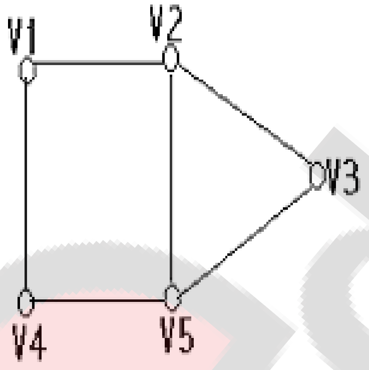
Graph

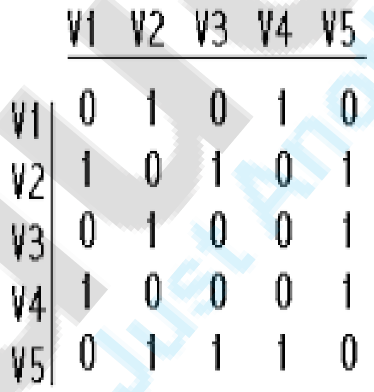
Graph Representation:

- **Adjacency matrix**
- **Adjacency List**

<u>**Adjacency matrix:**</u>

- n by n matrix, where n is number of vertices
  - A(m,n) = 1 iff (m,n) Є E ,
  - 0 otherwise
- If V_i is adjacent to V_j, then place 1 at the ith row and jth column and 1 at jth row and ith column; place 0 otherwise.
- The adjacency matrix is symmetric for a simple graph.

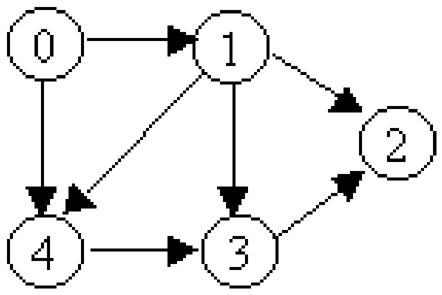




$$\begin{array}{c|ccccc} & V1 & V2 & V3 & V4 & V5 \\ \hline V1 & 0 & 1 & 0 & 1 & 0 \\ V2 & 1 & 0 & 1 & 0 & 1 \\ V3 & 0 & 1 & 0 & 0 & 1 \\ V4 & 1 & 0 & 0 & 0 & 1 \\ V5 & 0 & 1 & 1 & 1 & 0 \end{array}$$

- **Digraph -** for each directed edge (V_i, V_j), we place 1 at ith row and jth column

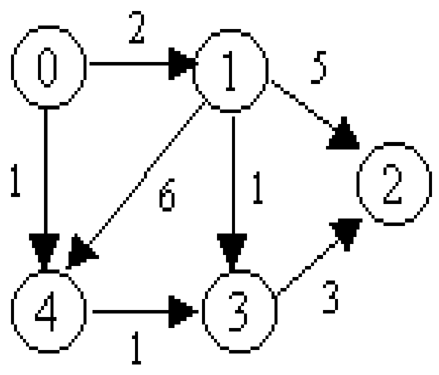
Adjacency matrix, in general, is not symmetric for a digraph



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

- **Weighted graph:**

A(m,n) = w (weight of edge), or positive infinity (or 0) otherwise

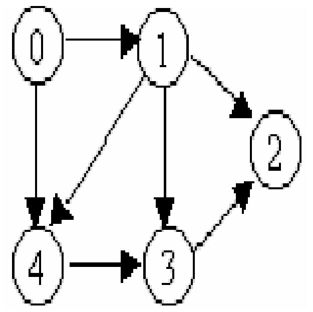
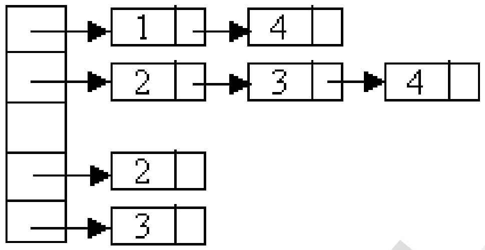


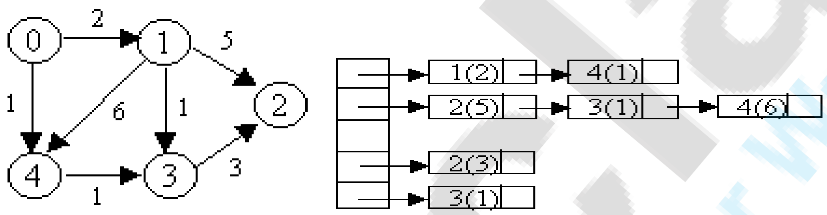
$$A = \begin{bmatrix} \infty & 2 & \infty & \infty & 1 \\ \infty & \infty & 5 & 1 & 6 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 3 & \infty & \infty \\ \infty & \infty & \infty & 1 & \infty \end{bmatrix}$$

- **Advantages**
  - Simple
- **Disadvantage**
  - Only one edge can be stored between any two vertices.
  - Additional information about graph can not be stored.
  - Insertion/ Deletion of node requires changing dimension of matrix.
  - Wastage of space. Even if no edge is present in graph (null graph) of n nodes, a n x n matrix will be allocated which will be containing all 0's (null matrix).

<u>Adjacency list</u>*:*

- For a graph G = (V,E) a list is formed for each element x of V, containing all nodes y such that (x,y) Є E.

-  

- Advantages
  - Insertion/ Deletion simpler.
  - Possible to store additional information of nodes and edges.
  - Better when adjacency matrix is **sparse**
  - 

Traversal:

- **Depth First Traversal:** Beginning at a starting node depth-first search (DFS) recursively visits the first yet undiscovered direct neighbor of every reached node before continuing in the same manner with the remaining neighbors at this node.

Algorithm DFT (G <graph>)

Pre: G is graph structure;Post: Graph traversed in Depth First

```
1. for (all v in G)
    1. visited[v] = false
2. for (all v in G)
    1. if (not visited[v])
        1. traverse(v)
```

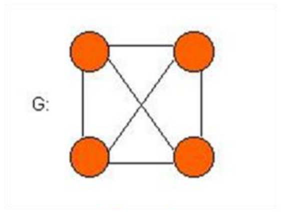
```
Algo traverse(v <vertex of graph>)
    1. visited[v] = true
    2. process v
    3. for (all w adjacent to v )
        1. if (not  visited[w])
            1. traverse(w)
```

- **Breadth First Traversal:** Beginning at a starting node breadth-first search visits all of its direct neighbors (having distance 1) before it in turn uses these already visited nodes as new starting nodes to continue with their direct neighbors. That way, it visits all reachable nodes at distance k to the original starting node before those at distance k+1.

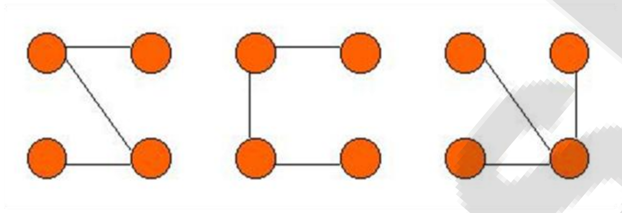
```
Algorithm BFS (G <graph>)

Pre:  G is graph structure;Post: Graph traversed in Breadth First
    1. for (all v in G)
        1. visited[v] = false
    2. for (all v in G)
        1. if (not visited[v])
            1. add(v, queue)
            2. while (queue not empty)
                1. v = delete (queue)
                2. if (not visited[v])
                    1. visited[v]=true
                    2. process v
                3. for (all w adjacent to v)
                    1. if (not visited[w])
        1. add(w, queue)
```

Spanning Trees:

- *A spanning tree of a connected graph is the smallest set of edges such that all nodes of the graph are connected. Addition of one more edge will result into a cycle.*



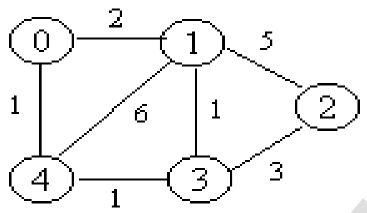
Graph



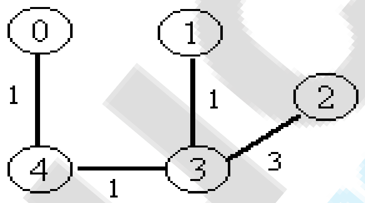
Three (of the many possible) spanning trees from graph

Minimum Spanning Trees (MST):

- A minimum spanning tree T of an graph G is a subgraph of G that connects all the vertices in G at the lowest total cost.



Graph G

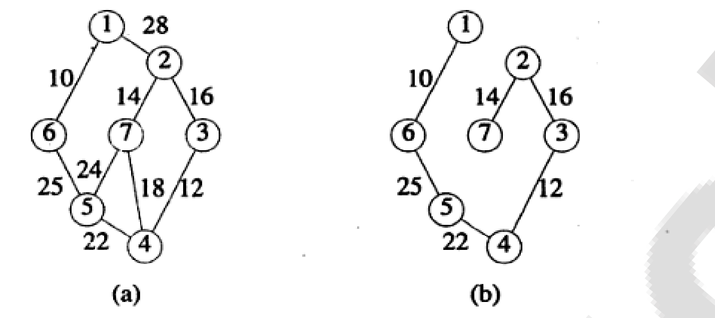


A Minimum Spanning Tree for G
(total cost = 6)

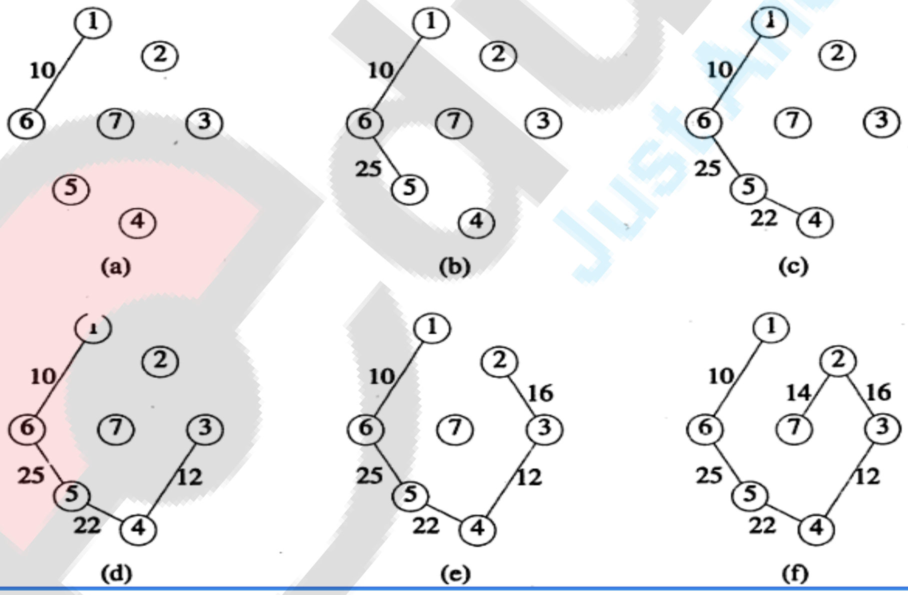
- If weights of edges in a network are unique, then there will be a unique MST. If there are duplicate weights, there can be more than one MST's.
- MST is used as one of the most important tools to analyze computer networks (e.g. cabling, network load capacity, optimal flow).
- Two algorithms:
  - Prim's algorithm
  - Kruskal's algorithm.

Prim's Algorithm:

- Include an edge of minimum cost in your Spanning Tree
- Repeatedly include the next edge (u, v)
  - u should be in the tree , v should not be.
  - Cost of (u, v) is minimum among all alternative edges (i.e., all edges connected from any vertex/node in the tree built so far).
- The algorithm stops when all the nodes have been reached



A graph and its minimum cost spanning tree

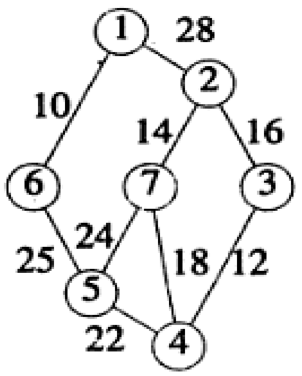


```
Algorithm Prim($E, cost, n, t$)
// $E$ is the set of edges in $G$. $cost[1:n, 1:n]$ is the cost
// adjacency matrix of an $n$ vertex graph such that $cost[i, j]$ is
// either a positive real number or $\infty$ if no edge $(i, j)$ exists.
// A minimum spanning tree is computed and stored as a set of
// edges in the array $t[1:n-1, 1:2]$. $(t[i,1], t[i,2])$ is an edge in
// the minimum-cost spanning tree. The final cost is returned.
{
    Let $(k, l)$ be an edge of minimum cost in $E$;
    $mincost := cost[k, l]$;
    $t[1,1] := k$; $t[1,2] := l$;
    for $i := 1$ to $n$ do  // Initialize near.
        if $(cost[i, l] < cost[i, k])$ then $near[i] := l$;
        else $near[i] := k$;
    $near[k] := near[l] := 0$;

    for $i := 2$ to $n-1$ do
    { // Find $n-2$ additional edges for $t$.
        Let $j$ be an index such that $near[j] \neq 0$ and
        $cost[j, near[j]]$ is minimum;
        $t[i,1] := j$; $t[i,2] := near[j]$;
        $mincost := mincost + cost[j, near[j]]$;
        $near[j] := 0$;
        for $k := 1$ to $n$ do // Update $near[\,]$.
            if $((near[k] \neq 0)$ and $(cost[k, near[k]] > cost[k, j]))$
                then $near[k] := j$;
    }
    return $mincost$;
}
```

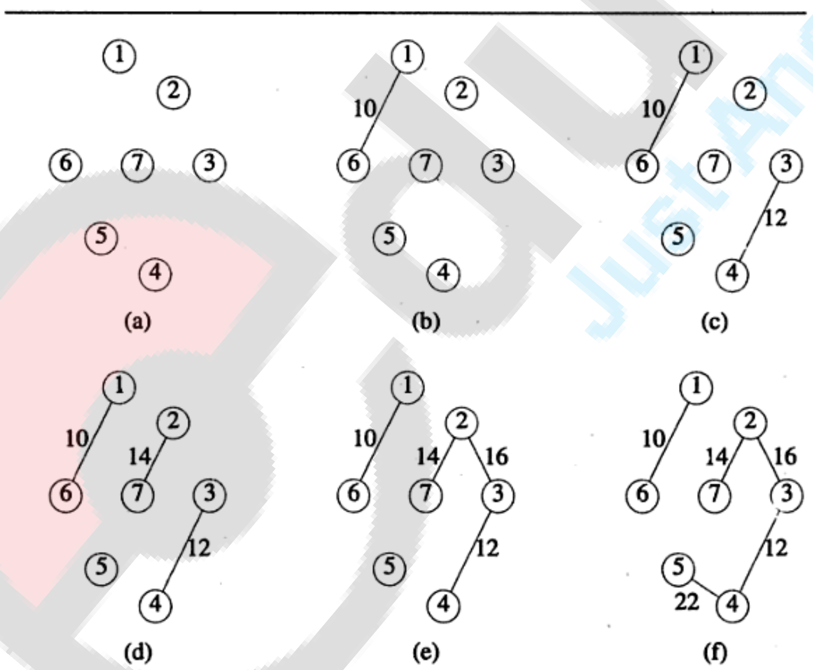
Kruskal's Algorithm:

- The main idea is to
  - start with a set (called forest) of singleton trees, and
  - merge two trees at a time, unless it creates a cycle in the merged tree, until the forest becomes one tree.

Find minimum spanning tree using kruskal's algorithm



(a)


(a)
(b)
(c)
(d)
(e)
(f)
10
12
14
16
22