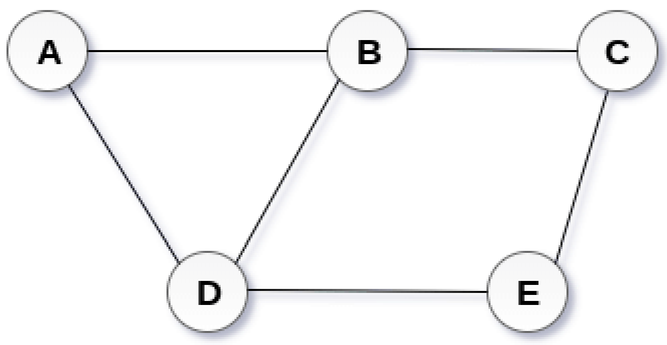
Graph

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.

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.



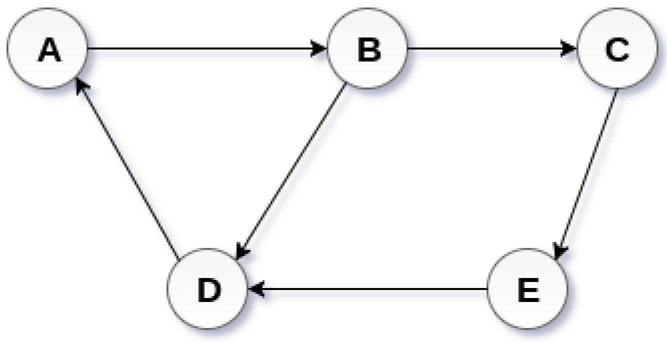
Undirected Graph

Directed and Undirected 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.

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.

A directed graph is shown in the following figure.



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.

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 $V_0=V_N$.

Simple Path

If all the nodes of the graph are distinct with an exception $V_0=V_N$, then such path P is called as 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.

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.

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.

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.

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.

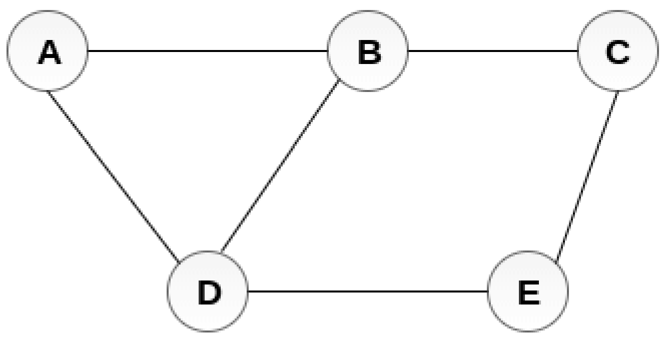
There are two ways to store Graph into the computer's memory. In this part of this tutorial, we discuss each one of them in detail.

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 M_{ij} in the adjacency matrix representation of an undirected graph G will be 1 if there exists an edge between V_i and V_j.

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



Undirected Graph

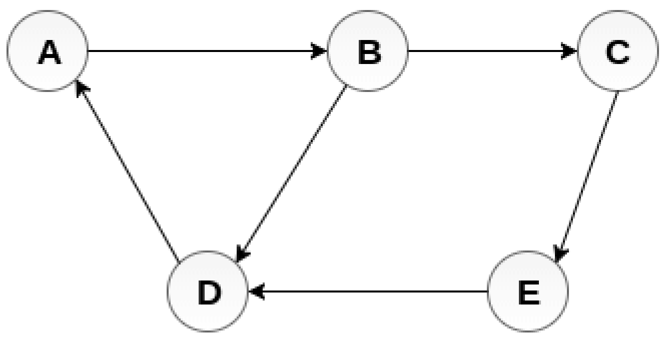
$$\begin{array}{c|ccccc|}
 & A & B & C & D & E \\
A & 0 & 1 & 0 & 1 & 0 \\
B & 1 & 0 & 1 & 1 & 0 \\
C & 0 & 1 & 0 & 0 & 1 \\
D & 1 & 1 & 0 & 0 & 1 \\
E & 0 & 0 & 1 & 1 & 0
\end{array}$$

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.

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



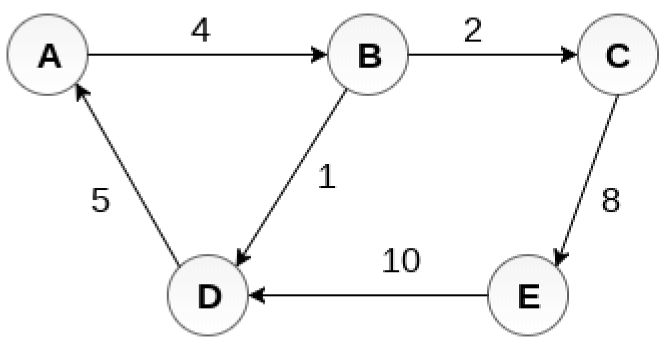
Directed Graph

$$\begin{array}{c|ccccc|}
 & A & B & C & D & E \\
A & 0 & 1 & 0 & 0 & 0 \\
B & 0 & 0 & 1 & 1 & 0 \\
C & 0 & 0 & 0 & 0 & 1 \\
D & 1 & 0 & 0 & 0 & 0 \\
E & 0 & 0 & 0 & 1 & 0
\end{array}$$

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.

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



Weighted Directed Graph

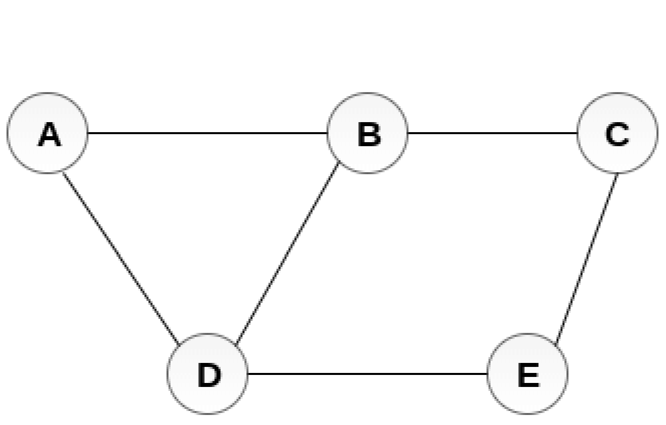
$$\begin{array}{c|ccccc|}
 & A & B & C & D & E \\
A & 0 & 4 & 0 & 0 & 0 \\
B & 0 & 0 & 2 & 1 & 0 \\
C & 0 & 0 & 0 & 0 & 8 \\
D & 5 & 0 & 0 & 0 & 0 \\
E & 0 & 0 & 0 & 10 & 0
\end{array}$$

Adjacency Matrix

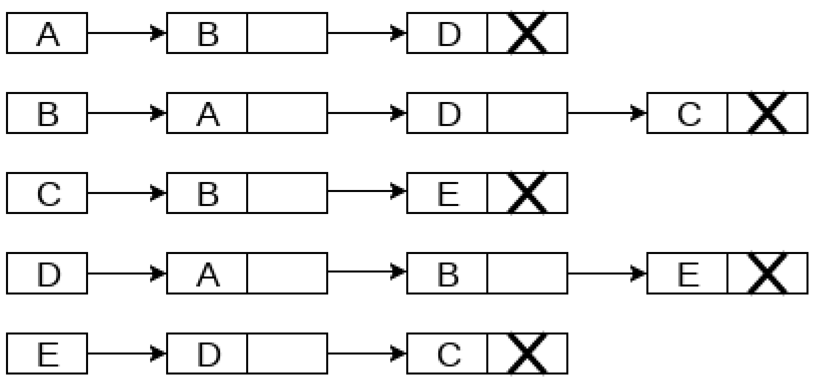
Linked Representation

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.



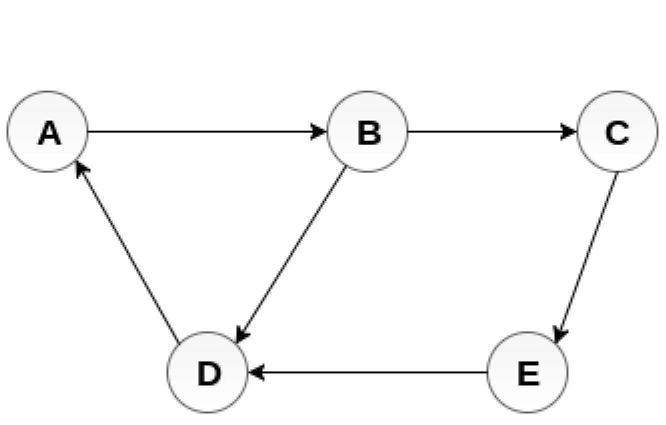
Undirected Graph



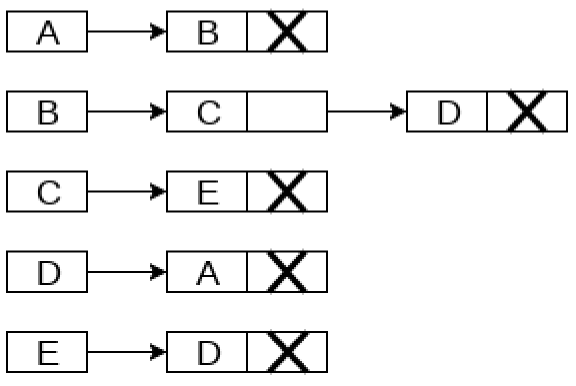
Adjacency List

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.



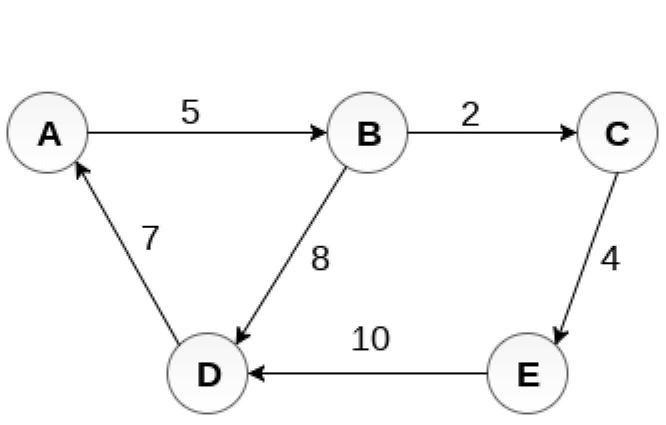
Directed Graph



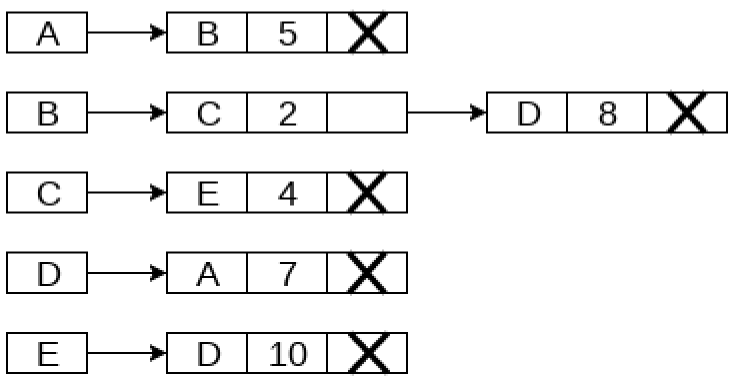
Adjacency List

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

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

Graph Traversal Algorithm

In this part of the tutorial we will discuss the techniques by using which, we can traverse all the vertices of the graph.

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.

- Breadth First Search
- Depth First Search

Breadth First Search (BFS) Algorithm

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.

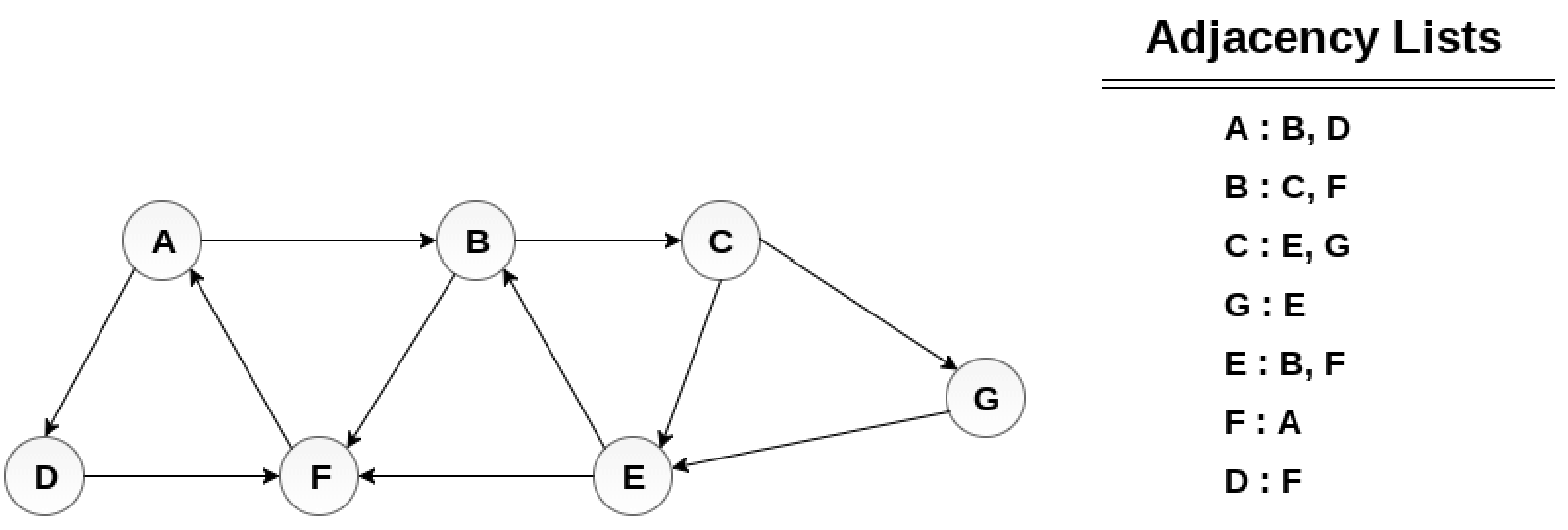
The algorithm of breadth first search is given below. The algorithm starts with examining the node A and all of its neighbours. In the next step, the neighbours of the nearest node of A are explored and process continues in the further steps. The algorithm explores all neighbours of all the nodes and ensures that each node is visited exactly once and no node is visited twice.

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:** EXIT

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

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

Lets start examining the graph from Node A.

1. Add A to QUEUE1 and NULL to QUEUE2.

1. QUEUE1 = {A}
2. QUEUE2 = {NULL}

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

1. QUEUE1 = {B, D}
2. QUEUE2 = {A}

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

1. QUEUE1 = {D, C, F}
2. 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.

1. QUEUE1 = {C, F}
2. QUEUE2 = { A, B, D}

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

1. QUEUE1 = {F, E, G}
2. 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.

1. QUEUE1 = {E, G}
2. 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.

1. QUEUE1 = {G}
2. 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**.

Depth First Search (DFS) Algorithm

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.

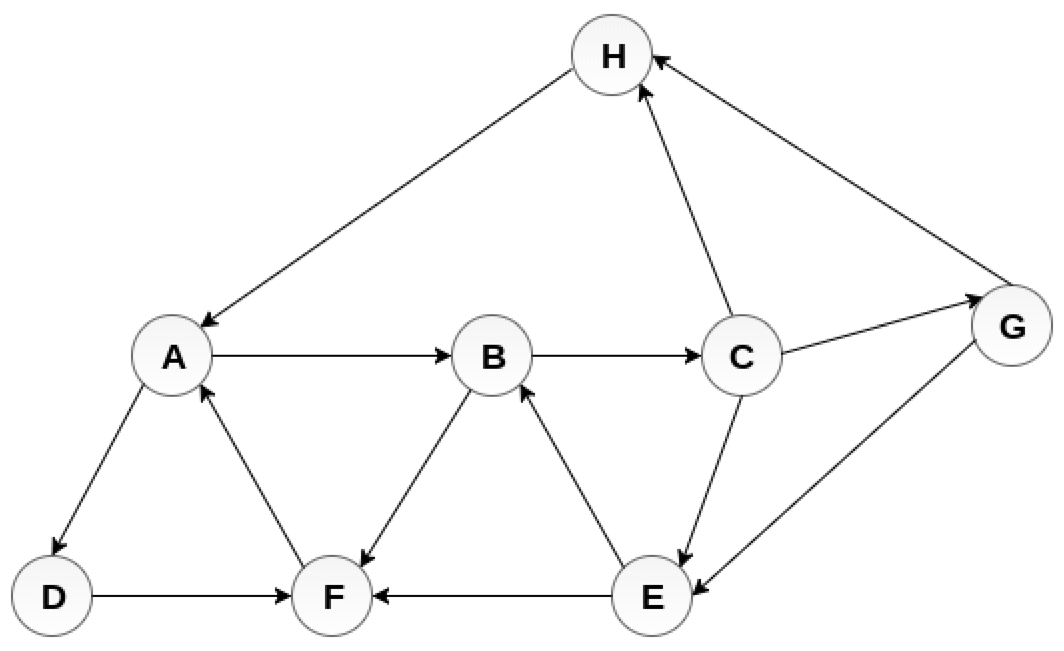
The data structure which is being used in DFS is stack. The process is similar to BFS algorithm. In DFS, the edges that leads to an unvisited node are called discovery edges while the edges that leads to an already visited node are called block edges.

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:** EXIT

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

Solution :

Push H onto the stack

1. 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.

1. Print H
2. 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.

1. Print A
2. 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.

1. Print D
2. Stack : B, F

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.

1. Print F
2. Stack : B

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

1. Print B
2. Stack : C

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

1. Print C
2. Stack : E, G

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

1. Print G
2. Stack : E

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

1. Print E
2. 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 :

1. H → A → D → F → B → C → G → E

Spanning Tree

In this topic, we will learn about the ***spanning tree*** and ***minimum spanning tree***. Before knowing about the spanning trees, we should know about some graphs.

The following are the types of graphs:

- **Undirected graph:** An undirected graph is a graph in which all the edges do not point to any particular direction, i.e., they are not unidirectional; they are bidirectional. It can also be defined as a graph in which set of V vertices and set of E edges, each edge connecting two different vertices.
- **Connected graph:** A connected graph is a graph in which a path always exists from a vertex to any other vertex. A graph is connected if we can reach any vertex from any other vertex by following edges in either direction.
- **Directed graph:** A directed graph is defined as a graph in which set of V vertices and set of Edges, each connecting two different vertices, but it is not mandatory that node points in the opposite direction also.

What is a Spanning tree?

If we have a graph containing V vertices and E edges, then the graph can be represented as:

G(V, E)

If we create the spanning tree from the above graph, then the spanning tree would have the same number of vertices as the graph, but the vertices are not equal. The edges in the spanning tree would be equal to the number of edges in the graph minus 1.

Suppose the spanning tree is represented as:

G\`(V\`, E\`)

where,

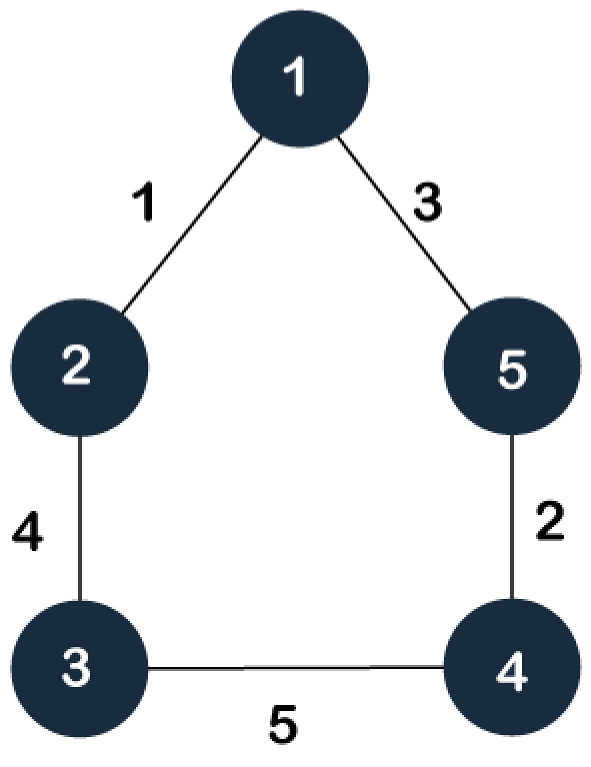
V=V\`

E\`є E -1

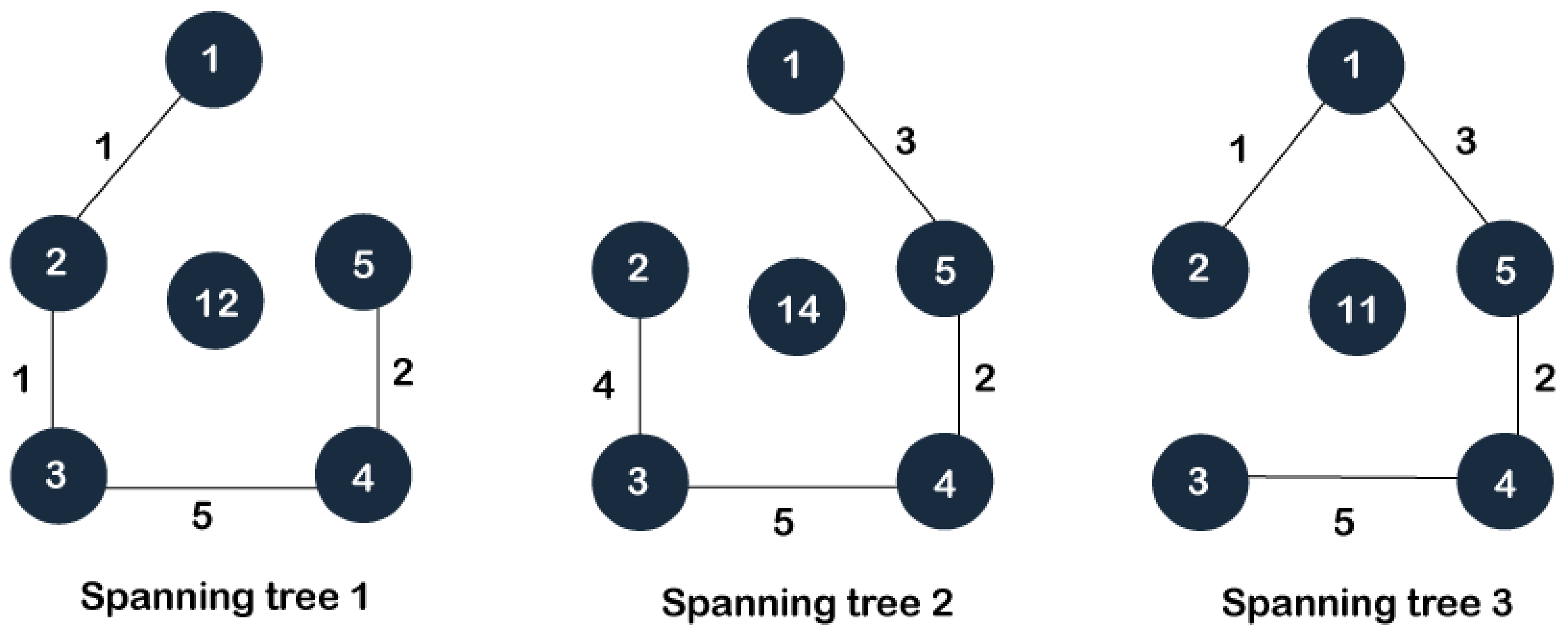
E\`=|V| - 1

Let's understand through an example.

Suppose we want to create the spanning tree of the graph, which is shown below:

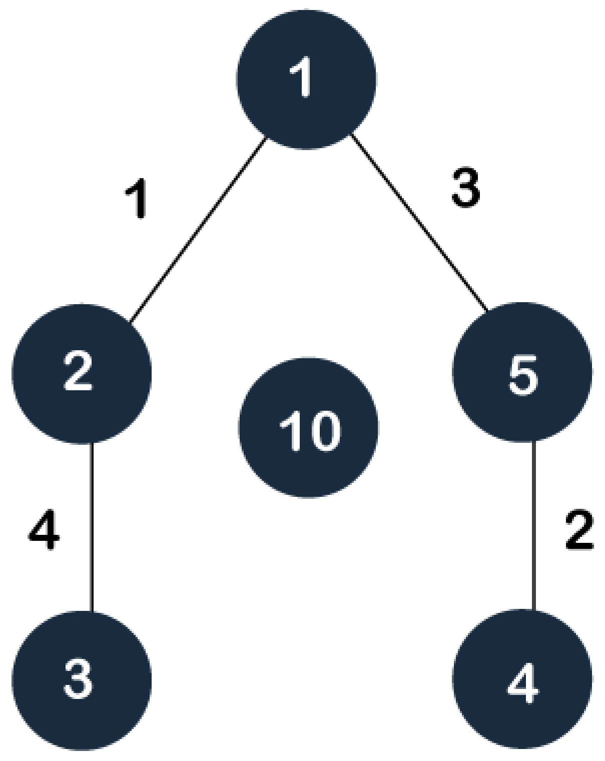


As we know, that spanning tree contains the same number of vertices as the graph, so the total number of vertices in the graph is 5; therefore, the spanning tree will also contain the 5 vertices. The edges in the spanning tree are equal to the number of vertices in the graph minus 1; therefore, the number of edges is 4. Three spanning trees can be created, which are shown below:



Minimum Spanning Trees

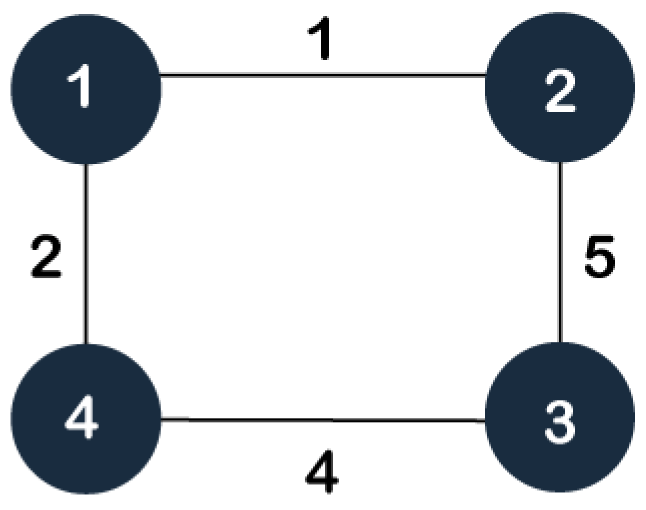
The minimum spanning tree is the tree whose sum of the edge weights is minimum. From the above spanning trees, the total edge weight of the spanning tree 1 is 12, the total edge weight of the spanning tree 2 is 14, and the total edge weight of the spanning tree 3 is 11; therefore, the total edge weight of the spanning tree 3 is minimum. From the above graph, we can also create one more spanning tree as shown below:



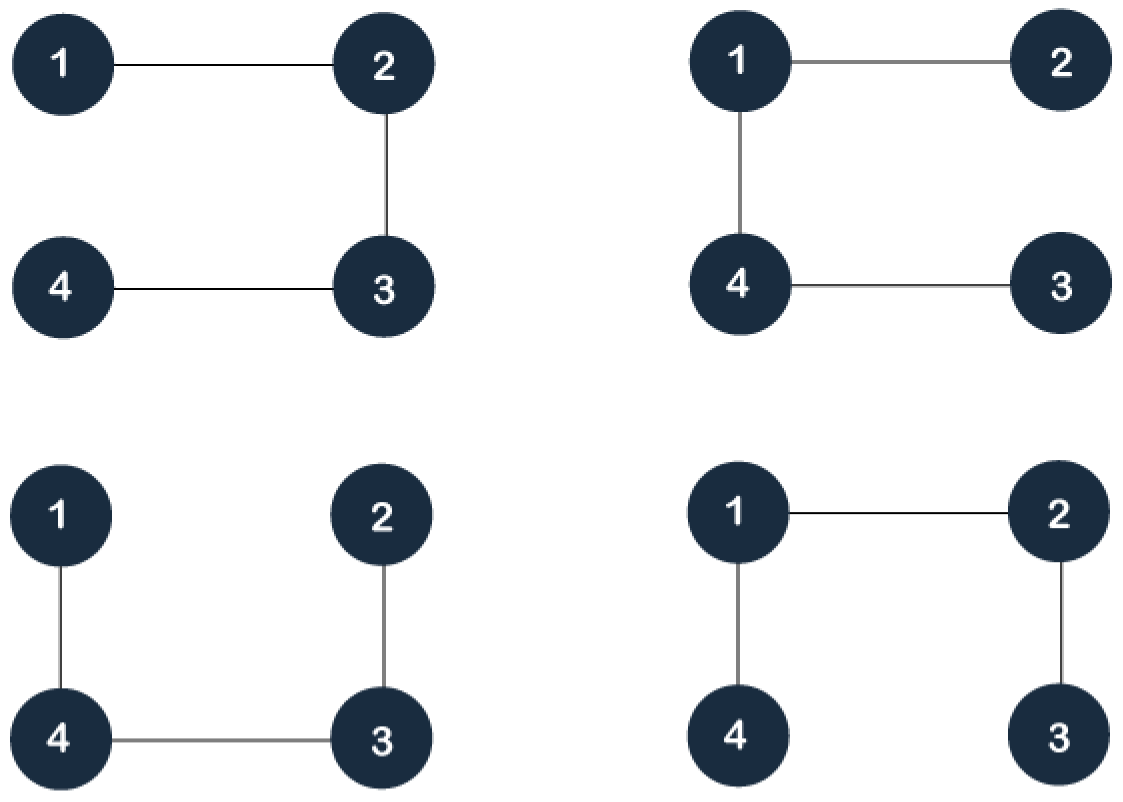
In the above tree, the total edge weight is 10 which is less than the above spanning trees; therefore, the minimum spanning tree is a tree which is having an edge weight, i.e., 10.

Properties of Spanning tree

- A connected graph can contain more than one spanning tree. The spanning trees which are minimally connected or we can say that the tree which is having a minimum total edge weight would be considered as the minimum spanning tree.
- All the possible spanning trees that can be created from the given graph G would have the same number of vertices, but the number of edges in the spanning tree would be equal to the number of vertices in the given graph minus 1.
- The spanning tree does not contain any cycle. Let's understand this property through an example. Suppose we have the graph which is given below:

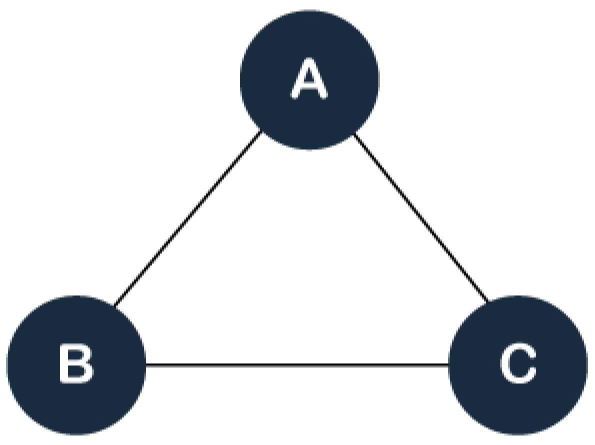


We can create the spanning trees of the above graph, which are shown below:

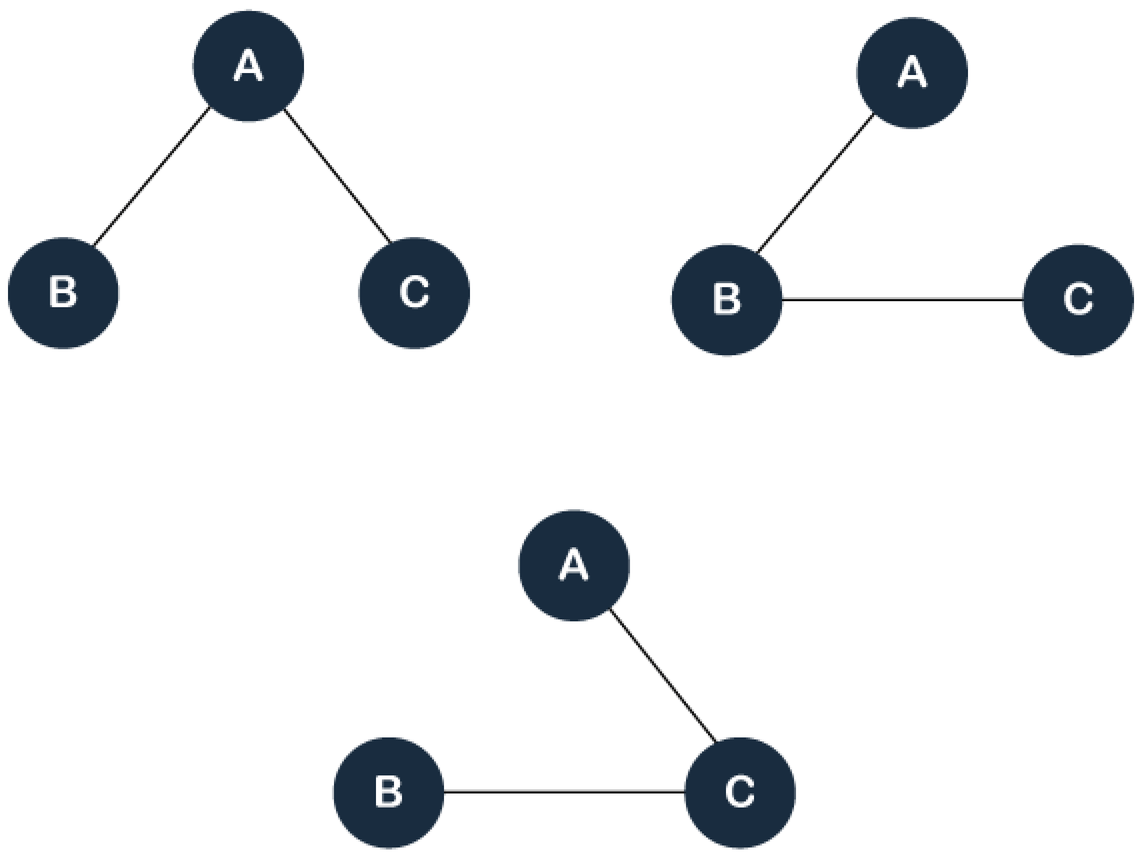


As we can observe in the above spanning trees that one edge has been removed. If we do not remove one edge from the graph, then the tree will form a cycle, and that tree will not be considered as the spanning tree.

- The spanning tree cannot be disconnected. If we remove one more edge from any of the above spanning trees as shown below:
  The above tree is not a spanning tree because it is disconnected now.
- If two or three edges have the same edge weight, then there would be more than two minimum spanning trees. If each edge has a distinct weight, then there will be only one or unique minimum spanning tree.
- A complete undirected graph can have $\mathbf{n^{n-2}}$ number of spanning trees where n is the number of vertices in the graph. For example, the value of n is 5 then the number of spanning trees would be equal to 125.
- Each connected and undirected graph contains at least one spanning tree.
- The disconnected graph does not contain any spanning tree, which we have already discussed.
- If the graph is a complete graph, then the spanning tree can be constructed by removing maximum (e-n+1) edges. Let's understand this property through an example.
  A complete graph is a graph in which each pair of vertices are connected. Consider the complete graph having 3 vertices, which is shown below:



We can create three spanning trees from the above graph shown as below:



According to this property, the maximum number of edges from the graph can be formulated as (e-n+1) where e is the number of edges, n is the number of vertices. When we substitute the value of e and n in the formula, then we get 1 value. It means that we can remove maximum 1 edge from the graph to make a spanning tree. In the above spanning trees, the one edge has been removed.

Applications of Spanning trees

The following are the applications of the spanning trees:

- **Building a network:** Suppose there are many routers in the network connected to each other, so there might be a possibility that it forms a loop. So, to avoid the formation of the loop, we use the tree data structure to connect the routers, and a minimum spanning tree is used to minimally connect them.
- **Clustering:** Here, clustering means that grouping the set of objects in such a way that similar objects belong to the same group than to the different group. Our goal is to divide the n objects into k groups such that the distance between the different groups gets maximized.

How can clustering be achieved?

First, we divide the n objects into k groups.

Secondly, we will combine these clusters iteratively by adding an edge between them.

Stop when we reach the k clusters.

Approach

One of the approaches that can be used to obtain clustering is to compute a minimum spanning tree. The minimum spanning tree can be simply calculated by dropping the k-1 most heavily weighted edge from the graph.