Preface

Graph theory is a well-known area of Discrete Mathematics which has so many theoretical developments and applications not only to different branches of Mathematics, but also to various other fields of basic sciences, technology, social sciences and computer science. Graphs are widely used as efficient and effective tools to model many types of practical and real-world problems in physical, biological, social and information systems. Graph-theoretical models and methods are based on mathematical combinatorics and related fields.

This book is written for the students of Computer Science, who study the subject Graph Theory under their university curriculum. The content of the book is prepared as per the syllabus of fifth semester Bachelor of Technology in Computer Science and engineering under APJ Abdul Kalam Kerala Technological University, Thiruvananthapuram, Kerala, India. I started writing the lecture notes one and a half year ago as per the requests and requirements of the under-graduate and post-graduate students of Computer Science and Computer Applications programmes under various universities in the state of Kerala, India.

In coming years, I also plan to extend the content of the book by including some major and important areas like graph colouring, matching, coverings and network flows. Then, students of other universities and programmes can also make use of the book.

On this occasion, I would like to thank all members of my family for the support they have offered to me during the preparation of the book. I also gratefully acknowledge the support and encouragements of my colleagues, the Principal and the Management of Vidya Academy of Science & Technology, Thrissur, India. My dear students, I will not forget your

involvements in the preparation of this book. I once again use this opportunity to thank all who have extended helping hands to me during this period.

Graph Theory has nominal dependence on many other areas of Mathematics and so students of even non-mathematics programmes can learn this subject very easily. But, I expect that the readers have a sound mathematical background for a better understanding of the subject. A proper understanding of the technique of writing proofs in a logical manner is essential for the readers or learners to have a fruitful learning experience.

The Centre for Studies in Discrete Mathematics (CSDM), Vidya Academy of Sciences & Technology, Thrissur, India. is the publisher of this book. CSDM is a noble initiative to encourage quality research in Discrete Mathematics and allied areas, in collaboration with other reputed institutions and centres of excellence. The Centre works closely with people associated with teaching and research in related disciplines like computer science and electronics engineering. CSDM publishes this book electronically as an open access material. Anybody can download the book from the website of the Centre and use it for his or her requirements by giving due acknowledgements to the publisher and the author.

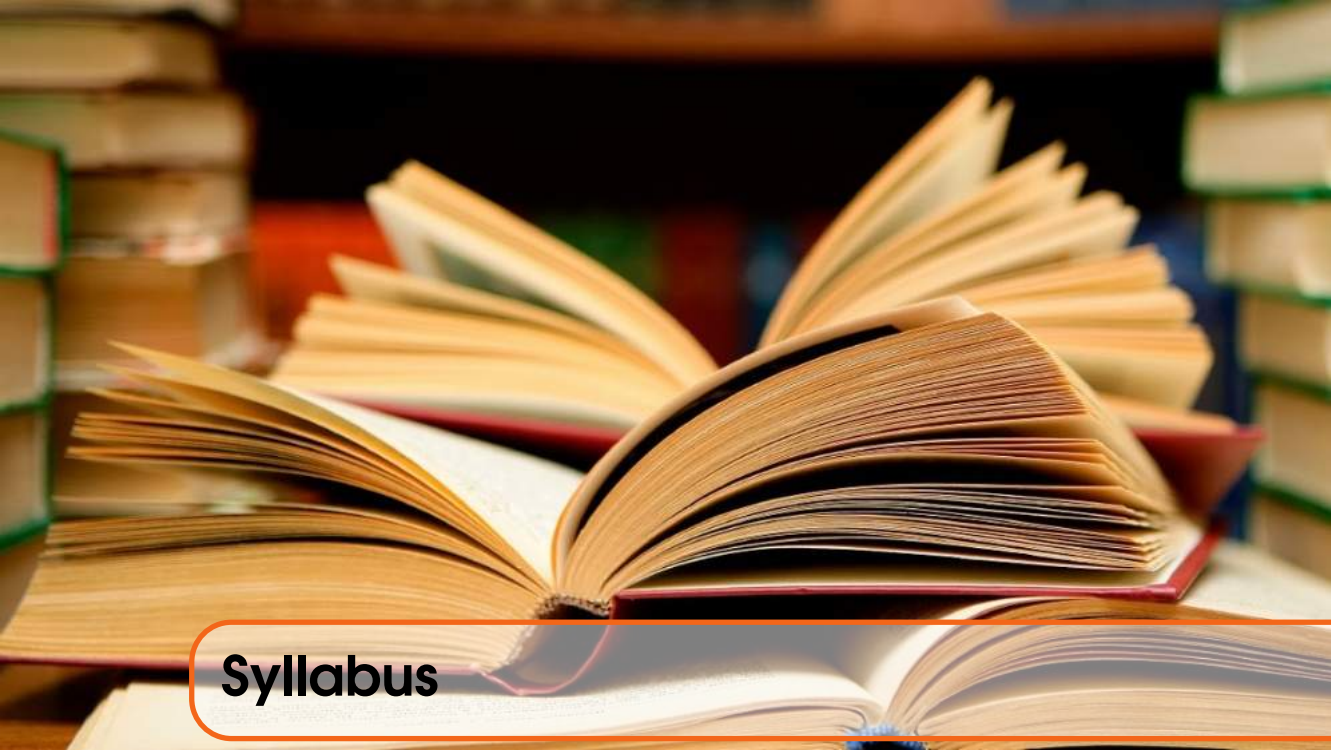
I wholeheartedly invite criticism and suggestions from the readers of this book for improving the content and presentation style.

Thrissur
November 2017.

Sudev Naduvath

Contents

SYLLABUS



Syllabus

Module-1

Introductory concepts - What is a graph? - applications of graphs - finite and infinite graphs - incidence and degree - isolated vertex and pendent vertex - null graph - paths and circuits - isomorphism of graphs - subgraphs - walks, paths and circuits - connected graphs - disconnected graphs.

Hours allotted: 09 Marks: 15%

Module-2

Euler graphs, Hamiltonian paths and circuits, Dirac's theorem for Hamiltonicity, travelling salesman problem. directed graphs – types of digraphs, digraphs and binary relation.

Hours allotted: 10 Marks: 15%

Module-3

Trees – properties, pendent vertex, distance and centres - rooted and binary trees, counting trees, spanning trees.

Hours allotted: 07 Marks: 15%

Module-4

Vertex connectivity, edge connectivity, cut-sets and cut-vertices, fundamental circuits, planar graphs, different representation of planar graphs, Euler's theorem, geometric dual, combinatorial dual.

Hours allotted: 09 Marks: 15%

Module-5

Matrix representation of graphs- adjacency matrix, incidence matrix, circuit matrix, fundamental circuit matrix and rank, cut-set matrix, path matrix

Hours allotted: 08 Marks: 20%

Module-6

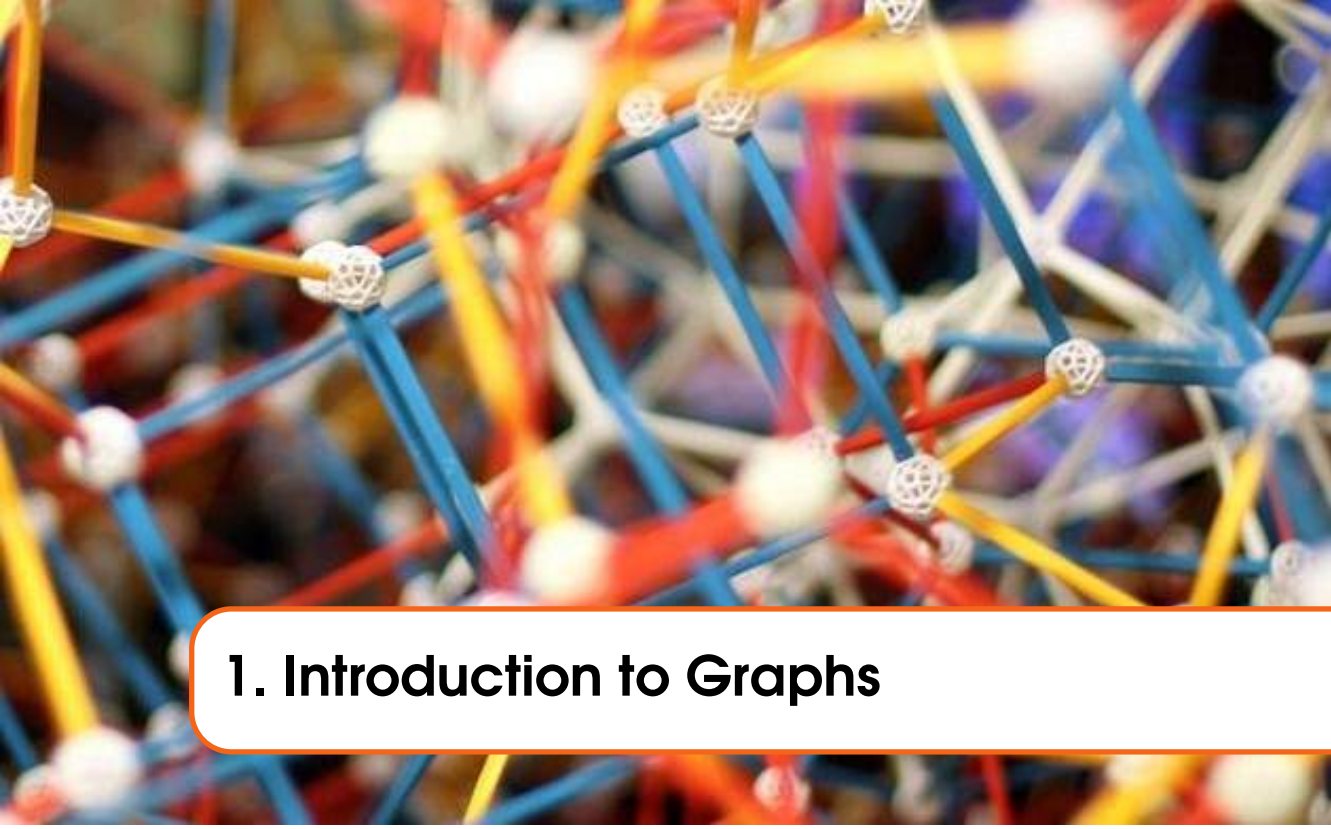
Graphs theoretic algorithms - algorithm for computer representation of a graph, algorithm for connectedness and components, spanning tree algorithms, shortest path algorithms.

Hours allotted: 07 Marks: 20%

Reference Books:

1. Douglas B. West, *Introduction to Graph Theory*, Prentice Hall of India Ltd., 2001.
2. Narasingh Deo, *Graph Theory*, PHI, 1979.
3. Robin J. Wilson, *Introduction to Graph Theory*, Longman Group Ltd., 2010.
4. R. Diestel, *Graph Theory*, free online edition, 2016:
   `diestel-graph-theory.com/basic.html`.

INTRODUCING GRAPHS



1. Introduction to Graphs

Graph Theory is a well-known area of discrete mathematics which deals with the study of graphs. A graph may be considered as a mathematical structure that is used for modelling the pairwise relations between objects.

Graph Theory has many theoretical developments and applications not only to different branches of mathematics, but also to various other fields of basic sciences, technology, social sciences, computer science etc. Graphs are widely used as efficient tools to model many types of practical and real-world problems in physical, biological, social and information systems. Graph-theoretical models and methods are based on mathematical combinatorics and related fields.

1.1 Basic Definitions

Definition 1.1.1 — Graph. A *graph* G can be considered as an ordered triple (V, E, ψ), where

(i) $V = \{v_1, v_2, v_3, \ldots\}$ is called the *vertex set* of G and the elements of V are called the *vertices* (or *points* or *nodes*);

(ii) $E = \{e_1, e_2, e_3, \ldots\}$ is the called the *edge set* of G and the elements of E are called *edges* (or *lines* or *arcs*); and

(iii) ψ is called the *adjacency relation*, defined by $\psi : E \to V \times V$, which defines the association between each edge with the vertex pairs of G.

Usually, the graph is denoted as $G = (V, E)$. The vertex set and edge set of a graph G are

also written as $V(G)$ and $E(G)$ respectively.

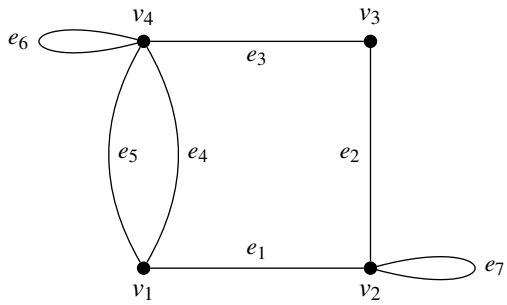


Figure 1.1: An example of a graph

If two vertices u and v are the (two) end points of an edge e, then we represent this edge by uv or vu. If $e = uv$ is an edge of a graph G, then we say that u and v are *adjacent vertices* in G and that e *joins* u and v. In such cases, we also say that u and v are adjacent to each other.

Given an edge $e = uv$, the vertex u and the edge e are said to be *incident with* each other and so are v and e. Two edges e_i and e_j are said to be *adjacent edges* if they are incident with a common vertex.

Definition 1.1.2 — Order and Size of a Graph. The *order* of a graph G, denoted by $\nu(G)$, is the number of its vertices and the *size* of G, denoted by $\varepsilon(G)$, is the number of its edges.

A graph with p-vertices and q-edges is called a (p,q)-*graph*. The $(1,0)$-graph is called a *trivial graph*. That is, a trivial graph is a graph with a single vertex. A graph without edges is called an *empty graph* or a *null graph*. The following figure illustrates a null graph of order 5.

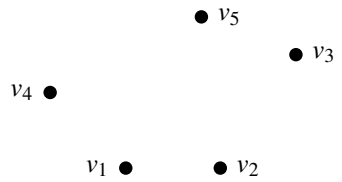


Figure 1.2: Null graph of order 5.

Definition 1.1.3 — Finite and Infinite Graphs. A graph with a finite number of vertices as well as a finite number of edges is called a *finite graph*. Otherwise, it is an *infinite graph*.

Definition 1.1.4 — Self-loop. An edge of a graph that joins a node to itself is called *loop* or a *self-loop*. That is, a loop is an edge uv, where $u = v$.

Definition 1.1.5 — Parallel Edges. The edges connecting the same pair of vertices are called *multiple edges* or *parallel edges*.

In Figure 1.2, the edges e_6 and e_7 are loops and the edges e_4 and e_5 are parallel edges.

Definition 1.1.6 — Simple Graphs and Multigraphs. A graph G which does not have loops or parallel edges is called a *simple graph*. A graph which is not simple is generally called a *multigraph*.

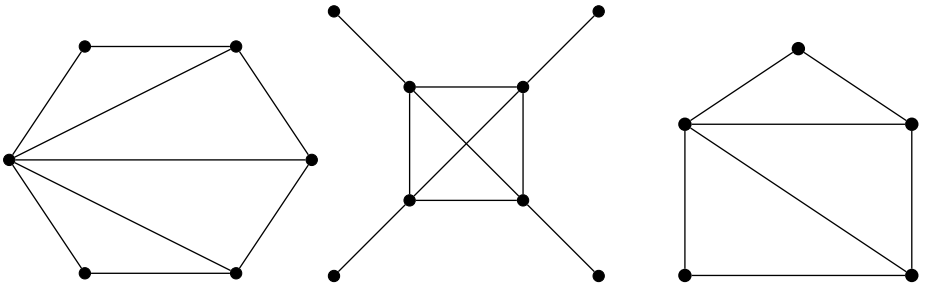


Figure 1.3: Some examples of simple graphs

1.2 Degrees and Degree Sequences in Graphs

Definition 1.2.1 — Degree of a vertex. The number of edges incident on a vertex v, with self-loops counted twice, is called the *degree* of the vertex v and is denoted by $deg_G(v)$ or $deg(v)$ or simply $d(v)$.

Definition 1.2.2 — Isolated vertex. A vertex having no incident edge is called an *isolated vertex*. In other words, isolated vertices are those with zero degree.

Definition 1.2.3 — Pendant vertex. A vertex of degree 1, is called a *pendent vertex* or an end vertex.

Definition 1.2.4 — Internal vertex. A vertex, which is neither a pendent vertex nor an isolated vertex, is called an *internal vertex* or an *intermediate vertex*.

Definition 1.2.5 — Minimum and Maximum Degree of a Graph. The *maximum degree of a graph* G, denoted by $\Delta(G)$, is defined to be $\Delta(G) = \max\{d(v) : v \in V(G)\}$. Similarly, the *minimum degree of a graph* G, denoted by $\delta(G)$, is defined to be $\delta(G) = \min\{d(v) : v \in V(G)\}$. Note that for any vertex v in G, we have $\delta(G) \leq d(v) \leq \Delta(G)$.

The following theorem is a relation between the sum of degrees of vertices in a graph G and the size of G.

Theorem 1.2.1 In a graph G, the sum of the degrees of the vertices is equal to twice the number of edges. That is, $\sum_{v \in V(G)} d(v) = 2\varepsilon$.

Proof. Let $S = \sum_{v \in V(G)} d(v)$. Notice that in counting S, we count each edge exactly twice. That is, every edge contributes degree 1 each to both of its end vertices and a loop provides degree 2 to the vertex it incidents with. Hence 2 to the sum of degrees of vertices in G. Thus, $S = 2|E| = 2\varepsilon$. ■

The above theorem is usually called the *first theorem on graph theory*. It is also known as the *hand shaking lemma*. The following two theorems are immediate consequences of the above theorem.

Theorem 1.2.2 For any graph G, $\delta(G) \leq \frac{2|E|}{|V|} \leq \Delta(G)$.

Proof. By Theorem-1, we have $2\varepsilon = \sum_{v \in V(G)} d(v)$. Therefore, note that $\frac{2|E|}{|V|} = \frac{\sum d(v)}{|V|}$, the average degree of G. Therefore, $\delta(G) \leq \frac{2|E|}{|V|} \leq \Delta(G)$. ■

Theorem 1.2.3 For any graph G, the number of odd degree vertices is always even.

Proof. Let $S = \sum_{v \in V(G)} d(v)$. By Theorem 1.2.1, we have $S = 2\varepsilon$ and hence S is always even. Let V_1 be the set of all odd degree vertices and V_2 be the set of all even degree vertices in G. Now, let $S_1 = \sum_{v \in V_1} d(v)$ and $S_2 = \sum_{v \in V_2} d(v)$. Note that S_2, being the sum of even integers, is also an even integer.

We also note that $S = S_1 + S_2$ (since V_1 and V_2 are disjoint sets and $V_1 \cup V_2 = V$). Therefore, $S_1 = S - S_2$. Being the difference between two even integers, S_1 is also an even integer. Since V_1 is a set of odd degree vertices, S_1 is even only when the number of elements in V_1 is even. That is, the number of odd degree vertices in G is even, completing the proof. ■

Definition 1.2.6 — Degree Sequence. The *degree sequence* of a graph of order n is the n-term sequence (usually written in descending order) of the vertex degrees. In Figure-1, $\delta(G) = 2$, $\Delta(G) = 5$ and the degree sequence of G is $(5,4,3,2)$.

Definition 1.2.7 — Graphical Sequence. An integer sequence is said to be *graphical* if it is the degree sequence of some graphs. A graph G is said to be the *graphical realisation* of an integer sequence S if S the degree sequence of G.

Problem 1.1 Is the sequence $S = \langle 5,4,3,3,2,2,2,1,1,1,1 \rangle$ graphical? Justify your answer.
Solution: The sequence $S = \langle a_i \rangle$ is graphical if every element of S is the degree of some vertex in a graph. For any graph, we know that $\sum_{v \in V(G)} d(v) = 2|E|$, an even integer. Here, $\sum a_i = 25$, not an even number. Therefore, the given sequence is not graphical. ■

Problem 1.2 Is the sequence $S = \langle 9,9,8,7,7,6,6,5,5\rangle$ graphical? Justify your answer.

Solution: The sequence $S = \langle a_i\rangle$ is graphical if every element of S is the degree of some vertex in a graph. For any graph, we know that $\sum_{v\in V(G)} d(v) = 2|E|$, an even integer. Here, $\sum a_i = 62$, an even number. But note that the maximum degree that a vertex can attain in a graph of order n is $n-1$. If S were graphical, the corresponding graph would have been a graph on 9 vertices and have $\Delta(G) = 9$. Therefore, the given sequence is not graphical. ■

Problem 1.3 Is the sequence $S = \langle 9,8,7,6,6,5,5,4,3,3,2,2\rangle$ graphical? Justify your answer.

Solution: The sequence $S = \langle a_i\rangle$ is graphical if every element of S is the degree of some vertex in a graph. For any graph, we know that $\sum_{v\in V(G)} d(v) = 2|E|$, an even integer. Here, we have $\sum a_i = 60$, an even number. Also, note that the all elements in the sequence are less than the number of elements in that sequence.

Therefore, the given sequence is graphical and the corresponding graph is drawn below. ■

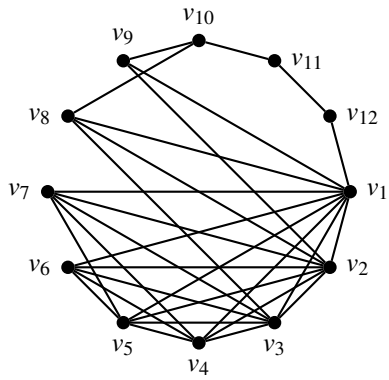


Figure 1.4: Graphical realisation of the degree sequence S.

1.2.1 Neighbourhoods

Definition 1.2.8 — Neighbourhood of a Vertex. The *neighbourhood* (or *open neighbourhood*) of a vertex v, denoted by $N(v)$, is the set of vertices adjacent to v. That is, $N(v) = \{x \in V : vx \in E\}$. The *closed neighbourhood* of a vertex v, denoted by $N[v]$, is simply the set $N(v)\cup\{v\}$.

Then, for any vertex v in a graph G, we have $d_G(v) = |N(v)|$. A special case is a loop that connects a vertex to itself; if such an edge exists, the vertex is said to belong to its own neighbourhood.

Given a set S of vertices, we define the neighbourhood of S, denoted by $N(S)$, to be the union of the neighbourhoods of the vertices in S. Similarly, the closed neighbourhood of S, denoted by $N[S]$, is defined to be $S\cup N(S)$.

Neighbourhoods are widely used to represent graphs in computer algorithms, in terms of the adjacency list and adjacency matrix representations. Neighbourhoods are also used in the clustering coefficient of graphs, which is a measure of the average density of its

neighbourhoods. In addition, many important classes of graphs may be defined by properties of their neighbourhoods, or by symmetries that relate neighbourhoods to each other.

1.3 Subgraphs and Spanning Subgraphs

Definition 1.3.1 — Subgraph of a Graph. A graph $H(V_1, E_1)$ is said to be a *subgraph* of a graph $G(V, E)$ if $V_1 \subseteq V$ and $E_1 \subseteq E$.

Definition 1.3.2 — Spanning Subgraph of a Graph. A graph $H(V_1, E_1)$ is said to be a *spanning subgraph* of a graph $G(V, E)$ if $V_1 = V$ and $E_1 \subseteq E$.

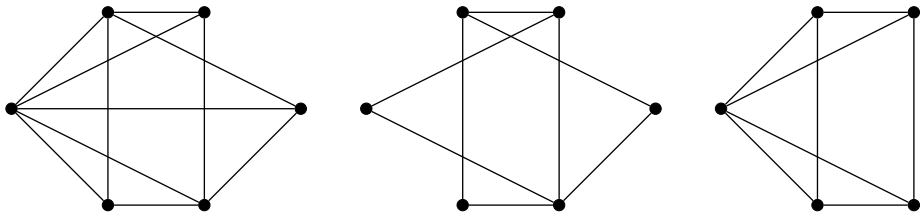


Figure 1.5: Examples of Subgraphs

In the above figure, the second graph is a spanning subgraph of the first graph, while the third graph is a subgraph of the first graph.

1.3.1 Induced Subgraphs

Definition 1.3.3 — Induced Subgraph. Suppose that V' be a subset of the vertex set V of a graph G. Then, the subgraph of G whose vertex set is V' and whose edge set is the set of edges of G that have both end vertices in V' is denoted by $G[V]$ or $\langle V \rangle$ called an *induced subgraph* of G.

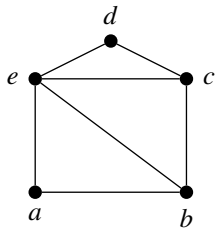
Definition 1.3.4 — Edge-Induced Subgraph. Suppose that E' be a subset of the edge set V of a graph G. Then, the subgraph of G whose edge set is E' and whose vertex set is the set of end vertices of the edges in E' is denoted by $G[E]$ or $\langle E \rangle$ called an *edge-induced subgraph* of G.

Figure 1.6 depicts an induced subgraph and an edge induced subgraph of a given graph.

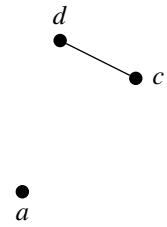
1.4 Fundamental Graph Classes

1.4.1 Complete Graphs

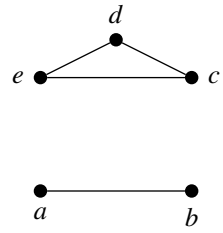
Definition 1.4.1 — Complete Graphs. A *complete graph* is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge. A complete graph



(a) The graph G



(b) The induced subgraph $G[a,c,d]$



(c) The induced subgraph $G[ab,cd,ce,de]$

Figure 1.6: Induced and edge-induced subgraphs of a graph G.

on n vertices is denoted by K_n.

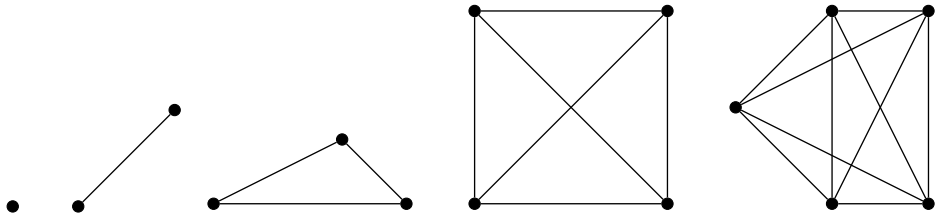


Figure 1.7: First few complete graphs

Problem 1.4 Show that a complete graph K_n has $\frac{n(n-1)}{2}$ edges.

Solution: Note that any two vertices in a complete graph are adjacent to each other. Hence, the number of edges in a complete graph is equal to the number of distinct pairs of vertices in it. Therefore, the number of such pairs of vertices in K_n is $\binom{n}{2} = \frac{n(n-1)}{2}$. That is, the number of edges in K_n is $\frac{n(n-1)}{2}$. ■

We can write an alternate solution to this problem as follows:

Solution: Note that every vertex in a complete graph K_n is adjacent to all other $n-1$ vertices in K_n. That is, $d(v) = n-1$ for all vertices in K_n. Since K_n has n vertices, we have $\sum \lim_{v \in V(K_n)} d(v) = n(n-1)$. Therefore, by the first theorem on graph theory, we have $2|E(K_n)| = n(n-1)$. That is, the number of edges in K_n is $\frac{n(n-1)}{2}$. ■

Problem 1.5 Show that the size of every graph of order n is at most $\frac{n(n-1)}{2}$.

Solution: Note that every graph on n vertices is a spanning subgraph of the complete graph K_n. Therefore, $E(G) \subseteq E(K_n)$. That is, $|E(G)| \leq |E(K_n)| = \frac{n(n-1)}{2}$. That is, any graph of order n can have at most $\frac{n(n-1)}{2}$ edges. ■

1.4.2 Bipartite Graphs

Definition 1.4.2 — Bipartite Graphs. A graph G is said to be a *bipartite graph* if its vertex set V can be partitioned into two sets, say V_1 and V_2, such that no two vertices in

the same partition can be adjacent. Here, the pair (V_1, V_2) is called the *bipartition* of G.

Figure 1.8 gives some examples of bipartite graphs. In all these graphs, the white vertices belong to the same partition, say V_1 and the black vertices belong to the other partition, say V_2.

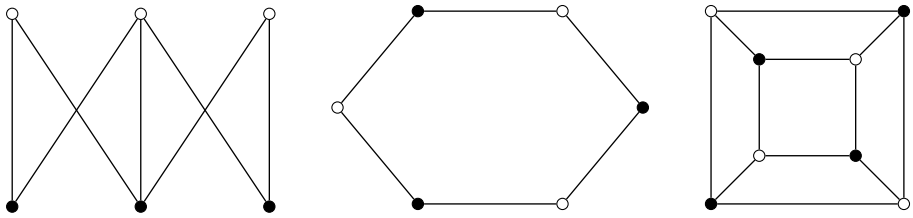


Figure 1.8: Examples of bipartite graphs

Definition 1.4.3 — Complete Bipartite Graphs. A bipartite graph G is said to be a *complete bipartite graph* if every vertex of one partition is adjacent to every vertex of the other. A complete bipartite graph with bipartition (X, Y) is denoted by $K_{|X|,|Y|}$ or $K_{a,b}$, where $a = |X|, b = |Y|$.

The following graphs are also some examples of complete bipartite graphs. In these examples also, the vertices in the same partition have the same colour.

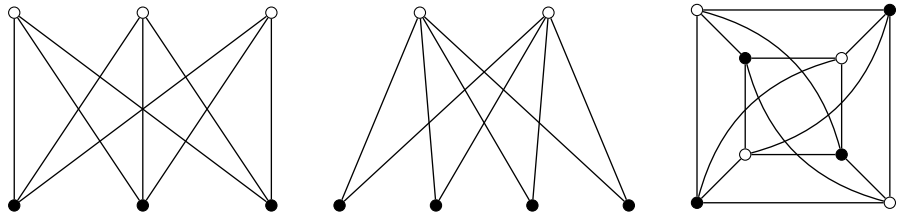


Figure 1.9: Examples of complete bipartite graphs

Problem 1.6 Show that a complete bipartite graph $K_{a,b}$ has ab vertices.

Solution: Let $K_{a,b}$ be a complete bipartite graph with bipartition (X, Y). Note that all a vertices in X have the same degree b and all b vertices in Y have the same degree a. Therefore, $\sum_{v \in V(K_{a,b})} d(v) = ab + ba = 2ab$. By the first theorem on graph theory, we have $2|E(K_{a,b})| = 2ab$. That is, $|E(K_{a,b})| = ab$. ■

Theorem 1.4.1 The complete graph K_n can be expressed as the union of k bipartite graphs if and only if $n \leq 2^k$.

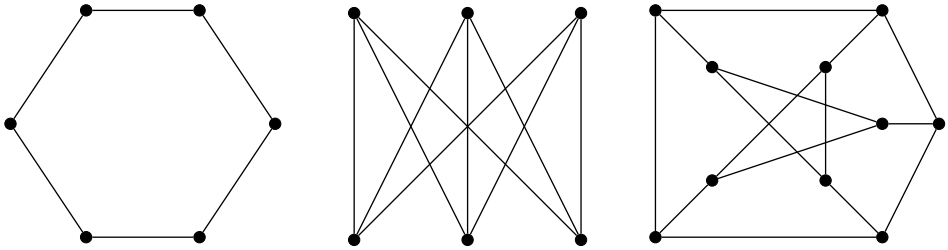
Proof. First assume that K_n can be expressed as the union of k bipartite graphs. We use the method of induction on k. First let $k = 1$. Note that K_n contains triangle K_3 (and K_3 is not bipartite) except for $n \leq 2$. Therefore, the result is true for $k = 1$.

Now assume that $k > 1$ and the result holds for all complete graphs having fewer than k complete bipartite components. Now assume that $K_n = G_1 \cup G_2 \cup \ldots, \cup G_k$, where each G_i is bipartite. Partition the vertex set V into two components such that the graph G_k has no edge within X or within Y. The union of other $k-1$ bipartite subgraphs must cover the complete subgraphs induced by X and Y. Then, by Induction hypothesis, we have $|X| \leq 2^{k-1}$ and $|YX| \leq 2^{k-1}$. Therefore, $n = |X| + |Y| \leq 2^{k-1} + 2^{k-1} = 2^k$. Therefore, the necessary part follows by induction. ■

1.4.3 Regular Graphs

Definition 1.4.4 — Regular Graphs. A graph G is said to be a *regular graph* if all its vertices have the same degree. A graph G is said to be a *k-regular graph* if $d(v) = k \; \forall \; v \in V(G)$. Every complete graph is an $(n-1)$-regular graph.

The degree of all vertices in each partition of a complete bipartite graph is the same. Hence, the complete bipartite graphs are also called *biregular graphs*. Note that, for the complete bipartite graph $K_{|X|,|Y|}$, we have $d_X(v) = |Y|$ and $d_Y(v) = |X|$.



(a) A 2-regular graph (b) A 3-regular graph (c) Petersen Graph

Figure 1.10: Examples of regular graphs

1.5 Isomorphic Graphs

Definition 1.5.1 — Isomorphism of Two Graphs. An *isomorphism* of two graphs G and H is a bijective function $f : V(G) \to V(H)$ such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H.

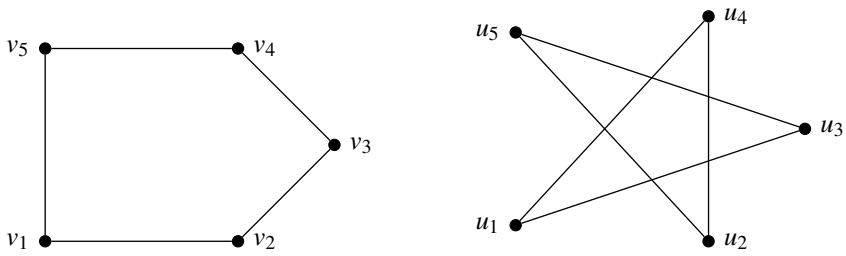
That is, two graphs G and H are said to be isomorphic if

(i) $|V(G)| = |V(H)|$,
(ii) $|E(G)| = |E(H)|$,
(iii) $v_i v_j \in E(G) \implies f(v_i)f(v_j) \in E(H)$.

This bijection is commonly described as *edge-preserving bijection*.

If an isomorphism exists between two graphs, then the graphs are called *isomorphic graphs* and denoted as $G \simeq H$ or $G \cong H$.

For example, consider the graphs given in Figure 1.11.



(a) G_1. (b) G_2.

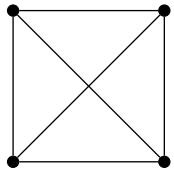
Figure 1.11: Examples of isomorphic graphs

In the above graphs, we can define an isomorphism f from the first graph to the second graph such that $f(v_1) = u_1, f(v_2) = u_3, f(v_3) = u_5, f(v_4) = u_2$ and $f(v_5) = u_4$. Hence, these two graphs are isomorphic.

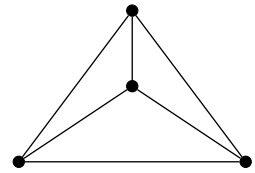
1.6 Exercises

1. Show that every loop-less graph G has a bipartite subgraph with at least $\frac{\varepsilon}{2}$ edges.
2. Verify whether graph isomorphism is an equivalence relation?
3. For $k > 0$, show that a k-regular bipartite graph has the same number of vertices in each of its partite sets.
4. Show that every simple graph on n vertices subgraph of K_n.
5. Show that every subgraph of a bipartite graph is bipartite.
6. Verify whether the integer sequences $(7,6,5,4,3,3,2)$ and $(6,6,5,4,3,3,1)$ are graphical.
7. Show that if G is simple and connected but not complete, then G has three vertices u, v and w such that $uv, vw \in E(G)$, but $uw \notin E$.
8. Show that every induced subgraph of a complete graph K_n is also a complete subgraph.
9. If G is an r-regular graph, then show that r divides the size of G.
10. Show that every subgraph of a bipartite graph is bipartite.
11. If G is an r-regular graph and r is odd, then show that $\frac{\varepsilon}{r}$ is an even integer.
12. Let G be a graph in which there is no pair of adjacent edges. What can you say about the degree of the vertices in G?
13. Check whether the following pairs of graphs are isomorphic? Justify your answer.

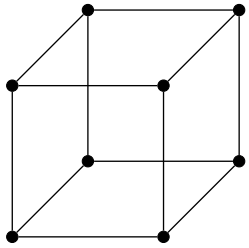
(a) G (b) H



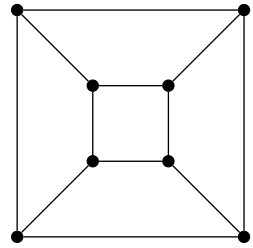
(a) G



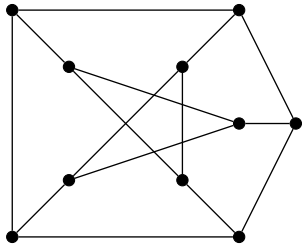
(b) H



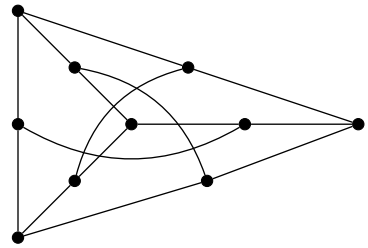
(a) G



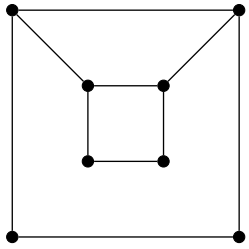
(b) H



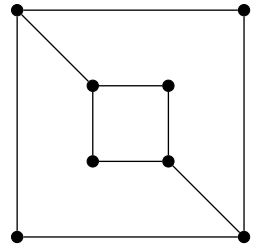
(a) G



(b) H



(a) G

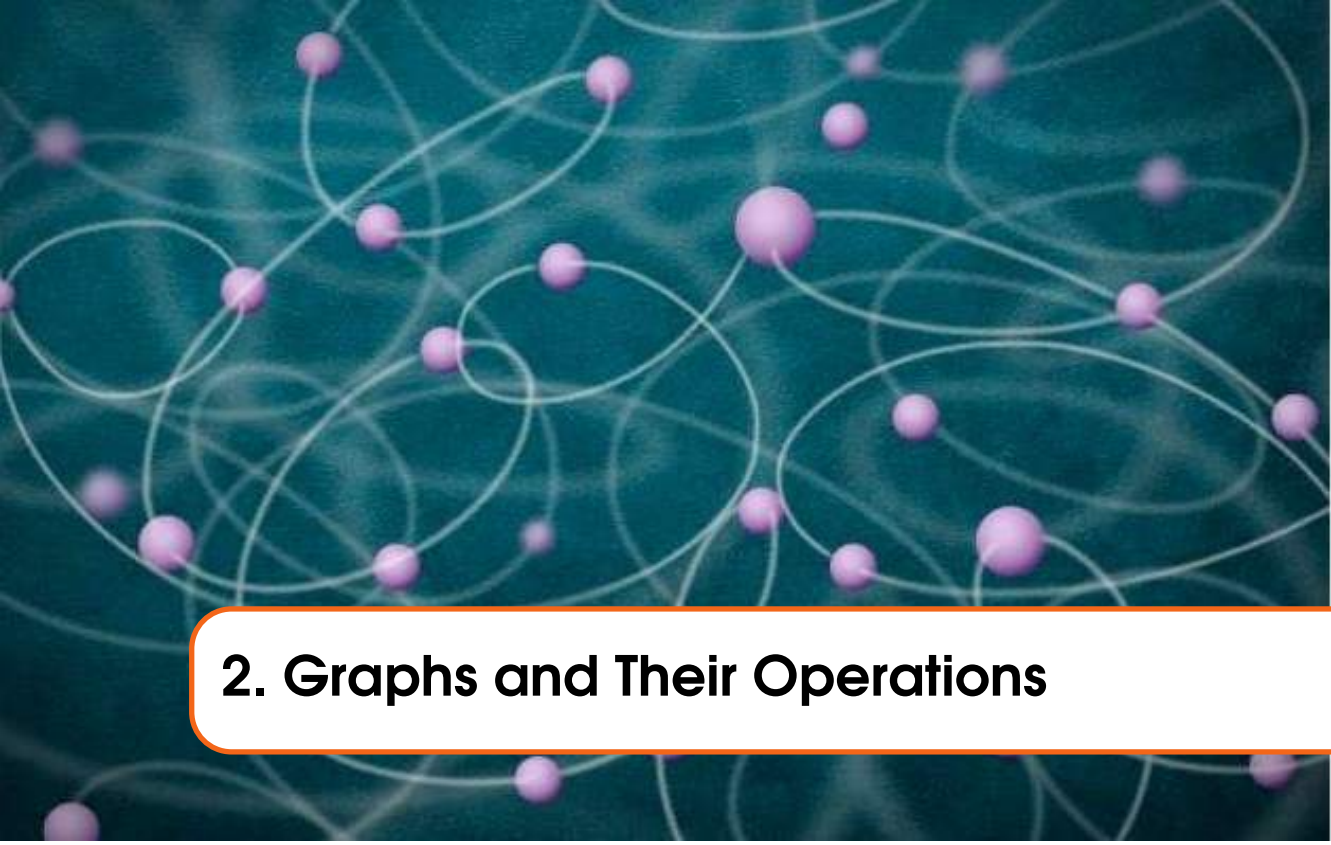


(b) H

14. Let G be a graph with n vertices and e edges and let m be the smallest positive integer such that $m \geq \frac{2e}{n}$. Prove that G has a vertex of degree at least m.
15. Prove that it is impossible to have a group of nine people at a party such that each one knows exactly five of the others in the group.
16. Let G be a graph with n vertices, t of which have degree k and the others have degree $k+1$. Prove that $t = (k+1)n - 2e$, where e is the number of edges in G.
17. Let G be a k-regular graph, where k is an odd number. Prove that the number of edges in G is a multiple of k.
18. Let G be a graph with n vertices and exactly $n-1$ edges. Prove that G has either a

vertex of degree 1 or an isolated vertex.

19. What is the smallest integer n such that the complete K_n has at least 500 edges?
20. Prove that there is no simple graph with six vertices, one of which has degree 2, two have degree 3, three have degree 4 and the remaining vertex has degree 5.
21. Prove that there is no simple graph on four vertices, three of which have degree 3 and the remaining vertex has degree 1.
22. Let G be a simple regular graph with n vertices and 24 edges. Find all possible values of n and give examples of G in each case.



2. Graphs and Their Operations

We have already seen that the notion of subgraphs can be defined for any graphs as similar to the definition of subsets to sets under consideration. Similar to the definitions of basic set operations, we can define the corresponding basic operations for graphs also. In addition to these fundamental graph operations, there are some other new and useful operations are also defined on graphs. In this chapter, we discuss some basic graph operation.

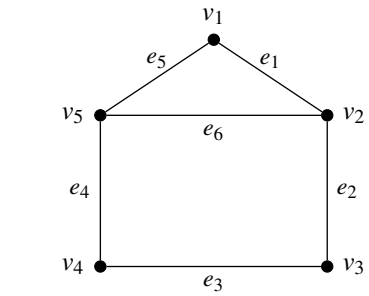
2.1 Union, Intersection and Ringsum of Graphs

Definition 2.1.1 — Union of Graphs. The *union* of two graphs G_1 and G_2 is a graph G, written by $G = G_1 \cup G_2$, with vertex set $V(G_1) \cup V(G_2)$ and the edge set $E(G_1) \cup E(G_2)$.

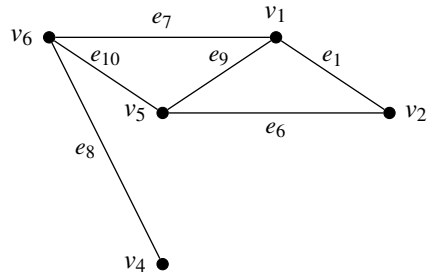
Definition 2.1.2 — Intersection of Graphs. The *intersection* of two graphs G_1 and G_2 is another graph G, written by $G = G_1 \cap G_2$, with vertex set $V(G_1) \cap V(G_2)$ and the edge set $E(G_1) \cap E(G_2)$.

Definition 2.1.3 — Ringsum of Graphs. The *ringsum* of two graphs G_1 and G_2 is another graph G, written by $G = G_1 \oplus G_2$, with vertex set $V(G_1) \cap V(G_2)$ and the edge set $E(G_1) \oplus E(G_2)$, where $\oplus$ is the symmetric difference (XOR Operation) of two sets.

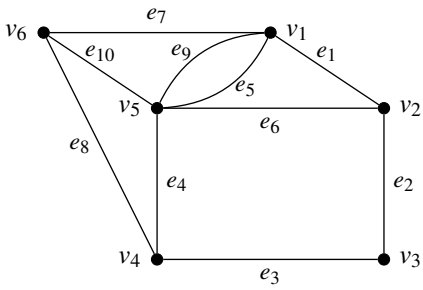
Figure 2.1 illustrates the union, intersection and ringsum of two given graphs.



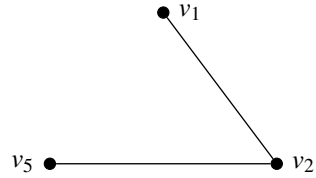
(a) G_1



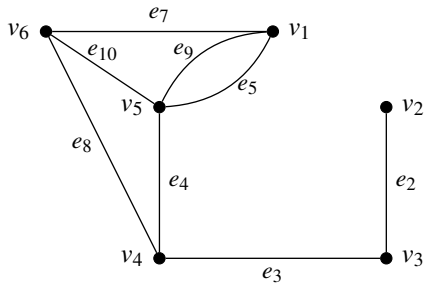
(b) G_2



(c) $G_1 \cup G_2$



(d) $G_1 \cap G_2$



(e) $G_1 \oplus G_2$

Figure 2.1: Illustrations to graph operations

1. The union, intersection and ringsum operations of graphs are commutative. That is, $G_1 \cup G_2 = G_2 \cup G_1, G_1 \cap G_2 = G_2 \cap G_1$ and $G_1 \oplus G_2 = G_2 \oplus G_1$.
2. If G_1 and G_2 are edge-disjoint, then $G_1 \cap G_2$ is a null graph, and $G_1 \oplus G_2 = G_1 \cup G_2$.
3. If G_1 and G_2 are vertex-disjoint, then $G_1 \oplus G_2$ is empty.
4. For any graph G, $G \cap G = G \cup G$ and $G \oplus G$ is a null graph.

Definition 2.1.4 — Decomposition of a Graph. A graph G is said to be *decomposed* into two subgraphs G_1 and G_2 , if $G_1 \cup G_2 = G$ and $G_1 \cap G_2$ is a null graph.

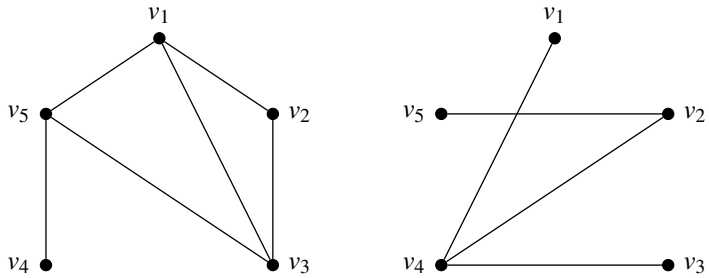
2.2 Complement of Graphs

Definition 2.2.1 — Complement of Graphs. The *complement* or *inverse* of a graph G, denoted by $\bar{G}$ is a graph with $V(G) = V(\bar{G})$ such that two distinct vertices of $\bar{G}$ are adjacent if and only if they are not adjacent in G.

R Note that for a graph G and its complement $\bar{G}$, we have

(i) $G_1 \cup \bar{G} = K_n$;
(ii) $V(G) = V(\bar{G})$;
(iii) $E(G) \cup E(\bar{G}) = E(K_n)$;
(iv) $|E(G)| + |E(\bar{G})| = |E(K_n)| = \binom{n}{2}$.

A graph and its complement are illustrated below.



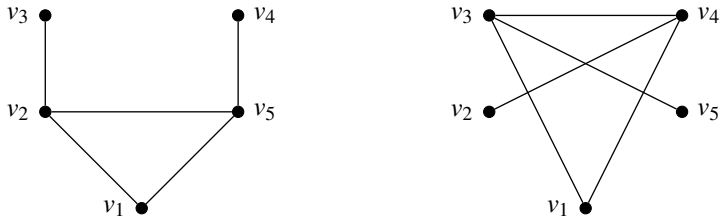
(a) G (b) $\bar{G}$

Figure 2.2: A graph and its complement

2.2.1 Self-Complementary Graphs

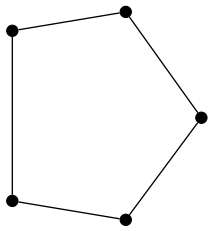
Definition 2.2.2 — Self-Complementary Graphs. A graph G is said to be *self-complementary* if G is isomorphic to its complement. If G is self complementary, then $|E(G)| = |E(\bar{G})| = \frac{1}{2}|E(K_n)| = \frac{1}{2}\binom{n}{2} = \frac{n(n-1)}{4}$.

The following are two examples of self complementary graphs.

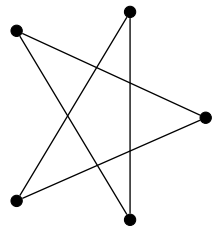


(a) Bull graph G. (b) $\bar{G}$.

Figure 2.3: Example of self-complementary graphs



(a) G_2



(b) G_2.

Figure 2.4: Example of self-complementary graphs

Problem 2.1 For any self-complementary graph G of order n, show that $n \equiv 0,1 \pmod 4$.

Solution: For self-complementary graphs, we have

(i) $V(G) = V(\bar{G})$;

(ii) $|E(G)| + |E(\bar{G})| = \frac{n(n-1)}{2}$;

(iii) $|E(G)| = |E(\bar{G})|$.

Therefore, $|E(G)| = |E(\bar{G})| = \frac{n(n-1)}{4}$. This implies, 4 divides either n or $n-1$. That is, for self-complementary graphs of order n, we have $n \equiv 0,1 \pmod 4$. ■

(Note that we say $a \equiv b \pmod n$, which is read as "a is congruent to b modulo n", if $a-b$ is completely divisible by n).

2.3 Join of Graphs

Definition 2.3.1 The *join* of two graphs G and H, denoted by $G+H$ is the graph such that $V(G+H) = V(G) \cup V(H)$ and $E(G+H) = E(G) \cup E(H) \cup \{xy : x \in V(G), y \in V(H)\}$.

In other words, the join of two graphs G and H is defined as the graph in which every edge of the first graph is adjacent to all vertices of the second graph.

Figure 2.5 illustrates the join of two graphs P_3 and P_4 and Figure 2.6 illustrates the join of two graphs C_5 and P_2.

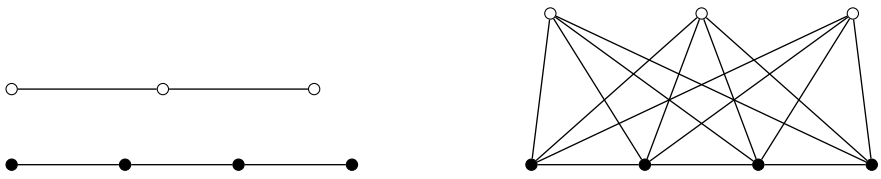
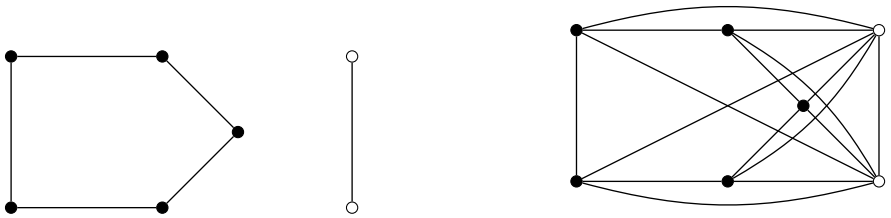


Figure 2.5: The join of the paths P_4 and P_3.

2.4 Deletion and Fusion

Definition 2.4.1 — Edge Deletion in Graphs. If e is an edge of G, then $G-e$ is the graph obtained by removing the edge of G. The subgraph of G thus obtained is called an *edge-deleted subgraph* of G. Clearly, $G-e$ is a spanning subgraph of G.



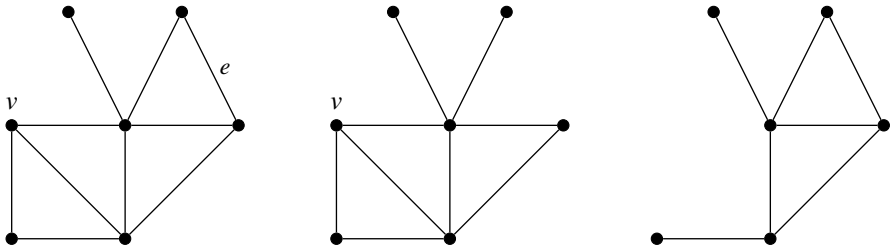
(a) C_5 (b) P_2 (c) $C_5 + P_2$.

Figure 2.6: The join of the cycle C_5 and the path P_2.

Similarly, vertex-deleted subgraph of a graph is defined as follows:

Definition 2.4.2 — Vertex Deletion in Graphs. If v is a vertex of G, then $G - v$ is the graph obtained by removing the vertex v and all edges G that are incident on v. The subgraph of G thus obtained is called an *vertex-deleted subgraph* of G. Clearly, $G - v$ will not be a spanning subgraph of G.

Figure 2.7 illustrates the edge deletion and the vertex deletion of a graph G.



(a) G (b) $G - e$ (c) $G - v$

Figure 2.7: Illustrations to edge deletion and vertex deletion

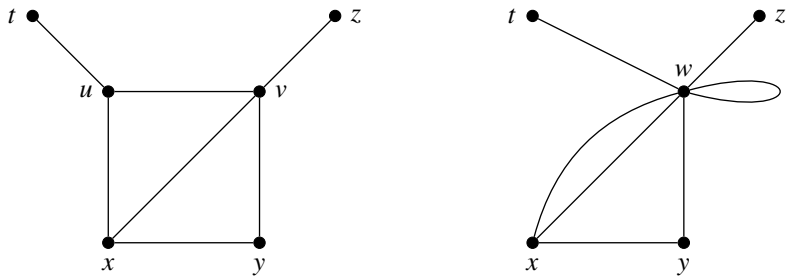
Definition 2.4.3 — Fusion of Vertices. A pair of vertices u and v are said to be *fused* (or *merged* or *identified*) together if the two vertices are together replaced by a single vertex w such that every edge incident with either u or v is incident with the new vertex w (see Figure 2.8).

Note that the fusion of two vertices does not alter the number of edges, but reduces the number of vertices by 1.

2.4.1 Edge Contraction

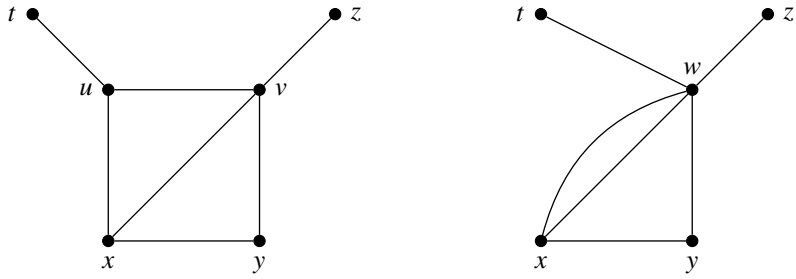
Definition 2.4.4 — Edge Contraction in Graphs. An *edge contraction* of a graph G is an operation which removes an edge from a graph while simultaneously merging its two end vertices that it previously joined. Vertex fusion is a less restrictive form of this operation.

A graph obtained by contracting an edge e of a graph G is denoted by $G \circ e$.



(a) G (b) $G-e$

Figure 2.8: Illustrations to fusion of two vertices



(a) G (b) $G \circ uv$

Figure 2.9: Illustrations to edge contraction of a graph.

2.5 Subdivision and Smoothing

2.5.1 Subdivision of a Graph

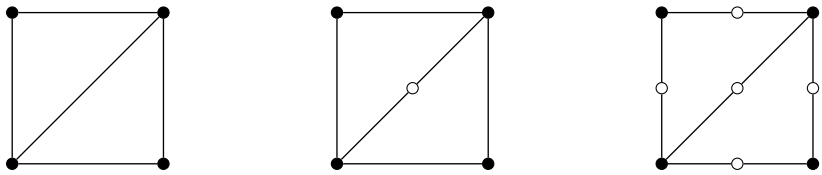
Definition 2.5.1 — Subdivision of an Edge. Let $e = uv$ be an arbitrary edge in G. The *subdivision* of the edge e yields a path of length 2 with end vertices u and v with a new internal vertex w (That is, the edge $e = uv$ is replaced by two new edges, uw and wv).

(a) G (b) G

Figure 2.10: Subdivision of an edge

Definition 2.5.2 — Subdivision of a Graph. A *subdivision* of a graph G (also known as an *expansion* of G) is a graph resulting from the subdivision of (some or all) edges in G (see 2.11). The newly introduced vertices in the subdivisions are represented by white vertices.

Definition 2.5.3 — Homeomorphic Graphs. Two graphs are said to be *homeomorphic* if both can be obtained by the same graph by subdivisions of edges.



(a) G (b) (c)

Figure 2.11: Illustrations to Subdivision of graphs

In Figure 2.11, the second and third graphs are homeomorphic, as they are obtained by subdividing the edges of the first graph in the figure.

2.5.2 Smoothing a Vertex

Definition 2.5.4 — Smoothing Vertices in Graphs. The reverse operation, *smoothing out* or *smoothing* a vertex w of degree 2 with regards to the pair of edges (e_i, e_j) incident on w, removes w and replaces the pair of edges (e_i, e_j) containing w with a new edge e that connects the other endpoints of the pair (e_i, e_j) (see the illustration).

(a) (b) G

Figure 2.12: Smoothing of the vertex w

Smoothing of a vertex of a graph G is also called an elementary transformation of G.

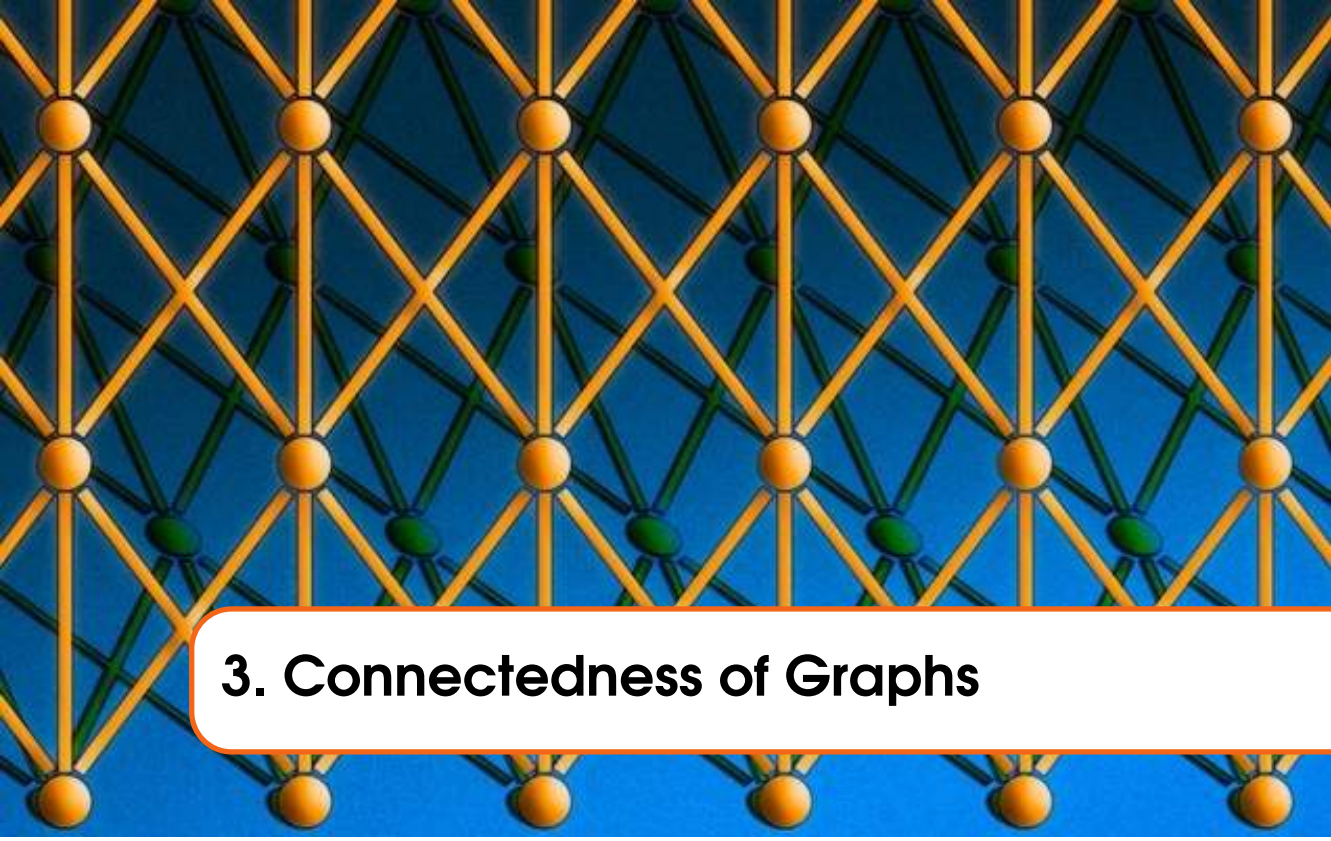
Problem 2.2 Show that a graph obtained by subdividing all edges of a graph G is a bipartite graph.

Solution: Let H be the subdivision of G. Let $X = V(G)$ and Y be the newly introduced vertices during subdivision. Clearly, $X \cup Y = V(H)$. Note that adjacency is not defined among the vertices of Y. When we subdivide an edge uv in G, then the edge uv will be removed and hence u and v becomes non-adjacent in H. Therefore, no two vertices in X can be adjacent in H. Thus, H is bipartite. ■

2.6 Exercises

1. Show that the complement of a complete bipartite graph is the disjoint union of two complete graphs.
2. The isomorphic image of a graph walk W is a walk of the same length.
3. For any graphs G and H, the ringsum $G \oplus H$ is empty if and only if $E(G) = E(H)$.
4. Show that the ringsum of two edge-disjoint collections of circuits is a collection of circuits.
5. For any graph G with six vertices, then G or its complement $\bar{G}$ contains a triangle.

6. Every graph G contains a bipartite spanning subgraph whose size is at least half the size of G.
7. Any graph G has a regular supergraph H of degree $\Delta(G)$ such that G is an induced subgraph of H.
8. how that if a self-complementary graph contains pendent vertex, then it must have at least another pendent vertex.
9. Draw all the non-isomorphic self complementary graphs on four vertices.
10. Prove that a graph with n vertices $(n > 2)$ cannot be bipartite if it has more than $\frac{n^2}{4}$ edges.
11. Verify whether the join of two bipartite graphs is bipartite. Justify your answer.
12. What is the order and size of the join of two graphs?
13. Does the join of two graphs hold commutativity? Illustrate with examples.



3. Connectedness of Graphs

3.1 Paths, Cycles and Distances in Graphs

Definition 3.1.1 — Walks. A *walk* in a graph G is an alternating sequence of vertices and connecting edges in G. In other words, a *walk* is any route through a graph from vertex to vertex along edges. If the starting and end vertices of a walk are the same, then such a trail is called a *closed walk*.

A walk can end on the same vertex on which it began or on a different vertex. A walk can travel over any edge and any vertex any number of times.

Definition 3.1.2 — Trails and Tours. A *trail* is a walk that does not pass over the same edge twice. A trail might visit the same vertex twice, but only if it comes and goes from a different edge each time. A *tour* is a trail that begins and ends on the same vertex.

Definition 3.1.3 — Paths and Cycles. A *path* is a walk that does not include any vertex twice, except that its first vertex might be the same as its last. A *cycle* or a *circuit* is a path that begins and ends on the same vertex.

Definition 3.1.4 — Length of Paths and Cycles. The *length* of a walk or circuit or path or cycle is the number of edges in it.

A path of order n is denoted by P_n and a cycle of order n is denoted by C_n. Every edge of G can be considered as a path of length 1. Note that the length of a path on n vertices is $n-1$.

A cycle having odd length is usually called an *odd cycle* and a cycle having even length is called an *even cycle*.

Definition 3.1.5 — Distance between two vertices. The *distance* between two vertices u and v in a graph G, denoted by $d_G(u,v)$ or simply $d(u,v)$, is the length (number of edges) of a *shortest path* (also called a *graph geodesic*) connecting them. This distance is also known as the *geodesic distance*.

Definition 3.1.6 — Eccentricity of a Vertex. The *eccentricity* of a vertex v, denoted by $\varepsilon(v)$, is the greatest geodesic distance between v and any other vertex. It can be thought of as how far a vertex is from the vertex most distant from it in the graph.

Definition 3.1.7 — Radius of a Graph. The *radius* r of a graph G, denoted by $rad(G)$, is the minimum eccentricity of any vertex in the graph. That is, $rad(G) = \min_{v \in V(G)} \varepsilon(v)$.

Definition 3.1.8 — Diameter of a Graph. The *diameter* of a graph G, denoted by $diam(G)$ is the maximum eccentricity of any vertex in the graph. That is, $diam(G) = \max_{v \in V(G)} \varepsilon(v)$.

Here, note that the diameter of a graph need not be twice its radius unlike in geometry. We can even see many graphs having same radius and diameter. Complete graphs are examples of the graphs with radius equals to diameter.

Definition 3.1.9 — Center of a Graph. A *center* of a graph G is a vertex of G whose eccentricity equal to the radius of G.

Definition 3.1.10 — Peripheral Vertex of a Graph. A *peripheral vertex* in a graph of diameter d is one that is distance d from some other vertex. That is, a peripheral vertex is a a vertex that achieves the diameter. More formally, a vertex v of G is peripheral vertex of a graph G, if $\varepsilon(v) = d$.

For a general graph, there may be several centers and a center is not necessarily on a diameter.

The distances between vertices in the above graph are given in Table 3.1. Note that a vertex v_i is represented by i in the table (to save the space).

Note that the radius of G is given by $r(G) = \min\{\varepsilon(v)\} = 4$ and the diameter of G is given by $diam(G) = \max\{\varepsilon(v)\} = 6$ and all eight central vertices are represented by white vertices in Figure 3.1.

Definition 3.1.11 — Geodetic Graph. A graph in which any two vertices are connected by a unique shortest path is called a *geodetic graph.*

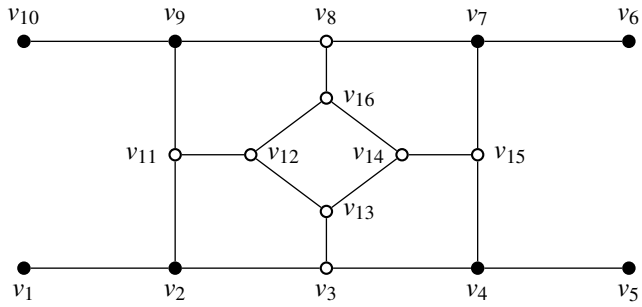


Figure 3.1: A graph with eight centers.

v	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	ε
1	0	1	2	3	4	6	5	4	3	4	2	3	3	4	4	4	6
2	1	0	1	2	3	5	4	3	2	3	1	2	2	3	3	3	5
3	2	1	0	1	2	4	3	4	3	4	2	2	1	1	2	3	4
4	3	2	1	0	1	3	2	3	4	5	5	3	2	2	1	3	5
5	4	3	2	1	0	4	3	4	5	6	4	4	3	3	2	4	6
6	6	5	4	3	4	0	1	2	3	4	4	4	4	3	2	3	6
7	5	4	3	2	3	1	0	1	2	3	3	3	3	2	1	2	5
8	4	3	4	3	4	2	1	0	1	2	2	2	3	2	2	1	4
9	3	2	3	4	5	3	2	1	0	1	1	2	3	3	3	2	5
10	4	3	4	5	6	4	3	2	1	0	2	3	4	4	4	3	6
11	2	1	2	3	4	4	3	2	1	1	0	1	2	3	4	2	4
12	3	2	2	3	4	4	2	1	2	3	1	0	1	2	3	1	4
13	3	2	1	2	3	4	3	3	3	4	2	1	0	1	2	2	4
14	4	3	2	2	3	3	2	2	3	4	3	2	1	0	1	1	4
15	4	3	2	1	2	2	1	2	3	4	4	3	2	1	0	2	4
16	4	3	3	3	4	3	2	1	2	3	2	3	2	1	2	0	4

Table 3.1: Eccentricities of vertices of the graph in Figure 3.1.

Theorem 3.1.1 If G is a simple graph with $diam(G) \geq 3$, then $diam(\bar{G}) \leq 3$.

Proof. If $diam(G) \geq 3$, then there exist at least two non-adjacent vertices u and v in G such that u and v have no common neighbours in G. Hence, every vertex x in $G - \{u,v\}$ is non-adjacent to u or v or both in G. This makes x adjacent to u or v or both in $\bar{G}$. Moreover, $uv \in E(\bar{G})$. So, for every pair of vertices x,y, there is an x,y path of length at most 3 in $\bar{G}$ through the edge uv. Hence, $diam(\bar{G}) \leq 3$. ■

3.2 Connected Graphs

Definition 3.2.1 — Connectedness in a Graph. Two vertices u and v are said to be *connected* if there exists a path between them. If there is a path between two vertices u and v, then u is said to be *reachable* from v and vice versa. A graph G is said to be *connected*

if there exist paths between any two vertices in G.

Definition 3.2.2 — Component of a Graph. A connected *component* or simply, a *component* of a graph G is a maximal connected subgraph of G.

Each vertex belongs to exactly one connected component, as does each edge. A connected graph has only one component.

A graph having more than one component is a *disconnected graph* (In other words, a disconnected graph is a graph which is not connected). The number of components of a graph G is denoted by $\omega(G)$.

In view of the above notions, the following theorem characterises bipartite graphs.

Theorem 3.2.1 A connected graph G is bipartite if and only if G has no odd cycles.

Proof. Suppose that G is a bipartite graph with bipartition (X,Y). Assume for contradiction that there exists a cycle $v_1, v_2, v_3, \ldots\ldots, v_k, v_1$ in G with k odd. Without loss of generality, we may additionally assume that $v_1 \in X$. Since G is bipartite, $v_2 \in Y$, $v_3 \in X$, $v_4 \in Y$ and so on. That is, $v_i \in X$ for odd values of i and $v_i \in Y$ for even values of i. Therefore, $v_k \in X$. But, then the edge $v_k, v_1 \in E$ is an edge with both endpoints in X, which contradicts the fact that G is bipartite. Hence, a bipartite graph G has no odd cycles.

Conversely, assume that G is a graph with no odd cycles. Let $d(u,v)$ denote the distance between two vertices u and v in G. Pick an arbitrary vertex $u \in V$ and define $X = \{x \in V(G) : d(x,u) \text{ is even}\}$. Clearly, $u \in X$ as $d(u,u) = 0$. Now, define another $Y = \{y \in V(G) : d(u,y) \text{ is odd}\}$. That is, $Y = V - X$. If possible, assume that there exists an edge $vw \in E(G)$ such that $v, w \in X$ (or $v, w \in Y$). Then, by construction $d(u,v)$ and $d(u,w)$ are both even (or odd). Let $P(u,w)$ and $P(u,v)$ be the shortest paths connecting u to w, and u to v respectively. Then, the cycle given by $P(u,w) \cup \{wv\} \cup P(v,u)$ has odd length $1 + d(u,w) + d(u,v)$, which is a contradiction. Therefore, no such edge wv may exist and G is bipartite. ■

Theorem 3.2.2 A graph G is disconnected if and only if its vertex set V can be partitioned into two non-empty, disjoint subsets V_1 and V_2 such that there exists no edge in G whose one end vertex is in subset V_1 and the other in the subset V_2.

Proof. Suppose that such a partitioning exists. Consider two arbitrary vertices u and v of G, such that $u \in V_1$ and $v \in V_2$. No path can exist between vertices u and v; otherwise, there would be at least one edge whose one end vertex would be in V_1 and the other in V_2. Hence, if a partition exists, G is not connected.

Conversely, assume that G is a disconnected graph. Consider a vertex u in G. Let V_1 be the set of all vertices that are joined by paths to u. Since G is disconnected, V_1 does not include all vertices of G. The remaining vertices will form a (nonempty) set V_2. No vertex in V_1 is joined to any vertex in V_2 by an edge. Hence, we get the required partition. ■

Theorem 3.2.3 If a graph has exactly has two vertices of odd degree, then there exists a path joining these two vertices.

Proof. Let G be a graph with two vertices v_1 and v_2 of odd degree and all other vertices of even degree. Then, by Theorem 1.2.3, both of them should lie in the same component of G. Since every component of G must be connected, there must be a path between v_1 and v_2. ■

Theorem 3.2.4 Let G be a graph with n vertices and k components. Then, G has at most $\frac{1}{2}(n-k)(n-k+1)$ edges.

Proof. Let G be a graph with n vertices and k components. Let the number of vertices in each of the k components of G be $n_1, n_2, \ldots, n_k$ respectively. Then we have,

$$n_1 + n_2 + \ldots + n_k = n;\ n_i \geq 1 \tag{3.1}$$

First, note that any connected graph on n vertices must have at least $n-1$ edges. The proof of the theorem is based on the inequality $\sum_{i=1}^{k} n_i^2 \leq n^2 - (k-1)(2n-k)$, which can be proved as follows.

$$\begin{aligned}
\sum_{i=1}^{k}(n_i - 1) &= n-k \\
\left(\sum_{i=1}^{k}(n_i - 1)\right)^2 &= (n-k)^2 \\
\sum_{i=1}^{k}(n_i^2 - 2n_i) + k + \textit{non-negative cross terms} &= n^2 + k^2 - 2nk \\
\sum_{i=1}^{k} n_i^2 - 2\sum_{i=1}^{k} n_i + k &\leq n^2 + k^2 - 2nk \\
\sum_{i=1}^{k} n_i^2 - 2n + k &\leq n^2 + k^2 - 2nk \\
\sum_{i=1}^{k} n_i^2 &\leq n^2 + k^2 - 2nk + 2n - k \\
\therefore \sum_{i=1}^{k} n_i^2 &\leq n^2 - (k-1)(2n-k).
\end{aligned}$$

Hence, we have

$$\sum_{i=1}^{k} n_i^2 \leq n^2 - (k-1)(2n-k) \tag{3.2}$$

Now, note that the number edges in K_n is $\frac{n(n-1)}{2}$. Hence, the maximum number of edges in i-th component of G (which is a simple connected graph) is $\frac{n_i(n_i-1)}{2}$. Therefore, the maximum

number of edges in G is

$$\begin{aligned}
\sum_{i=1}^{k} \frac{n_i(n_i-1)}{2} &= \sum_{i=1}^{k} \frac{n_i^2-n_i}{2} \\
&= \frac{1}{2}\sum_{i=1}^{k}(n_i^2-n_i) \\
&= \frac{1}{2}\left[\sum_{i=1}^{k} n_i^2 - \sum_{i=1}^{k} n_i\right] \\
&\leq \frac{1}{2}\left[n^2-(k-1)(2n-k)-n\right] \text{ (By Eq. (3.1) and Ineq. (3.2))} \\
&= \frac{1}{2}\left[n^2-2nk+k^2+2n-k-n\right] \\
&= \frac{1}{2}\left[n^2-2nk+k^2+n-k\right] \\
&= \frac{1}{2}\left[(n-k)^2+(n-k)\right] \\
&= \frac{1}{2}(n-k)(n-k+1).
\end{aligned}$$

■

Problem 3.1 Show that an acyclic graph on n vertices and k components has $n-k$ edges.

Solution. The solution follows directly from the first part of the above theorem.

Problem 3.2 Show that every graph on n vertices having more than $\frac{(n-1)(n-2)}{2}$ edges is connected.

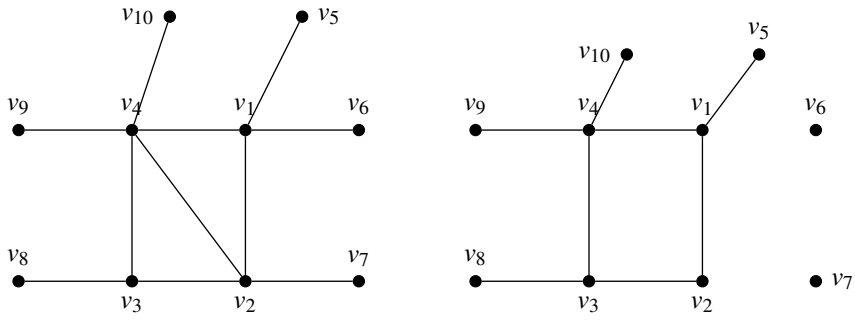
Solution. Consider the complete graph K_n and v be an arbitrary vertex of K_n. Now remove all $n-1$ edges of K_n incident on v so that it becomes disconnected with K_{n-1} as one component and the isolated vertex v as the second component. Clearly, this disconnected graph has $\frac{(n-1)(n-2)}{2}$ edges (all of which belong to the first component). Since all pairs of vertices in the first component K_{n-1} are any adjacent to each other, any new edge drawn must be joining a vertex in K_{n-1} and the isolated vertex v, making the revised graph connected. ■

3.3 Edge Deleted and Vertex Deleted Subgraphs

Definition 3.3.1 — Edge Deleted Subgraphs. Let $G(V,E)$ be a graph and $F \subseteq E$ be a set of edges of G. Then, the graph obtained by deleting F from G, denoted by $G-F$, is the subgraph of G obtained from G by removing all edges in F. Note that $V(G-F) = V(G)$. That is, $G-F = (V, E-F)$.

Note that any edge deleted subgraph of a graph G is a spanning subgraph of G.

Definition 3.3.2 — Vertex Deleted Subgraphs. Let $W \subseteq V(G)$ be a set of vertices of G. Then the graph obtained by deleting W from G, denoted by $G-W$, is the subgraph of G

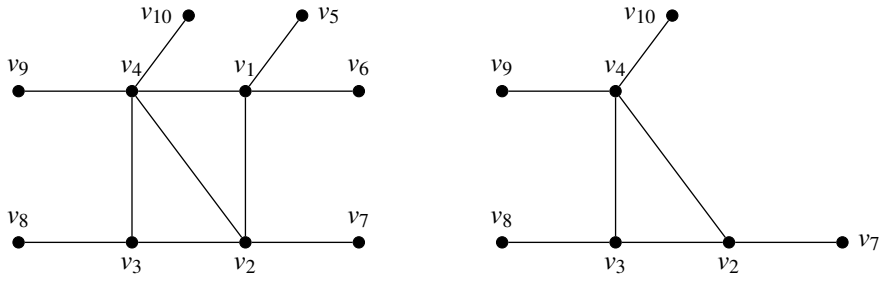


(a) G (b) $G' = G - \{v_1v_6, v_2v_4, v_2v_7\}$.

Figure 3.2: A graph and its edge deleted subgraph.

obtained from G by removing all vertices in W and all edges incident to those vertices.

See Figure 3.3 for illustration of a vertex-deleted subgraph of a given graph.



(a) G (b) $G' = G - \{v_1, v_5, v_6\}$.

Figure 3.3: A graph and its edge deleted subgraph.

Cut-Edges and Cut-Vertices

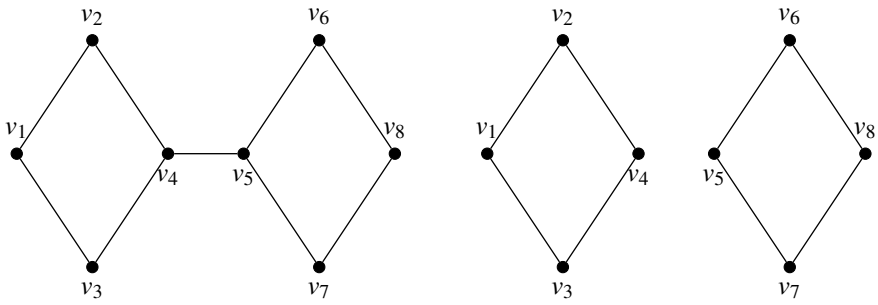
Definition 3.3.3 — Cut-Edge. An edge e of a graph G is said to be a *cut-edge* or a *bridge* of G if $G - e$ is disconnected.

In the above graph G, the edge v_4v_5 is a cut-edge, since $G - v_4v_5$ is a disconnected graph.

The following is a necessary and sufficient condition for an edge of a graph G to be a cut edge of G.

Theorem 3.3.1 An edge e of a graph G is a cut-edge of G if and only if it is not contained in any cycle of G.

Proof. Let $e = uv$ be a cut edge of G. Then, the vertices u and v must be in different components of $G - e$. If possible, let e is contained in cycle C in G. Then, $C - e$ is a path between u and v in $G - e$, a contradiction to the fact that u and v are in different components of $G - e$. Therefore, e can not be in any cycle of G.

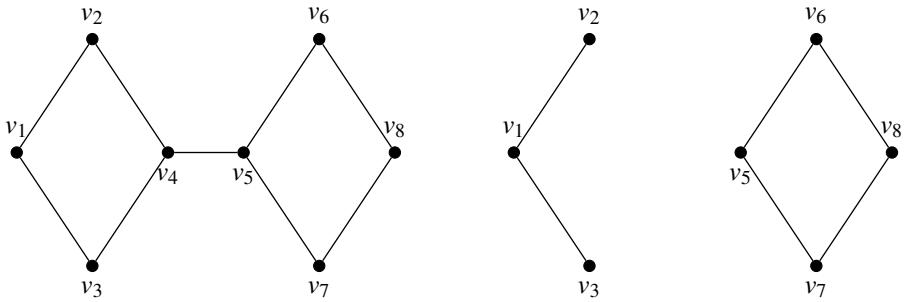


(a) Connected graph G (b) $G \circ uv$

Figure 3.4: Disconnected graph $G - v_4v_5$

Conversely, assume that e is not in any cycle of G. Then, there is no (u,v)-path other than e. Therefore, u and v are in different components of $G-e$. That is, $G-e$ is disconnected and hence e is a cut-edge of G. ■

Definition 3.3.4 — Cut-Vertex. A vertex v of a graph G is said to be a *cut-vertex* of G if $G-v$ is disconnected.



(a) Connected graph G (b) $G \circ uv$

Figure 3.5: disconnected graph $G - v_4$

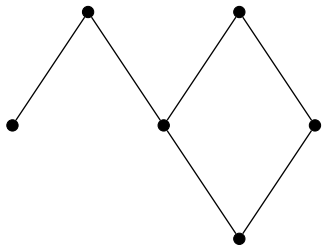
In graph G, v_4 is a cut-vertex as $G-v_4$ is a disconnected graph. Similarly, v_5 is also a cut-vertex of G.

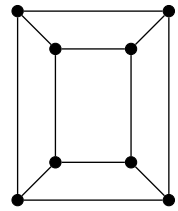
Since removal of any pendent vertex will not disconnect a given graph, every cut-vertex will have degree greater than or equal to 2. But, note that every vertex v, with $d(v) \geq 2$ need not be a cut-vertex.

3.4 Exercises

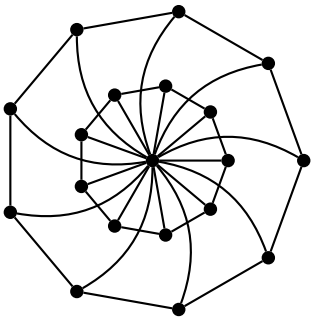
1. Show that every uv-walk contains a uv-path.
2. Show that every closed walk contains a cycle.

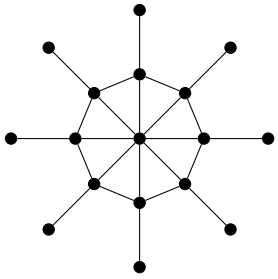
3. Show that every graph with n vertices and k edges, $n > k$ has $n-k$ components.
4. If every vertex of a graph G has degree greater than or equal to 2, then G has some cycles.
5. If G is a simple graph with $d(v) \geq k$, $\forall v \in V(G)$, then G contains a path of length at least k. If $k \geq 2$, then G contains a cycle of length $k+1$.
6. Show that if G is simple and $\delta(G) \geq k$, then G has a path of length k.
7. If a connected graph G is decomposed into two subgraphs G_1 and G_2, then show that there must be at least one common vertex between G_1 and G_2.
8. If we remove an edge e from a graph G and $G-e$ is still connected, then show that e lies along some cycle of G.
9. If the intersection of two paths is a disconnected graph, then show that the union of the two paths has at least one circuit.
10. If P_1 and P_2 are two different paths between two given vertices of a graph G, then show that $P_1 \oplus P_2$ is a circuit or a set of circuits in G.
11. Show that the complement of a complete bipartite graph is the disjoint union of two complete graphs.
12. For a simple graph G, with n vertices, if $\delta(G) = \frac{n-1}{2}$, then G is connected.
13. Show that any two longest paths in a connected graph have a vertex in common.
14. For $k \geq 2$, prove that a k-regular bipartite graph has no cut-edge.
15. Determine the maximum number of edges in a bipartite subgraph of the Petersen graph.
16. If H is a subgraph of G, then show that $d_G(u,v) \leq d_H(u,v)$.
17. Prove that if a connected graph G has equal order and size, then G is a cycle.
18. Show that eccentricities of adjacent vertices differ by at most 1.
19. Prove that if a graph has more edges than vertices then it must possess at least one cycle.
20. If the intersection of two paths is a disconnected graph, then show that the union of the two paths has at least one cycle.
21. The radius and diameter of a graph are related as $rad(G) \leq diam(G) \leq 2r(G)$.

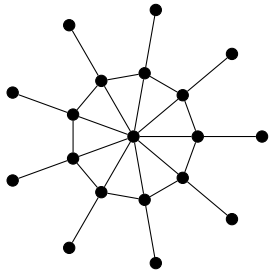


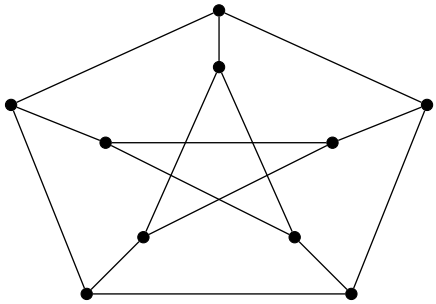


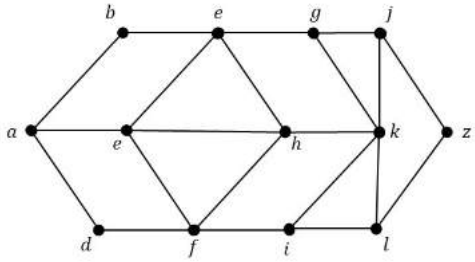
22. Find the eccentricity of the vertices and the radius, the diameter and center(s) of the following graphs:










b
e
g
j
a
e
h
k
z
d
f
i
l

II TRAVERSABILITY IN GRAPHS & DIGRAPHS

4. Traversability in Graphs

4.1 Königsberg Seven Bridge Problem

The city of *Königsberg* in Prussia (now Kaliningrad, Russia) was situated on either sides of the *Pregel River* and included two large islands which were connected to each other and the mainlands by *seven bridges* (see the below picture).

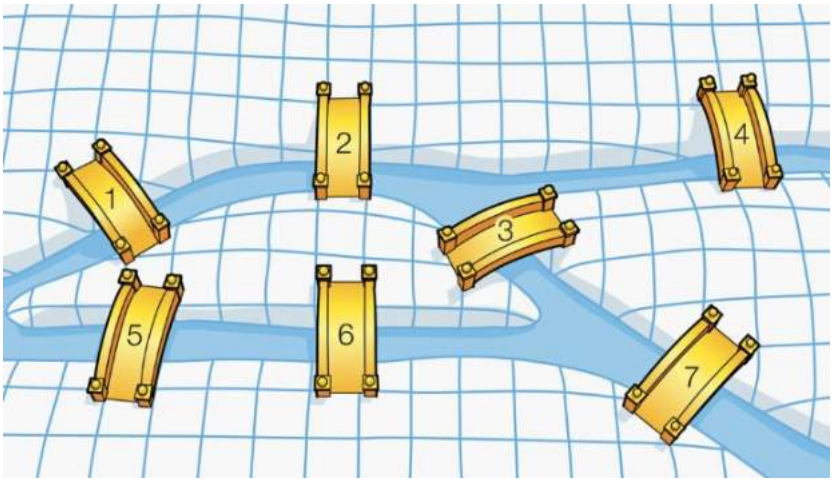


Figure 4.1: Königsberg's seven bridge problem.

The problem was to devise a walk through the city that would cross each bridge once and

only once, subject to the following conditions:

(i) The islands could only be reached by the bridges;

(ii) Every bridge once accessed must be crossed to its other end;

(iii) The starting and ending points of the walk are the same.

The Königsberg seven bridge problem was instrumental to the origination of Graph Theory as a branch of modern Mathematics.

In 1736, a Swiss Mathematician *Leonard Euler* introduced a graphical model to this problem by representing each land area by a vertex and each bridge by an edge connecting corresponding vertices (see the following figure).

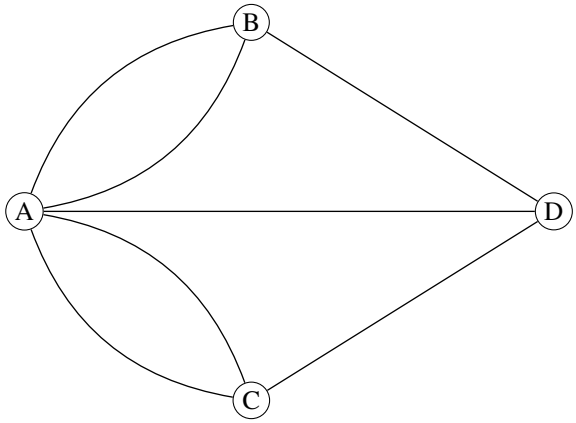


Figure 4.2: Graphical representation of seven bridge problem

Using this graphical model, Euler proved that no such walk (trail) exists.

4.2 Eulerian Graphs

Definition 4.2.1 — Traversable Graph. An *Eulerian trail* or *Euler walk* in an undirected graph is a walk that uses each edge exactly once. If an Euler trail exists in a given graph G, then G is called a *traversable graph* or a *semi-Eulerian graph*.

Definition 4.2.2 — Eulerian Graph. An *Eulerian cycle* or *Eulerian circuit* or *Euler tour* in an undirected graph is a cycle that uses each edge exactly once. If such an Euler cycle exists in the graph concerned, then the graph is called an *Eulerian graph* or a *unicursal graph*.

The following theorem characterises the class of Eulerian graphs:

Theorem 4.2.1 — Euler Theorem. A connected graph G is Eulerian if and only if every vertex in G is of even degree.

Proof. If G is Eulerian, then there is an Euler circuit, say P, in G. Every time a vertex is listed, that accounts for two edges adjacent to that vertex, the one before it in the list and the

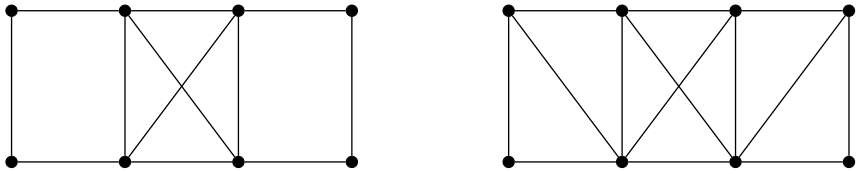
one after it in the list. This circuit uses every edge exactly once. So, every edge is accounted for and there are no repeats. Thus every degree must be even.

Conversely, let us assume that each vertex of G has even degree. We need to show that G is Eulerian. We prove the result by induction on the number of edges of G. Let us start with a vertex $v_0 \in V(G)$. As G is connected, there exists a vertex $v_1 \in V(G)$ that is adjacent to v_0. Since G is a simple graph and $d(v_i) \geq 2$, for each vertex $v_i \in V(G)$, there exists a vertex $v2 \in V(G)$, that is adjacent to v_1 with $v_2 \neq v_0$. Similarly, there exists a vertex $v_3 \in V(G)$, that is adjacent to v_2 with $v_3 \neq v_1$. Note that either $v_3 = v_0$, in which case, we have a circuit $v_0v_1v_2v_0$ or else one can proceed as above to get a vertex $v_4 \in V(G)$ and so on. As the number of vertices is finite, the process of getting a new vertex will finally end with a vertex v_i being adjacent to a vertex v_k, for some $i,\ 0 \leq i \leq k-2$. Hence, $v_i - v_{i+1} - v_{i+2} - \ldots - v_k - v_i$ forms a circuit, say C, in G.

If C contains every edge of G, then C gives rise to a closed Eulerian trail and we are done. So, let us assume that $E(C)$ is a proper subset of $E(G)$. Now, consider the graph G_1 that is obtained by removing all the edges in C from G. Then, G_1 may be a disconnected graph but each vertex of G_1 still has even degree. Hence, we can do the same process explained above to G_1 also to get a closed Eulerian trail, say C_1. As each component of G_1 has at least one vertex in common with C, if C_1 contains all edges of G_1, then $C \cup C_1$ is a closed Euler trail in G. If not, let G_2 be the graph obtained by removing the edges of C_1 from G_1. (That is, $G_2 = G_1 - E(C \cup C_1)$).

Since G is a finite graph, we can proceed to find out a finite number of cycles only. Let the process of finding cycles, as explained above, ends after a finite number of steps, say r. Then, the reduced graph $G_r = G_{r-1} - E(C_{r-1}) = G - E(C \cup C_1 \cup C_{r-1})$ will be an empty graph (null graph). Then, $C \cup C_1 \cup C_2 \ldots \cup C_{r-1}$ is a closed Euler trail in G. Therefore, G is Eulerian. This completes the proof. ■

Illustrations to an Eulerian graph and a non-Eulerian graph are given in Figure 4.3.



(a) An Eulerian graph. (b) A non-Eulerian graph

Figure 4.3: Examples of Eulerian and non-Eulerian graphs

In the first graph in Figure 4.3, every vertex has even degree and hence by Theorem 4.2.1, it is Eulerian. In the second graph, some vertices have odd degree and hence it is not Eulerian.

Note: In an Euler graph, it can be noted that every edge of G is contained in exactly one cycle of G. Hence, we have the following Theorem.

Theorem 4.2.2 A connected graph G is Eulerian if and only if it can be decomposed into edge-disjoint cycles.

Proof. Assume that G can be decomposed into edge-disjoint cycles. Since the degree of every vertex in a cycle is 2, the degree of every vertex in G is two or multiples of 2. That is, all vertices in G are even degree vertices. Then, by Theorem 4.2.1, G is Eulerian.

Converse part is exactly the same as that of Theorem 4.2.1. ■

Theorem 4.2.3 A connected graph G is traversable if and only if it has exactly two odd degree vertices.

Proof. In a traversable graph, there must be an Euler trail. The starting vertex and terminal vertex need not be the same. Therefore, these two vertices can have odd degrees. Remaining part of the theorem is exactly as in the proof of Theorem 4.2.1. ■

4.3 Chinese Postman Problem

In his job, a postman picks up mail at the post office, delivers it, and then returns to the post office. He must, of course, cover each street in his area at least once. Subject to this condition, he wishes to choose his route in such a way that walks as little as possible. This problem is known as the Chinese postman problem, since it was first considered by a Chinese mathematician, Guan in 1960.

We refer to the street system as a weighted graph (G,w) whose vertices represent the intersections of the streets, whose edges represent the streets (one-way or two-way) and the weight represents the distance between two intersections, of course, a positive real number. A closed walk that covers each edge at least once in G is called a *postman tour*. Clearly, the Chinese postman problem is just that of finding a minimum-weight postman tour. We will refer to such a postman tour as an optimal tour.

An algorithm for finding an optimal Chinese postman route is as follows:

S-1 : List all odd vertices.
S-2 : List all possible pairings of odd vertices.
S-3 : For each pairing find the edges that connect the vertices with the minimum weight.
S-4 : Find the pairings such that the sum of the weights is minimised.
S-5 : On the original graph add the edges that have been found in Step 4.
S-6 : The length of an optimal Chinese postman route is the sum of all the edges added to the total found in Step 4.
S-7 : A route corresponding to this minimum weight can then be easily found.

■ **Example 4.1** Consider the following weighted graph:

1. The odd vertices are A and H; There is only one way of pairing these odd vertices, namely AH;
2. The shortest way of joining A to H is using the path $\{AB, BF, FH\}$, a total length of 160;

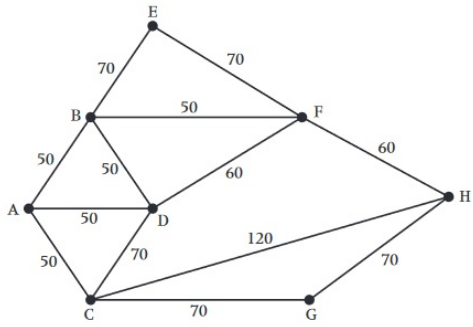


Figure 4.4: An example for Chinese Postman Problem

3. Draw these edges onto the original network.
4. The length of the optimal Chinese postman route is the sum of all the edges in the original network, which is $840m$, plus the answer found in Step 4, which is $160m$. Hence the length of the optimal Chinese postman route is $1000m$.
5. One possible route corresponding to this length is $ADCGHCABDFBEFHFBA$, but many other possible routes of the same minimum length can be found.

■

4.4 Hamiltonian Graphs

Definition 4.4.1 — Traceable Graphs. A *Hamiltonian path* (or *traceable path*) is a path in an undirected (or directed graph) that visits each vertex exactly once. A graph that contains a Hamiltonian path is called a *traceable graph*.

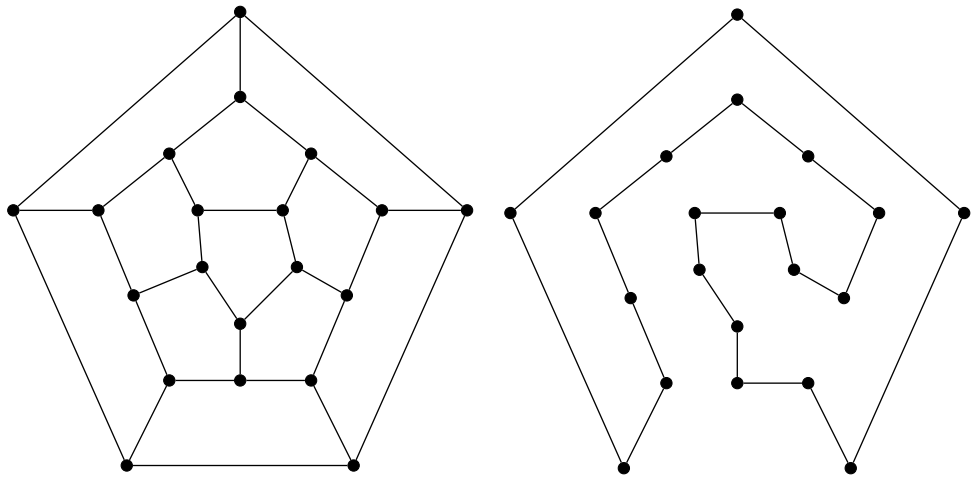
Definition 4.4.2 — Hamiltonian Graphs. A *Hamiltonian cycle*, or a *Hamiltonian circuit*, or a *vertex tour* or a *graph cycle* is a cycle that visits each vertex exactly once (except for the vertex that is both the start and end, which is visited twice). A graph that contains a Hamiltonian cycle is called a *Hamiltonian graph*.

Hamiltonian graphs are named after the famous mathematician *William Rowan Hamilton* who invented the Hamilton's puzzle, which involves finding a Hamiltonian cycle in the edge graph of the dodecahedron.

A necessary and sufficient condition for a graph to be a Hamiltonian is still to be determined. But there are a few sufficient conditions for certain graphs to be Hamiltonian. The following theorem is one of those results.

Theorem 4.4.1 — Dirac's Theorem. Every graph G with $n \geq 3$ vertices and minimum degree $\delta(G) \geq \frac{n}{2}$ has a Hamilton cycle.

Proof. Suppose that $G = (V, E)$ satisfies the hypotheses of the theorem. Then G is connected, since otherwise the degree of any vertex in a smallest component C of G would be at most

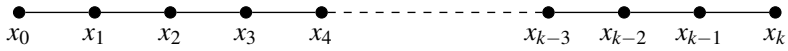


(a) (b)

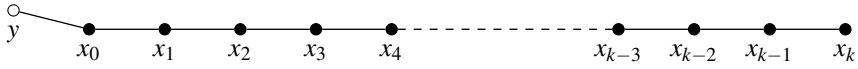
Figure 4.5: Dodecahedron and a Hamilton cycle in it.

$|C|-1<\frac{n}{2}$, contradicting the hypothesis $\delta(G)\geq\frac{n}{2}$.

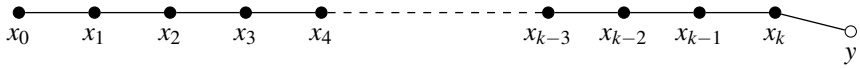
Let $P=x_0x_1\ldots x_k$ be a longest path in G, as seen in the figure given below:



Note that the length of P is k. Since P cannot be extended to a longer path, all the neighbours of x_0 lie on P. Assume the contrary. Let y be an adjacent vertex of x_0 which is not in P. Then, the path $P'=yx_0,x_2\ldots x_k$ is a path of length $k+1$ (see the graph given below), contradicting the hypothesis that P is the longest path in G.



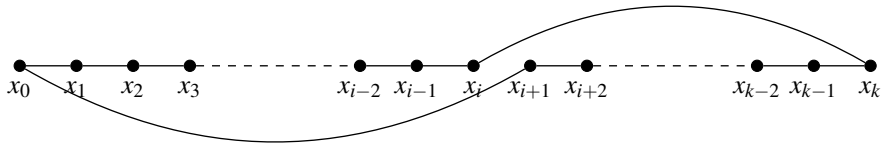
Similarly, we note that and all the neighbours of x_k will also lie on P, unless we reach at a contradiction as mentioned above (see the graph given below).



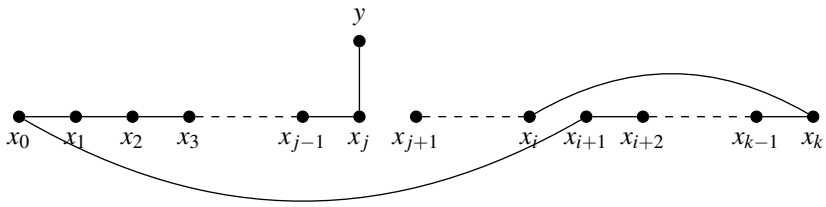
Hence, at least $\frac{n}{2}$ of the vertices $x_0,\ldots,x_{k-1}$ are adjacent to x_k, and at least $\frac{n}{2}$ of the vertices $x_1,\ldots,x_k$ are adjacent to x_0. Another way of saying the second part of the last sentence is: at least $\frac{n}{2}$ of the vertices $x_i\in\{x_0,\ldots,x_{k-1}\}$ are such that $x_0x_{i+1}\in E$. Combining both statements and using the pigeon-hole principle, we see that there is some x_i with $0\leq i\leq k-1$, $x_ix_k\in E$ and $x_0x_{i+1}\in E$.

Consider the cycle $C=x_0x_{i+1}x_{i+2}\ldots x_{k-1}x_kx_ix_{i-1}\ldots x_1x_0$ as given in the following graph.

We claim that the above cycle C is a Hamilton cycle of G. Otherwise, since G is connected, there would be some vertex x_j of C adjacent to a vertex y not in C, so that $e=x_jy\in E$. But



then we could attach e to a path ending in x_j containing k edges of C, constructing a path in G longer than P (see the graph given below), which is a contradiction to the hypothesis that P is the longest path in G.



Therefore, C must cover all vertices of G and hence it is a Hamiltonian cycle in G. This completes the proof. ■

Theorem 4.4.2 — Ore's Theorem. Let G be a graph with n vertices and let u and v be non-adjacent vertices in G such that $d(u)+d(v) \geq n$. Let $G+uv$ denote the super graph of G obtained by joining u and v by an edge. Then G is Hamiltonian if and only if $G+uv$ is Hamiltonian.

Proof. Let G be a graph with n vertices and suppose u and v are non-adjacent vertices in G such that $d(u)+d(v) \geq n$. Let $G' = G+uv$ be the super graph of G obtained by adding the edge uv. Note that, except for u and v, $d_G(x) = d_{G'}(x) \forall x \in V(G)$.

Let G be Hamiltonian. The only difference between G and G' is the edge uv. Then, obviously G' is also Hamiltonian as a Hamilton cycle in G will be a Hamilton cycle in G' as well.

Conversely, let G' be Hamiltonian. We have to show that G is Hamiltonian. Assume the contrary. Then, by (contrapositive of) Dirac's Theorem, we have $\delta(u) < \frac{n}{2}$ and $\delta(u) < \frac{n}{2}$ and hence we have $d(u)+d(v) < n$, which contradicts the hypothesis that $d(u)+d(v) \geq n$. Hence G is Hamiltonian. ■

The following theorem determines the number of edge-disjoint Hamilton cycles in a complete graph K_n, where n is odd.

Theorem 4.4.3 In a complete graph K_n, where $n \geq 3$ is odd, there are $\frac{n-1}{2}$ edge-disjoint Hamilton cycles.

Proof. Note that a complete graph has $\frac{n(n-1)}{2}$ edges and a hamilton cycle in K_n contains only n edges. Therefore, the maximum number of edge-disjoint hamilton cycles is $\frac{n-1}{2}$.

Now, assume that $n \geq 3$ and is odd. Construct a subgraph G of K_n as explained below:

The vertex v_1 is placed at the centre of a circle and the remaining $n-1$ vertices are placed on the circle, at equal distances along the circle such that the angle made at the centre by two points is $\frac{360}{n-1}$ degrees. The vertices with odd suffixes are placed along the upper half of the circle and the vertices with even suffixes are placed along the lower half circle. Then, draw edges v_iv_{i+1}, where $1 \leq i \leq n$, with the meaning that $v_{n+1} = v_1$, are drawn as shown in Figure 4.7.

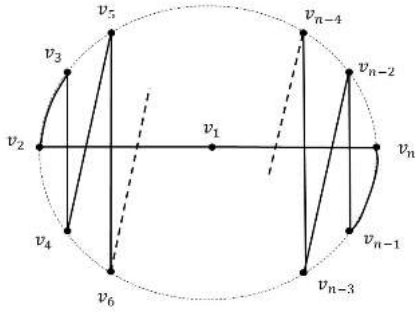


Figure 4.6: A Hamilton cycle G of K_n.

Clearly, the reduced graph G_1 is a cycle covering all vertices of K_n. That is If we rotate the vertices along the curve for $\frac{360}{n-1}$ degrees, we get another Hamilton subgraph G_2 of K_n, which has no common edges with G_1. (see figure 4.7).

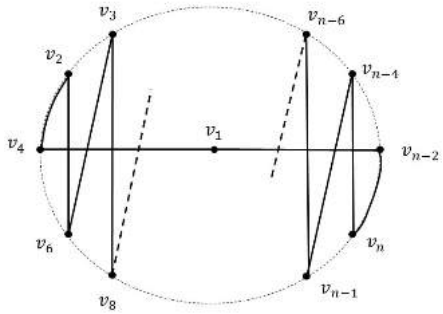


Figure 4.7: Another Hamilton cycle G_2 of K_n.

In a similar way, rotate the polygonal pattern clockwise by $\frac{360}{(n-1)}$ degrees. After $(n-1)$-th rotation, all vertices will be exactly as in Figure 4.6. Therefore, $n-1$ rotations are valid. But, it can be noted that the cycle G_i obtained after the i-th rotation and the cycle $G_{\frac{n-1}{2}+i}$ are isomorphic graphs, because all vertices in the upper half cycle in G_i will be in the lower half cycle in $G_{\frac{n-1}{2}+i}$ and vice versa, in the same order(see Figure 4.8).

That is, we have now that there are $\frac{n-1}{2}$ distinct such non-isomorphic edge-disjoint cycles

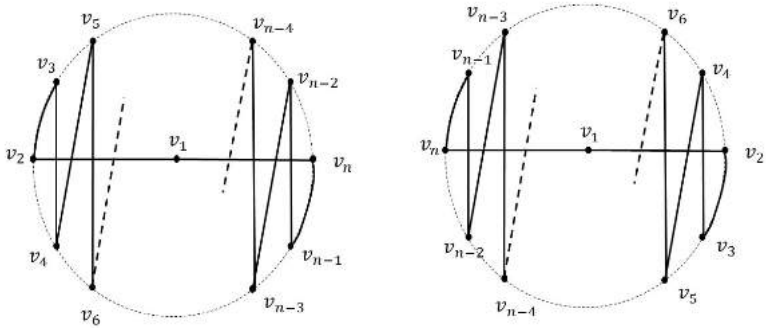


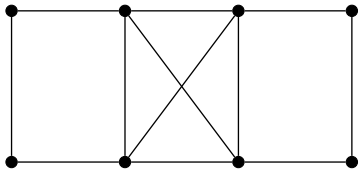
Figure 4.8: Isomorphic Hamilton cycles G_1 and $G_{\frac{n-1}{2}+1}$ of K_n.

in K_n. Hence, the number of edge-disjoint Hamilton cycles is $\frac{n-1}{2}$. ■

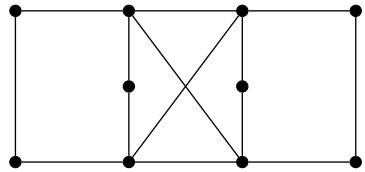
4.5 Some Illustrations

We can find out graphs, which are either Eulerian or Hamiltonian or simultaneously both, whereas some graph are neither Eulerian nor Hamiltonian. We note that the dodecahedron is an example for a Hamiltonian graph which is not Eulerian (see Figure 4.5a).

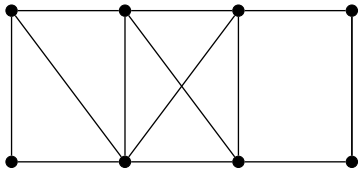
Let us now examine some examples all possible types of graphs in this category.



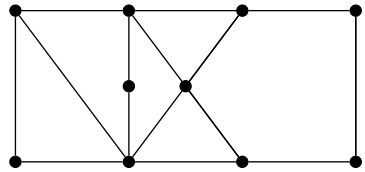
(a) A graph which is both Eulerian and Hamiltonian



(b) A graph which is Eulerian, but not Hamiltonian.



(c) A graph which is Hamiltonian, but not Eulerian.



(d) A graph which is neither Eulerian nor Hamiltonian.

Figure 4.9: Traversability in graphs

4.6 Weighted Graphs

Definition 4.6.1 — Weighted Graphs. A **weighted graph** is a graph G in which each edge e has been assigned a real number $w(e)$, called the *weight* (or *length*) of the edge e.

Figure 4.10 illustrates a weighted graph:

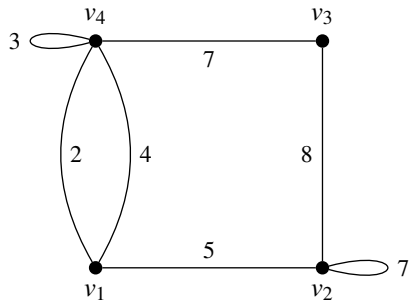


Figure 4.10: An example of a weighted graph

If H is a subgraph of a weighted graph, the weight $w(H)$ of H is the sum of the weights $w(e_1)+w(e_2)+\ldots+w(e_k)$ where $\{e_1,e_2,\ldots,e_k\}$ is the set of edges of H.

Many optimisation problems amount to finding, in a suitable weighted graph, a certain type of subgraph with minimum (or maximum) weight.

4.7 Travelling Salesman's Problem

Suppose a travelling salesman's territory includes several towns with roads connecting certain pairs of these towns. As a part of his work, he has to visit each town. For this, he needs to plan a round trip in such a way that he can visit each of the towns exactly once.

We represent the salesman's territory by a weighted graph G where the vertices correspond to the towns and two vertices are joined by a weighted edge if and only if there is a road connecting the corresponding towns which does not pass through any of the other towns, the edge's weight representing the length of the road between the towns.

Then, the problem reduces to check whether the graph G is a Hamiltonian graph and to construct a Hamiltonian cycle of minimum weight (or length) if G is Hamiltonian. This problem is known as the *Travelling Salesman Problem*.

It is sometimes difficult to determine if a graph is Hamiltonian as there is no easy characterisation of Hamiltonian graphs. Moreover, for a given a weighted graph G which is Hamiltonian there is no easy or efficient algorithm for finding an optimal circuit in G, in general. These facts make our problem difficult.

To find out an optimal Hamilton cycle, we use the following algorithm with an assumption that the given graph G is a weighted complete graph.

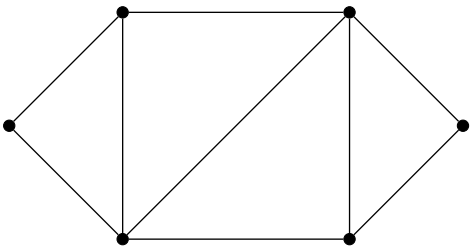
Two Optimal Algorithm

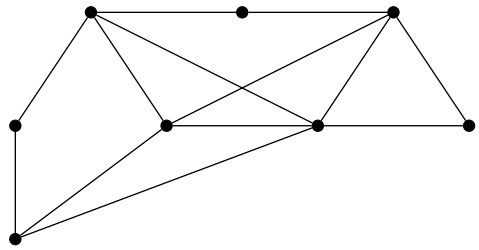
1. Let $C = v_1v_2, v_3\ldots, v_nv_1$ be any Hamiltonian cycle of the weighted graph G and let w be the weight of C. That is, $w = w(v_1v_2)+w(v_2,v_3)+\ldots+w(v_nv_1)$.

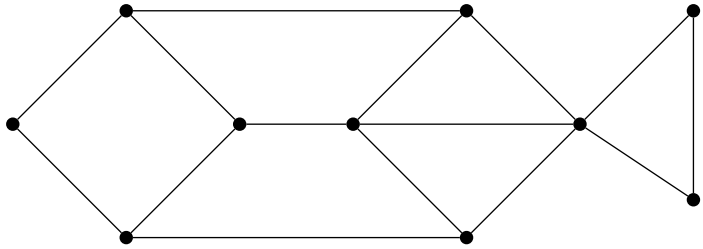
2. Set $i = 1$.
3. Set $j = i+2$.
4. Let $C_{ij} = v_1v_2v_2v_3 \dots v_iv_jv_{j-1} \dots v_{i+1}v_{j+1}v_{j+2} \dots v_nv_1$ be the Hamiltonian cycle and let w_{ij} denote the weight of C_{ij}, so that $w_{ij} = w - w(v_iv_{i+1}) - w(v_jv_{j+1}) + w(v_iv_j) + w(v_{i+1}v_{j+1})$.
   If $w_{ij} < w$, (that is, if $w(v_iv_j) + w(v_{i+1}v_{j+1}) < w(v_iv_{i+1}) + w(v_jv_{j+1})$), then replace C by C_{ij} and w by w_{ij} and return to Step 1, taking the sequence of vertices $v_1v_2v_3 \dots v_nv_1$ as given by our new C.
5. Set $j = j+1$. If $j < n$, do Step 4. Otherwise, set $i = i+1$. If $i < n-2$, do Step 3. Otherwise, stop.

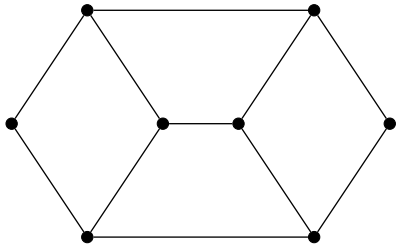
4.8 Exercises

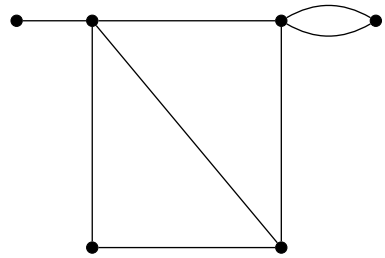
1. Draw a graph that has a Hamilton path, but not a Hamilton Cycle.

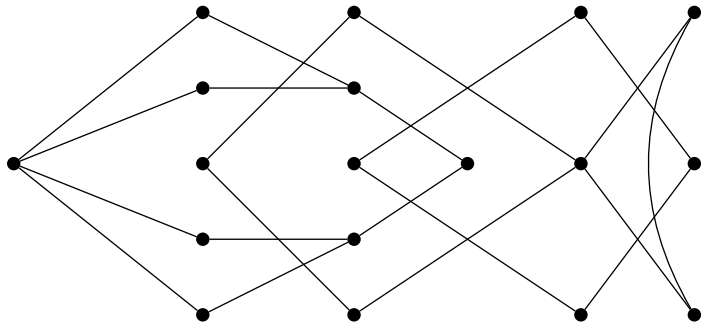






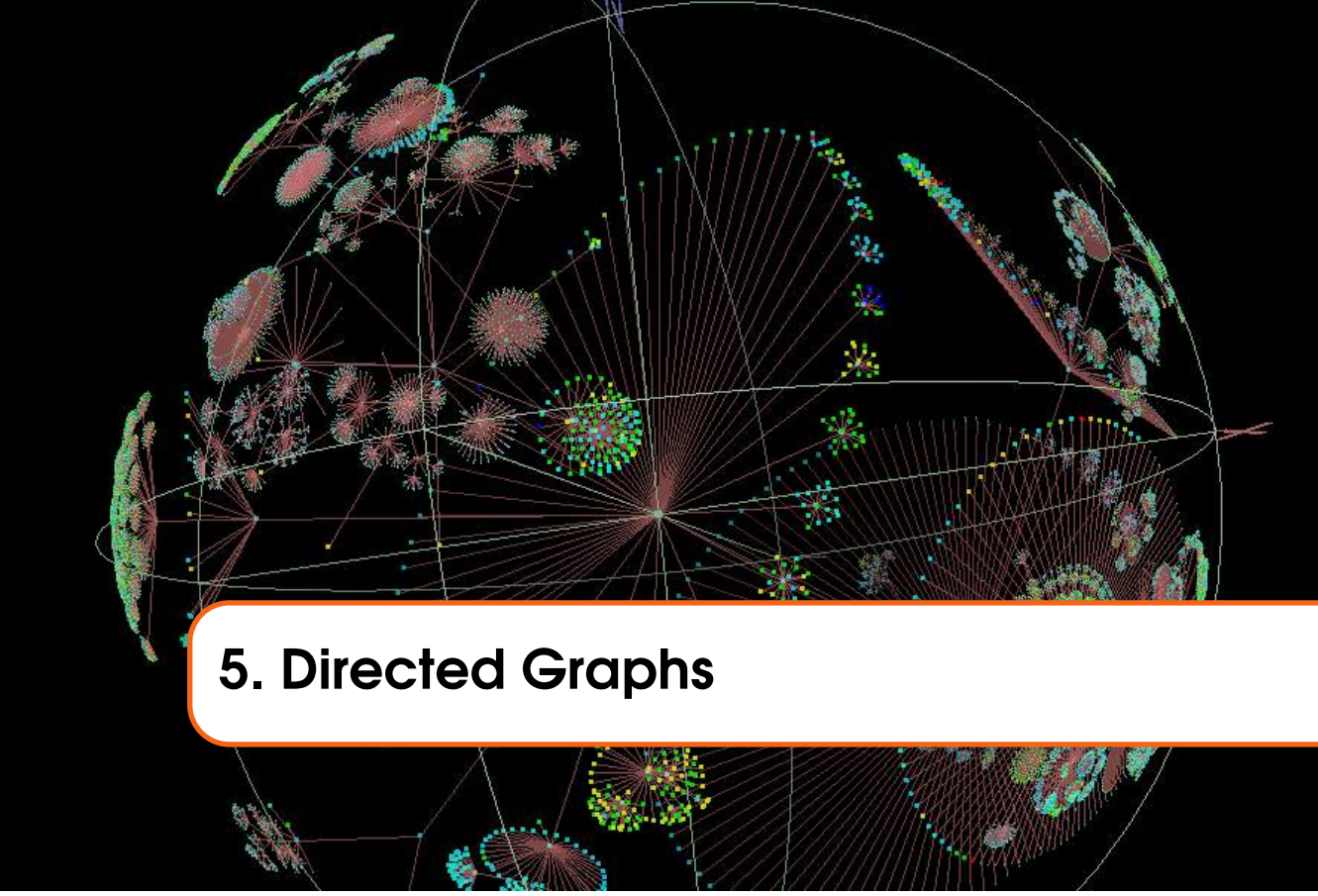






2. Show that if G is Eulerian,then every block (see Chapter 7 for the notion of blocks) of G is Eulerian.
3. Show that a Hamilton path of a graph G, if exists, is the longest path in G.
4. Show that if G is a self-complementary graph, then G has a Hamilton path.
5. Show that every complete graph $K_n; n \geq 3$ is Hamiltonian.
6. Verify whether Petersen graph is Eulerian. Justify your answer.
7. Verify whether Petersen graph is Hamiltonian. Justify your answer.
8. Verify whether the following graphs are Eulerian and Hamiltonian. Justify your answer.
9. Show that every even graph (a graph without odd degree vertices) can be decomposed into cycles.
10. Show that if either
    (a) G is not 2-connected, or
    (b) G is bipartite with bipartition (X,Y) where $|X| \leq |Y|$.

    Then G is non-Hamiltonian.
11. Characterise all simple Euler graphs having an Euler tour which is also a Hamiltonian cycle.
12. Let G be a Hamiltonian graph. Show that G does not have a cut vertex.
13. There are n guests at a dinner party, where $n \geq 4$. Any two of these guests know, between them, all the other $n-2$. Prove that the guests can be seated round a circular table so that each one is sitting between two people they know.
14. Let G be a simple k-regular graph, with $2k-1$ vertices. Prove that G is Hamiltonian.



5. Directed Graphs

5.1 Directed Graphs

Definition 5.1.1 — Directed Graphs. A *directed graph* or *digraph* G consists of a set V of vertices and a set E of edges such that $e \in E$ is associated with an ordered pair of vertices. In other words, if each edge of the graph G has a direction, then the graph is called a directed graph.

The directed edges of a directed graph are also called *arcs*. The initial vertex of an arc a is called the *tail* of a and the terminal vertex v is called the *head* of the arc a. An arc $e = (u, v)$ in a digraph D is a *loop* if $u = v$. Two arcs e, f are *parallel edges* if they have the same tails and the same heads. If D has no loops or parallel edges, then we say that D is *simple*.

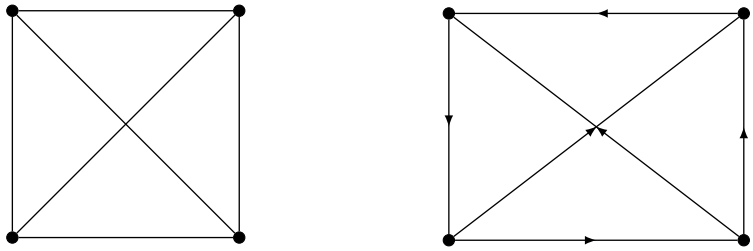
Definition 5.1.2 — Degrees in Digraphs. The *indegree* of vertex v in a directed graph D is the number of edges which are coming into the vertex v (that is, the number of incoming edges) and is denoted by $d^-(v)$. The *out-degree* vertex v in a directed graph D is the number of edges which are going out from the vertex v (that is, the number of outgoing edges) and is denoted by $d^+(v)$.

Definition 5.1.3 — Orientation of Graphs. If we assign directions to the edges of a given graph, then the new directed graph D is called an *orientation* of G.

Definition 5.1.4 — Underlying Graphs of Directed Graphs. If remove the directions of the edges of a directed graph D, then the reduced graph G is called the *underlying graph* of D.

Note that the orientation of a graph G is not unique. Every edge of G can take any one of the two possible directions. Therefore, a graph $G = (V, E)$ can have at most $2^{|E|}$ different orientations. But, a directed graph can have a unique underlying graph.

Figure 5.1b illustrates an undirected graph G and an orientation D of G.



(a) A graph G (b) Orientation of G

Figure 5.1: An undirected graph and one of its orientations.

Definition 5.1.5 — Sources and Sinks. A vertex with zero in-degree is called a *source* and a vertex with zero out-degree is called a *sink*.

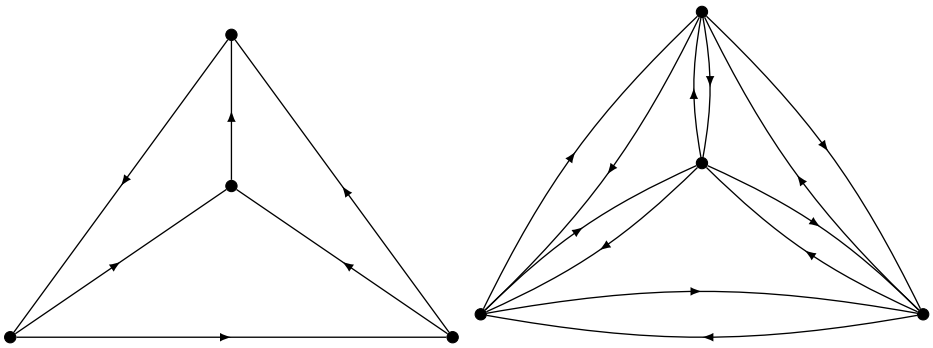
Theorem 5.1.1 In a directed graph D, the sum of the in-degrees and the sum of out-degrees of the vertices is equal to twice the number of edges. That is, $\sum_{v \in V(D)} d^+(v) = \sum_{v \in V(D)} d^-(v) = \varepsilon$.

Proof. Let $S^+ = \sum_{v \in V(D)} d^+(v)$ and $S^- = \sum_{v \in V(D)} d^-(v)$. Notice that every arc (edge) of D contributes exactly 1 to S^+ and 1 to S^-. That is, we count each edge exactly once in S^+ and once in S^-. Hence, $S^+ = S^- = |E| = \varepsilon$. ■

Definition 5.1.6 — Tournaments. If the edges of a complete graph are each given an orientation, the resulting directed graph is called a *tournament*.

Definition 5.1.7 — Complete Digraph. A *complete digraph* is a directed graph in which every pair of distinct vertices is connected by a pair of unique edges (one in each direction).

Definition 5.1.8 — Paths and Cycles in Directed Graphs. A *directed walk* in a digraph D is a sequence $v_0, e_1, v_2, \ldots, e_n, v_n$ so that $v_i \in V(D)$ for every $0 \leq i \leq n$, and e_i is an edge from v_{i-1} to v_i for every $1 \leq i \leq n$. We say that this is a walk from v_0 to v_n. If $v_0 = v_n$,



(a) A tournament on 4 vertices (b) A complete digraph on 4 vertices

Figure 5.2: A tournament and a complete digraph

then the walk is called a *closed directed walk* and if $v_0, v_1, \ldots, v_n$ are distinct we call it a *directed path*. In a directed path $v_0, e_1, v_2, \ldots, e_n, v_n$, if $v_0 = v_n$, then the directed path is called a *directed cycle*.

Definition 5.1.9 — Weakly and Strongly Connected Digraphs. A digraph D is said to be *weakly connected* if its underlying graph G is connected. A digraph D is said to be *strongly connected* if there is a directed path between any two vertices $u, v \in V(D)$.

Some observations on directed graphs are given below.

1. Let D be a digraph in which every vertex has outdegree at least 1. Then D contains a directed cycle.
2. A digraph D is *acyclic* if it has no directed cycles. The digraph D is acyclic if and only if there is an ordering $v_1, v_2, \ldots, v_n$ of $V(D)$ so that every edge (v_i, v_j) satisfies $i < j$.

Definition 5.1.10 — Eulerian Digraphs. A closed directed walk in a digraph D is called *Eulerian* if it contains every edge exactly once. A digraph D is said to be an *Eulerian digraph* if it contains an Eulerian closed directed walk.

The following theorem describes a necessary and sufficient condition for a digraph to be Eulerian.

Theorem 5.1.2 A digraph D, whose underlying graph is connected, is Eulerian if and only if $d^+(v) = d^-(v)$ for every $v \in V(D)$.

Proof. (*Necessary Part:*) First assume that D is Eulerian. Then, D consists of an Eulerian closed directed walk. That is, whenever there is an incoming edge to v, then there is an outgoing edge from v. Therefore, $d^+(v) = d^-(v)$ for every $v \in V(D)$.
(*Converse Part:*) Choose a closed walk $v_0, e_1, v_1, e_2, v_2 \ldots, e_n, v_n$ which uses each edge at most once and is maximum in length, subject to the given constraint. Suppose that this walk is not Eulerian. Then, as in the undirected case, it follows from the fact that the underlying

graph is connected that there exists an edge $e \in E(D)$ which does not appear in the walk so that e is incident with some vertex in the walk, say v_i. Let $H = D - \{e_1, e_2, \ldots, e_n\}$. Then, every vertex of H has indegree equal to its outdegree, so by the previous proposition, there is a list of directed cycles in H containing every edge exactly once. In particular, there is a directed cycle $C \subseteq H$ with $e \in C$. But then, the walk obtained by following $v_0, e_1, \ldots, v_i$, then following the directed cycle C from v_i back to itself, and then following $e_{i+1}, v_i, \ldots, v_n$ is a longer closed walk which contradicts our choice. This completes the proof. ■

5.2 Types of Directed graphs

Definition 5.2.1 — Symmetric Digraphs. *Symmetric directed graphs* are the directed graphs where all edges are bidirected (that is, for every arrow that belongs to the digraph, the corresponding inversed arrow also belongs to it). Note that, symmetric digraphs are the graphical representations of symmetric relations.

Definition 5.2.2 — Transitive Digraphs. A digraph D is said to be a *transitive digraph*, if any three vertices (x, y, z) such that edges (x, y) and (y, z) in G imply (x, z) in D. Unlabeled transitive digraphs are called *digraph topologies*.

Definition 5.2.3 — Directed Acyclic Graphs. *Directed acyclic graphs* (DAGs) are directed graphs with no directed cycles.

5.3 Networks

Definition 5.3.1 — Networks. A *network* is a directed graph $D + (V, A)$ in which a unique non-negative real number is assigned to every arc of D.

Definition 5.3.2 — Capacity Function. This assignment $c : A \to \mathbb{R}$ is called the *capacity function* of the network D. The non-negative real number assigned an arc a of D is called its *capacity* and is denoted by $c(a)$.

Definition 5.3.3 — Flow Function. To a network, we may assign another function $f : E \to \mathbb{R}$, called the *flow function*, that assigns a non-negative real number such that $0 \leq f(a) \leq c(a)$, for any arc $a \in A(D)$.

Definition 5.3.4 — In-flow and Out-flow in Networks. The *inflow* to a vertex v, denoted by $f_{in}(v)$, is defined by $f_{in}(v) = \sum_{uv \in A} f(uv)$ and the *outflow* from v, denoted by $f_{out}(v)$, is defined by $f_{out}(v) = \sum_{vw \in A} f(vw)$. Note that $f_{in}(v) = f_{out}(v)$ for $v \in V(D)$.

The vertex set of a network can be partitioned into three types:

(i) *Source:* A vertex with indegree 0 is called a *source vertex* and is denoted by s and the set of all source vertices is called the *source*, and is denoted by S.

(ii) *Sink:* A vertex with outdegree 0 is called a *sink vertex* and is denoted by t and the set of all sink vertices is called the *sink*, and is denoted by $\bar{S}$.

(iii) A vertex that is neither a source nor a sink is called an *intermediate vertex* and is denoted by s and the set of all source vertices is called the *intermediate*, and is denoted by I.

Note the following points:

(i) The flow over any arc can be no more than the capacity of that arc.

(ii) The inflow on any intermediate vertex is equal to the outflow of that vertex. That is, the flow is not obstructed or reduced at any intermediate vertices.

The value of a flow f, denoted by $val\ f$ is defined as

$$\begin{aligned} val\ f &= \sum_{v\in X} f_{out}(v) - f_{in}(v) \\ &= f_{out}(X) - f_{in}(X). \end{aligned}$$

Definition 5.3.5 — Cuts in Networks. A *cut* of a network D, denoted by $(S,\bar{S})$ or K, is the set of arcs $\{s\bar{s} : s \in S, \bar{s} \in \bar{S}\}$, whose removal disconnects the network into two components.

The *capacity* of a cut K, denoted by $capK$ and is defined as

$$cap\ K = \sum_{a\in K} c(a).$$

Consider a $u-v$ path P and flow f on a network D. For an arc $a \in P$, define the function $\ell_f(a)$ by

$$\ell_f(a) = \begin{cases} c(a) - f(a), & \text{if } a \text{ directs towards } v; \\ f(a), & \text{if } a \text{ points towards } u. \end{cases}$$

Then, the f-*augment* of P, denoted by $\ell_f(P)$ and is defined as $ell_f(P) = \min_{a\in P} \ell_f(a)$. A $u-v$ path P is said to be f-*augmenting* if and only if u is a source, v is a sink and the f-augment of P is positive. Given an f-augmenting path P, we can construct a new flow f' whose value is

$$val\ f' = val\ f + \ell_f(P).$$

That is, for every arc on the path P,

(i) increase the flow by a quantity $\ell_f(P)$, if it is a forward arc;

(ii) decrease the flow by a quantity $\ell_f(P)$, if it is a backward arc.

Our problem in this context is the natural optimisation problem: *What is the maximum value attained by any flow?*

Theorem 5.3.1 — Min-cut Max-Flow Theorem. For any network D, the value of the maximum flow is equal to the capacity of the minimum cut.

Proof. We need to show that maximum of all flows (the value of maximum flow) is equal to the minimum capacity of all cut capacities (the capacity of the minimum cut). To prove this

we the following steps.

Claim-1: For any flow f and any cut k in any given network D, $val\ f \leq cap\ K$.

Proof of Claim-1: Let $K = (S, \bar{S})$. If X comprised of all sources and intermediates, we have

$$val\ f = f_{out}(S) - f_{in}(S) = f_{out}(X) - f_{in}(X)$$

(as intermediate contributes nothing to the flow).

Now, let a be an arc with both its end vertices ae in S. Then, its inflow and outflow are both counted in S and hence make no impact on the net value of flow. Therefore, the only flows which have a positive impact on $val\ f$ are those which originate in S and terminate in $\bar{S}$, which must precisely pass through the cut $K = (S, \bar{S})$. Therefore, $val\ f \leq \sum_{a \in K} f(a) \leq \sum_{a \in K} c(a) = cap\ K$. Hence, our claim is valid.

Corollary 5.3.2 If f^* is the maximum flow and K^* is the minimum cut on the network, then $val\ f^* \leq cap\ K^*$.

Claim 2: Given a network, there exists a flow f and cut K on the network such that $val\ f = cap\ K$.

Proof of Claim-2: Let f be a flow in the network D such that there are no f-augmenting paths in D. (We can construct such a flow by starting with the zero flow, and adjust the flow as described in the augmenting procedure until no more augmenting is possible.)

Let S be the set consisting of al source vertices and all vertices v such that there exists an $x - v$ path from some source vertices x to v with positive f-augment. Then, note that $\bar{S}$ contains the network sinks. Now, Let $K = (S, \bar{S})$.

If possible, let $f(a) < c(a)$ for some arcs $a = s\bar{s} \in K$. Note that $x - s$ path with positive augment from source x can be extended to an $x - \bar{s}$ path with positive f-augment, which implies $\bar{s} \in S$, a contradiction.

We noted in the first conclusion that only the flows over the cut K positively impact $val\ f$. Then, we have $val\ f = cap\ K$. Hence our claim is true. Now, we proceed to prove the result as follows. Let f^* be the maximum flow and * be the minimum cut on a given network. Then, we have

(i) $val\ f \leq val\ f^*$; and

(ii) $cap\ K^* \leq cap\ K$.

Then, by Claim -1 and Claim-2, we note that there exists some flow f and cut K such that $cap\ K = val\ f \leq cap\ K^*$. This is possible only when $val\ f = cap\ K^*$, as K^* is the minimum cut.

By the corollary of Claim-1, we note that $val\ f^* \leq cap\ K^* = val\ f$, which is possible only when $val\ f^* = cap\ K^*$. That is, the value of the maximum flow is equal to the capacity of the minimum cut, completing the proof. ■

III TREES

6. Trees

Definition 6.0.1 — Tree. A graph G is called a *tree* if it is connected and has no cycles. That is, a tree is a connected acyclic (circuitless) graph.

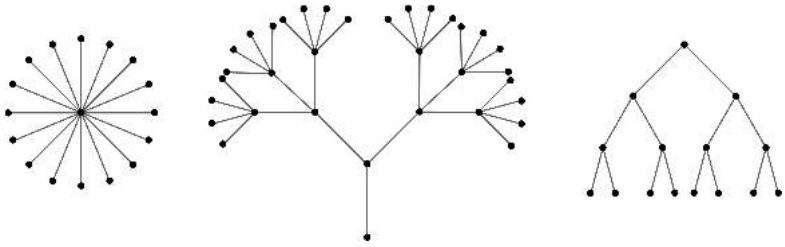


Figure 6.1: Examples of trees

Definition 6.0.2 — Tree. An acyclic graph may possibly be a disconnected graph whose components are trees. Such graphs are called **forests**.

6.1 Properties of Trees

Theorem 6.1.1 A graph is a tree if and only if there is exactly one path between every pair of its vertices.

Proof. Let G be a graph and let there be exactly one path between every pair of vertices in G. So G is connected. If G contains a cycle, say between vertices u and v, then there are

two distinct paths between u and v, which is a contradiction to the hypothesis. Hence, G is connected and is without cycles, therefore it is a tree.

Conversely, let G be a tree. Since G is connected, there is at least one path between every pair of vertices in G. Let there be two distinct paths, say P and P' between two vertices u and v of G. Then, the union of $P \cup P'$ contains a cycle which contradicts the fact that G is a tree. Hence, there is exactly one path between every pair of vertices of a tree. ■

Then, by Definition 3.1.11, we have the following result:

Theorem 6.1.2 All trees are geodetic graphs.

Theorem 6.1.3 A tree with n vertices has $n-1$ edges.

Proof. We prove the result by using mathematical induction on n, the number of vertices. The result is obviously true for $n = 1,2,3$. See illustrations in Figure 6.2.

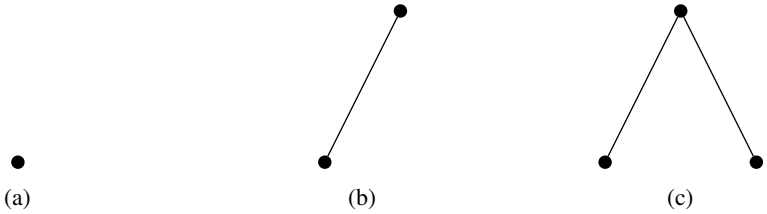


Figure 6.2: Trees with $n = 1,2,3$.

Let the result be true for all trees with fewer than n vertices. Let T be a tree with n vertices and let e be an edge with end vertices u and v. So, the only path between u and v is e. Therefore, deletion of e from T disconnects T.

Now, $T-e$ consists of exactly two components T_1 and T_2 say, and as there were no cycles to begin with, each component is a tree. Let n_1 and n_2 be the number of vertices in T_1 and T_2 respectively. Then, note that $n_1+n_2 = n$. Also, $n_1 < n$ and $n_2 < n$. Thus, by induction hypothesis, the number of edges in T_1 and T_2 are respectively n_1-1 and n_2-1. Hence, the number of edges in T is $n_1-1+n_2-1+1 = n_1+n_2-1 = n-1$. ■

Theorem 6.1.4 Any connected graph with n vertices and $n-1$ edges is a tree.

Proof. Let G be a connected graph with n vertices and $n-1$ edges. We show that G contains no cycles. Assume to the contrary that G contains cycles. Remove an edge from a cycle so that the resulting graph is again connected. Continue this process of removing one edge from one cycle at a time till the resulting graph H is a tree. As H has n vertices, so the number of edges in H is $n-1$. Now, the number of edges in G is greater than the number of edges in H. That is, $n-1 > n-1$, which is not possible. Hence, G has no cycles and therefore is a tree. ■

Theorem 6.1.5 Every edge of a tree is a cut-edge of G.

Proof. Since a tree T is an acyclic graph, no edge of T is contained in a cycle. Therefore, by Theorem 3.3.1, every edge of T is a cut-edge. ■

A graph is said to be *minimally connected* if removal of any one edge from it disconnects the graph. Clearly, a minimally connected graph has no cycles.

The following theorem is another characterization of trees.

Theorem 6.1.6 A graph is a tree if and only if it is minimally connected.

Proof. Let the graph G be minimally connected. Then, G has no cycles and therefore is a tree. Conversely, let G be a tree. Then, G contains no cycles and deletion of any edge from G disconnects the graph. Hence, G is minimally connected. ■

Theorem 6.1.7 A graph G with n vertices, $n-1$ edges and no cycles is connected.

Proof. Let G be a graph without cycles with n vertices and $n-1$ edges. We have to prove that G is connected. Assume that G is disconnected. So G consists of two or more components and each component is also without cycles. We assume without loss of generality that G has two components, say G_1 and G_2. Add an edge e between a vertex u in G_1 and a vertex v in G_2. Since there is no path between u and v in G, adding e did not create a cycle. Thus $G \cup \{e\}$ is a connected graph (tree) of n vertices, having n edges and no cycles. This contradicts the fact that a tree with n vertices has $n-1$ edges. Hence, G is connected. ■

Theorem 6.1.8 Any tree with at least two vertices has at least two pendant vertices.

Proof. Let the number of vertices in a given tree T be n, where $(n > 1)$. So the number of edges in T is $n-1$. Therefore, the degree sum of the tree is $2(n-1)$ (by the first theorem of graph theory). This degree sum is to be divided among the n vertices. Since a tree is connected it cannot have a vertex of zero degree. Each vertex contributes at least 1 to the above sum. Thus, there must be at least two vertices of degree exactly 1. That is, every tree must have at least two pendant vertices. ■

Theorem 6.1.9 Let G be a graph on n vertices. Then, the following statements are equivalent:

(i) G is a tree.
(ii) G is connected and has $n-1$ edges.
(iii) G is acyclic (circuitless) and has $n-1$ edges.
(iv) There exists exactly one path between every pair of vertices in G.
(v) G is a minimally connected graph.

Proof. The equivalence of these conditions can be established using the results $(i) \implies (ii), (ii) \implies (iii), (iii) \implies (iv), (iv) \implies (v)$ and $(v) \implies (i)$.

Part-$(i) \implies (ii)$: This part states that if G is a tree on n vertices, then G is connected and has $n-1$ edges. Since G is a tree, clearly, by definition of a tree it is connected. The remaining part follows from the result that every tree on n vertices has $n-1$ vertices.

Part-$(ii) \implies (ii)$: This part states that if G is connected and has $n-1$ edges, then G is acyclic and has $n-1$ edges. Clearly, This result follows from the result that a connected graph on n vertices and $n-1$ edges is acyclic.

Part-$(iii) \implies (iv)$: This part states that if G is an acyclic graph on n vertices and has $n-1$ edges, then there exists exactly one path between every pair of vertices in G. By a previous theorem, we have an acyclic graph G on n vertices and $n-1$ edges is connected. Therefore, G is a tree. Hence, by our first theorem, there exists exactly one path between every pair of vertices in G.

Part-$(iv) \implies (v)$: This part states that if there exists exactly one path between every pair of vertices in G, then G is minimally connected. Assume that every pair of vertices in G is connected by a unique path.

Let u and v be any two vertices in G and P be the unique (u,v)-path in G. Let e be any edge in this path P. If we remove the edge from P, then there will be no (u,v)-path in $G-e$. That is, $G-e$ is disconnected. Therefore, G is minimally connected.

Part-$(v) \implies (i)$: This part states that if G is minimally connected, then G is a tree. Clearly, G is connected as it is minimally connected. Since G is minimally connected, removal of any edge makes G disconnected. That is, every edge of G is a cut edge of G. That is, no edge of G is contained in a cycle in G. Therefore, G is acyclic and hence is a tree. ■

Theorem 6.1.10 A vertex v in a tree is a cut-vertex of T if and only if $d(v) \geq 2$.

Proof. Let v be a cut-vertex of a tree T. Since, no pendant vertex of a graph can be its cut-vertex, clearly we have $d(v) \geq 2$.

Let v be a vertex of a tree T such that $d(v) \geq 2$. Then v is called an *internal vertex* (or *intermediate vertex*) of T. Since $d(v) \geq 2$, there are two at least two neighbours for v in T. Let u and w be two neighbours of v. Then, $u-v-w$ is a $(u-w)$-path in G. By Theorem-1, we have the path $u-v-w$ is the unique $(u-w)$-path in G. Therefore, $T-v$ is disconnected and u and w are in different components of T. Therefore, v is a cut-vertex of T. This completes the proof. ■

6.2 Distances in Trees

Definition 6.2.1 — Metric. A *metric* on a set A is a function $d : A \times A \to [0,\infty)$, where $[0,\infty)$ is the set of non-negative real numbers and for all $x,y,z \in A$, the following conditions are satisfied:

1. $d(x,y) \geq 0$ (non-negativity or separation axiom);
2. $d(x,y) = 0 \Leftrightarrow x = y$ (identity of indiscernibles);

3. $d(x,y) = d(y,x)$ (symmetry);
4. $d(x,z) \leq d(x,y) + d(y,z)$ (sub-additivity or triangle inequality).

Conditions 1 and 2, are together called a *positive-definite function*.

A metric is sometimes called the *distance function*.

In view of the definition of a metric, we have

Theorem 6.2.1 The distance between vertices of a connected graph is a metric.

Definition 6.2.2 — Center of a graph. A vertex in a graph G with minimum eccentricity is called the *center* of G.

Theorem 6.2.2 Every tree has either one or two centers.

Proof. The maximum distance, $\max d(v,v_i)$ from a given vertex v to any other vertex occurs only when v_i is a pendant vertex. With this observation, let T be a tree having more than two vertices. Tree T has two or more pendant vertices.

Deleting all the pendant vertices from T, the resulting graph T' is again a tree. The removal of all pendant vertices from T uniformly reduces the eccentricities of the remaining vertices (vertices in T') by one. Therefore, the centers of T are also the centers of T'. From T', we remove all pendant vertices and get another tree T''. Continuing this process, we either get a vertex, which is a center of T, or an edge whose end vertices are the two centers of T. ■

6.3 Degree Sequences in Trees

Theorem 6.3.1 The sequence $\langle d_i \rangle; 1 \leq i \leq n$ of positive integers is a degree sequence of a tree if and only if (i) $d_i \geq 1$ for all $i, 1 \leq i \leq n$ and (ii) $\sum d_i = 2n-2$.

Proof. Since a tree has no isolated vertex, we have $d_i \geq 1$ for all i. Also, $\sum d_i = 2|E| = 2(n-1)$, as a tree with n vertices has $n-1$ edges.

We use induction on n to prove the converse part. For $n = 2$, the sequence is $\{1,1\}$ and is obviously the degree sequence of K_2 . Suppose the claim is true for all positive sequences of length less than n. Let $\langle d_i \rangle$ be the non-decreasing positive sequence of n terms, satisfying conditions (i) and (ii). Then $d_1 = 1$ and $d_n > 1$.

Now, consider the sequence $D' = \{d_2, d_3, \ldots, d_{n-1}, d_n - 1\}$, which is a sequence of length $n-1$. Obviously in D', we have $d_i \geq 1$ and $\sum d_i = d_2 + d_3 + \ldots + d_{n-1} + d_n - 1 = d_1 + d_2 + d_3 + \ldots + d_{n-1} + d_n - 1 - 1 = 2n - 2 - 2 = 2(n-1) - 2$ (because $d_1 = 1$). So D' satisfies conditions (i) and (ii) and by induction hypothesis, there is a tree T_0 realising D'. In T_0, add a new vertex and join it to the vertex having degree $d_n - 1$ to get a tree T. Therefore, the degree sequence of T is $\{d_1, d_2, \ldots, dn\}$. This completes the proof. ■

Theorem 6.3.2 Let T be a tree with k edges. If G is a graph whose minimum degree satisfies $\delta(G) \geq k$, then G contains T as a subgraph. In other words, G contains every tree of order at most $\delta(G)+1$ as a subgraph.

Proof. We use induction on k. If $k = 0$, then $T = K_1$ and it is clear that K_1 is a subgraph of any graph. Further, if $k = 1$, then $T = K_2$, and K_2 is a subgraph of any graph whose minimum degree is one.

Assume that the result is true for all trees with $k-1$ edges ($k \geq 2$) and consider a tree T with exactly k edges. We know that T contains at least two pendant vertices. Let v be one of them and let w be the vertex that is adjacent to v.

Consider the graph $T - v$. Since $T - v$ has $k-1$ edges, the induction hypothesis applies, so $T - v$ is a subgraph of G. We can think of $T - v$ as actually sitting inside G (meaning w is a vertex of G, too).

Since G contains at least $k+1$ vertices, and $T - v$ contains k vertices, there exist vertices of G that are not a part of the subgraph $T - v$. Further, since the degree of w in G is at least k, there must be a vertex u not in $T - v$ that is adjacent to w. The subgraph $T - v$ together with u forms the tree T as a subgraph of G. ■

6.4 On Counting Trees

A *labelled graph* is a graph, each of whose vertices (or edges) is assigned a unique name ($v_1, v_2, v_3, \ldots$ or $A, B, C, \ldots$) or labels ($1, 2, 3, \ldots$).

The distinct vertex labelled trees on 4 vertices are given in Figure 6.3.

The distinct unlabelled trees on 4 vertices are given in Figure 6.4.

6.5 Spanning Trees

Definition 6.5.1 — Spanning Tree. A *spanning tree* of a connected graph G is a tree containing all the vertices of G. A *spanning tree* of a graph is a maximal tree subgraph of that graph. A spanning tree of a graph G is sometimes called the *skeleton* or the *scaffold graph*.

Theorem 6.5.1 Every connected graph G has a spanning tree.

Proof. Let G be a connected graph on n vertices. Pick an arbitrary edge of G and name it e_1. If e_1 belongs to a cycle of G, then delete it from G. (Else, leave it unchanged and pick it another one). Let $G_1 = G - e_1$. Now, choose an edge e_2 of G_1. If e_2 belongs to a cycle of G_1, then remove e_2 from G_1. Proceed this step until all cycles in G are removed iteratively. Since G is a finite graph the procedure terminates after a finite number of times. At this stage, we get a subgraph T of G, none of whose edges belong to cycles. Therefore, T is a connected acyclic subgraph of G on n vertices and hence is a spanning tree of G, completing the proof. ■

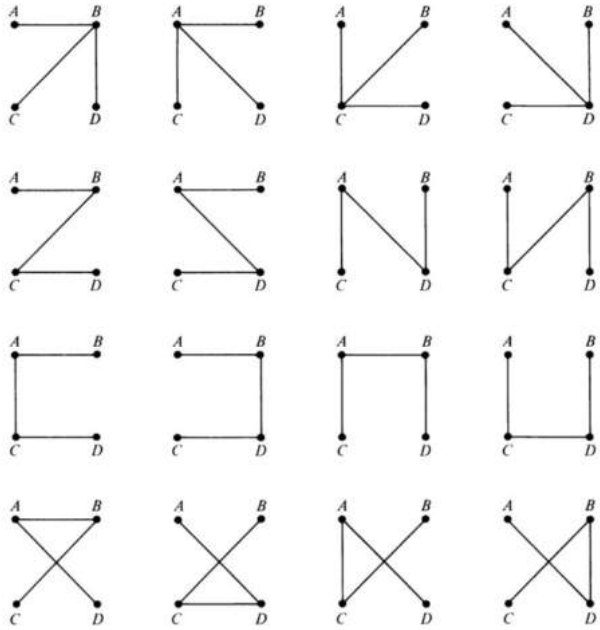


Figure 6.3: Distinct labelled trees on 4 vertices

Figure 6.4: Distinct unlabelled trees on 4 vertices

Definition 6.5.2 — Branches and Chords of Graphs. Let T be a spanning tree of a given graph G. Then, every edge of T is called a *branch* of T. An edge of G that is not in a spanning tree of G is called a *chord* (or a *tie* or a *link*).

Note that the branches and chords are defined in terms of a given spanning tree.

Theorem 6.5.2 Show that every graph with n vertices and ε edges has $n-1$ branches and $\varepsilon - n + 1$ chords.

Proof. Let G be a graph with n vertices and ε edges and let T be a spanning tree of G. Then, by Theorem 6.1.3, T has $n-1$ edges. Therefore, the number of branches is $n-1$. The number chords in G with respect to T is $|E(G)| - |E(T)| = \varepsilon - (n-1) = \varepsilon - n + 1$. ■

The set of all chords of a tree T is called a *chord set* or a *co-tree* or a *tie set* and is usually denoted by $\bar{T}$. Therefore, we have $T \cup \bar{T} = G$.

Definition 6.5.3 — Rank and Nullity of Graphs. The number of branches in a spanning tree T of a graph G is called its *rank*, denoted by $rank(G)$. The number of chords of a graph G is called its *nullity*, denoted by $nullity(G)$.

Therefore, we have

$$rank(G) + nullity(G) = |E(G)|, \text{ the size of } G.$$

Theorem 6.5.3 Let T and T' be two distinct spanning trees of a connected graph G and $e \in E(T) - E(T')$. Then, there exists an edge $e' \in E(T') - E(T)$ such that $T - e + e'$ is a spanning tree of G.

Proof. By Theorem 6.1.5, we know that every edge of a tree is a cut-edge. Let X_1 and X_2 be the two components of $T - e$. Since T' is a spanning tree and hence T' has an edge e' whose one end vertex in X_1 and the other in X_2. Therefore, the graph $T - e + e'$ is a graph on n vertices and $n-1$ edges and without cycles (since adding an edge between two vertices in two components of G will not create a cycle). Therefore, by Theorem 6.1.7, the graph $T - e + e'$ and hence is a tree. That is, $T - e + e'$ is a spanning tree of G. ■

Theorem 6.5.4 Let T and T' be two distinct spanning trees of a connected graph G and $e \in E(T) - E(T_2)$. Then, there exists an edge $e' \in E(T') - E(T)$ such that $T' + e - e'$ is a spanning tree of G.

Proof. Since T' is a spanning tree of G and e is not an edge of T', by Theorem 10.16b, $T' + e$ contains a unique cycle, say C which contains e'. Since T is a tree (hence acyclic), we can find an edge $e' \in E(C) - E(T)$. Therefore, removal of e' breaks the cycle C and hence $T' + e - e'$ is acyclic and has n vertices and $n-1$ edges. Thus, by Theorem 6.1.7, $T' + e - e'$ is connected and hence is a spanning tree of G. ■

Theorem 6.5.5 — Cayley's Theorem. For $n \geq 2$, the number of distinct spanning trees with n vertices is n^{n-2}.

Proof. Let $V = \{v_1, v_2, \ldots, v_n\}$ be the vertex set of K_n and let $N = \{1, 2, 3, \ldots, n\}$. Note that there exists a bijection f from V to N such that $f(v_i) = i$ and V can be replaced by N in this discussion. We note that n^{n-2} is the number of sequences of length $n-2$ that can be formed from N. Thus, in order to prove the theorem, we try to establish a one-to-one correspondence between the set of spanning trees of K_n and the set of such sequences length $n-2$ from N.

With each spanning tree T of K_n, we associate a unique sequence $(t_1, t_2, t_3, \ldots, t_{n-2})$ as follows. Regarding V as an ordered set, let s_1 be the first vertex of degree 1 in T; the vertex adjacent to s_1 is taken as t_1. Let s_2 the first vertex of degree 1 in $T - s_1$, and take the vertex adjacent to s_2 as t_2. This operation is repeated until t_{n-2} has been defined and a tree with just two vertices remains. Note that different spanning trees of K_n determine difference sequences.

Now, note that any vertex v of T occurs $d_T(v)-1$ times in the $(t_1,t_2,t_3,\ldots,t_{n-2})$. Thus, the vertices of degree one in T are precisely those that do not appear in this sequence. Therefore, to reconstruct T from the sequence $(t_1,t_2,\ldots,t_{n-2})$, we proceed as follows: Let s_1 be the first vertex of N not in $(t_1,t_2,\ldots,t_{n-2})$; join s_1 to t_1. Next, let s_2 be the first vertex of $N-\{s_1\}$ not in $(t_2,t_3\ldots,t_{n-2})$, and join s_2 to t_2. Proceed in this way until the $n-2$ edges $s_1t_1,s_2t_2,s_3t_3,\ldots,s_{n-}2t_{n-2}$ have been determined. T is now obtained by adding the edge joining the two remaining vertices of $N-\{s_1,s_2,\ldots,s_{n-2}\}$. We can verify that different sequences we get give rise to different spanning trees of Kn.

We have thus established the desired one-to-one correspondence between the distinct number of spanning trees of K_n and the distinct sequences from N of length $n-2$. Therefore, the number of distinct spanning trees of a graph G is n^{n-2}. ■

The above theorem can also be stated in terms of labelled trees as follows:

Theorem 6.5.6 — Cayley's Theorem on Labelled Trees. For $n \geq 2$, the number of labelled trees with n vertices is n^{n-2}.

Proof. Every labelled tree on n vertices can be treated as a spanning tree of K_n. Therefore, the theorem follows immediately from Theorem 6.5.5. ■

6.6 Fundamental Circuits

Theorem 6.6.1 A connected graph G is a tree if and only if adding an edge between any two vertices in G creates exactly one cycle (circuit).

Proof. First assume that G is a tree and let u,v be any two vertices of G. Then, by Theorem 6.1.1, there exists a unique path, say P, between u and v. Add an edge between these two vertices. Then, $P+uv$ is clearly a cycle in the graph $H=G+uv$. If possible, let uv be an edge in two cycles, say C and C' in H. Then, $C-uv$ and $C'-uv$ are two disjoint uv-paths in $H-uv=G$, contradicting the uniqueness of P. Hence, $P+uv$ is the only cycle in $G+uv$.

Conversely, assume that $P+uv$ is the only cycle in the graph $H=G+uv$. Then, $G=H-uv$ is a connected graph having no cycles. That is, G is a tree. ■

Theorem 6.6.2 Adding a chord of a connected graph G to the corresponding spanning tree T of G creates a unique cycle and in G.

Proof. The proof of the theorem is a consequence of Theorem 6.6.1. ■

Definition 6.6.1 — Fundamental Cycles. A cycle formed in a graph G by adding a chord of a spanning tree T f G is called a *fundamental circuit* or *fundamental cycle* in G.

Definition 6.6.2 — Cyclomatic Number. The *cyclomatic number* or *circuit rank*, or *cycle rank* of an undirected graph is the minimum number of edges that must be removed from the graph to break all its cycles, making it into a tree or forest.

Clearly, the cyclomatic number of a graph G is equal to the nullity of G.

Theorem 6.6.3 Any connected graph G with n vertices and ε edges components has $\varepsilon - n + 1$ fundamental cycles.

Proof. The number of chords corresponding to any spanning tree T of G on n vertices is $\varepsilon - n + 1$ (see Theorem 6.5.2). By Theorem 6.6.1, we know that corresponding to each chord in G, there exists a unique fundamental circuit in G. Therefore, the number of fundamental circuits in G is $\varepsilon - n + 1$. ■

Theorem 6.6.4 Any connected graph G with n vertices, ε edges and k components has $\varepsilon - n + k$ fundamental cycles.

Proof. Any spanning acyclic subgraph F of G with n vertices and k components (may be called a *spanning forest* of G) can have exactly $n - k$ edges (see the first part of Theorem 3.2.4). That is, the number of branches in F is $n - k$. Therefore, the number of chords is $\varepsilon - n - k$. Hence, by Theorem 6.6.1, we know that corresponding to each chord in G, there exists a unique fundamental circuit in G. Therefore, the number of fundamental circuits in G is $\varepsilon - n + k$. ■

6.7 Rooted Tree

Definition 6.7.1 — Rooted Tree. A *rooted tree* is a tree T in which one vertex is distinguished from all other vertices. This particular vertex is called the *root* of T.

Figure 6.5 illustrates some rooted trees on five vertices. In all these graphs, the white vertices represent the roots of the rooted trees concerned.

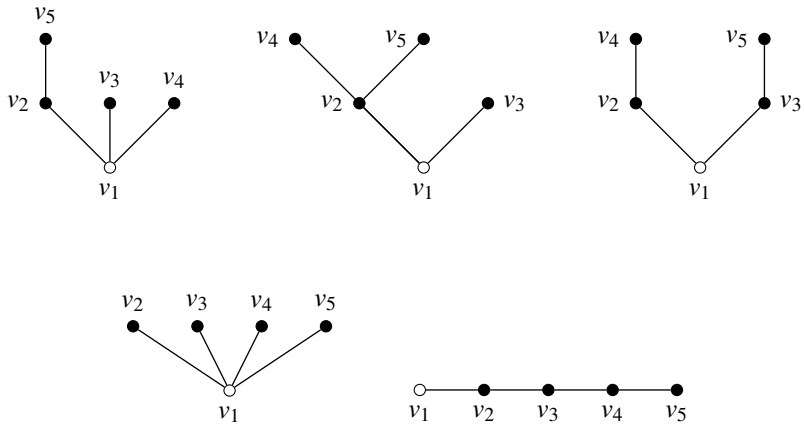


Figure 6.5: Some rooted trees on five vertices

In certain practical or real-life problems, we may have to calculate the lengths or total lengths of vertices of rooted trees from the roots. In some cases, we may also need to assign weights to the vertices of a tree.

Consider the following binary tree whose pendant vertices are assigned some weights.

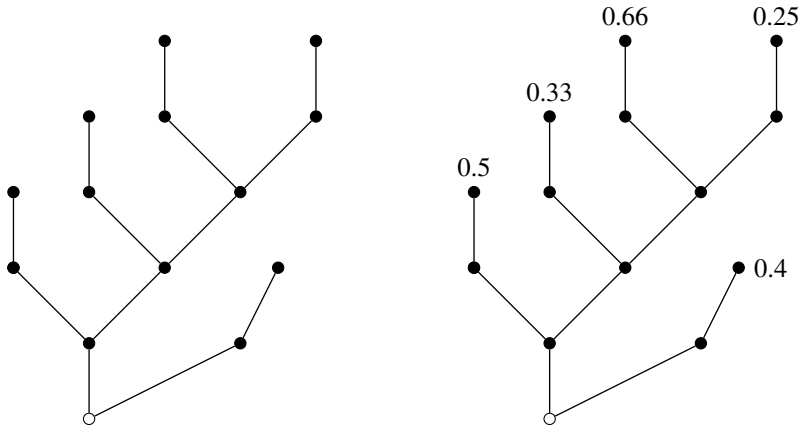


Figure 6.6: A rooted tree with and without weights to pendant vertices.

Definition 6.7.2 — Path Length of a Rooted Tree. The *path length* or (*external path length*) of a rooted tree T is the sum of the levels of all pendant vertices.

The path lengths of rooted trees are widely applied in the analysis of algorithms. The path length of the first rooted tree in Figure 6.6 is $2+3+4+5+5=19$.

Definition 6.7.3 — Weighted Path Length of a Rooted Tree. If every pendant vertex v_i of a tree T is assigned some positive real number w_i, then the *weighted path length* of T is defined as $\sum_i w_i \ell_i$, where ℓ_i is the level of the vertex v_i from the root.

The weighted path length of the graph in Figure 6.6 is $2\times 0.4+3\times 0.5+4\times 0.33+5(0.66+0.25)=8.17$.

6.8 Binary Tree

Definition 6.8.1 — Binary Tree. A *binary tree* is a rooted tree in which there is only one vertex of degree 2 and all other vertices have degree 3 or 1. The vertex having degree 2 serves as the root of a binary tree.

Theorem 6.8.1 The number of vertices in a binary tree is odd.

Proof. Let T be a binary tree. Note that the only vertex of T which has even degree is the root. We also have the result that the number of odd degree vertices in any graph is even. Hence, the number of vertices in T is odd. ■

Every non-pendant vertex of a binary (or rooted) tree is called its *internal vertex*. A vertex v of a binary tree is said to be *at level* ℓ if its distance from the root is ℓ.
A rooted tree with its levels is illustrated in Figure 6.7.

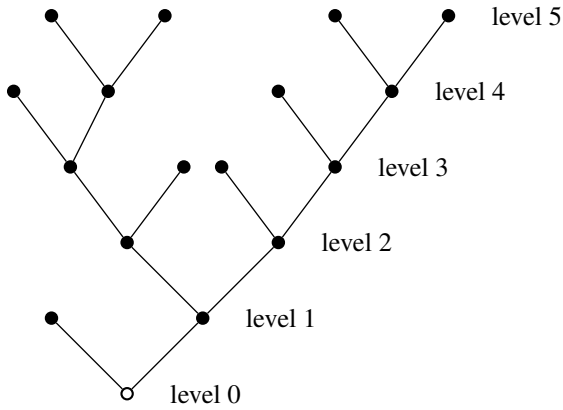


Figure 6.7: A 17-vertex binary tree with level 5.

Theorem 6.8.2 A binary tree on n vertices has $\frac{n+1}{2}$ pendant vertices.

Proof. Let p the number of pendant vertices in T. Then, the number of vertices of degree 3 is $n-p-1$. Then, we have

$$\begin{aligned} |E(T)| &= \frac{1}{2}\sum_{v\in V(T)} d(v) \text{(First Theorem on Graph Theory)} \\ &= \frac{1}{2}[2+p+3(n-p-1)] \\ &= n-1. \end{aligned}$$

Hence, we have $p = \frac{n+1}{2}$. This complete the proof. ■

The binary trees are widely used search procedures. In such procedures, each vertex of a binary tree represents a test and with two possible outcomes. Usually, we have to construct a binary tree on n vertices, for given values of n, with or without a fixed number of levels. This makes the study on the bounds on the number of levels.

Theorem 6.8.3 Let T be a k-level binary tree on n vertices. Then, $\lceil \log_2(n+1)-1 \rceil \leq k \leq \frac{n-1}{2}$, where $\lceil x \rceil$ represents the smallest integer greater than or equal to x (ceiling function).

Proof. Note that there is one vertex at level 0 (root), at most two vertices at level 1, at most four vertices at level 2, at most eight vertices at level 3 and proceeding like this, there are at most 2^k vertices at level k.

Figure6.7 is an example for a binary tree with fewer than 2^k vertices at the k-th level for some k, while Figure 6.8 is an example for a binary tree with exactly 2^k vertices at the k-th level for some k.

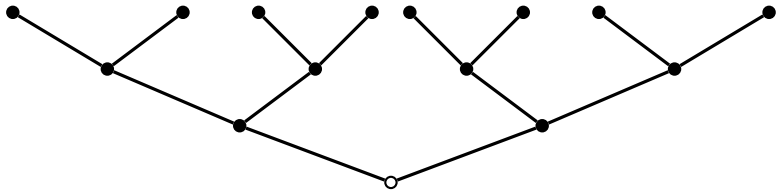


Figure 6.8: A rooted trees 2^k vertices at k-th level, where $k = 0, 1, 2, 3$

Therefore, we have

$$\begin{aligned} n &\leq \sum_{i=0}^{k} 2^i \\ &= \frac{2^{k+1}-1}{2-1} \quad \text{(Since the above series is a geometric series with } r = 2.\text{)} \\ \therefore n &\leq 2^{k+1}-1 \\ n+1 &\leq 2^{k+1} \\ \text{That is, } \log_2(n+1) &\leq k+1 \\ \text{Therefore, } \log_2(n+1)-1 &\leq k. \end{aligned}$$

Since k is an integer, the above equation becomes $\lceil \log_2(n+1)-1 \rceil \leq k$.

If we construct a binary tree T such that there are exactly two vertices at every level other than the root level, then T attains the maximum possible number of levels (see Figure 6.9).

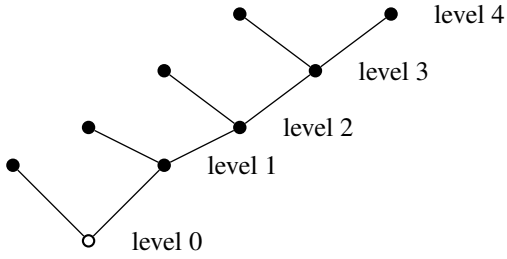


Figure 6.9: A 9-vertex binary tree with level 4.

From this figure, we can interpret that half of the $n-1$ edges of T are drawn towards the left and the remaining half are drawn to the right. Therefore, we can see that the maximum possible value of k is $\frac{n-1}{2}$. ■

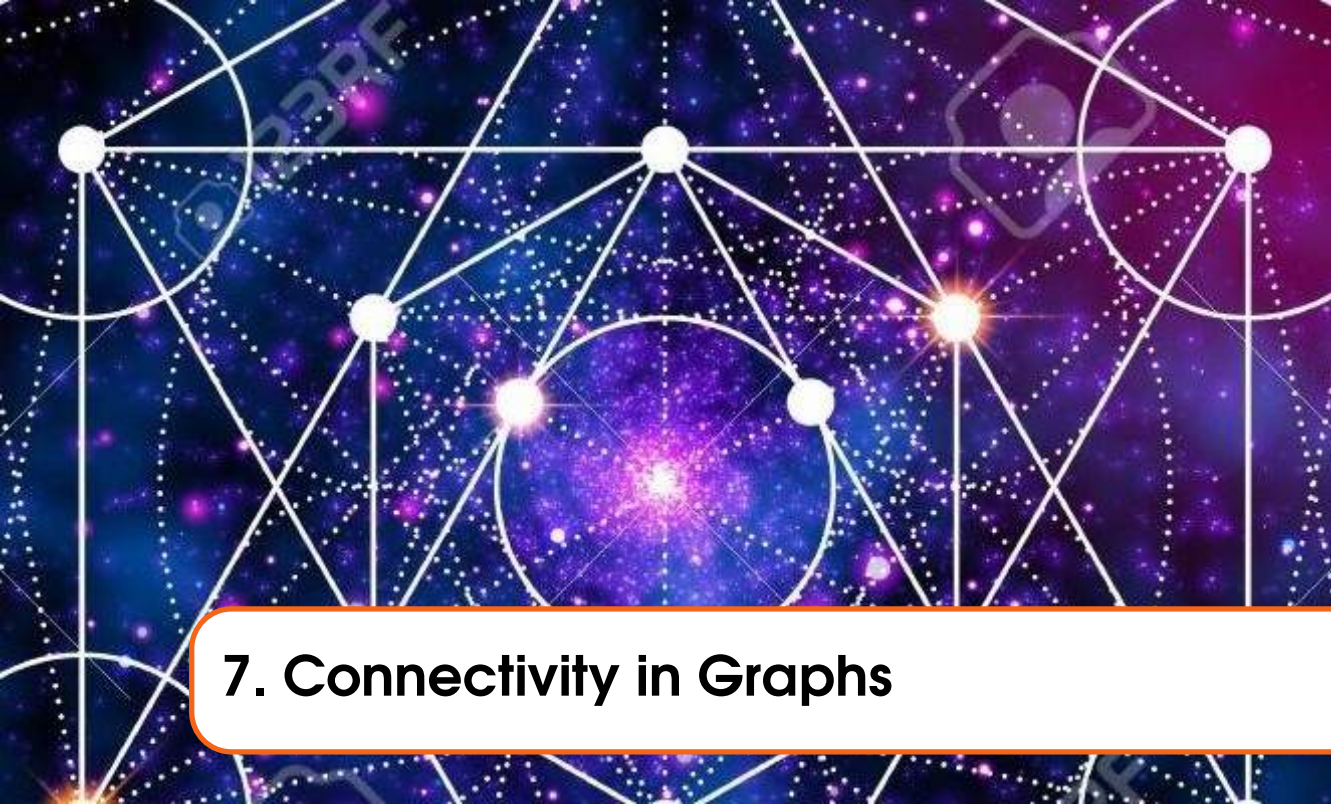
An ℓ-level binary tree is said to be a *complete binary tree* if it has 2^k vertices at the k-th

level for all $1 \leq k \leq \ell$. Figure 6.8 is a complete binary tree of level 3.

6.9 Exercises

1. Draw all unlabeled rooted trees on n vertices, where $n = 1,2,3,4,5,6$.
2. Show that a path is its own spanning tree.
3. Show that a pendant edge of a graph will be contained in all of its spanning trees.
4. What is the nullity of a complete K_n?
5. Show that every Hamilton path of a graph G (if exists) is a spanning tree of G.
6. Show that every cycle (or circuit) in G has at least one common edge with a chord set.
7. Prove that a graph G is a tree if and only if it is a loopless and has exactly one spanning tree.
8. Every graph with fewer edges than vertices has a component that is a tree.
9. Prove that a maximal acyclic subgraph of G consists of a spanning tree from each component of G.
10. Prove that every graph of order n and size ε has at least $\varepsilon - n + 1$ cycles.
11. Prove that a simple connected graph having exactly 2 vertices that are not cut-vertices is a path.
12. Let G be connected and let $e \in E$. Then, show that
    (a) e is in every spanning tree of G if and only if e is a cut edge of G;
    (b) e is in no spanning tree of G if and only if e is a loop of G.
13. Show that if each degree in G is even, then G has no cut edge

IV CONNECTIVITY & PLANARITY



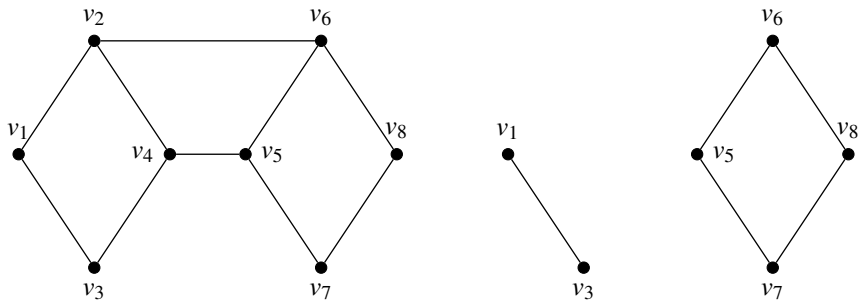
7. Connectivity in Graphs

7.1 Cut-Vertices and Vertex-Cuts of a Graph

First recall that a *cut-vertex* of a graph G is a vertex v in G such that $G-v$ is disconnected. A cut-vertex is also called a *cut-node* or an *articulation point*.

Definition 7.1.1 — Vertex-Cut. A subset W of the vertex set V of a graph G is said to be a *vertex-cut* or a *Separating Set* of G if $G-W$ is disconnected. In the above graph, the vertex set $W=\{v_3,v_4\}$ is a vertex-cut of G.

The following graph illustrates a vertex-cut of a graph.



(a) Connected graph G (b) Disconnected graph $G-\{v_2,v_4\}$

Figure 7.1: disconnected graph $G-\{v_4v_5,v_2v_6,v_3v_7\}$.

R If v is a cut-vertex of G, then $\{v\}$ is a vertex-cut of G. Any set of cut vertices in G is also a vertex-cut of G.

Theorem 7.1.1 Every internal vertex of a tree is a cut-vertex.

Proof. An internal vertex of a graph G is a vertex v with degree greater than or equal to 2. Therefore, the result follows from Theorem 6.1.10. ■

Theorem 7.1.2 Every connected graph on three or more vertices has at least two vertices which are not cut-vertices.

Proof. Let G be a connected graph of order $n \geq 3$ and let T be a spanning tree of G. Then, by Theorem 6.1.8, there exist at least two pendant vertices in T. Note that no pendant vertex can be a cut-vertex of a graph. Let v be one of the pendant vertices in T. Then, $T - v$ is also a connected (acyclic) graph. Hence, $T - v$ is the spanning tree of the graph $G - v$. Therefore, $G - v$ is connected and hence v is not a cut-vertex of G also. That is, the pendant vertices in a spanning tree of a graph G will not be the cut-vertices of G as well. Therefore, there exist at least two pendant vertices in G. ■

The following result is a necessary and sufficient condition for a vertex of a graph G to be a cut-vertex of G.

Theorem 7.1.3 A vertex v of connected graph G is a cut-vertex of G if and only if there exist two (or more) edges incident on v such that no cycles in G contain both (all) edges simultaneously.

Proof. Let v be an arbitrary vertex of a graph G and let $e_1 = uv$ and $e_2 = vw$ be two edges incident on v.

Assume v be cut-vertex of G. If possible, let e_1 and e_2 lie on the same cycle, say C. Then, the path uvw and the path $C - v$ are two distinct uw-paths in G. Hence, u and w lie in the same component of $G - v$, which is a contradiction to the assumption that v is a cut-vertex of G. Therefore, C contains any one of these two edges.

Conversely, assume that no cycle in G contains both edges e_1 and e_2 incident on v. Then, uvw is the only uw-path in G and hence u and w lie on two distinct components in $G - v$. Then, v is a cut-vertex of G, completing the proof. ■

The following result is another necessary and sufficient condition for a vertex of a graph G to be a cut-vertex of G.

Theorem 7.1.4 A vertex v of connected graph G is a cut-vertex of G if and only if there exist two vertices, say x and y, in G such that every uw-path passes through v.

Proof. Let v, x, y be three vertices of a graph G such that every xy-paths in G pass through v. Then, x and y are not connected in $G - v$. That is, $G - v$ is disconnected and hence v is a cut edge of G.

Let v be a cut vertex of a graph G. Let x and y be two vertices such that they lie on two components in $G-v$. If possible, let some xy-paths pass through v and some other do not. Let P be an xy-path which does not pass through v.

Pick two adjacent vertices u and w of v such that they lie on different components of $G-v$. Without loss of generality, let u and x are in the same component, say K_1, of $G-v$, while w and y are in the other component, say K_2, of $G-v$. Since K_1 is connected, there exists a path, say P_1, between u and x. Similarly, there exists a path P_2 between w and y in K_2. Then, we note that $P_1 \cup P \cup P_2 \cup uvw$ is a cycle consisting of both edges uv and vw in G, contradicting the statement of Theorem 7.1.3. Therefore, all xy-paths should pass through v. This completes the proof. ■

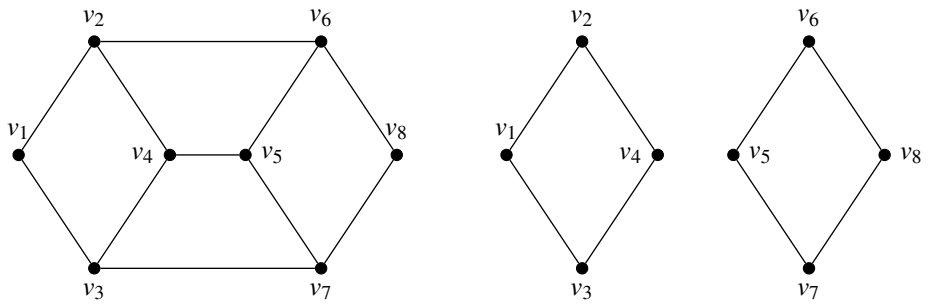
7.2 Cut-Sets of a Graph

Recall that an edge e of a graph G is a cut-edge of G if $G-e$ is disconnected.

Definition 7.2.1 — Edge-Cut. A subset F of the edge set E of a graph G is said to be an *edge-cut* of G if $G-E$ is disconnected.

Definition 7.2.2 — Bond of a graph. A minimal nonempty edge-cut of a graph G is called a *bond* of G.

The following graph illustrates a vertex-cut of a graph.



(a) Connected graph G (b) $G \circ uv$

Figure 7.2: Disconnected graph $G-\{v_4v_5, v_2v_6, v_3v_7\}$.

In the above graph G, the edge set $F = \{v_4v_5, v_2v_6, v_3v_7\}$ is an edge-cut, since the removal of F makes G disconnected.

Definition 7.2.3 — Cut-Set. A *cut-set* is a minimal edge-cut of G. That is, a cut-set of a graph G is a set of edges F of G whose removal makes the graph disconnected, provided the removal of no proper subset of F makes G disconnected.

A cut-set also called a *minimal cut-set*, a *proper cut-set*, a *simple cut-set*, or a *cocycle*. Note that the edge-cut F in the above example is a cut-set of G.

If a cut-set puts two vertices v_1 and v_2 into two different components, then, it is called a cut-set with regard to v_1 and v_2 .

R If e is a cut-edge of G, then $\{e\}$ is an edge-cut of G. Any set of cut-edges in G is also an edge-cut of G.

If $\{e\}$ is a cut-set of G (That is, when e is a cut edge of G), then it is customary to say that e is a cut-set of G. Hence, we have

Theorem 7.2.1 Every edge of a tree is a cut-set.

Proof. The proof is an immediate consequence of the fact that every edge of a tree is a cut-edge of G (see Theorem 6.1.5). ■

Theorem 7.2.2 Every cut-set in a graph G must contain at least one branch of every spanning tree of G.

Proof. Let F be a cut-set in G and T be any spanning tree of G. If F does not contain any edge of T, then, by Theorem 6.1.1, there will be a unique path between any pair of vertices in T and hence in $G-F$, contradicting the hypothesis that F is a cut-set in G. Therefore, every cut-set in a graph G must contain at least one branch of every spanning tree of G. ■

Theorem 7.2.3 In any connected graph G, any minimal set of edges consisting of at least one branch of every spanning tree of G is a cut-set.

Proof. Let F be a minimal set of edges consisting of at least one branch of every spanning tree T of G. Then, $G-F$ will remove at least one edge from every spanning tree of G. Therefore, $G-F$ is disconnected. Since F is minimal, we have $(G-F)+e$ contains a spanning tree of G. Therefore, F is a cut-set of G, completing the proof. ■

Theorem 7.2.4 Every cycle (circuit) in G has even number of edges in common with any cut-set of G.

Proof. Let S be a cut-set in G. Then, $G-S$ is disconnected. Let V_1 and V_2 be the two disjoint subsets of $V(G-S)$. Any cycle which lies entirely in V_1 (or V_2) does not have any common edge with a cut-set of G. If a cycle C has vertices in both V_1 and V_2, we have to traverse back and forth between V_1 and V_2 to traverse the cycle C. Since every edge in S has one end in V_1 and the other end in V_2, the number of edges common to C and S must be even. ■

The above theorem can easily be verified from the graph G in Figure 7.2.

Theorem 7.2.5 The ringsum of any two cut-sets in a connected graph G is a cut-set or an edge-disjoint union of cut-sets.

Proof. Let S_1 and S_2 be two cut-sets in a connected graph G. Let (V_1, V_2) be a unique partition of $V(G)$ with respect to S_1, whereas (U_1, U_2) be a unique partition of $V(G)$ with respect to S_2. Then, we have

$$V_1 \cup V_2 = V; V_1 \cap V_2 = \emptyset;$$
$$V_3 \cup V_4 = V; V_3 \cap V_4 = \emptyset.$$

Now, we have

$$(V_1 \cap V_4) \cup (V_2 \cap V_3) = V_1 \oplus V_3 = V_5;$$
$$(V_1 \cap V_3) \cup (V_2 \cap V_4) = V_2 \oplus V_4 = V_6.$$

It is to be noted that the ringsum $S_1 \oplus S_2$ consists of the edges that join the vertices in V_5 to those in V_6. Thus, the set of edges $S_1 \oplus S_2$ partitions $V(G)$ into two sets V_5 and V_6 such that

$$V_1 \cup V_2 = V \; ; \; V_1 \cap V_2 = \emptyset.$$

Hence, $S_1 \oplus S_2$ is a cut-set if subgraphs containing V_5 and V_6 remain connected in $G - (S_1 \oplus S_2)$. Otherwise, $S_1 \oplus S_2$ is just an edge-disjoint union of two cut-sets. ■

7.3 Fundamental Cut-Sets

Definition 7.3.1 — Fundamental Cut-Set. Let T be a spanning tree of a connected graph G. A cut set S of G containing exactly one branch of T is called a *fundamental cut-set* or a *basic cut-set* of G with regard to T.

Theorem 7.3.1 Every connected graph of order n has $n-1$ distinct fundamental cut-sets corresponding to any spanning tree of G.

Proof. Let G be a connected graph and T be a spanning tree of G. Then, by Theorem 7.2.2, any cut-set of G contains an edge of T. Consider an edge e of T. Clearly, $\{e\}$ is a cut-set in G and partitions $V = V(T)$ into two disjoint sets. Consider the same partition of V in G also and let S be a cut-set in G corresponding to this partition. Then, S consists of exactly one edge of T and all other elements of S will be chords of T and hence is a fundamental cut-set. Therefore, S can be considered as a unique fundamental cut-set of G with respect of the branch e of T in G. That is, every edge of T corresponds to a unique fundamental cut-set in G. Hence, the number of distinct fundamental cut-sets corresponding to T is $n-1$, the order of T. This completes the proof. ■

Theorem 7.3.2 With respect to a spanning tree, a chord c that determines a fundamental circuit C occurs in every fundamental cut-set associated with the branches in T and in no other.

Proof. Consider a spanning tree T of G. Let c be a chord with respect to T and let the fundamental circuit made by c be called C, consisting of k branches $b_1, b_2, \ldots, b_k$ in addition to c. That is, $C = \{c, b_1, b_2, \ldots, b_k\}$ be a fundamental circuit with respect to T. Every branch of any spanning tree is associated with a unique fundamental cut-set (see Theorem 7.2.2). Let S_1 be the fundamental cut-set associated with b_1, which consists of q chords other than b_1. That is, $S_1 = \{b_1, c_1, c_2, \ldots, c_q\}$ is a fundamental cut-set with respect to T. By Theorem 7.2.4, there must be an even number of edges common to C and S_1. The edge b_1 is common in C and S_1 and c is the only other edge that can possibly be in both C and S_1. Therefore, c is one of the chords $c_1, c_2, \ldots, c_q$.

The same argument holds for the fundamental cut-sets corresponding to $b_2, b_3, \ldots b_k$ and hence c is contained in every fundamental cut-set associated with branches in C.

If c belongs to any other fundamental cut-set S' other than those mentioned above, then there would be only one edge c common to S' and C, a contradiction to Theorem 7.2.4. Therefore, c is not contained in any other fundamental cut-sets other than those corresponding to $b_1, b_2, b_3, \ldots, b_k$. This completes the proof. ■

Theorem 7.3.3 With respect to a spanning tree, a branch b that determines a fundamental cut-set S is contained in every fundamental circuit associated with the chords in S and in no other.

Proof. Let the fundamental cut-set S determined by b be $S = \{b, c_1, c_2, \ldots, c_p\}$ and C' be the fundamental circuit determined by chord c_1. Then, $C' = \{c_1, b_1, b_2, \ldots, b_q\}$. Since the number of edges common to S and C' must be even, b_1 must be in C'. The same is true for the fundamental circuits formed by $c_2, c_3, \ldots, c_p$. As explained in previous theorem b will be one among $b_1, b_2, \ldots, b_q$.

If b_1 belongs to a fundamental circuit C" made by a chord other than $c_1, c_2, c_3, \ldots, c_p$, then since none of $c_1, c_2, c_3, \ldots, c_p$ is in C", b_1 is the only edge common to C" and S, contradicting the statement of Theorem 7.2.4. Hence, b does not belong to any fundamental circuit made by a chord other than $c_1, c_2, c_3, \ldots, c_p$. This completes the proof. ■

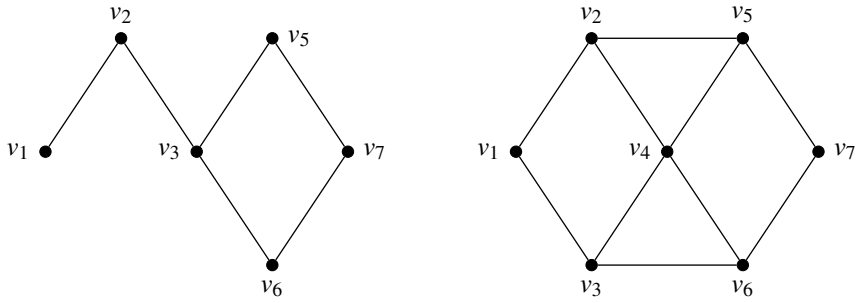
7.4 Connectivity in Graphs

Definition 7.4.1 — Separable Graph. A connected graph G (or a connected component of a graph) is said to be a *separable graph* if it has a *cut-vertex*.

Definition 7.4.2 — Non-separable Graph. A connected graph or a connected component of a graph, which is not separable, is called *non-separable graph*.

Figure 7.3(a) depicts a separable graph, while Figure 7.3(b) depicts a non-separable graph.

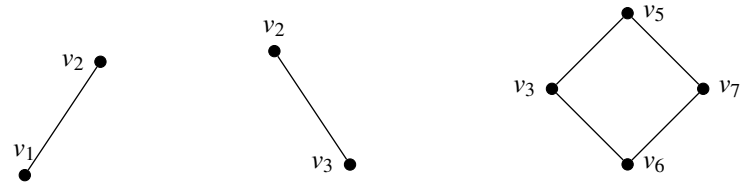
Definition 7.4.3 — Block. A non-separable subgraph of a separable graph G is called a *block* of G.



(a) A separable graph (b) A non-separable graph

Figure 7.3: Illustration to separable and non-separable graphs

In Figure 7.3(a), v_2 and v_3 are cut-vertices. Hence, the three blocks of that graph are given in Figure 7.4:



(a) Block 1 (b) Block 2 (c) Block 3

Figure 7.4: Illustration to blocks of a graph

Definition 7.4.4 — Edge Connectivity of a Graph. Let G be a graph (may be disconnected) having k components. The minimum number of edges whose deletion from G increases the number of components of G is called the *edge connectivity* of G. The edge connectivity of G is denoted by $\lambda(G)$.

R Note that

1. The number of edges in the smallest cut-set of a graph is its edge connectivity.
2. Since every edge of a tree T is a cut-set, the edge connectivity of T is 1.

Theorem 7.4.1 The edge connectivity of a graph G is less than or equal to its minimum degree. That is, $\lambda(G) \leq \delta(G)$.

Proof. Let v_i be a vertex in G such that $d(v_i) = \delta(G)$. Then, note that v_i can be separated from G only after the removal of $d(v_i)$ edges incident on v_i. Therefore, $\lambda(G) \leq \delta(G)$. ■

Definition 7.4.5 — Vertex Connectivity of a Graph. Let G be a graph (may be disconnected). The minimum number of vertices whose deletion from G increase the number of components of G is called the *vertex connectivity* of G. The vertex connectivity of G is denoted by $\kappa(G)$.

Theorem 7.4.2 The vertex connectivity of any graph G is less than or equal to the edge connectivity of G. That is, $\kappa(G) \leq \lambda(G)$.

Proof. Let $\lambda = \lambda(G)$ and $\kappa = \kappa(G)$. Since λ is the edge connectivity of G, there exists a cut-set S in G with λ edges. Note that the maximum number of vertices to be removed from G to delete λ edges from G is λ. Therefore, $\kappa \leq \lambda$, completing the proof. ■

Theorem 7.4.3 Every cut-set in a non-separable graph with more than two vertices contains at least two edges.

Proof. A graph is non-separable if its vertex connectivity is at least two. In view of Theorem 7.4.2, edge connectivity of a non-separable graph must be at least two which is possible if the graph has at least two edges. ■

Theorem 7.4.4 The maximum vertex connectivity of a connected graph G with n vertices and ε edges is $\lfloor \frac{2\varepsilon}{n} \rfloor$.

Proof. We know that $\frac{2\varepsilon}{n}$ is the average degree of G (see Theorem 1.2.2) and hence there must be at least one vertex in G whose degree is less than or equal to the number $\lfloor \frac{2\varepsilon}{n} \rfloor$. Also, we have $\delta(G) \leq \frac{2\varepsilon}{n}$ (see Theorem 1.2.2). By Theorem 7.4.2, we have $\kappa(G) \leq \lambda(G) \leq \lfloor \frac{2\varepsilon}{n} \rfloor$. Hence, $\kappa(G) \leq \lfloor \frac{2\varepsilon}{n} \rfloor$. ■

Theorem 7.4.5 — Whitney's Inequality. For any graph G, we have $\kappa(G) \leq \lambda(G) \leq \delta(G)$.

Proof. The proof follows immediately from Theorem 7.4.1 and Theorem 7.4.2. ■

Theorem 7.4.6 If G is a cubic graph (3-regular graph), then $\kappa(G) = \lambda(G)$.

Proof. Let S be a minimum vertex-cut of a cubic graph G. Then, $|S| = \kappa(G) \leq \lambda(G)$. Let H_1 and H_2 be two components of $G - S$. Since S is a minimal vertex-cut of G, every vertex in S must have a neighbour in H_1 and a neighbour in H_2. Since G is 3-regular, v cannot have two neighbours in H_1 and two neighbours in H_2.

Now, delete the edge from v to a member of $\{H_1, H_2\}$, when v has only one neighbour. If a path enters S via one vertex v_i in S and leaves for H_2 via another vertex v_j in S, adjacent to v_i, then we delete the edges to H_1 for both v_i and v_j. Clearly, the number of edges deleted here (both cases together) is equal to the number of vertices in S and these $\kappa(G)$ edges breaks all paths from H_1 to H_2. Hence, $\kappa(G) = \lambda(G)$. ■

Definition 7.4.6 A graph G is said to be *k-connected* if its vertex connectivity is k (That is, if $\kappa(G) = k$). A graph G is *k-edge connected* if its edge connectivity is k (That is, if $\lambda(G) = k$.)

Theorem 7.4.7 A graph G is k-connected if and only if there exist at least k disjoint paths (paths without common vertices) between any pair of vertices in G.

Proof. Assume that G is k-connected. Then, k vertices are to be removed to make G disconnected. If there are fewer than k disjoint paths between any two vertices u and v in G, then u and v will be in different components on removing fewer than k vertices from G, a contradiction to the fact that G is k-connected. Therefore, there must be at least k disjoint paths between any pair of vertices in G.

Conversely, assume that there exist at least k disjoint paths between any pair of vertices u and v in G. Then, we note that the removal of k vertices, one from each one of the k disjoint paths, leaves u and v in two different components. Moreover, removing fewer than k vertices will not leave u and v in two different components. Hence, G is k-connected. ■

Theorem 7.4.8 A graph G is k-edge connected if and only if there exist at least k edge-disjoint paths (paths that may have common vertices, but no common edges) between any pair of vertices in G and at least one pair of vertices is joined by exactly k edge disjoint paths.

Proof. The result is straight-forward from the fact that if two vertices u and v of a graph are connected by k-edge disjoint paths, then u and v are disconnected if and only if k edges, one from each of the k paths between them, are removed. ■

Theorem 7.4.9 — Whitney's Theorem. A graph with at least three vertices is a 2-connected graph if and only if there exist internally disjoint paths between any pair of vertices in G.

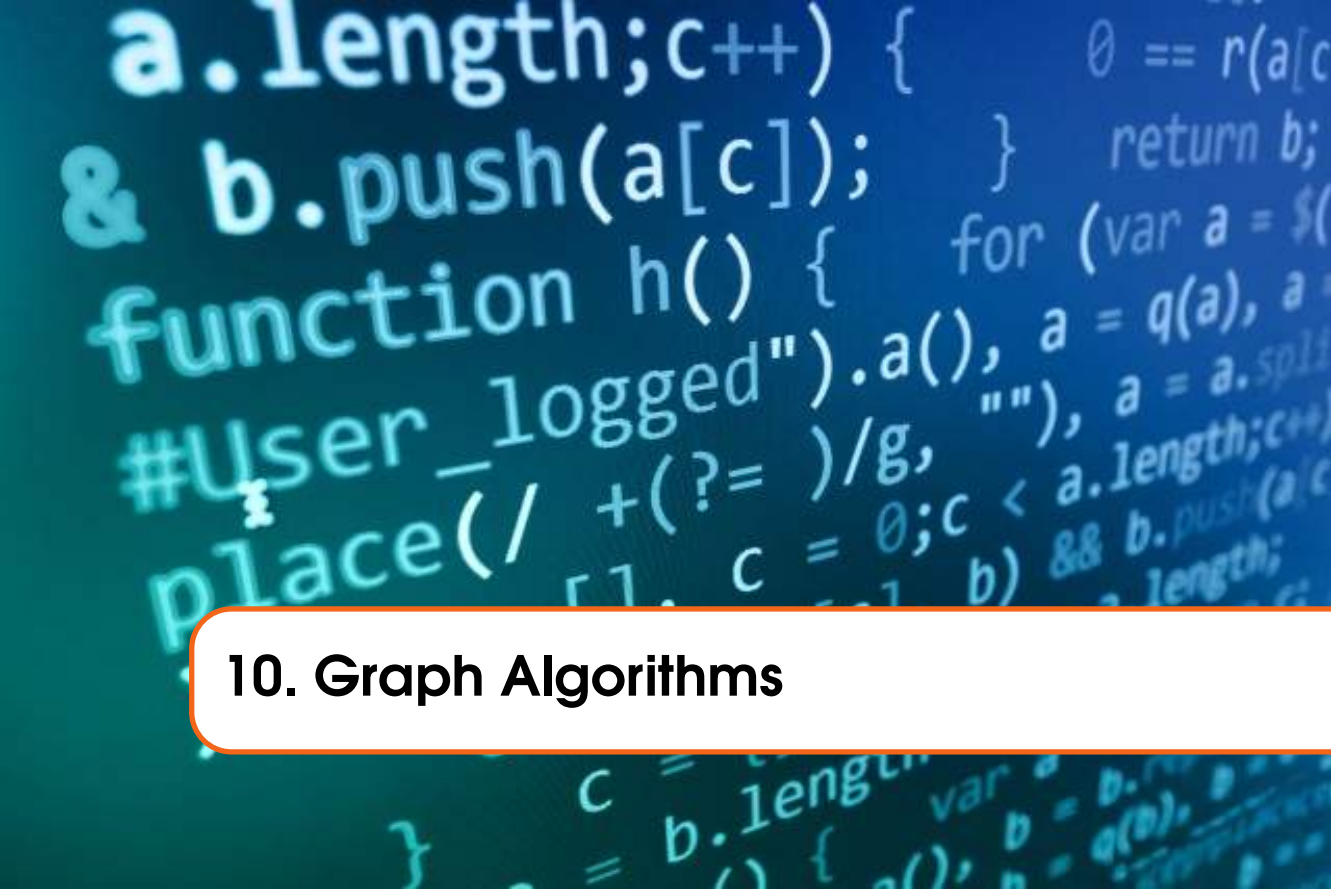
Proof. The proof is a specific case of Theorem 7.4.7, where $k = 2$. ■

7.5 Exercises

1. Show that every edge-cut is a disjoint union of bonds.
2. if G has a cut-edge, then show that G has a vertex v such that $\omega(G-v) > \omega(G)$. Is the converse true? Justify your answer.
3. Show that a simple connected graph that has exactly two vertices which are not cut-vertices is a path.
4. Show that if G is simple and $\delta > v-2$, then $\kappa(G) = 8$.
5. Show that if G has no even cycles, then each block of G is either K_1 or K_2, or an odd cycle.

6. Show that a connected graph which is not a block has at least two blocks that each contains exactly one cut vertex.
7. Let G be a connected graph with at least three vertices. Prove that if G has bridge then G has a cut vertex.
8. Let u and v be two vertices of a 2-connected graph. Then, show that there is a cycle in G passing through both u and v.
9. For a tree T with at least three vertices, show that there is a cut-vertex v of T such that every vertex adjacent to v, except for possibly one, has the degree 1.
10. Show that a cut-vertex of a graph is not a cut-vertex of its complement.
11. Show that $\kappa(G) \leq \frac{2\varepsilon}{n}$.
12. The ring sum of two distinct proper edge-cut sets is an edge-cut set.
13. Prove that any two connected graphs with n vertices, all of degree 2, are isomorphic.

VI GRAPH THEORETIC ALGORITHMS



10. Graph Algorithms

An algorithm may be considered as a set of instructions for solving certain mathematical problems when followed step by step will lead to the solutions of that problem. Usually, an algorithm is first written in ordinary language and then converted in to flowcharts and finally, written in a detailed and precise language so that the machine can execute it.

An algorithm must not only do what it is supposed to do, but also must do it efficiently, in terms of the requirements of both memory and computation time requirements as the function of the size of the input. In our case, the input will be graphs with order n and size ε.

10.1 Computer Representation of a Graph

A graph is generally presented to and is stored in a digital computer in one of the following forms:

1. *Adjacency Matrix:* Adjacency matrices are the mostly used form to represent a graph in computers. Assigning distinct numbers to the distinct vertices of G, the $n \times n$-binary matrix $X(G)$is used for representing G during the input, storage and output. Since each of the n^2 entries are either a 0 or a 1, the adjacency matrix requires n^2 bits of computational memory.

   As bits can be packed into words, each row of the adjacency matrix of G of order n may be written as a sequence of n^2-bits in $\lceil \frac{n}{w} \rceil$ machine words, where $w$is the word length (number of bits in a computer word). Therefore, the number of words required to store the adjacency matrix is $n\lceil \frac{n}{w} \rceil$.

   Since the adjacency matrix is symmetric, it will be sufficient if we have only the upper

diagonal elements of the matrix and only $\frac{n(n-1)}{2}$ bits of storage. But note that this increases the complexity and computation time, even though the storage space is saved. Another point is that adjacency matrix is defined for the graphs without parallel edges, this method cannot be used for representing graphs with parallel edges.

2. *Incidence Matrix:* Incidence matrices are also used to represent graphs. It requires $n \cdot \varepsilon$ bits of storage which might be more than the storage space required for the adjacency matrix (it is because the number of edges may be greater than the number of vertices in a vast majority of graphs). In certain practical cases such as electrical networks and switching networks incidence matrices are preferred than the adjacency matrices.
3. *Edge Listing:* Another way to represent a graph is just a list, or array, of $|E|$ edges, which is called an *edge list*. Clearly, parallel edges and self-loops can be included in this representation of a graph. Note that if we require b bits of storage for a vertex, then $2^{b-1} \leq n \leq 2^b$ and storing an edge requires $2b$ bits (Since an edge can be considered to be the pair of its end vertices). Therefore, the total storage required will be $2b \cdot \varepsilon$. The following example illustrates the edge-listing of the graph in Figure 10.1.

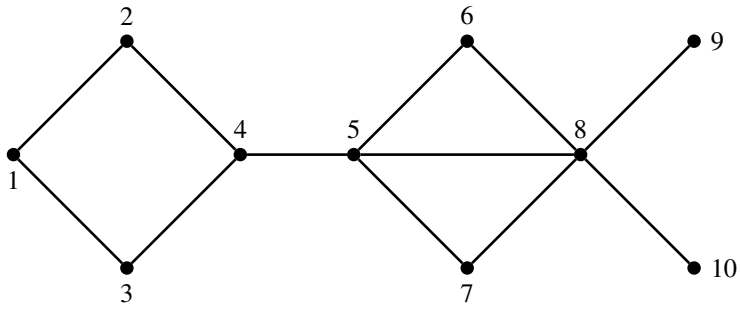


Figure 10.1: A graph G for computer input

The edge list of the graph in Figure 10.1, is written as: $L_E = \{1-2, 1-3, 2-4, 3-4, 4-5, 5-6, 5-7, 5-8, 6-8, 7-8, 8-9, 8-10\}$.
Edge listing is more economical (when compared to adjacency matrices), and hence we have $2b\varepsilon \leq n^2$. Edge listing is comparatively a convenient method for inputting a graph into the computer, but the retrieval and manipulation may be more difficult.

4. *Two Linear Arrays:* In this method, a graph is represented in two linear arrays, say F and H of vertices of the graph G, not necessarily disjoint, such that each edge has one end vertex in F and other in H. This method is found useful to represent weighted graphs and has the same storage requirements as that for edge listing.
The two-linear array of the graph in Figure 10.1 is written in the table given below:

F	1	1	2	3	4	5	5	5	6	7	8	8
H	2	3	4	4	5	6	7	8	8	8	9	10

Table 10.1: Two Linear Array representation of a graph.

5. *Successor Listing:* This method can efficiently be used for representing those graphs with $\frac{\varepsilon}{n}$ ratio is small. Here, n linear arrays are used to represent the vertices and their successors in G. For example, by writing the list "$i : j,k,l$", it is meant that the vertices with labels j,k and l are successors of the vertex with label i.

F	1	2	3	4	5	6	7	8	9	10
H	2,3	2,4	1,4	2,3,5	4,6,7,8	5,8	5,8	5,6,7,9,10	8	8

Table 10.2: Neighbour (successor) listing of a graph.

For undirected graphs, neighbours are listed and hence each edge will appear twice in the list. This is an obvious redundancy. But the successor listing is more efficient than adjacent matrices if $d_{av} < \lceil \frac{n}{w} \rceil - 1$, where d_{av} is the average degree of G.

10.2 Algorithm for Connectedness and Components

Given the adjacency matrix of a graph G, is it possible to determine whether the graph G is connected? If not connected, what is the number of components in G? The following algorithm checks the connectedness of graphs.

10.2.1 Connectedness and Components Algorithm

S-1 : Set $H = G$ and component count $c = 1$.
S-2 : Choose an arbitrary vertex v of G.
S-3 : Fuse all vertices in the neighbourhood $N(v)$ with v and call this vertex v.
S-4 : If the number of vertices which are non-adjacent to v is the same as that before the fusion, then go to Step-5. Otherwise, go to Step-2.
S-5 : Delete the vertex v (with all fused vertex) from H call this new graph H.
S-6 : If H has any vertex left in it, then let $c \leftarrow c+1$ and go to Step-2. Otherwise, go to Step-7.
S-7 : Print each c with its vertices. Stop.

For illustration, consider the graph given in Figure 10.2:

In the initial step, set component count $c = 1$ and choose $v = v_1$. Fuse the vertices v_2, v_3, v_5 and v_6 to v and call the new vertex v (and represent this by a white vertex). Then we have the following graph.

Here, we notice that the number of vertices that non-adjacent to v is different before and after fusion. So, we need the same action again. In the next iteration, we fuse the vertex v_8 and v_{10} to $v = v_1$. Then, we have the reduced graph as shown in Figure 10.4

Here also, the number of vertices that non-adjacent to v is different before and after fusion. So, in the next iteration,fuse the vertex v_{12} to $v = v_1$. Then, we have the reduced graph as shown in Figure 10.5

Here, we notice that the number of vertices that non-adjacent to v is the same before and after fusion. So, we can delete the vertex v_1 from H and set $c \leftarrow c+1$, (that is, $c = 2$). Now,

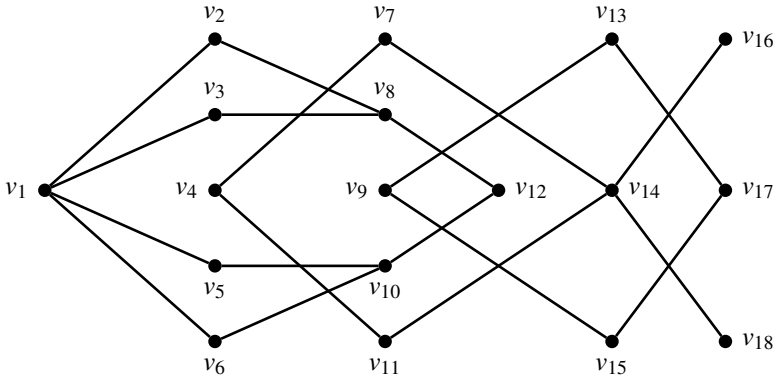


Figure 10.2: A graph G for computer input

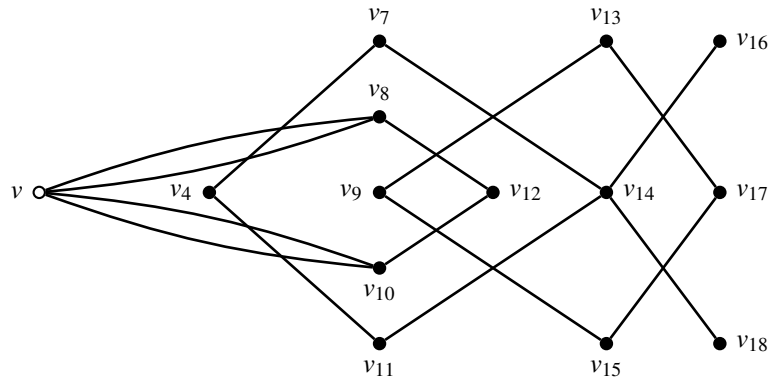


Figure 10.3: Components Algorithms Step-1

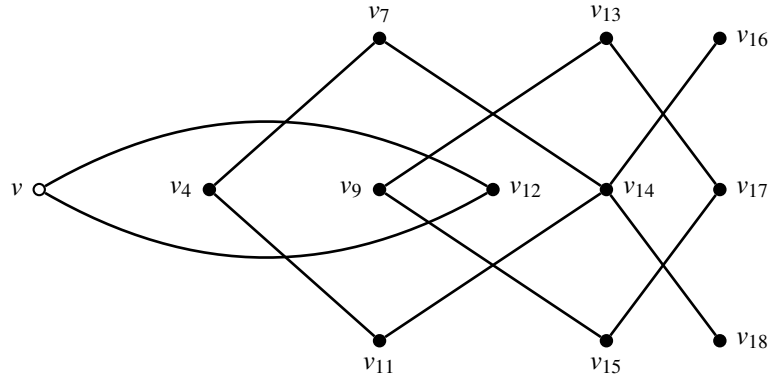


Figure 10.4: Components Algorithms Step-2

take $v = v_4$ in the reduced graph H and fuse the adjacent vertices v_7 and v_{11} to v. Then, we have the reduced graph as shown in Figure 10.6

At this stage, it can be noted that the number of vertices that non-adjacent to v is different

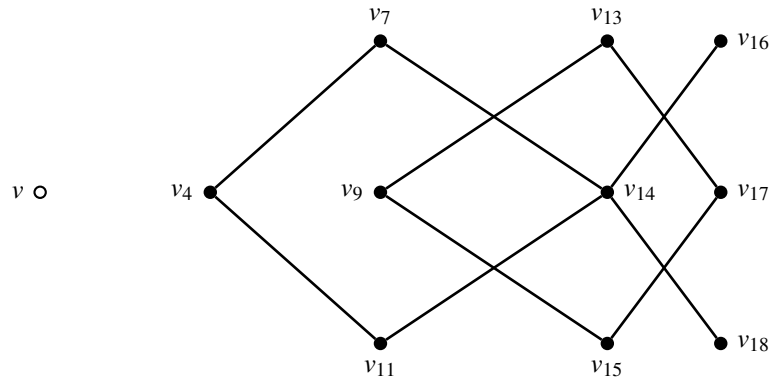


Figure 10.5: Components Algorithms Step-3

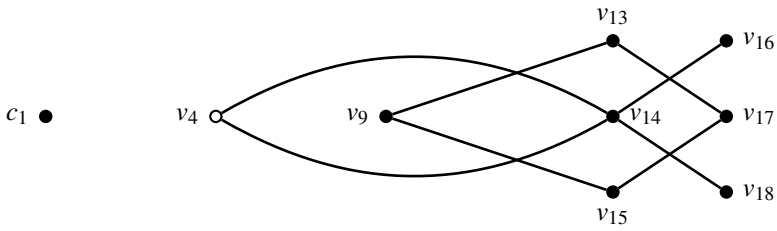


Figure 10.6: Components Algorithms Step-4

before and after fusion and hence we perform the same action again. That is, we fuse the vertex v_{14} to v. Then we get the reduced graph as shown in Figure 10.7.

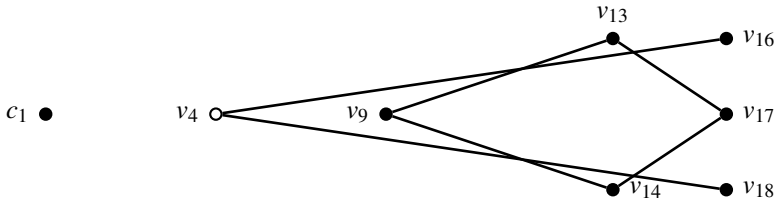


Figure 10.7: Components Algorithms Step-5

Here also, the number of vertices that non-adjacent to v is different before and after fusion and hence we fuse the vertex v_{16} and v_{18} to v. Then, v becomes an isolated vertex and hence we can delete v from H. In the new reduced graph H, take $v = v_9$ and set $c = 3$. Fuse the vertices v_{13} and v_{15} to v. Then, we have the reduced graph as given in Figure 10.8.

Finally fusing the vertex v_{16} to v, we have

In the above graph in Figure 10.9, each c_i corresponds to a component. Then, it can be understood that the given graph is disconnected and has three components which are as given in Figure 10.10.

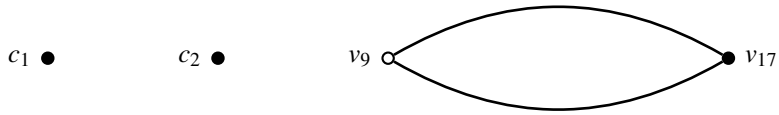


Figure 10.8: Components Algorithms Step-6

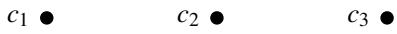
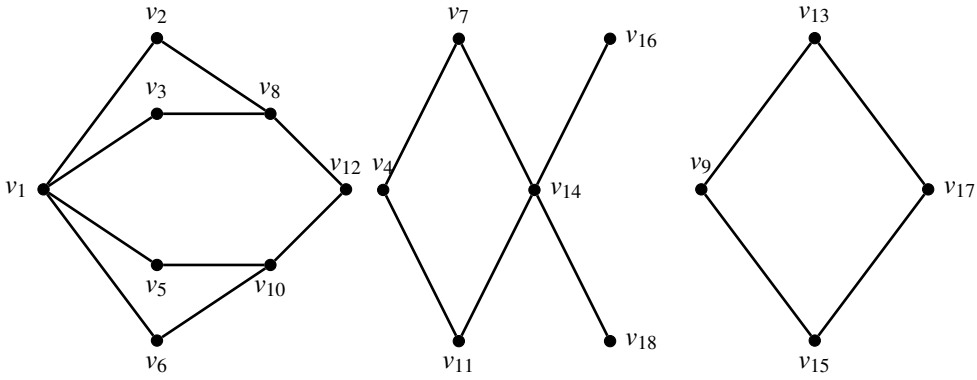


Figure 10.9: Components Algorithms Step-7



(a) Component-1 (b) Component-2 (c) Component-3

Figure 10.10: Components in a graph G.

10.3 Spanning Tree Algorithm

A *spanning tree algorithm* is the one which yields a spanning tree from a given connected graph and spanning forest from a given disconnected graph. Let G be a graph (connected or disconnected) with n vertices and ε edges. Then, this algorithm has the following steps.

S-1 : If it has loops, then, the loops may be discarded from G and the vertices be labeled $1,2,3,\ldots,n$ and the graph be arranged in two linear arrays F and H (as in linear array listing explained earlier) such that the i-th edge has end vertices $f_i \in F$ and $h_i \in H$.

S-2 : Choose an arbitrary edge $e_1 = f_1 h_1$ and set the component number $c = 1$.

S-3 : At the k-th stage, where $1 \leq k \leq \varepsilon$, one of the following situations may arise:

(a) If for the edge e_k, neither f_k nor h_k is included in any of the trees constructed so far in G, that edge may be named as a new tree and its end vertices are given the component number c after incrementing the value of c by 1.

(b) If vertex f_k is in some tree $T_i; i = 1,2,3,\ldots,c$ and h_k in tree $T_j; j = 1,2,3,\ldots,c; i \neq j$, the k-th edge is used to join these two vertices and hence every vertex in T_j is now given the component number of T_i. The value of c is decremented by 1.

(c) If both vertices are in the same tree, the edge $f_k h_k$ forms a fundamental circuit and is not considered further.

(d) If the vertex f_k is in a tree T_i and the vertex h_k is not in any tree, then add the edge $f_k h_k$ is added to T_i by assigning the component number of T_i to h_k also. If the vertex f_k is not in any tree and the vertex h_k is in a tree T_j, then add the edge $f_k h_k$ is added to T_j by assigning the component number of T_j to f_k also.

S-4 : Stop if some component number is assigned to all vertices of G.

For example, consider the graph given in Figure 10.2. Consider the edge $e_1 = v_1v_2$ (where $v_1 \in F$ and $v_2 \in H$). Both vertices are not currently in any tree of G and hence by Step S-3(a) add ${}_1v_2$ to the tree T_1 and assign the component number 1 to v_1 and v_2.

Now, consider the vertex pairs $(v_1, v_3), (v_1, v_4), (v_1, v_5)$ and (v_1, v_6). For all these vertex pairs, exactly one vertex (v_1) is in the tree T_1 and as per Step-S-3(d), all these vertices can be added to the tree T_1 and assign the component number 1.

Next, for the vertex pair (v_2, v_8), only v_2 currently belongs to the tree T_1. So, by Step-S-3(d), we can add v_2v_8 to the tree T_1 and give the component number 1.

For the vertex pair (v_3, v_8), both v_3 and v_8 belong to the tree T_1 and hence by Step S-3(c), we can never consider v_3v_8 to any tree of G further.

Similarly, if we consider v_6v_{10} to the tree T_1 we cannot consider v_5v_{10} and if the edge v_8v_{12} is included in T_1, then the edge $v_{10}v_{12}$ can never be considered to any tree further.

For the vertex pairs (v_4, v_7) and (v_4, v_{11}), none of the vertices is in any tree so far and hence by Step S-3(a), they can be included to a new tree T_2 and assign the component number 2.

Now, if we consider the edge v_7v_{14} to the tree T_2, then, by Step-S3(c), the edge $v_{11}v_{14}$ cannot be taken to any tree further.

For the vertex pairs $(v_{14}, v16)$ and (v_{14}, v_{18}), only v_{14} belongs to a tree in G, and hence both edges can be added to T_2 with component number 2.

Similarly, for the vertex pairs (v_9, v_{13}) and (v_9, v_{15}), none of the vertices is in any tree so far and hence by Step S-3(a), they can be included to a new tree T_3 and assign the component number 3.

Now, if we consider the edge $v_{13}v_{17}$ to the tree T_3, then, by Step-S3(c), the edge $v_{15}v_{17}$ cannot be considered for inclusion in any tree further.

Then, the spanning forest (as G is disconnected) of the graph G is as given in Figure 10.11.

10.4 Minimal Spanning Tree Algorithms

Many optimisation problems amount to finding, in a suitable weighted graph, a certain type of subgraph with the minimum (or maximum) weight. In the following problems, we are looking for a spanning tree of G of the minimum weight. Such a tree is called a *minimal spanning tree* or an *optimal spanning tree* for G. Minimum spanning trees have many interesting applications such as the design of a network. There are several algorithms for computing the minimum spanning trees. They all, however, are based on the same underlying property about cuts in a graph, which we will refer to as the *light-edge property*. One of the most important algorithms among these is Kruskal's algorithm.

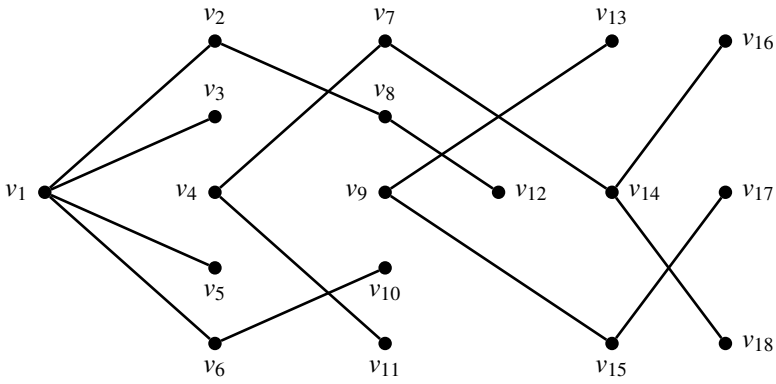


Figure 10.11: A spanning forest of a disconnected G

10.4.1 Kruskal's Algorithm

The procedure is explained in the following three steps.

(i) Choose an edge e_1, an edge of G, such that $w(e_1)$ is as small as possible and e_1 is not a loop.

(ii) If edges $e_1, e_2, \ldots, e_i$ have been chosen, then choose an edge e_{i+1}, not already, chosen, such that
   (a) the induced subgraph $G[\{e_1, \ldots, e_{i+1}\}]$ is acyclic; and
   (b) $w(e_{i+1})$ is as small as possible subject to Step-1.

(iii) If G has n vertices, stop after $n-1$ edges have been chosen. Otherwise, repeat Step 2.

A step by step illustration of Kruskal's Algorithm to find a minimal spanning tree of the graph given Figure 10.12 is given below.

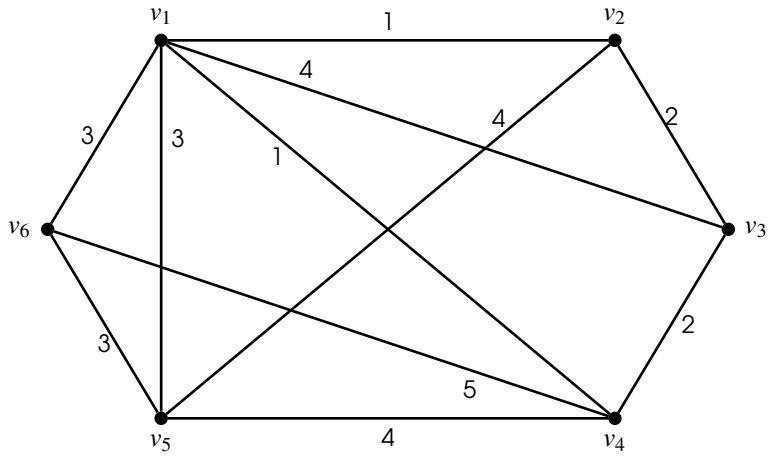


Figure 10.12: A weighted graph G

Figure 10.13 illustrates how to determine a minimal spanning tree of the weighted graph G given above using Kruskal's algorithm.

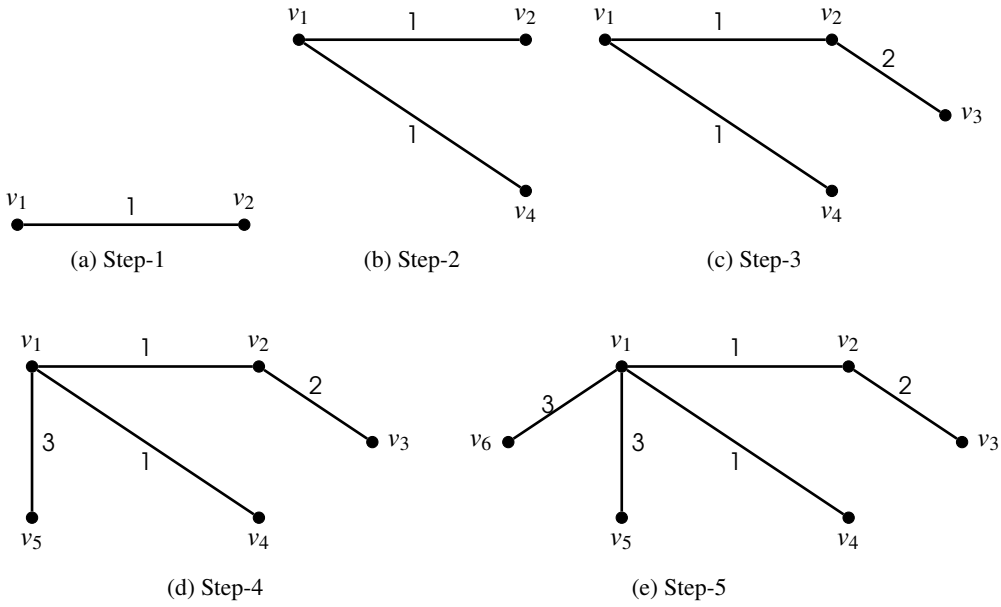


Figure 10.13: Illustration to Kruskal's Algorithm

Theorem 10.4.1 Kruskal's algorithm yields the shortest spanning tree of the graph G under consideration.

Proof. In this algorithm, we first choose an edge of G which has smallest weight among the edges of G which are not loops. Next, again avoiding loops, we choose from the remaining edges one of smallest weight possible which does not form a cycle with the edge already chosen. We repeat this process taking edges of smallest weight among those not already chosen, provided no cycle is formed with those that have been chosen. This condition ensures that we get acyclic subgraphs of G with weights as the minimum as possible. If the graph has n vertices we stop after having chosen $n-1$ edges. This is because by Theorem 6.1.7, any acyclic graph with n vertices and $n-1$ edges is connected and hence is a tree. Therefore, these $n-1$ edges form a minimal spanning tree of G. ■

10.4.2 Prim's Algorithm

This algorithm involves the following steps.

(i) Choose any vertex v_1 of G.

(ii) Choose an edge $e_1 = v_1v_2$ of G such that $v_2 \neq v_1$ and e_1 has the smallest weight among the edges of G incident with v_1.

(iii) If edges $e_1, e_2, \ldots, e_i$ have been chosen involving end points $v_1, v_2, \ldots, v_{i+1}$, choose an edge $e_{i+1} = v_jv_k$ with $v_j \in \{v_1, \ldots, v_{i+1}\}$ and $v_k \notin \{v_1, \ldots, v_{i+1}\}$ such that e_{i+1} has the smallest weight among the edges of G with precisely one end in $\{v_1, \ldots, v_{i+1}\}$.

(iv) Stop after $n-1$ edges have been chosen. Otherwise, repeat Step-3.

A step by step illustration of Prim's Algorithm to find a minimal spanning tree of the graph given Figure 10.12 is given below.

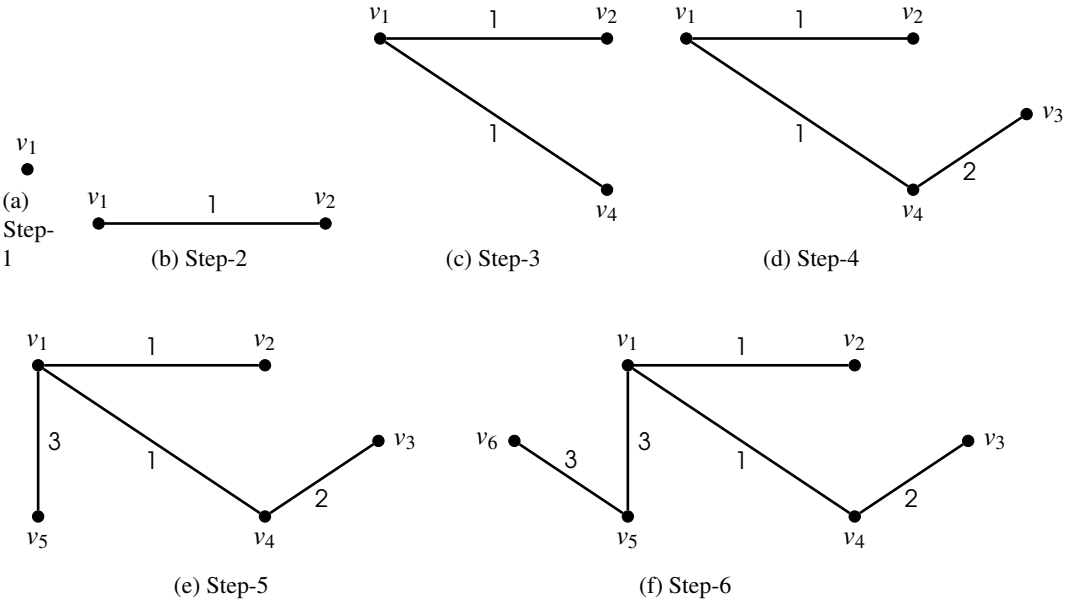


Figure 10.14: Illustration to Prim's Algorithm

Theorem 10.4.2 Prim's algorithm yields the shortest spanning tree of the graph G under consideration.

Proof. In this algorithm for finding a minimal spanning tree, we first choose a (any) vertex v_1 of the connected graph G. We next choose one of the edges incident with v_1 having the smallest weight in G which is not a loop, say $e_1 = v_1v_2$. Next, we choose an edge of smallest weight in G which is incident with either v_1 or v_2, but with the other end vertex neither of these. That is, we choose $e_2 = v_iv_3$, where $i = 1$ or $i = 2$. We repeat this process of taking edges of smallest weight one of whose ends is a vertex previously chosen and the other end becoming involved for the first time, until we have chosen $n-1$ edges (assuming the graph has n vertices). Clearly, this condition ensures that at each stage the subgraph develops satisfying the connectedness acyclic conditions and has the weight as minimum as possible. At the final stage, we have involved each of the n vertices of G and by the construction the resulting subgraph is connected, acyclic and has the weight as minimum as possible. Hence, the subgraph of G thus obtained is a minimal spanning tree. ■

1. Both algorithms have the same output. But, Kruskal's algorithm begins with forest and merge into a tree, while Prim's algorithm always stays as a tree.
2. Both algorithms have the same complexity $O(N \log N)$, as both involve sorting edges for lowest weight.

3. Kruskal's Algorithm usually has to check for the weights of the whole edges of the graph at each iteration and hence works best, if the number of edges is kept to a minimum. But, Prim's algorithm doesn't need to see the weights of all edges of the graph at once. It can deal with it one piece at a time. It also doesn't need to worry if adding an edge will create a cycle since this algorithm deals primarily with the nodes, and not the edges.
4. The running time of Kruskal's algorithm is $O(\varepsilon \log n)$, while the running time of Prim's algorithm is $O(\varepsilon + \log n)$.

10.5 Shortest Path Problems

Let G be a graph and let s,t be two specified vertices of G. In shortest spanning tree problems, we have to a path from s to t, if there is any, which uses the least number of edges. Such a path, if it exists, is called a *shortest path* from s to t.

The Breadth First Search (BFS) Technique

The Breadth First Search (BFS for short) technique can be explained as in the following algorithm.

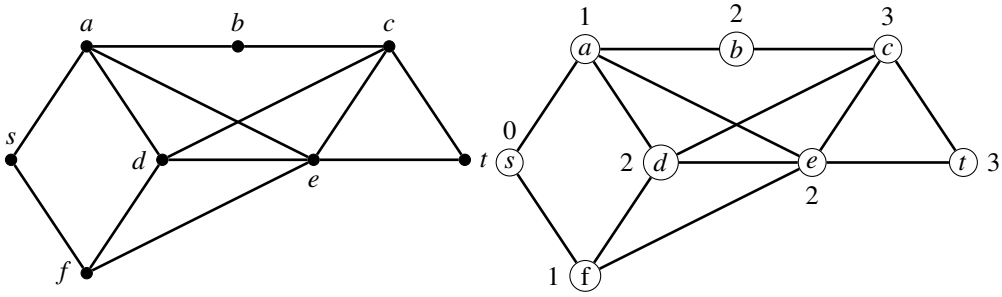
(S-1) Label vertex s with 0. Set $i = 0$.

(S-2) Find all unlabeled vertices in G which are adjacent to vertices labelled i. If there are no such vertices, then t is not connected to s by a path. If there are such vertices, label them $i+1$.

(S-3) If t is labeled, go to Step 4. If not, increase i to $i+1$ and go to Step 2.

(S-4) The length of a shortest path from s to t is $i+1$. Stop.

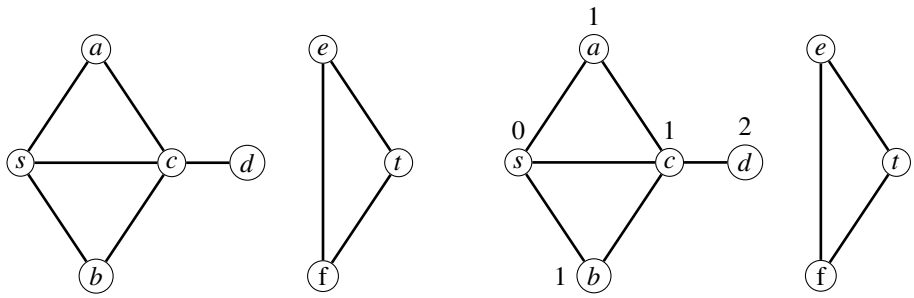
For example, for the graph of in Figure 10.15, first s is labelled 0. Then, the vertices a and f, that are adjacent to s, are labelled 1. After that, the vertices b,d and e are labelled 2. Then c and t are labelled 3. Since t is labelled 3, the length of a shortest path from s to t is 3.



(a) A Graph G (b) G shortest paths from s.

Figure 10.15: Illustration - BFS Algorithm

For the graph given in Figure **??**, first, s is labelled 0. Then a, b and c are labelled 1. Then, d is labelled 2. But now there are no unlabelled vertices adjacent to d, the only vertex labelled 2. Hence, by Step 2, it can be concluded that there is no path from s to t.



(a) A Graph G (b) G shortest paths from s.

Figure 10.16: Illustrtaion - BFS Algorithm

10.5.1 The Back-tracking Algorithm for a Shortest Path

1. Set $i = \lambda(t)$ and assign $v_i = t$.
2. Find a vertex u adjacent to v_i and with $\lambda(u) = i - 1$. Assign $v_{i-1} = u$.
3. If $i = 1$, stop. If not, decrease i to $i - 1$ and go to Step 2.

Example: In Figure 10.15 we have $\lambda(t) = 3$ and hence we start with $i = 3$ and $v_3 = t$ (Step-1). We can then choose e adjacent to $v_3 = t$ with $\lambda(e) = 2$ and assign $v_2 = e$. Next, we can choose f adjacent to $v_2 = e$ with $\lambda(f) = 1$ and assign $v_1 = 1$. Finally we take s adjacent to f with $\lambda(s) = 0$ and assign $v_0 = s$. This gives the shortest path $v_0v_1v_2v_3 = s - f - e - t$ from s to t.

10.5.2 Dijkstra's Algorithm

The *Dijkstra's algorithm* is an algorithm for finding the shortest paths between nodes in a graph, which may represent (for example, road networks). It was conceived by computer scientist *Edsger W. Dijkstra* in 1956. Dijkstra's original variant found the shortest path between two nodes, but a more common variant fixes a single node as the "source" vertex and finds shortest paths from the source to all other vertices in the graph, producing a shortest-path tree. Given a path P from vertex s to vertex t in a weighted graph G, we define the length of P to be the sum of the weights of its edges.

The steps involved in Dijkstra's Algorithm are as follows:

1. Set $\lambda(s) = 0$ and for all vertices $v \neq s$, $\lambda(v) = \infty$. Set $T = V$, the vertex set of G. (We will think of T as the set of vertices *uncoloured*.)
2. Let u be a vertex in T for which $\lambda(u)$ is the minimum.
3. If $u = t$, stop.
4. For every edge $e = uv$ incident with u, if $v \in T$ and $\lambda(v) > \lambda(u) + w(uv)$, change the value of $\lambda(v)$ to $\lambda(u) + w(uv)$ (That is, given an edge $e = uv$ from an *uncoloured* vertex v to u, change $\lambda(v)$ to $\min\{\lambda(v), \lambda(u) + w(uv)\}$.
5. Change T to $T - \{u\}$ and go to Step-2, (That is, *colour* u and then go back to Step-2 to find an *uncoloured* vertex with the minimum label).

Consider the weighted graph in Figure 10.17.

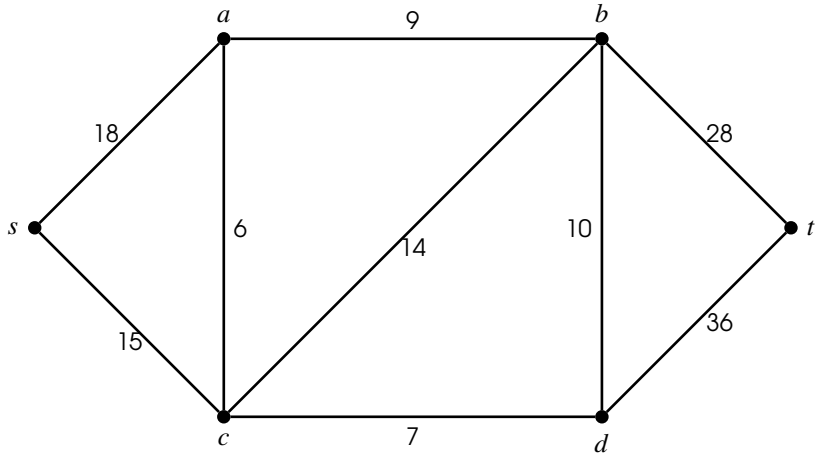


Figure 10.17: A weighted graph G

A step by step illustration of Dijkstra's Algorithm for the graph given in Figure **??** is explained as follows:

Step 1. The initial labelling is given by:

vertex v	s	a	b	c	d	t
$\lambda(v)$	0	∞	∞	∞	∞	∞
T	{s,	a,	b,	c,	d,	t}

Step 2. $u = s$ has $\lambda(u)$ a minimum (with value 0).

Step 4. There are two edges incident with u, namely sa and sc. Both a and c are in T, i.e., they are not yet coloured. $\lambda(a) = \infty > 18 = 0 + 18 = \lambda(s) + w(sa)$. Hence, $\lambda(a)$ becomes 18. Similarly, $\lambda(c)$ becomes 15.

Step 5. T becomes $T - \{s\}$, i.e., we colour s. Thus we have

vertex v	s	a	b	c	d	t
$\lambda(v)$	0	18	∞	15	∞	∞
T	{	a,	b,	c,	d,	t}

Step 2. $u = c$ has $\lambda(u)$ a minimum and u in T (with $\lambda(u) = 15$).

Step 4. There are 3 edges cv incident with $u = c$ having v in T, namely ca, cb, cd. Considering the edge ca, we have $\lambda(a) = 18 < 21 = 15 + 6 = \lambda(c) + w(ca)$ and hence $\lambda(a)$ remains as 18. Considering the edge cb, we have $\lambda(b) = \infty > 29 = 15 + 14 = \lambda(c) + w(cb)$. So, $\lambda(b)$ becomes 29. Similarly, considering the edge cd, we have $\lambda(d)$ becomes $15 + 7 = 22$.

Step 5. T becomes $T - \{c\}$, i.e., we colour c. Thus, we have

vertex v	s	a	b	c	d	t
$\lambda(v)$	0	18	29	15	22	∞
T	{	a,	b,		d,	t}

Step 2. $u = a$ has $\lambda(u)$ a minimum and for u in T (with $\lambda(u) = 18$).

Step 4. There is only one edge av incident with $u = a$ having v in T, namely ab. Then, $\lambda(b) = 29 > 27 = 18 + 9 = \lambda(a) + w(ab)$. So, $\lambda(b)$ becomes 27.

Step 5. T becomes $T - \{a\}$, i.e., we colour a. Thus, we have

vertex v	s	a	b	c	d	t
$\lambda(v)$	0	18	27	15	22	∞
T	{		b,		d,	t}

Step 2. $u = d$ has $\lambda(u)$ minimum for u in T (with $\lambda(u) = 22$).

Step 4. There are two edges dv incident with $u = d$ having v in T, namely db and dt. Considering the edge db, we have $\lambda(b) = 27 < 32 = 22 + 10 = \lambda(d) + w(db)$ and hence $\lambda(b)$ remains as 27. For the edge dt, we have $\lambda(t) = \infty > 58 = 22 + 36 = \lambda(d) + w(dt)$. Hence, $\lambda(t)$ becomes 58.

Step 5. T becomes $T - \{d\}$, i.e., we colour d. Thus, we have

vertex v	s	a	b	c	d	t
$\lambda(v)$	0	18	27	15	22	58
T	{		b,			t}

Step 2. $u = b$ has $\lambda(u)$ minimum for u in T (with $\lambda(u) = 27$).

Step 4. There is only one edge bv for v in T, namely bt. $\lambda(t) = 58 > 55 = 27 + 28 = \lambda(b) + w(bt)$ so $\lambda(t)$ becomes 55.

Step 5. T becomes $T - ab$, i.e., we colour b. Thus, we have

vertex v	s	a	b	c	d	t
$\lambda(v)$	0	18	27	15	22	55
T	{					t}

Step 2. $u = t$, the only choice. Step 3. Stop.

All steps, we did above can be written in a single table as follows:

Steps	s	a	b	c	d	t	T					
1	0	∞	∞	∞	∞	∞	s	a	b	c	d	t
2	0	18	∞	15	∞	∞		a	b	c	d	t
3	0	18	29	15	22	∞		a	b		d	t
4	0	18	27	15	22	∞			b		d	t
5	0	18	27	15	22	58			b			t
6	0	18	27	15	22	55						t

Note that Dijkstra's algorithm does not work with negative edges. The problem is that when we consider the closest vertex v not in the visited set, its shortest path may be through only the visited set and then extended by one edge out of the visited set to v.

Theorem 10.5.1 For any graph G, Dijkstra's algorithm computes the distance (weight) of shortest paths from a fixed vertex s to all other vertices in G.

10.5.3 Floyd-Warshall Algorithm

Notations in the algorithm:

1. Assume that the graph under consideration is represented by an $n \times n$ matrix $[w_{ij}]$ defined by
$$w_{ij} = \begin{cases} 0, & \text{if } v_i = v_j; \\ w(v_i v_j), & \text{if } v_i \neq v_j; \\ \infty, & \text{if } v_i v_j \notin E;. \end{cases}$$
2. D is the matrix defined by $D = [d_{ij}]$, where d_{ij} is the distance between the vertices v_i and v_j.
3. Let $d_{ij}^{(k)}$ be the length of the shortest path from v_i to v_j such that all intermediate vertices on the path (if any) are in set $\{v_1, v_2, \ldots, v_k\}$. Then, $d_{ij}^{(0)} = w_{ij}$, that is, no intermediate vertex between v_i and v_j. Also, let $D^{(k)} = [d_{ij}^{(k)}]$, where $1 \leq k \leq n$.

Floyd-Warshall Algorithm:

S-1 : Initailly set $D \leftarrow D^{(0)}$. Set $i = 1, j = 1$ and set $k = 1$.

S-2 : Let $D^{(k)} = [d_{ij}^{(k)}]$ such that

(a) if v_k is not on a (the) shortest $v_i - v_j$ path, then it has length $d_{ij}^{(k-1)}$.

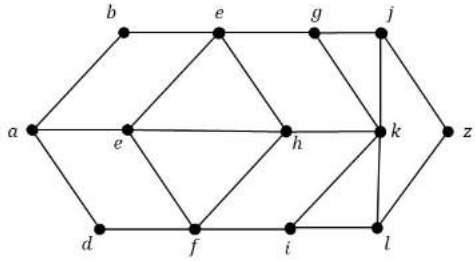
(b) if v_k is on a path, then it consists of a subpath from v_i to v_k and a subpath from v_k to v_j and must be as short as possible, namely they have lengths $d_{ik}^{(k-1)}$ and $d_{kj}^{(k-1)}$. Then, the shortest $v_i - v_j$ path has length $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.

Combining the two cases we get, $d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$.

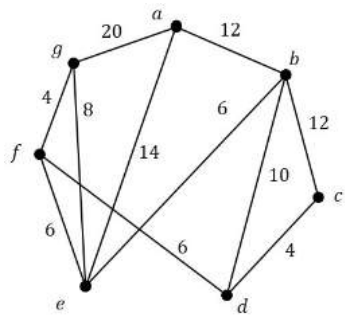
S-3 : If $D = D^{(n)}$, then print D. Stop.

10.6 Exercises

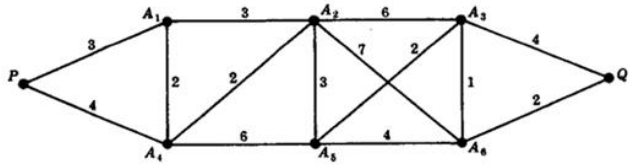
1. Carry out the BFS algorithm for the following graph to find the length of a shortest path from vertex a to vertex z, an example of such a shortest path, and the number of shortest paths from a to z.

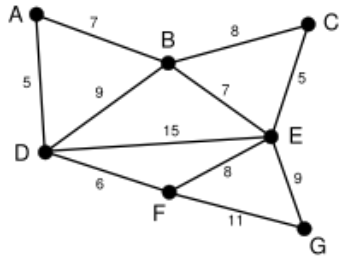


2. Use Dijkstra's algorithm on the connected weighted graphs of the following figure to find the length of the shortest paths from the vertex a to each of the other vertices and to give examples of such paths.

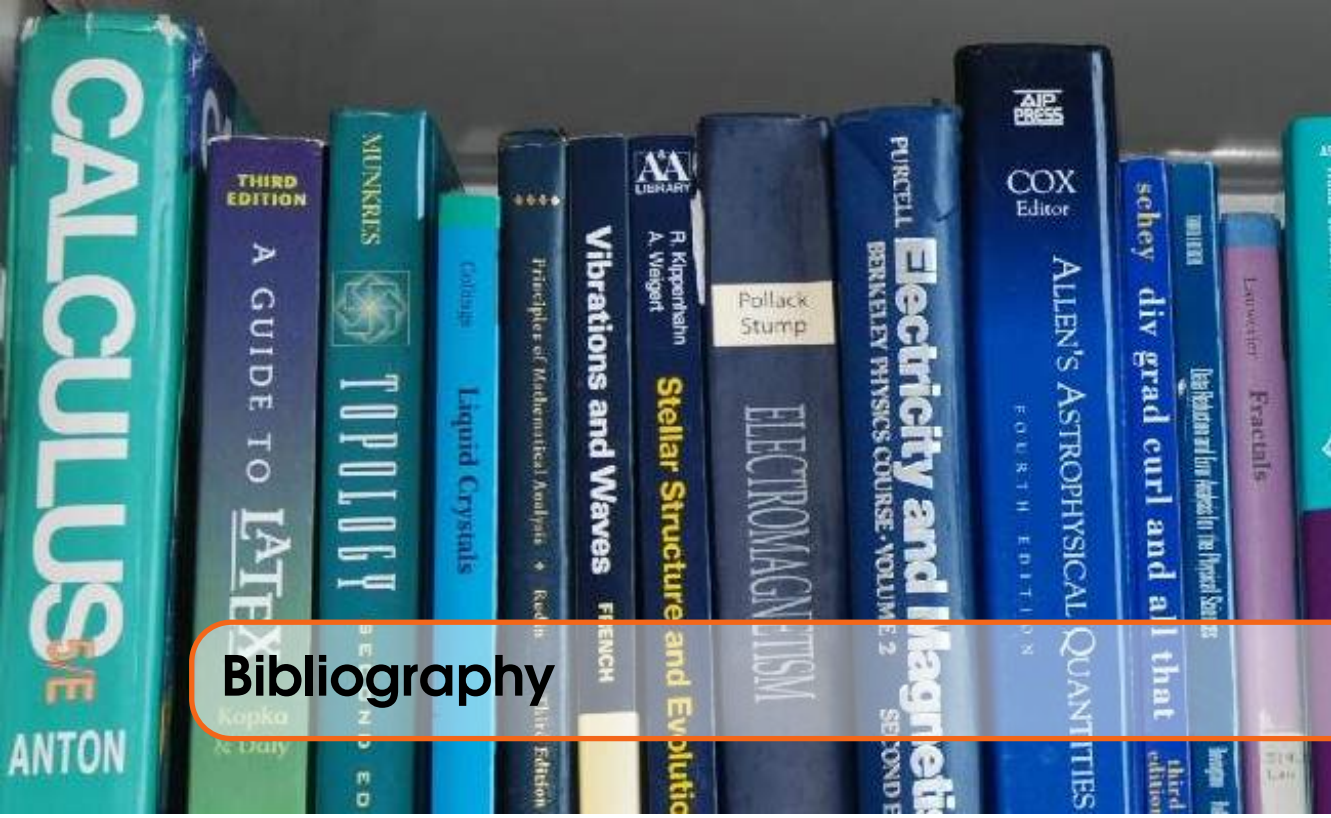


3. Use Kruskal's algorithm and Prim's algorithm to find minimal spanning trees of the following graphs:





REFERENCES



Bibliography

[1] G. Agnarsson and R. Greenlaw, (2007). *Graph theory: Modeling, applications & algorithms*, Pearson Education, New Delhi.

[2] S. Arumugam and S. Ramachandran, (2015), *Invitation to graph theory*, Scitech Publ., Kolkata, India.

[3] R. Balakrishnan and K. Ranganathan, (2012). *A textbook of graph theory*, Springer, New York.

[4] V.K. Balakrishnan, (1997). *Graph theory*, McGrawhill, New York.

[5] A. Benjamin, G. Chartrant and P. Zhang, (2015). *The fascinating world of graph theory*, Princeton Uty. Press, New Jersey.

[6] C. Berge, (1979). *Graphs and hypergraphs*, North Holland, Amsterdam.

[7] C. Berge, (2001). *The theory of graphs*, Dover Pub.

[8] B. Bollabás, (1998). *Modern graph theory*, Springer Int. Edn., New York.

[9] J.A. Bondy and U.S.R Murty, (2008). *Graph theory*, Springer.

[10] J.A. Bondy and U.S.R Murty, (1976). *Graph theory with applications*, North-Holland, New York.

[11] A. Brandstädt, V. B. Le and J. P. Spinrad, (1999). *Graph classes: A survey*, SIAM, Philadelphia.

[12] F.Buckley and F.Harary, (1990). *Distances in graphs*, Addison-Wesley Publishing Company.

[13] G. Chartrand and P. Zhang, (2005). *Introduction to graph theory*, McGraw-Hill Inc.

[14] G. Chartrand and L. Lesniak, (1996). *Graphs and digraphs*, CRC Press.

[15] J. Clark and D.A. Holton, (1995). *A first look at graph theory*, Allied Publishers Ltd., Mumbai, India.

[16] N. Deo, (2016). *Graph Theory with application to engineering and computer science*, Prentice Hall of India Pvt. Ltd., India.

[17] R. Diestel, (2010). *Graph theory*, Springer-Verlag, New York.

[18] J.C. Fournier, (2009). *Graph theory with applications*, John Wiley & Sons, Inc., New York.

[19] M.C. Golumbic and I.B.A. Hartman, (2005). *Graph theory, combinatorics and algorithms*, Springer.

[20] R.P. Grimaldi, (2014). *Discrete and combinatorial mathematics*, Pearson Education., New Delhi.

[21] J. Gross and J. Yellen, (1999). *Graph theory and its applications*, CRC Press.

[22] J. Gross and J. Yellen, (2004). *Handbook of graph theory*, CRC Press.

[23] F. Harary, (2001). *Graph theory*, Narosa Publ. House, New Delhi.

[24] F. Harary and E. M. Palmer, (1973). *Graphical enumeration*, Academic Press Inc.,

[25] J. B. Jensen and G. Gutin, (2007). *Digraphs: Theory, algorithms and applications*, Springer- Verlag.

[26] K. D. Joshi, (2003). *Applied discrete structures*, New Age International, New Delhi.

[27] T.A. McKee and F.R. McMorris, (1999). *Topics in intersection graph theory*, SIAM Monographs on Discrete Mathematics and Application, SIAM, Philadelphia.

[28] O. Ore, *Theory of Graphs*, (1962). American Math. Soc. Colloquium Pub., XXXV III.

[29] M.S. Rahman, (2017). *Basic graph theoy*, Springer, Switzerland.

[30] F.S. Roberts, (1978). *Graph theory and its practical applications to problems of society*, SIAM, Philadelphia.

[31] K. Rosen, (2011). *Discrete mathematics and its applications*, Tata McGraw Hill.

[32] R.J. Trudeau, (1993), *Introduction to graph theory*, Dover Publ., New York.

[33] S.S. Sarkar (2003). *A text book of discrete mathematics*, S. Chand & Co., New Delhi.

[34] G. Sethuraman, R. Balakrishnan, and R.J. Wilson, (2004). *Graph theory and its applications*, Narosa Pub. House, New Delhi.

[35] G. S. Singh, (2013). *Graph theory*, Prentice Hall of India, New Delhi.

[36] R. R. Stoll, (1979). *Set theory and Logic*, Dover pub., New York

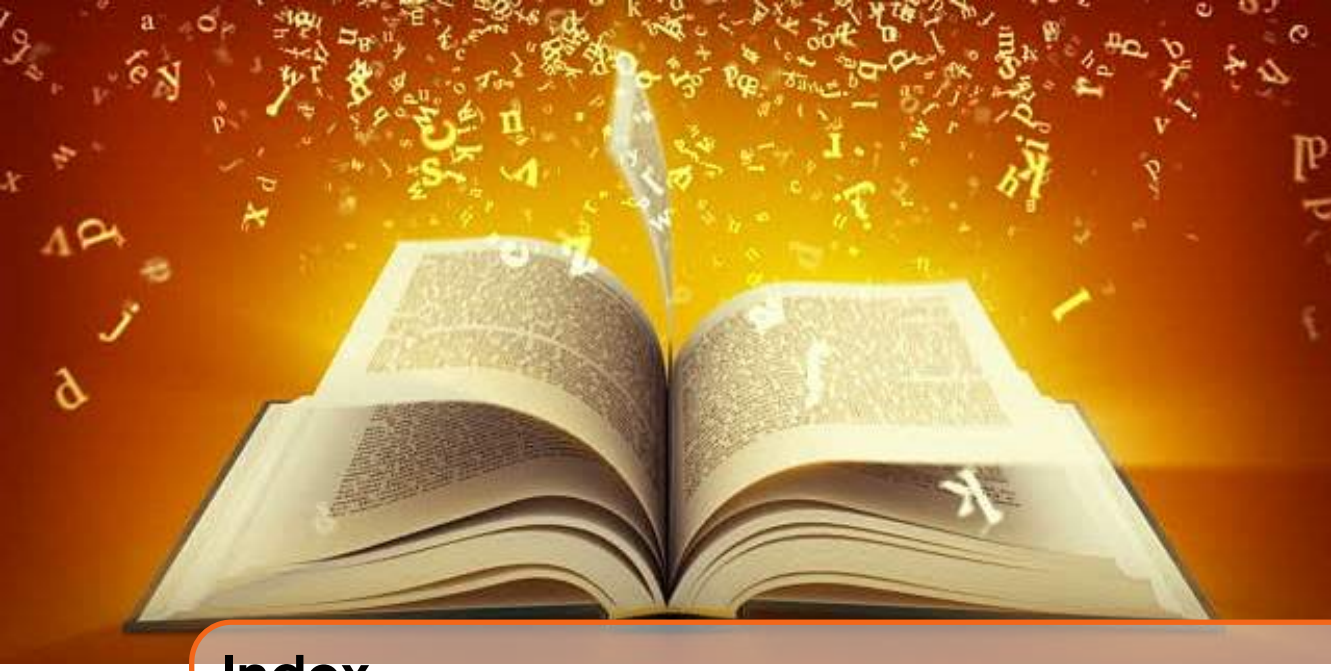
[37] D.B. West, (2001). *Introduction to graph theory*, Pearson Education Inc., Delhi.

LIST OF FIGURES & TABLES

List of Figures

List of Tables

INDEX



Index

ABOUT THE AUTHOR

The author, an authentic researcher in Graph Theory & Discrete Mathematics, has been an Associate Professor in the Department of Mathematics, Vidya Academy of Science & Technology,Thrissur, India, for around fifteen years. He completed his doctoral degree from Kannur University, Kerala, India.

Dr. Sudev has been writing reviews for the prestigious databases like Mathematical Reviews, Zb MATH, Computing Reviews and MAA Reviews. He is in the editorial board of more than twenty International Research Journals and a member of the reviewer panel of about forty journals.

Author's Website: http:// www.sudevnaduvath.in

The Centre for Studies in Discrete Mathematics (CSDM)

The centre for studies in Discrete Mathematics (CSDM) is a noble initiative of the Department of Applied Sciences, Vidya Academy of Science and Technology, Thrissur, India, to encourage quality research in Discrete Mathematics and allied areas in collaboration with other reputed instituitions and centers of excellence. The centre is envisaged to act as an avenue for our teachers to interact with experts of national and international repute in the specific subject areas concerned. The centre works closely with people associated with teaching and research in related disciplines like computer science and electronic engineering. (http://www.csdm.org.in)

ISBN: 978-93-5291-147-9