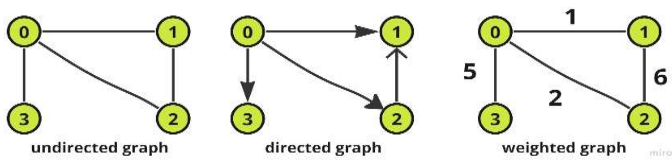
Graph

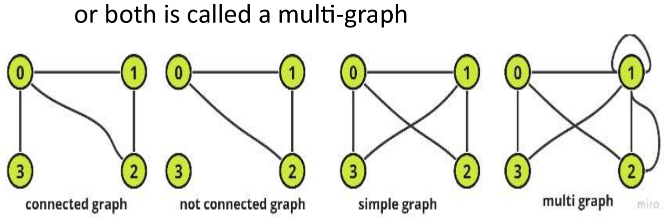
- A graph is a non-linear data structure consisting of nodes and edges .It has finite set of vertices (or nodes) and set of edges.
- Graph denoted as G(E,V) , where E=edge , V=vertices

Types of Graph

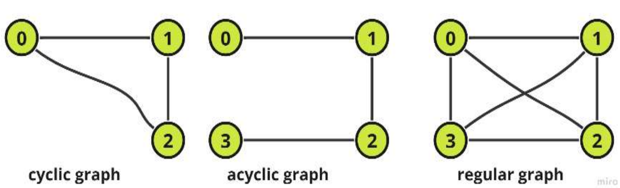
- **Undirected Graph**
  - when all the edges present between any vertices of the graph are un-directed or have not defined direction.
- **Directed Graph**
  - when all the edges present between any vertices of the graph are directed or have a defined direction.
- **Weighted Graph**
  - each edge of a graph has an associated numerical value, called a weight. Usually ,the edge weights are non- negative integers.



- **Connected Graph**
  - A connected graph is a graph in which there is a path between every pair of vertices.
- **Simple Graph**
  - A graph or directed graph which does not have any selfloop or parallel edges is called a simple graph
- **Multi-graph**
  - A graph which has either a self-loop or parallel edges or both is called a multi-graph



- **Acyclic graph**
  - If a graph (digraph) does not have any cycle then it is called as acyclic graph.
- **Cyclic graph**
  - A graph that has cycles is called a cyclic graph.
- **Regular graph**
  - all graph vertices should have the same degree.



Applications Graph

Graph theory is used to find shortest path in road or a network.

- Graphs are used to represent networks of communication.
- Graphs are used to represent data organization.
- Graphs are used for topological sorting etc.

Terminologies of graph

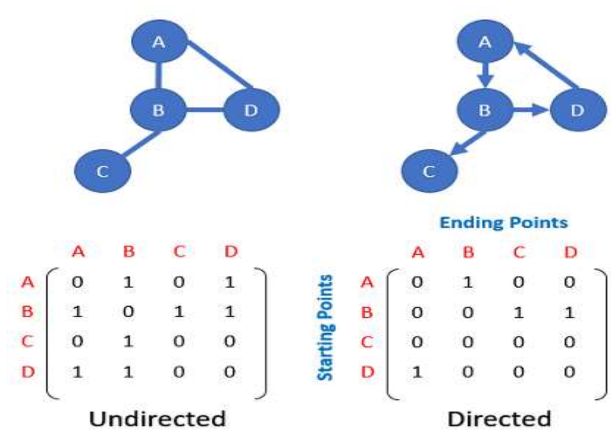
Term	Description
Vertex	individual data element is called vertex(node)
Edge	It is connecting link between two nodes
Undirected edge	It is a bidirectional edge.
Directed Edge	It is a unidirectional edge.
Weighted Edge	An edge with value (cost) on it.
Degree	total number of edges connected to a vertex .
Indegree	number of incoming edges connected to vertex.
Outdegree	total number of outgoing edges connected to vertex.
Self-loop	an edge that connects a vertex to itself.
Adjacency	Vertices are said to be adjacent if edge is connected.

Representations of graph

- Here are the two most common ways to represent a graph :
  1. Adjacency Matrix 2.Adjacency List

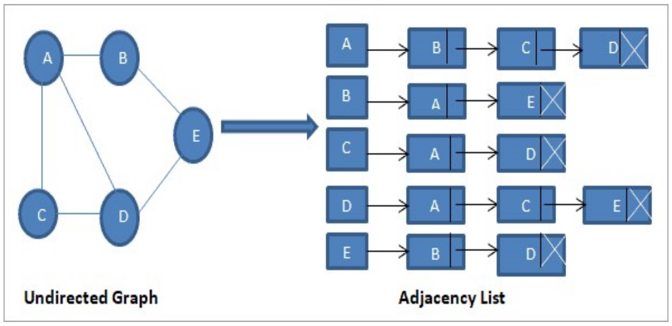
1.Adjacency Matrix

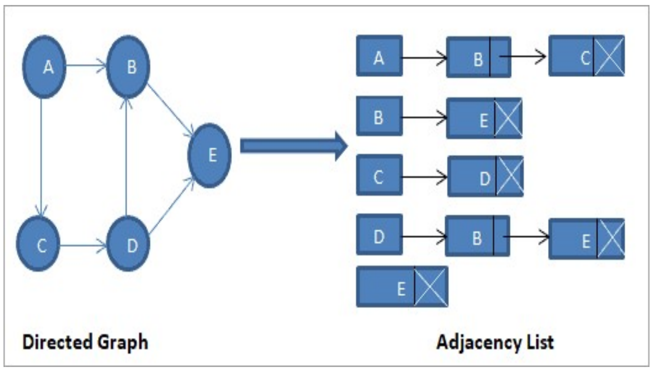
- An adjacency matrix is a way of representing a graph as a matrix of boolean (0's and 1's).
- Let's assume there are n vertices in the graph So, create a 2D matrix **adjMat[n][n]** having dimension n x n.
- If there is an edge from vertex i to j, mark **adjMat[i][j]** as 1.
- If there is no edge from vertex i to j, mark **adjMat[i][j]** as 0.



2. Adjacency List

- An array is used to store edges between two vertices
- The size of array is equal to number of vertices.
- Each index in array represents a specific vertex in the graph.
- The entry at the index i of the array contains a linked list containing the vertices that are adjacent to vertex i.





Graph traversal

- Graph is represented by its nodes and edges, so traversal of each node is the traversing in graph.
- There are two standard ways of traversing a graph
  1.Depth first search   2. Breadth first search

1.Depth-first-search [DFS]

- Dfs is the searching or traversing algorithm , in which we used the stack data structure .
- In dfs ,we will first focus on the depth then go to the breadth at that level.
- Time complexity : O(V + E) & space complexity :O(v)

Algorithm:

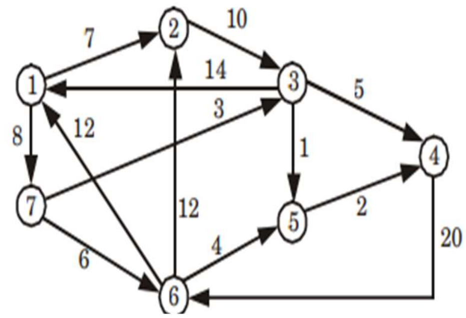
Step 1: SET STATUS = 1 (ready state) for each node in G
Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

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)

Implement DFS algorithm (Numerical)



Adjacency list of the given graph :

- $1 \rightarrow 2, 7$
- $2 \rightarrow 3$
- $3 \rightarrow 5, 4, 1$
- $4 \rightarrow 6$
- $5 \rightarrow 4$
- $6 \rightarrow 2, 5, 1$
- $7 \rightarrow 3, 6$

1. Initially set unvisited for all vertex
2. Push 1 onto stack
3. Stack =1
4. Pop 1 from stack , set vis[1]=1 and Push 2, 7 onto stack
   DFS = 1   stack = 2 7
5. Pop 7 from stack , set vis[7]=1 Push 3, 6;
   DFS = 1, 7   stack =2 3 6 .
6. Pop 6 from stack , set vis[6]=1 Push 5;
   DFS = 1, 7, 6 stack =2 3 5 .
7. Pop 5 from stack , set vis[5]=1 Push 4;
   DFS = 1, 7, 6, 5   stack = 2 3 4
8. Pop 4 from stack, set vis[4]=1
   DFS = 1, 7, 6, 5, 4 stack= 2,3
9. Pop 3 from stack , set vis[3]=1
   DFS = 1, 7, 6, 5, 4, 3 stack =2
10. Pop 2 from stack, set vis[2]=1
    DFS = 1, 7, 6, 5, 4, 3,2
11. Now, the stack is empty, so the depth first traversal of a given graph is **1, 7, 6, 5, 4, 3 2.**

2.Breadth-first-search [BFS]

- Bfs is the searching or traversing algorithm , in which we used the queue data structure .
- In bfs ,we will first focus on the breadth then go to the depth at that level.
- Time complexity : O(V + E) & space complexity :O(v)

Algorithm:

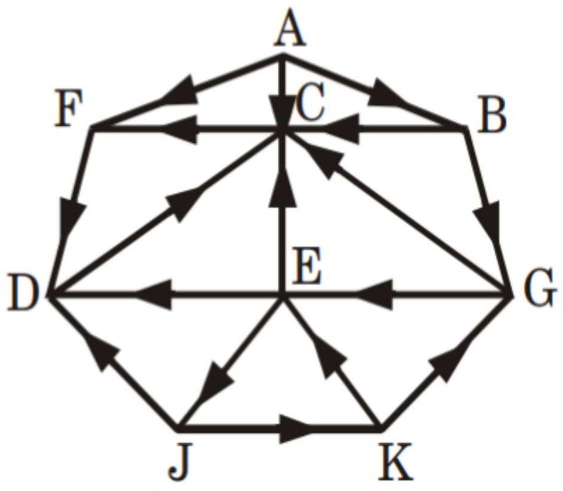
Step 1: SET STATUS = 1 (ready state) for each node in G
Step 2: Enqueue the starting node A into the queue and set its STATUS = 2 (waiting state)

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

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

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

Implement BFS algorithm (Numerical)



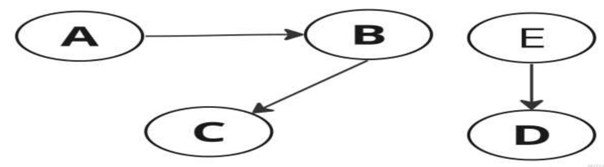
Adjacency list of the given graph :

- A → F, C, B
- B → G, C
- C → F
- D → C
- E → D, C, J
- F → D
- G → C, E
- J → D, K
- K → E, G

1. Initially set unvisited for all vertex
2. Push A into queue
   Queue =A
3. Delete A from queue, set vis[A]=1 , insert F,C,B into queue
   BFS = A queue = F,C,B
4. Delete F from queue, set vis[F]=1 and insert D into queue
   BFS = A,F queue = C,B,D
5. Delete C from queue, set vis[C]=1
   BFS = A,F ,C queue = B,D
6. Delete B from queue, set vis[B]=1 and insert G into queue
   BFS = A,F ,C,B queue = D,G
7. Delete D from queue, set vis[D]=1
   BFS = A,F ,C,B,D queue = G
8. Delete G from queue, set vis[G]=1 and insert E into queue
   BFS = A,F ,C,B,D,G queue = E
9. Delete E from queue, set vis[E]=1 and insert J into queue
   BFS = A,F ,C,B,D,G,E queue = J
10. Delete J from queue, set vis[J]=1 and insert K into queue
    BFS = A,F ,C,B,D,G,E,J queue = K
11. Delete K from queue, set vis[K]=1
    BFS = A,F ,C,B,D,G,E,J,K
12. Now, the queue is empty, so the breadth first traversal of a given graph is **A,F ,C,B,D,G,E,J,K**

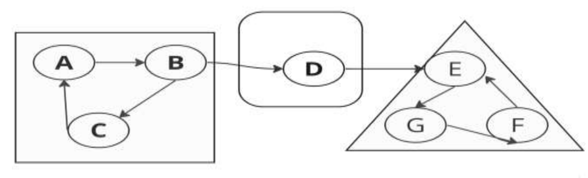
Connected component

- A connected component in a graph is a subgraph in which there is a path between every pair of vertices. In other words, all vertices in a connected component are mutually reachable.



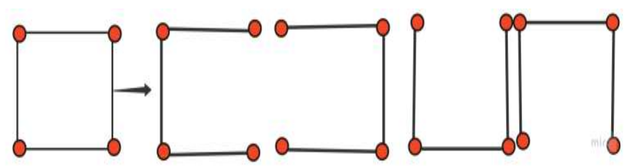
Strongly Connected component

- A directed graph is strongly connected if there is a path between all pairs of vertices.
- Kosaraju's algorithm is used to find strongly connected components in a graph.
- In this graph we have 3 strongly connected component
- 1. A B C 2. D 3. E F G



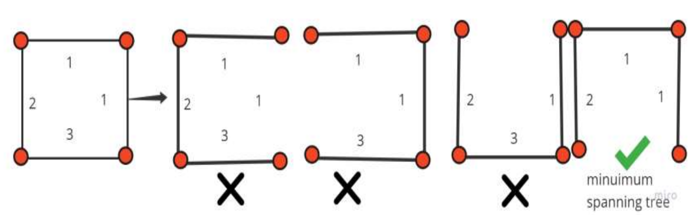
Spanning Tree

- A spanning tree of an undirected graph is a sub-graph that is a tree which contains all the vertices of graph.
- A spanning tree of a connected graph G contains all the vertices and has the edges which connect all the vertices. So, the number of edges will be 1 less than the number of nodes.
- A connected graph may have more than one spanning trees.



Minimum Spanning Tree [MST]

- In weighted graphs, a minimum spanning tree is spanning tree with the minimum possible sum of edge weights.



Prim's algorithm

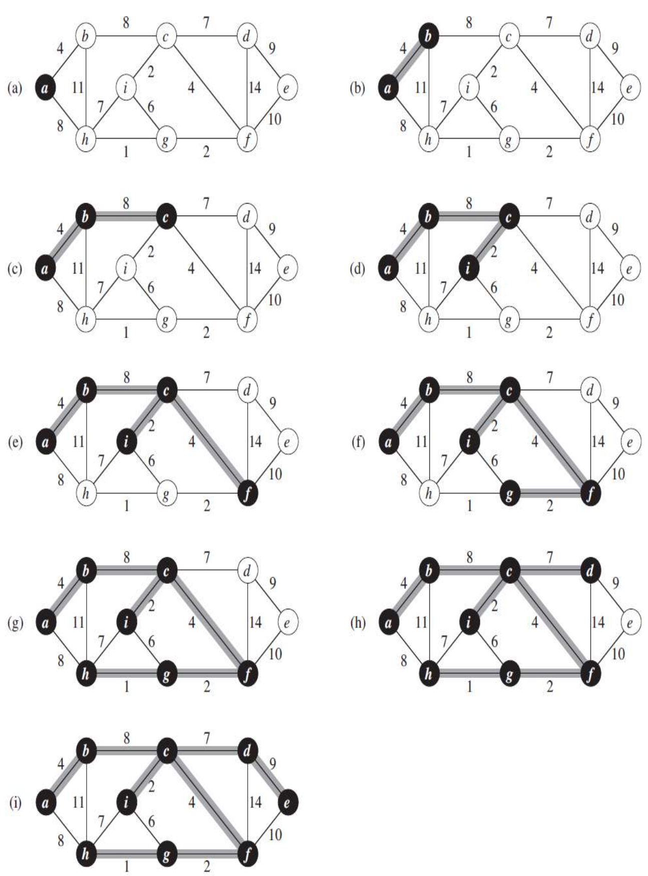
- It is a greedy algorithm that is used to find the minimum spanning tree from a graph.
- Prim's algorithm starts with the single node and explores all the adjacent nodes with all the connecting edges at every step. It is used for undirected graph .
- time complexity :O(E Log V)) & space complexity : O(V)

Algorithm:

Step1. Choose a starting vertex.
Step2. Repeat until there are fringe vertices:
a. Select the minimum-weight edge (e) connecting the tree and fringe vertex[not included].
b. Add the chosen edge and vertex to the minimum spanning tree (T).

Step3. Exit.

Implement Prim's algorithm (Numerical)



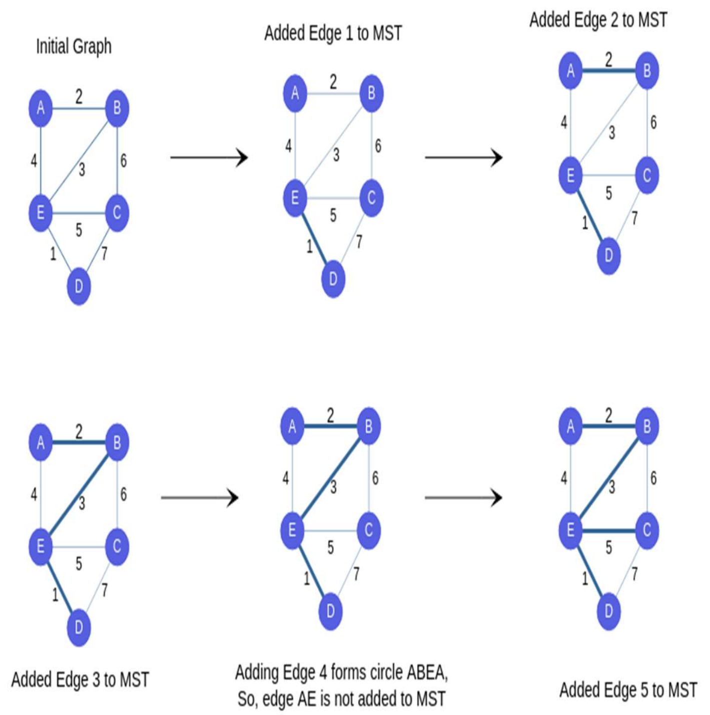
Kruskal's Algorithm

- It is a greedy algorithm that is used to find the minimum spanning tree from a graph.
- In Kruskal's algorithm, we start from edges with the lowest weight and keep adding the edges until the goal is reached.
- time complexity :O(E Log E)) & space complexity : O(V)

Algorithm :

1. Create a forest where each vertex is a separate tree.
2. Sort all the edges in non-decreasing order of their weights.
3. Pick the smallest edge. Check if it does not form a cycle then include int the spanning tree .
4. Repeat step 3 until there are (V-1) edges in the minimum spanning tree, where V is the number of vertices.

Implement kruskal's algorithm (Numerical)



Dijkstra algorithm

- Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a weighted graph,
- It is a type of Greedy Algorithm that only works on Weighted Graphs having positive weights.
- It can also be used for finding the shortest paths from a single node to a single destination
- time complexity : O(E log V) & space complexity : O(V)

Algorithm :

Step 1: First, we will mark the source node with a current distance of 0 and set the rest of the nodes to INFINITY.

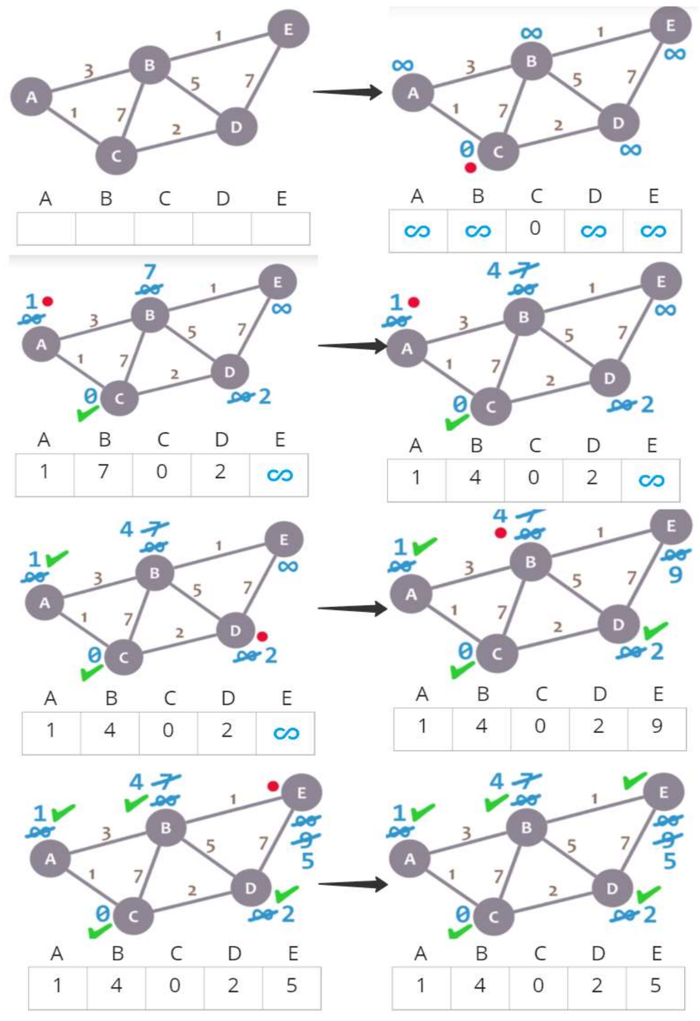
Step 2: We will then set the unvisited node with the smallest current distance as the current node "curr".

Step 3: For each neighbor N of the current node "curr":
If dist[curr]+weight[N] < dist[N] then Set dist[N] = dist[curr]+weight[N]

Step 4: We will then mark the current node "curr" as visited.

Step 5: We will repeat the process from 'Step 2' if there is any node unvisited left in the graph.

Implement dijkstra algorithm (Numerical)



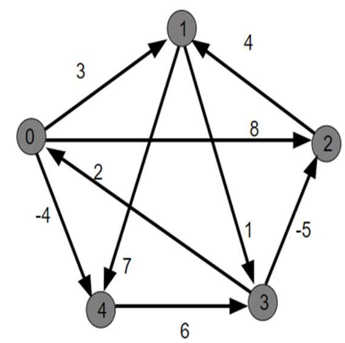
Floyd warshall algorithm

- It is a dynamic programming algorithm used to discover
- the shortest paths in a weighted graph
- In which includes both positive & negative weight cycles.
- It is also called all pair shortest path algorithm
- Time complexity : O(N^3) & space complexity: O(V^2)

Algorithm

1. Create a matrix 'dist' of size VxV (V is number of vertices) and initialize it with the weights of the edges in the graph.
   - If there is no direct edge between vertices i and j, set dist[i][j] to infinity.
   - If there is a direct edge between vertices i and j with weight w, set dist[i][j] to w.
2. For each vertex 'k' from 1 to V:
   a. For each pair of vertices 'i' and 'j':
      i. If dist[i][k] + dist[k][j] is less than dist[i][j], update dist[i][j] to dist[i][k] + dist[k][j]
3. The final 'dist' matrix contains the shortest path distances between all pairs of vertices.

Implement Floyd warshall algorithm (Numerical)



matrix A_0

	0	1	2	3	4
0	0	3	8	∞	-4
1	∞	0	∞	1	7
2	∞	4	0	∞	∞
3	2	∞	-5	0	∞
4	∞	∞	∞	6	0

matrix A_1

	0	1	2	3	4
0	0	3	8	∞	-4
1	∞	0	∞	1	7
2	∞	4	0	∞	∞
3	2	**5**	-5	0	**-2**
4	∞	∞	∞	6	0

matrix A_2

	0	1	2	3	4
0	0	3	8	**4**	-4
1	∞	0	∞	1	7
2	∞	4	0	**5**	**11**
3	2	5	-5	0	-2
4	∞	∞	∞	6	0

matrix A_3

	0	1	2	3	4
0	0	3	8	4	-4
1	∞	0	∞	1	7
2	∞	4	0	5	11
3	2	**-1**	-5	0	-2
4	∞	∞	∞	6	0

matrix A_4

	0	1	2	3	4
0	0	3	**-1**	4	-4
1	**3**	0	**-4**	1	**-1**
2	**7**	4	0	5	**3**
3	2	-1	-5	0	-2
4	**8**	**5**	**1**	6	0

Final matrix

	0	1	2	3	4
0	0	**1**	**-3**	**2**	-4
1	3	0	-4	1	-1
2	7	4	0	5	11
3	2	-1	-5	0	-2
4	8	5	1	6	0