A Turing Machine (TM) is a mathematical model which consists of an infinite length tape divided into cells on which input is given. It consists of a head which reads the input tape. A state register stores the state of the Turing machine. After reading an input symbol, it is replaced with another symbol, its internal state is changed, and it moves from one cell to the right or left. If the TM reaches the final state, the input string is accepted, otherwise rejected.

A TM can be formally described as a 7-tuple $(Q, X, \Sigma, \delta, q_0, B, F)$ where –

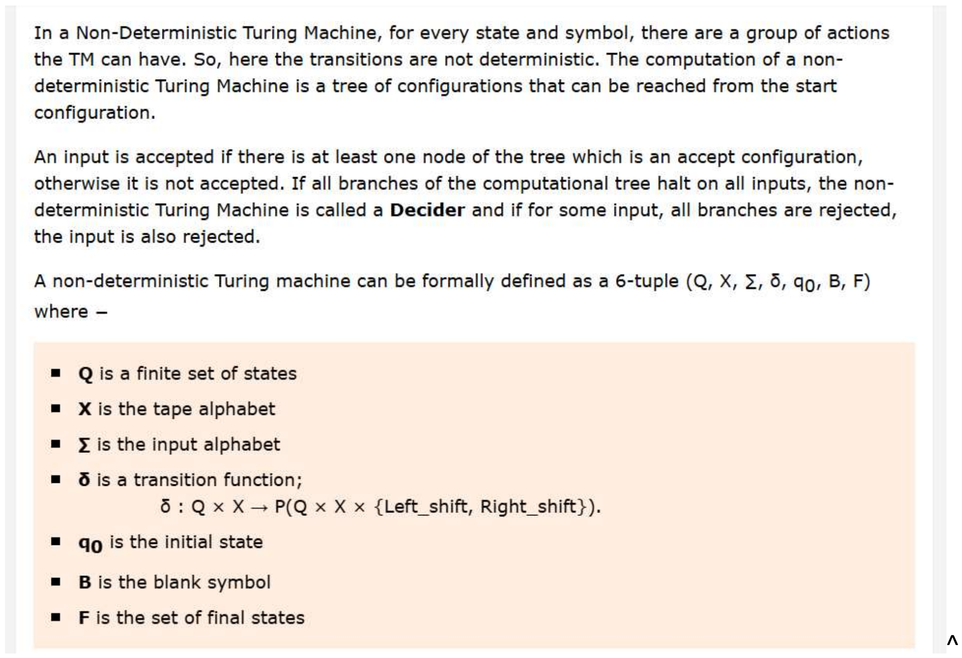
- **Q** is a finite set of states
- **X** is the tape alphabet
- **Σ** is the input alphabet
- **δ** is a transition function; $\delta : Q \times X \rightarrow Q \times X \times \{\text{Left\_shift}, \text{Right\_shift}\}$.
- **q_0** is the initial state
- **B** is the blank symbol
- **F** is the set of final states

3.2.3 Transition Diagrams

In some situations, **graphical** representation of the next-move (partial) function δ of a Turing Machine may give better idea of the behavior of a TM in comparison to the **tabular** representation of δ.

A **Transition Diagram** of the next-move functions δ of a TM is a graphical representation consisting of a finite number of nodes and (directed) labelled arcs between the nodes. Each node represents a state of the TM and a label on an arc from one state (say p) to a state (say q) represents the information about the required **input symbol say x** for the transition from p to q to take place **and the action** on the part of the control of the TM. The action part consists of (i) the symbol say y to be written in the current cell and (ii) the movement of the tape Head.

Then the label of an arc is generally written as x/(y, M) where M is L, R or N.

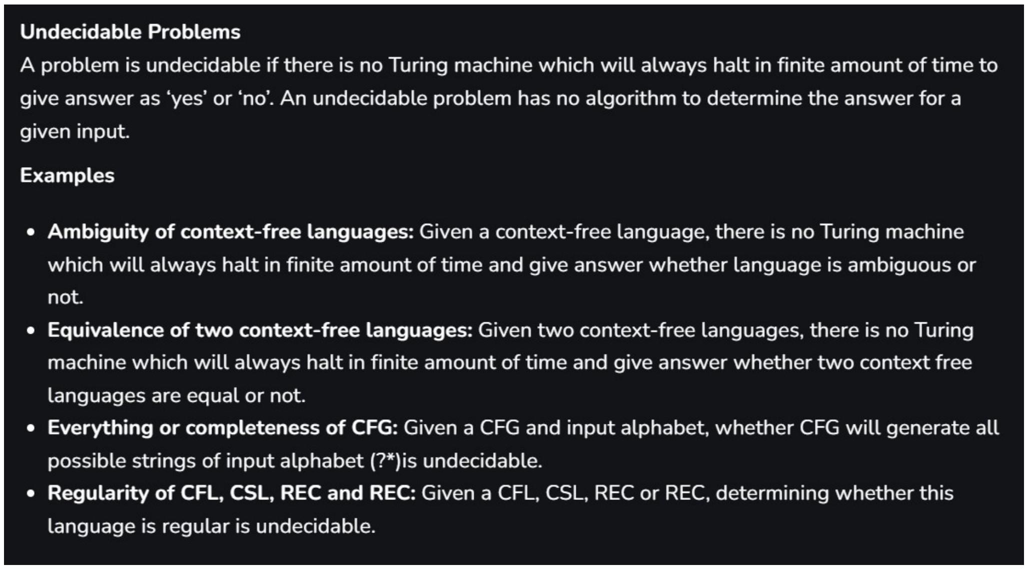


In a Non-Deterministic Turing Machine, for every state and symbol, there are a group of actions the TM can have. So, here the transitions are not deterministic. The computation of a non-deterministic Turing Machine is a tree of configurations that can be reached from the start configuration.

An input is accepted if there is at least one node of the tree which is an accept configuration, otherwise it is not accepted. If all branches of the computational tree halt on all inputs, the non-deterministic Turing Machine is called a **Decider** and if for some input, all branches are rejected, the input is also rejected.

A non-deterministic Turing machine can be formally defined as a 6-tuple $(Q, X, \Sigma, \delta, q_0, B, F)$ where −

- **Q** is a finite set of states
- **X** is the tape alphabet
- **Σ** is the input alphabet
- **δ** is a transition function;
  $\delta : Q \times X \rightarrow P(Q \times X \times \{Left\_shift, Right\_shift\})$.
- $\mathbf{q_0}$ is the initial state
- **B** is the blank symbol
- **F** is the set of final states



Undecidable Problems
A problem is undecidable if there is no Turing machine which will always halt in finite amount of time to give answer as 'yes' or 'no'. An undecidable problem has no algorithm to determine the answer for a given input.

Examples

- **Ambiguity of context-free languages:** Given a context-free language, there is no Turing machine which will always halt in finite amount of time and give answer whether language is ambiguous or not.
- **Equivalence of two context-free languages:** Given two context-free languages, there is no Turing machine which will always halt in finite amount of time and give answer whether two context free languages are equal or not.
- **Everything or completeness of CFG:** Given a CFG and input alphabet, whether CFG will generate all possible strings of input alphabet (?*)is undecidable.
- **Regularity of CFL, CSL, REC and REC:** Given a CFL, CSL, REC or REC, determining whether this language is regular is undecidable.

THE HALTING PROBLEM

There are many problems which are not computable. But, we start with a problem which is important and that at the same time gives us a platform for developing later results. One such problem is the halting problem. Algorithms may contain loops that may be infinite or finite in length. The amount of work done in an algorithm usually depends on the data input. Algorithms may consist of various numbers of loops, nested or in sequence. Informally, the **Halting problem** can be put as:

Given a Turing machine M and an input *w* to the machine M, determine if the machine M will eventually halt when it is given input *w*.

Trial solution: Just run the machine M with the given input *w*.

- If the machine M halts, we know the machine halts.
- But if the machine doesn't halt in a reasonable amount of time, we cannot conclude that it won't halt. Maybe we didn't wait long enough.

What we need is an algorithm that can determine the correct answer for any M and *w* by performing some analysis on the machine's description and the input. But, it is shown by Alan Turing that no such algorithm exists.

Q3 How TM is different from Finite Automata?

Ans.3 The *Finite Automata* is used only as **accepting devices** for languages in the sense that the automata, when given an input string from a language, tells whether the string is acceptable or not. ***The Turing Machines are designed to play at least the following three different roles:***

(i) **As accepting devices for languages**, similar to the role played by FAs .

(ii) **As a computer of functions**. In this role, a TM represents a particular function (say the SQUARE function which gives as output the square of the integer given as input). Initial input is treated as representing an argument of the function. And the (final) string on the tape when the TM enters the Halt State is treated as

representative of the value obtained by an application of the function to the argument represented by the initial string.

(iii) **As an enumerator of strings of a language** that outputs the strings of a language, one at a time, in some systematic order, i.e, as a list.

Q2 Define P and NP Complexity Classes.

P denotes the class of all problems, for each of which there is at least one *known* polynomial time Deterministic TM solving it.
NP denotes the class of all problems, for each of which, there is at least one known Non-Deterministic polynomial time solution. However, this solution may not be reducible to a polynomial time algorithm, i.e, to a polynomial time DTM.

We can define p and NP to be the classes of languages because problems from graph theory, combinatorics can often be formulated as language recognition problems. Consider a problem which requires an answer in form of " Yes" or "No" for each instance. Each instance of a problem can be encoded as a string and reformulate the problem as one of recognizing the language consisting of all the strings representing those instance of the problem whose answer is "Yes".

With this logic P and NP classes of complexities can be defined as classes of languages:

Definition

P = { L| L can be decided or accepted by a DTM(Deterministic TM) in polynomial time}

NP = { L| L can be decided or accepted by a Nondeterministic TM in polynomial time}

NP is a set of decision problems (with Yes or No answer) that can be solved by NDTM in polynomial time

2.2 PIGEONHOLE PRINCIPLE

Let us start with considering a situation where we have 10 boxes and 11 objects to be placed in them. Wouldn't you agree that regardless of the way the objects are placed in the two boxes at least one box will have more than one object in it? On the face of it, this seems obvious. This is actually an application of the pigeonhole principle, which we now state.

Theorem 1 (The Pigeonhole Principle): Let there be n boxes and (n+1) objects. Then, under any assignment of objects to the boxes, there will always be a box with more than one object in it.
This can be reworded as: if m pigeons occupy n pigeonholes, where m > n, then there is at least one pigeonhole with two or more pigeons in it.

1 The pigeonhole

Proof: Let us label the n pigeonholes 1, 2, …, n, and the m pigeons $p_1, p_2, \ldots, p_m$. Now, beginning with p_1, we assign one each of these pigeons the holes numbered 1, …, n, respectively. Under this assignment, each hole has one pigeon, but there are still (m–n) pigeons left. So, in whichever way we place these pigeons, at least one hole will have more than one pigeon in it. This completes the proof!

This result appears very trivial, but has many applications. For example, using it you can show that:

- if 8 people are picked in any way from a group, at least 2 of them will have been born on the same weekday.
- in any group of 13 people, at least two are born in the same month.

Principle of Inclusion and Exclusion (PIE) is a combinatorial method used to calculate the cardinality (size) of the union of multiple sets. The principle is particularly useful when sets overlap, and we want to avoid double-counting elements that belong to more than one set.

For example, when calculating the number of people who own a dog or a cat, PIE adds up all dog owners and subtracts all cat owners. This includes:

- Adding the sizes of each individual set.
- Subtracting the sizes of the pairwise intersections to avoid double-counting.
- Adding back the size of the intersection of all three sets, as it was subtracted too many times.
- Subtracting the sizes of intersection of four set, as it was added twice in the last step.
- and so on.

We can continue this till intersection of all the element is either added or subtracted at the end.

Formula for Principle of Inclusion and Exclusion

In general, for n sets $A_1, A_2, \ldots, A_n$:

$$|A_1 \cup A_2 \cup \cdots \cup A_n| = \sum_{i=1}^{n} |A_i| - \sum_{1 \le i < j \le n} |A_i \cap A_j| + \sum_{1 \le i < j < k \le n} |A_i \cap A_j \cap A_k| - \cdots + (-1)^{n+1} |A_1 \cap A_2 \cap \cdots \cap A_n|$$

Where,

- The first term sums the sizes of all individual sets.
- The second term subtracts the sizes of all pairwise intersections.
- The third term adds back the sizes of all three-way intersections.
- This alternating pattern continues, with the signs changing for each subsequent term, until the intersection of all N sets is reached.

Example 6: Show that $[(p \rightarrow q) \wedge \sim q] \rightarrow \sim p$ is a tautology.

Solution: $[(p \rightarrow q) \wedge \sim q] \rightarrow \sim p$

$\equiv [(\sim p \vee q) \wedge \sim q] \rightarrow \sim p$, using E18, and symmetricity of $\equiv$.

$\equiv [(\sim p \wedge \sim q) \vee (q \wedge \sim q)] \rightarrow \sim p$, by De Morgan's laws.

$\equiv [(\sim p \wedge \sim q) \vee F] \rightarrow \sim p$, since $q \wedge \sim q$ is always false.

$\equiv (\sim p \wedge \sim q) \rightarrow \sim p$, using E18.

Which is tautology.

Example 7: Show that $(p \rightarrow \sim q) \wedge (p \rightarrow \sim r) \equiv \sim [p \wedge (q \vee r)]$.

Solution: We shall start with the statement on the left hand side of the equivalence that we have to prove. Then, we shall apply the laws we have listed above, or the equivalence in E 18, to obtain logically equivalent statements. We shall continue this process till we obtain the statement on the right hand side of the equivalence given above. Now

$(p \rightarrow \sim q) \wedge (p \rightarrow \sim r)$

$\equiv (\sim p \vee q) \wedge (\sim p \vee \sim r)$, by E18

$\equiv \sim p \vee (\sim q \wedge \sim r)$, by distributivity

$\equiv \sim p \vee [\sim (q \vee r)]$, by De Morgan's laws

$\equiv \sim [p \wedge (q \vee r)]$, by De Morgan's laws

So we have proved the equivalence that we wanted to.

Dynamic programming follows the principal of optimality. If a problem have an optimal structure, then definitely it has principal of optimality.A problem has optimal sub structure if an optimal solution can be constructed efficiently from optimal solution of its sub-problems. Principal of optimality shows that a problem can be solved by taking a sequence of decision to solve the optimization problem. In dynamic programming in every stage we takes a decision. The Principle of Optimality states that components of a globally optimum solution must themselves be optimal.

A problem is said to satisfy the Principle of Optimality if the sub solutions of an optimal solution of the problem are themselves optimal solutions for their sub problems.

Examples:

- The shortest path problem satisfies the Principle of Optimality.
- This is because if $a, x_1, x_2, \ldots, x_n$is a shortest path from node a to node b in a graph, then the portion of x_i to x_j on that path is a shortest path from x_i to x_j.
- The longest path problem, on the other hand, does not satisfy the Principle of Optimality. Forexample the undirected graph of nodes a, b, c, d, and e and edges$(a, b), (b, c), (c, d), (d, e)$and (e, a). That is, G is a ring. The longest (noncyclic) path from a to d to a,b,c,d. The sub-path from b to c on that path is simply the edge b,c. But that is not the longest path from b to c. Rather b,a,e,d,c is the longest path. Thus, the sub path on a longest path is not necessarily a longest path.

Steps of Dynamic Programming:

Dynamic programming has involve four major steps:

1. Develop a mathematical notation that can express any solution and sub solution for the problem at hand.
2. Prove that the Principle of Optimality holds.
3. Develop a recurrence relation that relates a solution to its sub solutions, using the math notation of step 1. Indicate what the initial values are for that recurrence relation, and which term signifies the final solution.
4. Write an algorithm to compute the recurrence relation.
   - Steps 1 and 2 need not be in that order. Do what makes sense in each problem.
   - Step 3 is the heart of the design process. In high level algorithmic design situations, one can stop at step 3.
   - Without the Principle of Optimality, it won't be possible to derive a sensible recurrence relation in step 3.
   - When the Principle of Optimality holds, the 4 steps of DP are guaranteed to yield an optimal solution. No proof of optimality is needed.

- Matrix multiplication is a binary operation of multiplying two or more matrices one by one that are conformable for multiplication. For example two matrices A, B having the dimensions of $p \times q$ and $s \times t$ respectively; would be conformable for $A \times B$multiplication only if q==s and for $B \times A$ multiplication only if t==p.

- Matrix multiplication is associative in the sense that if A, B, and C are three matrices of order$m \times n$, $n \times p$ and $p \times q$ then the matrices (AB)C and A(BC) are defined as (AB)C = A (BC) and the product is an $m \times q$ matrix.
- Matrix multiplication is not commutative. For example two matrices A and B having dimensions$n \times n$ and$n \times n$ then the matrix AB = BA can't be defined. Because BA are not conformable for multiplication even if AB are conformable for matrix multiplication.
- For 3 or more matrices, matrix multiplication is associative, yet the number of scalar multiplications may very significantly depending upon how we pair the matrices and their product matrices to get the final product.

Example: Suppose there are three matrices A is 100×10, B is 10×50 and C is 50×5, then number of multiplication required for (AB)C is $AB = 100 \times 10 \times 5 = 50000$, $(AB)C = 100 \times 50 \times 5 = 25000$. Total multiplication for (AB)C= 75000 mulplication.Similarly number of multiplication required for A(BC) is BC = $10 \times 50 \times 5$ = 2500 and A(BC) = $100 \times 10 \times 5$ = 5000. Total multiplication for A(BC) = 7500

Binary Search Tree: A binary search tree is a binary tree such that all the key smaller than root are on the left side and all the keys greater than root are on the right side. Figure shown below is a binary search tree in which 40 is root node all the node in the left sub tree of 40 is smaller and all the node in the right sub tree of 40 is greater. Now question is why keys are arrange in order?. Because it gives an advantage in searching. Suppose I want to search 30, we will start searching from the root node. Check whether root is a 30, no, then 30 is smaller than 40, definitely 30 will be in left sub tree of 40. Go in the left sub tree, check left sub tree root is 30, no, 20 is smaller than 30, definitely 30 will be in the right hand side. Yes now 30 is found. So to search the 30 how many comparison are needed. Out of 7 elements to search 30 it, takes 3 comparison. Number of comparison required to search any element in a binary search tree is depend upon the number of levels in a binary search tree. We are searching for the key that are already present in the binary search tree, it is a successful search. Now let's assume that key is 9 and search is an unsuccessful search because 9 is not found in the binary search tree. Total number of comparison it takes is also 3. There are a two possibility to search any element successful search and unsuccessful search. In both the cases we need to know the amount of comparison we are doing. That amount of comparison is nothing but the cost of searching in a binary search tree.

What is Binomial Theorem?

Binomial Theorem is also called as Binomial Expansion describe the powers in algebraic equations. Binomial Theorem helps us to find the expanded polynomial without multiplying the bunch of binomials at a time. The expanded polynomial will always contain one term more than the power you are expanding.
Following formula shows the General formula to expand the algebraic equations by using Binomial Theorem:

$$(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$$

Where, n= positive integer power of algebraic equation and $\binom{n}{k}$= read as "n choose k".
According to theorem, expansion goes as following for any of the algebraic equation containing any positive power,

$$(a+b)^n = \binom{n}{0} a^n b^0 + \binom{n}{1} a^{n-1} b^1 + \binom{n}{2} a^{n-2} b^2 + \ldots + \binom{n}{n-1} a^1 b^{n-1} + \binom{n}{n} a^n b^n$$

Example 4: Reduce the following Boolean expressions to a simpler form.

(a) $X(x_1, x_2) = (x_1 \wedge x_2) \wedge (x_1 \wedge x'_2)$;
(b) $X(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_3)$.

Solution: (a) Here we can write

$$\begin{aligned}
(x_1 \wedge x_2) \wedge (x_1 \wedge x'_2) &= ((x_1 \wedge x_2) \wedge x_1) \wedge x'_2 && \text{(Associative law)}\\
&= (x_1 \wedge x_2) \wedge x'_2 && \text{(Absorption law)}\\
&= x_1 \wedge (x_2 \wedge x'_2) && \text{(Associative law)}\\
&= x_1 \wedge \mathbf{O} && \text{(Complementation law)}\\
&= \mathbf{O}. && \text{(Identity law)}
\end{aligned}$$

Thus, in its simplified form, the expression given in (a) above is **O**, i.e., a **null expression.**

(b) We can write

$$\begin{aligned}
&(x_1 \wedge x_2) \vee (x_1 \wedge x'_2 \wedge x_3) \vee (x_1 \wedge x_3)\\
&= [x_1 \wedge \{x_2 \vee (x'_2 \wedge x_3\}] \wedge (x_1 \wedge x_3) && \text{(Distributive law)}\\
&= [x_1 \wedge \{(x_2 \vee x'_2) \wedge (x_2 \vee x_3)\}] \wedge (x_1 \wedge x_3) && \text{(Distributive law)}\\
&= [x_1 \wedge \{\mathbf{I} \wedge (x_2 \vee x_3)\}] \wedge (x_1 \wedge x_3) && \text{(Complementation law)}\\
&= [x_1 \wedge (x_2 \vee x_3)] \wedge (x_1 \wedge x_3) && \text{(Identity law)}\\
&= [(x_1 \wedge x_2) \vee (x_1 \wedge x_3)] \wedge (x_1 \wedge x_3) && \text{(Distributive law)}\\
&= [(x_1 \wedge x_2) \wedge (x_1 \wedge x_3)] \vee [(x_1 \wedge x_3) \wedge (x_1 \wedge x_3)] && \text{(Distributive law)}\\
&= (x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_3) && \text{(Idemp.,\& assoc. laws)}\\
&= x_1 \wedge [(x_2 \wedge x_3) \vee x_3] && \text{(Distributive law)}\\
&= x_1 \wedge x_3 && \text{(Absorption law)}
\end{aligned}$$

Thus, the simplified form of the expression given in (b) is $(x_1 \wedge x_3)$.

1.4.1 Cartesian product

Very often we deal with several sets at a time, and we need to study their combined action. For instance, combinations of a set of teachers and a set of students. In such a situation we can take a product of these sets to handle them simultaneously. To understand this product let us first consider the following definitions.

Definition: An **ordered pair**, usually denoted by (x,y), is a pair of elements x and y of some sets. This is ordered in the sense that $(x,y) \neq (y,x)$ whenever $x \neq y$,that is , the order of placing of the element in the pair matters.

Any two ordered pairs (x,y) and (a,b) are equal iff x = a and y = b.

For example if, A= {a,b,c}and B= {x,y,z},then

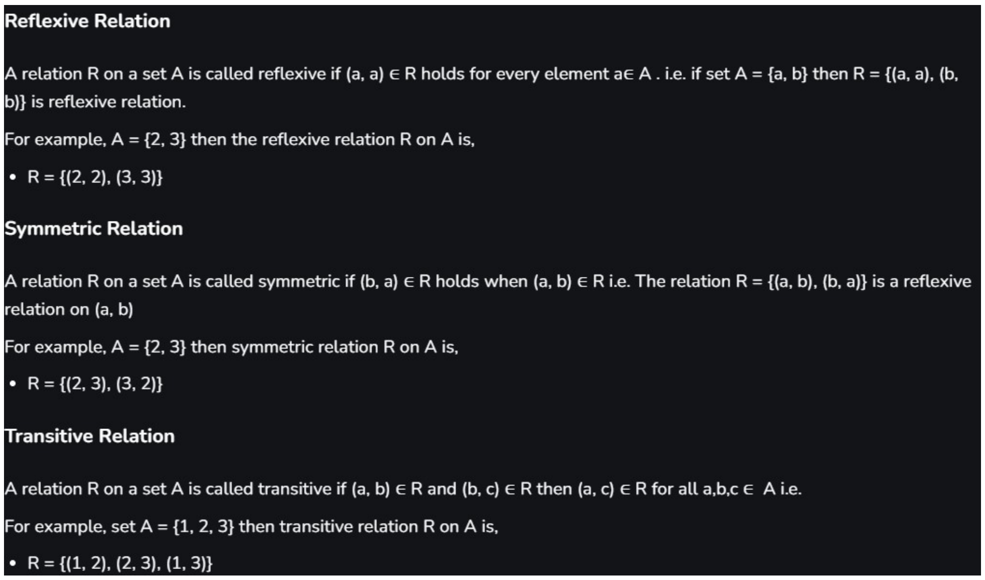
$A \times B = \{a,b,c\} \times \{x,y,z\} = \{(a,x), (a,y),(a,z),(b,x),(b,y),(b,z),(c,x),(c,y), (c,z)\}$,and
$B \times A = \{x,y,z\} \times \{a,b,c\} = \{(x,a),(x,b),(x,c),(y,a),(y,b),(y,c),(z,a),(z,b),(z,c)\}$.

Definition: A **relation between two sets A and B** is a subset of $A \times B$. Any subset of $A \times A$ is **a relation on the set A**.

For instance, if A ={1,2,3} and B = {p,q}, then the subset {(1,p),(2,q),(2,p)} is a relation on $A \times B$. And {(1,1), (2,3)} is a relation on A.

Also, $R= \{(x,y) \in \mathbf{N} \times \mathbf{N} : x > y\}$ is a relation on **N,** the set of natural numbers, since $R \subseteq \mathbf{N} \times \mathbf{N}$.

If $R \subseteq A \times B$,. we write ${}_x\mathbf{R}_y$ if and only if $(x,y) \in R$(${}_xR_y$ is read as 'x is related to y').



Reflexive Relation

A relation R on a set A is called reflexive if (a, a) ∈ R holds for every element a∈ A . i.e. if set A = {a, b} then R = {(a, a), (b, b)} is reflexive relation.

For example, A = {2, 3} then the reflexive relation R on A is,

- R = {(2, 2), (3, 3)}

Symmetric Relation

A relation R on a set A is called symmetric if (b, a) ∈ R holds when (a, b) ∈ R i.e. The relation R = {(a, b), (b, a)} is a reflexive relation on (a, b)

For example, A = {2, 3} then symmetric relation R on A is,

- R = {(2, 3), (3, 2)}

Transitive Relation

A relation R on a set A is called transitive if (a, b) ∈ R and (b, c) ∈ R then (a, c) ∈ R for all a,b,c ∈ A i.e.

For example, set A = {1, 2, 3} then transitive relation R on A is,

- R = {(1, 2), (2, 3), (1, 3)}

Definition: A **function from a non-empty set A to a non-empty set B** is a subset R of A**x**B such that for each a $\in$A $\exists$a unique b$\in$B such that (a,b)$\in$R. So, this relation satisfies the following two conditions:

(i) for each a $\in$A, there is some b$\in$B such that (a,b) $\in$R

(ii) if (a,b) $\in$R and (a,b′) $\in$R then b = b′.

We usually present functions as a rule associating elements of one set with another. So, let us present the definition again, with this view.

Definition: Let A and B be non-empty sets. A **function** (or a **mapping**) f from A to B is a rule that assigns to each element x in A **exactly one** element y in B. We write this as f: A $\rightarrow$ B, read it as 'f is a function from A to B'.

Note that

(i) to each a $\in$A, f assigns an element of B; and

(ii) to each a $\in$A, f assigns only **one element** of B.

So, for example, suppose A ={1,2,3}, B= {1,4,9,11} and f assigns to each member in A its square values. Then f is a function from A to B. But if A={1,2,3,4}, B={1,4,9,10} and f is the same rule, then f is not a function from A to B since no member of B is assigned to the element 4 in A.

1.5.1 Types of Functions

Here we shall look at different types of mappings.

Onto Mapping: A mapping f: A$\rightarrow$ B is said to an **onto** (or **surjective**) **mapping** if f(A)= B, that is, the range and co-domain coincide. In this case we say that **f maps A onto B**.

For example, *f*: **Z**$\rightarrow$ **Z** : f(x) = x+1, x$\in$**Z**, then every element y in the co-domain **Z** has a pre-image y–1 in the domain **Z**. Therefore, f(**Z**) = **Z** ,and f is an onto mapping.

Injective Mapping: A mapping *f*: A$\rightarrow$ B is said to be **injective** (or **one-one**) if the images of distinct elements of A under f are distinct, i.e., if $x_1 \neq x_2$ in A, then $f(x_1) \neq f(x_2)$ in B. This is briefly denoted by saying f is **1–1**.

For example f: **R**$\rightarrow$ **R** be defined by f(x) = 2x+1, x$\in$**R**, then for x_1 ,x_2 $\in$**R** ($x_1 \neq x_2$) we have $f(x_1) \neq f(x_2)$. So, f is **1–1**.

Bijective Mapping: A mapping f: A$\rightarrow$ B is said to be **bijective** (or **one-one onto,**) if f is both injective and surjective, i.e., one-one as well as onto.

For example, f: **Z**$\rightarrow$ **Z** : f(x) = x+2, x$\in$**Z** is both injective and surjective. So, f is bijective.

2.2.2 Kleene Closure Definition

Suppose an alphabet $\sum$, and define a language in which any string of letters from $\sum$ is a word, even the null string. We shall call this language the closure of the alphabet. We denote it by writing * after the name of the alphabet as a superscript, which is written as $\sum^*$. This notation is sometimes also known as **Kleene Star**.

For a given alphabet $\sum$, the language $\sum^*$ ($sigma^*$) consists of all possible strings, including the null string.

For example, If $\sum = \{z\}$, then, $\sum^* = L_1 = \{\wedge, z, zz, zzz \ldots..\}$

Example 8: If $s = \{cc, d\}$, then

$s^* = \{\wedge$ or any word composed of factors of cc and d$\}$
  $= \{\wedge$ or all strings of c's and d's in which c's occur in even clumps$\}$.
The string ccdcccd is not in s^* since it has a clump of c's of length 3.
$\{x : x \wedge =$ or $x = (cc)^{i_1}\, d^{j_1}\, (cc)^{i_2}\, d^{j_2} \ldots\ldots (cc)^{i_m}\, (d)^{j_m}\}$ where $i_1, j_1, \ldots\ldots i_m, j_m \geq 0$

Positive Closure: If we want to modify the concept of closure to refer to only the concatenation leading to non-null strings from a set s, we use the notation + instead of *. This plus operation is called positive closure.

Theorem 1: For any set s of strings prove that $s^* = (s^*)^* = s^{**}$

Proof: We know that every word in s^{**} is made up of factors from s^*.
Also, every factor from s^* is made up of factors from s.
Therefore, we can say that every word in s^{**} is made up of factors from s.

First, we show $s^{**} \subset s^*$. (i)
Let $x \in s^{**}$.... Then $x = x_1 \ldots.. x_n$ for some $x_1 \in s^*$ which implies $s^{**} \subset s^*$

Next, we show $s^* \subset s^{**}$.
$s^* \subset s^{**}$ (ii)

By above inclusions (i) and (ii), we prove that
$s^* = s^{**}$

Certain sets of strings or languages can be represented in algebraic fashion, then these algebraic expressions of languages are called **regular expressions**. Regular expressions are in **Bold** face. The symbols that appear in regular use of the letters of the alphabet $\sum$ are the symbol for the null string $\wedge$, parenthesis, the star operator, and the plus sign.

The set of regular expressions is defined by the following rules:

1. Every letter of $\sum$ can be made into a regular expression $\wedge$ itself is a regular expression.
2. Φ is a regular expression

1.2 MULTIPLICATION AND ADDITION PRINCIPLES

Let us start with considering the following situation: Suppose a shop sells six styles of pants. Each style is available in 8 lengths, six waist sizes, and four colours. How many different kinds of pants does the shop need to stock?

There are 6 possible types of pants; then for each type, there are 8 possible length sizes; for each of these, there are 6 possible waist sizes; and each of these is available in 4 different colours. So, if you sit down to count all the possibilities, you will find a huge number, and may even miss some out! However, if you apply the multiplication principle, you will have the answer in a jiffy!

So, what is the multiplication principle? There are various ways of explaining this principle. One way is the following:

Suppose that a task/procedure consists of a sequence of subtasks or steps, say, Subtask 1, Subtask 2,…, Subtask k. Furthermore, suppose that Subtask 1 can be performed in n_1, ways, Subtask 2 can be performed in n_2 ways after Subtask 1 has been performed, Subtask 3 can be performed in n_3 ways after Subtask 1 and Subtask 2 have been performed, and so on. Then **the multiplication principle** says that the number of ways in which the whole task can be performed is $n_1.n_2….n_k$.

Example 1: Suppose we want to choose two persons from a party consisting of 35 members as president and vice-president. In how many ways can this be done?

Solution: Here, Subtask 1 is 'choosing a president'. This can be done in 35 ways. Subtask 2 is 'choosing a vice-president'. For each choice of president, we can choose the vice-president in 34 ways. Therefore, the total number of ways in which Subtasks 1 and 2 can be done is $35 \times 34 = 1190$.

Example 2: There are three political parties, P_1, P_2 and P_3. The party P_1 has 4 members, P_2 has 5 members and P_3 has 6 members in an assembly. Suppose we want to select two persons, both from the same party, to become president and vice-president. In how many ways can this be done?

Solution: From P_1, we can do the task in $4 \times 3 = 12$ ways, using the multiplication principle. From P_2, it can be done in $5 \times 4 = 20$ ways. From P_3 it can be done in $6 \times 5 = 30$ ways. So, by the addition principle, the number of ways of doing the task is $12 + 20 + 30 = 62$.

1.4 COMBINATIONS

Let's begin by considering a situation where we want to choose a committee of 3 faculty members from a group of seven faculty members. In how many distinct ways can this be done? Here order doesn't matter, because choosing F_1, F_2, F_3 is the same as choosing F_2, F_1, F_3, and so on. (Here F_i denotes the ith faculty member.) So, for every choice of members, to avoid repetition, we have to divide by 3!. Thus, the number would be $\frac{7\times 6\times 5}{3!}=\frac{7!}{3!4!}$.

More generally, suppose there are n distinct objects and we want to select r objects, where $r \le n$, where the order of **the objects in the selection does not matter**. This is called **a combination** of n things taken r at a time. The number of ways of doing this is represented by ${}_nC_r$, ${}^nC_r$, C^n_r, $\binom{n}{r}$ or C (n, r). We will use the notation C(n, r), in conformity with the notation P(n, r) for permutations. We read C(n, r) as 'n choose r' to emphasize the fact that only **choice** is involved but **not ordering**.

In the example that we started the section with, you saw that the number of combinations was 7!/3!4!, i.e., $\frac{P(7,3)}{3!}$. In fact, this relationship between C(n, r) and P(n, r) is true in general. We have the following result.

Theorem 3: The number of combinations of n objects, taken r at a time, where

Example 12: A box contains 3 red, 3 blue and 4 white socks. In how many ways can 8 socks be pulled out of the box, one at a time, if order is important?

Solution: Let us first see what happens if order isn't important. In this case we count the number of solutions of $r+b+w = 8$, $0 \le r, b \le 3$, $0 \le w \le 4$. To apply Theorem 5, we write $x = 3 - r$, $y = 3 - b$, $z = 4 - 10$.

Then we have $x+y+z = 10 - 8 = 2$, and the number of solutions this has is $C(3+2-1,2) = 6$.

These 6 solutions are (1, 0, 1) (0, 1, 1), (1, 1, 0), (2, 0, 0), (0, 2, 0), (0, 0, 2). So, the corresponding solutions for (r, b, w) are

(3, 3, 2), (2, 3, 3), (3, 2, 3), (3, 1, 4), (2, 2, 4), (1, 3, 4).

Now, we consider order. From Theorem 2 we know that the number of ways of pulling out 3 red, 3 blue and 2 white socks in some order is $\frac{8!}{3!3!2!}$. This number would be the same if you had 2 red, 3 blue and 3 white socks, etc. By this reasoning and considering all different orderings, the number of possibilities is

$$3\left(\frac{8!}{3!3!2!}\right)+2\left(\frac{8!}{3!1!4!}\right)+\frac{8!}{2!2!4!}=3220.$$

Example 14: A die is rolled once. What are the probabilities of the following events?

(i) getting an even number,

(ii) getting at least 2,

(iii) getting at most 2,

(iv) getting at least 10.

Solution: If we call the events A, B, C and D, then we have $X = \{1, 2, 3, 4, 5, 6\}$, $A = \{2,4,6\}$, $B = \{2,3,4,5,6\}$, $C = \{1,2\}$, and $D = \phi$.

Hence, $P(A) = 3/6$, $P(B) = 5/6$, $P(C) = 2/6$, $P(D) = 0$.

Example 15: A coin is tossed n times. What is the probability of getting exactly r heads?

Solution: If H and T represent head and tail, respectively, then X consists of sequences of length n that can be formed using only the letters H and T. Therefore, $n(X) = 2^n$. The event A consists of those sequences in which there are precisely r Hs. So, $n(A) = C(n, r)$. Hence, the required probability is $C(n, r)/2^n$.

* * *

Example 16: Two dice, one red and one white, are rolled. What is the probability that the white die turns up a smaller number than the red die?

Solution: If the number on the red die is x and that on the white die is y, then X consists of the 36 pairs (x, y), where x and y can be any integer from $\{1, 2, 3, 4, 5, 6\}$.

For the event A, we need $x < y$. For $x = 1, 2, 3, 4, 5$, y can be $x + 1, x+2, \ldots, 6$, i.e., $6 - x$ in number. Thus, by the addition principle,

$$n(A) = \sum_{x=1}^{5} (6 - x) = 5 + 4 + 3 + 2 + 1 = 15.$$

Hence, $P(A) = 15/36 = 5/12$.

Example 15: A coin is tossed n times. What is the probability of getting exactly r heads?

Solution: If H and T represent head and tail, respectively, then X consists of sequences of length n that can be formed using only the letters H and T. Therefore, $n(X) = 2^n$. The event A consists of those sequences in which there are precisely r Hs. So, $n(A) = C(n, r)$. Hence, the required probability is $C(n, r)/2^n$.

* * *

Example 16: Two dice, one red and one white, are rolled. What is the probability that the white die turns up a smaller number than the red die?

Solution: If the number on the red die is x and that on the white die is y, then X consists of the 36 pairs (x, y), where x and y can be any integer from {1, 2, 3, 4, 5, 6}.

For the event A, we need $x < y$. For $x = 1, 2, 3, 4, 5$, y can be $x + 1, x+2,\ldots, 6$, i.e., $6 - x$ in number. Thus, by the addition principle,

$$n(A) = \sum_{x=1}^{5} (6 - x) = 5 + 4 + 3 + 2 + 1 = 15.$$

Hence, $P(A) = 15/36 = 5/12$.

Example 18: Suppose A and B are mutually exclusive events such that $P(A) = 0.3$ and $P(B) = 0.4$. What is the probability that

i) A does not occur?

ii) A or B occurs?

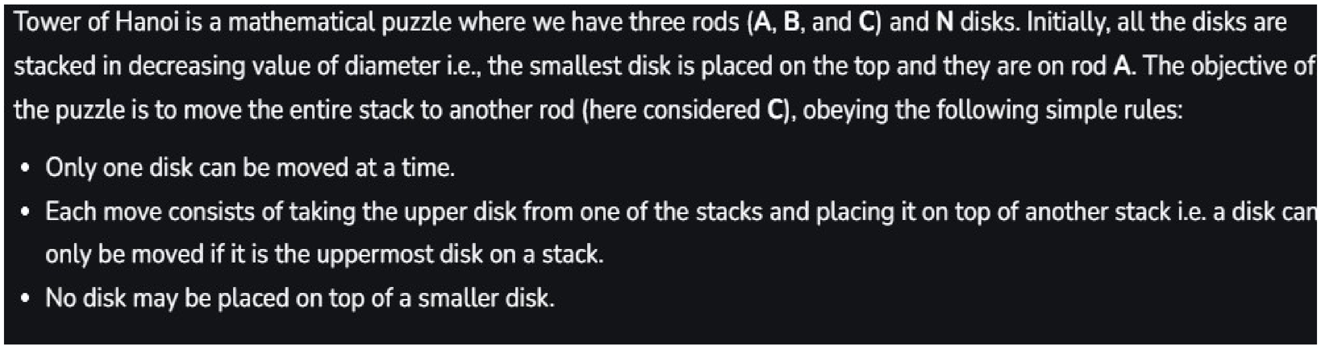
iii) Either A or B does not occur?

Solution:

i) This is $P(A^c) = 0.7$.

ii) This is $P(A \cup B) = 0.7$.

iii) This is $P(A^c \cup B^c) = P[(A \cap B)^c] = P(\phi^c) = P(X) = 1$


Tower of Hanoi is a mathematical puzzle where we have three rods (**A**, **B**, and **C**) and **N** disks. Initially, all the disks are stacked in decreasing value of diameter i.e., the smallest disk is placed on the top and they are on rod **A**. The objective of the puzzle is to move the entire stack to another rod (here considered **C**), obeying the following simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

ss

3.4 DIVIDE AND CONQUER TECHNIQUE TO SOLVE RECURRENCES

Often, to solve a problem, we can partition the problem into smaller problems, find solutions to the smaller problems and then combine the solutions to find a solution for the whole. We can repeatedly partition the problem till we reach a stage where we can solve the problem quite easily. This approach is called the **divide-and-conquer** approach. Let us start with an example. **Throughout this section we will assume that** $n = 2^k$.

Example 10: Consider the problem of finding the maximum and minimum of a list of n numbers. Here let us assume that $n = 2^k$ for some k to make things simpler for us. Let a_n be the number of comparisons required for finding the maximum and minimum of a list of n numbers. We can partition the list into two lists of size $\frac{n}{2}$ each. The maximum and minimum of each of the list can be found using $a_{\frac{n}{2}}$ comparisons. We can then compare minimum (resp. maximum) of the lists to get the minimum (resp. maximum) of the list, i.e., we will need two more comparisons. So, $a_n = 2a_{\frac{n}{2}} + 2$ for $n \geq 2$. $a_2 = 2$. We leave it as an exercise for you to check that $a_n = \frac{3}{2}n - 2$, when n is a power of 2.

Example 3: Calculate S_3^2 and S_4^2.

Solution: Using Theorem 4, we get $S_3^2 = S_2^1 + 2 \times S_2^2 = 1 + 2 \times 1 = 3$, and $S_4^2 = S_3^1 + 2 S_3^2 = 1 + 2 \times 3 = 7$.

Example 4: In how many ways can 20 students be grouped into 3 groups?

Solution: Theorem 6 says that this can be done in $S_{20}^1 + S_{20}^2 + S_{20}^3$ ways.

Now, using Theorem 3, we get this number to be

$$1 + \frac{1}{2}\sum_{k=0}^{2}(-1)^k C(2,2-k)(2-k)^{20} + \frac{1}{6}\sum_{k=0}^{3}(-1)^k C(3,3-k)(3-k)^{20}$$

$= 581,130,734.$

$$1^2 + 2^2 + 3^2 + \ldots + n^2 = \frac{n}{6}(n+1)(2n+1).$$

Since we want to prove it for every $n \in \mathbf{N}$, we take $m = 1$.

Step 1: $p(1)$ is $1^2 = \frac{1}{6}(1+1)(2+1)$, which is true

Step 2: Suppose for an arbitrary $k \in \mathbf{N}$, $p(k)$ is true, i.e.,

$$1^2 + 2^2 + \ldots + k^2 = \frac{k}{6}(k+1)(2k+1) \text{ is true.}$$

Step 3: To check if the assumption in step 2 implies that $p(k+1)$ is true. Let's see.

$$P(k+1) \text{ is } 1^2 + 2^2 + \ldots + k^2 + (k+1)^2 = \frac{k+1}{6}(k+2)(2k+3)$$

$$\Leftrightarrow (1^2 + 2^2 + \ldots + k^2) + (k+1)^2 \frac{k+1}{6}(k+2)(2k+3)$$

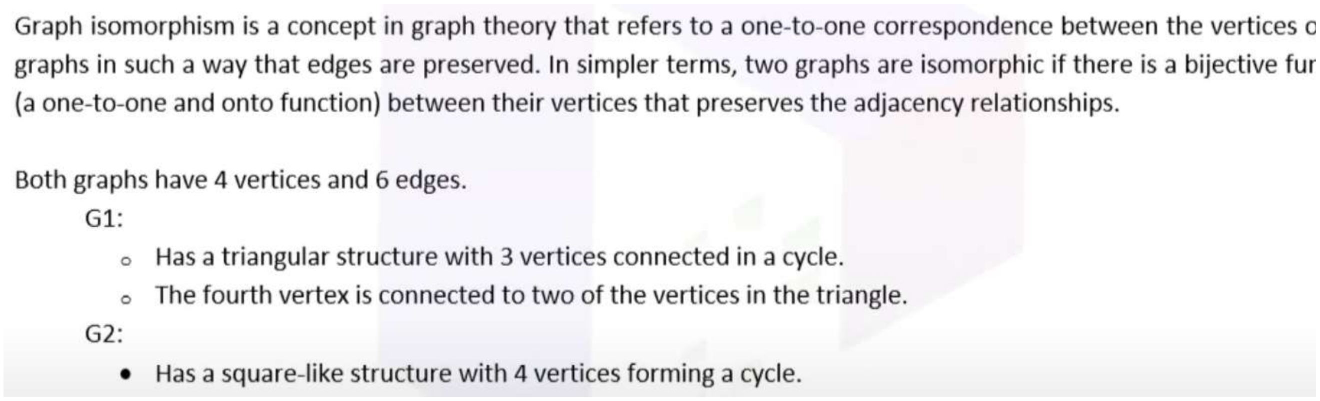
$$\Leftrightarrow \frac{k}{6}(k+1)(2k+1) + (k+1)^2 = \frac{k+1}{6}(k+2)(2k+3),$$

since $p(k)$ is true.

$$\Leftrightarrow \frac{k+1}{6}[k(2k+1) + 6(k+1)] = \frac{k+1}{6}(k+2)(2k+3)$$

$\Leftrightarrow 2k^2 + 7k + 6 = (k+2)(2k+3)$, dividing throughout by $\frac{k+1}{6}$, which is true.

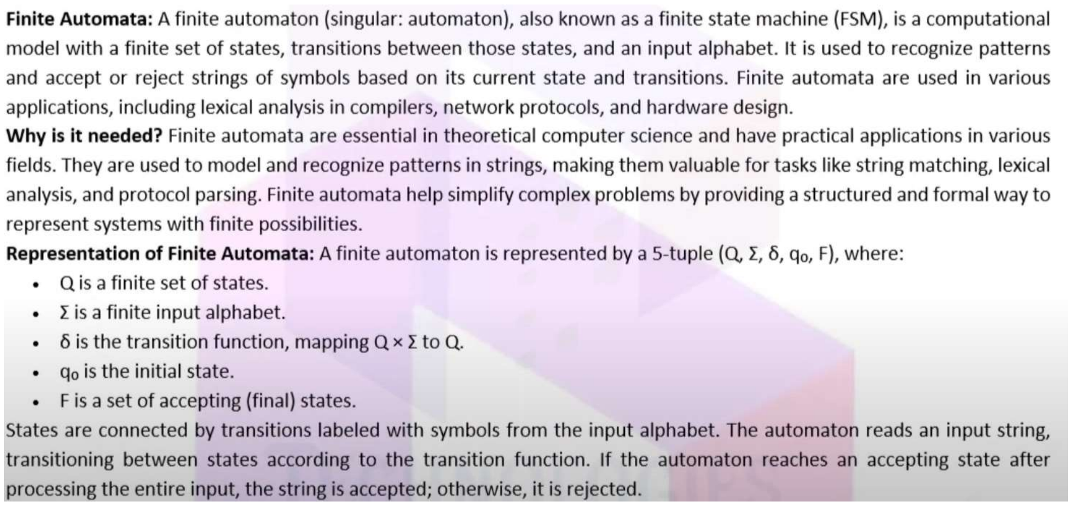
So, $p(k)$ is true implies that $p(k+1)$ is true.



Graph isomorphism is a concept in graph theory that refers to a one-to-one correspondence between the vertices o
graphs in such a way that edges are preserved. In simpler terms, two graphs are isomorphic if there is a bijective fur
(a one-to-one and onto function) between their vertices that preserves the adjacency relationships.

Both graphs have 4 vertices and 6 edges.

- G1:
  - Has a triangular structure with 3 vertices connected in a cycle.
  - The fourth vertex is connected to two of the vertices in the triangle.
- G2:
  - Has a square-like structure with 4 vertices forming a cycle.



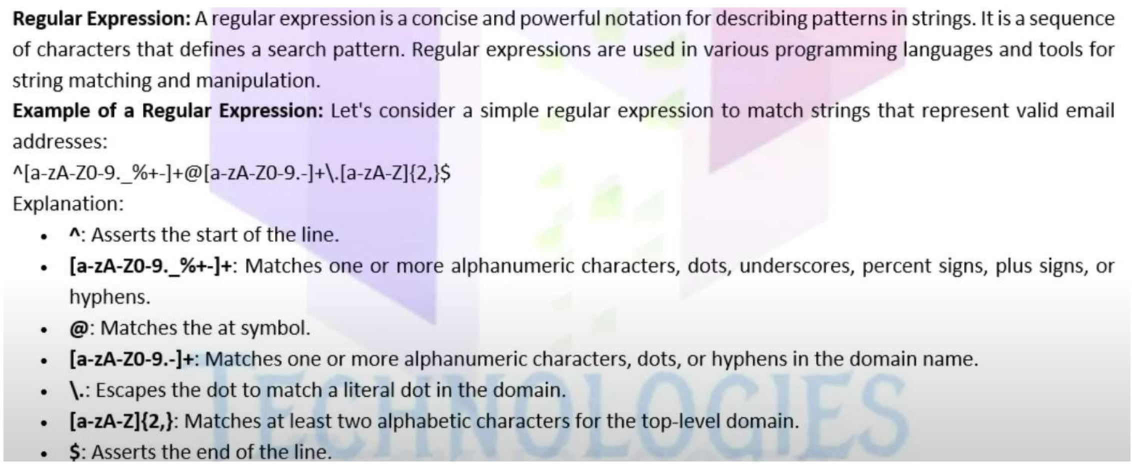
Finite Automata: A finite automaton (singular: automaton), also known as a finite state machine (FSM), is a computational model with a finite set of states, transitions between those states, and an input alphabet. It is used to recognize patterns and accept or reject strings of symbols based on its current state and transitions. Finite automata are used in various applications, including lexical analysis in compilers, network protocols, and hardware design.

Why is it needed? Finite automata are essential in theoretical computer science and have practical applications in various fields. They are used to model and recognize patterns in strings, making them valuable for tasks like string matching, lexical analysis, and protocol parsing. Finite automata help simplify complex problems by providing a structured and formal way to represent systems with finite possibilities.

Representation of Finite Automata: A finite automaton is represented by a 5-tuple ($Q, \Sigma, \delta, q_0, F$), where:

- Q is a finite set of states.
- Σ is a finite input alphabet.
- δ is the transition function, mapping $Q \times \Sigma$ to Q.
- q_0 is the initial state.
- F is a set of accepting (final) states.

States are connected by transitions labeled with symbols from the input alphabet. The automaton reads an input string, transitioning between states according to the transition function. If the automaton reaches an accepting state after processing the entire input, the string is accepted; otherwise, it is rejected.



Regular Expression: A regular expression is a concise and powerful notation for describing patterns in strings. It is a sequence of characters that defines a search pattern. Regular expressions are used in various programming languages and tools for string matching and manipulation.

Example of a Regular Expression: Let's consider a simple regular expression to match strings that represent valid email addresses:

```
^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$
```

Explanation:

- **^**: Asserts the start of the line.
- **[a-zA-Z0-9._%+-]+**: Matches one or more alphanumeric characters, dots, underscores, percent signs, plus signs, or hyphens.
- **@**: Matches the at symbol.
- **[a-zA-Z0-9.-]+**: Matches one or more alphanumeric characters, dots, or hyphens in the domain name.
- **\\.**: Escapes the dot to match a literal dot in the domain.
- **[a-zA-Z]{2,}**: Matches at least two alphabetic characters for the top-level domain.
- **$**: Asserts the end of the line.

Q10: Prove the following theorem by direct proof method: ''The square of an even integer is an even integer.''

Ans:-

To prove the theorem "The square of an even integer is an even integer" by direct proof, we need to start with the assumption that we have an even integer, and then show that the square of this even integer is also an even integer.

Theorem: For any even integer n, n^2 is also an even integer.

Direct Proof: Let n be an even integer. By definition, an even integer can be written as n=2k for some integer k.

Now, we want to show that n^2 is also an even integer.

$n^2=(2k)^2=4k^2$

Here, $4k^2$ is a multiple of 4, and any multiple of 4 is an even integer.

Therefore, n^2 is an even integer.

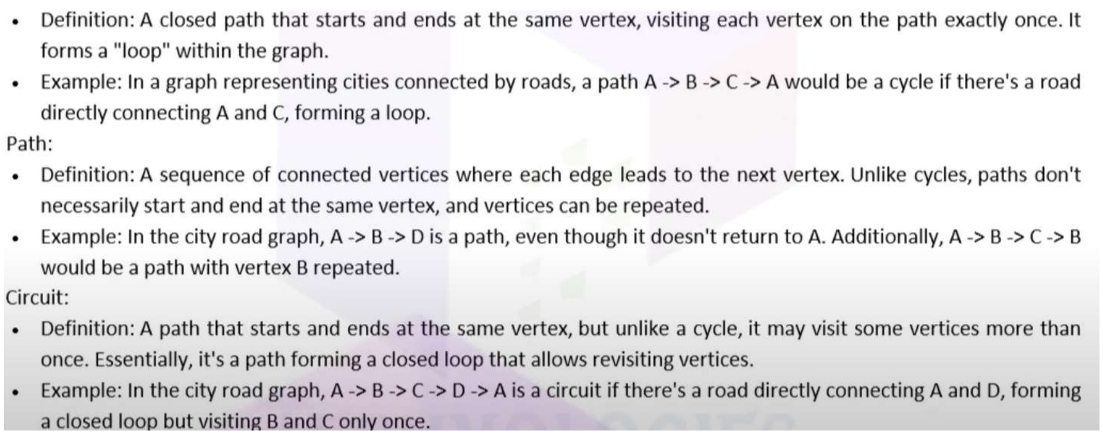
1. **Domain:** The domain of a function is the set of all possible input values (independent variable) for which the function is defined. It is the set of values that can be fed into the function to obtain corresponding output values.
2. **Co-domain:** The co-domain of a function is the set of all possible output values (dependent variable) that the function can produce. It is a larger set than the range, as it includes all possible output values, whether or not they are actually reached.
3. **Range:** The range of a function is the set of all actual output values that the function can produce for the given input values. It is a subset of the co-domain and represents the values that the function "hits" or reaches.

a. Complete Graph:

- Definition: A graph where every pair of vertices is connected by an edge. There are no "isolated" vertices without connections.
- Example: Consider a social network of 4 friends, A, B, C, and D. If all four friends are connected directly, forming a network where everyone knows everyone else, that would be a complete graph K4.

b. Degree of a Vertex:

- Definition: The number of edges connected to a particular vertex. It represents the vertex's "connectivity" within the graph.
- Example: In the K4 example above, each vertex (A, B, C, D) has a degree of 3, as each is connected to the other three friends.



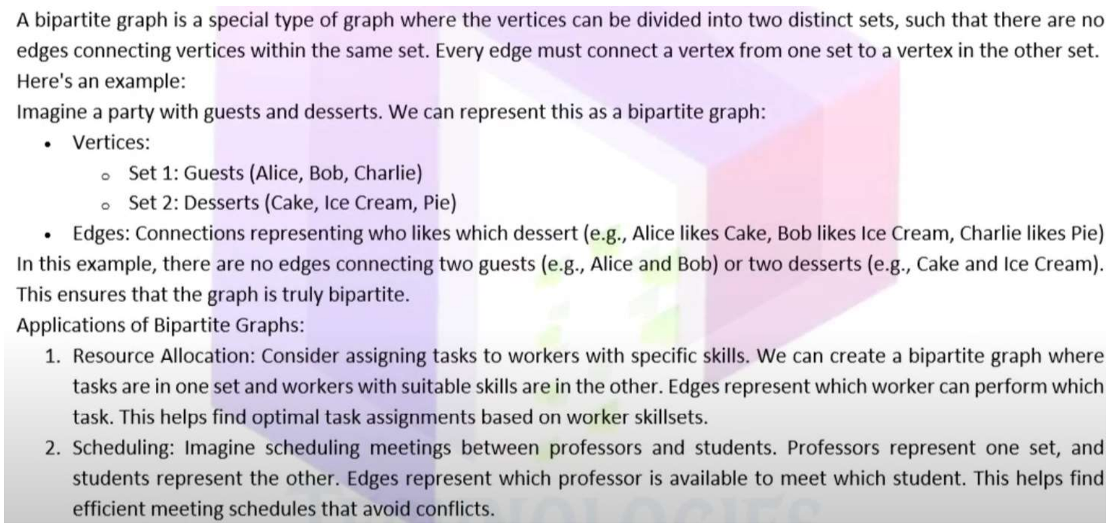
- Definition: A closed path that starts and ends at the same vertex, visiting each vertex on the path exactly once. It forms a "loop" within the graph.
- Example: In a graph representing cities connected by roads, a path A -> B -> C -> A would be a cycle if there's a road directly connecting A and C, forming a loop.

Path:

- Definition: A sequence of connected vertices where each edge leads to the next vertex. Unlike cycles, paths don't necessarily start and end at the same vertex, and vertices can be repeated.
- Example: In the city road graph, A -> B -> D is a path, even though it doesn't return to A. Additionally, A -> B -> C -> B would be a path with vertex B repeated.

Circuit:

- Definition: A path that starts and ends at the same vertex, but unlike a cycle, it may visit some vertices more than once. Essentially, it's a path forming a closed loop that allows revisiting vertices.
- Example: In the city road graph, A -> B -> C -> D -> A is a circuit if there's a road directly connecting A and D, forming a closed loop but visiting B and C only once.

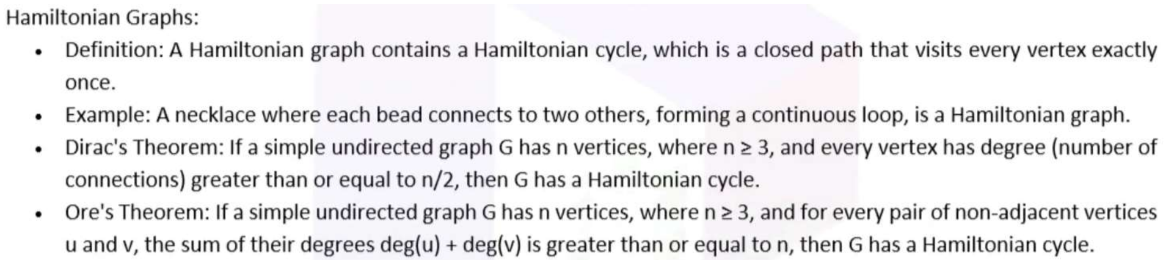


A bipartite graph is a special type of graph where the vertices can be divided into two distinct sets, such that there are no edges connecting vertices within the same set. Every edge must connect a vertex from one set to a vertex in the other set.
Here's an example:
Imagine a party with guests and desserts. We can represent this as a bipartite graph:

- Vertices:
  - Set 1: Guests (Alice, Bob, Charlie)
  - Set 2: Desserts (Cake, Ice Cream, Pie)
- Edges: Connections representing who likes which dessert (e.g., Alice likes Cake, Bob likes Ice Cream, Charlie likes Pie)

In this example, there are no edges connecting two guests (e.g., Alice and Bob) or two desserts (e.g., Cake and Ice Cream). This ensures that the graph is truly bipartite.
Applications of Bipartite Graphs:

1. Resource Allocation: Consider assigning tasks to workers with specific skills. We can create a bipartite graph where tasks are in one set and workers with suitable skills are in the other. Edges represent which worker can perform which task. This helps find optimal task assignments based on worker skillsets.
2. Scheduling: Imagine scheduling meetings between professors and students. Professors represent one set, and students represent the other. Edges represent which professor is available to meet which student. This helps find efficient meeting schedules that avoid conflicts.



Hamiltonian Graphs:

- Definition: A Hamiltonian graph contains a Hamiltonian cycle, which is a closed path that visits every vertex exactly once.
- Example: A necklace where each bead connects to two others, forming a continuous loop, is a Hamiltonian graph.
- Dirac's Theorem: If a simple undirected graph G has n vertices, where $n \geq 3$, and every vertex has degree (number of connections) greater than or equal to n/2, then G has a Hamiltonian cycle.
- Ore's Theorem: If a simple undirected graph G has n vertices, where $n \geq 3$, and for every pair of non-adjacent vertices u and v, the sum of their degrees deg(u) + deg(v) is greater than or equal to n, then G has a Hamiltonian cycle.

- Definition: An Eulerian graph contains an Eulerian path, which is a path that visits every edge exactly once. It may or may not return to the starting vertex. If it does, it's an Eulerian cycle.
- Example: A map where you can trace a route across every bridge or road exactly once is an Eulerian graph.
- Euler's Theorem: An undirected graph has an Eulerian cycle if and only if all vertices have even degree.

ey Differences:

- Focus: Hamiltonian graphs focus on visiting all vertices, while Eulerian graphs focus on visiting all edges.
- Cycles vs. Paths: Hamiltonian paths must be cycles (closed loops), while Eulerian paths can be open or closed.
- Degree conditions: Different degree conditions determine the existence of each type of path.

b. Vertex Coloring: Vertex coloring is a concept in graph theory where the vertices of a graph are assigned colors in such a way that no two adjacent vertices have the same color. The minimum number of colors needed for a proper vertex coloring is known as the chromatic number of the graph. Vertex coloring finds applications in scheduling, map coloring, and resource allocation problems.

c. Edge Coloring: Edge coloring is another concept in graph theory where the edges of a graph are assigned colors in such a way that no two incident edges have the same color. The minimum number of colors needed for a proper edge coloring is known as the chromatic index of the graph. Edge coloring is used in scheduling problems, network design, and various real-world scenarios.

d. Planar Graphs: A graph is called planar if it can be drawn on a plane without any edges crossing each other. The study of planar graphs is crucial in topology and graph theory. Euler's formula for planar graphs states that for any connected planar graph with V vertices, E edges, and F faces (including the exterior), the equation $V-E+F=2$ holds. Planar graphs have applications in circuit design, map representation, and network planning.

e. Pascal's Formula: Pascal's formula refers to various mathematical concepts, but one common interpretation is related to binomial coefficients. Pascal's formula states that the binomial coefficient (kn) (read as "n choose k") can be computed using the formula $(kn)=(k-1n-1)+(kn-1)$. This recursive formula represents the Pascal's Triangle, a triangular array of numbers where each number is the sum of the two directly above it. Pascal's formula is fundamental in combinatorics and has applications in probability, algebra, and calculus.

De Morgan's laws are a set of two logical rules that relate the negation of a conjunction (AND) or disjunction (OR) of two statements. These laws are as follows:

1.The negation of a conjunction is equivalent to the disjunction of the negations of the statements. In other words:

$\sim(p \land q) \Leftrightarrow \sim p \lor \sim q$

2.The negation of a disjunction is equivalent to the conjunction of the negations of the statements. In other words:

$\sim(p \lor q) \Leftrightarrow \sim p \land \sim q$

Demorgan's laws are useful in simplifying complex logical expressions by breaking them down into simpler parts. For example, consider the expression $\sim(p \land q)$. Applying the first Demorgan's law, we can rewrite this as $\sim p \lor \sim q$. Similarly, if we have $\sim(p \lor q)$, we can rewrite this as $\sim p \land \sim q$ using the second Demorgan's law.

An equivalence relation is a binary relation that is reflexive, symmetric, and transitive. In other words, it is a relation on a set that satisfies the following three properties:

1.Reflexivity: for any element a in the set, a is related to itself.

2.Symmetry: if a is related to b, then b is related to a.

3.Transitivity: if a is related to b and b is related to c, then a is related to c.

Equivalence relations are used to partition a set into subsets of elements that are related to each other. The equivalence classes are formed by grouping together all elements that are related to each other by the equivalence relation. Equivalence relations are commonly used in various fields of mathematics, including algebra, topology, and number theory.

The chromatic number of a graph is the smallest number of colors needed to color the vertices of the graph in such a way that no adjacent vertices have the same color. In other words, it is the minimum number of colors required to color all the vertices of the graph in such a way that no two adjacent vertices have the same color.

Formally, let G be a graph. A coloring of the vertices of G is an assignment of a color to each vertex such that no two adjacent vertices have the same color. The chromatic number of G, denoted by $\chi(G)$, is the smallest number of colors required for a proper coloring of G.

The concept of chromatic number is important in graph theory, as it provides a measure of the "colorability" of a graph. The chromatic number of a graph is related to many other properties of the graph, such as its planarity, its maximum degree, and its clique number. The chromatic number is also used in many practical applications, such as scheduling

g) Show that C6 is bipartite and K3 is not bipartite.

To show that C6 (cycle of length 6) is bipartite, we can divide its vertices into two sets, say A and B, as follows:

A = {1, 3, 5} B = {2, 4, 6}

Here, all the vertices in set A are connected only to vertices in set B, and vice versa. Thus, C6 is bipartite.