UNIT – IV

Q. Define Graph? Explain about the types of graph?

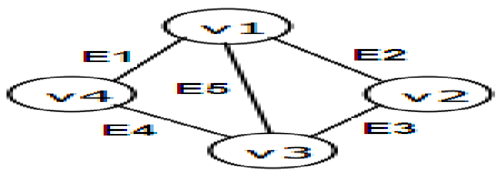
DEFINITION

"A graph G is a collection of two sets V & E. Where V is a finite non empty of vertices and E is a finite non empty set of edges".

Or

"A graph G is a defined as a set of objects called nodes and edges". G = (V, E). A graph G consists of two things:

- ✓ A set V of elements called nodes (or points or vertices).
- ✓ A set E of edges such that each edge e in E is identified with a unique pair (u, v) of nodes in V, denoted by e = (u, v)



Graph G

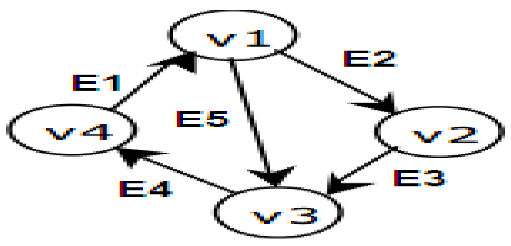
G = { { V1, V2, V3, V4}, { E1, E2, E3, E4, E5} }

TYPES OF GRAPHS

There are basically two types of graphs:

- ✓ Directed graphs
- ✓ Undirected graph

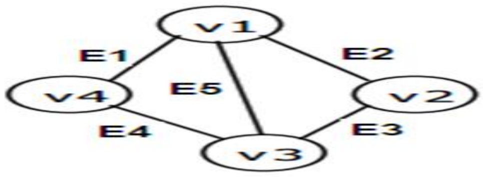
In a directed graph or digraph, the directions are shown on the edges. The edges between the vertices are ordered. In the below graph, the edge is in between the vertices V1 and V2. The vertex V1 is called head and the vertex V2 is called tail. The edge is the set of (V1, V2) and not (V2, V1).



Directed Graph

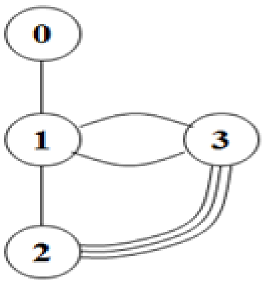
Dr. Ratna Raju Mukiri M.Tech(CSE), S.E.T., PhD.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

In undirected graph, there are no directions shown on the edges. The edges are not ordered. In the below graph, the edge E1 is the set of (V1, V2) and (V2, V1).

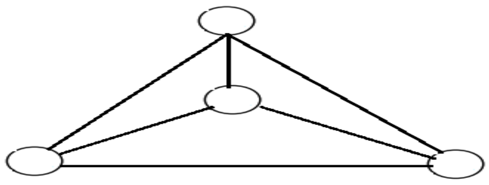


Undirected Graph

If there is more than one edge between the vertices then it is called multi – graph which is shown below.



A planar graph is a kind of graph that can be drawn in a plane and which contains no crossing edges. It is shown below.



Q. What are the properties of graphs?

PROPERTIES OF GRAPHS

Complete graph – if an undirected graph of n vertices contains n(n- 1) / 2 number of edges then it is called complete graph.

Sub graph – a sub graph G' of a graph G is a graph such that the set of vertices and set of edges of G' are power subset of the set of edges of G.

Connected graph – an undirected graph is said to be a connected graph if for every pair of distinct vertices V_i and V_j in V(G) there is a graph from V_i to V_j in G

Weighted graph – a weighted graph is a graph that contains weights along with its edges.

Path – a path is denoted using sequence of vertices and there exists an edge from one vertex to next vertex.

Cycle – a closed walk through the graph with repeated vertices so that the starting and ending vertex is same.

Indegree – the number of edges that are incident to that vertex is indegree.

Outdegree – the number of edges that are exiting out from the vertex is outdegree.

Degree – the number of edges associated with the vertex is degree of a vertex.

Self loop – self loop is the edge that connects the same vertex to itself.

Strongly connected graph - A Directed graph is called a strongly connected graph if for any two nodes I and J, there is a directed path from I to J and also from J to I.

Weakly connected graph - A Directed graph is called a weakly connected graph if for any two nodes I and J, there is a directed path from I to J or from J to I.

Source node - A node where the indegree is 0 but has a positive value for outdegree is called a source node. That is there are only outgoing arcs to the node and no incoming arcs to the node.

Sink node - A node where the outdegree is 0 and has a positive value for indegree is called the sink node. That is there is only incoming arcs to the node and no outgoing arcs the node.

Forest – it is a set of disjoint trees. If we remove the root node of a given tree then it becomes forest.

Q. Explain about different representations of graphs? (or) Explain how graphs are represented using adjacency list and adjacency matrix?

GRAPH REPRESENTION

There are different ways of representing digraphs. They are:

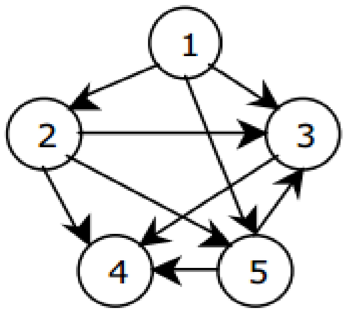
- ✓ Adjacency matrix.
- ✓ Adjacency List.

Adjacency matrix

In this representation, the adjacency matrix of a graph G is a two dimensional n x n matrix, say A = (ai,j), where

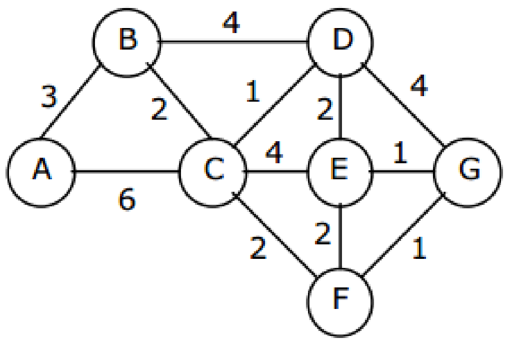
$$a_{i,j} = \begin{cases} 1 \text{ if there is an edge from } v_i \text{ to } v_j \\ 0 \text{ otherwise} \end{cases}$$

The matrix is symmetric in case of undirected graph, while it may be asymmetric if the graph is directed. This matrix is also called as Boolean matrix or bit matrix. The below figure shows the adjacency matrix representation of the graph G. The adjacency matrix is also useful to store multigraph as well as weighted graph.



	1	2	3	4	5
1	0	1	1	0	1
2	0	0	1	1	1
3	0	0	0	1	0
4	0	0	0	0	0
5	0	0	1	1	0

The adjacency matrix for a weighted graph is called as cost adjacency matrix. The below figure shows the cost adjacency matrix representation of the graph G.

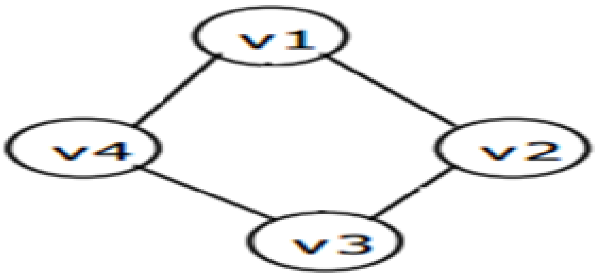


	A	B	C	D	E	F	G
A	0	3	6	∝	∝	∝	∝
B	3	0	2	4	∝	∝	∝
C	6	2	0	1	4	2	∝
D	∝	4	1	0	2	∝	4
E	∝	∝	4	2	0	2	1
F	∝	∝	2	∝	2	0	1
G	∝	∝	∝	4	1	1	0

Adjacency List

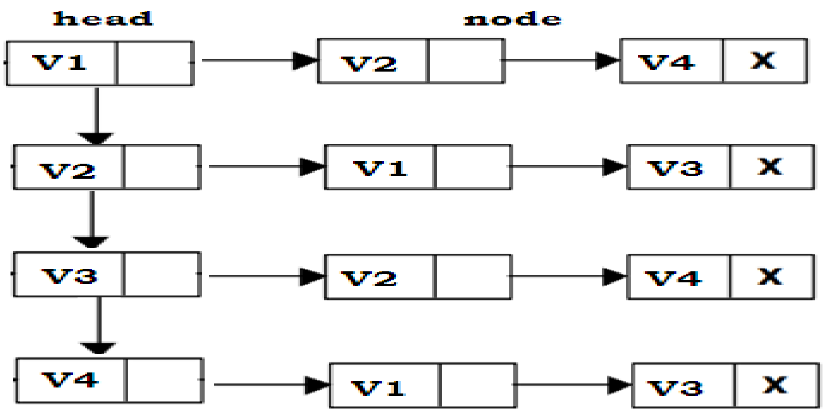
As we have problem with array, to overcome it we have selected the flexible data structure called linked lists. The type in which a graph is created with the linked list is called adjacency list.

We know that the graph is set of vertices and edges we will maintain two structures for vertices and edges respectively. The graph contains four vertices V1, V2, V3, V4 so to maintain them we use linked list of head nodes and adjacent nodes.



Graph G

```
struct head
{
        char data;
        struct head *down;
        struct head *next;
};
struct node
{
        char data;
        struct node *next1;
}
```



The down pointer helps us to go to each node in the graph whereas next pointer is for going to adjacent node of each of the head node.

Q. Explain about graph operations? (or) Explain about graph taversals?

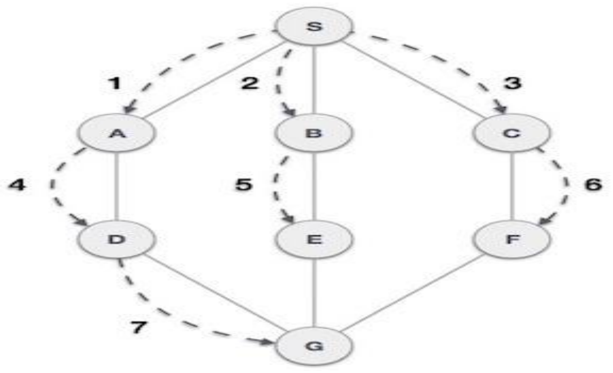
ELEMENTARY GRAPH OPERATIONS

There are two elementary graph operations called graph traversal techniques. They are as follows.

- ✓ Breadth First Search
- ✓ Depth First Search

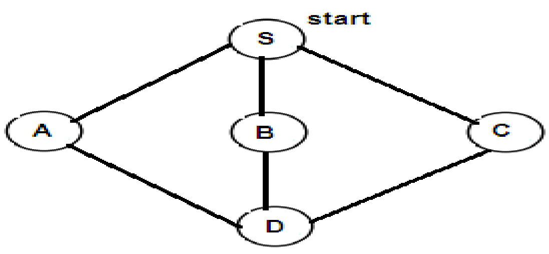
BREADTH FIRST SEARCH

To traverse a graph by BFS, first a vertex V_1 in the graph will be visited then all the adjacent vertices of it will be traversed. Suppose if V_1 has the adjacent vertices (V_1, V_2, …..V_n) then they will be printed first and the process continues for all vertices. The data structures used for keeping all vertices and their adjacent vertices is Queue. We also use array for visited vertices. The nodes that are visited are marked as 1. In BFS, traversing is done in breadth wise fashion.

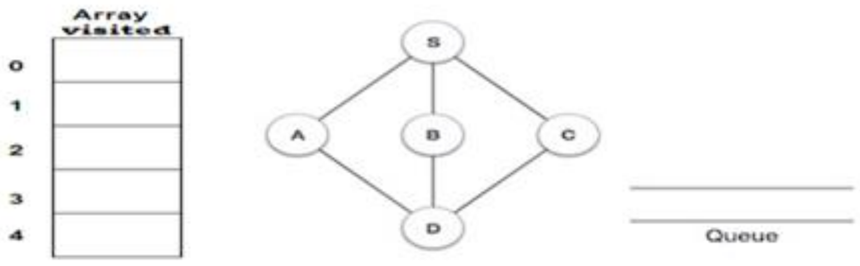


Algorithm

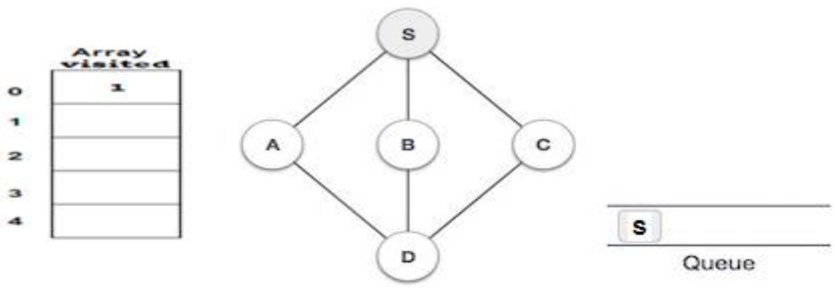
1. Create a graph, depending on the type of graph either it may be directed or undirected set the value of flag to 1 or 0.
2. Read the vertex from which you want to traverse the graph say V_i
3. Initialize the visited array to 1 at the index of V_i
4. Insert the visited vertex V_i in the queue
5. Visit the vertex which is at the front of the queue, delete it from the queue and place the adjacent vertices in the queue.
6. Repeat the step5 till queue is not empty.



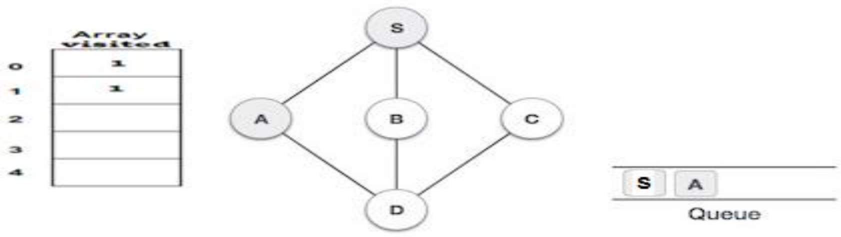
Step 1: Initialize the queue.



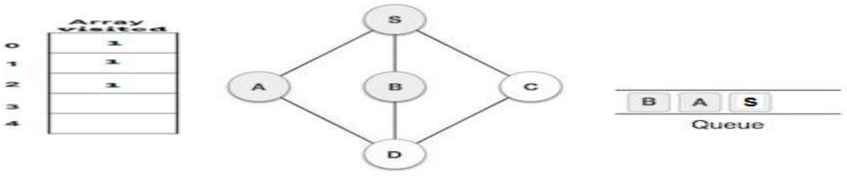
Step 2: Start with the vertex from visiting **S** (starting node), and mark it as visited.



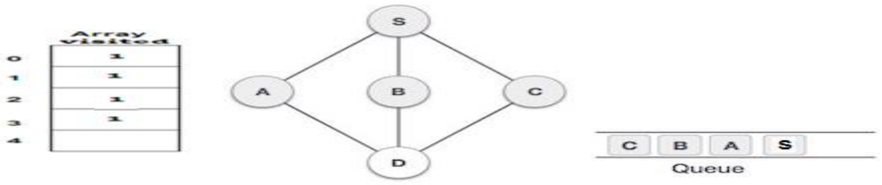
Step 3: We then see an unvisited adjacent node from **S**. In this example, we have three nodes but alphabetically we choose **A**, mark it as visited and enqueue it.



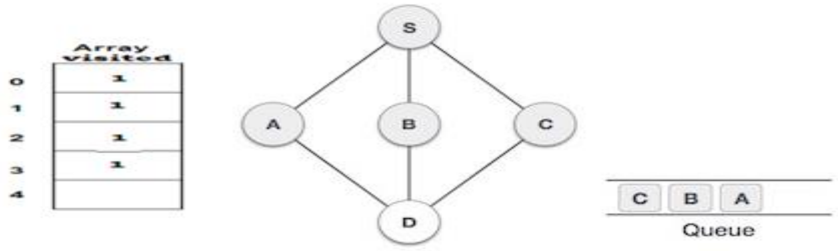
Step 4: Next, the unvisited adjacent node from **S** is **B**. We mark it as visited and enqueue it.



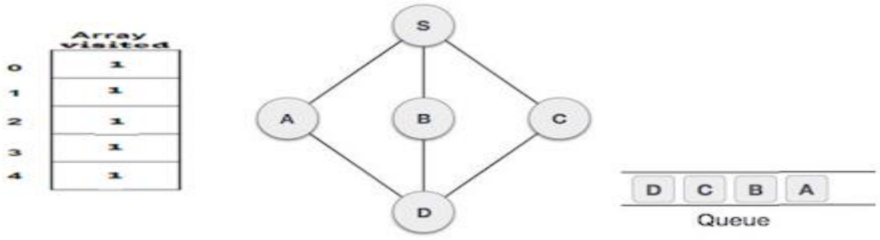
Step 5: Next, the unvisited adjacent node from **S** is **C**. We mark it as visited and enqueue it.



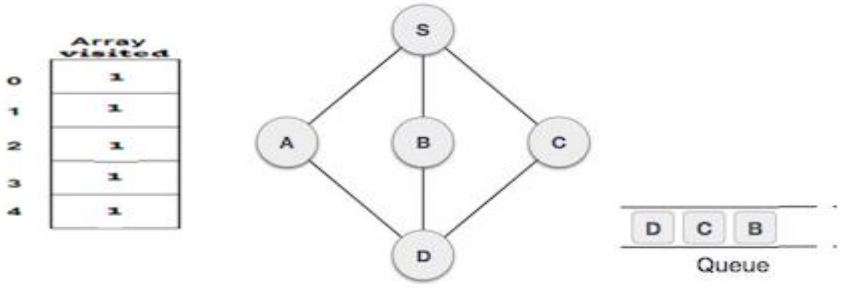
Step 6: Now, **S** is left with no unvisited adjacent nodes. So, we dequeue **S** and find **A**.



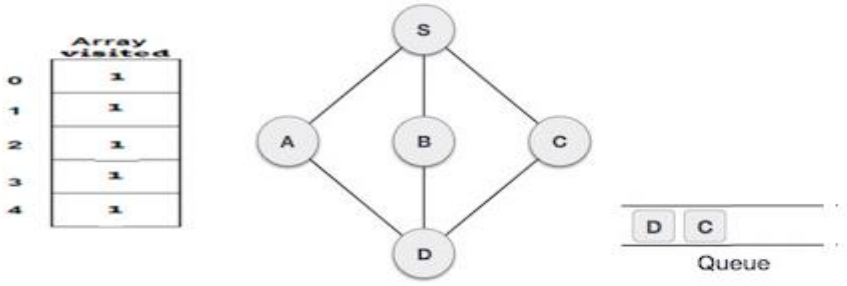
Step 7: Next, the unvisited adjacent node from **A** is **D**. We mark it as visited and enqueue it.



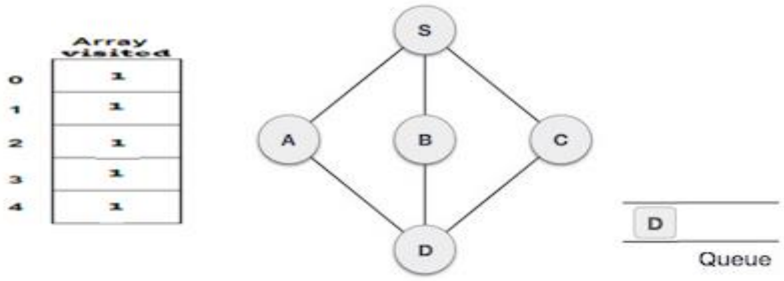
Step 7: Now, **A** is left with no unvisited adjacent nodes. So, we dequeue **A**.



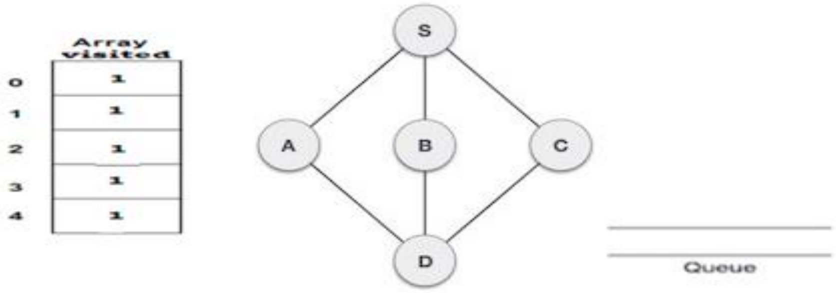
Step 8: Now, **B** is dequeued.



Step 9: Now, **C** is dequeued.

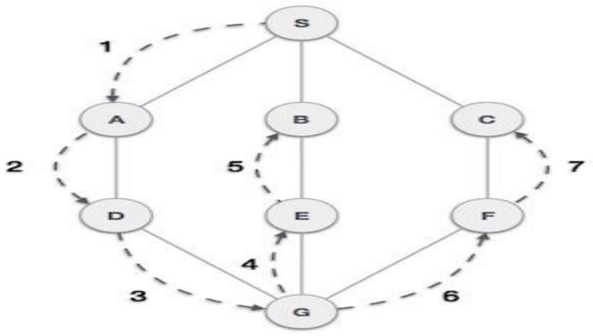


Step 10: Now, **D** is dequeued.



DEPTH FIRST SEARCH (DFS)

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.



Algorithm

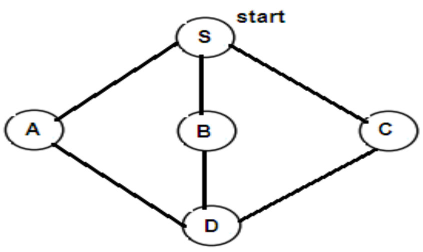
1. Create a graph, depending on the type of graph either it may be directed or undirected.

2. Read the vertex from which you want to traverse the graph say V_i

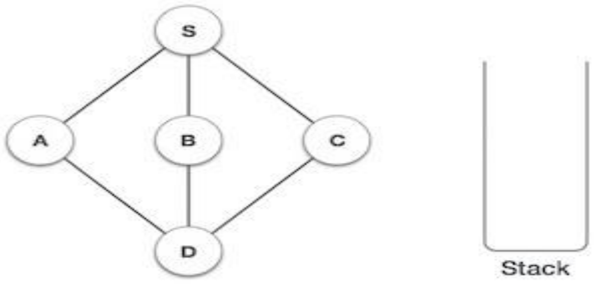
3. Insert the visited vertex V_i and push all the adjacent vertices into the Stack

4. If no adjacent vertices found ten pop the top vertex from the stack.

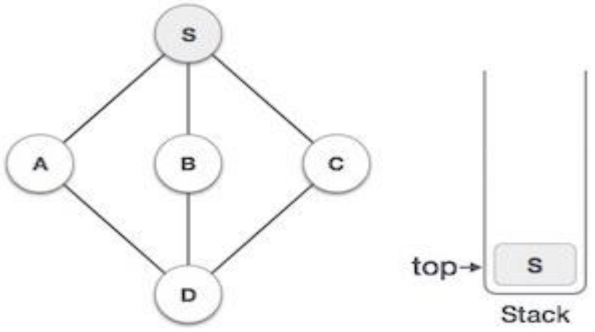
5. Repeat the step4 till stack is empty.



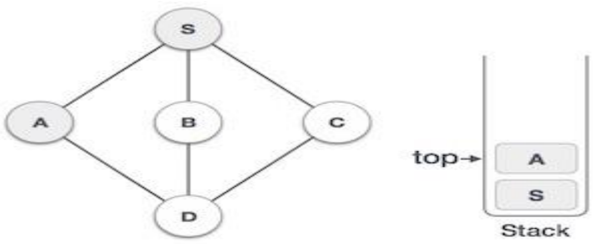
Step 1: Initialize the stack.



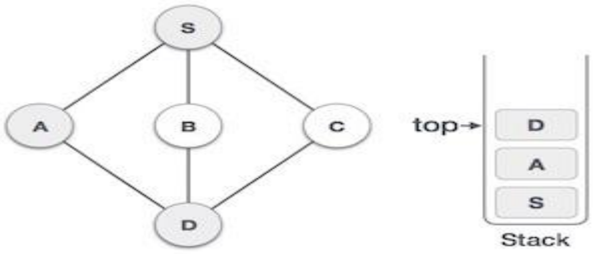
Step 2: Mark **S** as visited and put it onto the stack. See any unvisited adjacent node from **S**. We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order.



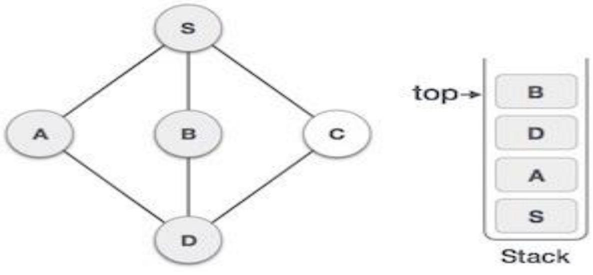
Step 3: Mark **A** as visited and put it onto the stack. See any unvisited adjacent node from A. Both **S** and **D** are adjacent to **A** but we are concerned for unvisited nodes only.



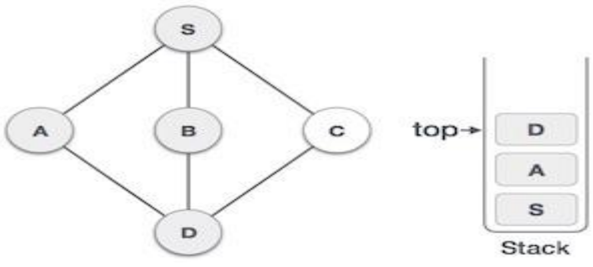
Step 4: Visit **D** and mark it as visited and put onto the stack. Here, we have **B** and **C** nodes, which are adjacent to **D** and both are unvisited. However, we shall again choose in an alphabetical order.



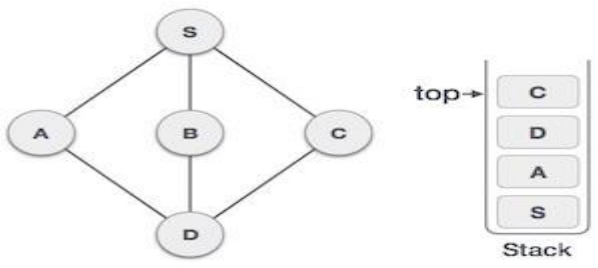
Step 5: We choose **B**, mark it as visited and put onto the stack. Here **B** does not have any unvisited adjacent node. So, we pop **B** from the stack.



Step 6: We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find **D** to be on the top of the stack.



Step 7: Only unvisited adjacent node is from **D** is **C** now. So we visit **C**, mark it as visited and put it onto the stack.

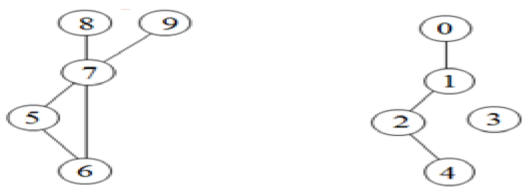


Step 8: So now we pop one by one node from stack until is empty.

Q. Define connected components, spanning trees, biconnected components and minimum cost spanning tree.

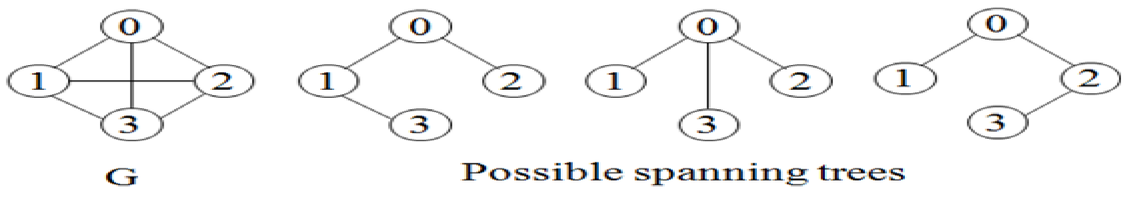
CONNECTED COMPONENTS

The maximal connected subgraph of a graph is called connected component of a graph.



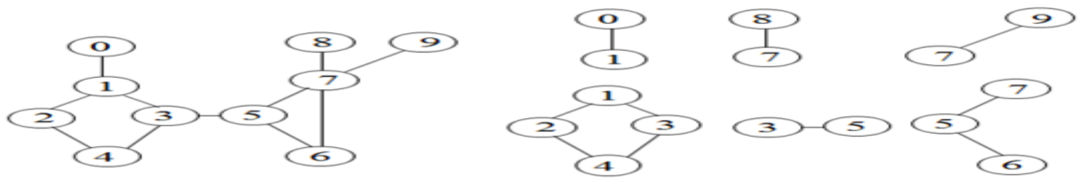
SPANNING TREES

A spanning tree of a graph G is a tree which has all vertices being covered with minimum possible number of edges and does not form a cycle.



BICONNECTED COMPONENTS

A biconnected graph is a connected graph that has no articulation points. A maximal connected subgraph H (no subgraph that is both biconnected and properly contains H).



MINIMUM COST SPANNING TREES

In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same gragh. Three different algorithms can be used

- ✓ Kruskal
- ✓ Prim

Q. Explain about Kruskal's algorithm to find the minimum cost spanning tree with an example?

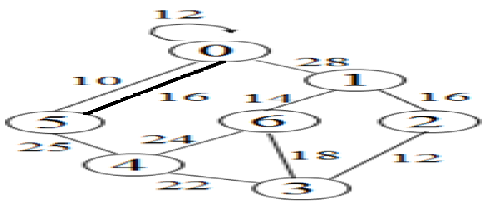
KRUSKAL'S ALGORITHM

This algorithm is used to find the minimum cost spanning tree using greedy approach. This algorithm treats the graph as a forest and every node in it has an individual tree. A tree connects to another only if and only iff it has least cost among the available options and do not violate the minimum spanning tree properties.

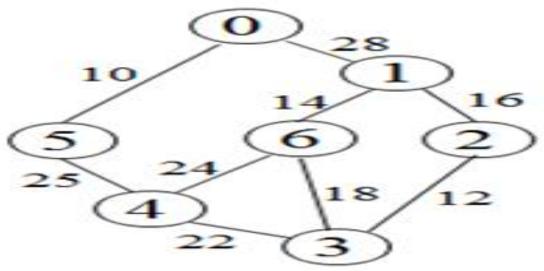
Algorithm

```
1. for each vertex V in graph G do
2. create a set with the element V
3. initialize priority queue Q containing all the edges in descending order of
their weights
4. define forest = NULL
5. while T has edges <= N-1 do
6. select an edge with minimum weight
7. if T(v) T(u)
        add (u,v) to forest
        union T(v) and T(u)
8. return T.
```

To explain kruskal's algorithm let us consider the following example:



Step 1: Remove all loops and remove all parallel edges



Step 2: Arrange all edges in their increasing order of their weight.

$0 \overset{10}{\text{——}} 5$

$2 \overset{12}{\text{——}} 3$

$1 \overset{14}{\text{——}} 6$

$1 \overset{16}{\text{——}} 2$

$3 \overset{18}{\text{——}} 6$

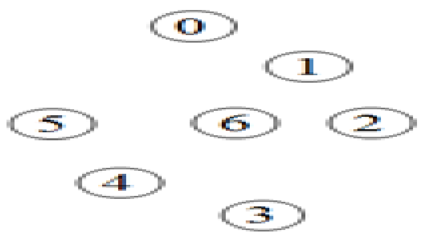
$3 \overset{22}{\text{——}} 4$

$4 \overset{24}{\text{——}} 6$

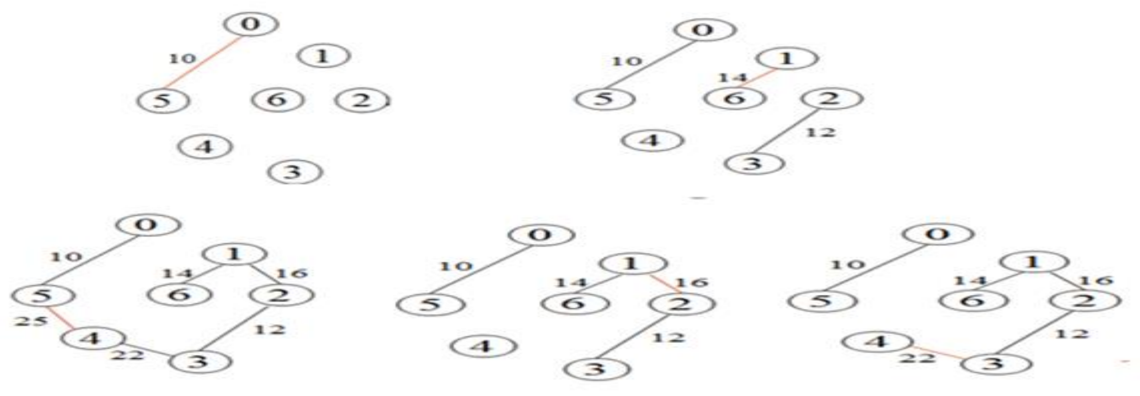
$4 \overset{25}{\text{——}} 5$

$0 \overset{28}{\text{——}} 1$

Step 3: Initially the tree is



Step 4: Add the edges which has the least cost / weight



Q. Explain about prim's algorithm to find the minimum cost spanning tree with an example?

PRIM'S ALGORITHM

Prim's algorithm is used to find minimum cost spanning tree. It (as Kruskal's algorithm) uses the greedy approach. Prim's algorithm shares a similarity with the **shortest path first** algorithms. Prim's algorithm, in contrast with Kruskal's algorithm, treats the nodes as a single tree and keeps on adding new nodes to the spanning tree from the given graph.

Algorithm

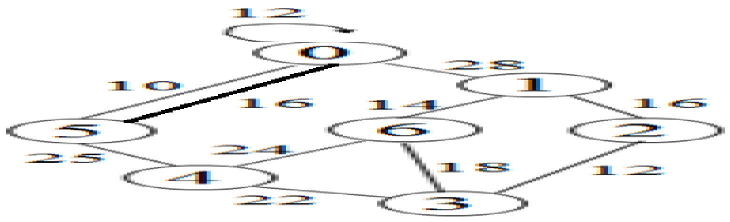
1. let G be a graph that is connected, weighted and undirected graph
2. create two sets V and V' such that

   V' is empty

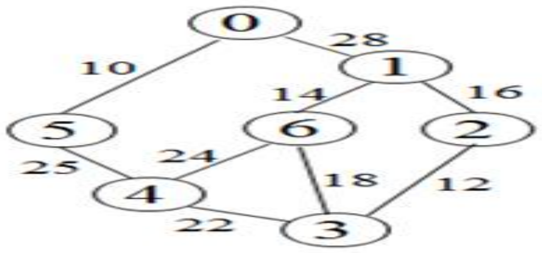
   V contains all the vertices in the graph

   select minimum weighted edge (i, j) from G and insert i & j into the set V'
3. repeat step 4 until V is not equal to V'
4. find all neighbors of all vertices which are in set V' such that one end – point of neighbor edge is in V and another not in V'.
5. sum up all selected edges weights and exit.

To explain prim's algorithm let us consider the following example:

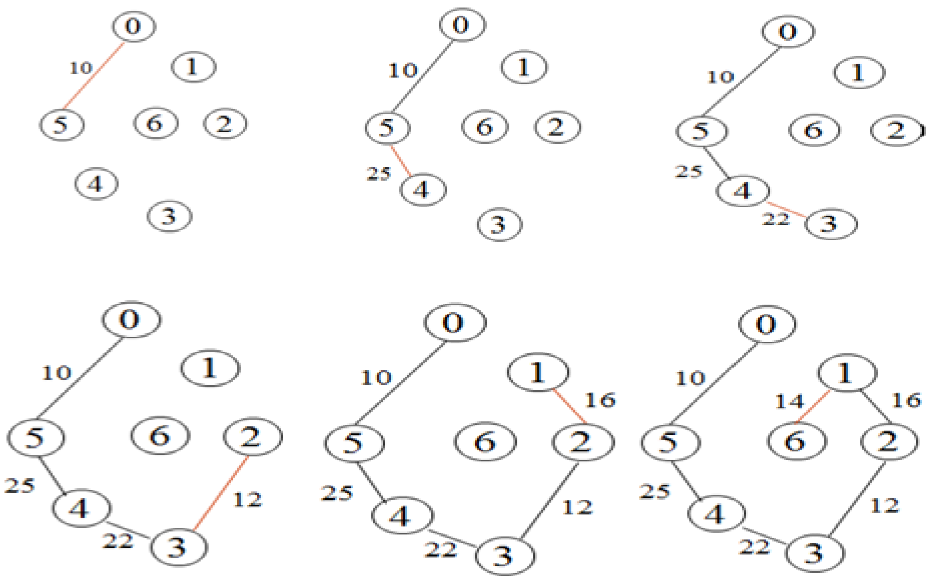


Step 1: Remove all loops and remove all parallel edges



Dr. Ratna Raju Mukiri M.Tech(CSE), S.E.T., PhD.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Step 2: apply prim’s algorithm now we get the spanning tree with minimum cost as follows



Q. Define shortest path and transitive closure?

SHORTEST PATH AND TRANSITIVE CLOSURE

The minimum cost path between the vertices in a graph is called the shortest path.

Transitive closure is a Boolean matrix in which the existence of directed paths of lengths between vertices is mentioned. It is generated by BFS and DFS. While computing transitive closure we have start with some vertex and find all edges which are reachable to every other vertex.

$$\begin{tikzcd} 0 \arrow[r] & 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & 4 \arrow[ll, bend right] \end{tikzcd}$$

(a) Digraph G

$$\begin{array}{c|ccccc} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 1 & 0 \\ 3 & 0 & 0 & 0 & 0 & 1 \\ 4 & 0 & 0 & 1 & 0 & 0 \end{array}$$

(b) Adjacency matrix A for G

$$\begin{array}{c|ccccc} 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 2 & 0 & 0 & 1 & 1 & 1 \\ 3 & 0 & 0 & 1 & 1 & 1 \\ 4 & 0 & 0 & 1 & 1 & 1 \end{array}$$

cycle

(c) transitive closure matrix A^+

There is a path of length > 0

$$\begin{array}{c|ccccc} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 2 & 0 & 0 & 1 & 1 & 1 \\ 3 & 0 & 0 & 1 & 1 & 1 \\ 4 & 0 & 0 & 1 & 1 & 1 \end{array}$$

reflexive

(d) reflexive transitive closure matrix A^*

There is a path of length ≥ 0

Q. Explain about single source all destinations non negatve edge cost (or) Explain about Dijkstra's Algorithm to find the shortest path from source to destination vertices.

SINGLE SOURCE / ALL DESTINATIONS NON NEGATIVE EDGE COST

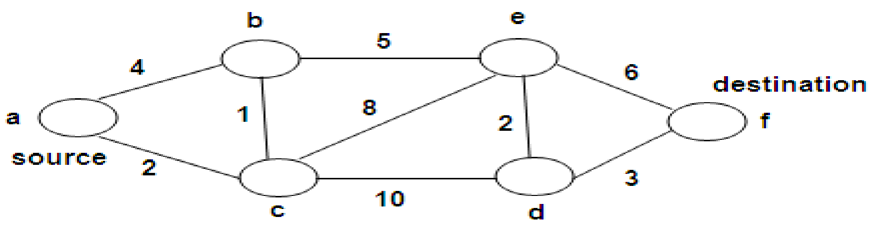
DIJKSTRA'S ALGORITHM

This algorithm is used to find the shortest path between the vertices in the graph. This algorithm finds the cost of shortest path from source vertex to destination vertex.

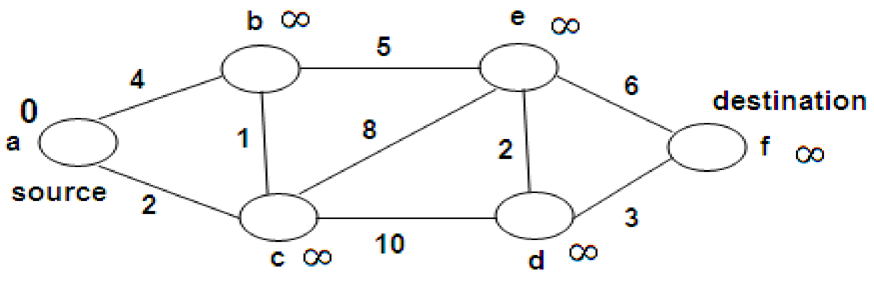
Algorithm

1. Assign every node a tentative distance.
2. set initial node as current and mark all nodes as unvisited.
3. for current node, consider all unvisited nodes and calculate tentative distance. Compare current distance with calculated distance and assign the smaller value.
4. when all the neighbors are considered of the current node mark it visited.
5. if the destination is marked visited then stop.
6. end

To explain about shortest path by using dijkstra's algorithm let us consider the following example

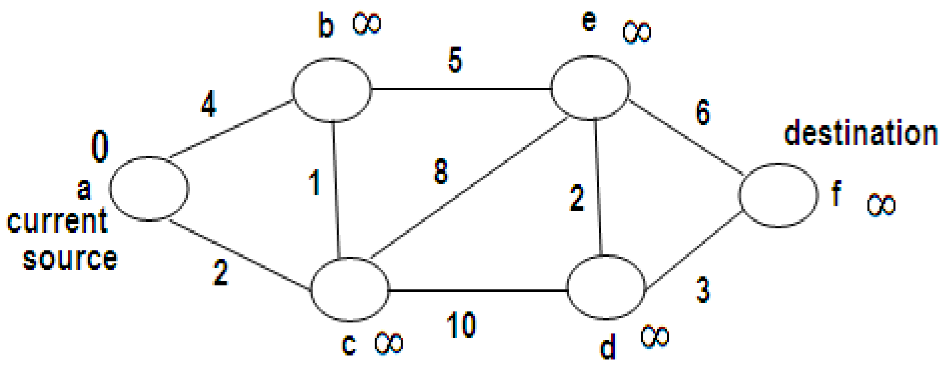


Step 1: Assign every node a tentative distance.

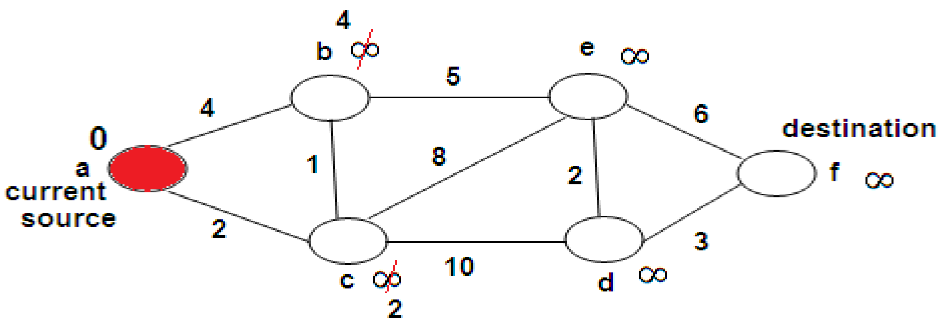


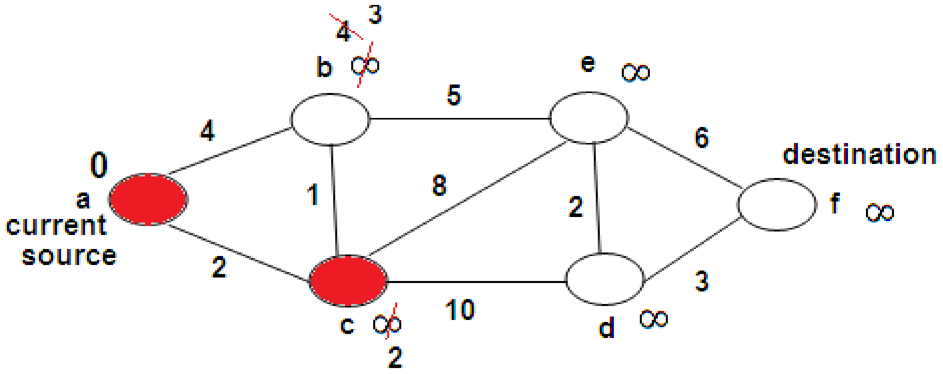
Dr. Ratna Raju Mukiri M.Tech(CSE), S.E.T., PhD.,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

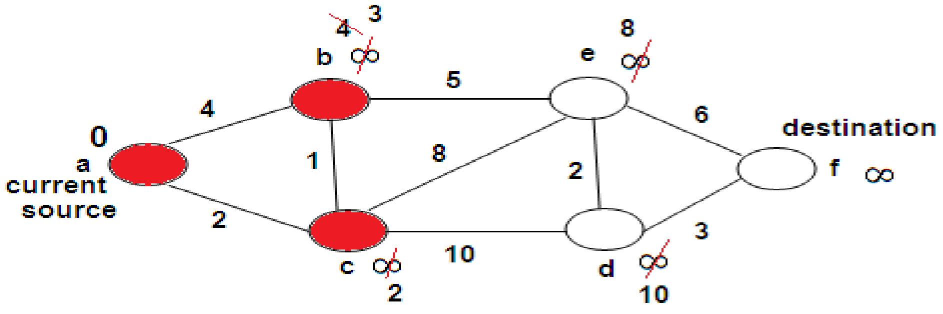
Step2: Set initial node as current and mark all nodes as unvisited.

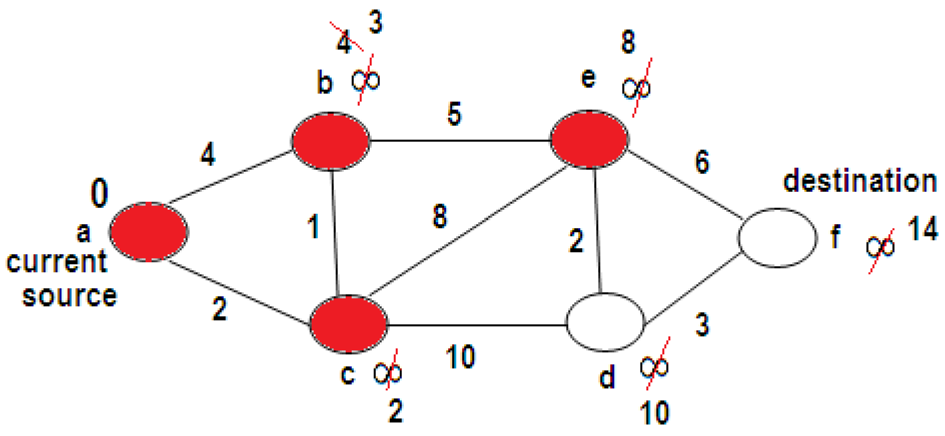


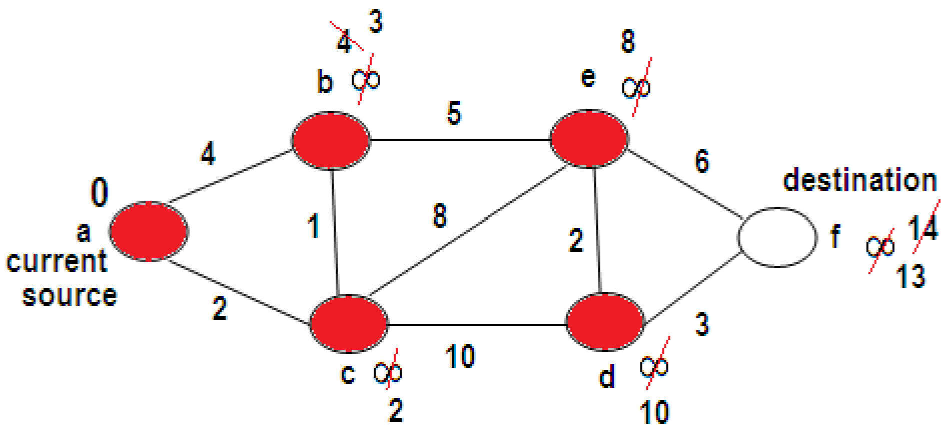
Step 3: For current node, consider all unvisited nodes and calculate tentative distance. Compare current distance with calculated distance and assign the smaller value.

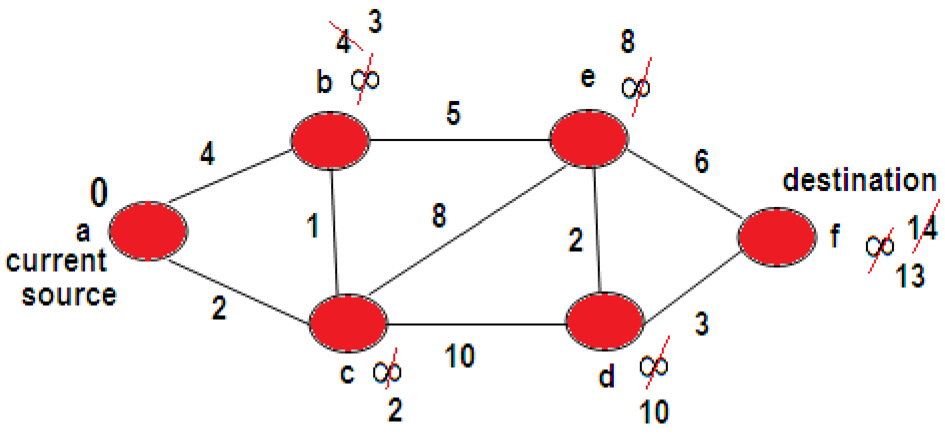












Step 4: if the destination is marked visited then stop.

Therefore the total cost to reach from source to destination using dijkstra's algorithm is 13. i.e. 2 + 1 + 5 + 2 + 3. The path followed is a – c – b – e – d – f.

Q. Explain about single source / all destinations: general weights, all – pairs shortest path, transitive closure (or) Explain about Floyd –Warshall's algorithm with an example?

SINGLE SOURCE / ALL DESTINATIONS: GENERAL WEIGHTS, ALL – PAIRS SHORTEST PATH, TRANSITIVE CLOSURE

FLOYD – WARSHALL'S ALGORITHM

This algorithm is used to find the shortest path between all pair of vertices where negative edges are allowed.

d_{ij}^{k} = weight of the shortest path from veytex i to j for which all interediate nodes are in {1, 2, ...k}

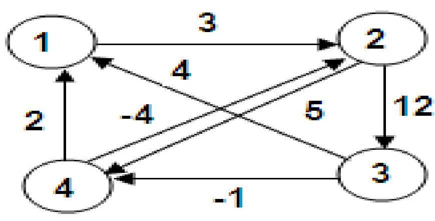
$$d_{ij}^{k} = \begin{cases} w_{ij} & \text{if } k=0 \\ \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}) & \text{if } k>=1 \end{cases}$$

$$d_{ij}^{k} = \begin{cases} 0 & \text{if i equal to j} \\ w_{ij} & \text{if i not equal to j} \\ \infty & \text{other wise} \end{cases}$$

Algorithm

```
Floyd_Warshall(w)
1.  n = rows(w)
2. $D^0$ = w
3.  for k=1 to n do
4. for i=1 to n do
5. for j=1 to n do
$d_{ij}^{k} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$
6. return $D^n$
```

To explain about Floyd_Warshall algorithm for shortest path let us consider the following example.



According to the algorithm, the distance from each vertex is noted in the following diagram

$$d_{ij}^{k} = \begin{cases} 0 & \text{if i equal to j} \\ w_{ij} & \text{if i not equal to j} \\ \infty & \text{other wise} \end{cases}$$

As per the formula if i = j then it is 0. So 1-1, 2-2, 3-3 and 4-4 values in the two dimensional are 0. W_{ij} if i is not equal to j. i.e the distance from i to j. for example 1-2 the distance is 3. As there is no path from 2-1 it is infinity(∞).

D^0

	1	2	3	4
1	0	3	∞	∞
2	∞	0	12	5
3	4	∞	0	-1
4	2	-4	∞	0

For k = 1, 2, 3, 4
i = 1, 2, 3, 4 and
j = 1, 2, 3, 4

We find D^1as for k=1, i=1, 2, 3, 4 j=1, 2, 3, 4. As per the formula if i = j then it is 0. So 1-1, 2-2, 3-3 and 4-4 values in the two dimensional are 0. W_{ij} if i is not equal to j. i.e the distance from i to j.

when i=1, j=2 since the formula applied is

$$d_{ij}^{k} = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$$

$$d_{ij}^{k^1} = \min(d_{ij}^{0}, d_{ik}^{0} + d_{kj}^{0})$$

$= \min(d_{12}, d_{11} + d_{12})$

= min(3, 0+3)

=3

When i=1, j=3

$$d^{k^1}_{ij} = \min(d^0_{ij} \; , d^0_{ik} + d^0_{kj})$$

$=\min(d_{13}, d_{11} + d_{13})$

$=\min(\infty, 0+\infty)$

$=\infty$

When i=1, j=4

$$d^{k^1}_{ij} = \min(d^0_{ij} \; , d^0_{ik} + d^0_{kj})$$

$=\min(d_{14}, d_{11}+d_{14})$

$=\min(\infty, 0+\infty)$

$=\infty$

When i=2, j=1

$$d^{k^1}_{ij} = \min(d^0_{ij} \; , d^0_{ik} + d^0_{kj})$$

$=\min(d_{21}, d_{21}+d_{11})$

$=\min(\infty, \infty + 0)$

$=\infty$

When i=2, j=3

$$d^{k^1}_{ij} = \min(d^0_{ij} \; , d^0_{ik} + d^0_{kj})$$

$=\min(d_{23}, d_{21}+d_{13})$

$=\min(12, \infty + \infty)$

$=12$

When i=2, j=4

$$d^{k^1}_{ij} = \min(d^0_{ij} \; , d^0_{ik} + d^0_{kj})$$

$=\min(d_{24}, d_{21} + d_{14})$

$=\min(5, \infty +\infty)$

$=5$

When i=3, j=1

$$d^{k^1}_{ij} = \min(d^0_{ij}, d^0_{ik} + d^0_{kj})$$

$=\min(d_{31}, d_{31} + d_{11})$

$=\min(4, 4 + 0)$

$=4$

When i=3, j=2

$$d^{k^1}_{ij} = \min(d^0_{ij}, d^0_{ik} + d^0_{kj})$$

$=\min(d_{32}, d_{31} + d_{12})$

$=\min(\infty, 4 + 3)$

$=7$

When i=3, j=4

$$d^{k^1}_{ij} = \min(d^0_{ij}, d^0_{ik} + d^0_{kj})$$

$=\min(d_{34}, d_{31} + d_{14})$

$=\min(-1, 4 + \infty)$

$=-1$

When i=4, j=1

$$d^{k^1}_{ij} = \min(d^0_{ij}, d^0_{ik} + d^0_{kj})$$

$=\min(d_{41}, d_{41} + d_{11})$

$=\min(2, 2 + 0)$

$=2$

When i=4, j=2

$$d^{k^1}_{ij} = \min(d^0_{ij}, d^0_{ik} + d^0_{kj})$$

$=\min(d_{42}, d_{41} + d_{12})$

$=\min(-4, 2 + 3)$

$=-4$

When i=4, j=3

$$d^{k^1}_{ij} = \min(d^0_{ij}, d^0_{ik} + d^0_{kj})$$

$=\min(d_{43}, d_{41} + d_{13})$

$=\min(\infty, 2 + \infty)$

$=\infty$

k = 1

D^1

	1	2	3	4
1	0	3	∞	∞
2	∞	0	12	5
3	4	7	0	-1
4	2	-4	∞	0

Similarly by doing so using the formula we can get D2, D3 and D4 when k=2, k=3 and k=4

k = 2

D^2

	1	2	3	4
1	0	3	15	8
2	∞	0	12	5
3	4	7	0	-1
4	2	-4	8	0

k = 3

D^3

	1	2	3	4
1	0	3	15	8
2	16	0	12	5
3	4	7	0	-1
4	2	-4	8	0

k = 4

D^4

	1	2	3	4
1	0	3	15	8
2	7	0	12	5
3	1	-5	0	-1
4	2	-4	8	0