UNIT – I

LOGIC

LOGIC

TF Statement

A proposition is a collection of declarative statements that has either a truth value "true" or a truth value "false". A propositional consists of propositional variables and connectives. We denote the propositional variables by capital letters (A, B, etc). The connectives connect the propositional variables.

Eg

- "Man is Mortal", it returns truth value "TRUE"
- "12 + 9 = 3 – 2", it returns truth value "FALSE"

The following is not a Proposition −

- "A is less than 2". It is because unless we give a specific value of A, we cannot say whether the statement is true or false.

Connectives

A Logical Connective is a symbol which is used to connect two or more propositional or predicate logics in such a manner that resultant logic depends only on the input logics and the meaning of the connective used.

Generally there are five connectives which are ,

- OR (∨)
- AND (∧)
- Negation/ NOT (¬)
- Implication / if-then (→)
- If and only if (⇔).

OR (∨)

The OR operation of two propositions A and B (written as A ∨ B) is true if at least any of the propositional variable A or B is true.

The truth table is as follows −

A	B	A ∨ B
True	True	True
True	False	True
False	True	True
False	False	False

AND (∧)

The AND operation of two propositions A and B (written as $A \land B$) is true if both the propositional variable A and B is true.

The truth table is as follows −

A	B	A ∧ B
True	True	True
True	False	False
False	True	False
False	False	False

Negation (¬)

The negation of a proposition A (written as ¬ A) is false when A is true and is true when A is false.

The truth table is as follows −

A	¬ A

A	¬ A
True	False
False	True

<u>Implication / if-then (→)</u>

An implication A → B is the proposition “if A, then B”. It is false if A is true and B is false. The rest cases are true.

The truth table is as follows −

A	B	A → B
True	True	True
True	False	False
False	True	True
False	False	True

<u>If and only if (⇔)</u>

A ⇔ B is bi-conditional logical connective which is true when p and q are same, i.e. both are false or both are true.

The truth table is as follows −

A	B	A ⇔ B
True	True	True
True	False	False
False	True	False
False	False	True

Atomic and Compound Statements

An atomic sentence is an atomic formula containing no variables. It follows that an atomic sentence contains no logical connectives, variables or quantifiers. A sentence consisting of one or more sentences and a logical connective is a compound (or molecular) sentence.

Eg

It is raining - simple statement

Jack **and** jill went up the hill - compound statement

Well-formed Formula (wff)

Not all strings can represent propositions of predicate logic. Those that produce a proposition when their symbols are interpreted are called well-formed formulas of the first order predicate logic. A predicate name followed by a list of variables such as P(x, y), where P is a predicate name, and x and y are variables, is called an atomic formula.

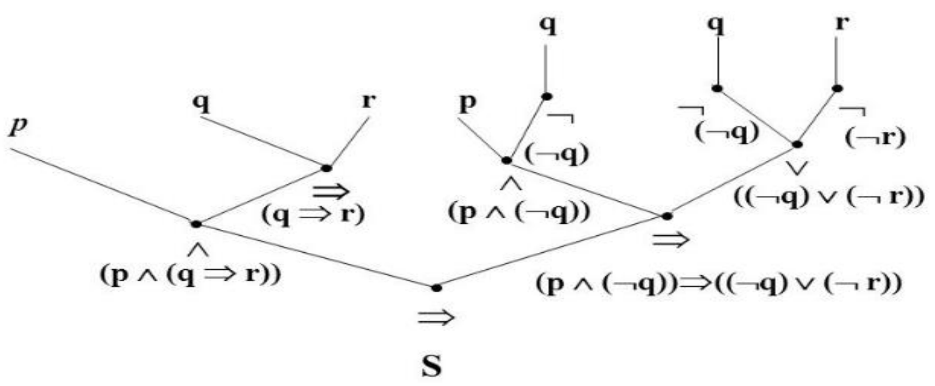
Wffs are constructed using the following rules:

1. True and False are wffs.
2. Each propositional constant (i.e. specific proposition).
3. Each atomic formula (i.e. a specific predicate with variables) is a wff.
4. If A and B are wffs, then so are $\neg A$, $(A \vee B)$, $(A \wedge B)$, $(A \rightarrow B)$, and $(A \leftrightarrow B)$.

Parsing Tree

Every wff we can associate a tree called a parsing tree.

Eg



The Truth Table of a Formula

A **truth table** shows how the truth or falsity of a compound statement depends on the truth or falsity of the simple statements from which it's constructed.

Eg

Construct a truth table for the formula $\neg P \wedge (P \rightarrow Q)$.

P	Q	$\neg P$	$P \rightarrow Q$	$\neg P \wedge (P \rightarrow Q)$
T	T	F	T	F
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Tautology

A **tautology** is a formula which is "always true" --- that is, it is true for every assignment of truth values to its simple components. You can think of a tautology as a *rule of logic*.

The opposite of a tautology is a **contradiction**, a formula which is "always false". In other words, a contradiction is false for every assignment of truth values to its simple components.

Eg

Show that $(P \rightarrow Q) \vee (Q \rightarrow P)$ is a tautology.

I construct the truth table for $(P \rightarrow Q) \vee (Q \rightarrow P)$ and show that the formula is always true.

P	Q	$P \rightarrow Q$	$Q \rightarrow P$	$(P \rightarrow Q) \vee (Q \rightarrow P)$
T	T	T	T	T
T	F	F	T	T
F	T	T	F	T
F	F	T	T	T

The last column contains only T's. Therefore, the formula is a tautology.

Tautological Implications and Equivalence of Formulae

Tautologies by adding ones involving the conditional and the biconditional. From now on, we use small letters like p and q to denote atomic statements only, and uppercase letters like A and B to denote statements of all types, compound or atomic.

We first look at some tautological implications, tautologies of the form A→B. You should check the truth table of each of the statements we give to see that they are, indeed, tautologies.

Eg

p	**q**	**p→q**	**(p→q) ∧p**	**[(p→q) ∧p] →q**
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

REVIEW QUESTIONS

1) Define Well Formed formula.
2) What is the truth table for conditional statement?
3) Discuss about tautological implication.
4) Explain about Connectives.
5) Show that (P→Q)∧(R→Q)and (P∨R)→Q are equivalent.
6) Prove whether the following formula (P∧(P↔Q))→Q) is a tautology or not.
7) What is meant by atomic and compound statements.
8) Define tautology.

UNIT – II

NORMAL FORMS

- ➢ **Principal Normal Forms**
- ➢ **Theory of Inference**
- ➢ **Open Statements**
- ➢ **Quantifiers**
- ➢ **Valid Formulae and Equivalence**
- ➢ **Theory of Inference for Predicate Calculus**

NORMAL FORMS

Principal Normal Form

Disjunctive Normal Forms (DNF)

A formula which is equivalent to a given formula and which consists of a sum of elementary products is called a disjunctive normal form of given formula.

Eg

$(P \wedge \sim Q) \vee (Q \wedge R) \vee (\sim P \wedge Q \wedge \sim R)$

- The DNF of formula is not unique.

Conjunctive Normal Form (CNF)

A formula which is equivalent to a given formula and which consists of a product of elementary products is called a conjunctive normal form of given formula.

Eg

$(P\sim \vee Q) \wedge (Q \vee R) \wedge (\sim P \vee Q \vee \sim R)$

- The CNF of formula is not unique.
- If every elementary sum in CNF is tautology, then given formula is also tautology.

Principle Disjunctive Normal Form (PDNF)

An equivalent formula consisting of disjunctions of minterms only is called the principle disjunctive normal form of the formula.

It is also known as **sum-of-products canonical form**.

Eg

$(P \wedge \sim Q \wedge \sim R) \vee (P \wedge \sim Q \wedge R) \vee (\sim P \wedge \sim Q \wedge \sim R)$

- The minterm consists of conjunctions in which each statement variable or its negation, but not both, appears only once.
- The minterms are written down by including the variable if its truth value is T and its negation if its truth value is F.

Principle Conjunctive Normal Form (PCNF)

An equivalent formula consisting of conjunctions of maxterms only is called the principle conjunctive normal form of the formula.

It is also known as **product-of-sums canonical form**.

Eg

$(P \vee \sim Q \vee \sim R) \wedge (P \vee \sim Q \vee R) \wedge (\sim P \vee \sim Q \vee \sim R)$

- The maxterm consists of disjunctions in which each variable or its negation, but not both, appears only once.
- The dual of a minterm is called a maxterm.
- Each of the maxterm has the truth value F for exactly one combination of the truth values of the variables.
- The maxterms are written down by including the variable if its truth value is F and its negation if its truth value is T.

Eg

Obtain the PDNF of $(\rceil P \vee \rceil Q) \rightarrow (P \leftrightarrow \rceil Q)$

P	**Q**	**$\rceil P \vee \rceil Q$**	**P $\rceil Q$**	**$(\rceil P \vee \rceil Q) \rightarrow (P \leftrightarrow \rceil Q)$**
T	T	F	F	T
T	F	T	T	T
F	T	T	T	T
F	F	T	F	F

From the above table

$(\rceil P \vee \rceil Q) \rightarrow (P \leftrightarrow \rceil Q) \Leftrightarrow (P \wedge Q) \vee (P \wedge \rceil Q) \vee (\rceil P \wedge Q)$
$\Leftrightarrow (\rceil P \wedge Q) \vee (P \wedge \rceil Q) \vee (P \wedge Q)$

Eg

Obtain PDNF for $P \rightarrow ((P \rightarrow Q \wedge \rceil(\rceil Q \vee \rceil P)))$.

Solution

$P \rightarrow ((P \rightarrow Q \wedge \rceil(\rceil Q \vee \rceil P))) \Leftrightarrow P \rightarrow ((P \rightarrow Q \wedge (P \wedge Q)))$
$\Leftrightarrow P \rightarrow ((P \rightarrow P \wedge Q))$
$\Leftrightarrow P \rightarrow (\rceil P \vee (P \wedge Q))$
$\Leftrightarrow \rceil P \vee (\rceil P \vee (P \wedge Q))$
$\Leftrightarrow \rceil P \vee (P \wedge Q)$
$\Leftrightarrow (\rceil P \wedge (Q \vee \rceil Q)) \vee (P \wedge Q)$
$\Leftrightarrow (\rceil P \wedge Q) \vee (\rceil P \wedge \rceil Q) \vee (P \wedge Q)$
$\Leftrightarrow (\rceil P \wedge \rceil Q) \vee (\rceil P \wedge Q) \vee (P \wedge Q)$

Eg

Obtain PCNF for A : $(\neg P \to R) \wedge ((Q \to P) \wedge (P \to Q))$.

Solution

$A \Leftrightarrow (P \vee R) \wedge ((\neg Q \vee P) \wedge (\neg P \vee Q))$
$\Leftrightarrow (P \vee R \vee (Q \wedge \neg Q)) \wedge (P \vee \neg Q \vee (R \wedge \neg R)) \wedge (\neg P \vee Q \vee (R \wedge \neg R))$
$\Leftrightarrow (P \vee Q \vee R) \wedge (P \vee \neg Q \vee R) \wedge (P \vee \neg Q \vee R) \wedge (P \vee \neg Q \vee \neg R) \wedge (\neg P \vee Q \vee R) \wedge (\neg P \vee Q \vee \neg R)$
$\Leftrightarrow (P \vee Q \vee R) \wedge (P \vee \neg Q \vee R) \wedge (P \vee \neg Q \vee \neg R) \wedge (\neg P \vee Q \vee R) \wedge (\neg P \vee Q \vee \neg R)$

Theory of Inference

A **proof** is an argument from **hypotheses** (assumptions) to a **conclusion**. Each step of the argument follows the laws of logic. In mathematics, a statement is not accepted as valid or correct unless it is accompanied by a proof. This insistence on proof is one of the things that sets mathematics apart from other subjects.

Writing proofs is difficult; there are no procedures which you can follow which will guarantee success. *The patterns which proofs follow are complicated, and there are a lot of them.* You can't expect to do proofs by following rules, memorizing formulas, or looking at a few examples in a book.

For this reason, I'll start by discussing **logic proofs**. Since they are more highly patterned than most proofs, they are a good place to start. They'll be written in **column format**, with each step justified by a **rule of inference**. Most of the rules of inference will come from tautologies. Since a tautology is a statement which is "always true", it makes sense to use them in drawing conclusions.

Like most proofs, logic proofs usually begin with **premises** --- statements that you're allowed to assume. The **conclusion** is the statement that you need to prove. The idea is to operate on the premises using rules of inference until you arrive at the conclusion.

Rule of Premises. You may write down a **premise** at any point in a proof.

The second rule of inference is one that you'll use in most logic proofs. It is sometimes called **modus ponendo ponens**, but I'll use a shorter name.

Modus Ponens. If you know P and $P \to Q$, you may write down Q.

In the rules of inference, it's understood that symbols like "P" and "Q" may be replaced by *any* statements, including compound statements. I'll say more about this later.

Here is a simple proof using modus ponens:

1.	P	Premise
2.	$P \to Q$	Premise
3.	Q	Modus ponens (1, 2)

I'll write logic proofs in 3 columns. The statements in logic proofs are numbered so that you can refer to them, and the numbers go in the first column. The actual statements go in the second column. The third column contains your justification for writing down the statement.

Thus, statements 1 (P) and 2 ($P \to Q$) are premises, so the rule of premises allows me to write them down. Modus ponens says that if I've already written down P and $P \to Q$ --- on *any* earlier lines, in either order --- then I may write down Q. I did that in line 3, citing the rule ("Modus ponens") and the lines (1 and 2) which contained the statements I needed to apply modus ponens.

As I noted, the "P" and "Q" in the modus ponens rule can actually stand for compound statements --- they don't have to be "single letters". For example:

1.	$(A \vee \neg B) \to \neg C$	Premise
2.	$\neg D$	Premise
3.	$A \vee \neg B$	Premise
4.	$\neg C$	Modus ponens (1, 3)

There are several things to notice here. First, $A \vee \neg B$ is taking the place of P in the modus ponens rule, and $\neg C$ is taking the place of Q. That is, $A \vee \neg B$ and $\neg C$ are *compound statements* which are substituted for "P" and "Q" in modus ponens.

Notice also that the if-then statement $(A \vee \neg B) \to \neg C$ is listed first and the "if"-part $\neg C$ is listed second. It doesn't matter which one has been written down first, and long as both pieces have already been written down, you may apply modus ponens.

Finally, the statement $\neg D$ didn't take part in the modus ponens step. Perhaps this is part of a bigger proof, and $\neg D$ will be used later. The fact that it came between the two modus ponens pieces doesn't make a difference.

As usual in math, you have to be sure to apply rules *exactly*. For example, this is *not* a valid use of modus ponens:

1.	P	Premise
2.	$(P \wedge Q) \to R$	Premise
3.	R	INVALID modus ponens!

Do you see why? To use modus ponens on the if-then statement $(P \wedge Q) \to R$, you need the "if"-part, which is $P \wedge Q$. You only have P, which is just *part of* the "if"-part. That's not good enough.

Double Negation. In any statement, you may substitute P for $\neg\neg P$ or $\neg\neg P$ for P (and write down the new statement).

For example, in this case I'm applying double negation with P replaced by $A \to \neg C$:

1.	$\neg[\neg(A \to \neg C)]$	Premise
2.	$A \to \neg C$	Double negation (1)

You can also apply double negation "inside" another statement:

1.	$A \to \neg\neg B$	Premise
2.	$A \to B$	Double negation (1)

Double negation comes up often enough that, we'll bend the rules and allow it to be used without doing so as a separate step or mentioning it explicitly. I'll demonstrate this in the examples for some of the other rules of inference.

Modus Tollens. If you know $\neg Q$ and $P \to Q$, you may write down $\neg P$.

This is a simple example of modus tollens:

1.	$\neg Q$	Premise
2.	$P \to Q$	Premise
3.	$\neg P$	Modus tollens (1, 2)

In the next example, I'm applying modus tollens with P replaced by C and Q replaced by $A \to B$:

1.	$\neg(A \to B)$	Premise
2.	$C \to (A \to B)$	Premise
3.	$\neg C$	Modus tollens (1, 2)

The last example shows how you're allowed to "suppress" double negation steps. Do you see how this was done? If I wrote the double negation step explicitly, it would look like this:

1.	$\neg(A \to B)$	Premise
2.	$C \to (A \to B)$	Premise
3.	$C \to \neg\neg(A \to B)$	Double negation (2)
4.	$\neg C$	Modus tollens (1, 3)

When you apply modus tollens to an if-then statement, be sure that you have the *negation* of the "then"-part. The following derivation is incorrect:

1.	Q	Premise
2.	$P \to Q$	Premise
3.	$\neg P$	INVALID modus tollens!

To use modus tollens, you need $\neg Q$, not Q.

This is also incorrect:

1.	Q	Premise
2.	$P \to Q$	Premise
3.	P	INVALID modus tollens!

This looks like modus ponens, but backwards. There is no rule that allows you to do this: The deduction is invalid.

Disjunctive Syllogism. If you know $\neg P$ and $P \vee Q$, you may write down Q.

Here's a simple example of disjunctive syllogism:

1.	$\neg P$	Premise
2.	$P \vee Q$	Premise
3.	Q	Disjunctive syllogism (1, 2)

In the next example, I'm applying disjunctive syllogism with $A \vee B$ replacing P and D replacing Q in the rule:

1.	$\neg(A \vee B)$	Premise
2.	$(A \vee B) \vee D$	Premise
3.	D	Disjunctive syllogism (1, 2)

In the next example, notice that P is the same as $\neg\neg P$, so it's the negation of $\neg P$.

1.	P	Premise
2.	$\neg P \vee (Q \to R)$	Premise
3.	$Q \to R$	Disjunctive syllogism (1, 2)

This is another case where I'm skipping a double negation step. Without skipping the step, the proof would look like this:

1.	P	Premise
2.	$\neg P \vee (Q \to R)$	Premise
3.	$\neg\neg P$	Double negation (1)
4.	$Q \to R$	Disjunctive syllogism (2, 3)

DeMorgan's Law. In any statement, you may substitute:

1. $\neg(P \vee Q)$ for $\neg P \wedge \neg Q$.

2. $\neg P \wedge \neg Q$ for $\neg(P \vee Q)$.

3. $\neg(P \wedge Q)$ for $\neg P \vee \neg Q$.

4. $\neg P \vee \neg Q$ for $\neg(P \wedge Q)$.

As usual, after you've substituted, you write down the new statement.

DeMorgan's Law tells you how to distribute $\neg$ across $\wedge$ or $\vee$, or how to factor $\neg$ out of $\wedge$ or $\vee$. To distribute, you attach $\neg$ to each term, then change $\wedge$ to $\vee$ or $\vee$ to $\wedge$. To factor, you factor $\neg$ out of each term, then change $\wedge$ to $\vee$ or $\vee$ to $\wedge$.

Note that it only applies (directly) to "or" and "and". We'll see how to negate an "if-then" later.

Here's DeMorgan applied to an "or" statement:

1.	$\neg(\neg P \vee \neg Q)$	Premise
2.	$P \wedge Q$	DeMorgan (1)

Notice that a literal application of DeMorgan would have given $\neg\neg P \wedge \neg\neg Q$. I changed this to $P \wedge Q$, once again suppressing the double negation step.

Conditional Disjunction. If you know $P \to Q$, you may write down $\neg P \vee Q$.

If you know $\neg P \vee Q$, you may write down $P \to Q$.

Here's the first direction:

1.	$\neg P \vee Q$	Premise
2.	$P \to Q$	Conditional disjunction (1)

And here's the second:

1.	$P \to Q$	Premise
2.	$\neg P \vee Q$	Conditional disjunction (1)

The first direction is key: Conditional disjunction allows you to convert "if-then" statements into "or" statements.

We'll see below that biconditional statements can be converted into pairs of conditional statements. Together with conditional disjunction, this allows us in principle to reduce the five logical connectives to three (negation, conjunction, disjunction). But DeMorgan allows us to change conjunctions to disjunctions (or vice versa), so in principle we could do everything with just "or" and "not". The reason we don't is that it would make our statements much longer: The use of the other connectives is like shorthand that saves us writing.

In additional, we can solve the problem of negating a conditional that we mentioned earlier.

1.	$\neg(P \to Q)$	Premise
2.	$\neg(\neg P \vee Q)$	Conditional disjunction (1)
3.	$P \wedge \neg Q$	DeMorgan (2)

We've derived a new rule! Let's write it down.

Negating a Conditional. If you know $\neg(P \to Q)$, you may write down $P \wedge \neg Q$.

If you know $P \wedge \neg Q$, you may write down $\neg(P \to Q)$.

The first direction is more useful than the second. Personally, I tend to forget this rule and just apply conditional disjunction and DeMorgan when I need to negate a conditional. But you may use this if you wish.

Constructing a Conjunction. If you know P and Q, you may write down $P \wedge Q$.

Think about this to ensure that it makes sense to you. If $P \wedge Q$ is true, you're saying that P is true and that Q is true. So on the other hand, you need both P true and Q true in order to say that $P \wedge Q$ is true.

Here's an example. Notice that I put the pieces in parentheses to group them after constructing the conjunction.

1.	$P \leftrightarrow Q$	Premise
2.	$P \vee Q$	Premise
3.	$(P \leftrightarrow Q) \wedge (P \vee Q)$	Constructing a conjunction (1, 2)

Rule of Syllogism. If you know $P \rightarrow Q$ and $Q \rightarrow R$, then you may write down $P \rightarrow R$.

The Rule of Syllogism says that you can "chain" syllogisms together. For example:

1.	$P \rightarrow (Q \vee R)$	Premise
2.	$(Q \vee R) \rightarrow \neg S$	Premise
3.	$P \rightarrow \neg S$	Rule of syllogism (1, 2)

Definition of Biconditional. If you know $P \leftrightarrow Q$, you may write down $P \rightarrow Q$ and you may write down $Q \rightarrow P$. If you know $P \rightarrow Q$ and $Q \rightarrow P$, you may write down $P \leftrightarrow Q$.

First, a simple example:

1.	$P \leftrightarrow Q$	Premise
2.	$Q \rightarrow P$	Definition of biconditional (1)

By the way, a standard mistake is to apply modus ponens to a biconditional ("$\leftrightarrow$"). Modus ponens applies to conditionals ("$\rightarrow$"). So this isn't valid:

1.	$P \leftrightarrow Q$	Premise
2.	Q	Premise
3.	P	INVALID - Not modus ponens!

With the same premises, here's what you need to do:

1.	$P \leftrightarrow Q$	Premise
2.	Q	Premise
3.	$Q \rightarrow P$	Definition of biconditional (1)
4.	P	Modus ponens (2, 3) □

Decomposing a Conjunction. If you know $P \wedge Q$, you may write down P and you may write down Q.

This rule says that you can decompose a conjunction to get the individual pieces:

1.	$P \wedge (Q \leftrightarrow \neg R)$	Premise
2.	P	Decomposing a conjunction (1)
2.	$Q \leftrightarrow \neg R$	Decomposing a conjunction (1)

Note that you *can't* decompose a disjunction!

1.	$P \vee Q$	Premise
2.	P	INVALID!

What's wrong with this? If you know that $P \vee Q$ is true, you know that *one* of P or Q must be true. The problem is that you don't know *which one* is true, so you can't *assume* that either one *in particular* is true.

On the other hand, it is easy to *construct* disjunctions.

Constructing a Disjunction. If you know P, and Q is *any* statement, you may write down $P \vee Q$.

This says that if you know a statement, you can "or" it with *any other statement* to construct a disjunction.

1.	P	Premise
2.	$P \vee$ "Calvin sleeps with a night light."	Constructing a disjunction (1)

Notice that it doesn't matter what the other statement is! Once you know that P is true, any "or" statement with P must be true: An "or" statement is true if at least one of the pieces is true.

The next two rules are stated for completeness. They are easy enough that, as with double negation, we'll allow you to use them without a separate step or explicit mention.

Commutativity of Conjunctions. In any statement, you may substitute $P \wedge Q$ for $Q \wedge P$ (and write down the new statement).

Commutativity of Disjunctions. In any statement, you may substitute $P \vee Q$ for $Q \vee P$ (and write down the new statement).

Here is commutativity for a conjunction:

1.	$P \wedge \neg Q$	Premise
2.	$\neg Q \wedge P$	Commutativity (1)

Here is commutativity for a disjunction:

1.	$(Q \to P) \vee R$	Premise
2.	$R \vee (Q \to P)$	Commutativity (1) □

Before I give some examples of logic proofs, I'll explain where the rules of inference come from. You've probably noticed that the rules of inference correspond to tautologies. In fact, you can start with tautologies and use a small number of **simple inference rules** to derive all the other inference rules.

Three of the simple rules were stated above: The Rule of Premises, Modus Ponens, and Constructing a Conjunction. Here are two others. We've been using them without mention in some of our examples if you look closely.

Equivalence You may replace a statement by another that is logically equivalent. (Recall that P and Q are **logically equivalent** if and only if $P \leftrightarrow Q$ is a tautology.)

For instance, since P and $\neg\neg P$ are logically equivalent, you can replace P with $\neg\neg P$ or $\neg\neg P$ with P. This is Double Negation. As I mentioned, we're saving time by not writing out this step.

Substitution. You may take a known tautology and substitute for the simple statements.

This amounts to my remark at the start: In the statement of a rule of inference, the simple statements ("P", "Q", and so on) may stand for compound statements. "May stand for" is the same as saying "may be substituted with". We've been doing this without explicit mention.

Here's an example. The Disjunctive Syllogism tautology says

$$(\neg P \wedge (P \vee Q)) \rightarrow Q.$$

Suppose you have $\neg(A \wedge D)$ and $(A \wedge D) \vee C$ as premises. Here's how you'd apply the simple inference rules and the Disjunctive Syllogism tautology:

1.	$\neg(A \wedge D)$	Premise
2.	$(A \wedge D) \vee C$	Premise
3.	$\neg(A \wedge D) \wedge ((A \wedge D) \vee C)$	Constructing a conjunction (1, 3)
4.	$\neg(A \wedge D) \wedge ((A \wedge D) \vee C) \rightarrow C$	Substitution (Disjunctive Syllogism)
5.	C	Modus ponens (3, 4)

Notice that I used four of the five simple inference rules: the Rule of Premises, Modus Ponens, Constructing a Conjunction, and Substitution. In line 4, I used the Disjunctive Syllogism tautology $(\neg P \wedge (P \vee Q)) \rightarrow Q$ by substituting

$$A \wedge D \quad \text{for} \quad P \quad \text{and} \quad C \quad \text{for} \quad Q.$$

(Some people use the word "instantiation" for this kind of substitution.)

The advantage of this approach is that you have only five simple rules of inference. The disadvantage is that the proofs tend to be longer. With the approach I'll use, Disjunctive Syllogism is a rule of inference, and the proof is:

1.	$\neg(A \wedge D)$	Premise
2.	$(A \wedge D) \vee C$	Premise
3.	C	Disjunctive syllogism (1,2)

Here are some proofs which use the rules of inference. In each case, some **premises** --- statements that are assumed to be true --- are given, as well as a statement to prove. A proof consists of using the rules of inference to produce the statement to prove from the premises.

Example. Premises: $\left\{\begin{array}{c} \neg A \to (C \wedge D) \\ A \to B \\ \neg B \end{array}\right.$.

Prove: C.

1.	$A \to B$	Premise
2.	$\neg B$	Premise
3.	$\neg A$	Modus tollens (1,2)
4.	$\neg A \to (C \wedge D)$	Premise
5.	$C \wedge D$	Modus ponens (3,4)
6.	C	Decomposing a conjunction (5)

Example. Premises: $\left\{\begin{array}{c} P \wedge Q \\ P \to \neg(Q \wedge R) \\ S \to R \end{array}\right.$.

Prove: $\neg S$.

1.	$P \wedge Q$	Premise
2.	P	Decomposing a conjunction (1)
3.	Q	Decomposing a conjunction (1)
4.	$P \to \neg(Q \wedge R)$	Premise
5.	$\neg(Q \wedge R)$	Modus ponens (3,4)
6.	$\neg Q \vee \neg R$	DeMorgan (5)
7.	$\neg R$	Disjunctive syllogism (3,6)
8.	$S \to R$	Premise
9.	$\neg S$	Modus tollens (7,8) □

Example. Premises: $\left\{\begin{array}{c} \neg(A \vee B) \to C \\ \neg A \\ \neg C \end{array}\right.$.

Prove: B.

1.	$\neg(A \vee B) \to C$	Premise
2.	$\neg C$	Premise
3.	$A \vee B$	Modus tollens (1,2)
4.	$\neg A$	Premise
5.	B	Disjunctive syllogism (3,4)

Open Statements

An open statement in x associates with the name of each object in a collection, called the universe of the open statement, a logical statement. Such a logical statement is called a component of the open statement. It is obtained by replacing the x (or any other variable letter) in the open statement with the name or symbol of the object.

Eg

Suppose the open statement in x is: "**x** is greater than 3". The universe is the collection **{1,2,3,4,5}** of the first five natural numbers. Replace the x by each of these numbers to get the five components (or component statements):

1 is greater than 3	component is F	**1** receives an F
2 is greater than 3	component is F	**2** receives an F
3 is greater than 3	component is F	**3** receives an F
4 is greater than 3	component is T	**4** receives an T
5 is greater than 3	component is T	**5** receives an T

Quantifiers

We need quantifiers to formally express the meaning of the words "all" and "some". The two most important quantifiers are: **Universal quantifier**, "For all". Symbol:$\forall$ **Existential quantifier**, "There exists". Symbol:$\exists$$\forall x$ $P(x)$asserts that $P(x)$is true for every x in the domain. $\exists x$ $P(x)$asserts that $P(x)$is true for some x in the domain. The quantifiers are said to bind the variable x in these expressions. Variables in the scope of some quantifier are called bound variables. All other variables in the expression are called free variables. A propositional function that does not contain any free variables isa proposition and has a truth value.

The Universal Quantifier

The expression: $\forall x\ P(x)$, denotes the **universal quantification** of the atomic formula P(x). Translated into the English language, the expression is understood as: "*For all x, P(x)* holds", "*for each x, P(x)* holds" or "*for every x, P(x)* holds". $\forall$ is called the **universal quantifier**, and $\forall x$ means all the objects x in the universe. If this is followed by *P(x)* then the meaning is that *P(x)* is true for every object *x* in the universe. For example, "All cars have wheels" could be transformed into the propositional form, $\forall x\ P(x)$, where:

- *P(x)* is the predicate denoting: ***x* has wheels**, and
- the universe of discourse is only populated by cars.

The Existential Quantifier

The expression: $\exists xP(x)$, denotes the **existential quantification** of *P(x)*. Translated into the English language, the expression could also be understood as: "There exists an *x* such that *P(x)*" or "There is at least one *x* such that *P(x)*" $\exists$ is called the **existential quantifier**, and $\exists x$ means at least one object *x* in the universe. If this is followed by *P(x)* then the meaning is that *P(x)* is true for at least one object *x* of the universe. For example, "*Someone loves you*" could be transformed into the propositional form, $\exists x\ P(x)$, where:

- *P(x)* is the predicate meaning: ***x* loves you**,
- The universe of discourse contains (but is not limited to) all living creatures.

<u>Eg</u>

<u>Premises</u>:

a. “It’s not sunny and it’s colder than yesterday”$\neg p \land q$

b. “We will go swimming only if it’s sunny.”$r \rightarrow p$

c. “If we don’t go swimming then we will take canoe trip.”$\neg r \rightarrow s$

d. “If we take a canoe trip, then we will be home by sunset.”$s \rightarrow t$

Conclusion: “We will be home by sunset.”t.

<u>Solution</u>

(1)	$\neg p \land q$	Premise
(2)	$\neg p$	Simplification rule using (1)
(3)	$r \rightarrow p$	Premise
(4)	$\neg r$	MT using (2) (3)
(5)	$\neg r \rightarrow s$	Premise
(6)	s	MP using (4) (5)
(7)	$s \rightarrow t$	Premise
(8)	t	MP using (6) (7)

This is a valid argument showing that from the premises (a), (b), (c)and (d), we can prove the **conclusion** t.

<u>Eg</u>

Suppose $P \rightarrow Q$; $\neg P \rightarrow R$; $Q \rightarrow S$. Prove that $\boldsymbol{\neg R \rightarrow S}$**.**

<u>Solution</u>

(1)	$P \rightarrow Q$	Premise
(2)	$\neg P \lor Q$	Logically equivalent to (1)
(3)	$\neg P \rightarrow R$	Premise
(4)	$P \lor R$	Logically equivalent to (3)
(5)	$Q \lor R$	Apply resolution rule to (2)(4)
(6)	$\neg R \rightarrow Q$	Logically equivalent to (5)
(7)	$Q \rightarrow S$	Premise
(8)	$\neg R \rightarrow S$	Apply HS rule to (6)(7).

Theory of Inference for Predicate Calculus

Rule of Inference	Name
(i) $\forall x P(x) \rightarrow p(c)$ for an arbitrary element c.	Universal Specification(US)
(ii) P(c) for an arbitrary element c $\forall x P(x)$.	Universal Generalization(UG)
(iii) $\exists x P(x) \rightarrow p(c)$ for some element c.	Existential Specification(ES)
(iv) p(c) for some element c $\exists x P(x)$.	Existential Generalization(EG)

Eg

Suppose: all natural numbers are integers; there exists a natural number; **Prove** that there exists an integer.

Solution

We can formalize this problem as follows.
N(x): x is a natural number.
I(x): x is an integer.
Premise:$\forall x(N(x)\rightarrow I(x)), \exists x\, N(x)$ **Need to prove**:$\exists x\, I(x)$

(1)	$\exists x\, N(x)$	Premise
(2)	N(c)	Apply existential specification rule to (1)
(3)	$\forall x(N(x)\rightarrow I(x))$	Premise
(4)	$N(c)\rightarrow I(c)$	Apply universal specification rule to (3)
(5)	I(c)	Apply MP rule to (2)(4)
(6)	$\exists x\, I(x)$	Apply existential generalization rule to (5)

REVIEW QUESTIONS

1) Define conjunctive normal form.
2) What is meant by Open statement.
3) Obtain PCNF and PDNF of $(\neg p\rightarrow r)\wedge(q\leftrightarrow p)$.
4) Obtain the conjunctive normal form $\neg(p\vee q)\leftrightarrow(p\wedge q)$
5) Obtain PDNF of $(P\wedge Q)\vee(\neg P\wedge R)\vee(Q\wedge R)$
6) What is disjunctive normal? give example.
7) Define PCNF and PDNF.
8) What is universal quantifier ? Give example.

UNIT – III
GRAPH THEORY

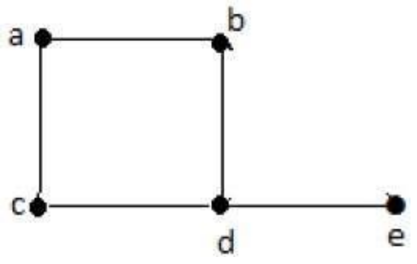
GRAPH THEORY

Basic Concepts

Graph

A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as **vertices**, and the links that connect the vertices are called **edges**.

Formally, a graph is a pair of sets **(V, E)**, where **V** is the set of vertices and E is the set of edges, connecting the pairs of vertices. Take a look at the following graph



In the above graph,

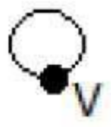
V = {a, b, c, d, e}

E = {ab, ac, bd, cd, de}

Loop

In a graph, if an edge is drawn from vertex to itself, it is called a loop.

Eg

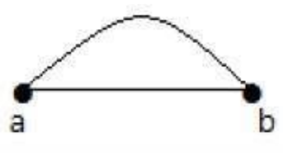


In the above graph, V is a vertex for which it has an edge (V, V) forming a loop.

Parallel Edges

In a graph, if a pair of vertices is connected by more than one edge, then those edges are called parallel edges.

Eg



In the above graph, ‘a’ and ‘b’ are the two vertices which are connected by two edges ‘ab’ and ‘ab’ between them. So it is called as a parallel edge.

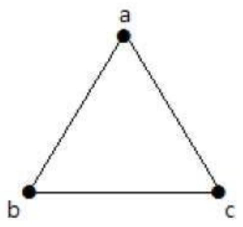
Simple Graph

A graph **with no loops** and no **parallel edges** is called a simple graph.

- The maximum number of edges possible in a single graph with ‘n’ vertices is ${}^nC_2$ where ${}^nC_2 = n(n – 1)/2$.
- The number of simple graphs possible with ‘n’ vertices = $2^{{}^nc_2} = 2^{n(n-1)/2}$.

Eg

In the following graph, there are 3 vertices with 3 edges which is maximum excluding the parallel edges and loops. This can be proved by using the above formulae.



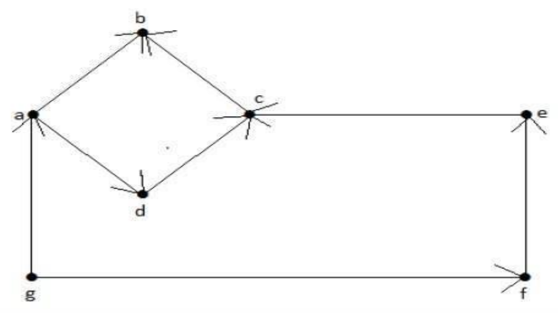
The maximum number of edges with n=3 vertices −

$$
\begin{aligned}
{}^nC_2 &= n(n–1)/2 \\
&= 3(3–1)/2 \\
&= 6/2 \\
&= 3 \text{ edges}
\end{aligned}
$$

Directed Graph

In a directed graph, each edge has a direction.

Eg



In the above graph, we have seven vertices ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, and ‘g’, and eight edges ‘ab’, ‘cb’, ‘dc’, ‘ad’, ‘ec’, ‘fe’, ‘gf’, and ‘ga’. As it is a directed graph, each edge bears an arrow mark that shows its direction. Note that in a directed graph, ‘ab’ is different from ‘ba’.

Degree of Vertex

It is the number of vertices adjacent to a vertex V.

Notation − deg(V).

In a simple graph with n number of vertices, the degree of any vertices is −

$$\deg(v) \leq n - 1 \ \forall \ v \in G$$

A vertex can form an edge with all other vertices except by itself. So the degree of a vertex will be up to the **number of vertices in the graph minus 1**. This 1 is for the self-vertex as it cannot form a loop by itself. If there is a loop at any of the vertices, then it is not a Simple Graph.

Degree of vertex can be considered under two cases of graphs −

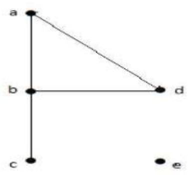
- Undirected Graph
- Directed Graph

Degree of Vertex in an Undirected Graph

An undirected graph has no directed edges. Consider the following examples.

Eg

Take a look at the following graph −



In the above Undirected Graph,

- deg(a) = 2, as there are 2 edges meeting at vertex ‘a’.
- deg(b) = 3, as there are 3 edges meeting at vertex ‘b’.
- deg(c) = 1, as there is 1 edge formed at vertex ‘c’

- So ‘c’ is a **pendent vertex**.
- deg(d) = 2, as there are 2 edges meeting at vertex ‘d’.
- deg(e) = 0, as there are 0 edges formed at vertex ‘e’.
- So ‘e’ is an **isolated vertex**.

Degree of Vertex in a Directed Graph

In a directed graph, each vertex has an **indegree** and an **outdegree**.

Indegree of a Graph

- **Indegree** of vertex V is the number of edges which are coming into the vertex V.
- **Notation** − deg−(V).

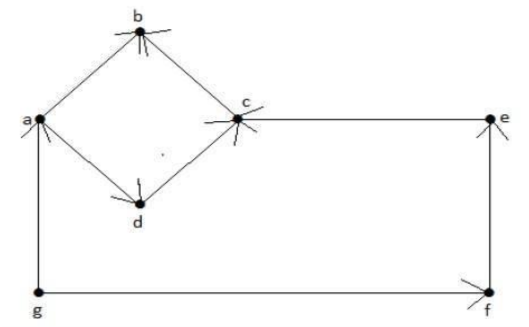
Outdegree of a Graph

- **Outdegree** of vertex V is the number of edges which are going out from the vertex V.
- **Notation** − deg+(V).

Consider the following examples.

Eg

Take a look at the following directed graph. Vertex ‘a’ has two edges, ‘ad’ and ‘ab’, which are going outwards. Hence its outdegree is 2. Similarly, there is an edge ‘ga’, coming towards vertex ‘a’. Hence the indegree of ‘a’ is 1.



The indegree and outdegree of other vertices are shown in the following table −

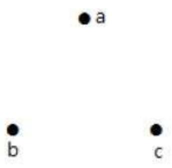
Vertex	Indegree	Outdegree
a	1	2
b	2	0

c	2	1
d	1	1
e	1	1
f	1	1
g	0	2

Null Graph

A graph having no edges is called a Null Graph.

Eg

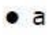


In the above graph, there are three vertices named ‘a’, ‘b’, and ‘c’, but there are no edges among them. Hence it is a Null Graph.

Trivial Graph

A **graph with only one vertex** is called a Trivial Graph.

Eg



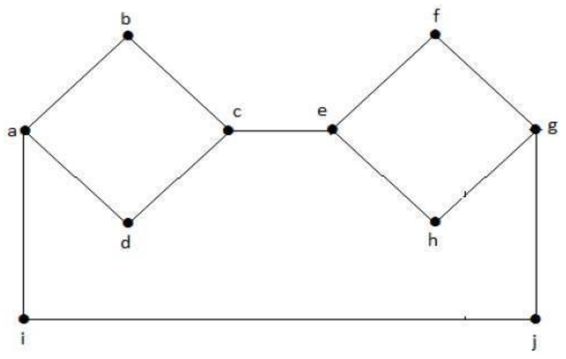
In the above shown graph, there is only one vertex ‘a’ with no other edges. Hence it is a Trivial graph.

Connected Graph

A graph G is said to be connected **if there exists a path between every pair of vertices**. There should be at least one edge for every vertex in the graph. So that we can say that it is connected to some other vertex at the other side of the edge.

Eg

In the following graph, each vertex has its own edge connected to other edge. Hence it is a connected graph.

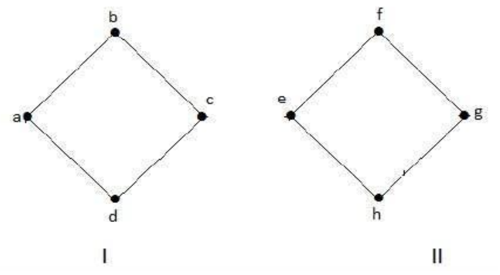


Disconnected Graph

A graph G is disconnected, if it does not contain at least two connected vertices.

Eg

The following graph is an example of a Disconnected Graph, where there are two components, one with 'a', 'b', 'c', 'd' vertices and another with 'e', 'f', 'g', 'h' vertices.



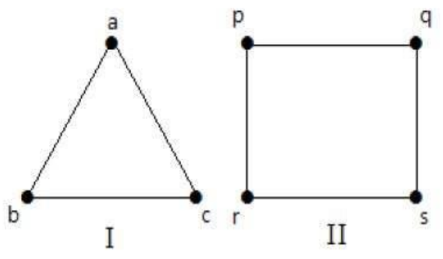
The two components are independent and not connected to each other. Hence it is called disconnected graph.

Regular Graph

A graph G is said to be regular, **if all its vertices have the same degree**. In a graph, if the degree of each vertex is 'k', then the graph is called a 'k-regular graph'.

Eg

In the following graphs, all the vertices have the same degree. So these graphs are called regular graphs.



In both the graphs, all the vertices have degree 2. They are called 2-Regular Graphs.

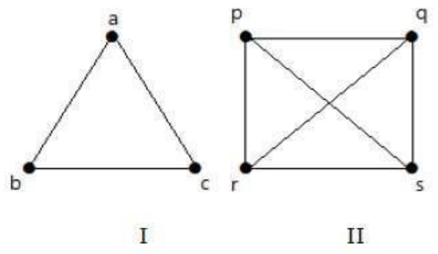
Complete Graph

A simple graph with ‘n’ mutual vertices is called a complete graph and it is **denoted by ‘K_n’**. In the graph, **a vertex should have edges with all other vertices**, then it called a complete graph.

In other words, if a vertex is connected to all other vertices in a graph, then it is called a complete graph.

Eg

In the following graphs, each vertex in the graph is connected with all the remaining vertices in the graph except by itself.

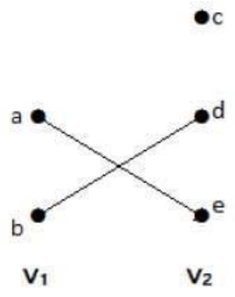


Bipartite Graph

A simple graph G = (V, E) with vertex partition V = {V_1, V_2} is called a bipartite graph **if every edge of E joins a vertex in V1 to a vertex in V_2**.

In general, a Bipartite graph has two sets of vertices, let us say, V1 and V2, and if an edge is drawn, it should connect any vertex in set V_1 to any vertex in set V_2.

Eg



In this graph, you can observe two sets of vertices − V_1 and V_2. Here, two edges named ‘ae’ and ‘bd’ are connecting the vertices of two sets V_1 and V_2.

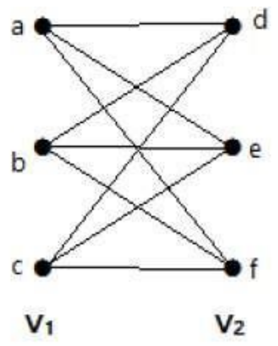
Complete Bipartite Graph

A bipartite graph ‘G’, G = (V, E) with partition V = {V_1, V_2} is said to be a complete bipartite graph if every vertex in V_1 is connected to every vertex of V_2.

In general, a complete bipartite graph connects each vertex from set V_1 to each vertex from set V_2.

Eg

The following graph is a complete bipartite graph because it has edges connecting each vertex from set V_1 to each vertex from set V_2.



If $|V_1| = m$ and $|V_2| = n$, then the complete bipartite graph is denoted by $K_{m,n}$.

- $K_{m,n}$ has $(m+n)$ vertices and (mn) edges.
- $K_{m,n}$ is a regular graph if $m=n$.

In general, a **complete bipartite graph is not a complete graph**.

Isomorphic Graphs

Two graphs G_1 and G_2 are said to be isomorphic if −

- Their number of components (vertices and edges) are same.
- Their edge connectivity is retained.

Note

If $G_1 \equiv G_2$ then −

$|V(G_1)| = |V(G_2)|$

$|E(G_1)| = |E(G_2)|$

Degree sequences of G_1 and G_2 are same.

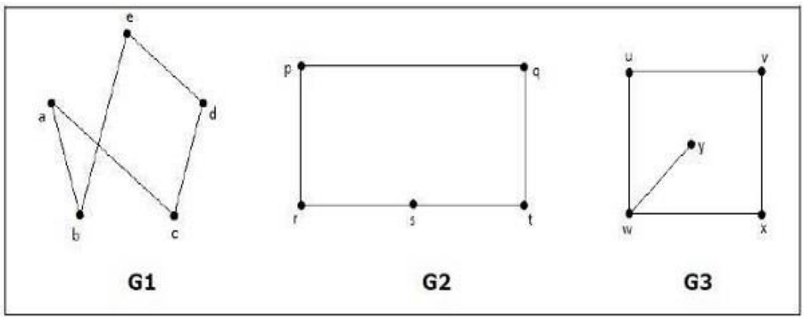
If the vertices $\{V_1, V_2, .. Vk\}$ form a cycle of length K in G_1, then the vertices $\{f(V_1), f(V_2),\dots f(Vk)\}$ should form a cycle of length K in G_2.

All the above conditions are necessary for the graphs G_1 and G_2 to be isomorphic, but not sufficient to prove that the graphs are isomorphic.

- $(G_1 \equiv G_2)$ if and only if $(G_1{-} \equiv G_2{-})$ where G_1 and G_2 are simple graphs.
- $(G_1 \equiv G_2)$ if the adjacency matrices of G_1 and G_2 are same.
- $(G_1 \equiv G_2)$ if and only if the corresponding subgraphs of G_1 and G_2 (obtained by deleting some vertices in G1 and their images in graph G_2) are isomorphic.

Eg

Which of the following graphs are isomorphic?



In the graph G_3, vertex 'w' has only degree 3, whereas all the other graph vertices has degree 2. Hence G3 not isomorphic to G_1 or G_2.

Matrix Representation of Graphs

A graph can be represented using Adjacency Matrix way.

Adjacency Matrix

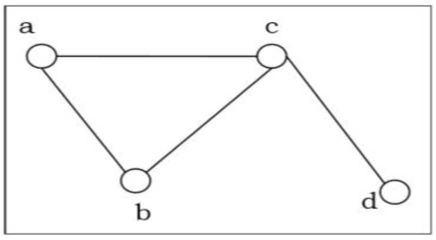
An Adjacency Matrix A[V][V] is a 2D array of size V × V where V is the number of vertices in a undirected graph. If there is an edge between V_x to V_y then the value of $A[V_x][V_y]=1$ and $A[V_y][V_x]=1$, otherwise the value will be zero.

For a directed graph, if there is an edge between V_x to V_y, then the value of $A[V_x][V_y]=1$, otherwise the value will be zero.

Adjacency Matrix of an Undirected Graph

Let us consider the following undirected graph and construct the adjacency matrix –

Eg



Adjacency matrix of the above undirected graph will be −

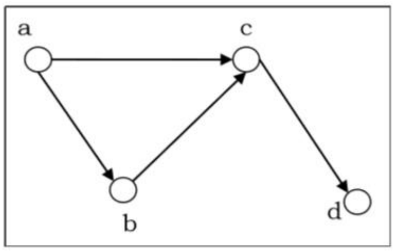
	a	**B**	**C**	**D**
a	0	1	1	0
b	1	0	1	0
c	1	1	0	1

d	0	0	1	0

Adjacency Matrix of a Directed Graph

Let us consider the following directed graph and construct its adjacency matrix –

Eg



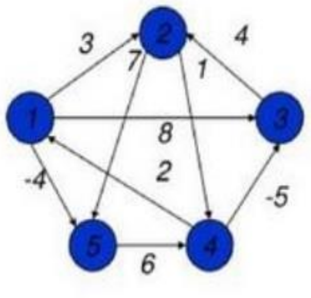
Adjacency matrix of the above directed graph will be −

	a	**b**	**C**	**D**
A	0	1	1	0
B	0	0	1	0
C	0	0	0	1
D	0	0	0	0

Warshall Algorithm

The all pair shortest path algorithm is also known as Floyd-Warshall algorithm is used to find all pair shortest path problem from a given weighted graph. As a result of this algorithm, it will generate a matrix, which will represent the minimum distance from any node to all other nodes in the graph.

Eg



$$\begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

At first the output matrix is same as given cost matrix of the graph. After that the output matrix will be updated with all vertices k as the intermediate vertex.

The time complexity of this algorithm is O(V3), here V is the number of vertices in the graph.

Algorithm

```
Begin
  for k := 0 to n, do
    for i := 0 to n, do
      for j := 0 to n, do
        if cost[i,k] + cost[k,j] < cost[i,j], then
          cost[i,j] := cost[i,k] + cost[k,j]
        done
      done
    done
    display the current cost matrix
End
```

Tree

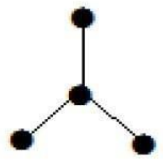
A **connected acyclic graph** is called a tree. In other words, a connected graph with no cycles is called a tree.

The edges of a tree are known as **branches**. Elements of trees are called their nodes. The nodes without child nodes are called **leaf nodes**.

A tree with ‘n’ vertices has ‘n-1’ edges. If it has one more edge extra than ‘n-1’, then the extra edge should obviously has to pair up with two vertices which leads to form a cycle. Then, it becomes a cyclic graph which is a violation for the tree graph.

Eg

The graph shown here is a tree because it has no cycles and it is connected. It has four vertices and three edges, i.e., for ‘n’ vertices ‘n-1’ edges as mentioned in the definition.



Note − Every tree has at least two vertices of degree one.

Center of a Tree

The center of a tree is a vertex with minimal eccentricity. The eccentricity of a vertex ‘X’ in a tree ‘G’ is the maximum distance between the vertex ‘X’ and any other vertex of the tree. The maximum eccentricity is the tree diameter. If a tree has only one center, it is called Central Tree and if a tree has only more than one centers, it is called Bi-central Tree. Every tree is either central or bi-central.

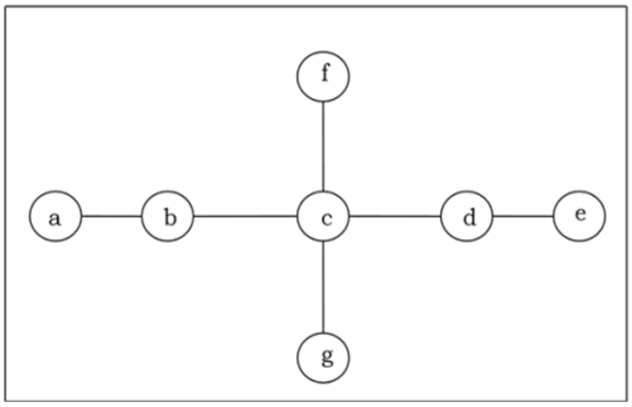
Stps to find centers of a tree

Step 1 − Remove all the vertices of degree 1 from the given tree and also remove their incident edges.

Step 2 − Repeat step 1 until either a single vertex or two vertices joined by an edge is left. If a single vertex is left then it is the center of the tree and if two vertices joined by an edge is left then it is the bi-center of the tree.

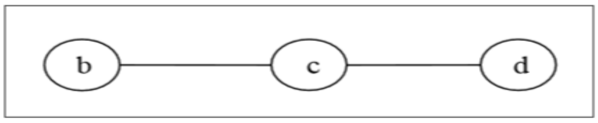
Eg

Find out the center/bi-center of the following tree −

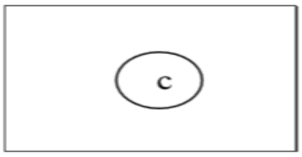


Solution

At first, we will remove all vertices of degree 1 and also remove their incident edges and get the following tree −



Again, we will remove all vertices of degree 1 and also remove their incident edges and get the following tree −



Finally we got a single vertex ‘c’ and we stop the algorithm. As there is single vertex, this tree has one center ‘c’ and the tree is a central tree.

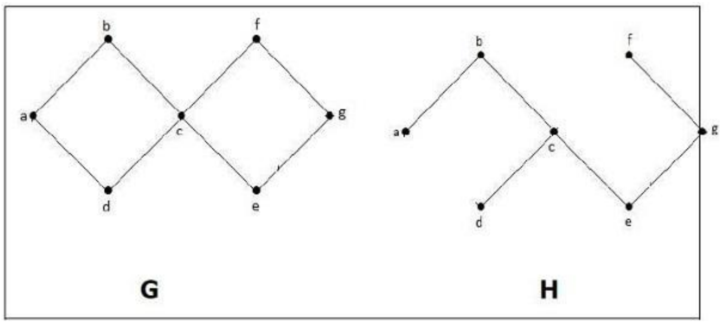
Spanning Trees

Let G be a connected graph, then the sub-graph H of G is called a spanning tree of G if −

- H is a tree
- H contains all vertices of G.

A spanning tree T of an undirected graph G is a subgraph that includes all of the vertices of G.

Eg



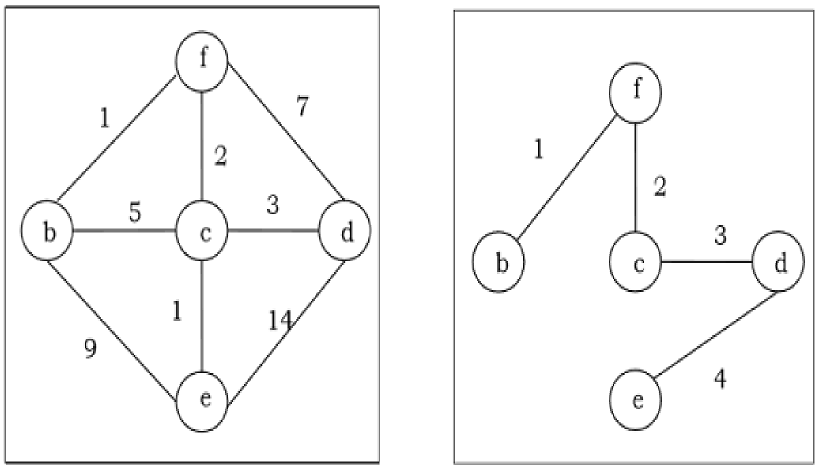
In the above example, G is a connected graph and H is a sub-graph of G.

Clearly, the graph H has no cycles, it is a tree with six edges which is one less than the total number of vertices. Hence H is the Spanning tree of G.

Minimum Spanning Tree

A spanning tree with assigned weight less than or equal to the weight of every possible spanning tree of a weighted, connected and undirected graph G, it is called minimum spanning tree (MST). The weight of a spanning tree is the sum of all the weights assigned to each edge of the spanning tree.

Eg



Kruskal's Algorithm

Kruskal's algorithm is a greedy algorithm that finds a minimum spanning tree for a connected weighted graph. It finds a tree of that graph which includes every vertex and the total weight of all the edges in the tree is less than or equal to every possible spanning tree.

Algorithm

Step 1 − Arrange all the edges of the given graph $G (V,E)$ in ascending order as per their edge weight.

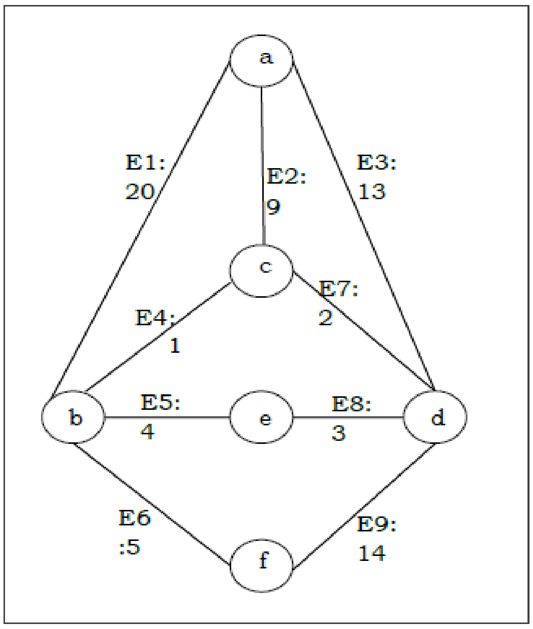
Step 2 − Choose the smallest weighted edge from the graph and check if it forms a cycle with the spanning tree formed so far.

Step 3 − If there is no cycle, include this edge to the spanning tree else discard it.

Step 4 − Repeat Step 2 and Step 3 until \$(V-1)\$ number of edges are left in the spanning tree.

Eg

Suppose we want to find minimum spanning tree for the following graph G using Kruskal's algorithm.



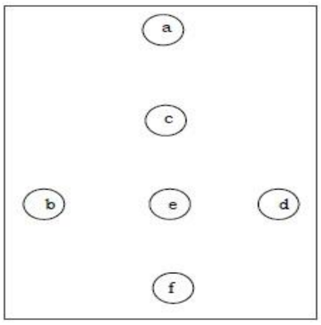
Solution

From the above graph we construct the following table −

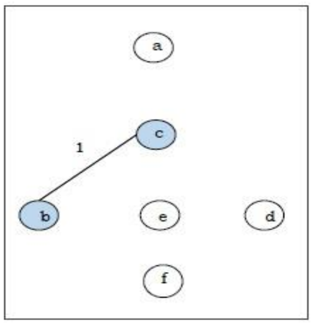
Edge No.	Vertex Pair	Edge Weight
E1	(a, b)	20
E2	(a, c)	9
E3	(a, d)	13
E4	(b, c)	1
E5	(b, e)	4
E6	(b, f)	5
E7	(c, d)	2
E8	(d, e)	3
E9	(d, f)	14

Now we will rearrange the table in ascending order with respect to Edge weight −

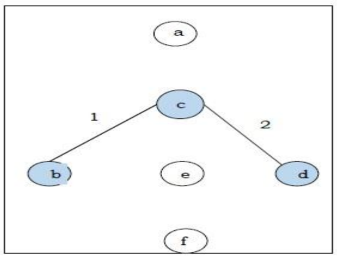
Edge No.	Vertex Pair	Edge Weight
E4	(b, c)	1
E7	(c, d)	2
E8	(d, e)	3
E5	(b, e)	4
E6	(b, f)	5
E2	(a, c)	9
E3	(a, d)	13
E9	(d, f)	14
E1	(a, b)	20



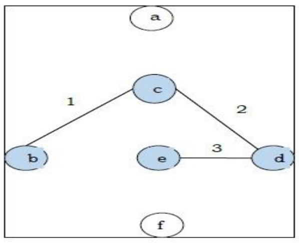
After adding vertices



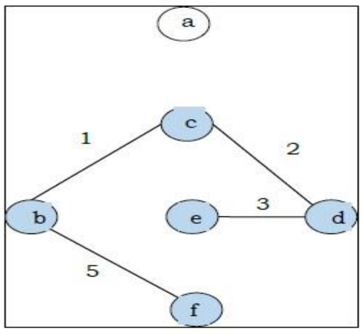
After adding edge E4



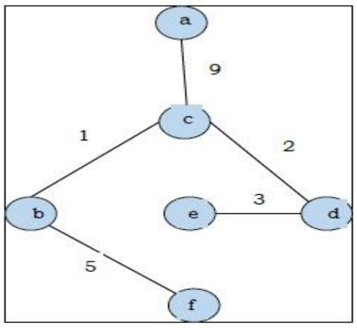
After adding edge E7



After adding edge E8



After adding edge E6
(don't add E5 since it forms cycle)



After adding edge E2

Since we got all the 5 edges in the last figure, we stop the algorithm and this is the minimal spanning tree and its total weight is $(1 + 2 + 3 + 5 + 9) = 20$.

Prim's Algorithm

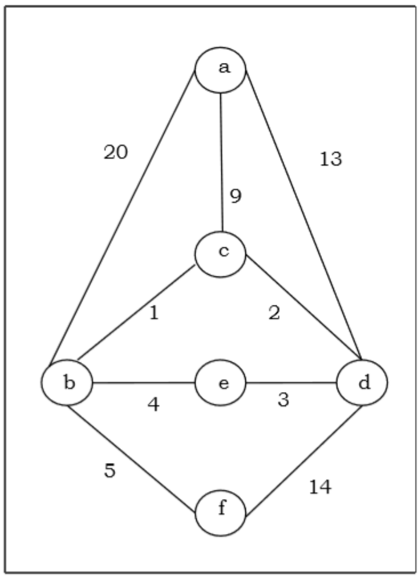
Prim's algorithm, discovered in 1930 by mathematicians, Vojtech Jarnik and Robert C. Prim, is a greedy algorithm that finds a minimum spanning tree for a connected weighted graph. It finds a tree of that graph which includes every vertex and the total weight of all the edges in the tree is less than or equal to every possible spanning tree. Prim's algorithm is faster on dense graphs.

Algorithm

- Initialize the minimal spanning tree with a single vertex, randomly chosen from the graph.
- Repeat steps 3 and 4 until all the vertices are included in the tree.
- Select an edge that connects the tree with a vertex not yet in the tree, so that the weight of the edge is minimal and inclusion of the edge does not form a cycle.
- Add the selected edge and the vertex that it connects to the tree.

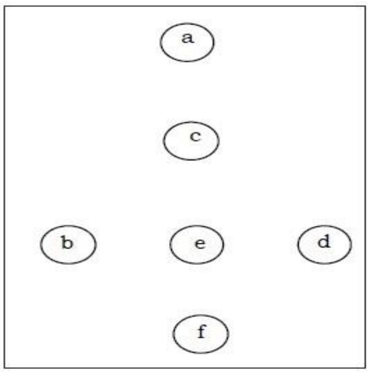
Eg

Suppose we want to find minimum spanning tree for the following graph G using Prim's algorithm.

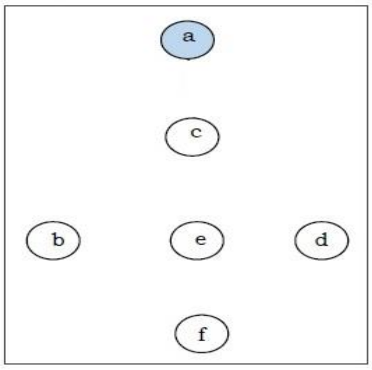


Solution

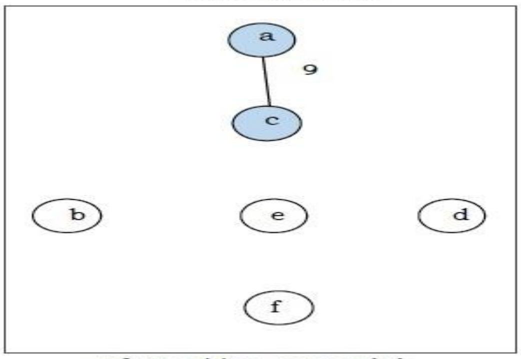
Here we start with the vertex ‘a’ and proceed.



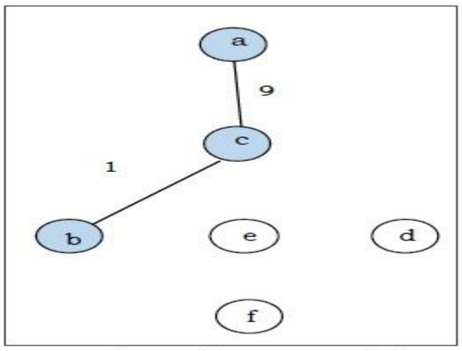
No vertices added



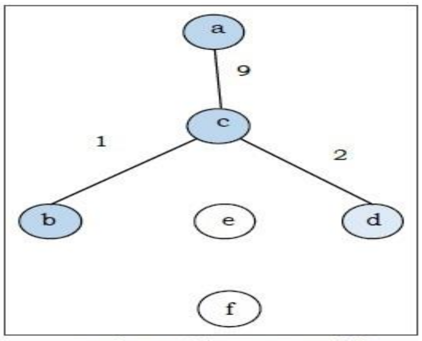
After adding vertex ‘a’

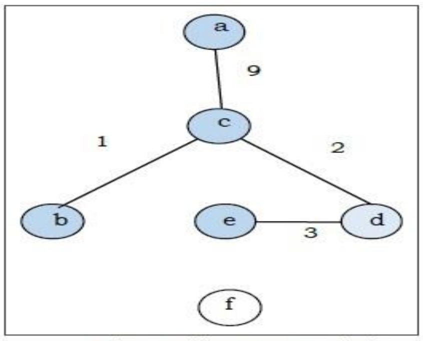


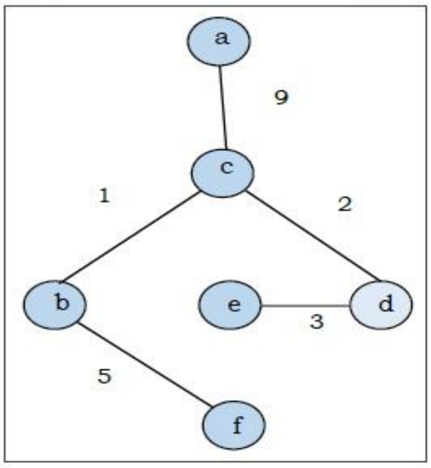
After adding vertex ‘c’



After adding vertex ‘b’


After adding vertex 'd'


After adding vertex 'e'


After adding vertex 'f'

This is the minimal spanning tree and its total weight is $(1 + 2 + 3 + 5 + 9) = 20$.

Shortest Path Problem

Dijkstra's Algorithm

Dijkstra's algorithm solves the single-source shortest-paths problem on a directed weighted graph $G = (V, E)$, where all the edges are non-negative (i.e., $w(u, v) \geq 0$ for each edge $(u, v) \in E$).

In the following algorithm, we will use one function ***Extract-Min()***, which extracts the node with the smallest key.

Eg

Let us consider vertex ***1*** and ***9*** as the start and destination vertex respectively. Initially, all the vertices except the start vertex are marked by ∞ and the start vertex is marked by ***0***.

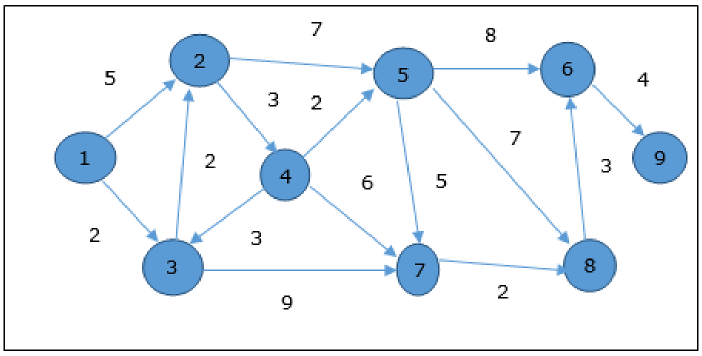
Vertex	Initial	Step1 V_1	Step2 V_3	Step3 V_2	Step4 V_4	Step5 V_5	Step6 V_7	Step7 V_8	Step8 V_6
1	0	0	0	0	0	0	0	0	0

2	∞	5	4	4	4	4	4	4	4
3	∞	2	2	2	2	2	2	2	2
4	∞	∞	∞	7	7	7	7	7	7
5	∞	∞	∞	11	9	9	9	9	9
6	∞	∞	∞	∞	∞	17	17	16	16
7	∞	∞	11	11	11	11	11	11	11
8	∞	∞	∞	∞	∞	16	13	13	13
9	∞	∞	∞	∞	∞	∞	∞	∞	20

Hence, the minimum distance of vertex ***9*** from vertex ***1*** is ***20***. And the path is

$$1 \to 3 \to 7 \to 8 \to 6 \to 9$$

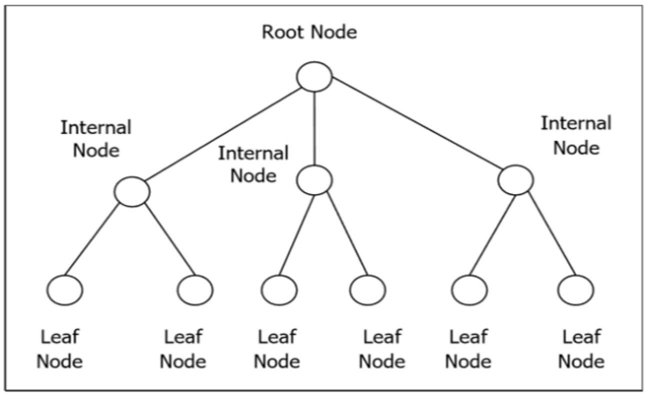
This path is determined based on predecessor information.



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.

Eg



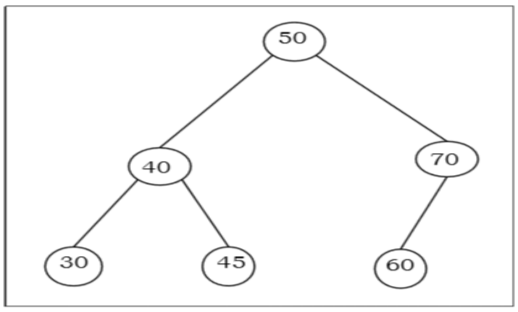
Binary Search Tree

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

- ‘X’ in left sub-tree of vertex V, Value(X) \le Value (V)
- ‘Y’ in right sub-tree of vertex V, Value(Y) \ge 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.

Eg



Traversing a Binary Tree

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree −

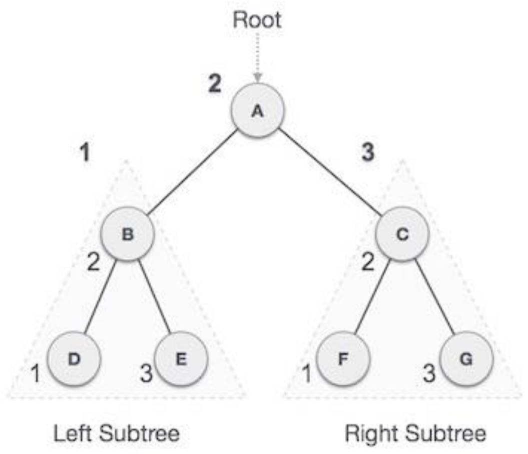
- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

In-order Traversal

In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.

If a binary tree is traversed **in-order**, the output will produce sorted key values in an ascending order.



We start from **A**, and following in-order traversal, we move to its left subtree **B**. **B** is also traversed in-order. The process goes on until all the nodes are visited. The output of inorder traversal of this tree will be −

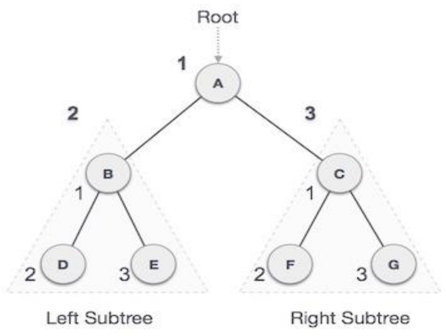
D → B → E → A → F → C → G

Algorithm

Until all nodes are traversed −
Step 1 − Recursively traverse left subtree.
Step 2 − Visit root node.
Step 3 − Recursively traverse right subtree.

Pre-order Traversal

In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.



We start from **A**, and following pre-order traversal, we first visit **A** itself and then move to its left subtree **B**. **B** is also traversed pre-order. The process goes on until all the nodes are visited. The output of pre-order traversal of this tree will be −

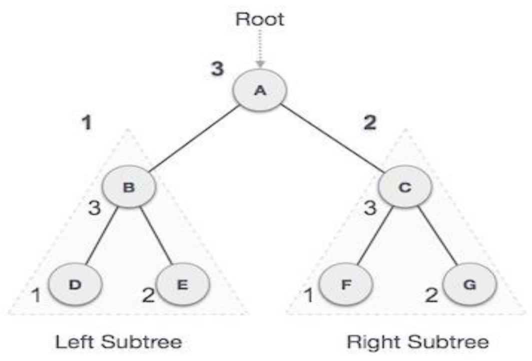
$$A \to B \to D \to E \to C \to F \to G$$

Algorithm

Until all nodes are traversed −
Step 1 − Visit root node.
Step 2 − Recursively traverse left subtree.
Step 3 − Recursively traverse right subtree.

Post-order Traversal

In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.



We start from **A**, and following Post-order traversal, we first visit the left subtree **B**. **B** is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be −

$$D \to E \to B \to F \to G \to C \to A$$

Algorithm

Until all nodes are traversed −
Step 1 − Recursively traverse left subtree.
Step 2 − Recursively traverse right subtree.
Step 3 − Visit root node.

REVIEW QUESTIONS

1) Define Binary Tree.
2) Define indegree and outdegree.
3) Explain the representation of trees.
4) Explain in detail about matrix representation of graphs with example.
5) Define spanning tree.
6) Define Rooted tree.

7) Explain traversal of binary tree.

8) What is meant by complete asymmetric graph.

UNIT – IV

LINEAR PROGRAMMING PROBLEM(LPP)

- **Mathematical Formulation**
- **Graphical Solution**
- **Slack and Surplus Variable**
- **Simplex Method**
- **Two Phase Method**

LINEAR PROGRAMMING PROBLEM(LPP)

Decision Variables and their Relationships

The decision variable refers to any candidate (person, service, projects, jobs, tasks) competing with other decision variables for limited resources. These variables are usually interrelated in terms of utilization of resources and need simultaneous solutions, i.e., the relationship among these variables should be linear.

Objective Function

The Linear Programming Problem must have a well defined objective function to optimize the results. For instance, minimization of cost or maximization of profits. It should be expressed as linear function of decision variables ($Z = X1 + X2$, where Z represents the objective, i.e., minimization/maximization, X1 and X2 are the decision variables directly affecting the Z value).

Constraints

There would be limitations on resources which are to be allocated among various competing activities. These must be capable of being expressed as linear equalities or inequalities in terms of decision variables.

Non-Negativity Restrictions

All variables must assume non-negative values. If any of the variable is unrestricted in sign, a tool can be employed which will enforce the negativity without changing the original information of a problem.

Mathematical Formulation of Linear Programming Problems(LPP)

Steps for formulating LPP,

1. Identify the nature of the problem (maximization/minimization problem).
2. Identify the number of variables to establish the objective function.
3. Formulate the constraints.

4. Develop non-negativity constraints.

Eg

A firm manufactures 2 types of products A & B and sells them at a profit or ` 2 on type A & ` 3 on type B. Each product is processed on 2 machines G & H. Type a requires1 minute of processing time on G and 2 minutes on H. Type B requires one minute on G &1 minute on H. The machine G is available for not more than 6 hrs. 40 mins. while machine H is available for 10 hrs. during any working day. Formulate the problem as LPP.

Solution

Let, x_1 be the no. of products of type A.

x_2 be the no. of products of type B.

Since the profit on type A is ` 2 per product, $2x_1$ will be the profit on selling x_1 units of type A. Similarly $3x_2$ will be the profit on selling x_2 units of type B.

Hence the objective function will be,Maximize 'Z' = $2x_1 + 3x_2$ is subject to constraints.

Since machine 'G' takes one minute on 'A' and one minute on 'B', the total number of minutes required is given by $x1 + x2$. Similarly, on machine 'H' $2x1 + x2$. But 'G' is not available for more than 400 minutes. Therefore, $x1 + x2 \leq 400$ and H is not available for more than 600 minutes, therefore, $2x1 + x2 \leq 600$ and $x1, x2, \geq 0$, i.e.,

$x_1 + x_2 \leq 400$ (Time availability constraints)

$2x1 + x2 \leq 600$

$x1, x2 \geq 0$ (Non-negativity constraints)

Graphical Solutions under Linear Programming

Steps

1. Consider each inequality constraint as an equation.
2. Plot each equation on the graph as each will geometrically represent a straight line.
3. Plot the feasible region, every point on the line will satisfy the equation on the line.
4. If the inequality constraint corresponding to that line is less than or equal to, then the region below the line lying in the 1st quadrant (as shown in above graph) is shaded (due to non-negativity of variables); for the inequality constraint with greater than or equal to sign, the region above the line in the 1st quadrant is shaded. The points lying in common region will satisfy all the constraints simultaneously. Hence, it is called feasible region.

5. Identify the co-ordinates of the corner points.

6. Find the 'Z' value by substituting the co-ordinates of corner points to the objective functions.

Eg

Maximize 'Z' $=3x_1 + 5x_2$

(Subject to constraints)

$x_1 + 2x_2 \leq 2,000$

$x_1 + x_2 \leq 1,500$

$x_2 \leq 600$

$x1, x2 \leq 0$

Solution

Step 1: Convert the inequalities into equalities and find the divisible of the equalities.

Equation	x_1	x_2
$x_1 + 2x_2 = 2,000$	2,000	1,000
$x_1 + x_2 = 1,500$	1,500	1,500
$x_2 = 600$	---	600

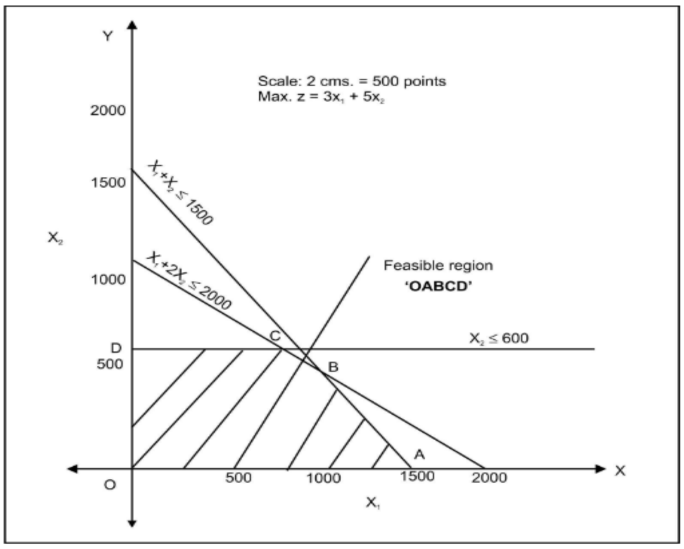
Step 2: Fix up the graphic scale.

Maximum points =2,000

Minimum points =600

2 cms =500 points

Step 3: Graph the data



Step 4: Find the co-ordinates of the corner points

Corner Points	X_1	x_2
O	0	0
A	1,500	0
B	1,000	500
C	800	600
D	0	600

At ‘B’: $x_1 + 2x_2 = 2,000$ (1)

$x_1 + x_2 = 1,500$ (2)

$x_2 = 500$

Notes

Put $x_2 =$ 500 in eq. (1),

$x_1 + 2(500) = 2,000$

Therefore $x_1 = 2,000 – 1,000$

Therefore $x_1 = 1,000$

At ‘C’: $x_1 + 2x_2 = 2,000$ ………(1)

$x_2 = 600$ ………(2)

Put $x_2 =$ 600 in eq. (1),

$x_1 + 2(600) = 2,000$

$x_1 = 2,000 – 1,200$

Therefore $x_1 = 800$

Step 5*:* Substitute the co-ordinates of corner points into the objective function.

Maximize ‘Z’ $= 3x_1 + 5x_2$

At ‘O’, Z = 0 + 0 = 0

At ‘A’, Z = 3 (1,500) + 5 (0) = 4,500

At ‘B’, Z = 3 (1,000) + 5 (500) = 5,500

At ‘C’, Z = 3 (800) + 5 (600) = 5,400

At ‘C’, Z = 3 (0) + 5 (600) = 3,000

<u>Result</u>

A maximum profit of ` 5,500 can be earned by producing 1,000 dolls of basic version and 500 dolls of deluxe version.

Slack and Surplus Variables

To convert an equation of the form

$\mathbf{a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{1n}x_n \leq b_i}$ to standard form

we introduce a slack variable y_i to obtain

$$\mathbf{a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{1n}x_n + y_i = b_i.}$$

To convert an equation of the form

$\mathbf{a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{1n}x_n \geq b_i}$ to standard form

we introduce a surplus variable y_i to obtain

$$\mathbf{a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{1n}x_n - y_i = b_i.}$$

Eg

Maximise 'Z' = $4x1 + 3x2$

[Subject to constraints]

$2x1 + x2 \leq 30$

$x1 + x2 \geq 24$

Where, $x1, x2 \geq 0$ [Non-negativity constraints]

Standard Form

Convert the inequalities into equalities adding slack and surplus variables.

Maximise 'Z' = $4x1 + 3x2$

[Subject to constraints]

$2x1 + x2 + x3 = 30$

$x1 + x2 - x4 = 24$

Where, $x1, x2, x3, x4 \geq 0$ [Non-negativity constraints]

Simplex Method of Linear Programming

Steps:

i) Convert the inequalities into equalities by adding slack variables, surplus variables or artificial variables, as the case may be.

ii) Identify the coefficient of equalities and put them into a matrix form AX = B

Where "A" represents a matrix of coefficient, "X" represents a vector of unknown quantities and B represents a vector of constants, leads to AX = B [This is according to system of equations].

iii) Tabulate the data into the first iteration of Simplex Method.

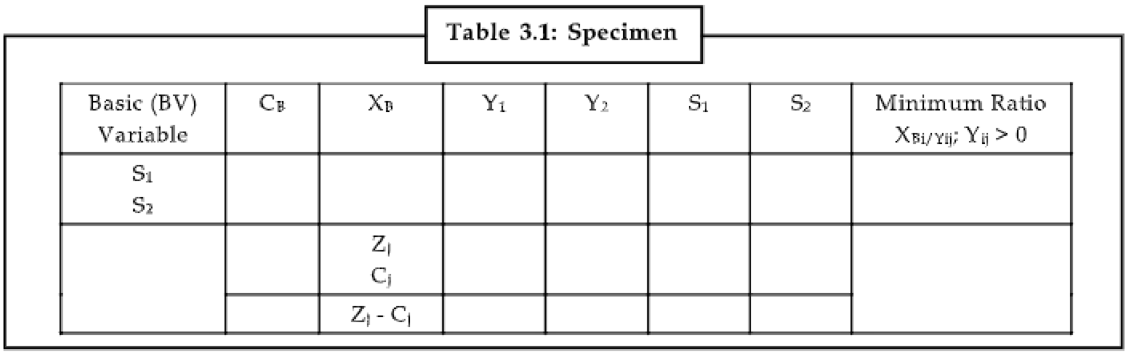


Table 3.1: Specimen

Basic (BV) Variable	C_B	X_B	Y_1	Y_2	S_1	S_2	Minimum Ratio X_{Bi}/Y_{ij}; $Y_{ij} > 0$
S_1 S_2							
		Z_j C_j					
		$Z_j - C_j$					

(a) Cj is the coefficient of unknown quantities in the objective function.

$Z_j = \square C_{Bi}Y_{ij}$ (Multiples and additions of coefficients in the table, i.e., $C_{B1} \times Y_{11} + C_{B2} \times Y_{12}$)

(b) Identify the Key or Pivotal column with the minimum element of Zj - Cj denoted as 'KC' throughout to the problems in the chapter.

(c) Find the 'Minimum Ratio' i.e., X_{Bi}/Y_{ij}.

(d) Identify the key row with the minimum element in a minimum ratio column. Key row is denoted as 'KP'.

(e) Identify the key element at the intersecting point of key column and key row, which is put into a box □ throughout to the problems in the chapter.

iv) Reinstate the entries to the next iteration of the simplex method.

a) The pivotal or key row is to be adjusted by making the key element as '1' and dividing the other elements in the row by the same number.

b) The key column must be adjusted such that the other elements other than key elements should be made zero.

c) The same multiple should be used to other elements in the row to adjust the rest of the elements. But, the adjusted key row elements should be used for deducting out of the earlier iteration row

d) The same iteration is continued until the values of $Z_j - C_j$ become either '0' or positive.

v) Find the 'Z' value given by C_B, X_B.

Eg

Solve by Simplex method.

Maximise $'Z' = 5x_1 + 3x_2$ [Subject to constraints]

$$x_1 + x_2 \leq 2$$

$$5x_1 + 2x_2 \leq 10$$

$$3x_1 + 8x_2 \leq 12$$

Where, $x_1, x_2 \geq 0$ [Non-negativity constraints]

Solution:

Step 1: Conversion of inequalities into equalities adding slack variables

$$x_1 + x_2 + x_3 = 2$$

$$5x_1 + 2x_2 + x_4 = 10$$

$$3x_1 + 8x_2 + x_5 = 12$$

Where, x_3, x_4 and x_5 are slack variables.

Step 2: Fit the data into the matrix form AX = B

$$A = \begin{pmatrix} Y_1 & Y_2 & S_1 & S_2 & S_3 \\ x_1 & x_2 & x_3 & x_4 & x_5 \\ 1 & 1 & 1 & 0 & 0 \\ 5 & 2 & 0 & 1 & 0 \\ 3 & 8 & 0 & 0 & 1 \end{pmatrix} \quad X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = B = \begin{pmatrix} 2 \\ 10 \\ 12 \end{pmatrix}$$

Step 3: Fit the data into first iteration of Simplex Method

BV	C_B	X_B	Y_1	Y_2	S_1	S_2	S_3	Min. Ratio
S_1	0	2	$\boxed{1}$	1	1	0	0	2/1 = 2(KR) →
S_2	0	10	5	2	0	1	0	10/5 = 2
S_3	0	12	3	8	0	0	1	12/3 = 4
		Z_j	0	0				
		C_j	5	3				
		$Z_j - C_j$	-5	-3				
			(↑ KC)					

Therefore, $Z = C_B X_B$

$$= (0\times2) + (0\times10) + (0\times12)$$

$$= 0$$

Step 4: Fit the data into second iteration of Simplex Method.

BV	C_B	X_B	Y_1	Y_2	S_1	S_2	S_3	Min. Ratio
Y_1	5	$2/1 = 2$	$1/1 = 1$	$1/1 = 1$	–	–	–	–
S_2	0	$10 - 2(5) = 0$	$5 - 1(5) = 0$	$2-1(5) = -3$	–	–	–	–
S_3	0	$12 - 2(3) = 6$	$3 - 1(3) = 0$	$8-1(3) = 5$	–	–	–	–
		Z_j	5	5				
		C_j	5	3				
		$Z_j - C_j$	0	2				

Therefore, $Z = C_B X_B$

$= (5 \times 2) + (0 \times 0) + (0 \times 6)$

Therefore, $Z = 10$

Therefore, Maximum value of 'Z' = 10

Two Phase Method

In the preceding section we observed that it was frequently necessary to add artificial variables to the constraints to obtain an initial basic feasible solution to an L.P. problem. If the problem is to be solved, the artificial variables must be driven to zero. The two-phase method is another method to handle these artificial variables. Here the L.P. problem is solved in two phases.

Phase I

In this phase we find an *i.b.f.s.* to the original problem. For this all artificial variables are to be driven to zero. To do this an *artificial objective function* (w) is created which is the sum of all the artificial variables. This *new* objective function is then *minimized,* subject to the constraints of the given (original) problem, using the simplex method. At the end of phase I, three cases arise:

1. If the minimum value of $w > 0$, and at least one artificial variable appears in the basis at a positive level, then the given problem has no feasible solution and the procedure terminates.
2. If the minimum value of $w = 0$, and no artificial variable appears in the basis, then a basic feasible solution to the given problem is obtained. The artificial variable column (s) is/are deleted for phase II computations.
3. If the minimum value of $w = 0$ and one or more artificial variables appear in the basis at zero level, then a feasible solution to the original problem is obtained. However, we must take care of this artificial variable and see that it never becomes positive during phase II computations. Zero cost coefficient is assigned to this artificial variable and it is retained in the initial table of phase II. If this variable remains in the basis at zero level in all phase II computations, there is no problem. However, the problem arises if it becomes positive in some iteration. In such a case, a slightly different approach is adopted in selecting the outgoing variable. The lowest non-negative replacement ratio criterion is not adopted to find the outgoing variable. Artificial variable (or one of the artificial variables if there are more than one) is selected as the outgoing variable. The simplex method can then be applied as usual to obtain the optimal basic feasible solution to the given L.P. problem.

Phase II

When phase I results in (2) or (3), we go on to phase II to find optimum solution to the given L.P. problem. The basic feasible solution found at the end of phase I is now used as a starting solution for the original problem. In other words, the final table of phase I becomes the starting table of phase II in which the *artificial (auxiliary) objective function* is replaced by the original objective function. The simplex method is then applied to arrive at the optimum solution. Artificial variables which do not appear in the basis may be deleted.

Remarks: 1. In phase I, the iterations are stopped as soon as the value of the new (artificial) objective function becomes zero because this is its minimum value. There is no need to continue till the optimality is reached if this value becomes zero earlier than that.

2. Note that the new objective function is always of minimization type regardless of whether the original problem is of maximization or minimization type.

Eg

Use the two-phase simplex method to

maximize $Z = 5x_1 + 3x_2,$

subject to the constraints $2x_1 + x_2 \leq 1,$

$x_1 + 4x_2 \geq 6,$

$x_1, x_2, \geq 0.$

Solution

Phase I

It consists of the following steps:

Step 1. Set up the Problem in the Standard Form

The original objective function $Z = 5x_1 + 3x_2$ is temporarily set aside during the phase I solution. The given constraints, after the introduction of slack, surplus and artificial variables take the form :

$$2x_1 + x_2 + s_1 = 1,$$
$$x_1 + 4x_2 - s_2 + A_1 = 6,$$
$$x_1, x_2, s_1, s_2, A_1 \geq 0.$$

The new objective function is

$$\text{minimize } w = A_1.$$

Now the simplex method requires that a variable which appears in one equation must appear in all the equations. This is done by proper placement of a zero coefficient. Thus the problem for phase I in standard form becomes

$$\begin{aligned} \text{minimize } w &= 0x_1 + 0x_2 + 0s_1 + 0s_2 + A_1, \\ \text{subject to } \quad & 2x_1 + x_2 + s_1 + 0s_2 + 0A_1 = 1, \\ & x_1 + 4x_2 + 0s_1 - s_2 + A_1 = 6, \\ & x_1, x_2, s_1, s_2, A_1 \geq 0. \end{aligned}$$

Step 2. Find an Initial Basic Feasible Solution

Substituting $x_1 = x_2 = s_2 = 0$ in the constraint equations we get $s_1 = 1$, $A_1 = 6$ as the initial basic feasible solution.

	c_j	0	0	0	0	1			
c_B	Basis	x_1	x_2	s_2	s_1	A_1	b	θ	
0	s_1	2	(1)	0	1	0	1	1	←
1	A_1	1	4	– 1	0	1	6	3/2	
	$Z_j = \Sigma c_B a_{ij}$	1	4	– 1	0	1	6		
	$c_j – Z_j$	– 1	– 4	1	0	0			
			↑			*Initial b.f.s. for phase I problem*			

Step 3. Perform Optimality Test

Since $c_j – Z_j$ is negative under some columns (minimization problem), table is not optimal.

Step 4. Iterate Towards an Optimal Solution

x_2 is incoming variable, s_1 is outgoing variable and (1) is the key element. In table , s_1 is replaced by x_2.

	c_j	0	0	0	0	1	
c_B	Basis	x_1	x_2	s_2	s_1	A_1	b
0	x_2	2	1	0	1	0	1
1	A_1	– 7	0	– 1	– 4	1	2
	$Z_j = \Sigma c_B a_{ij}$	– 7	0	– 1	– 4	1	2
	$c_j – Z_j$	7	0	1	4	0	
				Optimal basic feasible solution for phase I problem			

$\because$ $c_j – Z_j$ is either positive or zero under all columns, an optimal basic feasible solution to the auxiliary L.P.P. has been obtained.

However, since $w = A_1 = 2$ (> 0) and artificial variable A_1 appears in the basis at a positive level ($A_1 = 2$), the given problem *does not possess a feasible solution* and the procedure stops.

REVIEW QUESTIONS

1) Define slack and surplus variable.
2) What is optimum basic feasible solution.
3) Write the algorithm for graphical method.
4) Write mathematical formulation of LPP.
5) Write algorithm for Simplex method.
6) Define non negativity constraints.

UNIT – V

TRANSPORTATION PROBLEM(TP)

- **Transportation Table**
- **Solution of Transportation Problem**
- **Testing for Optimality**
- **Assignment Problem**
- **The Assignment Method**
- **Special Cases in Assignment Problems**

TRANSPORTATION PROBLEM(TP)

Transportation problem is a particular class of linear programming, which is associated with day-to-day activities in our real life and mainly deals with logistics. It helps in solving problems on distribution and transportation of resources from one place to another. The goods are transported from a set of sources (e.g., factory) to a set of destinations (e.g., warehouse) to meet the specific requirements. In other words, transportation problems deal with the transportation of a product manufactured at different plants (supply origins) to a number of different warehouses (demand destinations). The objective is to satisfy the demand at destinations from the supply constraints at the minimum transportation cost possible. To achieve this objective, we must know the quantity of available supplies and the quantities demanded.

Mathematical Formulation of TP

The transportation problem applies to situations where a single commodity is to be transported from various sources of supply **(origins)** to various demands **(destinations).**

Let there be m sources of supply S_1, S_2, ………………S_m having a_i (i = 1, 2,......m) units of supplies respectively to be transported among n destinations D_1, D_2 ………D_n with b_j (j = 1,2…..n) units of requirements respectively. Let C_{ij} be the cost for shipping one unit of the commodity from source i, to destination j for each route. If x_{ij} represents the units shipped per route from source i, to destination j, then the problem is to determine the transportation schedule which minimizes the total transportation cost of satisfying supply and demand conditions.

The transportation problem can be stated mathematically as a linear programming problem as below:

Minimize $$Z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

Subject to constraints,

$$\sum_{j=1}^{n} x_{ij} = a_{i,} \quad i = 1,2,\ldots..m \text{ (supply constraints)}$$

$$\sum_{i=1}^{n} x_{ij} = b_{j,} \quad j = 1,2,\ldots..m \text{ (demand constraints)}$$

and x_{ij} ³ 0 for all $\quad i = 1,2,\ldots..m$ *and,*

$\qquad j = 1,2,\ldots..m$

Transportation Table

The Transportation problem can also be represented in a tabular form

Let c_{ij} be the cost of transporting a unit of the product from ith origin to jth destination.

a_i be the quantity of the commodity available at source i,

b_j be the quantity of the commodity needed at destination j, and

x_{ij} be the quantity transported from ith source to jth destination

Tabular Representation of Transportation Model

		To			Supply
	D_1	D_2	...	D_3	
From					a_1
S_1	C_{11} X_{11}	C_{12} X_{12}	...	C_{1n}	a_1
S_2	C_{21} X_{21}	C_{22} X_{22}	...	C_{2n}	a_2
. . .	. . .	. . .	... ...	. . .	. . .
S_m	C_{m1} x_{m1}	C_{m2} x_{m2}	...	C_{mn}	a_m
b_j Demand	b_1	b_2	...	b_n	$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$$

If the total supply is equal to total demand, then the given transportation problem is a balanced One.

Initial Basic Feasible Solution

Step 1: Formulate the Problem

Formulate the given problem and set up in a matrix form. Check whether the problem is a balanced or unbalanced transportation problem. If unbalanced, add dummy source (row) or dummy destination (column) as required.

Step 2: Obtain the Initial Feasible Solution

The initial feasible solution can be obtained by any of the following three methods.

1. Northwest Corner Method (NWC)
2. Row and Column Minima Method (RCMM)
3. Vogel's Approximation Method (VAM)

The transportation cost of the initial basic feasible solution through Vogel's approximation method, VAM will be the least when compared to the other two methods which gives the value nearer to the optimal solution or optimal solution itself. Algorithms for all the three methods to find the initial basic feasible solution are given.

Northwest Corner Method(NWC)

1. Select the North-west (i.e., upper left) corner cell of the table and allocate the maximum possible units between the supply and demand requirements. During allocation, the transportation cost is completely discarded (not taken into consideration).
2. Delete that row or column which has no values (fully exhausted) for supply or demand.
3. Now, with the new reduced table, again select the North-west corner cell and allocate the available values.
4. Repeat steps (2) and (3) until all the supply and demand values are zero.
5. Obtain the initial basic feasible solution.

<u>Eg</u>

Find Initial Basic Feasible Solution to the following TP by NWC method.

	Retail shops				
Factories	**1**	**2**	**3**	**4**	**Supply**
1	3	5	7	6	50
2	2	5	8	2	75
3	3	6	9	2	25
Demand	20	20	50	60	

Solution:

	Retail shops				
Factories	**1**	**2**	**3**	**4**	**Supply**
1	3 ^{20}	5 ^{20}	7 ^{10}	6	~~50~~
2	2	5	8 ^{40}	2 ^{35}	~~75~~
3	3	6	9	2 ^{25}	~~25~~
Demand	~~20~~	~~20~~	~~50~~	~~60~~	

As under the process of NWC method, we allocate $x_{11} = 20$. Now demand for the first column is satisfied, therefore, eliminate that column.

Proceeding in this way, we observe that

$$x_{12} = 20, x_{13} = 10, x_{23} = 40, x_{24} = 35, x_{34} = 25.$$

Delete the row if supply is exhausted.

Delete the column if demand is satisfied.

Here, number of retail shops (n) = 4, and

Number of factories (m) = 3

Number of basic variables = m + n – 1 = 3 + 4 – 1 = 6.

Initial basic feasible solution:

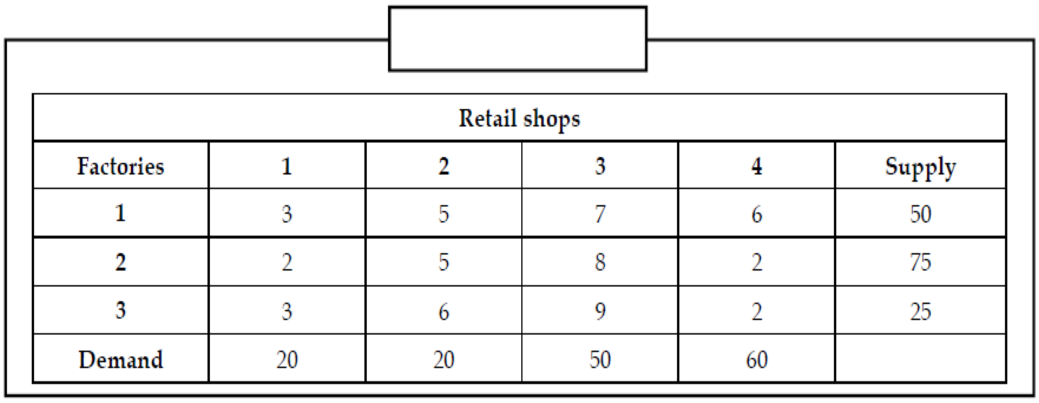
$$20 \times 3 + 20 \times 5 + 10 \times 7 + 40 \times 8 + 35 \times 2 + 25 \times 2 = 670$$

Row and Column Minima Method (RCMM)

1. Select the smallest transportation cost cell available in the entire table and allocate the supply and demand.
2. Delete that row/column which has exhausted. The deleted row/column must not be considered for further allocation.
3. Again select the smallest cost cell in the existing table and allocate. (*Note:* In case, if there are more than one smallest costs, select the cells where maximum allocation can be made)
4. Obtain the initial basic feasible solution.

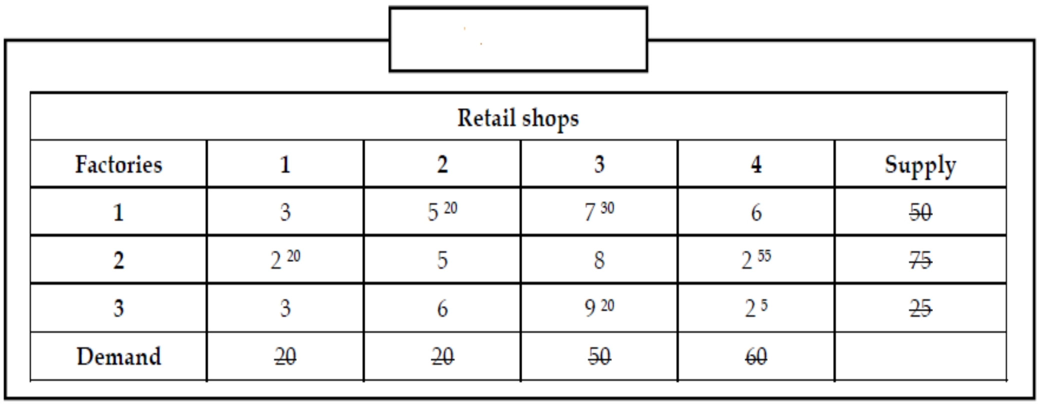
Eg

Find Initial Basic Feasible Solution to the following TP by RCMM method.



	Retail shops				
Factories	**1**	**2**	**3**	**4**	**Supply**
1	3	5	7	6	50
2	2	5	8	2	75
3	3	6	9	2	25
Demand	20	20	50	60	

Applying the least cost method-



	Retail shops				
Factories	**1**	**2**	**3**	**4**	**Supply**
1	3	5^{20}	7^{30}	6	~~50~~
2	2^{20}	5	8	2^{55}	~~75~~
3	3	6	9^{20}	2^{5}	~~25~~
Demand	~~20~~	~~20~~	~~50~~	~~60~~	

We observe that $c_{21} = 2$, which is the minimum transportation cost. So, $x_{21} = 20$.

Proceeding in this way, we observe that $x_{24} = 55$, $x_{34} = 5$, $x_{12} = 20$, $x_{13} = 30$, $x_{33} = 20$.

Number of basic variables = $m + n - 1 = 3 + 4 - 1 = 6$.

The initial basic feasible solution:

$$= 20 \times 2 + 55 \times 2 + 5 \times 2 + 20 \times 5 + 30 \times 7 + 20 \times 9$$

$$= 650.$$

Vogel's Approximation Method (VAM)

1. Calculate penalties for each row and column by taking the difference between the smallest cost and next highest cost available in that row/column. If there are two smallest costs, then the penalty is zero.
2. Select the row/column, which has the largest penalty and make allocation in the cell having the least cost in the selected row/column. If two or more equal penalties exist, select one where a row/column contains minimum unit cost. If there is again a tie, select one where maximum allocation can be made.
3. Delete the row/column, which has satisfied the supply and demand.
4. Repeat steps (1) and (2) until the entire supply and demands are satisfied.
5. Obtain the initial basic feasible solution.

Eg

Find Initial Basic Feasible Solution to the following TP by VAM method.

	Destination				
Origin	**1**	**2**	**3**	**4**	**Supply**
1	20	22	17	4	120
2	24	37	9	7	75
3	34	37	20	15	25
Demand	60	40	30	110	240

Solution:

Solving the problem through Vogel's Approximation Method, we get the Table 5.18

	Destination					
Origin	**1**	**2**	**3**	**4**	**Supply**	**Penalty**
1	20	22^{40}	17	4	~~120~~ 80	13
2	24	37	9	7	70	2
3	32	37	20	15	50	5
Demand	60	~~40~~	30	110	240	
Penalty	4	15	8	3		

The highest penalty occurs in the second column. The minimum c_{ij} in this column is c_{12} (i.e., 22). Hence, $x_{12} = 40$ and the second column is eliminated.

Now again calculate the penalty.

Origin	1	2	3	4	Supply	Penalty
1	20	22^{40}	17	4^{80}	~~120~~	13
2	24	37	9	7	70	2
3	32	37	20	15	50	5
Demand	60	~~40~~	30	110	240	
Penalty	4		8	3		

The highest penalty occurs in the first row. The minimum cij in this row is c_{14} (i.e., 4). So $x_{14} = 80$ and the first row is eliminated.

Final table:

Now assuming that you can calculate the values yourself, we reach the final table as in Table 5.20

Destination											
Origin	**1**	**2**	**3**	**4**	**Supply**	**Penalty**					
1	20	**22** ^{40}	17	**4** ^{80}	~~120~~	3	13	-	-	-	-
2	**24** 10	37	**9** ^{30}	**7** ^{30}	~~70~~	2	2	2	17	24	24
3	**32** 50	37	20	15	~~50~~	5	5	5	17	32	-
Demand	~~60~~	~~40~~	~~30~~	~~110~~	240						
Penalty	4	15	8	3							
	4	-	8	3							
	8	-	11	8							
	8	-	-	8							
	8	-	-	-							
	24	-	-	-							

The initial basic feasible solution:

$$= 22 \times 40 + 4 \times 80 + 24 \times 10 + 9 \times 30 + 7 \times 30 + 32 \times 50$$

$$= 3520$$

Test for Optimality

Stepping Stone Method

It is a method for computing optimum solution of a transportation problem.

Steps Involved:

Step 1: Determine an initial basic feasible solution using any one of the following:

(a) North West Corner Rule

(b) Matrix Minimum Method

(c) Vogel Approximation Method

Step 2: Make sure that the number of occupied cells is exactly equal to m+n-1, where m is the number of rows and n is the number of columns.

Step 3: Select an unoccupied cell.

Step 4: Beginning at this cell, trace a closed path using the most direct route through at least three occupied cells used in a solution and then back to the original occupied cell and moving with only horizontal and vertical moves. The cells at the turning points are called "Stepping Stones" on the path.

Step 5: Assign plus (+) and minus (-) signs alternatively on each corner cell of the closed path just traced, starting with the plus sign at unoccupied cell to be evaluated.

Step 6: Compute the net change in the cost along the closed path by adding together the unit cost figures found in each cell containing a plus sign and then subtracting the unit costs in each square containing the minus sign.

Step 7: Check the sign of each of the net changes. If all the net changes computed are greater than or equal to zero, an optimum solution has been reached. If not, it is possible to improve the current solution and decrease the total transportation cost.

Step 8: Select the unoccupied cell having the most negative net cost change and determine the maximum number of units that can be assigned to a cell marked with a minus sign on the closed path corresponding to this cell. Add this number to the unoccupied cell and to all other cells on the path marked with a plus sign. Subtract this number from cells on the closed path marked with a minus sign.

Step 9: Repeat the procedure until you get an optimum solution

Eg

Consider the Following TP. Find the Optimum solution.

Factory	**D**	**E**	**F**	**G**	**Capacity**
A	4	6	8	6	700
B	3	5	2	5	400
C	3	9	6	5	600
Requirement	400	450	350	500	1700

Solution: First, we find out an initial basic feasible solution by Matrix Minimum Method

Factory	**D**	**E**	**F**	**G**	**Capacity**
A	4	6^{450}	8	6^{250}	700
B	3^{50}	5	2^{350}	5	400
C	3^{350}	9	6	5^{250}	600
Requirement	400	450	350	500	1700

Here, m + n – 1 = 6. So the solution is not degenerate.

The cell AD (4) is empty so allocate one unit to it. Now draw a closed path from AD.

Factory	**D**	**E**	**F**	**G**	**Capacity**
A	4^{+1}	6^{450}	8	6^{249}	700
B	3^{50}	5	2^{350}	5	400
C	3^{349}	9	6	5^{251}	600
Requirement	400	450	350	500	1700

The increase in the transportation cost per unit quantity of reallocation is + 4 – 6 + 5 – 3 = 0.

This indicates that every unit allocated to route AD will neither increase nor decrease the transportation cost. Thus, such a reallocation is unnecessary.

Choose another unoccupied cell. The cell BE is empty so allocate one unit to it.

Factory	**D**	**E**	**F**	**G**	**Capacity**
A	4	6^{449}	8	6^{251}	700
B	3^{49}	5^{+1}	2^{350}	5	400
C	3^{351}	9	6	5^{249}	600
Requirement	400	450	350	500	1700

The increase in the transportation cost per unit quantity of reallocation is + 5 – 6 + 6 – 5 + 3 – 3 = 0

This indicates that every unit allocated to route BE will neither increase nor decrease the transportation cost. Thus, such a reallocation is unnecessary.

The allocations for other unoccupied cells are:

Unoccupied cells	**Increase in cost per unit of reallocation**	**Remarks**
CE	+ 9 – 6 + 6 – 5 = 4	Cost Increases
CF	+ 6 – 3 + 3 – 2 = 4	Cost Increases
AF	+ 8 – 6 + 5 – 3 + 3 – 2 = 5	Cost Increases
BG	+ 5 – 5 + 3 – 3 = 0	Neither increase nor decrease

Since all the values of unoccupied cells are greater than or equal to zero, the solution obtained is optimum.

Minimum transportation cost is:

$$6 \times 450 + 6 \times 250 + 3 \times 250 + 2 \times 250 + 3 \times 350 + 5 \times 250 = ₹\,7350$$

Assignment Problem

L.P is used in solving problems faced in assigning the 'equal number of jobs to equal number of workers so as to maximise profit or minimize cost'. Hence it is called one-to-one assignment. Say for instance, there are 'n' jobs to be performed and 'n' number of persons are available for doing these jobs and each person can do one job at a time though with varying degree of efficiency. Say let C_{ij} be the total cost : here C=cost, I = individual and j = job. So, a problem arises

as to which worker is to be assigned which job as to minimize the total job cost. The simple matrix would go like this:

Cost Matrix Job

Individual		**1**	**2**	**J**	**n**
	1	C_{11}	C_{12}	C_{1j}	C_{1n}
	2	C_{21}	C_{22}	C_{2j}	C_{2n}
	i	C_{i1}	C_{i2}	C_{ij}	C_{in}
	n	C_{n1}	C_{n2}	C_{nj}	C_{nn}

Although these types of problems could be solved by using transportation algorithm but a more efficient method called the assignment algorithm is used to solve such typical problems.

$C_{ij} \rightarrow$ indicates the cost of assigning i^th job to j^th individual

$X_{ij} \rightarrow$ (reference index) which indicates whether i^th job is assigned to j^th person or not.

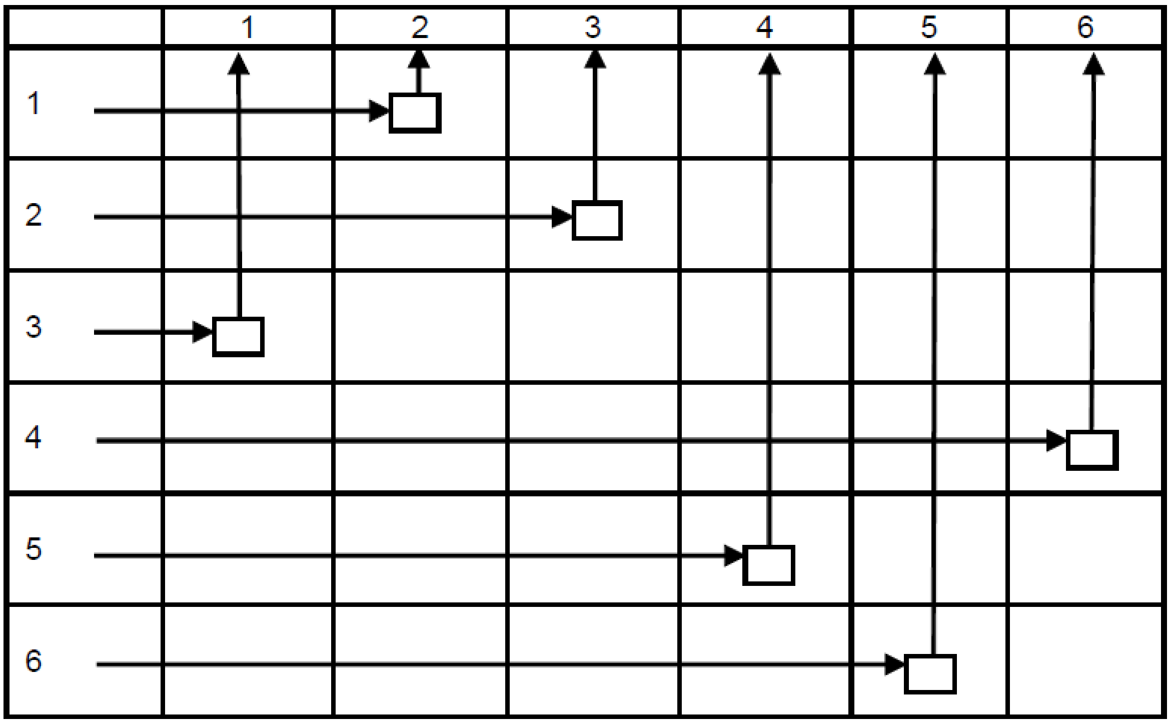
X_{ij} = 1 if i^th job are assigned to j^th person.

0 if i^th job are assigned to j^th person.

Consider the example:

There are 6 persons and 6 jobs to be allotted.

Let the assignment of jobs be as shown below:



The assignments are $1 \to 2, 2 \to 3, 3 \to 1, 4 \to 6, 5 \to 4, 6 \to 5$.

Here, $x_{11} = 0$ since the first job is not allotted to first person.

$X_{12} = 1$ since the first job is allotted to second person.

Similarly, $x_{21} = x_{22} = x_{32} = x_{34} = 0$, and

$x_{31} = x_{46} = x_{54} = x_{65} = 1$

Sum of all jobs assigned to first person = $\sum_{i=1}^{6} X_{i1} = 1$

Since only one job can be allotted to a person.

$\sum_{j=1}^{6} X_{ij}$ = Sum of all persons with first job = 1, since a job can be assigned to only one person.

In general, sum of all jobs assigned to jth person = 1

i.e., $\sum_{i=j}^{n} X_{ij} = 1$

And sum of all persons with ith job = 1

i.e., $\sum_{i=j}^{n} X_{ij} = 1$

And initial basic feasible solution can be found out by following:

1. Reduction Theorem
2. Hungarian Approach

Similarly, many real life problems can be solved such as assigning number of classes, for number of rooms, number of drivers to number of trucks or vice versa, number of teachers to number of classes, etc.

Reduction Theorem can be used for solving assignment problems with an objective of minimization of costs. For such maximization assignment problems, commonly used rules are:

1. Blind fold assignment/assignment by intuition.
2. Converting the maximization problem into minimization by considering the largest element in the whole matrix.
3. Converting the maximization problem into minimization by using negative signs for all the elements in the profit matrix.

Types of Assignment Problem

The assignment problems are of two types. It can be either

(i) Balanced or

(ii) Unbalanced.

If the number of rows is equal to the number of columns or if the given problem is a square matrix, the problem is termed as a *balanced assignment problem*. If the given problem is not a square matrix, the problem is termed as an *unbalanced assignment problem*.

If the problem is an unbalanced one, add dummy rows / dummy columns as required so that the matrix becomes a square matrix or a balanced one. The cost or time values for the dummy cells are assumed as zero.

Mathematical Formulation of AP

Minimize the total cost which is given by,

$$Z = \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} X_{ij}$$

Where, $i = 1, 2, 3, \ldots\ldots\ldots n$

$j = 1, 2, 3, \ldots\ldots\ldots n$

Subject to restriction

$X_{ij} = 1$ (One job is done by one worker)

$= 0$ (No job is assigned)

$\Sigma X_{ij} = 1$ (only one job be assigned to one person)
Where, $j = 1, 2, 3, \ldots\ldots\ldots n$

$\Sigma X_{ij} = 1$ (only one person can do one job at a time)

Where, $i = 1, 2, 3, \ldots\ldots\ldots n$

$C_{ij} \rightarrow$ indicates the cost of assigning i^{th} job to j^{th} individual or vice versa, $Vi_{ij} = 1$ to n.

$X_{ij} \rightarrow$ indicates whether i^{th} job is assigned to j^{th} person or not.

$X_{ij} = 1$ if ith job is assigned to jth person '0' otherwise.

Theorem Statement

It states that in an assignment problem if we add or subtract a constant to every element of any row or column of the cost matrix (C_{ij}), then an assignment that minimizes the total cost on one matrix will also minimize the total cost on the other matrix.

Proof:

Let $X_{ij} = X_{ij}$ $\quad$ X_{ij} = elements of first cost matrix.

X_{ij} = elements of second cost matrix.

$$Z = \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} X_{ij} : X_{ij} \geq 0$$

$$Z = \sum_{i=1}^{n} \sum_{j=1}^{n} \left(C_{ij} \pm X_{ij} \pm V_j\right) X_{ij}$$

$X_{ij} = i = 1.2.3……………n$

$j = 1.2.3……………n$

$U_i \rightarrow$ ith row constant taken for reduction.

$V_j \rightarrow$ jth column constant taken for reduction.

Where, U_i and V_j are considered to be constant.

$$Z_1 = \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} X_{ij} - \sum U_i \sum_{i-1}^{n} X_{ij} - \sum_{j=1}^{n} V_j \sum_{j=1}^{n} X_{ij}$$

$$Z_1 = Z - \sum_{i-1}^{n} U_i - \sum_{j=1}^{n} V_j$$

$$\sum_{j=1}^{n} X_{ij} = \sum_{i=1}^{n} x_{ij} = 1$$

Since, terms that are subtracted from 'Z' to give 'Z_1' are independent of x_{ij}, it follows that 'Z' is minimized whenever 'Z_1' is minimized and it can be proved conversely.

Eg

Find the minimum cost for the following problem:

	Persons			
	I	**II**	**III**	**IV**
A	10	12	19	11
B	5	10	7	8
C	12	14	13	11
D	8	15	11	9

Tasks

(i) Row-wise reduction

	I	II	III	IV
A	0	2	9	1
B	0	5	2	3
C	1	3	2	0
D	0	7	3	1

(ii) Column-wise reduction

	I	II	III	IV
A	0	$\boxed{0}$	7	1
B	0	3	$\boxed{0}$	3
C	1	1	0	0
D	$\boxed{0}$	5	1	1

Hence, the assignment is

A	→	II	12
B	→	III	07
C	→	IV	11
D	→	I	08
			₹ 38

Here minimized cost $= \Sigma\Sigma\, C_{ij}X_{ij}$ = ₹ 38

Special Case in Assignment Problems

Maximization Case in Assignment Problem

In maximization problem, the objective is to maximize profit, revenue, etc. Such problems can be solved by converting the given maximization problem into a minimization problem.

1. Change the signs of all values given in the table or another method is,
2. Select the highest element in the entire assignment table and subtract all the elements of the table from the highest element.

Alpha Corporation has 4 plants, each of which can manufacture any one of the 4 products. Production cost differs from one plant to another plant, so also the sales revenue. Given the revenue and the cost data below, obtain which product each plant should produce to maximize the profit.

Sales revenue (₹ in 000s)

Product		1	2	3	4
Plant	A	50	68	49	62
	B	60	70	51	74
	C	55	67	53	70
	D	58	65	54	69

Production Cost (₹ in' 000s)

Product		1	2	3	4
Plant	A	49	60	45	61
	B	55	63	45	69
	C	52	62	49	68
	D	55	64	48	66

Solution:

Step 1: Determination of profit matrix.

Product		1	2	3	4
Plant	A	1	8	4	1
	B	5	7	6	5
	C	3	5	4	2
	D	3	1	6	3

Step 2: Conversion of profit matrix into cost matrix.

Product		**1**	**2**	**3**	**4**
Plant	A	7	0	4	7
	B	3	1	2	3
	C	5	3	4	6
	D	5	7	2	5

Using Reduction Rules

Step 3: Row-wise reduction of the matrix.

Product		**1**	**2**	**3**	**4**
Plant	A	7	0	4	7
	B	2	0	1	2
	C	2	0	1	3
	D	3	5	0	3

Step 4: Column-wise reduction of the matrix.

Product		**1**	**2**	**3**	**4**
Plant	A	5	0	4	5
	B	0	0	1	0
	C	0	0	1	1
	D	1	5	0	1

Step 5: Trial Assignment.

Product		**1**	**2**	**3**	**4**
Plant	A	5	$\boxed{0}$	4	5
	B	0	0	1	$\boxed{0}$
	C	$\boxed{0}$	0	1	1
	D	1	5	$\boxed{0}$	1

Step 6: Determination of profit associated with the assignment.

Plant	**Product**	**Total Profit (Rs.)**
A	2	8,000
B	4	5,000
C	1	3,000
D	3	6,000
	Total Profit	22,000

REVIEW QUESTIONS

1) Define assignment problem.
2) Define IBFS.
3) Write algorithm for VAM method.
4) Write the mathematical formulation of TP.
5) What is meant by unbalanced TP.
6) Write the steps for assignment problem.