Graphs

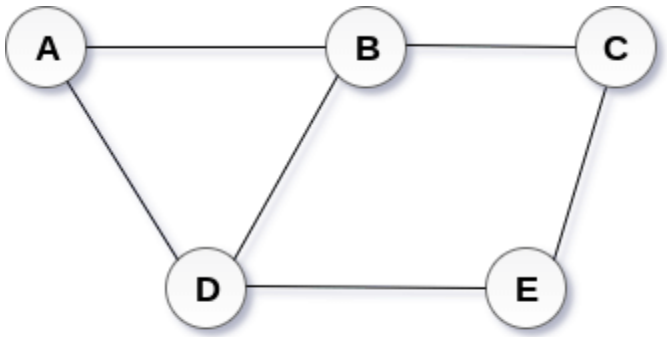
A graph G consists of set of vertices, V, and a set of edges, E. Each edge is a pair (v, w) , where v, w V.

- Edges are sometimes referred to as arcs.
- If the pair is ordered, then the graph is ***directed graph*** otherwise it is ***undirected graph.***
- Verted w is adjacent to v if and only if (v, w) E.
- Vertex v is adjacent to w and w is adjacent to v if (v, w) E in an undirected graph.

Applications of Graphs

1. **To represent the Networks (Electronic circuits, Transportation networks, Highway network, Flight network, Computer networks, etc)**
2. **For representing the dependency of tables in a database**

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.

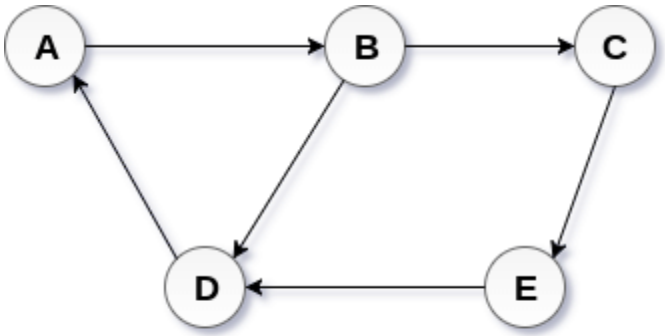


Undirected Graph

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.



Directed Graph

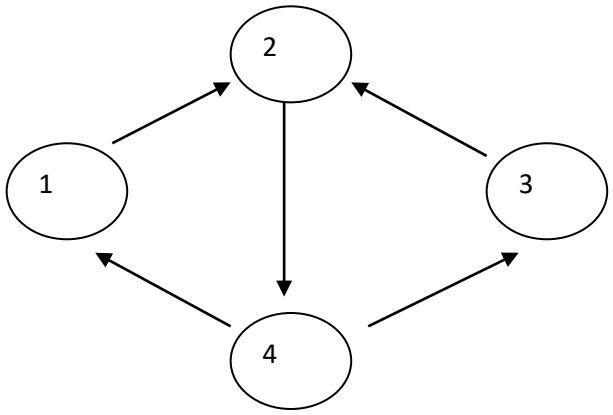
Representation of Graphs:

Graphs are represented using the following approaches

1. **Adjacency Matrix** – A matrix of size n X n is maintained, where n is the number of vertices. If there is an edge from vertex ***v*** to vertex ***w*** then adj_Mat[v, w] = 1, if it is a unweighted graph, otherwise fill with the cost of the edge. If there is no edge between v and w then set the value to 0. *(Used in dense graphs – Number of edges closed to the maximum possible number of edges)*
2. **Adjacency List** - In this representation all the vertices connected toa vertex ***v*** are listed on an adjacency list for that vertex ***v.*** *(Used in sparse graphs – Graphs with only fewer edges)*

3.Adjacency Set – *Similar to adjacency list but instead of using the Linked Lists Disjoint sets are used.*

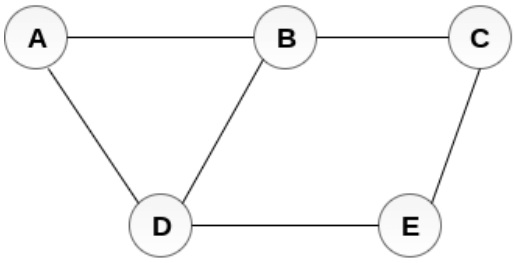
Example – Consider the following graph



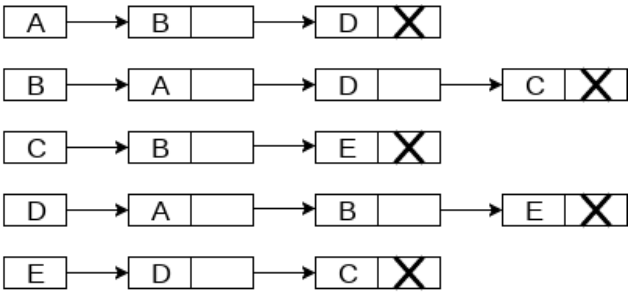
Adjacency matrix:

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

Adjacency List:



Undirected Graph



Adjacency List

Graph Traversals:

To work with the graphs we need a mechanism to visit the nodes in the graph, there exist two popular techniques for traversing the graphs:

1. Depth First Search (DFS)
2. Breadth First Search (BFS)

Depth First Search:

DFS works similar to the pre-order traversal technique of the tree. It works in the following manner

"***Starting at some vertex, v, we process v and then recursively all the vertices adjacent to v. To avoid cycles we need to remember which nodes are visited.***"

- By starting at vertex v it considers the edges from v to other vertices.
- If the edge leads to an already visited vertex, then backtrack to the current vertex v.
- If an edge leads to unvisited vertex, then go to that vertex and start processing that vertex.
- The new vertex becomes the current vertex.
- Follow this procedure until we reach dead-end. At this point do the backtracking.

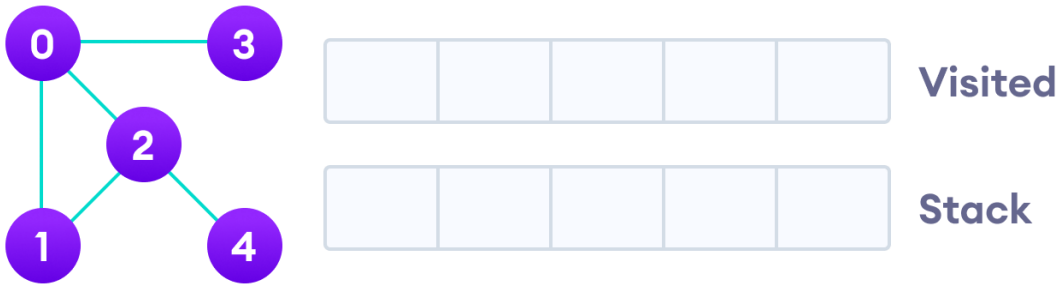
Algorithm

- **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

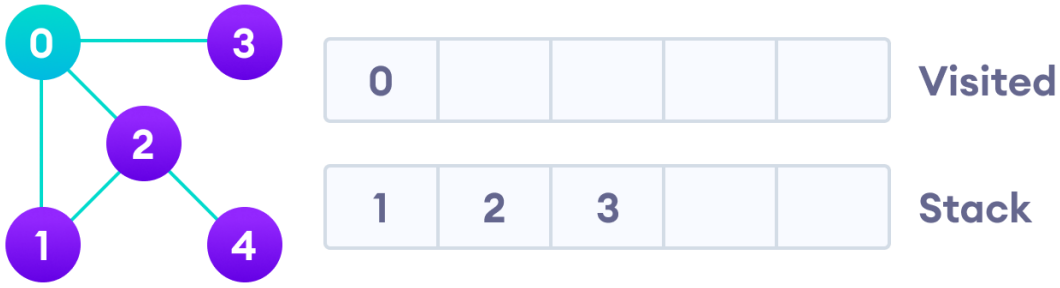
- **Step 4:** Pop the top node N. Process it and set its STATUS = 3 (processed state)
- **Step 5:** Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their
  STATUS = 2 (waiting state)
  [END OF LOOP]
- **Step 6:** EXIT

For Example:

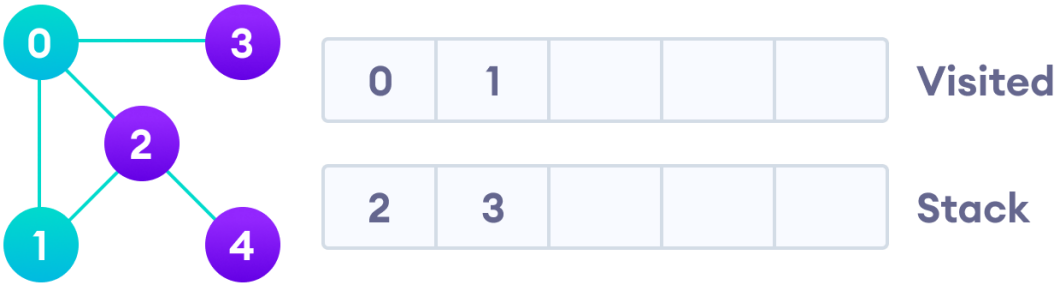
Let's see how the Depth First Search algorithm works with an example. We use an undirected graph with 5 vertices.



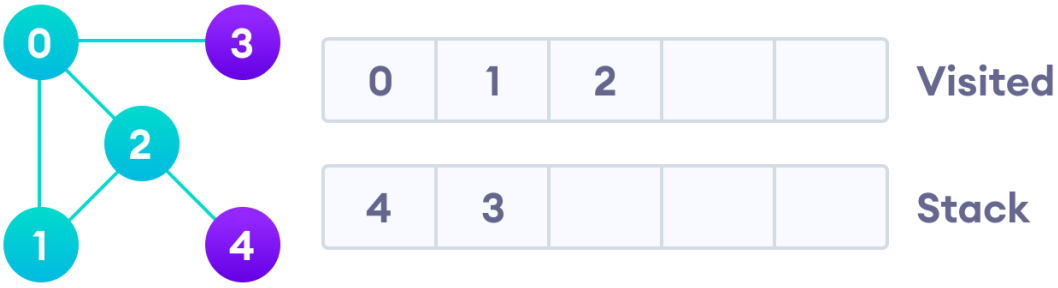
We start from vertex 0, the DFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.



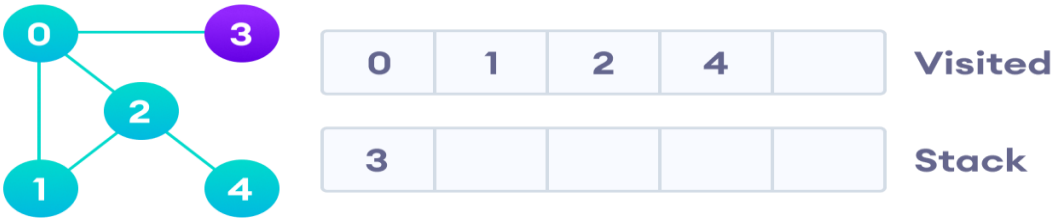
Next, we visit the element at the top of stack i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.



Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.

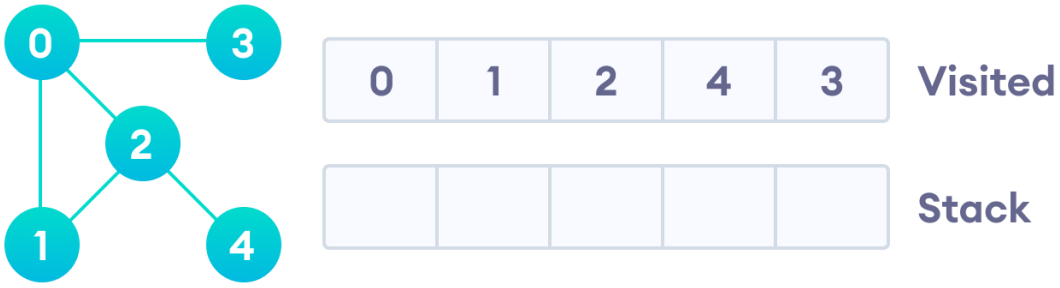


Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.



Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.

After we visit the last element 3, it doesn't have any unvisited adjacent nodes, so we have completed the Depth First Traversal of the graph.



Breadth First Search:

"Starting at some vertex, which is at level 0. We process v and then process all the vertices at level (vertices whose distance is one from the start vertex). In the second stage it process all the vertices in level 2 (these are vertices which are adjacent to level one). This process is continued till all the levels are processed. To avoid cycles we need to remember which nodes are visited."

BFS uses queue data structure.

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

Minimum Spanning Tree:

Minimum Spanning Tree of an undirected graph G is a tree formed from graph edges that connects all the vertices of G at lowest total cost.

MSP exists if and only if the graph is connected.

The following are the two algorithms exist to find the MSPs :

1. Prims Algorithm
2. Kruskal Algorithm

Both these algorithms works on the principle of *Greedy Techniques*

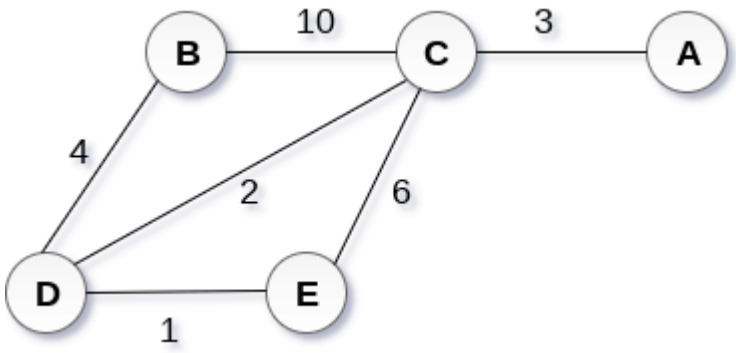
Prims algorithm:

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.

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



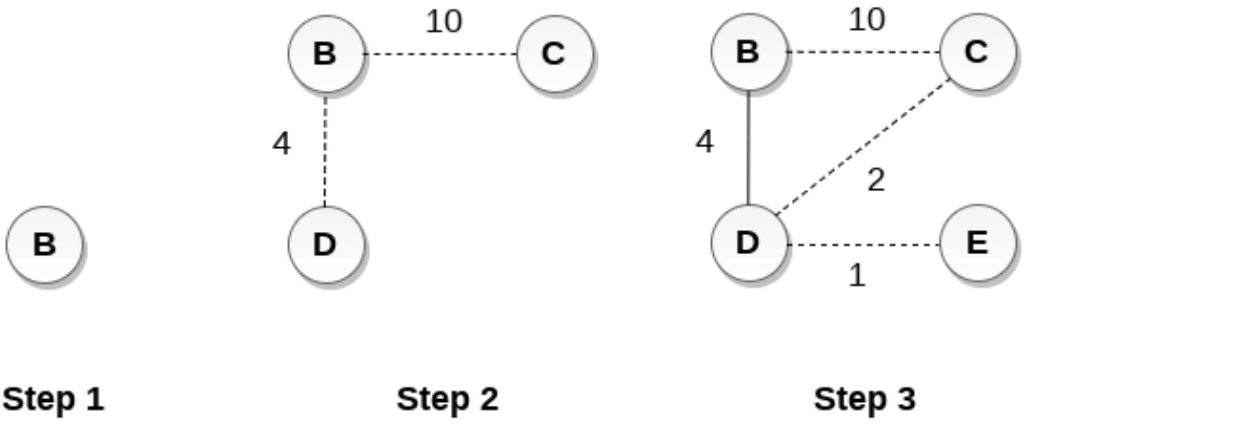
Solution

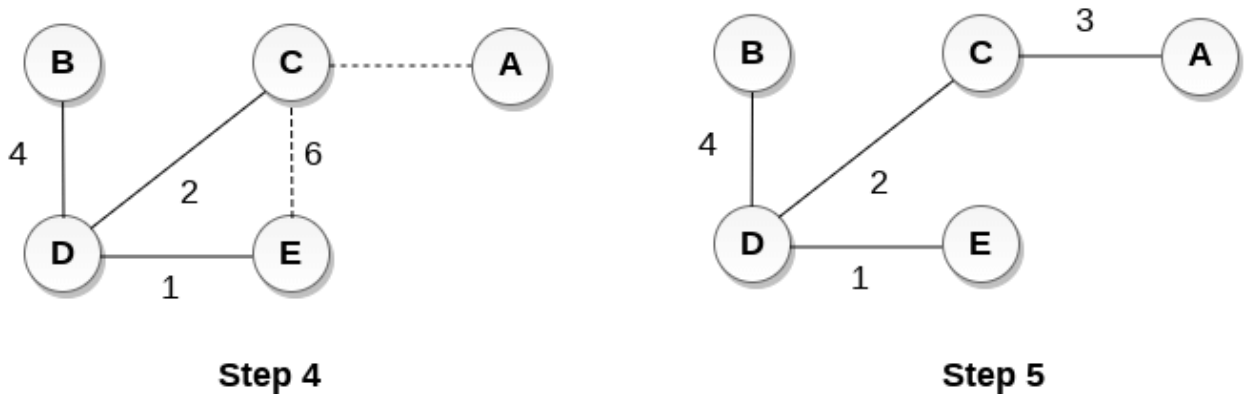
- **Step 1 :** Choose a starting vertex B.
- **Step 2:** Add the vertices that are adjacent to A. the edges that connecting the vertices are shown by dotted lines.
- **Step 3:** Choose the edge with the minimum weight among all. i.e. BD and add it to MST. Add the adjacent vertices of D i.e. C and E.
- **Step 3:** Choose the edge with the minimum weight among all. In this case, the edges DE and CD are such edges. Add them to MST and explore the adjacent of C i.e. E and A.
- **Step 4:** Choose the edge with the minimum weight i.e. CA. We can't choose CE as it would cause cycle in the graph.

The graph produces in the step 4 is the minimum spanning tree of the graph shown in the above figure.

The cost of MST will be calculated as;

cost(MST) = 4 + 2 + 1 + 3 = 10 units.





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.

The Kruskal's algorithm is given as follows.

Algorithm

- **Step 1:** Create a forest in such a way that each graph is a separate tree.
- **Step 2:** Create a priority queue Q that contains all the edges of the graph.
- **Step 3:** Repeat Steps 4 and 5 while Q is NOT EMPTY
- **Step 4:** Remove an edge from Q
- **Step 5:** IF the edge obtained in Step 4 connects two different trees, then Add it to the forest (for combining two trees into one tree).
  ELSE
  Discard the edge
- **Step 6:** END