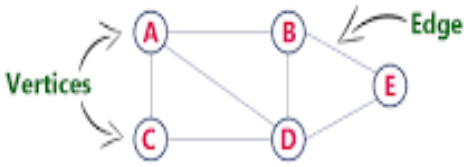
GRAPH

Introduction

Graph is a non **linear** data structure, it contains a set of points known as nodes (or vertices) and set of linkes known as edges (or Arcs) which connets the vertices. A graph is defined as follows... Generally, a graph G is represented as G = (V , E), where V is set of vertices and E is set of edges.



G={ A,B,C,D,E}

Comparison between Graph and Trees

BASIS FOR COMPARISON	TREE	GRAPH
Path	Only one between two vertices.	More than one path is allowed.
Root node	It has exactly one root node.	Graph doesn't have a root node.
Loops	No loops are permitted.	Graph can have loops.
Complexity	Less complex	More complex comparatively
Traversal techniques	Pre-order, In-order and Post-order.	Breadth-first search and depth-first search.
Number of edges	n-1 (where n is the number of nodes)	Not defined
Model type	Hierarchical	Network

Types of Graph

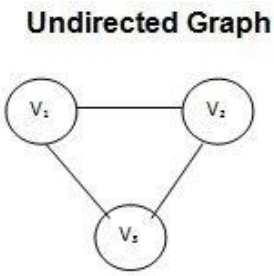


Figure 1: An Undirected Graph

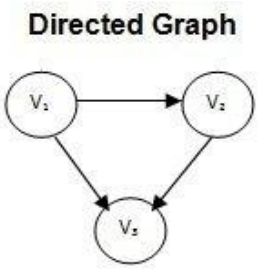
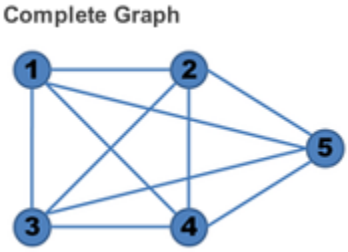


Figure 2: A Directed Graph

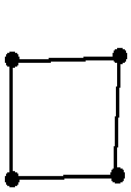
Complete Graph:

If an undirected graph of n vertices consists of n(n-1)/2 number of edges then that graph is called a complete graph.

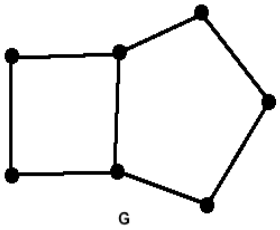


Subgraph:

A subgraph G' of graph G is a graph such that the set of vertices and set of edges of G' are proper subset of the set of edges of G.

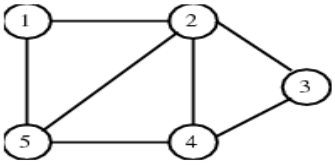


S
(Subgraph of G)

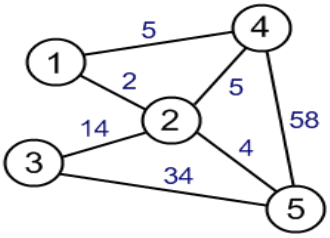


G

Connected graph:



Weighted graph:



Path:

A path is denoted using sequence of vertices and there exists an edge from one vertex to the next vertex.

Cycle:

A closed walk through the graph with repeated vertices, mostly having the same starting and ending vertex is called a cycle.

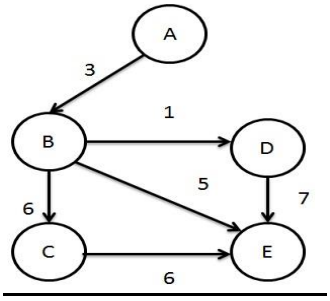
Component:

The maximal connected sub graph of a graph is called components of a graph.

In-degree and out-degree

In-degree of vertex is the number of edges that incident to that vertex.

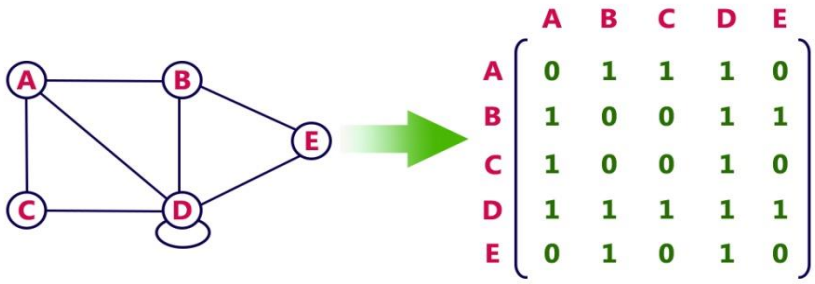
Out-degree of vertex is total number of edges that are going away from the vertex.



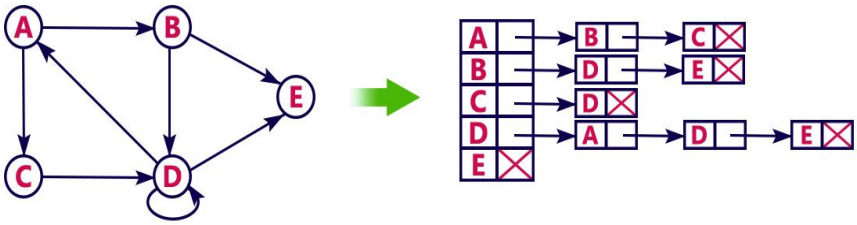
Vertices	In-degree	Out-degree
A	0	1
B	1	3
C	1	1
D	1	1
E	3	0

Representation of Graphs:

1. Adjacency Matrix Representation



2. Adjacency List Representation

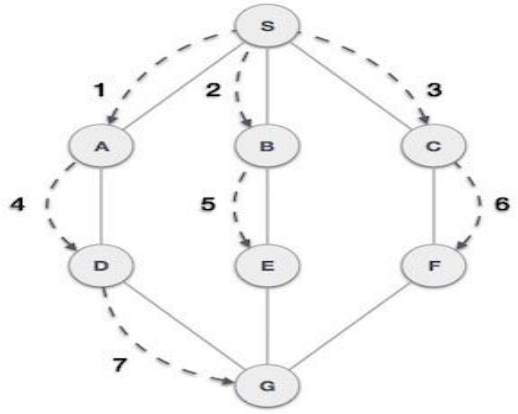


Display of Graph

Graph traversal (also known as graph search) refers to the process of visiting (checking and/or updating) each vertex in a graph. Such traversals are classified by the order in which the vertices are visited.

1. Breath first search
2. Depth first search

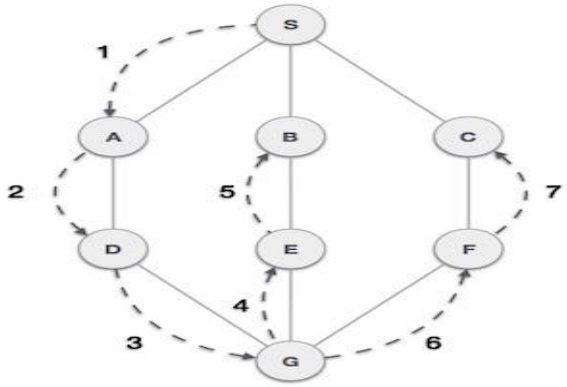
1. Breadth First Search (BFS) algorithm traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



As in the example given above, 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- S A B C D E F G

2. 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.



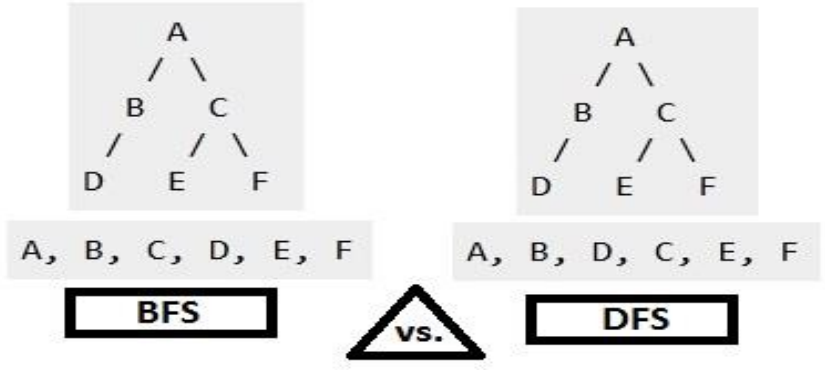
As in the example given above, DFS algorithm traverses from S to A to D to G to E to B first, then to F and lastly to C. It employs the following rules.

- 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-S A D G E B F C

Difference between BFS and DFS

Sr.No	BFS	DFS
1.	BFS visit nodes **level by level** in Graph.	DFS visit nodes of graph **depth wise**. It visits nodes until reach a leaf or a node which doesn't have non-visited nodes.
2.	A node is fully explored before any other can begin.	Exploration of a node is suspended as soon as another unexplored is found.
3.	Uses Queue data structure to store Un-explored nodes.	Uses Stack data structure to store Un-explored nodes.
4.	BFS is slower and require more memory.	DFS is faster and require less memory.
5.	Some Applications: ▪ Finding all connected components in a graph. ▪ Finding the shortest path between two nodes. ▪ Finding all nodes within one connected component.	Some Applications: ▪ Topological Sorting. ▪ Finding connected components. ▪ Solving puzzles such as maze. ▪ Finding strongly connected components. ▪ Finding articulation points (cut vertices) of the graph.

Example:



Spanning Trees:

A spanning tree S is a subset of a tree T in which all the vertices of tree T are present but it may not contain all the edges.

Minimum Spanning Trees:

The minimum spanning tree of a weighted connected graph G is called minimum spanning tree if its weight is minimum.

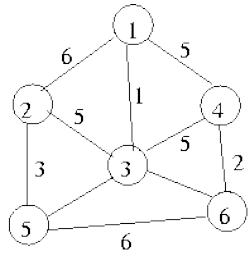
Prim's algorithm

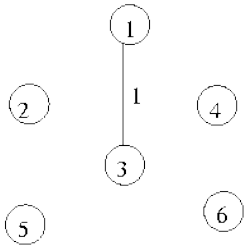
Step-1: Select the pair with minimum weight

Step-2: Select the adjacent vertex and select the minimum weighted edge using this adjacent vertex. The selected vertex should not form the circuit.

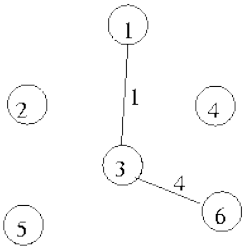
Step-3: Repeat step 1 and 2 until all the vertices are getting covered.

Example

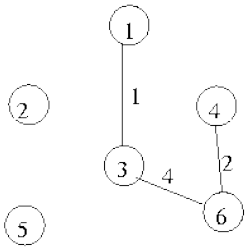




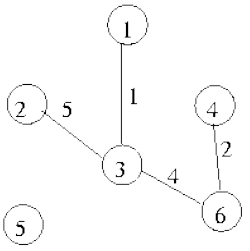
Iteration 1. U = {1}



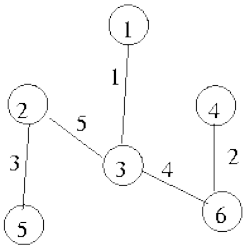
Iteration 2. U = {1,3}



Iteration 3. U = {1,3,6}



Iteration 4. U = {1,3,6,4}



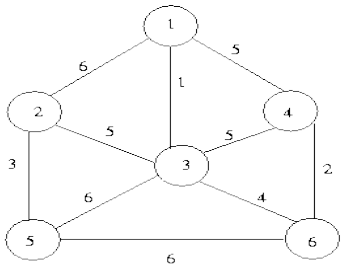
Iteration 5. U = {1,3,6,4,2}

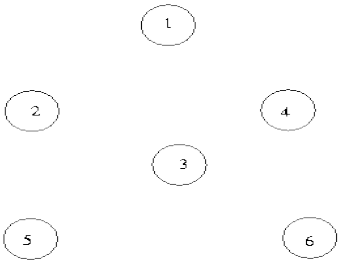
Kruskal's algorithm

Step-1: Select the pair with minimum weight each time and make the union of the selected edges. The selected adjacent vertex should not form the circuit.

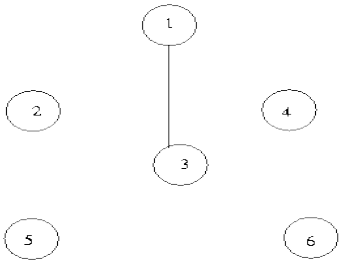
Step-2: Repeat step 1 until all the vertices are getting covered.

Example

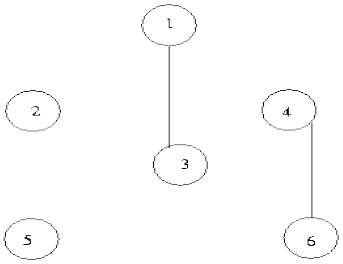




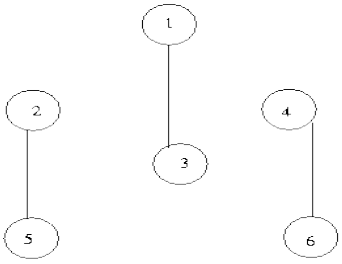
Initial Configuration



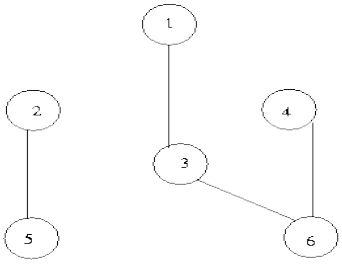
step1. choose (1,3)



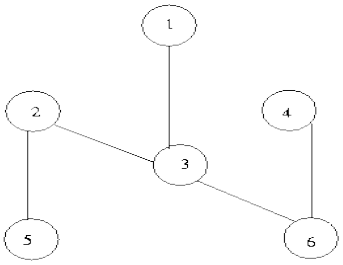
step2. choose (4,6)



step3. choose (2,5)



step4. choose (3,6)



step5. choose (2,3)

- **Difference between Prim's and Kruskal's algorithm**

Sr.No	Kruskal's algorithm	Prim's algorithm
1.	Select the shortest edge in a network	Select any vertex
2.	Select the next shortest edge which does not create a cycle	Select the shortest edge connected to that vertex
3.	Repeat step 2 until all vertices have been connected	Select the shortest edge connected to any vertex already connected
4.	Kruskal's Begins with forest and merge into tree.	Repeat step 3 until all vertices have been connected
5.	**Complexity** O(NlogN) comparison sort for edges	**Complexity** O(NlogN) search the least weight edge for every vertices.

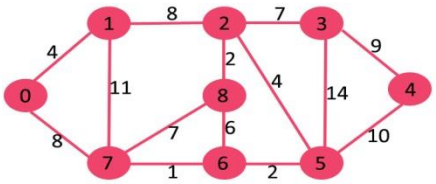
Shortest Path

The shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

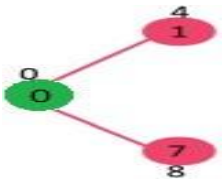
The problem of finding the shortest path between two intersections on a road map (the graph's vertices correspond to intersections and the edges correspond to road segments, each weighted by the length of its road segment) may be modeled by a special case of the shortest path problem in graphs.

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks

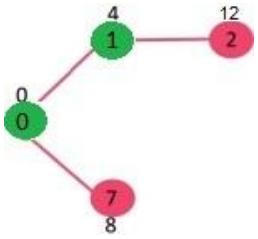
Example



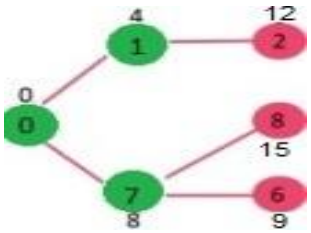
Step-1:



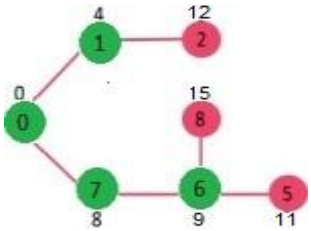
Step-2:



Step-3:



Step-4:



Step-5: