6.12 SUGGESTED READINGS

Anuranjan Misra, *Discrete Mathematics*, Acme Learning pvt ltd.

Richard Johnsonbaugh, *Discrete Mathematics*, Prentice Hall

V. K. Balakrishnan, *Introductory Discrete Mathematics*, Courier Dover Publications,

R. C. Penner, *Discrete Mathematics: Proof Techniques and Mathematical Structures*, World Scientific, 1999

Mike Piff, *Discrete Mathematics: An Introduction for Software* Engineers, Cambridge University Press, 1991

UNIT IV

LESSON

7

GRAPH THEORY

CONTENTS

7.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Understand graphs with examples such as walks, paths, trees, etc.
- Discuss bipartite and complete bipartite graphs
- Discuss eulerian and Hamiltonian graphs
- Discuss isomorphism and homeomorphismgraphs
- Discuss matrix representation of graphs

7.1 INTRODUCTION

The word *graph* refers to a specific mathematical structure usually represented as a diagram consisting of points joined by lines. In applications, the points may, for instance, correspond to chemical atoms, towns, electrical terminals or anything that can be connected in pairs. The lines may be chemical bonds, roads, wires or other connections. Applications of graph theory are found in communications, structures and mechanisms, electrical networks, transport systems, social networks and computer science.

7.2 GRAPHS – BASICS

A graph is a mathematical structure comprising a set of vertices, V, and a set of edges, E, which connect the vertices. It is usually represented as a diagram consisting of points, representing the **vertices** (or **nodes**), joined by lines, representing the **edges** (Figure 7.1). It is also formally denoted by G (V, E).

We have shown below some examples of graph:

In a **labelled graph** the vertices have labels or names (Figure 7.2).

In a **weighted graph** each edge has a **weight** associated with it (Figure 7.3).

A **digraph** (**directed graph**) is a diagram consisting of points, called vertices, joined by directed lines, called **arcs** (Figure 7.4).

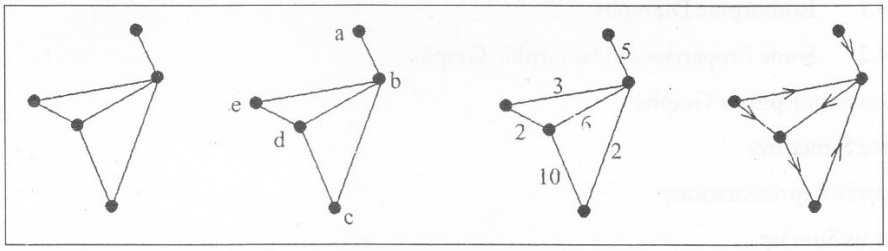


Figure 7.1 Figure 7.2 Figure 7.3 Figure 7.4

Two or more edges joined the same pair of vertices are called **multiple edges** (Figure 7.5). An edge joining a vertex to itself is a loop (Figure 7.5).

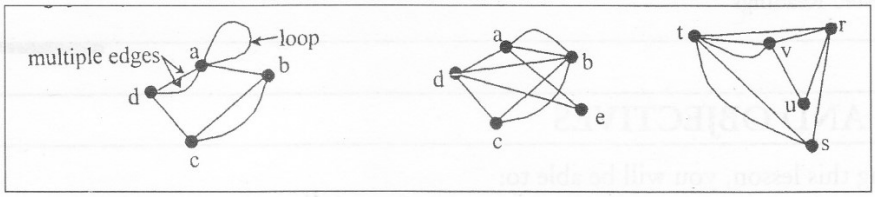


Figure 7.5 Figure 7.6

A graph with no multiple edges and no loops is a simple graph (e.g., Figure 7.1).

Two vertices joined by an edge are said to be adjacent.

Vertices are incident with the edges which joins them and an edge is incident with the vertices it joins.

Two graphs G and H are isomorphic if H can be obtained by relabelling the vertices of G, i.e. there is a one-one correspondence between the vertices and G and those of H such that the number of edges joining each pair of vertices in G is equal to the number of edges joining the corresponding pair of vertices in H (Figure 7.6 a $\leftrightarrow$ v, b $\leftrightarrow$ t, c $\leftrightarrow$ s, d $\leftrightarrow$ r, e $\leftrightarrow$ u).

A subgraph of G is a graph all of whose vertices and edges are vertices and edges of G (Figure 7.7 shows a series of subgraph of G).

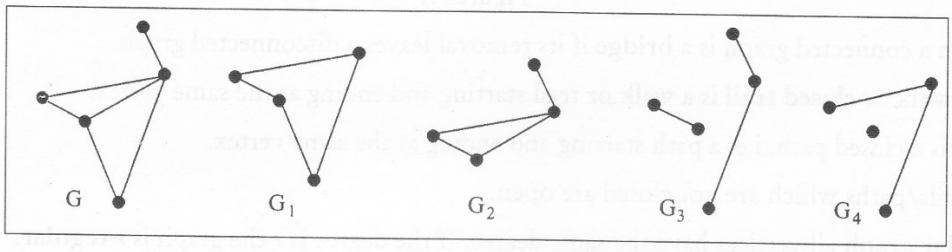


Figure 7.7

The **degree** of a vertex v is the number of edges incident with v, Loops count as 2.

The **degree sequence** of a graph G is the sequence obtained by listing, in ascending order with repeats, the degrees of the vertices of G (*e.g.* in Figure 7.7 the degree sequence of G is (1, 2, 2, 3, 4).)

The **Handshaking Lemma** states that the sum of the degrees of the vertices of a graph is equal to that twice the no. of edge this follow reality from the fact that each edge join two vertices necessarily distinct) and so contributes 1 to the degree of each of those vertices.

A **walk** of length k in a graph is a succession of k edges joining two vertices. NB Edges can occur more than once in a walk.

A trail is walk in which all the edges (but not necessarily all the vertices) are distinct.

A **path** is a walk in which all the edges and all the vertices are distinct.

So, in Figure 7.8, *abdcbde* is a walk of length 6 between *a* and *e*. It is not a trail (because edge *bd* is traversed twice). The walk *adcbde* is a trail length 5 between *a* and *e*. It is not a path (because vertex *d* is visited twice). The walk *abcde* is a path of length 4 between *a* and *e*.

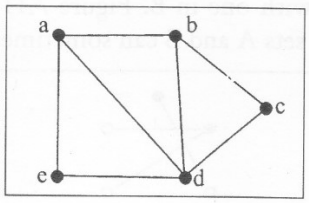


Figure 7.8

A **connected graph** has a path between every pair of vertices. A **disconnected graph** is a graph which is not connected. e.g., Figure 7.7, G and the subgraphs G_1 and G_2 are connected whilst G_3 and G_4 are disconnected.

Every disconnected graph can be split into a number of connected subgraphs called its components. It may not be immediately obvious that a graph is disconnected. For instance Figure 7.9 shows 3 graphs, each disconnected and comprising 3 components.

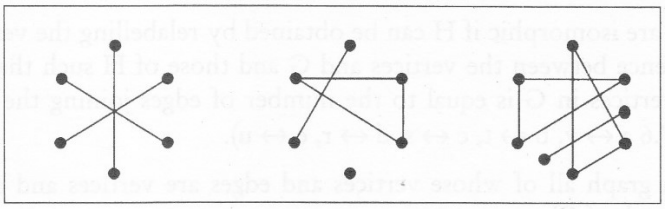


Figure 7.9

An edge in a connected graph is a **bridge** if its removal leaves a disconnected graph.

A **closed walk** or **closed trail** is a walk or trail starting and ending at the same vertex.

A **circuit** is a closed path, i.e. a path starting and ending at the same vertex.

Walks/trails/paths which are not closed are **open.**

In a **regular graph** all vertices have the same degree. If the degree is r the graph is ***r*-regular.**

If G is r-regular with m vertices it must have $1/2\ mr$ edges (from the Handshaking Theorem).

A **complete graph** is a graph in which every vertex is joined to every other vertex by exactly one edge. The complete graph with m vertices is denoted by K_m. K_m is $(m-1)$-regular and so has $1/2m\ (m-1)$ edges.

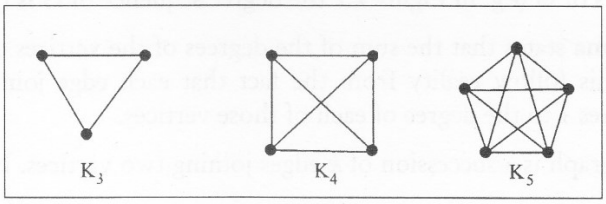


Figure 7.10

A null graph is a graph with no edges. The null graph with in vertices is denoted Nm is o-regular.

A cycle graph consines of a single cycle of vertices a edges. The cycle graph with m vertices is denoted cm.

A **bipartite graph** is a graph whose vertices can be split into two subsets A and B in such a way that every edge of G joins a vertex in A with one in B. Figure 7.11 shows some bipartite graphs. Notice that the allocation of the nodes to the sets A and B can sometimes be done in several ways.

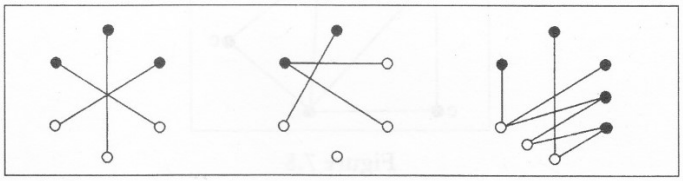


Figure 7.11

A **complete bipartite graph** is a bipartite graph in which every vertex in A is joined to every vertex in B by exactly one edge. The complete bipartite graph with r vertices in A and s vertices in B is denoted $K_{r,s}$. Figure 7.12 shows some complete bipartite graphs.

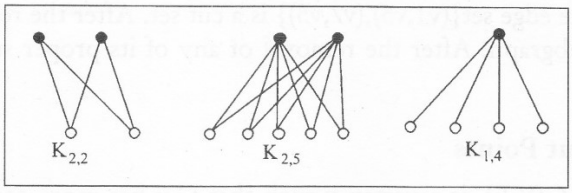


Figure 7.12

A **tree** is a connected graph with no cycles. In a tree there is just one path between each pair of vertices. Figure 7.13 shows some trees. Every tree is a bipartite graph. Start at any node, assign each node connected to that node to the other set, then repeat the process with those nodes!

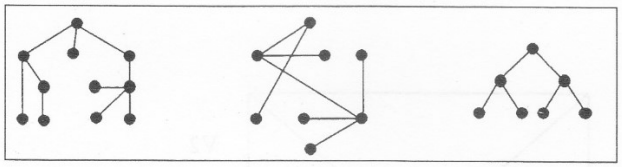


Figure 7.13

A **path graph** is a tree consisting of a single path through all its vertices. The path graph within vertices is denoted P_m. Figure 7.14 shows some path graphs.

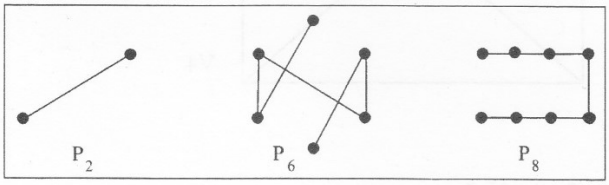
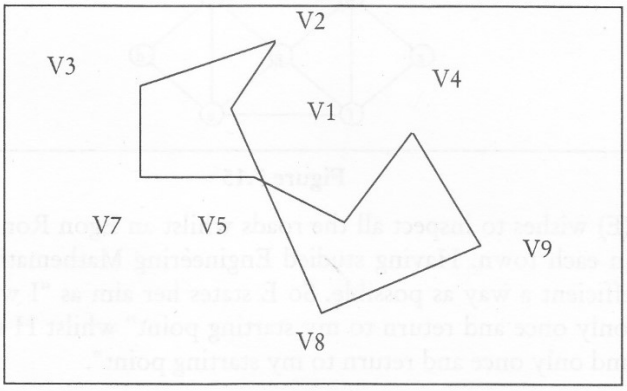


Figure 7.14

7.2.1 Cut Set

A Cut Set for connected graph G(V, E) is a smallest set of edges such that removal of the set, disconnects the graph whereas the removal of any proper subset of this set, left a connected subgraph.

Example 1: Consider the graph shown below. Determine the cutest for this graph.



Solution: For this graph, the edge set{(v1,v5),(v7,v5)} is a cut set. After the removal of this set, we have left with a disconnected subgraph. After the removal of any of its proper subset, we have left with a connected sub graph.

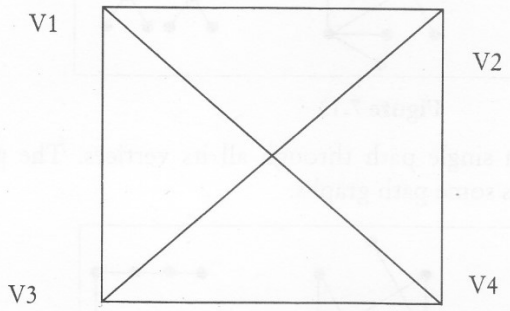
7.2.2 Cut Vertices or Cut Points

A cut point for the graph G(V,E), is a vertex v such that G-v has more connected components than Gor disconnected. It is obtained by deleting the vertex from the graph G and also deleting all the edges incident on v.

Example 2: Consider the graph below and determine the subgraphs

i. G-v1

ii. G-v3



Solution:

(i) The subgraph G-v1 is v2-v4-v3-v5

(ii) The subgraph G-v3 is v1-v2-v5-v3

7.3 EULERIAN AND HAMILTONIAN GRAPHS

Consider the following map of 7 towns and the roads connecting them.

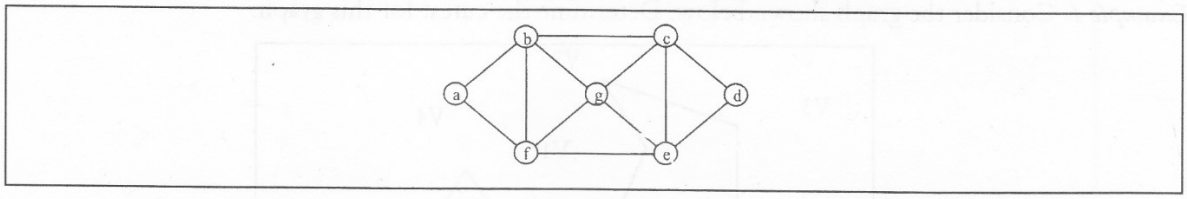


Figure 7.15

A highway engineer (E) wishes to inspect all the roads whilst an Egon Ronay inspector (H) wishes to dine in a restaurant in each town. Having studied Engineering Mathematics, each wishes to achieve their objective in as efficient a way as possible. So E states her aim as "I wish, if possible, to traverse every road once and only once and return to my starting point" whilst H says "I wish, if possible, to visit each town once and only once and return to my starting point".

A range of real problems give rise to versions of these two objectives, so graph theory has formalised them in the following way.

An **Eulerian graph** is a connected graph wishes contains a closed trail which includes every edge. The trail is called an **Eulerian trail.**

A **Hamiltonian graph** is a connected graph which contains a cycle which includes every vertex. The cycle is called an **Hamiltonian cycle.**

So E is saying "I want an Eulerian trail" and H is saying "I want a Hamiltonian cycle". Considering the map in Figure 7.15 as a graph, both an Eulerian trail and a Hamiltonian cycle exist, for instance *abcdecgefgbfa* and *abcdegfa* respectively. So the graph is both Eulerian and Hamiltonian.

In Figure 7.16 we see some more examples of Eulerian and Hamiltonian graphs.

- Graph 1 (the graph of the map in Figure 7.16) is both Eulerian and Hamiltonian.
- Graph 2 is Eulerian (*e.g. bcgefgb*) but not Hamiltonian.
- Graph 3 is Hamiltonian (*e.g. bcgefb*) but not Eulerian.
- Graph 4 is neither Eulerian nor Hamiltonian.

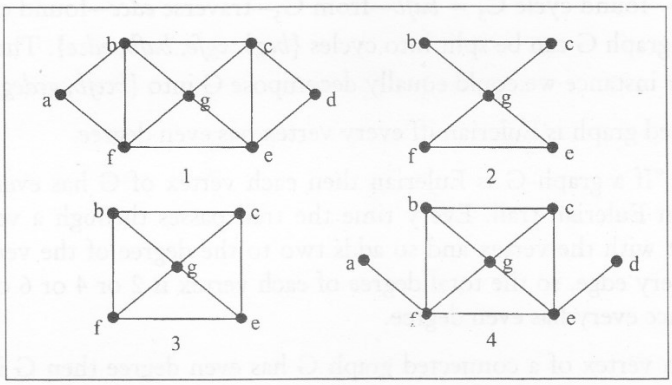


Figure 7.16

7.3.1 Eulerian Graphs

Theorem 1. A connected graph is Eulerian iff every vertex has even degree.

To prove this we first need a simpler theorem.

Theorem 2. If G is a graph all of whose vertices have even degree, then G can be split into cycles no two of which have an edge in common.

Proof. Let G be a graph all of whose vertices have even degree. Start at any vertex u and traverse edges in an arbitrary manner, without repeating any edge. Since every vertex has even degree it is always possible to find a different edge by which to leave a vertex. Since there is a finite number of vertices, eventually we must arrive at a vertex, v say, which we have already visited. The edges visited since the previous visit to v constitute a closed cycle, C_1 say. Remove all the edges in C_1 from the graph, leaving a subgraph G_1 say. Since we have removed a closed cycle of edges the vertices of G_1 will either have the same degree as the vertices of G or degrees 2 less than the equivalent vertices of G—either way G_1 is a graph all of whose vertices have been degree. We repeat the process with G_1, finding a cycle C_2, removing the edges in this cycle from G_1 and leaving G_2. Continue in this way until there are no edges

left. Then we have a set of cycles, C_1, C_2, C_3, ... which together include all edges of G and no two of which have an edge in common.

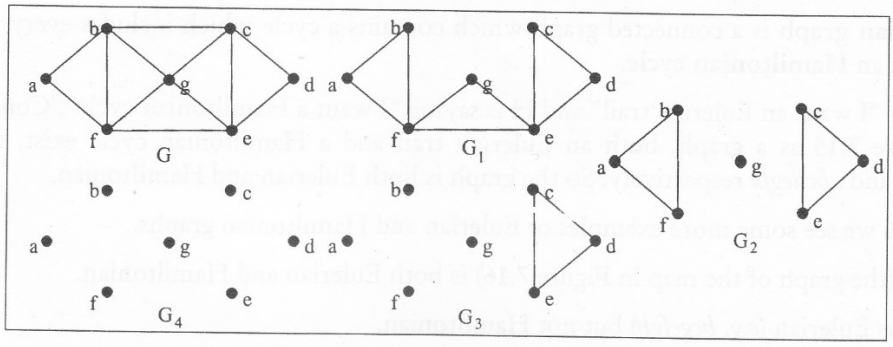


Figure 7.17

For instance, traverse *abcgb*—found cycle C_1 = *bcgb*—form G_1—traverse *degfe*—found cycle C_2 = *egfe*—form G_2—traverse *bafb*—found cycle C_3 = *bafb*—from G_3—traverse *edce*—found cycle C_4 = *edce*—from G_4—no edges left. The graph G can be split into cycles {*bcgb*, *egfe*, *bafb*, *edce*}. The decomposition is, of course, not unique. For instance we could equally decompose G into {*bcefb*, *gcdeg*, *abgfa*}.

Theorem 3. A connected graph is Eulerian iff every vertex has even degree.

Proof. First we prove "If a graph G is Eulerian then each vertex of G has even degree." Since G is Eulerian there exists an Eulerian trail. Every time the trail passes through a vertex it traverses two different edges incident with the vertex and so adds two to the degree of the vertex. Since the trail is Eulerian it traverses every edge, so the total degree of each vertex is 2 or 4 or 6 or ...., i.e. of the form $2k$, k = 1, 2, 3, ... Hence every has even degree.

Now we prove "If each vertex of a connected graph G has even degree then G is Eulerian." Since all the vertices of G have even degree, by theorem 2, G can be decomposed into a set of cycles no two of which have an edge in common. We will fit these cycles together to create an Eulerian trail. Start at any vertex of a cycle C_1. Travel round C_1 until we find a vertex which belongs also to another cycle, C_2 say. Travel round C_2 and then continue along C_1 until we reach the starting point. We have closed trail C_{12} which includes all the edges of C_1 and C_2. If this includes all the edges of G we have the required Eulerian trail, otherwise repeat the process starting at any point of C_{12} and travelling around it until we come to a vertex which is a member of another cycle, C_3 say. Travel round C_3 and then continue along C_{12} thus creating a closed trail C_{123}. Continue the process until we have a trail which includes all the edges of G and that will be an Eulerian trail in G.

We have proved both "If a graph G is Eulerian then each vertex of G has even degree" and "If each vertex of a connected graph G has even degree then G is Eulerian" and so we have "A connected graph is Eulerian iff every vertex has even degree".

A **semi-Eulerian graph** is a connected graph which contains an open trail which includes every edge. The trail is called a semi-Eulerian trail.

Theorem 4. A connected graph is semi-Eulerian iff exactly two vertices have odd degree.

Proof. (*a*) If G is a semi-Eulerian graph then there is an open trail which includes every edge. Let u and v be the vertices at the start and end of this trail. Add the edge uv to the graph. The graph is now

Eulerian and so every vertex has even degree by theorem 1. If the added edge is now removed the degrees of the vertices u and v edge are reduced by one and so are odd, the degrees of all other vertices are unaltered and are even. So if G is semi-Eulerian it has exactly two vertices of odd degree.

(*b*) Suppose G is a connected graph with exactly two vertices of odd degree. Let those two vertices be u and v. Add an edge uv to the graph. Now every vertex of G has even degree and so G is Eulerian. The Eulerian trail in G includes every edge and so includes the edge uv. Now remove the edge uv, then there is a trail starting at vertex u and ending at vertex v (or vice versa) which includes every edge. Hence if G is a connected graph with exactly two vertices of odd degree then G is semi Eulerian.

Hence we see that a connected graph is semi-Eulerian iff exactly two vertices have odd degree.

7.3.2 Hamiltonian Graphs

No simple necessary and sufficient condition for a graph to be Hamiltonian is known—this is an open area of research in graph theory.

But we can identify some classes of graph which are Hamiltonian. Obviously the cycle graph C_n is Hamiltonian for all n. The complete graph K_n is Hamiltonian for all $n \geq 3$.—obvious because, if the vertices are denoted $\{v_1, v_1, \ldots v_n\}$ then the path $v_1v_2v_3 \ldots v_nv_1$ is a Hamiltonian path.

If we add an edge to a Hamiltonian graph then the resulting graph is Hamiltonian—the Hamiltonian cycle in the original graph is also a Hamiltonian cycle in the enhanced graph. Adding edges may make a non-Hamiltonian graph into a Hamiltonian graph but cannot convert a Hamiltonian graph into a non-Hamiltonian one so graphs with high vertex degrees are more likely to be Hamiltonian then graphs with small vertex degrees. Ore's theorem is one possible more precise statement relating Hamiltonian graphs and their vertex degrees.

Ore's Theorem (stated without proof) : If G is a simple connected graph with n vertices ($n \geq 3$) then G is Hamiltonian if deg (v) + deg (w) $\geq n$ for every non-adjacent pair of vertices v and w.

If G is a simple connected graph with n vertices ($n \geq 3$) then G is Hamiltonian if deg (v) $\geq n/2$ for every vertex v. This follows from Ore's theorem. From this we can determine that all the complete bipartite graphs $K_{p,p}$ are Hamiltonian (the degree of every vertex is p, the graph has $2p$ vertices, hence deg (v) $\geq n/2$ ($=p$) for every vertex).

A **semi-Hamiltonian graph** is a connected graph which contains a path, but not a cycle, which includes every vertex. The path is called a **semi-Hamiltonian path.**

7.4 ISOMORPHISM

Two graphs G_1 and G_2 are said to be isomorphic to each other if there is a one to one correspondence between their vertices and between their edges so that the incidence relationship is maintained.

It means that if in graph G_1 an edge e_k is incident with vertices v_1 and v_j then in graph G_2 its corresponding edge e'_k must be incident with the vertices v'_i and v'_j that correspondent to the vertices v_i and v_j respectively.

The following two graphs G_1 and G_2 are isomorphic graphs.

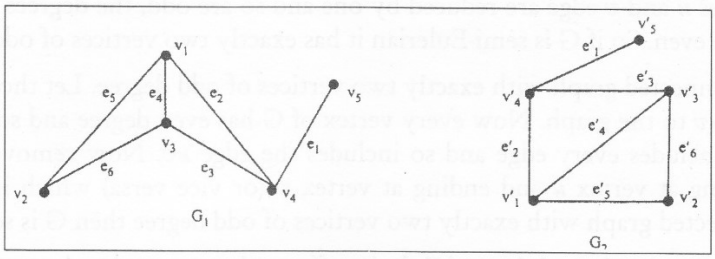


Figure 7.18

Vertices v_1, v_2, v_3, v_4 and v_5 in G_1 corresponds to v'_1 v'_2, v'_3, v'_4 and v'_5 respectively in G_2. Edges e_1, e_2, e_3, e_4, e_5 and e_6 in G_1 corresponds to e'_1, e'_2, e'_3, $e¢_4$, e'_5 and e'_6 respectively in G_2.

Here we can see that if any edge is incident with two vertices in G_1 then its corresponding edge shall be incident with the corresponding vertices in G_2 e.g. edges e_1, e_2 and e_3 are incident on vertex v_4, then the corresponding edges e'_1, e'_2 and e'_3 shall be incident on the corresponding vertex v'_4. In the way the incidence relationship shall be preserved.

In fact isomorphic graphs are the same graphs drawn differently. The difference is in the names or labels of their vertices and edges. The following two graphs are also isomorphic graphs in which vertices a, b, c, d, p, q, r, and s in G_1 corresponds to vertices v_1 v_2, v_3, v_4, v_5, v_6, v_7 and v_8 respectively in G_2 and edges e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11} and e_{12} in G_1 corresponds to edges e'_1, e'_2, e'_3, e'_4, e'_5, e^x_6, e'_7, e'_8, e'_9, e'_{10}, e'_{11} and e'_{12} in G_2 to preserve the incidence relationship.

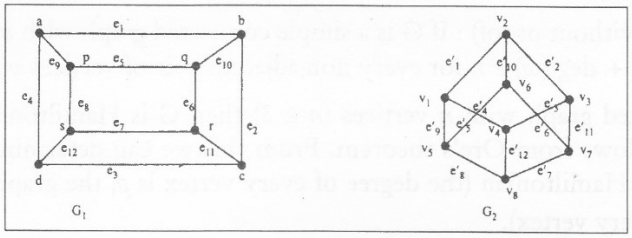


Figure 7.19

The incidence relationship between vertices and edges in between corresponding vertices and edges in G_2.

The following two graphs are not isomorphic

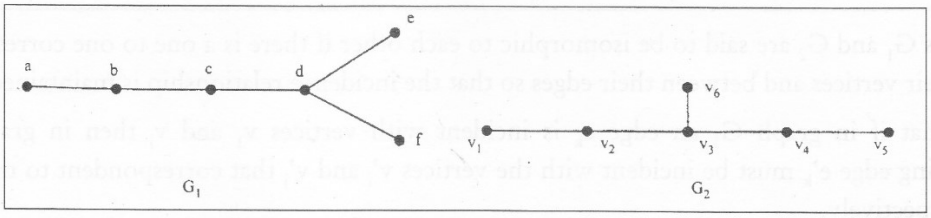


Figure 7.20

Vertex d in G_1 corresponds to vertex v_3 in G_2 as these are the only two vertices of degree 3.

In G_1, there are two pendant vertices adjacent to vertex d, while in G_2 there is only one pendant vertex adjacent to the corresponding vertex v_3. Thus the relationship of adjacency and incidence is not preserved and the two graphs are not isomorphic.

There is no simple and efficient criterion to identify isomorphic graphs.

7.4.1 Isomorphic Digraphs

Two digraphs are said to be isomorphic if,

(a) Their corresponding undirected graphs are isomorphic.

(b) Directions of the corresponding edges also agree.

The following two digraphs are not isomorphic because the directions of the two corresponding edges e_4 and e'_4 do not agree (although their corresponding undirected graphs are isomorphic).

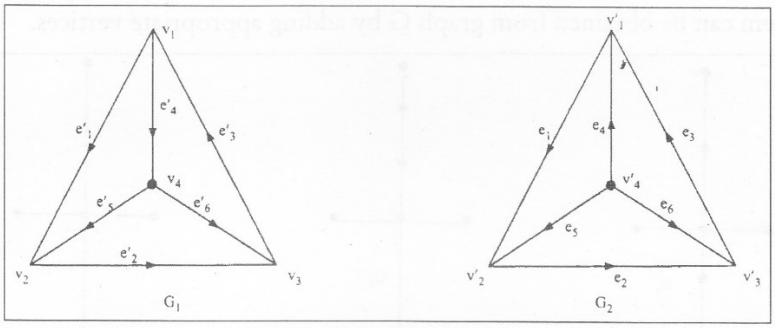


Figure 7.21

Isomorphism may also be defined as follows:

Isomorphism from a graph $G_1 = (V_1, E_1)$ to $G_2 = (V_2, E_2)$ is defined as mapping $f : V_1 \rightarrow V_2$ such that

(a) f is one-one and onto

(b) edge $v_i\ v_j \in E_1$ if and only if $f(v_1) \cdot f(v_j) \in E_2$ where $f(v_1)$ and $f(v_j)$ are the images of v_i and v_j respectively in graph G_2

7.4.2 Some Properties of Isomorphic Graphs

1. Number of vertices in isomorphic graphs is the same.
2. Number of edges in isomorphic graphs is also the same.
3. Each one of the isomorphic graphs has an equal number of vertices with a given degree.

This property is utilized in identifying two non-isomorphic graphs by writing down the degree sequence of their respective vertices.

Illustration: The degree sequence of graph G_1 is 4, 2, 2, 2, 2, 2, 1, 1 and that of G_2 is 3, 3, 2, 2, 2, 2, 1, 1 which are not the same. Therefore G_1 and G_2 are non-isomorphic

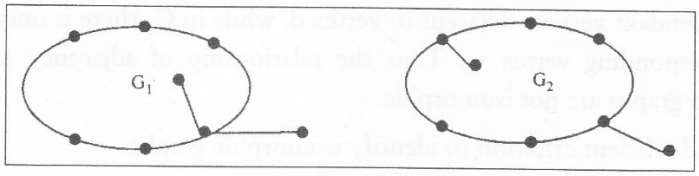


Figure 7.22

7.5 HOMEOMORPHISM GRAPHS

Let G_1 be a given graph. Another graph G_2 can be obtained from this graph by dividing an edge of G, with additional vertices.

Two graphs G_1 and G_2 are said to be Homeomorphic if these can be obtained from the same graph or isomorphic graphs by this method. Graphs G_1 and G_2, though not isomorphic, are Homeomorphic because each of them can be obtained from graph G by adding appropriate vertices.

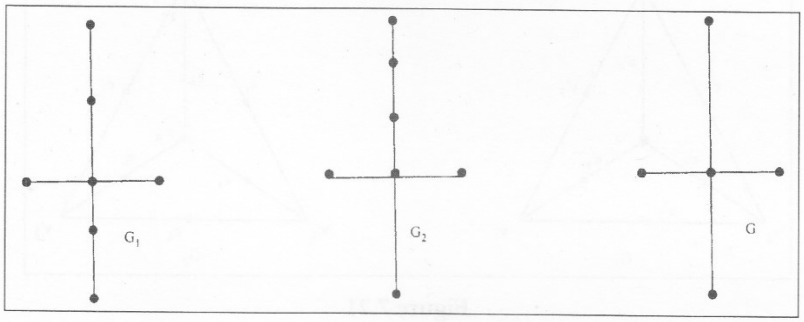


Figure 7.23

7.6 TREE STRUCTURES

A tree is a connected graph which has no cycler.

Trees are a relatively simple type of graph but they are also very important. Many applications use trees as a mathematical representation, e.g. decision trees in OR, some utility networks, linguistic analysis, family trees, organisation trees.

Trees have a number of special properties as follows:

(*a*) It is obvious, from the constructive process of building all possible trees step by step from the simplest tree (one vertex, no edges) that a tree with n vertices has exactly $n - 1$ edges.

(*b*) When a new vertex and edges is added to a tree, no cycle is created (since the new edge joins an existing vertex to a new vertex) and the tree remains connected.

(*c*) There is exactly one path from any vertex in a tree to any other vertex—if there were two or more paths between any two vertices then the two paths would form a cycle and the graph would not be a tree.

(*d*) Because there is exactly one path between any two vertices then there is one (and only one) edge joining any two adjacent vertices. If this edge is removed, the graph is no longer connected (and so is not a tree). So the removal of any edge from a tree disconnects the graph.

(*e*) Since there is a path between any two vertices, if an edge is added to the tree joining two existing vertices then a cycle is formed comprising the existing path between the two vertices together with the new edge.

All these properties can be used to define a tree. If T is a graph with n vertices then the following are all equivalent definitions of a tree:

- T is connected and has no cycles.
- T has $n - 1$ edges and has no cycles.
- T is connected and has $n - 1$ edges.
- Any two vertices of T are connected by exactly one path.
- T is connected and the removal of any edge disconnects T.
- T contains no cycles but the addition of any new edge creates a cycle.

A **spanning tree** in a connected graph G is a subgraph which includes every vertex and a tree. For instance Figure 7.24 below shows the complete graph K_5 and several possible spanning trees. Large and complex graphs may have very many spanning trees.

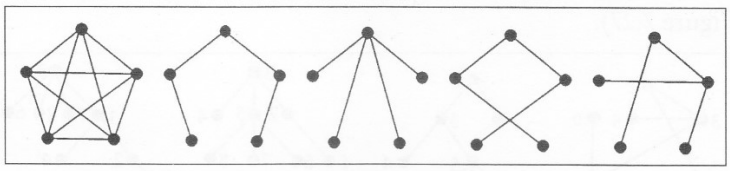


Figure 7.24

A spanning tree may be found by the building-up method or the cutting-down method. The building-up algorithm is "Select edges of the graph, one by one, such a way that no cycles are created; repeating until all vertices are included" and the cutting-down method is "Choose any cycle in the graph and remove one of its edges; repeating until no cycles remain."

For instance, in Figure 7.25 below a spanning tree in the graph G is built up by selecting successively edges *ab* (1st diagram), then *ce* (2nd diagram), then *bg*, then *ge* (3rd diagram), then *gf* and finally *de* (final diagram).

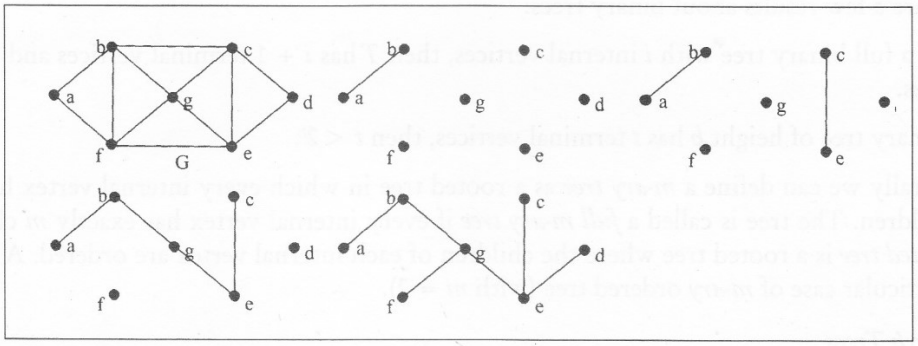


Figure 7.25

In Figure 7.26, a spanning tree in the graph G of Figure 7.25 is derived by the cutting down method, by successively finding cycles and removing edges. First cycle is *abcdefa*—remove *bc* (1st diagram).

Cycle *bgfb*—remove *fb* (2nd diagram). Cycle *gedcg*—remove *cd*. Cycle *cgec*—remove *gc*. Cycle *gfabg*—removed *ab*. Cycle *gfeg*—remove *fe*. No more cycles so this is a spanning tree.

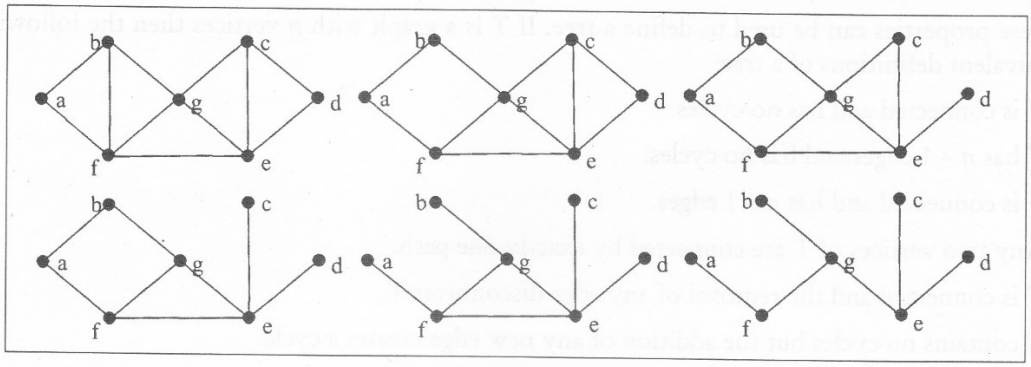


Figure 7.26

A **rooted tree** is a tree in which one vertex is selected as a root and all other edges branch out from the root vertex. Any given tree can be drawn as a rooted tree in a variety of ways depending on the choice of root vertex (see Figure 7.27).

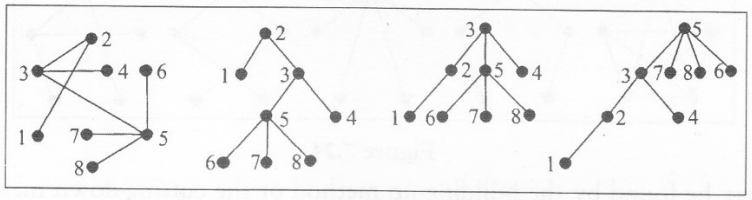


Figure 7.27

Binary Trees

A *binary tree* is a rooted tree in which each vertex has at most two children, designated as *left child* and *right child*. If a vertex has one child, that child is designated as either a left child or a right child, but not both. A *full binary tree* is a binary tree in which each vertex has exactly two children or none. The following are a few results about binary trees:

1. If T is a full binary tree with i internal vertices, then T has $i + 1$ terminal vertices and $2i + 1$ total vertices.
2. If a binary tree of height h has t terminal vertices, then $t < 2^h$.

More generally we can define a *m-ary tree* as a rooted tree in which every internal vertex has no more than m children. The tree is called a *full m-ary tree* if every internal vertex has exactly m children. An *ordered rooted tree* is a rooted tree where the children of each internal vertex are ordered. A binary tree is just a particular case of *m-ary* ordered tree (with $m = 2$).

Binary Search Trees

Assume S is a set in which elements (which we will call "data") are ordered; e.g., the elements of S can be numbers in their natural order, or strings of alphabetic characters in lexicographic order. A *binary search tree* associated to S is a binary tree T in which data from S are associate with the vertices of T so

that, for each vertex u in T, each data item in the left subtree of v is less than the data item in v, and each data item in the right subtree of v is greater than the data item in v.

Example: Figure 7.28 below, contains a binary search tree for the set S = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. In order to find a element we start at the root and compare it to the data in the current vertex (initially the root). If the element is greater we continue through the right child, if it is smaller we continue through the left child, if it is equal we have found it. If we reach a terminal vertex without founding the element, then that element is not present in S.

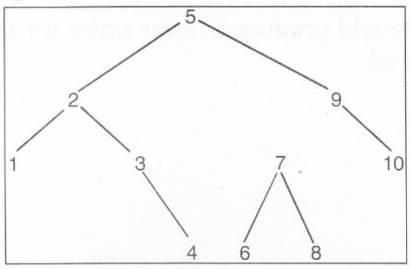


Figure 7.28: Binary Search Tree

Making a Binary Search Tree. We can store data in a binary search tree by randomly choosing data from *S* and placing it in the tree in the following way: The first data chosen will be the root of the tree. Then for each subsequent data item, starting at the root we compare it to the data in the current vertex v. If the new data item is greater than the data in the current vertex then we move to the right child, if it is less we move to the left child. If there is no such child then we create one and put the new data in it. For instance, the tree in Figure 7.29 below has been made from the following list of words choosing them in the order they occur: "IN A PLACE OF LA MANCHA WHOSE NAME I DO NOT WANT TO REMEMBER".

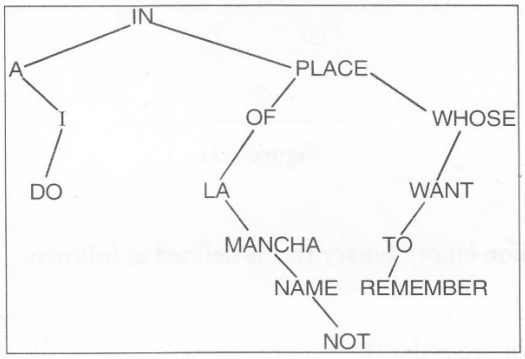


Figure 7.29: Another Binary Search Tree

Traversal of a Binary Tree

Tree traversal is one of the most common operations performed on tree data structures. It is a way in which each node in the tree is visited exactly once in a systematic manner. There are many applications that essentially require traversal of binary trees. For example, a binary tree could be used to represent an arithmetic expression as shown in Figure 7.30.

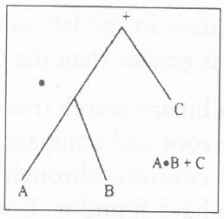


Figure 7.30

The full binary tree traversal would produce a linear order for the nodes in a binary tree. There are three ways of binary tree traversal,

(i) In-order traversal

(ii) Pre-order traversal

(iii) Post-order traversal

Inorder Traversal

The in-order traversal of a non-empty tree is defined as follows:

(i) Traverse the left subtree inorder (L).

(ii) Visit the root node (N)

(iii) Traverse the right subtree in order (R).

Illustration: In Figure 7.31 inorder traversal of a binary tree is DBFEGAC.

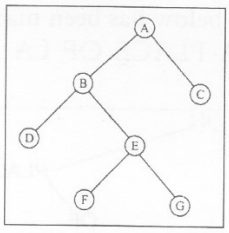


Figure 7.31

Pre-order Traversal

The pre-order traversal of a non-empty binary tree is defined as follows:

(i) Visit the root node (N).

(ii) Traverse the left subtree in pre-order (L)

(iii) Traverse the right subtree in pre-order (R)

Illustration: In Figure 7.31 the preorder traversal of a binary tree is ABDEFGC.

Postorder Traversal

The postorder traversal of non-empty binary tree is defined as follows:

(i) Traverse the left subtree in postorder (L).

(ii) Traverse the right subtree in postorder (R).

(iii) Visit the root node (N).

Illustration: In Figure 7.31 the postorder traversal of a binary tree is DFGEBCA.

Level of a Vertex in a Full Binary Tree

In a binary tree the distance of a vertex v_i from the root of the tree is called the level of v_i and is denoted by l_i. Thus level of the root is zero. Levels of the various vertices in the following tree have been denoted by numbers written adjacent to the respective vertices.

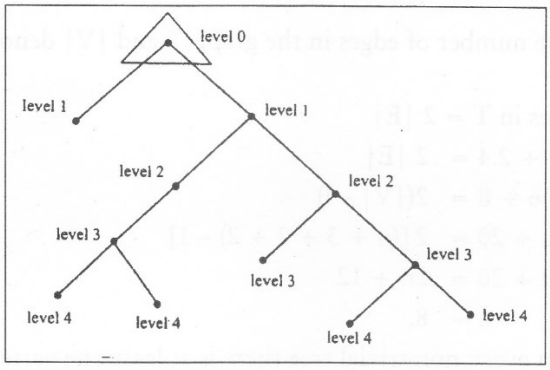


Figure 7.32: Twelve Vertex Binary Tree of Level 4

Number of Vertices of Different levels in a Binary Tree

In a binary tree there will be two edges adjacent to the root vertex v_0. Let these edges be v_0u_1 and v_0v_1. Levels of each of the vertices u_1 and v_1 is 1. So maximum number of vertices of level 0 is $1(=2^0)$ and maximum number of vertices with level 1 is $=2^1$.

Again there can be either 0 or 2 edges adjacent to each of the vertices u_1 and v_1. Let these edges be u_1u_2, u_1u_3, v_1v_2 and v_1v_3. Levels of each of the four vertices u_2, u_3, v_2, v_3 is 2. So maximum number of vertices of level 2 is $4(= 2^2)$. In a similar way the levels of the 8 vertices that will be obtained by adding two edges to each of the four vertices u_2, u_3, v_2, v_3 shall be 3. So maximum number of vertices each of level 3 is $8(=2^3)$. Not more than two edges can be added to any of the vertices so obtained to keep the degree of that vertex as 3.

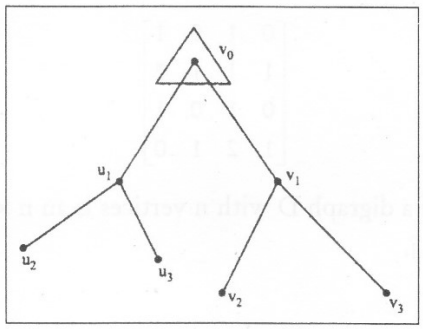


Figure 7.33

Proceeding in this way we see that the maximum number of vertices in a n level binary tree at levels 0, 1, 2, 3,... shall be 2^0. 2^1, 2^2, 2^3,... respectively.

whose sum = $2^0 + 2^1 + 2^2 + 2^3 + \ldots + 2^n$.

The maximum level of any vertex in a binary tree (denoted by l_{max}) is called *height of the tree.* The minimum possible height of an n vertex binary tree is $[\log_2(n+1) - 1]$ which is equal *to* the smallest integer $\geq [\log_2 (n + 1) - 1]$ and Max. $l_{max} = \frac{n-1}{2}$

Example 3: If a tree T has n vertices of degree 1, 3 vertices of degree 2, 2 vertices of 3 and 2 vertices of degree 4 find the value of n.

Solution: Let $|E|$ denote the number of edges in the graph T and $|V|$ denote the number of vertices in the same graph T.

Sum of degrees of all vertices in T = $2|E|$

$$n.1 + 3.2 + 2.3 + 2.4 = 2|E|$$

or $$n + 6 + 6 + 8 = 2(|V| - 1)$$

or $$n + 20 = 2[(n + 3 + 2 + 2) - 1]$$

or $$n + 20 = 2n + 12$$

or $$n = 8.$$

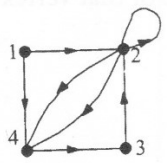
Theorem 5: To prove that in every non-trivial tree there is at least one vertex of degree one.

Proof: Let us start at vertex v_1. If $d(v_1) = 1$, then the theorem is already proved. If $d(v_1) > 1$, then we can move to a vertex say v_2 that is adjacent to v_1. Now if $d(v_2) > 1$, again move to another vertex say v_3 that is adjacent to v_2. In this way we can continue to produce a path v_1, v_2, v_3 .... (without repetition of any vertex, in order to avoid formation of circuit as the graph is a tree). As the graph is finite, this path must end at some vertex whose degree shall be one because we shall only enter this vertex and cannot exit from it.

7.7 MATRIX REPRESENTATIONS

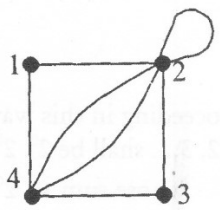
The adjacency matrix A(G) of a graph G with n vertices is an $n \times n$ matrix with a_{ij} being the number of edges joining of edges joining the vertices i and j.

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 2 \\ 0 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{bmatrix}$$



The adjacency matrix A(D) of a digraph D with n vertices is an n × n matrix with a_{ij} being the number of arcs from vertex i to vertex j.

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



What is the number of walks of length 2 from vertex i to vertex j? There are, for instance, two walks of length 2 from 1 to 4. And so on. The matrix of these is

$$\begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 1 & 2 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

and we see that this is just $A(D)^2$. In fact this generalises.

Theorem 6. The number of walks of length k from vertex i to vertex j in a digraph D with n vertices is given by the ijth element of the matrix Ak where A is the adjacency matrix of the digraph.

Proof. We prove this result by mathematical induction.

Assume that the result is true for $k \leq K - 1$. We will show that it is then also true for K. Consider any walk from vertex i to vertex j of length K. Such a walk consists of a walk of length $K - 1$ from vertex i to a vertex p which i adjacent to vertex j followed by a walk of length 1 from vertex p to vertex j. The number of such walks is $[A^K - 1]_{ip} \times A_{pj}$. The total number of walks of length k from vertex i to vertex j will then be the sum of the walks through any p, i.e. $\sum_{p=1}^{n} \left[A^{K-1}\right]_p A_{pj}$ but this is just the expression for the ij'th element of the matrix $A^{K-1}A = A^K$ so the result is true for $k = K$. But the result is certainly true for walks of length 1, i.e. $k = 1$, because that is the definition of the adjacency matrix A. Hence the theorem is true for all k.

Now we can create a method of automating the determination of whether a given digraph is strongly connected or not. For the graph to be strongly connected there must be paths (of any length) from every vertex to every other vertex. The length of these paths cannot exceed $n - 1$ where n is the number of vertices in the graph (otherwise a path would be visiting at least one vertex at least twice). So the number of paths from a vertex i to a vertex j of any length from 1 to $n - 1$ is the sum of the ijth elements of the matrices A, A^2, A^3, ... A^{n-1}. So we introduce the matrix $B = A + A^2 + A^3 + \ldots A^{n-1}$ whose element B_{ij} represent the number of paths between all the vertices. If any off-diagonal element of B is zero then there are no paths from some vertex i to some vertex j. The digraph is strongly connected provided all the off-diagonal elements are non-zero!

Theorem 7. If A is the adjacency matrix of a digraph D with n vertices and B is the matrix $B = A + A^2 + A^3 + \ldots A^{n-1}$ then D is strongly connected iff each nondiagonal element of B is greater then 0.

Proof. To prove this theorem we must show both "if each non diagonal element of B is greater then 0 then D is strongly connected" and "if D is strongly connected then each non diagonal element of B is greater then 0".

Firstly, let D be a digraph and suppose that each non-diagonal element of the matrix $B > 0$, i.e. $B_{ij} > 0$ for all $i \neq j$. Then $[Ak]_{ij} > 0$ for some $k \in [1, n - 1]$, i.e. there is a walk of some length k between 1 and n - 1 from every vertex i to every vertex j. So the digraph is strongly connected.

Secondly, suppose the digraph is strongly connected. Then, by definition, there is a path from every vertex i to every j. Since the digraph has n vertices the path is of length no more than $n - 1$.

Hence, for all $i \neq j$, $[Ak]_{ij} > 0$ for some $k \leq n - 1$. Hence, for all $i \neq j$, $B_{ij} > 0$.

Returning to the example of the digraph D in figure 4.2 above we have

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, A^2 = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 1 & 2 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 0 \end{bmatrix}, A^3 = \begin{bmatrix} 0 & 2 & 2 & 2 \\ 0 & 3 & 2 & 2 \\ 0 & 1 & 2 & 2 \\ 0 & 1 & 0 & 2 \end{bmatrix},$$

$$B = A + A^2 + A^3 = \begin{bmatrix} 0 & 4 & 3 & 5 \\ 0 & 5 & 4 & 6 \\ 0 & 3 & 2 & 4 \\ 0 & 2 & 1 & 2 \end{bmatrix}$$

so the graph is not strongly connected because we cannot get from vertex 2, vertex 3 or vertex 4 to vertex 1. Inspecting the digraph that is intuitively obviously! But, of course, this method is valid for large and complex digraphs which are less amenable to ad hoc analysis.

If we are only interested in whether there is at least one path from vertex i to vertex j (rather than wanting to know how many paths), then all of this can also be done using Boolean matrices. In this case the Boolean matrix (in which $a_{ij} = 1$ if there is at least one arc from vertex i to vertex j and 0 otherwise) of the graph is

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

and the calculation of A^2, A^3 etc. is done using Boolean arithmetic (so $\times$ is replaced by $\wedge$ and $+$ by $\vee$) so

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, A^2 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}, A^3 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix},$$

$$R = A \cup A^2 \cup A^3 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

A^2 is a matrix in which $(A^2)_{ij} = 1$ if there is at least one walk of length 2 from vertex i to vertex j and 0 otherwise and R is a matrix in which $R_{ij} = 1$ if there is at least one walk of length less than n from vertex i to vertex j and 0 otherwise. R is called the reachability matrix. In general R is easier and quicker to compute than B because it uses Boolean arithmetic. But, better, there is an even faster method called Warhsall's algorithm.

Let D = (V, A) where V = $\{v_1, v_2, \ldots v_n\}$ and there are no multiple arcs. Warshall's algorithm computes a sequence of $n + 1$ matrices, $M_0, M_1, \ldots M_n$. For each $k \in [0 \ldots n]$, $[M_k]_{ij} = 1$ iff there is a path in G from v_i to v_j whose interior nodes come only from $\{v_1, v_2, \ldots v_k\}$. Warshall's algorithm is

```
procedure Warshall (var M:n × n matrix);
{initially M = A, the adjacency matrix of G}
begin
    for k:= 1 to n do
        for i:= 1 to n do
            for j:= 1 to n do
                M[i, j] := M[i, j]  (M[i, k] ∧ M[k, j]);
end;
```

Example 4: Find the reachability matrix of the digraph G using Warhall's algorithm.

Lets look at this in detail. The following table shows the derivation of the elements of M_1 from those of M_0. Notice that we use the updated elements of M as soon as they are available!

k	i	j		
1	1	1	$M_1[1,1] := M_0[1,1] \vee (M_0[1,1] \wedge M_0[1,1])$	$M_1[1,1] := 0 \vee (0 \wedge 0) = 0$
1	1	2	$M_1[1,2] := M_0[1,2] \vee (M_1[1,1] \wedge M_0[1,2])$	$M_1[1,2] := 1 \vee (0 \wedge 1) = 1$
1	1	3	$M_1[1,3] := M_0[1,3] \vee (M_1[1,1] \wedge M_0[1,3])$	$M_1[1,3] := 0 \vee (0 \wedge 0) = 0$
1	1	4	$M_1[1,4] := M_0[1,4] \vee (M_1[1,1] \wedge M_0[1,4])$	$M_1[1,4] := 1 \vee (0 \wedge 1) = 1$
1	2	1	$M_1[2,1] := M_0[2,1] \vee (M_0[2,1] \wedge M_1[1,1])$	$M_1[2,1] := 0 \vee (0 \wedge 0) = 0$
1	2	2	$M_1[2,2] := M_0[2,2] \vee (M_1[2,1] \wedge M_1[1,2])$	$M_1[2,2] := 1 \vee (0 \wedge 1) = 1$
1	2	3	$M_1[2,3] := M_0[2,3] \vee (M_1[2,1] \wedge M_1[1,3])$	$M_1[2,3] := 0 \vee (0 \wedge 0) = 0$
1	2	4	$M_1[2,4] := M_0[2,4] \vee (M_1[2,1] \wedge M_1[1,4])$	$M_1[2,4] := 1 \vee (0 \wedge 1) = 1$
1	3	1	$M_1[3,1] := M_0[3,1] \vee (M_0[3,1] \wedge M_1[1,1])$	$M_1[3,1] := 0 \vee (0 \wedge 0) = 0$
1	3	2	$M_1[3,2] := M_0[3,2] \vee (M_1[3,1] \wedge M_1[1,2])$	$M_1[3,2] := 1 \vee (0 \wedge 1) = 1$
1	3	3	$M_1[3,3] := M_0[3,3] \vee (M_1[3,1] \wedge M_1[1,3])$	$M_1[3,3] := 0 \vee (0 \wedge 0) = 0$
1	3	4	$M_1[3,4] := M_0[3,4] \vee (M_1[3,1] \wedge M_1[1,4])$	$M_1[3,4] := 0 \vee (0 \wedge 1) = 0$
1	4	1	$M_1[4,1] := M_0[4,1] \vee (M_0[4,1] \wedge M_1[1,1])$	$M_1[4,1] := 0 \vee (0 \wedge 0) = 0$
1	4	2	$M_1[4,2] := M_0[4,2] \vee (M_1[4,1] \wedge M_1[1,2])$	$M_1[4,2] := 0 \vee (0 \wedge 1) = 0$
1	4	3	$M_1[4,3] := M_0[4,3] \vee (M_1[4,1] \wedge M_1[1,3])$	$M_1[4,3] := 1 \vee (0 \wedge 0) = 1$
1	4	4	$M_1[4,4] := M_0[4,4] \vee (M_1[4,1] \wedge M_1[1,4])$	$M_1[4,4] := 0 \vee (0 \wedge 1) = 0$

$$M_0 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, M_1 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, M_2 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix},$$

$$M_3 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}, M_4 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

The major advantage of Warshall's algorithm is that it is computationally more efficient. The number of operations for a digraph with n vertices is $O(n^3)$ whilst the number of operations required to compute R from $R = A \cup A^2 \cup A^3 \cup ... \cup A^{n-1}$ is $O(n^4)$.

To see this proceed thus:

Let an and take m and an or s msec. For the power method, to compute each element of A^2 takes $n*m + (n - 1)*s$. A has n^2 elements to compute the whole of A^2 takes $n^2*(n*m + (n - 1)*s)$. To compute A^3 we can multiply A times A^2 and this just takes the same time as computing A^2. For a graph with n vertices there are $n - 2$ matrix multiplications and then an or of $n - 1$ matrices, so the total time is $(n - 1)*(n^2*(n*m + (n - 1)*s)) + n^2*(n - 2)*s = (n^4 - 2*n^3)*(m + s) = O(n^4)$.

For Warshall's algorithm, each basic operation takes one and and one or. The triple loop means there are n^3 basic operations so the total time taken is $n^3*(m + s) = O(n^3)$.

Overall therefore, for a small graph we can compute the reachability matrix by hand using the power method relatively quickly. But if we are looking at a larger graph (think of 50 vertices and then consider the situation for 500 or 5000 vertices), we need computational help and Warshall's algorithm will take 1/n of the compute time taken by the power method.

Check Your Progress

Which of the graphs K_8, $K_{4,4}$, C_6, $K_{2,5}$ are Eulerian graphs (use theorem 1 to decide). For those which are Eulerian, find an Eulerian trail.

7.8 LET US SUM UP

The word *graph* refers to a specific mathematical structure usually represented as a diagram consisting of points joined by lines. In applications, the points may, for instance, correspond to chemical atoms, towns, electrical terminals or anything that can be connected in pairs. The lines may be chemical bonds, roads, wires or other connections. Applications of graph theory are found in communications, structures and mechanisms, electrical networks, transport systems, social networks and computer science.

A walk of length k in a graph is a succession of k edges joining two vertices. NB Edges can occur more than once in a walk.

A **path** is a walk in which all the edges and all the vertices are distinct.

A **tree** is a connected graph with no cycles. In a tree there is just one path between each pair of vertices.

7.9 KEYWORDS

Graph: A graph is a mathematical structure comprising a set of vertices, V, and a set of edges, E, which connect the vertices.

Digraph: A digraph is a diagram consisting of points, called vertices, joined by directed lines, called arcs.

Subgraph: A subgraph of G is a graph all of whose vertices and edges are vertices and edges of G.

Degree: The degree of a vertex v is the number of edges incident with v, Loops count as 2.

Walk: A walk of length k in a graph is a succession of k edges joining two vertices.

Path: A path is a walk in which all the edges and all the vertices are distinct.

Eulerian Graph: An Eulerian graph is a connected graph wishes contains a closed trail which includes every edge.

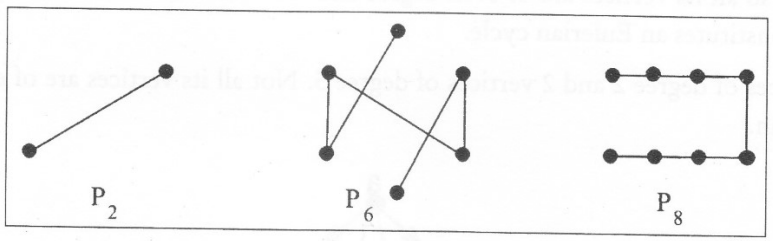
Hamiltonian Graph: A Hamiltonian graph is a connected graph which contains a cycle which includes every vertex.

7.10 QUESTIONS FOR DISCUSSION

1. Draw the graphs whose vertices and edges are as follows. In each case say if the graph is a simple graph.

   (a) V = $\{u, v, w, x\}$, E = $\{uv, vw, wx, vx\}$

   (b) V = {1, 2, 3 4, 5, 6, 7, 8}, E = {12, 22, 23, 34, 35, 67, 68, 78}



   (c) V = $\{n, p, q, r, s, t\}$, E = $\{np, nq, nt, rs, rt, st, pq\}$

2. Which of graph B, C and D are isomorphic to graph A? State the corresponding vertices in each isomorphic pair.

3. Which of the graphs P, Q, ... W are subgraphs of G?

4. Write down the degree sequence of each of the following graphs:

5. Graphs G_1 and G_2 have the same degree sequence—and they necessarily isomorphic? If your answer is no, give a counter example.
6. Graphs G_1 and G_2 are isomorphic. Do they necessarily have the same degree sequence? If your answer is no, give a counter example.
7. Use the Handshaking Lemma to prove that the number of vertices of odd degree in any graph must be even.
8. Draw the graphs K_5, N_5 and C_5.
9. Draw the complete bipartite graphs $K_{2,3}$, $K_{3,5}$, $K_{4,4}$. How many edges and vertices does each graph have? How many edges and vertices would you expect in the complete bipartite graphs $K_{r,}$
10. By finding a Hamiltonian cycle show that the complete bipartite graph $K_{3,3}$ is Hamiltonian. Show that the complete bipartite graph $K_{2,4}$ is not Hamiltonian. What condition on r and s is necessary for the complete bipartite graph $K_{r,s}$ to be Hamiltonian?
11. Draw the graphs corresponding to the following adjacency matrices.

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 2 & 0 & 1 & 1 \\ 2 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 2 \\ 1 & 1 & 0 & 2 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

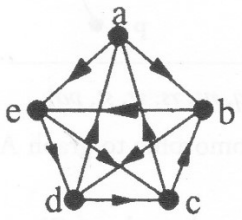
Check Your Progress: Modal Answers

K_8 is 7-regular, so all its vertices are of odd degree and it is not Eulerian.

$K_{4,4}$ is 4-regular, so all its vertices are of even degree and it is Eulerian. An Eulerian trail, referred to the diagram below, is *aebfcgdhcedfagbha*.

C_6 is 2-regular, so all its vertices are of even degree and it is Eulerian. Since it's a cycle graph the whole graph constitutes an Eulerian cycle.

$K_{2,5}$ has 5 vertices of degree 2 and 2 vertices of degree 5. Not all its vertices are of even degree so it is not Eulerian.



7.11 SUGGESTED READINGS

Anuranjan Misra, *Discrete Mathematics*, Acme Learning pvt ltd.

Richard Johnsonbaugh, *Discrete Mathematics*, Prentice Hall

V. K. Balakrishnan, *Introductory Discrete Mathematics*, Courier Dover Publications,

R. C. Penner, *Discrete Mathematics: Proof Techniques and Mathematical Structures*, World Scientific, 1999

Mike Piff, *Discrete Mathematics: An Introduction for Software* Engineers, Cambridge University Press, 1991

UNIT V

LESSON

8

GRAPH COLOURING

CONTENTS

8.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Understand graph colouring
- Understand partitioning and covering
- Discuss planer graphs and directed graphs
- Discuss chromatic ploynomial in graphs

8.1 INTRODUCTION

In graph theory, graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called "colors" to elements of a graph subject to certain constraints. In its simplest form, it is a way of coloring the vertices of a graph such that no two adjacent vertices share the same color; this is called a vertex coloring. Similarly, an edge coloring assigns a color to each edge so that no two adjacent edges share the same color, and a face coloring of a planar graph assigns a color to each face or region so that no two faces that share a boundary have the same color.

Vertex coloring is the starting point of the subject, and other coloring problems can be transformed into a vertex version. For example, an edge coloring of a graph is just a vertex coloring of its line graph, and a face coloring of a planar graph is just a vertex coloring of its planar dual. However, non-vertex coloring problems are often stated and studied as is. That is partly for perspective, and partly because some problems are best studied in non-vertex form, as for instance is edge coloring.

The convention of using colors originates from coloring the countries of a map, where each face is literally colored. This was generalized to coloring the faces of a graph embedded in the plane. By planar duality it became coloring the vertices, and in this form it generalizes to all graphs. In mathematical and computer representations it is typical to use the first few positive or nonnegative integers as the "colors". In general one can use any finite set as the "color set". The nature of the coloring problem depends on the number of colors but not on what they are.

Graph coloring enjoys many practical applications as well as theoretical challenges. Beside the classical types of problems, different limitations can also be set on the graph, or on the way a color is assigned, or even on the color itself. It has even reached popularity with the general public in the form of the popular number puzzle Sudoku. Graph coloring is still a very active field of research.

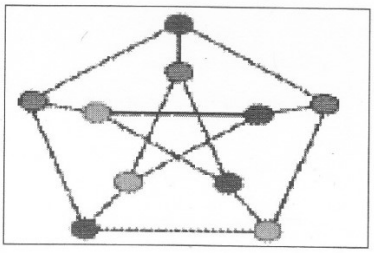


Figure 8.1: A proper vertex colouring of the Petersen graph with 3 colors, the minimum number possible

8.2 DEFINITION AND TERMINOLOGY

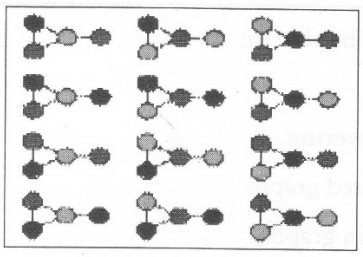


Figure 8.2: This graph can be 3-coloured in 12 different ways.

8.2.1 Vertex Colouring

When used without any qualification, a **coloring** of a graph is almost always a proper vertex coloring, namely a labelling of the graph's vertices with colors such that no two vertices sharing the same edge have the same color. Since a vertex with a loop could never be properly coloured, it is understood that graphs in this context are loopless.

The terminology of using *colors* for vertex labels goes back to map coloring. Labels like *red* and *blue* are only used when the number of colors is small, and normally it is understood that the labels are drawn from the integers {1, 2, 3, ...}.

A coloring using at most k colors is called a (proper) ***k*-coloring.** The smallest number of colors needed to color a graph G is called its **chromatic number,** $\chi(G)$. A graph that can be assigned a (proper) k-coloring is ***k*-colorable,** and it is ***k*-chromatic** if its chromatic number is exactly *k*. A subset of vertices assigned to the same color is called a *color class*, every such class forms an independent set. Thus, a *k*-coloring is the same as a partition of the vertex set into *k* independent sets, and the terms *k-partite* and *k-colorable* have the same meaning.

8.2.2 Chromatic Polynomial

In the Figure 8.3 all nonisomorphic graphs on 3 vertices and their chromatic polynomials. The empty graph E_3 (red) admits a 1-coloring, the others admit no such colorings. The green graph admits 12 colorings with 3 colors.

The **chromatic polynomial** counts the number of ways a graph can be colored using no more than a given number of colors. For example, using three colors, the graph in the image to the right can be colored in 12 ways. With only two colors, it cannot be colored at all. With four colors, it can be colored in 24 + 4Å"12 = 72 ways: using all four colors, there are 4! = 24 valid colorings (*every* assignment of four colors to *any* 4-vertex graph is a proper coloring); and for every choice of three of the four colors, there are 12 valid 3-colorings. So, for the graph in the example, a table of the number of valid colorings would start like this:

Available colors	1	2	3	4	...
Number of colorings 0	0	12	72	...	

The chromatic polynomial is a function P(G, t) that counts the number of *t*-colorings of G. As the name indicates, for a given G the function is indeed a polynomial in *t*. For the example graph, $P(G, t) = t(t - 1)^2(t - 2)$, and indeed P(G, 4) = 72.

The chromatic polynomial includes at least as much information about the colorability of G as does the chromatic number. Indeed, χ is the smallest positive integer that is not a root of the chromatic polynomial $\chi(G) = \min\{k:P(G,k) > 0\}$.

Chromatic polynomials for certain graphs:

Triangle K_3 $t(t - 1)(t - 2)$

Complete graph K_n

Tree with *n* vertices $t(t - 1)^{n - 1}$

Cycle C_n $(t - 1)^n + (-1)^n(t - 1)$

Petersen graph $t(t - 1)(t - 2)(t^7 - 12t^6 + 67t^5 - 230t^4 + 529t^3 - 814t^2 + 775t - 352)$

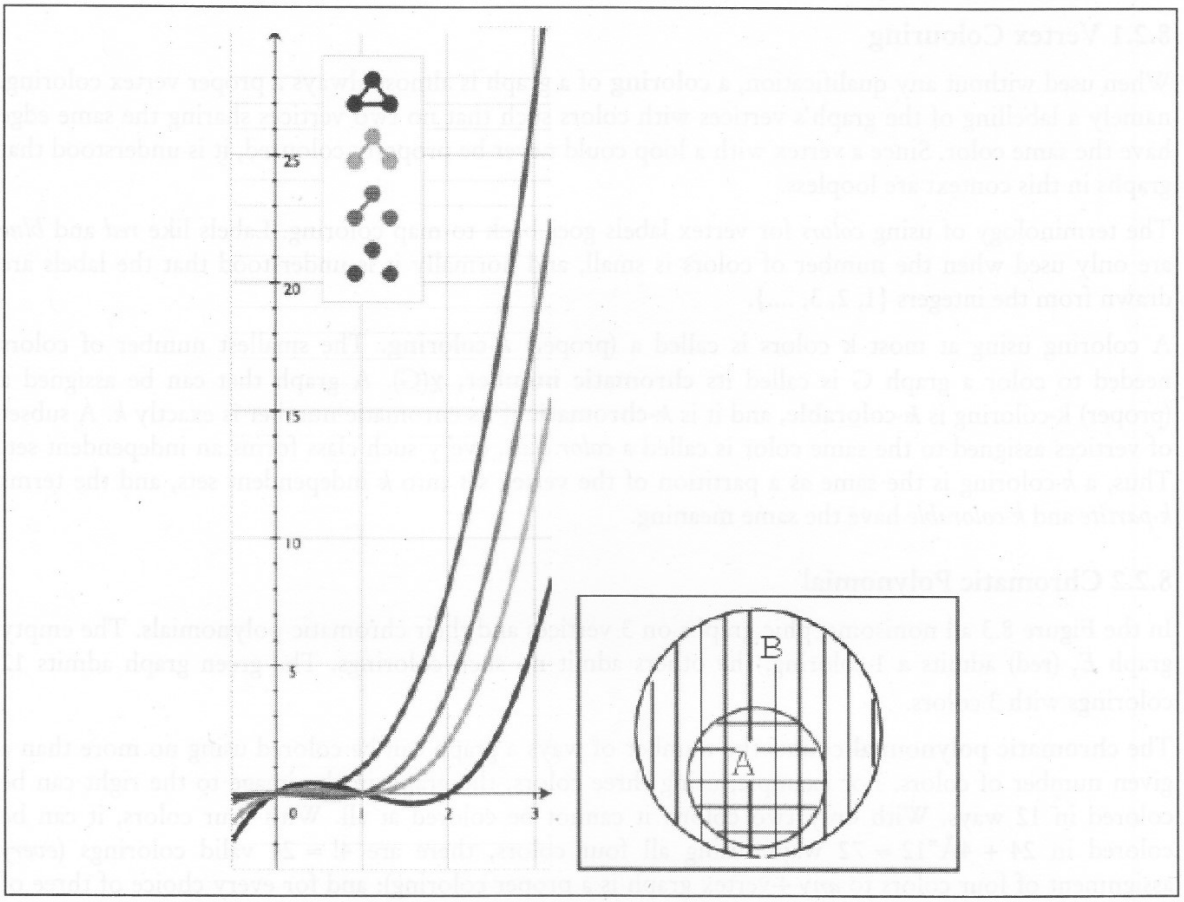


Figure 8.3: Chromatic Polynomial

8.2.3 Edge Colouring

An **edge coloring** of a graph, is a proper coloring of the *edges*, meaning an assignment of colors to edges so that no vertex is incident to two edges of the same color. An edge coloring with k colors is called a k-edge-coloring and is equivalent to the problem of partitioning the edge set into k matchings. The smallest number of colors needed for an edge coloring of a graph G is the **chromatic index,** or **edge chromatic number,** $\chi 2$ (G). A **Tait coloring** is a 3-edge coloring of a cubic graph. The four color theorem is equivalent to the assertion that every planar cubic bridgeless graph admits a Tait coloring.

8.3 PROPERTIES

Bounds on the Chromatic Number

Assigning distinct colors to distinct vertices always yields a proper coloring, so

$$1 \leq \chi(G) \leq n$$

The only graphs that can be 1-colored are edgeless graphs, and the complete graph K_n of n vertices requires $\chi(K_n) = n$ colors. In an optimal coloring there must be at least one of the graph's m edges between every pair of color classes, so

$$\chi(G)(\chi(G)-1) \leq 2m$$

If G contains a clique of size k, then at least k colors are needed to color that clique; in other words, the chromatic number is at least the clique number:

$$\chi(G) \geq \omega(G)$$

For interval graphs this bound is tight.

The 2-colorable graphs are exactly the bipartite graphs, including trees and forests. By the four color theorem, every planar graph can be 4-colored.

A greedy coloring shows that every graph can be colored with one more color than the maximum vertex degree,

$$\chi(G) \leq \Delta(G)+1$$

Complete graphs have ÷(G) = n and Ä(G) = n " 1, and odd cycles have ÷(G) = 3 and Ä(G) = 2, so for these graphs this bound is best possible. In all other cases, the bound can be slightly improved; Brooks' theorem states that

Brooks' theorem: $\chi(G) \leq \Delta(G)$ for a connected, simple graph G, unless G is a complete graph or an odd cycle.

Graphs with High Chromatic Number

Graphs with large cliques have high chromatic number, but the opposite is not true. The Grötzsch graph is an example of a 4-chromatic graph without a triangle, and the example can be generalised to the Mycielskians.

Mycielski's Theorem: There exist triangle-free graphs with arbitrarily high chromatic number.

From Brooks's theorem, graphs with high chromatic number must have high maximum degree. Another local property that leads to high chromatic number is the presence of a large clique. But colorability is not an entirely local phenomenon: A graph with high girth looks locally like a tree, because all cycles are long, but its chromatic number need not be 2:

Theorem (Erdõs): There exist graphs of arbitrarily high girth and chromatic number.

Bounds on the Chromatic Index

An edge coloring of G is a vertex coloring of its line graph L(G), and vice versa. Thus,

$$\chi(G) = \chi'(L(G))$$

There is a strong relationship between edge colorability and the graph's maximum degree Ä(G). Since all edges incident to the same vertex need their own color, we have

$$\chi'(G) \geq \Delta(G)$$

Moreover,

König's theorem: $\chi'(G)$ = D(G) if G is bipartite.

In general, the relationship is even stronger than what Brooks's theorem gives for vertex coloring:

Vizing's Theorem: A graph of maximal degree D has edge-chromatic number D or D + 1.

Other Properties

For planar graphs, vertex colorings are essentially dual to nowhere-zero flows.

About infinite graphs, much less is known. The following is one of the few results about infinite graph coloring:

If all finite subgraphs of an infinite graph G are k-colorable, then so is G, under the assumption of the axiom of choice.

Open Problems

The chromatic number of the plane, where two points are adjacent if they have unit distance, is unknown, although it is one of 4, 5, 6, or 7. Other open problems concerning the chromatic number of graphs include the Hadwiger conjecture stating that every graph with chromatic number k has a complete graph on k vertices as a minor, the Erdős–Faber–Lovász conjecture bounding the chromatic number of unions of complete graphs that have at exactly one vertex in common to each pair, and the Albertson conjecture that among k-chromatic graphs the complete graphs are the ones with smallest crossing number.

When Birkhoff and Lewis introduced the chromatic polynomial in their attack on the four-color theorem, they conjectured that for planar graphs G, the polymomial P(G,t) has no zeros in the region [4, ∞]. Although it is known that such a chromatic polynomial has no zeros in the region [5, ∞] and that P(G, 4) ≠ 0, their conjecture is still unresolved. It also remains an unsolved problem to characterize graphs which have the same chromatic polynomial and to determine which polynomials are chromatic.

8.4 APPLICATIONS

8.4.1 Scheduling

Vertex coloring models to a number of scheduling problems. In the cleanest form, a given set of jobs need to be assigned to time slots, each job requires one such slot. Jobs can be scheduled in any order, but pairs of jobs may be in *conflict* in the sense that they may not be assigned to the same time slot, for example because they both rely on a shared resource. The corresponding graph contains a vertex for every job and an edge for every conflicting pair of jobs. The chromatic number of the graph is exactly the minimum *makespan*, the optimal time to finish all jobs without conflicts.

Details of the scheduling problem define the structure of the graph. For example, when assigning aircrafts to flights, the resulting conflict graph is an interval graph, so the coloring problem can be solved efficiently. In bandwidth allocation to radio stations, the resulting conflict graph is a unit disk graph, so the coloring problem is 3-approximable.

8.4.1 Register Allocation

A compiler is a computer program that translates one computer language into another. To improve the execution time of the resulting code, one of the techniques of compiler optimization is register allocation, where the most frequently used values of the compiled program are kept in the fast processor registers. Ideally, values are assigned to registers so that they can all reside in the registers when they are used.

The textbook approach to this problem is to model it as a graph coloring problem. The compiler constructs an *interference graph*, where vertices are symbolic registers and an edge connects two nodes if they are needed at the same time. If the graph can be colored with k colors then the variables can be stored in k registers.

8.4.3 Other Applications

The problem of coloring a graph has found a number of applications, including pattern matching.

The recreational puzzle Sudoku can be seen as completing a 9-coloring on given specific graph with 81 vertices.

8.5 GRAPH PARTITIONING AND COVERING

Graph Partitioning

Partitioning a graph is dividing the graph into two or more large pieces while minimizing the size of the graph between. That is dividing the large graph into number of sub graphs is known as graph partitioning. Partion of V into disjoints sub sets, where the sub graph created is a complete graph.

Graph Covering

Consider a graph G(V,E). Graph Covering is a collection of vertices of subsets of vertex V, where each vertex shows a complete sub graph of G, and there is some vertex for each edge.

8.6 PLANAR GRAPHS

A graph G is planar if it can be drawn on a plane in such a way that is no two edges meet except at a vertex with which they both incident. Any such drawing is a plane drawing of G. A graph is non-planar if no plane drawing of it exists. Figure 8.4 shows a common representation of the graph K_4 and Figure 8.5 shows three possible plane drawings of K_4.

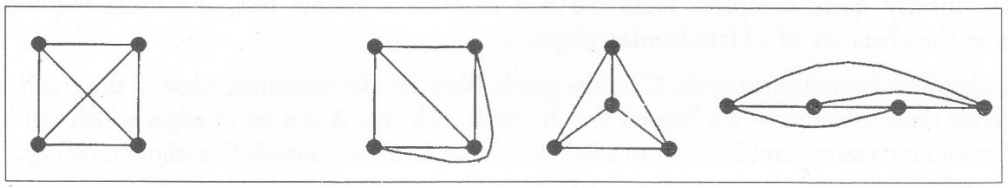


Figure 8.4 Figure 8.5

The complete bipartite graph $K_{3,3}$ and the complete graph K_5 are important simple examples of non-planar graphs. (You will prove this in Graph Theory Exercises 4!).

Any plane drawing of a planar graph G divides the set of points of the plane not lying on G into regions called faces. The region outside the graph is of infinite extent and is called the infinite face.

The degree of a face f of a connected planar graph G, denoted deg (f), is the number of edges encountered in a walk around the boundary of f. If all faces have the same degree, g, then G is face-regular of degree g.

In Figure 8.4 each plane drawing of the graph K_4 has 4 faces (including the infinite face) each face being of degree 3 so K_4 is face-regular of degree 3.

In any plane drawing of a planar graph, the sum of all the face degrees is equal to twice the number of edges.

In Figure 8.5 the plane drawing of the graph K_4 has 4 face (including the infinite face) each face being of degree 3 so K_4 is face-regular of degree 3. The sum of the face degrees is therefore 12 whilst the number of edges is 6.

Euler's Formula for Planar Graphs

If n, m and f denote respectively the number of vertices, edges and faces in a plane drawing of a connected plane graph G then $n - m + f = 2$.

In Figure 8.4 we have $n = 4$, $m = 6$ and $f = 4$ satisfying Euler's formula.

Proof of Euler's Formula

A plane drawing of any connected planar graph G can be constructed by taking a spanning tree of G and adding edges to it, one at a time, until a plane drawing of G is obtained.

We prove Euler's formula by showing that

(*a*) for any spanning tree G, $n - m + f = 2$ and

(*b*) adding an edge to the spanning tree does not change the value of $n - m + f$.

Let T be any spanning tree of G. We may draw T in a plane without crossings. T has n vertices $n - 1$ edges and 1 face (the infinite face). Thus $n - m + f = n - (n - 1) + 1 = 2$ so we have shown (*a*).

Now if we add an edge to T either it joins two different vertices, or it joins a vertex to itself (it is a loop). In either case it divides some face into two faces, so adding one face. Hence we have increased m, the number of edges, and f, the number of faces, by one each. The value of the expression $n - m + f$ is unchanged. We add more edges, one at a time, and at each addition the value of $n - m + f$ is unchanged.

Hence we have shown (b) and so proved Euler's theorem.

It is useful to be able to test a graph for planarity. There are a variety of algorithms for determining planarity, mostly quite complex. Here we will describe a simple test, the cycle method, which determines the planarity of a Hamiltonian graph.

First we identify a Hamiltonian cycle, C, in the graph. Now list the remaining edges of the graph, and then try to divide those edges into two disjoint sets A and B such that A is a set of edges which can be drawn inside C without crossings and B is a set of edges which can be drawn outside C without crossings.

Example: Determine whether the graph in Figure 8.6 is planar.

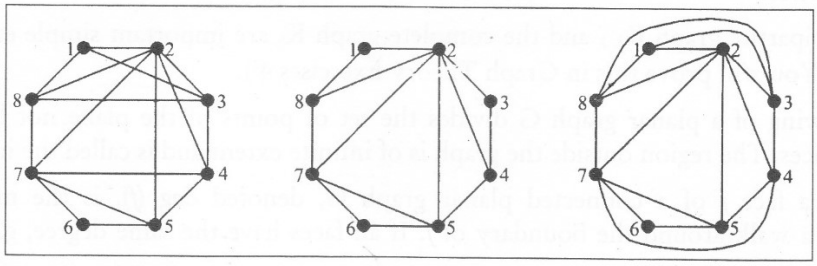


Figure 8.6 **Figure 8.7** **Figure 8.8**

First find a Hamiltonian cycle. 123456781 will do. Draw a plane drawing of the cycle. The remaining edges are {24, 25, 27, 28, 31, 38, 47, 57}. Take the first edge, 24, and put it in set A. Take the second edge, 25, and put it in set A if compatible—it is so put it in set A. Consider the next edge, 27—it is

comparable with set A so add it to set A. At this point we have A = {24, 25, 27}, B = { }. The next edge is 28—it is compatible with set A so add it to set A (A = {24, 25, 27, 28}). The next edge is 31 which is not compatible with set A so put it in set B(B = {31}). The next edge is 38 which is not compatible with set A so put it in set B(B = {31, 38}). The next edge is 47 which is not compatible with set A so put it in set B (B = {31, 38, 47}). The next edge 57 which is compatible with set A so put it in set A(A = 24, 25, 27, 28, 57}). Figure 8.6 shows the Hamiltonian cycle 123456781 and the edges in set A drawn inside the cycle. Now if we can add the edges from set B, all outside the cycle, without crossings then we have a plane drawing of the graph and it will be planar. Figure 8.7 shows that the edges in set B can be drawn in that way so the graph is planar and Figure 8.7 is a plane drawing.

8.7 DIRECTED GRAPHS

A **digraph (directed graph)** is a diagram consisting of points, called vertices, joined by directed lines, called arcs.

A Digraph G is defined as an ordered pair(V,E), where V is the set of points called vertices and E is the set of edges. Each edge in the graph G is assigned a direction and is identified with an ordered pair(u,v) where u is the initial vertex and v is the end vertex.

8.8 CHROMATIC POLYNOMIAL

The **chromatic polynomial** counts the number of ways a graph can be colored using no more than a given number of colors. It has been discussed in detail under graph colouring as above.

Five Colour Theorm

Theorm: The vertices of every planer graph can be properly coloured with five colours.

Proof: We will prove this theorm by induction. All the graphs with 1,2,3,4,or,5 vertices can be properly coloured with five colours. Now let us assume that every planer graph with n-1 vertices can be properly coloured with five colours. If we prove that any planer graph G with n vertices will require no more than five colours,we have done.

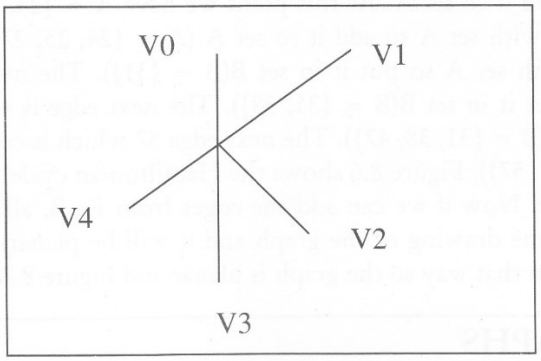
Consider the planer graph G with n vertices.

Since G is planer, it must have atleast one vertex with degree five or less. Assume this vertex to be 'u'.

Let G1 be a graph of n-1 vertices obtained from G by deleting vertex 'u'. The G1 graph requires no more than five colours. Let the vertices in G1 have been properly coloured and now add to it 'u' and all the edges incident on u. If the degree of u is 1,2,3, or, 4, a proper colour to u can be easily assigned.

Now,consider the case in which degree of u is 5. and all the five colours have been used in colouring the vertices. Adjacent to u. as shown in figure below.

Thus, if there is no path between v4 and v2, coloured alternately with colours 5 and 3 of all vertices connected to v2 through vertices of alternating colours 5 and 3. This interchange will colour vertex v2 with colour 5 and yet keep G, properly coloured. As vertex v4 is still with colour 5, the colour 3 is left over with which to colour vertex u which proves the theorm.



Check Your Progress

1. What is Chromatic Polynomial?
2. Define Planer graph.
3. If G is a simple, connected, planar graph with $n(\geq 3)$ vertices and m edges, and if g is the length of the shortest cycle in G, show that

   $$m \leq g(n-2)/(g-2)$$

8.9 LET US SUM UP

In graph theory, graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called "colors" to elements of a graph subject to certain constraints. In its simplest form, it is a way of coloring the vertices of a graph such that no two adjacent vertices share the same color; this is called a vertex coloring. Similarly, an edge coloring assigns a color to each edge so that no two adjacent edges share the same color, and a face coloring of a planar graph assigns a color to each face or region so that no two faces that share a boundary have the same color. The chromatic polynomial counts the number of ways a graph can be colored using no more than a given number of colors. For example, using three colors, the graph in the image to the right can be colored in 12 ways. A Tait coloring is a 3-edge coloring of a cubic graph. The four color theorem is equivalent to the assertion that every planar cubic bridgeless graph admits a Tait coloring. A graph G is planar if it can be drawn on a plane in such a way that is no two edges meet except at a vertex with which they both incident. Any such drawing is a plane drawing of G. A graph is non-planar if no plane drawing of it exists. A digraph (directed graph) is a diagram consisting of points, called vertices, joined by directed lines, called arcs. Vertex coloring is the starting point of the subject, and other coloring problems can be transformed into a vertex version.

8.10 KEYWORDS

Graph Colouring: Graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called "colors" to elements of a graph subject to certain constraints

Vertex Coloring: It is the starting point of the subject, and other coloring problems can be transformed into a vertex version.

Edge Coloring: It is a proper coloring of the *edges*, meaning an assignment of colors to edges so that no vertex is incident to two edges of the same color.

Digraph (Directed Graph): It is a diagram consisting of points, called vertices, joined by directed lines, called arcs.

8.11 QUESTIONS FOR DISCUSSION

1. What do you understand by coloring of graphs?
2. Describe vertex coloring.
3. What do you mean by edge coloring?
4. What are the main properties of coloring of a graph?
5. Discuss applications of coloring of a graph.
6. What is Five Colour Theorm?

Check Your Progress: Modal Answers

1. The chromatic polynomial counts the number of ways a graph can be colored using no more than a given number of colors. For example, using three colors, the graph in the image to the right can be colored in 12 ways. With only two colors, it cannot be colored at all. With four colors, it can be colored in 24 + 4Å"12 = 72 ways: using all four colors, there are 4! = 24 valid colorings
2. A graph G is planar if it can be drawn on a plane in such a way that is no two edges meet except at a vertex with which they both incident. Any such drawing is a plane drawing of G.
3. In a plane drawing of a planar graph the edges surrounding a face are a cycle. Thus, if g is the length of the shortest cycle in a planar graph, the degree of each face in the plane drawing is $\geq g$. Therefore the sum of the face degrees is $\geq gf$. The Handshaking Lemma tells us that the sum of the face degrees is twice the number of edges = $2m$, so $2m \geq gf$. Now Euler's formula tells us that $n - m + f = 2$ so $f = m + 2 - n$. Hence $2m \geq gf = g(m + 2 - n)$. Hence, we have $2m \geq g(m + 2 - n)$. Hence $g(n - 2) \geq (g - 2)m$, i.e. $g(n - 2)/(g - 2) > m$.

8.12 SUGGESTED READINGS

Anuranjan Misra, *Discrete Mathematics*, Acme Learning pvt ltd.

Richard Johnsonbaugh, *Discrete Mathematics*, Prentice Hall

V. K. Balakrishnan, *Introductory Discrete Mathematics*, Courier Dover Publications,

R. C. Penner, *Discrete Mathematics: Proof Techniques and Mathematical Structures*, World Scientific, 1999

Mike Piff, *Discrete Mathematics: An Introduction for Software* Engineers, Cambridge University Press, 1991