

# STUDY MATERIAL ON GRAPH THEORY

2023

PREPARED BY

**DR. RAKESH SARKAR**

ASSISTANT PROFESSOR

DEPARTMENT OF MATHEMATICS

GOUR MAHAVIDYALYA

MANGALBARI, MALDA

# 1 Graph Theory

## 1.1 Introduction

The first occurrence of graph in the Mathematical history is considered to be the classical “Konigsberg Bridge Problem”. The problem is stated by the great mathematician L. Euler who lived in Konigsberg, as below:

“Konigsberg is divided into four parts by river Pregel and connected by seven bridges. Is it possible to tour Konigsberg along a path that crosses every bridge once and only once and return to the starting point?”

The diagram is given as below:



Fig:1.1

In proving the fact that the problem is unsolvable, Euler represented above image in the form of a “graph”, as follows.

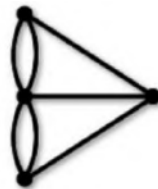


Fig:1.2

The points denote the land and the lines denote bridges.

In fact, rather than only proving that above problem is unsolvable, Euler introduced type of graphs, which can be traceable by starting from one point, traversing every line once and returning to the starting point, which is termed as Euler Graph. In Chapter 4, we shall discuss Euler Graphs in detail.

In modern times, Graph theory is applicable in many areas, such as, Chemistry, Electronics, and Networks, to name a few.

In this chapter, we will get acquainted ourselves with terminology and basic concepts of Graph Theory

## 1.2 APPLICATIONS OF GRAPH THEORY

Though graph “theory” appears to be a theoretical and hence pure mathematical term, we shall be amazed to know the areas in which it can be applied. In this section, we shall just quote a few in which graph theory is applied.

1. Graph Theory is helpful in making robots function autonomously.
2. Graph Theory is used to solve actual crimes.
3. Mathematics, often called the universal language, also forms a bridge between languages. Machine translators use Graph Theory to achieve good translations efficiently.
4. Descriptions of cellular activity involve a combination of continuous models. The analysis of cells requires usage of Graph Theory as well.
5. Researchers use graph theory to find near-optimal solutions saving industry time and money. (Travelling salesman’s problem, Chinese postman’s problem).
6. The Graph Theory is applicable in Road and Rail Traffic network.
7. The Graph Theory is applicable in planning tournaments (such as football, chess).
8. Hierarchy in the office, such as Chairperson is the root and people work under him are at various levels.
9. The Graph Theory is present in the virtual world of internet such as www (world wide web), social networking, searching data, data mining, and so on.

### 1.3 BASIC GRAPH THEORY DEFINITIONS AND NOTATIONS

**Simple Graph:** Simple graph is a set  $G (V(G), E(G))$ ,  $V(G)$  the set of vertices (points) and  $E(G)$  the set of edges (lines) disjoint from  $V(G)$ , together with an incidence function  $G$ , that associates with each edge of  $G$  a distinct unordered pair of vertices of  $G$ .

**Directed Graph:** A directed graph or digraph is a graph  $G (V, E)$  in which edges are ordered pairs  $(u, v)$  where  $u, v \in V$  That is if there is an edge from  $u$  to  $v$ , there may or may not be an edge from  $v$  to  $u$ .

#### 1.3.1 Example 1:

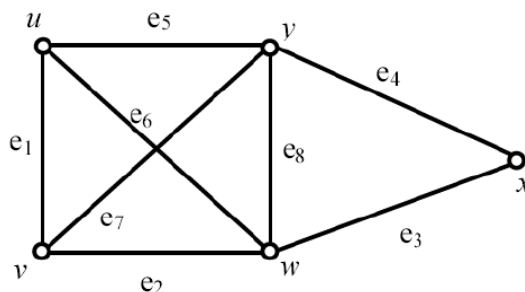


Fig:1.3

#### 1.3.2 Example 2:

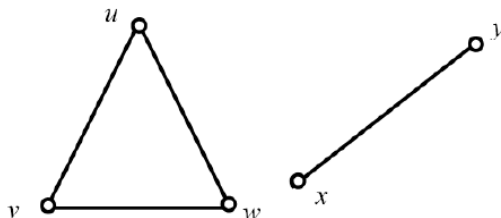


Fig:1.4

In the Example 1 graph  $G$  has,  $V = \{u, v, w, x, y\}$  and

$$E = \{ e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8$$

}

In the Example 2, graph  $G$  has,  $V = \{u, v, w, x, y\}$  and  $E = \{uv, uw, vw, xy\}$ .

**Note:** 1. Typically number of vertices in a graph  $G$  is denoted by letter  $p$  and edges by  $q$ .

2. A graph in which both vertex set and edge set is finite is called as finite graph.

3. In this and the subsequent chapters, we shall mainly discuss finite graphs.
4. The graph with no vertices (and hence no edges) is termed as null graph.
5. A graph with just one vertex is termed as trivial graph.
6. We are discussing non-trivial and non-null graph.
7. In a graph, if an edge with identical end vertices is called as loop.
8. In a graph, two or more edges with same end vertices are termed as parallel edges.
9. A graph, in which loop and / or parallel edges are permitted, is termed as Multigraph.
10. Graph of Fig. 1.6 is a digraph or directed graph.

In the graph of Fig. 1.2, that is the graph of Königsberg's bridge problem, we can observe parallel edges and the graph of Fig. 1.5, there is a loop at vertex  $u$ .

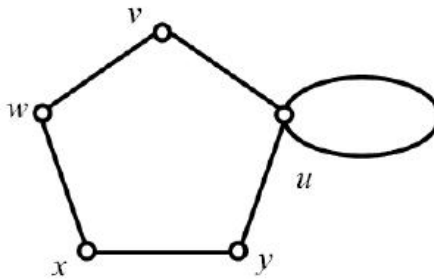


Fig:1.5

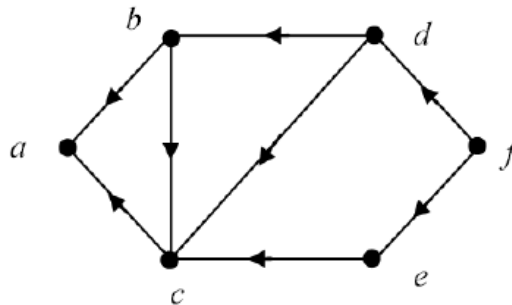


Fig:1.6

**Degree:** The degree of a vertex of a graph is the number of edges incident to the vertex, with loops counted twice. We denote degree of vertex  $v$  in a graph  $G$  is denoted by  $\deg G (v)$ .

In Fig.1.3, degree of  $u$  is 3, where as in Fig.1.5, degree of  $u$  is 4.

**Note:** In case of directed graph, every vertex has two types of degrees, in-degree (that is number of edges entering the vertex) and out-degree (that is number of edges leaving the vertex). For the graph of Fig. 1.6, in-deg ( $a$ ) =

out-deg (a) = 1; in-deg (b) = 1, out-deg (b) = 2; in-deg (c) = 3, out-deg (c) = 1; in-deg (d) = 1, out-deg (d) = 2; in-deg (e) = outdeg (e) = 1; indeg (f) = 0, out-deg (f) = 2.

**Walk:** A walk consists of an alternating sequence of vertices and edges consecutive elements of which are incident, which begins and ends with a vertex.

In Fig. 1.3,  $\{ue_1ve_7ye_7ve_2w\}$  is a walk.

**Trail:** A trail is a walk in which no edges are repeated.

In Fig. 1.3

$$ue_1ve_2we_6ue_5y$$

is a trail.

**Path:** A path is a trail in which no vertices (except possibly the end vertices) are repeated.

In Fig 1.3,

$$ue_1ve_7ye_8we_3x$$

is a path

**Circuit:** A circuit is a closed trail (that is end vertices are same) with at least one edge is known as Circuit. In Fig. 1.3,

$$ue_1ve_7ye_8we_3xe_4ye_5u$$

is a circuit. It can also be written, only in terms of vertices as:  $uvywxyu$ .

**Cycle:** A cycle is a circuit in which no edge is repeated.

In Fig. 1.3,

$$ue_1ve_2we_3xe_4ye_5u$$

is a cycle.

It can also be written as  $uvwxyu$ , in terms of vertices alone.

**Subgraph:** A graph  $H = (H(V), H(E))$  is called as subgraph of  $G = (G(V), G(E))$ , if  $H(V) \subseteq G(V)$  and  $H(E) \subseteq G(E)$

Fig. 1.6 below is a subgraph of Fig. 1.3

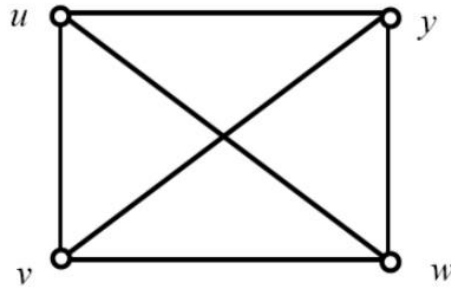


Fig:1.7

Let  $G(V, E)$  be a graph. We can obtain a subgraph from a graph in any one of the following ways.

1. A subgraph  $H$ , can be obtained by deleting vertex subset  $U$  of  $V$  and by deleting all the edges from  $E$  which are incident with a vertex in  $U$ .

2. A subgraph  $H$ , can be obtained by deleting an edge set  $D$  that is subset of  $E$  and vertex set of  $H$  is same as vertex set of  $G$ . Such a subgraph is called as spanning subgraph of  $G$ .

**Connected Graph:** Graph  $G$  is connected if and only if there exists a walk between any pair of vertices.

Graph in Fig. 1.3 is connected.

**Disconnected Graph:** Graph  $G$  is disconnected if and only if there exists at least one pair of vertices which is not connected by a walk.

Graph if Fig. 1.4 is disconnected, as there is no walk between vertices  $u$  and  $x$ .

The distance  $d(u, v)$  between two vertices  $u$  and  $v$  of a graph  $G$  is the length of the shortest path (often termed as geodesic) joining them if any; otherwise  $d(u, v) = \infty$

In a simple connected graph, distance is a metric; that is for all vertices  $u$ ,  $v$ , and  $w$ ,

1.  $d(u, v) \geq 0$  and  $d(u, v) = 0$  if and only if  $u = v$ .
2.  $d(u, v) = d(v, u)$
3.  $d(u, v) + d(v, w) \geq d(u, w)$

The diameter  $d(G)$  of a connected graph  $G$  is the length of any longest geodesic. In the graph of Fig. 1.3,  $d(G)$  is 2.]

**Complement of a Graph:** Let  $G$  be a simple graph. The complement  $G^c$  of  $G$  is the simple graph whose vertex set is  $V$  (that is same the vertex set of  $G$ ) and whose edges are the pairs of nonadjacent vertices of  $G$ .

Fig. 1.8 below is the complement of the graph of Fig. 1.3

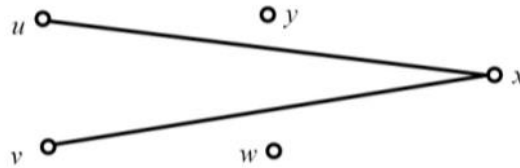


Fig:1.8

**Components:** Connected component or Component of a graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph.

For example graph of Fig. 1.9 below is made up of three components.



Fig. 1.9

**Proof:** Let  $G(V, E)$  be a disconnected and non-trivial graph and

$$G^c$$

be its complement. Let  $u, v$  be any two vertices in  $V$ . If there is no edge  $uv$  in  $G$ , then  $uv$  will be an edge in  $G^c$ . If the edge  $uv$  exists in  $G$ , then vertices  $u$  and  $v$  belong to same component (say  $H$ ) of  $G$ . As  $G$  is disconnected it has at least two components. Let  $w$  be a vertex in  $V$  which belongs to a component other than  $H$ . Then, there are no edges  $uw$  and  $wv$  in  $G$ . Hence,  $uw$  and  $wv$  be edges in

$$G^c$$

, and hence we get a  $u$ - $v$  path ( $u$ - $w$ - $v$ ) in

$$G^c$$

. Thus, any between two arbitrary vertices  $u, v$  of  $V$  there is a path in  $G^c$ . Hence,  $G^c$  is connected. Note: The converse of the above result is not true. That is complement of a connected graph need not be connected. As an example consider the graphs in the Fig.1.10 below.

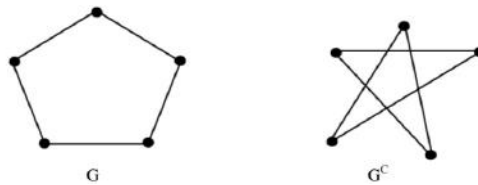


Fig:1.09

## 1.4 DIFFERENT TYPES OF GRAPHS

In this section we shall define and draw different types of graphs which will be useful for us in further discussion. **Complete Graph:** A graph  $G$  is said to be complete, if every vertex of  $G$  is connected to every other vertex in the vertex set of  $G$ . 1.5 Mathematical Representation of Graph

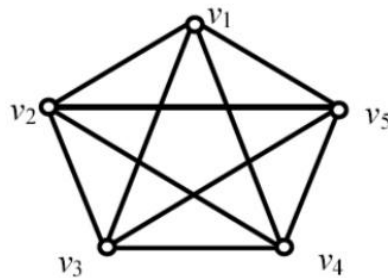
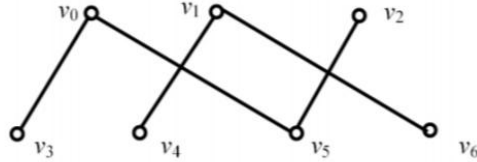


Fig:1.10

**Bipartite Graph:** A graph  $G$  is said to be bipartite, if vertex set is divided into two disjoint sets such that no two vertices in the same set are connected.



Fig. 1.11 below is an example of a bipartite graph in which  $v = v_1 \cup v_2$   
and  $v_1 = \{v_0, v_1, v_2\}$   
and  $v_2 = \{v_3, v_4, v_5, v_6\}$



Lemma 1.4.1: If  $G$  is a bipartite graph having partitions  $X$  and  $Y$ , then  
 $\sum_{v \in X} deg(v) = \sum_{v \in Y} deg(v)$

Proof: We shall prove this lemma by induction on number of edges of  $G$ .  
Let  $|X| = r$  and  $|Y| = s$ , for  $r, s \geq 1$ . (For if  $r = s = 1$ , then only one edge can  
be drawn and the lemma is trivially true.)

Take subgraph of  $G$  consisting of only vertices of  $G$ . Now, we shall start with  
an induction. Add one edge from any vertex of  $X$  and any vertex from  $Y$ . Then,  
 $\sum_{v \in X} deg(v) = \sum_{v \in Y} deg(v)$ .

Now, suppose this is true for  $n - 1$  edges, then on adding one more edge,  
exactly 1 is added to both

$$\sum_{v \in X} deg(v) = \sum_{v \in Y} deg(v)$$

$K_{m,n}$ : If a vertex set of a bipartite graph is partitioned into sets of sizes  $m$   
and  $n$ , respectively and every vertex in the first set connected to every vertex  
in the set two then such a bipartite graph is known as complete bipartite graph  
and is denoted by  $K_{m,n}$ .

$P_n$  is a path on  $n$  vertices.

$C_n$  is a cycle on  $n$  vertices.

It is very interesting to note the following theorems.

**Theorem :** Let  $G$  be a graph in which all vertices have degree at least two.  
Then  $G$  contains a cycle.

**Proof:** If  $G$  has a loop, it contains a cycle of length one, and if  $G$  has parallel  
edges, it contains a cycle of length two. So we may assume that  $G$  is simple.

Let  $P = [v_0 v_1 \dots v_{k_1} v_{k_2}]$  be a longest path in  $G$ . Because the degree of  $v_{k_1}$   
is at least two, it has a neighbour  $v$  different from  $v_{k_2}$ . If  $v$  is not on  $P$ , the path  
 $v_0 v_1 \dots v_{k_1} v v_{k_1} v_{k_2} \dots v_{k_1} v_{k_2} v$  is longer than  $P$ , which contradicts that  $P$  is a longest path.  
Therefore,  $v = v_i$ , for some  $i, 0 \leq i \leq k_2$ .

**Theorem:** Let  $G$  be undirected graph.  $G$  is bipartite if and only if it has no odd cycles.

**Proof:** Let  $G$  be bipartite graph. Let if possible it has an odd cycle. Let the cycle be  $v_1v_2\dots v_{2k+1}v_1$ . Let the two disjoint vertex sets of  $G$  be  $A$  and  $B$ . Then, we have  $v_1 \in A, v_2 \in B, v_3 \in A, v_4 \in B$  and so on  $v_{2k+1} \in A, v_1 \in B$ , a contradiction that  $G$  is bipartite, as  $v_1 \in A$  as well as  $v_1 \in B$ .

Thus,  $G$  has no odd cycle.

Conversely, let  $G$  has no odd cycles. We have to show that  $G$  is bipartite.

Without loss of generality, let  $G$  be connected, as the same logic can be applied to each of the components.

Choose any vertex  $v$  in the vertex set  $V$  of  $G$ .

Let  $A$  be the set of vertices such that the shortest path from  $v$  to each of the vertex in  $V$  is of odd length and  $B$  be the set of vertices such that the shortest path from  $v$  to each of the vertex in  $V$  is of even length. Then,  $v \in B$ . Also  $A \cup B = V$  and  $A \cap B = \phi$ .

We shall prove that  $A, B$  is the partition of  $G$ .

For, if not, there exists two incident vertices  $x_1$  and  $x_2$ , both in  $A$  or both in  $B$ . Without loss of generality, let both are in  $A$ . Then,  $x_1-v$  there is a path of odd length,  $v-x_2$  there is a path of odd length and hence  $v-x_2-x_1-v$  is in an odd cycle in  $G$ , a contradiction.

Previous theorem says that an even graph contains a cycle. Now let us prove even stronger result. That is an even graph can be partitioned into cycle and conversely.

**Theorem:** Let  $G (V, E)$  be an even graph. Then the edge set  $E$  of  $G$  can be partitioned into cycles such that no two cycles will share an edge.

**Proof:** Let  $G (V,E)$  be a graph whose vertices are all even. If there is more than one vertex in  $G$ , then each vertex must have degree greater than 0. Begin at any vertex  $u$ . Since the graph is connected (if the graph is not connected then the argument will be applied to separate components), there must be an edge  $u, u_1$  for some vertex  $u_1 \neq u$ . Since  $u_1$  has even degree greater than 0, there is an edge  $u_1, u_2$ . These two edges make a trail from  $u$  to  $u_2$ . Continue this trail, leaving each vertex on an edge that was not previously used, until we reach a vertex  $v$  that we have met before. (Note:  $v$  may or may not be the same vertex as  $u$ . It does not matter either way.) The edges of the trail between the two occurrences of  $v$  must form a cycle. Call the cycle formed by this process  $C_1$ . If  $C_1$  covers all the edges of  $G$ , the proof is complete. Otherwise, remove the edges forming  $C_1$  from the graph, leaving graph, say,  $G_1$ . All the vertices in  $G_1$  are still even. So pick some vertex  $u'$  in  $G_1$ . Repeat the same process as before, starting with an edge  $\{u', u'_1\}$ . By the same argument, we can generate a new cycle  $C_2$ , which has no edges in common with  $C_1$ . If  $C_2$  covers all the rest of the edges of  $G$ , then we are done. Otherwise, remove the edges forming  $C_2$  from the graph, getting graph  $G_2$ , which again contains only even vertices. We continue in this way until we have used up all the edges of  $G$ . By this time we have a number of cycles,  $C_1, C_2, \dots, C_k$  which between them contain all

the edges of  $G$  but no two of them have an edge in common. The converse of previous Theorem is also true and is obvious. The readers are encouraged to prove the same.

**Regular Graph:** A graph is said to be  $k$  – regular, if degree of every vertex  $v$  of  $G$  is  $k$ . A complete graph on  $p$  vertices is  $p-1$  regular. Fig. 1.12 below gives two examples of 3 – regular graphs.

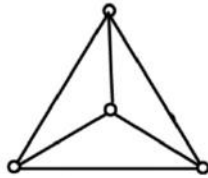


Fig:1.12(a)

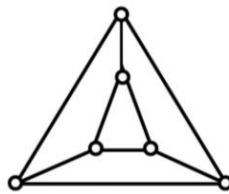


Fig:1.12(b)

**Tree:** A connected graph having no cycle is called as a tree.

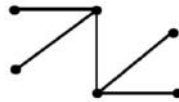


Fig:1.13(Tree)

**Petersen’s Graph :** This graph on 10 vertices and 15 edges is very famous because it tends to be a counter-example to many generalizations of ideas that work for smaller graphs. As a rule of thumb, check any conjecture on the Petersen graph before trying to prove it.

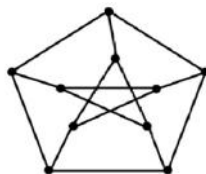


Fig:1.14(Petersen’s Graph)

## 1.5 MATHEMATICAL REPRESENTATION OF GRAPH

Even though the pictorial representation of the Graph gives very good idea of the problem under consideration, to solve the problem mathematically as well as electronically, we need to have mathematical representation of the same. The best way to represent a graph is using matrices. There are two types of matrices we shall discuss to represent graph, adjacency matrix and incidence matrix.

**Incidence Matrix:** Let  $G (V, E)$  be a graph having  $n$  vertices and  $m$  edges. The incidence matrix of  $G$  is an  $n \times m$  matrix;  $M_G := (m_{ve})$ , where  $m_{ve} = x$  where  $x$  is the number of times vertex  $v$  is incident with edge  $e$ .

The incidence matrix of Fig. 1.3 is

$$M = \begin{bmatrix} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 \\ u & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ v & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ w & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ x & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ y & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

**Adjacency Matrix:** Let  $G (V, E)$  be a graph having  $n$  vertices. The adjacency matrix of  $G$  is an  $n \times n$  matrix;  $A_G := (m_{uv})$ , where

$m_{uv} = 0$  if there is no edge from  $u$  to  $v$   
 $= 1$  if there an edge from  $u$  to  $v$   
 $= 2$  if  $u = v$  and there is a loop from  $u$  to itself.

The adjacency matrix of Fig. 1.3 is

$$A = \begin{bmatrix} & u & v & w & x & y \\ u & 0 & 1 & 1 & 0 & 1 \\ v & 1 & 0 & 1 & 0 & 1 \\ w & 1 & 1 & 0 & 1 & 1 \\ x & 0 & 0 & 1 & 0 & 1 \\ y & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Thus, we observe that in a simple graph,  $A$  is symmetric matrix and sum of the row (/column) elements is degree of the corresponding vertex.

Now let us prove some results based on the discussion above.

**Theorem:** The sum of the degrees of the vertices of a graph  $G$  is twice the number of edges. That is:  $\sum degv = 2q$ , where  $q$  is number of edges in the graph.

**Proof:** Consider incidence matrix  $M$  of graph  $G$ . Sum of entries in every row is precisely,  $deg(v)$ , and hence sum of all the entries in  $M$  is  $\sum deg$  However, sum of every column is 2 as every edge has two end vertices and there are  $q$  columns corresponding to  $q$  edges and hence sum of all the entries in  $M$  is  $2q$ . Thus, we get the required result.

**Corollary 1.5.1:** In any graph, the number of vertices of odd degree is even.

**Proof:** The proof is obvious, as if such vertices are odd in number then  $\sum degv$  will be odd, which contradicts Theorem 1.3.1. If  $G (V, E)$  is a graph having no multiple edges, then it can be represented using Adjacency list, which specifies the list of adjacent vertices to every vertex in the graph. For example, the adjacency list for the graph of Fig. 1.3 is:

$u \rightarrow \{v, w, y\}; v \rightarrow \{u, w, y\}; w \rightarrow \{u, v, x, y\}; x \rightarrow \{w, y\}$  and  $y \rightarrow \{u, v, w, x\}$

When a simple graph contains relatively few edges, that is, when it is sparse, it is usually preferable to use adjacency lists rather than an adjacency matrix to represent the graph.

Whereas if a simple graph is dense, that is, suppose that it contains many edges, say, more than half of all possible edges, then using an adjacency matrix to represent the graph is usually preferable over using adjacency lists.

Computationally speaking, adjacency matrices are more convenient than adjacency lists.

## 1.6 ISOMORPHISM

Let us have a look at the following graphs.

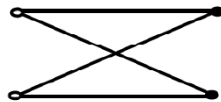


Fig:1.15(a)



Fig:1.15(b)

If the graph in Fig. 1.15(a) is made up of a string, we can change the corners appropriately to get the graph of Fig. 1.15(b). The highlighted and non-highlighted vertices will correspond in these two graphs. Thus, we can say these graphs are similar though they do not appear to be the same. This gives us an intuitive idea about isomorphism.

Now, let us define graph isomorphism formally.

**Definition:** Isomorphism: Two graphs  $G_1$  and  $G_2$  are isomorphic (written as  $G_1 \cong G_2$ ), if and only if there exists a one-to-one correspondence between their vertex sets, which preserves adjacency. We make the following observations from the definition above.

1. There exists a bijection  $f$ , from vertex set  $V_1$  of  $G_1$  to the vertex set  $V_2$  of  $G_2$ .
2. Number of vertices and edges in both the graphs are same.
3. If  $uv$  is an edge in  $G_1$  then  $f(u)f(v)$  is an edge in  $G_2$ .
4. For any vertex  $v$  in  $G_1$ , degree of  $v$  in  $G_1$  is same as degree of  $f(v)$  in  $G_2$ .

Once you see that graphs are isomorphic, it is easy to prove it. However, proving that they are not isomorphic can be sometimes very complex. It is not practically possible to check all possible correspondences. Hence, to show that two graphs are non-isomorphic, we try to find some intrinsic property that differs between the two graphs in question. In the following examples we shall check whether the given pair of graphs are isomorphic.

Example 1.6.1:

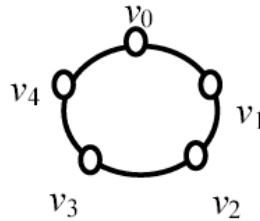


Fig:1.16(a)

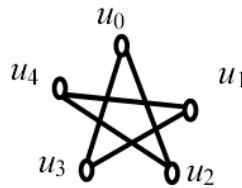


Fig:1.16(b)

In the figures above, let us have correspondence as:  $v_0 \rightarrow u_0, v_1 \rightarrow u_2, v_2 \rightarrow u_4, v_3 \rightarrow u_1$  and  $v_4 \rightarrow u_3$ . Then, this one-to-one correspondence defines isomorphism between these two graphs.

Example 1.6.2:



Fig:1.17(a)

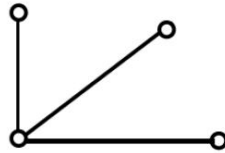


Fig:1.17(b)

In the both the graphs of Fig. 1.17, there are 4 vertices and 4 edges each. However, in Fig. 1.17(a), 2 vertices are of degree 2 and 2 are of degree 1 and in Fig. 1.17(b), there are 3 vertices of degree 1 and one is of degree 3. Thus, these two graphs are not isomorphic. Example 1.6.3:

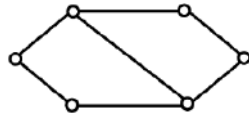


Fig:1.18(a)

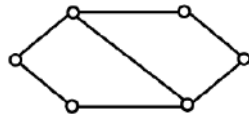


Fig:1.18(b)

In the both the graphs of Fig. 1.18, there are 6 vertices and 7 edges each. However, in Fig. 1.15(a), there is one cycle of length 3 and in Fig. 1.15(b) has no cycle of length 3.

Thus, these two graphs are not isomorphic.

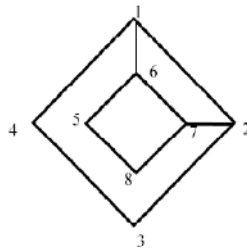


Fig:1.19(a)

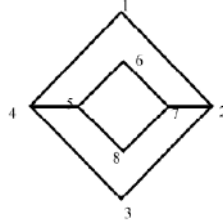


Fig:1.19(b)

In the both the graphs of Fig. 1.19, there are 8 vertices and 10 edges each. Also there are 4 vertices of degree 3 and 4 of degree 2 in each of the graphs. However, in Fig. 1.19 (a) degree of vertices 1 and 3 is 2 and both are connected to vertices 2 and 4 of degree 3 and in Fig. 1.19 (b) vertices 3 and 4 are of degree 2 and these are connected to one vertex of degree 2 and one vertex of degree 3. Hence, we cannot find one-to-one correspondence between these two graphs and thus they are not isomorphic.

**Definition 1.6.2: Automorphism:** An automorphism of a graph is an isomorphism of a graph to itself.

In case of a simple graph, automorphism is just a permutation  $\alpha$  of its vertex set which preserves adjacency. The automorphisms of a graph reflect its symmetries. For example, if  $u$  and  $v$  are two vertices of a simple graph, and if there is an automorphism  $\alpha$  which maps  $u$  to  $v$ , then  $u$  and  $v$  are alike in the graph, and are referred to as similar vertices. Graphs in which all vertices are similar, such as the complete graph  $K_n$ , the complete bipartite graph  $K_{n,n}$  are called vertex-transitive.

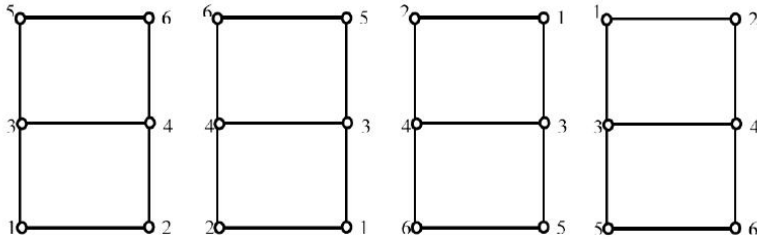


Fig:1.20

The grid graphs of Fig. 1.20 have four automorphisms,  $(1, 2, 3, 4, 5, 6)$ ,  $(2, 1, 4, 3, 6, 5)$ ,  $(5, 6, 3, 4, 1, 2)$ , and  $(6, 5, 4, 3, 2, 1)$ . These correspond to the graph itself, the graph flipped left-to-right, the graph flipped up-down, and the graph flipped left-to-right and up-down, respectively, illustrated above.



## 2 CONNECTIVITY

### 2.1 Introduction

In the first chapter, we have defined connected graphs and component. In this chapter, we shall discuss graph connectivity in more details as it is of great importance in the practical applications. In the computer network, it will be crucial to know whether data can be transferred if one of the nodes or links fails. Some connected graphs can be disconnected by removing some vertices or edges. In this chapter, we shall understand the concept of connectivity further.

### 2.2 CUT VERTICES, BRIDGES AND BLOCKS

To understand the importance of connectivity intuitively, let us have a look at the following graphs of Fig2.1. All the graphs are on 5 vertices.

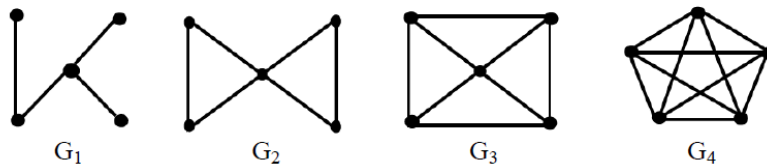


Fig:2.1

$G_1$  is a minimal connected graph; deleting any edge disconnects it.  $G_2$  cannot be disconnected by the deletion of a single edge, but can be disconnected by the deletion of one vertex, its cut vertex. There are no cut edges or cut vertices in  $G_3$ , but even so  $G_3$  is clearly not as well connected as  $G_4$ , the complete graph on five vertices. Thus, intuitively, each successive graph is better connected than the previous one. We now introduce two parameters of a graph, its connectivity and edge connectivity, which measure the extent to which it is connected. For any graph  $G$ ,

- $\delta$ : Minimum degree of the graph.
- $\Delta$ : Maximum degree of the graph.
- $k$ : Connectivity of the graph that is minimum number of vertices that are to be removed to make the graph disconnected or a trivial.
- $k'$ : Edge connectivity of the graph that is minimum number of edges that are to be removed to make the graph disconnected or a trivial.

Thus, a connected graph is termed as  $k$ -connected, if we need to remove  $k$  vertices to disconnect the graph  $G$ .

In the graph  $G_1$  of Fig. 2.1,  $\delta = k = k' = 1$ .

In the graph  $G_2$  of Fig. 2.1,  $\delta = 2$ ,  $k = 1$ ,  $k' = 2$ .

In the graph  $G_3$  of Fig. 2.1,  $\delta = 3$ ,  $k = 0$ ,  $k' = 3$ .

Students are encouraged to find these parameters for the graph  $G_4$  of Fig. 2.1.

Before we proceed to prove some interesting results, let us define certain terms related to connectivity.

**Definition 2.1.1: Cut vertex (Cut point):** Let  $G (V, E)$  be a connected graph. Vertex  $v \in V$ , is called as cut vertex, if on removing  $v$  along with all its incident edges from the graph, resulting graph is disconnected.

**Definition 2.1.2: Bridge:** Let  $G (V, E)$  be a connected graph. An  $e \in E$  is called as bridge, if on removing  $e$  from  $G$ , resulting graph is disconnected.

**Definition 2.1.3: Non-separable Graph:** A connected, non-trivial graph having no cut vertices is called as non-separable graph.

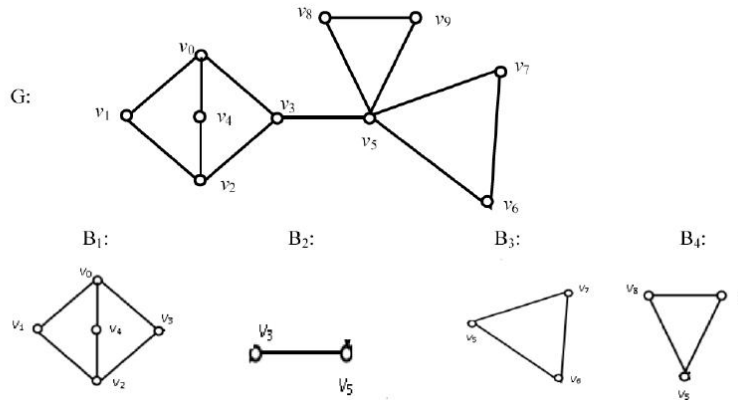


Fig:2.2

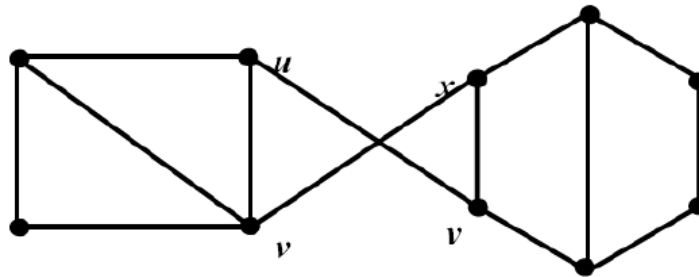


Fig:2.3

The graph of Fig. 2.3 is 2-connected (i.e.  $k = 2$ , e.g., vertices  $u, v$ ) and it has  $k' = 2$  (i.e. on removing edges  $uy$  and  $vx$ , graph is disconnected).

**Definition 2.1.4: Separation:** A separation of  $G$  of order  $k$  is a pair of subgraphs  $(H, K)$  with  $H \cup K = G$  and  $E(H \cap K) = \emptyset$  and  $V(H) \cap V(K) = k$ . Such a separation is proper if  $V(H) \setminus V(K)$  and  $V(K) \setminus V(H)$  are nonempty.

E.g. Separation of graph in Fig. 2.3 is:

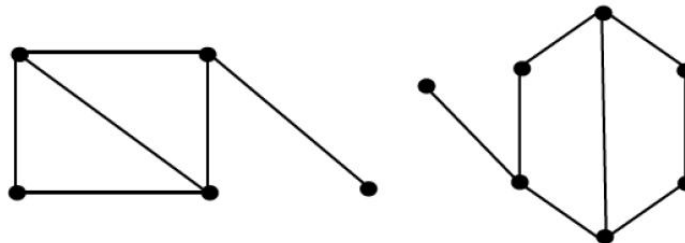


Fig:2.4

Thus, from the definition of a separation, we observe that, a connected graph has a cut vertex if and only if it has 1 – separation. (i.e.  $|V(H) \cap V(K)|=1$ )

**Definition 2.1.4: Block:** A block of a graph is a maximal nonseparable subgraph. If  $G$  is non-separable, then  $G$  itself is a block.

In the connected graph  $G$  having vertex set  $\{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$  of Fig.2.2, edge  $v_3v_5$  is a bridge. The subgraphs  $B_1, B_2, B_3$ , and  $B_4$  are blocks of the given graph.

We can observe that end vertices of a bridge are cut-vertices and an edge is a bridge if and only if it is not on any cycle.

From the above discussion we can easily prove following theorems.

**Theorem 2.1.1:** Let  $v$  be a vertex of a connected graph  $G (V, E)$ . The following statements are equivalent.

- i.  $v$  is a cut vertex of  $G$ .
- ii. There exist vertices  $u$  and  $w$ , distinct from  $v$  such that  $v$  is on every  $u$ - $w$  path.
- iii. There exists a partition of the set of vertices  $V-\{v\}$  into subsets  $U$  and  $W$  such that for any vertex  $u \in U$  and  $w \in W$ , the point  $v$  is on every  $u$ - $w$  path.

**Theorem 2.1.2:** Let  $x$  be an edge of a connected graph  $G (V, E)$ . The following statements are equivalent.

- i.  $x$  is a bridge of  $G$ .
- ii.  $x$  is not on any cycle of  $G$ .
- iii. There exist vertices  $u, v$  of  $G$  such that  $x$  is on every path joining  $u$  and  $v$ .
- iv. There exists a partition of  $V$  into subsets  $U$  and  $W$  such that for any point  $u \in U$  and  $w \in W$ , the edge  $x$  is on every path joining  $u$  and  $w$ .

Remarks:

- If a block  $B$  has at least three vertices, then  $B$  is 2-connected.
- If an edge is a block of  $G$ , then it is a cut-edge of  $G$ .

**Theorem 2.1.3:** Two blocks in a graph share at most one vertex.

**Proof:** Let, if possible,  $B_1$  and  $B_2$  are two blocks of  $G$ , sharing two or more vertices. Then the deletion of any one of the vertices will not disconnect  $B_1$  or  $B_2$ . Thus,  $B_1 \cap B_2$  is a subgraph of  $G$  having no cut vertex and a block of  $G$ , which contradicts maximality of  $B_1$  and  $B_2$ .

- Blocks of  $G$  partition its edge set.
- If two blocks share a vertex, then it must be a cut-vertex of  $G$ .

**Definition 2.1.5: Block Graph:** Block graph of a graph  $G$  is a bipartite graph  $H$  in which one partition set consists of cut vertices of  $G$  and the other has a vertex  $b_i$  for every block  $B_i$  of  $G$ . We include  $(v, b_i)$  as an edge if cut vertex  $v$  is in block  $B_i$ .

Let us illustrate the definition with the help of following graph.

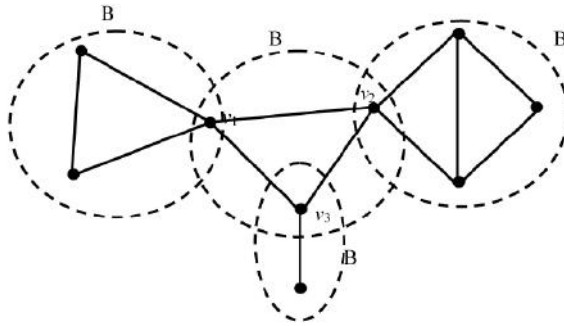


Fig:2.4

The graph of above Fig.2.4 has four blocks  $B_1, B_2, B_3, B_4$  and three cut vertices  $v_1, v_2, v_3$ . Hence, Block graph of the same is as below:

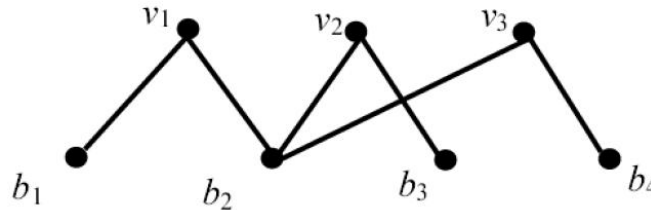


Fig:2.5

**Theorem 2.1.4:** The block graph of a connected graph is a tree.

**Proof:** We have seen that a block graph of a connected graph is a bipartite graph. Let  $B(G)$  be a block graph of a connected graph  $G$ . By adding edges in a block of  $G$ , we do not change  $B(G)$ , so let us assume that blocks are complete graphs. Since  $G$  is connected,  $B(G)$  is also connected. Now, we shall show that  $B(G)$  has no cycle.

Let if possible, it has a cycle  $C$  such that its alternate vertices correspond to cut vertices and blocks of  $G$ . It can be shown in the diagram of Fig. 2.6 below. White points represent vertices with respect to blocks and black points represent vertices with respect to cut vertices.

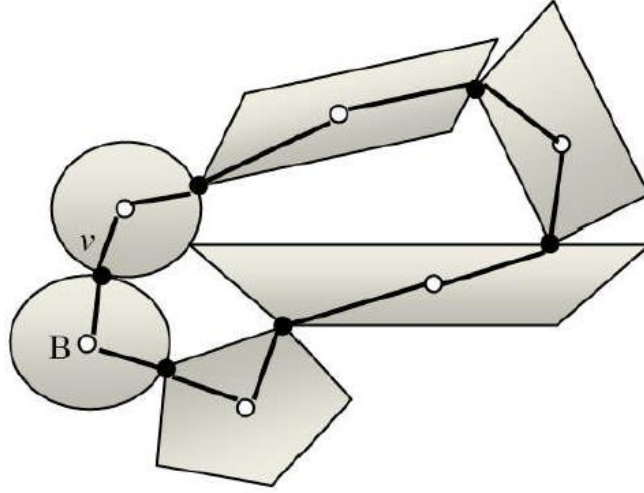


Fig:2.6

Let cut vertices of  $G$  in order along  $C$  be  $v_0, v_1, v_2, \dots, v_k, v_0$ , then  $v_0 v_1 v_2 \dots v_k v_0$  is a cycle  $C \subseteq G$ . If  $B \in C$ , then  $B \cup C$  is a subgraph of  $G$  consisting of complete graph  $B$  together with a cycle  $C$  in which one edge is in  $B$  and atleast one edge not in  $B$ . Therefore,  $B \cup C$  has no cut vertex, which contradicts maximality of  $B$ .

Now, we shall prove the relationship among  $k$ ,  $k'$  and  $\delta$ .

**Theorem 2.1.5:** For any graph  $G$ ,  $k \leq k' \leq \delta$ .

**Proof:** We shall prove second part of the inequality first. If  $G$  has not edges, then  $k' = 0$ . Otherwise, if every edge incident to a vertex of minimum degree is removed, the resulting graph is disconnected. In both these extreme cases,  $k' \leq \delta$ .

Now we shall prove  $k \leq k'$ . Here we need to consider many cases. If  $G$  is disconnected or trivial,  $k = k' = 0$ . If  $G$  is connected and has a bridge at edge  $e$ , then  $k = 1$ , because either  $G$  has a cut-vertex which is an end vertex of  $e$  or  $G$  is  $K_2$ . Finally, let  $G$  has edge connectivity  $k'$ , such that  $k' \geq 2$ . Then, removal of  $k' - 1$  of these edges will produce a bridge, say,  $e = uv$ . For each of these  $k' - 1$  edges, select an incident vertex different from  $u$  or  $v$ . Removal of these  $k' - 1$  vertices also removes these  $k' - 1$  edges and possibly more. If the resulting graph is disconnected then  $k \leq k'$ . If not,  $e = uv$  is still a bridge in this resulting graph and removal of  $u$  or  $v$  will result into either trivial or disconnected graph, giving  $k \leq k'$ .

Following graph in Fig. 2.3 has  $k = 2$ ,  $k' = 3$  and  $\delta = 4$ .

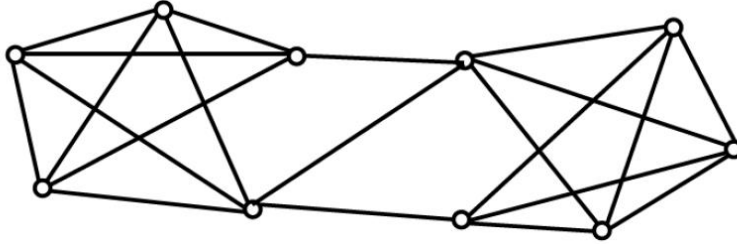


Fig:2.6

### 2.3 MENGER'S THEOREMS

Menger's theorem gives characterisation of connectivity of finite, connected graphs. If  $u$  and  $v$  are any two vertices in a connected graph, then two paths connecting  $u$  and  $v$  are said to be disjoint or vertex disjoint, if they have no vertex (and hence no edge) in common. The paths are said to be edge disjoint, if there is no edge common.

Menger discusses minimum number of disjoint paths between any pair of vertices. Menger proved the results for vertex-connectivity as well as edge-connectivity.

Before we discuss Menger's theorem, let us have a look at the graph of Fig. 2.7 below:

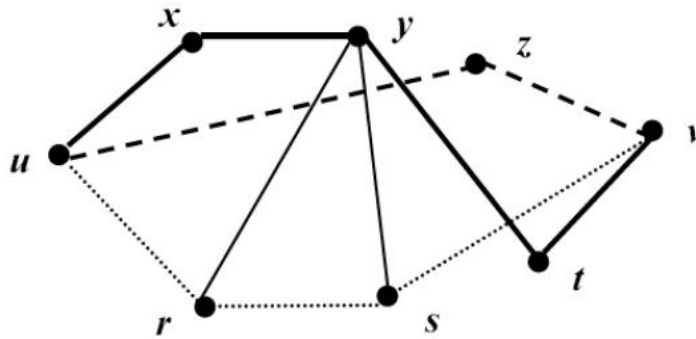


Fig:2.7

In the Fig. 2.7, we observe that there are three vertices disjoint from  $u$  to  $v$ , viz.  $u-x-y-t-v$ ,  $u-z-v$  and  $u-r-s-v$ .

Vertex version of Menger's theorem discusses about the vertex disjoint path.

**Theorem 2.2.1:(Menger's Theorem – Vertex Form):** The minimum number of vertices separating two non-adjacent vertices  $s$  and  $t$  is the maximum number of number of disjoint paths connecting  $s$  and  $t$ .

**Proof:** If  $k$  vertices separate  $s$  and  $t$ , then obviously there can be no more than  $k$  disjoint paths. We shall show that these are exactly  $k$ . That is if there are  $k$  vertices separate  $s$  and  $t$  then exactly  $k$  internally disjoint paths separate  $s$  and  $t$ . This is true if  $k = 1$ . So let  $k \geq 1$  and if possible the result is wrong. That is it takes less than  $k$  disjoint paths to disconnect  $s$  and  $t$ . Let  $h$  be the smallest such  $k$ , and let  $F$  be a graph with the minimum number of vertices for which the theorem fails for  $h$ . We remove edges from  $F$  until we obtain a graph  $G$  such that  $h$  vertices are required to separate  $s$  and  $t$  in  $G$  but for any edge  $x$  of  $G$ , only  $h - 1$  vertices are required to separate  $s$  and  $t$  in  $G - x$ . We first investigate the properties of this graph  $G$ , and then complete the proof of the theorem.

By the definition of  $G$ , for any edge  $x$  of  $G$ , there exist a set  $S(x)$  of  $h - 1$  vertices which separates  $s$  and  $t$  in  $G - x$ . Now  $G - S(x)$  contains at least one  $s$ - $t$  path, since it takes  $h$  points to separate  $s$  and  $t$  in  $G$ . Each such  $s$ - $t$  path must contain the edge  $x = uv$ , since it is not a path in  $G - x$ . So  $u, v \notin S(x)$  and if  $u \neq s, t$ , then  $S(x) \cup \{u\}$  separates  $s$  and  $t$  in  $G$ . If there is a point  $w$  adjacent to both  $s$  and  $t$  in  $G$ , then  $G - w$  requires  $(h - 1)$  points to separate  $s$  and  $t$  and so it has  $h - 1$  disjoint  $s$ - $t$  paths. Replacing  $w$ , we have  $h$  disjoint  $s$ - $t$  paths in  $G$ . So we have shown:

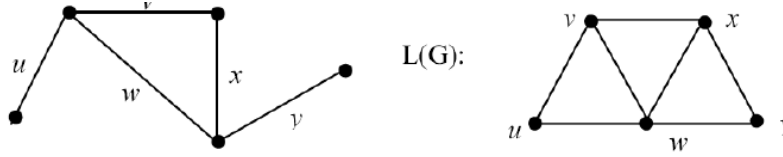
(I) No point is adjacent to both  $s$  and  $t$  in  $G$ .

Let  $W$  be any collection of  $h$  points separating  $s$  and  $t$  in  $G$ . An  $s$ - $W$  path is a path joining  $s$  with some  $w_i \in W$  and containing no other point of  $W$ , call the collections of all  $s$ - $W$  paths and  $W$ - $t$  paths  $P_s$  and  $P_t$ , respectively. Then each  $s$ - $t$  path begins with a member of  $P_s$  and ends with a member of  $P_t$  because every such path contains a point of  $W$ . Moreover, the paths in  $P_s$  and  $P_t$ , have the points of  $W$  and no others in common, since it is clear that each  $w_i$  is in at least one path in each collection and, if some other point were in both an  $s$ - $W$  and a  $W$ - $t$  path, then there would be an  $s$ - $t$  path containing no point of  $W$ . Finally, either  $P_s - W = s$  or  $P_t - W = t$ , since, if not, then both  $P_s$  plus the lines  $w_1t, w_2t, \dots$  and  $P_t$  plus the lines  $sw_1, sw_2, \dots$  are graphs with fewer points than  $G$  in which  $s$  and  $t$  are nonadjacent and  $h$ -connected, and therefore in each there are  $h$  disjoint  $s$ - $t$  paths. Combining the  $s$ - $W$  and  $W$ - $t$  portions of these paths, we can construct  $h$  disjoint  $s$ - $t$  paths in  $G$ , and thus have a contradiction. Therefore we have proved:

(II) Any collection  $W$  of  $h$  points separating  $s$  and  $t$  is adjacent either to  $s$  or to  $t$ . Now we can complete the proof of the theorem. Let  $P = \{s, u_1, u_2, \dots, t\}$  be a shortest  $s$ - $t$  path in  $G$  and let  $u_1u_2 = x$ . Note that by (I),  $u_2 \neq t$ . Form  $S(x) = \{v_1, v_2, \dots, v_{k-1}\}$  as above, separating  $s$  and  $t$  in  $G - x$ . By (I),  $u_1t \notin G$  and hence by II with  $W = S(x) \cup u_1$ ,  $sv_i \in G$ , for all  $i$ . Thus, by (I),  $v_it \notin G$ , for every  $i$ . However, if we pick  $W = S(x) \cup u_2$  instead, we have by (II) that  $su_2 \in G$ , contradicting our choice of  $P$  as a shortest  $s$ - $t$  path, and completing the proof of the theorem.

**Definition 2.2.1: Line Graph:** Let  $G$  be any graph. Line graph of  $G$ , denoted by  $L(G)$  is such that each vertex in  $L(G)$  represents an edge in  $G$  and if two edges are adjacent in  $G$  then there is an edge between two vertices of  $L(G)$  corresponding to these two edges.

Fig. 2.8 below gives an example of a graph and its line graph.



**Fig. 2.8**

## 2.4 CONSTRUCTION OF RELIABLE COMMUNICATION NETWORK

As we have seen in the first chapter, one of the applications of graph theory is to represent graph as a communication network. While constructing this network, it is necessary to make it sure that it does not get disconnected too often. Higher the connectivity and the edge connectivity of the network, the more reliable the network is. Minimum cost spanning tree constructed using Kruskal's algorithm (discussed in chapter 3), has connectivity 1 and hence it is not much reliable. Therefore, we try to generalise the problem.

Let  $k$  be given integer and let  $G$  be a weighted graph. Our aim is to find minimum weight  $k$ -connected spanning subgraph. If  $k = 1$ , the problem can be solved using Kruskal's method. For  $k > 1$ , the problem is difficult and no solution is known. However, if  $G$  is a complete graph having unit weight, then it can be solved by then method given below.

If  $G$  is a complete graph on  $n$  vertices having unit weight, then the problem is simply to find a minimum  $m$ -connected ( $m < n$ ) spanning subgraph, with as few edges as possible. We shall denote by  $f(m, n)$ , the least number of edges an  $m$ -connected graph on  $n$  vertices can have and denote such a graph by  $H_{m,n}$ . Structure of  $H_{m,n}$  depends on parities of  $m$  and  $n$ , and we have three cases as below.

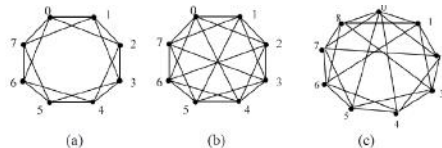
Case 1: Let  $m$  be even,  $m=2r$ . Number the vertices of graph on  $n$  vertices from 0 to  $n - 1$ .

Connect vertices  $i$  and  $j$  if  $j - i = s \pmod n$  is such that  $0 \leq s \leq r$ .

Case 2: Let  $m$  be odd,  $m = 2r + 1$  and  $n$  even. First draw  $H_{m, n}$  and then add edges from  $i$  to  $i + (n/2)$ .

Case 3: Let  $m$  be odd,  $m = 2r + 1$  and  $n$  is also odd. Then  $H_{2r+1, n}$  is constructed first by drawing  $H_{2r,n}$  and then by adding edges joining vertex 0 to vertices  $\frac{n-1}{2}$  and  $\frac{n+1}{2}$  vertex  $i$  to vertex  $i + \frac{n-1}{2}$  for  $1 \leq i \leq \frac{n-1}{2}$

These three cases are shown in the Fig. 2.9(a), (b) and (c) below.



**Fig:2.9**



## 2.5 DIJKSTRA'S ALGORITHM

If  $G(V, E)$  is any connected graph and  $u, v$  are any two arbitrary vertices in  $V$ . Then, there may exist multiple paths from  $u$  to  $v$ . One of the very important and practical applications of Graph theory is to find the shortest path from one vertex (node) to the other. The term "shortest" may refer to minimum distance, least cost or least time.

One of the most important algorithms which is of prime importance in Graph Theory is credited to a computer scientist Dijkstra. Main reason for the popularity of Dijkstra's algorithm is that it provides exact optimal solution to a large class of shortest path problems, and it is important theoretically, practically as well as educationally.

Before we proceed to discuss the example and algorithm, let us have a look at its salient features.

- Algorithm gives solution to single source shortest path problems.
- It works on both directed and undirected graphs.
- All edges must have non-negative weights.
- Graph must be connected.

Before we proceed to the algorithm, let us understand the same with the help of an example

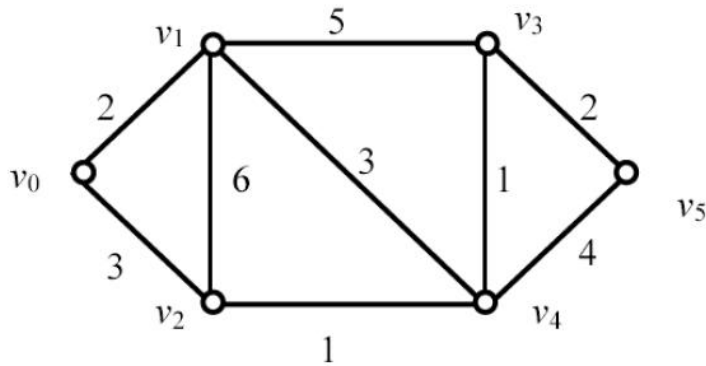


Fig:2.10

Let us say, we have to find minimum distance from  $v_0$  to every other vertex and the shortest path from  $v_0$  to every other vertex.

We shall start with vertex  $v_0$  and subsequently visit every vertex and once all the vertices are visited, we will terminate the procedure.

Let us introduce some parameters for the same.

$L(i)$  : Shortest distance between  $v_0$  and vertex  $i$ .  $V$  : Set of visited vertices  
 $U$  : Set of vertices not visited so far  $w(i, j)$ : Weight of edge connecting vertices  $i$  and  $j$   $P(i)$  : Shortest path from  $v_0$  to  $i$ .

To begin with, we shall have  $L(v_0) = 0$  and  $L(v_1) = L(v_2) = L(v_3) = L(v_4) = L(v_5) = \infty$ .  $V = \{v_0\}$ ,  $U = \{v_1, v_2, v_3, v_4, v_5\}$ ,  $P(v_1) = P(v_2) = P(v_3) = P(v_4) = P(v_5) = \emptyset$ .

Initially, let  $v_0$  be only visited vertex and hence  $V = \{v_0\}$ . We check all the edges having one end point in  $V$  and other in  $U$  for their distances and update  $V$  by picking up  $v_k$  for which  $L(v_k)$  is minimum. We keep on updating paths from  $v_0$  to  $v_k$  every time. We shall continue the process till all the vertices of the graph are visited.

**Step I:**

$V = \{v_0\}$ ,  $P(v_0) = \{v_0\}$ . Now observe all the vertices adjacent to  $v_0$  and update these parameters as below:

$$L(v_1) = \min \{L(v_1), L(v_0) + w(v_0, v_1)\} = \min \{\infty, 0 + 2\} = 2, P(v_1) = \{v_0, v_1\}$$

$$L(v_2) = \min \{L(v_2), L(v_0) + w(v_0, v_2)\} = \min \{\infty, 0 + 3\} = 3, P(v_2) = \{v_0, v_2\}$$

As  $L(v_1)$  is minimum,  $V = \{v_0, v_1\}$ .

**Step II:**

$$L(v_2) = \min \{L(v_2), L(v_1) + w(v_1, v_2)\} = \min \{3, 2 + 6\} = 3, P(v_2) = \{v_0, v_1, v_2\}$$

$$L(v_3) = \min \{L(v_3), L(v_1) + w(v_1, v_3)\} = \min \{\infty, 2 + 5\} = 7, P(v_3) = \{v_0, v_1, v_3\}$$

$$L(v_4) = \min \{L(v_4), L(v_1) + w(v_1, v_4)\} = \min \{\infty, 2 + 3\} = 5, P(v_4) = \{v_0, v_1, v_4\}$$

As  $L(v_2)$  is minimum,  $V = \{v_0, v_1, v_2\}$

**Step III:**

$$L(v_3) = \min \{L(v_3), L(v_2) + w(v_2, v_3)\} = \min \{7, 3 + \infty\} = 7, P(v_3) = \{v_0, v_1, v_2, v_3\}$$

$$L(v_4) = \min \{L(v_4), L(v_2) + w(v_2, v_4)\} = \min \{5, 3 + 1\} = 4, P(v_4) = \{v_0, v_2, v_4\}$$

As  $L(v_4)$  is minimum,  $V = \{v_0, v_1, v_2, v_4\}$

**Step IV:**

$$L(v_3) = \min \{L(v_3), L(v_4) + w(v_4, v_3)\} = \min \{7, 4 + 1\} = 5, P(v_3) = \{v_0, v_2, v_4, v_3\}$$

$$L(v_5) = \min \{L(v_5), L(v_4) + w(v_4, v_5)\} = \min \{\infty, 4 + 4\} = 8, P(v_5) = \{v_0, v_2, v_4, v_5\}$$

As  $L(v_3)$  is minimum,  $V = \{v_0, v_1, v_2, v_4, v_3\}$

**Step V:**

$$L(v_5) = \min \{L(v_5), L(v_3) + w(v_3, v_5)\} = \min \{8, 5 + 2\} = 7, P(v_5) = \{v_0, v_2, v_4, v_3, v_5\}$$

Thus,  $V = \{v_0, v_1, v_2, v_3, v_4, v_5\}$ .

All the vertices are visited and we have found the shortest path and distance from  $v_0$  to every other vertex.

In fact, what we have presented above is Dijkstra's shortest path method.

Now, let us present the steps of the algorithm

**Dijkstra's Shortest Path Algorithm:**

Let  $G(V, E)$  be any connected, weighted graph, with  $G_V = \{v_0, v_1, v_2, \dots, v_n\}$ .

**Parameters:**

$L(v_i)$  : Shortest distance between source  $s = v_0$  and vertex  $v_i$ .

$P(v_i)$  : Set of vertices giving shortest path from  $s$  to  $v_i$ .

$V$  : Set of visited vertices

$U$  : Set of vertices not visited so far

$w(v_i, v_j)$  : weight of an edge from  $v_i$  to  $v_j$ .

$z$  : Destination vertex

**Initial Values:**

$V = \phi$ ,  $U = GV$ ,  $L(s = v_0) = 0$ ,  $P(s = v_0) = \{v_0\}$

$L(v_i) = \infty$ ; for  $1 \leq i \leq n$   $P(v_i) = \phi$ ; for  $1 \leq i \leq n$

**Procedure:**

While  $z \in U$

  Begin

$u$  : vertex in  $U$  with  $L(u)$  minimum

$V := V \cup u$

    For every  $x \in U$

      Begin

        If ( $L(x) > L(u) + w(u, x)$ ) then

$L(x) := L(u) + w(u, x)$

$P(x) := P(u) \cup x$

        End If

    End

  End

### 3 TREES

#### 3.1 Introduction

In the chapter 2, we have defined trees and also its relationship with block graphs. In this chapter, we shall explore trees in details and also discuss a few practical applications of the same.

First, we will be enumerating all the trees on  $n$  vertices. Before that, let us draw trees on 7 vertices. A Few of such trees are shown in Fig. 3.1.

Following figure gives trees on 7 vertices.

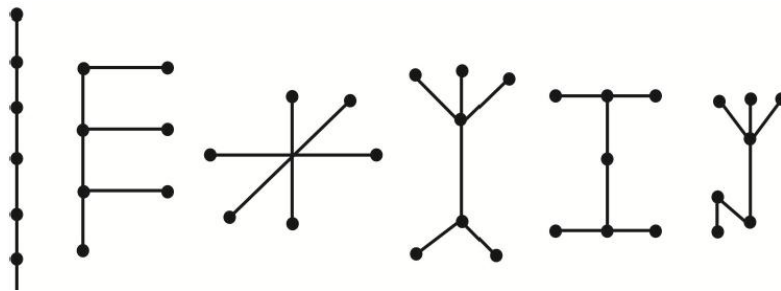


Fig:3.1

Let us know a few more terms related to tree. As tree is an acyclic graph, acyclic graphs are called as Forest.

We have seen a spanning subgraph of a graph  $G$  in the chapter 1. If a spanning subgraph of a connected graph is a tree, it is called as spanning tree. Spanning trees are of utmost importance in the applications of graph theory.

Fig. 3.2(b) below is a spanning tree of the graph of Fig.3.2(a).

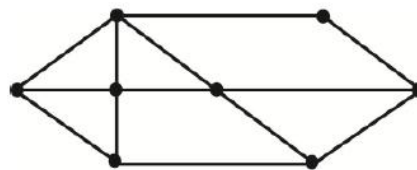


Fig:3.2(a)

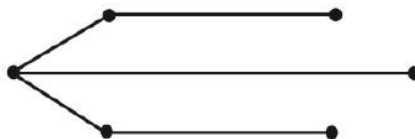


Fig:3.2(b)

We also observe the following facts about trees from the definition of tree and the discussion so far.

1. In a tree, any two vertices are connected by exactly one path. (For more than one path would result into a cycle.)

2. From Theorem 1.4.1, we know that if degree of every vertex is at least two, then the graph contains a cycle. And hence every tree has at least one vertex of degree at most one. In fact, if the tree is nontrivial then it has a vertex of degree exactly one. Vertex of degree one in a tree is termed as leaf.

3. Result: For a tree  $p = q + 1$ , or number of vertices is exactly one more than the number of edges.

**Proof:** Let  $T$  be a tree having  $p$  vertices and  $q$  edges. If  $T$  is trivial, then  $p = 1$  and  $q = 0$ . Hence, the result is true. If  $T$  is  $K_2$ , then  $p = 2$  and  $q = 1$ . Again the result holds true. Let  $T$  be non-trivial and not  $K_2$ . We shall prove the result by induction on number of vertices. Let the result be true for a tree having less than  $p$  vertices. Now,  $T$  has  $p$  vertices and it is non-trivial. Hence, it has at least one leaf, say at vertex  $v$ . Then,  $T - v$  is again a tree having  $p - 1$  ( $< p$ ) vertices and  $q - 1$  edges. By induction hypothesis,  $p - 1 = (q - 1) + 1$ . Simplifying we get  $p = q + 1$ , as required.

4. Every non-trivial graph has at least two leaves. This follows easily from the theorem above, as  $\sum d_i = 2q = 2(p - 1)$ .

5. Let  $e = uv$  be any edge in tree  $T$ . Then,  $e$  has to be its cut edge, for if not, after removing  $e$  from  $T$ ,  $T$  will be connected and hence, we will get a  $uv$  path in  $T - e$ , a contradiction to property 1 above.

6. Every non-leaf vertex of a tree is its cut vertex. For, let  $u$  be a vertex of a tree having degree at least 2 and  $v$  and  $w$  are its adjacent vertices. Then,  $uv$  and  $uw$  are the only paths from  $u$  to  $v$  and  $u$  to  $w$  respectively. Hence, on removing  $u$  from the tree, it becomes disconnected.

7. Every tree is bipartite. To see that, choose any vertex  $v$  from tree. Now divide vertices of the tree in two sets  $A$  and  $B$  such that,  $u \in A$ , if  $d(v, u)$  is even and  $u \in B$ , if  $d(v, u)$  is odd. By this choice,  $v \in A$ . Thus,  $A \cup B = V$  (vertex set of the tree) and  $A \cap B = \phi$ .

8. A graph is connected if and only if it has a spanning tree. A spanning tree is a connected graph. From a connected graph, delete one edge at a time to remove cycles and we get a spanning tree

## 3.2 CHARACTERISATION OF TREES

**Theorem 3.2.1:** Let  $G(V, E)$  be a graph having  $p$  vertices and  $q$  edges. The following statements are equivalent for  $G$ .

- i.  $G$  is a tree.
- ii. Every two points of  $G$  are joined by a unique path.
- iii.  $G$  is connected and  $p = q + 1$ .
- iv.  $G$  is acyclic and  $p = q + 1$ .
- v.  $G$  is acyclic and any two nonadjacent vertices of  $G$  are joined by an edge  $e$ , then  $G + e$  has exactly one cycle.
- vi.  $G$  is connected and every edge of  $G$  is in cut edge.

This characterisation of tree can be proved from the discussion so far.

### 3.3 EDGE CUTS AND BONDS

First, we shall start with edge cuts.

Let  $G(V, E)$  be a graph and  $X, Y$  by subsets of  $V$ , not necessarily distinct. We denote  $E[X, Y]$  to be the set of all edges of  $G$  having one end in  $X$  and the other in  $Y$  and by  $e(X, Y)$  their number. If  $X = Y$ , we simply write as  $E(X)$  and  $e(X)$  for  $E[X, X]$  and  $e(X, X)$ . If  $Y = V - X$ , the set  $E[X, Y]$  is called as the edge cut associated with  $X$  and is denoted by  $\delta(X)$ . Obviously,  $\delta(X) = \delta(V - X)$  and  $\delta(V) = \phi$ .

Let us illustrate edge cuts with an example.

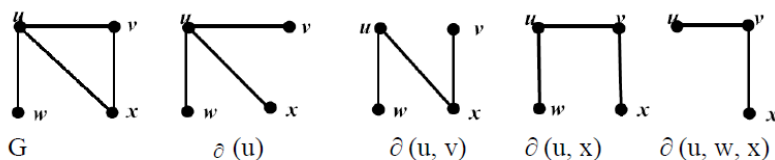


Fig:3.3

The minimal edge cut of a graph is called as bond, thus bond is an edge cut such that none of its edge-subset is an edge cut. To illustrate, let us have a look at the following figure.

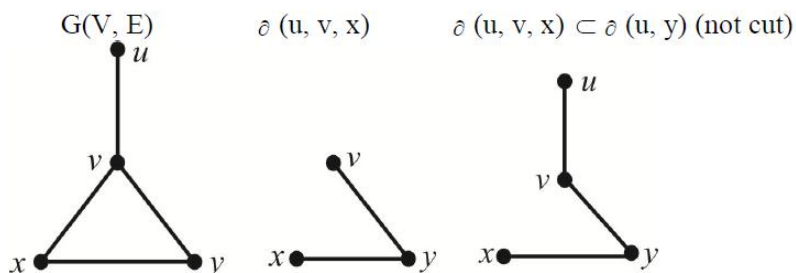


Fig:3.4

In the Fig. 3.4,  $\delta(u, v, x)$  is a bond whereas  $\delta(u, y)$  is not as it has a subset which is an edge cut.

Now we shall discuss two important results associated with edge cuts.

### 3.4 GRAPHS AND VECTOR SPACE

Before, we define a vector space on graph, let us first define a binary operation - symmetric difference (denoted by  $\Delta$ ), on graphs. Let  $G(V, E)$  be a graph, having  $p$  vertices and  $q$  edges.

Let  $E = \{e_1, e_2, \dots, e_q\}$ . Let  $E_1$  and  $E_2$  be two subsets of  $E$ . We define,  $E_1 \Delta E_2$  as:

$E_1 \Delta E_2 = E_1 \cup E_2 - E_1 \cap E_2$ . Thus, for a graph in Fig. 3.5 below, we shall illustrate  $\Delta$ .

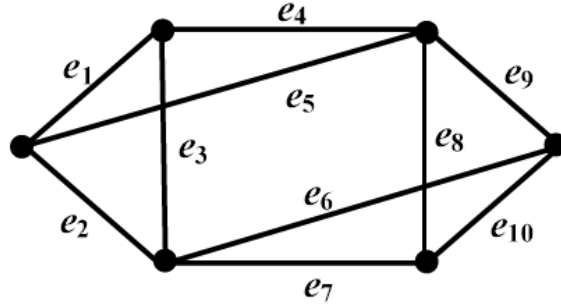


Fig:3.5

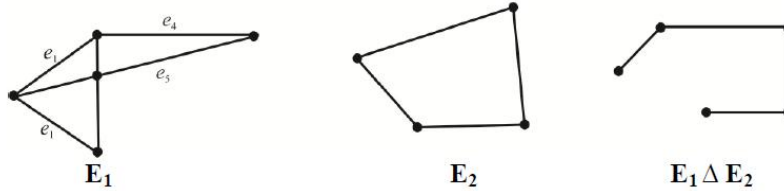


Fig:3.6

Fig. 3.5 shows a graph on 8 vertices and 10 edges.  $E_1$  and  $E_2$  are two subsets of edge set  $E$  of the graph. Then the Fig. 3.6 shows  $E_1 \Delta E_2$

$$E_1 = \{e_1, e_2, e_4, e_5\}, E_2 = \{e_2, e_5, e_7, e_8\}, \text{ then } E_1 \Delta E_2 = \{e_1, e_4, e_7, e_8\}$$

**Notations :** If  $X$  and  $Y$  are two subsets of edge set  $E$  of a graph  $G(V, E)$ , having  $p$  vertices and  $q$  edges, we associate vectors  $X = (x_1, x_2, \dots, x_q)$  and  $Y = (y_1, y_2, \dots, y_q)$ , such that,  $x_i = 1, y_j = 1$ , if edge  $x_i \in X$ , edge  $y_j \in Y$ , else  $x_i = 0, y_j = 0$ . We define operation  $\Delta$  on the elements of  $X, Y$  with a rule:  $1 \Delta 1 = 0, 0 \Delta 0 = 0, 1 \Delta 0 = 1$  and  $0 \Delta 1 = 1$ . Thus, for the edge sets  $E_1$  and  $E_2$  in the example above, we have, vector  $E_1 = (1, 1, 0, 1, 1, 0, 0, 0, 0, 0)$  and vector  $E_2 = (0, 1, 0, 0, 1, 0, 1, 1, 0, 0)$  and hence vector  $E_1 \Delta E_2 = (1, 0, 0, 1, 0, 0, 1, 1, 0, 0)$ . Thus,  $E_1 \Delta E_2 = \{e_1, e_4, e_7, e_8\}$ , as we have obtained earlier.

Thus, the  $q$ -vector representing the symmetric difference of  $q$ -vectors  $E_1$  and  $E_2$  is in fact the  $q$ -vector of symmetric difference of  $E_1$  and  $E_2$ . But,  $E_1$  and  $E_2$  are two subgraphs of  $G$ . Hence, we have defined binary operation on the subgraphs in the form of a vector. The set of all  $2^q$   $q$ -vectors, (all zeros indicate null graph), is a set of all edge induced subgraphs of  $G$ . We denote it by  $\epsilon(G)$ . This set forms a vector space on the field  $GF(2)$  or  $Z_2$ . This can be easily verified.

1.  $\epsilon(G)$  is an abelian group under  $\Delta$ .

a. Let  $X, Y$  be any two vectors (edge sets) in  $\epsilon(G)$ . Then,  $X \Delta Y$  is also a vector in  $\epsilon(G)$ , and hence  $\epsilon(G)$  is closed under  $\Delta$ .

- b. Vector associated with  $\phi$  is  $(0, 0, \dots, 0)$  and  $X \Delta (0, 0, \dots, 0) = X = (0, 0, \dots, 0)$ . Thus, identity exists.
  - c. If  $X$  is any edge vector, then,  $X \Delta X = 0$  (zero vector). This shows the existence of inverse with respect to  $\Delta$ .
  - d. The operation  $\Delta$  is clearly commutative.
2. Scalar multiplication on vectors is distributive.
  3. Scalar multiplication is associative.

Also observe that vectors  $(1, 0, 0, \dots, 0)$  associated with edge set  $e_1$ ,  $(0, 1, 0, \dots, 0)$  associated with edge set  $\{e_2\}$ ,  $\dots$ ,  $(0, 0, \dots, 0, 1)$  associated with edge set  $\{e_q\}$ , forms a basis for the vector space. Thus, the dimension of the vector space  $\epsilon(G)$  is  $q$ . **Definition 3.3.1: Fundamental Cycle:** Let  $T$  be any spanning tree of a connected graph  $G$ .

Adding just one edge to a spanning tree will create a cycle; such a cycle is called a fundamental cycle with respect to a spanning tree  $T$ .

There is a distinct fundamental cycle for each edge; thus, there is a one-to-one correspondence between fundamental cycles and edges not in the spanning tree. For a connected graph with  $p$  vertices, any spanning tree will have  $p - 1$  edges, and thus, for a graph of  $q$  edges and any one of its spanning trees will have  $q - p + 1$  fundamental cycles.

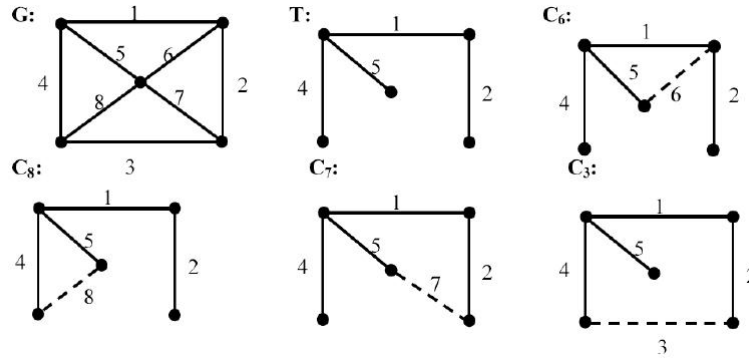


Fig:3.7

**Note:** Number on the edge indicate just an edge number, so 3 means  $e_3$ .

In the above Fig. 3.7,  $T$  is a spanning tree of graph  $G$  and  $C_3, C_6, C_7$  and  $C_8$  are fundamental cycles of  $G$  with respect to  $T$ . ( $C_i$  is obtained from  $T$  by adding an edge  $e_i$ ).


For any given spanning tree the set of all  $q - p + 1$  fundamental cycles forms a cycle basis, a basis for the cycle subspace of  $\epsilon(G)$ .

### 3.5 CAYLEY'S FORMULA

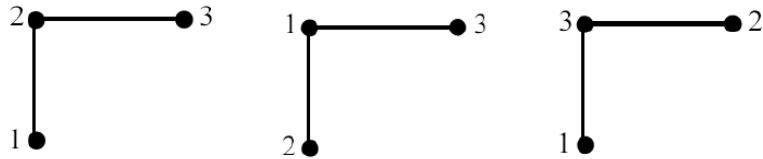
Cayley's formula counts the number of labelled trees on  $n$  vertices. In other words, it counts the number of spanning trees of a complete graph  $K_n$ . However, it does not count the number of non isomorphic trees on  $n$  vertices.



Before we proceed to the formula, let us find number of labelled trees for small values of  $n$  such as 2, 3, 4 and then we shall generalise using Cayley's formula.

For  $n = 2$ ,   $T_2 = 1$

For  $n = 3$ ,  $T_2 = 3$



For  $n=4$   $t_4=14$

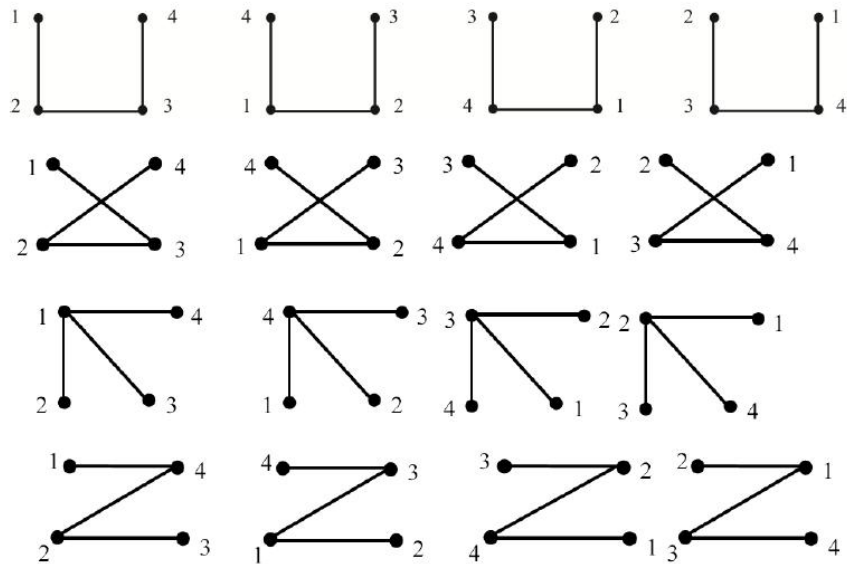


Fig:3.8

Let  $T_n$  denote number of labelled trees on  $n$  vertices. Then, Cayley's Formula states that:  $T_n = n^{(n-2)}$ .

Now let us count number of labelled trees on  $n$  vertices. In fact, Cayley's formula gives this count and many proofs of the formula are available. We shall use the simplest algorithm for counting that is "Prüfer Encoding". Before we proceed to the proof of the formula, let us understand "Prüfer Encoding".

**Prüfer Encoding:** The most straight forward method of showing that a set has a certain number of elements is to find a bijection between that set and some other set with a known number of elements. In this case, we are going to find a bijection between the set of Prüfer sequences and the set of spanning trees.

A Prüfer sequence is a sequence of  $n - 2$  numbers, each being one of the numbers from 1 to  $n$ . We observe that, there are  $n^{n-2}$  Prüfer sequences for any given  $n$ , where we allow repetitions. The following is an algorithm that can be used to encode any tree into a Prüfer sequence. Let  $T_n$  be a set of all trees on  $n$  vertices.

**Algorithm (Coding):**

1. Take any tree,  $t \in T_n$ , whose vertices are labelled from 1 to  $n$  in any manner.
2. Let  $i = 1, t_1 = t$ .
3. From  $t_i$ , choose vertex  $v$  with the smallest label whose degree is equal to 1, and write down the value of its only neighbour, say  $a_i$  ( $1 \leq i \leq n-1$ ) (We have already shown that any tree must have at least two leaf vertices).
4. Construct tree  $t_{i+1}$  from  $t_i$  by removing vertex  $v$  and edge  $va_i$ .
5. Update  $i$  to  $i + 1$ . Repeat from step 3 for the new, smaller tree. Continue until only one vertex remains.
6. Drop last neighbour from the list to get a sequence of  $n - 2$  vertices. (In fact, we observe that the last in the least is always the vertex with the highest label.) Now, we shall apply this algorithm to a tree on 8 vertices below.

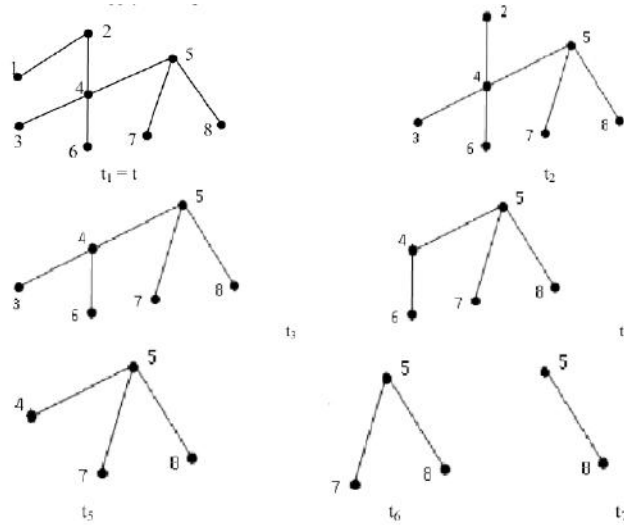


Fig:3.9

- Step 1: Remove vertex  $v = 1$  and  $a_1 = 2$
- Step 2: Remove vertex  $v = 2$  and  $a_2 = 4$
- Step 3: Remove vertex  $v = 3$  and  $a_3 = 4$

Step 4: Remove vertex  $v = 6$  and  $a_4 = 4$

Step 5: Remove vertex  $v = 4$  and  $a_5 = 5$

Step 6: Remove vertex  $v = 7$  and  $a_6 = 5$

Step 7: Remove vertex  $v = 5$  and  $a_7 = 8$

Now, as per the algorithm, we have to drop  $a_7 = 8$  and hence the Prüfer sequence is: 2, 4, 4, 4, 5, 5. We observe that the vertex label frequency in the sequence is  $\text{deg}(\text{vertex}) - 1$ . Also a vertex with degree one never appears in the sequence.

Thus, we have seen how to construct a Prüfer Sequence from a tree. Now is the time to construct a tree from a given Prüfer sequence  $P$ . We shall first discuss the algorithm for the same.

Algorithm (Decoding):

1.  $P$  is given Prüfer sequence and  $L = 1, 2, \dots, n-1, n$ .
2. Let  $j$ : First label in  $P$  and  $k$ : the least number in  $L$  that does not occur in  $P$ .
3. Connect an edge between  $j$  and  $k$ .
4. Remove  $j$  and  $k$  from  $P$  and  $L$  respectively.
5. Perform steps 2 to 4, till  $P$  is non-empty.
6. There will be exactly two numbers in  $L$  remaining. Join an edge between these two.

Thus, we get a tree corresponding to given Prüfer sequence.

Now, let us apply it on the Prüfer sequence  $P = 2, 4, 4, 4, 5, 5$ .

Step 1:  $P = 2, 4, 4, 4, 5, 5$ ;  $L = 1, 2, 3, 4, 5, 6, 7, 8$ ;  $j = 2$ ;  $k = 1$

Step 2:  $P = 4, 4, 4, 5, 5$ ;  $L = 2, 3, 4, 5, 6, 7, 8$ ;  $j = 4$ ;  $k = 2$

Step 3:  $P = 4, 4, 5, 5$ ;  $L = 3, 4, 5, 6, 7, 8$ ;  $j = 4$ ;  $k = 3$

Step 4:  $P = 4, 5, 5$ ;  $L = 4, 5, 6, 7, 8$ ;  $j = 4$ ;  $k = 6$

Step 5:  $P = 5, 5$ ;  $L = 4, 5, 7, 8$ ;  $j = 5$ ;  $k = 4$

Step 6:  $P = 5$ ;  $L = 5, 7, 8$ ;  $j = 5$ ;  $k = 7$

Step 7:  $P = \phi$ ;  $L = 5, 8$

Step 8: Connect edge 5-8.

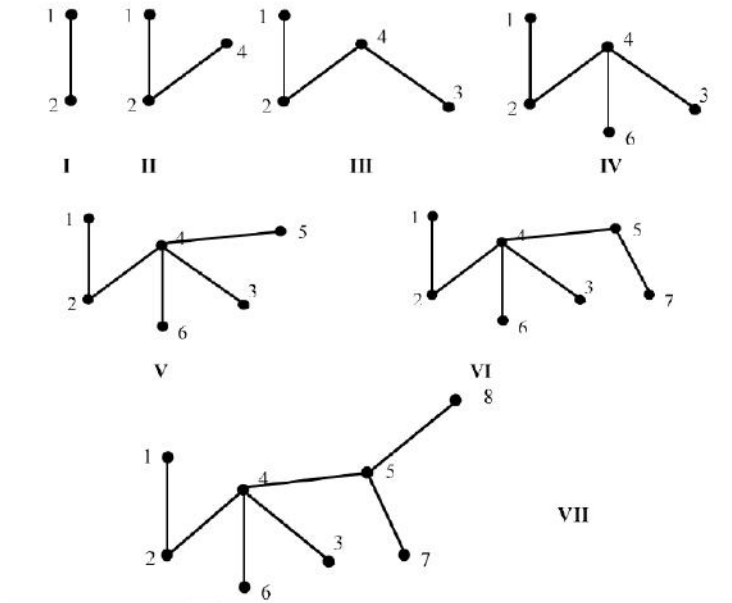


Fig:3.10

Following the above steps, we have now reconstructed our original tree on 8 vertices, as in Fig. 3.9. It may be oriented differently, but all of the vertices are adjacent to their correct neighbours, and so we have the correct tree back. Since there were no ambiguities on how to encode the tree or decode the sequence, we can see that for every tree there is exactly one corresponding Prüfer Sequence, and for each Prüfer Sequence there is exactly one corresponding tree. More formally, the encoding function can be thought of as taking a member of the set of spanning trees on  $n$  vertices,  $T_n$ , to the set of Prüfer Sequences with  $n-2$  terms,  $P_n$ . Decoding would then be the inverse of the encoding function, and we have seen that composing these two functions results in the identity map. If we let  $f$  be the encoding function, then the above statements can be summarized as follows:  $f: T_n \rightarrow P_n$ ,  $f^{-1}: P_n \rightarrow T_n$ , and  $f^{-1} f = I$ .

Since we have found a bijective function between  $T_n$  and  $P_n$ , we know that they must have the same number of elements. We know that  $|P_n| = n^{n-2}$ , and so  $|T_n| = n^{n-2}$ .

### 3.6 ROOTED AND BINARY TREES

rooted tree  $T(x)$  is a tree  $T$  with a specified vertex  $x$ , called the root of  $T$ . An orientation of a rooted tree in which every vertex but the root has in-degree one is called a branching. We refer to a rooted tree or branching with root  $x$  as an  $x$ -tree or  $x$ -branching, respectively

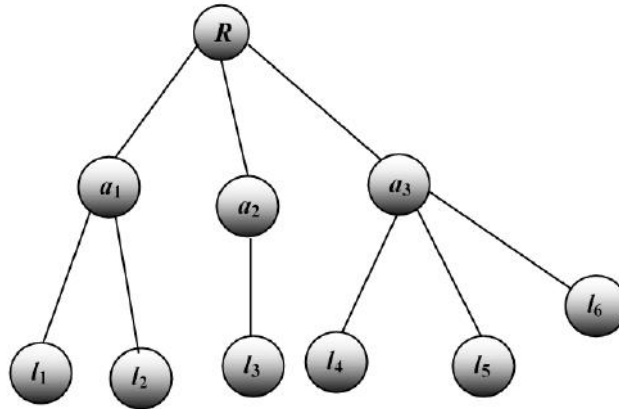


Fig:3.13

In the Fig. 3.13 above, R is root having in-degree 0 and out-degree 3. The vertices a1, a2 and a3 have in-degree 1 and out-degrees 2, 1 and 3 respectively. Vertices l1, l2, l3, l4, l5 and l6 have in-degree 1 each and out-degree 0. Vertices having out degree 0 are termed as leaves.

Vertices a1, a2 and a3 are called as children of R and R is the parent of these vertices. Vertices a1, a2 and a3 are siblings, as they have same parent.

Thus, rooted tree is a directed graph.

Binary trees are special kind of rooted trees. In binary trees there are maximum 2 children to any vertex. Vertex having no child is as before termed as leaf.

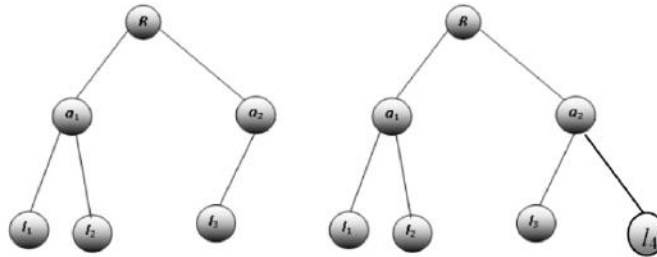


Fig:3.14(a)(b)

Rooted trees and branching are effective tools in designing of efficient algorithms for the purpose of reachability. There are certain terminologies exclusively associated with rooted binary trees.

- The depth of a node is the number of edges from the root to the node.
- The height of a node is the number of edges from the node to the deepest leaf.
- The height of a tree is a height of the root. (Thus, height of tree in Fig. 3.14(a) is 2)

- A complete binary tree is a binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from left to right. (Fig. 3.14(b))

As graphs and trees have many real life applications, the vertices are usually used to store some data and we may need to search or traverse to the node to access the data. Hence, traversal is of utmost importance in graphs and trees. Now, shall look at some algorithms for the tree and graph traversals.

### 3.7.1 Breadth First Search:

In most types of tree-search, the criterion for selecting a vertex to be added to the tree depends on the order in which the vertices already in the tree  $T$  were added. A tree-search in which the adjacency lists of the vertices of  $T$  are considered on a first-come-first-served basis, that is, in increasing order of their time of incorporation into  $T$ , is known as breadth-first search. In order to implement this algorithm efficiently, vertices in the tree are kept in a queue; this is just a list  $Q$  which is updated either by adding a new element to one end (the tail/ rear of  $Q$ ) or removing an element from the other end (the head/front of  $Q$ ). At any moment, the queue  $Q$  comprises all vertices from which the current tree could potentially be grown.

Initially, at time  $t = 0$ , the queue  $Q$  is empty. Whenever a new vertex is added to the tree, it joins  $Q$ . At each stage, the adjacency list of the vertex at the head of  $Q$  is scanned for a neighbour to add to the tree. If every neighbour is already in the tree, this vertex is removed from  $Q$ . The algorithm terminates when  $Q$  is once more empty. It returns not only the tree (given by its predecessor function  $p$ ), but also records the level of each vertex in the tree and, more importantly, their distances from  $r$  in  $G$ . It also returns a function  $t$  which records the time of incorporation of each vertex into the tree  $T$ . We keep track of the vertices in  $T$  by colouring them black. The notation  $G(x)$  signifies a graph  $G$  with a specified vertex (or root)  $x$ . Recall that an  $x$ -tree is a tree rooted at vertex  $x$ .

**Algorithm:** Breadth-First Search (BFS) Input: a connected graph  $G(r)$   
Output: an  $r$ -tree  $T$  in  $G$  with predecessor function  $p$ , a level function  $l$ , such that  $l(v) = d_G(r,v)$  for all  $v \in V$ , and a time function  $t$

- 1: set  $i := 0$  and  $Q := \phi$
- 2: increment  $i$  by 1
- 3: colour  $r$  black
- 4: set  $l(r) := 0$  and  $t(r) := i$
- 5: append  $r$  to  $Q$
- 6: while  $Q$  is nonempty do
- 7: consider the head  $x$  of  $Q$
- 8: if  $x$  has an uncoloured neighbour  $y$  then
- 9: increment  $i$  by 1
- 10: colour  $y$  black
- 11: set  $p(y) := x$ ,  $l(y) := l(x) + 1$  and  $t(y) := i$
- 12: append  $y$  to  $Q$
- 13: else

- 14: remove x from Q
- 15: end if
- 16: end while
- 17: return (p,l, t).

Before we discuss the algorithm in detail, let us first implement it on the following graph.

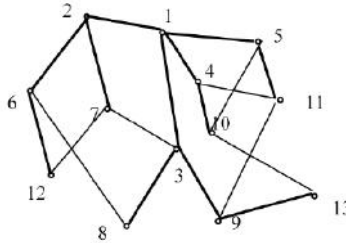


Fig:3.15

Now, let us apply BFS on the Fig. 3.15.

<i>i</i>	Colour	Length ( <i>l</i> )	Predecessor ( <i>p</i> )	Time ( <i>t</i> )
0	1 <i>B</i>	$l(1) = 0$	-	-
1	2 <i>B</i>	$l(2) = 1$	1	1
2	3 <i>B</i>	$l(3) = 1$	1	2
3	4 <i>B</i>	$l(4) = 1$	1	3
4	5 <i>B</i>	$l(5) = 1$	1	4
Vertex 1 removed from Q				
5	6 <i>B</i>	$l(6) = 2$	2	5
6	7 <i>B</i>	$l(7) = 2$	2	6
Vertex 2 removed from Q				
7	8 <i>B</i>	$l(8) = 2$	3	7
8	9 <i>B</i>	$l(9) = 2$	3	8
Vertex 3 removed from Q				
9	10 <i>B</i>	$l(10) = 2$	4	9
Vertex 4 removed from Q				
10	11 <i>B</i>	$l(11) = 2$	5	10
Vertex 5 removed from Q				
11	12 <i>B</i>	$l(12) = 3$	6	11
Vertex 6 removed from Q				
Vertex 7 removed from Q				
Vertex 8 removed from Q				
12	13 <i>B</i>	$l(13) = 3$	9	12
Vertex 9 removed from Q				
Vertex 10 removed from Q				
Vertex 11 removed from Q				
Vertex 12 removed from Q				
Vertex 13 removed from Q				

Table:3.1

Q:  $\phi \rightarrow 1 \rightarrow 12 \rightarrow 123 \rightarrow 1234 \rightarrow 12345 \rightarrow 2345 \rightarrow 23456 \rightarrow 234567 \rightarrow 34567 \rightarrow 345678 \rightarrow 3456789 \rightarrow 456789 \rightarrow 45678910 \rightarrow 5678910 \rightarrow 567891011 \rightarrow 67891011 \rightarrow 6789101112 \rightarrow 789101112 \rightarrow 89101112 \rightarrow 9101112 \rightarrow 910111213 \rightarrow 10111213 \rightarrow 111213 \rightarrow 1213 \rightarrow 13 \rightarrow \phi$ .

Based on this, the highlighted edges in the Fig. 3.15 show the BFS tree.

### 3.7.2 Depth-First Search

Depth-first search is a tree-search in which the vertex added to the tree  $T$  at each stage is one which is a neighbour of as recent addition to  $T$  as possible. In other words, we first scan the adjacency list of the most recently added vertex  $x$  for a neighbour not in  $T$ . If there is such a neighbour, we add it to  $T$ . If not, we backtrack to the vertex which was added to  $T$  just before  $x$  and examine its neighbours, and so on. The resulting spanning tree is called a depth-first search tree or DFS-tree.

This algorithm may be implemented efficiently by maintaining the vertices of  $T$  whose adjacency lists have yet to be fully scanned, not in a queue as we did for breadth-first search, but in a stack. A stack is simply a list, one end of which is identified as its top; it may be updated either by adding a new element as its top or else by removing its top element. In depth-first search, the stack  $S$  is initially empty. Whenever a new vertex is added to the tree  $T$ , it is added to  $S$ . At each stage, the adjacency list of the top vertex is scanned for a neighbour to add to  $T$ . If all of its neighbours are found to be already in  $T$ , this vertex is removed from  $S$ .

The algorithm terminates when  $S$  is once again empty. As in breadth-first search, we keep track of the vertices in  $T$  by colouring them black. Associated with each vertex  $v$  of  $G$  are two times: the time  $f(v)$  when  $v$  is incorporated into  $T$  (that is, added to the stack  $S$ ), and the time  $l(v)$  when all the neighbours of  $v$  are found to be already in  $T$ , the vertex  $v$  is removed from  $S$ , and the algorithm backtracks to  $p(v)$ , the predecessor of  $v$  in  $T$ . The time increments by one with each change in the stack  $S$ . In particular,  $f(r) = 1$ ,  $l(v) = f(v) + 1$  for every leaf  $v$  of  $T$ , and  $l(r) = 2n$ .

**Algorithm:** Depth-First Search

**Input:** a connected graph  $G$

**Output:** a rooted spanning tree of  $G$  with predecessor function  $p$ , and two time functions  $f$  and  $l$ .

- 1: set  $i := 0$  and  $S :=$
- 2: choose any vertex  $r$  (as root)
- 3: increment  $i$  by 1
- 4: colour  $r$  black
- 5: set  $f(r) := i$
- 6: add  $r$  to  $S$  (that is push  $r$  on stack  $S$ )
- 7: while  $S$  is nonempty do
- 8: consider the top vertex  $x$  of  $S$
- 9: increment  $i$  by 1
- 10: if  $x$  has an uncoloured neighbour  $y$  then
- 11: colour  $y$  black
- 12: set  $p(y) := x$  and  $f(y) := i$



```

13: add y to the top of S (that is push y on stack S)
14: else
15: set l(x) := i
16: remove x from S (that is pop x from the stack S)
17: end if
18: end while
19: return (p, f, l)

```

Now, let us apply DFS above for the graph below: Highlighted edges in the graph indicate DFS tree.

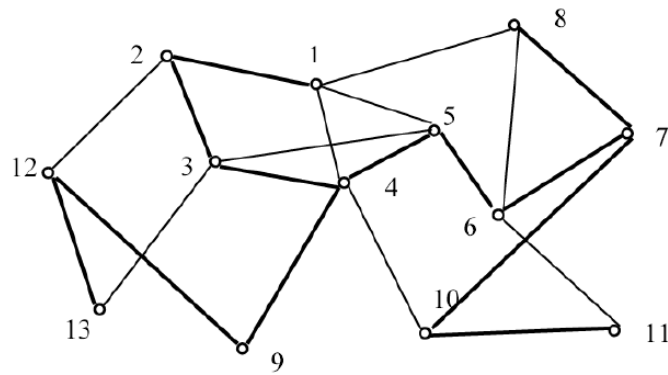


Fig:3.16

$i$	Colour	$f$	$L$	$P$
0	-	-	-	-
1	1 B	1	<b><u>Push(1)</u></b>	-
2	2 B	2	<b><u>Push(2)</u></b>	1
3	3 B	3	<b><u>Push(3)</u></b>	2
4	4 B	4	<b><u>Push(4)</u></b>	3
5	5 B	5	<b><u>Push(5)</u></b>	4
6	6 B	6	<b><u>Push(6)</u></b>	5
7	7 B	7	<b><u>Push(7)</u></b>	6
8	8 B	8	<b><u>Push(8)</u></b>	7
9	<b><i>Pop 8</i></b>	-	$l(8)=9$	-
10	10 B	10	<b><u>Push(10)</u></b>	7
11	11 B	10	<b><u>Push(11)</u></b>	10
12	<b><i>Pop 11</i></b>	-	$l(11)=12$	-
13	<b><i>Pop 10</i></b>	-	$l(10)=13$	-
14	<b><i>Pop 7</i></b>	-	$l(7)=14$	-
15	<b><i>Pop 6</i></b>	-	$l(6)=15$	-
16	<b><i>Pop 5</i></b>	-	$l(5)=16$	-
17	9 B	17	<b><u>Push(9)</u></b>	4
18	12 B	18	<b><u>Push(12)</u></b>	9
19	13 B	19	<b><u>Push(13)</u></b>	12
20	<b><i>Pop 13</i></b>	-	$l(19)=20$	-
21	<b><i>Pop 12</i></b>	-	$l(12)=21$	-
22	<b><i>Pop 9</i></b>	-	$l(9)=22$	-
23	<b><i>Pop 4</i></b>	-	$l(4)=23$	-
24	<b><i>Pop 3</i></b>	-	$l(3)=24$	-
25	<b><i>Pop 2</i></b>	-	$l(2)=25$	-
26	<b><i>Pop 1</i></b>	-	$l(1)=26$	-

Table:3.2

**Note:** Unused columns are used to indicate push and pop.

## 4 Matching and Ramsey Theory

Matching theory is used to find the similarity between the graphs. It is an important tool in the fields like computer vision and pattern recognition. Matching has applications in flow networks, scheduling and planning, modeling bonds in chemistry, graph coloring, the stable marriage problem, neural networks in artificial intelligence and more. In image recognition applications, the results of image segmentation in image processing typically produces data graphs with number of vertices much larger than in the model graphs data expected to match against. Perfect Matching theory is also known as graph isomorphism problem. Many graph matching algorithm exist in order to optimize for the parameters. First we have to go through some definitions.

**Matching in Graph:** A matching in graph  $G$  is a set  $M = \{e_1, e_2, e_3, \dots, e_k\}$  of edges such that each vertex  $v \in V(G)$  appears in at most one edge of  $M$  i.e  $e_i \cap e_j = \phi$  for all  $i, j$ .

The size of matching is the number of edges that appears in the matching.

**M- Saturated vertex:** A vertex  $v$  is called as M- saturated if for some  $e = \{xy\}$ ,  $e \in M$ . If  $x$  is not saturated then it is unsaturated.  $2|M|$  = number of M- saturated vertices.

**Perfect Matching:** A perfect matching in a graph  $G$  is a matching in which every vertex of  $G$  appears exactly once, i.e Which saturates every vertex or a matching of size exactly

$$\frac{n}{2}$$

**Maximum Matching:** A matching  $m$  is called maximum if no other matching in  $G$  has a larger size.

**M- alternating Path:** Let  $M$  be a matching in a graph  $G$ . A path  $P$  in  $G$  is said to be M-alternating if every other edge in  $P$  appear in  $M$

**M- augmenting Path:** An M-augmenting path is an M-alternating path  $P = \{v_1, v_2, v_3, \dots, v_k\}$  such that both  $v_1, v_k$  are not vertices in  $M$ .

**Berg theorem:** Let  $M$  be matching in a graph  $G$  Then  $M$  is a maximum matching if and only if there does not exist any M- augmenting path in  $G$ .

**Proof.** Suppose that  $M$  is a matching in  $G$ , such that there exist an M-augmenting path say  $P$ . Notice that  $P$  must have odd length. Since its edges alternate between edges in  $M$  and edges in  $G \setminus M$ , and further both begins and ends with edges from  $G \setminus M$ . Let  $M_0$  be the set of edges in  $P$  that are not in  $M$ . Then notice that  $M_0 \oplus M$  is a matching in  $G$ . so  $M$  is not a maximum matching. Hence if  $M$  is a maximum matching, there can not be any M-augmenting path in  $G$ . Conversely assume that  $M$  is a matching having no M-augmented path in  $G$ . Let  $M_2$  be a maximum matching in  $G$ . Note that by above argument, there is no  $M_2$  augmenting path in  $G$ . Let  $H$  be the subgraph of  $G$  having  $E(H) = \{e \in M \text{ but } e \notin M_2\} \cup \{e \in M_2 \text{ but } e \notin M\}$ . Let us consider the possible components of  $H$ . Note that every vertex has degree 0, 1, or 2 in  $H$ . Vertices of degree 0 are disregarded as their components are trivial. If a component has all vertices of degree 2. It must be an even cycle alternating

between edges of  $M$  and edges of  $M_2$ . If a component has a vertex degree 1 it must be a path, alternating between edges of  $M$  and edges of  $M_2$ . Note that neither  $M$  nor  $M_2$  has an augmented path  $G$ , we must have that such a component begins with an edge from one matching and ends with an edge from the other matching. In either case every non-trivial component of  $H$  has exactly half of its edges from  $M$  and exactly half of its edges from  $M_2$ . Hence we must have that

$$|M \setminus M_2| = |M_2 \setminus M|$$

. Hence

$$|M| = |M \cap M_2| + |M \setminus M_2| = |M \cap M_2| + |M_2 \setminus M| = |M_2|$$

. and thus  $M$  is also a maximum matching in  $G$ .

## 5 MATCHING IN BIPARTITE GRAPHS

Let  $G$  be a graph with vertex set partitioned into two subsets  $A$  and  $B$  such that every edge in  $G$  has one end point in  $A$  and the other in  $B$ . Such a graph is called a bipartite graph. We will use the notation  $G(A, B)$  for a bipartite graph. Suppose that  $A$  and  $B$  are subsets of  $G$ . We can say that  $A$  can match with  $B$  if there exists a matching  $M = \{e_1, e_2, e_3, \dots, e_k\}$  such that each  $e_i$  has one vertex in  $A$  and the other in  $B$  and every vertex in  $A$  and  $B$  appears in the matching.

### 5.1 Neighbour set of $S$ in Graph $G$ :

$N(v) = \{u \in V(G) \mid u \text{ is adjacent to } v\}$  is called as the set of neighbours of  $v$ . Given a set  $S \subset V(G)$ , we write  $N(S) = \{v \in V(G) \mid v \text{ is adjacent to at least one vertex in } S\}$  is the set of vertices that are adjacent to at least one vertex in  $S$ .

**Hall's Theorem** Let  $G(A, B)$  be a bipartite graph. Suppose that

$$|A| \leq |B|$$

. Then there exists a matching  $M$  of size

$$|A|$$

in  $G$  if and only if for every subset  $S$

$$S \subset A, \text{ we have that } |N(S)| \geq |S|$$

. In particular, if

$$|A| = |B|$$

, then  $G$  has a perfect matching under this condition.

**proof** Let  $A = \{u_1, u_2, \dots, u_k\}$  and  $B = \{v_1, v_2, \dots, v_l\}$  with  $k \leq l$ . First suppose that there exists a matching  $M$  of size  $|A|$  in  $G$ . Since  $G$  is bipartite, every edge of  $M$  includes one vertex of  $A$ . By possible relabeling of  $B$ , suppose that  $M = \{e_1,$

$e_2, \dots, e_k$  where  $e_i = \{u_i, v_i\}$  for each  $i$ . Let  $S \subset A$  with  $S = \{u_1, u_2, \dots, u_t\}$ . Then  $v_j \in N(S)$  for all  $1 \leq j \leq t$ . and hence  $|N(S)| \geq t = |S|$ . Conversely we prove by induction on  $|A|$ . First, note that if  $|A| = 1$ , the theorem is trivial. Let us assume that the theorem holds for  $|A| = k - 1$ . Let  $G(A, B)$  be a bipartite graph with  $|A| = k$  satisfying Hall's condition. we consider two cases.

**case 1:** For every proper subset  $S$  of  $A$ ,  $|N(S)| \geq |S| + 1$ . Consider  $u_1$ , without loss of generality suppose that  $u_1$  is adjacent to  $v_1$  (and possibly some other vertices also). Consider the subgraph  $H$  of  $G$  on  $A \setminus \{u_1\}, B \setminus \{v_1\}$ , i.e remove both vertices  $u_1$  and  $v_1$  from consideration. Let  $S$  be a subset of  $A \setminus \{u_1\}$ . Then note that  $N_H(S)$  is either equal to  $N_G(S)$  or has been reduced in size by 1 due to removal of  $v_1$ . In either case,  $|N_H(S)| \geq |N_G(S)| - 1 \geq |S|$ . And hence  $H$  satisfies Hall's condition. We can therefore obtain a matching in  $H$  of size  $|A| - 1$  by the induction hypothesis. adding the edge  $\{u_1, v_1\}$  to such a matching produces a matching in  $G$  of size  $|A|$ .

**Case 2:**  $A$  contains a proper subset  $S$  having  $|S| = |N(S)|$ . Note that since  $S$  is a proper subset of  $A$ , by induction hypothesis we have that the subgraph of  $G$  on  $(S, N(S))$  satisfies Hall's condition and hence we may find a matching on this subgraph of size  $|S|$ . Let  $H$  be the subgraph of  $G$  on  $(A \setminus S, B \setminus N(S))$ : Note that we have removed the same number of vertices from both  $A$  and  $B$ . Let  $T$  be a subset of  $A \setminus S$ . Then notice that  $N_G(S \cup T) = N_G(S) \cup N_H(T)$  and these two neighborhoods are disjoint. Furthermore by Hall's condition we have  $|N_G(S \cup T)| \geq |S \cup T| = |S| + |T|$ . Therefore  $|N_H(T)| = |N_G(S \cup T)| - |N_G(S)| \geq |S| + |T| - |S| = |T|$ . and hence  $H$  satisfies Hall's condition. We thus can find a matching in  $H$  of size  $|A \setminus S|$ . Taking the union of this matching with the matching on  $S$  gives a matching in  $G$  of size  $|A|$  as desired.

**Some obvious features of perfect Matching:** (1) If  $G$  has an odd number of vertices, then it has no perfect matching. (2) If  $G$  has any isolated vertices then it has no perfect matching. (3) If  $G$  has a component of odd size, then it has no perfect matching.

**Notation:** For any given graph  $G$ ,  $o(G)$  denote the number of odd components of  $G$ .

**Tutte's Theorem:** Let  $G$  be a graph. Then  $G$  contains a perfect matching if and only if for every proper subset  $S \subset V(G)$  we have  $o(G \setminus S) \leq |S|$ .

**Proof.** We first consider the forward implication. Suppose that  $G$  contains a perfect matching  $M$ . Let  $S = \{v_1, v_2, \dots, v_k\} \subset V(G)$  be a proper subset of  $V(G)$ . let  $G_1, G_2, \dots, G_n$  be the odd components of  $G \setminus S$ . Since  $G_i$  is odd, some vertex  $u_i$  of  $G_i$  must be matched under  $M$  with a vertex  $v_i$  of  $S$ .  $o(G \setminus S) = n \leq |S|$  Let us now consider the backward implication. Let  $G$  be a graph on  $n$  vertices. We shall prove this by induction on  $n$ . Note the base case is when  $n=2$ . Then  $G$  is  $K_2$ . It satisfies Tutte's condition and has a perfect matching. Thus theorem hold for  $n=2$ . We assume that if graph has  $n-2$  or fewer vertices (note, we may delete two vertices since no odd graph has a perfect matching and hence  $n$  is even.) then the theorem holds. Now we have a graph  $G$  on  $n$  vertices in which Tutte's condition is satisfied. We consider two case:

**case 1:** For every proper subset  $S$  of  $V(G)$ ,  $o(G \setminus S) \leq |S| - 1$ . Note that as  $n$  is even, we must have that  $o(G \setminus S)$  and  $|S|$  are of the same parity. so

infact we have  $o(G \setminus S) \leq |S| - 2$  for every proper subset of  $V(G)$ . Fix an edge  $uv$  and consider  $H = G \setminus \{u, v\}$  having  $n-2$  vertices. Let  $T \subset V(H)$ . Note that  $o(H \setminus T) = O(G \setminus (T \cup \{u, v\})) \leq |T \cup \{u, v\}| - 2 = |T|$ . Hence  $H$  satisfies Tutte's criterion and thus  $H$  has a perfect matching  $M_0$ . Taking  $M = M_0 \cup \{uv\}$  yields a perfect matching in  $G$ .

**case 2:** There exist a proper subset  $S$  of  $V(G)$  with  $o(G \setminus S) = |S|$ . Let  $S$  be the largest proper subset of  $V(G)$  having  $o(G \setminus S) = |S| = k$  and let  $S = \{v_1, v_2, \dots, v_k\}$ . We first claim that  $G \setminus S$  contains only odd components. Let if  $H$  be an even component of  $G \setminus S$ . Then fix any vertex  $u_0 \in V(H)$ . Note that  $H \setminus \{u_0\}$  must have atleast one odd component as it has an odd number of vertices and hence  $|S \cup \{u_0\}| \geq o(G \setminus (S \cup \{u_0\})) \leq |S \cup \{u_0\}|$ . yielding a strictly larger set satisfying the hypothesis of the case. Hence no even component exist. Let  $G_1, G_2, \dots, G_k$  be the  $k$ -components of  $G \setminus S$  and each of these is odd. Create a bipartite graph  $B$  as follows. Let  $V = S$  and let  $U = \{u_1, u_2, \dots, u_k\}$ . Put an edge  $u_i v_j$  in  $B$  if  $v_j$  is adjacent to some vertex in  $G_i$ .

**Claim:**  $B$  satisfies Hall's criterion. Suppose  $T$  is a proper subset of  $U$  with  $|T| = t$ . Suppose that  $|N(T)| < |T|$ . Let  $S_0 = \{v_{i_1}, v_{i_2}, \dots, v_{i_r}\} = N(T) \subset S$ , with  $r < t$ . Note that for each  $u_i \in T$ , we have that  $G_i$  is a component of  $G \setminus S$  since it is adjacent to no other vertices in  $S$ . Hence  $o(G \setminus S') \geq t > r = |S'|$ . This is a contradiction of Tutte's condition. Hence  $B$  satisfies Hall's criterion. Therefore, there exist a perfect matching  $M$  in  $B$ . Without loss of generality by relabel the components of  $G \setminus S$  so that we have, for every  $v_i \in S$ . That  $v_i$  is adjacent to some vertex of  $G_i$  call this vertex as  $u_i$ . Let us first show that we can find a perfect matching in  $G_j \setminus \{u_j\}$ . If  $|V(G_j)| = 1$  then we are done. if not, suppose that  $|V(G_j)| \geq 3$ . Let  $H = G_j \setminus \{u_j\}$ . Let  $W \subset V(H)$  be a proper subset. Note that  $o(H \setminus W) = o(G \setminus (S \cup \{u_j\} \cup W)) - (k - 1)$ . Since we have all the odd components  $G_1, G_2, \dots, G_k$ . Excepting  $G_j$  counted in the right hand side, which can be rectified by simply subtracting  $k-1$ . Moreover due to maximality of  $S$ . We have  $o(G \setminus (S \cup \{u_j\} \cup W)) < |S \cup \{u_j\} \cup W| = |W| + k + 1$  and hence  $o(H \setminus W) < |W| + k + 1 - (k - 1) = |W| + 2$ . As  $o(H \setminus W)$  and  $|W|$  must have same parity. This implies  $o(H \setminus W) = |W|$ . Hence  $H$  satisfies Tutte's criterion. Thus by induction we can form a perfect matching in  $H$ . By performing the above procedure. We thus can form a perfect matching  $M_j$  in  $G_j \setminus \{u_j\}$  for all  $j$ . Taking  $M_1 \cup M_2 \cup \dots \cup M_j \cup \{u_1 v_1, u_2 v_2, \dots, u_k v_k\}$  yields a perfect matching in  $G$ .

## 6 Independent sets and covering

**Independent set (Stable set):** Let  $G(V, E)$  be a graph. A independent set is a subset  $C$  of  $V$  such that no two vertices of  $C$  are adjacent in  $G$ . An independent set is maximum if  $G$  has no independent set  $C'$  with  $|C'| > |C|$ .

**Vertex cover:** A vertex cover is a set  $W$  of  $V$  such that every edge of  $G$  has atleast one end in  $W$ . Definition 10. **Edge cover:** A edge cover is a subset  $F$  of  $E$  such that for each vertex  $v$  there exist  $e \in F$  satisfying  $v \in e$ . Note: An edge cover can exist only if  $G$  has no isolated vertices.

**Notation:**

$\alpha(G) = \max\{|C|/C\}$  is an independent set.

$\tau(G) = \min\{|W|/W\}$  is a vertex cover

$\nu(G) = \max\{|M|/M\}$  is a Matching

$\rho(G) = \min\{|F|/F\}$  is an edge cover

**Note:**  $\alpha(G) \leq \rho(G)$  and  $\nu(G) \leq \tau(G)$

**Theorem** A set  $C \subset V$  is an independent set of  $G$  if and only if  $V \setminus C$  is a vertex covering of  $G$ .

**Proof** By definition,  $C$  is an independent set of  $G$  if and only if no edge of  $G$  has both ends in  $C$  equivalently if and only if each edge has at least one end in  $V \setminus C$ . But this is so if and only if  $V \setminus C$  is a vertex covering of  $G$ . Corollary  $\alpha(G) + \tau(G) = n$ . Proof. Let  $C$  be a maximum independent set of  $G$  and  $W$  be a minimum vertex covering of  $G$ . Then by above theorem  $V \setminus W$  is an independent set and  $V \setminus C$  is a vertex covering. Therefore  $n - \tau(G) = |V \setminus W| \leq \alpha(G)$ . (1).  $n - \alpha(G) = |V \setminus C| \geq \tau(G)$ . (2). From (1) and (2) we have  $\alpha(G) + \tau(G) = n$ .

**Gallai's theorem:** If  $G=(V, E)$  is a graph without isolated vertices then  $\nu(G) + \rho(G) = |V|$ .

**Proof** Let  $M$  be a matching of size  $\nu(G)$ . Let  $U$  be the set of  $M$ -unsaturated vertices (vertices which are not end point of any edge in  $M$ ). Since  $G$  has no isolated vertex and  $M$  is maximum, there exist a set  $E'$  of  $|U|$  edges, one incident with each vertex in  $U$ . Clearly,  $M \cup E'$  is an edge covering of  $G$ , and so  $\rho(G) \leq |M \cup E'| = \nu(G) + (n - 2\nu(G)) = n - \nu(G)$ . (1) Now let  $L$  be a minimum edge covering of  $G$ , set  $H=G[L]$  and let  $M$  be a maximum matching in  $H$ . Denote the set of  $M$ -unsaturated vertices in  $H$  by  $U$ . Since  $M$  is maximum,  $H[U]$  has no links and therefore  $|L| - \nu(G) = |L \setminus M| \geq |U| = n - 2\nu(G)$ . (2)  $|L| + |M| \geq n$  Because  $H$  is a subgraph of  $G$ ,  $M$  is a matching in  $G$  and so  $\rho(G) + \nu(G) \geq |L| + |M| \geq n$ . (3) from (1) and (2) we get  $\rho(G) + \nu(G) = n$  Let  $M$  be a matching and  $K$  be a covering such that  $|M| = |K|$  then,  $M$  is a maximum matching and  $K$  is a minimum covering. Proof. If  $M'$  is a maximum matching and  $K'$  is minimum covering then  $|M| \leq |M'| \leq |K'| \leq |K|$  Since  $|M| = |K|$ , it follows that  $|M| = |M'|$  and  $|K| = |K'|$ .

**Koings matching Theorem** In a bipartite graph, the number of edges in a maximum matching is equal to the number of vertices in a minimum covering.

**Proof** Let  $G$  be a bipartite graph with bipartition  $(X, Y)$  and let  $M'$  be a maximum matching of  $G$ . Let  $U$  be the set of  $M'$ -unsaturated vertices in  $X$  and Let  $Z$  be the set of all vertices connected by  $M'$ -alternating paths to vertices of  $U$ . Let set  $S = Z \cap X$  and  $T = Z \cap Y$ . Then by Hall's theorem, we have that every vertex in  $T$  is  $M'$ -saturated and  $N(S)=T$ . Define  $K' = (X \setminus S) \cup T$ . Every edge of  $G$  must have at least one of its ends in  $K'$  or otherwise, there would be an edge with one end in  $S$  and one end in  $Y \setminus T$ , contradicting  $N(S)=T$ . Thus  $K'$  is a covering of  $G$  and Clearly  $|M'| = |K'|$ . By above lemma  $K'$  is a minimum covering. The theorem follows.

**Konigs's edge covering theorem** In a bipartite graph  $G$  with no isolated vertex, the number of vertices in a maximum independent set is equal to the number of edge in a minimum edge covering.

**Proof** let  $G$  be a bipartite graph with no isolated vertex, By Gallai's theorem, we have  $\alpha(G) + \tau(G) = \nu(G) + \rho(G)$  and since  $G$  is bipartite, it follows

from Konig's matching theorem  $\nu(G) = \tau(G)$ . Thus  $\alpha(G) = \rho(G)$ .

## 7 The Personnel Assignment problem:

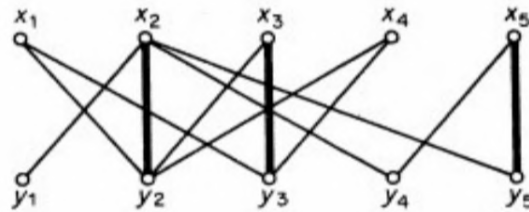
In a certain company,  $n$  workers  $X_1, X_2, \dots, X_n$  are available for  $n$  jobs  $y_1, y_2, \dots, y_n$ , each worker being qualified for one or more of these jobs. Can all the men be assigned, one man per job, two jobs for which they are qualified? This is the personnel assignment problem. We construct a bipartite graph  $G$  with bipartition  $(X, Y)$ , where  $X = \{x_1, x_2, \dots, x_n\}$ ,  $Y = \{y_1, y_2, \dots, y_n\}$  and  $x_i$  is joined to  $y_j$  if and only if worker  $x_i$  is qualified for job  $y_j$ . The problem becomes one of determining whether or not  $G$  has a perfect matching. According to Hall's theorem either  $G$  has such a matching or there is a subset  $S$  of  $X$  such that  $|N(S)| < |S|$ . Now we present an algorithm to solve the personnel assignment problem.

**Algorithm:** Start with an arbitrary matching  $M$ .

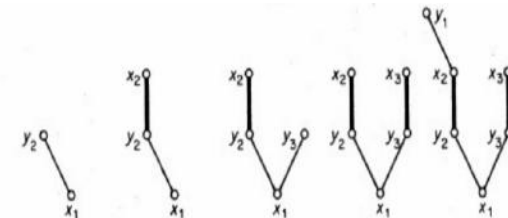
(1) If  $M$  saturates every vertex in  $X$  then stop otherwise, Let  $u$  be an  $M$ -unsaturated vertex in  $X$ . Set  $S = \{u\}$  and  $T = \emptyset$

(2) If  $N(S) = T$  then  $|N(S)| < |S|$ , Since  $|T| = |S| - 1$  then stop. since by Hall's theorem there is no matching that saturates every vertex in  $X$ . Otherwise, let  $y \in N(S) \setminus T$ .

(3) If  $y$  is  $M$ -saturated, let  $yz \in M$ . Replace  $S$  by  $S \cup \{z\}$  and  $T$  by  $T \cup \{y\}$  and go to step 2. (observe that  $|T| = |S| - 1$  is maintained after this replacement.) Otherwise, let  $P$  be an  $M$ -augmenting  $(u - y)$  path. Replace  $M$  by  $M' = M \oplus (P)$  and go to step 1. Consider a graph given below with initial matching  $M = \{x_2y_2, x_3y_3, x_5y_5\}$ .

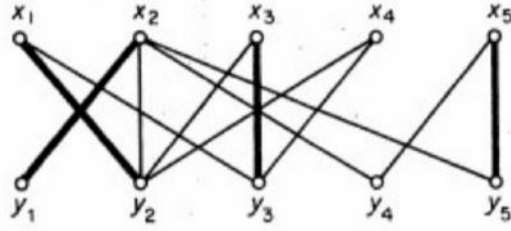


An  $M$ -alternating tree is grown, starting with  $x_1$  and the  $M$ -augmenting path  $x_1y_2x_2y_1$  found, as shown in figure below.

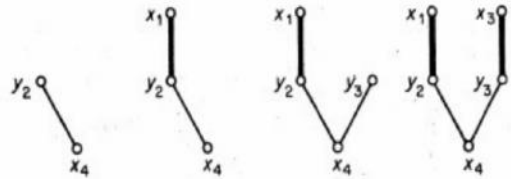




This result in a new matching  $M' = \{x_1y_2, x_2y_1, x_3y_3, x_5y_5\}$  as shown in figure below.

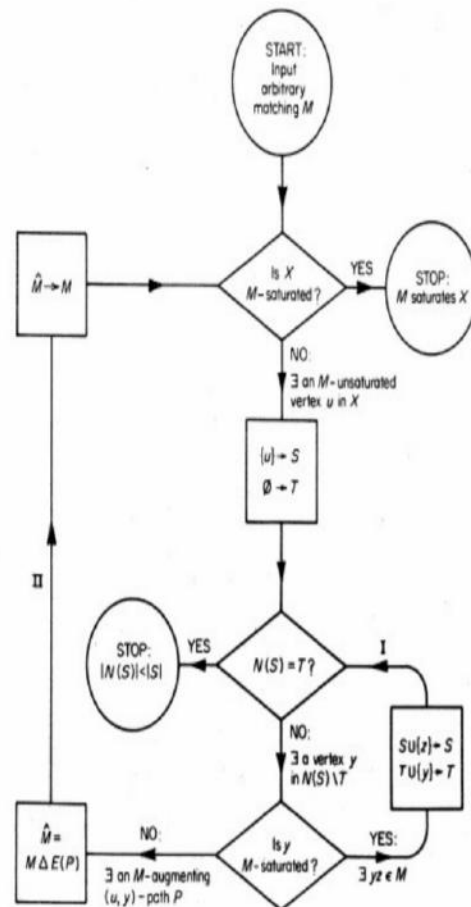


An  $M'$ -alternating tree is now grown from  $x_4$ . Since there is no  $M'$ - augmenting path with origin  $x_4$ , the algorithm terminates.



The set  $S = \{x_1, x_3, x_4\}$  with neighbour set  $N(S) = \{y_2, y_3\}$  shows that  $G$  has no perfect matching.

## 7.1 Flow Chart of Hungarian Method:



The Hungarian Method

## 8 Ramsey Number

**Clique:** A clique of a simple graph  $G$  is a subset  $S$  of  $V(G)$  such that any two vertices of  $S$  are adjacent.  $S$  is a clique of  $G$  if and only if  $S$  is an independent set of  $G^c$ . If  $G$  has no large cliques, then  $G$  has a large independent set. The above remark was first proved by Ramsey (1930).

**Question:** Among 6 people, there are either 3 who know each other or 3 who do not know each other.

**Proof:** Let 1,2,3,4,5,6 be the 6 people. Consider these 6 people as vertices of graph.  $i$  vertex is connected to  $j$  vertex by an edge then it means  $i$  and  $j$  know each other. otherwise they do not know each other. Let us select vertex

1 and join the with some of remaining vertices. By pi- genhole principle, either 1 knows 3 people or 1 does not know 3 people i.e 1 is connected to either 3 vertices or not connected to 3 vertices by an edge.

**Case1:**If 1 is adjacent to three vertices . If 1 is adjacent to three vertices say a, b, c. If two of a, b, c are adjacent then 1 with those two vertices form K3. This means 1 and other two people know each other.

Otherwise if none of a, b, c are adjacent. Then we get three isolated vertices, i.e we get three people they do not know each other.

**Case2:** 1 is not adjacent to three vertices. If 1 is not adjacent to three vertices a, b, c. If two of a, b, c are not adjacent then we get three vertices isolated. means Three people do not know each other.

**Definition** If we color the edges of  $K_n$  with red or blue, then there is either a set of  $r$  vertices such that edges among them are colored red or a set of  $b$  vertices such that edges among them are colored blue.

**Ramsey Numbers:** For any positive integers  $k$  and  $l \leq 2$  there exist a smallest integer  $t=r(k, l)$  such that any graph on  $t$  vertices contains either a clique of  $k$  vertices or an independent set of  $l$  vertices.  $r(1,l)=r(k,1)=1$   $r(2, l)=l$ ,  $r(k, 2)=k$ ,  $r(k, l)=r(l, k)$

**Theorem** For any two integers  $k \leq 2$  and  $l \leq 2$  then prove that  $r(k, l) \geq r(k, l - 1) + r(k - 1, l)$ . Furthermore, if  $r(k, l - 1)$  and  $r(k - 1, l)$  are both even, then strict inequality holds .

**Proof.**Let  $G$  be a graph on  $r(k, l - 1) + r(k - 1, l)$  vertices, and let  $v \in V$  . We distinguish two cases:

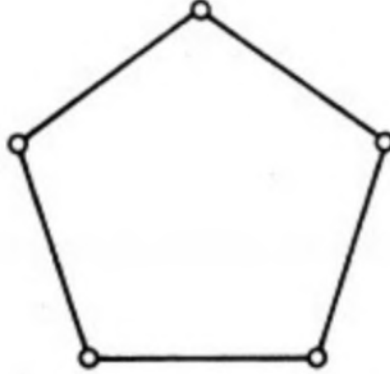
**Case1:**  $v$  is non adjacent to a set of atleast  $r(k, l - 1)$  vertices.  $G[S]$  contains either a clique of  $k$  vertices or an independent set of  $l - 1$  vertices and therefore  $G[S \cup \{v\}]$  contains either a clique of  $k$  vertices or an independent set of  $l$  vertices.

**Case2:**  $v$  is adjacent to a set  $T$  of atleast  $r(k - 1, l)$  vertices.  $G[T]$  contains either a clique of  $k-1$  vertices or an independent set of  $l$  vertices. Therefore  $G[T \cup \{v\}]$  contains either a cilque of  $k$  vertices or an independent set of  $l$  vertices. Since one of case(1) and case(2) must hold because the number of vertices to which  $v$  is non adjacent plus the number of vertices to which  $v$  is adjacent is equal to  $r(k, l - 1) + r(k - 1, l) - 1$ .

**Proof.** Thus  $r(k, l) \leq r(k, l - 1) + r(k - 1, l)$  Now suppose that  $r(k, l - 1)$  and  $r(k-1, l)$  are both even and let  $G$  be a graph on  $r(k, l - 1) + r(k - 1, l) - 1$  vertices. Since  $G$  has an odd number of vertices, then some vertices  $v$  is of even degree; in particular ,  $v$  cannot be adjacent to precisely  $r(k-1, l)-1$  vertices. Consequently, either case1 or case2 of above holds and therefore  $G$  contains either a clique of  $k$  vertices or an independent set of  $l$  vertices. Thus  $r(k, l) \leq r(k, l - 1) + r(k - 1, l) - 1$  as stated.

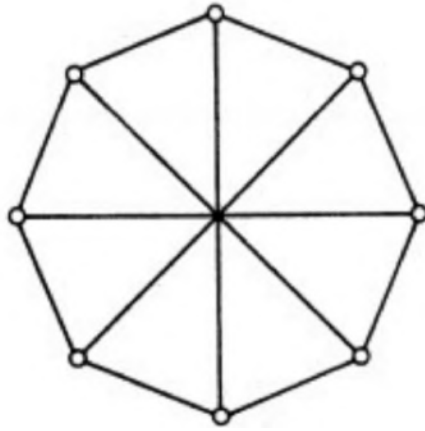
**Theorem** Prove that  $r(3, 3) = 6$ ,  $r(3, 4) = 9$ ,  $r(3, 5) = 14$ ,  $r(4, 4) = 18$ .

**Proof.(i):** From Theorem above we can get,  $r(3, 3) \geq r(3, 2) + r(2, 3) = 3 + 3 = 6 \cdot \cdot \cdot (1)$  From figure below , The 5 cycle contains no clique of three vertices and no independent set of three vertices.



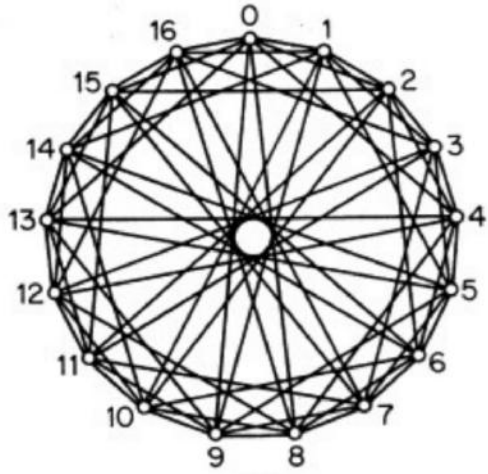
It shows that  $r(3, 3) \leq 6 \dots (2)$ . from (1) and (2)  $r(3, 3) = 6$

**Proof.** (ii) From Theorem above we can get,  $r(3, 4) \leq r(3, 3) + r(2, 4) - 1 = 6 + 4 - 1 = 9 \dots (1)$  The graph of figure below , contains no clique of three vertices and no independent set of four vertices.



It shows that  $r(3, 4) \leq 9 \dots (2)$ . from (1) and (2)  $r(3, 4) = 9$

**Proof.** (iii) From Theorem above we can get  $r(3, 5) \leq r(3, 4) + r(2, 5) = 9 + 5 = 14 \dots (1)$  The graph of figure below , contains no clique of three vertices and no independent set of five vertices.



It shows that  $r(4, 4) \leq 18 \cdot \dots (2)$ . from (1) and (2)  $r(4, 4) = 18$

Ramsey graph: A  $(k, l)$ - Ramsey graph is a graph on  $r(k, l) - 1$  vertices that contains neither a clique of  $k$  vertices nor an independent set of  $l$  vertices. Such a graph exist for all  $k \leq 2$  and  $l \leq 2$ . All the graph shown in the above theorem represent Ramsey- graph.