COMBINATORICS & GRAPH THEORY

MCA 206

SELF LEARNING MATERIAL

DIRECTORATE

OF DISTANCE EDUCATION

SWAMI VIVEKANAND SUBHARTI UNIVERSITY

MEERUT – 250 005,

UTTAR PRADESH (INDIA)

SLM Module Developed By :

Author:

Reviewed by :

Assessed by:

Study Material Assessment Committee, as per the SVSU ordinance No. VI (2)

Published by: Gayatri Sales

Typeset at: Micron Computers **Printed at:** Gayatri Sales, Meerut.

COMBINATORICS & GRAPH THEORY (MCA-206)

Unit 1

Rules of sum and products, Permutation, Combination, Permutation groups and application, Probability,Ramsey theory, Discrete numeric function and generating function, Combinatorial problems, Difference equation.

Unit II

Recurrence Relation-Introduction, Linear recurrence relation with constant coefficient, Homogeneous solution, Particular solution, Total solution, Solution by the method of generating function.

Unit III

Graphs, sub-graphs, some basic properties, Walks, Path & circuits, Connected graphs, Disconnected graphs and component, Eular and Hamiltonian graphs, Various operation on graphs, Tree and fundamental circuits, Distance diameters, Radius and pendent vertices, Rooted and binary trees, Counting trees, Spanning trees, Finding all spanning trees of a graph and a weighted graph.

Unit IV

Cut-sets and cut vertices, some properties, All cut sets in a graph, Fundamental circuit and cut sets, Connectivity and seperatability, Network flows, mincut theorem, Planar graphs, Combinatorial and geometric dual, Kuratowski to graph detection of planarity, Geometric dual, Some more criterion of planarity, Thickness and Crossings, Vector space of a graph and vectors, basis vectors, cut set vector, circuit vector, circuit and cut set verses sub spaces, orthogonal vector and sub space. Incidence matrix of graphs, sub matrices of A(G), circuit matrix, cut set matrix, path matrix and relationship among Af, Bf, Cf, fundamental circuit matrix and range of Bf adjacency matrix, rank nullity

theorem.

Unit V

Coloring and covering partitioning of graph, Chromatic number, Chromatic partitioning, Chromatic polynomials, Matching, covering, Four color problem, Directed graph, Types of directed graphs, Directed paths and connectedness, Euler digraph, Trees with directed edges, Fundamental circuit in digraph, Matrices A, B, C of digraph adjacency matrix of digraph, Enumeration and its types, Counting of labeled and unlabeled trees, Polya’s theorem, Graph enumeration with polyas theorem, Graph theoretic algorithm.

Unit I

Rules of sum and products

The sum and product rules of probability refer to methods of figuring out the probability of two events, given the probabilities of each event. The sum rule is for finding the probability of either of two events that cannot occur simultaneously. The product rule is for finding the probability of both of two events that are independent.

Explaining the Sum Rule

Write the sum rule and explain it in words. The sum rule is given by P(A + B) = P(A) + P(B). Explain that A and B are each events that could occur, but cannot occur at the same time.

Give examples of events that cannot occur simultaneously and show how the rule works. One example: The probability that the next person walking into class will be a student and the probability that the next person will be a teacher. If the probability of the person being a student is 0.8 and the probability of the person being a teacher is 0.1, then the probability of the person being either a teacher or student is 0.8 + 0.1 = 0.9.

Give examples of events that can occur at the same time, and show how the rule fails. One example: The probability that the next flip of a coin is heads or that the next person walking into the class is a student. If the probability of heads is 0.5 and the probability of the next person being a student is 0.8, then the sum is 0.5 + 0.8 = 1.3; but probabilities must all be between 0 and 1.

Product Rule

Write the rule and explain the meaning. The product rule is P(E_F) = P(E)_P(F) where E and F are events that are independent. Explain that independence means that one event occurring has no effect on the probability of the other event occurring.

Give examples of how the rule works when events are independent. One example: When picking cards from a deck of 52 cards, the probability of getting an ace is 4/52 = 1/13, because there are 4 aces among the 52 cards (this should have been explained in an earlier lesson). The probability of picking a heart is 13/52 = 1/4. The probability of picking the ace of hearts is 1/4*1/13 =1/52.

Give examples where the rule fails because the events are not independent. One example: The probability of picking an ace is 1/13, the probability of picking a two is also 1/13. But the probability of picking an ace and a two in the same card is not 1/13*1/13, it is 0, because the events are not independent.

In mathematics, a permutation of a set is, loosely speaking, an arrangement of its members into a sequence or linear order, or if the set is already ordered, a rearrangement of its elements. The word "permutation" also refers to the act or process of changing the linear order of an ordered set.[1]

Permutations differ from combinations, which are selections of some members of a set regardless of order. For example, written as tuples, there are six permutations of the set {1,2,3}, namely: (1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), and (3,2,1). These are all the possible orderings of this three-element set. Anagrams of words whose letters are different are also permutations: the letters are already ordered in the original word, and the anagram is a reordering of the letters. The study of permutations of finite sets is an important topic in the fields of combinatorics and group theory.

Permutations are used in almost every branch of mathematics, and in many other fields of science. In computer science, they are used for analyzing sorting algorithms; in quantum physics, for describing states of particles; and in biology, for describing RNA sequences.

The number of permutations of n distinct objects is n factorial, usually written as n!, which means the product of all positive integers less than or equal to n.

Technically, a permutation of a set S is defined as a bijection from S to itself.[2][3] That is, it is a function from S to S for which every element occurs exactly once as an image value. This is related to the rearrangement of the elements of S in which each element s is replaced by the corresponding f(s). For example, the permutation (3,1,2) mentioned above is described by the function defined as:

> The collection of all permutations of a set form a group called the symmetric group of the set. The group operation is the composition (performing two given rearrangements in succession), which results in another rearrangement. As properties of permutations do not depend on the nature of the set elements, it is often the permutations of the set that are considered for studying permutations.

In elementary combinatorics, the k-permutations, or partial permutations, are the ordered arrangements of k distinct elements selected from a set. When k is equal to the size of the set, these are the permutations of the set.

Permutation

A permutation, also called an "arrangement number" or "order," is a rearrangement of the elements of an ordered list S into a one-to-one correspondence with S itself. The number of permutations on a set of n elements is given by $n!$ (n factorial; Uspensky 1937, p. 18). For example, there are $2! = 2 \cdot 1 = 2$ permutations of $\{1, 2\}$, namely $\{1, 2\}$ and $\{2, 1\}$, and $3! = 3 \cdot 2 \cdot 1 = 6$ permutations of $\{1, 2, 3\}$, namely $\{1, 2, 3\}$, $\{1, 3, 2\}$, $\{2, 1, 3\}$, $\{2, 3, 1\}$, $\{3, 1, 2\}$, and $\{3, 2, 1\}$. The permutations of a list can be found in the Wolfram Language using the command Permutations[list]. A list of length n can be tested to see if it is a permutation of 1, ..., n in the Wolfram Language using the command PermutationListQ[list].

Sedgewick (1977) summarizes a number of algorithms for generating permutations, and identifies the minimum change permutation algorithm of Heap (1963) to be generally the fastest (Skiena 1990, p. 10). Another method of enumerating permutations was given by Johnson (1963; Séroul 2000, pp. 213-218).

The number of ways of obtaining an ordered subset of k elements from a set of n elements is given by

$$_nP_k \equiv \frac{n!}{(n-k)!}$$

$\{1, 2\}$, $\{1, 3\}$, $\{1, 4\}$, $\{2, 1\}$, $\{2, 3\}$, $\{2, 4\}$, $\{3, 1\}$, $\{3, 2\}$, $\{3, 4\}$, $\{4, 1\}$, $\{4, 2\}$, and $\{4, 3\}$.
The unordered subsets containing k elements are known as the k-subsets of a given set.

A representation of a permutation as a product of permutation cycles is unique (up to the ordering of the cycles). An example of a cyclic decomposition is the permutation $\{4, 2, 1, 3\}$ of $\{1, 2, 3, 4\}$. This is denoted $(2)(143)$, corresponding to the disjoint permutation cycles (2) and (143). There is a great deal of freedom in picking the representation of a cyclic decomposition since (1) the cycles are disjoint and can therefore be specified in any order, and (2) any rotation of a given cycle specifies the same cycle (Skiena 1990, p. 20). Therefore, (431)(2), (314)(2), (143)(2), (2)(431), (2)(314), and (2)(143) all describe the same permutation.

Another notation that explicitly identifies the positions occupied by elements before and after application of a permutation on n elements uses a $2 \times n$ matrix, where the first row is $(123 \ldots n)$ and the second row is the new arrangement. For example, the permutation which switches elements 1 and 2 and fixes 3 would be written as

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{bmatrix}.$$

Any permutation is also a product of transpositions. Permutations are commonly denoted in lexicographic or transposition order. There is a correspondence between a permutation and a pair of Young tableaux known as the Schensted correspondence.

The number of wrong permutations of n objects is $[n!/e]$ where $[x]$ is the nearest integer function. A permutation of n ordered objects in which no object is in its natural place is called a derangement (or sometimes, a complete permutation) and the number of such permutations is given by the subfactorial $!n$.

Using

$$(x+y)^n = \sum_{r=0}^{n} \binom{n}{r} x^{n-r} y^r$$

with $x = y = 1$ gives

$$2^n = \sum_{r=0}^{n} \binom{n}{r},$$

so the number of ways of choosing 0, 1, ..., or n at a time is 2^n.

The set of all permutations of a set of elements 1, ..., n can be obtained using the following recursive procedure

```
  1 2
  /
2 1
```

```
  1   2 3
      /
  1 3 2
  /
3 1   2
|
3 2   1
  \
  2 3 1
      \
  2   1 3
```

Consider permutations in which no pair of consecutive elements (i.e., rising or falling successions) occur. For $n = 1$, 2, ... elements, the numbers of such permutations are 1, 0, 0, 2, 14, 90, 646, 5242, 47622, ... (OEIS A002464).

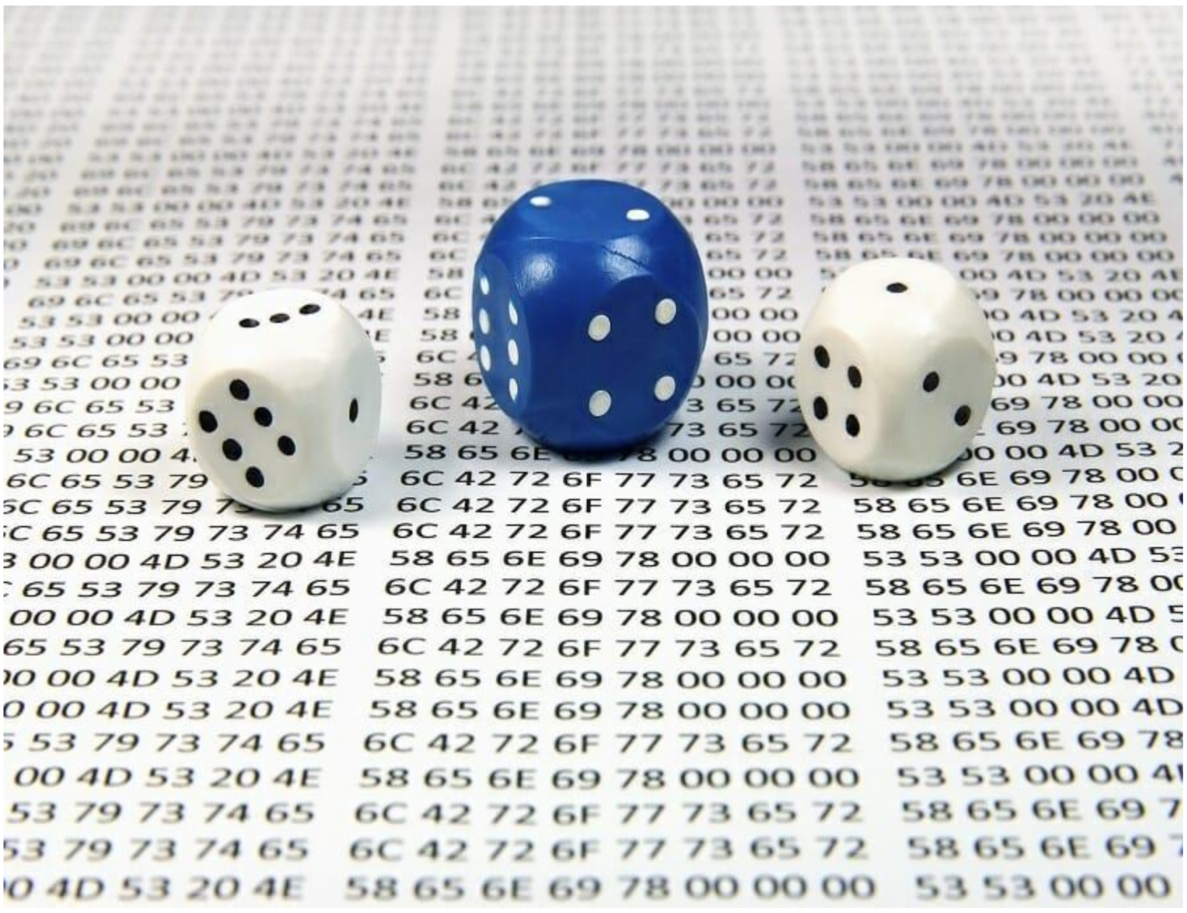
Let the set of integers 1, 2, ..., N be permuted and the resulting sequence be divided into increasing runs. Denote the average length of the nth run as N approaches infinity, L_n. The first few values are summarized in the following table, where e is the base of the natural logarithm (Le Lionnais 1983, pp. 41-42; Knuth 1998).

n	L_n	OEIS	approximate
1	$e-1$	A091131	1.7182818...
2	e^2-2e	A091132	1.9524...
3	$e^3-3e^2+\frac{3}{2}e$	A091133	1.9957...

Combination
The number of

Combination

A combination is a mathematical technique that determines the number of possible arrangements in a collection of items where the order of the selection does not matter. In combinations, you can select the items in any order.



Combinations can be confused with permutations. However, in permutations, the order of the selected items is essential. For example, the arrangements ab and ba are equal in combinations (considered as one arrangement), while in permutations, the arrangements are different.

Combinations are studied in combinatorics but are also used in different disciplines, including mathematics and finance.

Formula for Combination

Mathematically, the formula for determining the number of possible arrangements by selecting only a few objects from a set with no repetition is expressed in the following way:

$$C(n,k) = \binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

Where:

- n – the total number of elements in a set
- k – the number of selected objects (the order of the objects is not important)
- ! – factorial

Factorial (noted as “!”) is a product of all positive integers less or equal to the number preceding the factorial sign. For example, 3! = 1 x 2 x 3 = 6.

Note that the formula above can be used only when the objects from a set are selected without repetition.

Example of Combination

You are a portfolio manager in a small hedge fund. You’ve decided to create a new fund that will attract risk-taking investors. The fund will include stocks of fast-growing companies that offer high growth potential. Your team of analysts identified the stocks of 20 companies that suit the profile.

Since it is a new fund, you’ve decided to include five stocks with equal weight in the initial portfolio, and after one year, you will review the performance of the portfolio and add new stocks if the fund’s performance is successful. Currently, you want to identify the number of possible portfolios you can create from the stocks identified by your analysts.

The investment decision-making is an example of a combination problem. Since you are going to develop a portfolio in which all stocks will be of equal weights, the order of the selected stocks does not influence the portfolio. For example, the portfolio ABC and CBA would be equal to each other because of the similar weights (33.3% each) of each stock.

Permutation groups and application

Being a subgroup of a symmetric group, all that is necessary for a set of permutations to satisfy the group axioms and be a permutation group is that it contain the identity permutation, the inverse permutation of each permutation it contains, and be closed under composition of its permutations.[2] A general property of finite groups implies that a finite nonempty subset of a symmetric group is again a group if and only if it is closed under the group operation.[3]

The degree of a group of permutations of a finite set is the number of elements in the set. The order of a group (of any type) is the number of elements (cardinality) in the group. By Lagrange's theorem, the order of any finite permutation group of degree n must divide n! since n-factorial is the order of the symmetric group Sn.

Notation

Since permutations are bijections of a set, they can be represented by Cauchy's two-line notation.[4] This notation lists each of the elements of M in the first row, and for each element, its image under the permutation below it in the second row. If is a permutation of the set then,

For instance, a particular permutation of the set {1,2,3,4,5} can be written as:

this means that σ satisfies σ(1)=2, σ(2)=5, σ(3)=4, σ(4)=3, and σ(5)=1. The elements of M need not appear in any special order in the first row. This permutation could also be written as:

Permutations are also often written in cyclic notation (cyclic form)[5] so that given the set M = {1,2,3,4}, a permutation g of M with g(1) = 2, g(2) = 4, g(4) = 1 and g(3) = 3 will be written as (1,2,4)(3), or more commonly, (1,2,4) since 3 is left unchanged; if the objects are denoted by single letters or digits, commas and spaces can also be dispensed with, and we have a notation such as (124). The permutation written

above in 2-line notation would be written in cyclic notation as

Composition of permutations–the group product

The product of two permutations is defined as their composition as functions, so is the function that maps any element x of the set to . Note that the rightmost permutation is applied to the argument first,[6][7] because of the way function application is written. Some authors prefer the leftmost factor acting first, [8] [9] [10] but to that end permutations must be written to the right of their argument, often as a superscript, so the permutation acting on the element results in the image . With this convention, the product is given by . However, this gives a different rule for multiplying permutations. This convention is commonly used in the permutation group literature, but this article uses the convention where the rightmost permutation is applied first.

Since the composition of two bijections always gives another bijection, the product of two permutations is again a permutation. In two-line notation, the product of two permutations is obtained by rearranging the columns of the second (leftmost) permutation so that its first row is identical with the second row of the first (rightmost) permutation. The product can then be written as the first row of the first permutation over the second row of the modified second permutation. For example, given the permutations, the product QP is:

The composition of permutations, when they are written in cyclic form, is obtained by juxtaposing the two permutations (with the second one written on the left) and then simplifying to a disjoint cycle form if desired. Thus, in cyclic notation the above product would be given by:

Since function composition is associative, so is the product operation on permutations: . Therefore, products of two or more permutations are usually written without adding parentheses to express grouping; they are also usually written without a dot or other sign to indicate multiplication (the dots of the previous example were added for emphasis, so would simply be written as

Neutral element and inverses

The identity permutation, which maps every element of the set to itself, is the neutral element for this product. In two-line notation, the identity is

In cyclic notation, e = (1)(2)(3)...(n) which by convention is also denoted by just (1) or even ().[11]

Since bijections have inverses, so do permutations, and the inverse σ^{-1} of σ is again a permutation. Explicitly, whenever σ(x)=y one also has $\sigma^{-1}(y)=x$. In two-line notation the inverse can be obtained by interchanging the two lines (and sorting the columns if one wishes the first line to be in a given order). For instance

To obtain the inverse of a single cycle, we reverse the order of its elements. Thus,

To obtain the inverse of a product of cycles, we first reverse the order of the cycles, and then we take the inverse of each as above. Thus,

Having an associative product, an identity element, and inverses for all its elements, makes the set of all permutations of M into a group, Sym(M); a permutation group.

Examples

Consider the following set G1 of permutations of the set M = {1, 2, 3, 4}:

e = (1)(2)(3)(4) = (1)

This is the identity, the trivial permutation which fixes each element.

a = (1 2)(3)(4) = (1 2)

This permutation interchanges 1 and 2, and fixes 3 and 4.

b = (1)(2)(3 4) = (3 4)

Like the previous one, but exchanging 3 and 4, and fixing the others.

ab = (1 2)(3 4)

This permutation, which is the composition of the previous two, exchanges simultaneously 1 with 2, and 3 with 4.

G1 forms a group, since aa = bb = e, ba = ab, and abab = e. This permutation group is isomorphic, as an abstract group, to the Klein group V4.

As another example consider the group of symmetries of a square. Let the vertices of a square be labeled 1, 2, 3 and 4 (counterclockwise around the square starting with 1 in the top left corner). The symmetries are determined by the images of the vertices, that can, in turn, be described by permutations. The rotation by 90° (counterclockwise) about the center of the square is described by the permutation (1234). The 180° and 270° rotations are given by (13)(24) and (1432), respectively. The reflection about the horizontal line through the center is given by (12)(34) and the corresponding vertical line reflection is (14)(23). The reflection about the 1,3−diagonal line is (24) and reflection about the 2,4−diagonal is (13). The only remaining symmetry is the identity (1)(2)(3)(4). This permutation group is abstractly known as the dihedral group of order 8.

Group actions

In the above example of the symmetry group of a square, the permutations "describe" the movement of the vertices of the square induced by the group of symmetries. It is common to say that these group elements are "acting" on the set of vertices of the square. This idea can be made precise by formally defining a group action.[12]

Let G be a group and M a nonempty set. An action of G on M is a function f: G × M → M such that

f(1, x) = x, for all x in M (1 is the identity (neutral) element of the group G), and

f(g, f(h, x)) = f(gh, x), for all g,h in G and all x in M.

This last condition can also be expressed as saying that the action induces a group homomorphism from G into Sym(M).[12] Any such homomorphism is called a (permutation) representation of G on M.

For any permutation group, the action that sends (g, x) → g(x) is called the natural action of G on M. This is the action that is assumed unless otherwise indicated.[12] In the example of the symmetry group of the square, the group's action on the set of vertices is the natural action. However, this group also induces an action on the set of four triangles in the square, which are: t1 = 234, t2 = 134, t3 = 124 and t4 = 123. It also acts on the two diagonals: d1 = 13 and d2 = 24.

Group element	Action on triangles	Action on diagonals
(1)	(1)	(1)
(1234)	(t1 t2 t3 t4)	(d1 d2)
(13)(24)	(t1 t3)(t2 t4)	(1)
(1432)	(t1 t4 t3 t2)	(d1 d2)
(12)(34)	(t1 t2)(t3 t4)	(d1 d2)
(14)(23)	(t1 t4)(t2 t3)	(d1 d2)
(13)	(t1 t3)	(1)
(24)	(t2 t4)	(1)

Transitive actions

The action of a group G on a set M is said to be transitive if, for every two elements s, t of M, there is some group element g such that g(s) = t. Equivalently, the set M forms a single orbit under the action of G.[13] Of the examples above, the group {e, (1 2), (3 4), (1 2)(3 4)} of permutations of {1, 2, 3, 4} is not transitive (no group element takes 1 to 3) but the group of symmetries of a square is transitive on the vertices.

Primitive actions

A permutation group G acting transitively on a non-empty finite set M is imprimitive if there is some nontrivial set partition of M that is preserved by the action of G, where "nontrivial" means that the partition isn't the partition into singleton sets nor the partition with only one part. Otherwise, if G is transitive but does not preserve any nontrivial partition of M, the group G is primitive.

For example, the group of symmetries of a square is primitive on the vertices: if they are numbered 1, 2, 3, 4 in cyclic order, then the partition {{1, 3}, {2, 4}} into opposite pairs is preserved by every group element. On the other hand, the full symmetric group on a set M is always imprimitive.

Cayley's theorem

Any group G can act on itself (the elements of the group being thought of as the set M) in many ways. In particular, there is a regular action given by (left) multiplication in the group. That is, f(g, x) = gx for all g and x in G. For each fixed g, the function fg(x) = gx is a bijection on G and therefore a permutation of the set of elements of G. Each element of G can be thought of as a permutation in this way and so G is isomorphic to a permutation group; this is the content of Cayley's theorem.

For example, consider the group G1 acting on the set {1, 2, 3, 4} given above. Let the elements of this group be denoted by e, a, b and c = ab = ba. The action of G1 on itself described in Cayley's theorem gives the following permutation representation:

fe ↦ (e)(a)(b)(c)
fa ↦ (ea)(bc)
fb ↦ (eb)(ac)
fc ↦ (ec)(ab).

Isomorphisms of permutation groups

If G and H are two permutation groups on sets X and Y with actions f1 and f2 respectively, then we say that G and H are permutation isomorphic (or isomorphic as permutation groups) if there exists a bijective map λ : X → Y and a group isomorphism ψ : G → H such that

λ(f1(g, x)) = f2(ψ(g), λ(x)) for all g in G and x in X.[14]

If X = Y this is equivalent to G and H being conjugate as subgroups of Sym(X).[15] The special case where G = H and ψ is the identity map gives rise to the concept of equivalent actions of a group.[16]

In the example of the symmetries of a square given above, the natural action on the set {1,2,3,4} is equivalent to the action on the triangles. The bijection λ between the sets is given by i ↦ ti. Thenatural action of group G1 above and its action on itself (via left multiplication) are not equivalent as the natural action has fixed points and the second action does not.

Oligomorphic groups

When a group G acts on a set S, the action may be extended naturally to the Cartesian product Sn of S, consisting of n-tuples of elements of S: the action of an element g on the n-tuple (s1, ..., sn) is given by

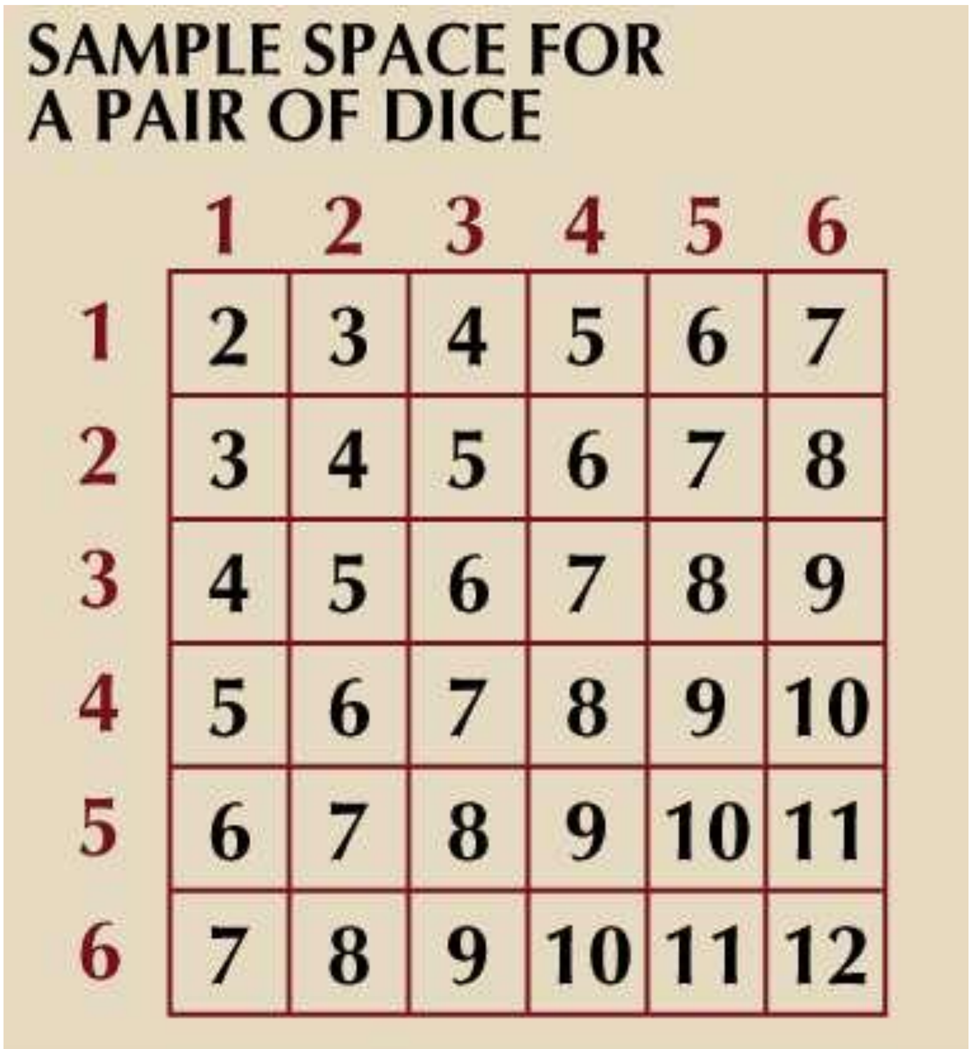
g(s1, ..., sn) = (g(s1), ..., g(sn)).

The group G is said to be oligomorphic if the action on Sn has only finitely many orbits for every positive integer n.[17][18] (This is automatic if S is finite, so the term is typically of interest when S is infinite.)

The interest in oligomorphicgroups is partly based on their application to model theory, for example when considering automorphisms in countably categorical theories.

Probability

The fundamental ingredient of probability theory is an experiment that can be repeated, at least hypothetically, under essentially identical conditions and that may lead to different outcomes on different trials. The set of all possible outcomes of an experiment is called a “sample space.” The experiment of tossing a coin once results in a sample space with two possible outcomes, “heads” and “tails.” Tossing two dice has a sample space with 36 possible outcomes, each of which can be identified with an ordered pair (i, j), where i and j assume one of the values 1, 2, 3, 4, 5, 6 and denote the faces showing on the individual dice. It is important to think of the dice as identifiable (say by a difference in colour), so that the outcome (1, 2) is different from (2, 1). An “event” is a well-defined subset of the sample space. For example, the event “the sum of the faces showing on the two dice equals six” consists of the five outcomes (1, 5), (2, 4), (3, 3), (4, 2), and (5, 1).



SAMPLE SPACE FOR A PAIR OF DICE

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

sample space for a pair of dice
Sample space for a pair of dice.
Encyclopædia Britannica, Inc.

A third example is to draw n balls from an urn containing balls of various colours. A generic outcome to this experiment is an n-tuple, where the ith entry specifies the colour of the ball obtained on the ith draw (i = 1, 2,…, n). In spite of the simplicity of this experiment, a thorough understanding gives the theoretical basis for opinion polls and sample surveys. For example, individuals in a population favouring a particular candidate in an election may be identified with balls of a particular colour, those favouring a different candidate may be identified with a different colour, and so on. Probability theory provides the basis for learning about the contents of the urn from the sample of balls drawn from the urn; an application is to learn about the electoral preferences of a population on the basis of a sample drawn from that population.

Another application of simple urn models is to use clinical trials designed to determine whether a new treatment for a disease, a new drug, or a new surgical procedure is better than a standard treatment. In the simple case in which treatment can be regarded as either success or failure, the goal of the clinical trial is to discover whether the new treatment more frequently leads to success than does the standard treatment. Patients with the disease can be identified with balls in an urn. The red balls are those patients who are cured by the new treatment, and the black balls are those not cured. Usually there is a control group, who receive the standard treatment. They are represented by a second urn with a possibly different fraction of red balls. The goal of the experiment of drawing some number of balls from each urn is to discover on the basis of the sample which urn has the larger fraction of red balls. A variation of this idea can be used to test the efficacy of a new vaccine. Perhaps the largest and most famous example was the test of the Salk vaccine for poliomyelitis conducted in 1954. It was organized by the U.S. Public Health Service and involved almost two million children. Its success has led to the almost complete elimination of polio as a health problem in the industrialized parts of the world. Strictly speaking, these applications are problems of statistics, for which the foundations are provided by probability theory.

In contrast to the experiments described above, many experiments have infinitely many possible outcomes. For example, one can toss a coin until “heads” appears for the first time. The number of possible tosses is n = 1, 2,…. Another example is to twirl a spinner. For an idealized spinner made from a straight line segment having no width and pivoted at its centre, the set of possible outcomes is the set of all angles that the final position of the spinner makes with some fixed direction, equivalently all real numbers in $[0, 2\pi)$. Many measurements in the natural and social sciences, such as volume, voltage, temperature, reaction time, marginal income, and so on, are made on continuous scales and at least in theory involve infinitely many possible values. If the repeated measurements on different subjects or at different times on the same subject can lead to different outcomes, probability theory is a possible tool to study this variability.

Because of their comparative simplicity, experiments with finite sample spaces are discussed first. In the early development of probability theory, mathematicians considered only those experiments for which it seemed reasonable, based on considerations of symmetry, to suppose that all outcomes of the experiment were “equally likely.” Then in a large number of trials all outcomes should occur with approximately the same frequency. The probability of an event is defined to be the ratio of the number of cases favourable to the event—i.e., the number of outcomes in the subset of the sample space defining the event—to the total number of cases. Thus, the 36 possible outcomes in the throw of two dice are assumed equally likely, and the probability of obtaining “six” is the number of favourable cases, 5, divided by 36, or 5/36.

Now suppose that a coin is tossed n times, and consider the probability of the event “heads does not occur” in the n tosses. An outcome of the experiment is an n-tuple, the kth entry of which identifies the result of the kth toss. Since there are two possible outcomes for each toss, the number of elements in the sample space is 2n. Of these, only one outcome corresponds to having no heads, so the required probability is 1/2n.

It is only slightly more difficult to determine the probability of “at most one head.” In addition to the single case in which no head occurs, there are n cases in which exactly one head occurs, because it can occur on the first, second,…, or nth toss. Hence, there are n + 1 cases favourable to obtaining at most one head, and the desired probability is (n + 1)/2n.

Probability

Probability means possibility. It is a branch of mathematics that deals with the occurrence of a random event. The value is expressed from zero to one. Probability has been introduced in Maths to predict how likely events are to happen.

Learn More here: Study Mathematics

The meaning of probability is basically the extent to which something is likely to happen. This is the basic probability theory, which is also used in the probability distribution, where you will learn the possibility of outcomes for a random experiment. To find the probability of a single event to occur, first, we should know the total number of possible outcomes.

Probability Definition in Math

Probability is a measure of the likelihood of an event to occur. Many events cannot be predicted with total certainty. We can predict only the chance of an event to occur i.e. how likely they are to happen, using it. Probability can range in from 0 to 1, where 0 means the event to be an impossible one and 1 indicates a certain event. Probability for Class 10 is an important topic for the students which explains all the basic concepts of this topic. The probability of all the events in a sample space adds up to 1.

For example, when we toss a coin, either we get Head OR Tail, only two possible outcomes are possible (H, T). But if we toss two coins in the air, there could be three possibilities of events to occur, such as both the coins show heads or both shows tails or one shows heads and one tail, i.e.(H, H), (H, T),(T, T).

Formula for Probability

The probability formula is defined as the possibility of an event to happen is equal to the ratio of the number of favourable outcomes and the total number of outcomes.

Probability of event to happen P(E) = Number of favourable outcomes/Total Number of outcomes

Sometimes students get mistaken for “favourable outcome” with “desirable outcome”. This is the basic formula. But there are some more formulas for different situations or events.

Examples and Solutions

1) There are 6 pillows in a bed, 3 are red, 2 are yellow and 1 is blue. What is the probability of picking a yellow pillow?

Ans: The probability is equal to the number of yellow pillows in the bed divided by the total number of pillows, i.e. 2/6 = 1/3.

2) There is a container full of coloured bottles, red, blue, green and orange. Some of the bottles are picked out and displaced. Sumit did this 1000 times and got the following results:

- No. of blue bottles picked out: 300
- No. of red bottles: 200
- No. of green bottles: 450
- No. of orange bottles: 50

a) What is the probability that Sumit will pick a green bottle?

Ans: For every 1000 bottles picked out, 450 are green.

Therefore, P(green) = 450/1000 = 0.45

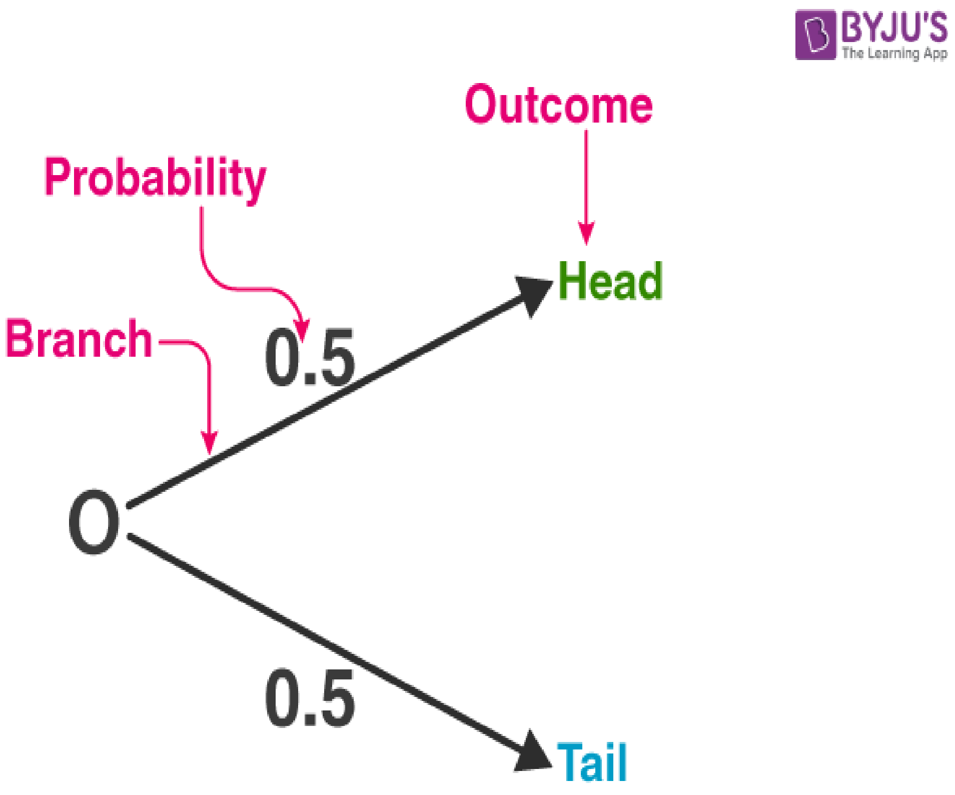
b) If there are 100 bottles in the container, how many of them are likely to be green?

Ans: The experiment implies that 450 out of 1000 bottles are green.

Therefore, out of 100 bottles, 45 are green.

Probability Tree

The tree diagram helps to organize and visualize the different possible outcomes. Branches and ends of the tree are two main positions. Probability of each branch is written on the branch, whereas the ends are containing the final outcome. Tree diagram is used to figure out when to multiply and when to add. You can see below a tree diagram for the coin:



Types of Probability

There are three major types of probabilities:

- Theoretical Probability
- Experimental Probability
- Axiomatic Probability

Theoretical Probability

It is based on the possible chances of something to happen. The theoretical probability is mainly based on the reasoning behind probability. For example, if a coin is tossed, the theoretical probability of getting a head will be ½.

Experimental Probability

It is based on the basis of the observations of an experiment. The experimental probability can be calculated based on the number of possible outcomes by the total number of trials. For example, if a coin is tossed 10 times and heads is recorded 6 times then, the experimental probability for heads is 6/10 or, 3/5.

Axiomatic Probability

In axiomatic probability, a set of rules or axioms are set which applies to all types. These axioms are set by Kolmogorov and are known as Kolmogorov's three axioms. With the axiomatic approach to probability, the chances of occurrence or non-occurrence of the events can be quantified. The axiomatic probability lesson covers this concept in detail with Kolmogorov's three rules (axioms) along with various examples.

Conditional Probability is the likelihood of an event or outcome occurring based on the occurrence of a previous event or outcome.

Probability of an Event

Assume an event E can occur in r ways out of a sum of n probable or possible equally likely ways. Then the probability of happening of the event or its success is expressed as;

P(E) = r/n

The probability that the event will not occur or known as its failure is expressed as:

P(E') = (n-r)/n = 1-(r/n)

E' represents that the event will not occur.

Therefore, now we can say;

P(E) + P(E') = 1

This means that the total of all the probabilities in any random test or experiment is equal to 1.

What are Equally Likely Events?

When the events have the same theoretical probability of happening, then they are called equally likely events. The results of a sample space are called equally likely if all of them have the same probability of occurring. For example, if you throw a die, then the probability of getting 1 is 1/6. Similarly, the probability of getting all the numbers from 2,3,4,5 and 6, one at a time is 1/6. Hence, the following are some examples of equally likely events when throwing a die:

- Getting 3 and 5 on throwing a die
- Getting an even number and an odd number on a die
- Getting 1, 2 or 3 on rolling a die

are equally likely events, since the probabilities of each event are equal.

Complementary Events

The possibility that there will be only two outcomes which states that an event will occur or not. Like a person will come or not come to your house, getting a job or not getting a job, etc. are examples of complementary events. Basically, the complement of an event occurring in the exact opposite that the probability of it is not occurring. Some more examples are:

- It will rain or not rain today
- The student will pass the exam or not pass.
- You win the lottery or you don't.

Probability Density Function

The Probability Density Function (PDF) is the probability function which is represented for the density of a continuous random variable lying between a certain range of values. Probability Density Function explains the normal distribution and how mean and deviation exists. The standard normal distribution is used to create a database or statistics, which are often used in science to represent the real-valued variables, whose distribution are not known.

Probability Terms and Definition

Some of the important probability terms are discussed here:

Term	Definition	Example
Sample Space	The set of all the possible outcomes to occur in any trial	1. Tossing a coin, Sample Space (S) = {H,T} 2. Rolling a die, Sample Space (S) = {1,2,3,4,5,6}
Sample Point	It is one of the possible results	In a deck of Cards: • 4 of hearts is a sample point. • the queen of clubs is a sample point.
Experiment or Trial	A series of actions where the outcomes are always uncertain.	The tossing of a coin, Selecting a card from a deck of cards, throwing a dice.
Event	It is a single outcome of an experiment.	Getting a Heads while tossing a coin is an event.
Outcome	Possible result of a trial/experiment	T (tail) is a possible outcome when a coin is tossed.

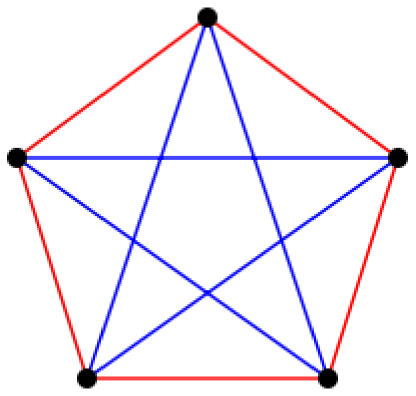
Term	Definition	Example
Complimentary event	The non-happening events. The complement of an event A is the event, not A (or A’)	Standard 52-card deck, A = Draw a heart, then A’ = Don’t draw a heart
Impossible Event	The event cannot happen	In tossing a coin, impossible to get both head and tail at the same time

In combinatorial mathematics, Ramsey's theorem, in one of its graph-theoretic forms, states that one will find monochromatic cliques in any edge labelling (with colours) of a sufficiently large complete graph. To demonstrate the theorem for two colours (say, blue and red), let r and s be any two positive integers.[1] Ramsey's theorem states that there exists a least positive integer R(r, s) for which every blue-red edge colouring of the complete graph on R(r, s) vertices contains a blue clique on r vertices or a red clique on s vertices. (Here R(r, s) signifies an integer that depends on both r and s.)

Ramsey's theorem is a foundational result in combinatorics. The first version of this result was proved by F. P. Ramsey. This initiated the combinatorial theory now called Ramsey theory, that seeks regularity amid disorder: general conditions for the existence of substructures with regular properties. In this application it is a question of the existence of monochromatic subsets, that is, subsets of connected edges of just one colour.

An extension of this theorem applies to any finite number of colours, rather than just two. More precisely, the theorem states that for any given number of colours, c, and any given integers n1, …, nc, there is a number, R(n1, …, nc), such that if the edges of a complete graph of order R(n1, ..., nc) are coloured with c different colours, then for some i between 1 and c, it must contain a complete subgraph of order ni whose edges are all colour i. The special case above has c = 2 (and n1 = r and n2 = s).

R(3, 3) = 6[edit]



A 2-edge-labeling of K5 with no monochromatic K3

Suppose the edges of a complete graph on 6 vertices are coloured red and blue. Pick a vertex, v. There are 5 edges incident to v and so (by the pigeonhole principle) at least 3 of them must be the same colour. Without loss of generality we can assume at least 3 of these edges, connecting the vertex, v, to vertices, r, s and t, are blue. (If not, exchange red and blue in what follows.) If any of the edges, (r, s), (r, t), (s, t), are also blue then we have an entirely blue triangle. If not, then those three edges are all red and we have an entirely red triangle. Since this argument works for any colouring, any K6 contains a monochromatic K3, and therefore R(3, 3) ≤ 6. The popular version of this is called the theorem on friends and strangers.

An alternative proof works by double counting. It goes as follows: Count the number of ordered triples of vertices, x, y, z, such that the edge, (xy), is red and the edge, (yz), is blue. Firstly, any given vertex will be the middle of either 0 × 5 = 0 (all edges from the vertex are the same colour), 1 × 4 = 4 (four are the same colour, one is the other colour), or 2 × 3 = 6 (three are the same colour, two are the other colour) such triples. Therefore, there are at most 6 × 6 = 36 such triples. Secondly, for any non-monochromatic triangle (xyz), there exist precisely two such triples. Therefore, there are at most 18 non-monochromatic triangles. Therefore, at least 2 of the 20 triangles in the K6 are monochromatic.

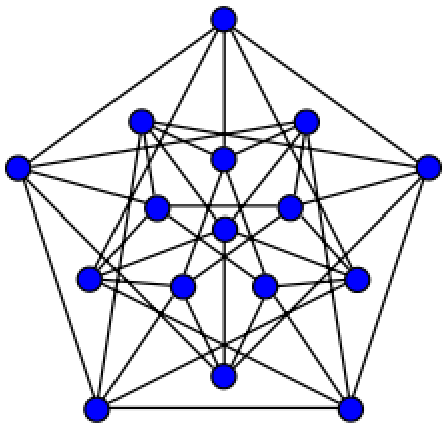
Conversely, it is possible to 2-colour a K5 without creating any monochromatic K3, showing that R(3, 3) > 5. The unique[2] colouring is shown to the right. Thus R(3, 3) = 6.

The task of proving that R(3, 3) ≤ 6 was one of the problems of William Lowell Putnam Mathematical Competition in 1953, as well as in the Hungarian Math Olympiad in 1947.

A multicolour Ramsey number is a Ramsey number using 3 or more colours. There are (up to symmetries) only two non-trivial multicolour Ramsey numbers for which the exact value is known, namely R(3, 3, 3) = 17 and R(3, 3, 4) = 30.[3]

Suppose that we have an edge colouring of a complete graph using 3 colours, red, green and blue. Suppose further that the edge colouring has no monochromatic triangles. Select a vertex v. Consider the set of vertices that have a red edge to the vertex v. This is called the red neighbourhood of v. The red neighbourhood of v cannot contain any red edges, since otherwise there would be a red triangle consisting of the two endpoints of that red edge and the vertex v. Thus, the induced edge colouring on the red neighbourhood of v has edges coloured with only two colours, namely green and blue. Since R(3, 3) = 6, the red neighbourhood of v can contain at most 5 vertices. Similarly, the green and blue neighbourhoods of v can contain at most 5 vertices each. Since every vertex, except for v itself, is in one of the red, green or blue neighbourhoods of v, the entire complete graph can have at most 1 + 5 + 5 + 5 = 16 vertices. Thus, we have R(3, 3, 3) ≤ 17.

To see that R(3, 3, 3) ≥ 17, it suffices to draw an edge colouring on the complete graph on 16 vertices with 3 colours that avoids monochromatic triangles. It turns out that there are exactly two such colourings on K16, the so-called untwisted and twisted colourings. Both colourings are shown in the figures to the right, with the untwisted colouring on the top, and the twisted colouring on the bottom.



Clebsch graph

If we select any colour of either the untwisted or twisted colouring on K16, and consider the graph whose edges are precisely those edges that have the specified colour, we will get the Clebsch graph.

It is known that there are exactly two edge colourings with 3 colours on K15 that avoid monochromatic triangles, which can be constructed by deleting any vertex from the untwisted and twisted colourings on K16, respectively.

It is also known that there are exactly 115 edge colourings with 3 colours on K14 that avoid monochromatic triangles, provided that we consider edge colourings that differ by a permutation of the colours as being the same.

2-colour case

The theorem for the 2-colour case, can be proved by induction on r + s.[4] It is clear from the definition that for all n, R(n, 2) = R(2, n) = n. This starts the induction. We prove that R(r, s) exists by finding an explicit bound for it. By the inductive hypothesis R(r − 1, s) and R(r, s − 1) exist.

Lemma 1. R(r, s) ≤ R(r − 1, s) + R(r, s − 1):

Proof. Consider a complete graph on R(r − 1, s) + R(r, s − 1) vertices whose edges are coloured with two colours. Pick a vertex v from the graph, and partition the remaining vertices into two sets M and N, such that for every vertex w, w is in M if (v, w) is blue, and w is in N if (v, w) is red. Because the graph has R(r − 1, s) + R(r, s − 1) = |M| + |N| + 1 vertices, it follows that either |M| ≥ R(r − 1, s) or |N| ≥ R(r, s − 1). In the former case, if M has a red Ks then so does the original graph and we are finished. Otherwise M has a blue Kr−1 and so M ∪ {v} has a blue Kr by the definition of M. The latter case is analogous. Thus the claim is true and we have completed the proof for 2 colours.

In this 2-colour case, if R(r − 1, s) and R(r, s − 1) are both even, the induction inequality can be strengthened to:[5]

R(r, s) ≤ R(r − 1, s) + R(r, s − 1) − 1.

Proof. Suppose p = R(r − 1, s) and q = R(r, s − 1) are both even. Let t = p + q − 1 and consider a two-coloured graph of t vertices. If is degree of -th vertex in the graph, then, according to the Handshaking lemma, is even. Given that t is odd, there must be an even . Assume is even, M and N are the vertices incident to vertex 1 in the blue and red subgraphs, respectively. Then both and are even. According to the Pigeonhole principle, either , or . Since is even, while is odd, the first inequality can be strengthened, so either or . Suppose . Then either the M subgraph has a red and the proof is complete, or it has a blue which along with vertex 1 makes a blue . The case is treated similarly.

Computational complexity

Erdős asks us to imagine an alien force, vastly more powerful than us, landing on Earth and demanding the value of R(5, 5) or they will destroy our planet. In that case, he claims, we should marshal all our computers and all our mathematicians and attempt to find the value. But suppose, instead, that they ask for R(6, 6). In that case, he believes, we should attempt to destroy the aliens.

— Joel Spencer[7]

A sophisticated computer program does not need to look at all colourings individually in order to eliminate all of them; nevertheless it is a very difficult computational task that existing software can only manage on small sizes. Each complete graph Kn has 1/$\boxed{2}$n(n − 1) edges, so there would be a total of cn(n − 1)/$\boxed{2}$ graphs to search through (for c colours) if brute force is used.[8] Therefore, the complexity for searching all possible graphs (via brute force) is O(cn2) for c colourings and at most n nodes.

The situation is unlikely to improve with the advent of quantum computers. The best known algorithm[citation needed] exhibits only a quadratic speedup (c.f. Grover's algorithm) relative to classical computers, so that the computation time is still exponential in the number of colours.[9

Known values

As described above, R(3, 3) = 6. It is easy to prove that R(4, 2) = 4, and, more generally, that R(s, 2) = s for all s: a graph on s − 1 nodes with all edges coloured red serves as a counterexample and proves that R(s, 2) ≥ s; among colourings of a graph on s nodes, the colouring with all edges coloured red contains a s-node red subgraph, and all other colourings contain a 2-node blue subgraph (that is, a pair of nodes connected with a blue edge.)

Using induction inequalities, it can be concluded that R(4, 3) ≤ R(4, 2) + R(3, 3) − 1 = 9, and therefore R(4, 4) ≤ R(4, 3) + R(3, 4) ≤ 18. There are only two (4, 4, 16) graphs (that is, 2-colourings of a complete graph on 16 nodes without 4-node red or blue complete subgraphs) among 6.4 × 1022 different 2-colourings of 16-node graphs, and only one (4, 4, 17) graph (the Paley graph of order 17) among 2.46 × 1026 colourings.[6] (This was proven by Evans, Pulham and Sheehan in 1979.) It follows that R(4, 4) = 18.

The fact that R(4, 5) = 25 was first established by Brendan McKay and Stanisław Radziszowski in 1995.[10]

The exact value of R(5, 5) is unknown, although it is known to lie between 43 (Geoffrey Exoo (1989)[11]) and 48 (Angeltveit and McKay (2017)[12]) (inclusive).

In 1997, McKay, Radziszowski and Exoo employed computer-assisted graph generation methods to conjecture that R(5, 5) = 43. They were able to construct exactly 656 (5, 5, 42) graphs, arriving at the same set of graphs through different routes. None of the 656 graphs can be extended to a (5, 5, 43) graph.[13]

For R(r, s) with r, s > 5, only weak bounds are available. Lower bounds for R(6, 6) and R(8, 8) have not been improved since 1965 and 1972, respectively.[3]

R(r, s) with r, s ≤ 10 are shown in the table below. Where the exact value is unknown, the table lists the best known bounds. R(r, s) with r, s < 3 are given by R(1, s) = 1 and R(2, s) = s for all values of s.

The standard survey on the development of Ramsey number research is the Dynamic Survey 1 of the Electronic Journal of Combinatorics, by Stanisław Radziszowski, which is periodically updated.[3] Where not cited otherwise, entries in the table below are taken from the March 2017 edition. (Note there is a trivial symmetry across the diagonal since R(r, s) = R(s, r).)

Values / known bounding ranges for Ramsey numbers R(r, s) (sequence A212954 in the OEIS)

s r	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2		2	3	4	5	6	7	8	9	10
3			6	9	14	18	23	28	36	40–42
4				18	25[10]	36–41	49–61	59[14]–84	73–115	92–149
5					43–48	58–87	80–143	101–216	133–316	149[14]–442
6						102–165	115[14]–298	134[14]–495	183–780	204–1171
7							205–540	217–1031	252–1713	292–2826

8								282–1870	329–3583	343–6090
9									565–6588	581–12677
10						79823556				

Asymptotics

The inequality R(r, s) ≤ R(r − 1, s) + R(r, s − 1) may be applied inductively to prove that

In particular, this result, due to Erdős and Szekeres, implies that when r = s,An exponential lower bound,

was given by Erdős in 1947 and was instrumental in his introduction of the probabilistic method. There is obviously a huge gap between these two bounds: for example, for s = 10, this gives 101 ≤ R(10, 10) ≤ 48620. Nevertheless, exponential growth factors of either bound have not been improved to date and still stand at 4 and √2 respectively. There is no known explicit construction producing an exponential lower bound. The best known lower and upper bounds for diagonal Ramsey numbers currently stand at

due to Spencer and Conlon respectively.For the off-diagonal Ramsey numbers R(3, t), it is known that they are of order ; this may be stated equivalently as saying that the smallest possible independence number in an n-vertex triangle-free graph is

The upper bound for R(3, t) is given by Ajtai, Komlós, and Szemerédi, the lower bound was obtained originally by Kim, and was improved by Griffiths, Morris, Fiz Pontiveros, and Bohman and Keevash, by analysing the triangle-free process. More generally, for off-diagonal Ramsey numbers, R(s, t), with s fixed and t growing, the best known bounds

due to Bohman and Keevash and Ajtai, Komlós and Szemerédi respectively.

Infinite graphs

A further result, also commonly called Ramsey's theorem, applies to infinite graphs. In a context where finite graphs are also being discussed it is often called the "Infinite Ramsey theorem". As intuition provided by the pictorial representation of a graph is diminished when moving from finite to infinite graphs, theorems in this area are usually phrased in set-theoretic terminology.[15]

Theorem. Let X be some infinite set and colour the elements of X(n) (the subsets of X of size n) in c different colours. Then there exists some infinite subset M of X such that the size n subsets of M all have the same colour.

Proof: The proof is by induction on n, the size of the subsets. For n = 1, the statement is equivalent to saying that if you split an infinite set into a finite number of sets, then one of them is infinite. This is evident. Assuming the theorem is true for n ≤ r, we prove it for n = r + 1. Given a c-colouring of the (r + 1)-element subsets of X, let a0 be an element of X and let Y = X \ {a0}. We then induce a c-colouring of the r-element subsets of Y, by just adding a0 to each r-element subset (to get an (r + 1)-element subset of X). By the induction hypothesis, there exists an infinite subset Y1 of Y such that every r-element subset of Y1 is coloured the same colour in the induced colouring. Thus there is an element a0 and an infinite subset Y1 such that all the (r + 1)-element subsets of X consisting of a0 and r elements of Y1 have the same colour. By the same argument, there is an element a1 in Y1 and an infinite subset Y2 of Y1 with the same properties. Inductively, we obtain a sequence {a0, a1, a2, …} such that the colour of each (r + 1)-element subset (ai(1), ai(2), …, ai(r + 1)) with i(1) < i(2) < ... < i(r + 1) depends only on the value of i(1). Further, there are infinitely many values of i(n) such that this colour will be the same. Take these ai(n)'s to get the desired monochromatic set.

A stronger but unbalanced infinite form of Ramsey's theorem for graphs, the Erdős–Dushnik–Miller theorem, states that every infinite graph contains either a countably infinite independent set, or an infinite clique of the same cardinality as the original graph.

Discrete Maths | Generating Functions-Introduction and Prerequisites

Last Updated: 13-12-2019

Prerequisite – Combinatorics Basics, Generalized PnC Set 1, Set 2

Definition : Generating functions are used to represent sequences efficiently by coding the terms of a sequence as coefficients of powers of a variable (say) in a formal power series.

Now with the formal definition done, we can take a minute to discuss why should we learn this concept.

This concept can be applied to solve many problems in mathematics. There is a huge chunk of mathematics dealing with just generating functions.

It can be used to solve various kinds of Counting problems easily.

It can be used to solve recurrence relations by translating the relation in terms of sequence to a problem about functions.

It can be used to prove combinatorial identities.

In simple words generating functions can be used to translate problems about sequences to problems about functions which are comparatively easy to solve using maneuvers.Some basic prerequisitesBefore we start let’s go through some basic Combinatorics formulaA generating function is a “formal” power series in the sense that we usually regard x as a placeholder rather than a number. Only in rare cases will we actually evaluate a generating function by letting x take a real number value, so we generally ignore the issue of convergence of real numbers is the infinite

series:
Note: we'll indicate the correspondence between a sequence and its generating function with a double-sided arrow.

Some Important sequences and their results…

If you want to know more about the derivation of the given sequences you may refer

Note that all the above series is infinite. There is one peculiar result of finite series which will be very handy so it's recommended to remember these results to solve problems quickly.

An important result:

Operation on generating Functions:

Scaling

Addition

Right Shift

Differentiation

We can use these operations to get new sequences from known sequences, and new generating functions from known generating functions.

Scaling:

Multiplying a generating function by a constant scales every term in the associated sequence by the same consta

Addition:

Adding generating functions corresponds to adding the two sequences term by term.
Right Shifting

Differentiation:

In general, differentiating a generating function has two effects on the corresponding sequence: each term is multiplied by its index and the entire sequence is shifted left one place.

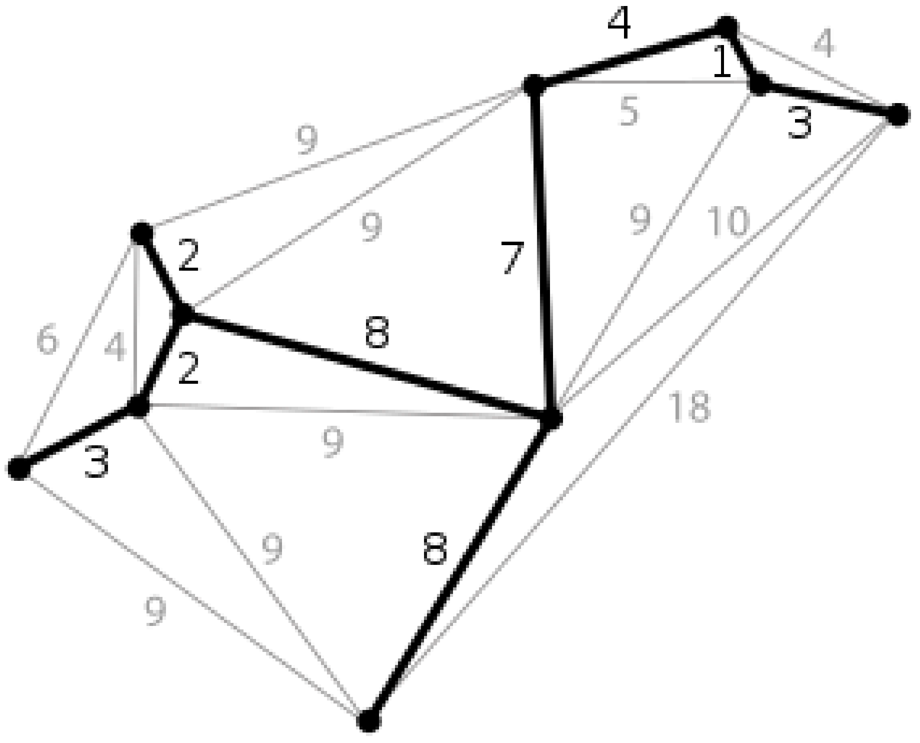
Example:

What sequence is represented by the following series :

SolutionBy now you must have got this, the coefficient of a0 = 1, a1 = 0, a2 = 4, a3 = 0, a4 = 1, a5 = 1/999, a6 = 100.

So sequence is:

Combinatorial problem



A minimum spanning tree of a weighted planar graph. Finding a minimum spanning tree is a common problem involving combinatorial optimization.

Combinatorial optimization is a subfield of mathematical optimization that is related to operations research, algorithm theory, and computational complexity theory. It has important applications in several fields, including artificial intelligence, machine learning, auction theory, software engineering, applied mathematics and theoretical computer science.

Combinatorial optimization is a topic that consists of finding an optimal object from a finite set of objects.[1] In many such problems, exhaustive search is not tractable. It operates on the domain of those optimization problems in which the set of feasible solutions is discrete or can be reduced to discrete, and in which the goal is to find the best solution. Typical problems are the travelling salesman problem ("TSP"), the minimum spanning tree problem ("MST"), and the knapsack problem.

Some research literature[2] considers discrete optimization to consist of integer programming together with combinatorial optimization (which in turn is composed of optimization problems dealing with graph structures) although all of these topics have closely intertwined research literature. It often involves determining the way to efficiently allocate resources used to find solutions to mathematical problems.

Methods

There is a large amount of literature on polynomial-time algorithms for certain special classes of discrete optimization, a considerable amount of it unified by the theory of linear programming. Some examples of combinatorial optimization problems that fall into this framework are shortest paths and shortest-path trees, flows and circulations, spanning trees, matching, and matroid problems.

For NP-complete discrete optimization problems, current research literature includes the following topics:

- polynomial-time exactly solvable special cases of the problem at hand (e.g. see fixed-parameter tractable)
- algorithms that perform well on "random" instances (e.g. for TSP)
- approximation algorithms that run in polynomial time and find a solution that is "close" to optimal
- solving real-world instances that arise in practice and do not necessarily exhibit the worst-case behavior inherent in NP-complete problems (e.g. TSP instances with tens of thousands of nodes[6]).

Combinatorial optimization problems can be viewed as searching for the best element of some set of discrete items; therefore, in principle, any sort of search algorithm or metaheuristic can be used to solve them. Perhaps the most universally applicable approaches are branch-and-bound (an exact algorithm which can be stopped at any point in time to serve as heuristic), branch-and-cut (uses linear optimisation to generate bounds), dynamic programming (a recursive solution construction with limited search window) and tabu search (a greedy-type swapping algorithm). However, generic search algorithms are not guaranteed to find an optimal solution first, nor are they guaranteed to run quickly (in polynomial time). Since some discrete optimization problems are NP-complete, such as the traveling salesman problem[citation needed], this is expected unless P=NP.

NP optimization problem

An NP-optimization problem (NPO) is a combinatorial optimization problem with the following additional conditions.[8] Note that the below referred polynomials are functions of the size of the respective functions' inputs, not the size of some implicit set of input instances.

- the size of every feasible solution is polynomially bounded in the size of the

  given instance
- the languages and can be recognized in polynomial time, and
- is polynomial-time computable.

This implies that the corresponding decision problem is in NP. In computer science, interesting optimization problems usually have the above properties and are therefore NPO problems. A problem is additionally called a P-optimization (PO) problem, if there exists an algorithm which finds optimal solutions in polynomial time. Often, when dealing with the class NPO, one is interested in optimization problems for which the decision versions are NP-complete. Note that hardness relations are always with respect to some reduction. Due to the connection between approximation algorithms and computational optimization problems, reductions which preserve approximation in some respect are for this subject preferred than the usual Turing and Karp reductions. An example of such a reduction would be the L-reduction. For this reason, optimization problems with NP-complete decision versions are not necessarily called NPO-complete.[9]

NPO is divided into the following subclasses according to their approximability:[8]

- NPO(I): Equals FPTAS. Contains the Knapsack problem.
- NPO(II): Equals PTAS. Contains the Makespan scheduling problem.
- NPO(III): :The class of NPO problems that have polynomial-time algorithms which computes solutions with a cost at most c times the optimal cost (for minimization problems) or a cost at least of the optimal cost (for maximization problems). In Hromkovič's book, excluded from this class are all NPO(II)-problems save if P=NP. Without the exclusion, equals APX. Contains MAX-SAT and metric TSP.
- NPO(IV): :The class of NPO problems with polynomial-time algorithms approximating the optimal solution by a ratio that is polynomial in a logarithm of the size of the input. In Hromkovic's book, all NPO(III)-problems are excluded from this class unless P=NP. Contains the set cover problem.
- NPO(V): :The class of NPO problems with polynomial-time algorithms approximating the optimal solution by a ratio bounded by some function on n. In Hromkovic's book, all NPO(IV)-problems are excluded from this class unless P=NP. Contains the TSP and Max Clique problems.

Difference equation

In mathematics and in particular dynamical systems, a linear difference equation[1]:ch. 17[2]:ch. 10 or linear recurrence relation sets equal to 0 a polynomial that is linear in the various iterates of a variable—that is, in the values of the elements of a sequence. The polynomial's linearity means that each of its terms has degree 0 or 1. Usually the context is the evolution of some variable over time, with the current time period or discrete moment in time denoted as t, one period earlier denoted as t − 1, one period later as t + 1, etc.

An nth order linear difference equation is one that can be written in terms of parameters a1, ..., an and b as

The equation is called homogeneous if $b = 0$ and nonhomogeneous if $b \neq 0$. Since the longest time lag between iterates appearing in the equation is n, this is an nth order equation, where n could be any positive integer. When the longest lag is specified numerically so n does not appear notationally as the longest time lag, n is occasionally used instead of t to index iterates.

In the most general case the coefficients ai and b could themselves be functions of t; however, this article treats the most common case, that of constant coefficients. If the coefficients ai are polynomials in t the equation is called a linear recurrence equation with polynomial coefficients.

The solution of such an equation is a function of t, and not of any iterate values, giving the value of the iterate at any time. To find the solution it is necessary to know the specific values (known as initial conditions) of n of the iterates, and normally these are the n iterates that are oldest. The equation or its variable is said to be stable if from any set of initial conditions the variable's limit as time goes to infinity exists; this limit is called the steady state.

Difference equations are used in a variety of contexts, such as in economics to model the evolution through time of variables such as gross domestic product, the inflation rate, the exchange rate, etc. They are used in modeling such time series because values of these variables are only measured at discrete intervals. In econometric applications, linear difference equations are modeled with stochastic terms in the form of autoregressive (AR) models and in models such as vector autoregression (VAR) and autoregressive moving average (ARMA) models that combine AR with other features.

Unit II

Recurrence

Categories

In this chapter, we will discuss how recursive techniques can derive sequences and be used for solving counting problems. The procedure for finding the terms of a sequence in a recursive manner is called recurrence relation. We study the theory of linear recurrence relations and their solutions. Finally, we introduce generating functions for solving recurrence relations.

Definition

A recurrence relation is an equation that recursively defines a sequence where the next term is a function of the previous terms (Expressing FnFn as some combination of FiFi with i<ni<n).
Example − Fibonacci series − Fn=Fn−1+Fn−2Fn=Fn−1+Fn−2, Tower of Hanoi − Fn=2Fn−1+1Fn=2Fn−1+1

Linear Recurrence Relations

A linear recurrence equation of degree k or order k is a recurrence equation which is in the

Recurrence relations	Initial values	Solutions
Fn = Fn-1 + Fn-2	a1 = a2 = 1	Fibonacci number
Fn = Fn-1 + Fn-2	a1 = 1, a2 = 3	Lucas Number
Fn = Fn-2 + Fn-3	a1 = a2 = a3 = 1	Padovan sequence
Fn = 2Fn-1 + Fn-2	a1 = 0, a2 = 1	Pell number

format xn=A1xn−1+A2xn−1+A3xn−1+…Akxn−kxn=A1xn−1+A2xn−1+A3xn−1+…Ak xn−k(AnAn is a constant and Ak≠0Ak≠0) on a sequence of numbers as a first-degree polynomial.

These are some examples of linear recurrence equations −

How to solve linear recurrence relation

Suppose, a two ordered linear recurrence relation is
− Fn=AFn−1+BFn−2Fn=AFn−1+BFn−2 where A and B are real numbers.

The characteristic equation for the above recurrence relation is −

x2−Ax−B=0x2−Ax−B=0

Three cases may occur while finding the roots −

Case 1 − If this equation factors as (x−x1)(x−x1)=0(x−x1)(x−x1)=0 and it produces two distinct real roots x1x1 and x2x2, then Fn=axn1+bxn2Fn=ax1n+bx2n is the solution. [Here, a and b are constants]

Case 2 − If this equation factors as (x−x1)2=0(x−x1)2=0 and it produces single real root x1x1, then Fn=axn1+bnxn1Fn=ax1n+bnx1n is the solution.

Case 3 − If the equation produces two distinct complex roots, x1x1 and x2x2 in polar form x1=r∠θx1=r∠θ and x2=r∠(−θ)x2=r∠(−θ),
then Fn=rn(acos(nθ)+bsin(nθ))Fn=rn(acos(nθ)+bsin(nθ)) is the solution.

Problem 1

Solve the recurrence
relation Fn=5Fn−1−6Fn−2Fn=5Fn−1−6Fn−2 where F0=1F0=1 and F1=4F1=4

Solution

The characteristic equation of the recurrence relation is −

x2−5x+6=0,x2−5x+6=0,

So, (x−3)(x−2)=0(x−3)(x−2)=0

Hence, the roots are −

x1=3x1=3 and x2=2x2=2

The roots are real and distinct. So, this is in the form of case 1

Hence, the solution is −

Fn=axn1+bxn2Fn=ax1n+bx2n

Here, Fn=a3n+b2n (As x1=3 and x2=2)Fn=a3n+b2n (As x1=3 and x2=2)

Therefore,

1=F0=a30+b20=a+b1=F0=a30+b20=a+b
4=F1=a31+b21=3a+2b4=F1=a31+b21=3a+2b
Solving these two equations, we get a=2a=2 and b=−1b=−1

Hence, the final solution is −

Fn=2.3n+(−1).2n=2.3n−2nFn=2.3n+(−1).2n=2.3n−2n

Problem 2

Solve the recurrence relation
− Fn=10Fn−1−25Fn−2Fn=10Fn−1−25Fn−2 where F0=3F0=3 and F1=17F1=17

Solution

The characteristic equation of the recurrence relation is −

x2−10x−25=0x2−10x−25=0

So (x−5)2=0(x−5)2=0
Hence, there is single real root x1=5x1=5

As there is single real valued root, this is in the form of case 2

Hence, the solution is −

Fn=axn1+bnxn1Fn=ax1n+bnx1n
3=F0=a.50+(b)(0.5)0=a3=F0=a.50+(b)(0.5)0=a
17=F1=a.51+b.1.51=5a+5b17=F1=a.51+b.1.51=5a+5b
Solving these two equations, we get a=3a=3 and b=2/5b=2/5
Hence, the final solution is − Fn=3.5n+(2/5).n.2nFn=3.5n+(2/5).n.2n

Problem 3

Solve the recurrence relation Fn=2Fn−1−2Fn−2Fn=2Fn−1−2Fn−2 where F0=1F0=1 and F1=3F1=3

Solution

The characteristic equation of the recurrence relation is −

x2−2x−2=0x2−2x−2=0

Hence, the roots are −

x1=1+ix1=1+i and x2=1−ix2=1−i

In polar form,

x1=r∠θx1=r∠θ and x2=r∠(−θ),x2=r∠(−θ), where r=2–√r=2 and θ=π4θ=π4

The roots are imaginary. So, this is in the form of case 3.

Hence, the solution is −

Fn=(2–√)n(acos(n.⊓/4)+bsin(n.⊓/4))Fn=(2)n(acos(n.⊓/4)+bsin(n.⊓/4))
1=F0=(2–√)0(acos(0.⊓/4)+bsin(0.⊓/4))=a1=F0=(2)0(acos(0.⊓/4)+bsin(0.⊓/4))=a
3=F1=(2–√)1(acos(1.⊓/4)+bsin(1.⊓/4))=2–√(a/2–√+b/2–√)3=F1=(2)1(acos(1.⊓/4)+bsin(1.⊓/4))=2(a/2+b/2)
Solving these two equations we get a=1a=1 and b=2b=2

Hence, the final solution is −

Fn=(2–√)n(cos(n.π/4)+2sin(n.π/4))Fn=(2)n(cos(n.π/4)+2sin(n.π/4))

Non-Homogeneous Recurrence Relation and Particular Solutions

A recurrence relation is called non-homogeneous if it is in the form

Fn=AFn−1+BFn−2+f(n)Fn=AFn−1+BFn−2+f(n) where f(n)≠0f(n)≠0
Its associated homogeneous recurrence relation is Fn=AFn–1+BFn−2Fn=AFn–1+BFn−2
The solution (an)(an) of a non-homogeneous recurrence relation has two parts.

First part is the solution (ah)(ah) of the associated homogeneous recurrence relation and the second part is the particular solution (at)(at).
an=ah+atan=ah+at

Solution to the first part is done using the procedures discussed in the previous section.

To find the particular solution, we find an appropriate trial solution.

Let f(n)=cxnf(n)=cxn ; let x2=Ax+Bx2=Ax+B be the characteristic equation of the associated homogeneous recurrence relation and let x1x1 and x2x2 be its roots.

- If x≠x1x≠x1 and x≠x2x≠x2, then at=Axnat=Axn
- If x=x1x=x1, x≠x2x≠x2, then at=Anxnat=Anxn
- If x=x1=x2x=x1=x2, then at=An2xnat=An2xn

Example

Let a non-homogeneous recurrence relation be Fn=AFn–1+BFn−2+f(n)Fn=AFn–1+BFn−2+f(n) with characteristic roots x1=2x1=2 and x2=5x2=5. Trial solutions for different possible values of f(n)f(n) are as follows –

f(n)	Trial solutions
4	A
5.2n	An2n
8.5n	An5n
4n	A4n
2n2+3n+1	An2+Bn+C

Problem

Solve the recurrence relation Fn=3Fn−1+10Fn−2+7.5nFn=3Fn−1+10Fn−2+7.5n where F0=4F0=4 and F1 =3F1=3

Solution

This is a linear non-homogeneous relation, where the associated homogeneous equation is Fn=3Fn−1+10Fn−2Fn=3Fn−1+10Fn−2 and f(n)=7.5nf(n)=7.5n

The characteristic equation of its associated homogeneous relation is −

x2−3x−10=0x2−3x−10=0

Or, (x−5)(x+2)=0(x−5)(x+2)=0
Or, x1=5x1=5 and x2=−2x2=−2
Hence ah=a.5n+b.(−2)nah=a.5n+b.(−2)n , where a and b are constants.

Since f(n)=7.5nf(n)=7.5n, i.e. of the form c.xnc.xn, a reasonable trial solution of at will be AnxnAnxn
at=Anxn=An5nat=Anxn=An5n

After putting the solution in the recurrence relation, we get −

An5n=3A(n–1)5n−1+10A(n–2)5n−2+7.5nAn5n=3A(n–1)5n−1+10A(n–2)5n−2+7.5n
Dividing both sides by 5n−25n−2, we get
An52=3A(n−1)5+10A(n−2)50+7.52An52=3A(n−1)5+10A(n−2)50+7.52
Or, 25An=15An−15A+10An−20A+17525An=15An−15A+10An−20A+175
Or, 35A=17535A=175
Or, A=5A=5
So, Fn=An5n=5n5n=n5n+1Fn=An5n=5n5n=n5n+1

The solution of the recurrence relation can be written as −

Fn=ah+atFn=ah+at
=a.5n+b.(−2)n+n5n+1=a.5n+b.(−2)n+n5n+1
Putting values of F0=4F0=4 and F1=3F1=3, in the above equation, we get a=−2a=−2 and b=6b=6

Hence, the solution is −

Fn=n5n+1+6.(−2)n−2.5nFn=n5n+1+6.(−2)n−2.5n

Generating Functions

Generating Functions represents sequences where each term of a sequence is expressed as a coefficient of a variable x in a formal power series.

Mathematically, for an infinite sequence,
say a0,a1,a2,…,ak,…,a0,a1,a2,…,ak,…, the generating function will be −
Gx=a0+a1x+a2x2+⋯+akxk+⋯=∑k=0∞akxkGx=a0+a1x+a2x2+⋯+akxk+⋯=∑k=0∞ak xk

Some Areas of Application

Generating functions can be used for the following purposes −

- For solving a variety of counting problems. For example, the number of ways to make change for a Rs. 100 note with the notes of denominations Rs.1, Rs.2, Rs.5, Rs.10, Rs.20 and Rs.50
- For solving recurrence relations
- For proving some of the combinatorial identities
- For finding asymptotic formulae for terms of sequences

Problem 1

What are the generating functions for the sequences {ak}{ak} with ak=2ak=2 and ak=3kak=3k?

Solution

When ak=2ak=2, generating function, G(x)=∑∞k=02xk=2+2x+2x2+2x3+…G(x)=∑k=0∞2xk=2+2x+2x2+2x3+…

When ak=3k,G(x)=∑∞k=03kxk=0+3x+6x2+9x3+……ak=3k,G(x)=∑k=0∞3kxk=0+3x+6x2+9x3+……

Problem 2

What is the generating function of the infinite series; 1,1,1,1,…1,1,1,1,…?

Solution

Here, ak=1ak=1, for 0≤k≤∞0≤k≤∞
Hence, G(x)=1+x+x2+x3+…⋯=1(1−x)G(x)=1+x+x2+x3+…⋯=1(1−x)

Some Useful Generating Functions

- For ak=ak,G(x)=∑∞k=0akxk=1+ax+a2x2+……⋯=1/(1−ax)ak=ak,G(x)=∑k=0∞akxk=1+ax+a2x2+……⋯=1/(1−ax)
- For ak=(k+1),G(x)=∑∞k=0(k+1)xk=1+2x+3x2……⋯=1(1−x)2ak=(k+1),G(x)=∑k=0∞(k+1)xk=1+2x+3x2……⋯=1(1−x)2
- For ak=cnk,G(x)=∑∞k=0cnkxk=1+cn1x+cn2x2+……⋯+x2=(1+x)nak=ckn,G(x)=∑k=0∞cknxk=1+c1nx+c2nx2+……⋯+x2=(1+x)n
- For ak=1k!,G(x)=∑∞k=0xkk!=1+x+x22!+x33!……⋯=exak=1k!,G(x)=∑k=0∞xkk!=1+x+x22!+x33!……⋯=ex

Linear Recurrence Relations with Constant Coefficients

A Recurrence Relations is called linear if its degree is one.

The general form of linear recurrence relation with constant coefficient is

C0 yn+r+C1 yn+r-1+C2 yn+r-2+⋯+Cr yn=R (n)

Where C0,C1,C2......Cn are constant and R (n) is same function of independent variable n.

A solution of a recurrence relation in any function which satisfies the given equation.

Linear Homogeneous Recurrence Relations with Constant Coefficients:

The equation is said to be linear homogeneous difference equation if and only if R (n) = 0 and it will be of order n.

The equation is said to be linear non-homogeneous difference equation if R (n) ≠ 0.

Example1: The equation ar+3+6ar+2+12ar+1+8ar=0 is a linear non-homogeneous equation of order 3.

Example2: The equation ar+2-4ar+1+4ar= 3r + 2r is a linear non-homogeneous equation of order 2.

A linear homogeneous difference equation with constant coefficients is given by

C0 yn+C1 yn-1+C2 yn-2+⋯......+Cr yn-r=0 .... equation (i)

Where C0,C1,C2.....Cn are constants.

The solution of the equation (i) is of the form $A\propto_1^K$, where ∝1 is the characteristics root and A is constant.

Substitute the values of A ∝K for yn in equation (1), we have

C0 A∝K+C1 A∝K-1+C2 A∝K-2+⋯....+Cr A∝K-r=0.......equation (ii)

After simplifying equation (ii), we have

C0 ∝r+C1 ∝r-1+C2 ∝r-2+⋯Cr=0..........equation (iii)

The equation (iii) is called the characteristics equation of the difference equation.

If ∝1 is one of the roots of the characteristics equation, then $A\propto_1^K$ is a homogeneous solution to the difference equation.

To find the solution of the linear homogeneous difference equations, we have the four cases that are discussed as follows:

Case1: If the characteristic equation has n distinct real roots∝1, ∝2, ∝3,.......∝n.

Thus, $\alpha_1^K, \alpha_2^K, \ldots\ldots\ \alpha_n^K$ are all solutions of equation (i).

Also, we have $A_1 \propto_1^K, A_2 \propto_1^K, \ldots\ldots\ldots A_n \propto_n^K$ are all solutions of equation (i). The sums of solutions are also solutions.

Hence, the homogeneous solutions of the difference equation are

$$y_k = A_1 \propto_1^K, A_2 \propto_2^K, \ldots\ldots\ldots A_n \propto_n^K.$$

Case2: If the characteristics equation has repeated real roots.

If ∝1=∝2, then (A1+A2 K) $\propto_1^K$ is also a solution.

If ∝1=∝2=∝3 then (A1+A2 K+A3 K2) $\propto_1^K$ is also a solution.

Similarly, if root ∝1 is repeated n times, then.

(A1+A2 K+A3 K2+......+An Kn-1) $\propto_1^K$

The solution to the homogeneous equation.

Case3: If the characteristics equation has one imaginary root.

If α+iβ is the root of the characteristics equation, then α-iβ is also the root, where α and β are real.

Thus, (α+iβ)K and (α-iβ)K are solutions of the equations. This implies

(α+iβ)K A1+α-iβ)K A2

Is also a solution to the characteristics equation, where A1 and A2 are constants which are to be determined.

Case4: If the characteristics equation has repeated imaginary roots.

When the characteristics equation has repeated imaginary roots,

(C1+C2 k) (α+iβ)K +(C3+C4 K)(α-iβ)K

Is the solution to the homogeneous equation.

Example1: Solve the difference equation ar-3ar-1+2ar-2=0.

Solution: The characteristics equation is given by

s2-3s+2=0 or (s-1)(s-2)=0
⇒ s = 1, 2

Therefore, the homogeneous solution of the equation is given by

ar=C1r+C2.2r.

Example2: Solve the difference equation 9yK+2-6yK+1+yK=0.

Solution: The characteristics equation is

9s2-6s+1=0 or (3s-1)2=0

$\Rightarrow$ s = $\frac{1}{3}$ and $\frac{1}{3}$

Therefore, the homogeneous solution of the equation is given by

yK=(C1+C2 k).$\left(\frac{1}{3}\right)^{K}$

Example3: Solve the difference equation yK-yK-1-yK-2=0.

Solution: The characteristics equation is s2-s-1=0

s=$\frac{1 \pm \sqrt{1+4}}{2} = \frac{1 \pm \sqrt{5}}{2}$

Therefore, the homogeneous solution of the equation is

$$y_K = C_1 \left[\frac{1+\sqrt{5}}{2}\right]^{K} + C_2 \left[\frac{1-\sqrt{5}}{2}\right]^{K}$$

Example4: Solve the difference equation yK+4+4yK+3+8yK+2+8yK+1+4yK=0.

Solution: The characteristics equation is s4+4s3+8s2+8s+4=0
(s2+2s+2) (s2+2s+2)=0
s = -1±i,-1±i

Therefore, the homogeneous solution of the equation is given by

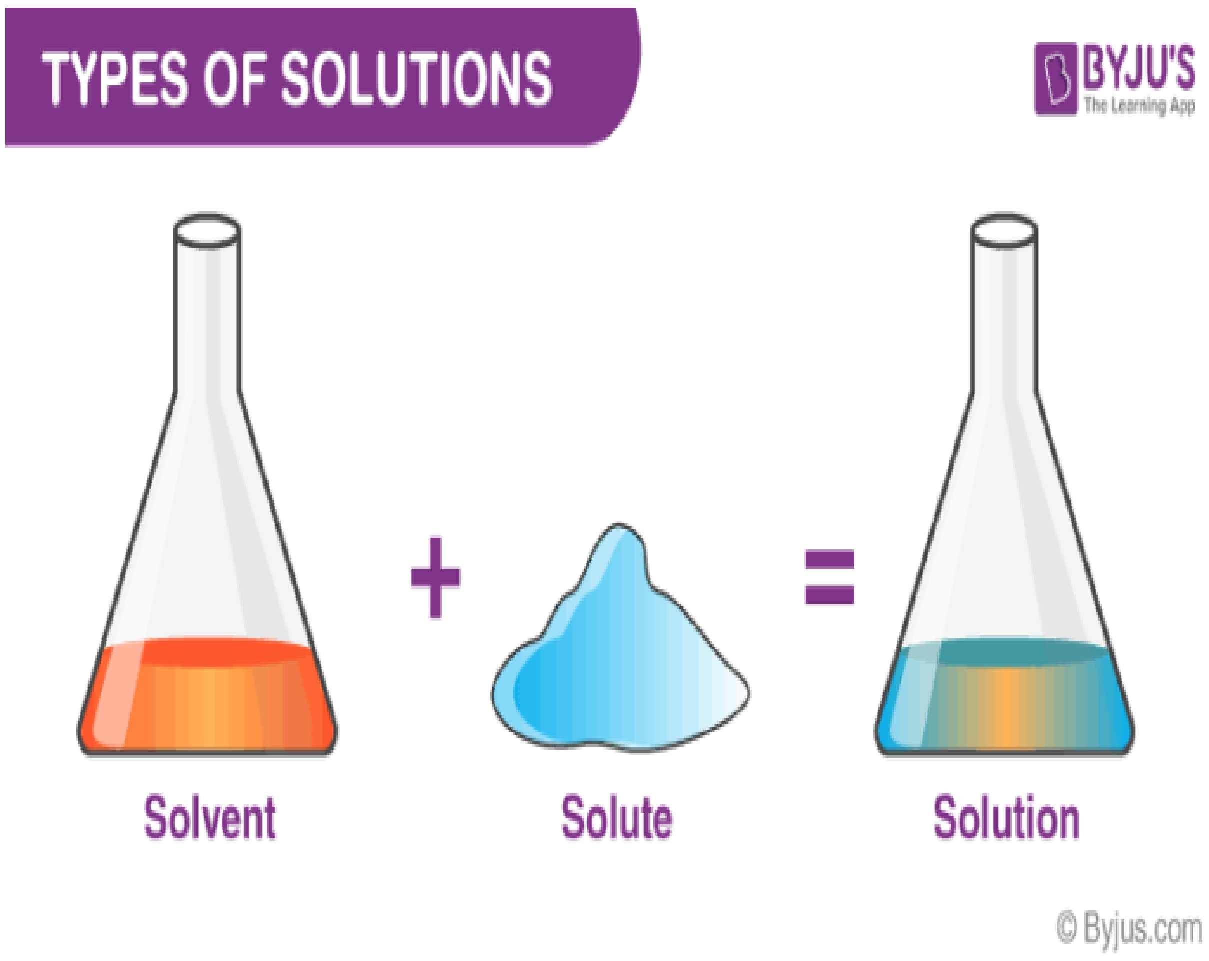
yK=(C1+C2 K)(-1+i)K+(C3 +C4 K)(-1-i)K

Homogeneous solution

Define Solution

a homogenous mixture which mainly comprises of two components namely solute and solvent.
For example, salt and sugar is a good illustration of a solution. A solution can be categorized into several components.

On the basis of physical states of solvent and solute can be categorized as solid, liquid and gaseous solutions.



In solid solutions, solute and solvent are in the solid-state. For example ceramics, and polymer blends. In liquid solutions, solid, gas or liquid is mixed in a liquid state. Gaseous solutions are usually homogenous mixtures of gases like air. Depending upon a number of solutions and solute, it can be classified into dilute and concentrated solutions.

Different Types of Solutions

Depending upon the dissolution of the solute in the solvent, solutions can be categorized into supersaturated solution, unsaturated and saturated solutions.

- A supersaturated solution comprises a large amount of solute at a temperature wherein it will be reduced as a result the extra solute will crystallize quickly.
- An unsaturated solution is a solution in which a solvent is capable of dissolving any more solute at a given temperature.

- A saturated solution can be defined as a solution in which a solvent is not capable of dissolving any more solute at a given temperature.

The solutions are of two forms, depending on whether the solvent is water or not.

- Aqueous solution – When a solute is dissolved in water the solution is called an aqueous solution. Eg, salt in water, sugar in water and copper sulfate in water.
- Non-aqueous solution – When a solute is dissolved in a solvent other than water, it is called a non-aqueous solution. Eg, iodine in carbon tetrachloride, sulphur in carbon disulfide, phosphorus in ethyl alcohol.

Solutions are spoken of as having two components, the solvent, and the solute. Another classification of the solution depends on the amount of solute added to the solvent.

- A dilute solution contains a small amount of solute in a large amount of solvent.
- A concentrated solution contains a large amount of solute dissolved in a small amount of solven

47,916

Mixtures

A mixture is composed of two or more substance, but they are not chemically combined. In contrast, the compound contains various elements that are bonded to each other. For instance, consider a mixture of salt that is when salt is dissolved in water it is a mixture but ideally, salts consist of two components namely sodium and chlorine.

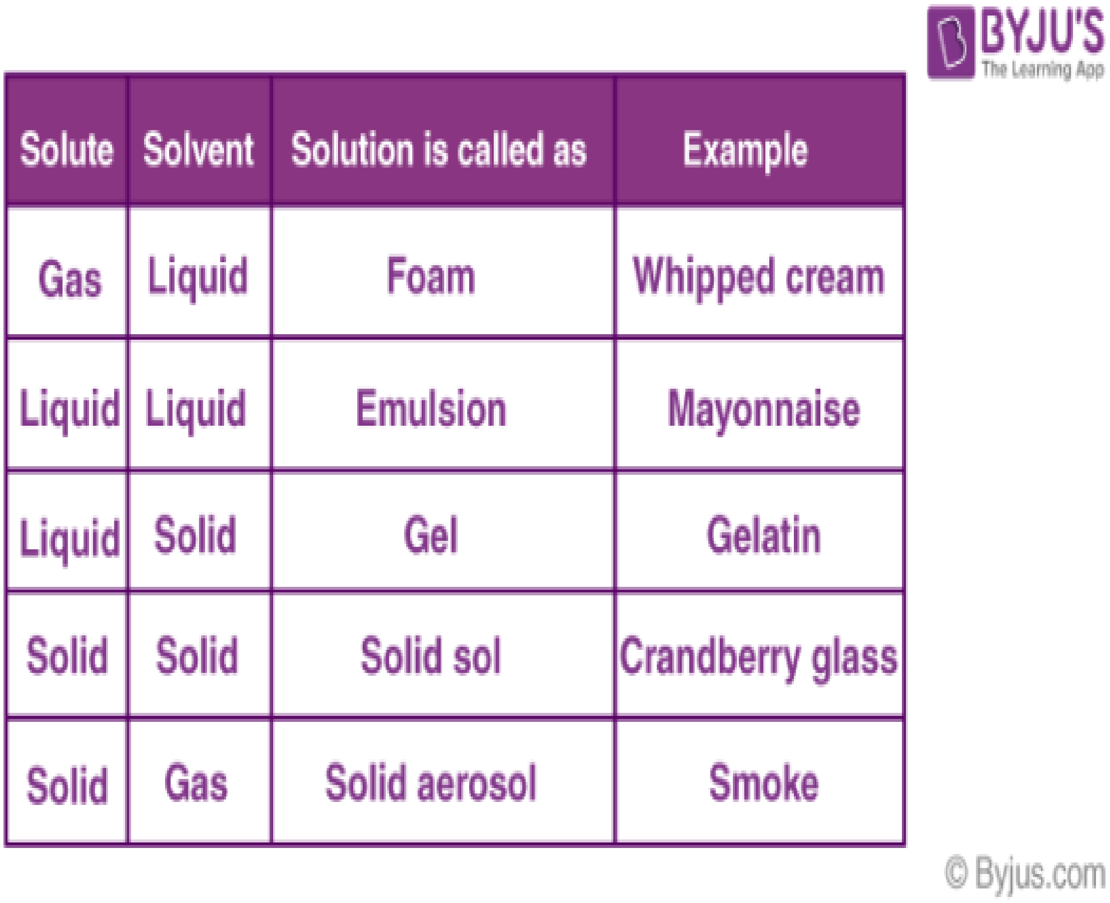
Here Sodium and Chlorine are bonded together with the electrostatic force of attraction to form sodium chloride even though there is no chemical bond between water and salt in the mixture. Hence, matter can be classified as mixtures, compounds, and elements. Further mixtures can be classified as homogeneous and heterogeneous mixtures.

Homogenous and Heterogeneous Solutions

Homogeneous solutions are solutions with uniform composition and properties throughout the solution. For example a cup of coffee, perfume, cough syrup, a solution of salt or sugar in water etc.

Heterogeneous solutions are solutions with non-uniform composition and properties throughout the solution. A solution of oil and water, water and chalk powder and solution of water and sand etc.

Examples



Solute	Solvent	Solution is called as	Example
Gas	Liquid	Foam	Whipped cream
Liquid	Liquid	Emulsion	Mayonnaise
Liquid	Solid	Gel	Gelatin
Solid	Solid	Solid sol	Crandberry glass
Solid	Gas	Solid aerosol	Smoke

© Byjus.com

Aerated drinks, Salt-water or Sugar water mixtures, fruit juices are some examples for solutions. Some solutions are heterogeneous in nature, and they are termed as suspension.

Such suspended particles can be seen quite clearly in the solution. Hence, when light is passed through such solutions, it scatters in different directions. Medicated syrups are one of the finest examples of this.

Frequently Asked Questions – FAQs

What is the true solution?

The True Solution is a homogeneous combination of two or more components immersed in a solvent with a particle size of less than 10-9 m or 1 nm. Example: The basic solution of sugar in water. By using philtre paper that is often not noticeable to the naked eye, particles can not be separated from real solutions.

What type of solution is vinegar?

Vinegar, which can contain flavourings, is an aqueous solution of acetic acid and trace chemicals. Usually, vinegar contains 5 to 8 percent acetic acid by amount. The fermentation of ethanol or sugars by acetic acid bacteria normally produces acetic acid.

What is solution and its type?

A solution is a combination of liquid and solute molecules that is homogeneous. Water, concrete, or gaseous may be a solution. In comparison, a combination of liquids , gases and solids may be a solution. In certain cases, the solution consists of a number of various kinds of solutes, such as salts, oxygen, and organic compounds, including seawater.

What are liquids two examples?

Examples of liquids at room temperatures include water, mercury, palm oil, ethanol. While francium, cesium, gallium, and rubidium liquefy at slightly elevated temperatures, Mercury is the only metallic element that is a liquid at room temperature.

What type of solution is formed when two liquids do not mix?

Whether two fluids can be combined to form a solution, they are considered “miscible.” Whether two fluids can not be mixed to form a solution, they are considered “immiscible.” Alcohol and water are an example of miscible fluids. Oil and water are an example of immiscible liquids.

Particular Solution

(a) Homogeneous Linear Difference Equations and Particular Solution:

We can find the particular solution of the difference equation when the equation is of homogeneous linear type by putting the values of the initial conditions in the homogeneous solutions.

Example1: Solve the difference equation 2ar-5ar-1+2ar-2=0 and find particular solutions such that a0=0 and a1=1.

Solution: The characteristics equation is 2s2-5s+2=0
(2s-1)(s-2)=0
s = $\frac{1}{2}$ and 2.

Therefore, the homogeneous solution of the equation is given by

ar(h)= C1$\left(\frac{1}{2}\right)^r$ +C2 .2r..........equation (i)

Putting r=0 and r=1 in equation (i), we get

$a0=C1+C2=0$...........equation (a)

$a1=\frac{1}{2}\ C1+2C2=1$...........equation.(b)

Solving eq (a) and (b), we have

$C1=-\frac{2}{3}$ and $C2=\frac{2}{3}$

Hence, the particular solution is

$$a_{r(P)}=-\frac{2}{3}.\left(\frac{1}{2}\right)^{r}+\frac{2}{3}.(2)^{r}$$

Example2: Solve the difference equation ar-4ar-1+4ar-2=0 and find particular solutions such that a0=0 and a1=6.

Solution: The characteristics equation is

$s2-4s+4=0$ or $(s-2)2=0$ $\qquad$ $s = 2, 2$

Therefore, the homogeneous solution of the equation is given by

$ar(n)=(C1+C2\ r).2r$.............. equation (i)

Putting r = 0 and r = 1 in equation (i), we get

$a0=(C1+0).20 = 1$ $\qquad$ $\therefore C1=1$

$a1=(C1+C2).2=6$ $\qquad$ $\therefore C1+C2=3\Rightarrow C2=2$

Hence, the particular solution is

$ar(P)=(1+2r).2r.$

Example3: Solve the difference equation 9ar-6ar-1+ar-2=0 satisfying the conditions a0=0 and a1=2.

Solution: The characteristics equation is

$9s2-6s+1=0$ or $(3s-1)2=0$

$s = \frac{1}{3},\frac{1}{3}$

Therefore, the homogeneous solution of the equation is given by

$ar(h)=(C1+C2\ r).\left(\frac{1}{3}\right)^{r}$ ....equation (i)

Putting r = 0 and r = 1 in equation (i), we get

$a0=C\_1=0$

$a1= (C1+C2).\frac{1}{3}=2.$ $\qquad$ $\therefore C1+C2=6\Rightarrow C2=6$

Hence, the particular solution is

$a_r(P)=6r.\left(\frac{1}{3}\right)^r$.

(b) Non-Homogeneous Linear Difference Equations and Particular Solution:

There are two methods to find the particular solution of a non-homogeneous linear difference equation. These are as follows:

1. Undetermined coefficients method
2. E and ∆ operator method.

1. Undetermined Coefficients Method: This method is used to find a particular solution of non-homogeneous linear difference equations, whose R.H.S term R (n) consist of terms of special forms.

In this method, firstly we assume the general form of the particular solutions according to the type of R (n) containing some unknown constant coefficients, which have to be determined. Then according to the difference equation, we will determine the exact solution.

The general form of a particular solution to be assumed for the special forms of R (n), to find the exact solution is shown in the table.

Form of R (n)	General form to be assumed
Z, here z is constant	A
Zr, here z is constant	Zr
P (r), a polynomial of degree n	A0 rn+A1 rn-1+⋯..An
Zr. P (r), here P(r) is a polynomial of the nth degree in r. Z is a constant.	[A0 rn+A1 rn-1+⋯..An].Zr

Example1: Find the particular solution of the difference equation ar+2-3ar+1+2ar=Zr ....equation (i)

Where Z is some constant.

Solution: The general form of solution is = A. Zr

Now putting this solution on L.H.S of equation (i), we get

= A Zr+2-3AZr+1+2AZr=(Z2-3Z+2) A Zr.........equation (ii)

Equating equation (ii) with R.H.S of equation (i), we get

(Z2-3Z+2)A=1

A = $\frac{1}{(Z^2 - 3Z + 2)} = \frac{1}{(Z-1)(Z-2)}$ (Z≠1, Z≠2)

Therefore, the particular solution is $\frac{Z^r}{(Z-1)(Z-2)}$

Example2: Find the particular solution of the difference equation ar+2-5ar+1+6ar=5r ....equation (i)

Solution: Let us assume the general form of the solution= A. 5r.

Now to find the value of A, put this solution on L.H.S of the equation (i), then this becomes

= A. 5r+2-5.A5r+1+6.A5r

= 25A. 5r-25A.5r+6A.5r

= 6A.5r ....equation (ii)

Equating equation (ii) to R.H.S of equation (i), we get

A =$\frac{1}{6}$

Therefore, the particular solution of the difference equation is =$\frac{1}{6}$.5r.

Example3: Find the particular solution of the difference equation ar+ 2+ar+1+ar=r.2r..........equation (i)

Solution: Let us assume the general form of the solution = (A0+A1r). 2^r

Now, put these solutions in the L.H.S of the equation (i), we get

= 2r+2 [A0+A1 (r+2)]+2r+1 [A0+A1 (r+1)]+2r (A0+A1 r)

= 4. 2r (A0+A1 r+2A1)+2.2r (A0+A1 r+A1)+2r (A0+A1 r)

= r. 2r (7A1)+2r (7A0+10A1)............equation (ii)

Equating equation (ii) with R.H.S of equation (i), we get

7A1=1 ∴ A1=$\frac{1}{7}$

7A0+10A1=0 ∴ A0=$\frac{-10}{49}$

Therefore, the particular solution is $2^r\left(\frac{-10}{49} + \frac{1}{7}r\right)$

2. E and ∆ operator Method:

Definition of Operator E: The operator of E on f(x) means that give an increment to the value of x in the function. The operation of E is, put (x+h) in the function wherever there is x. Here h is increment quantity. So $Ef(x) = f(x+h)$

Here, E is operated on f(x), therefore, E is a symbol known as shift operator.

Definition of OperatorΔ: The operation Δ is an operation of two steps.

Firstly, x in the function is incremented by a constant and then former is subtracted from the later i.e.,

$$\Delta f(x)=f(x+h)-f(x)$$

Theorem1: Prove that $E \cong 1+\Delta$.

Proof: The operation of Δ on f(x) is of two steps. First, increment the value of x in the function. So, whenever, there is x in f(x) put x+h (here h is constant increment), which means operation of E on f(x) i.e.,

$$f(x+h)=Ef(x).$$

Second, subtract the original function from the value obtained in the first step, hence

$$\Delta f(x)=Ef(x)-1f(x)=(E-1)f(x)$$

So, the operation of Δ on f(x) is equivalent to the operation of (E-1) on f(x).

Therefore, we have

$$E \cong 1+\Delta.$$

Theorem2: Show that $E^n f(x)=f(x+nh)$.

Proof: We know that $E f(x) =f (x+h)$

Now $E^n f(x)=E.E.E.E.........n \text{ times } f(x)$

$= E^{n-1} [E f(x)] = E^{n-1} f(x+h)$

$= E^{n-2} [E f(x+h)] = E^{n-2} f(x+2h)$

....

....

$= E f[x+ (n-1) h] = f(x+nh)$.

Theorem3: Show that $E Cf(x) = CE f(x)$

Proof: We know that $E C f(x) = C f(x+h) = CE f(x+h)$. Hence Proved.

There is no effect of the operation of E on any constant. Therefore, the operation of E on any constant will be equal to constant itself.

By E and Δ operator method, we will find the solution of

$$C_0 y_{n+r}+C_1 y_{n+r-1}+C_2 y_{n+r-2}+\cdots+C_n y_n=R(n) \text{..............equation (i)}$$

Equation (i) can be written as

$$C_0 E^r y_n+C_1 E^{r-1} y_n+C_2 E^{r-2} y_n+\cdots+C_n y_n=R(n)$$

$(C_0 E^r+C_1 E^{r-1}+C_2 E^{r-2}+\cdots+C_n) y_n=R(n)$

Putting $C_0 E^r+C_1 E^{r-1}+C_2 E^{r-2}+\cdots+C_n=P(E)$

So $P(E) y_n=R(n)$

$\therefore \quad y_n=\frac{R(n)}{P(E)}$................equation (ii)

To find the particular solution of (ii) for different forms of R (n), we have the following cases.

Case1: When R (n) is some constant A.

We know that, the operation of E on any constant will be equal to the constant itself i.e.,

$EA=A$

Therefore, $P(E) A = (C_0 E^r+C_1 E^{r-1}+C_2 E^{r-2}+\cdots+C_n)A$

$= (C_0+C_1+C_2+\cdots+C_n)A$

$= P(1) A$

$\frac{A}{P(E)} = \frac{1}{P(1)}.A$

Therefore, using equation (ii), the particular solution of (i) is

$y_n=\frac{A}{P(1)}, P(1)\neq 0$

P (1) is obtained by putting E = 1 in P (E).

Case2: When R (n) is of the form $A. Z^n$, where A and Z are constants

We have, $P(E)(A. Z^n)=\{C_0 E^r+C_1 E^{r-1}+\cdots+ C_n\}(A.Z^n)$

$=A\{C_0 Z^{r+n}+C_1 Z^{r+n-1}+\cdots+C_n Z^n\}$

$= A\{C_0 Z^r+C_1 Z^{r-1}+\cdots+C_n\}. Z^n$

$=AP(Z).Z^n$

To get, P (Z) put E=Z in P (E)

Therefore, $\frac{A.Z^n}{P(E)}=\frac{A.Z^n}{P(Z)}$, provided $P(Z) \neq 0$

Thus, $y_n=\frac{A.Z^n}{P(Z)}, P(Z) \neq 0$

If A = 1, then $y_n=\frac{Z^n}{P(Z)}$

When P (Z) = 0 then for equation

(i) $(E-Z) y_n= A. Z^n$

For this, the particular solution becomes A. $\frac{1}{(E-Z)}$ Zn=A. n Zn-1

(ii) (E-Z)2 yn= A. Zn

For this, the particular solution becomes $A.\frac{1}{(E-Z)^2}.Z^n=\frac{A.n(n-1)}{2!}.Z^{n-2}$

(iii) (E-Z)3 yn= A. Zn

For this, the particular solution becomes $A.\frac{1}{(E-Z)^3}.Z^n=\frac{A.n(n-1)(n-2)}{3!}.Z^{n-3}$ and so on.

Case3: When R (n) be a polynomial of degree m is n.

We know that E≅1+∆
So, P (E) =P (1+∆)

$$\frac{1}{P(E)}=\frac{1}{P(1+\Delta)}$$

Which can be expanded in ascending power of ∆ as far as upto ∆m

⇒ $\frac{1}{P(E)}=\frac{1}{P(1+\Delta)}$ =(b0+b1 ∆+b2 ∆+⋯.+bm ∆m+⋯)
⇒ $\frac{1}{P(E)}$.R(n)=(b0+b1 ∆+b2 ∆+⋯.+bm ∆m+⋯).R(n)
= b0 R(n)+b1 ∆ R(n)+⋯+bm ∆m R(n)

All other higher terms will be zero because R (n) is a polynomial of degree m.

Thus, the particular solution of equation (i), in this case will be

yn=b0 R(n)+b1 ∆ R(n)+⋯.+bm ∆m R(n).

Case4: When R (n) is of the form R(n).Zn,where R(n) is a polynomial of degree m and Z is some constant

We have Er[Zn R(n)]=Zr+n R (n+r)=Zr.Zn.Er.R(n)=Zn (ZE)rR(n)

Similarly, we have
$\frac{1}{P(E)}$ [Zn R(n)]=Zn $\frac{1}{P(ZE)}$.(R(n))= Zn [P(Z+Z∆)]-1.R(n)

Thus, the particular solution of equation (i), in this case will be
yn=Zn [P(Z+Z∆)]-1.R(n)

Example1: Find the particular solution of the difference equation
2ar+1-ar=12.

Solution: The above equation can be written as
(2E-1) ar=12

The particular solution is given by

ar= $\frac{1}{(2E-1)}$.12

Put E=1, in the equation. The particular solution is ar=12

Example2: Find the particular solution of the difference equation ar-4ar-1+4ar-2=2r.

Solution: The above equation can be written as
(E2-4E+4) ar=2r

Therefore, P (E) = E2-4E+4 = (E-2)2

Thus, the particular solution is given by

$$a_r = \frac{1}{(E-2)^2} . 2^r = \frac{r(r-1)}{2} . 2^{r-2}$$

$$a_r = r(r-1). 2^{r-3}$$

Total Solution

The total solution or the general solution of a non-homogeneous linear difference equation with constant coefficients is the sum of the homogeneous solution and a particular solution. If no initial conditions are given, obtain n linear equations in n unknowns and solve them, if possible to get total solutions.

If y(h) denotes the homogeneous solution of the recurrence relation and
y(p) indicates the particular solution of the recurrence relation then, the total solution or the general solution y of the recurrence relation is given by
y =y(h)+y(p).

Example: Solve the difference equation
ar-4ar-1+4ar-2=3r+2r...........equation (i)

Solution: The homogeneous solution of this equation is obtained by putting R.H.S equal to zero i.e.,
ar-4ar-1+4ar-2=0

The homogeneous solution is ar(h)= (C1+C2 r).2r

The equation (i) can be written as (E2-4E+4) ar=3r+2r

The particular solution is given as

$$a_{r(p)} = \frac{1}{(E^2-4E+4)}.(3r+2^r) = \frac{1}{(E-2)^2}.(3r) + \frac{1}{(E-2)^2}.2^r$$

$$= 3.\frac{1}{(1-\Delta)^2}(r) + \frac{r(r-1)}{2!}.2^{r-2} = 3(1-\Delta)^{-2}(r) + \frac{r(r-1)}{2!}.2^{r-2}$$

$$=3(1+2\Delta).[r]+\frac{r(r-1)}{2!}.2^{r-2}=3(r+2)+r(r-1).2^{r-3}$$

$$a_{r(p)} = 3(r+2)+r(r-1).2^{r-3}$$

Therefore, the total solution is $a_r = (C_1 + C_2 r).2^r + 3(r+2) + r(r-1).2^{r-3}$

Generating Functions

There is an extremely powerful tool in discrete mathematics used to manipulate sequences called the generating function. The idea is this: instead of an infinite sequence (for example: 2,3,5,8,12,…2,3,5,8,12,…) we look at a single function which encodes the sequence. But not a function which gives the nnth term as output. Instead, a function whose power series (like from calculus) “displays” the terms of the sequence. So for example, we would look at the power series 2+3x+5x2+8x3+12x4+⋯2+3x+5x2+8x3+12x4+⋯ which displays the sequence 2,3,5,8,12,…2,3,5,8,12,… as coefficients.
An infinite power series is simply an infinite sum of terms of the form cnxncnxn were cncn is some constant. So we might write a power series like this:
∞∑k=0ckxk.∑k=0∞ckxk.

or expanded like this
c0+c1x+c2x2+c3x3+c4x4+c5x5+⋯.c0+c1x+c2x2+c3x3+c4x4+c5x5+⋯.

When viewed in the context of generating functions, we call such a power series a generating series. The generating series generates the sequence
c0,c1,c2,c3,c4,c5,….c0,c1,c2,c3,c4,c5,….

In other words, the sequence generated by a generating series is simply the sequence of coefficients of the infinite polynomial.
Example5.1.1
What sequence is represented by the generating series 3+8x2+x3+x57+100x6+⋯?3+8x2+x3+x57+100x6+⋯?
Solution

Now you might very naturally ask why we would do such a thing. One reason is that encoding a sequence with a power series helps us keep track of which term is which in the sequence. For example, if we write the sequence 1,3,4,6,9,…,24,41,…1,3,4,6,9,…,24,41,… it is impossible to determine which term 2424 is (even if we agreed that the first term was supposed to be a0a0). However, if we wrote the generating series instead, we would have 1+3x+4x2+6x3+9x4+⋯+24x17+41x18+⋯.1+3x+4x2+6x3+9x4+⋯+24x17+41x18+⋯. Now it is clear that 24 is the 17th term of the sequence (that is, a17=24a17=24). Of course to get this benefit we could have displayed our sequence in any number of ways, perhaps $\boxed{1}0\boxed{3}1\boxed{4}2\boxed{6}3\boxed{9}4\cdots\boxed{24}17\boxed{41}18\cdots$,1031426394⋯24174118⋯, but we do not do this. The reason is that the generating series looks like an ordinary power series (although we are interpreting it differently) so we can do things with it that we ordinarily do with power series such as write down what it converges to.
For example, from calculus we know that the power series 1+x+x22+x36+x424+⋯+xnn!+⋯1+x+x22+x36+x424+⋯+xnn!+⋯ converges to the function ex.ex. So we can use exex as a way of talking about the sequence of coefficients of the power series for ex.ex. When we write down a nice compact function which has an infinite power series that we view as a generating series, then we call that function a generating function. In this example, we would say 1,1,12,16,124,…,1n!,… has generating function ex1,1,12,16,124,…,1n!,… has generating function ex

Building Generating Functions

The exex example is very specific. We have a rather odd sequence, and the only reason we know its generating function is because we happen to know the Taylor series for ex.ex. Our goal now is to gather some tools to build the generating function of a particular given sequence.
Let's see what the generating functions are for some very simple sequences. The simplest of all: 1, 1, 1, 1, 1, …. What does the generating series look like? It is simply 1+x+x2+x3+x4+⋯.1+x+x2+x3+x4+⋯. Now, can we find a closed formula for this power series? Yes! This particular series is really just a geometric series with common ratio x.x. So if we use our “multiply, shift and subtract” technique from Section 2.2, we have
S=1+x+x2+x3+⋯−xS−−=   x+x2+x3+x4+⋯−――――――――――(1−x)S=1S=1+x+x2+x3+⋯−xS_=   x+x2+x3+x4+⋯_(1−x)S=1

Therefore we see that
1+x+x2+x3⋯=11−x1+x+x2+x3⋯=11−x

You might remember from calculus that this is only true on the interval of convergence for the power series, in this case when |x|<1.|x|<1. That is true for us, but we don't care. We are never going to plug anything in for x,x, so as long as there is some value of xx for which the generating function and generating series agree, we are happy. And in this case we are happy.
1,1,1,…1,1,1,…
The generating function for 1,1,1,1,1,1,…1,1,1,1,1,1,… is 11−x11−x

Let's use this basic generating function to find generating functions for more sequences. What if we replace xx by −x.−x. We get
11+x=1−x+x2−x3+⋯ which generates 1,−1,1,−1,…11+x=1−x+x2−x3+⋯ which generates 1,−1,1,−1,…

If we replace xx by 3x3x we get
11−3x=1+3x+9x2+27x3+⋯ which
generates 1,3,9,27,…11−3x=1+3x+9x2+27x3+⋯ which generates 1,3,9,27,…

By replacing the xx in 11−x11−x we can get generating functions for a variety of sequences, but not all. For example, you cannot plug in anything for xx to get the generating function for 2,2,2,2,….2,2,2,2,…. However, we are not lost yet. Notice that each term of 2,2,2,2,…2,2,2,2,… is the result of multiplying the terms of 1,1,1,1,…1,1,1,1,… by the constant 2. So multiply the generating function by 2 as well.
21−x=2+2x+2x2+2x3+⋯ which generates 2,2,2,2,…21−x=2+2x+2x2+2x3+⋯ which generates 2,2,2,2,…

Similarly, to find the generating function for the sequence 3,9,27,81,…,3,9,27,81,…, we note that this sequence is the result of multiplying each term of 1,3,9,27,…1,3,9,27,… by 3. Since we have the generating function for 1,3,9,27,…1,3,9,27,… we can say
31−3x=3⋅1+3⋅3x+3⋅9x2+3⋅27x3+⋯ which
generates 3,9,27,81,…31−3x=3⋅1+3⋅3x+3⋅9x2+3⋅27x3+⋯ which
generates 3,9,27,81,…

What about the sequence 2,4,10,28,82,…?2,4,10,28,82,…? Here the terms are always 1 more than powers of 3. That is, we have added the sequences 1,1,1,1,…1,1,1,1,… and 1,3,9,27,…1,3,9,27,… term by term. Therefore we can get a generating function by adding the respective generating functions:
2+4x+10x2+28x3+⋯=(1+1)+(1+3)x+(1+9)x2+(1+27)x3+⋯=1+x+x2+x3+⋯+1+3x+9x2+27x3+⋯=11−x+11−3x2+4x+10x2+28x3+⋯=(1+1)+(1+3)x+(1+9)x2+(1+27)x3+⋯=1+x+x2+x3+⋯+1+3x+9x2+27x3+⋯=11−x+11−3x

The fun does not stop there: if we replace xx in our original generating function by x2x2 we get
11−x2=1+x2+x4+x6⋯ which generates 1,0,1,0,1,0,….11−x2=1+x2+x4+x6⋯ which generates 1,0,1,0,1,0,….

How could we get 0,1,0,1,0,1,…?0,1,0,1,0,1,…? Start with the previous sequence and shift it over by 1. But how do you do this? To see how shifting works, let's first try to get the generating function for the sequence 0,1,3,9,27,….0,1,3,9,27,…. We know that 11−3x=1+3x+9x2+27x3+⋯.11−3x=1+3x+9x2+27x3+⋯. To get the zero out front, we need the generating series to look like x+3x2+9x3+27x4+⋯x+3x2+9x3+27x4+⋯ (so there is no constant term). Multiplying by xx has this effect. So the generating function for 0,1,3,9,27,…0,1,3,9,27,… is x1−3x.x1−3x. This will also work to get the generating function for 0,1,0,1,0,1,…:0,1,0,1,0,1,…:
x1−x2=x+x3+x5+⋯ which generates 0,1,0,1,0,1…x1−x2=x+x3+x5+⋯ which generates 0,1,0,1,0,1…

What if we add the
sequences 1,0,1,0,1,0,…1,0,1,0,1,0,… and 0,1,0,1,0,1,…0,1,0,1,0,1,… term by term? We should get 1,1,1,1,1,1….1,1,1,1,1,1…. What happens when we add the generating functions? It works (try it)!
11−x2+x1−x2=11−x.11−x2+x1−x2=11−x.

Here's a sneaky one: what happens if you take the derivative of 11−x?11−x? We get 1(1−x)2.1(1−x)2. On the other hand, if we differentiate term by term in the power series, we
get (1+x+x2+x3+⋯)′=1+2x+3x2+4x3+⋯(1+x+x2+x3+⋯)′=1+2x+3x2+4x3+⋯ which is the generating series for 1,2,3,4,….1,2,3,4,…. This says
1,2,3,…1,2,3,…
The generating function for 1,2,3,4,5,…1,2,3,4,5,… is 1(1−x)2.1(1−x)2.
Take a second
derivative: 2(1−x)3=2+6x+12x2+20x3+⋯.2(1−x)3=2+6x+12x2+20x3+⋯. So 1(1−x)3= 1+3x+6x2+10x3+⋯1(1−x)3=1+3x+6x2+10x3+⋯ is a generating function for the triangular numbers, 1,3,6,10…1,3,6,10… (although here we
have a0=1a0=1 while T0=0T0=0 usually).

Differencing

We have seen how to find generating functions from 11−x11−x using multiplication (by a constant or by xx), substitution, addition, and differentiation. To use each of these, you must notice a way to transform the sequence 1,1,1,1,1…1,1,1,1,1… into your desired sequence. This is not always easy. It is also not really the way we have analyzed sequences. One thing we have considered often is the sequence of differences between terms of a sequence. This will turn out to be helpful in finding generating functions as well. The sequence of differences is often simpler than the original sequence. So if we know a generating function for the differences, we would like to use this to find a generating function for the original sequence.
For example, consider the sequence 2,4,10,28,82,….2,4,10,28,82,…. How could we move to the sequence of first differences: 2,6,18,54,…?2,6,18,54,…? We want to subtract 2 from the 4, 4 from the 10, 10 from the 28, and so on. So if we subtract (term by term) the
sequence 0,2,4,10,28,…0,2,4,10,28,… from 2,4,10,28…,2,4,10,28…, we will be set. We can get the generating function for 0,2,4,10,28,…0,2,4,10,28,… from the generating function for 2,4,10,28…2,4,10,28… by multiplying by x.x. Use AA to represent the generating function for 2,4,10,28,82,…2,4,10,28,82,… Then:
A=2+4x+10x2+28x3+82x4+⋯−xA−—=0+2x+4x2+10x3+28x4+82x5+⋯−——————
——————(1−x)A=2+2x+6x2+18x3+54x4+⋯A=2+4x+10x2+28x3+82x4+⋯−xA_=0 +2x+4x2+10x3+28x4+82x5+⋯_(1−x)A=2+2x+6x2+18x3+54x4+⋯

While we don't get exactly the sequence of differences, we do get something close. In this particular case, we already know the generating function AA (we found it in the previous section) but most of the time we will use this differencing technique to find A:A: if we have the generating function for the sequence of differences, we can then solve for A.A.
Example5.1.2
Find a generating function for 1,3,5,7,9,….1,3,5,7,9,….
Solution

Now that we have a generating function for the odd numbers, we can use that to find the generating function for the squares:
Example5.1.3
Find the generating function for 1,4,9,16,….1,4,9,16,…. Note we take 1=a0.1=a0.
Solution

In each of the examples above, we found the difference between consecutive terms which gave us a sequence of differences for which we knew a generating function. We can generalize this to more complicated relationships between terms of the sequence. For example, if we know that the sequence satisfies the recurrence relation an=3an−1−2an−2?an=3an−1−2an−2? In other words, if we take a term of the sequence and subtract 3 times the previous term and then add 2 times the term before that, we get 0 (since an−3an−1+2an−2=0an−3an−1+2an−2=0). That will hold for all but the first two terms of the sequence. So after the first two terms, the sequence of results of these calculations would be a sequence of 0's, for which we definitely know a generating function.
Example5.1.4
The sequence 1,3,7,15,31,63,…1,3,7,15,31,63,… satisfies the recurrence relation an=3an−1−2an−2.an=3an−1−2an−2. Find the generating function for the sequence.
Solution

Multiplication and Partial Sums

What happens to the sequences when you multiply two generating functions? Let's see: A=a0+a1x+a2x2+⋯A=a0+a1x+a2x2+⋯ and B=b0+b1x+b2x2+⋯.B=b0+b1x+b2 x2+⋯. To multiply AA and B,B, we need to do a lot of distributing (infinite FOIL?) but keep in mind we will group like terms and only need to write down the first few terms to see the pattern. The constant term is a0b0.a0b0. The coefficient of xx is a0b1+a1b0.a0b1+a1b0. And so on. We get:
AB=a0b0+(a0b1+a1b0)x+(a0b2+a1b1+a2b0)x2+(a0b3+a1b2+a2b1+a3b0)x3+⋯AB=a0b0+(a0b1+a1b0)x+(a0b2+a1b1+a2b0)x2+(a0b3+a1b2+a2b1+a3b0)x3+⋯

Example5.1.5
“Multiply” the sequence 1,2,3,4,…1,2,3,4,… by the sequence 1,2,4,8,16,….1,2,4,8,16,….
Solution

Consider the special case when you multiply a sequence by 1,1,1,….1,1,1,…. For example, multiply 1,1,1,…1,1,1,… by 1,2,3,4,5….1,2,3,4,5…. The first term is 1·1=1.1·1=1. Then 1·2+1·1=3.1·2+1·1=3. Then 1·3+1·2+1·1=6.1·3+1·2+1·1=6. The next term will be 10. We are getting the triangular numbers. More precisely, we get the sequence of partial sums of 1,2,3,4,5,….1,2,3,4,5,…. In terms of generating functions, we take 11−x11−x (generating 1,1,1,1,1…1,1,1,1,1…) and multiply it by 1(1−x)21(1−x)2 (generating 1,2,3,4,5,…1,2,3,4,5,…) and this give 1(1−x)3.1(1−x)3. This should not be a surprise as we found the same generating function for the triangular numbers earlier.
The point is, if you need to find a generating function for the sum of the first nn terms of a particular sequence, and you know the generating function for that sequence, you can multiply it by 11−x.11−x. To go back from the sequence of partial sums to

the original sequence, you look at the sequence of differences. When you get the sequence of differences you end up multiplying by 1−x,1−x, or equivalently, dividing by 11−x.11−x. Multiplying by 11−x11−x gives partial sums, dividing
by 11−x11−x gives differences.

Solving Recurrence Relations with Generating Functions

We conclude with an example of one of the many reasons studying generating functions is helpful. We can use generating functions to solve recurrence relations.
Example5.1.6
Solve the recurrence relation an=3an−1−2an−2an=3an−1−2an−2 with initial conditions a0=1a0=1 and a1=3.a1=3.
Solution

We can now add generating functions to our list of methods for solving recurrence relations.

Exercises

Find the generating function for each of the following sequences by relating them back to a sequence with known generating function.

a. 4,4,4,4,4,….4,4,4,4,4,….
b. 2,4,6,8,10,….2,4,6,8,10,….
c. 0,0,0,2,4,6,8,10,….0,0,0,2,4,6,8,10,….
d. 1,5,25,125,….1,5,25,125,….
e. 1,−3,9,−27,81,….1,−3,9,−27,81,….
f. 1,0,5,0,25,0,125,0,….1,0,5,0,25,0,125,0,….
g. 0,1,0,0,2,0,0,3,0,0,4,0,0,5,….0,1,0,0,2,0,0,3,0,0,4,0,0,5,….

Solution

Find the sequence generated by the following generating functions:

a. 4x1−x.4x1−x.

b. 11−4x.11−4x.

c. x1+x.x1+x.

d. 3x(1+x)2.3x(1+x)2.

e. 1+x+x2(1−x)21+x+x2(1−x)2 (Hint: multiplication).

Solution

Show how you can get the generating function for the triangular numbers in three different ways:

a. Take two derivatives of the generating function for 1,1,1,1,1,…1,1,1,1,1,…

b. Use differencing.

c. Multiply two known generating functions.

Solution

Use differencing to find the generating function for 4,5,7,10,14,19,25,….4,5,7,10,14,19,25,….

Solution

Find a generating function for the sequence with recurrence relation an=3an−1−an−2an=3an−1−an−2 with initial terms a0=1a0=1 and a1=5.a1=5.

Solution

Use the recurrence relation for the Fibonacci numbers to find the generating function for the Fibonacci sequence.

Solution

Use multiplication to find the generating function for the sequence of partial sums of Fibonacci numbers, S0,S1,S2,…S0,S1,S2,… where S0=F0,S0=F0, S1=F0+F1,S1=F0+F1, S2=F0+F1+F2,S2=F0+F1+F2, S3=F0+F1+F2+F3S3=F0+F1+F2+F3 and so on.

Solution

Find the generating function for the sequence with closed formula an=2(5n)+7(−3)n.an=2(5n)+7(−3)n.

Solution

Find a closed formula for the nnth term of the sequence with generating function 3x1−4x+11−x.3x1−4x+11−x.

Solution

Find a7a7 for the sequence with generating function 2(1−x)2·x1−x−x2.2(1−x)2·x1−x−x2.

Solution

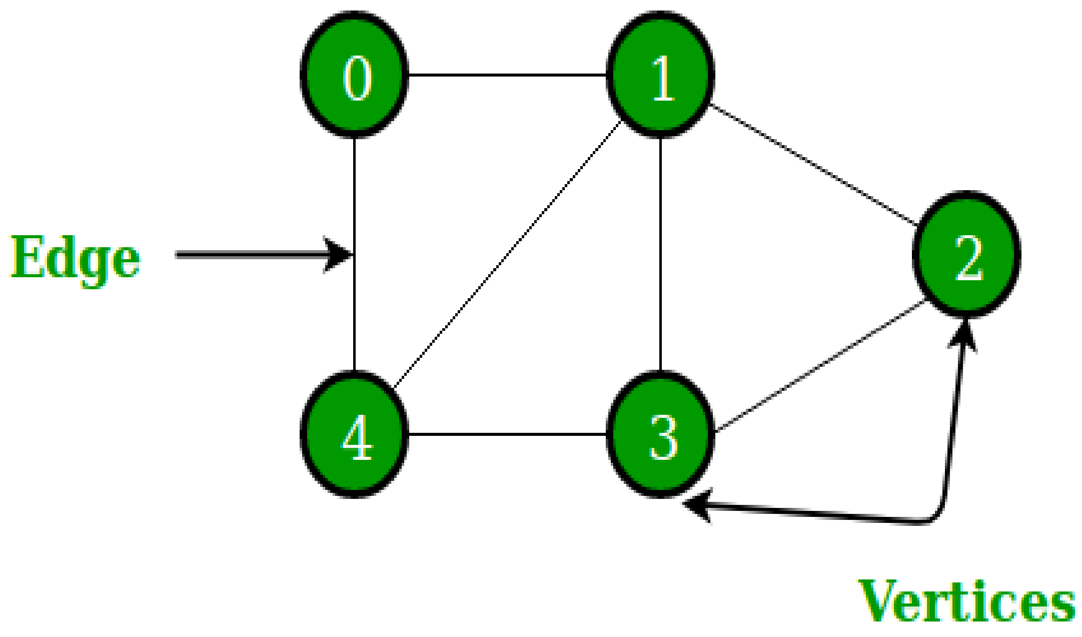
Explain how we know that 1(1−x)21(1−x)2 is the generating function for 1,2,3,4,….1,2,3,4,….

Unit III

Graph

A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph can be defined as,

A Graph consists of a finite set of vertices(or nodes) and set of Edges which connect a pair of nodes.



In the above Graph, the set of vertices V = {0,1,2,3,4} and the set of edges E = {01, 12, 23, 34, 04, 14, 13}.

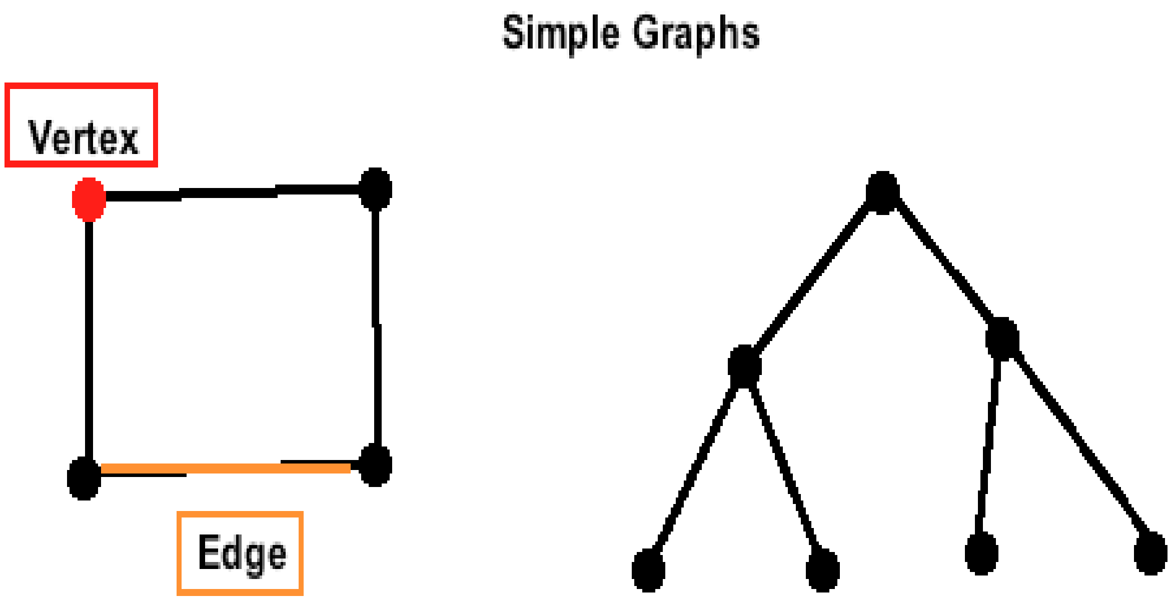
Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, locale etc.

Graphs and Subgraphs

Simple Graphs

We will first define the most fundamental of graphs, a simple graph:

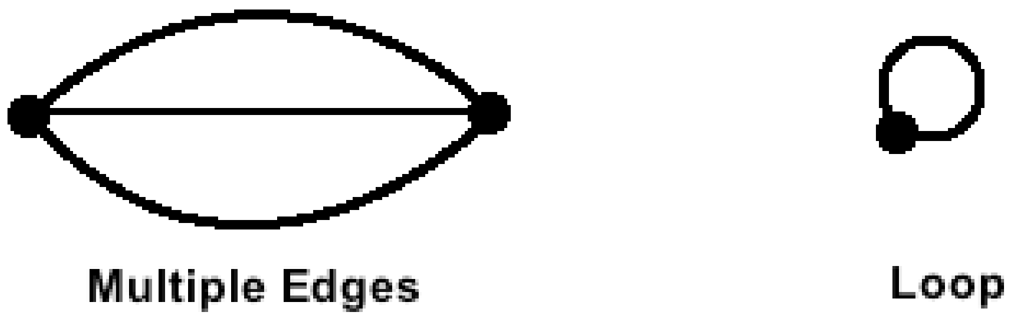
We will graphically denote a vertex with a little dot or some shape, while we will denote edges with a line connecting two vertices. Note that these edges do not need to be straight like the conventional geometric interpretation of an edge. For example, the following graphs are simple graphs.



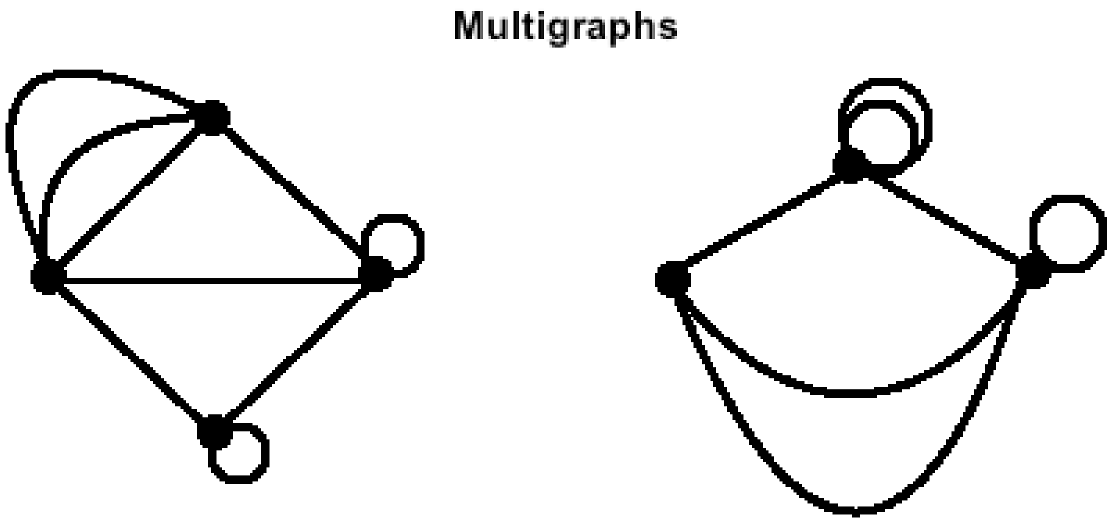
Multigraphs

Definition: A Multigraph is a graph G that contains either multiple edges or loops.

We consider a multiple edge to be to lines from a vertex x to a vertex y. One the other hand, we consider a loop to be an edge that wraps around back to itself.



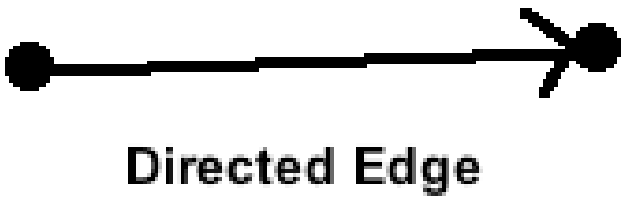
The following graphs are multigraphs:



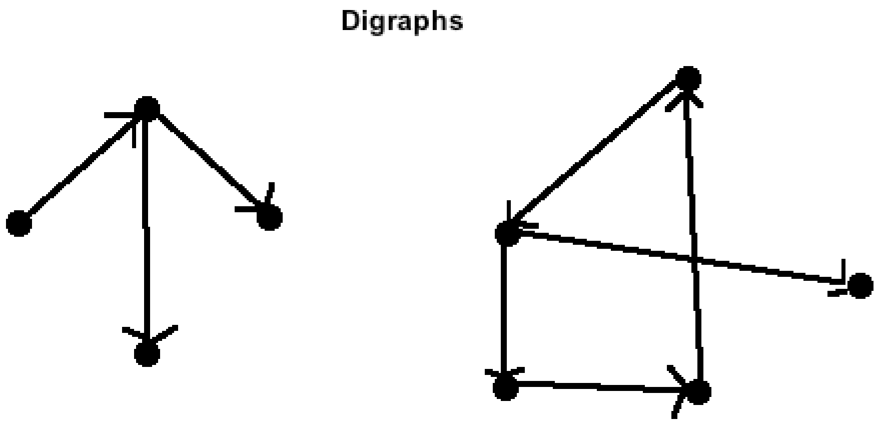
Digraphs (Directed Graphs)

Definition: A Digraph is a graph D that contains directed edges (also called arcs) instead of regular edges.

We note that a directed edge (or arc) contains some sort of direction from one vertex to another:



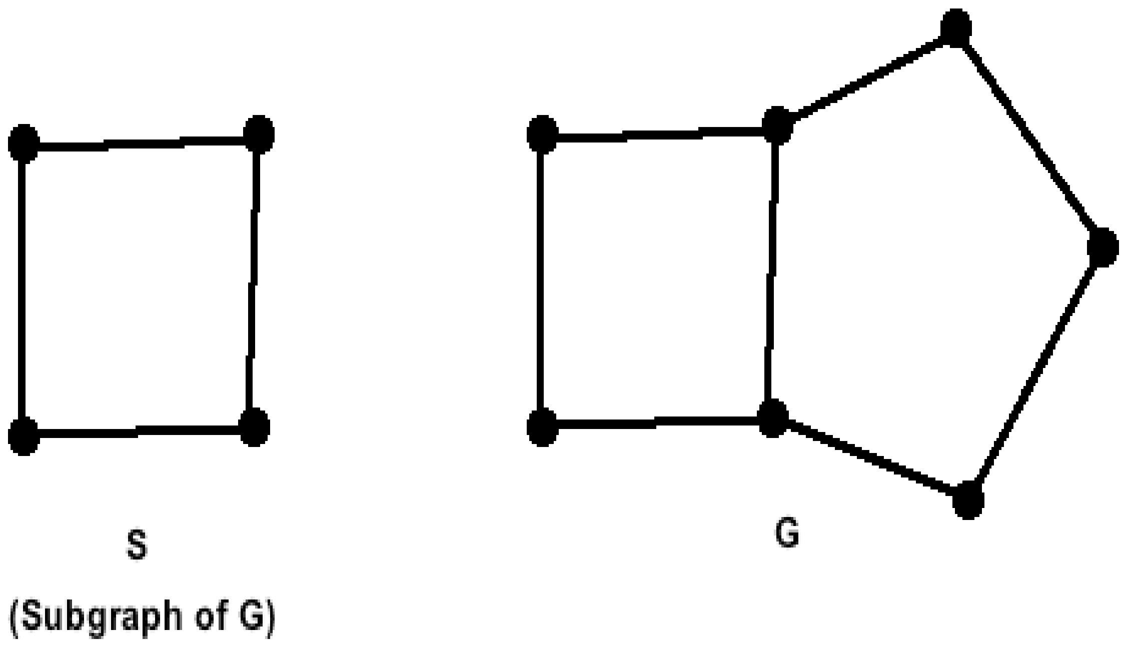
The following graphs are digraphs:



Subgraphs

Definition: A Subgraph S of a graph G is a graph whose vertex set V(S) is a subset of the vertex set V(G), that is V(S)⊆V(G), and whose edge set E(S) is a subset of the edge set E(G), that is E(S)⊆E(G).

Essentially, a subgraph is a graph within a larger graph. For example, the following graph S is a subgraph of G:



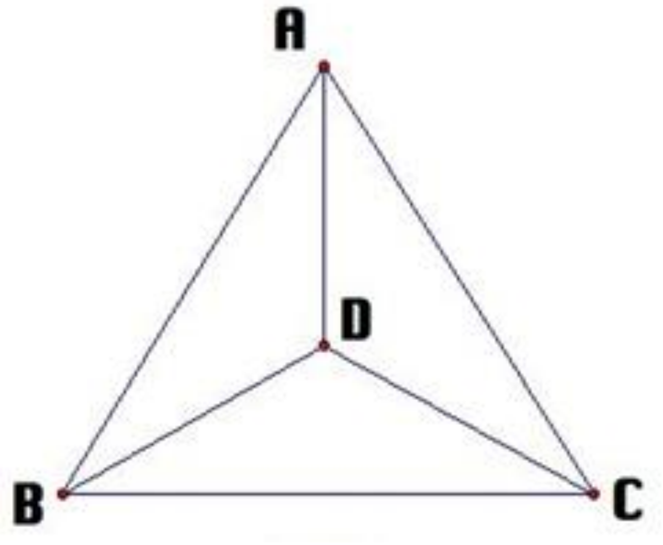
Basic Properties of Graph Theory

Properties of graph theory are basically used for characterization of graphs depending on the structures of the graph. Following are some basic properties of graph theory:

1 Distance between two vertices

Distance is basically the number of edges in a shortest path between vertex X and vertex Y. If there are many paths connecting two vertices, then the shortest path is considered as the distance between the two vertices. Distance between two vertices is denoted by d(X, Y).

Example



Suppose, we want to find the distance between vertex B and D, then first of all we have to find the shortest path between vertex B and D.

There are many paths from vertex B to vertex D:

- B -> C -> A -> D, length = 3
- B -> D, length = 1(Shortest Path)
- B -> A -> D, length = 2
- B -> C -> D, length = 2
- B -> C -> A -> D, length = 3

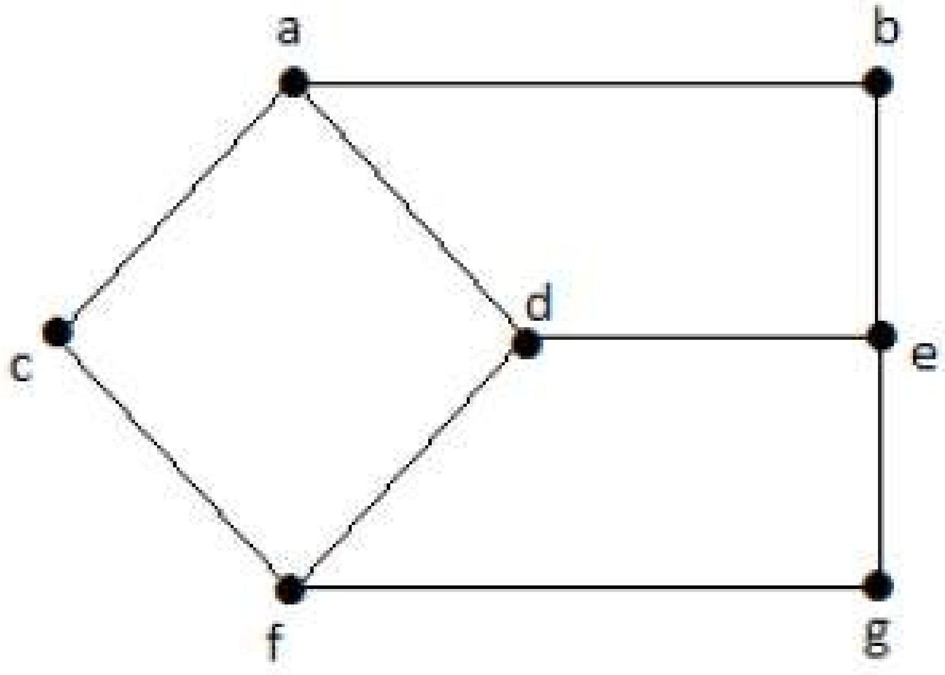
Hence, the minimum distance between vertex B and vertex D is 1.

2. Eccentricity of a vertex

Eccentricity of a vertex is the maximum distance between a vertex to all other vertices. It is denoted by e(V).

To count the eccentricity of vertex, we have to find the distance from a vertex to all other vertices and the highest distance is the eccentricity of that particular vertex.

Example



In the above example, if we want to find the maximum eccentricity of vertex 'a' then:

- The distance from vertex a to b is 1 (i.e. ab)
- The distance from vertex a to c is 1 (i.e. ac)
- The distance from vertex a to f is 2 (i.e. ac -> cf or ad -> df)
- The distance from vertex a to d is 1 (i.e. ad)
- The distance from vertex a to e is 2 (i.e. ab -> be or ad -> de)
- The distance from vertex a to g is 3 (i.e. ab -> be -> eg or ac -> cf -> fg etc.)

Hence, the maximum eccentricity of vertex 'a' is 3, which is a maximum distance from vertex ?a? to all other vertices.

Similarly, maximum eccentricities of other vertices of the given graph are:

- e(b) = 3
- e(c) = 3
- e(d) = 2
- e(e) = 3
- e(f) = 3
- e(g) = 3

3. Radius of connected Graph

The radius of a connected graph is the minimum eccentricity from all the vertices. In other words, the minimum among all the distances between a vertex to all other vertices is called as the radius of the graph. It is denoted by r(G).

Example

From the example of 5.2, r(G) = 2, which is the minimum eccentricity for the vertex 'd'.

4. Diameter of a Graph

Diameter of a graph is the maximum eccentricity from all the vertices. In other words, the maximum among all the distances between a vertex to all other vertices is considered as the diameter of the graph G. It is denoted by d(G).

Example

From the above example, if we see all the eccentricities of the vertices in a graph, we will see that the diameter of the graph is the maximum of all those eccentricities.

Diameter of graph d(G) = 3, which is the maximum eccentricity.

5. Central point

If the eccentricity of the graph is equal to its radius, then it is known as central point of the graph.

Or

If r(V) = e(V), then V is the central point of the graph G.

Example

From the above example, 'd' is the central point of the graph. i.e.

1. e(d) = r(d) = 2

6. Centre

The set of all the central point of the graph is known as centre of the graph.

Example

From the example of 5.2, {'d'} is the centre of the graph.

7. Circumference

The total number of edges in the longest cycle of graph G is known as the circumference of G.

Example

In the above example, the circumference is 6, which is derived from the longest path a -> c -> f -> g -> e -> b -> a or a -> c -> f -> d -> e -> b -> a.

8. Girth

The total number of edges in the shortest cycle of graph G is known as girth. It is denoted by g(G).

Example

In the above example, the girth of the graph is 4, which is derived from the shortest cycle a -> c -> f -> d -> a, d -> f -> g -> e -> d or a -> b -> e -> d -> a.

9. Sum of degrees of vertices Theorem

For non-directed graph G = (V,E) where, Vertex set V = {V1, V2, ... Vn} then,

$$\sum_{i=1}^{n} \deg(V_i) = 2|E|$$

In other words, for any graph, the sum of degrees of vertices equals twice the number of edges.

Corollary 1

For directed graph G = (V, E) where, Vertex Set V = {V1, V2, ... Vn} then,

$$\sum_{i=1}^{n} \deg^{+}(V_i) = |E| = \sum_{i=1}^{n} \deg^{-}(V_i)$$

Corollary 2

The number of vertices in any non- directed graph with odd degree is even.

Walks Path & circuits

Mathematics | Walks, Trails, Paths, Cycles and Circuits in Graph

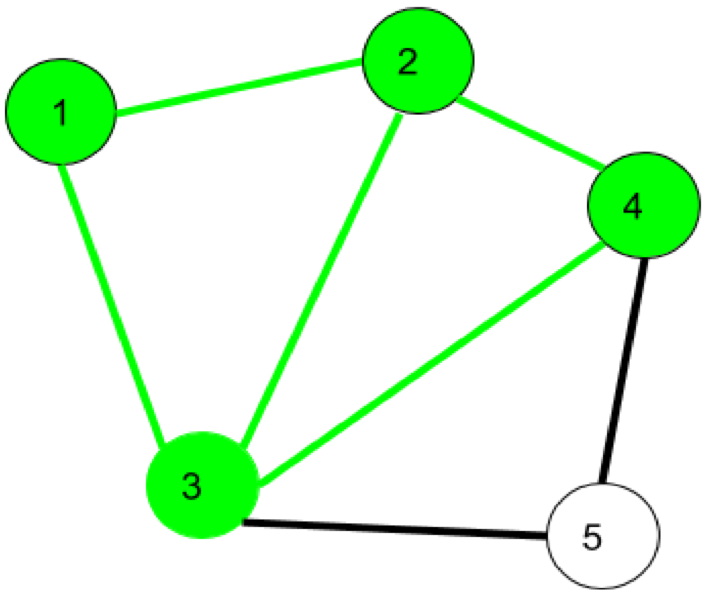
Last Updated: 21-05-2020

Prerequisite – Graph Theory Basics – Set 1

1. Walk –

A walk is a sequence of vertices and edges of a graph i.e. if we traverse a graph then we get a walk.

Vertex can be repeated
Edges can be repeated



Here 1->2->3->4->2->1->3 is a walk

Walk can be open or closed.

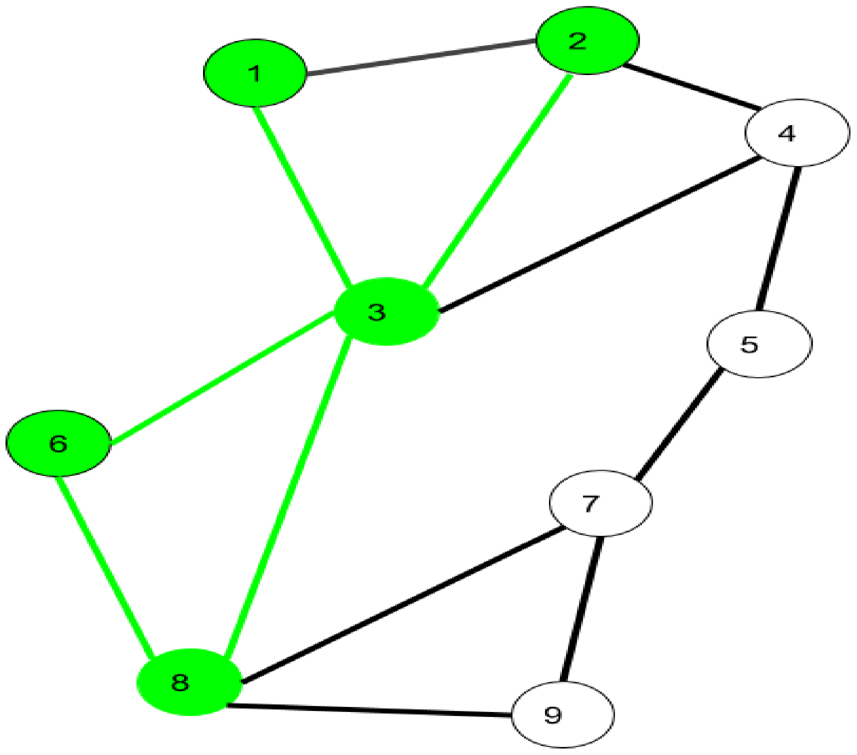
Walk can repeat anything (edges or vertices).

Open walk-A walk is said to be an open walk if the starting and ending vertices are different i.e. the origin vertex and terminal vertex are different.
Closed walk-A walk is said to be a closed walk if the starting and ending vertices are identical i.e. if a walk starts and ends at the same vertex, then it is said to be a closed walk.

In the above diagram:
1->2->3->4->5->3-> is an open walk.
1->2->3->4->5->3->1-> is a closed walk.

2. Trail –

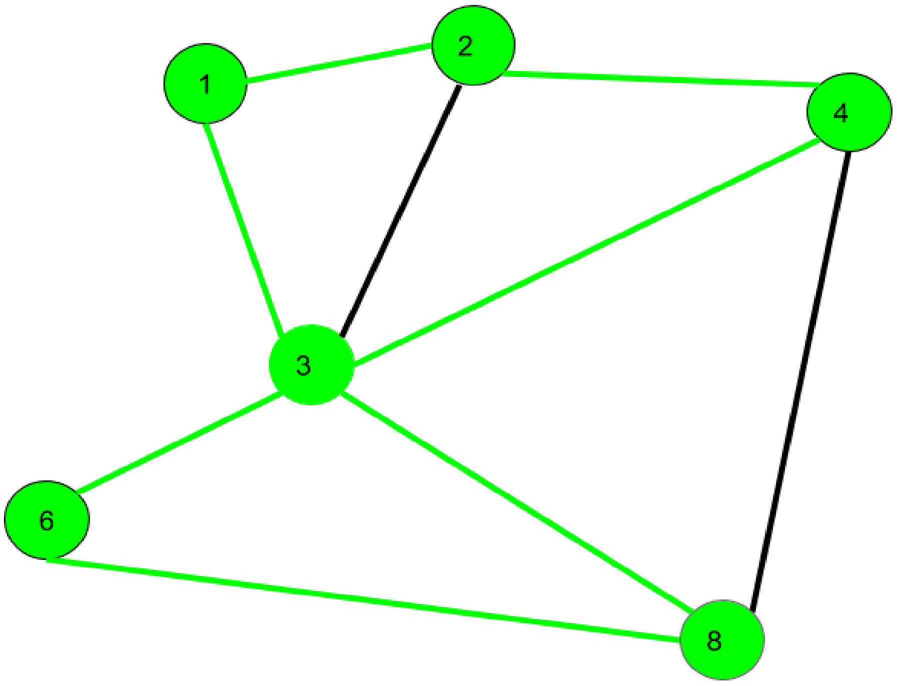
Trail is an open walk in which no edge is repeated.
Vertex can be repeated



Here 1->3->8->6->3->2 is trail
Also 1->3->8->6->3->2->1 will be a closed trail

3. Circuit –

Traversing a graph such that not an edge is repeated but vertex can be repeated and it is closed also i.e. it is a closed trail.
Vertex can be repeated
Edge not repeated



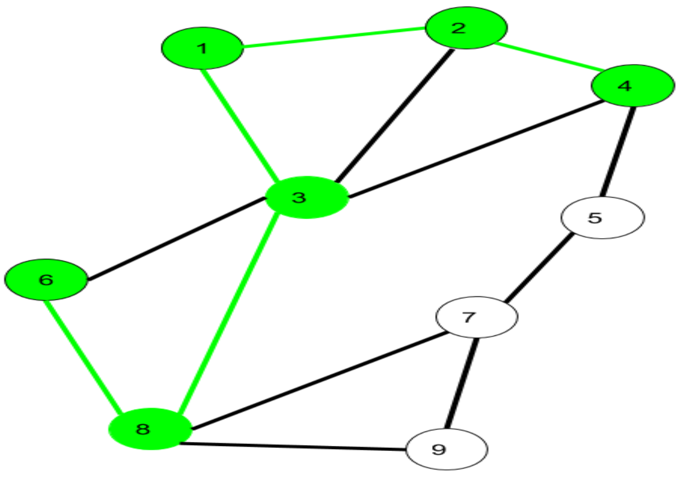
Here 1->2->4->3->6->8->3->1 is a circuit

Circuit is a closed trail.

These can have repeated vertices only.

4. Path –

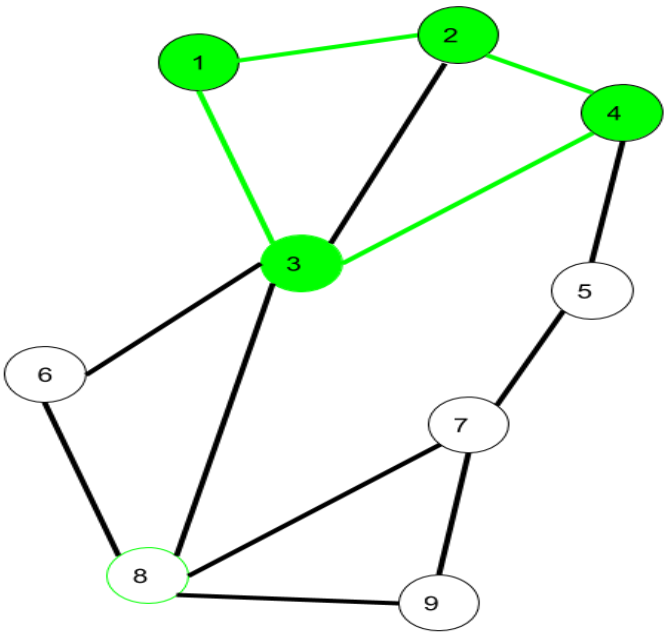
It is a trail in which neither vertices nor edges are repeated i.e. if we traverse a graph such that we do not repeat a vertex and nor we repeat an edge. As path is also a trail, thus it is also an open walk.
Vertex not repeated
Edge not repeated



Here 6->8->3->1->2->4 is a Path

5. Cycle –

Traversing a graph such that we do not repeat a vertex nor we repeat a edge but the starting and ending vertex must be same i.e. we can repeat starting and ending vertex only then we get a cycle.
Vertex not repeated
Edge not repeated

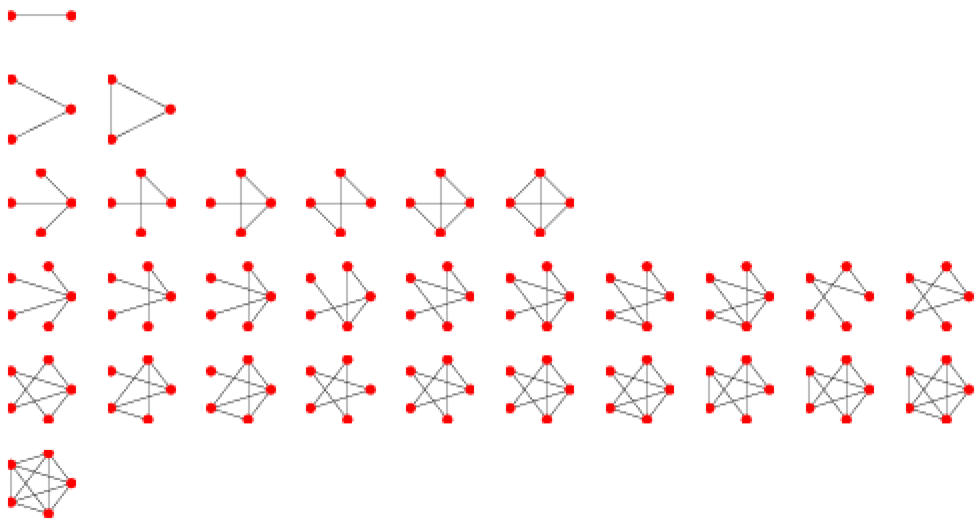


Here 1->2->4->3->1 is a cycle.

Cycle is a closed path.

These can not have repeat anything (neither edges nor vertices).

Connected Graph



A connected graph is graph that is connected in the sense of a topological space, i.e., there is a path from any point to any other point in the graph. A graph that is not connected is said to be disconnected. This definition means that the null graph and singleton graph are considered connected, while empty graphs on $n \geq 2$ nodes are disconnected.

According to West (2001, p. 150), the singleton graph K_1, "is annoyingly inconsistent" since it is connected (specifically, 1-connected), but for consistency in discussing connectivity, it is considered to have vertex connectivity $\kappa(K_1) = 0$.

If A is the adjacency matrix of a simple graph G, then entry (i, j) of A^k is the number of k-walks from vertex i to vertex j. As a result, a graph on $n > 1$ nodes is connected iff

$$\sum_{k=1}^{n-1} \mathsf{A}^k$$

has no matrix element equal to zero.

A connected graph on $n > 1$ nodes satisfies

$$\sum_{i=1}^{n} \rho(v_i) \geq \frac{1}{2}(n-1),$$

where $\rho(v_i)$ is the vertex degree of vertex i (and where the inequality can be made strict except in the case of the singleton graph K_1). However while this condition is necessary for a graph to be connected, it is not sufficient; an arbitrary graph satisfying the above inequality may be connected or disconnected.

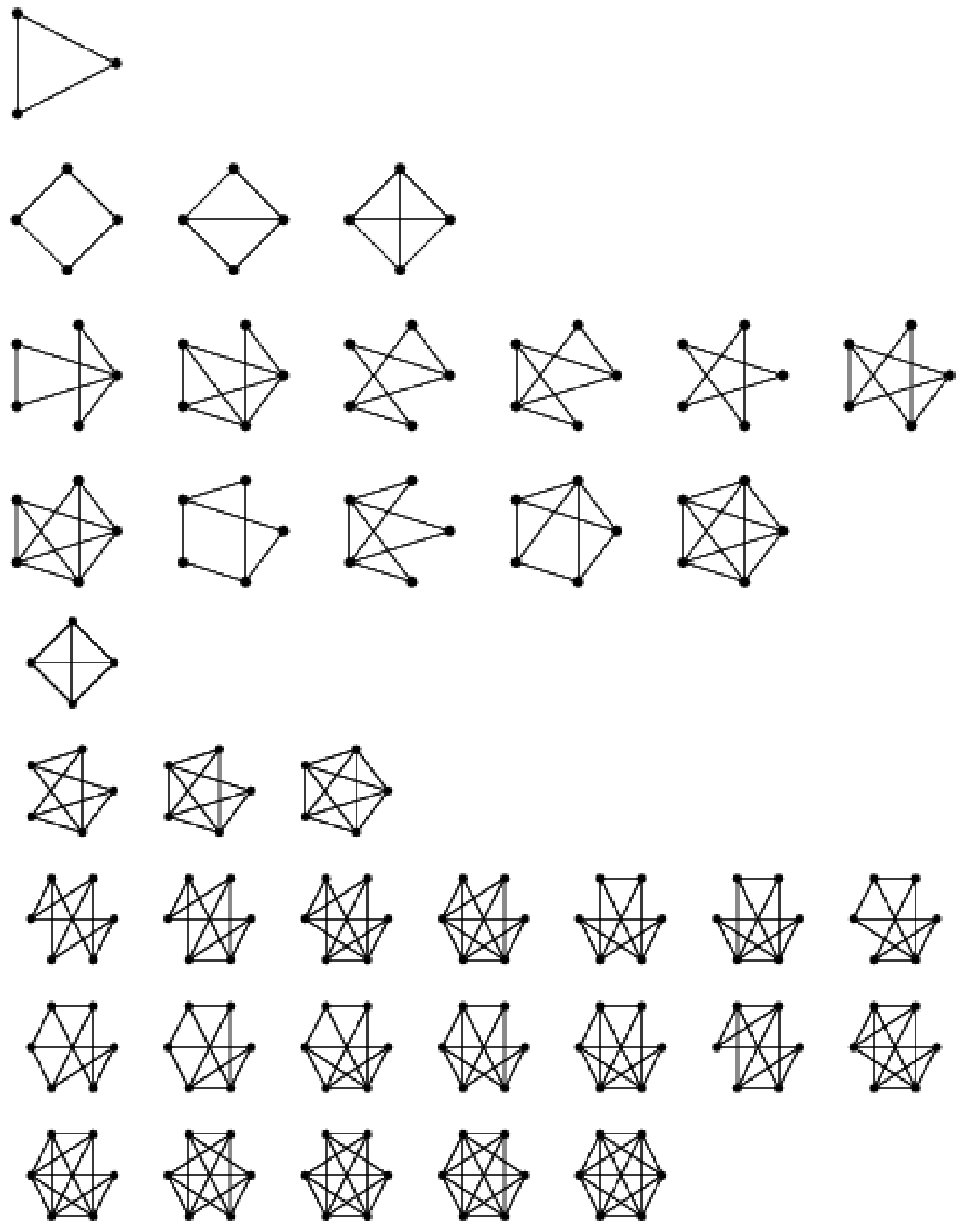
The number of n-node connected unlabeled graphs for $n = 1$, 2, ... are 1, 1, 2, 6, 21, 112, 853, 11117, 261080, ... (OEIS A001349). The total number of (not necessarily connected) unlabeled n-node graphs is given by the Euler transform of the preceding sequence, 1, 2, 4, 11, 34, 156, 1044, 12346, ... (OEIS A000088; Sloane and Plouffe 1995, p. 20). Furthermore, in general, if a_n is the number of unlabeled connected graphs on n nodes satisfying some property, then the Euler transform b_n is the total number of unlabeled graphs (connected or not) with the same property. This application of the Euler transform is called Riddell's formula.

The numbers of connected labeled graphs on n-nodes are 1, 1, 4, 38, 728, 26704, ... (OEIS A001187), and the total number of (not necessarily connected) labeled n-node graphs is given by the exponential transform of the preceding sequence: 1, 2, 8, 64, 1024, 32768, ... (OEIS A006125; Sloane and Plouffe 1995, p. 19).

An efficient enumeration of connected graphs on n nodes can be done using the program geng (part of nauty) by B. McKay using the syntax geng -c n. However, since the order in which graphs are returned by the geng program changes as a function of time as improvements are made, the canonical ordering given on McKay's website is used here and in GraphData.

A graph may be tested in the Wolfram Language to see if it is a connected graph using ConnectedGraphQ[g].

If G is disconnected, then its complement $\overline{G}$ is connected (Skiena 1990, p. 171; Bollobás 1998). However, the converse is not true, as can be seen using the example of the cycle graph C_5 which is connected and isomorphic to its complement.

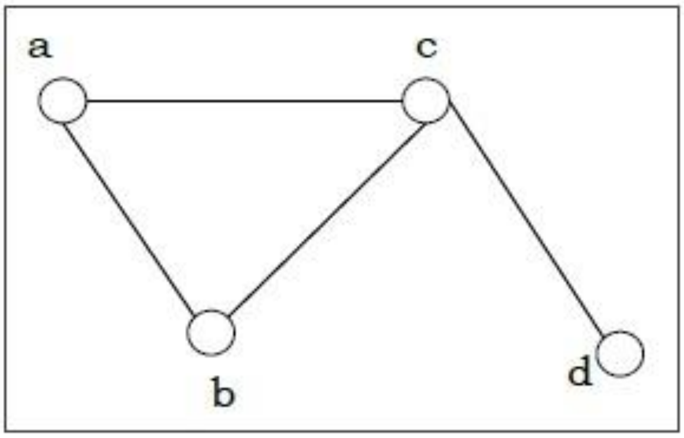


One can also speak of k-connected graphs (i.e., graphs with vertex connectivity k) in which each vertex has degree at least k (i.e., the minimum of the degree sequence is $\geq k$). 1-connected graphs are therefore connected with minimal degree ≥ 1. The following table gives the number of k-connected graphs on n vertices for small k.

k	OEIS	sequence
1	A001349	1, 1, 2, 6, 21, 112, 853, 11117, ...
2	A004108	0, 0, 1, 3, 11, 61, 507, 7442, ...
3	A007112	0, 0, 0, 1, 3, 19, 150, 2589, ...

Connected Graph

A graph is connected if any two vertices of the graph are connected by a path.

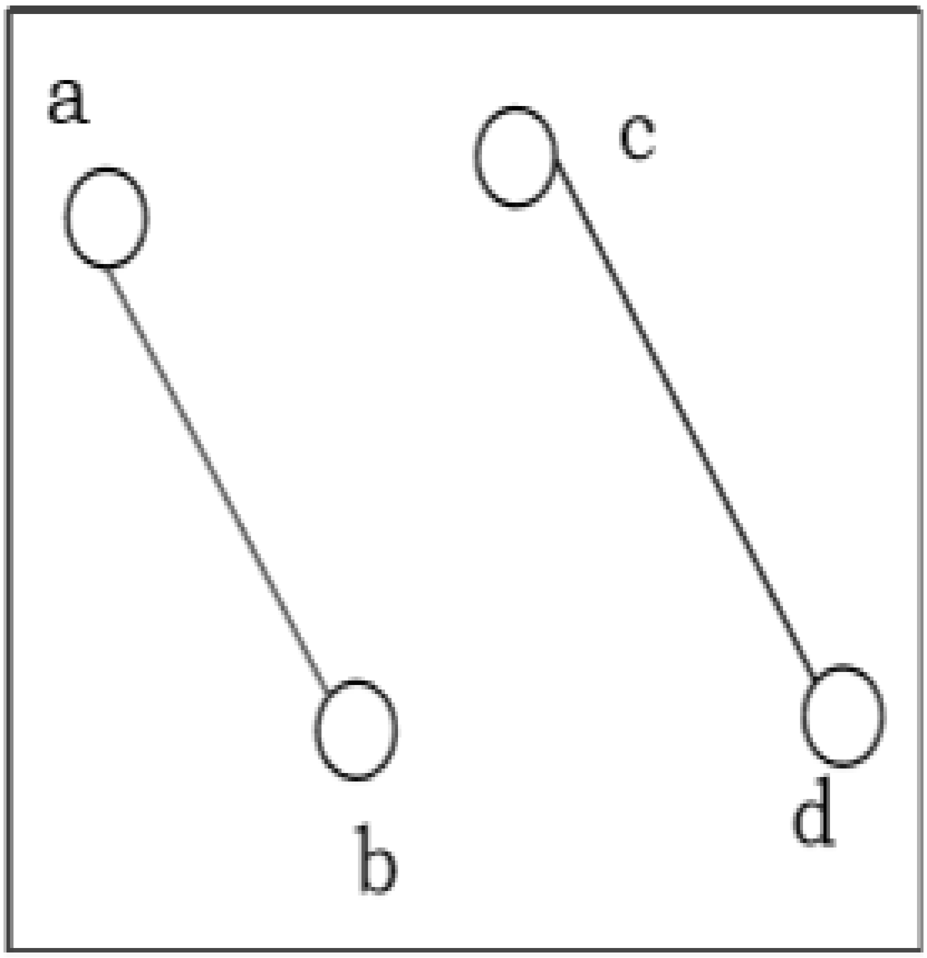


Vertex 1	Vertex 2	PATH
a	b	a b
a	c	a b c, a c
a	d	a b c d, a c d
b	c	b a c , b c
c	d	c d

Vertex 1	Vertex 2	PATH

Disconnected Graph

A graph is disconnected if at least two vertices of the graph are not connected by a path. If a graph G is disconnected, then every maximal connected subgraph of G is called a connected component of the graph G.



a	b	a b
a	c	Not Available
a	d	Not Available
b	c	Not Available
c	d	c d

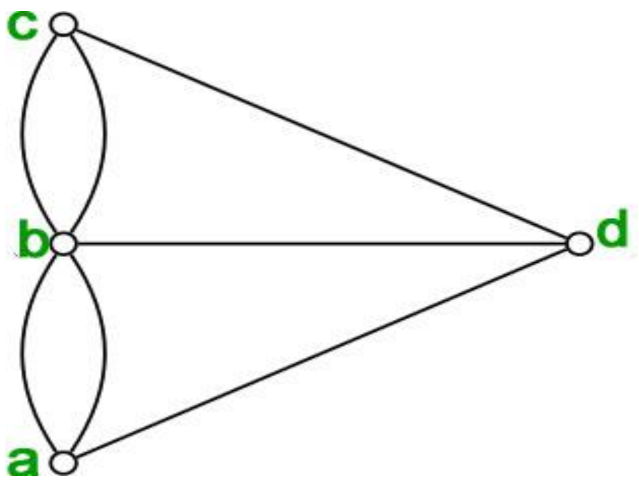
Mathematics | Euler and Hamiltonian Paths

Certain graph problems deal with finding a path between two vertices such that each edge is traversed exactly once, or finding a path between two vertices while visiting each vertex exactly once. These paths are better known as Euler path and Hamiltonian path respectively.
The Euler path problem was first proposed in the 1700's.

Euler paths and circuits :

- An Euler path is a path that uses every edge of a graph exactly once.
- An Euler circuit is a circuit that uses every edge of a graph exactly once.
- An Euler path starts and ends at different vertices.
- An Euler circuit starts and ends at the same vertex.

The Konigsberg bridge problem's graphical representation :



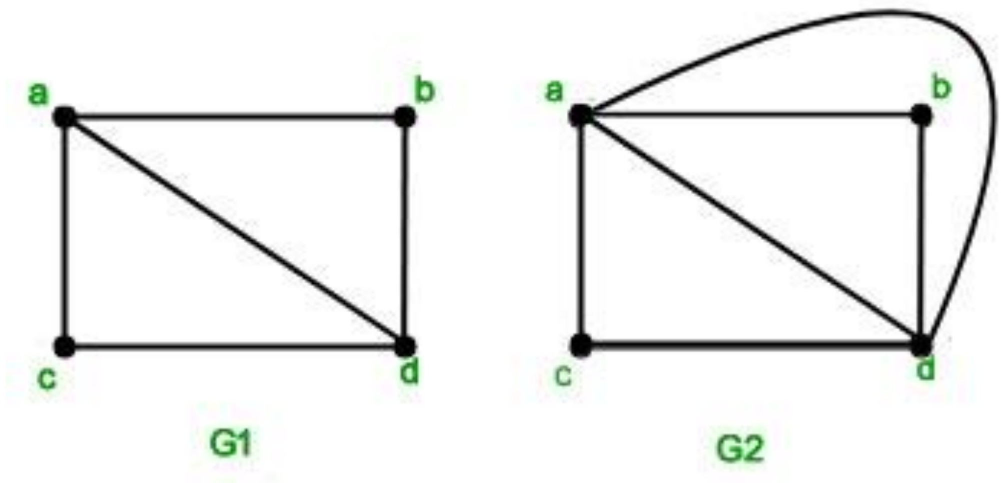
There are simple criteria for determining whether a multigraph has a Euler path or a Euler circuit. For any multigraph to have a Euler circuit, all the degrees of the vertices must be even.

Theorem – “A connected multigraph (and simple graph) with at least two vertices has a Euler circuit if and only if each of its vertices has an even degree.”

Proof of the above statement is that every time a circuit passes through a vertex, it adds twice to its degree. Since it is a circuit, it starts and ends at the same vertex, which makes it contribute one degree when the circuit starts and one when it ends. In this way, every vertex has an even degree.
Since the Koningsberg graph has vertices having odd degrees, a Euler circuit does not exist in the graph.

Theorem – “A connected multigraph (and simple graph) has an Euler path but not an Euler circuit if and only if it has exactly two vertices of odd degree.”
The proof is an extension of the proof given above. Since a path may start and end at different vertices, the vertices where the path starts and ends are allowed to have odd degrees.

- Example – Which graphs shown below have an Euler path or Euler circuit?



- Solution – has two vertices of odd degree and and the rest of them have even degree. So this graph has an Euler path but not an Euler circuit. The path starts and ends at the vertices of odd degree. The path is- .
  has four vertices all of even degree, so it has a Euler circuit. The circuit is –
  .

Hamiltonian paths and circuits :
Hamilonian Path – A simple path in a graph that passes through every vertex exactly once is called a Hamiltonian path.

Hamilonian Circuit – A simple circuit in a graph that passes through every vertex exactly once is called a Hamiltonian circuit.

Unlike Euler paths and circuits, there is no simple necessary and sufficient criteria to determine if there are any Hamiltonian paths or circuits in a graph. But there are certain criteria which rule out the existence of a Hamiltonian circuit in a graph, such as-if there is a vertex of degree one in a graph then it is impossible for it to have a Hamiltonian circuit.

There are certain theorems which give sufficient but not necessary conditions for the existence of Hamiltonian graphs.

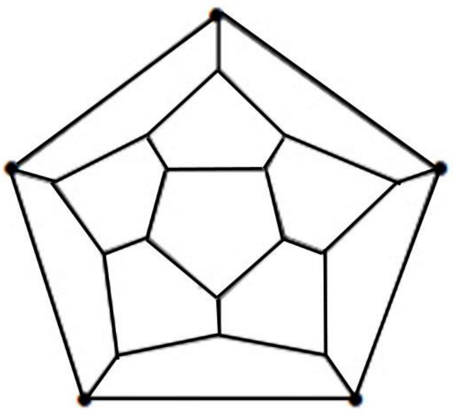
Dirac's Theorem- "If is a simple graph with vertices with such that the degree of every vertex in is at least , then has a Hamiltonian circuit."

Ore's Theorem- "If is a simple graph with vertices with such that for every pair of non-adjacent vertices and in , then has a Hamiltonian circuit."

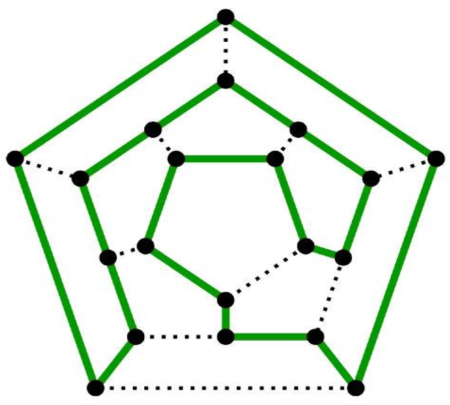
As mentioned above that the above theorems are sufficient but not necessary conditions for the existence of a Hamiltonian circuit in a graph, there are certain graphs which have a Hamiltonian circuit but do not follow the conditions in the above-mentioned theorem. For example, the cycle has a Hamiltonian circuit but does not follow the theorems.

There are many practical problems which can be solved by finding the optimal Hamiltonian circuit. One such problem is the Travelling Salesman Problem which asks for the shortest route through a set of cities.

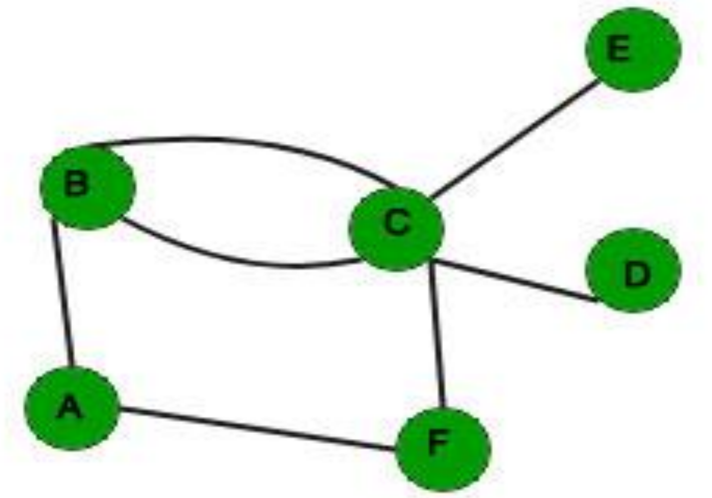
- Example 1- Does the following graph have a Hamiltonian Circuit?



- Solution- Yes, the above graph has a Hamiltonian circuit. The solution is –



- Example 2- Does the following graph have a Hamiltonian Circuit?



- Solution- No the above graph does not have a Hamiltonian circuit as there are two vertices with degree one in the graph.

Operation on graphs

Graph Operations – Extracting sub graphs

In this section we will discuss about various types of sub graphs we can extract from a given Graph.

Sub graph

Getting a sub graph out of a graph is an interesting operation. A sub graph of a graph G(V,E) can be obtained by the following means:

- Removing one or more vertices from the vertex set.
- Removing one or more edges from the edge family.
- Removing either vertices or edges from the graph.

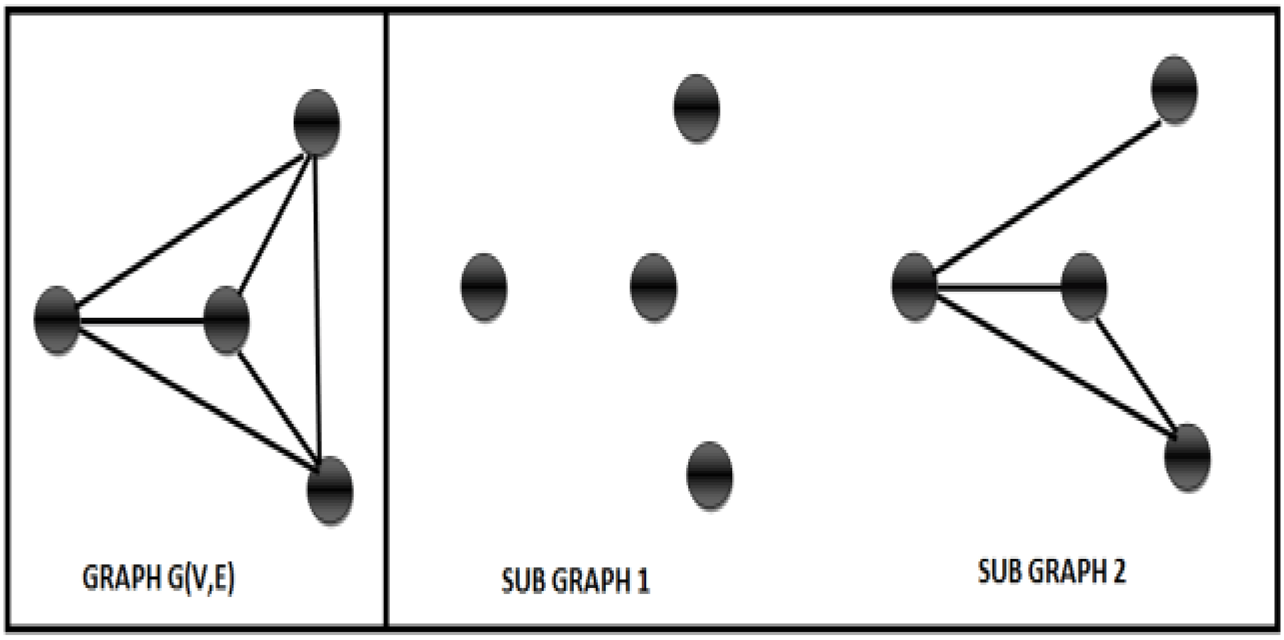
Points worth noting

- The vertices of sub graphs are subsets of the original vertices
- The edges of sub graphs are subsets of the original edges

We can extract sub graphs for simple graphs, directed graphs, multi edge graphs and all types of graphs.

The Null Graph is always a sub graph of all the graphs. There can be many sub graphs for a graph.

Below is a graph and its sub graphs.

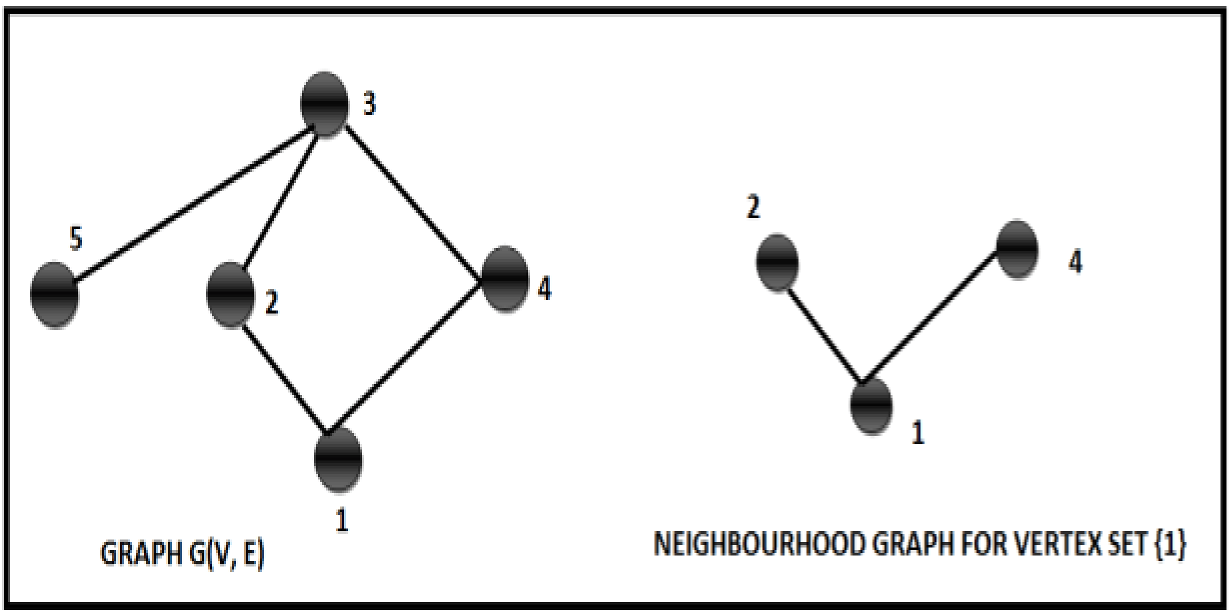


Neighbourhood graph

The neighbourhood graph of a graph G(V,E) only makes sense when we mention it with respect to a given vertex set. For e.g. if V = {1,2,3,4,5} then we can find out the Neighbourhood graph of G(V,E) for vertex set {1}.

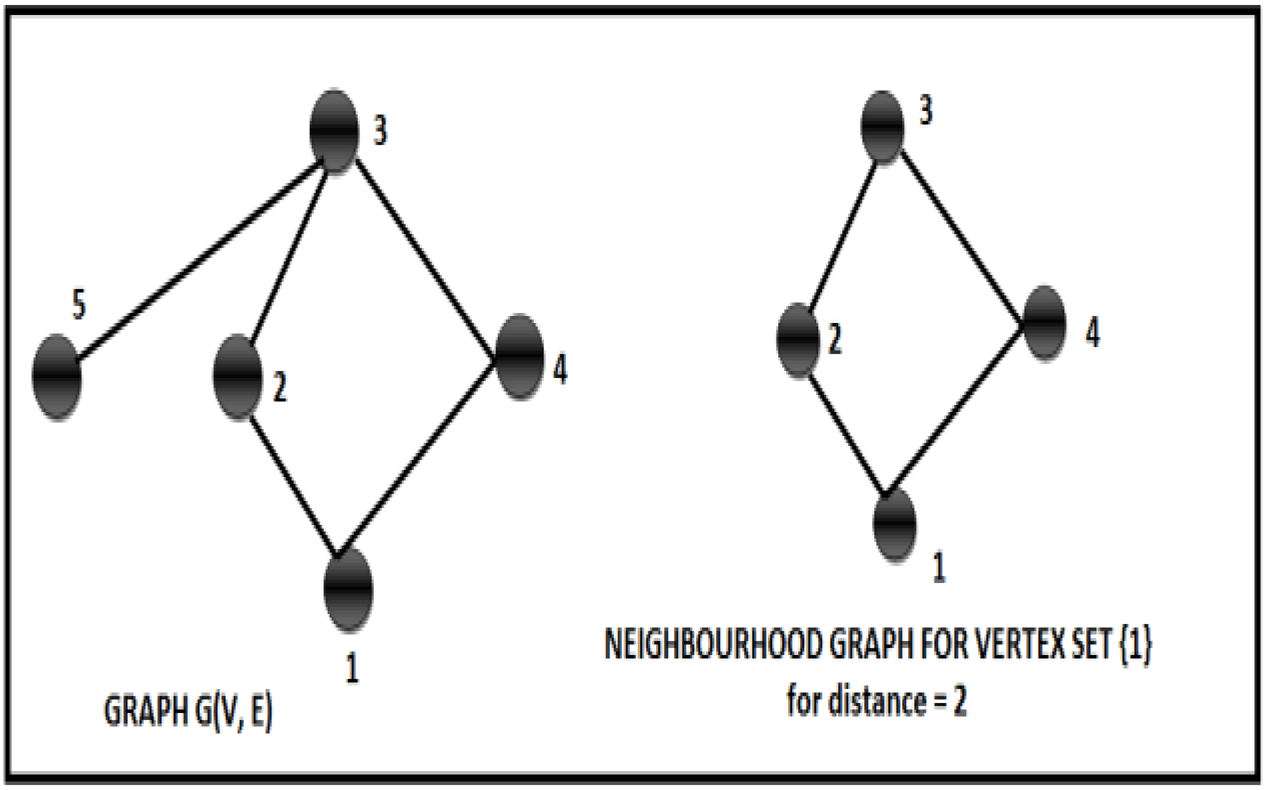
So, the neighbourhood graphs contains the vertices 1 and all the edges incident on them and the vertices connected to these edges.

Below is a graph and its neighbourhood graphs as described above.



We can also extract neighbourhood graph for a distance more than 1, which means that we do not stop at the first neighbour and also extend it to second, third and more.

Another example for neighbourhood graph up to distance 2 is given below:



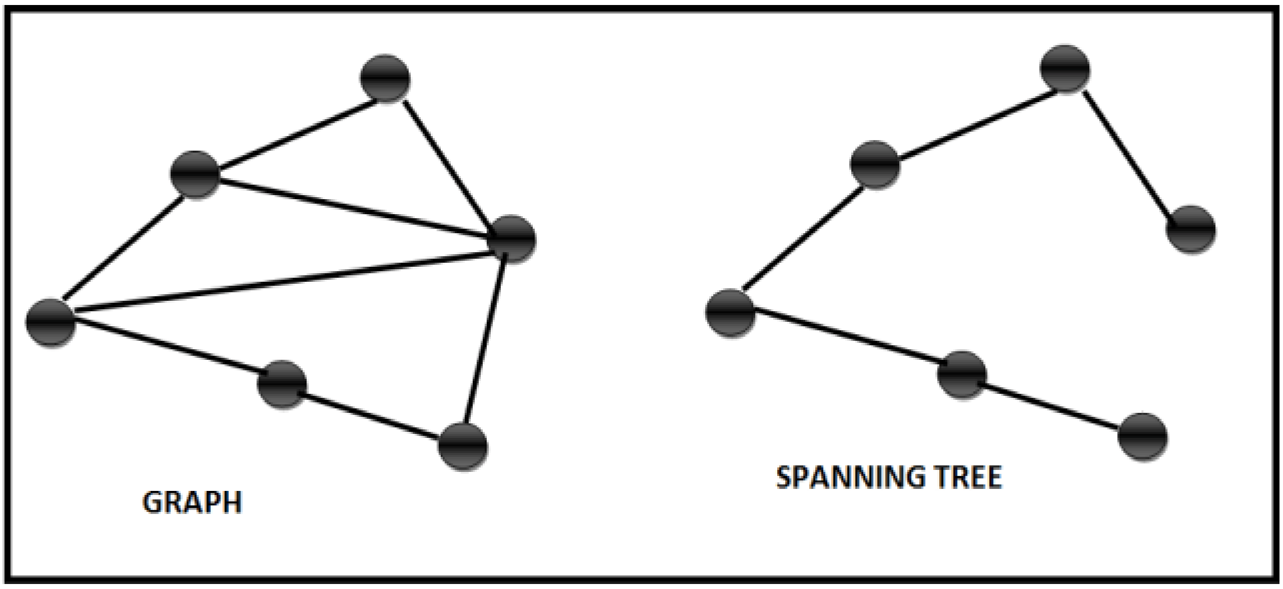
Spanning Tree

A spanning tree of a connected graph G(V,E) is a sub graph that is also a tree and connects all vertices in V. For a disconnected graph the spanning tree would be the spanning tree of each component respectively.

There is an interesting set of problems related to finding the minimum spanning tree (which we will be discussing in upcoming posts). There are many algorithms available to solve this problem, for e.g.: Kruskal's, Prim's etc. Note that the concept of minimum spanning tree mostly makes sense in case of weighted graphs. If the graph is not weighted, we consider all the weights to be one and any spanning tree becomes the minimum spanning tree.

We can use traversals like Breadth First Scan and Depth First Scan to fight the Spanning Tree. We can find spanning tree for undirected graphs, directed graphs, multi graphs as well.

Below is an example of spanning tree for the given graph.



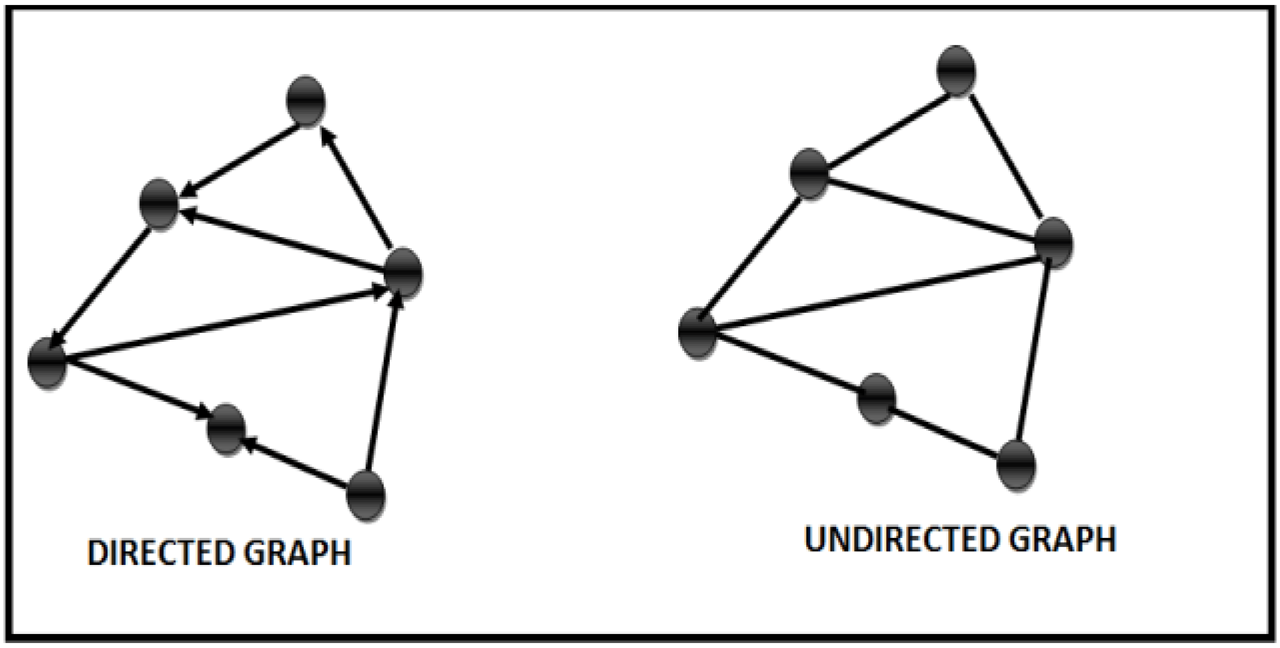
Graph Operations – Conversions of Graphs

In this section we discuss about converting one graph into another graph. Which means all the graphs can be converted into any of the below forms.

Conversion from Directed Graph to Undirected graph

This is the simplest conversions possible. A directed graph has directions represented by arrows, in this conversion we just remove all the arrows and do not store the direction information. Below is an example of the conversion.

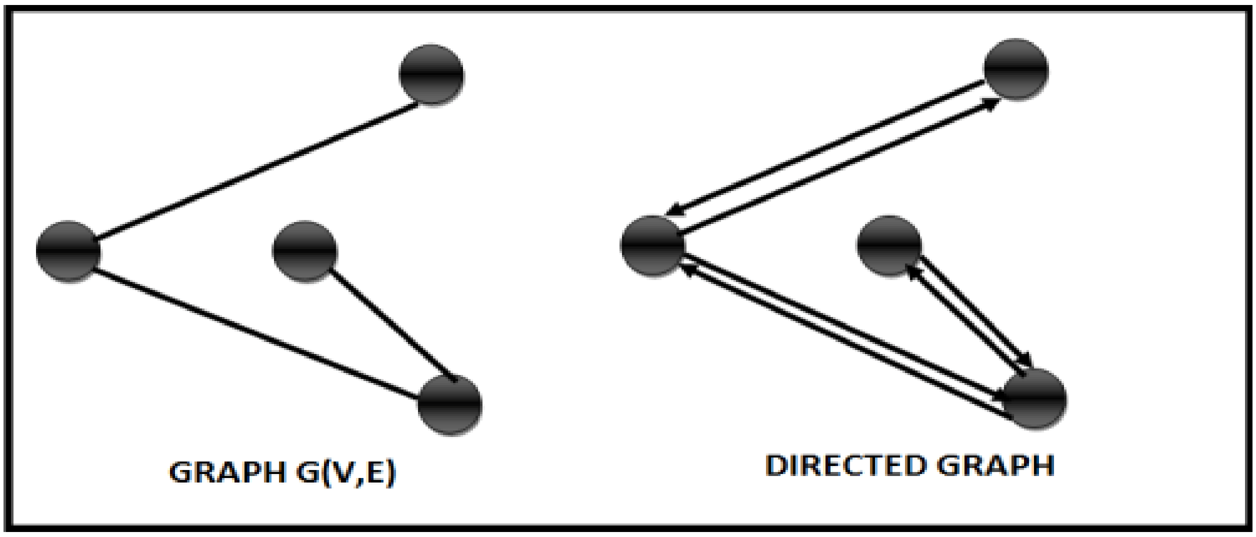
Please note that the graph remains unchanged in terms of its structure. However, we can choose to remove edges if there are multi edges. But it is strictly not required.



Conversion from Undirected Graph to Directed graph

This conversion gives a directed graph given an undirected graph G(V,E). It is the exact reverse of the above. The trick to achieve this is to add one edge for each existing edge in the edge family E. Once the extra edges are added, we just assign opposite direction to each pair of edges between connecting vertices.

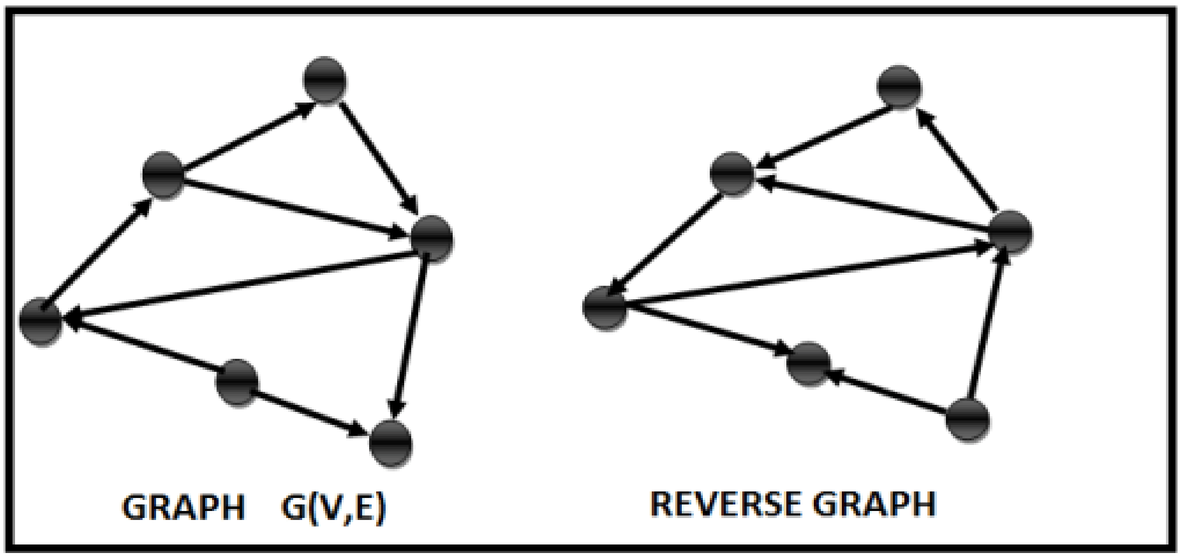
Below is a demonstration for the same.



Reversing a graph

Reversing a graph is an operation meant for directed graphs. To reverse a graph we reverse the direction of the edges. The reverse graph has the same vertex set as the original graph. As the edges are reversed, the adjacency matrix for this graph is the Transpose of the adjacency matrix for the original graph (Left to the user to draw and check if this holds true).

Below is the illustration of reversing a graph G(V,E)

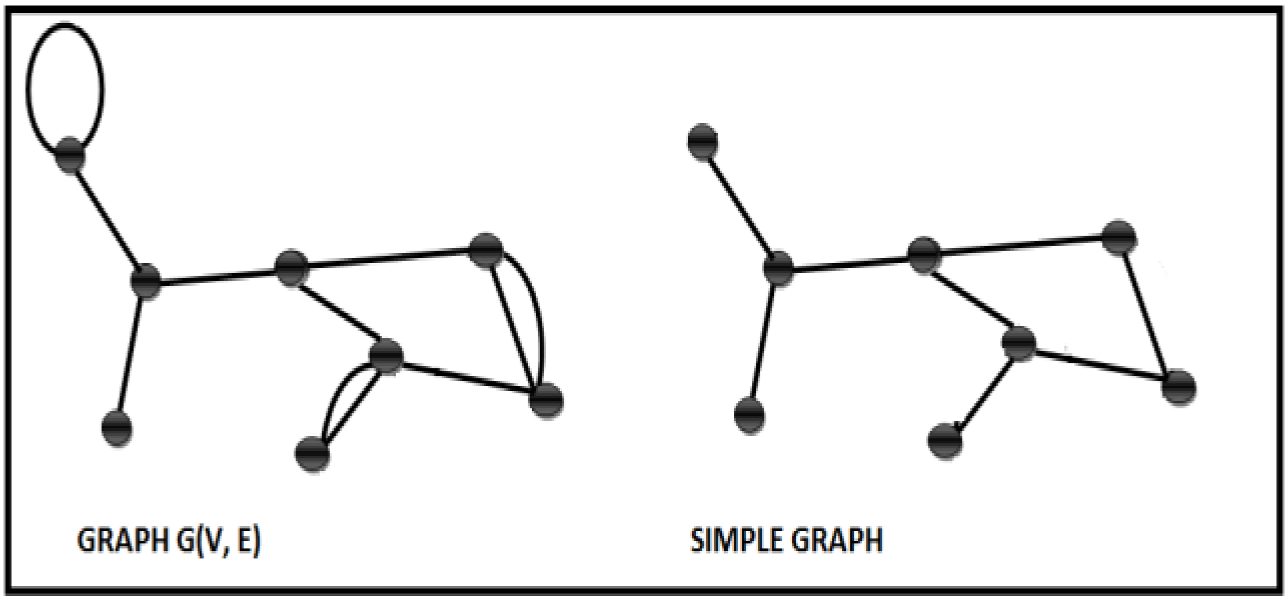


Deriving a Simple graph

The operation is to derive a simple graph out of any given graph. A simple graph by definition must not contain any self loops or multi edges. As we understand that a graph can contain loops and multiple edges, this operation shall remove the loops and multiple edges from the graph G(V,E) to obtain a simple graph.

The adjacency matrix of such a graph will surely have the diagonal elements as zero, because there is no self loops.

Below is the image to demonstrate this operation.



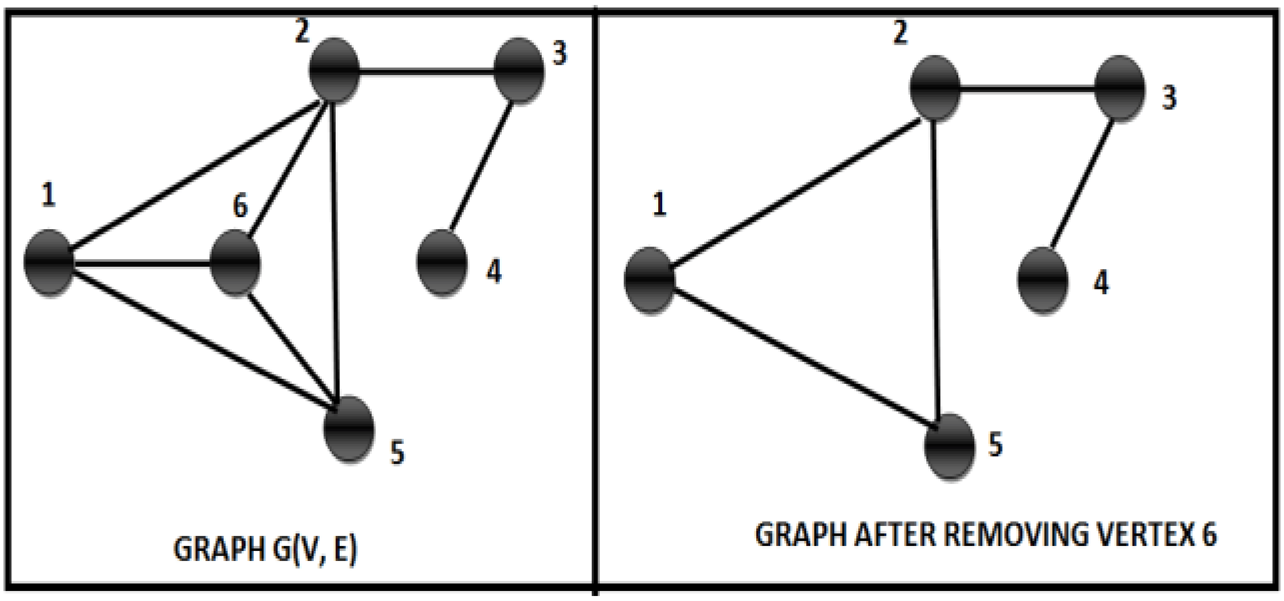
Graph Operations – Modifications of Graphs

This section talks about some important modifications on a graph. These operations will majorly change the structure of the underlying graph.

Delete Vertex

What happens when we delete a vertex from a given Graph G(V, E)?
We cannot successfully remove a vertex if we do not remove all the edges incident on the vertex. Once we remove the vertex then the adjacency matrix will not contain the row and column for the corresponding vertex.
This operation changes the vertex set and the edge family of the graph.

Below image helps in understanding the removal of vertex.

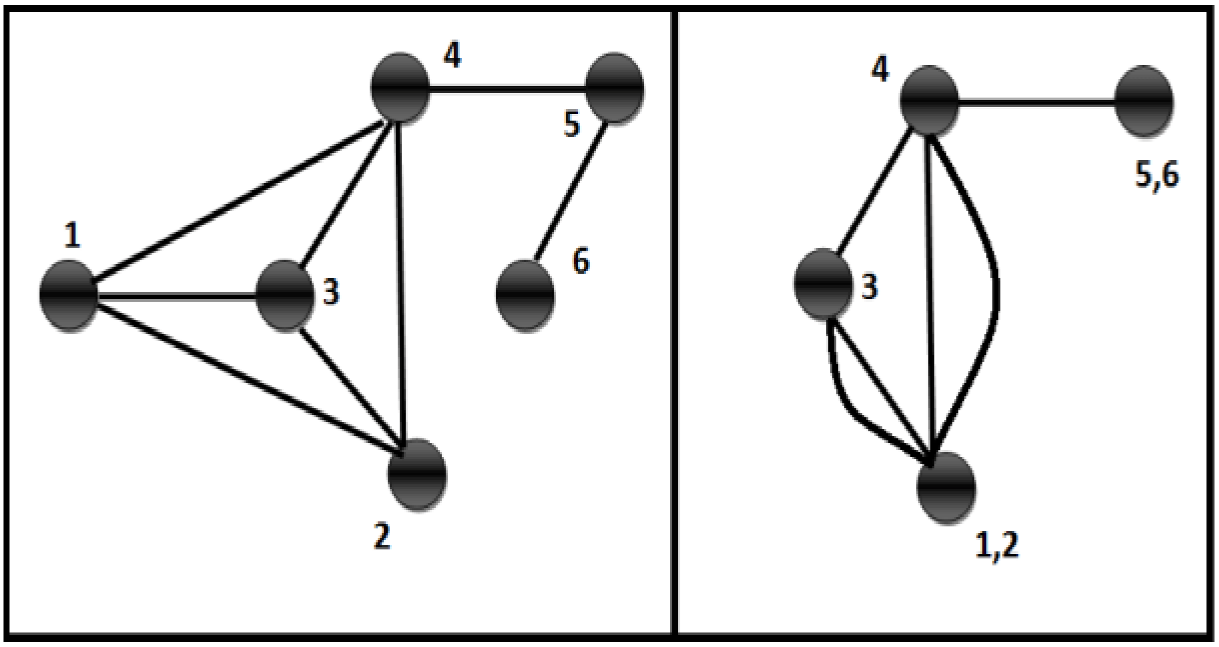


Contract Vertex

One of the important operations on a graph is contraction. It can be done by contracting two vertices into one. Also, it can be done by contracting an edge, which we will see later in this article.

We cannot contract one vertex, for contraction we need a set of vertices, minimum two. Contraction can only be done when there is an edge between the two vertices. The operation basically removes all the edges between the two vertices.

In the below illustration vertices 1,2 are contracted and 5,6 are also contracted. Please notice the removal of edges between them.



Tree and fundamental circuits

In graph theory, a tree is an undirected graph in which any two vertices are connected by exactly one path, or equivalently a connected acyclic undirected graph.[1] A forest is an undirected graph in which any two vertices are connected by at most one path, or equivalently an acyclic undirected graph, or equivalently a disjoint union of trees.[2]

A polytree[3] (or directed tree[4] or oriented tree[5][6] or singly connected network[7]) is a directed acyclic graph (DAG) whose underlying undirected graph is a tree. A polyforest (or directed forest or oriented forest) is a directed acyclic graph whose underlying undirected graph is a forest.

The various kinds of data structures referred to as trees in computer science have underlying graphs that are trees in graph theory, although such data structures are generally rooted trees. A rooted tree may be directed, called a directed rooted tree,[8][9] either making all its edges point away from the root—in which case it is called an arborescence[4][10] or out-tree[11][12]—or making all its edges point towards the root—in which case it is called an anti-arborescence[13] or in-tree.[11][14] A rooted tree itself has been defined by some authors as a directed graph.[15][16][17] A rooted forest is a disjoint union of rooted trees. A rooted forest may be directed, called a directed rooted forest, either making all its edges point away from the root in each rooted tree—in which case it is called a branching or out-forest—or making all its edges point towards the root in each rooted tree—in which case it is called an anti-branching or in-forest.

Distance diameters

Geodesic distance" redirects here. For distances on the surface of a sphere, see Great-circle distance. For distances on the surface of the Earth, see Geodesics on an ellipsoid. For geodesics in differential geometry, see Geodesic.

In the mathematical field of graph theory, the distance between two vertices in a graph is the number of edges in a shortest path (also called a graph geodesic) connecting them. This is also known as the geodesic distance.[1] Notice that there may be more than one shortest path between two vertices.[2] If there is no path connecting the two vertices, i.e., if they belong to different connected components, then conventionally the distance is defined as infinite.

In the case of a directed graph the distance between two vertices and is defined as the length of a shortest directed path from to consisting of arcs, provided at least one such path exists.[3] Notice that, in contrast with the case of undirected graphs, does not necessarily coincide with , and it might be the case that one is defined while the other is not.

Related concepts

A metric space defined over a set of points in terms of distances in a graph defined over the set is called a graph metric. The vertex set (of an undirected graph) and the distance function form a metric space, if and only if the graph is connected.

The eccentricity of a vertex is the greatest distance between and any other vertex; in symbols that is . It can be thought of as how far a node is from the node most distant from it in the graph.

The radius of a graph is the minimum eccentricity of any vertex or, in symbols, .

The diameter of a graph is the maximum eccentricity of any vertex in the graph. That is, is the greatest distance between any pair of vertices or, alternatively, . To find the diameter of a graph, first find the shortest path between each pair of vertices. The greatest length of any of these paths is the diameter of the graph.

A central vertex in a graph of radius is one whose eccentricity is —that is, a vertex that achieves the radius or, equivalently, a vertex such that .

A peripheral vertex in a graph of diameter is one that is distance from some other

vertex—that is, a vertex that achieves the diameter. Formally, is peripheral if .

A pseudo-peripheral vertex has the property that for any vertex , if is as far away from as possible, then is as far away from as possible. Formally, a vertex u is

pseudo-peripheral, if for each vertex v with holds .

The partition of a graph's vertices into subsets by their distances from a given starting vertex is called the level structure of the graph.

A graph such that for every pair of vertices there is a unique shortest path connecting them is called a geodetic graph. For example, all trees are geodetic.[4]

Algorithm for finding pseudo-peripheral vertices

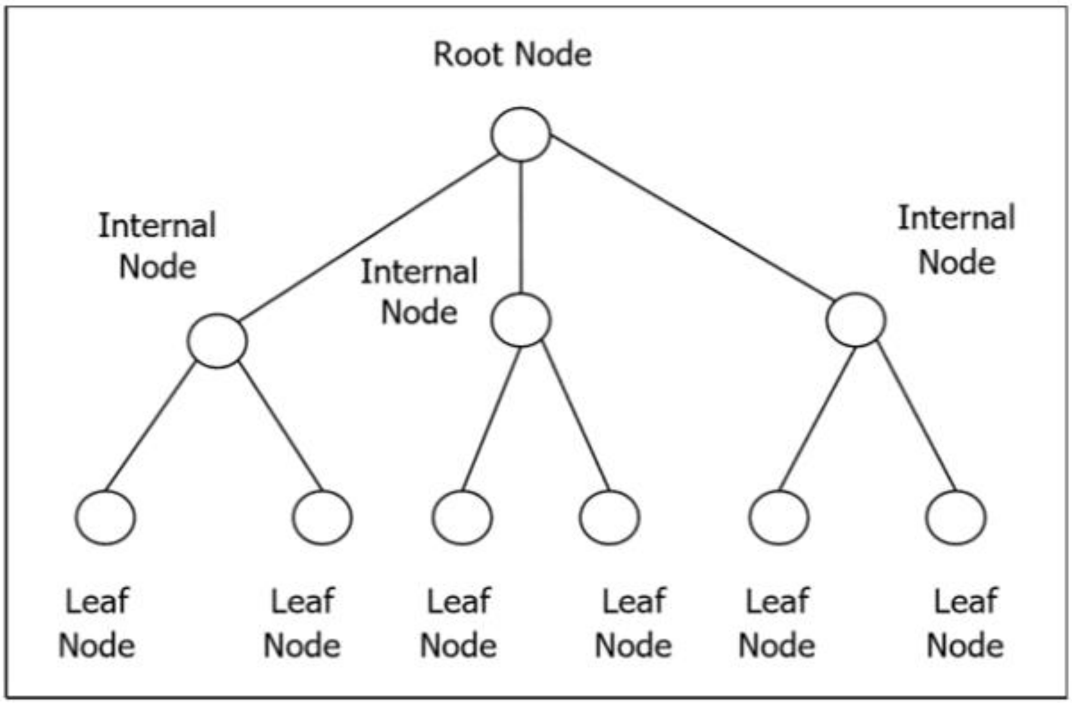
Often peripheral sparse matrix algorithms need a starting vertex with a high eccentricity. A peripheral vertex would be perfect, but is often hard to calculate. In most circumstances a pseudo-peripheral vertex can be used. A pseudo-peripheral vertex can easily be found with the following algorithm:

1. Choose a vertex .
2. Among all the vertices that are as far from as possible, let be one with minimal degree.
3. If then set and repeat with step 2, else is a pseudo-peripheral vertex.

Radius and pendent vertices

Rooted Tree

A rooted tree G is a connected acyclic graph with a special node that is called the root of the tree and every edge directly or indirectly originates from the root. An ordered rooted tree is a rooted tree where the children of each internal vertex are ordered. If every internal vertex of a rooted tree has not more than m children, it is called an m-ary tree. If every internal vertex of a rooted tree has exactly m children, it is called a full m-ary tree. If m = 2, the rooted tree is called a binary tree.



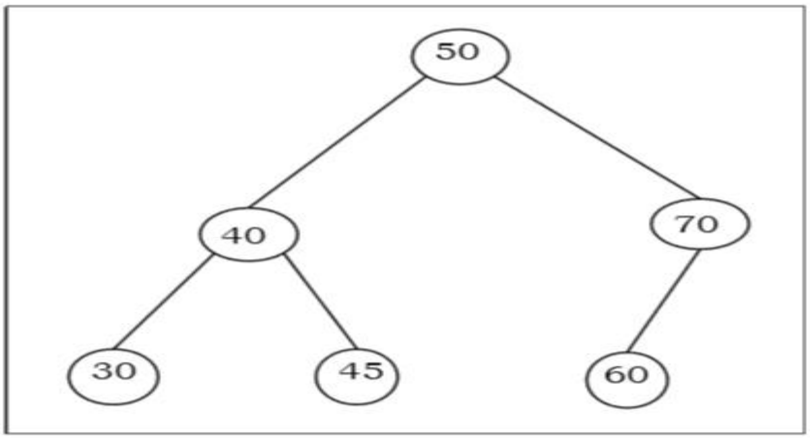
Binary Search Tree

The binary Search tree is a binary tree which satisfies the following property −

- X in left sub-tree of vertex V, Value(X) ≤ Value (V)
- Y in right sub-tree of vertex V, Value(Y) ≥ Value (V)

So, the value of all the vertices of the left sub-tree of an internal node V are less than or equal to V and the value of all the vertices of the right sub-tree of the internal node V are greater than or equal to V. The number of links from the root node to the deepest node is the height of the Binary Search Tree.

Example



Algorithm to search for a key in BST

```
BST_Search(x, k)
if( x = NIL or k =Value[x])
return x;
if( k <Value[x])
return BST_Search (left[x], k);
else
return BST_Search (right[x], k)
```

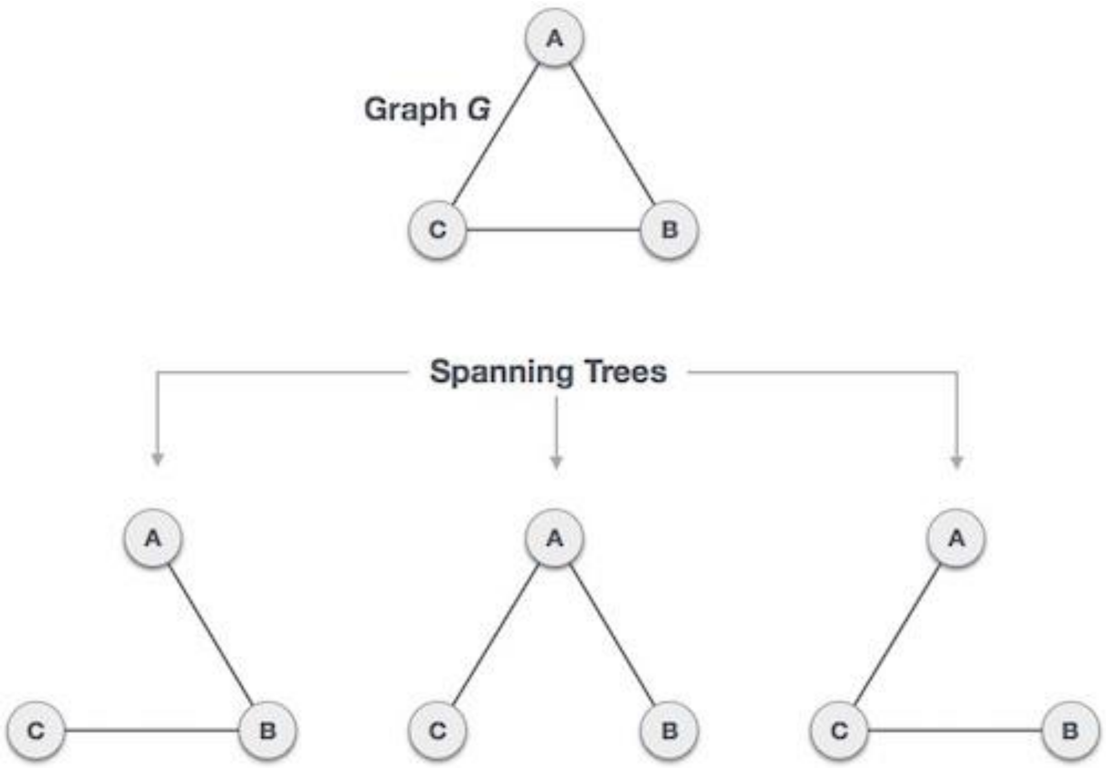
Complexity of Binary search tree

	Average Case	Worst case
Space Complexity	O(n)	O(n)
Search Complexity	O(log n)	O(n)
Insertion Complexity	O(log n)	O(n)
Deletion Complexity	O(log n)	O(n)

Spanning trees

A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected..

By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree. A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.



We found three spanning trees off one complete graph. A complete undirected graph can have maximum nn-2 number of spanning trees, where n is the number of nodes. In the above addressed example, n is 3, hence 33−2 = 3 spanning trees are possible.

General Properties of Spanning Tree

We now understand that one graph can have more than one spanning tree. Following are a few properties of the spanning tree connected to graph G −

- A connected graph G can have more than one spanning tree.
- All possible spanning trees of graph G, have the same number of edges and vertices.
- The spanning tree does not have any cycle (loops).
- Removing one edge from the spanning tree will make the graph disconnected, i.e. the spanning tree is minimally connected.
- Adding one edge to the spanning tree will create a circuit or loop, i.e. the spanning tree is maximally acyclic.

Mathematical Properties of Spanning Tree

- Spanning tree has n-1 edges, where n is the number of nodes (vertices).
- From a complete graph, by removing maximum e - n + 1 edges, we can construct a spanning tree.
- A complete graph can have maximum nn-2 number of spanning trees.

Thus, we can conclude that spanning trees are a subset of connected Graph G and disconnected graphs do not have spanning tree.

Application of Spanning Tree

Spanning tree is basically used to find a minimum path to connect all nodes in a graph. Common application of spanning trees are −

- Civil Network Planning
- Computer Network Routing Protocol
- Cluster Analysis

Let us understand this through a small example. Consider, city network as a huge graph and now plans to deploy telephone lines in such a way that in minimum lines we can connect to all city nodes. This is where the spanning tree comes into picture.

Minimum Spanning Tree (MST)

In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph. In real-world situations, this weight can be measured as distance, congestion, traffic load or any arbitrary value denoted to the edges.

Minimum Spanning-Tree Algorithm

We shall learn about two most important spanning tree algorithms here −

- Kruskal's Algorithm
- Prim's Algorithm

Finding all spanning trees of a graph and a weighted graph.

Complete Weighted Graph: A graph in which an edge connects each pair of graph vertices and each edge has a weight associated with it is known as a complete weighted graph.
The number of spanning trees for a complete weighted graph with n vertices is n(n-2).
Proof: Spanning tree is the subgraph of graph G that contains all the vertices of the graph. Therefore, the number of spanning trees of a complete weighted graph is the same as the number of labeled trees (need not be binary) with n vertices.
The Prüfer sequence of a labeled tree of n vertices is a unique sequence of length (n-2) associated with the tree. Also, for a given Prüfer sequence of length (n-2) on the labels 1 to n, there is a unique labeled tree with the given Prüfer sequence. Therefore, we have a bijection between the set A of labeled trees with n vertices and the set B of Prüfer sequences of size n-2 on the labels 1 to n. This can be proved as follows –

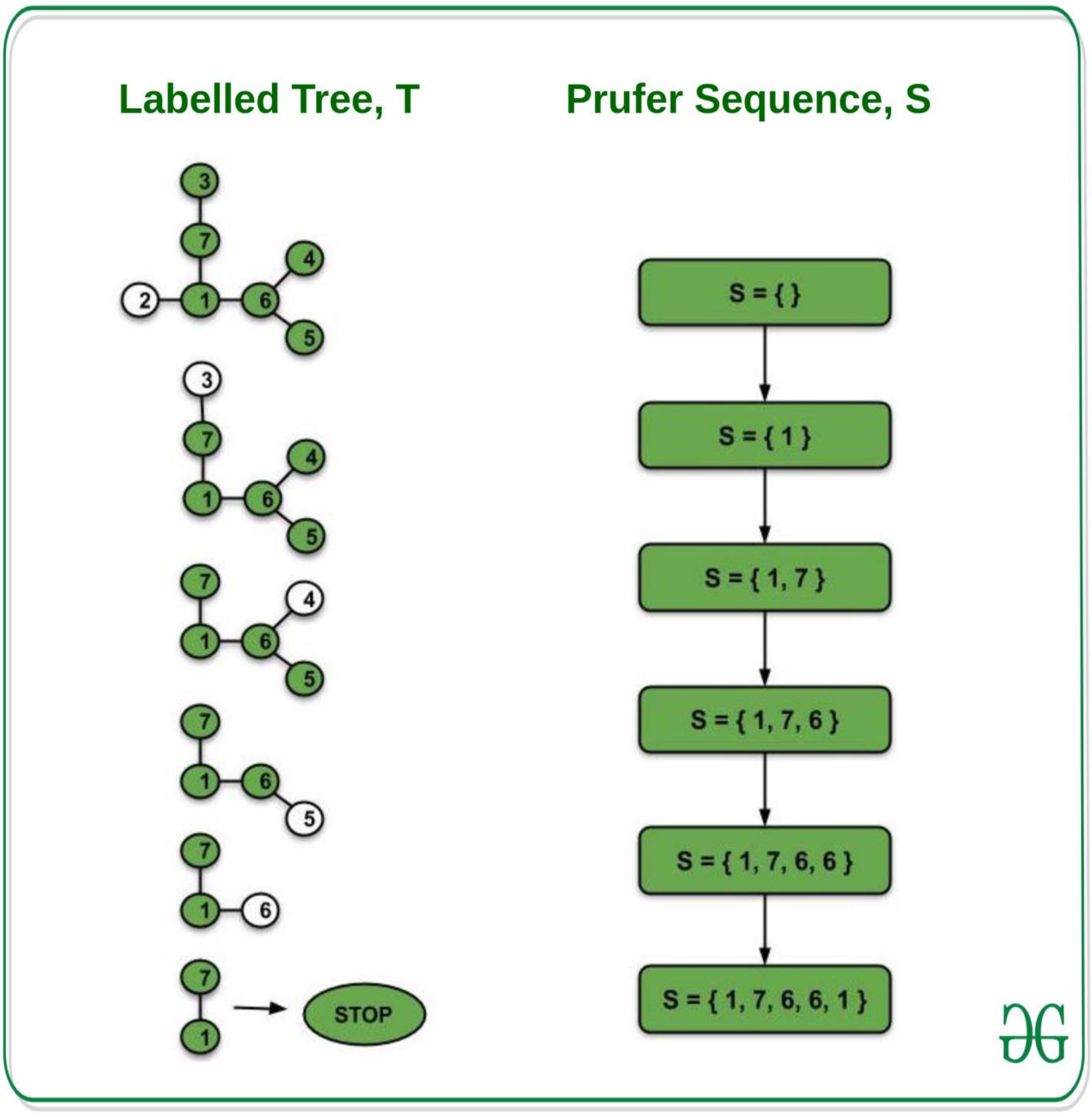
Let T be a labeled tree with vertices 1,2,…,n, and S as a Prüfer sequence of size (n-2). Thus, T and S are the elements of sets A and B, respectively.

(i) Labeled tree (T) –> Prufer sequence (S)
Constructing the Prüfer sequence of a labeled tree –

Initially, let S = NULL.

Procedure –

- Find the leaf node(L) of T with the smallest label.
- Add the neighbour of L to S.
- Delete the leaf node, L.
- Repeat the above steps until there are only two nodes left in the tree (Only one spanning tree is possible).
- We constructed the Prüfer sequence S associated with the labeled tree T.



Observations –

- No leaf node is appended to S.
- Every vertex V of tree T is added to S, a total of degree(V)-1 times.
- The tree T has n vertices and hence (n-1) edges.
- Number of terms in S = Sum of (degree(V) – 1) for all vertices V belonging to the tree T = Sum of degree of all vertices of tree T – (1+1+…+1..n times) = 2(no. of edges) – n = 2*(n-1) – n = n-2. (Since the sum of the degree of all vertices of a tree = 2*no. of edges of the tree).
- Therefore, T is analogous to the Prüfer's sequence S of length (n-2).

(ii) Prufer Sequence (S) –> Labeled Tree (T)

Constructing the labeled tree from its Prüfer sequence–

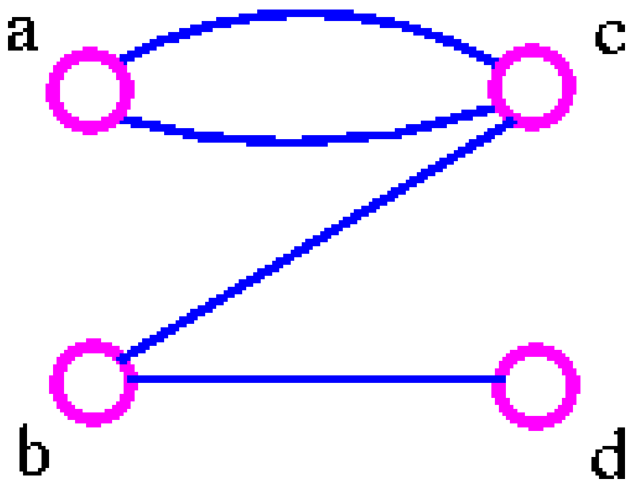
Procedure-

- Let L = {1, 2, …, n} be the set of labels (vertices of T).
- Let S = {a1,a2,…,a(n-2)} be the Prüfer sequence of size (n-2) where each ai belongs to L.
- Find the smallest element x that belongs to L but is not in S.
- Connect x and the first element of S (a1) through an edge.
- Delete a1 from S, x from L (Thus, S:=S-{a1} and, L:=L-{x}).
- Similarly, find y, the smallest element belonging to L and not in S.
- Connect y and first element of S(a2).
- Remove y from L and a2 from S (Thus, S:=S-{a2} and, L:=L-{y}).
- Continue the above process till two items are left in L.
- Connect these two items in the tree formed so far.

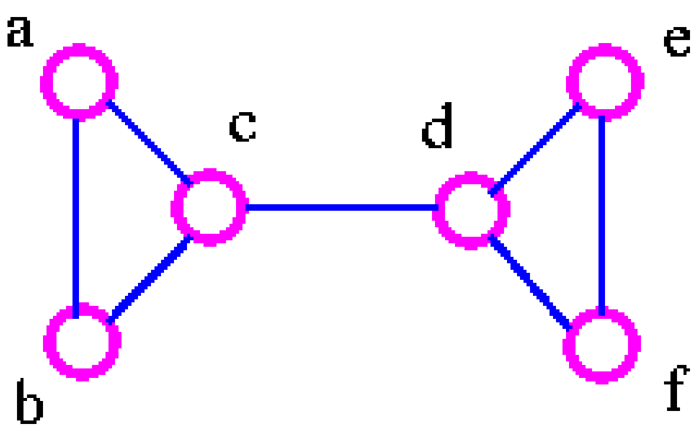
Unit IV

All cut sets in a graph

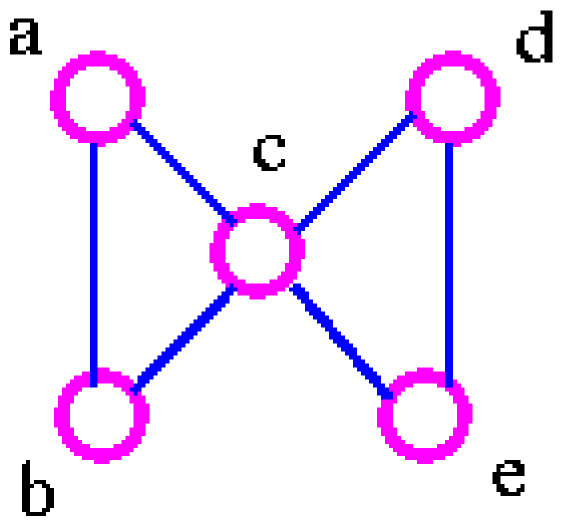
Cut Edge (Bridge)   A bridge is a single edge whose removal disconnects a graph.



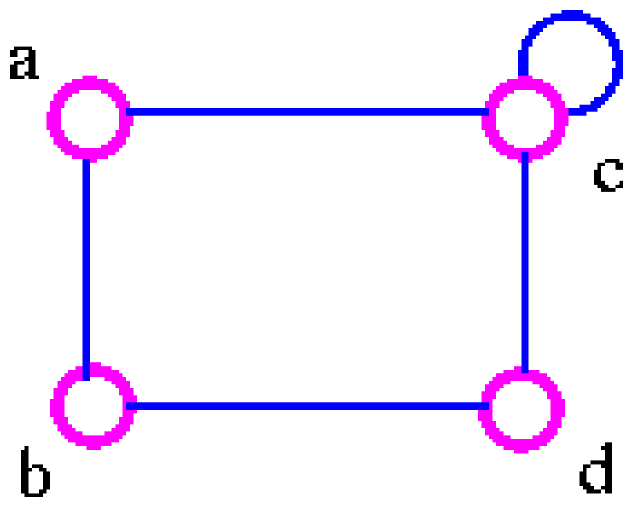
The above graph G1 can be split up into two components by removing one of the edges bc or bd. Therefore, edge bc or bd is a bridge.



The above graph G2 can be disconnected by removing a single edge, cd. Therefore, edge cd is a bridge.



The above graph G3 cannot be disconnected by removing a single edge, but the removal of two edges (such as ac and bc) disconnects it.



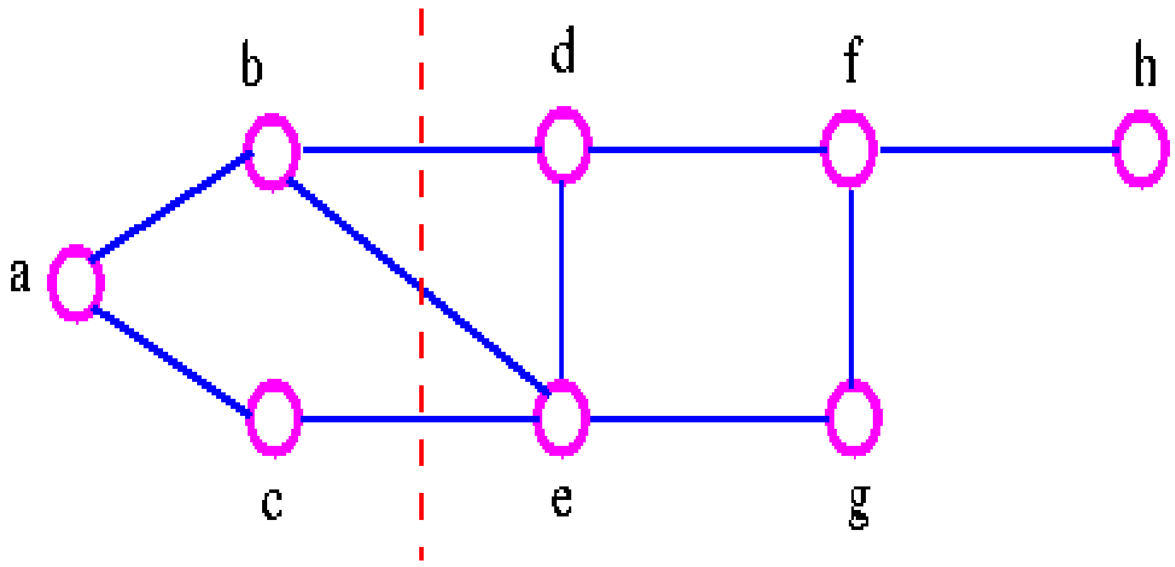
The above graph G4 can be disconnected by removing two edges such as ac and dc.

Cut Set

A cut set of a connected graph G is a set S of edges with the following properties

- The removal of all edges in S disconnects G.
- The removal of some (but not all) of edges in S does not disconnects G.

As an example consider the following graph



We can disconnect G by removing the three edges bd, bc, and ce, but we cannot disconnect it by removing just two of these edges. Note that a cut set is a set of edges in which no edge is redundant.

Cut-Vertex

A cut-vertex is a single vertex whose removal disconnects a graph.

It is important to note that the above definition breaks down if G is a complete graph, since we cannot then disconnects G by removing vertices. Therefore, we make the following definition.

Connectivity of Complete Graph

The connectivity k(kn) of the complete graph kn is n-1. When n-1 ≥ k, the graph kn is said to be k-connected.

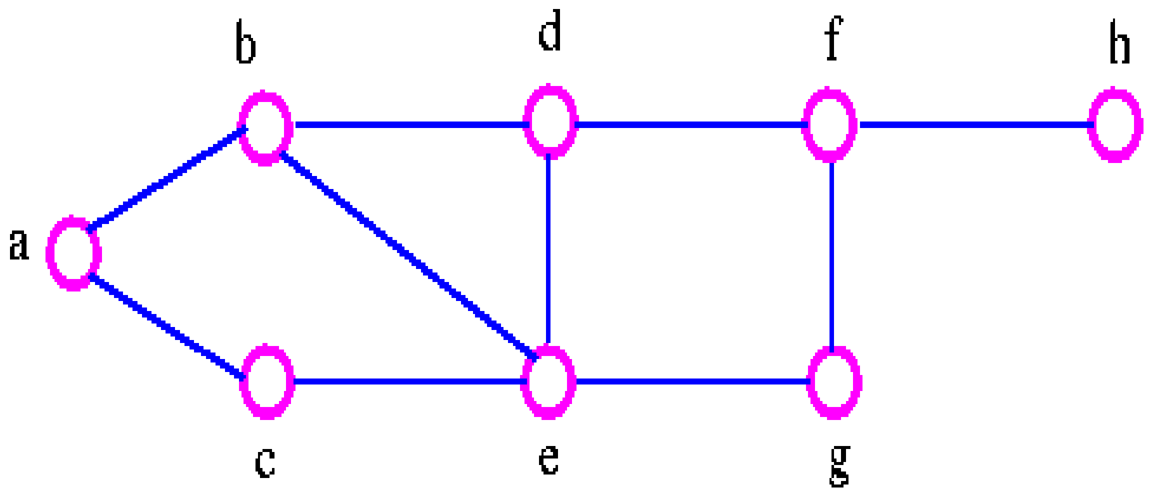
Vertex-Cut set

A vertex-cut set of a connected graph G is a set S of vertices with the following properties.

a. the removal of all the vertices in S disconnects G.

b. the removal of some (but not all) of vertices in S does not disconnects G.

Consider the following graph



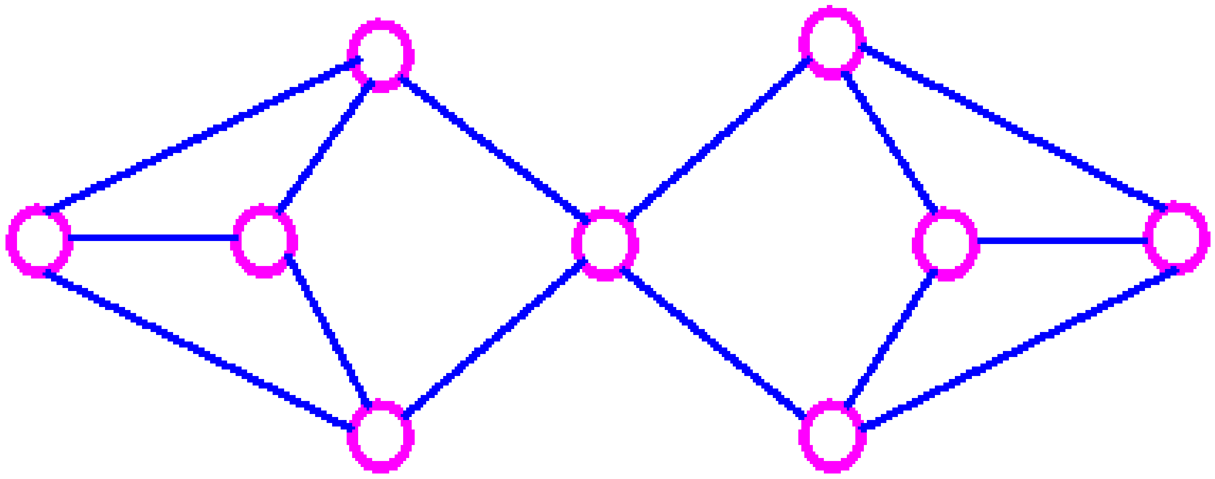
We can disconnects the graph by removing the two vertices b and e, but we cannot disconnect it by removing just one of these vertices. the vertex-cutset of G is {b, e}.

Note that the connectivity k(G) does not exceed the edge-connectivity λ(G). This inequality holds for all connected graph.

Formally, for any connected graph G we have

$K(G) \leq \lambda(G) \leq \delta(G)$

where δ(G) is the smallest vertex-degree in G. But it is certainly possible for both inequality in above theorem to be strict inequalities (that is, $k(G) < \lambda(G) < \delta(G)$) For example, in the following graph,



$K(G)=1$, $\lambda(G) = 2$, and $\delta(G) = 3$.

Cut Vertex

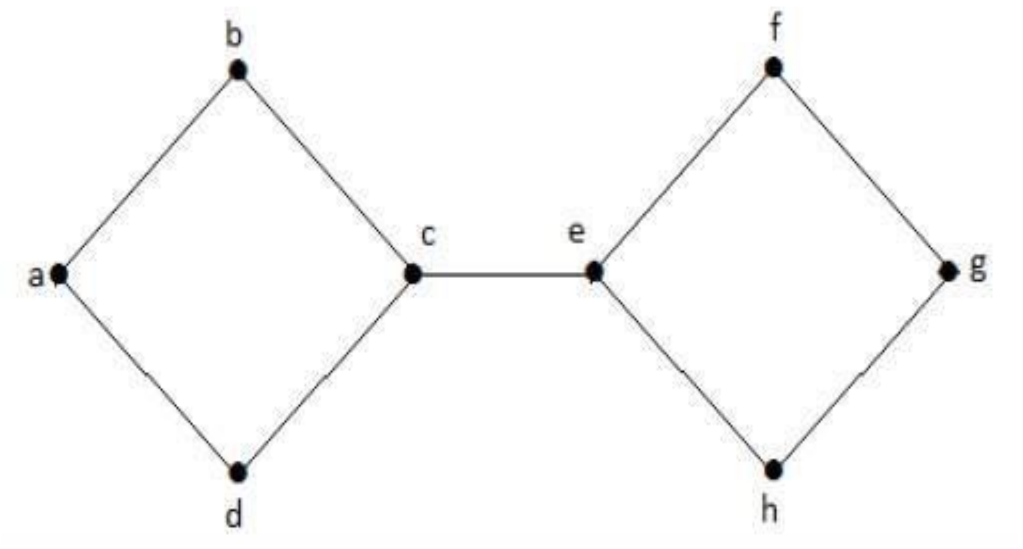
Let 'G' be a connected graph. A vertex V ∈ G is called a cut vertex of 'G', if 'G-V' (Delete 'V' from 'G') results in a disconnected graph. Removing a cut vertex from a graph breaks it in to two or more graphs.

Note − Removing a cut vertex may render a graph disconnected.

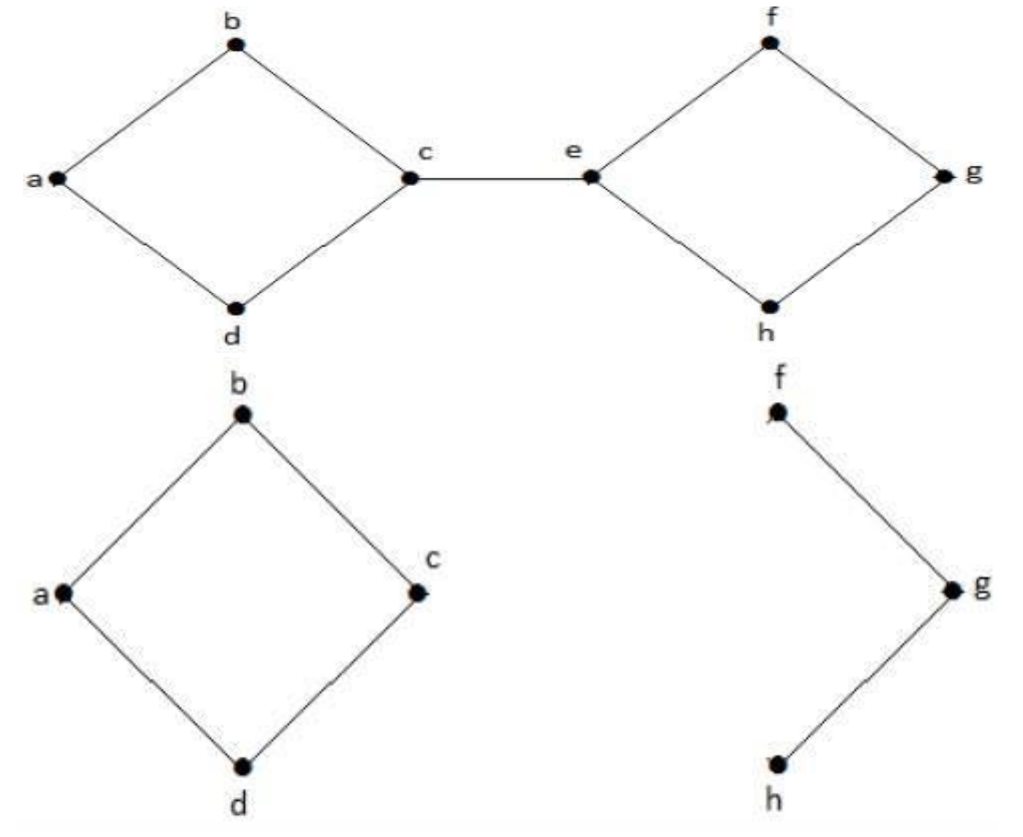
A connected graph 'G' may have at most (n–2) cut vertices.

Example

In the following graph, vertices 'e' and 'c' are the cut vertices.



By removing 'e' or 'c', the graph will become a disconnected graph.



Without 'g', there is no path between vertex 'c' and vertex 'h' and many other. Hence it is a disconnected graph with cut vertex as 'e'. Similarly, 'c' is also a cut vertex for the above graph.

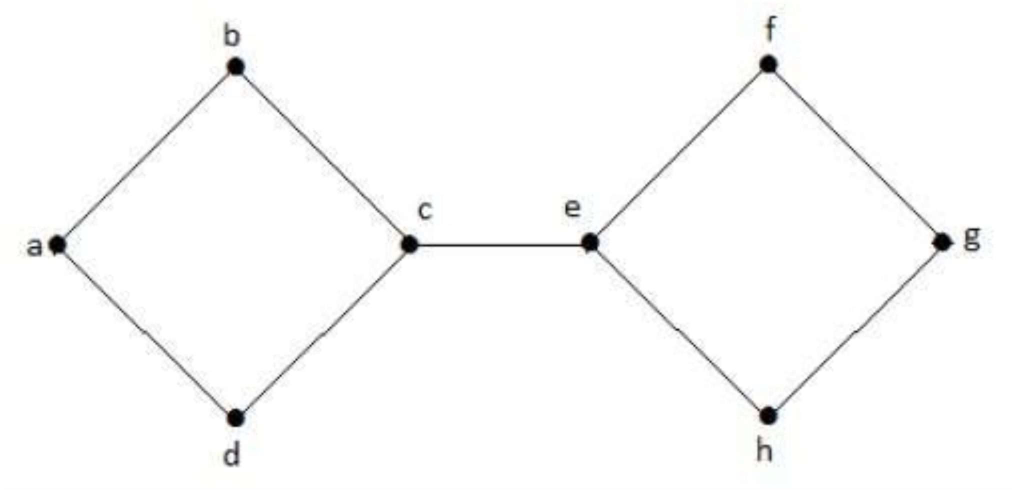
Cut Edge (Bridge)

Let 'G' be a connected graph. An edge 'e' ∈ G is called a cut edge if 'G-e' results in a disconnected graph.

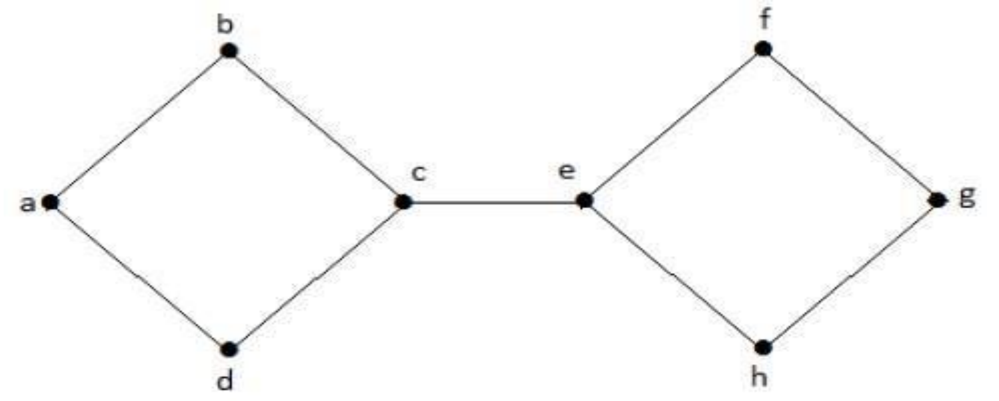
If removing an edge in a graph results in to two or more graphs, then that edge is called a Cut Edge.

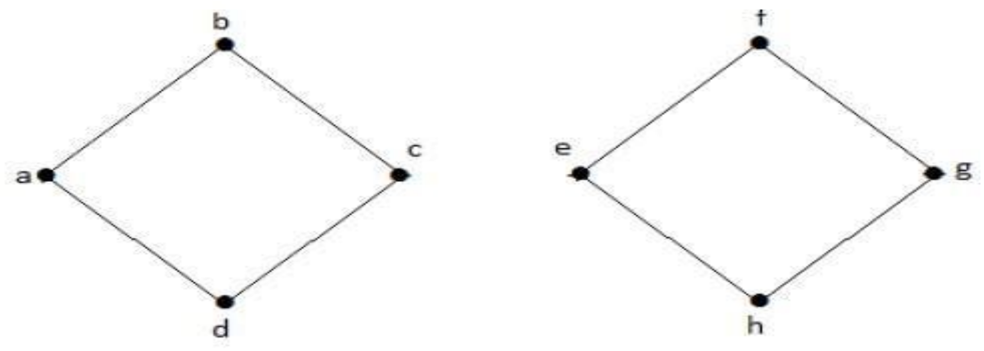
Example

In the following graph, the cut edge is [(c, e)]



By removing the edge (c, e) from the graph, it becomes a disconnected graph.





In the above graph, removing the edge (c, e) breaks the graph into two which is nothing but a disconnected graph. Hence, the edge (c, e) is a cut edge of the graph.

Note − Let 'G' be a connected graph with 'n' vertices, then

- a cut edge e ∈ G if and only if the edge 'e' is not a part of any cycle in G.
- the maximum number of cut edges possible is 'n-1'.
- whenever cut edges exist, cut vertices also exist because at least one vertex of a cut edge is a cut vertex.
- if a cut vertex exists, then a cut edge may or may not exist.

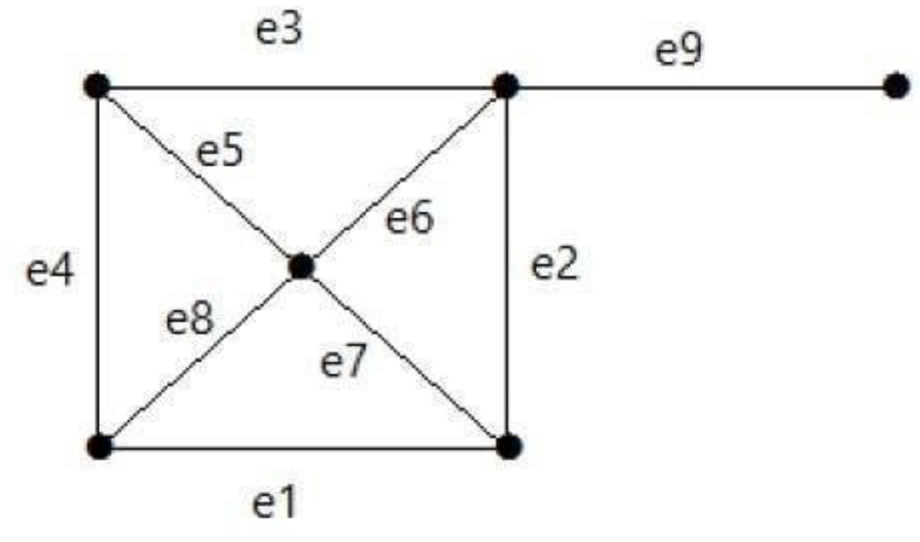
Cut Set of a Graph

Let 'G'= (V, E) be a connected graph. A subset E' of E is called a cut set of G if deletion of all the edges of E' from G makes G disconnect.

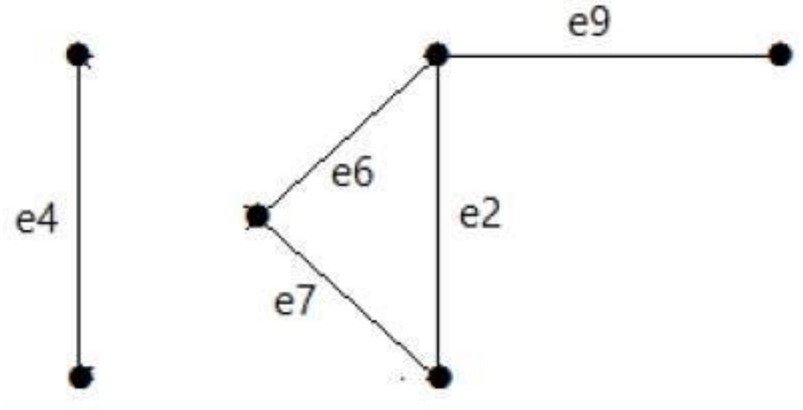
If deleting a certain number of edges from a graph makes it disconnected, then those deleted edges are called the cut set of the graph.

Example

Take a look at the following graph. Its cut set is E1 = {e1, e3, e5, e8}.



After removing the cut set E1 from the graph, it would appear as follows −

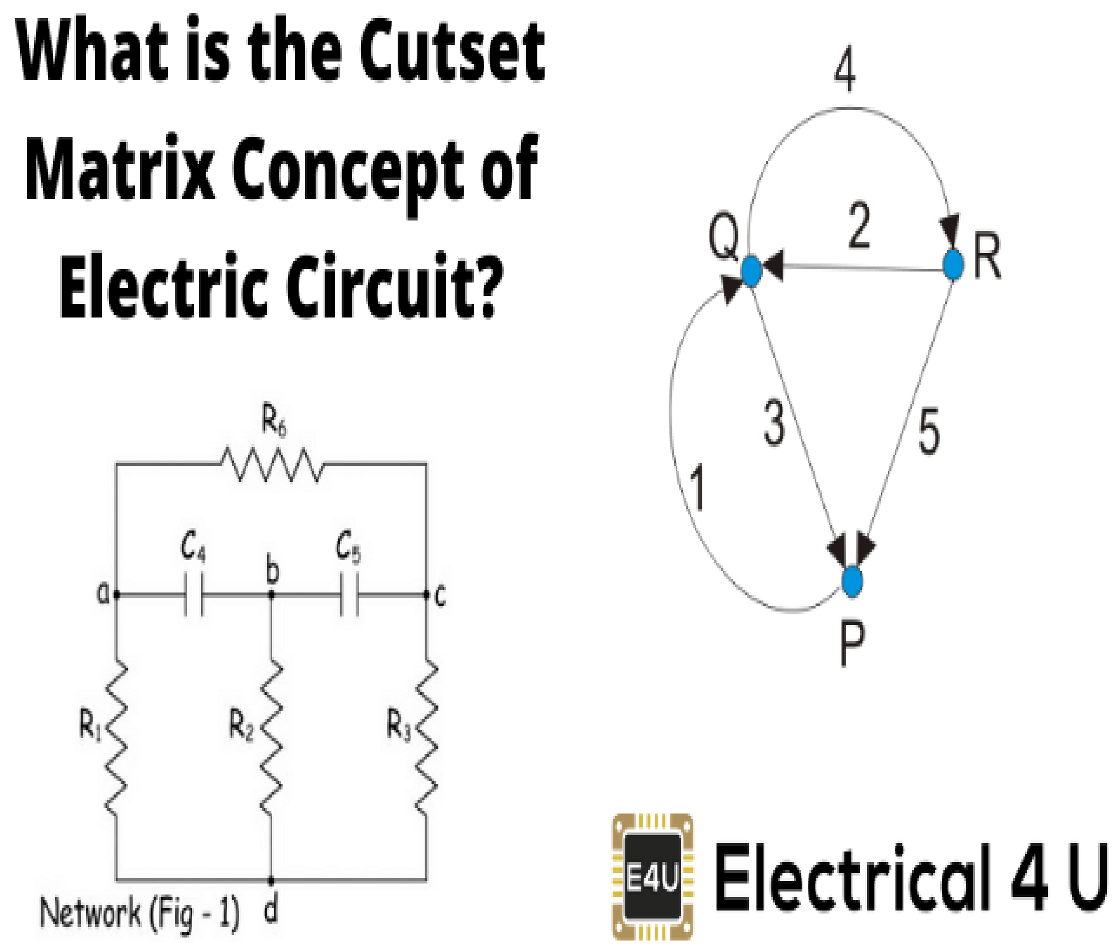


Similarly there are other cut sets that can disconnect the graph −

- E3 = {e9} – Smallest cut set of the graph.
- E4 = {e3, e4, e5}

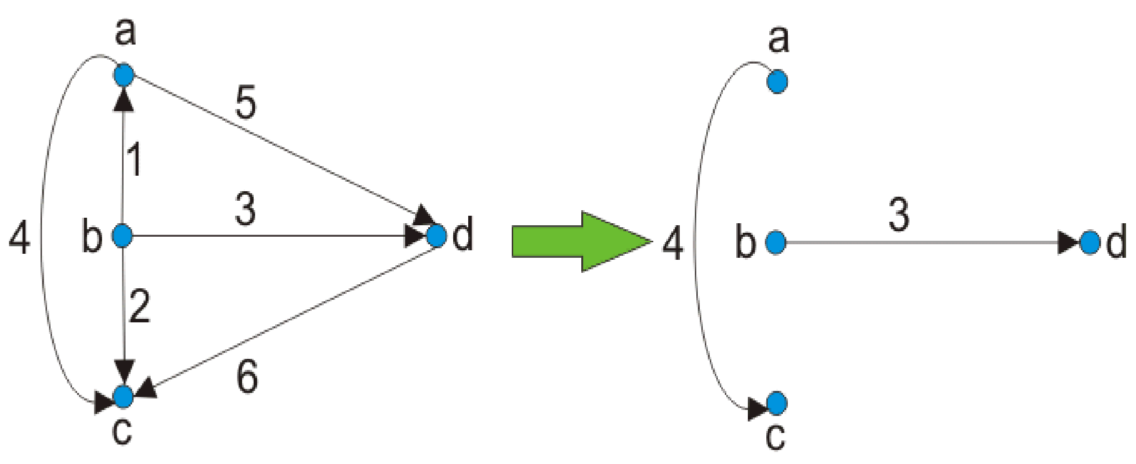
Fundamental circuit and cut sets

Cutset Matrix Concept of Electric Circuit



When we talk of cut set matrix in graph theory, we generally talk of fundamental cut-set matrix. A cut-set is a minimum set of branches of a connected graph such that when removed these branches from the graph, then the graph gets separated into 2 distinct parts called sub-graphs and the cut set matrix is the matrix which is obtained by row-wise taking one cut-set at a time. The cutset matrix is denoted by symbol [Qf].

Example of Cutsets Matrix of a Circuit



Two sub-graphs are obtained from a graph by selecting cut-sets consisting of branches [1, 2, 5, 6].
Thus, in other words we can say that fundamental cut set of a given graph with reference to a tree is a cut-set formed with one twig and remaining links. Twigs are the branches of tree and links are the branches of co-tree.
Thus, the number of cutset is equal to the number of twigs.
[Number of twigs = N – 1]
Where, N is the number of nodes of the given graph or drawn tree.

1.	Branchase ⇒	1	2	3	.	.	b
Cutsets							
C1							
C2							
C3							
.							
.							

The orientation of cut-set is the same as that of twig and that is taken positive.

Cn							

Steps to Draw Cut Set Matrix

There are some steps one should follow while drawing the cut-set matrix. The steps are as follows-

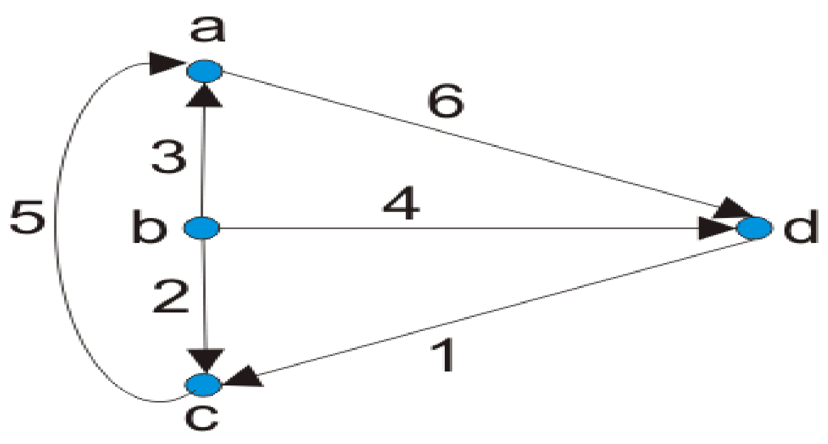
2. Draw the graph of given network or circuit (if given).
3. Then draw its tree. The branches of the tree will be twig.
4. Then draw the remaining branches of the graph by dotted line. These branches will be links.
5. Each branch or twig of tree will form an independent cut-set.
6. Write the matrix with rows as cut-set and column as branches.

n = number of cut-set.
b = number of branches.

Orientation in Cut Set Matrix

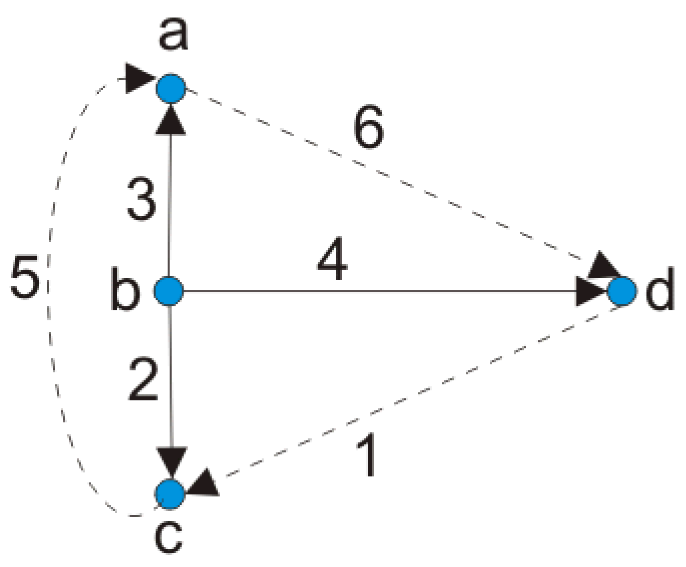
Qij = 1; if branch J is in cut-set with orientation same as that of tree branch.
Qij = -1; if branch J is in cut-set with orientation opposite to that of branch of tree.
Qij = 0; if branch J is not in cut-set.

Example 1

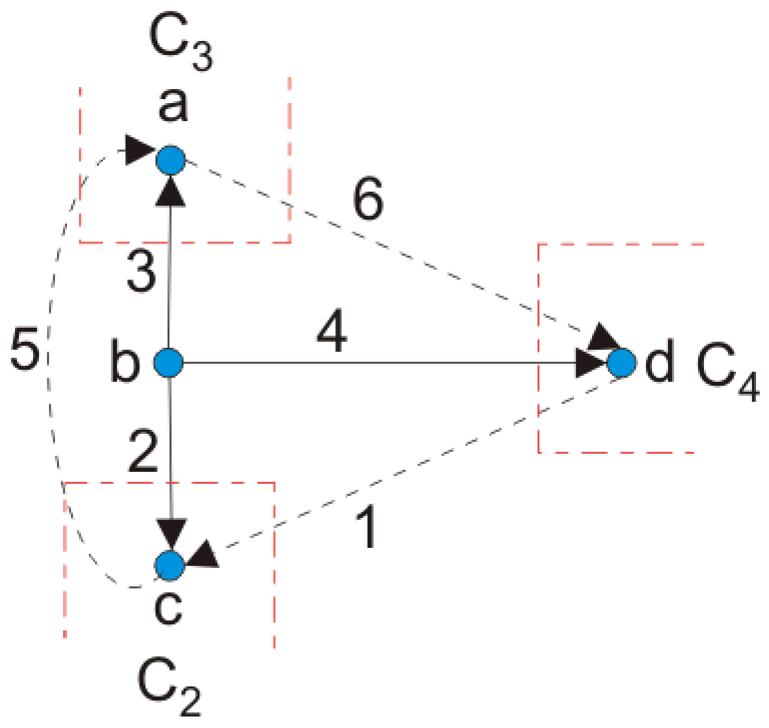


Draw the cut-set matrix for the following graph.
Answer:

Step 1: Draw the tree for the following graph.



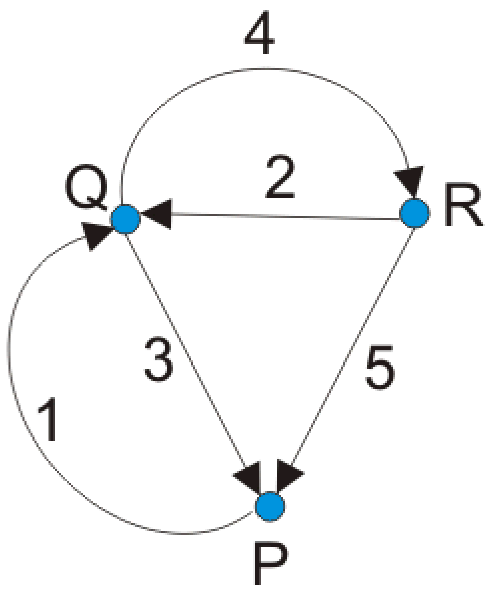
Step 2: Now identify the cut-set. Cut-set will be that node which will contain only one twig and any number of links.



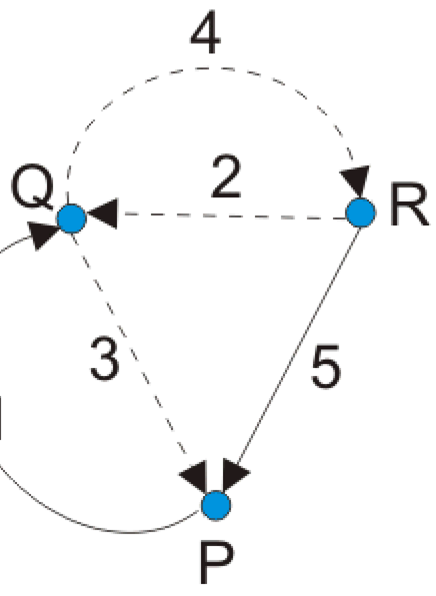
Here C2, C3 and C4 are cut-sets.
Step 3: Now draw the matrix.

	Branchase ⇒	1	2	3	4	5	6
Cutsets							
C2		+1	+1	0	0	-1	0
C3		0	0	+1	0	+1	-1
C4		-1	0	0	+1	0	+1

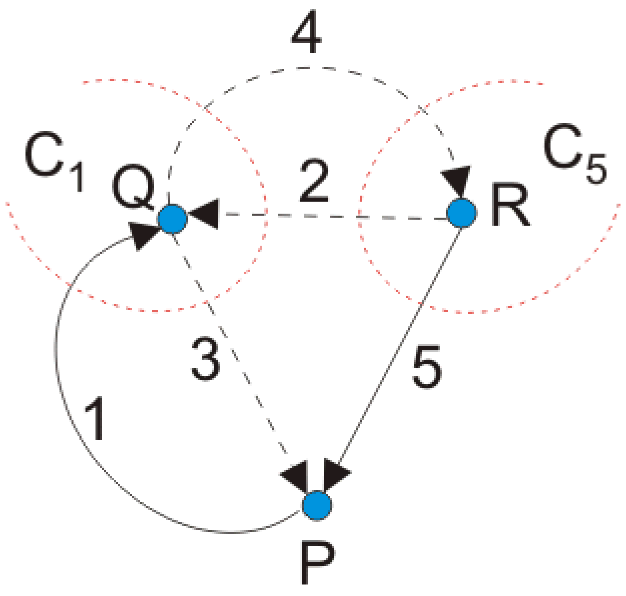
This is the required matrix.
Example 2:



Draw the cut-set of the given graph.
Answer:
Again in this question we have to repeat the same steps as done in previous question.
Step 1: Draw the tree for the following graph.



Step 2: Now identify the cut-set. Cut-set will be that node which will contain only one twig and any number of links.



Here C1 and C5 are cut-sets.
Step 3: Now draw the matrix.

	Branchase ⇒	1	2	3	4	5
Cutsets						
C1		+1	+1	-1	-1	0
C5		0	-1	0	-1	+1

This is the required matrix.
Points to remember

There are some key points which should be remembered. They are:-

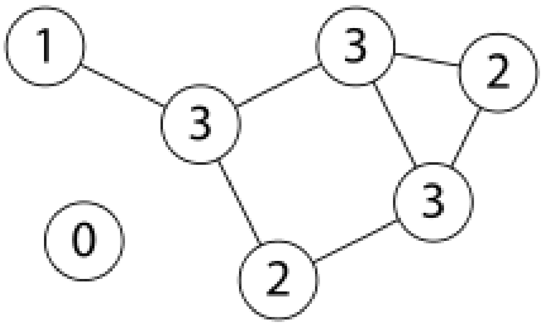
- In cutset matrix, the orientation of twig is taken positive.
- Each cut-set contains only one twig.
- Cut-set can have any number of links attached to it.
- The relation between cut-set matrix and KCL is that

$$[Q_f] \times I_b = 0$$

Connectivity and seperatability

In mathematics and computer science, connectivity is one of the basic concepts of graph theory: it asks for the minimum number of elements (nodes or edges) that need to be removed to separate the remaining nodes into isolated subgraphs.[1] It is closely related to the theory of network flow problems. The connectivity of a graph is an important measure of its resilience as a network.

Connected vertices and graphs



With vertex 0, this graph is disconnected. The rest of the graph is connected.

In an undirected graph G, two vertices u and v are called connected if G contains a path from u to v. Otherwise, they are called disconnected. If the two vertices are additionally connected by a path of length 1, i.e. by a single edge, the vertices are called adjacent.

A graph is said to be connected if every pair of vertices in the graph is connected. This means that there is a path between every pair of vertices. An undirected graph that is not connected is called disconnected. An undirected graph G is therefore disconnected if there exist two vertices in G such that no path in G has these vertices as endpoints. A graph with just one vertex is connected. An edgeless graph with two or more vertices is disconnected.

A directed graph is called weakly connected if replacing all of its directed edges with undirected edges produces a connected (undirected) graph. It is unilaterally connected or unilateral (also called semiconnected) if it contains a directed path from u to v or a directed path from v to u for every pair of vertices u, v.[2] It is strongly connected, or simply strong, if it contains a directed path from u to v and a directed path from v to u for every pair of vertices u, v

Components and cuts

A connected component is a maximal connected subgraph of an undirected graph. Each vertex belongs to exactly one connected component, as does each edge. A graph is connected if and only if it has exactly one connected component.

The strong components are the maximal strongly connected subgraphs of a directed graph.

A vertex cut or separating set of a connected graph G is a set of vertices whose removal renders G disconnected. The vertex connectivity κ(G) (where G is not a complete graph) is the size of a minimal vertex cut. A graph is called k-vertex-connected or k-connected if its vertex connectivity is k or greater.

More precisely, any graph G (complete or not) is said to be k-vertex-connected if it contains at least k+1 vertices, but does not contain a set of k − 1 vertices whose removal disconnects the graph; and κ(G) is defined as the largest k such that G is k-connected. In particular, a complete graph with n vertices, denoted Kn, has no vertex cuts at all, but κ(Kn) = n − 1.

A vertex cut for two vertices u and v is a set of vertices whose removal from the graph disconnects u and v. The local connectivity κ(u, v) is the size of a smallest vertex cut separating u and v. Local connectivity is symmetric for undirected graphs; that is, κ(u, v) = κ(v, u). Moreover, except for complete graphs, κ(G) equals the minimum of κ(u, v) over all nonadjacent pairs of vertices u, v.

2-connectivity is also called biconnectivity and 3-connectivity is also called triconnectivity. A graph G which is connected but not 2-connected is sometimes called separable.

Analogous concepts can be defined for edges. In the simple case in which cutting a single, specific edge would disconnect the graph, that edge is called a bridge. More generally, an edge cut of G is a set of edges whose removal renders the graph disconnected. The edge-connectivity λ(G) is the size of a smallest edge cut, and the local edge-connectivity λ(u, v) of two vertices u, v is the size of a smallest edge cut disconnecting u from v. Again, local edge-connectivity is symmetric. A graph is called k-edge-connected if its edge connectivity is k or greater.

A graph is said to be maximally connected if its connectivity equals its minimum degree. A graph is said to be maximally edge-connected if its edge-connectivity equals its minimum degree.[3]

Super- and hyper-connectivity

A graph is said to be super-connected or super-κ if every minimum vertex cut isolates a vertex. A graph is said to be hyper-connected or hyper-κ if the deletion of each minimum vertex cut creates exactly two components, one of which is an isolated vertex. A graph is semi-hyper-connected or semi-hyper-κ if any minimum vertex cut separates the graph into exactly two components.[4]

More precisely: a G connected graph is said to be super-connected or super-κ if all minimum vertex-cuts consist of the vertices adjacent with one (minimum-degree) vertex. A G connected graph is said to be super-edge-connected or super-λ if all minimum edge-cuts consist of the edges incident on some (minimum-degree) vertex.[5]

A cutset X of G is called a non-trivial cutset if X does not contain the neighborhood N(u) of any vertex $u \notin X$. Then the superconnectivity κ1 of G is:

$$\kappa 1(G) = \min\{|X| : X \text{ is a non-trivial cutset}\}.$$

A non-trivial edge-cut and the edge-superconnectivity λ1(G) are defined analogously.[6]

Menger's theorem

One of the most important facts about connectivity in graphs is Menger's theorem, which characterizes the connectivity and edge-connectivity of a graph in terms of the number of independent paths between vertices.

If u and v are vertices of a graph G, then a collection of paths between u and v is called independent if no two of them share a vertex (other than u and v themselves). Similarly, the collection is edge-independent if no two paths in it share an edge. The number of mutually independent paths between u and v is written as κ′(u, v), and the number of mutually edge-independent paths between u and v is written as λ′(u, v).

Menger's theorem asserts that for distinct vertices u,v, λ(u, v) equals λ′(u, v), and if u is also not adjacent to v then κ(u, v) equals κ′(u, v).[7][8] This fact is actually a special case of the max-flow min-cut theorem.

Computational aspects

The problem of determining whether two vertices in a graph are connected can be solved efficiently using a search algorithm, such as breadth-first search. More generally, it is easy to determine computationally whether a graph is connected (for example, by using a disjoint-set data structure), or to count the number of connected components. A simple algorithm might be written in pseudo-code as follows:

1. Begin at any arbitrary node of the graph, G

2. Proceed from that node using either depth-first or breadth-first search, counting all nodes reached.
3. Once the graph has been entirely traversed, if the number of nodes counted is equal to the number of nodes of G, the graph is connected; otherwise it is disconnected.

By Menger's theorem, for any two vertices u and v in a connected graph G, the numbers κ(u, v) and λ(u, v) can be determined efficiently using the max-flow min-cut algorithm. The connectivity and edge-connectivity of G can then be computed as the minimum values of κ(u, v) and λ(u, v), respectively.

In computational complexity theory, SL is the class of problems log-space reducible to the problem of determining whether two vertices in a graph are connected, which was proved to be equal to L by Omer Reingold in 2004.[9] Hence, undirected graph connectivity may be solved in O(log n) space.

The problem of computing the probability that a Bernoulli random graph is connected is called network reliability and the problem of computing whether two given vertices are connected the ST-reliability problem. Both of these are #P-hard.[10]

Number of connected graphs

The number of distinct connected labeled graphs with n nodes is tabulated in the On-Line Encyclopedia of Integer Sequences as sequence A001187, through n = 16. The first few non-trivial terms are

n	graphs
2	1
3	4
4	38
5	728
6	26704
7	1866256
8	251548592

Bounds on connectivity

- The vertex-connectivity of a graph is less than or equal to its edge-connectivity. That is, $\kappa(G) \le \lambda(G)$. Both are less than or equal to the minimum degree of the graph, since deleting all neighbors of a vertex of minimum degree will disconnect that vertex from the rest of the graph.[1]

- For a vertex-transitive graph of degree d, we have: $2(d + 1)/3 \le \kappa(G) \le \lambda(G) = d$.[11]

- For a vertex-transitive graph of degree $d \le 4$, or for any (undirected) minimal Cayley graph of degree d, or for any symmetric graph of degree d, both kinds of connectivity are equal: $\kappa(G) = \lambda(G) = d$.[12]

Other properties

- Connectedness is preserved by graph homomorphisms.

- If G is connected then its line graph L(G) is also connected.

- A graph G is 2-edge-connected if and only if it has an orientation that is strongly connected.

- Balinski's theorem states that the polytopal graph (1-skeleton) of a k-dimensional convex polytope is a k-vertex-connected graph.[13] Steinitz's previous theorem that any 3-vertex-connected planar graph is a polytopal graph (Steinitz theorem) gives a partial converse.

- According to a theorem of G. A. Dirac, if a graph is k-connected for $k \ge 2$, then for every set of k vertices in the graph there is a cycle that passes through all the vertices in the set.[14][15] The converse is true when $k = 2$.

Network flow

In graph theory, a flow network (also known as a transportation network) is a directed graph where each edge has a capacity and each edge receives a flow. The amount of flow on an edge cannot exceed the capacity of the edge. Often in operations research, a directed graph is called a network, the vertices are called nodes and the edges are called arcs. A flow must satisfy the restriction that the amount of flow into a node equals the amount of flow out of it, unless it is a source, which has only outgoing flow, or sink, which has only incoming flow. A network can be used to model traffic in a computer network, circulation with demands, fluids in pipes, currents in an electrical circuit, or anything similar in which something travels through a network of nodes.

Definition

A network is a graph G = (V, E), where V is a set of vertices and E is a set of V’s edges – a subset of V × V – together with a non-negative function $c: V \times V \to \mathbb{R}^{\infty}$, called the capacity function. Without loss of generality, we may assume that if (u, v) ∈ E then (v, u) is also a member of E, since if (v, u) ∉ E then we may add (v, u) to E and then set c(v, u) = 0.

If two nodes in G are distinguished, a source s and a sink t, then (G, c, s, t) is called a flow network.[1]

Flows

There are various notions of a flow function that can be defined in a flow graph. Flow functions model the net flow of units between pairs of nodes, and are useful when asking questions such as what is the maximum number of units that can be transferred from the source node s to the sink node t? The simplest example of a flow function is known as a pseudo-flow.

A pseudo-flow is a function $f : V \times V \to \mathbb{R}$ that satisfies the following two constraints for all nodes u and v:

- Skew symmetry: Only encode the net flow of units between a pair of nodes u and v (see intuition below), that is: f (u, v) = −f (v, u).
- Capacity constraint: An arc's flow cannot exceed its capacity, that is: f (u, v) ≤ c(u, v).

Given a pseudo-flow f in a flow network, it is often useful to consider the net flow entering a given node v, that is, the sum of the flows entering v.
The excess function $xf : V \to \mathbb{R}$ is defined by xf (u) = ∑v ∈ V f (v, u). A node u is said to be active if xf (u) > 0, deficient if xf (u) < 0 or conserving if xf (u) = 0.

These final definitions lead to two strengthenings of the definition of a pseudo-flow:

A pre-flow is a pseudo-flow that, for all v ∈ V \{s}, satisfies the additional constraint:

- Non-deficient flows: The net flow entering the node v is non-negative, except for the source, which "produces" flow. That is: xf (v) ≥ 0 for all v ∈ V \{s}.

A feasible flow, or just a flow, is a pseudo-flow that, for all v ∈ V \{s, t}, satisfies the additional constraint:

- Flow conservation: The net flow entering the node v is 0, except for the source, which "produces" flow, and the sink, which "consumes" flow. That is: xf (v) = 0 for all v ∈ V \{s, t}.

The value of a feasible flow f, denoted | f |, is the net flow into the sink t of the flow network. That is, | f | = xf (t).

Intuition

In the context of flow analysis, there is only an interest in considering how units are transferred between nodes in a holistic sense. Put another way, it is not necessary to distinguish multiple arcs between a pair of nodes:

Given any two nodes u and v, if there are two arcs from u to v with capacities 5 and 3 respectively, this is equivalent to considering only a single arc between u and v with capacity 8 — it is only useful to know that 8 units can be transferred from u to v, not how they can be transferred.

Again, given two nodes u and v, if there is a flow of 5 units from u to v, and another flow of 3 units from v to u, this is equivalent to a net flow of 2 units from u to v, or a net flow of −2 units from v to u (so sign indicates direction) — it is only useful to know that a net flow of 2 units will flow between u and v, and the direction that they will flow, not how that net flow is achieved.

For this reason, the capacity function $c: V \times V \to \mathbb{R}_\infty$, which does not allow for multiple arcs starting and ending at the same nodes, is sufficient for flow analysis. Similarly, it is enough to impose the skew symmetry constraint on flow functions to ensure that flow between two vertices is encoded by a single number (to indicate magnitude), and a sign (to indicate direction) — by knowing the flow between u and v you implicitly, via skew symmetry, know the flow between v and u. These simplifications of the model aren't always immediately intuitive, but they are convenient when it comes time to analyze flows.

The capacity constraint simply ensures that a flow on any one arc within the network cannot exceed the capacity of that arc.

Concepts useful to flow problems

The residual capacity of an arc with respect to a pseudo-flow f, denoted cf, is the difference between the arc's capacity and its flow. That is, cf (e) = c(e) - f(e). From this we can construct a residual network, denoted Gf (V, Ef), which models the amount of available capacity on the set of arcs in G = (V, E). More formally, given a flow network G, the residual network Gf has the node set V, arc set Ef = {e ∈ V × V : cf (e) > 0} and capacity function cf.

This concept is used in Ford–Fulkerson algorithm which computes the maximum flow in a flow network.

Note that there can be a path from u to v in the residual network, even though there is no path from u to v in the original network. Since flows in opposite directions cancel out, decreasing the flow from v to u is the same as increasing the flow from u to v.

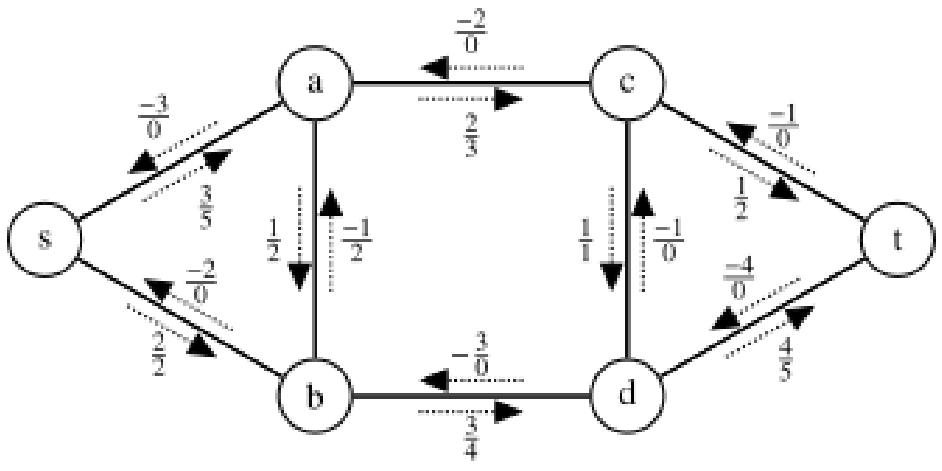
Augmenting paths

An augmenting path is a path (u1, u2, ..., uk) in the residual network, where u1 = s, uk = t, and cf (ui, ui + 1) > 0. A network is at maximum flow if and only if there is no augmenting path in the residual network Gf.

Multiple sources and/or sink

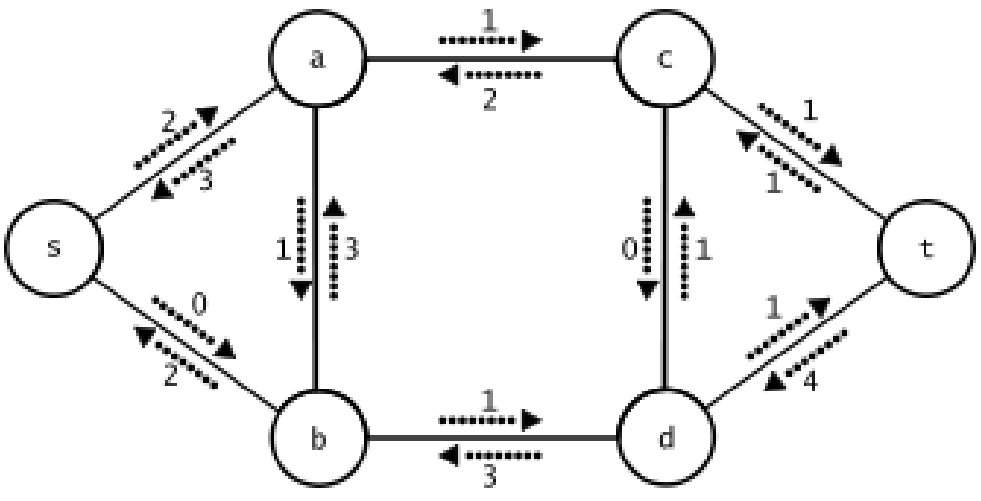
Sometimes, when modeling a network with more than one source, a supersource is introduced to the graph.[2] This consists of a vertex connected to each of the sources with edges of infinite capacity, so as to act as a global source. A similar construct for sinks is called a supersink.[3]

Example



A flow network showing flow and capacity

To the left you see a flow network with source labeled s, sink t, and four additional nodes. The flow and capacity is denoted . Notice how the network upholds skew symmetry, capacity constraints and flow conservation. The total amount of flow from s to t is 5, which can be easily seen from the fact that the total outgoing flow from s is 5, which is also the incoming flow to t. We know that no flow appears or disappears in any of the other nodes.



Residual network for the above flow network, showing residual capacities.

Below you see the residual network for the given flow. Notice how there is positive residual capacity on some edges where the original capacity is zero, for example for the edge . This flow is not a maximum flow. There is available capacity along the paths , and , which are then the augmenting paths. The residual capacity of the first path is.[citation needed] Notice that as long as there exists some path with a positive residual capacity, the flow will not be maximum. The residual capacity for some path is the minimum residual capacity of all edges in that path.

Applications

Picture a series of water pipes, fitting into a network. Each pipe is of a certain diameter, so it can only maintain a flow of a certain amount of water. Anywhere that pipes meet, the total amount of water coming into that junction must be equal to the amount going out, otherwise we would quickly run out of water, or we would have a buildup of water. We have a water inlet, which is the source, and an outlet, the sink. A flow would then be one possible way for water to get from source to sink so that the total amount of water coming out of the outlet is consistent. Intuitively, the total flow of a network is the rate at which water comes out of the outlet.

Flows can pertain to people or material over transportation networks, or to electricity over electrical distribution systems. For any such physical network, the flow coming into any intermediate node needs to equal the flow going out of that node. This conservation constraint is equivalent to Kirchhoff's current law.

Flow networks also find applications in ecology: flow networks arise naturally when considering the flow of nutrients and energy between different organisms in a food web. The mathematical problems associated with such networks are quite different from those that arise in networks of fluid or traffic flow. The field of ecosystem network analysis, developed by Robert Ulanowicz and others, involves using concepts from information theory and thermodynamics to study the evolution of these networks over time.

Classifying flow problems

The simplest and most common problem using flow networks is to find what is called the maximum flow, which provides the largest possible total flow from the source to the sink in a given graph. There are many other problems which can be solved using max flow algorithms, if they are appropriately modeled as flow networks, such as bipartite matching, the assignment problem and the transportation problem. Maximum flow problems can be solved efficiently with the push–relabel algorithm. The max-flow min-cut theorem states that finding a maximal network flow is equivalent to finding a cut of minimum capacity that separates the source and the sink, where a cut is the division of vertices such that the source is in one division and the sink is in another.

Well-known algorithms for the Maximum Flow Problem

Inventor(s)	Year	Time complexity (with n nodes and m arcs)
Dinic's algorithm	1969	O(mn2)
Edmonds–Karp algorithm	1972	O(m2n)

MPM (Malhotra, Pramodh-Kumar and Maheshwari) algorithm[4]	1978	O(n3)
James B. Orlin[5]	2013	O(mn)

In a multi-commodity flow problem, you have multiple sources and sinks, and various "commodities" which are to flow from a given source to a given sink. This could be for example various goods that are produced at various factories, and are to be delivered to various given customers through the same transportation network.

In a minimum cost flow problem, each edge has a given cost , and the cost of sending the flow across the edge is . The objective is to send a given amount of flow from the source to the sink, at the lowest possible price.

In a circulation problem, you have a lower bound on the edges, in addition to the upper bound . Each edge also has a cost. Often, flow conservation holds for all nodes in a circulation problem, and there is a connection from the sink back to the source. In this way, you can dictate the total flow with and . The flow circulates through the network, hence the name of the problem.

In a network with gains or generalized network each edge has a gain, a real number (not zero) such that, if the edge has gain g, and an amount x flows into the edge at its tail, then an amount gx flows out at the head.

In a source localization problem, an algorithm tries to identify the most likely source node of information diffusion through a partially observed network. This can be done in linear time for trees and cubic time for arbitrary networks and has applications ranging from tracking mobile phone users to identifying the originating source of disease outbreaks.[6]

mincut theorem

In computer science and optimization theory, the max-flow min-cut theorem states that in a flow network, the maximum amount of flow passing from the source to the sink is equal to the total weight of the edges in a minimum cut, i.e. the smallest total weight of the edges which if removed would disconnect the source from the sink.

The max-flow min-cut theorem is a special case of the duality theorem for linear programs and can be used to derive Menger's theorem and the Kőnig–Egerváry theorem.[1]

Definitions and statement

The theorem relates two quantities: the maximum flow through a network, and the minimum capacity of a cut of the network, that is, the minimum capacity is achieved by the flow.

To state the theorem, each of these quantities must first be defined.

Let N = (V, E) be a directed graph, where V denotes the set of vertices and E is the set of edges. Let s ∈ V and t ∈ V be the source and the sink of N, respectively.

The capacity of an edge is a mapping denoted by cuv or c(u, v) where u,v ∈ V. It represents the maximum amount of flow that can pass through an edge.

Flows

A flow is a mapping denoted by or , subject to the following two constraints:

1. Capacity Constraint: For every edge ,

2. Conservation of Flows: For each vertex apart from and (i.e. the source and sink, respectively), the following equality holds:

A flow can be visualized as a physical flow of a fluid through the network, following the direction of each edge. The capacity constraint then says that the volume flowing through each edge per unit time is less than or equal to the maximum capacity of the edge, and the conservation constraint says that the amount that flows into each vertex equals the amount flowing out of each vertex, apart from the source and sink vertices.

Cuts

The other half of the max-flow min-cut theorem refers to a different aspect of a network: the collection of cuts. An s-t cut C = (S, T) is a partition of V such that s ∈ S and t ∈ T. That is, s-t cut is a division of the vertices of the network into two parts, with the source in one part and the sink in the other. The cut-set of a cut C is the set of edges that connect the source part of the cut to the sink part:

{\displaystyle X_{C}:=\{(u,v)\in E\ :\ u\in S,v\in T\}=(S\times T)\cap E.}Thus, if all the edges in the cut-set of C are removed, then no positive flow is possible, because there is no path in the resulting graph from the source to the sink.

The capacity of an s-t cut is the total weight of its edges,

{\displaystyle c(S,T)=\sum \nolimits _{(u,v)\in X_{C}}c_{uv}=\sum \nolimits _{(i,j)\in E}c_{ij}d_{ij},}where {\displaystyle d_{ij}=1} if {\displaystyle i\in S} and {\displaystyle j\in T}, {\displaystyle 0} otherwise.

There are typically many cuts in a graph, but cuts with smaller weights are often more difficult to find.

Minimum s-t Cut Problem. Minimize c(S, T), that is, determine S and T such that the capacity of the S-T cut is minimal.

Main theorem

The main theorem links the maximum flow through a network with the minimum cut of the network.

Max-flow min-cut theorem. The maximum value of an s-t flow is equal to the minimum capacity over all s-t cuts.

Linear program formulation

The max-flow problem and min-cut problem can be formulated as two primal-dual linear programs.

	Max-flow (Primal)	Min-cut (Dual)
variables	[a variable per edge]	[a variable per edge] [a variable per non-terminal node]
objective	maximize [max total flow from source]	minimize [min total capacity of edges in cut]
constraints	subject to [a constraint per edge and a constraint per non-terminal node]	subject to [a constraint per edge]
sign constraints		

The max-flow LP is straightforward. The dual LP is obtained using the algorithm described in dual linear program. The resulting LP requires some explanation. The interpretation of the variables in the min-cut LP is:

The minimization objective sums the capacity over all the edges that are contained in the cut.

The constraints guarantee that the variables indeed represent a legal cut:[3]

The constraints (equivalent to) guarantee that, for non-terminal nodes u,v, if u is

in S and v is in T, then the edge (u,v) is counted in the cut ( ).

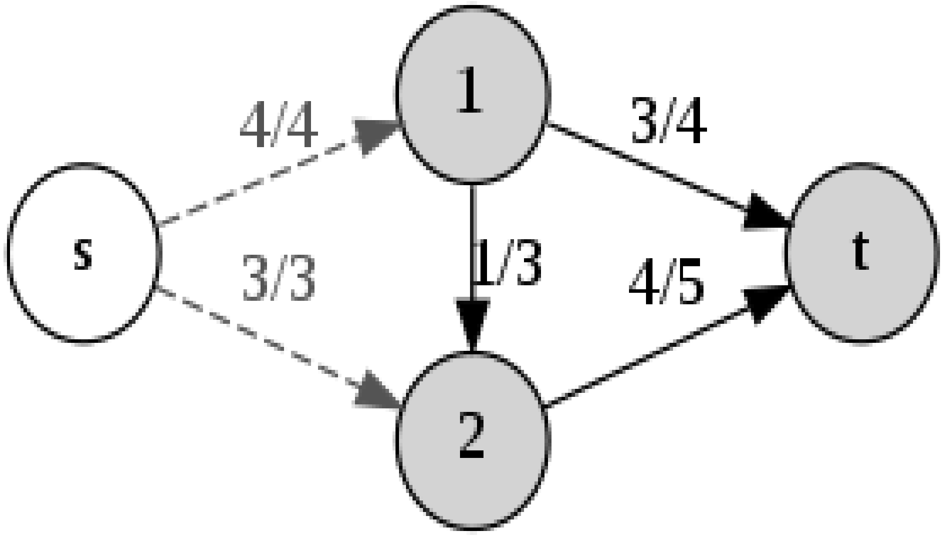
The constraints (equivalent to) guarantee that, if v is in T, then the edge (s,v) is counted in the cut (since s is by definition in S).

The constraints (equivalent to) guarantee that, if u is in S, then the edge (u,t) is counted in the cut (since t is by definition in T).

Note that, since this is a minimization problem, we do not have to guarantee that an edge is not in the cut - we only have to guarantee that each edge that should be in the cut, is summed in the objective function.

The equality in the max-flow min-cut theorem follows from the strong duality theorem in linear programming, which states that if the primal program has an optimal solution, x*, then the dual program also has an optimal solution, y*, such that the optimal values formed by the two solutions are equal.

Example



A network with the value of flow equal to the capacity of an s-t cut

The figure on the right is a network having a value of flow of 7. The numerical annotation on each arrow, in the form x/y, indicates the flow (x) and the capacity (y). The flows emanating from the source total seven (4+3=7), as do the flows into the sink (3+4=7).

The vertex in white and the vertices in grey form the subsets S and T of an s-t cut, whose cut-set contains the dashed edges. Since the capacity of the s-t cut is 7, which equals the value of flow, the max-flow min-cut theorem indicates that the value of flow and the capacity of the s-t cut are both optimal in this network.

Note that the flow through each of the dashed edges is at full capacity: this is the 'bottleneck' of the system. By contrast there is spare capacity in the right-hand part of the network. In particular, the flow from node one to node two need not be equal to 1. If there were no flow between nodes one and two, then the inputs to the sink would change to 4/4 and 3/5; the total flow would still be seven (4+3=7). On the other hand, if the flow from node one to node two were doubled to 2, then the inputs to the sink would change to 2/4 and 5/5; the total flow would again remain at seven (2+5=7).

Application

Cederbaum's maximum flow theorem

The maximum flow problem can be formulated as the maximization of the electrical current through a network composed of nonlinear resistive elements.[4] In this formulation, the limit of the current Iin between the input terminals of the electrical network as the input voltage Vin approaches , is equal to the weight of the minimum-weight cut set.

Generalized max-flow min-cut theorem

In addition to edge capacity, consider there is capacity at each vertex, that is, a mapping denoted by c(v), such that the flow f has to satisfy not only the capacity constraint and the conservation of flows, but also the vertex capacity constraint

In other words, the amount of flow passing through a vertex cannot exceed its capacity. Define an s-t cut to be the set of vertices and edges such that for any path from s to t, the path contains a member of the cut. In this case, the capacity of the cut is the sum the capacity of each edge and vertex in it.

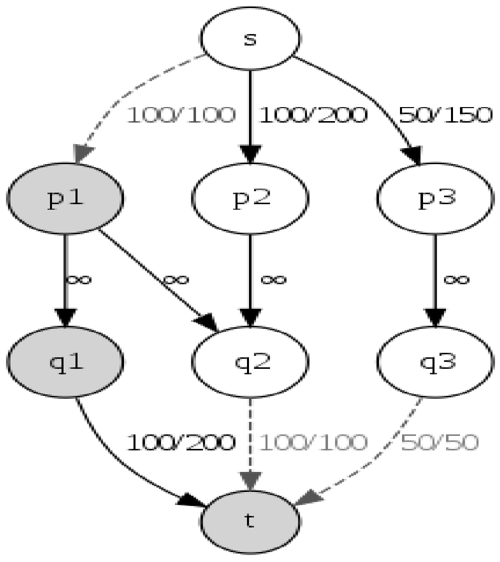
In this new definition, the generalized max-flow min-cut theorem states that the maximum value of an s-t flow is equal to the minimum capacity of an s-t cut in the new sense.

Menger's theorem

In the undirected edge-disjoint paths problem, we are given an undirected graph G = (V, E) and two vertices s and t, and we have to find the maximum number of edge-disjoint s-t paths in G.

The Menger's theorem states that the maximum number of edge-disjoint s-t paths in an undirected graph is equal to the minimum number of edges in an s-t cut-set.

Project selection problem



A network formulation of the project selection problem with the optimal solution

In the project selection problem, there are n projects and m machines. Each project pi yields revenue r(pi) and each machine qj costs c(qj) to purchase. Each project requires a number of machines and each machine can be shared by several projects. The problem is to determine which projects and machines should be selected and purchased respectively, so that the profit is maximized.

Let P be the set of projects not selected and Q be the set of machines purchased, then the problem can be formulated as,

Since the first term does not depend on the choice of P and Q, this maximization problem can be formulated as a minimization problem instead, that is,

The above minimization problem can then be formulated as a minimum-cut problem by constructing a network, where the source is connected to the projects with capacity r(pi), and the sink is connected by the machines with capacity c(qj). An edge (pi, qj) with infinite capacity is added if project pi requires machine qj. The s-t cut-set represents the projects and machines in P and Q respectively. By the max-flow min-cut theorem, one can solve the problem as a maximum flow problem.

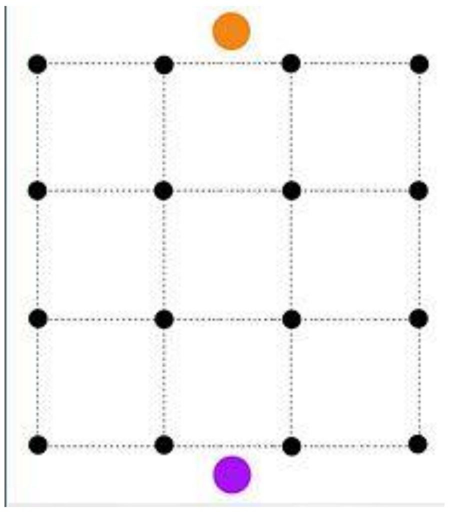
	Project r(pi)	Machine c(qj)	
1	100	200	Project 1 requires machines 1 and 2.
2	200	100	Project 2 requires machine 2.
3	150	50	Project 3 requires machine 3.

The figure on the right gives a network formulation of the following project selection problem:

The minimum capacity of an s-t cut is 250 and the sum of the revenue of each project is 450; therefore the maximum profit g is 450 − 250 = 200, by selecting projects p2 and p3.

The idea here is to 'flow' the project profits through the 'pipes' of the machine. If we cannot fill the pipe, the machine's return is less than its cost, and the min cut algorithm will find it cheaper to cut the project's profit edge instead of the machine's cost edge.

Image segmentation problem



Each black node denotes a pixel.

In the image segmentation problem, there are n pixels. Each pixel i can be assigned a foreground value fi or a background value bi. There is a penalty of pij if pixels i, j are adjacent and have different assignments. The problem is to assign pixels to foreground or background such that the sum of their values minus the penalties is maximum.

Let P be the set of pixels assigned to foreground and Q be the set of points assigned to background, then the problem can be formulated as,

This maximization problem can be formulated as a minimization problem instead, that is,

The above minimization problem can be formulated as a minimum-cut problem by constructing a network where the source (orange node) is connected to all the pixels with capacity fi, and the sink (purple node) is connected by all the pixels with capacity bi. Two edges (i, j) and (j, i) with pij capacity are added between two adjacent pixels. The s-t cut-set then represents the pixels assigned to the foreground in P and pixels assigned to background in Q.

History

An account of the discovery of the theorem was given by Ford and Fulkerson in 1962:[5]

"Determining a maximal steady state flow from one point to another in a network subject to capacity limitations on arcs ... was posed to the authors in the spring of 1955 by T.E. Harris, who, in conjunction with General F. S. Ross (Ret.) had formulated a simplified model of railway traffic flow, and pinpointed this particular problem as the central one suggested by the model. It was not long after this until the main result, Theorem 5.1, which we call the max-flow min-cut theorem, was conjectured and established.[6] A number of proofs have since appeared."[7][8][9]

Proof

Let G = (V, E) be a network (directed graph) with s and t being the source and the sink of G respectively.

Consider the flow f computed for G by Ford–Fulkerson algorithm. In the residual graph (Gf) obtained for G (after the final flow assignment by Ford–Fulkerson algorithm), define two subsets of vertices as follows:

A: the set of vertices reachable from s in Gf

Ac: the set of remaining vertices i.e. V − A

Claim. value(f) = c(A, Ac), where the capacity of an s-t cut is defined by

.

Now, we know, for any subset of vertices, A. Therefore, for value(f) = c(A, Ac) we need:

All outgoing edges from the cut must be fully saturated.

All incoming edges to the cut must have zero flow.

To prove the above claim we consider two cases:

In G, there exists an outgoing edge such that it is notsaturated, i.e., f (x, y) < cxy. This implies, that there exists a forward edge from x to y in Gf, therefore there exists a path from s to y in Gf, which is a contradiction. Hence, any outgoing edge (x, y) is fully saturated.

In G, there exists an incoming edge such that it carries some non-zero flow, i.e., f (y, x) > 0. This implies, that there exists a backward edge from x to y in Gf, therefore there exists a path from s to y in Gf, which is again a contradiction. Hence, any incoming edge (y, x) must have zero flow.

Both of the above statements prove that the capacity of cut obtained in the above described manner is equal to the flow obtained in the network. Also, the flow was obtained by Ford-Fulkerson algorithm, so it is the max-flow of the network as well.

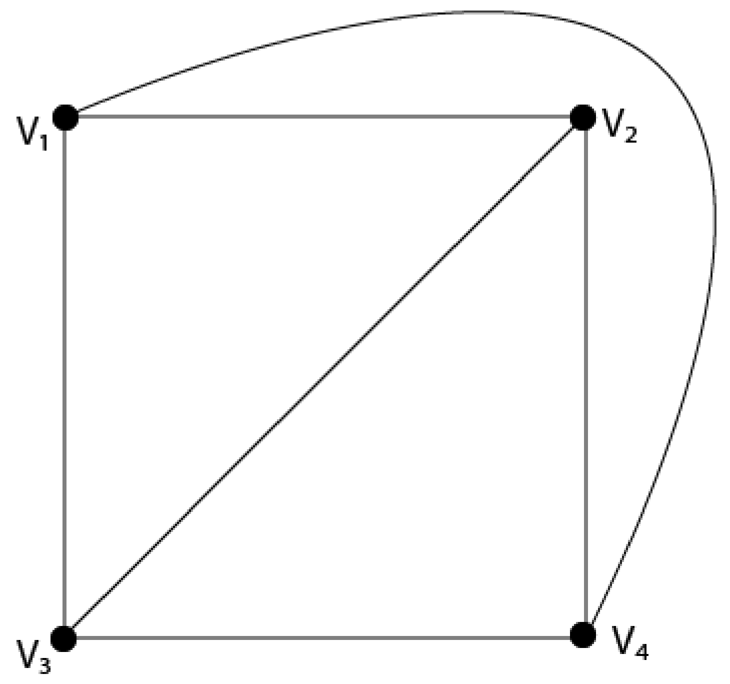
Also, since any flow in the network is always less than or equal to capacity of every cut possible in a network, the above described cut is also the min-cut which obtains the max-flow.

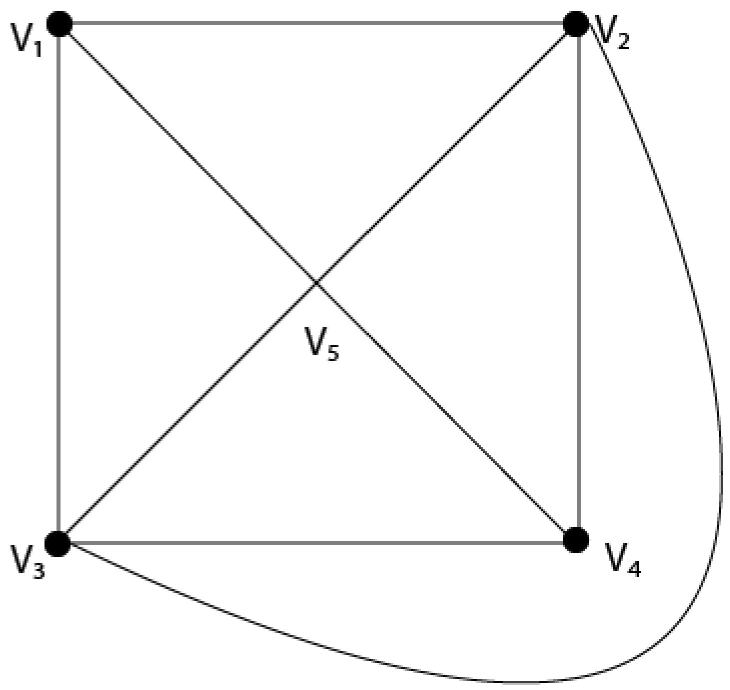
A corollary from this proof is that the maximum flow through any set of edges in a cut of a graph in equal to the minimum capacity of all previous cuts.

Planar Graph:

A graph is said to be planar if it can be drawn in a plane so that no edge cross.

Example: The graph shown in fig is planar graph.



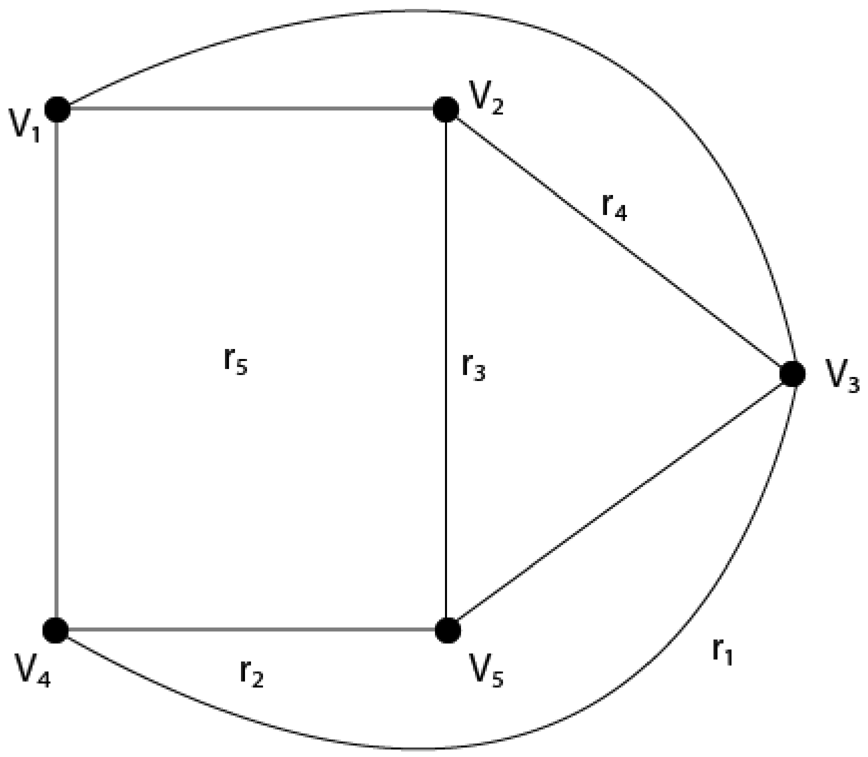


Region of a Graph: Consider a planar graph G=(V,E).A region is defined to be an area of the plane that is bounded by edges and cannot be further subdivided. A planar graph divides the plans into one or more regions. One of these regions will be infinite.

Finite Region: If the area of the region is finite, then that region is called a finite region.

Infinite Region: If the area of the region is infinite, that region is called a infinite region. A planar graph has only one infinite region.

Example: Consider the graph shown in Fig. Determine the number of regions, finite regions and an infinite region.



Solution: There are five regions in the above graph, i.e. r1,r2,r3,r4,r5.

There are four finite regions in the graph, i.e., r2,r3,r4,r5.

There is only one finite region, i.e., r1

Properties of Planar Graphs:

1. If a connected planar graph G has e edges and r regions, then $r \leq \frac{2}{3}$ e.
2. If a connected planar graph G has e edges, v vertices, and r regions, then v-e+r=2.
3. If a connected planar graph G has e edges and v vertices, then 3v-e≥6.
4. A complete graph Kn is a planar if and only if n<5.
5. A complete bipartite graph Kmn is planar if and only if m<3 or n>3.

Example: Prove that complete graph K4 is planar.

Solution: The complete graph K4 contains 4 vertices and 6 edges.

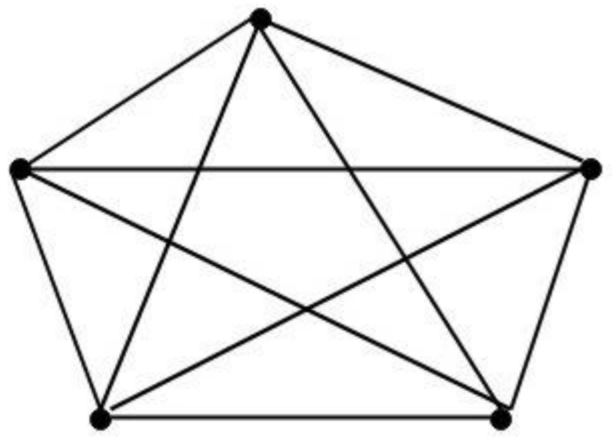
We know that for a connected planar graph 3v-e≥6.Hence for K4, we have 3x4-6=6 which satisfies the property (3).

Thus K4 is a planar graph. Hence Proved.

Non-Planar Graph:

A graph is said to be non planar if it cannot be drawn in a plane so that no edge cross.

Example: The graphs shown in fig are non planar graphs.



K_5

These graphs cannot be drawn in a plane so that no edges cross hence they are non-planar graphs.

Properties of Non-Planar Graphs:

A graph is non-planar if and only if it contains a subgraph homeomorphic to K5 or K3,3

Example1: Show that K5 is non-planar.

Solution: The complete graph K5 contains 5 vertices and 10 edges.

Now, for a connected planar graph $3v-e\geq6$.

Hence, for K5, we have $3 \times 5-10=5$ (which does not satisfy property 3 because it must be greater than or equal to 6).

Thus, K5 is a non-planar graph.

Example2: Show that the graphs shown in fig are non-planar by finding a subgraph homeomorphic to K5 or K3,3.

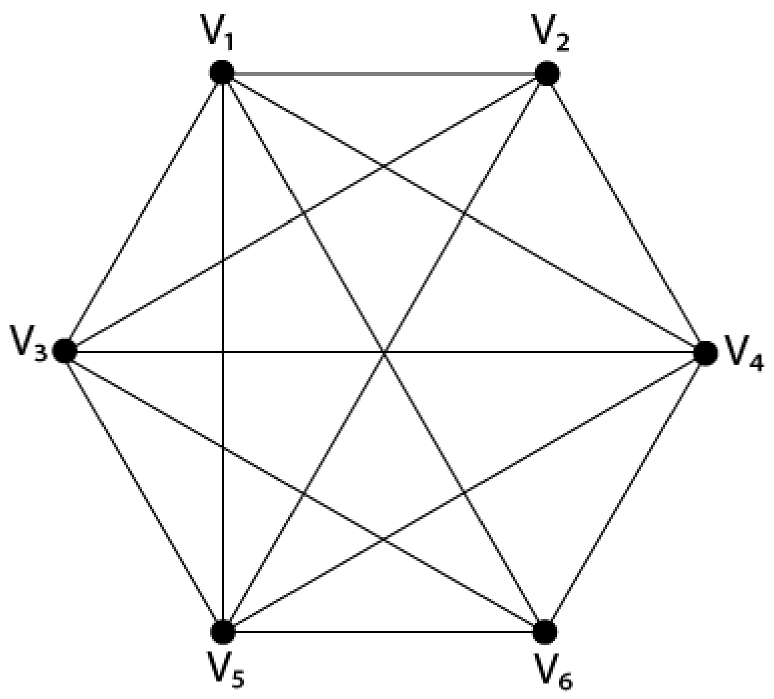


Fig: G_1

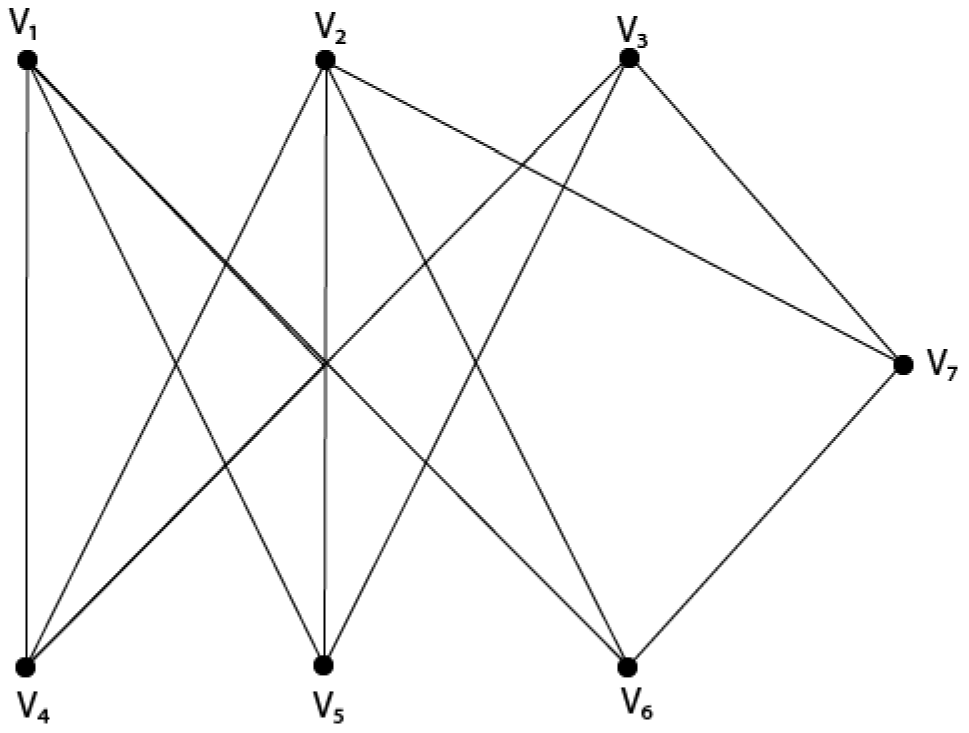


Fig: G_2

Solution: If we remove the edges (V1,V4),(V3,V4)and (V5,V4) the graph G1,becomes homeomorphic to K5.Hence it is non-planar.

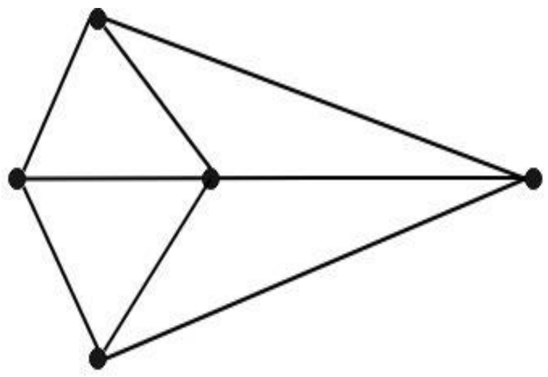
If we remove the edge V2,V7) the graph G2 becomes homeomorphic to K3,3.Hence it is a non-planar.

Graph Coloring:

Suppose that G= (V,E) is a graph with no multiple edges. A vertex coloring of G is an assignment of colors to the vertices of G such that adjacent vertices have different colors. A graph G is M-Colorable if there exists a coloring of G which uses M-Colors.

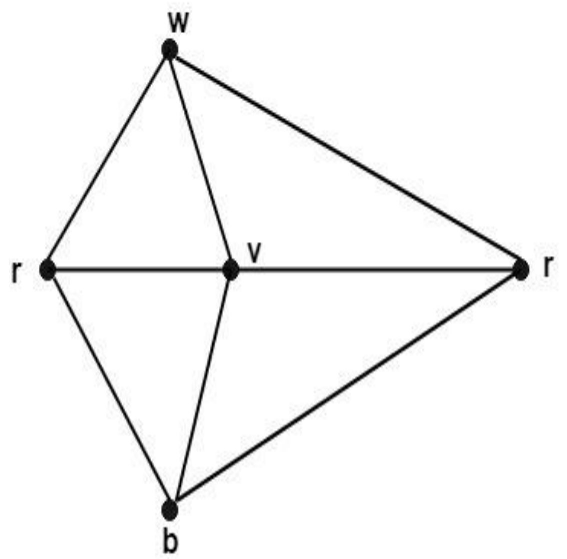
Proper Coloring: A coloring is proper if any two adjacent vertices u and v have different colors otherwise it is called improper coloring.

Example: Consider the following graph and color C={r, w, b, y}.Color the graph properly using all colors or fewer colors.



The graph shown in fig is a minimum 3-colorable, hence x(G)=3

Solution: Fig shows the graph properly colored with all the four colors.



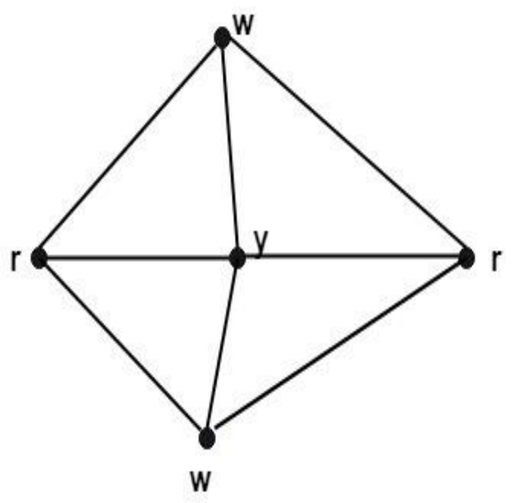
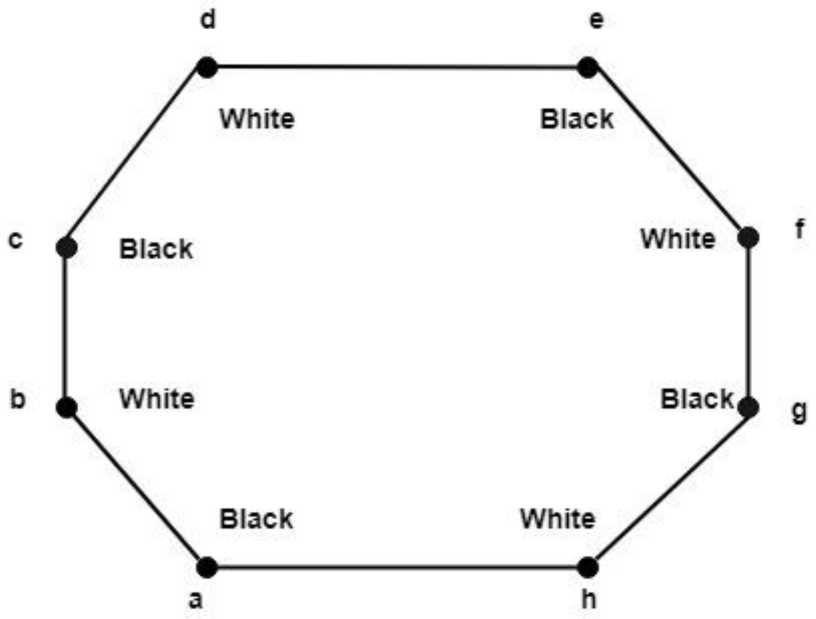


Fig shows the graph properly colored with three colors.



Chromatic number of G: The minimum number of colors needed to produce a proper coloring of a graph G is called the chromatic number of G and is denoted by x(G).

Example: The chromatic number of Kn is n.

Solution: A coloring of Kn can be constructed using n colours by assigning different colors to each vertex. No two vertices can be assigned the same colors, since every two vertices of this graph are adjacent. Hence the chromatic number of Kn=n.

Applications of Graph Coloring

Some applications of graph coloring include:

- Register Allocation
- Map Coloring
- Bipartite Graph Checking
- Mobile Radio Frequency Assignment
- Making a time table, etc.

State and prove Handshaking Theorem.

Handshaking Theorem: The sum of degrees of all the vertices in a graph G is equal to twice the number of edges in the graph.

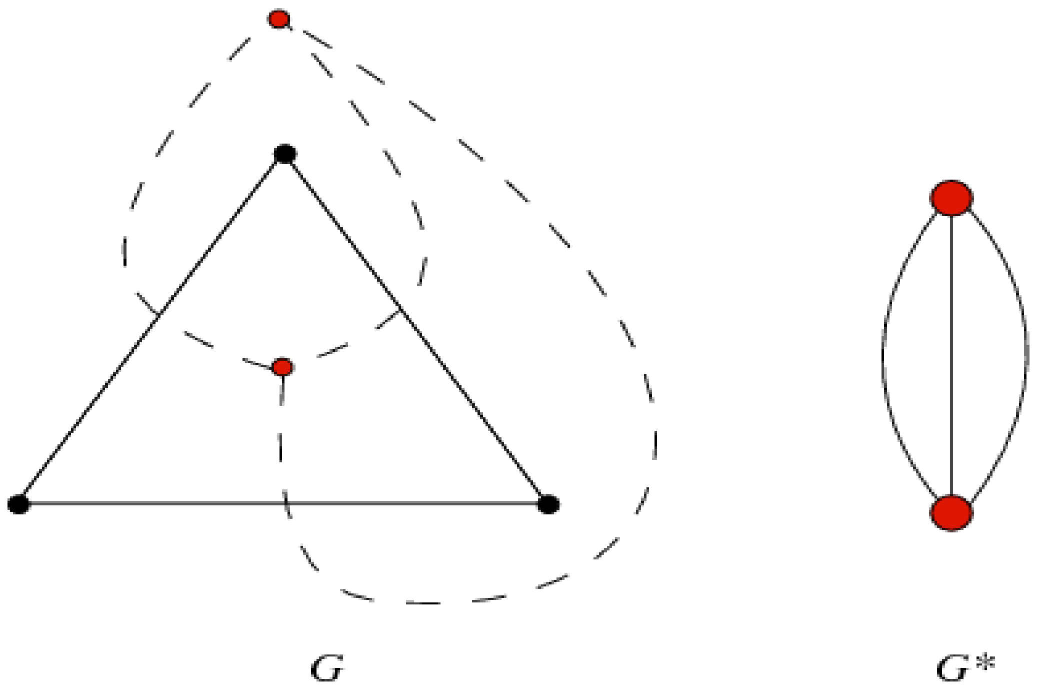
Mathematically it can be stated as:

$\sum v \in V deg(v)=2e$

Proof: Let G = (V, E) be a graph where V = {v1,v2, ...} be the set of vertices and E = {e1,e2 . . . .} be the set of edges. We know that every edge lies between two vertices so it provides degree one to each vertex. Hence each edge contributes degree two for the graph. So the sum of degrees of all vertices is equal to twice the number of edges in G.

Hence, $\sum_{v \in V} \deg(v) = 2e$

Geometric Dual Graph



Given a planar graph G, its geometric dual G^* is constructed by placing a vertex in each region of G (including the exterior region) and, if two regions have an edge x in common, joining the corresponding vertices by an edge X^* crossing only x. The result is always a planar pseudograph. However, an abstract graph with more than one embedding on the sphere can give rise to more than one dual.

Whitney showed that the geometric dual graph and combinatorial dual graph are equivalent (Harary 1994, p. 115), and so may simply be called "the" dual graph.

Kuratowski to graph detection of planarity

In graph theory, Kuratowski's theorem is a mathematical forbidden graph characterization of planar graphs, named after Kazimierz Kuratowski. It states that a finite graph is planar if and only if it does not contain a subgraph that is a subdivision of K5 (the complete graph on five vertices) or of K3,3 (complete bipartite graph on six vertices, three of which connect to each of the other three, also known as the utility graph).

Statement

A planar graph is a graph whose vertices can be represented by points in the Euclidean plane, and whose edges can be represented by simple curves in the same plane connecting the points representing their endpoints, such that no two curves intersect except at a common endpoint. Planar graphs are often drawn with straight line segments representing their edges, but by Fáry's theorem this makes no difference to their graph-theoretic characterization.

A subdivision of a graph is a graph formed by subdividing its edges into paths of one or more edges. Kuratowski's theorem states that a finite graph G is planar, if it is not possible to subdivide the edges of K5 or K3,3, and then possibly add additional edges and vertices, to form a graph isomorphic to G. Equivalently, a finite graph is planar if and only if it does not contain a subgraph that is homeomorphic to K5 or K3,3.

Kuratowski subgraphs

If G is a graph that contains a subgraph H that is a subdivision of K5 or K3,3, then H is known as a Kuratowski subgraph of G.[1] With this notation, Kuratowski's theorem can be expressed succinctly: a graph is planar if and only if it does not have a Kuratowski subgraph.

The two graphs K5 and K3,3 are nonplanar, as may be shown either by a case analysis or an argument involving Euler's formula. Additionally, subdividing a graph cannot turn a nonplanar graph into a planar graph: if a subdivision of a graph G has a planar drawing, the paths of the subdivision form curves that may be used to represent the edges of G itself. Therefore, a graph that contains a Kuratowski subgraph cannot be planar. The more difficult direction in proving Kuratowski's theorem is to show that, if a graph is nonplanar, it must contain a Kuratowski subgraph.

Algorithmic implications

A Kuratowski subgraph of a nonplanar graph can be found in linear time, as measured by the size of the input graph.[2] This allows the correctness of a planarity testing algorithm to be verified for nonplanar inputs, as it is straightforward to test whether a given subgraph is or is not a Kuratowski subgraph.[3] Usually, non-planar graphs contain a large number of Kuratowski-subgraphs. The extraction of these subgraphs is needed, e.g., in branch and cut algorithms for crossing minimization. It is possible to extract a large number of Kuratowski subgraphs in time dependent on their total size.[4]

History

Kazimierz Kuratowski published his theorem in 1930.[5] The theorem was independently proved by Orrin Frink and Paul Smith, also in 1930,[6] but their proof was never published. The special case of cubic planar graphs (for which the only minimal forbidden subgraph is K3,3) was also independently proved by Karl Menger in 1930.[7] Since then, several new proofs of the theorem have been discovered.[8]

In the Soviet Union, Kuratowski's theorem was known as either the Pontryagin–Kuratowski theorem or the Kuratowski–Pontryagin theorem,[9] as the theorem was reportedly proved independently by Lev Pontryagin around 1927.[10] However, as Pontryagin never published his proof, this usage has not spread to other places.[11]

Related results

A closely related result, Wagner's theorem, characterizes the planar graphs by their minors in terms of the same two forbidden graphs K5 and K3,3. Every Kuratowski subgraph is a special case of a minor of the same type, and while the reverse is not true, it is not difficult to find a Kuratowski subgraph (of one type or the other) from one of these two forbidden minors; therefore, these two theorems are equivalent.[12]

An extension is the Robertson-Seymour theorem.

Some more criterion of planarity

In graph theory, the planarity testing problem is the algorithmic problem of testing whether a given graph is a planar graph (that is, whether it can be drawn in the plane without edge intersections). This is a well-studied problem in computer science for which many practical algorithms have emerged, many taking advantage of novel data structures. Most of these methods operate in O(n) time (linear time), where n is the number of edges (or vertices) in the graph, which is asymptotically optimal. Rather than just being a single Boolean value, the output of a planarity testing algorithm may be a planar graph embedding, if the graph is planar, or an obstacle to planarity such as a Kuratowski subgraph if it is not.

Planarity criteria

Planarity testing algorithms typically take advantage of theorems in graph theory that characterize the set of planar graphs in terms that are independent of graph drawings. These include

- Kuratowski's theorem that a graph is planar if and only if it does not contain a subgraph that is a subdivision of K5 (the complete graph on five vertices) or K3,3 (the utility graph, a complete bipartite graph on six vertices, three of which connect to each of the other three).

- Wagner's theorem that a graph is planar if and only if it does not contain a minor (subgraph of a contraction) that is isomorphic to K5 or K3,3.

- The Fraysseix–Rosenstiehl planarity criterion, characterizing planar graphs in terms of a left-right ordering of the edges in a depth-first search tree.

The Fraysseix–Rosenstiehl planarity criterion can be used directly as part of algorithms for planarity testing, while Kuratowski's and Wagner's theorems have indirect applications: if an algorithm can find a copy of K5 or K3,3 within a given graph, it can be sure that the input graph is not planar and return without additional computation.

Other planarity criteria, that characterize planar graphs mathematically but are less central to planarity testing algorithms, include:

- Whitney's planarity criterion that a graph is planar if and only if its graphic matroid is also cographic,
- Mac Lane's planarity criterion characterizing planar graphs by the bases of their cycle spaces,
- Schnyder's theorem characterizing planar graphs by the order dimension of an associated partial order, and
- Colin de Verdière's planarity criterion using spectral graph theory.

Algorithms

Path addition method

The classic path addition method of Hopcroft and Tarjan[1] was the first published linear-time planarity testing algorithm in 1974. An implementation of Hopcroft and Tarjan's algorithm is provided in the Library of Efficient Data types and Algorithms by Mehlhorn, Mutzel and Näher [2] [3] .[4] In 2012, Taylor [5] extended this algorithm to generate all permutations of cyclic edge-order for planar embeddings of biconnected components.

Vertex addition method

Vertex addition methods work by maintaining a data structure representing the possible embeddings of an induced subgraph of the given graph, and adding vertices one at a time to this data structure. These methods began with an inefficient O(n2) method conceived by Lempel, Even and Cederbaum in 1967.[6] It was improved by Even and Tarjan, who found a linear-time solution for the s,t-numbering step,[7] and by Booth and Lueker, who developed the PQ tree data structure. With these improvements it is linear-time and outperforms the path addition method in practice.[8] This method was also extended to allow a planar embedding (drawing) to be efficiently computed for a planar graph.[9] In 1999, Shih and Hsu simplified these methods using the PC tree (an unrooted variant of the PQ tree) and a postorder traversal of the depth-first search tree of the vertices.[10]

Edge addition method

In 2004, John Boyer and Wendy Myrvold [11] developed a simplified O(n) algorithm, originally inspired by the PQ tree method, which gets rid of the PQ tree and uses edge additions to compute a planar embedding, if possible. Otherwise, a Kuratowski subdivision (of either K5 or K3,3) is computed. This is one of the two current state-of-the-art algorithms today (the other one is the planarity testing algorithm of de Fraysseix, de Mendez and Rosenstiehl[12][13]). See [14] for an experimental comparison with a preliminary version of the Boyer and Myrvold planarity test. Furthermore, the Boyer–Myrvold test was extended to extract multiple Kuratowski subdivisions of a non-planar input graph in a running time linearly dependent on the output size.[15] The source code for the planarity test[16][17] and the extraction of multiple Kuratowski subdivisions[16] is publicly available. Algorithms that locate a Kuratowski subgraph in linear time in vertices were developed by Williamson in the 1980s.[18]

Construction sequence method

A different method uses an inductive construction of 3-connected graphs to incrementally build planar embeddings of every 3-connected component of G (and hence a planar embedding of G itself).[19] The construction starts with K4 and is defined in such a way that every intermediate graph on the way to the full component is again 3-connected. Since such graphs have a unique embedding (up to flipping and the choice of the external face), the next bigger graph, if still planar, must be a refinement of the former graph. This allows to reduce the planarity test to just testing for each step whether the next added edge has both ends in the external face of the current embedding. While this is conceptually very simple (and gives linear running time), the method itself suffers from the complexity of finding the construction sequence.

Thickness and Crossings

In graph theory, the thickness of a graph G is the minimum number of planar graphs into which the edges of G can be partitioned. That is, if there exists a collection of k planar graphs, all having the same set of vertices, such that the union of these planar graphs is G, then the thickness of G is at most k.[1][2] In other words, the thickness of a graph is the minimum number of planar subgraphs whose union equals to graph G.[3]

Thus, a planar graph has thickness 1. Graphs of thickness 2 are called biplanar graphs. The concept of thickness originates in the 1962 conjecture of Frank Harary: For any graph on 9 points, either itself or its complementary graph is non-planar. The problem is equivalent to determining whether the complete graph K9 is biplanar (it is not, and the conjecture is true).[4] A

comprehensive[3] survey on the state of the arts of the topic as of 1998 was written by Petra Mutzel, Thomas Odenthal and Mark Scharbrodt.[5]

Specific graphs

The thickness of the complete graph on n vertices, Kn, is

except when n = 9, 10 for which the thickness is three.[6][7]

With some exceptions, the thickness of a complete bipartite graph Ka,b is generally:[8][9]

Related problem

Every forest is planar, and every planar graph can be partitioned into at most three forests. Therefore, the thickness of any graph G is at most equal to the arboricity of the same graph (the minimum number of forests into which it can be partitioned) and at least equal to the arboricity divided by three.[2][10] The thickness of G is also within constant factors of another standard graph invariant, the degeneracy, defined as the maximum, over subgraphs of G, of the minimum degree within the subgraph. If an n-vertex graph has thickness t then it necessarily has at most t(3n − 6) edges, from which it follows that its degeneracy is at most 6t − 1. In the other direction, if a graph has degeneracy D then it has arboricity, and thickness, at most D

Thickness is closely related to the problem of simultaneous embedding.[11] If two or more planar graphs all share the same vertex set, then it is possible to embed all these graphs in the plane, with the edges drawn as curves, so that each vertex has the same position in all the different drawings. However, it may not be possible to construct such a drawing while keeping the edges drawn as straight line segments.

A different graph invariant, the rectilinear thickness or geometric thickness of a graph G, counts the smallest number of planar graphs into which G can be decomposed subject to the restriction that all of these graphs can be drawn simultaneously with straight edges. The book thickness adds an additional restriction, that all of the vertices be drawn in convex position, forming a circular layout of the graph. However, in contrast to the situation for arboricity and degeneracy, no two of these three thickness parameters are always within a constant factor of each other.[12]

Computational complexity

It is NP-hard to compute the thickness of a given graph, and NP-complete to test whether the thickness is at most two.[13] However, the connection to arboricity allows the thickness to be approximated to within an approximation ratio of 3 in polynomial time.

Vector space of a graph and vectors

Definition

Let be a finite undirected graph. The vertex space of G is the vector space over the finite field of two elements of all functions . Every element of naturally corresponds the subset of V which assigns a 1 to its vertices. Also every subset of V is uniquely represented in by its characteristic function. The edge space is the -vector space freely generated by the edge set E. The dimension of the vertex space is thus the number of vertices of the graph, while the dimension of the edge space is the number of edges.

These definitions can be made more explicit. For example, we can describe the edge space.

Vector Basis

A vector basis of a vector space V is defined as a subset $v_1, \ldots, v_n$ of vectors in V that are linearly independent and span V. Consequently, if $(v_1, v_2, \ldots, v_n)$ is a list of vectors in V, then these vectors form a vector basis if and only if every $v \in V$ can be uniquely written as

$$v = a_1 v_1 + a_2 v_2 + \ldots + a_n v_n,$$

where a_1, ..., a_n are elements of the base field.

When the base field is the reals so that $a_i \in \mathbb{R}$ for $i = 1, \ldots, n$, the resulting basis vectors are n-tuples of reals that span n-dimensional Euclidean space $\mathbb{R}^n$. Other possible base fields include the complexes $\mathbb{C}$, as well as various fields of positive characteristic considered in algebra, number theory, and algebraic geometry.

A vector space V has many different vector bases, but there are always the same number of basis vectors in each of them. The number of basis vectors in V is called the dimension of V. Every spanning list in a vector space can be reduced to a basis of the vector space.

The simplest example of a vector basis is the standard basis in Euclidean space $\mathbb{R}^n$, in which the basis vectors lie along each coordinate axis. A change of basis can be used to transform vectors (and operators) in a given basis to another.

Given a hyperplane defined by

$$x_1 + x_2 + x_3 + x_4 + x_5 = 0,$$

a basis can be found by solving for x_1 in terms of x_2, x_3, x_4, and x_5. Carrying out this procedure,

$$x_1 = -x_2 - x_3 - x_4 - x_5,$$

so

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = x_2 \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + x_3 \begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + x_4 \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + x_5 \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix},$$

and the above vectors form an (unnormalized) basis.

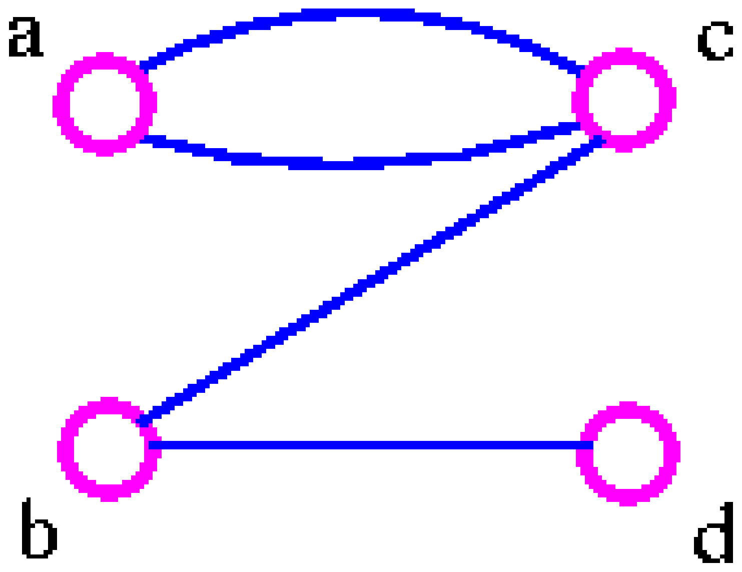
Given a matrix A with an orthonormal basis, the matrix corresponding to a change of basis, expressed in terms of the original $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_n$ is

$$\mathsf{A}' = [\, \mathsf{A}\hat{\mathbf{x}}_1 \quad \ldots \quad \mathsf{A}\hat{\mathbf{x}}_n \,].$$

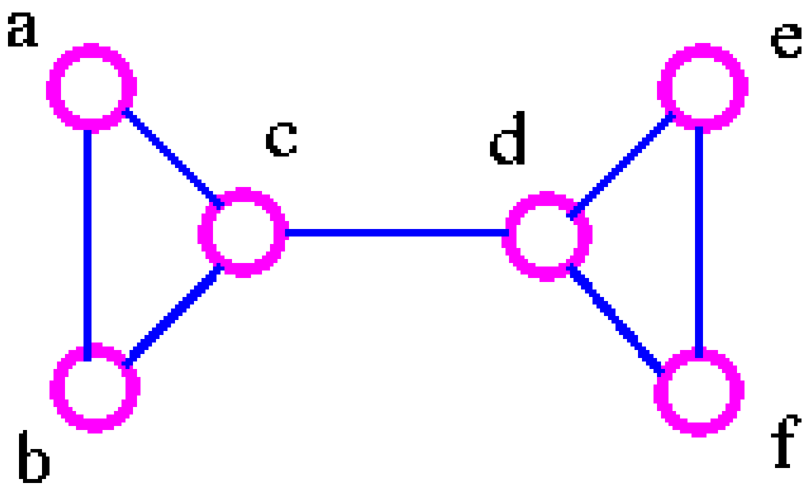
When a vector space is infinite dimensional, then a basis exists as long as one assumes the axiom of choice. A subset of the basis which is linearly independent and whose span is dense is called a complete set, and is similar to a basis. When V is a Hilbert space, a complete set is called a Hilbert basis.

cut set vector

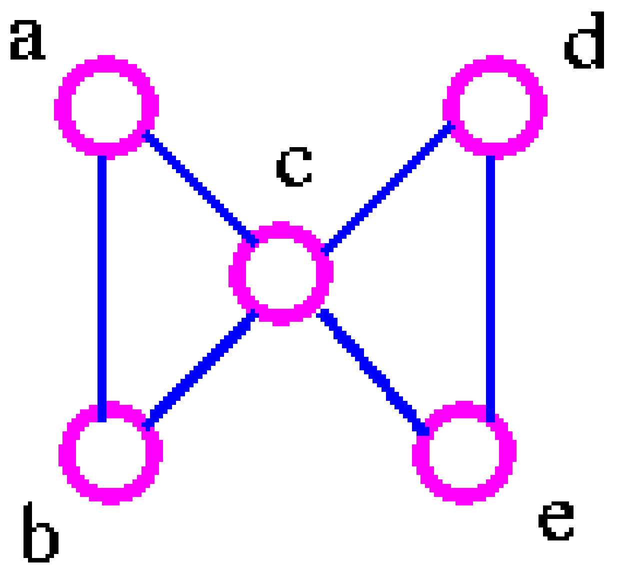
Cut Edge (Bridge)   A bridge is a single edge whose removal disconnects a graph.



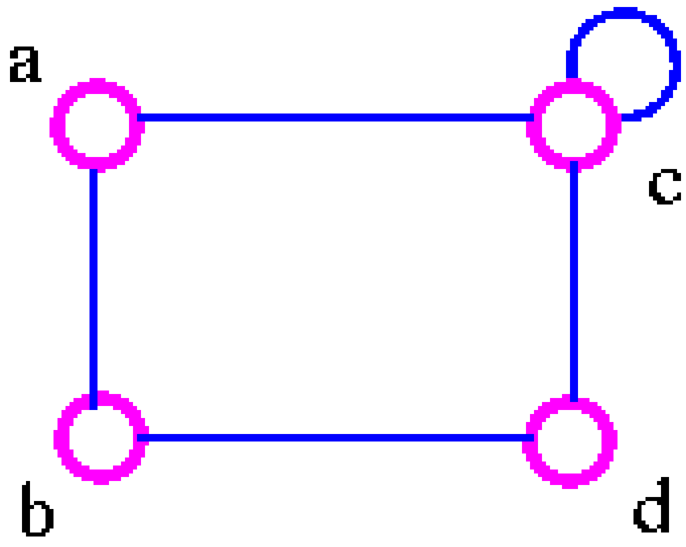
The above graph G1 can be split up into two components by removing one of the edges bc or bd. Therefore, edge bc or bd is a bridge.



The above graph G2 can be disconnected by removing a single edge, cd. Therefore, edge cd is a bridge.



The above graph G3 cannot be disconnected by removing a single edge, but the removal of two edges (such as ac and bc) disconnects it.



The above graph G4 can be disconnected by removing two edges such as ac and dc.

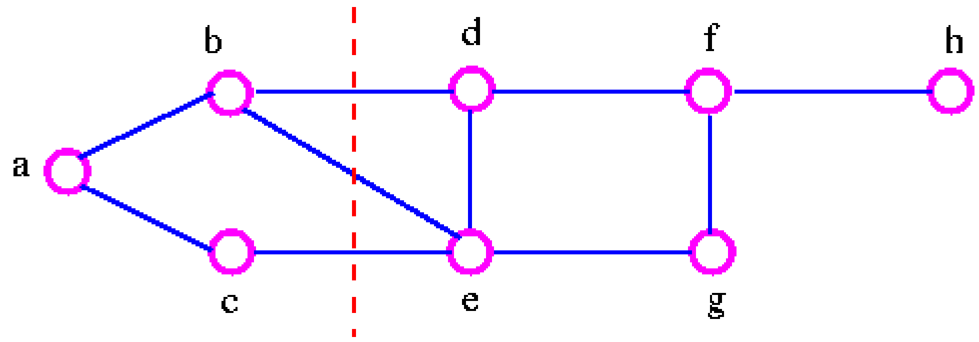
Cut Set

A cut set of a connected graph G is a set S of edges with the following properties

- The removal of all edges in S disconnects G.

- The removal of some (but not all) of edges in S does not disconnects G.

As an example consider the following graph



We can disconnect G by removing the three edges bd, bc, and ce, but we cannot disconnect it by removing just two of these edges. Note that a cut set is a set of edges in which no edge is redundant

Cut-Vertex

A cut-vertex is a single vertex whose removal disconnects a graph.

It is important to note that the above definition breaks down if G is a complete graph, since we cannot then disconnects G by removing vertices. Therefore, we make the following definition.

Connectivity of Complete Graph

The connectivity k(kn) of the complete graph kn is n-1. When n-1 ≥ k, the graph kn is said to be k-connected.

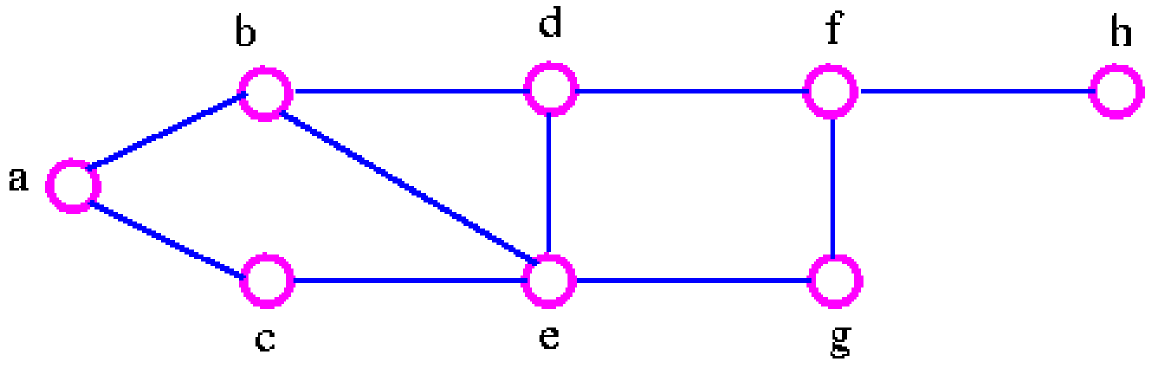
Vertex-Cut set

A vertex-cut set of a connected graph G is a set S of vertices with the following properties.

a. the removal of all the vertices in S disconnects G.

b. the removal of some (but not all) of vertices in S does not disconnects G.

Consider the following graph



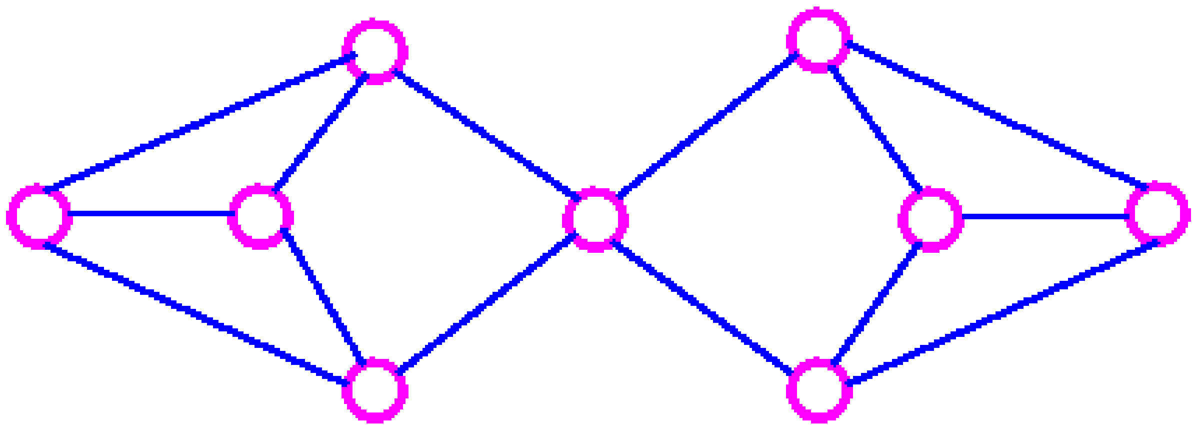
We can disconnects the graph by removing the two vertices b and e, but we cannot disconnect it by removing just one of these vertices. the vertex-cutset of G is {b, e}.

Note that the connectivity k(G) does not exceed the edge-connectivity λ(G). This inequality holds for all connected graph.

Formally, for any connected graph G we have

$K(G) \leq \lambda(G) \leq \delta(G)$

where δ(G) is the smallest vertex-degree in G. But it is certainly possible for both inequality in above theorem to be strict inequalities (that is, $k(G) < \lambda(G) < \delta(G)$) For example, in the following graph,



$K(G)=1$, $\lambda(G) = 2$, and $\delta(G) = 3$.

circuit and cut set verses sub spaces,

Orthogonal vectors

Orthogonal is just another word for perpendicular. Two vectors are orthogonal if the angle between them is 90 degrees. If two vectors are orthogonal, they form a right triangle whose hypotenuse is the sum of the vectors. Thus, we can use the Pythagorean theorem to prove that the dot product xTy = yTx is zero exactly when x and y are orthogonal. (The length squared ||x|| 2 equals xTx.) Note that all vectors are orthogonal to the zero vector.

Orthogonal subspaces

Subspace S is orthogonal to subspace T means: every vector in S is orthogonal to every vector in T. The blackboard is not orthogonal to the floor; two vectors in the line where the blackboard meets the floor aren't orthogonal to each other. In the plane, the space containing only the zero vector and any line through the origin are orthogonal subspaces. A line through the origin and the whole plane are never orthogonal subspaces. Two lines through the origin are orthogonal subspaces if they meet at right angles.

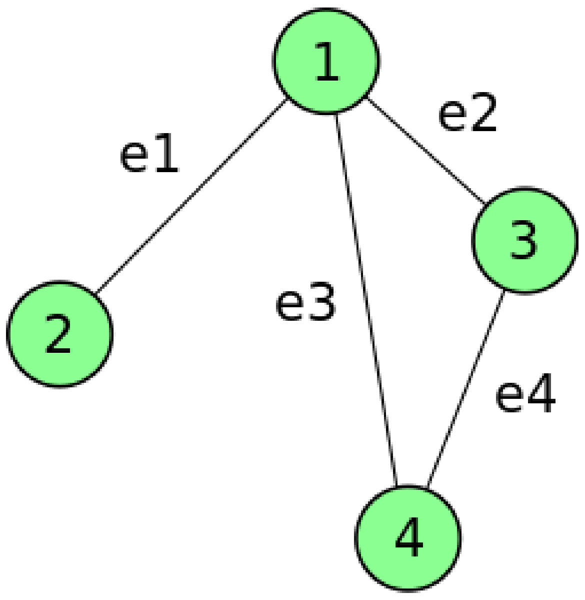
Nullspace is perpendicular to row space

The row space of a matrix is orthogonal to the nullspace, because Ax = 0 means the dot product of x with each row of A is 0. But then the product of x with any combination of rows of A must be 0. The column space is orthogonal to the left nullspace of A because the row space of AT is perpendicular to the nullspace of AT. In some sense, the row space and the nullspace of a matrix subdivide Rn 1 2 5 into two perpendicular subspaces.

Incidence matrix of graphs

In mathematics, an incidence matrix is a matrix that shows the relationship between two classes of objects. If the first class is X and the second is Y, the matrix has one row for each element of X and one column for each element of Y. The entry in row x and column y is 1 if x and y are related (called incident in this context) and 0 if they are not. There are variations; see below.

Graph theory



An undirected graph.

In graph theory an undirected graph has two kinds of incidence matrices: unoriented and oriented.

The unoriented incidence matrix (or simply incidence matrix) of an undirected graph is a n × m matrix B, where n and m are the numbers
of vertices and edges respectively, such that Bi,j = 1 if the vertex vi and edge ej are incident and 0 otherwise.

For example, the incidence matrix of the undirected graph shown on the right is a matrix consisting of 4 rows (corresponding to the four vertices, 1–4) and 4 columns (corresponding to the four edges, e1–e4):

	e1	e2	e3	e4
1	1	1	1	0
2	1	0	0	0
3	0	1	0	1
4	0	0	1	1

=

If we look at the incidence matrix, we see that the sum of each column is equal to 2. This is because each edge has a vertex connected to each end.

The incidence matrix of a directed graph is a n × m matrix B where n and m are the number of vertices and edges respectively, such that Bi,j = −1 if the edge ej leaves vertex vi, 1 if it enters vertex vi and 0 otherwise (many authors use the opposite sign convention).

The oriented incidence matrix of an undirected graph is the incidence matrix, in the sense of directed graphs, of any orientation of the graph. That is, in the column of edge e, there is one 1 in the row corresponding to one vertex of e and one −1 in the row corresponding to the other vertex of e, and all other rows have 0. The oriented incidence matrix is unique up to negation of any of the columns, since negating the entries of a column corresponds to reversing the orientation of an edge.

The unoriented incidence matrix of a graph G is related to the adjacency matrix of its line graph L(G) by the following theorem:

where A(L(G)) is the adjacency matrix of the line graph of G, B(G) is the incidence matrix, and Im is the identity matrix of dimension m.

The discrete Laplacian (or Kirchhoff matrix) is obtained from the oriented incidence matrix B(G) by the formula

The integral cycle space of a graph is equal to the null space of its oriented incidence matrix, viewed as a matrix over the integers or real or complex numbers.

The binary cycle space is the null space of its oriented or unoriented incidence matrix, viewed as a matrix over the two-element field.

Signed and bidirected graphs

The incidence matrix of a signed graph is a generalization of the oriented incidence matrix. It is the incidence matrix of any bidirected graph that orients the given signed graph. The column of a positive edge has a 1 in the row corresponding to one endpoint and a −1 in the row corresponding to the other endpoint, just like an edge in an ordinary (unsigned) graph. The column of a negative edge has either a 1 or a −1 in both rows. The line graph and Kirchhoff matrix properties generalize to signed graphs.

Multigraphs

The definitions of incidence matrix apply to graphs with loops and multiple edges. The column of an oriented incidence matrix that corresponds to a loop is all zero, unless the graph is signed and the loop is negative; then the column is all zero except for ±2 in the row of its incident vertex.

Hypergraphs

Because the edges of ordinary graphs can only have two vertices (one at each end), the column of an incidence matrix for graphs can only have two non-zero entries. By contrast, a hypergraph can have multiple vertices assigned to one edge; thus, a general matrix of non-negative integers describes a hypergraph.

Incidence structures

The incidence matrix of an incidence structure C is a p × q matrix B (or its transpose), where p and q are the number of points and lines respectively, such that Bi,j = 1 if the point pi and line Lj are incident and 0 otherwise. In this case, the incidence matrix is also a biadjacency matrix of the Levi graph of the structure. As there is a hypergraph for every Levi graph, and vice versa, the incidence matrix of an incidence structure describes a hypergraph.

Finite geometries

An important example is a finite geometry. For instance, in a finite plane, X is the set of points and Y is the set of lines. In a finite geometry of higher dimension, X could be the set of points and Y could be the set of subspaces of dimension one less than the dimension of the entire space (hyperplanes); or, more generally, X could be the set of all subspaces of one dimension d and Y the set of all subspaces of another dimension e, with incidence defined as containment.

Polytopes

In a similar manner, the relationship between cells whose dimensions differ by one in a polytope, can be represented by an incidence matrix.[1]

Block designs

Another example is a block design. Here X is a finite set of "points" and Y is a class of subsets of X, called "blocks", subject to rules that depend on the type of design. The incidence matrix is an important tool in the theory of block designs. For instance, it can be used to prove Fisher's inequality, a fundamental theorem of balanced incomplete 2-designs (BIBDs), that the number of blocks is at least the number of points.[2] Considering the blocks as a system of sets, the permanent of the incidence matrix is the number of systems of distinct representatives (SDRs).

sub matrices of A(G)

Given the matrices A and B (as shown), find AB using the partitionings indicated:

$$\mathbf{A}=\left[\begin{array}{cc|c} 1 & -1 & 2 \\ \hline 3 & 0 & 4 \\ 0 & 1 & 2 \end{array}\right], \quad \mathbf{B}=\left[\begin{array}{ccc|c} 5 & 2 & 0 & 2 \\ 1 & -1 & 3 & 1 \\ \hline 0 & 1 & 1 & 4 \end{array}\right].$$

4.

Partition the given matrices A and B and, using the results, find AB.
A=[4100220000100012],B=[3200−11000021001−1].

5.Compute A2 for the matrix A given in Problem 4 by partitioning A into block diagonal form.

6.Compute B2 for the matrix B given in Problem 4 by partitioning B into block diagonal form.

7.Use partitioning to compute A2 and A3 for

A=[1000000200000001000000100000001000000].

What is An for any positive integral power of n > 3?

8.Use partitioning to compute A2 and A3 for

A=[0−100000−1000000002−2−40000−13400001−2−30000000−10000000−1].

What is An for any positive integral power of n?

TRANSFORMATIONS

In Applied Dimensional Analysis and Modeling (Second Edition), 2007
Theorem 9-7.

The interchange of any two columns of A causes the interchange of the same two columns in C.

Proof.

Suppose we have two Dimensional Sets. The first has submatrices A1, B1, C1, and the second A2, B2, C2. Then, by assumption, B1 = B2, and
(9-25)A2=A1·T

where Z is the appropriate permutation matrix causing the interchange of the two designated columns in A1 Now by (9-25) and the Fundamental Formula
C2=−(A2−1·B2)T=−((A1·T)−1·B2)T=−(T−1·A1−1·B2)T

from which, since B1 = B2,
(9-26)C2=−(A1−1·B2)T·(T−1)T=C1(T−1)T

But Z is orthogonal, thus its transpose is equal to its inverse. Hence Z−1 = ZT, so that (Z−1)T = Z. Substituting this result into (9-26), we obtain
(9-27)C2=C1·T

Comparing (9-25) with (9-27), we see that A1 and C1 undergo the same transformation, and therefore if Z causes the interchange of any particular two columns in A1, it also causes the interchange of the same two columns in C1. This proves the theorem.

The rank–nullity theorem is a theorem in linear algebra, which asserts that the dimension of the domain of a linear map is the sum of its rank (the dimension of its image) and its nullity (the dimension of its kernel) .

Stating the theorem

Let , be vector spaces, where is finite dimensional. Let be a linear transformation. Then

,

where and One can refine this theorem via the splitting lemma to be a statement about an isomorphism of spaces, not just dimensions. Explicitly, since T induces an isomorphism from to , the existence of a basis for V that extends any given basis of implies, via the splitting lemma, that . Taking dimensions, the Rank-Nullity theorem follows immediately.

Matrices

Since matrices immediately come to mind when discussing linear maps. In the case of an matrix, the dimension of the domain is , the number of columns in the matrix. Thus the Rank-Nullity theorem for a given matrix immediately becomes

Proofs

Here we provide two proofs. The first[3] operates in the general case, using linear maps. The second proof[4] looks at the homogeneous system for with rank and shows explicitly that there exists a set of linearly independent solutions that span the kernel of . While the theorem requires that the domain of the linear map be finite-dimensional, there is no such assumption on the codomain. This means that there are linear maps not given by matrices for which the theorem applies. Despite this, the first proof is not actually more general than the second: since the image of the linear map is finite-dimensional, we can represent the map from its domain to its image by a matrix, prove the theorem for that matrix, then compose with the inclusion of the image into the full codomain.

Unit V

Coloring and covering partitioning of graph

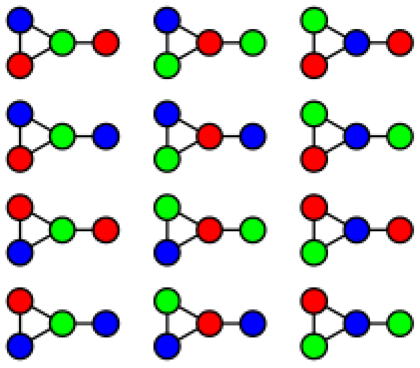
In graph theory, graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called "colors" to elements of a graph subject to certain constraints. In its simplest form, it is a way of coloring the vertices of a graph such that no two adjacent vertices are of the same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

Vertex coloring is usually used to introduce graph coloring problems, since other coloring problems can be transformed into a vertex coloring instance. For example, an edge coloring of a graph is just a vertex coloring of its line graph, and a face coloring of a plane graph is just a vertex coloring of its dual. However, non-vertex coloring problems are often stated and studied as-is. This is partly pedagogical, and partly because some problems are best studied in their non-vertex form, as in the case of edge coloring.

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

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

Definition and terminology



This graph can be 3-colored in 12 different ways.

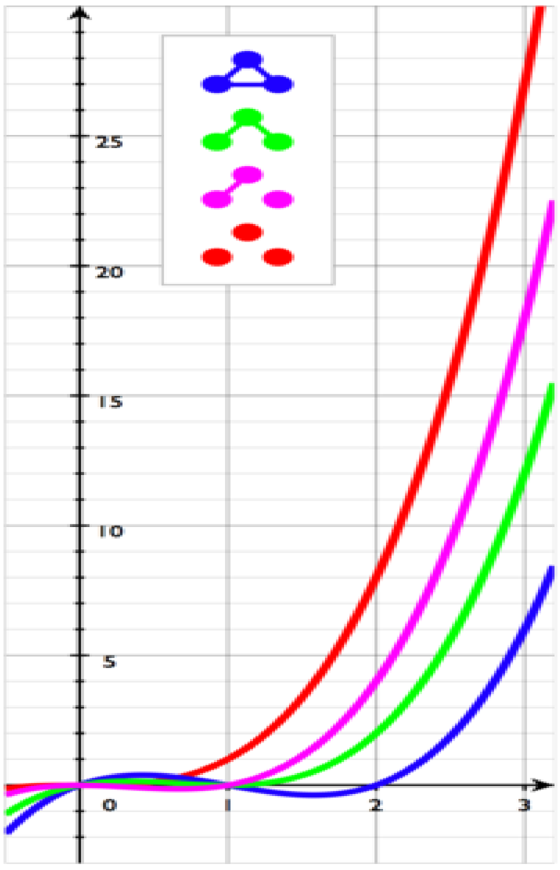
Vertex coloring

When used without any qualification, a coloring of a graph is almost always a proper vertex coloring, namely a labeling of the graph’s vertices with colors such that no two vertices sharing the same edge have the same color. Since a vertex with a loop (i.e. a connection directly back to itself) could never be properly colored, it is understood that graphs in this context are loopless.

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

A coloring using at most k colors is called a (proper) k-coloring. The smallest number of colors needed to color a graph G is called its chromatic number, and is often denoted χ(G). Sometimes γ(G) is used, since χ(G) is also used to denote the Euler characteristic of a graph. A graph that can be assigned a (proper) k-coloring is k-colorable, and it is k-chromatic if its chromatic number is exactly k. A subset of vertices assigned to the same color is called a color class, every such class forms an independent set. Thus, a k-coloring is the same as a partition of the vertex set into k independent sets, and the terms k-partite and k-colorable have the same meaning.

Chromatic number



All non-isomorphic graphs on 3 vertices and their chromatic polynomials. The empty graph E3 (red) admits a 1-coloring; the others admit no such colorings. The green graph admits 12 colorings with 3 colors.

Chromatic polynomials

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

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

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

The chromatic polynomial includes at least as much information about the colorability of G as does the chromatic number. Indeed, χ is the smallest positive integer that is not a root of the chromatic polynomial

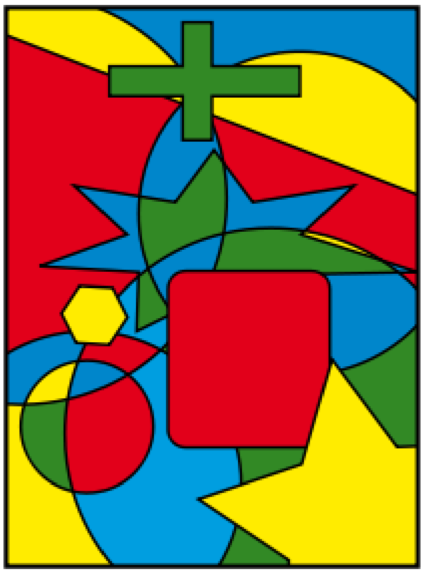
Chromatic polynomials for certain graphs

Triangle K3	
Complete graph Kn	
Tree with n vertices	

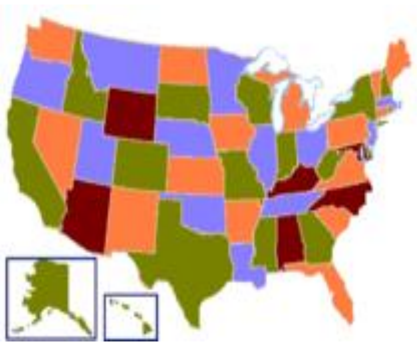
Cycle Cn	
Petersen graph	

.

Four color theorem



Example of a four-colored map



A four-coloring of a map of the states of the United States (ignoring lakes).

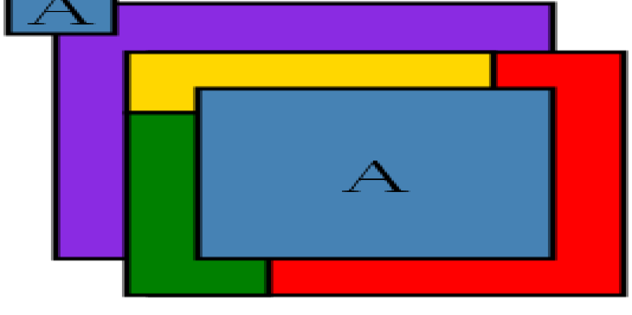
In mathematics, the four color theorem, or the four color map theorem, states that, given any separation of a plane into contiguous regions, producing a figure called a map, no more than four colors are required to color the regions of the map so that no two adjacent regions have the same color. Adjacent means that two regions share a common boundary curve segment, not merely a corner where three or more regions meet.[1] It was the first major theorem to be proved using a computer. Initially, this proof was not accepted by all mathematicians because the computer-assisted proof was infeasible for a human to check by hand.[2] Since then the proof has gained wide acceptance, although some doubters remain.[3]

The four color theorem was proved in 1976 by Kenneth Appel and Wolfgang Haken after many false proofs and counterexamples (unlike the five color theorem, proved in the 1800s, which states that five colors are enough to color a map). To dispel any remaining doubts about the Appel–Haken proof, a simpler proof using the same ideas and still relying on computers was published in 1997 by Robertson, Sanders, Seymour, and Thomas. Additionally, in 2005, the theorem was proved by Georges Gonthier with general-purpose theorem-proving software.

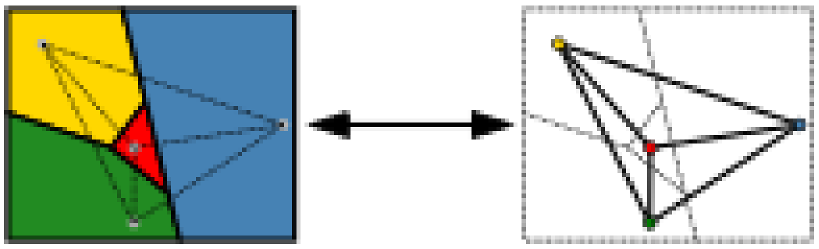
Precise formulation of the theorem

In graph-theoretic terms, the theorem states that for loopless planar graph , the chromatic number of its dual graph is The intuitive statement of the four color theorem – "given any separation of a plane into contiguous regions, the regions can be colored using at most four colors so that no two adjacent regions have the same color" – needs to be interpreted appropriately to be correct.

First, regions are adjacent if they share a boundary segment; two regions that share only isolated boundary points are not considered adjacent. Second, bizarre regions, such as those with finite area but infinitely long perimeter, are not allowed; maps with such regions can require more than four colors.[4] (To be safe, we can restrict to regions whose boundaries consist of finitely many straight line segments. It is allowed that a region entirely surround one or more other regions.) Note that the notion of "contiguous region" (technically: connected open subset of the plane) is not the same as that of a "country" on regular maps, since countries need not be contiguous (e.g., the Cabinda Province as part of Angola, Nakhchivan as part of Azerbaijan, Kaliningrad as part of Russia, and Alaska as part of the United States are not contiguous). If we required the entire territory of a country to receive the same color, then four colors are not always sufficient. For instance, consider a simplified map:



In this map, the two regions labeled A belong to the same country. If we wanted those regions to receive the same color, then five colors would be required, since the two A regions together are adjacent to four other regions, each of which is adjacent to all the others. A similar construction also applies if a single color is used for all bodies of water, as is usual on real maps. For maps in which more than one country may have multiple disconnected regions, six or more colors might be required.



A map with four regions, and the corresponding planar graph with four vertices.

A simpler statement of the theorem uses graph theory. The set of regions of a map can be represented more abstractly as an undirected graph that has a vertex for each region and an edge for every pair of regions that share a boundary segment. This graph is planar: it can be drawn in the plane without crossings by placing each vertex at an arbitrarily chosen location within the region to which it corresponds, and by drawing the edges as curves without crossings that lead from one region's vertex, across a shared boundary segment, to an adjacent region's vertex. Conversely any planar graph can be formed from a map in this way. In graph-theoretic terminology, the four-color theorem states that the vertices of every planar graph can be colored with at most four colors so that no two adjacent vertices receive the same color, or for short.

Directed graph

In mathematics, and more specifically in graph theory, a directed graph (or digraph) is a graph that is made up of a set of vertices connected by edges, where the edges have a direction associated with them.

Definition

In formal terms, a directed graph is an ordered pair G = (V, A) where[1]

- V is a set whose elements are called vertices, nodes, or points;
- A is a set of ordered pairs of vertices, called arrows, directed edges (sometimes simply edges with the corresponding set named E instead of A), directed arcs, or directed lines.

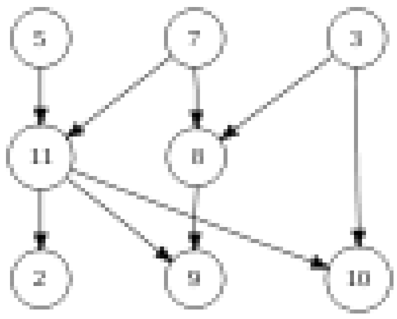
It differs from an ordinary or undirected graph, in that the latter is defined in terms of unordered pairs of vertices, which are usually called edges, arcs, or lines.

The aforementioned definition does not allow a directed graph to have multiple arrows with the same source and target nodes, but some authors consider a broader definition that allows directed graphs to have such multiple arrows (namely, they allow the arrows set to be a multiset). More specifically, these entities are addressed as directed multigraphs (or multidigraphs).

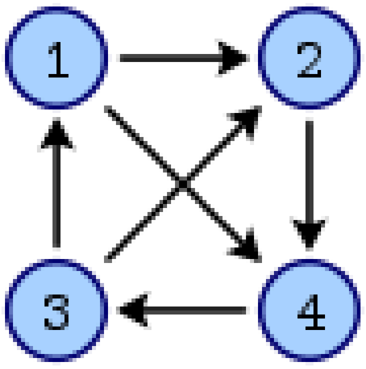
On the other hand, the aforementioned definition allows a directed graph to have loops (that is, arrows that directly connect nodes with themselves), but some authors consider a narrower definition that doesn't allow directed graphs to have loops.[2] More specifically, directed graphs without loops are addressed as simple directed graphs, while directed graphs with loops are addressed as loop-digraphs (see section Types of directed graphs).

Types of directed graphs

Subclasses



A simple directed acyclic graph



A tournament on 4 vertices

- Symmetric directed graphs are 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).

- Simple directed graphs are directed graphs that have no loops (arrows that directly connect vertices to themselves) and no multiple arrows with same source and target nodes. As already introduced, in case of multiple arrows the entity is usually addressed as directed multigraph. Some authors describe digraphs with loops as loop-digraphs.[2]

    - Complete directed graphs are simple directed graphs where each pair of vertices is joined by a symmetric pair of directed arrows (it is equivalent to an undirected complete graph with the edges replaced by pairs of inverse arrows). It follows that a complete digraph is symmetric.

  - Oriented graphs are directed graphs having no bidirected edges (i.e. at most one of (x, y) and (y, x) may be arrows of the graph). It follows that a directed graph is an oriented graph if and only if it hasn't any 2-cycle.[3]

    - Tournaments are oriented graphs obtained by choosing a direction for each edge in undirected complete graphs.

    - Directed acyclic graphs (DAGs) are directed graphs with no directed cycles.

      - Multitrees are DAGs in which no two distinct directed paths from a single starting vertex meet back at the same ending vertex.

      - Oriented trees or polytrees are DAGs formed by orienting the edges of undirected acyclic graphs.

        - Rooted trees are oriented trees in which all edges of the underlying undirected tree are directed either away from or towards the root.

Digraphs with supplementary properties

This list is incomplete; you can help by adding missing items with reliable sources.

- Weighted directed graphs (also known as directed networks) are (simple) directed graphs with weights assigned to their arrows, similarly to weighted graphs (which are also known as undirected networks or weighted networks).[2]

  - Flow networks are weighted directed graphs where two nodes are distinguished, a source and a sink.

- Rooted directed graphs (also known as flow graphs) are digraphs in which a vertex has been distinguished as the root.

  - Control flow graphs are rooted digraphs used in computer science as a representation of the paths that might be traversed through a program during its execution.

- Signal-flow graphs are directed graphs in which nodes represent system variables and branches (edges, arcs, or arrows) represent functional connections between pairs of nodes.

- Flow graphs are digraphs associated with a set of linear algebraic or differential equations.

- State diagrams are directed multigraphs that represent finite state machines.

- Commutative diagrams are digraphs used in category theory, where the vertices represent (mathematical) objects and the arrows represent morphisms, with the property that all directed paths with the same start and endpoints lead to the same result by composition.

- In the theory of Lie groups, a quiver Q is a directed graph serving as the domain of, and thus characterizing the shape of, a representation V defined as a functor, specifically an object of the functor category FinVctKF(Q) where F(Q) is the free category on Q consisting of paths in Q and FinVctK is the category of finite-dimensional vector spaces over a field K. Representations of a quiver label its vertices with vector spaces and its edges (and hence paths) compatibly with linear transformations between them, and transform via natural transformations.

Directed paths and connectedness

When dealing with directed graphs, we define two kinds of connectedness, strong and weak. Strong connectedness of a directed graph is defined as follows:

Definition (Strong Connectedness of a Directed Graph)   A directed graph is strongly connected   if there is a path in G between every pair of vertices in .

For example, Figure shows the directed graph given by

$$\begin{aligned}\mathcal{V} &= \{a, b, c, d, e, f\}\\ \mathcal{E} &= \{(a, b), (b, c), (b, e), (c, a), (d, e), (e, f), (f, d),\}\end{aligned}$$

Notice that the graph is not connected! E.g., there is no path from any of the vertices in to any of the vertices in . Nevertheless, the graph ``looks'' connected in the sense that it is not made of up of separate parts in the way that the graph in Figure is.

This idea of ``looking'' connected is what weak connectedness represents. To define weak connectedness we need to introduce first the notion of the undirected graph that underlies a directed graph: Consider a directed graph . The underlying undirected graph is the graph where $\hat{\mathcal{E}}$ represents the set of undirected edges that is obtained by removing the arrowheads from the directed edges in G:

$$\hat{\mathcal{E}} = \left\{ \{v, w\} : (v, w) \in \mathcal{E} \vee (w, v) \in \mathcal{E} \right\}.$$

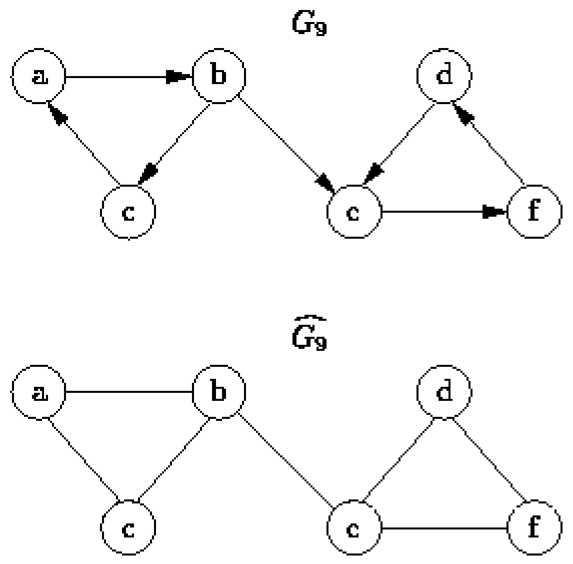


Figure: An Weakly Connected Directed Graph and the Underlying Undirected Graph

Weak connectedness of a directed graph is defined with respect to its underlying, undirected graph:

Definition (Weak Connectedness of a Directed Graph) A directed graph is weakly connected   if the underlying undirected graph $\widehat{G}$ is connected.

For example, since the undirected graph $\widehat{G_9}$ in Figure ❑ is connected, the directed graph G_9 is weakly connected. Consider what happens when we remove the edge (b,e) from the directed graph G_9. The underlying undirected graph that we get is G_8 in Figure ❑. Therefore, when we remove edge (b,e) from G_9, the graph that remains is neither strongly connected nor weakly connected.

A traversal of a directed graph (either depth-first or breadth-first) starting from a given vertex will only visit all the vertices of an undirected graph if there is a path from the start vertex to every other vertex. Therefore, a simple way to test whether a directed graph is strongly connected uses $|\mathcal{V}|$ traversals--one starting from each vertex in $\mathcal{V}$. Each time the number of vertices visited is counted. The graph is strongly connected if all the vertices are visited in each traversal.

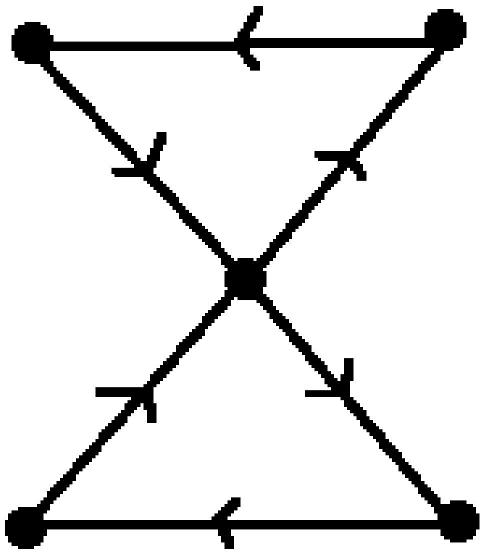
Program ❑ shows how this can be implemented. It shows the IsConnected member function of the Digraph class which returns the Boolean value true if the graph is strongly connected.

```
bool Digraph::IsConnected () const
{
    for (Vertex::Number v = 0; v < numberOfVertices; ++v)
    {
        CountingVisitor visitor;
        DepthFirstTraversal (
            PreOrder (visitor), SelectVertex (v));
        if (visitor.Count () != numberOfVertices)
            return false;
    }
    return true;
}
```

Eulerian Digraphs

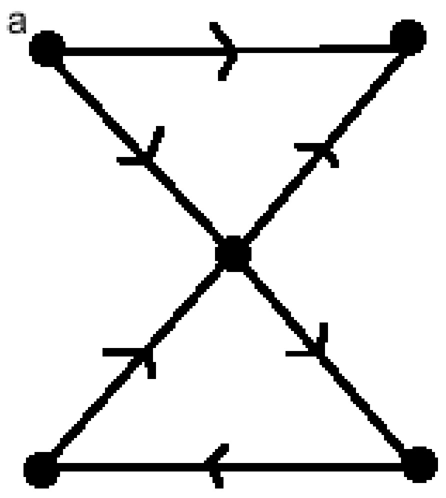
Definition: A digraph D is Eulerian if it contains a closed trail (a walk containing no repeated arcs) th
arcs in D. This closed trail is known as an Eulerian Trail.

Essentially, the same conditions for regular graphs being Eulerian hold for digraphs being Eulerian, however, we are now regarding Eulerian trails with regards to arcs. For example, the following digraph is Eulerian:



Eulerian

We should also not that the underlying graph being Eulerian does not imply that a digraph with that underlying graph is Eulerian. For example, the following orientation of the arcs of the graph above no longer make D Eulerian:



NOT Eulerian

This is because we must start our trail at vertex a and end at vertex a. This is of course, impossible, since the in-degree of vertex a is 0.

Trees with directed edges

In graph theory, a tree is an undirected graph in which any two vertices are connected by exactly one path, or equivalently a connected acyclic undirected graph.[1] A forest is an undirected graph in which any two vertices are connected by at most one path, or equivalently an acyclic undirected graph, or equivalently a disjoint union of trees.[2]

A polytree[3] (or directed tree[4] or oriented tree[5][6] or singly connected network[7]) is a directed acyclic graph (DAG) whose underlying undirected graph is a tree. A polyforest (or directed forest or oriented forest) is a directed acyclic graph whose underlying undirected graph is a forest.

The various kinds of data structures referred to as trees in computer science have underlying graphs that are trees in graph theory, although such data structures are generally rooted trees. A rooted tree may be directed, called a directed rooted tree,[8][9] either making all its edges point away from the root—in which case it is called an arborescence[4][10] or out-tree[11][12]—or making all its edges point towards the root—in which case it is called an anti-arborescence[13] or in-tree.[11][14] A rooted tree itself has been defined by some authors as a directed graph.[15][16][17] A rooted forest is a disjoint union of rooted trees. A rooted forest may be directed, called a directed rooted forest, either making all its edges point away from the root in each rooted tree—in which case it is called a branching or out-forest—or making all its edges point towards the root in each rooted tree—in which case it is called an anti-branching or in-forest.

The term "tree" was coined in 1857 by the British mathematician Arthur Cayley.[18]

Tree

A tree is an undirected graph G that satisfies any of the following equivalent conditions:

- G is connected and acyclic (contains no cycles).
- G is acyclic, and a simple cycle is formed if any edge is added to G.
- G is connected, but would become disconnected if any single edge is removed from G.
- G is connected and the 3-vertex complete graph K3 is not a minor of G.
- Any two vertices in G can be connected by a unique simple path.

If G has finitely many vertices, say n of them, then the above statements are also equivalent to any of the following conditions:

- G is connected and has n − 1 edges.
- G is connected, and every subgraph of G includes at least one vertex with zero or one incident edges. (That is, G is connected and 1-degenerate.)
- G has no simple cycles and has n − 1 edges.

As elsewhere in graph theory, the order-zero graph (graph with no vertices) is generally not considered to be a tree: while it is vacuously connected as a graph (any two vertices can be connected by a path), it is not 0-connected (or even (−1)-connected) in algebraic topology, unlike non-empty trees, and violates the "one more vertex than edges" relation. It may, however, be considered as a forest consisting of zero trees.

An internal vertex (or inner vertex or branch vertex) is a vertex of degree at least 2. Similarly, an external vertex (or outer vertex, terminal vertex or leaf) is a vertex of degree 1.

An irreducible tree (or series-reduced tree) is a tree in which there is no vertex of degree 2 (enumerated at sequence A000014 in the OEIS).

Fores

A forest is an undirected graph in which any two vertices are connected by at most one path. Equivalently, a forest is an undirected acyclic graph. Equivalently, a forest is an undirected graph, all of whose connected components are trees; in other words, the graph consists of a disjoint union of trees. As special cases, the order-zero graph (a forest consisting of zero trees), a single tree, and an edgeless graph, are examples of forests. Since for every tree V − E = 1, we can easily count the number of trees that are within a forest by subtracting the difference between total vertices and total edges. TV − TE = number of trees in a forest.

Polytree

A polytree[3] (or directed tree[4] or oriented tree[5][6] or singly connected network[7]) is a directed acyclic graph (DAG) whose underlying undirected graph is a tree. In other words, if we replace its directed edges with undirected edges, we obtain an undirected graph that is both connected and acyclic.

Some authors restrict the phrase "directed tree" to the case where the edges are all directed towards a particular vertex, or all directed away from a particular vertex (see arborescence).

Polyforest

A polyforest (or directed forest or oriented forest) is a directed acyclic graph whose underlying undirected graph is a forest. In other words, if we replace its directed edges with undirected edges, we obtain an undirected graph that is acyclic.

Some authors restrict the phrase "directed forest" to the case where the edges of each connected component are all directed towards a particular vertex, or all directed away from a particular vertex (see branching).

Rooted tree

A rooted tree is a tree in which one vertex has been designated the root.[20] The edges of a rooted tree can be assigned a natural orientation, either away from or towards the root, in which case the structure becomes a directed rooted tree. When a directed rooted tree has an orientation away from the root, it is called an arborescence[4] or out-tree;[11] when it has an orientation towards the root, it is called an anti-arborescence or in-tree.[11] The tree-order is the partial ordering on the vertices of a tree with $u < v$ if and only if the unique path from the root to v passes through u. A rooted tree which is a subgraph of some graph G is a normal tree if the ends of every edge in G are comparable in this tree-order whenever those ends are vertices of the tree (Diestel 2005, p. 15). Rooted trees, often with additional structure such as ordering of the neighbors at each vertex, are a key data structure in computer science; see tree data structure.

In a context where trees are supposed to have a root, a tree without any designated root is called a free tree.

A labeled tree is a tree in which each vertex is given a unique label. The vertices of a labeled tree on n vertices are typically given the labels 1, 2, ..., n. A recursive tree is a labeled rooted tree where the vertex labels respect the tree order (i.e., if $u < v$ for two vertices u and v, then the label of u is smaller than the label of v).

In a rooted tree, the parent of a vertex v is the vertex connected to v on the path to the root; every vertex has a unique parent except the root which has no parent.[20] A child of a vertex v is a vertex of which v is the parent.[20] An ascendant of a vertex v is any vertex which is either the parent of v or is (recursively) the ascendant of the parent of v. A descendant of a vertex v is any vertex which is either the child of v or is (recursively) the descendant of any of the children of v. A sibling to a vertex v is any other vertex on the tree which has the same parent as v.[20] A leaf is a vertex with no children.[20] An internal vertex is a vertex that is not a leaf.[20]

The height of a vertex in a rooted tree is the length of the longest downward path to a leaf from that vertex. The height of the tree is the height of the root. The depth of a vertex is the length of the path to its root (root path). This is commonly needed in the manipulation of the various self-balancing trees, AVL trees in particular. The root has depth zero, leaves have height zero, and a tree with only a single vertex (hence both a root and leaf) has depth and height zero. Conventionally, an empty tree (a tree with no vertices, if such are allowed) has depth and height −1.

A k-ary tree is a rooted tree in which each vertex has at most k children.[21] 2-ary trees are often called binary trees, while 3-ary trees are sometimes called ternary trees.

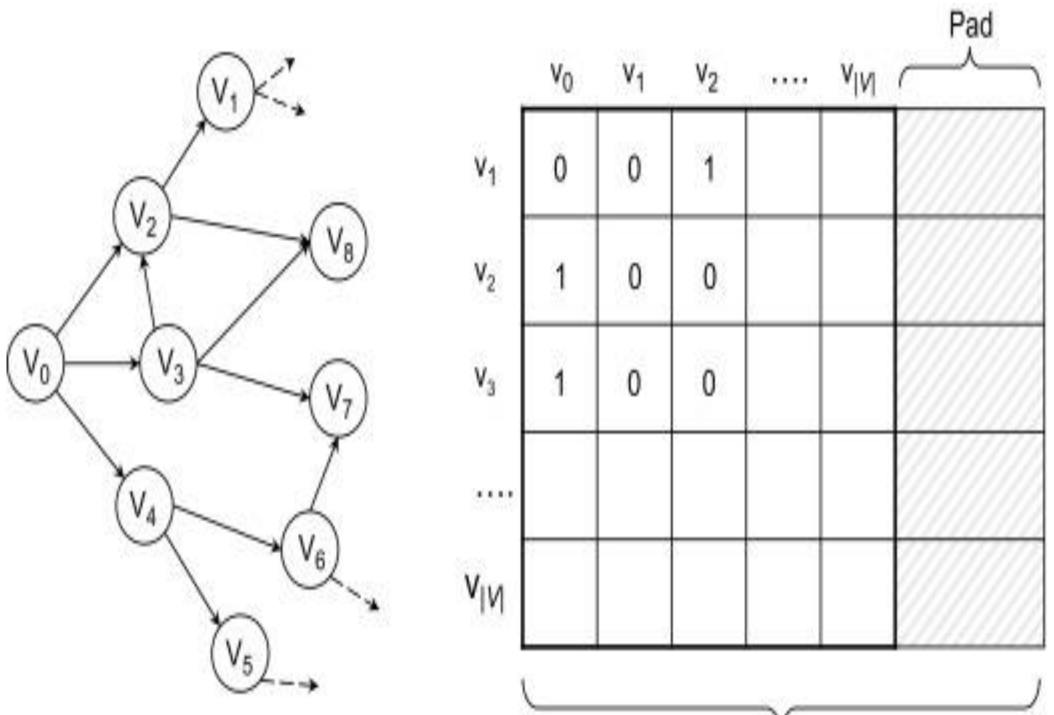
Ordered tree

An ordered tree (or plane tree) is a rooted tree in which an ordering is specified for the children of each vertex.[20][22] This is called a "plane tree" because an ordering of the children is equivalent to an embedding of the tree in the plane, with the root at the top and the children of each vertex lower than that vertex. Given an embedding of a rooted tree in the plane, if one fixes a direction of children, say left to right, then an embedding gives an ordering of the children. Conversely, given an ordered tree, and conventionally drawing the root at the top, then the child vertices in an ordered tree can be drawn left-to-right, yielding an essentially unique planar embedding.

Matrices A, B, C of digraph adjacency matrix of digraph

Graph algorithms on GPUs
F. Busato, N. Bombieri, in Advances in GPU Research and Practice, 2017

1.1 Adjacency Matrices
An adjacency matrix allows representing a graph with a V × V matrix M = [f(i, j)] where each element f(i, j) contains the attributes of the edge (i, j). If the edges do not have an attribute, the graph can be represented by a boolean matrix to save memory space (Fig. 1).



Common algorithms that use this representation are all-pair shortest path (APSP) and transitive closure [3–9]. If the graph is weighted, each value of f(i, j) is defined as follows:
Mi,j0ifi=jwi,jifi≠jandi,j∈E∞ifi≠jandi,j∉E

On GPUs, both directed and undirected graphs represented by an adjacency matrix take O(|V |2) memory space, because the whole matrix is stored in memory with a large continuous array. In GPU architectures, it is also important, for performance, to align the matrix with memory to improve coalescence of memory accesses. In this context, the Compute Unified Device Architecture (CUDA) language provides the function cudaMallocPitch [10] to pad the data allocation, with the aim of meeting the alignment requirements for memory coalescing. In this case the indexing changes are as follows:
M[i·V+j]→M[i·pitch+j]

The O(|V |2) memory space required is the main limitation of the adjacency matrices. Even on recent GPUs, they allow handling of fairly small graphs. For example, on a GPU device with 4 GB of DRAM, graphs that can be represented through an adjacency matrix can have a maximum of only 32,768 vertices (which, for actual graph datasets, is considered restrictive). In general, adjacency matrices best represent small and dense graphs (i.e., |E|≈|V |2). In some cases, such as for the all-pairs shortest path problem, graphs larger than the GPU memory are partitioned and each part is processed independently [7–9].

Graph theory

Mary Attenborough, in Mathematics for Electrical Engineering and Computing, 2003
19.3 Matrix representation of a graph
The incidence matrix and adjacency matrix
The incidence matrix of a graph G is a |V| ×|E| matrix. The element aij= the number of times that vertex viis incident with the edge ej.
The adjacency matrix of G is the |V| × |V| matrix. aij= the number of edges joining viand vjThe incidence matrix for the graph in Figure 19.2 is given by
e1e2e3e4e5e6e7e8e9v1v2v3v4v5v6(10 0 0 0 0 1 0 0 1100110010110000200011 00000000110001000001100)

and the adjacency matrix by
v1v2v3v4v5v6v1v2v3v4v5v6(0 1 0 0 0 1 101021011100001010020100110000

Graphs in SQL
Joe Celko, in Joe Celko's SQL for Smarties (Fifth Edition), 2015
27.4 Adjacency Matrix Model
An adjacency matrix is a square array whose rows are out-node and columns are in-nodes of a graph. A one in a cell means that there is edge between the two nodes. Using the graph in figure 30.1, we would have an array like this:

	A	B	C	D	E	F	G	H
A	1	1	1	0	0	0	0	0

	A	B	C	D	E	F	G	H
B	0	1	0	1	0	0	0	0
C	0	0	1	1	0	0	1	0
D	0	0	0	1	1	1	0	0
E	0	0	0	0	1	0	0	1
F	0	0	0	0	0	1	0	0
G	0	0	0	0	0	0	1	1
H	0	0	0	0	0	0	0	1

Many graph algorithms are based on the adjacency matrix model and can be translated into SQL. Go back to the chapter on modeling matrices in SQL and in particular matrix multiplication in SQL. For example, Dijkstra's algorithm for the shortest distances between each pair of nodes in a graph looks like this in pseudo-code.

```
FOR k = 1 TO n
DO FOR i = 1 TO n
DO FOR j = 1 TO n
IF a[i,k] + a[k,j] < a[i,j]
THEN a[i,j] = a[i,k] + a[k,j]
END IF;
END FOR;
END FOR;
END FOR;
```

You need to be warned that for a graph of (n) nodes, the table will be of size (n^2). The algorithms often run in (n^3) time. The advantage it has is that once you have completed a table it can be used for look ups rather than recomputing distances over and over

Graphs in SQL

Joe Celko, in Joe Celko's Trees and Hierarchies in SQL for Smarties (Second Edition), 2012

12.1.4 Adjacency Matrix Model

An adjacency matrix is a square array whose rows are out-node and columns are in-nodes of a graph. A one in a cell means that there is edge between the two nodes. Using the following graph, we would have an array like this:

	A	B	C	D	E	F	G	H
A	1	1	1	0	0	0	0	0
B	0	1	0	1	0	0	0	0
C	0	0	1	1	0	0	1	0
D	0	0	0	1	1	1	0	0
E	0	0	0	0	1	0	0	1
F	0	0	0	0	0	1	0	0
G	0	0	0	0	0	0	1	1
H	0	0	0	0	0	0	0	1

Many graph algorithms are based on the adjacency matrix model and can be translated into SQL. Go to the appropriate chapter for the details of modeling matrices in SQL and, in particular, look at the section on matrix multiplication in SQL. For example, Dijkstra's algorithm for shortest distances between each pair of nodes in a graph looks like this in this array pseudo-code.

```
FOR k = 1 TO n
DO FOR i = 1 TO n
DO FOR j = 1 TO n
IF a[i,k] + a[k,j] < a[i,j]
THEN a[i,j] = a[i,k] + a[k,j]
END IF;
END FOR;
END FOR;
END FOR;
```

You need to be warned that for a graph of (n) nodes, the table will be of size (n^2). The algorithms often run in (n^3) time. The advantage it has is that once you have completed a table, it can be used for lookups rather than recomputing distances over and over.

Running the query against the data set …

```
INSERT INTO AdjacencyListGraph
VALUES ('a', 'd', 1),
('d', 'e', 1),
('e', 'c', 1),
('c', 'b', 1),
('b', 'd', 1),
('a', 'e', 5);
```

Gives the result SET …

source_node	dest_node	min_wgt
a	b	4
a	c	3
a	d	1
a	e	2
b	c	3
b	d	1
b	e	2
c	b	1
c	d	2
c	e	3
d	b	3
d	c	2
d	e	1
e	b	2
e	c	1
e	d	3

Doing the Dijkstra algorithm would probably execute significantly faster in a language with arrays than in SQL

A novel graph clustering algorithm based on discrete-time quantum random walk
S.G. Roy, A. Chakrabarti, in Quantum Inspired Computational Intelligence, 2017
12 Proposed Graph-Based Quantum Clustering Algorithm

We identify clusters within a given general graph G(V, E). If there is any cluster structure present in the graph, the observed QCL simulation result identifies the number of clusters within that graph. The proposed quantum algorithm to identify clusters is given below:

Input: Graph adjacency matrix
Output: Number of clusters within the graph and nodes within each clusters.
Quantum clustering algorithm: Design discrete-time quantum random walk circuit for the graph (as explained below):

- Start at the origin node: x = 0.
- Choose the coin operator (assumed here to be the here Hadamard coin operator):
    - H|x,0〉 →|x,0〉 +|x,1〉 2,
    - H|x,1〉 →|x,0〉 −|x,1〉 2

  .
- The shift operator S helps to move the quantum walker from one quantum state to another state of the superposition states according to the permutation circuit.
- Apply ĈU=S(I⊗Ĉ) on the initial quantum state, where I is the identity operator, S is the shift operator, and C is the coin operator.
  (If U transformation for initial states is repeatedly used, then the resulting superposition state |ψ〉 will contain more nodes of the graph. The random walker traversed the graph faster.)
- Repeat step 4 for r consecutive transformations of U until a steady state (convergence) is reached (i.e., all nodes with all the coin states are visited by the quantum walker).
- Interpret the quantum simulation result for initial clustering. The nodes with the same visiting probability distribution values are now in the same cluster.
- Step 6 may generate many clusters (the number of nodes is less than the threshold value). Apply the merge and split model to fulfill the threshold criteria and to identify the actual cluster sets.
- End.

12.0.2 Merge and split model

- Let pi be the probability distribution values of node i = 0, 1, 2, n. Depending on the QCL simulation result, group the nodes into different sets of cluster Clusterj, where j = 1, 2, …, n, according to their visiting probability distribution values.
- The k-dimensional tree method is discussed in [34, 35]. We use the one-dimensional tree method (as our nodes are one-dimensional) in our merge and split model to identify the clusters.

- Assume we have set P, which consists of visiting probability distributions values for each cluster Clusterj, where j = 1, 2, …, n.

- Set P is divided into two subregions in accordance with their average probability distribution values:$1N\sum i=1NPi$ (N denotes the number of clusters obtained in step 6 of the proposed quantum clustering algorithm and Pi is the visiting probability distribution values for each cluster). One region has values greater than the average probability distribution value (pavg). The other region has values less than the average probability distribution value.

- Repeat until threshold criteria are reached.

Observation of the proposed algorithm shows that the nodes with the same or nearly the same visiting probability values tend to lie within the same

Computer Aided Molecular Design: Theory and Practice
P.M. Harper, ... R. Gani, in Computer Aided Chemical Engineering, 2003
Generation Algorithm for Level-3
The level-3 generation algorithm transforms the group based connectivity information (the adjacency matrix from level-2) into atom-based information. This is achieved by expanding each group into its corresponding atom-based adjacency matrix and replacing the groups in the group based description with additional rows and columns to allow for group expansion. When performing the group expansion into an atomic representation it is possible to experience that one group based description yields more than one atomic description. This is the case with compounds containing any of the groups listed in Table 5. It can be noted that the additional representations appear in the cases where the original groups have a ring element with 1 or more free connections because of the ambiguously defined distance (in the ring) between the free bonds (as in ortho/meta/para) or between hetero-atoms and bonds in aromatic rings (as in Pyridine derivates)

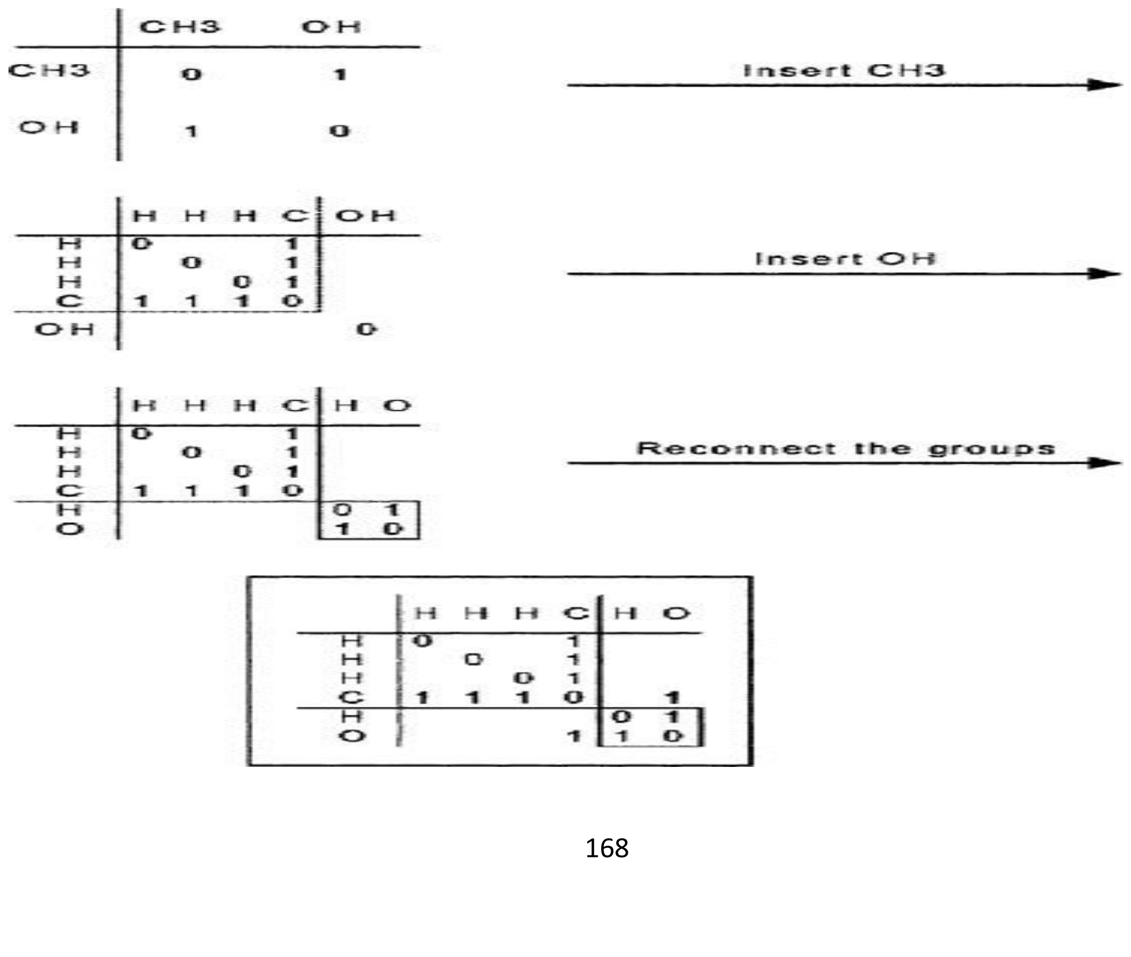
First-order group	Number of isomers on an atomic basis
C_5H_4N	3
C_5H_3N	6
C_4H_3S	2
C_4H_2S	4

The algorithm for generation of atomic adjacency matrices from group-based ones consists of the following steps:

- Set the matrix A equal to the group based matrix from Level 2

- List the groups in the compound
- For each of the groups in the compound:
    - Load the corresponding atom based matrix or matrices (for groups with ambiguous 2 dimensional representation)
    - Insert the atom based matrix in the place of the corresponding group in A. If the particular group has multiple representations create a corresponding number of copies of A
- Identify the atoms taking part in the original bonds between groups
- Reconnect the molecule by establishing connections between the atoms identified in point 3
- Stop

After performing the conversion the net result is a series of compounds described using atoms and how they are interconnected. Furthermore all 2D structural variations on the atomic level have been generated. This conversion process is illustrated in Figure 16.



$$\begin{array}{c|cc} & CH3 & OH \\ \hline CH3 & 0 & 1 \\ OH & 1 & 0 \end{array}$$

$$\begin{array}{c|cccc|c} & H & H & H & C & OH \\ \hline H & 0 & & & 1 & \\ H & & 0 & & 1 & \\ H & & & 0 & 1 & \\ C & 1 & 1 & 1 & 0 & \\ \hline OH & & & & & 0 \end{array}$$

$$\begin{array}{c|cccc|cc} & H & H & H & C & H & O \\ \hline H & 0 & & & 1 & & \\ H & & 0 & & 1 & & \\ H & & & 0 & 1 & & \\ C & 1 & 1 & 1 & 0 & & \\ \hline H & & & & & 0 & 1 \\ O & & & & & 1 & 0 \end{array}$$

$$\boxed{\begin{array}{c|cccc|cc} & H & H & H & C & H & O \\ \hline H & 0 & & & 1 & & \\ H & & 0 & & 1 & & \\ H & & & 0 & 1 & & \\ C & 1 & 1 & 1 & 0 & & 1 \\ \hline H & & & & & 0 & 1 \\ O & & & & 1 & 1 & 0 \end{array}}$$

Elements of Graph Theory

Jean-Michel Réveillac, in Optimization Tools for Logistics, 2015

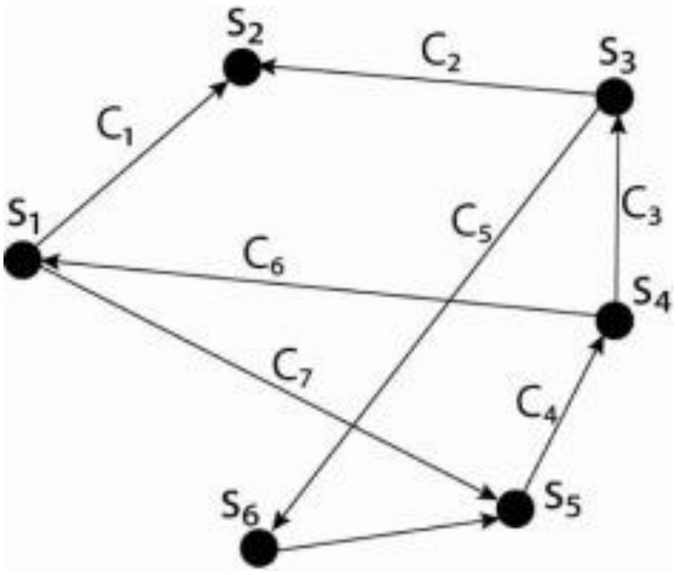
2.3 Directed graph or digraph

When the edges in a graph have a direction, we get a digraph or oriented graph. An oriented graph G is formed of two sets, the first S is the set containing the vertices {s1, s2,…, sn} and the second C contains the edges {c1, c2,…, cn}, denoted as G=(S, C).

An edge c is a pair or an ordered couple of vertices. If c={x, y), then edge c goes from x to y. The initial endpoint of c is y and the final endpoint is y.
The external degree d+(x) is denoted as the number of edges with x as the initial endpoint and the internal degree d-(x) is the number of edges with x as a final endpoint, meaning that the degree of a vertex of a diagraph is d(x)=d+(x)+d-(x).
A directed graph can be symmetric if the direction of the edges is reversed from the initial graph.

2.3.1 Path and circuit in a digraph

A path is a sequence of vertices and edges that link one vertex to another.
In a digraph, what we call distance is the length of the shortest path linking two vertices.
In Figure 2.20, the distance of d(s4, s1)=1, d(s5, s2)=3, d(s2, s3)=∞ (no path).



A diagraph is said to be strongly connected if each vertex can be reached from the other vertices by at least one path.

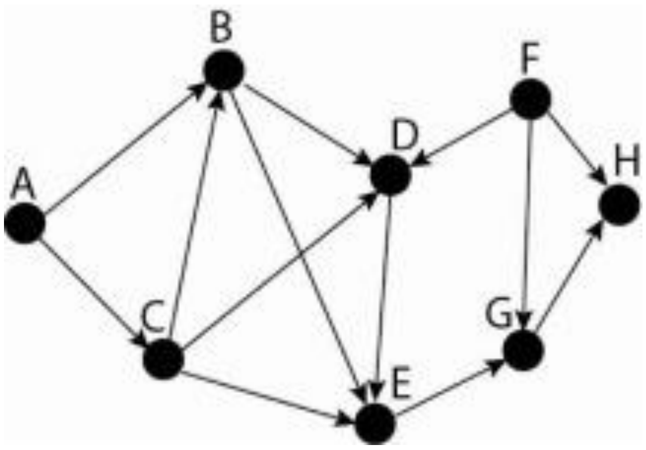
2.3.2 Absence of circuit in a digraph

For applications related particularly to task scheduling, digraphs without circuits are very important. This type of graph is also said to be acyclic.

Let G=(S, C) a diagraph. It is without circuit if and only if the set of its vertices has been topologically sorted.

Topological sorting is a depth-first search of the graph where a vertex is always visited before its successors, each vertex being given a number denoting the order.

In Figure 2.21, we consider digraph G=(S, C) with S={A, B, C, D, E, F, G, H} and C={(A, B), (A, C), (B, D), (B, E), (C, B), (C, D), (C, E), (D, F), (D, E), (E, G), (F, G), (F, H), (G, H)}.



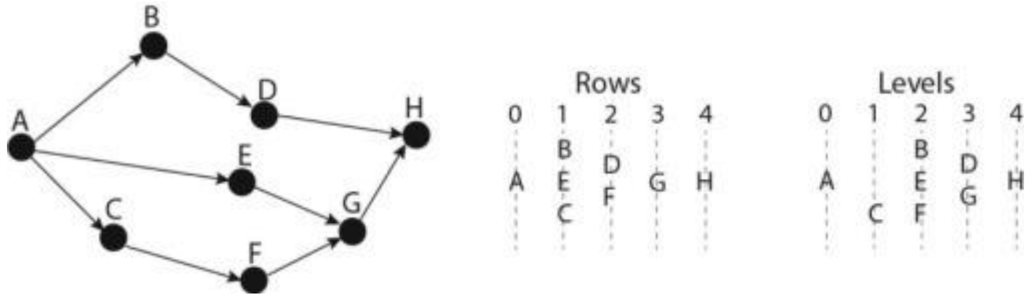
An acceptable topological order could be the following sequence of vertices: F, A, C, B, D, E, G, H, meaning that digraph G has no circuit.

A digraph without circuit contains at least a vertex x of the inferior degree or is equal to 0, so d-x(x) = 0.

To express the fact that a digraph has no circuit we can also use the notion of row r(x) and level n(x).

Let G=(S, C) a digraph. Row r(x), with x ∈ S, is the number of edges in the longest path with x as a terminal endpoint. The level n(x), with x ∈ S, is the number of edges in the longest path with x as an initial endpoint.

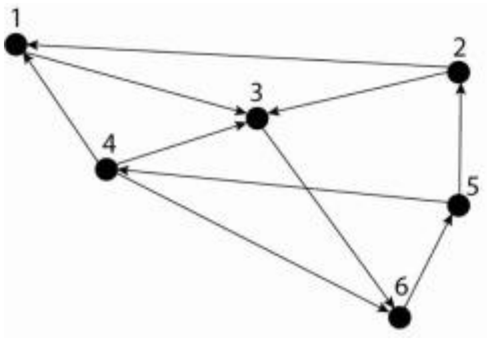
Each digraph without circuit can be ordered by ascending row or descending level.



2.3.3 Adjacency matrix

A digraph can be represented by an adjacency matrix. This is a double entry table with n lines and m columns representing the vertices of the digraph and whose intersections designate a vertex. This matrix is always square and it always has 0 on its diagonal unless it is a loop. It is not symmetric.

On the right, Figure 2.23 shows the adjacency matrix representing the graph. When an edge joins two edges the value in the matrix is 1.

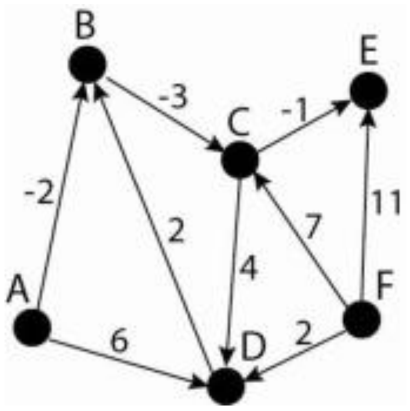


$$M=\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

There are other possible uses for the adjacency matrix, which has very interesting properties. These uses will be described in the following chapters of this book.

2.3.4 Valued graph matrix

As for the adjacency matrix, a valued graph can be represented by a square matrix. Each coefficient corresponds to the value (weight, cost) represented by an edge.
If G = (S, C, v) is the graph in Figure 2.24, the valuation matrix of G can be defined as the square matrix M=m(i, j) with a size n × n respecting:



Mij=v(i,j)if(i,j)∈Cotherwise∞

We get the associated matri

Nodes, Edges, and Network Measures

Adjacency matrix

An alternative to the adjacency list is an adjacency matrix. In an adjacency matrix, a grid is set up that lists all the nodes on both the X-axis (horizontal) and the Y-axis (vertical). Then, values are filled in to the matrix to indicate if there is or is not an edge between every pair of nodes. Typically, a 0 indicates no edge and a 1 indicates an edge

Notice a couple of things about this matrix. First, the diagonal is all zeroes because there are no edges between a node and itself in our example. Some networks do allow for self-loops. For example, in an email network, if a person emails himself, there could be a link from one node to itself, and thus there would be a 1 on the diagonal. Second, the matrix is symmetric. The numbers in the first row are the same as the numbers in the first column. The numbers in the second row are the same as the numbers in the second column. This is because the graph is undirected. Just as in the adjacency list, where the order of pairs in an undirected graph didn't matter

If we have a directed network, the matrix will not necessarily be symmetric. For example, consider the small network in Figure 2.5. In this case, there are edges from A to C, and C to A, and from A to B, but the reciprocal edge from B to A is absent. Thus, we only record a 1 for the A–B edge, and record a 0 for the B–A edge. The adjacency matrix would look like this:

In the examples we have seen so far, we have been recording a 1 in the matrix to indicate an edge is present, and a 0 when there is no edge. This scheme can be altered to show the weight of an edge as well. To do this, we replace the 1 with the edge weight. Using the values from Figure 2.4, we would have a weight of 4 between Tom Hanks and Gary Sinise. The matrix would look like this:

Parallel Program Development

Peter S. Pacheco, in An Introduction to Parallel Programming, 2011
6.2.11 Implementation of tree search using MPI and static partitioning
The vast majority of the code used in the static parallelizations of tree search using Pthreads and OpenMP is taken straight from the second implementation of serial, iterative tree search. In fact, the only differences are in starting the threads, the initial partitioning of the tree, and the Update_best_tour function. We might therefore expect that an MPI implementation would also require relatively few changes to the serial code, and this is, in fact, the case.

There is the usual problem of distributing the input data and collecting the results. In order to construct a complete tour, a process will need to choose an edge into each vertex and out of each vertex. Thus, each tour will require an entry from each row and each column for each city that's added to the tour, so it would clearly be advantageous for each process to have access to the entire adjacency matrix. Note that the adjacency matrix is going to be relatively small. For example, even if we have 100 cities, it's unlikely that the matrix will require more than 80,000 bytes of storage, so it makes sense to simply read in the matrix on process 0 and broadcast it to all the processes.

Once the processes have copies of the adjacency matrix, the bulk of the tree search can proceed as it did in the Pthreads and OpenMP implementations. The principal differences lie in

- partitioning the tree,
- checking and updating the best tour, and
- after the search has terminated, making sure that process 0 has a copy of the best tour for output.

Enumerated Types

An enumerated type is a type whose legal values consist of a fixed set of constants. Common examples include compass directions, which take the values North, South, East and West and days of the week, which take the values Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, and Saturday.

In the Java programming language, you define an enumerated type by using the enum keyword. For example, you would specify a days of the week enumerated type as:

```
enum Days { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
FRIDAY, SATURDAY };
```

Notice that by convention the names of an enumerated type's values are spelled in uppercase letters.

You should use enumerated types any time you need to represent a fixed set of constants. That includes natural enumerated types such as the planets in our solar system, the days of the week, and the suits in a deck of cards as well as sets where you know all possible values at compile time, for example the choices on a menu, rounding modes, command line flags, and so on.

Java programming language enumerated types are much more powerful than their counterparts in other languages, which are just glorified integers.

The enum declaration defines a class (called an enum type). These are the most important properties of enum types:

- Printed values are informative.
- They are typesafe.

- They exist in their own namespace.
- The set of constants is not required to stay fixed for all time.
- You can switch on an enumeration constant.
- They have a static values method that returns an array containing all of the values of the enum type in the order they are declared. This method is commonly used in combination with the for-each construct to iterate over the values of an enumerated type. (See The for Statement.)
- You can provide methods and fields, implement interfaces, and more.
- They provide implementations of all the Object methods. They are Comparable and Serializable, and the serial form is designed to withstand changes in the enum type.

In the following example, Planet is an enumerated type that represents the planets in the solar system. A Planet has constant mass and radius properties. Each enum constant is declared with values for the mass and radius parameters that are passed to the constructor when it is created. Note that the constructor for an enum type is implicitly private. If you attempt to create a public constructor for an enum type, the compiler displays an error message.

```
public enum Planet {
MERCURY (3.303e+23, 2.4397e6),
VENUS   (4.869e+24, 6.0518e6),
EARTH   (5.976e+24, 6.37814e6),
MARS    (6.421e+23, 3.3972e6),
JUPITER (1.9e+27,   7.1492e7),
SATURN  (5.688e+26, 6.0268e7),
URANUS  (8.686e+25, 2.5559e7),
NEPTUNE (1.024e+26, 2.4746e7),
PLUTO   (1.27e+22,  1.137e6);

private final double mass;   //in kilograms
private final double radius; //in meters
Planet(double mass, double radius) {
this.mass = mass;
this.radius = radius;
}
public double mass()   { return mass; }
public double radius() { return radius; }

//universal gravitational constant  (m3 kg-1 s-2)
public static final double G = 6.67300E-11;

public double surfaceGravity() {
return G * mass / (radius * radius);
}
public double surfaceWeight(double otherMass) {
return otherMass * surfaceGravity();
}
}
```

In addition to its properties, Planet has methods that allow you to retrieve the surface gravity and weight of an object on each planet. Here is a sample program that takes your weight on earth (in any unit) and calculates and prints your weight on all of the planets (in the same unit):

```
public static void main(String[] args) {
double earthWeight = Double.parseDouble(args[0]);
double mass = earthWeight/EARTH.surfaceGravity();
for (Planet p : Planet.values()) {
System.out.printf("Your weight on %s is %f%n",
p, p.surfaceWeight(mass));
}
}
```

Here's the output:

```
$ java Planet 175
Your weight on MERCURY is 66.107583
Your weight on VENUS is 158.374842
Your weight on EARTH is 175.000000
Your weight on MARS is 66.279007
Your weight on JUPITER is 442.847567
Your weight on SATURN is 186.552719
Your weight on URANUS is 158.397260
Your weight on NEPTUNE is 199.207413
Your weight on PLUTO is 11.703031
```

The Pólya enumeration theorem, also known as the Redfield–Pólya theorem and Pólya counting, is a theorem in combinatorics that both follows from and ultimately generalizes Burnside's lemma on the number of orbits of a group action on a set. The theorem was first published by J. Howard Redfield in 1927. In 1937 it was independently rediscovered by George Pólya, who then greatly popularized the result by applying it to many counting problems, in particular to the enumeration of chemical compounds.

The Pólya enumeration theorem has been incorporated into symbolic combinatorics and the theory of combinatorial species

Simplified, unweighted version

Let X be a finite set and let G be a group of permutations of X (or a finite symmetry group that acts on X). The set X may represent a finite set of beads, and G may be a chosen group of permutations of the beads. For example, if X is a necklace of n beads in a circle, then rotational symmetry is relevant so G is the cyclic group Cn, while if X is a bracelet of n beads in a circle, rotations and reflections are relevant so G is the dihedral group Dn of order 2n. Suppose further that Y is a finite set of colors — the colors of the beads — so that YX is the

set of colored arrangements of beads (more formally: YX is the set of functions .) Then the group G acts on YX. The Pólya enumeration theorem counts the number of orbits under G of colored arrangements of beads by the following formula:

where is the number of colors and c(g) is the number of cycles of the group element g when considered as a permutation of X.

Full, weighted version

In the more general and more important version of the theorem, the colors are also weighted in one or more ways, and there could be an infinite number of colors provided that the set of colors has a generating function with finite coefficients. In the univariate case, suppose that

is the generating function of the set of colors, so that there are fw colors of weight w for each integer w ≥ 0. In the multivariate case, the weight of each color is a vector of integers and there is a generating function f(t1, t2, ...) that tabulates the number of colors with each given vector of weights.

The enumeration theorem employs another multivariate generating function called the cycle index:

where n is the number of elements of X and ck(g) is the number of k-cycles of the group element g as a permutation of X.

A colored arrangement is an orbit of the action of G on the set 'YX (where Y is the set of colors and YX denotes the set of all functions φ: X→Y). The weight of such an arrangement is defined as the sum of the weights of φ(x) over all x in X. The theorem states that the generating function F of the number of colored arrangements by weight is given by:

To reduce to the simplified version given earlier, if there are m colors and all have weight 0, then f(t) = m and

Graph theoretic algorithm

There are different ways to store graphs in a computer system. The data structure used depends on both the graph structure and the algorithm used for manipulating the graph. Theoretically one can distinguish between list and matrix structures but in concrete applications the best structure is often a combination of both. List structures are often preferred for sparse graphs as they have smaller memory requirements. Matrix structures on the other hand provide faster access for some applications but can consume huge amounts of memory. Implementations of sparse matrix structures that are efficient on modern parallel computer architectures are an object of current investigation.

List structures include the edge list, an array of pairs of vertices, and the adjacency list, which separately lists the neighbors of each vertex: Much like the edge list, each vertex has a list of which vertices it is adjacent to.

Matrix structures include the incidence matrix, a matrix of 0's and 1's whose rows represent vertices and whose columns represent edges, and the adjacency matrix, in which both the rows and columns are indexed by vertices. In both cases a 1 indicates two adjacent objects and a 0 indicates two non-adjacent objects. The degree matrix indicates the degree of vertices. The Laplacian matrix is a modified form of the adjacency matrix that incorporates information about the degrees of the vertices, and is useful in some calculations such as Kirchhoff's theorem on the number of spanning trees of a graph. The distance matrix, like the adjacency matrix, has both its rows and columns indexed by vertices, but rather than containing a 0 or a 1 in each cell it contains the length of a shortest path between two vertices.

Problems

Enumeration

There is a large literature on graphical enumeration: the problem of counting graphs meeting specified conditions. Some of this work is found in Harary and Palmer (1973).

Subgraphs, induced subgraphs, and minors[edit]

A common problem, called the subgraph isomorphism problem, is finding a fixed graph as a subgraph in a given graph. One reason to be interested in such a question is that many graph properties are hereditary for subgraphs, which means that a graph has the property if and only if all subgraphs have it too. Unfortunately, finding maximal subgraphs of a certain kind is often an NP-complete problem. For example:

- Finding the largest complete subgraph is called the clique problem (NP-complete).

One special case of subgraph isomorphism is the graph isomorphism problem. It asks whether two graphs are isomorphic. It is not known whether this problem is NP-complete, nor whether it can be solved in polynomial time.

A similar problem is finding induced subgraphs in a given graph. Again, some important graph properties are hereditary with respect to induced subgraphs, which means that a graph has a property if and only if all induced subgraphs also have it. Finding maximal induced subgraphs of a certain kind is also often NP-complete. For example:

- Finding the largest edgeless induced subgraph or independent set is called the independent set problem (NP-complete).

Still another such problem, the minor containment problem, is to find a fixed graph as a minor of a given graph. A minor or subcontraction of a graph is any graph obtained by taking a subgraph and contracting some (or no) edges. Many graph properties are hereditary for minors, which means that a graph has a property if and only if all minors have it too. For example, Wagner's Theorem states:

- A graph is planar if it contains as a minor neither the complete bipartite graph K3,3 (see the Three-cottage problem) nor the complete graph K5.

A similar problem, the subdivision containment problem, is to find a fixed graph as a subdivision of a given graph. A subdivision or homeomorphism of a graph is any graph obtained by subdividing some (or no) edges. Subdivision containment is related to graph properties such as planarity. For example, Kuratowski's Theorem states:

- A graph is planar if it contains as a subdivision neither the complete bipartite graph K3,3 nor the complete graph K5.

Another problem in subdivision containment is the Kelmans–Seymour conjecture:

- Every 5-vertex-connected graph that is not planar contains a subdivision of the 5-vertex complete graph K5.

Another class of problems has to do with the extent to which various species and generalizations of graphs are determined by their point-deleted subgraphs. For example: