CHAPTER 5
GRAPHS AND MULTIGRAPHS

5.1 INTRODUCTION

A graph G consists of two things:

(1) A set **V** of elements called **nodes** (or **points** or **vertices**)

(2) A set **E** of edges such that each edge e in **E** is identified with a unique (unordered) pair [u,v] of nodes in V, denoted by e = [u,v]

A graph is a **non linear data** structure.

Sometimes we indicate the parts of a graph by writing G = (V, E)

Suppose e = [u,v]. Then the nodes u and v are called endpoints of e, and u and v are said to be adjacent nodes or neighbors. The degree of a node u, written deg (u), is the number of edges containing u. If deg(u) = 0-that is, If u does not belong to any edge-then u is called an isolated node.

A path P of length n from a node u to a node v is defined as a sequence of n+ 1 node.

$$P= (v_0, v_1, v_2,.......,v_n)$$

Such that u = v_0; v_{i-1} is adjacent to v_i for i=1,2, ...n; and v_n=v. The path P is said to be closed if $v_0 = v_n$. The path P is said to be simple if all the nodes are distinct, with the exception that v_0 may equal v_n ; that is P is simple if the nodes v_0, v_1,......v_{n-1} are distinct and the nodes v_1, v_2,....,v_n are distinct. A cycle is a closed simple path with length 3 or more. A cycle of length k is called a k-cycle.

5.2 CONNECTED GRAPH & NON CONNECTED GRAPH

1. CONNECTED GRAPH

A graph is connected if there is a path between 2 vertices of the graph, i.e. if we can travel from any vertex to any vertex.

V = {A, B, C, D}

E = { a, b, c}

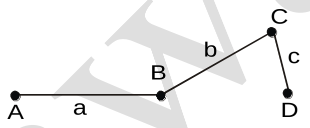


Figure 5.1

2. NON CONECTED GRAPH

Figure 5.2 is a non-connected graph because in no case can we travel to the vertex C. Figure 5.1 is a connected graph as all the vertices are reachable graph from any of the vertice. We can also reach the vertex D from A through B and C.

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

E = {a, b, c}

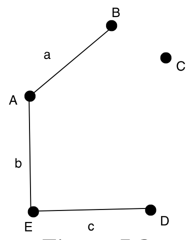


Figure 5.2

MULTIPLE EDGE

e_1 and e_2 are called multiple edge if they connect from the same pair of vertices.

i.e. e_i= (u,v) and e_2 = (u,v)

for figure 5.3

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

E= {e_1, e_2, e_3, e_4, e_5, e_6, e_7}

e_1 = (A, B)

e_2 = (A, B)

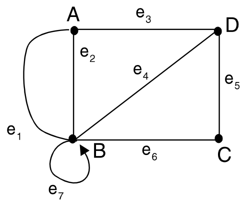


Figure 5.3

LOOPS

An edge e is called a loop if it has identical end points.
for figure 5.4
V = {A, B, C, D}
E = {a, b, c, d, e, f}
e = (D, D)
Hence, e forms a loop.

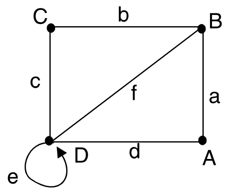


Figure 5.4

5.3 COMPLETE GRAPH

A graph is called a complete graph if for each pair of vertices, there is an edge in the graph.
for figure 5.5
V = {A, B, C, D}
E = {a, b, c, d, e, f}
If we take any pair of vertex, say **(A, C)** or **(D, B)**, we have an edge for it i.e. f and e respectively.

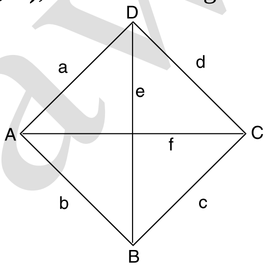


Figure 5.5

DEGREE OF VERTEX

Degree of a vertex is defined as the number of edge incident on it.
Let us take example of figure 5.6
V = {A, B, C, D, E}
E = {a, b, c, d, e, f, g}

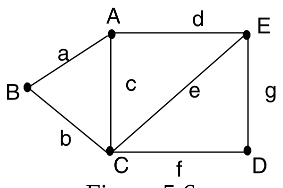


Figure 5.6

Vertex	Degree of Vertex
A	3
B	2
C	4
D	2

E	3

5.4 DIRECTED GRAPHS

A directed graph **G**, also called a **digraph**, is the same as a multigraph except that each edge e in G is assigned a direction, or in other words, each edge e is identified with an ordered pair (u,v) of nodes in G rather than an unordered pair [u,v].

Suppose G is a directed graph with a directed edge e = (u,v). Then e is also called an arc.

Moreover, the following terminology is used :

(1) e begins at u and ends at v.
(2) u is the origin or initial point of e and v is the destination or terminal point of e.
(3) u is a predecessor of v, and v is a successor or neighbor of u.
(4) u is adjacent to v and v is adjacent to u.

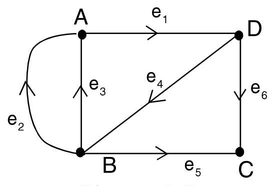


Figure .5.7

In figure 5.7

V = {A, B, C, D}

E = {e_1, e_2, e_3, e_4, e_5, e_6 }

Similarly

e_1 = (A, D) Here it cannot be (D, A)

e_2 = (B, A)

e_3 = (B, A)

e_4 = (D, B)

For the above example

Vertex	**Indegree**	**Outdegree**
A	2	1
B	1	3
C	2	0
D	1	2

Source and Sink

A vertex V is known as source if it has a positive outdegree, but 0 indegree.

A vertex, V is known as sink if it has a zero outdegree, but a positive indegree. For example in figure 5.7 C is the sink vertices, as can be seen from Table

Connected or Strongly Connected

A directed graph is said to be connected or strongly connected if for each pair of vertices U and V, there exists a path from U to V and from V to U.

V = {A, B, C, D}

E = {a, b, c, d, e, f, g, h}

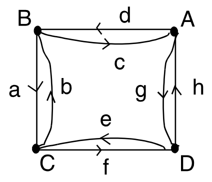


Figure 5.8

Reachable

A vertex V is said to be reachable from vertex U if there is a directed path from vertex U to V for example, in the figure5.8 each of the vertices is reachable from any vertex, such as vertex D is reachable from vertex B as there is a directed path from D to B via the vertex A.

Q. 5.1 consider the (undirected) graph G in figure 5.8 (a) Describe G formally in terms of its set V of nodes and its set E of edges. (b) Find the degree of each node.

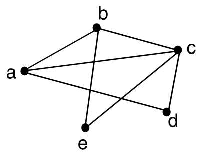


Figure 5.8 (a)

Solution :

(a) There are 5 nodes a, b, c d and e: hence V = {a, b, c, d, e}. There are 7 pairs [x,y] of nodes such that node x is connected with node y ; hence
E = { [a,b], [a,c], [a,d,], [b,c], [b,e], [c,d], [c,e]}

(b) The degree of a node is equal to the number of edges to which it belongs: for example, deg (a) = 3, since a belongs to three edges, [a,b], [a,c] and [a,d]. Similarly, deg (b)=3, deg(c) = 4, deg (d) = 2 and deg (e) = 2.

Q. 5.2 Consider the multigraphs in Fig. 5.8 (b) Which of them are (a) connected; (b) loop-free (i.e., without loops) ; (c) graphs ?

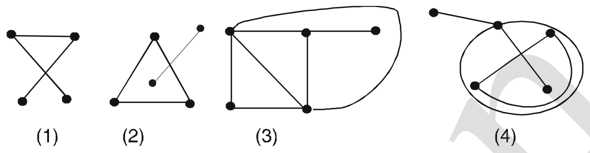


Figure 5.8(b)

Solution :

(a) Only multigraphs 1 and 3 are connected.

(b) Only multigraphs 4 has a loop (i.e. an edge with the same endpoints).

(c) Only multigraphs 1 and 2 are graphs, Multigraph 3 has multiple edges, and multigraph 4 has multiple edges and a loop.

Q. 5.3 Consider the connected graph G in fig.5.8 (c) Find all simple paths from node A to node F. (b) Find the distance between A and F. (c) Find the diameter of G. (The diameter of G is the maximum distance existing between any two of its nodes.)

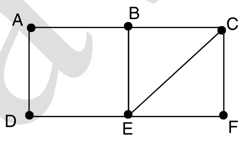


Figure 5.8(c)

Solution :

(a) A simple path from A to F is path such that no node and hence no edge is repeated. There are seven such simple paths:
(A, B, C, F) (A, B, E, F) (A, D, E, F) (A, D, E, C, F)
(A, B, C, E, F) (A, B, E, C, F) (A, D, E, B, C, F)

(b) The distance from A to F equals 3, since there is a simple path, (A, B, C, F), from A to F of length 3 and there is no shorter path from A to F.

(c) The distance between A and F equals 3 and the distance between any two nodes does not exceed 3; hence the diameter of the graph G equals 3.

Q. 5.4 Consider the (directed) graph G in fig 5.8 (d) Find all the simple paths from X to Z (b) Find all the simple paths from Y to Z. (c) Find indeg(Y) and outdeg (Y) (d) Are there any sources or sinks ?

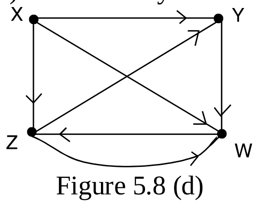


Figure 5.8 (d)

Solution :

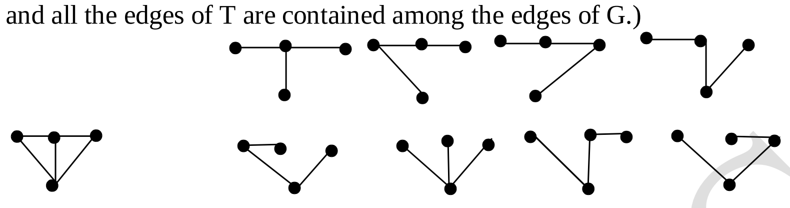
(a) There are three simple paths from X to Z : (X,Z) (X.W.Z) and (X, Y,W,Z).

(b) There is only one simple path from Y to Z (Y, W, Z).

(c) Since two edges enter Y (i.e. end at Y), we have indeg (Y) = 2. Since only one edge leaves Y (i.e., begins at Y) outdeg (Y) = 1.

(d) X is a source, since no edge enters X (i.e.indeg(x) = 0) but some edges leave X (i.e., outdeg (X) > 0). There are no sinks, since each node has a nonzero outdegree(i.e. each node is the initial point of some edge.)

Q. 5.5 find all spanning trees of the graph G shown in Fig. (a) (A tree T is called a spanning tree of a connected graph G if T has the same as G and all the edges of T are contained among the edges of G.)



(a) (b)

Solution:

There are eight such spanning trees, as shown in fig. (b) Since G has 4 nodes, each spanning tree T must have 4-1 = 3 edges. Thus each spanning tree can be obtained by deleting 2 of the 5 edges of G. This can be done in 10 ways. except that two of them lead to disconnected graphs. Hence the eight spanning trees shown are all the spanning trees of G.

REPRESENTATION OF A GRAPH

There are two ways by which a graph can be maintained in memory of a computer.

1. **Sequential representation or Array Representation** of G is by means of its adjacency matrix A
2. **Linked representation of G** is by means of linked lists of neighbors.

5.5 SEQUENTIAL REPRESENTATION OR ADJACENCY MATRIX

1. Adjacency matrix

Suppose G is a simple directed graph with n nodes, and suppose the nodes of G have been ordered and are called v_1, v_2,......v_n Then the adjacency matrix A = (a_{ij}) of the graph G is the n x n matrix defined as follows :

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is adjacent to } v_j \text{ that is if there is an edge } (v_i, v_j) \\ 0 & \text{otherwise} \end{cases}$$

Such matrix A, which contains entries of 0 and 1, is called bit matrix or Boolean matrix for a directed graph.

$a_{ij} = a_{ji}$ that is the adjacency matrix A of the directed graph G does depend on the ordering of the nodes of G, a different ordering of the nodes may result in a different adjacency matrix.

for a undirected graph G

The adjacency matrix A of G will be a symmetric matrix, i.e. one in which $a_{ij}=a_{ji}$ for every i and j

The adjacency representation of graph in fig.5.9 is as follow

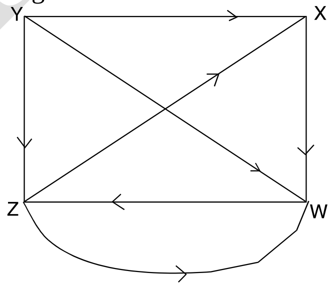


Figure 5.9

	X	Y	Z	W
X	0	0	0	1
Y	1	0	1	1
Z	1	0	0	1
W	0	0	1	0

$$A = 1 \begin{array}{cccc} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array}$$

In first row, the entry of 1 in the fourth column correspond to the edge between X and W and remaining entries in first row are O means there is no edge between X-X, X-Y, X-Z. Respectively.

Similarly In second row third and fourth column are entry are 1 means there in an edge between (Y and Z) and (Y and W), the size of matrix is equal to the numbers of vertices in a graph.

Consider the power A, A^2, A^3....of the adjacency matrix A of graph G. Let

$a_k(i,j)$ = the ij entry in the matrix A^k

Observe that $a_1(i,j) = a_{ij}$ gives the number of paths of length 1 from node v_i to node v_j. One can show that $a_2(i,j)$ gives the number of paths of length 2 from v_i to v_j.

Consider again the graph G in fig.5.9 adjacency matix A is given, the powers A^2, A^3 and A^4 of the matrix A follow:

$$A^2 = \begin{array}{cccc} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{array} \qquad A^3 = \begin{array}{cccc} 1 & 0 & 0 & 1 \\ 1 & 0 & 2 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{array} \qquad A^4 = \begin{array}{cccc} 0 & 0 & 1 & 1 \\ 2 & 0 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 1 & 0 & 1 & 1 \end{array}$$

Accordingly, in particular, there is a path of length 2 from v_4 to v_1, there are two paths of length 3 from v_2 to v_3, and there are 3 paths of length 4 from v_2 to v_4 (Here, v1= X, v_2= Y, v_3 = Z and v_4 = W)

Suppose we now define the matrix B as follows:

$B_r = A + A^2 + A^3 + \ldots\ldots + A^r$

5.6 PATH MATRIX

Let G be a simple directed graph with m nodes, $v_1, v_2 \ldots\ldots v_m$. The path matrix or reachability matrix of G is the m-square matrix $P = (P_{ij})$ defined as follows:

$$p_{ij} = \begin{cases} 1 & \text{if there is a path from } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

Suppose there is a path from v_i to v_j. Then there must be a simple path from v_i to v_j when v_i v_j, or there must be a cycle from v_i to v_j when v_i=v_j.

Let A be the adjacency matrix and let P= (p_{ij}) be the path matrix of a diagraph G. Then P_{ij} = 1 if and only if there is a nonzero number in the ij entry of the matrix.

$B_m = A + A_2 + A_3 + \ldots\ldots + A_m$

Consider the graph G with m= 4 nodes in fig. 5.9 Adding the matrix A, A^2,A^3 and A^4, we obtain the following matrix B_4, and replacing the nonzero entries in B_4 by 1, we obtain the path matrix P of the graph G:

$$B_4 = \begin{array}{cccc} 1 & 0 & 2 & 3 \\ 5 & 0 & 6 & 8 \\ 3 & 0 & 3 & 5 \\ 2 & 0 & 3 & 3 \end{array} \quad \text{and } P = \begin{array}{cccc} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{array}$$

5.7 PATH MATRIX USING WARSHALL'S ALGORITHM

Let's find out the path matrix of a Graph G with n number of nodes $V_1, V_2, V_3 \ldots V_n$. And also $P_o, P_1, P_2, \ldots P_n$ are the matrix of size n x n. Let $P_k[i,j]$ is the ith row jth column entry of matrix P_k.

1 If there is a simple path for V_i to V_j which does not use any other nodes except possibly $V_1, V_2, V_3 \ldots V_k$,

0 otherwise

We can also say that

$P_0[i,j] = 1$ if there is edge between V_i to V_j

$P_1[i,j] = 1$ if there is path between V_i to V_j which does not use any other nodes except V_1.

$P_2[i,j] = 1$ if there is path between V_i to V_j which does not use any other nodes except V_1 and V_2.

The entry in the path matrix $P_k[i,j]$ can be one only if one of the following case are there:

1. There is path between V_i and V_j that does not use any other vertex except $V_1, V_2 - V_{k-1}$

P_{k-1} [i,j] = 1 i.e.V_iV_j

2. There is path from V_i to V_k and V_k to V_j both does not use any other vertex except V_1 V_2 V_3 - V_{k-1}
   P_{k-1} [i,k] = 1 and P_{k-1}[k,j] = 1
   i.e. V_iV_k…………………….V_kV_j

So accordingly the elements in P_k [i,j] can be given as follows:

P_k [i,j] = P_{k-1} [i,j] or (P_{k-1} [i,k] AND P_{k-1} [k,j])

Warshall's Algorithm to Find Path Matrix.

Graph G is represented as a adjacency matrix G. And we are finding Path Matrix P. N is total Number of nodes

Algorithm 5.1

Warshall's Algorithm to Find Path Matrix.

```
Algorithm Warshalls_Path (Graph G)
{
          for (i = 1 to N)
          {
               for (j = 1 to N)
               {
                    if (G [i], [j] = 0)
                         P [i] [j] =0
                    else
                         P [i] [j] = 1
               }
          }
          for (k=1 to N)
          {
               for (i =1 to N)
               {
                    for (j=1 to N)
                    {
                      P[i] [j] = P[i] [j] OR (P[i] [k] AND P[k] [j])
                    }
               }
          }
}
```

5.9 SHORTEST PATH ALGORITHM

Algorithm 5.2

Let G be directed graph with n nodes and each edge e assigned a W(e) called weight or length of the edge e, we write the weight matrix W as follows.

W_{ij}= W (e) weight of an edges from V_i to V_j 0 if no edge.

Let's find now matrix S which tell us the shortest paths between two nodes.

S = S_{ij} is the length of Shortest path from V_i to V_j.

Lets modify the previous algorithm to find S.

S_k [i, j] = MINIMUM (S_{k-1} [i,j], S_{k-1} [i,k] + S_k[k,j])

Hence a formal algorithm can be written by just replacing statement

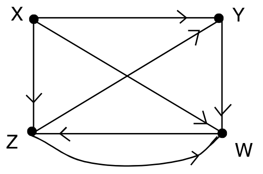
P[i] [j] = P[i][j] OR (P[i] [k] AND P [k] [j])

By S[i] [j] = MIN (S [i] [j], S[i] [k] + S[k] [j])

Rest every thing is same. Where MIN ( ) is a function returns a value which is minimum.

Q. 5.6 consider the graph G in fig. 5.8 (d) suppose the nodes are stored in memory in an array DATA as follows:

DATA: X, Y, Z, W
(a) Find the adjacency matrix A of the graph G.
(b) Find the path matrix P of G using powers of the adjacency matrix A.
(c) Is G strongly connected?



Solution:

(a) The nodes are normally ordered according to the way they appear in memory ; that is we assume v_1= X v_2 = Y v_3=Z and v_4 = W. the adjacency matrix A of G follows:

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

Here a_{ij} = 1 if there is node from v_i to v_j ; otherwise a_{ij} = 0.

(b) Since G has 4 nodes, compute A^2, A^3, A^4 and $B_4 = A + A^2 + A^3 + A^4$:

$$A^2 = \begin{matrix} 0&1&1&2 \\ 0&0&1&0 \\ 0&0&1&1 \\ 0&1&0&1 \end{matrix} \qquad A^3 = \begin{matrix} 0&1&2&2 \\ 0&1&0&1 \\ 0&1&1&1 \\ 0&0&1&1 \end{matrix}$$

$$A^4 = \begin{matrix} 0&2&2&3 \\ 0&0&1&1 \\ 0&1&1&2 \\ 0&1&1&1 \end{matrix} \qquad B^4 = \begin{matrix} 0&5&6&8 \\ 0&1&2&3 \\ 0&3&3&5 \\ 0&2&3&5 \end{matrix}$$

The path matrix P is now obtained by setting P_{ij} = 1 wherever there is a nonzero entry in the matrix B_4 Thus

$$P = \begin{matrix} 0&1&1&1 \\ 0&1&1&1 \\ 0&1&1&1 \\ 0&1&1&1 \end{matrix}$$

(c) The path matrix shows that there is no path from v_2 to v_1. In fact, there is no path from any node to v_1. Thus G is not strongly connected.

Q. 5.7 consider the graph G in fig 5. 8 (d) . and its adjacency matrix A obtained in Prob, 5.6 Find the path matrix P of G using Warshall's algorithm rather than the powers of A.

Solution:

Compute the matrices P_0, P_1, P_2, P_3 and P_4 where initially P_0 = A and

$P_k [i,j] = P_{k-1}[i,j], v(P_{k-1}[i,j]\ \ P_{k-1}[k,j])$

That is

$P_k[i,j] = 1$ if $P_{k-1}[i,j] =1$ or both $P_{k-1}[i,k]=1$ and $P_{k-1}[k,j] =1$

Then :

$$P_1 = \begin{matrix} 0&1&1&1 \\ 0&0&0&1 \\ 0&1&0&1 \\ 0&0&1&0 \end{matrix} \qquad P_2 = \begin{matrix} 0&1&1&1 \\ 0&0&0&1 \\ 0&1&0&1 \\ 0&0&1&0 \end{matrix}$$

$$P_3 = \begin{matrix} 0&1&1&1 \\ 0&0&0&1 \end{matrix} \qquad P_4 = \begin{matrix} 0&1&1&1 \\ 0&1&1&1 \end{matrix}$$

```
0 1 0 1        0 1 1 1
0 1 1 1        0 1 1 1
```

Observe that $P_0 = P_1 = P_2 = A$. The changes in P_3 occur for the following reasons:

$P_3(4,2) = 1$ because $P_2(4,3) = 1$ and $P_2(3,2)=1$

$P_3(4,4) = 1$ because $P_2(4,3)=1$ and $P_2(3,4)=1$

The changes in P_4 occur similarly. The last matrix, P_4 is the required path matrix P of the graph G.

Q. 5.8 consider the (undirected) weighted graph G in fig below. Suppose the nodes are stored in memory in an array DATA as follows.

DATA: A, B, C, X, Y

Find the weight matrix $W = (w_{ij})$ of the graph G.

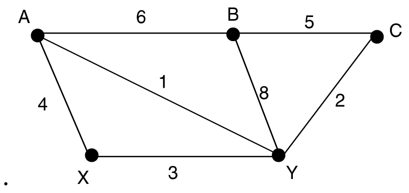


Figure (a)

Solution :

Assuming $v_1 = A$, $v_2 = B$, $V_3 = C$, $v_4 = X$ and $v_5 = Y$, we arrive at the following weight matrix

$$W = \begin{matrix} 0 & 6 & 0 & 4 & 1 \\ 6 & 0 & 5 & 0 & 8 \\ 0 & 5 & 0 & 0 & 2 \\ 4 & 0 & 0 & 0 & 3 \\ 1 & 8 & 2 & 3 & 0 \end{matrix}$$

Here W_{ij} denotes the weight of the edge v_i to v_j. Since G is undirected, W is symmetric matrix, that is $W_{ij} = W_{ji}$

Q. 5.9 suppose G is graph (undirected) which is cycle-free, that is, without cycles, Let $P= (p_{ij})$ be the path matrix of G.

(a) When can an edge $[V_i, V_j]$ be added to G so that G is still cycle - free ?

(b) How does the path matrix P change when an edge $[v_i, v_j]$ is added to G ?

Solution :

(a) The edge $[v_i, v_j]$ will form a cycle when it is added to G if and only if there already is a path between v_i and v_j. Hence the edge may be added to G when $P_{ij} = 0$.

(b) First set $p_{ij}= 1$, since the edge is a path from v_i to v_j. Also, set $p_{st}= 1$ if $p_{si} = 1$ and $p_{jt}= 1$. In other words, If there are both a path P_1 from v_s to v_i and a path P_2 from v_j to v_t then P_1, $[v_i, v_j]$ P_2 will form a path from v_s to v_t.

Q. 5.10 a minimum spanning tree T of a weighted graph G is spanning tree of G which has the minimum weight among all the spanning trees of G.

(a) Describe an algorithm to find a minimum spanning tree T of a weighted graph G.

(b) Find a minimum spanning tree T of the graph in Fig. (a) in problem 5.8

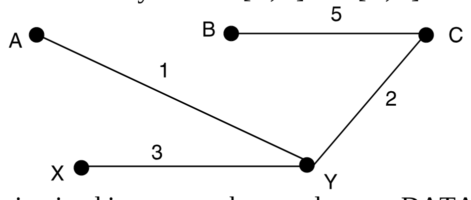
Solution:

(a) Algorithm This algorithm finds a minimum spanning tree T of a weighted graph G.

```
1. Order all the edges of G according to increasing weights.
2. Initialize T to be a graph consisting of the same nodes as G and no edges.
3. Repeat the following M -1 times, where M is the number nodes in G.
   Add to T an edge E of G with minimum weight such that E does not form a cycle in T.
[End of loop].
4. Exit.
```

Step 3 may be implemented using the results of Prob 5.9 Problem 5.9(a) tells us which edge e may be added to T so that no cycle is formed - i.e. so that T is still cycle - free-and prob. 5.9 (b) tell us how to keep track of the path matrix P of T as each edge e is added to T.

(b) Apply Algorithm P to obtain the minimum spanning tree T in fig. Although [A,X] has less weight than [B,C], we cannot add [A,X] to T, since it would form a cycle with [A,Y] and [Y,X].



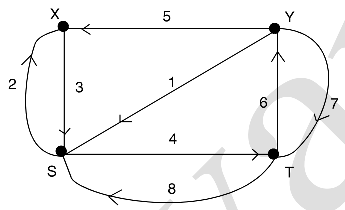
Q. 5.11 suppose a weighted graph G is maintained in memory by a node array DATA and a weight matrix W as follows:

Data: X, Y,S, T

$$W = \begin{matrix} 0 & 0 & 3 & 0 \\ 5 & 0 & 1 & 7 \\ 2 & 0 & 0 & 4 \\ 0 & 6 & 8 & 0 \end{matrix}$$

Draw a picture of G.

The picture appears in Figure below. The nodes are labeled by the entries in DATA. Also, if w_{ij} 0, then there is an edge from v_i to v_j with weight w_{ij} (We assume v_1 = X, v_2 = Y, v_3=S and v_4 = T, the order in which the nodes appear in the array DATA).



5.10 LINK LIST REPRESENTATION

This types of graph representation has two parts

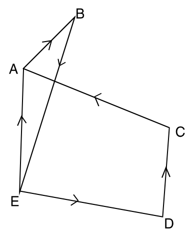
1. Vertex Nodes.
2. Edge Nodes.

Vertex Node

Address of Next vertex	Name of vertex	Address of edge

Edge Node

Address of the vertex	Name of edge	Address of next edge



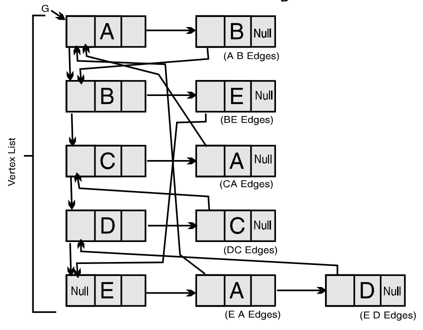


Figure 5.10

O. 5.12 suppose a graph G is maintained in memory in the form.
GRAPH (NODE, NEXT, ADJ, START, DEST, LINK)
Write a procedure which finds the indegree INDEG and the outdegree OUTDEG of each node of G.

Solution:

First we traverse the node list, using the pointer PTR in order to initialize the arrays INDEG and OUTDEG to zero. Then we traverse the node list, using the pointer PTRA, and for each value of PTRA, we traverse the list of neighbors of NODE [PTRA], using the pointer PTRB. Each time an edge is encountered. PTRA gives the location of its initial node and DEST [PTRB] gives the location of its terminal node. Accordingly, each edge updates the arrays INDEG and OUTDEG as follows:

OUTDEG [PTRA] : = OUTDEG[PTRA] + 1

and

INDEG [DEST [PTRB]]: = INDEG [DEST [PTRB]] +1

The formal procedure follows:

```
DEGREE(NODE,NEXT,ADJ, START, DEST, LINK, INDEG, OUTDEG)
```

Procedure P :

```
DEGREE(NODE,NEXT,ADJ, START, DEST, LINK, INDEG, OUTDEG)
This procedure finds the indegree INDEG and outdegree OUTDEG of each node in the
graph G in memory :
1. [Initialize arrays INDEG and OUTDEG]
    (a) Set PTR: START.
    (b) Repeat while PTR! = NULL: [Traverses node list.]
            (i) Set INDEG[PTR]:=0 and OUTDEG [PTR] : =0
            (ii) Set PTR : = NEXT [PTR]
        [End of loop.]
2. Set PTRA: = START.
3. Repeat Steps 4 to 6 while PTRA!= NULL : [Traverse node list.]
4.      Set PTRB : = ADJ[PTRA]
5.      Repeat while PTRB != NULL [Traverse list of neighbors]
            (a) Set OUTDEG[PTRA]:=OUTDEG[PTRA] + 1 and
                INDEG [DEST [PTRB]]: = INDEG [DEST [PTRB]] + 1.
            (b) Set PTRB : = LINK [PTRB]
        [End of inner loop using pointer PTRB.]
6.      Set PTRA : = NEXT [PTRA]
    [End of step 3 outer loop using the pointer PTRA.]
7. Return.
```

Q.5.13 Suppose G is a finite undirected graph. Then G consists of finite number of disjoint connected components. Describe an algorithm which finds the number NCOMP of connected components of G. Furthermore, the algorithm should assign a component number COMP (N) to every node N in the same connected component of G such that component numbers range from 1 to NCOMP.

Solution :

The general idea of the algorithm is to use a breadth - first or depth first search to find all nodes N reachable from a starting node A and to assign them the same component number. The algorithm follows.

Algorithm: Finds the connected components of an undirected graph G.

```
1. Initially set COMP (N): = 0 for every node N in G, and initially set L: = 0.
2. Find a node A such that COMP (A) = 0. If no such node A exists, then:
        Set NCOMP: = L, and Exit.
   Else
        Set L: = L + 1 and Set COMP (A): = L
```

3. Find all nodes N in G which are reachable from A (using a breadth first search or a depth first search) and set COMP (N) = L for each such node N.
4. Return to Step 2.

5.13 TRAVERSING A GRAPH

During the execution of our algorithms, each node N of G will be in one of three states, called the status on N, as follows:

STATUS = 1	:	(Ready state) The initial state of the node N.
STATUS = 2	:	(Waiting state) The node N is on the queue or stack, waiting to be processed.
STATUS = 3	:	(Processed state) The node N has been processed.

We now discuss the two searches separately.

5.14 BREADTH-FIRST SEARCH

The general idea behind a breadth-first search beginning at a starting node A is as follows. First we examine the starting node A. Then we examine all the neighbors' of A. Then we examine all the neighbors of the neighbors of A. And so on. Naturally, we need to keep track of the neighbors of a node, and we need to guarantee that no node is processed more than once. This is accomplished by using a queue to hold nodes that are waiting to be processed, and by using a field STATUS which tells us the current status of any node. The algorithm follows.

```
ALGORITHM BREADTH-FIRST SEARCH

This algorithm executes a breadth-first search on a graph G beginning at a starting node A.
1. Initialize all nodes to the ready state (STATUS = 1)
2. Put the starting node A in QUEUE and change its status to the waiting state
   (STATUS = 2).
3. Repeat Steps 4 and 5 until QUEUE is empty:
4.        Remove the front node N of QUEUE. Process N and change the status of N to
          the processed state (STATUS = 3).
5.        Add to the rear of QUEUE all the neighbors of N that are in the steady state
          (STATUS =1), and change their status to the waiting state (STATUS = 2).
   [End of Step 3 loop].
6. Exit.
```

Algorithm 5.6:

5.15 DEPTH - FIRST SEARCH

The general idea behind a depth - first search beginning at a starting node A is as follows. First we examine the starting node A. Then we examine each node N along a path P which begins at A ; that is, we process a neighbor of A, then a neighbor of a neighbor of A, and so on. After coming to a "dead end.'' that is, to the end of the path P, we backtrack on P until we can continue along another path P'.and so on. (This algorithm is similar to the inorder traversal of a binary tree, and the algorithm is also similar to the way one might travel through a maze.) The algorithm is very similar to the breadth-first search except now we use a stack instead of the queue. Again, a field STATUS is used to tell us the current status of node. The algorithm follows.

Algorithm 5.7:

```
ALGORITHM DEPTH - FIRST SEARCH
```

```
This algorithm executes a depth-first search on a graph G beginning at a starting node A.
1. Initialize all nodes to the ready state (STATUS =1)
2. Push the starting node A onto STACK and change its status to the waiting state
   (STATUS = 2).
3. Repeat steps 4 and 5 until STACK is empty.
4.      Pop the top node N of STACK. Process N and Change its status to the
        processed state (STATUS = 3).
5.      Push onto STACK all the neighbors of N that are still in the ready state
        (STATUS = 1) and change their status to the waiting state (STATUS = 2).
[End of Step 3 loop.]
6. Exit.
```

5.16 TOPOLOGICAL SORT

Algorithm 5.8:

```
ALGORITHM TOPOLOGICAL SORT

This algorithm finds a topological sort T of graph S without cycles.
1. Find the indegree INDEG (N) of each node N of S. (This can be done by traversing
   each adjacency list)
2. Put in a queue all the nodes with zero indegree.
3. Repeat steps 4 and 5 until the queue is empty.
4.     Remove the front node N of the queue (by setting FRONT: = FRONT + 1)
5.     Repeat the following for each neighbor M of the node N:
           (a) Set INDEG (M) = INDEG (M) -1
               [This deletes the edge from N to M.]
           (b) If INDEG (M) = 0, then: Add M to the rear of the queue.
           [End of loop.]
   [End of Step 3 loop.]
6. Exit.
```

5.17 Spanning Tree

A spanning tree for an undirected graph G is a graph T consisting of the nodes of G together with enough edges from G such that:

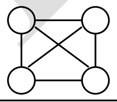
1) There is a path between each pair of nodes in T.

2) There is no simple cycle in T.

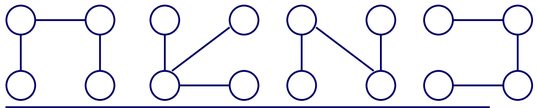
It should be clear that if G is connected if and only if there is a spanning tree for G. Thus, there is a spanning tree for every connected component of a graph.

If a graph G= (V, E) contain N nodes, then the spanning tree for that graph contains N-1 edges. The edges of spanning tree are subset of E.

A "minimal spanning tree" for a weighted graph G is a spanning tree such that the sum of its weights is less than or equal to the sum of its heights of every other spanning tree for G. That is in a minimal spanning tree the sum of weights of the edges is as small as possible.



A connected, undirected graph



Four of the spanning trees of the graph

All these are the spanning trees for the above graph G.

The minimum spanning tree is a tree because it is acyclic, it is spanning because it covers every edge, and it is minimum .Example for minimum

Spanning tree is wire a house with a minimum spanning tree of a weighted Graph.

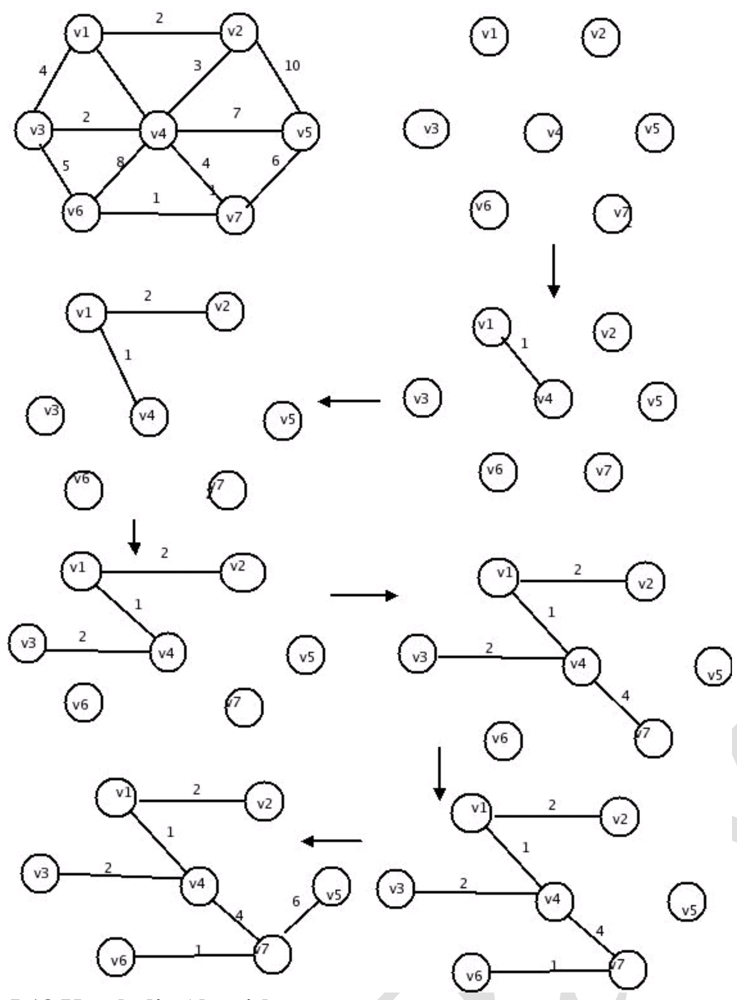
1)Prim's Algorithm

2)krushkal's Algorithm

Prim's Algorithm:-

One way to compute a minimum spanning tree is to grow the tree in successive stages. In each stage, one node is picked as a root, and we add an edge, and thus an associated vertex, to the tree.

```
Prim's algorithm
    T = a spanning tree containing a single node s;
    E = set of edges adjacent to s;
    while T does not contain all the nodes {
        remove an edge (v, w) of lowest cost from E
        if w is already in T then discard edge (v, w)
        else {
            add edge (v, w) and node w to T
            add to E the edges adjacent to w
        }
    }
```



5.18 Kruskal's Algorithm:-

It is continually selects the edges in order of smallest weight and accept an edge if it does not cause a cycle.

Kruskal's algorithm

```
T = empty spanning tree;
E = set of edges;
N = number of nodes in graph;
while T has fewer than N - 1 edges {
        remove an edge (v, w) of lowest cost from E
        if adding (v, w) to T would create a cycle
                then discard (v, w)
                else add (v, w) to T
}
```

Finding an edge of lowest cost can be done just by sorting the edges