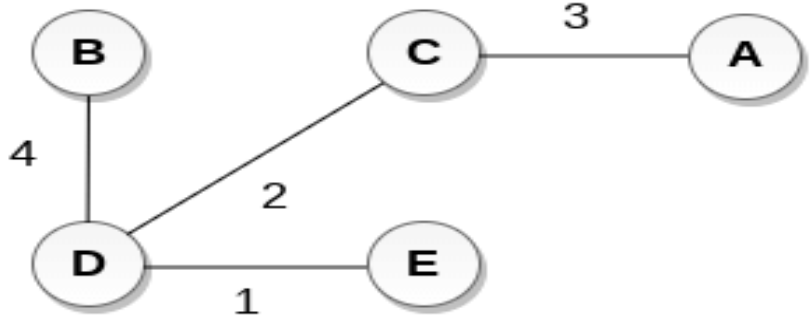
Step 2



Step 3



Step 4



Step 5

• Kruskal’s algorithm

- Kruskal's Algorithm is used to find the minimum spanning tree for a connected weighted graph.
- The main target of the algorithm is to find the subset of edges by using which, we can traverse every vertex of the graph.
- Kruskal's algorithm follows greedy approach which finds an optimum solution at every stage instead of focusing on a global optimum**.**

• Kruskal’s algorithm

- **Algorithm:**

Step 1: Create a forest F in such a way that every vertex of the graph is a separate tree.

Step 2: Create a set E that contains all the edges of the graph.

Step 3: Repeat Steps 4 and 5 while E is NOT EMPTY and F is not spanning

Step 4: Remove an edge from E with minimum weight

Step 5: IF the edge obtained in Step 4 connects two different trees, then add it to the forest F

(for combining two trees into one tree).

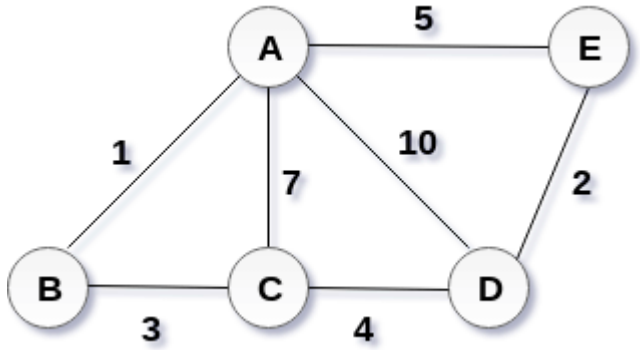
ELSE

Discard the edge

Step 6: END

• Kruskal’s algorithm

- Example:



Solution:

the weight of the edges given as :

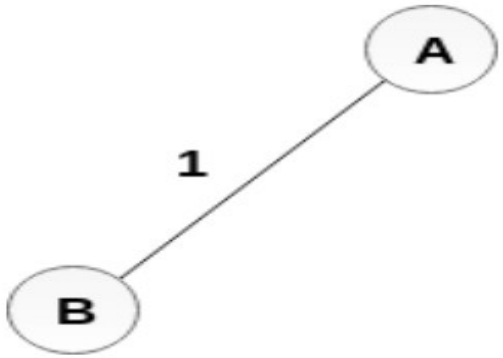
Edge	AE	AD	AC	AB	BC	CD	DE
Weight	5	10	7	1	3	4	2

Sort the edges according to their weights.

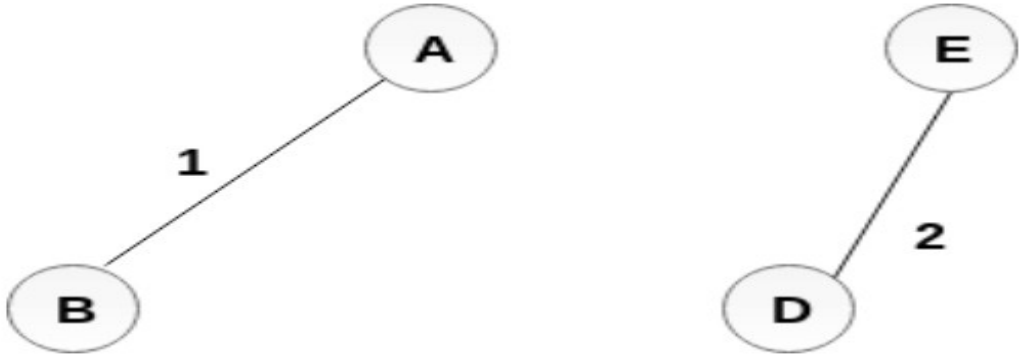
Edge	AB	DE	BC	CD	AE	AC	AD
Weight	1	2	3	4	5	7	10

• Kruskal’s algorithm

Add AB to the MST;

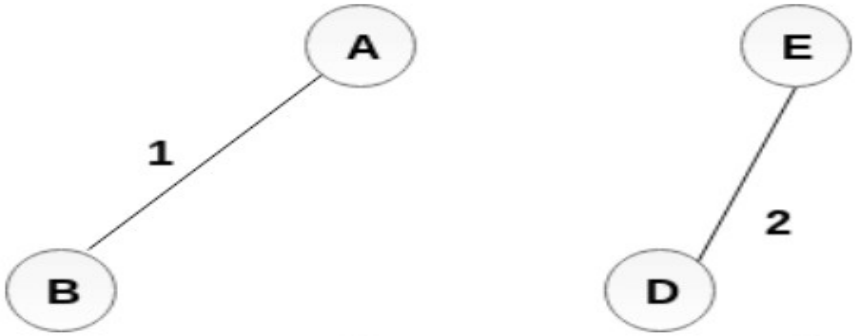


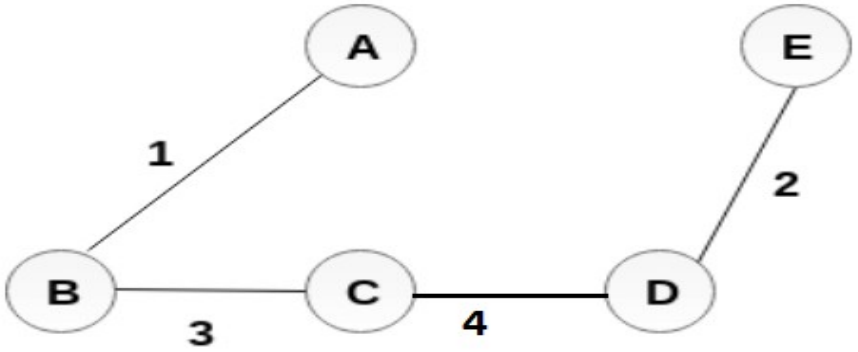
Add DE to the MST;



• Kruskal’s algorithm

Add BC to the MST;





The next step is to add AE, but we can't add that as it will cause a cycle.

The next edge to be added is AC, but it can't be added as it will cause a cycle.

The next edge to be added is AD, but it can't be added as it will contain a cycle.

Hence, the final MST is the one which is shown in the step 4.