Binary Search Tree

Paper Name : *Data Structure and Algorithm*
Paper Code : *CS*302

Department of Computer science & Engineering
Siliguri Institute of Technology

September 4, 2019

Outlines

- Definition
- Representation
- Operation
- Complexity
- Application

Definition

- Definition : A Binary search Tree is a Binary Tree in which every node value grater than of its right child and less then of its left child.
- Example :

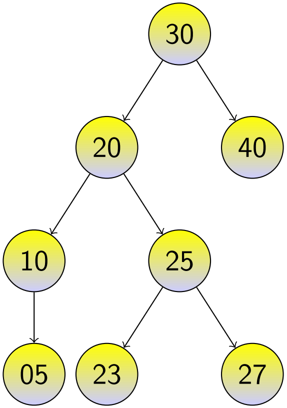


Figure: Example of Binary Search Tree.

Representation

A Binary search Tree can represent in two way: array representation and Linked list representation

- Array Representation:

30	20	40	10	25	-	-	5	-	23	27

Table: Array Representation of BST

- Linked list Representation:

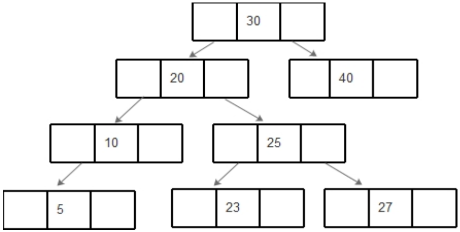


Figure: linked list Representation

Operation of BST

In BST there are four basic operation: Traversal, searching, Insert a node, Delete a node

- Traversal:
- Searching:
- Insert a node:
- Delete a node:

Traversal

Traversal means visiting each node exactly once.

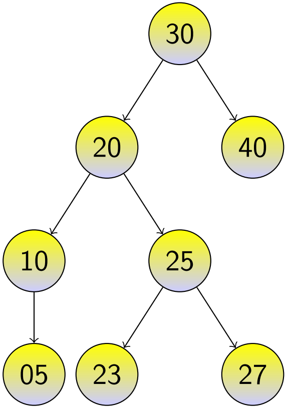


Figure: Example of Binary Search Tree.

- In Order Traversal:The traversing sequence is 5,10,20,23,25,27,30,40
- Pre Order Traversal:The traversing sequence is

Search a node from BST

Search a node from a Tree means the desired node exist or not in a BST. Two way to search a node : recursive way and Non recursive way.

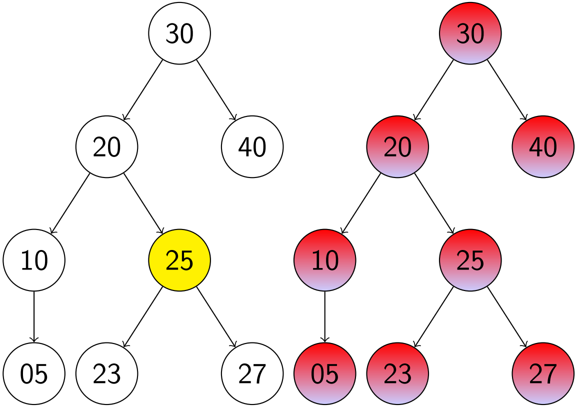


Figure: *SearchNode*25*and*21

Algorithm for Searching

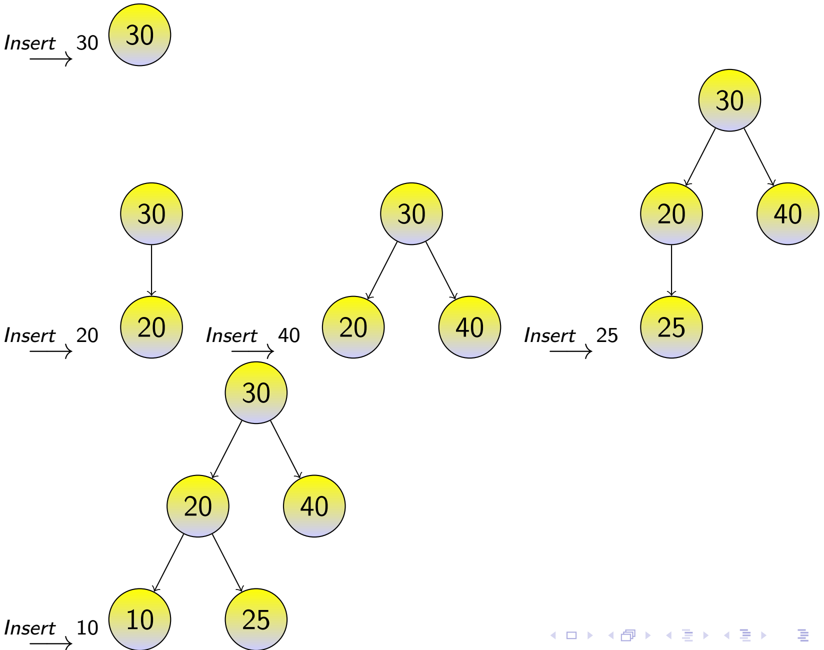
Algorithm 1 Recursive Search algorithm

INPUT: Binary search Tree (T) and current node i.e present node PN under scanning . Searched item/key is K

OUTPUT: KEY ELEMENT FOUND if the function return 1 i.e K in T other wise KEY ELEMENT NOT FOUND the function return 0.

```
Recursive Searching($TNode * PN$,$K$)
**if** $PN == NULL$ **then**
    Return 0
**else if** $K == PN \rightarrow Data$ **then**
    Return 1
**else if** $K \leq PN \rightarrow Data$ **then**
    Return Recursive Searching($PN \rightarrow Lchild, K$)
**else**
    Return Recursive Searching($PN \rightarrow Rchild, K$)
**end if**
```

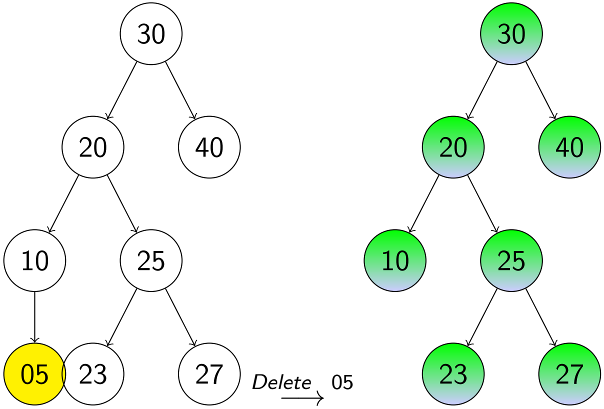
Insert a node in a BST



Delete a node from a BST

During delete a node there are three possibility :
i) Deleted node does not have any child
ii) Deleted node have only one child
iii)Deleted node have two child

- No child:



Delete One child

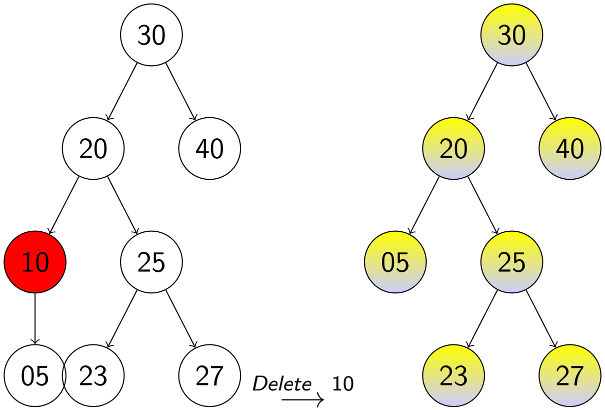


Figure: *Detete* 10 from Binary Search Tree.

Delete Two child

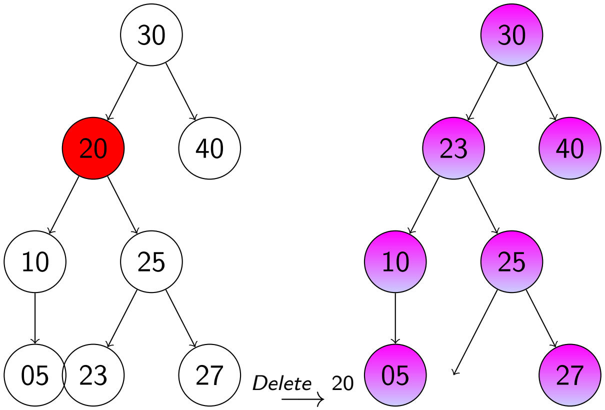


Figure: Detete 20 from Binary Search Tree.

Complexity of BST

Table: Complexity of BST operations

Operations	Best Case	Average Case	Worst Case
Traversal	O(N)	O(N)	O(N)
Searching	O(1)	O(log N)	O(N)
Insert	O(1)	O(log N)	O(N)
Delete	O(1)	O(log N)	O(N)

Applications of BST

(i) BST Used in many searching application where data is constantly entering or leaving such as map and set object in many language library
(ii) Storing a set of Names and being able to look up based on a prefix of the name.
(iii) BST is Used to express arithmetic expressions
(iv) To implement Huffman Coding Algorithm Binary search tree is used

Thank You

Mendelian genetics

Epistasis

Epistasis is a form of genetic interaction in which one gene masks the phenotypic expression of another.

or

Epistasis is an interaction between two non allelic genes in which one gene supresses the expression of another affecting the same character.

The expressed gene is called epistatic, while the supressed gene is said to be hypostatic.

Difference between Dominance and epistasis →

Dominance	Epistasis
i) Involves intra-allelic gene interaction.	i) Involves inter-allelic gene interaction.
ii) One allel hides the effect of other allele at the same gene pair.	ii) One gene hides the effect of other gene at different gene loci.

Types of Epistasis →

i) Dominant Epistasis (12:3:1) → A dominant allele (eg. A), of gene hides the effect of allele of another gene (eg. B) and express itself phenotypically.

⇒ The B allele (hypostatic) will be expressed only when gene locus A contains two recessive (aa) alleles.

⇒ Thus genotype AA BB or Aa Bb and AAbb or Aa bb produce the same phenotype.

→ Genotype aaBB or aa Bb and aabb produce two additional phenotype.

This type of dominant epistasis modifies the classical ratio of 9:3:3:1 into 12:3:1.

Epistatic alleles	Hypostatic alleles	Phenotypic expression
aa	bb	b
aa	BB, Bb	B
AA, Aa	BB, Bb, bb	A

Example →

Studied in summer squash (*Cucurbita pepo*)

→ Common fruit color - white, yellow and green.
→ White (W) is dominant over, colored squash.
→ Yellow (Y) is dominant over, green squash.
→ Pure breeding white fruited variety is crossed with the double recessive green variety, F_1 hybrid are all white.
→ When the hybrid are selfed - white, yellow and green fruited plants arise in the ratio of 12:3:1.

The effect of dominant gene 'Y' is masked by the dominant gene 'W' (epistatic gene)

* P WWYY x wwyy
(white) (green)

F_1: WwYy
(white)

gametes

♂/♀	WY	Wy	wY	wy
WY	WWYY white	WWYy white	WwYY white	WwYy white
Wy	WWYy white	WWyy white	WwYy white	Wwyy white
wY	WwYY white	WwYy white	wwYY Yellow	wwYy Yellow
wy	WwYy white	Wwyy white	wwYy Yellow	wwyy green

F2: White : Yellow : Green
12 : 3 : 1.

ii) Recessive epistasis (9:3:4) → Recessive epistasis occurs when the recessive alleles of one gene locus (aa the epistasis locus) supress the phenotypic expression of the alleles of another gene. (BB, Bb or bb alleles). This type of epistasis is called recessive epistasis.

▨ let us guess, the four phenotypic classes correspond to the genotype A_B_, A_bb, aaB_, and aaB_. If either of the one singly homozygous recessive genotype (i.e A-bb or aaB_) has the same phenotype as the double homozygous recessive (aabb), then a 9:3:4 phenotype ratio will be obtained.

▨ for eg - in the labrador Retriever breed of dogs, the B encodes a gene for an importance step in the production of melanin. The dominant allele, B is more efficient at pigment production than the recessive b allele, thus B hair appears black, and bb appears brown. A second locus, which we will call E, controls the deposition of melanin in the hairs. At least one functional E allele is required to deposite any pigment, wheather it is black or brown. Thus, all retrievers that are ee fail to deposit any melanin (and so appear pale yellow), regardless of the genotype at the B locus.

The ee genotype is therefore said to be epistatic to both the B and b alleles. Since the homozygous ee phenotype masks the phenotype of the B locus. The B/b locus is said to be hypostatic to the ee genotype. Because the masking allele is in this case is recessive, this is called recessive epistasis.

So,

$$\begin{tikzcd}
\boxed{BBEE} \arrow[dr] & \times & \boxed{bbee} \arrow[dl] \\
\text{Black} & \text{F1 generation} & \text{golden (pale yellow) (no pigment)} \\
& \boxed{BbEe}\ (\text{Black}) \arrow[dl, "\text{gametes}"'] \arrow[d] \arrow[dr] \arrow[drr] & \\
BE & Be \quad bE & be
\end{tikzcd}$$

F2 generation.

B- Black.
bb- brown
ee - golden/pale yellow.

9 : 3 : 4
Black Brown Yellow.

	BE	Be	bE	be
BE	BBEE. Black	BBEe. Black	BbEE Black	BbEe Black
be	BBEe Black	BBee. Yellow	BbEe Black	Bbee Yellow
bE	BbEE Black	BbEe Black	bbEE brown.	bbEe brown
be	BbEe Black	Bbee Yellow	bbEe brown	bbee . yellow.

iii) Duplicate Recessive Gene (9:7) → If both gene loci have homozygous recessive alleles and both of them produce identical phenotype the F2 ratio 9:3:3:1 would be 9:7. The genotype aaBB, aaBb, AAbb, Aabb and aabb produce same phenotype. Both dominant alleles when are present together only then they can complement each other this is known as complementary gene.

It is called complementary gene ~~dt~~ because the function of either A or B gene function has the same phenotype and they work together to produce a final product.

For, example consider a biochemical pathway, in which a colorless substrate is converted by the action of gene A to another colorless product, which is then converted by the action of gene B to a visible pigment. loss of function of either A or B or both. will have same result. No pigment production

$$\begin{tikzcd}\text{Colorless Compound} \arrow[r, "\text{gene A}"] & \text{colorless compound} \arrow[r, "\text{gene B}"] & \text{Purple Pigment}\end{tikzcd}$$

Fig: simplified pathway showing complementary gene action of A and B.

Scanned with CamScanner

$$\begin{tikzcd}
F_1 \text{ generation:} & \boxed{AAbb} \text{ (white flower)} \arrow[dr] & \times & \boxed{aaBB} \text{ (white flower)} \arrow[dl] \\
& & \boxed{AaBb} \text{ (Purple color flower)} \arrow[dll, "\text{gametes}"'] \arrow[dl] \arrow[d] \arrow[dr] & \\
F_2 \text{ generation:} \quad AB & Ab & aB & ab
\end{tikzcd}$$

ratio: 9:7
(purple : white)

♀/♂	AB	Ab	aB	ab
AB	AABB purple	AABb purple	AaBB purple	AaBb purple
Ab	AABb purple	AAbb white	AaBb purple	Aabb white
aB	AaBB purple	AaBb purple	aaBB white	aaBb white
ab	AaBb purple	Aabb white	aaBb white	aabb white

$$\begin{tikzcd}
\text{Colorless precursor} \arrow[r, "\text{Allele A}", "\text{pigment change catalyse}"'] \arrow[d] & \text{Colorless Precursor Q} \arrow[r, "\text{Allele B}", "\text{pigment change complete}"'] & \text{Purple pigment.} \\
{}
\end{tikzcd}$$

In this case dominant alleles on both locus are required hence wherever A and B both are present they result into purple effect masking the white.

IV) <u>Duplicate Dominant Gene (15:1)</u> → The dominant alleles of both the gene produce the same phenotypic effect giving the ratio 15:1.

In this case at least one of the dominant allele is necessary for the phenotypic effect e.g AABB, AaBb, Aabb, aaBB, aaBb give one phenotype.

In the absence of all the dominant gene (only in case of aabb) the recessive phenotype will be expressed. The duplicate gene are also called pseudoalleles.

Yet, another pigmentation pathway, in this case, in wheat, provides an example of this duplicate gene action. The biosynthesis of red pigment near the surface of wheat seeds involves many gene, two of which we will label A and B. Normal, red coloration of the wheat seed is maintained if function of either of these genes is lost in homozygous mutant (e.g in either aa B–, or A–bb). Only the doubly recessive mutant (aabb), which lacks function of both genes, shows a phenotype that differs from that produce by any of the other genotypes. A resonable interpretation of this result is that both gene encodes the same biological function, and either one alone is sufficient for the normal activity of that pathway.

F$_1$ progeny:

$$\begin{tikzcd}
\text{AABB (Red)} \arrow[dr] & \times & \text{aabb (white)} \arrow[dl] \\
& \boxed{\text{AaBb}}\ \text{(Red)} \arrow[dll, "\text{gametes}"'] \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{AB} & \text{Ab} & \text{aB} \quad \text{ab}
\end{tikzcd}$$

F$_2$ progeny.

	AB	Ab	aB	ab
AB	AABB Red	AABb Red	AaBB Red	AaBb Red
Ab	AABb Red	AAbb Red	AaBb Red	Aabb Red
aB	AaBB Red	AaBb Red	aaBB Red	aaBb Red
ab	AaBb Red	Aabb Red	aBab Red	aabb White

F$_2$ generation ratio →

Red : white
15 : 1

V) Polymeric gene interaction (9:6:1) → Two dominant alleles have similar effect when they are separate, but produce enhanced effect when they come together. Such gene interaction is known as polymeric gene interaction. The joint effect of two alleles appears to be additive or cumulative, but each of the two gene shows complete dominance, hence they cannot be considered as additive genes. In case of additive effect, gene show lack of dominance.

A well known example of polymeric gene interaction is fruit shape in summer squash. There are three types of fruit shape in this plant, viz, disc, spherical and long. The disc shape is controlled by two dominant genes (A and B), the spherical shape is produced by either dominant allele (A or B) and long shaped fruits develop in double recessive (aabb) plants.

A cross between disc shape (AABB) and long shape (aabb) strains produce disc shape fruits in F1. Inter-mating of F1 plants produced plants with disc, spherical, and long shape fruits in 9:6:1 ratio in F2. This can be explained as follow →

$$\begin{tikzcd}
\text{F}_1 \text{ progeny} & \boxed{AABB}\ \text{(Disc shape fruit)} & \times \arrow[d] & \boxed{aabb}\ \text{(long shape fruit)} \\
 & & \boxed{AaBb}\ \text{(Disc shape fruit)} \arrow[dll, "\text{gametes.}"'] \arrow[dl] \arrow[d] \arrow[dr] & \\
AB & Ab & aB & ab
\end{tikzcd}$$

F2 generation

Disc : spherical : long
9 : 6 : 1.

	AB	Ab	aB	ab.
AB	AABB Disc	AABb Disc.	AaBB Disc	AaBb Disc
Ab	AABb Disc	AAbb spherical	AaBb Disc	Aabb spherical
aB	AaBb Disc	AaBb Disc	aaBB spherical	aaBb spherical
ab.	AaBb Disc	Aabb spherical	aaBb spherical	aabb. Long

Here, plants with A- B— (9/16) genotypes produce disc shape fruits, those with A-bb-(3/16) and aa B-(3/16) genotypes produce spherical fruits and plants with aabb (1/16) genotype produce long fruits. Thus, in F2, normal dihybrid segregation ratio 9:3:3:1 is modified to 9:6:1 ratio. Similar gene action is also found in barley for awn length.

vi) Duplicate recessive interaction (13:3) → The dominant allele (A), either in homozygous or heterozygous condition, of one gene and the homozygous recessive allele (bb), of other gene produces the same phenotype.

In F2 generation, progenies having A (homozygous or heterozygous) or bb (homozygous) will not allow the c gene to be expressed.

Genotype AABB, AABb, AaBb and Aabb produce same phenotype and the genotype aaBB, aa Bb, and aabb produce another but same phenotype.

Scanned with CamScanner

Example: complete dominance at both gene pairs, but one gene when dominant epistatic to the other, and the second gene when homozygous recessive, epistatic to the first.

Feather colour of Fowl →

Gene pair 'A': colour inhibition is dominant to colour appearance.

Gene pair 'B': colour is dominant to white.

Interaction: Dominant color inhibition prevents color when color is present, color gene, when homozygous recessive prevents color even when dominant inhibitor is absent.

$$\begin{tikzcd}
\text{F}_1 \text{ Progeny: } \boxed{AABB} \text{ (White leg horn)} \arrow[r, phantom, "\times"] & \boxed{aabb} \text{ (White plymouth Rock)} \arrow[d] \\
 & \boxed{AaBb} \text{ (White)} \arrow[dll, "\text{gametes}"'] \arrow[dl] \arrow[d] \arrow[dr] \\
AB & Ab & aB & ab
\end{tikzcd}$$

F2 Progeny:

Ratio:
13 : 3
White : Colored

	AB	Ab	aB	ab
AB	AABB white	AABb white	AaBB white	AaBb. white
Ab	AABb white	AAbb colored	AaBb white	Aabb colored
aB	AaBB white	AaBb white	aaBB white	aaBb white
ab	AaBb white	Aabb colored	aaBb white	aabb white

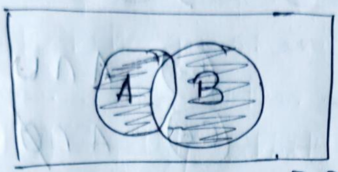
Symmetric Differences:

The symmetric difference of sets A and B, denoted by $A \oplus B$, consists of those elements which belong to A or B but not both.

$$A \oplus B = (A \cup B) \setminus (A \cap B)$$

or

$$A \oplus B = (A \setminus B) \cup (B \setminus A).$$



$A \oplus B$ is shaded.

Fundamental Products:

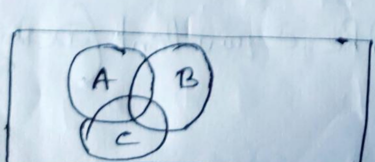
Consider n distinct sets $A_1, A_2, \ldots A_n$.
A fundamental product of the sets is a set of the form
$A_1^* \cap A_2^* \cap A_3^* \cap \ldots \cap A_n^*$ where $A_i^* = A$ or $A_i^* = A^c$.
We note that:

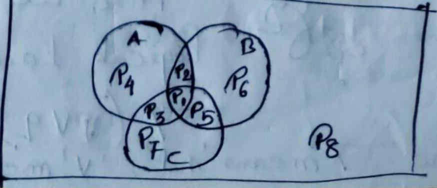
i) There are $m = 2^n$ such fundamental products.
ii) Any two such fundamental products are disjoint.
iii) The universal set U is the union of all fundamental products.

Eg: There are 3 sets A, B, C. The following lists the $m = 2^3 = 8$ fundamental products of the set A, B, C:

$P_1 = A \cap B \cap C$, $P_2 = A \cap B \cap C^c$, $P_3 = A \cap B^c \cap C$, $P_4 = A \cap B^c \cap C^c$

$P_5 = A^c \cap B \cap C$, $P_6 = A^c \cap B \cap C^c$, $P_7 = A^c \cap B^c \cap C$, $P_8 = A^c \cap B^c \cap C^c$





Algebra of Sets.

Laws of the algebra of sets.

Idempotent laws : 1a $A \cup A = A$ 1b $A \cap A = A$

Associative " : 2a $(A \cup B) \cup C = A \cup (B \cup C)$ 2b $(A \cap B) \cap C = A \cap (B \cap C)$

Commutative : 3a $A \cup B = B \cup A$ 3b $A \cap B = B \cap A$.

Distributive : 4a $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ 4b $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

Identity : 5a $A \cup \emptyset = A$ 5b $A \cap U = A$

6a $A \cup U = U$ 6b $A \cap \emptyset = \emptyset$.

Involution : 7 $(A^C)^C = A$

Complement : 8a $A \cup A^C = U$ 8b $A \cap A^C = \emptyset$

9a $U^C = \emptyset$ 9b $\emptyset^C = U$.

DeMorgan's law : 10a $(A \cup B)^C = A^C \cap B^C$ 10b $(A \cap B)^C = A^C \cup B^C$.

Proof of DeMorgan's law :

$$(A \cup B)^C = \{x \mid x \notin (A \text{ or } B)\}$$
$$= \{x \mid x \notin A \text{ and } x \notin B\}$$
$$= A^C \cap B^C.$$

~~$(A \cap B)^C = \{x \mid x \notin (A \text{ and } B)\}$~~

Here we use the equivalent (DeMorgan's) logical law:

$$\neg(p \vee q) = \neg p \wedge \neg q$$

where $\neg$ means 'not', '$\vee$' means 'or' and '$\wedge$' means 'and'.

Duality :

Suppose, E is an equation of set algebra. The dual E^* of E is the equation obtained by replacing each occurrence of $\cup, \cap, U$ & $\emptyset$ in E by $\cap, \cup, \emptyset$ and U respectively.

eg : $(U \cap A) \cup (B \cap A) = A$

The dual of A is

$$A^* = (\emptyset \cup A) \cap (B \cup A)$$

Observe ~~that the~~ pairs in laws in table are dual of each other. ~~The~~ It is a fact of set algebra, called the Principle of duality, that if any equation E is an identity then its dual E^* is also an identity.

Qn

(1) Let $U = \{1, 2, 3, \dots\dots 9\}$ be the universal set

$A = \{1, 2, 3, 4, 5\}$, $B = \{4, 5, 6, 7\}$, $C = \{5, 6, 7, 8, 9\}$,

$D = \{1, 3, 5, 7, 9\}$, $E = \{2, 4, 6, 8\}$, $F = \{1, 5, 9\}$.

Find i) $A \cup B$, ii) $A \cap B$, iii) $A \cup C$, iv) $A \cap C$, v) $D \cup F$, vi) $D \cap F$

vii) A^c, B^c, C^c, D^c, E^c

viii) $A \setminus B$, $B \setminus A$, $D \setminus E$.

ix) $A \oplus B$, $C \oplus D$, $E \oplus F$.

(2) ~~Prove $B \setminus A = B \cap A^c$~~

In a survey of 120 people, it was found that

65 read Newsweek Magazine.
45 " Times.
42 " Fortune.

20 read both Newsweek & Time

25 read both Newsweek & Fortune.

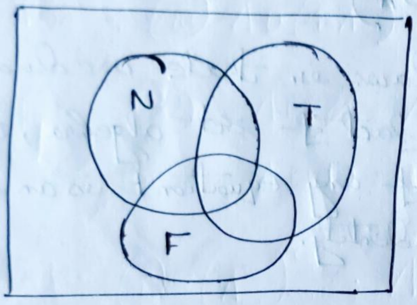
15 read both Time & Fortune.

8 reads all three magazines.

a) Find the no. of people who read at least one of the three magazines.

b) Fill the correct no. of people in ~~[illegible]~~ each of the eight regions of venn diagram where N, T & F denotes the set of people who read Newsweek, Time & Fortune respectively.

c) Find the no. of people who read exactly one magazine.



ⓐ $n(N \cup T \cup F) = n(N) + n(T) + n(F) - n(N \cap T)$

$- n(N \cap F) - n(T \cap F) + n(N \cap T \cap F)$

$= 65 + 45 + 42 - 20 - 25 - 15 + 8 = 100$.

ⓑ 8 read all three magazines.

20-8=12 read Newsweek and Times but not all three magazin..

25-8 = 17 " Newsweek & Fortune " " "

15-8 = 7 " Times & Fortune " " "

65 - 12 - 8 - 17 = 28 read only Newsweek.

45 - 12 - 8 - 7 = 18 read only Times.

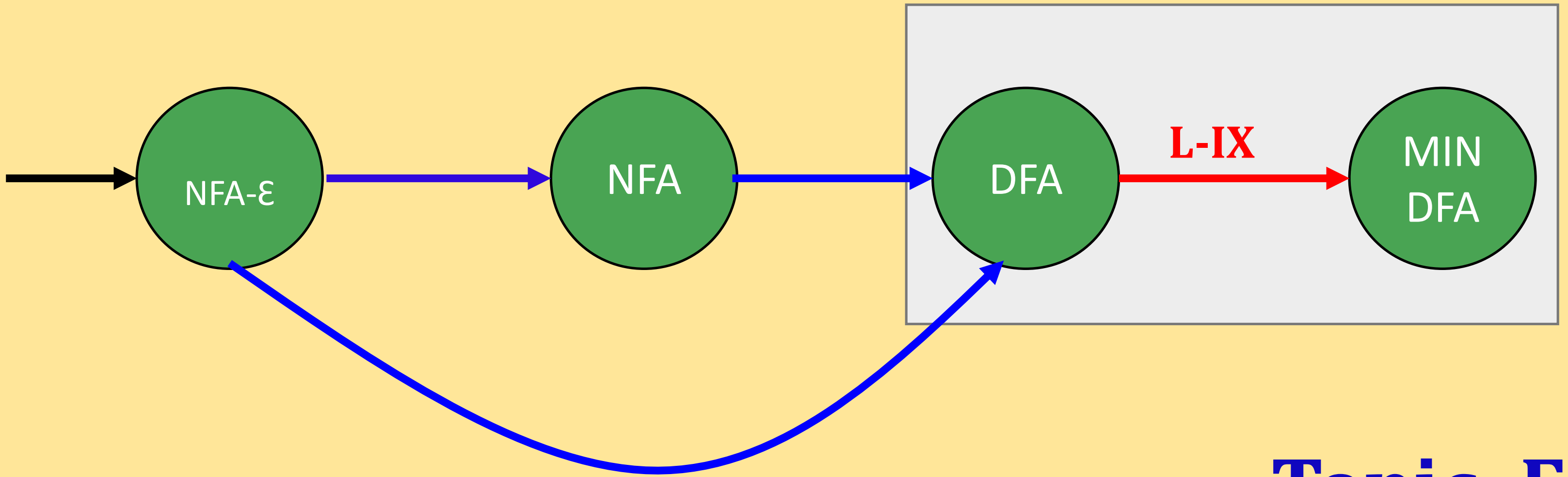
42 - 17 - 8 - 7 = 10 read only Fortune.

120 - 100 = 20 read no magazine at all.

c) (28 + 18 + 0) = 56 read exactly one of the magazines.

Formal Language & Automata Theory

PCC-CS 403



Topic: Finite Automata [FA]

DFA Minimization(Revisited)

Lecture – IX

Prof. Mithun Roy

Minimization of DFA

Minimization of DFA **means reducing the number of states from given FA**. Thus, we get the FSM(finite state machine) with redundant states after minimizing the FSM.

We have to follow the various steps to minimize the DFA. These are as follows:

Step 1: **Remove all the states that are unreachable from the initial state** via any set of the transition of DFA.

Step 2: Draw the transition table for all pair of states.

Step 3: Now split the transition table into two tables T1 and T2. **T1 contains all final states, and T2 contains non-final states.**

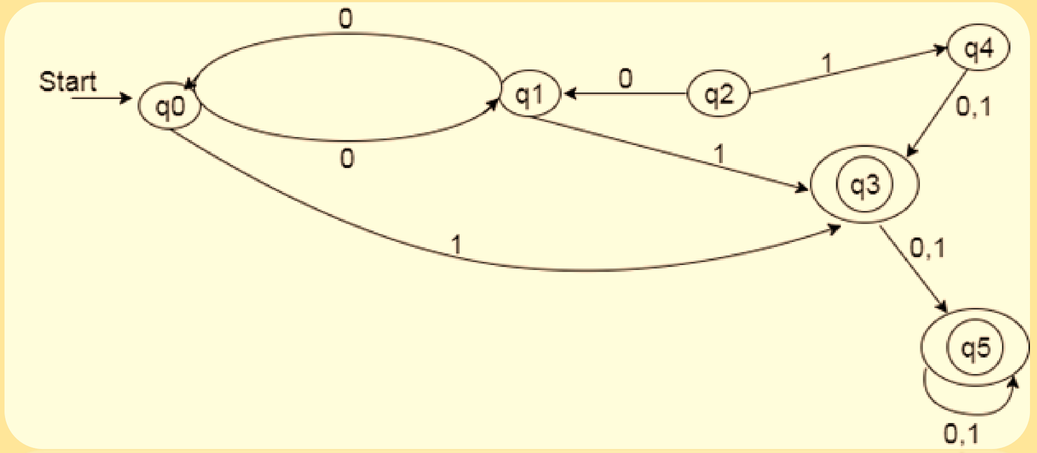
Step 4: Find similar rows from T1 such that: 1. δ (q, a) = p 2. δ (r, a) = p
That means, **find the two states which have the same value of a and b and remove one of them**.

Step 5: **Repeat step 3 until we find no similar rows available in the transition table T1.**

Step 6: Repeat step 3 and step 4 for table T2 also.

Step 7: **Now combine the reduced T1 and T2 tables**. The combined transition table is the transition table of minimized DFA.

Example - I



Step 1: In the given DFA, q2 and q4 are the unreachable states so remove them.

Step 2: Draw the transition table for the rest of the states.

Q | Σ	0	1
→ q0	q1	q3
q1	q0	q3
*q3	q5	q5
*q5	q5	q5

Step 3: Now divide rows of transition table into two sets as:

1. One set contains those rows, which start from non-final states:

Q | Σ	0	1
→ q0	q1	q3
q1	q0	q3

2. Another set contains those rows, which starts from final states.

Q | Σ	0	1
*q3	q5	q5
*q5	q5	q5

Step 4: Set 1 has no similar rows so set 1 will be the same.

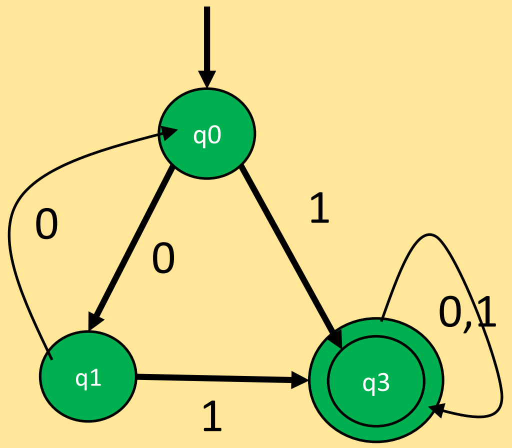
Step 5: In set 2, row 1 and row 2 are similar since q3 and q5 transit to the same state on 0 and 1. So skip q5 and then replace q5 by q3 in the rest.

Q | Σ	0	1
*q3	q3	q3

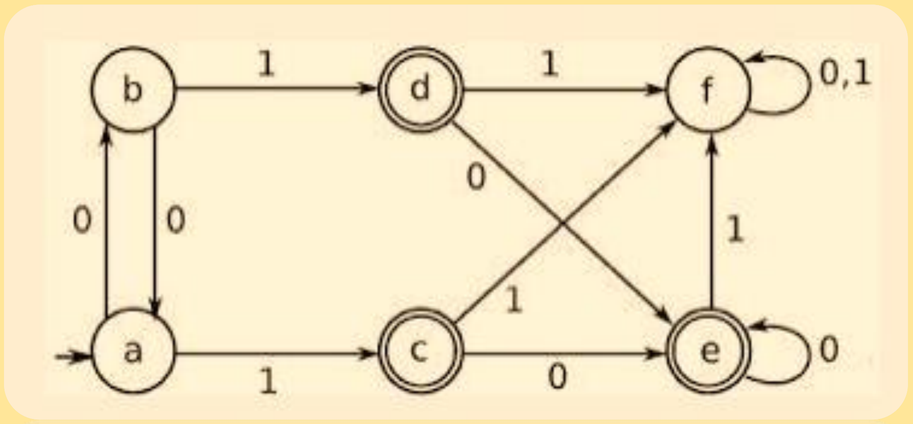
Step 6: Now combine set 1 and set 2 as:

Q | Σ	0	1
→ q0	q1	q3
q1	q0	q3
*q3	q3	q3

So, The Minimized DFA.



Example - II



Step 1: In the given DFA, all the state are reachable states.

Step 2: Draw the transition table for the rest of the states.

Q | Σ	0	1
→ a	b	c
b	a	d
*c	e	f
*d	e	f
*e	e	f
f	f	f

Step 3: Now divide rows of transition table into two sets as:

1. One set contains those rows, which start from non-final states:

Q | Σ	0	1
→ a	b	c
b	a	~~d,~~ c
f	f	f

2. Another set contains those rows, which starts from final states.

Q | Σ	0	1
*c	e	f
*d	e	f
*e	e	f

Step 4: Set 1 has no similar rows so set 1 will be the same.

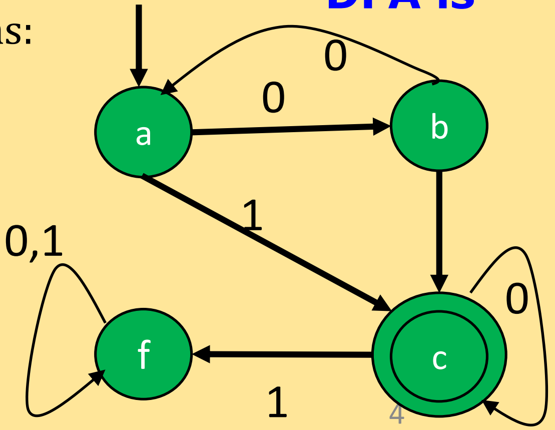
Step 5: In set 2, row 1,2 and row 3 are similar since c, d and e transit to the same state on 0 and 1. So skip e & d and then replace e by c in the rest.

Q | Σ	0	1
*c	c	f

Step 6: Now combine set 1 and set 2 as:

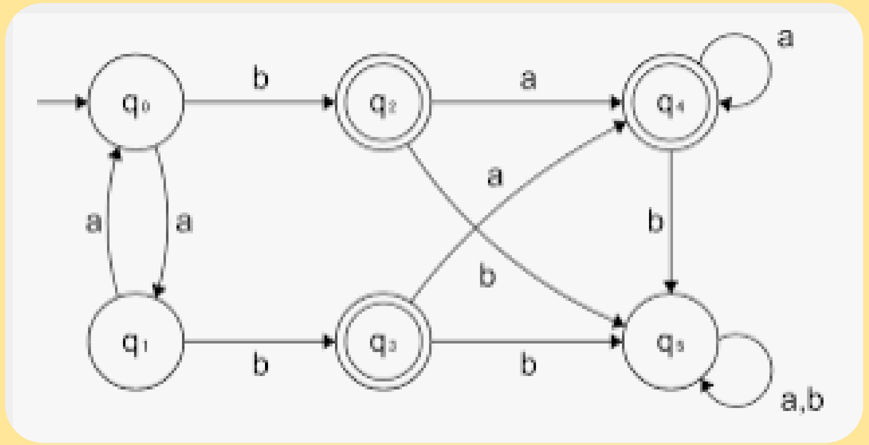
Q | Σ	0	1
→ a	b	c
b	a	c
f	f	f
*c	c	f

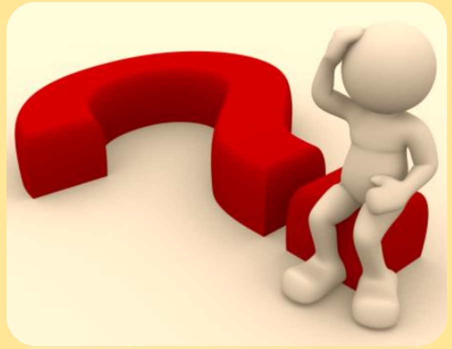
So, The Minimized DFA is



Homework – VIII

Minimized the given DFA





Thank You

Design & Analysis of Algorithm

PCC-CS 404

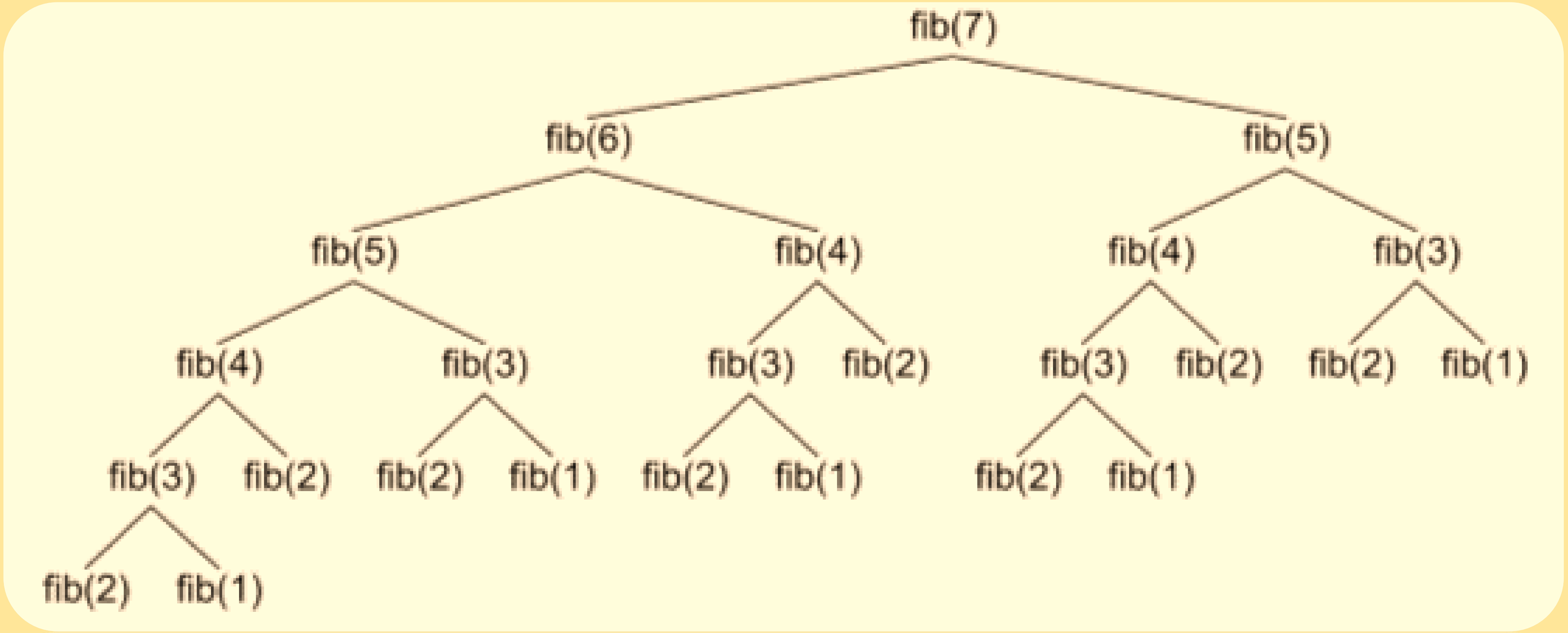
Topic: Dynamic Programming

[Matrix Chain Multiplication]

Lecture – VII

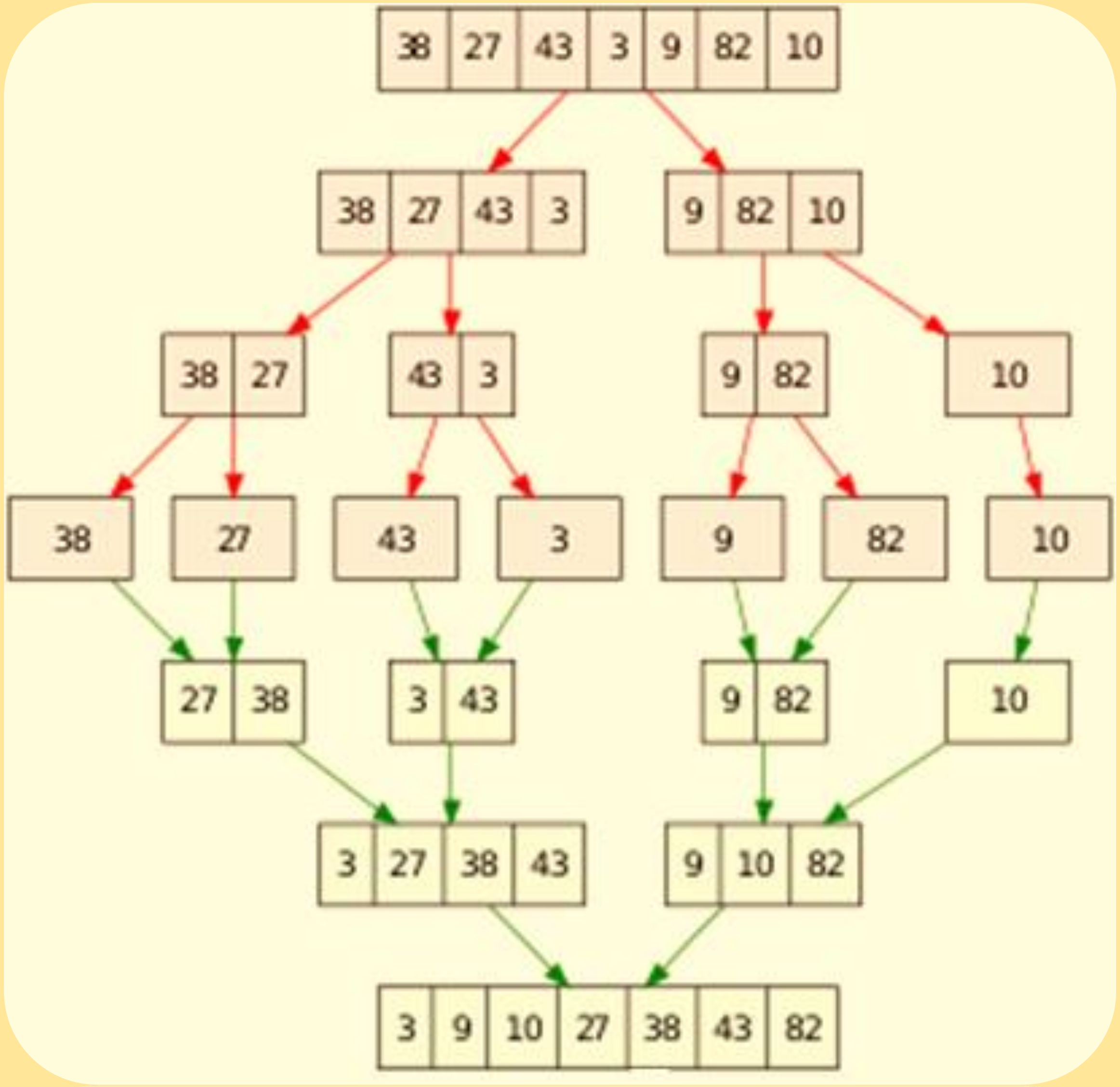
Prof. Mithun Roy

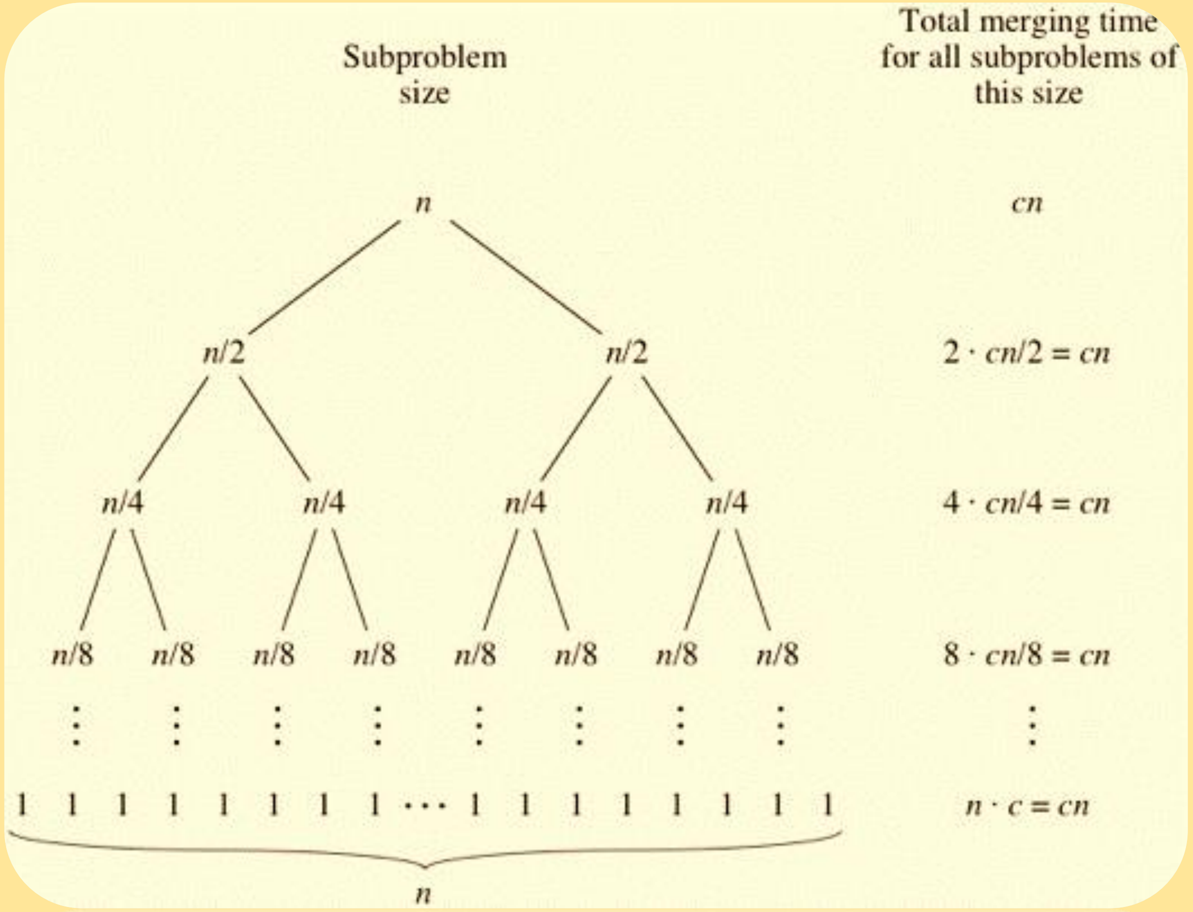
Method



$$\mathbf{T[n] = T[n-1] + T[n-2], T[0] = 1, T[1] = 1}$$
$$\boldsymbol{T[2] = T[1] + T[0] = 2\,, T[3] = T[2] + T[1] = 3, \ldots.}$$

Merge Sort Method





Method

$$B_{2\times2} \times C_{2\times3} = B_{p\times q} \times C_{q\times r} = R_{p\times r}\ (p \times q \times r)$$

$$A_1, A_2, A_3$$

$$A_{10\times20}, A_{20\times30}, A_{30\times40}$$

$$(\boldsymbol{A_1}\times \boldsymbol{A_2}) \times \boldsymbol{A_3} = 6000 + R_{10\times30} \times A_3 = 6000 + 12000 = 18000$$

$$A_1 \times (A_2 \times A_3) = A_1 \times R_{20\times40} + 24000 = 8000 + 24000 = 32000$$

$$(A_1\times A_2) \times A_3$$

Example of Matrix Chain Multiplication

Example: We are given the sequence {4, 10, 3, 12, 20, and 7}. The matrices have size 4 x 10, 10 x 3, 3 x 12, 12 x 20, 20 x 7. We need to compute M [i,j], 0 ≤ i, j≤ 5. We know M [i, i] = 0 for all i.

$$M(i,j) = \begin{cases} 0 & , i = j \\ \min\limits_{i \le k < j} M(i,k) + M(k+1,j) + p_{i-1}p_k p_j & , i < j \end{cases}$$

$$A_{1_{4\times10}}, A_{2_{10\times3}}, A_{3_{3\times12}}, A_{4_{12\times20}}, A_{5_{20\times7}}$$

$p_0 = 4$, $p_1 = 10$, $p_2 = 3$, $p_3 = 12$, $p_4 = 20$ and $p_5 = 7$

$$M(1,2) = M(1,1) + M(2,2) + p_0p_1p_2 = 120$$
$$M(2,3) = M(2,2) + M(3,3) + p_1p_2p_3 = 360$$
$$M(3,4) = M(3,3) + M(4,4) + p_2p_3p_4 = 720$$
$$M(4,5) = M(4,4) + M(5,5) + p_3p_4p_5 = 1680$$

j

1	2	3	4	5	i
0	120/ 1				1
	0	360/ 2			2
		0	720/ 3		3
			0	1680 /4	4
				0	5

Example of Matrix Chain Multiplication

Example: We are given the sequence {4, 10, 3, 12, 20, and 7}. The matrices have size 4 x 10, 10 x 3, 3 x 12, 12 x 20, 20 x 7. We need to compute M [i,j], 0 ≤ i, j≤ 5. We know M [i, i] = 0 for all i.

$$\mathrm{M}(i,j) = \begin{cases} 0 & , i = j \\ \min\limits_{i \le k < j} M(i,k) + M(k+1,j) + p_{i-1}p_kp_j & , i < j \end{cases}$$

$$A_{1_{4\times10}}, A_{2_{10\times3}}, A_{3_{3\times12}}, A_{4_{12\times20}}, A_{5_{20\times7}}$$

$$\mathrm{p}_0 = 4\,, \mathrm{p}_1 = 10\,, \mathrm{p}_2 = 3\,, \mathrm{p}_3 = 12, \mathrm{p}_4 = 20 \text{ and } \mathrm{p}_5 = 7$$

j

1	2	3	4	5	i
0	120/1	264/2			1
	0	360/2			2
		0	720/3		3
			0	1680/4	4
				0	5

$$M(1,3) = min \begin{cases} M(1,1) + M(2,3) + p_0p_1p_3 = 360 + 480 > 264 \\ M(1,2) + M(3,3) + p_0p_2p_3 = 120 + 144 = 264 \end{cases}$$

Example of Matrix Chain Multiplication

Example: We are given the sequence {4, 10, 3, 12, 20, and 7}. The matrices have size 4 x 10, 10 x 3, 3 x 12, 12 x 20, 20 x 7. We need to compute M [i,j], 0 ≤ i, j≤ 5. We know M [i, i] = 0 for all i.

$$M(i,j) = \begin{cases} 0 & ,i = j \\ \min\limits_{i\le k<j} M(i,k) + M(k+1,j) + p_{i-1}p_kp_j & ,i < j \end{cases}$$

$$A_{1_{4\times10}}, A_{2_{10\times3}}, A_{3_{3\times12}}, A_{4_{12\times20}}, A_{5_{20\times7}}$$

$$p_0 = 4\,, p_1 = 10\,, p_2 = 3\,, p_3 = 12, p_4 = 20 \text{ and } p_5 = 7$$

$$M(2,4) = \min \begin{cases} M(2,2) + M(3,4) + p_1p_2p_4 = 720 + 600 = 1320 \\ M(2,3) + M(4,4) + p_1p_3p_4 = 360 + 2400 > 1320 \end{cases}$$

j

1	2	3	4	5	i
0	120/1	264/2			1
	0	360/2	1320/2		2
		0	720/3		3
			0	1680/4	4
				0	5

Example of Matrix Chain Multiplication

Example: We are given the sequence {4, 10, 3, 12, 20, and 7}. The matrices have size 4 x 10, 10 x 3, 3 x 12, 12 x 20, 20 x 7. We need to compute M [i,j], 0 ≤ i, j≤ 5. We know M [i, i] = 0 for all i.

$$M(i,j) = \begin{cases} 0 & ,i = j \\ \min\limits_{i\le k<j} M(i,k) + M(k+1,j) + p_{i-1}p_k p_j & ,i < j \end{cases}$$

$$A_{1_{4\times10}}, A_{2_{10\times3}}, A_{3_{3\times12}}, A_{4_{12\times20}}, A_{5_{20\times7}}$$

$$\mathrm{p}_0 = 4\ ,\mathrm{p}_1 = 10\ ,\mathrm{p}_2 = 3\ ,\mathrm{p}_3 = 12, \mathrm{p}_4 = 20 \text{ and } \mathrm{p}_5 = 7$$

j

1	2	3	4	5	i
0	120/1	264/2			1
	0	360/2	1320/2		2
		0	720/3	1140/4	3
			0	1680/4	4
				0	5

$$\mathrm{M}(3,5) = \min \begin{cases} \mathrm{M}(3,3) + \mathrm{M}(4,5) + \mathrm{p}_2\mathrm{p}_3\mathrm{p}_5 = 1680 + 252 > 1140 \\ \mathrm{M}(3,4) + \mathrm{M}(5,5) + \mathrm{p}_2\mathrm{p}_4\mathrm{p}_5 = 720 + 420 = 1140 \end{cases}$$

Example of Matrix Chain Multiplication

Example: We are given the sequence {4, 10, 3, 12, 20, and 7}. The matrices have size 4 x 10, 10 x 3, 3 x 12, 12 x 20, 20 x 7. We need to compute M [i,j], 0 ≤ i, j≤ 5. We know M [i, i] = 0 for all i.

$$\mathrm{M}(i,j) = \begin{cases} 0 & ,i = j \\ \min\limits_{i \le k < j} M(i,k) + M(k+1,j) + p_{i-1}p_kp_j & ,i < j \end{cases}$$

$$A_{1_{4\times10}}, A_{2_{10\times3}}, A_{3_{3\times12}}, A_{4_{12\times20}}, A_{5_{20\times7}}$$

$$\mathrm{p}_0 = 4\,, \mathrm{p}_1 = 10\,, \mathrm{p}_2 = 3\,, \mathrm{p}_3 = 12, \mathrm{p}_4 = 20 \text{ and } \mathrm{p}_5 = 7$$

j

1	2	3	4	5	i
0	120/1	264/2	1080/2		1
	0	360/2	1320/2		2
		0	720/3	1140/4	3
			0	1680/4	4
				0	5

$$\mathrm{M}(1,4) = min \begin{cases} M(1,1) + M(2,4) + p_0p_1p_4 = 1320 + 800 > 1080 \\ M(1,2) + M(3,4) + p_0p_2p_4 = 120 + 720 + 240 = 1080 \\ M(1,3) + M(4,4) + p_0p_3p_4 = 264 + 960 > 1080 \end{cases}$$

Example of Matrix Chain Multiplication

Example: We are given the sequence {4, 10, 3, 12, 20, and 7}. The matrices have size 4 x 10, 10 x 3, 3 x 12, 12 x 20, 20 x 7. We need to compute M [i,j], 0 ≤ i, j≤ 5. We know M [i, i] = 0 for all i.

$$\mathrm{M}(i,j) = \begin{cases} 0 & , i = j \\ \min\limits_{i \le k < j} M(i,k) + M(k+1,j) + p_{i-1}p_kp_j & , i < j \end{cases}$$

$$A_{1_{4\times10}}, A_{2_{10\times3}}, A_{3_{3\times12}}, A_{4_{12\times20}}, A_{5_{20\times7}}$$

$$\mathrm{p}_0 = 4\,, \mathrm{p}_1 = 10\,, \mathrm{p}_2 = 3\,, \mathrm{p}_3 = 12, \mathrm{p}_4 = 20 \text{ and } \mathrm{p}_5 = 7$$

j

1	2	3	4	5	i
0	120/1	264/2	1080/2		1
	0	360/2	1320/2	1350/2	2
		0	720/3	1140/4	3
			0	1680/4	4
				0	5

$$M(2,5) = min \begin{cases} M(2,2) + M(3,5) + p_1p_2p_5 = 1140 + 210 = 1350 \\ M(2,3) + M(4,5) + p_1p_3p_5 = 360 + 1680 + 840 > 1350 \\ M(2,4) + M(5,5) + p_1p_4p_5 = 1320 + 1400 > 1350 \end{cases}$$

Example of Matrix Chain Multiplication

Example: We are given the sequence {4, 10, 3, 12, 20, and 7}. The matrices have size 4 x 10, 10 x 3, 3 x 12, 12 x 20, 20 x 7. We need to compute M [i,j], 0 ≤ i, j≤ 5. We know M [i, i] = 0 for all i.

$$\mathrm{M}(i,j) = \begin{cases} 0 & ,i=j \\ \min\limits_{i\le k<j} M(i,k)+M(k+1,j)+p_{i-1}p_kp_j & ,i<j \end{cases}$$

$$A_{1_{4\times10}}, A_{2_{10\times3}}, A_{3_{3\times12}}, A_{4_{12\times20}}, A_{5_{20\times7}}$$

$$\mathrm{p}_0 = 4\,, \mathrm{p}_1 = 10\,, \mathrm{p}_2 = 3\,, \mathrm{p}_3 = 12, \mathrm{p}_4 = 20 \text{ and } \mathrm{p}_5 = 7$$

$$M(1,5) = min \begin{cases} M(1,1)+M(2,5)+p_0p_1p_5 = 1350+280 > 1344 \\ M(1,2)+M(3,5)+p_0p_2p_5 = 120+1140+84 = 1344 \\ M(1,3)+M(4,5)+p_0p_3p_5 = 264+1680+336 > 1344 \\ M(1,4)+M(5,5)+p_0p_4p_5 = 1080+560 > 1344 \end{cases}$$

j

1	2	3	4	5	
0	120/1	264/2	1080/2	1344/2	1
	0	360/2	1320/2	1350/2	2
		0	720/3	1140/4	3
			0	1680/4	4
				0	5

i

Algorithm

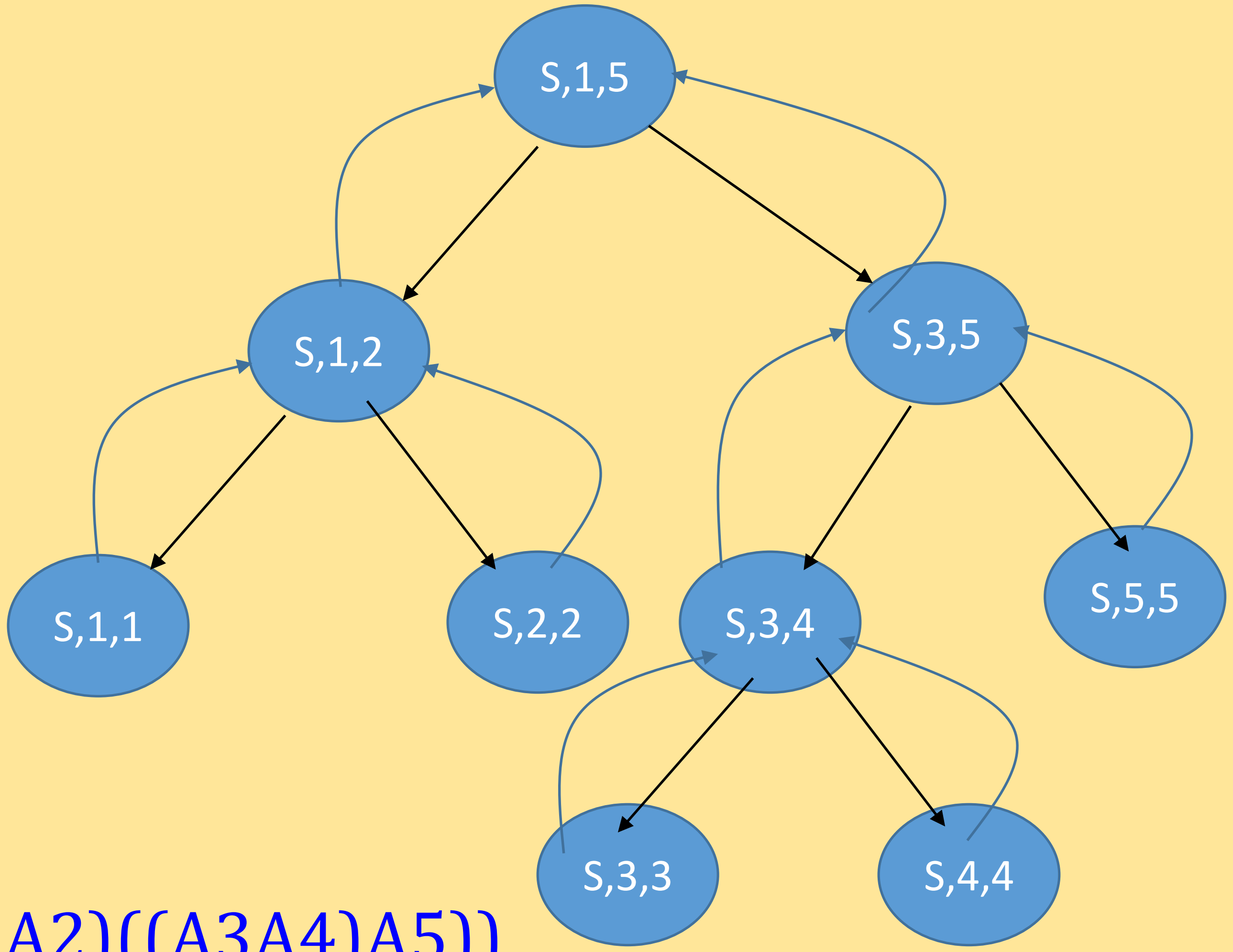
```
PRINT-OPTIMAL-PARENS(S,i,j)
if i = j then
  print Ai;
else
  print "(";
  PRINT-OPTIMAL-PARENS(S,i,S(i,j));
  PRINT-OPTIMAL-PARENS(S,S(i,j)+1,j);
  print ")";
end
```

j

1	2	3	4	5	
0	1	2	2	2	1
	0	2	2	2	2
		0	3	4	3
			0	4	4
				0	5

i

S Table



((A1A2)((A3A4)A5))

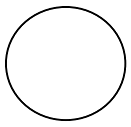
In this way if you multiply then optimal number of scalar multiplication is required.

IT-5th Semester
Operating System (PCC-CS502)
Lecture-13

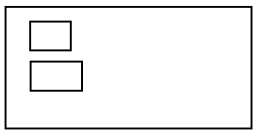
Resource Allocation Graphs:

Deadlocks can be described more precisely in terms of a directed graph called a system resource allocation graph. This graph consists of a set of vertices V and a set of edges E. The set of vertices is portioned into two different types of nodes P={P0, P1… Pn}, the set of the active processes in the system, and R={R0, R1… Rn}, the set consisting of all resource types in the system. A directed edge from a process Pi to resource type Rj signifies that process Pi requested an instance of Rj and is waiting for that resource. A directed edge from Rj to Pi signifies that an instance of Rj has been allocated to Pi.

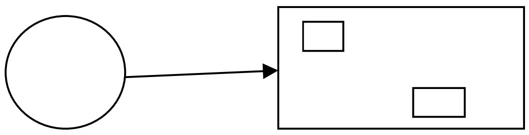
• Process



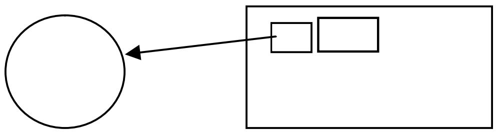
• Resource Type with 2 instances

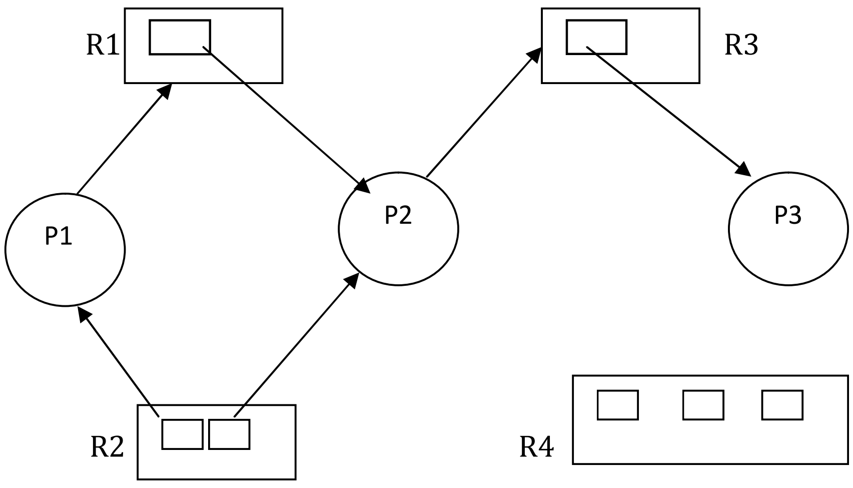


• Pi requests instance of Rj



• Pi is holding an instance of Rj





The resource allocation graph shown above depicts the following situation:
P= {P1, P2, P3 }
R= {R1, R2, R3}
E= {P1 → R1, P2 → R3, R1 → P2, R2 → P2, R2 → P1, P3 → R3}

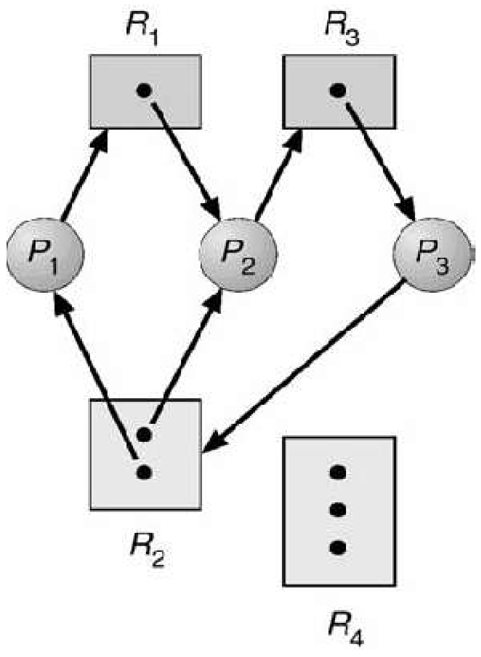
Resource Instances
One instance of resource type R1
Two instances of resource type R2
One instance of resource type R3
Three instances of resource type R4

Process States
Process P1 is holding an instance of resource R2, and is waiting for an instance of resource R1.
Process P2 is holding an instance of resource R1 and R2, and is waiting for an instance of resource R3.
Process P3 is holding an instance of resource R3.

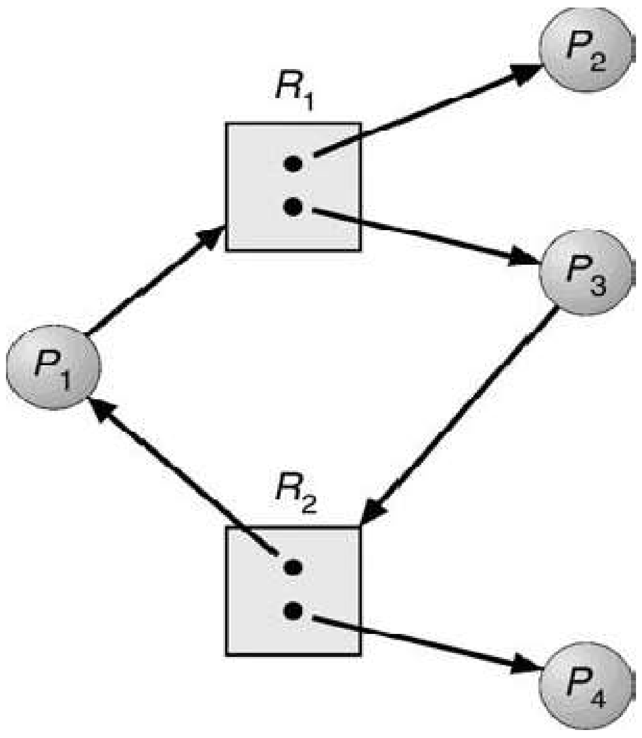
Given the definition of a resource allocation graph, it can be shown that if the graph contains no cycles, then no process is deadlocked.

If the graph contains cycles then:

- If only one instance per resource type, then a deadlock exists.
- If several instances per resource type, possibility of deadlock exists.



Here is a resource allocation graph with a deadlock. There are two cycles in this graph:
{P1 → R1, R1 → P2, P2 → R3, R3 → P3, P3 → R2, R2 → P1} and
{P2 → R3, R3 → P3, P3 → R2, R2 → P2}
No process will release an already acquired resource and the three processes will remain in the deadlock state.



The graph shown above has a cycle but there is no deadlock because processes P2 and P4 do not require further resources to complete their execution and will release the resources they are currently hold in finite time. These resources can then be allocated to P1 and P3 for them to resume their execution.

RESEARCH METHODOLOGY

NATURE & MEANING OF RESEARCH

In the modern complex world every society today is faced with serious social, economic & political problems. These problems need systematic, intelligent and Practical solutions. Problem solving is technical process. It requires the accumulation of new knowledge. Research provides the means for accumulating knowledge & wisdom. In other words, research is a systematic effort of gathering analysis & interpretation of problems confronted by humanity. It is a thinking process and scientific method of studying a problem and finding solution. It is an in-depth analysis based on reflective thinking.

DEFINITIONS

Research in common parlance refers to a search for knowledge. One can also define research as a scientific and systematic search for pertinent information on a specific topic. Research is an academic activity and the term should be used in a technical sense.

a) -William Emory defines Research as "any organised enquiry designed and carried out to provide information for solving a problem"

b) The new Oxford English Dictionary defines research is "the scientific investigation into and study of material, sources etc in order to establish facts and the reach new conclusions".

c) Redman and Mory defines, research as "a systematised effort to gain new knowledge''.

d) "A careful investigation or inquiry specially through search for new facts in any branch of knowledge" Advanced Leaner's Dictionary.

CHARACTERISTICS OF RESEARCH

The above definitions reveal the following characteristics of Research

1. Research is a systematic and critical investigation into a phenomenon.
2. It is not mere compilation of facts.
3. It adopts scientific method.
4. It is objective & Logical
5. It is based on empirical evidence.
6. Research is directed towards finding answers to questions

7. It emphasis the generalisation of theories and principles.

OBJECTIVES OF RESEARCH

The objectives of Research can be grouped under the following heads

1. To gain familiarity with a phenomenon or to achieve new insights to it.
2. To portray accurately the characteristics of a particular individual situation or a group.
3. To determine the frequency with which something occurs or with which it is associated with something else.
4. To test a hypothesis or a casual relationship between variables.

MOTIVATIONS IN RESEARCH

What makes people to undertake research?

The answer is as follows.

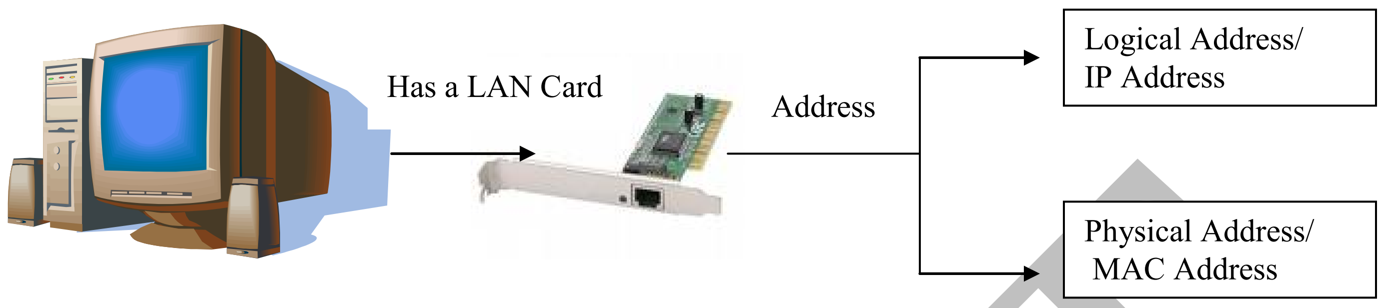
1. Desire to get a research degree along with its benefits.
2. Desire to face the challenge in the solving the unsolved Problem.
3. Desire to get intellectual joy of doing some creative work.
4. Desire to be of service to Society.
5. Desire to get respectability

IMPORTANCE OF RESEARCH

"All progress is born of enquiry. Doubt is often better than overconfidence, for it leads to enquiry & enquiry leads to investigation". Research has an important role in guiding social plan. Knowledge of the society & the cultural behaviour of the people require proper planning for their well development. Because knowledge & cultural behaviour of human being are interdependent. A reliable knowledge is needed for planning & this is possible only through research. Knowledge is a kind of power with which one can face the implication of a particular Phenomenon. Research provides the basis for all govt policies in our economic system.

Research help us in making predictions. Eg. Chernobil Nuclear, nuclear plant disastrour,
Bhopal gas disastrour. Research is equally important in seeking answer to various social problems In addition to this, the significance of research can be understood with the following points.

ADDRESSING



Overview of IPv4 Addressing Scheme

i) The identifier used in the network layer in the Internet model to identify each device connected to the Internet is called the Internet address or IP Address.

ii) IP address is 32 bits (IPv4) or 128 bits (IPv6).

iii) It is represented by dotted decimal notation.

iv) IP Address is **unique**.
They are unique in the sense that each address defines one and only one connection to the Internet. Two devices in the Internet can never have the same address at the same time.

v) The IP Addresses are **universal** in the sense that the addressing system must be accepted by any host that wants to be connected to the Internet.

Notations

We use notations to show an IPv4 address: **binary notation** and **dotted-decimal notation**.

Binary Notation

- In binary notation, the IPv4 address is displayed as 32 bits. Each octet is often referred to as a byte. So it is common to hear an IPv4 address referred to as a 32-bit address or a 4-byte address.
- The following is an example of an IPv4 address in binary notation:
  01110101 10010101 00011101 00000010

Dotted-Decimal Notation

- To make the IPv4 address more compact and easier to read, Internet addresses are usually written in decimal form with a decimal point (dot) separating the bytes. The following is the dotted-decimal notation of the above address: 117.149.29.2

```
10000000 . 00001011 . 00000011 . 00011111
   ↓           ↓          ↓            ↓
  128    .    11    .     3     .      31
```

CSE/IT

Question1.
Change the following IPv4 addresses from binary notation to dotted-decimal notation.
a. 10000001 00001011 00001011 11101111
b. 11000001 10000011 00011011 11111111

Solution
We replace each group of 8 bits with its equivalent decimal number (see Appendix B) and add dots for separation.
a. 129.11.11.239
b. 193.131.27.255

Question2.
Change the following IPv4 addresses from dotted-decimal notation to binary notation.
a. 111.56.45.78
b. 221.34.7.82

Solution
We replace each decimal number with its binary equivalent (see Appendix B).
a..01101111 00111000 00101101 01001110
b. 11011101 00100010 00000111 01010010

Question3.
Find the error, if any, in the following IPv4 addresses.
a. 111.56.045.78
b. 221.34.7.8.20
c. 75.45.301.14
d. 11100010.23.14.67

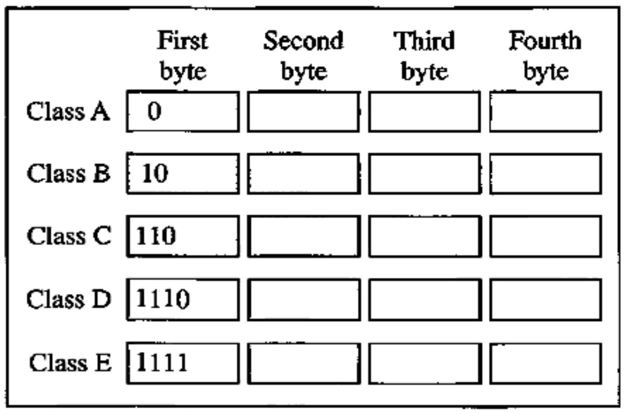
Solution
a. There must be no leading zero (045).
b. There can be no more than four numbers in an IPv4 address.
c. Each number needs to be less than or equal to 255 (301 is outside this range).
d. A mixture of binary notation and dotted-decimal notation is not allowed.

CSE/IT

Classful Addressing

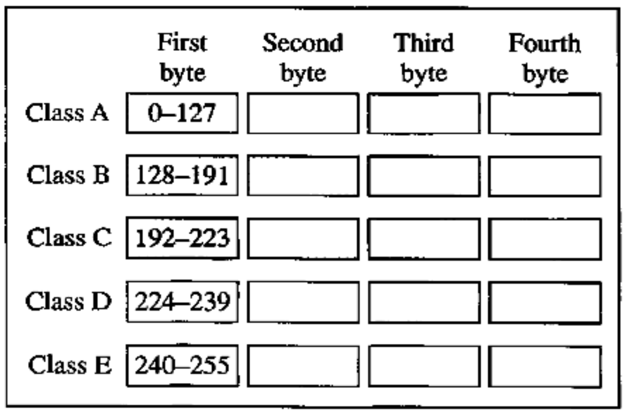
- IPv4 addressing when started a few decades ago, used the concept of classes. This architecture is called classful addressing.
- In classful addressing, the address space is divided into five classes: A, B, C, D, and E. Each class occupies some part of the address space.
- In mid 1990s, a new architecture, called classless addressing was introduced, which will eventually supersede the original architecture.

Finding the classes in binary and dotted decimal notation.



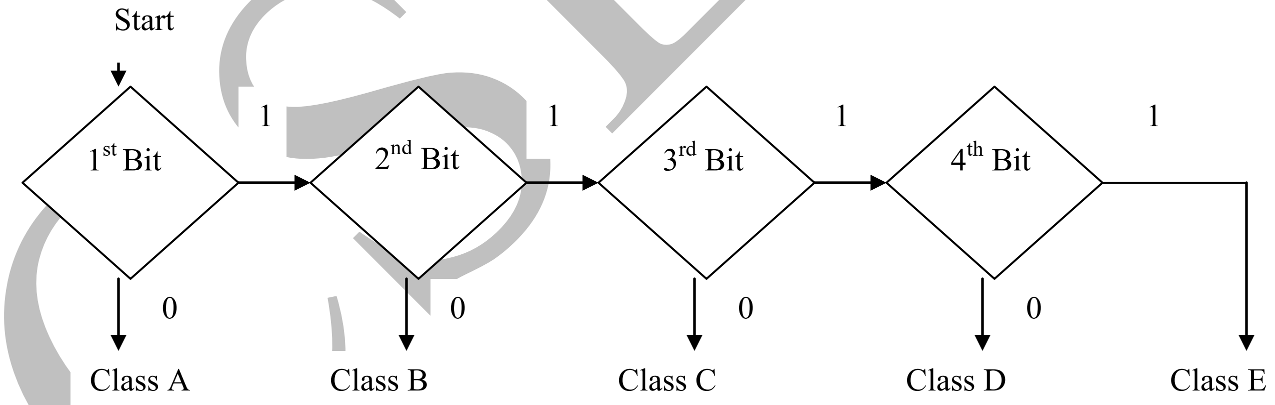
	First byte	Second byte	Third byte	Fourth byte
Class A	0			
Class B	10			
Class C	110			
Class D	1110			
Class E	1111			

a. Binary notation



	First byte	Second byte	Third byte	Fourth byte
Class A	0–127			
Class B	128–191			
Class C	192–223			
Class D	224–239			
Class E	240–255			

b. Dotted-decimal notation

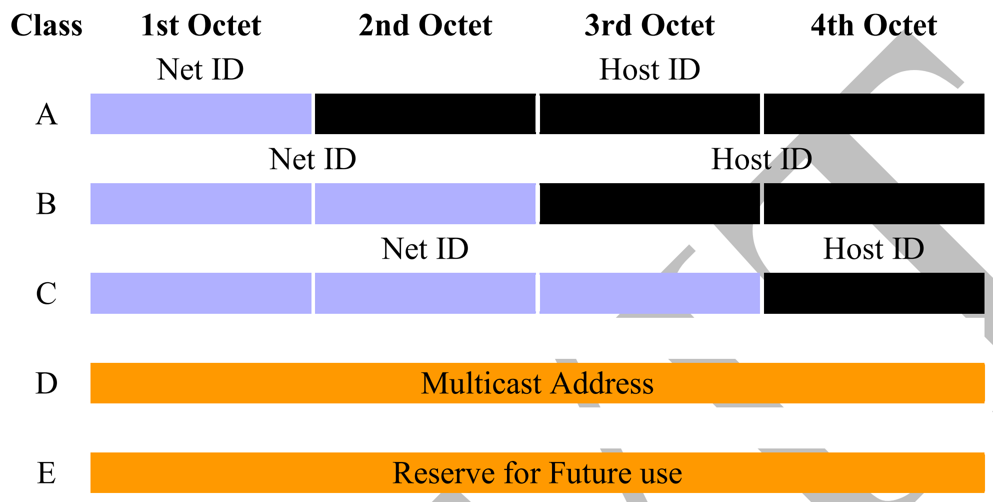


Question4.
Find the class of each address.
a. 00000001 00001011 00001011 11101111
b. 11000001 10000011 00011011 11111111
c. 14.23.120.8
d. 252.5.15.111
Solution
a. The first bit is 0. This is a class A address.
b. The first 2 bits are 1; the third bit is 0. This is a class C address.
c. The first byte is 14 (between 0 and 127); the class is A.
d. The first byte is 252 (between 240 and 255); the class is E.

There are two parts of an IP address:

- Network ID
- Host ID

The various classes of networks specify additional or fewer octets to designate the network ID versus the host ID.



Question5. Find the Network ID and Broadcast ID of different classes?
Answer.

<u>For Class A Network</u>

Network Id → X.0.0.0
Broadcast Id → X.255.255.255

<u>For Class B Network</u>

Network Id → X.X.0.0
Broadcast Id → X.X.255.255

<u>For Class C Network</u>

Network Id → X.X.X.0
Broadcast Id → X.X.X.255

Note : where X is variable ; X belongs to 0 to 127 for class A

CSE/IT

Note:

Class A

1st address → 0.0.0.0
Last Address → 127.255.255.255

So, Maximum number of network → 128
Valid number of n/w → 126
Because, we can not assign first and last address in any host machine.
First address is used for identifying a n/w (known as **network address**)
And last address is used for **broadcasting** purpose.

Class B	**Class C**	**Class D**
1st address → 128.0.0.0 Last Address → 191.255.255.255 Maximum no. of n/w→2^{16} Maximum no. of Host→2^{16}	1st address → 192.0.0.0 Last Address → 223.255.255.255 Maximum no. of n/w→2^{24} Maximum no. of Host→2^{8}	1st address → 224.0.0.0 Last Address → 239.255.255.255

Class E

1st address → 240.0.0.0 Last Address → 255.255.255.255

Network Address :

i) The network address is an address that defines the network itself.
ii) It can not be assign to a host
iii) Network address plays a very important role in classful addressing. A network address has several properties –
   a) All host bits are 0's
   b) Router can route a packet based on network address.
iv) In classful addressing, the network address is one that is assigned to the organization.
v) For network address →

Net id	Host id
Specific	All 0's

Note : Network Address = Binary AND operation of (IP Address + Subnet Mask)

Eg. 192.168.10.0 = 192.168.10.45 + 255.255.255.0

```
11000000.10101000.00001010.00101101   →192.168.10.45
11111111.11111111.11111111.00000000   →255.255.255.0
___________________________________

11000000.10101000.00001010.00000000   →192.168.10.0
```

Question 6.
Find out the network address of 132.6.17.85?
Answer:
132.6.17.85 is a class B IP address. The first 2 bytes defines the netid. We can find the n/w address by replacing the hosted bytes (17.85) with 0's. Therefore, the network address is 132.6.0.0.

MASK

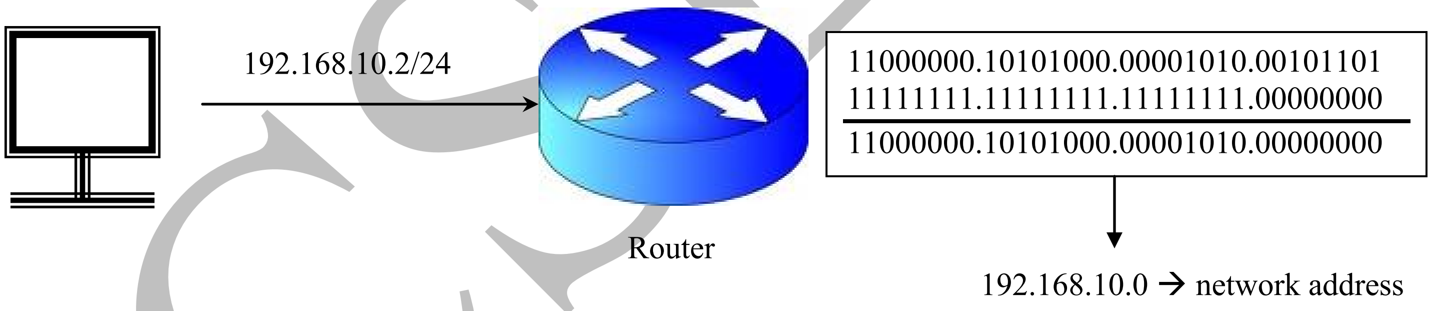
➩ When a router receives a packet with a destination address, it needs to route the packet. The routing is based on the network address and sub-network address.

➩ the router outside the organization route the packet based on the network address; and the router inside the organization routes the packet based on sub-network address.
Eg. When a parcel reaches in a post office, they are routed according to zip code. When they reach the post office serving that zip code, the parcel are routed according to the street address.

➩ Now the question is how can a router find the network address or subnetwork address?
A network administrator knows the network and sub-network address, but a router does not. A 32 bit number, called mask is the key to solve this problem.

➩ the router outside the organization use a <u>default mask</u>,; the router inside the organization use a <u>subnet mask.</u>
When a router receives a 32 bit binary IP and mask values, it performs a binary AND operation on it, and the result will be the desire network address.

Eg. →



Default Mask:

Although the length of the netid and hostid (in bits) is predetermined in classful addressing, we can also use a mask (also called the default mask), a 32-bit number made of contiguous 1s followed by contiguous 0s. The masks for classes A, B, and C are shown in Table. The concept does not apply to classes D and E.

Class	*Binary*	*Dotted-Decimal*	*CIDR*
A	11111111 00000000 00000000 00000000	255.0.0.0	/8
B	11111111 11111111 00000000 00000000	255.255.0.0	/16
C	11111111 11111111 11111111 00000000	255.255.255.0	/24

The mask can help us to find the netid and the hostid. For example, the mask for a class A address has eight 1s, which means the first 8 bits of any address in class A define the netid; the next 24 bits define the hostid. The last column of Table shows the mask in the form **/n** where n can be 8, 16, or 24 in classful addressing. This notation is

CSE/IT

also called **slash notation or Classless Interdomain Routing (CIDR) notation**. The notation is used in classless addressing, which we will discuss later. We introduce it here because it can also be applied to classful addressing.

Subnet Mask:

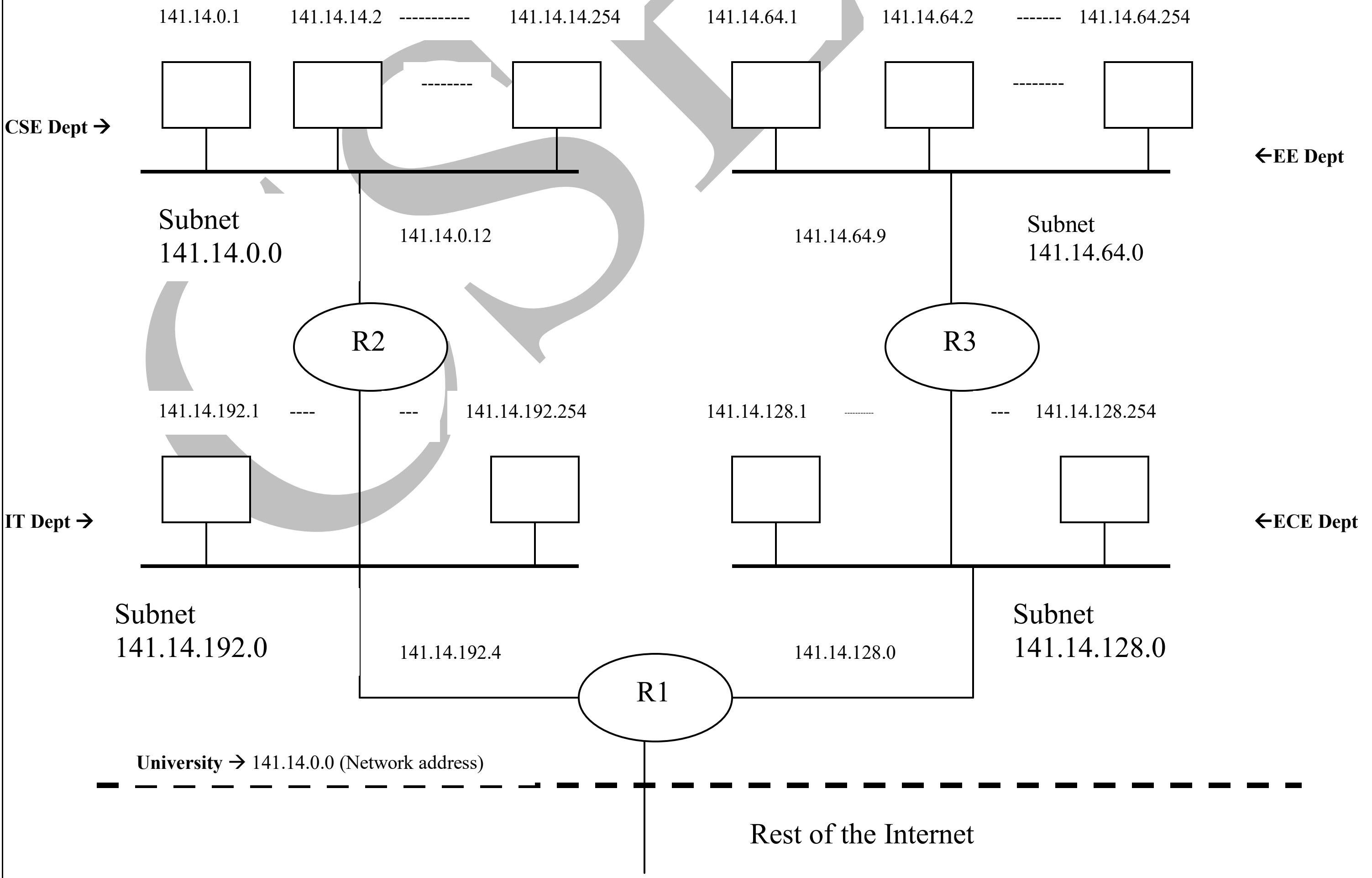
The number of 1's in a subnetmask is more than the number of 1s in the corresponding default mask.
Eg. → for class B, default mask is /16

Default mask of class B → 255.255.0.0
→ 11111111.11111111.00000000.00000000
Subnet mask → 255.255.224.0
→ 11111111.11111111.11100000.00000000

- The number of subnets is determined by the number of extra 1's.
- If the number of extra 1 is n. then the number of subnet is 2^n
- If the number of subnet is N, then the number of extra 1's is $\log_2 N$.

Subnetting:

- In subnetting, a network is divided into several smaller groups with each subnetwork (or subnet) having its own subnetwork address.
- Often an organization needs to assemble the host into groups; the network needs to be divided into several subnetworks.
- E.g. → A university may want to group its host according to departments. In this case the university has one n/w address, but needs several subnetwork addresses. The outside world knows the organization by its n/w address. Inside the organization of each subnetwork is recognized by its subnetwork address.



SE/IT

Supernetting

The time came when most of the class A and class B addresses were depleted; however, there was still a huge demand for midsize blocks. The size of a class C block with a maximum number of 256 addresses did not satisfy the needs of most organizations. Even a midsize organization needed more addresses. One solution was supernetting. In supernetting, an organization can combine several class C blocks to create a larger range of addresses. In other words, several networks are combined to create a super- network or a supernet. An organization can apply for a set of class C blocks instead of just one. For example, an organization that needs 1000 addresses can be granted four contiguous class C blocks. The organization can then use these addresses to create one supernetwork. Supernetting decreases the number of 1 s in the mask. For example, if an organization is given four class C addresses, the mask changes from/24 to/22. We will see that classless addressing eliminated the need for supernetting.

Question7: What is the difference between IP address and MAC address?
Answer.

IP Address	MAC Address
1. It is logical address.	1. It is physical address.
2. IP address is dynamic.	2. MAC address is Static.
3. IPv4 is 32 bits, and IPv6 is 128 bits.	3. MAC address is 48 bits.
4. IP address is represented by dotted decimal notation.	4. MAC address is represented by dotted hexadecimal notation.
5. It is user define address.	5. It is manufacturing address.
6. It is classify into 5 classes. i.e. class A, class B, class C, class D, class E	6. No such type of classification.
7. Each IP address has two parts. Netid and hosted.	7. No such type of division present in MAC address.

Note:

Formula to determine number of hosts on a given network

• Given that there are N host bits in an address, the number of hosts for that network is 2^N - 2. Two addresses are subtracted for the network address and the broadcast address.

• 8 host bits: 2^8 - 2 = 254 hosts

• 16 host bits: 2^{16} - 2 = 65534 hosts

• 24 host bits: 2^{24} - 2 = 16777214 hosts

CSEIT

⇘ Public IP and Private IP addresses

Public IP Address:

• Most IP addresses are public addresses. Public addresses are registered as belonging to a specific organization.

• Internet Service Providers (ISP) and extremely large organizations obtain blocks of public addresses from the IANA (Internet Assigned Numbers Authority). Other organizations obtain public addresses from their ISPs.

• Public IP addresses are routed across the Internet, so that hosts with public addresses may freely communicate with one another globally.

• No organization is permitted use public addresses that are not registered with that organization!

Private IP Address:

• The following are private addresses.
– Class A range: 10.0.0.0 through 10.255.255.255.
– Class B range: 172.16.0.0 through 172.31.255.255.
– Class C range: 192.168.0.0 through 192.168.255.255.

• Private addresses may be used by any organization, without any requirement for registration.

• Because private addresses are ambiguous - can't tell where they're coming from or going to because anyone can use them - private addresses are not permitted to be routed across the Internet.

• ISPs block private addresses from being routed across their infrastructure.

• Note: The use of private addresses, network address translation (NAT), and proxy servers solved the IP address shortage problem for the short and medium terms.

⇘ Reserved addresses

→ 0.0.0.0 is the default IP address, and it is used to specify a default route.
The default route will be discussed later (routing section).

→ Addresses beginning with 127 are reserved for internal loopback addresses.
It is common to see 127.0.0.1 used as the internal loopback address on many devices.
Try pinging this address on a PC or Unix station.

CSE/IT

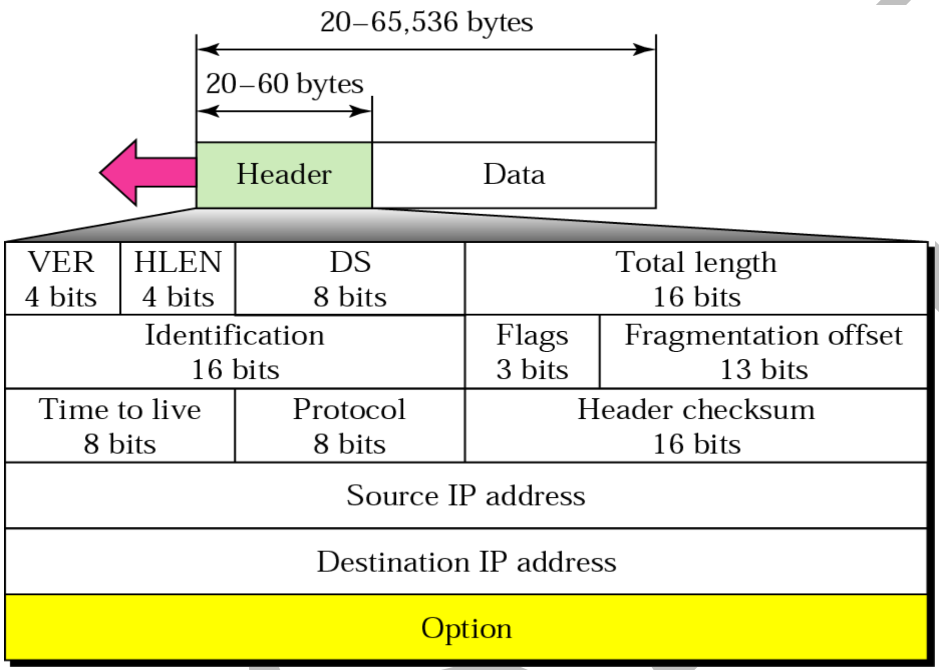
IP Datagram

What is Datagram?

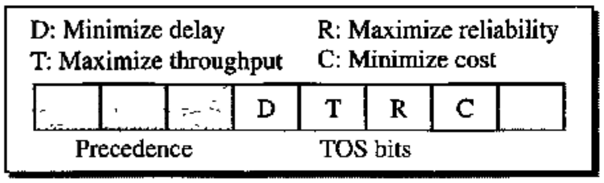
- Packets in the network layer are called Datagram.
- A datagram is a variable length packet consisting of two parts-----
  - I) Header
  - II) Data

*** the header is 20 – 60 bytes in length and contains information essential to routing and delivery.

IP datagram frame format



- **VER (version) : -** This 4bits field defines, which version of IP address we used (IPv4 or IPv6)
- **HLEN (Header Length):-** This field defines the header length, which is variable (20 – 60 bytes)
- **DS (Differentiate Services):-** This field defines the different service type.



Service type

In this interpretation, the first 3 bits are called precedence bits. The next 4 bits are called type of service (TOS) bits, and the last bit is not used.

a. Precedence is a 3-bit subfield ranging from 0 (000 in binary) to 7 (111 in binary). The precedence defines the priority of the datagram in issues such as congestion. If a router is congested and needs to discard some datagram's, those datagram's with lowest precedence are discarded first. Some datagram's in the Internet are more important than others. For example, a datagram used for network management is much more urgent and important than a datagram containing optional information for a group.

b. TOS bits is a 4-bit subfield with each bit having a special meaning. Although a bit can be either 0 or 1, one and only one of the bits can have the value of 1 in each datagram. The bit patterns and their interpretations are given in diagram bellow. With only 1 bit set at a time, we can have five different types of services.

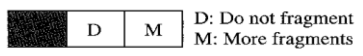
TOS Bits	*Description*
0000	Normal (default)
0001	Minimize cost
0010	Maximize reliability
0100	Maximize throughput
1000	Minimize delay

- **Total Length :-** This field defines the total length (header + Data) of the IP Datagram in bytes. Length of the IP datagram is limited to 65535 (2^{16}-1) bytes.

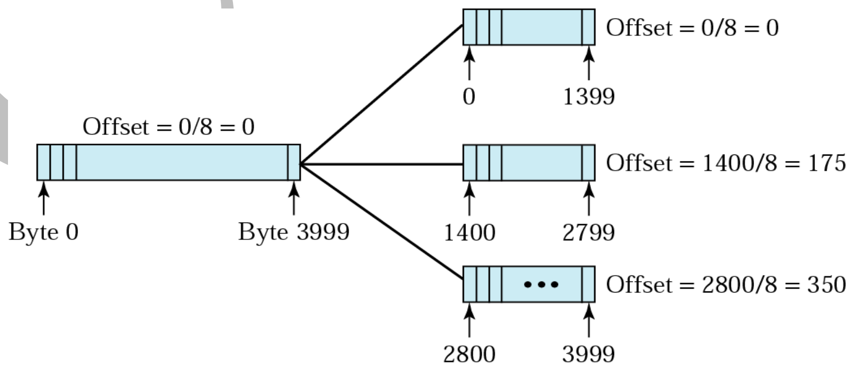
- **Identification:-** This field is required to set an identification number to each Datagram.

- **Flags:-** This is a 3-bit field. The first bit is reserved. The second bit is called the ***do not fragment*** bit. If its value is 1, the machine must not fragment the datagram. If its value is 0, the datagram can be fragmented if necessary. The third bit is called the more fragment bit. If its value is 1, it means the datagram is not the last fragment; there are more fragments after this one. If its value is 0, it means this is the last or only fragment.

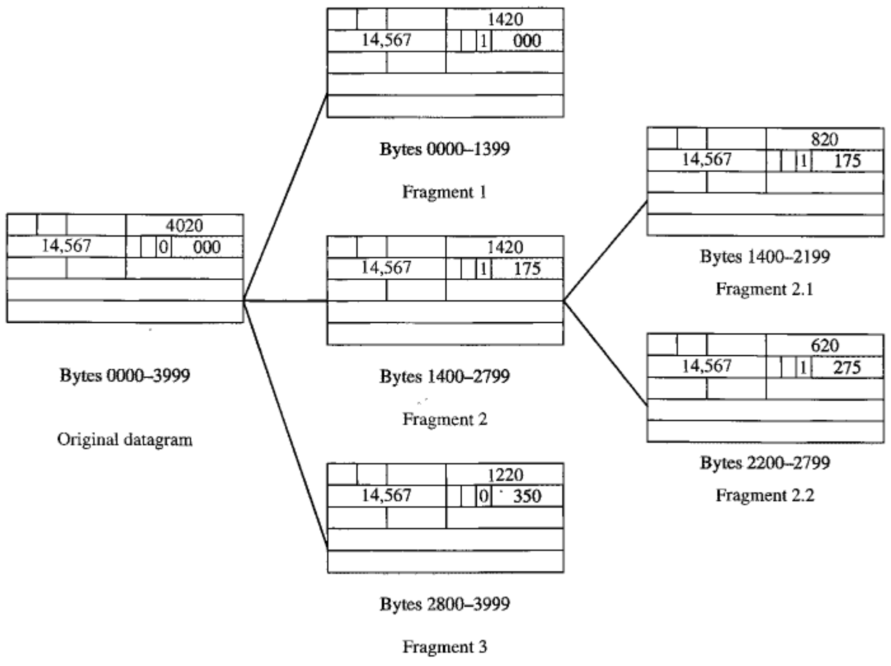
Flags used in fragmentation



- **Fragmentation offset:** This 13-bit field shows the relative position of this fragment with respect to the whole datagram. It is the offset of the data in the original datagram measured in units of 8 bytes. Figure (bellow) shows a datagram with a data size of 4000 bytes fragmented into three fragments.
The bytes in the original datagram are numbered 0 to 3999. The first fragment carries bytes 0 to 1399. The offset for this datagram is 0/8 = 0. The second fragment carries bytes 1400 to 2799; the offset value for this fragment is 1400/8 = 175. Finally, the third fragment carries bytes 2800 to 3999. The offset value for this fragment is 2800/8 = 350.



CS/IT

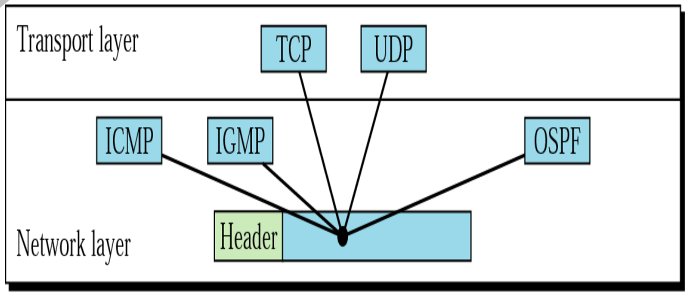


- **Time to live:-** Time to live is a limit on the period of time or number of iterations or transmission in computer and n/w technology that a unit of data (Eg. Packet) can experience before it should discarded. Or this field is used to control the maximum number of hops (routers) visited by the datagram.

- **Protocol:-** This field defines the higher-level protocol that uses the service of the IP layer. An IP datagram can encapsulate data from several higher level protocols. Such as TCP, UDP, ICMP. This field defines the final destination protocol to which the IP datagram should be delivered.

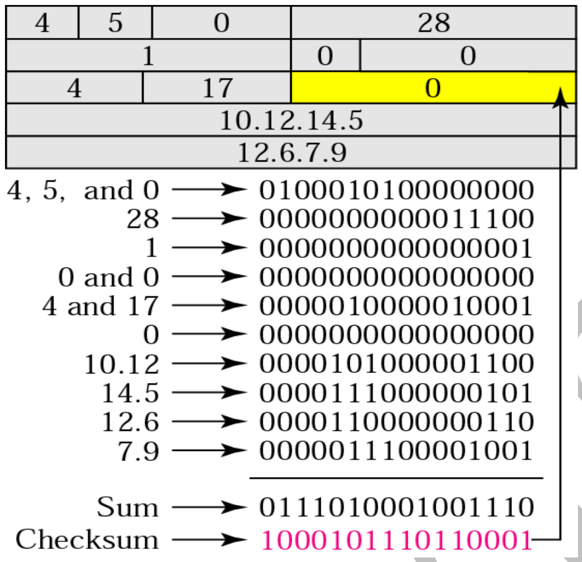
- The IP multiplexed and demultiplexed data from different higher level protocols.

Value	protocol
1	ICMP
2	IGMP
6	TCP
17	UDP
89	OSPF



CSE

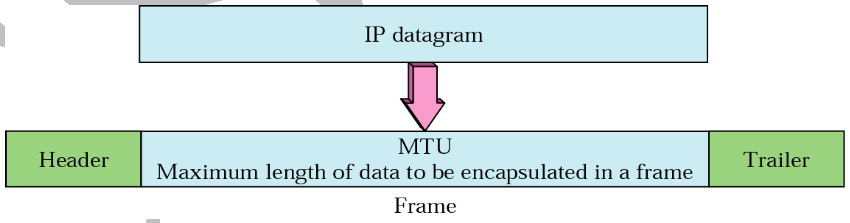
- **Checksum:-** Checksum field is used for error detection purpose in IP datagram. The header of IP packets changes with each visited router, but data do not. So, the checksum include only the part that has changed.



- **Source Address**:- This field defines the source IP address, and this field remain unchanged during the time the IP datagram travels from source host to destination host.

- **Destination address:-** This field defines the IP address of the destination

- **Options:-** As name implies, are not required for every datagram. They are used for network testing and debugging.



CSE/IT

Prepared by M.R.Chakraborty
Not for distribution without permission

Introduction

Basics of electricity

M.R.Chakraborty
Department of EE
Siliguri Institute of Technology

Outline

Prepared by M.R.Chakraborty
Not for distribution without permission

Prepared by M.R.Chakraborty
Not for distribution without permission

1 About electricity

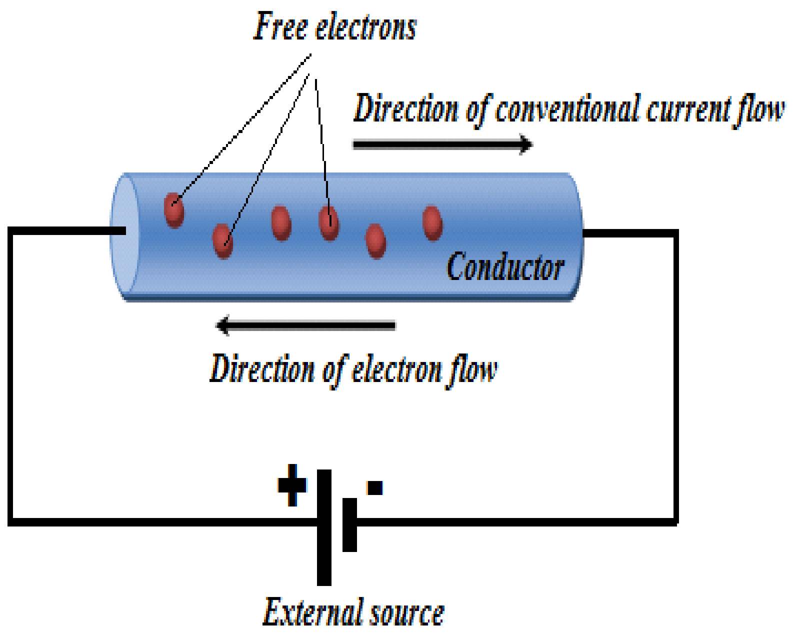
Electrical energy is a secondary energy source and also referred to as an energy carrier.

Electrical energy has the following advantages over all other forms (chemical, heat, light & mechanical) of energy

1. Cheapness
2. Convenient & efficient transmission
3. Easy control
4. Cleanliness
5. Greater flexibility – as it can be taken to any corner of the house, factory etc.
6. Utilization in versatile form

Prepared by M.R.Chakraborty
Not for distribution without permission

1 About electricity


Free electrons
Direction of conventional current flow
Conductor
Direction of electron flow
+
-
External source

By definition **electric current is the rate of flow of charge through a medium subjected to an external influence**. Mathematically

$$I = \frac{Q}{t}$$

Where,
I = Average current flow through the medium
Q = Charge
t= time

Definition of Ampere: When a charge of one coulomb passes through a medium in one second, the medium is said to be carrying a current of one Ampere

As 1 coulomb is equal to 6.24×10^{18} number of electrons, so 1 Ampere means flow of 6.24×10^{18} number of electrons

1 About electricity

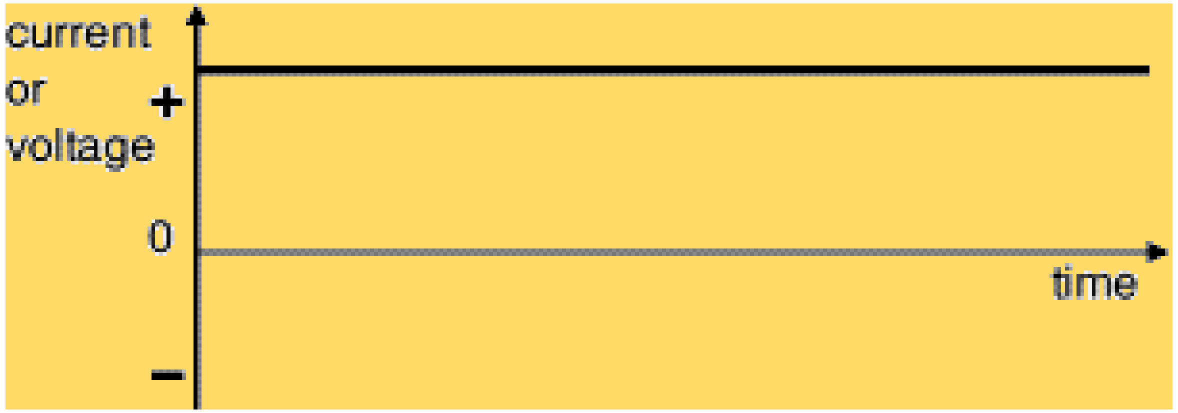
Prepared by M.R.Chakraborty
Not for distribution without permission

Electric current is divided into two types:

- **Directional Current (DC)**
- **Alternating Current (AC)**

Directional (Direct) Current

A non-varying, unidirectional electric current (Example: Current produced by batteries)



Characteristics:

- Direction of the flow of positive and negative charges does not change with time
- Direction of current (direction of flow for positive charges) is constant with time
- Potential difference (voltage) between two points of the circuit does not change polarity with time

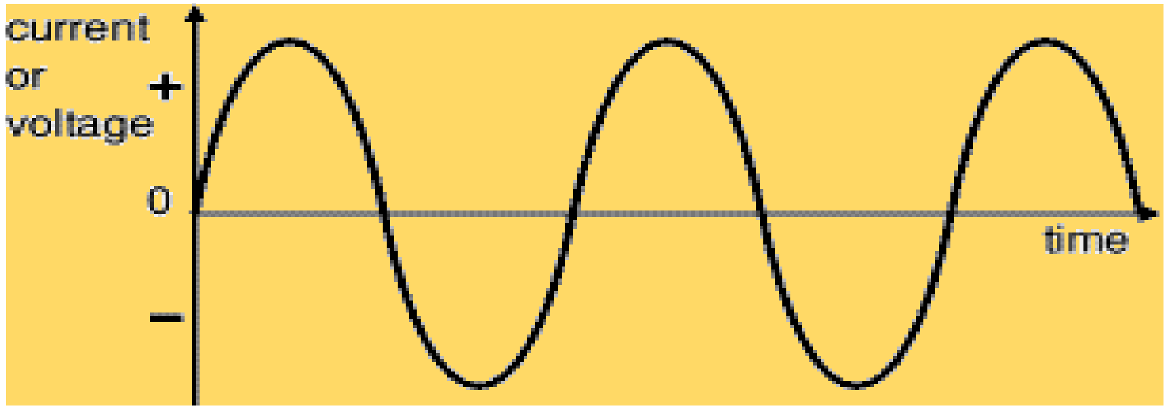
1 About electricity

Prepared by M.R.Chakraborty
Not for distribution without permission

Alternating Current

A current which reverses in regularly recurring intervals of time and which has alternately positive and negative values, and occurring a specified number of times per second.

(Example: Household electricity produced by generators, Electricity supplied by utilities.)



Characteristics:

- Direction of the current reverses periodically with time
- Voltage (tension) between two points of the circuit changes polarity with time.
- In 50 cycles AC, current reverses direction 100 times a second (two times during one cycle)

2 Ohm’s law

Prepared by M.R.Chakraborty
Not for distribution without permission

The potential difference (V) between any two points of a conductor is directly proportional to the current (I) flowing through the conductor provided temperature and all other physical quantities (e.g. length, cross – sectional area etc.) remain unchanged.

Mathematically we can write, $V \alpha I$

$$or,\ V = RI$$

R is the constant of proportionality & termed as 'Resistance'

Resistance is the property of a material by virtue of which it opposes the flow of electrons through it. The unit of Resistance is “ohm” (Ω).

From Ohm's Law $R = \dfrac{V}{I}$

For V = 1 Volt & I = 1 Amp

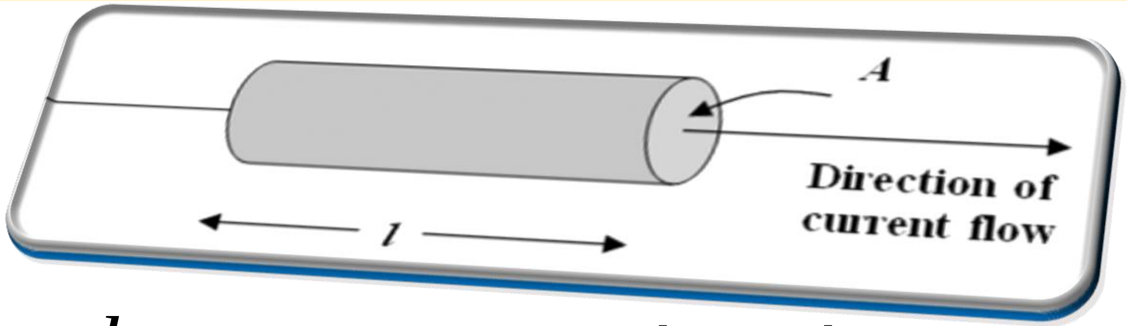
R=1 ohm

Thus **1 ohm is defined** as the amount of resistance which opposes a flow of 1 amp current through a conductor when the potential difference across the conductor is 1 Volt.

2.1 Laws of Resistance

Prepared by M.R.Chakraborty
Not for distribution without permission

- Resistance of a conductor is directly proportional to its length (*l*) in the direction of the current flow.
- Resistance of a conductor is inversely proportional to its cross-sectional area (A) perpendicular to the direction of flow of current



Thus from the above $R \, \alpha \, l$

and $R \, \alpha \, \frac{1}{A}$

Combining the two we get $R \alpha \frac{l}{A}$

Or, $R = \rho \frac{l}{A}$

Where 'ρ' is the proportionality constant & is called ***resistivity or specific resistance*** of the conductor.

If R = 1 ohm , l = 1 meter, A = 1 sq meter, Then, ρ = 1 ohm-meter $\longrightarrow$ So ***unit of resistivity is ohm-meter***.

Again, if l = 1 meter and A = 1 meter^2 , Then $R = \rho$

So **specific resistance or resistivity of a material is defined** as the value of resistance between two opposite faces of a cube of that material with a dimension of (1 m) x (1 m) x (1 m).

2.2 Conductance and Conductivity

Prepared by M.R.Chakraborty
Not for distribution without permission

Conductance is the reciprocal of Resistance.

From ohm's law the current through a conductor is given by $I = \frac{V}{R}$

Or, $I = GV$, Where $G = \frac{1}{R}$

The parameter **'G'** is called **conductance of the conductor** and it is reciprocal of resistance R. Conductance is the measurement of inducement which it offers to its flow but resistance is the measurement of opposition which it offers to the flow of current.

Unit of conductance is Siemens or mho.

Again, from the laws of resistance, $R = \rho . \frac{l}{A}$

Or $\frac{1}{G} = \rho \frac{l}{A}$

Or, $G = \frac{1}{\rho} . \frac{A}{l} = \sigma \frac{A}{l}$

The parameter **'σ' is called conductivity**.
It is *reciprocal of resistivity*.
Unit of conductivity is Siemens/meter.

2.3 Temperature co-efficient of Resistance

Prepared by M.R.Chakraborty
Not for distribution without permission

The ratio of change in resistance per ^{O}C and the resistance at the reference temperature is called temperature co-efficient of resistance (α) of a material.

Let, R_1 = resistance at $t_1{}^{O}C$
R_2 = resistance at $t_2{}^{O}C$
α_1 = temp coefficient of resistance at $t_1{}^{O}C$
α_2 = temp coefficient of resistance at $t_2{}^{O}C$

$$\mathbf{R_2 = R_1 (1+ \alpha t)} \qquad [\text{where, } t = t_2 - t_1]$$

Generally reference or base temp is taken as $0^{O}C$.

Let, α_0 = temp coefficient of resistance at base temp $0^{O}C$
R_0 = resistance of base / reference temp of $0^{O}C$
R_t = resistance of the material at any temp $t\,^{O}C$

$$\mathbf{R_t = R_0 (1+ \alpha_0 t)}$$

Similarly we can say, $\rho_t = \rho_0(1+\alpha_0 t)$

Also we can arrive at $\alpha_t = \dfrac{\alpha_0}{1+\alpha_0 t}$

Temp co-efficient of resistance is given by $\alpha_0 = \dfrac{R_t - R_0}{R_0 t}$

Temp. co-efficient of resistance is
- **i. positive for metallic conductors**
- **ii. negative for carbon, electrolytes, semiconductors, insulators**
- **iii. negligibly small for alloys like eureka, manganese, nickel chromium, constantan etc.**

3.1 Resistance in Circuits - Series connection

Prepared by M.R.Chakraborty
Not for distribution without permission

When the resistances are connected in series:

- Current through equivalent resistance R_{eq} is equal to the current through each of the original resistors (all have same current).
- Voltage across the equivalent resistance R_{eq} is the sum of the voltages over the original resistors.

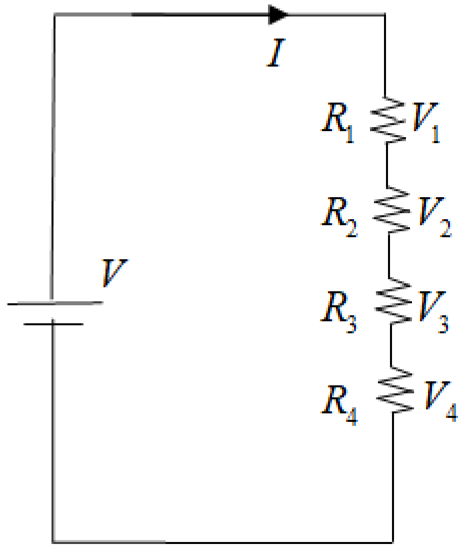
From figure, $V = V_1 + V_2 + V_3 + V_4$

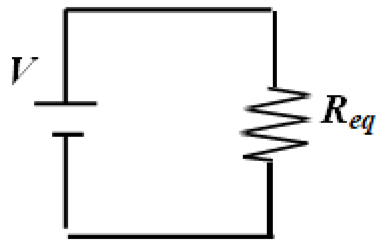
Or, $IR_{eq} = IR_1 + IR_2 + IR_3 + IR_4$

[Where, 'R' is the equivalent resistance of the circuit]

or, $R_{eq} = R_1 + R_2 + R_3 + R_4$

Or, $$\frac{1}{G_{eq}} = \frac{1}{G_1} + \frac{1}{G_2} + \frac{1}{G_3} + \frac{1}{G_4}$$





Criterion of series circuit

i. Same amount of current should flow through all the elements of the circuit.
ii. Individual resistors have their individual voltage drops.
iii. Nature of voltage drops are additive.
iv. Summation of all voltage drops must equal to the supply voltage.
v. Resistances are additive.

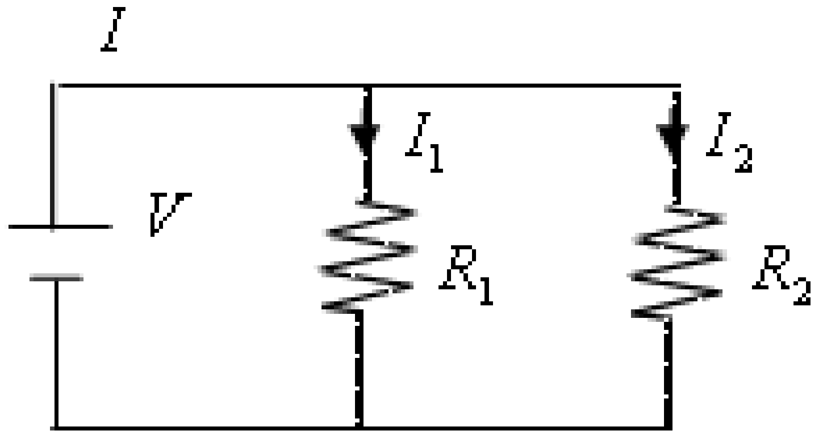
3.2 Resistance in Circuits - Parallel connection

Prepared by M.R.Chakraborty
Not for distribution without permission

When the resistances are connected in parallel:

- The individual resistors have the same voltage as the single equivalent resistor.
- The current through the equivalent resistor is the sum of the currents through the individual resistors.
- Individual voltages and currents can be recovered using Ohm's law or current division.

From figure, $\mathbf{I = I_1 + I_2}$



or, $$\frac{V}{R_{eq}} = \frac{V}{R_1} + \frac{V}{R_2}$$

[Where, ' R_{eq} ' is the equivalent resistance of the circuit]

or, $$\frac{1}{R_{eq}} = \frac{1}{R_1} + \frac{1}{R_2}$$

In terms of conductance the equivalent conductance will be $G_{eq} = G_1 + G_2$

Criterion of parallel circuit

i. **Same potential difference is applied across all the resistances**
ii. **Different resistances have their individual currents**
iii. **Conductances are additive.**

4.1 Basic quantities: Potential & potential difference

Prepared by M.R.Chakraborty
Not for distribution without permission

The amount of work done to move a unit charged particle from a reference point to a designated point in a static electric field is called **potential**.

The difference in electric potential between two points in an electric field is called **potential difference**.

It is measured in **volts**.

4.2 Basic quantities: Work

Prepared by M.R.Chakraborty
Not for distribution without permission

In an electric circuit, current flows through a conductor when potential difference is applied to it i.e. some work is done. So we can say that **whenever there is transfer of charge, some electric work is done**.

Electric work is given by, $W = VQ = VIt$

Where,	V = potential (volt)
	Q = charge (coulomb)
	I = current (amp)
	t = time (sec)

The unit of electrical work is **Joule**.

4.3 Basic quantities: Power

Prepared by M.R.Chakraborty
Not for distribution without permission

The **rate at which electric work is performed** is called electric power.

It is given by, $\mathbf{P=\dfrac{amount\ of\ work\ done}{time\ to\ do\ the\ work}=\dfrac{W}{t}=\dfrac{VIt}{t}=VI}$

The unit of electric power is **Watt**.

4.3 Basic quantities: Energy

Prepared by M.R.Chakraborty
Not for distribution without permission

Total **amount of electric work done in a certain period of time** is called electric energy.

It is given by, **E = power×time =P×t=VIt**

The unit is **watt-sec**. But watt-sec unit is very small if we think of the amount of electric energy consumed. So we use bigger unit like **KWh**.

1KWh = 1000 Wh = 1000 x 3600 watt-sec = 36 x 10^5 watt-sec.

Prepared by M.R.Chakraborty
Not for distribution without permission

5 Brief history of electricity

Electricity

Thales	BC624~546	Triboelectric effect
Gray	1670~1736	Division of conductor and insulator
Du Fay	1698~1739	Electric fluid (vitreous and resinous fluid)
Franklin	1706~1790	Electric charge and conservation of charge
Galvani	1737~1798	Biological electricity
Coulomb	1736~1806	Electrostatic force
Volta	1745~1827	Chemical electricity, battery
Oersted	1777~1851	Magnetic phenomenon induced from the electric wire
Ampere	1775~1836	Magnetic field
Faraday	1791~1867	Electromagnetic induction
Maxwell	1831~1879	Electromagnetic field
Hertz	1887~1975	Electromagnetic wave
Tesla	1856~1944	Alternative generator, wireless communication
Einstein	1879~1955	Photoelectric effect

Magnetism

Gilbert	1540~1603	magnetic field of the Earth

Numerical

Prepared by M.R.Chakraborty
Not for distribution without permission

1. **The resistance temperature co-efficient of aluminium at 0^0C is 0.0045 per ^{0}C. Find the co-efficient for a temperature of 30^0 C.**

Solution:

We know that,

$$\alpha_t = \frac{\alpha_0}{1+\alpha_0 t}$$

Here $\alpha_0 = 0.0045, t = 30^0$

Therefore, $\alpha_{30} = \dfrac{\alpha_0}{1+\alpha_0 \times 30} = \dfrac{0.0045}{1+0.0045\times 30} = 0.00396$

6 Numerical

Prepared by M.R.Chakraborty
Not for distribution without permission

2. A conductor has a resistance of 10 ohm at 20^0 C. At 50^0 C its resistance increases to 14 ohm. Determine the temperature co-efficient of the conductor at 0^0 C.

Solution:

We know that,

$R_t = R_0\,(1+\alpha_0 t)$

Here,

$R_{20} = 10\Omega$

$R_{50} = 14\Omega$

Therefore,

$$R_{20} = R_0(1+\alpha_0 \times 20) = 10$$

$$R_{50} = R_0(1+\alpha_0 \times 50) = 14$$

$$\therefore \frac{1+\alpha_0 \times 20}{1+\alpha_0 \times 50} = \frac{10}{14}$$

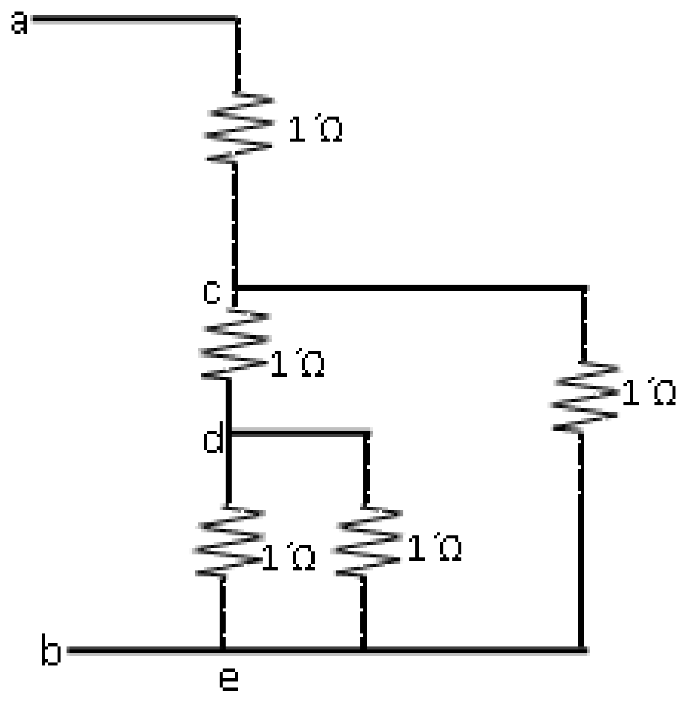
$$or, 10 + 500\alpha_0 = 14 + 280\alpha_0$$

$$or, 220\alpha_0 = 4$$

$$or, \alpha_0 = \frac{4}{220} = 0.0182\,/^0\,C$$

Prepared by M.R.Chakraborty
Not for distribution without permission

3. Determine the resistance across ‘ab’ for the circuit shown in the figure.



Solution:

This is a series-parallel combination of resistances.

The resistance across ***de***, $R_{de} = \frac{1\times1}{1+1} = 0.5\Omega$

The resistance R_{de} is in series with R_{cd}.

So, the resistance, $R_{cd} + R_{de} = 1 + 0.5 = 1.5\Omega$

Now, the total resistance across ***ce***, $R_{ce} = \frac{1.5\times1}{1.5+1} = \frac{1.5}{2.5} = 0.6\Omega$

But this resistance R_{ce} is in series with R_{ac}.

So the resistance across ***ab*** is given by,

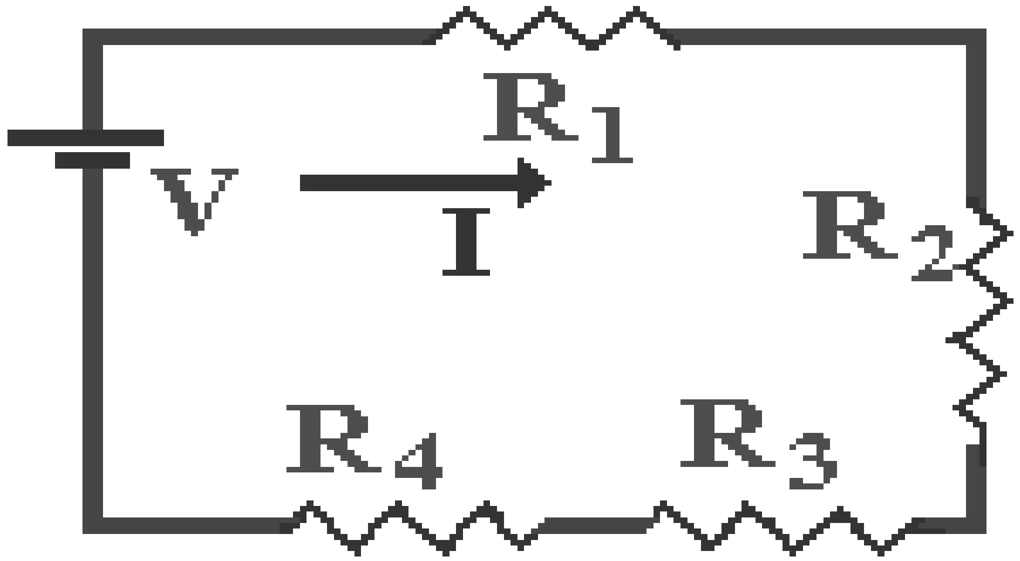
$$R_{ab} = R_{ac} + R_{ce} = 1 + 0.6 = 1.6\Omega$$

Assignment

Prepared by M.R.Chakraborty
Not for distribution without permission

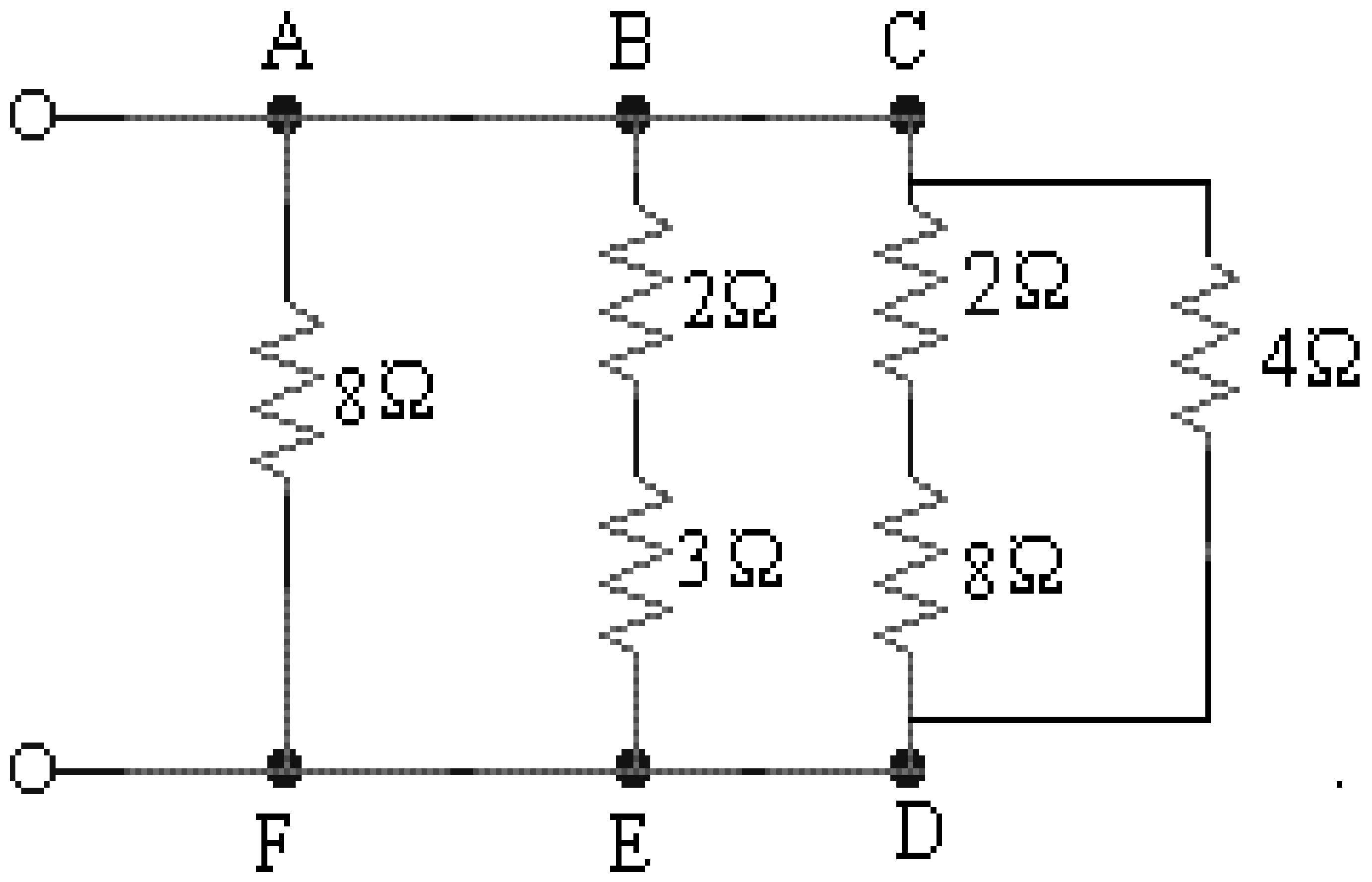
1. The current flowing in a circuit containing four resistors connected in series is I = 2 Amp. The potential drops across the first, second and third resistors are, respectively: V_1 = 8 V, V_2 = 10V and V_3= 6V. The equivalent resistance of the circuit is R = 30 Ω.

   Find the total voltage supplied by the battery and also current, voltage drop, and resistance of each resistor in the circuit.



Prepared by M.R.Chakraborty
Not for distribution without permission

2. Find the equivalent resistance of the circuit shown.



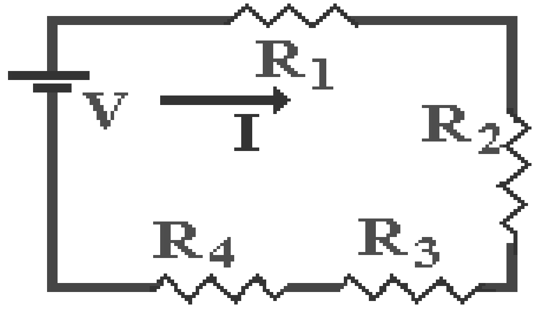
6 Solution of assignment

Prepared by M.R.Chakraborty
Not for distribution without permission

1. **The current flowing in a circuit containing four resistors connected in series is I = 2 Amp. The potential drops across the first, second and third resistors are, respectively: V_1 = 8 V, V_2 = 10V and V_3= 6V. The equivalent resistance of the circuit is R = 30 Ω.**

   Find the total voltage supplied by the battery and also current, voltage drop, and resistance of each resistor in the circuit.

Solution:



As the resistors are in series, so the current flow through each resistance is same.

Using the Ohm's Law, we can find the resistances of the first, second and third resistors.

$$R_1 = \frac{V_1}{I} = \frac{8}{2} = 4\Omega$$
$$R_2 = \frac{V_2}{I} = \frac{10}{2} = 5\Omega$$
$$R_3 = \frac{V_3}{I} = \frac{6}{2} = 3\Omega$$

As it is a series circuit, so the equivalent resistance is the sum of the individual resistances.

$$R_{eq} = R_1 + R_2 + R_3 + R_4$$
$$\text{or}, R_4 = R_{eq} - (R_1 + R_2 + R_3)$$
$$\text{or}, R_4 = 30 - (4 + 5 + 3) = 18\Omega$$

The current flowing through R_4 is also 2A as it is a series circuit.
Using Ohm's Law again, we can find the voltage across R_4 as below.

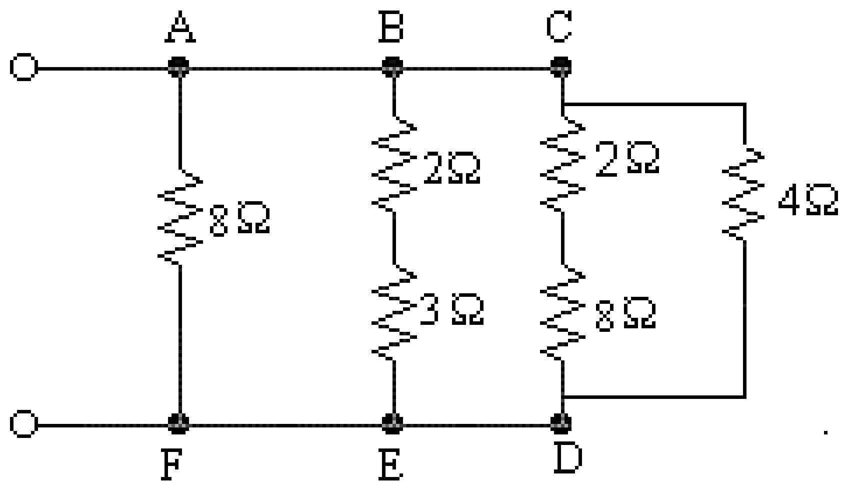
$$V_4 = IR_4 = 2 \times 18 = 36V$$

Therefore, total voltage V = V_1 + V_2 + V_3 + V_4 = 8+10+6+36 = 60 V

Solution of assignment

Prepared by M.R.Chakraborty
Not for distribution without permission

2. Find the equivalent resistance of the circuit shown.



Solution:

Equivalent resistance $R_{eq} = R_{AF} \parallel R_{BE} \parallel R_{CD}$

From the circuit given,

$R_{AF} = 8\ \Omega$

$R_{BE} = (2 + 3)\ \Omega = 5\ \Omega$ (as it is a series connection)

$$R_{CD} = 4\|(2+8) = 4\|10 = \frac{4 \times 10}{4+10} = 2.86\Omega$$

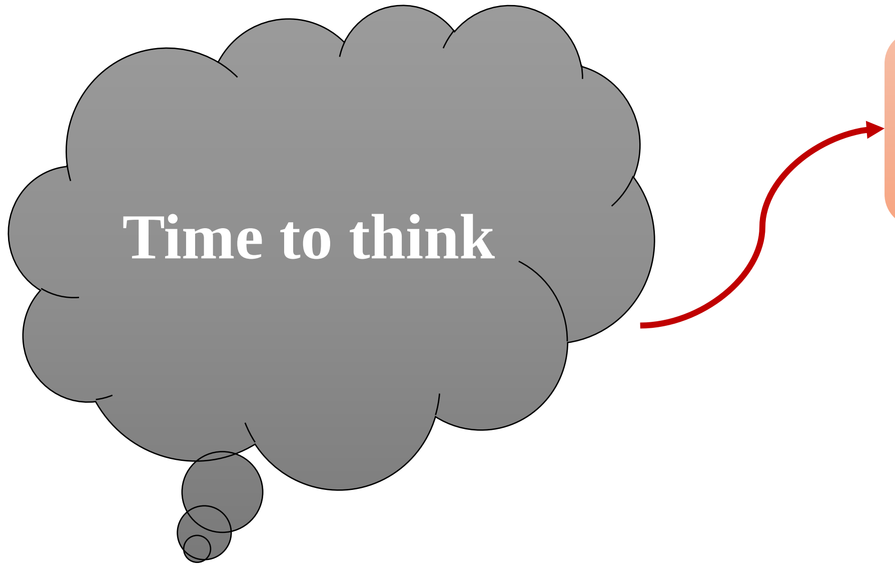
Therefore, **$R_{eq} = R_{AF} \parallel R_{BE} \parallel R_{CD} = (8 \parallel 5 \parallel 2.86)\ \Omega = 1.48\ \Omega$**

Prepared by M.R.Chakraborty
Not for distribution without permission

Sample questions

1. What do you mean by current? What is charge? Explain the relationship between charge and current.
2. Write down the classification of current. Explain in brief.
3. Discuss the differences between a.c. and d.c.
4. Explain Ohm’s law. Define resistance from it.
5. What do you mean by resistance? Name the factors affecting resistance.
6. What do you understand by resistivity, temperature co-efficient of resistance and conductivity.

Prepared by M.R.Chakraborty
Not for distribution without permission



Did I learn the topics properly?

Confused how to judge that?

Prepared by M.R.Chakraborty
Not for distribution without permission

How to Conduct a self-check on learning

1. Did I understand all the topics discussed in the class?
2. Did I understand the concepts clearly?
3. Can I apply the theoretical knowledge acquired in practical field?
4. Am I able to solve the numerical problems related to the topics?

Prepared by M.R.Chakraborty
Not for distribution without permission

If the answers are YES to the questions of previous slides

Congratulations !

You are on correct path of learning.

Don’t hesitate to

Prepared by M.R.Chakraborty
Not for distribution without permission

There is no alternate of studying books…….

Go through your book to know more about it.

Prepared by M.R.Chakraborty
Not for distribution without permission

Thank you

Mendelian genetics

Epistasis

Epistasis is a form of genetic interaction in which one gene masks the phenotypic expression of another.

or

Epistasis is an interaction between two non allelic genes in which one gene supresses the expression of another affecting the same character.

The expressed gene is called epistatic, while the supressed gene is said to be hypostatic.

Difference between Dominance and epistasis →

Dominance	Epistasis
i) Involves intra-allelic gene interaction.	i) Involves inter-allelic gene interaction.
ii) One allel hides the effect of other allele at the same gene pair.	ii) One gene hides the effect of other gene at different gene loci.

Types of Epistasis →

i) Dominant Epistasis (12:3:1) → A dominant allele (eg. A), of gene hides the effect of allele of another gene (eg. B) and express itself phenotypically.

⇒ The B allele (hypostatic) will be expressed only when gene locus A contains two recessive (aa) alleles.

⇒ Thus genotype AA BB or Aa Bb and AAbb or Aa bb produce the same phenotype.

⇒ Genotype aaBB or aa Bb and aabb produce two additional Phenotype.

Scanned with CamScanner

This type of dominant epistasis modifies the classical ratio of 9:3:3:1 into 12:3:1.

Epistatic alleles	Hypostatic alleles	Phenotypic expression
aa	bb	b
aa	BB, Bb	B
AA, Aa	BB, Bb, bb	A

Example →

Studied in summer squash (<u>cucurbita pepo</u>)

→ Common fruit color - white, yellow and green.
→ White (W) is dominant over, Colored squash.
→ Yellow (Y) is dominant over, green squash.
→ Pure breeding white fruited variety is crossed with the double recessive green variety, F_1 hybrid are all white.
→ When the hybrid are selfed - white, yellow and green fruited plants arise in the ratio of 12:3:1.

The effect of dominant gene 'Y' is masked by the dominant gene 'W' (epistatic gene)

* P WWYY x wwyy
(white) (green)

F_1: WwYy
(white)

gametes → WY, Wy, wY, wy

♂/♀	WY	Wy	wY	wy
WY	WWYY white	WWYy white	WwYY white	WwYy white
Wy	WWYy white	WWyy white	WwYy white	Wwyy white
wY	WwYY white	WwYy white	wwYY Yellow	wwYy Yellow
wy	WwYy white	Wwyy white	wwYy Yellow	wwyy green

F_2: White : Yellow : Green
12 : 3 : 1.

Scanned with CamScanner

ii) <u>Recessive epistasis (9:3:4)</u> → Recessive epistasis occurs when the recessive alleles of one gene locus (aa- the epistasis locus) supress the phenotypic expression of the alleles of another gene. (BB, Bb or bb alleles). This type of epistasis is called recessive epistasis.

▨ let us guess, the four phenotypic classes correspond to the genotype A_B_, A_bb, aaB_, and aaB_. If either of the one singly homozygous recessive genotype (i.e A-bb or aaB_) has the same phenotype as the double homozygous recessive (aabb), then a <u>9:3:4 phenotype</u> ratio will be obtained.

▨ for eg - in the labrador Retriever breed of dogs, the B encodes a gene for an importance step in the production of melanin. The dominant allele, B is more efficient at pigment production than the recessive b allele, thus B hair appears black, and bb appears brown. A second locus, which we call E, controls the deposition of melanin in the hairs. At least one functional E allele is required to deposite any pigment, wheather it is black or brown. Thus, all retrievers that are ee fail to deposit any melanin (and so appear pale yellow), regardless of the genotype of the B locus.

The ee genotype is therefore said to be <u>epistatic</u> to both the B and b alleles. Since the homozygous ee phenotype masks the phenotype of the B locus. The B/b locus is said to be <u>hypostatic</u> to the ee genotype. Because the masking allele is in this case is recessive, this is called <u>recessive epistasis</u>.

So,

$$\begin{tikzcd}
\boxed{BBEE} \arrow[dr, no head] & \times & \boxed{bbee} \arrow[dl, no head] \\
 & \boxed{BbEe} \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] & & \\
BE & Be & bE & be
\end{tikzcd}$$

Black   golden (pale yellow)
                     (no pigment)

F1 generation.

gametes. (Black)

F2 generation.

B- Black.
bb- brown
ee - golden/pale yellow.

9 : 3 : 4
Black Brown Yellow.

	BE	Be	bE	be
BE	BBEE Black	BBEe Black	BbEE Black	BbEe Black
Be	BBEe Black	BBee Yellow	BbEe Black	Bbee Yellow
bE	BbEE Black	BbEe Black	bbEE brown	bbEe brown
be	BbEe Black	Bbee Yellow	bbEe brown	bbee yellow

iii) Duplicate Recessive Gene (9:7) → If both gene loci have homozygous recessive alleles and both of them produce identical phenotype the F2 ratio 9:3:3:1 would be 9:7. The genotype aaBB, aaBb, AAbb, Aabb and aabb produce same phenotype. Both dominant alleles when are present together only then they can complement each other This is known as complementary gene.

It is called complementary gene ~~dt~~ because the function of either A or B gene function has the same phenotype and they work together to produce a final product.

For, example consider a biochemical pathway, in which a colorless substrate is converted by the action of gene A to another colorless product, which is then converted by the action of gene B to a visible pigment. loss of function of either A or B or both. will have same result. No pigment production

$$\begin{tikzcd}\text{Colorless Compound} \arrow[r, "\text{gene A}"] & \text{colorless Compound} \arrow[r, "\text{gene B}"] & \text{Purple Pigment}\end{tikzcd}$$

Fig: simplified pathway showing complementary gene action of A and B.

$$\begin{tikzcd}
\text{F1 generation: } \boxed{AAbb} \text{ (white flower)} \arrow[dr] & \times & \boxed{aaBB} \text{ (white flower)} \arrow[dl] \\
& \boxed{AaBb} \text{ (Purple color flower)} \arrow[dll, "\text{gametes}"'] \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{F2 generation: } AB & Ab & aB & ab
\end{tikzcd}$$

ratio: 9:7
(purple : white)

♀/♂	AB	Ab	aB	ab
AB	AABB purple	AABb purple	AaBB purple	AaBb purple
Ab	AABb purple	AAbb white	AaBb purple	Aabb white
aB	AaBB purple	AaBb purple	aaBB white	aaBb white
ab	AaBb purple	Aabb white	aaBb white	aabb white

$$\begin{tikzcd}
\text{Colorless precursor} \arrow[r, "\text{Allele A}", "\text{pigment change catalyse}"'] & \text{Colorless Precursor Q} \arrow[r, "\text{Allele B}", "\text{pigment change complete}"'] & \text{Purple pigment}
\end{tikzcd}$$

In, this case dominant alleles on both locus are required hence wherever A and B both are present they result into purple effect masking the white.

IV) Duplicate Dominant Gene (15:1) → The dominant alleles of both the gene produce the same phenotypic effect giving the ratio 15:1.

In this case at least one of the dominant allele is necessary for the phenotypic effect e.g AABB, AaBb, Aabb, aaBB, aaBb give one phenotype.

In the absence of all the dominant gene (only in case of aabb) the recessive phenotype will be expressed. The duplicate gene are also called psedoalleles.

Yet, another pigmentation pathway, in this case, in wheat, provides an example of this duplicate gene action. The biosynthesis of red pigment near the surface of wheat seeds involves many gene, two of which we will label A and B. Normal, red coloration of the wheat seed is maintained if function of either of these genes is lost in homozygus mutant (e.g in either aa B–, or A – bb). Only the doubly recessive mutant (aabb), which lacks function of both genes, shows a phenotype that differs from that produce by any of the other genotypes. A resonable interpretation of this result is that both gene encodes the same biological function, and either one alone is sufficient for the normal activity of that pathway.

F1 progeny:

$$\begin{tikzcd}
\text{AABB (Red)} \arrow[dr] & \times & \text{aabb (White)} \arrow[dl] \\
& \boxed{\text{AaBb}} \text{ (Red)} \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr, "\text{gametes}"'] & \\
\text{AB} & \text{Ab} & \text{aB} & \text{ab}
\end{tikzcd}$$

F2 progeny.

	AB	Ab	aB	ab
AB	AABB Red	AABb Red	AaBB Red	AaBb Red
Ab	AABb Red	AAbb Red	AaBb Red	Aabb Red
aB	AaBB Red	AaBb Red	aaBB Red	aaBb Red
ab	AaBb Red	Aabb Red	aBab Red	aabb White

F2 generation ratio →

Red : white
15 : 1

V) Polymeric gene interaction (9:6:1) → Two dominant alleles have similar effect when they are separate, but produce enhanced effect when they come together. Such gene interaction is known as polymeric gene interaction. The joint effect of two alleles appears to be additive or cumulative, but each of the two gene show complete dominance, hence they cannot be considered as additive genes. In case of additive effect, gene show lack of dominance.

A well known example of polymeric gene interaction is fruit shape in summer squash. There are three types of fruit shape in this plant, viz, disc, spherical and long. The disc shape is controlled by two dominant genes (A and B), the spherical shape is produced by either dominant allele (A or B) and long shaped fruits develop in double recessive (aabb) plants.

A cross between disc shape (AABB) and long shape (aabb) strains produce disc shape fruits in F1. Inter-mating of F1 plants produced plants with disc, spherical, and long shape fruits in 9:6:1 ratio in F2. This can be explained as follow →

F_1 progeny

$$\begin{tikzcd}
 & \boxed{AABB} \text{ (Disc shape fruit)} & \times \arrow[d] & \boxed{aabb} \text{ (Long shape fruit)} & \\
 & & \boxed{AaBb} \text{ (Disc shape fruit)} \arrow[dll, "\text{gametes}"'] \arrow[dl] \arrow[dr] \arrow[drr] & & \\
AB & Ab & & aB & ab
\end{tikzcd}$$

F2 generation

Disc : spherical : long
9 : 6 : 1.

	AB	Ab	aB	ab.
AB	AABB Disc	AABb Disc.	AaBB Disc	AaBb Disc
Ab	AABb Disc	AAbb spherical	AaBb Disc	Aabb. spherical
aB	AaBb Disc	AaBb Disc	aaBB spherical	aaBb. spherical
ab.	AaBb Disc	Aabb spherical	aaBb spherical	aabb. Long

Here, plants with A- B– (9/16) genotypes produce disc shape fruits, those with A-bb-(3/16) and aaB-(3/16) genotypes produce spherical fruits and plants with aabb(1/16) genotype produce long fruits. Thus, in F2, normal dihybrid segregation ratio 9:3:3:1 is modified to 9:6:1 ratio. Similar gene action is also found in barley for awn length.

vi) Duplicate recessive interaction (13:3) → The dominant allele (A), either in homozygous or heterozygous condition, of one gene and the homozygous recessive allele (bb), of other gene produces the same phenotype.

In F2 generation, progenies having A (homozygous or heterozygous) or bb (homozygous) will not allow the c gene to be expressed.

Genotype AABB, AABb, AaBb and Aabb produce same phenotype and the genotype aaBB, aaBb, and aabb produce another but same phenotype.

Scanned with CamScanner

Example: complete dominance at both gene pairs, but one gene when dominant epistatic to the other, and the second gene when homozygous recessive, epistatic to the first.

Feather colour of Fowl →

Gene pair 'A': colour inhibition is dominant to colour appearance.

Gene pair 'B': colour is dominant to white.

Interaction: Dominant color inhibition prevents color when color is present, color gene, when homozygous recessive prevents color even when dominant inhibitor is absent.

F1 progeny: $\boxed{AABB}$ (white leghorn) × $\boxed{aabb}$ (white plymouth Rock).

$$\begin{tikzcd}
\boxed{AABB} \arrow[r, phantom, "\times"] & \boxed{aabb} \arrow[d] \\
& \boxed{AaBb} \text{ (white)} \arrow[dll, "\text{gametes}"'] \arrow[dl] \arrow[d] \arrow[dr] \\
AB & Ab & aB & ab
\end{tikzcd}$$

F2 progeny:

Ratio:
13 : 3
white : colored

	AB	Ab	aB	ab
AB	AABB white	AABb white	AaBB white	AaBb. white
Ab	AABb white	AAbb colored	AaBb white	Aabb colored
aB	AaBB white	AaBb white	aaBB white	aaBb white
ab	AaBb white	Aabb colored	aaBb white	aabb white.

Digital Control System

Course Code: PE-EE-601A

Module – 6

Lecture 1

Stability of discrete-time system

Stability concepts and definition used in connection with continuous-time systems are directly applicable for a discrete-time system. Like continuous-time systems, the discrete-time systems also has to satisfy two notions of stability, namely

a. Zero input stability
b. BIBO stability

An initially relaxed system is said to be BIBO stable if for every bounded input $r(k);\ k \geq 0$ the output $y(k);\ k \geq 0$ is bounded for all value of *k*.

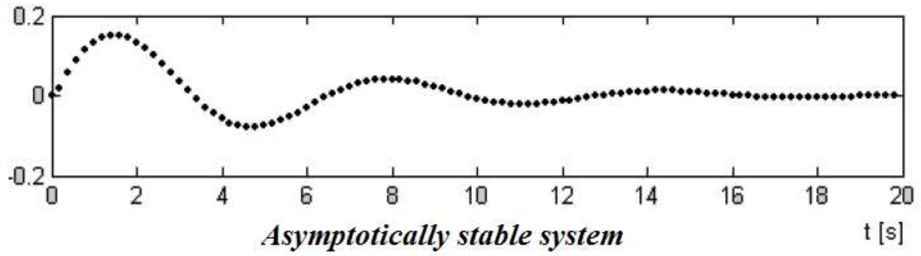
For a linear time invariant system to be BIBO stable, it is necessary and sufficient that

$$\sum_{k=0}^{\alpha} |g(k)| < \alpha$$

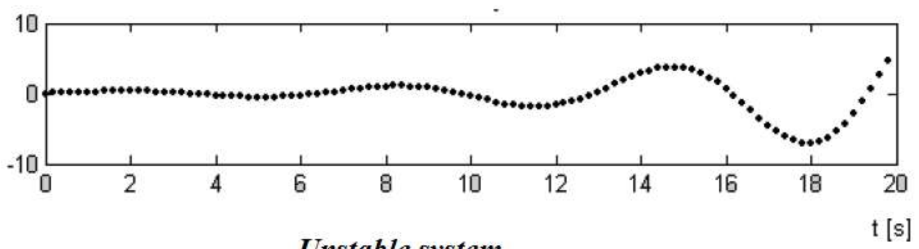
Where $g(k)$ is the response of the system to a unit sample sequence or impulse.

Stability of discrete-time system

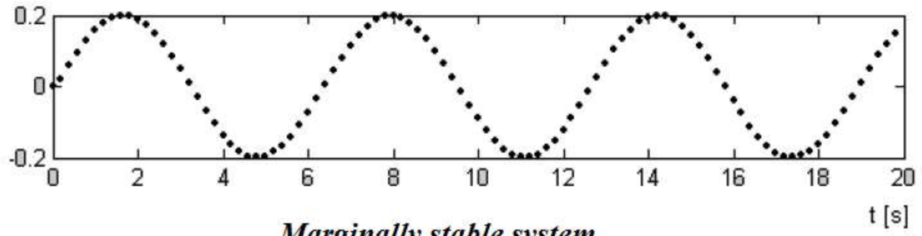
The impulse response for the different stability properties are illustrated in the figure



Asymptotically stable system



Unstable system



Marginally stable system

Stability of discrete-time system

Mapping between the *s* plane and the *z* plane

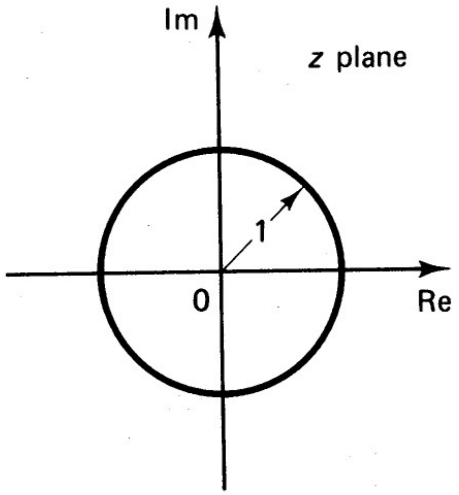
Relation between z and s

$$z = e^{Ts}$$

- The pole and zero locations in the z plane are related to the pole and zero locations in the s plane.
- The dynamic behavior of the discrete-time control system depends on the sampling period T.
  - Locations of poles and zeros in the z plane depend on T

Stability of discrete-time system

Mapping between the *s* plane and the *z* plane



The complex variables z and s are related by the equation

$z = e^{Ts}$

$s = \sigma + jw$

$z = e^{Ts} = e^{T(\sigma + jw)} = e^{T\sigma} e^{jTw} = e^{T\sigma} e^{j(Tw + 2\pi k)}$

Note: frequencies differ in integral multiples of the sampling frequency $\frac{2\pi}{T}$, are mapped into the same location in the z plane. jw axis in the s plane corresponds to $|z| = 1$. The interior of the unit circle corresponds to the left half of the s plane. The exterior of the unit corresponds to the right half of the plane.

- The left half of the *s* plane

$$|z| = e^{T\sigma} < 1 \qquad \text{(interior of the unit circle)}$$

- The $j\omega$ axis in the *s* plane corresponds to

$$|z| = 1 \quad \text{(unit circle)}$$

Stability of discrete-time system

Mapping between the *s* plane and the *z* plane

Primary strip and Complementary strips

$$z = e^{Ts}$$
$$\angle z = \omega T$$

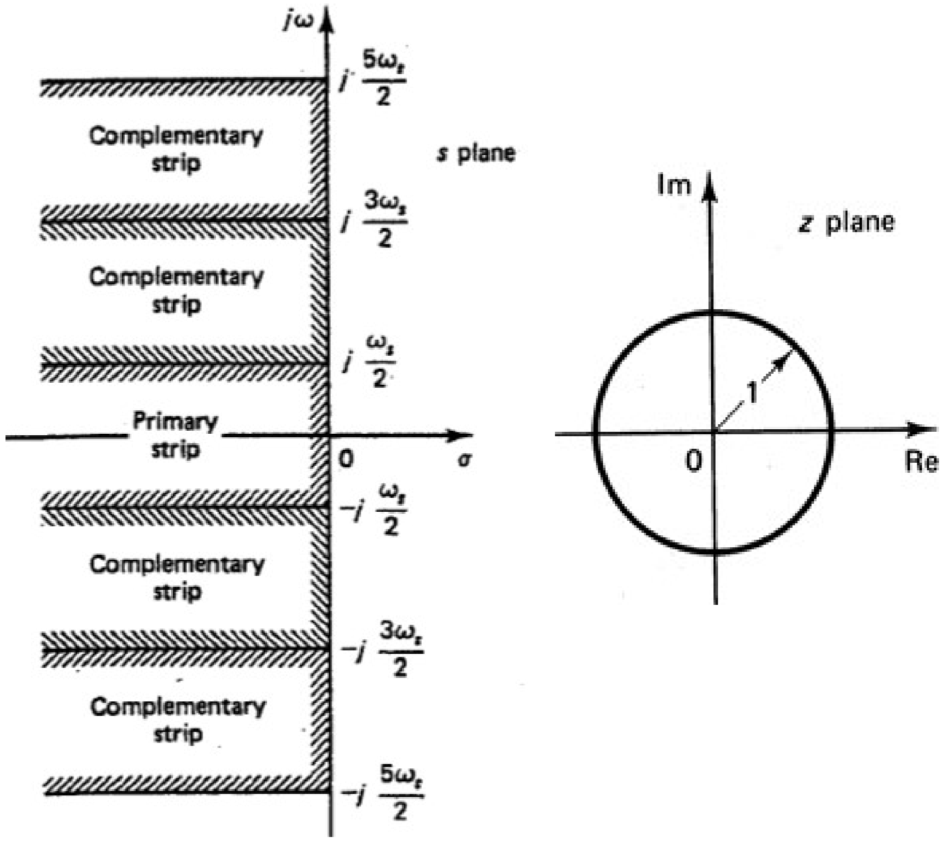
1) $\angle z = wT$ the angle of z varies from $-\infty \; to \; \infty$ as w varies from $-\infty \; to \; \infty$. as point moves from $-j\frac{1}{2}w_s$ to $j\frac{1}{2}w_s \; (w_s = \frac{2\pi}{T})$

$\angle z = wT$ varies from $-\pi \; to \; \pi$

This is the primary strip.

2) Each other strip with range of w_s will trace the z plane in one circle.

As the point in the s plane moves from $-\infty$ to ∞ on the $j\omega$ axis, we trace the unit circle in the z plane an infinite number of times.



SUBJECT: ENERGY MANAGEMENT & AUDIT
CODE : EE 801C

Module 2 : Energy Scenario

Lecture 1

Energy Resources

Energy is one of the major inputs for the economic development of any country. In the case of the developing countries, the energy sector assumes a critical importance in view of the ever-increasing energy needs requiring huge investments to meet them.
Energy can be classified into several types based on the following criteria:

- Primary and Secondary energy
- Commercial and Non-commercial energy
- Renewable and Non-Renewable energy

Primary and Secondary energy

Primary energy sources are those that are either found or stored in nature. Common primary energy sources are coal, oil, natural gas, and biomass (such as wood). Other primary energy sources available include nuclear energy from radioactive substances, thermal energy stored in earth's interior, and potential energy due to earth's gravity.
Primary energy sources are mostly converted in industrial utilities into secondary energy sources; for example coal, oil or gas converted into steam and electricity. Primary energy can also be used directly. Some energy sources have non-energy uses, for example coal or natural gas can be used as a feedstock in fertiliser plants.

Commercial and Non commercial energy

Commercial Energy

The energy sources that are available in the market for a definite price are known as commercial energy. By far the most important forms of commercial energy are electricity, coal and refined petroleum products. Commercial energy forms the basis of industrial, agricultural, transport and commercial development in the modern world. In the industrialized countries, commercialized fuels are predominant source not only for economic production, but also for many household tasks of general population. Examples: Electricity, lignite, coal, oil, natural gas etc.

Non-Commercial Energy

The energy sources that are not available in the commercial market for a price are classified as non-commercial energy. Non-commercial energy sources include fuels such as firewood, cattle dung and agricultural wastes, which are traditionally gathered, and not bought at a price used especially in rural households. These are also called traditional fuels. Non-commercial energy is often ignored in energy accounting. Example: Firewood, agro waste in rural areas; solar energy for water heating, electricity generation, for drying grain, fish and fruits; animal power for transport, threshing, lifting water for irrigation, crushing sugarcane; wind energy for lifting water and electricity generation.

Renewable and Non-Renewable Energy

Renewable energy is energy obtained from sources that are essentially inexhaustible. Examples of renewable resources include wind power, solar power, geothermal energy, tidal power and hydroelectric power. The most important feature of renewable energy is that it can be harnessed without the release of harmful pollutants.
Non-renewable energy is the conventional fossil fuels such as coal, oil and gas, which are likely to deplete with time.

Primary Energy Resources

Coal

The proven global coal reserve was estimated to be 9,84,453 million tonnes by end of 2003. The USA had the largest share of the global reserve (25.4%) followed by Russia (15.9%), China (11.6%). India was 4th in the list with 8.6%.

Oil

The global proven oil reserve was estimated to be 1147 billion barrels by the end of 2003. Saudi Arabia had the largest share of the reserve with almost 23%. (One barrel of oil is approximately 160 litres)

Gas

The global proven gas reserve was estimated to be 176 trillion cubic metres by the end of 2003. The Russian Federation had the largest share of the reserve with almost 27%.

Indian Energy Scenario

Coal dominates the energy mix in India, contributing to 58% of the total primary energy production. Over the years, there has been a marked increase in the share of natural gas in primary energy production from 10% in 1994 to 13% in 1999. There has been a decline in the share of oil in primary energy production from 20% to 17% during the same period.

Coal Supply

India has huge coal reserves, at least 84,396 million tonnes of proven recoverable reserves (at the end of 2003). This amounts to almost 8.6% of the world reserves and it may last for about 230 years at the current Reserve to Production (R/P) ratio. In contrast, the world's proven coal reserves are expected to last only for 192 years at the current R/P ratio.
Reserves/Production (R/P) ratio- If the reserves remaining at the end of the year are divided by the production in that year, the result is the length of time that the remaining reserves would last if production were to continue at that level.
India is the fourth largest producer of coal and lignite in the world. Coal production is concentrated in these states (Andhra Pradesh, Uttar Pradesh, Bihar, Madhya Pradesh, Maharashtra, Orissa, Jharkhand, West Bengal).

Oil Supply

Oil accounts for about 36 % of India's total energy consumption. India today is one of the top ten oil-guzzling nations in the world. According to the US Energy Information Administration (EIA), India is currently ranked behind the United States and China as the world's third-largest oil consumer. It consumed 206.2 million tonnes (over 4 million bpd) in the 2017-18 fiscal year. India's oil demand is projected to rise by 5.8 million barrels per day (bpd) by 2040.

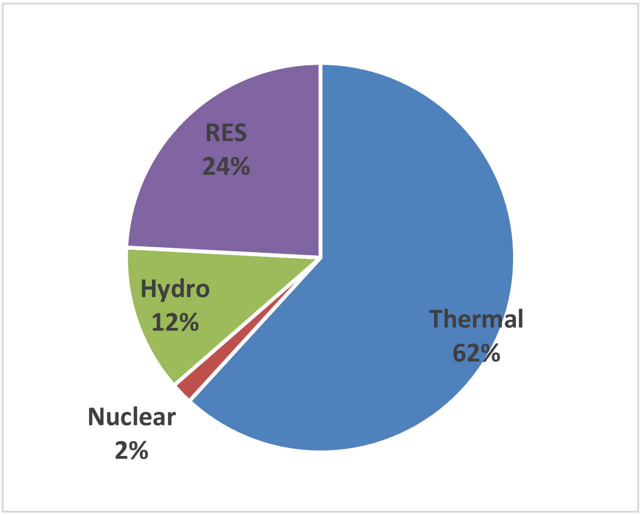
Natural Gas Supply

Natural gas accounts for about 8.9 per cent of energy consumption in the country. The current demand for natural gas is about 96 million cubic metres per day (mcmd) as against availability of 67 mcmd. By 2007, the demand is expected to be around 200 mcmd. Natural gas reserves are estimated at 660 billion cubic meters.

Electrical Energy Supply

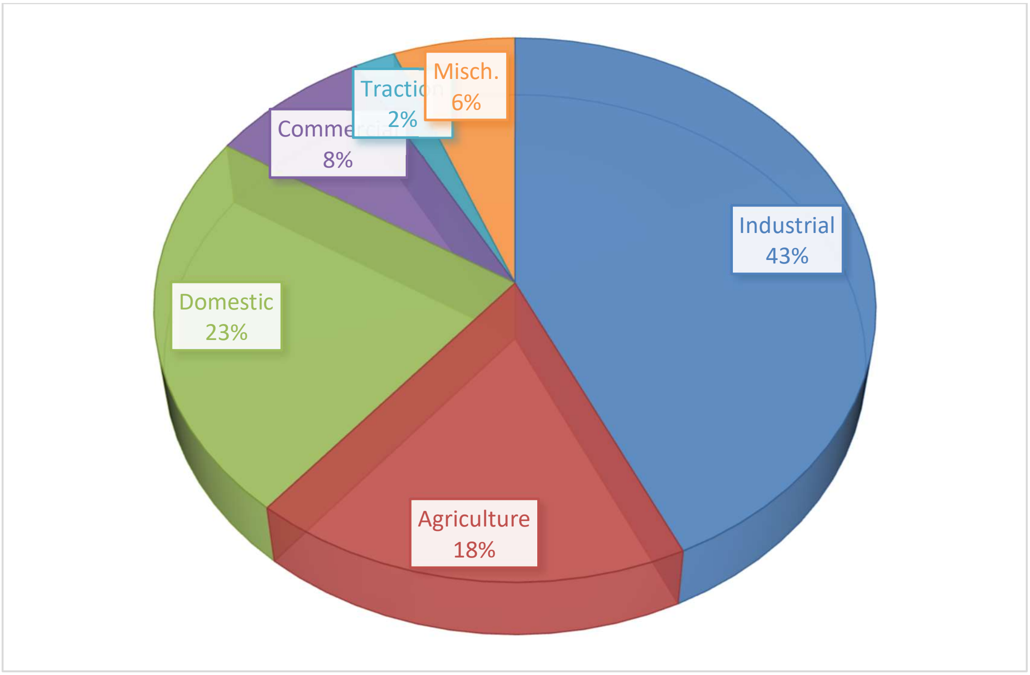
The all India installed capacity of electric power generating stations under utilities is 3,70,107 MW as on **28.02.2021.** following table shows the details of the same

Thermal	2,33,171
Nuclear	6,780
Hydro	46,209
RES	91,154
Total	**3,79,130**



SectorWise Energy Consumption in India

Average electricity usage in India stands 1,181 kWh per capita in 2018-2019. The major commercial energy consuming sectors in the country are classified as shown in the Figure. As seen from the figure, industry remains the biggest consumer of commercial energy and its share in the overall consumption is 43%.



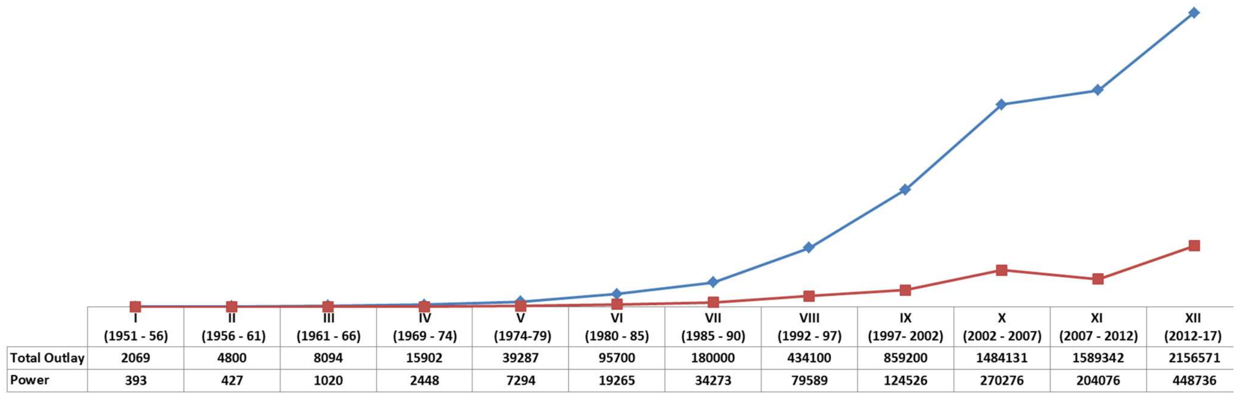
Energy Needs of Growing Economy

Economic growth is desirable for developing countries, and energy is essential for economic growth. However, the relationship between economic growth and increased energy demand is not always a straightforward linear one. For example, under present conditions, 6% increase in India's Gross Domestic Product (GDP) would impose an increased demand of 9 % on its energy sector.

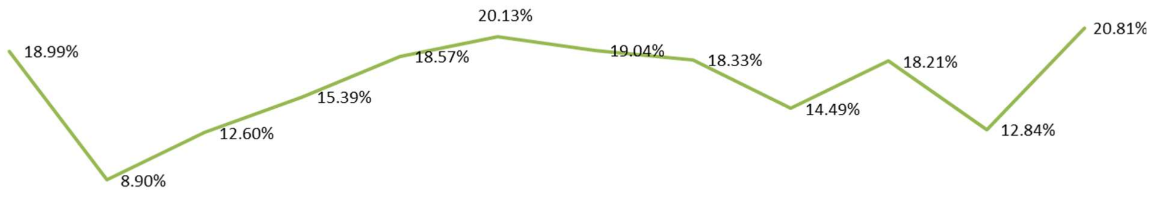
In this context, the ratio of energy demand to GDP also termed as **energy intensity** is a useful indicator. A high ratio reflects energy dependence and a strong influence of energy on GDP growth. The developed countries, by focusing on energy efficiency and lower energy-intensive routes, maintain their energy to GDP ratios at values of less than 1. The ratios for developing countries are much higher.

India's Energy Needs

The plan outlay vis-à-vis share of energy is given in Figure below. As seen from the Figure, in an average of 18.0% of the total five-year plan outlay is spent on the energy sector.



	I (1951 - 56)	II (1956 - 61)	III (1961 - 66)	IV (1969 - 74)	V (1974-79)	VI (1980 - 85)	VII (1985 - 90)	VIII (1992 - 97)	IX (1997- 2002)	X (2002 - 2007)	XI (2007 - 2012)	XII (2012-17)
Total Outlay	2069	4800	8094	15902	39287	95700	180000	434100	859200	1484131	1589342	2156571
Power	393	427	1020	2448	7294	19265	34273	79589	124526	270276	204076	448736



	I (1951 - 56)	II (1956 - 61)	III (1961 - 66)	IV (1969 - 74)	V (1974-79)	VI (1980 - 85)	VII (1985 - 90)	VIII (1992 - 97)	IX (1997- 2002)	X (2002 - 2007)	XI (2007 - 2012)	XII (2012-17)
%	18.99%	8.90%	12.60%	15.39%	18.57%	20.13%	19.04%	18.33%	14.49%	18.21%	12.84%	20.81%

Per Capita Energy Consumption

The per capita energy consumption is too low for India as compared to developed countries. It is just 4% of USA and 20% of the world average. The per capita consumption is likely to grow in India with growth in economy thus increasing the energy demand.

Energy Intensity

Energy intensity is energy consumption per unit of GDP. Energy intensity indicates the development stage of the country. India's energy intensity is 3.7 times of Japan, 1.55 times of USA, 1.47 times of Asia and 1.5 times of World average.

Long Term Energy Scenario for India

Coal

Coal is the predominant energy source for power production in India, generating approximately 70% of total domestic electricity. Energy demand in India is expected to increase over the next 10-15 years; although new oil and gas plants are planned, coal is expected to remain the dominant fuel for power generation. Despite significant increases in total installed capacity during the last decade, the gap between electricity supply and demand continues to increase. The resulting shortfall has had a negative impact on industrial output and economic growth.
However, to meet expected future demand, indigenous coal production will have to be greatly expanded. Production currently stands at around 290 Million tonnes per year, but coal demand is expected to more than double by 2010. Indian coal is typically of poor quality and as such requires to be beneficiated to improve the quality; Coal imports will also need to increase dramatically to satisfy industrial and power generation requirements.

Oil

India's demand for petroleum products is likely to rise from 97.7 million tonnes in 2001-02 to around 139.95 million tonnes in 2006-07, according to projections of the Tenth Five-Year Plan. The plan document puts compound annual growth rate (CAGR) at 3.6 % during the plan period. Domestic crude oil production is likely to rise marginally from 32.03 million tonnes in 2001-02 to 33.97 million tonnes by the end of the 10th plan period (2006-07). India's self sufficiency in oil has consistently declined from 60% in the 50s to 30% currently. Same is expected to go down to 8% by 2020. Around 92% of India's total oil demand by 2020 has to be met by imports.

Natural Gas

India's natural gas production is likely to rise from 86.56 million cmpd in 2002-03 to 103.08 million cmpd in 2006-07. It is mainly based on the strength of a more than doubling of production by private operators to 38.25 mm cmpd.

Electricity

India currently has a peak demand shortage of around 14% and an energy deficit of 8.4%. Keeping this in view and to maintain a GDP (gross domestic product) growth of 8% to 10%, the Government of India has very prudently set a target of 215,804 MW power generation capacity by March 2012 from the level of 100,010 MW as on March 2001, that is a capacity addition of 115,794 MW in the next 11 years. In the area of nuclear power the objective is to achieve 20,000 MW of nuclear generation capacity by the year 2020.

Energy from Biomass

Lecture 1

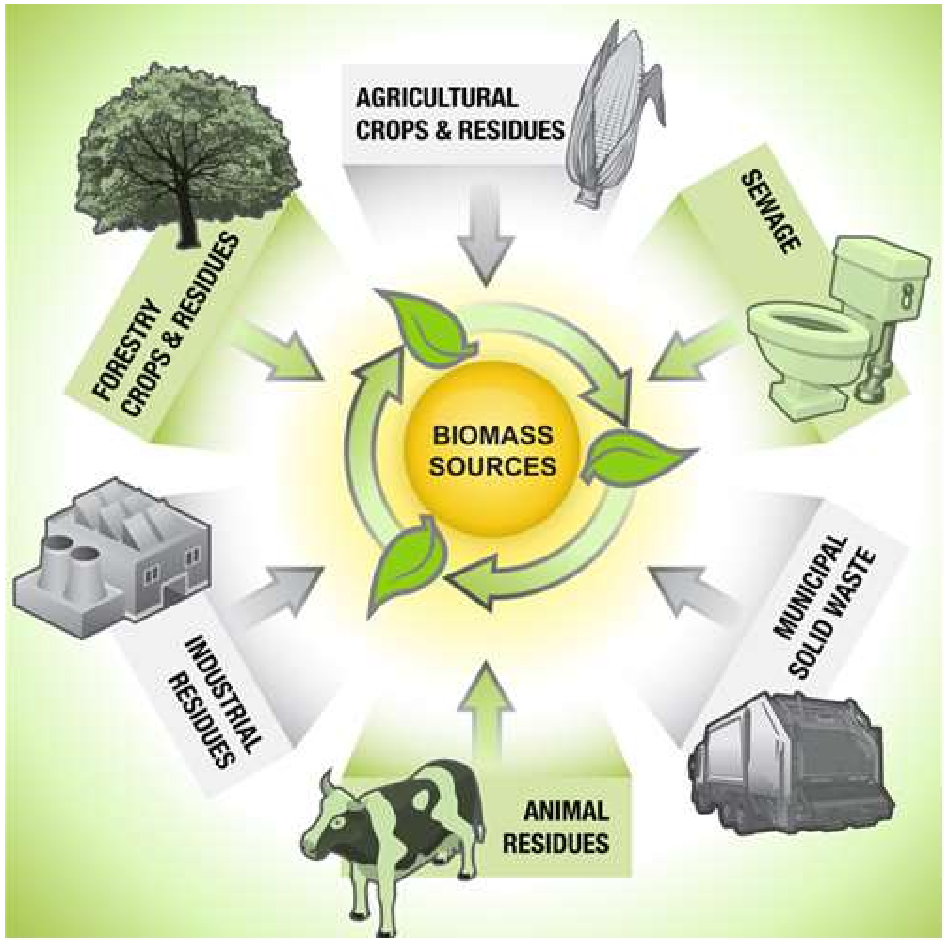
Topics

Introduction
Biomass Conversion Technologies

Introduction

Biomass is a renewable energy resource derived from the ***carbonaceous waste*** of various human and natural activities. It is derived from numerous sources, including the by-products from the timber industry, agricultural crops, raw material from the forest, major parts of household waste and wood.

Biomass is any sort of vegetation-trees, grasses, plants parts such as leaves, stems and twigs, and ocean plants. From it, we can extract a wealth of stored energy.

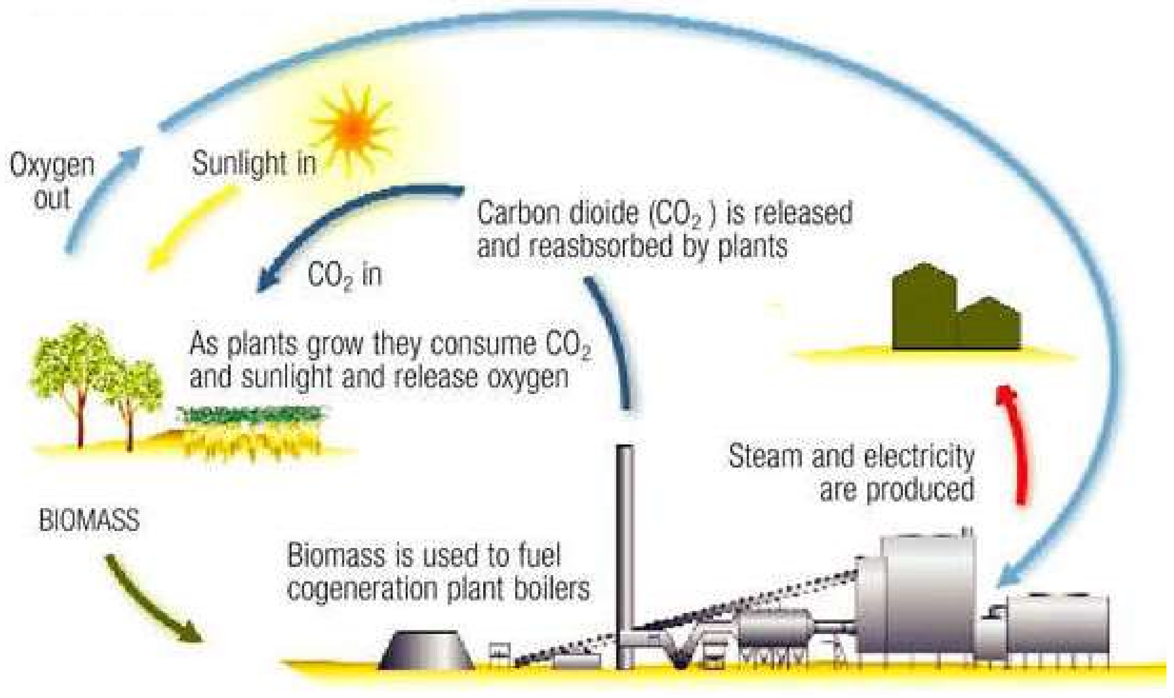


Introduction

During photosynthesis, plants combine carbon dioxide from the air and water from the ground to form carbohydrates, which form the building blocks of biomass.

The solar energy that drives photosynthesis is stored in the chemical bonds of the structural components of biomass. If we burn biomass efficiently and extract the energy stored in the chemical bonds, oxygen from the atmosphere combines with carbon in the plants to produce carbon dioxide and water.

The process is cyclic and renewable because the carbon dioxide is then available to produce new biomass.



Oxygen out
Sunlight in
Carbon dioide (CO_2) is released and reasbsorbed by plants
CO_2 in
As plants grow they consume CO_2 and sunlight and release oxygen
Steam and electricity are produced
BIOMASS
Biomass is used to fuel cogeneration plant boilers

Because the released carbon dioxide is used by growing plants in their photosynthesis process, there is no new carbon dioxide produced or released to the atmosphere. Therefore, there's no net contribution to global carbon dioxide emissions.

Introduction

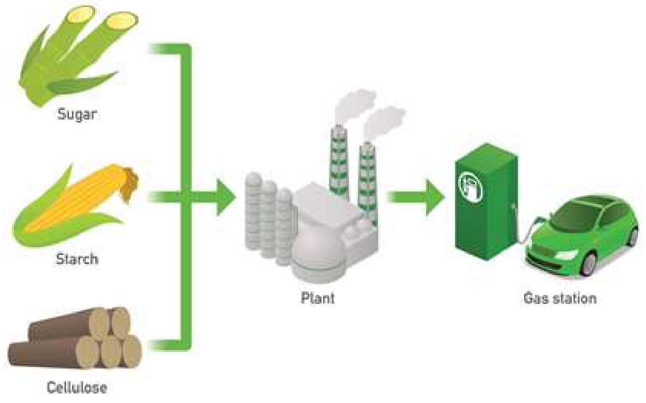
Followings are the major biomass applications:

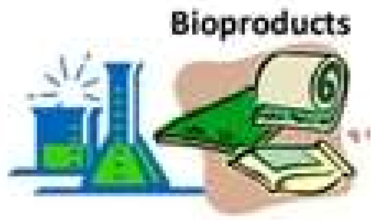
Biopower — Burning biomass directly, or converting it into gaseous or liquid fuels that burn more efficiently, to generate electricity

Biofuels — Converting biomass into liquid fuels for transportation

Bioproduct — Converting biomass into chemicals for making plastics and other products that typically are made from petroleum

Ironbridge, United Kingdom – 740MW
With 740MW capacity, the Ironbridge power plant located in the Severn Gorge, UK, is the world's biggest biomass power plant. Ironbridge was previously a coal-fired power station with an installed capacity of 1,000MW. Two units of the plant were converted for biomass-based power generation in 2013.



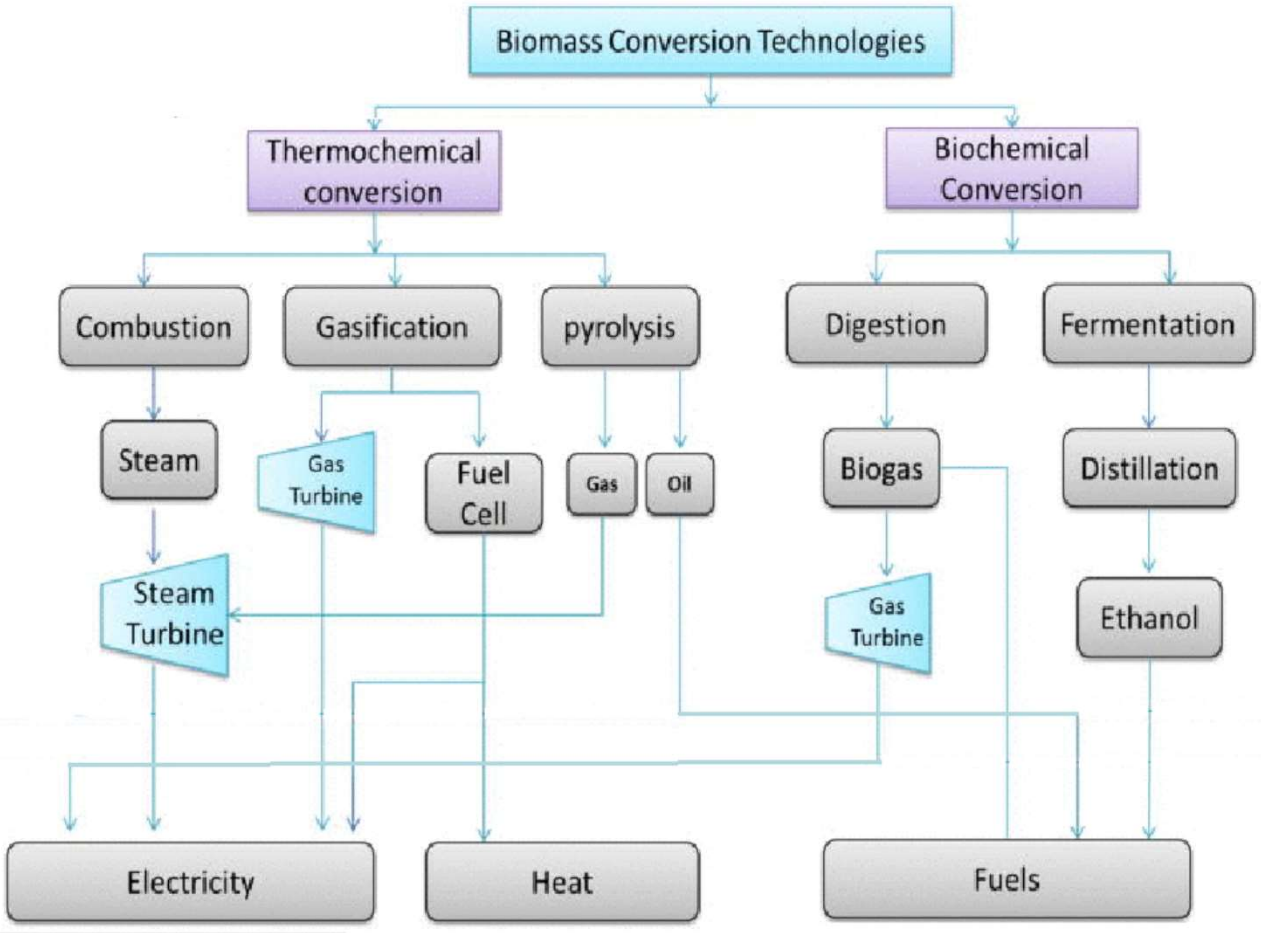


Biomass Conversion Technologies

A wide range of technologies have been developed to utilise the biomass resource. These vary from direct combustion in burner systems, to the production of more advanced biofuels, such as pyrolysis, through a variety of processing techniques.

Biomass is converted to energy through various processes, including. Direct combustion (burning) to produce heat. In general the conversion technologies used are :

- ***Thermochemical conversion*** to produce solid, gaseous, and liquid fuels.
- ***Biochemical conversion*** to produce liquid and gaseous fuels

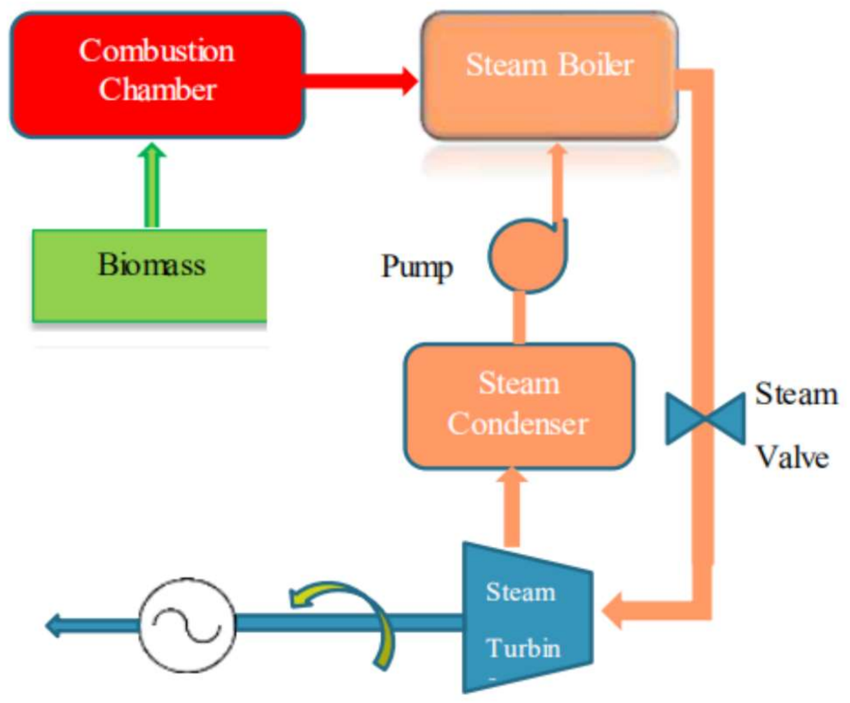


Biomass Conversion Technologies

Thermochemical conversion

Combustion

Combustion is an exothermic chemical reaction, in which biomass is burned in the presence of air. In this process chemical energy which is stored in the biomass is converted in the mechanical and electrical energies. This process is suitable for dry biomass containing moisture less than 50%. Biomass is burned at the temperature of 800-1000 °C. This process is used for domestic applications as well as commercially in biomass power plants in order to produce electricity.



Biomass Conversion Technologies

Thermochemical conversion

Gasification

Gasification is a process that exposes a solid fuel to high temperatures and limited oxygen, to produce a gaseous fuel. This is a mix of gases such as carbon monoxide, carbon dioxide, nitrogen, hydrogen and methane and is called producer gas.

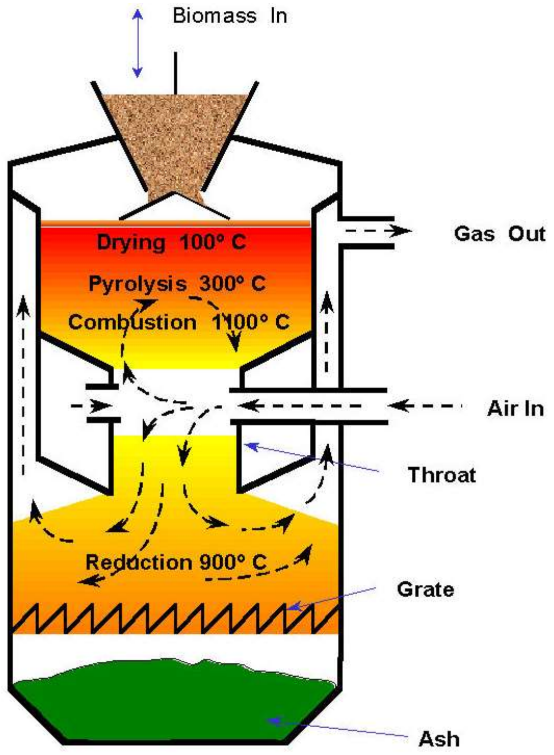
Gasification is conducted in a closed chamber called gasifier. The figure shows different sections inside a gasifier.

Drying

Biomass fuels consist of moisture ranging from 5 to 35%. At the temperature of around 150°C, the water is removed and converted into steam.

Pyrolysis

Pyrolysis is the themal decomposition of biomass fuels in the absence of oxygen. Pyrolysis involves release of three kinds of products : solid, liquid and gases. The temperature in this zone is within 150°C to 700°C.



Biomass Conversion Technologies

Thermochemical conversion

Gasification

Combustion

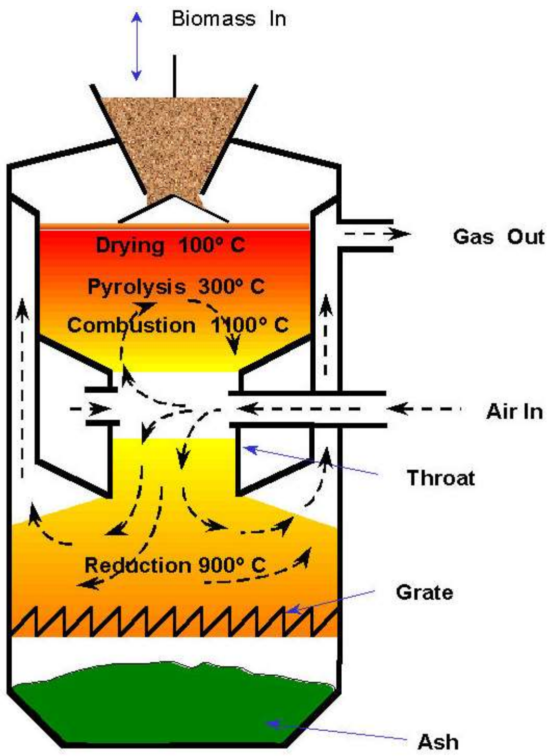
Introduced air in the combustion zone. Air contains, besides oxygen and water vapours, inert gases such as nitrogen and argon. These inert gases are considered to be non-reactive with fuel constituents. The oxidation takes place at the temperature of 700-2000^o c.

Reduction

In reduction zone, a number of high temperature chemical reactions take place in the absence of oxygen to produce CO & Methane

There are **various types of gasifiers**, but main of them are as following.

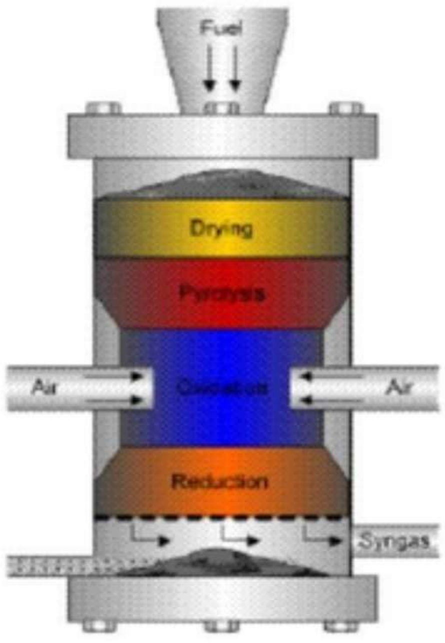
a. **Downdraft** gasifier
b. **Updraft** gasifier
c. **Cross-draft** gasifier



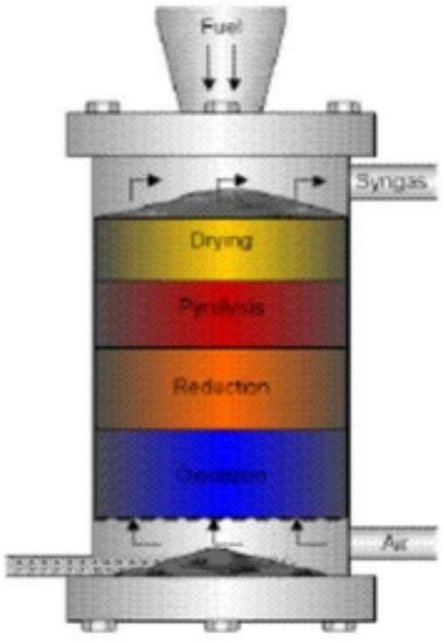
Biomass Conversion Technologies

Thermochemical conversion

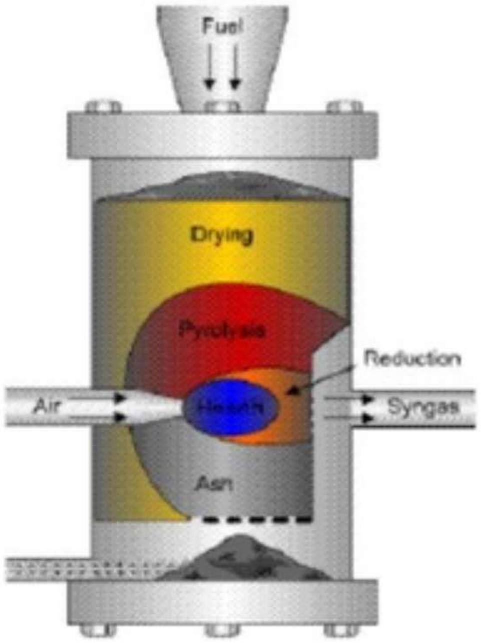
Gasification



Downdraft



Updraft



Cross-draft

Biomass Conversion Technologies

Thermochemical conversion

Gasification

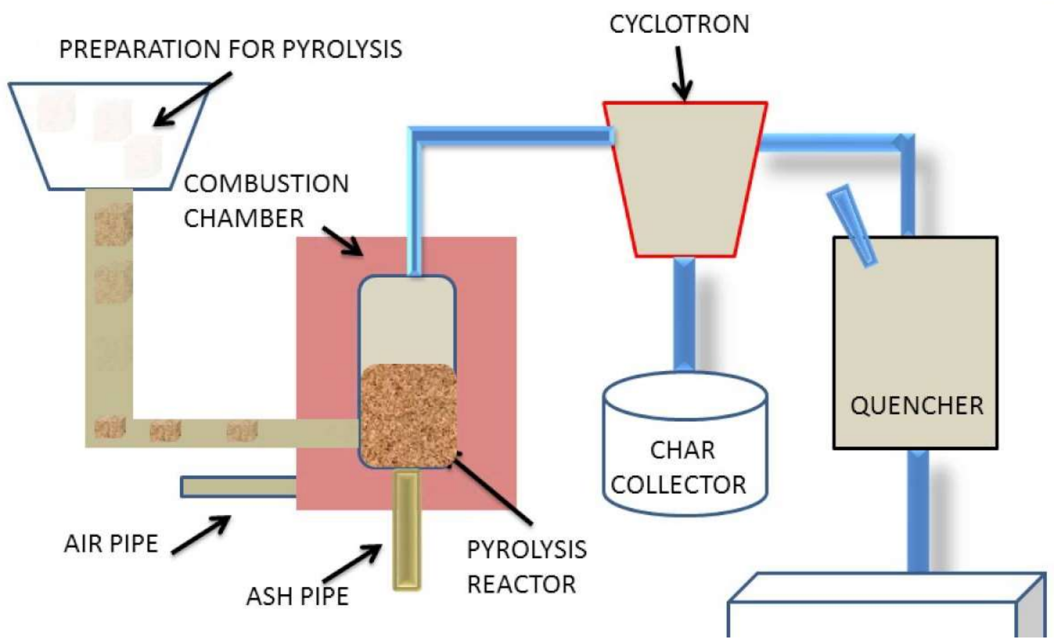
	Downdraft	Updraft	Cross-draft
Opeartion	Biomass is introduced from the top and moves downward. air is introduced at the top and flows downward. Producer gas is extracted at the bottom.	Biomass is introduced from the top and moves downward. air is introduced at the bottom and flows upward. Some drying occurs. Producer gas is extracted at the top.	Biomass is introduced from the top and moves downward. air is introduced at the bottom and flows across the bed. Some drying occurs. Producer gas is extracted opposite the air nozzle at the grate.
Advantages	Tar & particulates in the producer gas is lower.	• Can handle higher moisture biomass • Higher temperature can destroy some toxins, minerals & metals. • Higher tar content.	• Simplest of design. • Stronger circulation in hot zone • Lower temperature allow the use of less expensive construction material.
Disadvantages	• Feed size limits • Scale limitation • Low heating value • Moisture-sensitive	• Feed size limits • High tar yields • Scale limitations • Low heating value gas	• Complicated to operate • Forms slag • High carbon (33%) content in ash.

Biomass Conversion Technologies

Thermochemical conversion

Pyrolysis

Pyrolysis is the basic thermo-chemical process for converting solid biomass to a more useful liquid fuel, commonly called a bio-oil. This bio-oil can be used in existing oil-fired burners (with very little adjustment) to generate heat & electricity. The process involves heating solid biomass to a temperature of around 800^{o}C, in the absence of oxygen. This forces the volatile substances out of the biomass, leaving a small quantity of solid biomass (char). The volatiles are then collected in liquid form as the bio-oil.



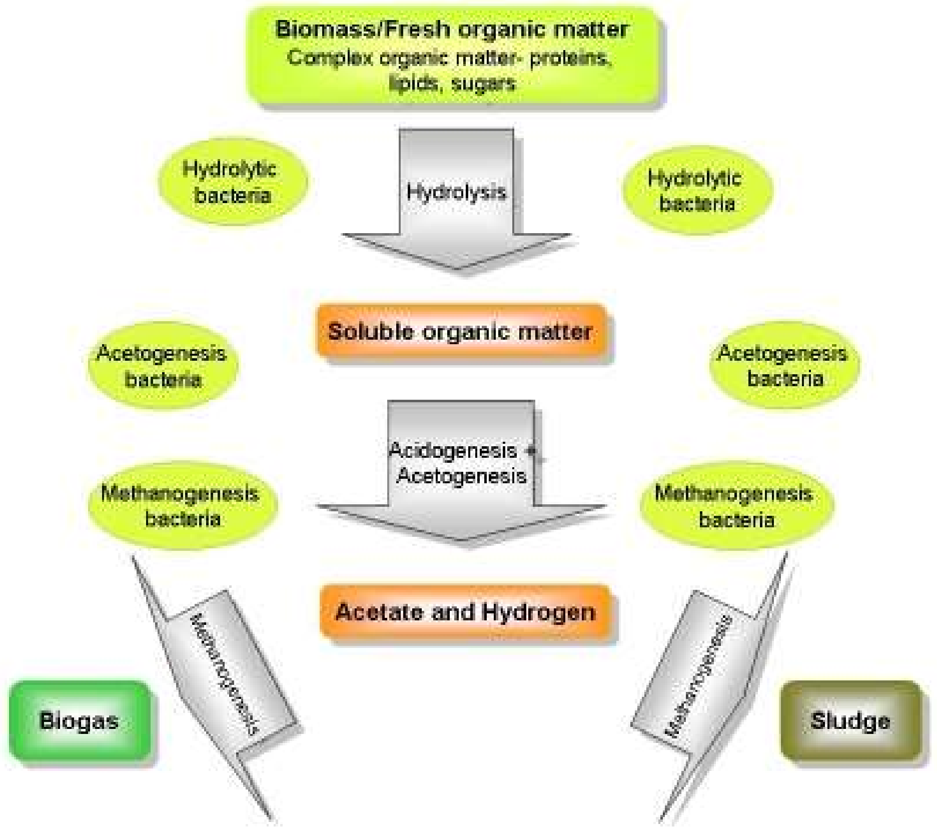
Biomass Conversion Technologies

Biochemical conversion

Digestion

Biomass digestion works by the action of anaerobic bacteria. These microorganisms usually live at the bottom of swamps or in other places where there is no air, consuming dead organic matter to produce, among other things, methane and hydrogen.

We can put these bacteria to work for us. By feeding organic matter such as animal dung or human sewage into tanks – called digesters - and adding bacteria, we can collect the emitted gas to use as an energy source. This can be a very efficient means of extracting usable energy from such biomass – up to two-thirds of the fuel energy of the animal dung is recovered.

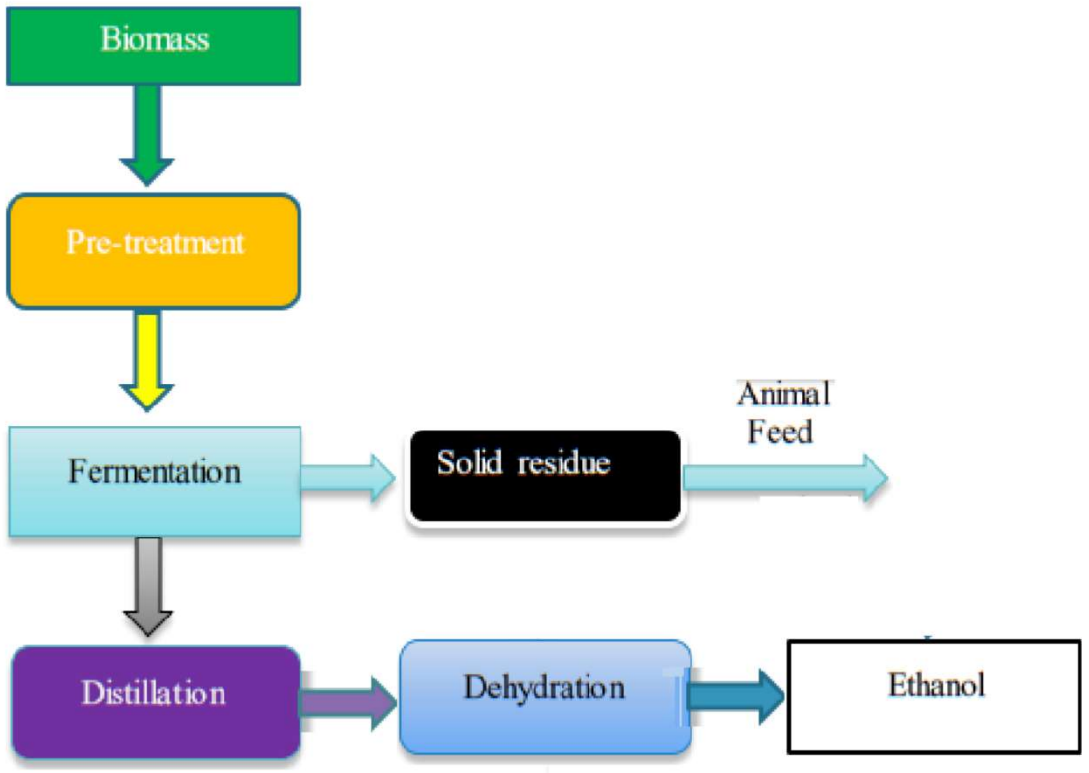


Biomass Conversion Technologies

Biochemical conversion

Fermentation

This process can be used on certain sugar producing energy crops to produce ethanol, a simple alcohol. It uses a simple and well established method; yeast is added to the biomass and the mixture is then allowed to ferment under specific conditions. The resulting brew is then distilled to produce 'bio-ethanol'. This can be used on it’s own in specialised combustion engines or it can be mixed with petrol to produce 'gasohol'.



Thank you for your attention

IPv6

Internet Protocol version 6 is a new addressing protocol designed to incorporate all the possible requirements of future Internet known to us as Internet version 2. This protocol as its predecessor IPv4, works on the Network Layer (Layer-3). Along with its offering of an enormous amount of logical address space, this protocol has ample features to which address the shortcoming of IPv4.

Why New IP Version?

So far, IPv4 has proven itself as a robust routable addressing protocol and has served us for decades on its best-effort-delivery mechanism.

IPv4 is 32 bits long and offers around 4,294,967,296 (2^{32}) addresses. This address space was considered more than enough that time. Given below are the major points that played a key role in the birth of IPv6:

- Internet has grown exponentially and the address space allowed by IPv4 is saturating. There is a requirement to have a protocol that can satisfy the needs of future Internet addresses that is expected to grow in an unexpected manner.
- IPv4 on its own does not provide any security feature. Data has to be encrypted with some other security application before being sent on the Internet.
- Data prioritization in IPv4 is not up to date. Though IPv4 has a few bits reserved for Type of Service or Quality of Service, but they do not provide much functionality.
- IPv4 enabled clients can be configured manually or they need some address configuration mechanism. It does not have a mechanism to configure a device to have globally unique IP address.

:IPv6 – Features:

The successor of IPv4 is not designed to be backward compatible. Trying to keep the basic functionalities of IP addressing, IPv6 is redesigned entirely. It offers the following features:

- **Larger Address Space**

  In contrast to IPv4, IPv6 uses 4 times more bits to address a device on the Internet. This much of extra bits can provide approximately 3.4×10^{38} different combinations of addresses. This address can accumulate the aggressive requirement of address allotment for almost everything in this world. According to an estimate, 1564 addresses can be allocated to every square meter of this earth.

- **Simplified Header**

  IPv6's header has been simplified by moving all unnecessary information and options (which are present in IPv4 header) to the end of the IPv6 header. IPv6 header is only twice as bigger than IPv4 provided the fact that IPv6 address is four times longer.

- **End-to-end Connectivity**

  Every system now has unique IP address and can traverse through the Internet without using NAT or other translating components. After IPv6 is fully implemented, every host can directly reach other hosts on the Internet, with some limitations involved like Firewall, organization policies, etc.

- **Auto-configuration**

  IPv6 supports both stateful and stateless auto configuration mode of its host devices. This way, absence of a DHCP server does not put a halt on inter segment communication.

- **Faster Forwarding/Routing**

  Simplified header puts all unnecessary information at the end of the header. The information contained in the first part of the header is adequate for a Router to take routing decisions, thus making routing decision as quickly as looking at the mandatory header.

- **IPSec**

  Initially it was decided that IPv6 must have IPSec security, making it more secure than IPv4. This feature has now been made optional.

- **No Broadcast**

  Though Ethernet/Token Ring are considered as broadcast network because they support Broadcasting, IPv6 does not have any broadcast support any more. It uses multicast to communicate with multiple hosts.

- **Anycast Support**

  This is another characteristic of IPv6. IPv6 has introduced Anycast mode of packet routing. In this mode, multiple interfaces over the Internet are assigned same Anycast IP address. Routers, while routing, send the packet to the nearest destination.

- **Mobility**

  IPv6 was designed keeping mobility in mind. This feature enables hosts (such as mobile phone) to roam around in different geographical area and remain connected with the same IP address. The mobility feature of IPv6 takes advantage of auto IP configuration and Extension headers.

- **Enhanced Priority Support**

  IPv4 used 6 bits DSCP (Differential Service Code Point) and 2 bits ECN (Explicit Congestion Notification) to provide Quality of Service but it could only be used if the end-to-end devices support it, that is, the source and destination device and underlying network must support it.

  In IPv6, Traffic class and Flow label are used to tell the underlying routers how to efficiently process the packet and route it.

- **Smooth Transition**

  Large IP address scheme in IPv6 enables to allocate devices with globally unique IP addresses. This mechanism saves IP addresses and NAT is not required. So devices can send/receive data among each other, for example, VoIP and/or any streaming media can be used much efficiently.

  Other fact is, the header is less loaded, so routers can take forwarding decisions and forward them as quickly as they arrive.

- **Extensibility**

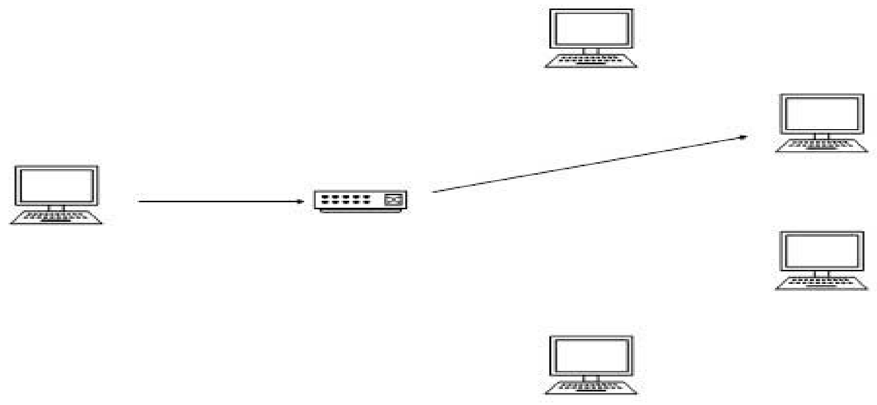
  One of the major advantages of IPv6 header is that it is extensible to add more information in the option part. IPv4 provides only 40-bytes for options, whereas options in IPv6 can be as much as the size of IPv6 packet itself.

Addressing Modes

In computer networking, addressing mode refers to the mechanism of hosting an address on the network. IPv6 offers several types of modes by which a single host can be addressed. More than one host can be addressed at once or the host at the closest distance can be addressed.

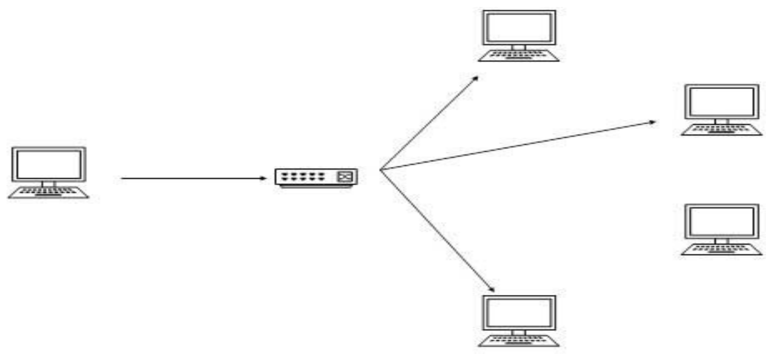
Unicast:

In unicast mode of addressing, an IPv6 interface (host) is uniquely identified in a network segment. The IPv6 packet contains both source and destination IP addresses. A host interface is equipped with an IP address which is unique in that network segment.When a network switch or a router receives a unicast IP packet, destined to a single host, it sends out one of its outgoing interface which connects to that particular host.



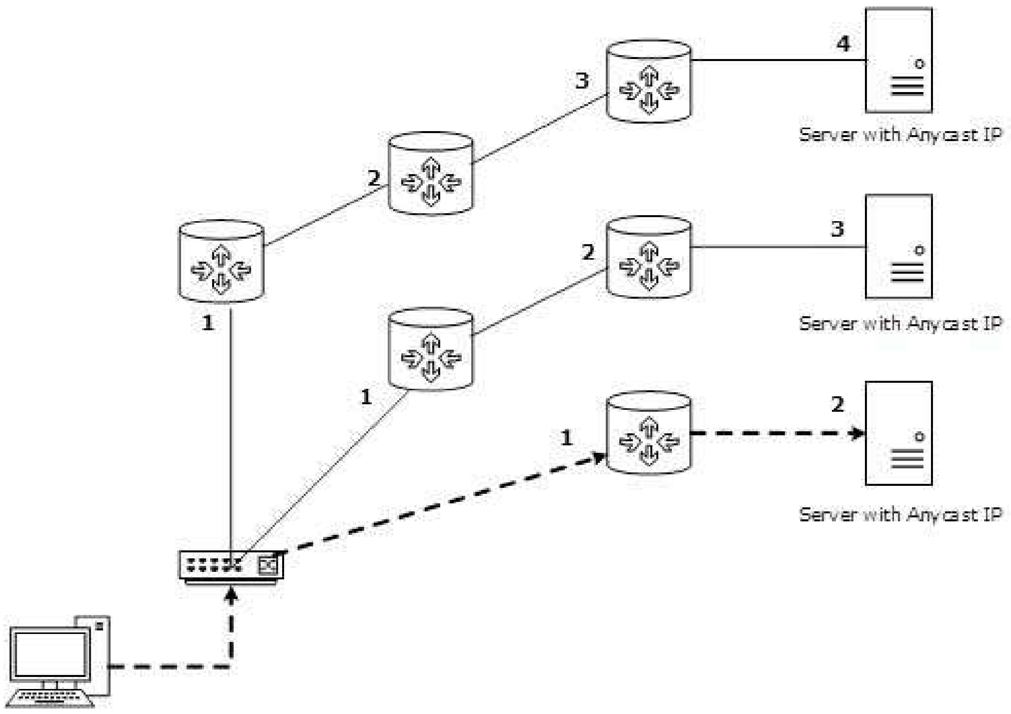
Multicast

The IPv6 multicast mode is same as that of IPv4. The packet destined to multiple hosts is sent on a special multicast address. All the hosts interested in that multicast information, need to join that multicast group first. All the interfaces that joined the group receive the multicast packet and process it, while other hosts not interested in multicast packets ignore the multicast information.



Anycast

IPv6 has in introduced a new type of addressing, which is called Anycast addressing. In this addressing mode, multiple interfaces (hosts) are assigned same Anycast IP address. When a host wishes to communicate with a host equipped with an Anycast IP address, it sends a Unicast message. With the help of complex routing mechanism, that Unicast message is delivered to the host closest to the Sender in terms of Routing cost.



Let's take an example of TutorialPoints.com Web Servers, located in all continents. Assume that all the Web Servers are assigned a single IPv6 Anycast IP Address. Now when a user from Europe wants to reach TutorialsPoint.com the DNS points to the server that is physically located in Europe itself. If a user from India tries to reach Tutorialspoint.com, the DNS will then point to the Web Server physically located in Asia. Nearest or Closest terms are used in terms of Routing Cost.

In the above picture, when a client computer tries to reach a server, the request is forwarded to the server with the lowest Routing Cost.

IPv6 - Address Types & Formats

Address Structure

An IPv6 address is made of 128 bits divided into eight 16-bits blocks. Each block is then converted into 4-digit Hexadecimal numbers separated by colon symbols.

For example, given below is a 128 bit IPv6 address represented in binary format and divided into eight 16-bits blocks:

```
0010000000000001 0000000000000000 0011001000111000 1101111111100001
0000000001100011 0000000000000000 0000000000000000 1111111011111011
```

Each block is then converted into Hexadecimal and separated by ':' symbol:

```
2001:0000:3238:DFE1:0063:0000:0000:FEFB
```

Even after converting into Hexadecimal format, IPv6 address remains long. IPv6 provides some rules to shorten the address. The rules are as follows:

Rule.1: Discard leading Zero(es):

In Block 5, 0063, the leading two 0s can be omitted, such as (5th block):

```
2001:0000:3238:DFE1:63:0000:0000:FEFB
```

Rule.2: If two of more blocks contain consecutive zeroes, omit them all and replace with double colon sign ::, such as (6th and 7th block):

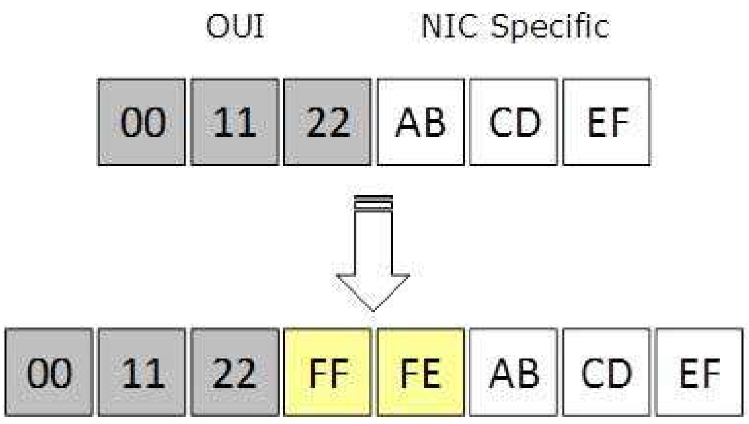
```
2001:0000:3238:DFE1:63::FEFB
```

Consecutive blocks of zeroes can be replaced only once by :: so if there are still blocks of zeroes in the address, they can be shrunk down to a single zero, such as (2nd block):

```
2001:0:3238:DFE1:63::FEFB
```

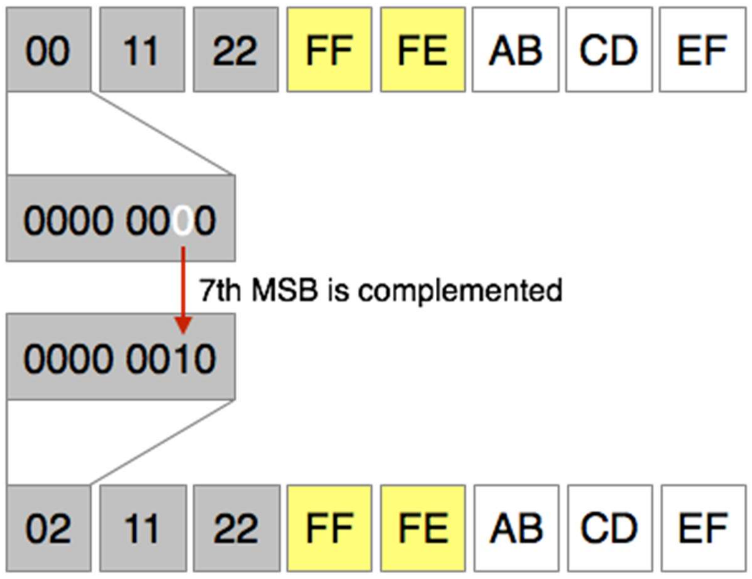
Interface ID

IPv6 has three different types of Unicast Address scheme. The second half of the address (last 64 bits) is always used for Interface ID. The MAC address of a system is composed of 48-bits and represented in Hexadecimal. MAC addresses are considered to be uniquely assigned worldwide. Interface ID takes advantage of this uniqueness of MAC addresses. A host can auto-configure its Interface ID by using IEEE's Extended Unique Identifier (EUI-64) format. First, a host divides its own MAC address into two 24-bits halves. Then 16-bit Hex value 0xFFFE is sandwiched into those two halves of MAC address, resulting in EUI-64 Interface ID.



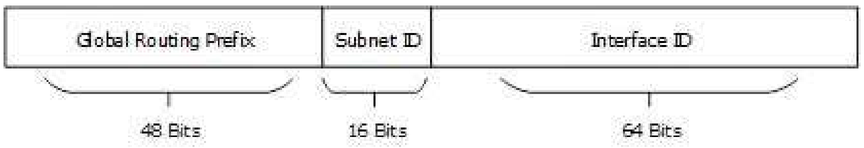
Conversion of EUI-64 ID into IPv6 Interface Identifier

To convert EUI-64 ID into IPv6 Interface Identifier, the most significant 7th bit of EUI-64 ID is complemented. For example:



Global Unicast Address

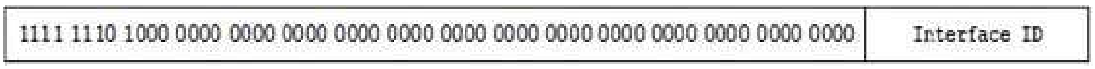
This address type is equivalent to IPv4's public address. Global Unicast addresses in IPv6 are globally identifiable and uniquely addressable.



Global Routing Prefix: The most significant 48-bits are designated as Global Routing Prefix which is assigned to specific autonomous system. The three most significant bits of Global Routing Prefix is always set to 001.

Link-Local Address

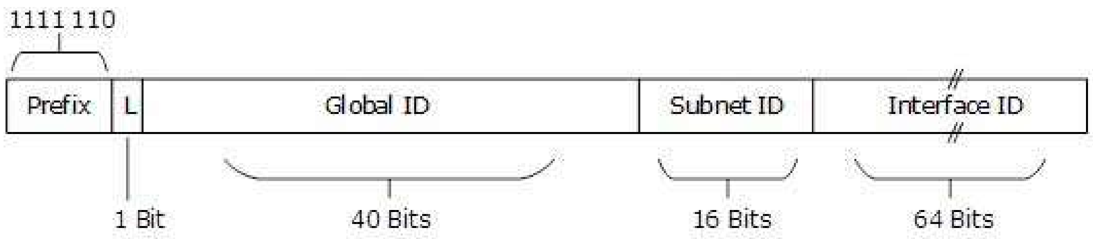
Auto-configured IPv6 address is known as Link-Local address. This address always starts with FE80. The first 16 bits of link-local address is always set to 1111 1110 1000 0000 (FE80). The next 48-bits are set to 0, thus:



Link-local addresses are used for communication among IPv6 hosts on a link (broadcast segment) only. These addresses are not routable, so a Router never forwards these addresses outside the link.

Unique-Local Address

This type of IPv6 address is globally unique, but it should be used in local communication. The second half of this address contain Interface ID and the first half is divided among Prefix, Local Bit, Global ID and Subnet ID.



Prefix is always set to 1111 110. L bit, is set to 1 if the address is locally assigned. So far, the meaning of L bit to 0 is not defined. Therefore, Unique Local IPv6 address always starts with 'FD'.

IPv6 - Special Addresses

Version 6 has slightly complex structure of IP address than that of IPv4. IPv6 has reserved a few addresses and address notations for special purposes. See the table below:

IPv6 Address	Meaning
::/128	Unspecified Address
::/0	Default Route
::1/128	Loopback Address

- As shown in the table, the address 0:0:0:0:0:0:0:0/128 does not specify anything and is said to be an unspecified address. After simplifying, all the 0s are compacted to ::/128.
- In IPv4, the address 0.0.0.0 with netmask 0.0.0.0 represents the default route. The same concept is also applied to IPv6, address 0:0:0:0:0:0:0:0 with netmask all 0s represents the default route. After applying IPv6 rule, this address is compressed to ::/0.

- Loopback addresses in IPv4 are represented by 127.0.0.1 to 127.255.255.255 series. But in IPv6, only 0:0:0:0:0:0:0:1/128 represents the Loopback address. After loopback address, it can be represented as ::1/128.

Reserved Multicast Address for Routing Protocols

IPv6 Address	Routing Protocol
FF02::5	OSPFv3
FF02::6	OSPFv3 Designated Routers
FF02::9	RIPng
FF02::A	EIGRP

- The above table shows the reserved multicast addresses used by interior routing protocol.
- The addresses are reserved following the same rules of IPv4.

Reserved Multicast Address for Routers/Node

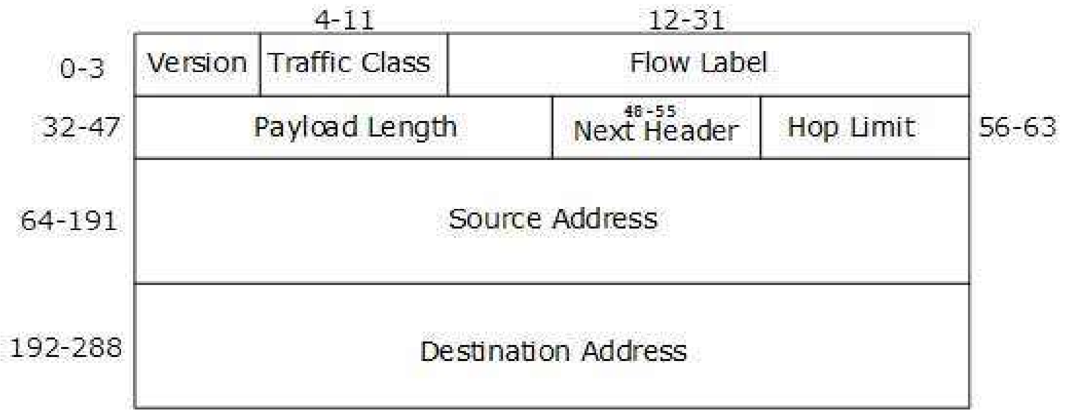
IPv6 Address	Scope
FF01::1	All Nodes in interface-local
FF01::2	All Routers in interface local
FF02::1	All Nodes in link-local
FF02::2	All Routers in link-local
FF05::2	All Routers in site-local

- These addresses help routers and hosts to speak to available routers and hosts on a segment without being configured with an IPv6 address. Hosts use EUI-64 based auto-configuration to self-configure an IPv6 address and then speak to available hosts/routers on the segment by means of these addresses.

IPv6 - Headers

The wonder of IPv6 lies in its header. An IPv6 address is 4 times larger than IPv4, but surprisingly, the header of an IPv6 address is only 2 times larger than that of IPv4. IPv6 headers have one Fixed Header and zero or more Optional (Extension) Headers. All the necessary information that is essential for a router is kept in the Fixed Header. The Extension Header contains optional information that helps routers to understand how to handle a packet/flow.

Fixed Header



IPv6 fixed header is 40 bytes long and contains the following information.

S.N.	Field & Description
1	**Version** (4-bits): It represents the version of Internet Protocol, i.e. 0110.
2	**Traffic Class** (8-bits): These 8 bits are divided into two parts. The most significant 6 bits are used for Type of Service to let the Router Known what services should be provided to this packet. The least significant 2 bits are used for Explicit Congestion Notification (ECN).
3	**Flow Label** (20-bits): This label is used to maintain the sequential flow of the packets belonging to a communication. The source labels the sequence to help the router identify that a particular packet belongs to a specific flow of information. This field helps avoid re-ordering of data packets. It is designed for streaming/real-time media.
4	**Payload Length** (16-bits): This field is used to tell the routers how much information a particular packet contains in its payload. Payload is composed of Extension Headers and Upper Layer data. With 16 bits, up to 65535 bytes can be indicated; but if the Extension Headers contain Hop-by-Hop Extension Header, then the payload may exceed 65535 bytes and this field is set to 0.
5	**Next Header** (8-bits): This field is used to indicate either the type of Extension Header, or if the Extension Header is not present then it indicates the Upper Layer PDU. The values for the type of Upper Layer PDU are same as IPv4's.
6	**Hop Limit** (8-bits): This field is used to stop packet to loop in the network infinitely. This is same as TTL in IPv4. The value of Hop Limit field is decremented by 1 as it passes a link (router/hop). When the field reaches 0 the packet is discarded.
7	**Source Address** (128-bits): This field indicates the address of originator of the packet.
8	**Destination Address** (128-bits): This field provides the address of intended recipient of the packet.

Extension Headers

In IPv6, the Fixed Header contains only that much information which is necessary, avoiding those information which is either not required or is rarely used. All such information is put between the Fixed Header and the Upper layer header in the form of Extension Headers. Each Extension Header is identified by a distinct value.

When Extension Headers are used, IPv6 Fixed Header's Next Header field points to the first Extension Header. If there is one more Extension Header, then the first Extension Header's 'Next-Header' field points to the second one, and so on. The last Extension Header's 'Next-Header' field points to the Upper Layer Header. Thus, all the headers points to the next one in a linked list manner.

If the Next Header field contains the value 59, it indicates that there are no headers after this header, not even Upper Layer Header.

The following Extension Headers must be supported as per RFC 2460:

Extension Header	Next Header Value	Description
Hop-by-Hop Options header	0	read by all devices in transit network
Routing header	43	contains methods to support making routing decision
Fragment header	44	contains parameters of datagram fragmentation
Destination Options header	60	read by destination devices
Authentication header	51	information regarding authenticity
Encapsulating Security Payload header	50	encryption information

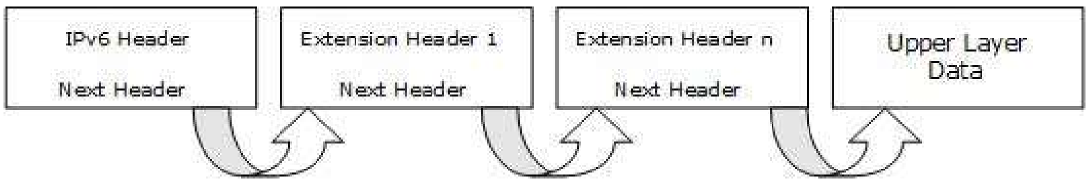
The sequence of Extension Headers should be:

IPv6 header
Hop-by-Hop Options header
Destination Options header[1]
Routing header
Fragment header
Authentication header
Encapsulating Security Payload header
Destination Options header[2]
Upper-layer header

These headers:

- 1. should be processed by First and subsequent destinations.
- 2. should be processed by Final Destination.

Extension Headers are arranged one after another in a linked list manner, as depicted in the following diagram:



IPv6 - Communication

In IPv4, a host that wants to communicate with another host on the network needs to have an IP address acquired either by means of DHCP or by manual configuration. As soon as a host is equipped with some valid IP address, it can speak to any host on the subnet. To communicate on layer-3, a host must also know the IP address of the other host. Communication on a link, is established by means of hardware embedded MAC Addresses. To know the MAC address of a host whose IP address is known, a host sends ARP broadcast and in return, the intended host sends back its MAC address.

In IPv6, there are no broadcast mechanisms. It is not a must for an IPv6 enabled host to obtain an IP address from DHCP or manually configured, but it can auto-configure its own IP.

ARP has been replaced by ICMPv6 Neighbor Discovery Protocol.

Neighbor Discovery Protocol

A host in IPv6 network is capable of auto-configuring itself with a unique link-local address. As soon as host gets an IPv6 address, it joins a number of multicast groups. All communications related to that segment take place on those multicast addresses only. A host goes through a series of states in IPv6:

- **Neighbor Solicitation**: After configuring all IPv6's either manually, or by DHCP Server or by auto-configuration, the host sends a Neighbor Solicitation message out to FF02::1/16 multicast address for all its IPv6 addresses in order to know that no one else occupies the same addresses.
- **DAD (Duplicate Address Detection)**: When the host does not listen from anything from the segment regarding its Neighbor Solicitation message, it assumes that no duplicate address exists on the segment.
- **Neighbor Advertisement**: After assigning the addresses to its interfaces and making them up and running, the host once again sends out a Neighbor Advertisement message telling all other hosts on the segment, that it has assigned those IPv6 addresses to its interfaces.

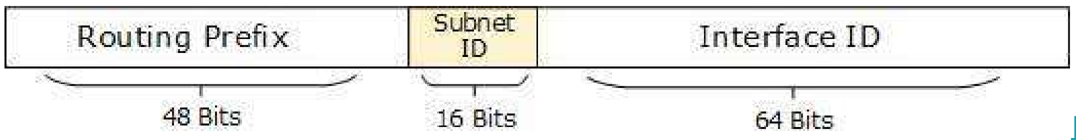
Once a host is done with the configuration of its IPv6 addresses, it does the following things:

- **Router Solicitation**: A host sends a Router Solicitation multicast packet (FF02::2/16) out on its segment to know the presence of any router on this segment. It helps the host to configure the router as its default gateway. If its default gateway router goes down, the host can shift to a new router and makes it the default gateway.
- **Router Advertisement**: When a router receives a Router Solicitation message, it response back to the host, advertising its presence on that link.
- **Redirect**: This may be the situation where a Router receives a Router Solicitation request but it knows that it is not the best gateway for the host. In this situation, the router sends back a Redirect message telling the host that there is a better 'next-hop' router available. Next-hop is where the host will send its data destined to a host which does not belong to the same segment.

IPv6 - Subnetting

In IPv4, addresses were created in classes. Classful IPv4 addresses clearly define the bits used for network prefixes and the bits used for hosts on that network. To subnet in IPv4, we play with the default classful netmask which allows us to borrow host bits to be used as subnet bits. This results in multiple subnets but less hosts per subnet. That is, when we borrow host bits to create a subnet, it costs us in lesser bit to be used for host addresses.

IPv6 addresses use 128 bits to represent an address which includes bits to be used for subnetting. The second half of the address (least significant 64 bits) is always used for hosts only. Therefore, there is no compromise if we subnet the network.



[Image: IPv6 Subnetting]

16 bits of subnet is equivalent to IPv4's Class B Network. Using these subnet bits, an organization can have another 65 thousands of subnets which is by far, more than enough.

Thus routing prefix is /64 and host portion is 64 bits. We can further subnet the network beyond 16 bits of Subnet ID, by borrowing host bits; but it is recommended that 64 bits should always be used for hosts addresses because auto-configuration requires 64 bits.

IPv6 subnetting works on the same concept as Variable Length Subnet Masking in IPv4.

/48 prefix can be allocated to an organization providing it the benefit of having up to /64 subnet prefixes, which is 65535 sub-networks, each having 2^{64}hosts. A /64 prefix can be assigned to a point-to-point connection where there are only two hosts (or IPv6 enabled devices) on a link.

Transition From IPv4 to IPv6

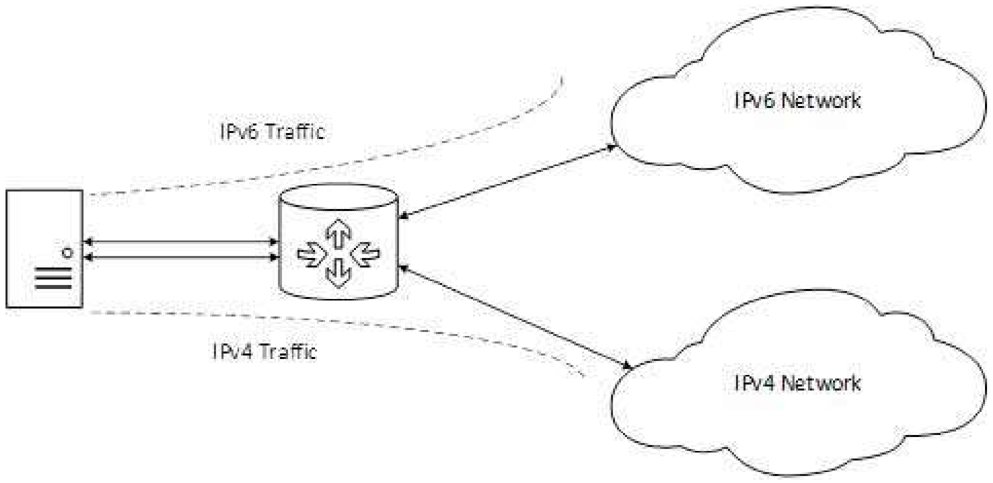
Advertisements

Complete transition from IPv4 to IPv6 might not be possible because IPv6 is not backward compatible. This results in a situation where either a site is on IPv6 or it is not. It is unlike implementation of other new technologies where the newer one is backward compatible so the older system can still work with the newer version without any additional changes.

To overcome this short-coming, we have a few technologies that can be used to ensure slow and smooth transition from IPv4 to IPv6.

Dual Stack Routers

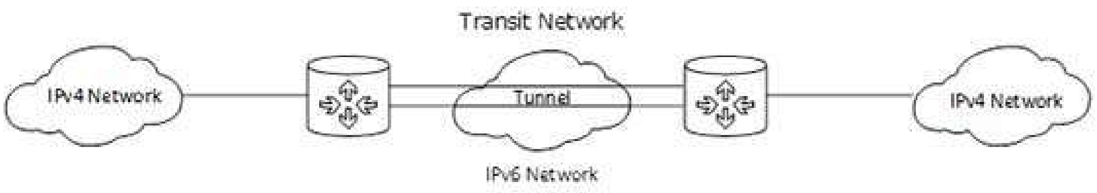
A router can be installed with both IPv4 and IPv6 addresses configured on its interfaces pointing to the network of relevant IP scheme.



In the above diagram, a server having IPv4 as well as IPv6 address configured for it can now speak with all the hosts on both the IPv4 as well as the IPv6 networks with the help of a Dual Stack Router. The Dual Stack Router, can communicate with both the networks. It provides a medium for the hosts to access a server without changing their respective IP versions.

Tunneling

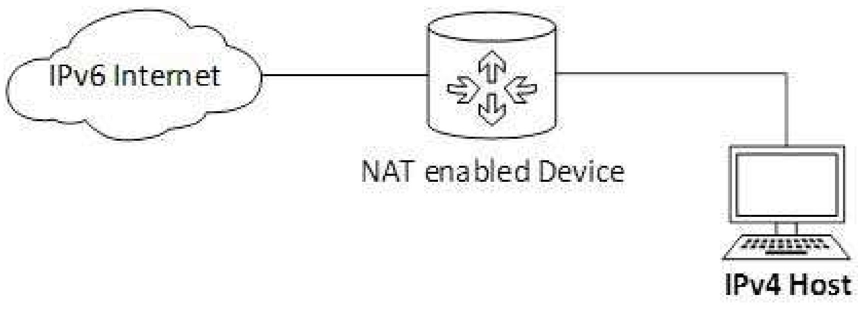
In a scenario where different IP versions exist on intermediate path or transit networks, tunneling provides a better solution where user's data can pass through a non-supported IP version.



The above diagram depicts how two remote IPv4 networks can communicate via a Tunnel, where the transit network was on IPv6. Vice versa is also possible where the transit network is on IPv6 and the remote sites that intend to communicate are on IPv4.

NAT Protocol Translation

This is another important method of transition to IPv6 by means of a NAT-PT (Network Address Translation – Protocol Translation) enabled device. With the help of a NAT-PT device, actual can take place happens between IPv4 and IPv6 packets and vice versa. See the diagram below:



A host with IPv4 address sends a request to an IPv6 enabled server on Internet that does not understand IPv4 address. In this scenario, the NAT-PT device can help them communicate. When the IPv4 host sends a request packet to the IPv6 server, the NAT-PT device/router strips down the IPv4 packet, removes IPv4 header, and adds IPv6 header and passes it through the Internet. When a response from the IPv6 server comes for the IPv4 host, the router does vice versa.

IPv6 - Routing

Routing concepts remain same in case of IPv6 but almost all routing protocols have been redefined accordingly. We discussed earlier, how a host speaks to its gateway. Routing is a process to forward routable data choosing the best route among several available routes or path to the destination. A router is a device that forwards data that is not explicitly destined to it.

There exists two forms of routing protocols:

- **Distance Vector Routing Protocol**: A router running distance vector protocol advertises its connected routes and learns new routes from its neighbors. The routing cost to reach a destination is calculated by means of hops between the source and destination. A router generally relies on its neighbor for best path selection, also known as "routing-by-rumors". RIP and BGP are Distance Vector Protocols.
- **Link-State Routing Protocol**: This protocol acknowledges the state of a Link and advertises to its neighbors. Information about new links is learnt from peer routers. After all the routing information has been converged, the Link-State Routing Protocol uses its own algorithm to calculate the best path to all available links. OSPF and IS-IS are link state routing protocols and both of them use Dijkstra's Shortest Path First algorithm.

Routing protocols can be divided in two categories:

- **Interior Routing Protocol**: Protocols in this categories are used within an autonomous system or organization to distribute routes among all routers inside its boundary. Examples: RIP, OSPF.
- **Exterior Routing Protocol**: An Exterior Routing Protocol distributes routing information between two different autonomous systems or organization. Examples: BGP.

Routing protocols

- **RIPng**

  RIPng stands for Routing Information Protocol Next Generation. This is an Interior Routing Protocol and is a Distance Vector Protocol. RIPng has been upgraded to support IPv6.

- **OSPFv3**

  Open Shortest Path First version 3 is an Interior Routing Protocol which is modified to support IPv6. This is a Link-State Protocol and uses Djikrasta's Shortest Path First algorithm to calculate best path to all destinations.

- **BGPv4**

  BGP stands for Border Gateway Protocol. It is the only open standard Exterior Gateway Protocol available. BGP is a Distance Vector protocol which takes Autonomous System as calculation metric, instead of the number of routers as Hop. BGPv4 is an upgrade of BGP to support IPv6 routing.

Protocols Changed to Support IPv6

- **ICMPv6**: Internet Control Message Protocol version 6 is an upgraded implementation of ICMP to accommodate IPv6 requirements. This protocol is used for diagnostic functions, error and information message, statistical purposes. ICMPv6's Neighbor Discovery Protocol replaces ARP and helps discover neighbor and routers on the link.

- **DHCPv6**: Dynamic Host Configuration Protocol version 6 is an implementation of DHCP. IPv6 enabled hosts do not require any DHCPv6 Server to acquire IP address as they can be auto-configured. Neither do they need DHCPv6 to locate DNS server because DNS can be discovered and configured via ICMPv6 Neighbor Discovery Protocol. Yet DHCPv6 Server can be used to provide these information.

- **DNS**: There has been no new version of DNS but it is now equipped with extensions to provide support for querying IPv6 addresses. A new AAAA (quad-A) record has been added to reply IPv6 query messages. Now the DNS can reply with both IP versions (4 & 6) without any change in the query format.

IPv6 - Mobility

When a host is connected to a link or network, it acquires an IP address and all communication take place using that IP address on that link. As soon as, the same host changes its physical location, that is, moves into another area / subnet / network / link, its IP address changes accordingly, and all the communication taking place on the host using old IP address, goes down.

IPv6 mobility provides a mechanism for the host to roam around different links without losing any communication/connection and its IP address.

Multiple entities are involved in this technology:

- **Mobile Node**: The device that needs IPv6 mobility.

- **Home Link**: This link is configured with the home subnet prefix and this is where the Mobile IPv6 device gets its Home Address.

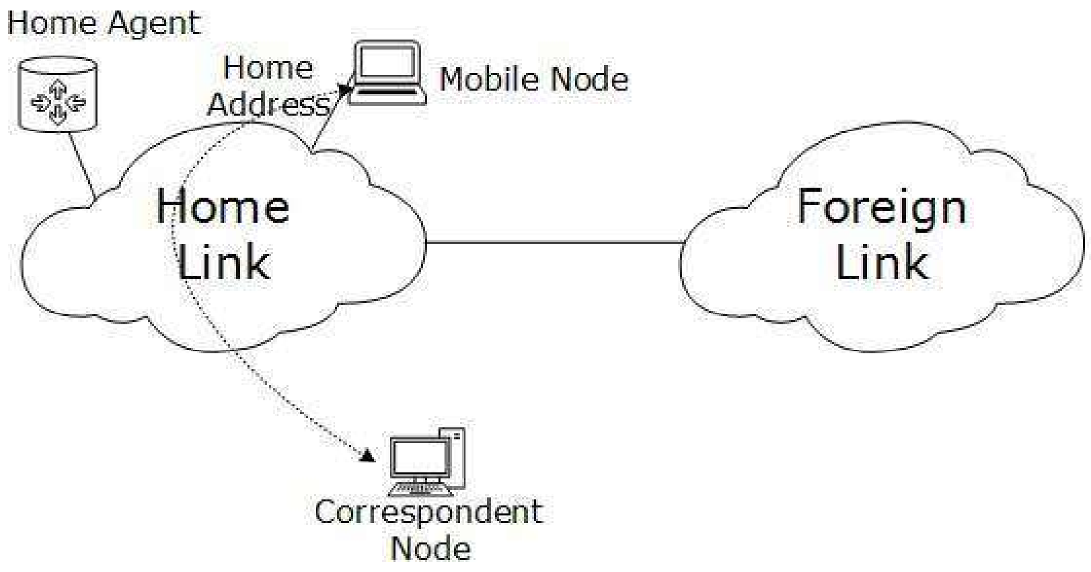
- **Home Address**: This is the address which the Mobile Node acquires from the Home Link. This is the permanent address of the Mobile Node. If the Mobile Node remains in the same Home Link, the communication among various entities take place as usual.

- **Home Agent**: This is a router that acts as a registrar for Mobile Nodes. Home Agent is connected to Home Link and maintains information about all Mobile Nodes, their Home Addresses, and their present IP addresses.

- **Foreign Link**: Any other Link that is not Mobile Node's Home Link.
- **Care-of Address**: When a Mobile Node gets attached to a Foreign Link, it acquires a new IP address of that Foreign Link's subnet. Home Agent maintains the information of both Home Address and Care-of Address. Multiple Care-of addresses can be assigned to a Mobile Node, but at any instance, only one Care-of Address has binding with the Home Address.
- **Correspondent Node**: Any IPv6 enabled device that intends to have communication with Mobile Node.

Mobility Operation

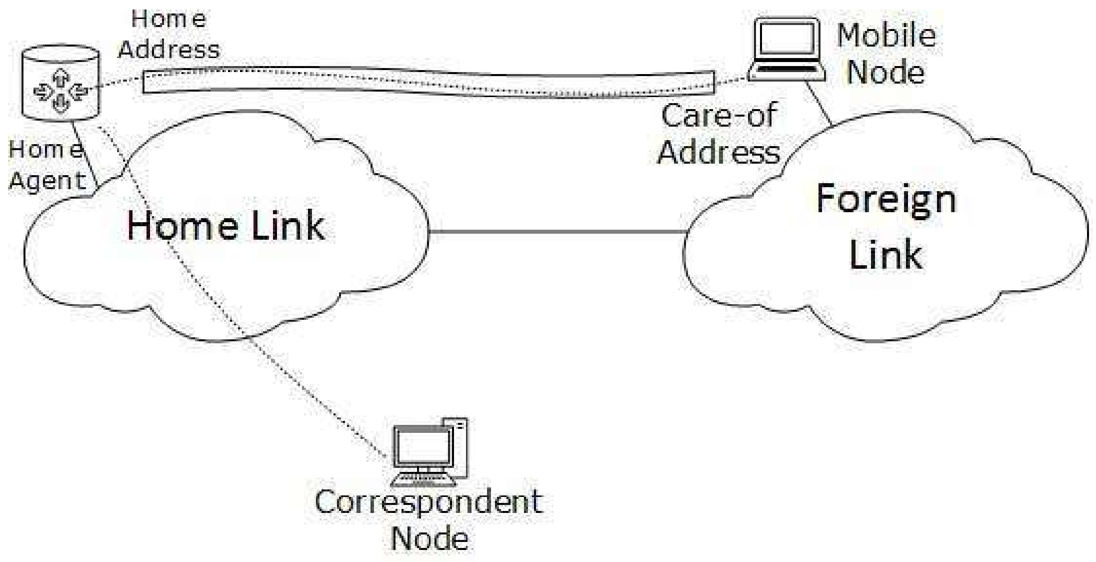
When Mobile Node stays in its Home Link, all communications take place on its Home Address as shown below:



Mobile Node connected to Home Link]

When a Mobile Node leaves its Home Link and is connected to some Foreign Link, the Mobility feature of IPv6 comes into play. After getting connected to a Foreign Link, the Mobile Node acquires an IPv6 address from the Foreign Link. This address is called Care-of Address. The Mobile Node sends a binding request to its Home Agent with the new Care-of Address. The Home Agent binds the Mobile Node's Home Address with the Care-of Address, establishing a Tunnel between both.

Whenever a Correspondent Node tries to establish connection with the Mobile Node (on its Home Address), the Home Agent intercepts the packet and forwards to Mobile Node's Care-of Address over the Tunnel which was already established.



Mobile Node connected to Foreign Link]

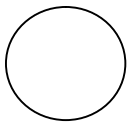
Route Optimization

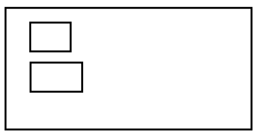
When a Correspondent Node initiates a communication by sending packets to Mobile the Node on the Home Address, these packets are tunneled to the Mobile Node by the Home Agent. In Route Optimization mode, when the Mobile Node receives a packet from the Correspondent Node, it does not forward replies to the Home Agent. Rather, it sends its packet directly to the Correspondent Node using Home Address as Source Address. This mode is optional and not used by default.

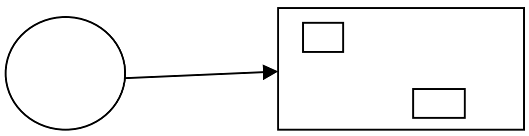
IT-5th Semester
Operating System (PCC-CS502)
Lecture-13

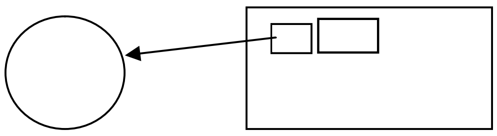
Resource Allocation Graphs:

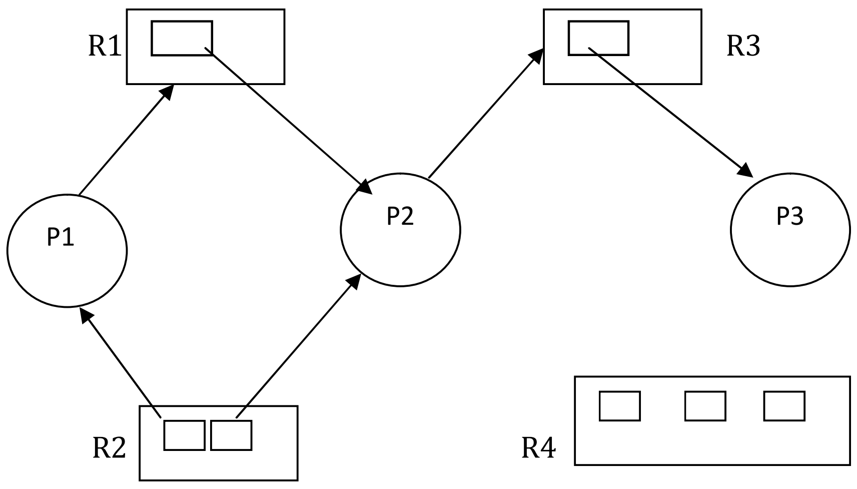
Deadlocks can be described more precisely in terms of a directed graph called a system resource allocation graph. This graph consists of a set of vertices V and a set of edges E. The set of vertices is portioned into two different types of nodes P={P0, P1… Pn}, the set of the active processes in the system, and R={R0, R1… Rn}, the set consisting of all resource types in the system. A directed edge from a process Pi to resource type Rj signifies that process Pi requested an instance of Rj and is waiting for that resource. A directed edge from Rj to Pi signifies that an instance of Rj has been allocated to Pi.

• Process 

• Resource Type with 2 instances 

• Pi requests instance of Rj 

• Pi is holding an instance of Rj 



The resource allocation graph shown above depicts the following situation:
P= {P1, P2, P3 }
R= {R1, R2, R3}
E= {P1 → R1, P2 → R3, R1 → P2, R2 → P2, R2 → P1, P3 → R3}

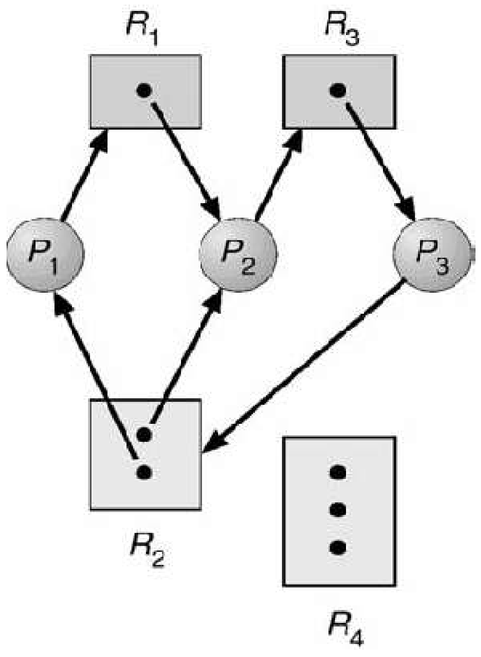
Resource Instances
One instance of resource type R1
Two instances of resource type R2
One instance of resource type R3
Three instances of resource type R4

Process States
Process P1 is holding an instance of resource R2, and is waiting for an instance of resource R1.
Process P2 is holding an instance of resource R1 and R2, and is waiting for an instance of resource R3.
Process P3 is holding an instance of resource R3.

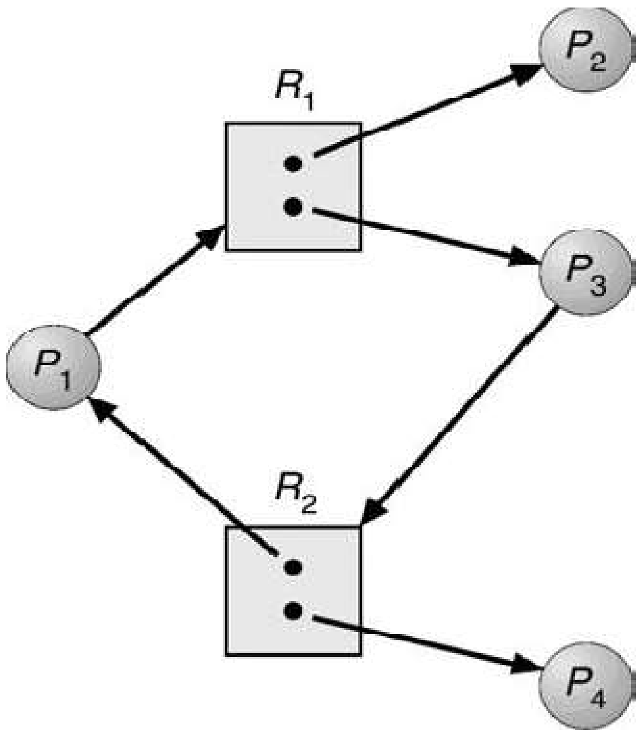
Given the definition of a resource allocation graph, it can be shown that if the graph contains no cycles, then no process is deadlocked.

If the graph contains cycles then:

- If only one instance per resource type, then a deadlock exists.
- If several instances per resource type, possibility of deadlock exists.



Here is a resource allocation graph with a deadlock. There are two cycles in this graph:
{P1 → R1, R1 → P2, P2 → R3, R3 → P3, P3 → R2, R2 → P1} and
{P2 → R3, R3 → P3, P3 → R2, R2 → P2}
No process will release an already acquired resource and the three processes will remain in the deadlock state.



The graph shown above has a cycle but there is no deadlock because processes P2 and P4 do not require further resources to complete their execution and will release the resources they are currently hold in finite time. These resources can then be allocated to P1 and P3 for them to resume their execution.

UNIT III – INTERMEDIATE CODE GENERATION

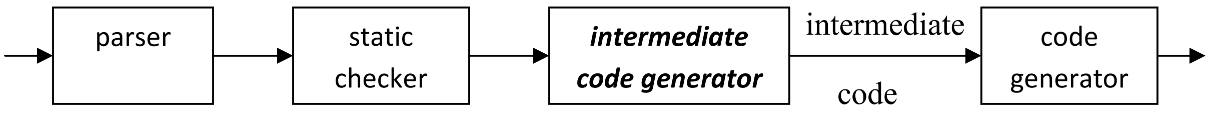
INTRODUCTION

The front end translates a source program into an intermediate representation from which the back end generates target code.

Benefits of using a machine-independent intermediate form are:

1. Retargeting is facilitated. That is, a compiler for a different machine can be created by attaching a back end for the new machine to an existing front end.
2. A machine-independent code optimizer can be applied to the intermediate representation.

Position of intermediate code generator



INTERMEDIATE LANGUAGES

Three ways of intermediate representation:

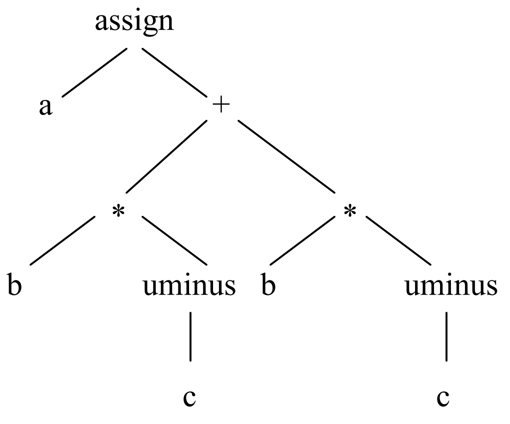
- Syntax tree
- Postfix notation
- Three address code

The semantic rules for generating three-address code from common programming language constructs are similar to those for constructing syntax trees or for generating postfix notation.

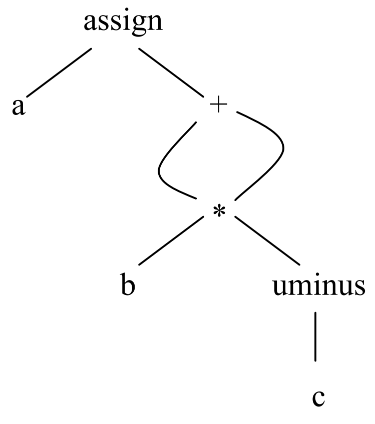
Graphical Representations:

Syntax tree:

A syntax tree depicts the natural hierarchical structure of a source program. A **dag (Directed Acyclic Graph)** gives the same information but in a more compact way because common subexpressions are identified. A syntax tree and dag for the assignment statement **a : = b * - c + b * - c** are as follows:



(a) Syntax tree



(b) Dag

Postfix notation:

Postfix notation is a linearized representation of a syntax tree; it is a list of the nodes of the tree in which a node appears immediately after its children. The postfix notation for the syntax tree given above is

a b c uminus * b c uminus * + assign

Syntax-directed definition:

Syntax trees for assignment statements are produced by the syntax-directed definition. Non-terminal S generates an assignment statement. The two binary operators + and * are examples of the full operator set in a typical language. Operator associativities and precedences are the usual ones, even though they have not been put into the grammar. This definition constructs the tree from the input a : = b * - c + b* - c.

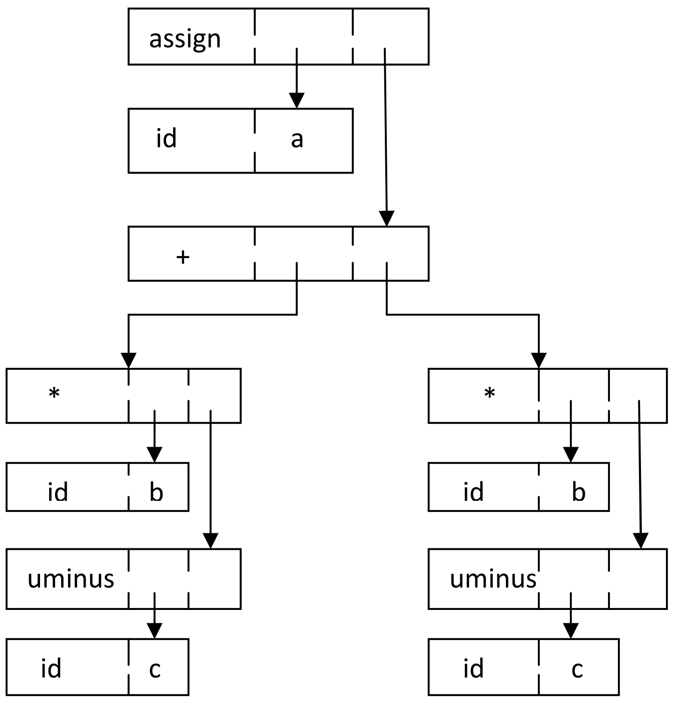
PRODUCTION	SEMANTIC RULE
S → id : = E	S.nptr : = mknode('assign',mkleaf(id, id.place), E.nptr)
E → E_1 + E_2	E.nptr : = mknode('+', E_1.nptr, E_2.nptr)
E → E_1 * E_2	E.nptr : = mknode('*', E_1.nptr, E_2.nptr)
E → - E_1	E.nptr : = mknode('uminus', E_1.nptr)
E → (E_1)	E.nptr : = E_1.nptr
E → id	E.nptr : = mkleaf(id, id.place)

Syntax-directed definition to produce syntax trees for assignment statements

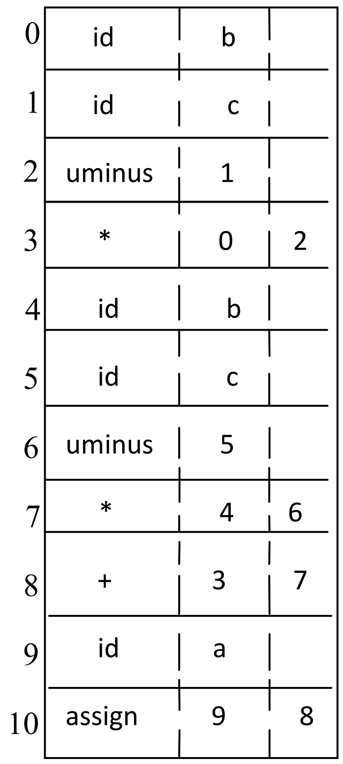
The token **id** has an attribute *place* that points to the symbol-table entry for the identifier. A symbol-table entry can be found from an attribute **id**.*name*, representing the lexeme associated with that occurrence of **id.** If the lexical analyzer holds all lexemes in a single array of characters, then attribute *name* might be the index of the first character of the lexeme.

Two representations of the syntax tree are as follows. In (a) each node is represented as a record with a field for its operator and additional fields for pointers to its children. In (b), nodes are allocated from an array of records and the index or position of the node serves as the pointer to the node. All the nodes in the syntax tree can be visited by following pointers, starting from the root at position 10.

Two representations of the syntax tree



(a)



0	id	b	
1	id	c	
2	uminus	1	
3	*	0	2
4	id	b	
5	id	c	
6	uminus	5	
7	*	4	6
8	+	3	7
9	id	a	
10	assign	9	8

(b)

Three-Address Code:

Three-address code is a sequence of statements of the general form

x : = y *op* z

where x, y and z are names, constants, or compiler-generated temporaries; *op* stands for any operator, such as a fixed- or floating-point arithmetic operator, or a logical operator on boolean-valued data. Thus a source language expression like x+ y*z might be translated into a sequence

t_1 : = y * z
t_2 : = x + t_1

where t_1 and t_2 are compiler-generated temporary names.

Advantages of three-address code:

- ➢ The unraveling of complicated arithmetic expressions and of nested flow-of-control statements makes three-address code desirable for target code generation and optimization.
- ➢ The use of names for the intermediate values computed by a program allows three-address code to be easily rearranged – unlike postfix notation.

Three-address code is a linearized representation of a syntax tree or a dag in which explicit names correspond to the interior nodes of the graph. The syntax tree and dag are represented by the three-address code sequences. Variable names can appear directly in three-address statements.

Three-address code corresponding to the syntax tree and dag given above

$t_1 := -c$	$t_1 := -c$
$t_2 := b * t_1$	$t_2 := b * t_1$
$t_3 := -c$	$t_5 := t_2 + t_2$
$t_4 := b * t_3$	$a := t_5$
$t_5 := t_2 + t_4$	
$a := t_5$	
(a) Code for the syntax tree	**(b) Code for the dag**

The reason for the term “three-address code” is that each statement usually contains three addresses, two for the operands and one for the result.

Types of Three-Address Statements:

The common three-address statements are:

1. Assignment statements of the form **x : = y *op* z**, where ***op*** is a binary arithmetic or logical operation.
2. Assignment instructions of the form **x : = *op* y**, where ***op*** is a unary operation. Essential unary operations include unary minus, logical negation, shift operators, and conversion operators that, for example, convert a fixed-point number to a floating-point number.
3. *Copy statements* of the form **x : = y** where the value of *y* is assigned to *x*.
4. The unconditional jump goto L. The three-address statement with label L is the next to be executed.
5. Conditional jumps such as **if *x relop y* goto L**. This instruction applies a relational operator (<, =, >=, etc.) to *x* and *y*, and executes the statement with label L next if *x* stands in relation

relop to y. If not, the three-address statement following if *x relop y* goto L is executed next, as in the usual sequence.

6. *param x* and *call p, n* for procedure calls and *return y*, where y representing a returned value is optional. For example,

```
param x1
param x2
  . . .
param xn
call p,n
```

generated as part of a call of the procedure p(x_1, x_2, …. ,x_n).

7. Indexed assignments of the form x : = y[i] and x[i] : = y.

8. Address and pointer assignments of the form x : = &y , x : = *y, and *x : = y.

Syntax-Directed Translation into Three-Address Code:

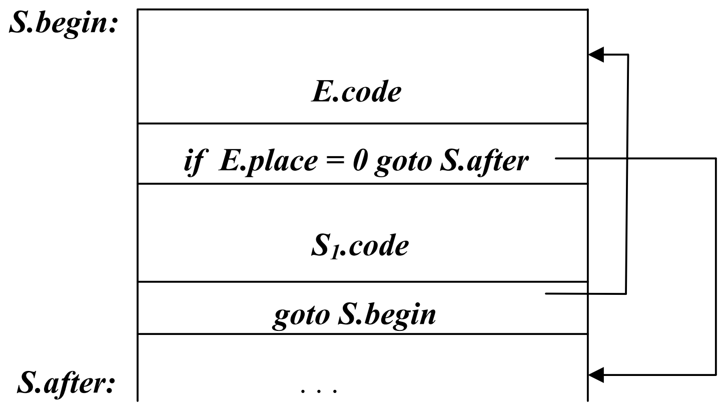
When three-address code is generated, temporary names are made up for the interior nodes of a syntax tree. For example, **id : =** ***E*** consists of code to evaluate ***E*** into some temporary t, followed by the assignment **id**.*place* : = **t.**

Given input a : = b * - c + b * - c, the three-address code is as shown above. The synthesized attribute *S.code* represents the three-address code for the assignment *S.*
The nonterminal *E* has two attributes :
1. *E.place*, the name that will hold the value of *E* , and
2. *E.code*, the sequence of three-address statements evaluating *E.*

Syntax-directed definition to produce three-address code for assignments

PRODUCTION	SEMANTIC RULES
S → id : = E	***S.code : = E.code || gen(id.place ':=' E.place)***
E → E_1 + E_2	***E.place := newtemp;*** ***E.code := E_1.code || E_2.code || gen(E.place ':=' E_1.place '+' E_2.place)***
E → E_1 * E_2	***E.place := newtemp;*** ***E.code := E_1.code || E_2.code || gen(E.place ':=' E_1.place '*' E_2.place)***
E → - E_1	***E.place := newtemp;*** ***E.code := E_1.code || gen(E.place ':=' 'uminus' E_1.place)***
E → (E_1)	***E.place : = E_1.place;*** ***E.code : = E_1.code***
E → id	***E.place : = id.place;*** ***E.code : = ' '***

Semantic rules generating code for a while statement



PRODUCTION	SEMANTIC RULES
S → **while** *E* **do** *S₁*	***S.begin := newlabel;*** ***S.after := newlabel;*** ***S.code := gen(S.begin ':') ||*** ***E.code ||*** ***gen ('if' E.place '=' '0' 'goto' S.after)||*** ***S₁.code ||*** ***gen ('goto' S.begin) ||*** ***gen (S.after ':')***

- The function *newtemp* returns a sequence of distinct names $t_1, t_2, \ldots$.. in response to successive calls.
- Notation *gen(x ':=' y '+' z)* is used to represent three-address statement x := y + z. Expressions appearing instead of variables like *x, y* and *z* are evaluated when passed to *gen*, and quoted operators or operand, like '+' are taken literally.
- Flow-of–control statements can be added to the language of assignments. The code for ***S*** → **while** ***E*** **do** ***S₁*** is generated using new attributes *S.begin* and *S.after* to mark the first statement in the code for *E* and the statement following the code for S, respectively.
- The function *newlabel* returns a new label every time it is called.
- We assume that a non-zero expression represents true; that is when the value of ***E*** becomes zero, control leaves the while statement.

Implementation of Three-Address Statements:

A three-address statement is an abstract form of intermediate code. In a compiler, these statements can be implemented as records with fields for the operator and the operands. Three such representations are:

- Quadruples
- Triples
- Indirect triples

Quadruples:

- A quadruple is a record structure with four fields, which are, ***op, arg1, arg2*** and ***result.***
- The *op* field contains an internal code for the operator. The three-address statement **x : = y op z** is represented by placing *y* in *arg1*, *z* in *arg2* and *x* in *result.*
- The contents of fields arg1, arg2 and result are normally pointers to the symbol-table entries for the names represented by these fields. If so, temporary names must be entered into the symbol table as they are created.

Triples:

- To avoid entering temporary names into the symbol table, we might refer to a temporary value by the position of the statement that computes it.
- If we do so, three-address statements can be represented by records with only three fields: *op, arg1* and *arg2.*
- The fields *arg1* and *arg2*, for the arguments of *op*, are either pointers to the symbol table or pointers into the triple structure (for temporary values).
- Since three fields are used, this intermediate code format is known as *triples*.

	op	*arg1*	*arg2*	*result*
(0)	uminus	c		t_1
(1)	*	b	t_1	t_2
(2)	uminus	c		t_3
(3)	*	b	t_3	t_4
(4)	+	t_2	t_4	t_5
(5)	: =	t_3		a

(a) Quadruples

	op	*arg1*	*arg2*
(0)	uminus	c	
(1)	*	b	(0)
(2)	uminus	c	
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	assign	a	(4)

(b) Triples

Quadruple and triple representation of three-address statements given above

A ternary operation like x[i] : = y requires two entries in the triple structure as shown as below while x : = y[i] is naturally represented as two operations.

	op	*arg1*	*arg2*
(0)	[ ] =	x	i
(1)	assign	(0)	y

(a) x[i] : = y

	op	*arg1*	*arg2*
(0)	= [ ]	y	i
(1)	assign	x	(0)

(b) x : = y[i]

Indirect Triples:

- Another implementation of three-address code is that of listing pointers to triples, rather than listing the triples themselves. This implementation is called indirect triples.
- For example, let us use an array statement to list pointers to triples in the desired order. Then the triples shown above might be represented as follows:

	statement
(0)	(14)
(1)	(15)
(2)	(16)
(3)	(17)
(4)	(18)
(5)	(19)

	op	*arg1*	*arg2*
(14)	uminus	c	
(15)	*	b	(14)
(16)	uminus	c	
(17)	*	b	(16)
(18)	+	(15)	(17)
(19)	assign	a	(18)

Indirect triples representation of three-address statements

DECLARATIONS

As the sequence of declarations in a procedure or block is examined, we can lay out storage for names local to the procedure. For each local name, we create a symbol-table entry with information like the type and the relative address of the storage for the name. The relative address consists of an offset from the base of the static data area or the field for local data in an activation record.

Declarations in a Procedure:

The syntax of languages such as C, Pascal and Fortran, allows all the declarations in a single procedure to be processed as a group. In this case, a global variable, say *offset*, can keep track of the next available relative address.

In the translation scheme shown below:

- Nonterminal P generates a sequence of declarations of the form **id : *T.***
- Before the first declaration is considered, *offset* is set to 0. As each new name is seen , that name is entered in the symbol table with offset equal to the current value of *offset*, and *offset* is incremented by the width of the data object denoted by that name.
- The procedure *enter(name, type, offset*) creates a symbol-table entry for *name*, gives its type *type* and relative address *offset* in its data area.
- Attribute *type* represents a type expression constructed from the basic types *integer* and *real* by applying the type constructors *pointer* and *array*. If type expressions are represented by graphs, then attribute *type* might be a pointer to the node representing a type expression.
- The width of an array is obtained by multiplying the width of each element by the number of elements in the array. The width of each pointer is assumed to be 4.

Computing the types and relative addresses of declared names

P → D	***{ offset : = 0 }***
D → D ; D	

D → id : T	***{ enter(id.name, T.type, offset);
	offset : = offset + T.width }***

T → integer	***{ T.type : = integer;
	T.width : = 4 }***

T → real	***{ T.type : = real;
	T.width : = 8 }***

T → array [num] of T_1	***{ T.type : = array(num.val,*** T_1***.type);
	T.width : = num.val X*** T_1***.width }***

T → $\uparrow$ T_1	***{ T.type : = pointer (*** T_1***.type);
	T.width : = 4 }***

Keeping Track of Scope Information:

When a nested procedure is seen, processing of declarations in the enclosing procedure is temporarily suspended. This approach will be illustrated by adding semantic rules to the following language:

$P \rightarrow D$

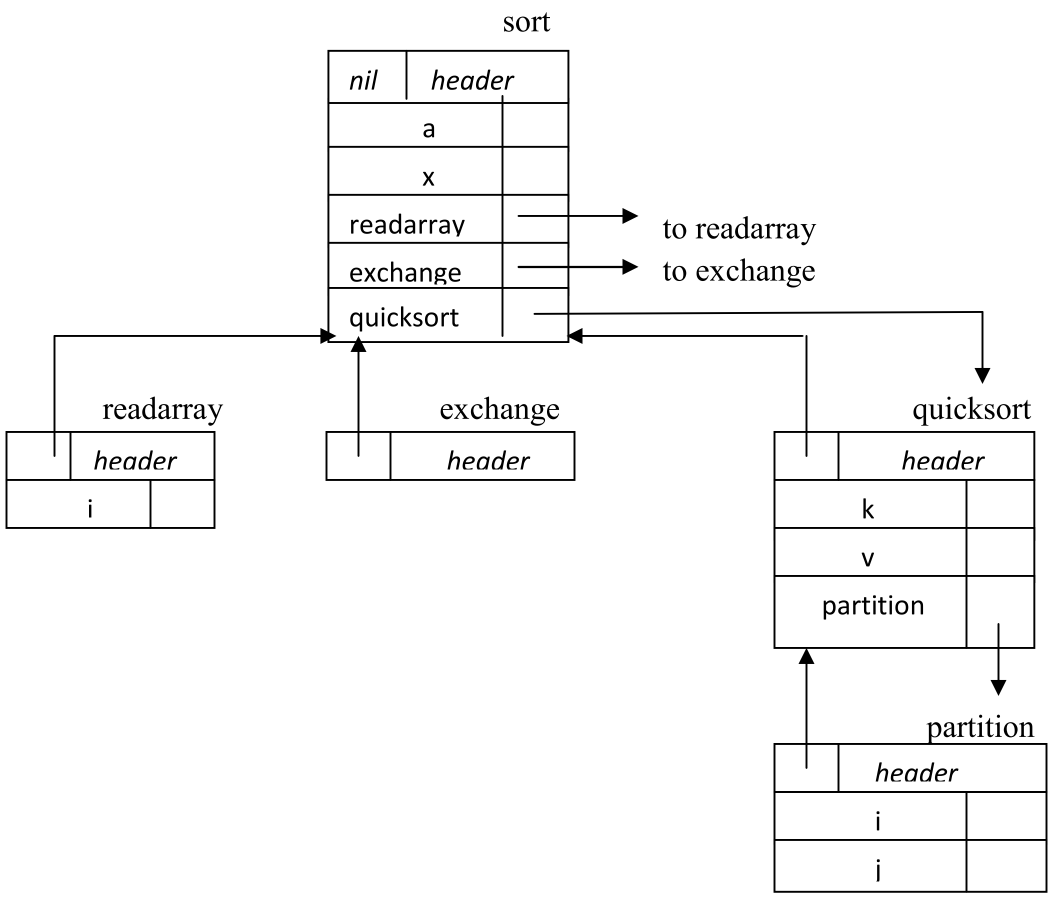
$D \rightarrow D\ ;\ D$ | **id** $: T$ | **proc id** $;\ D\ ;\ S$

One possible implementation of a symbol table is a linked list of entries for names.

A new symbol table is created when a procedure declaration $D \rightarrow$ ***proc id*** $D_1;S$ is seen, and entries for the declarations in D_1 are created in the new table. The new table points back to the symbol table of the enclosing procedure; the name represented by id itself is local to the enclosing procedure. The only change from the treatment of variable declarations is that the procedure *enter* is told which symbol table to make an entry in.

For example, consider the symbol tables for procedures *readarray, exchange*, and *quicksort* pointing back to that for the containing procedure *sort*, consisting of the entire program. Since *partition* is declared within *quicksort*, its table points to that of *quicksort*.

Symbol tables for nested procedures



The semantic rules are defined in terms of the following operations:

1. *mktable(previous)* creates a new symbol table and returns a pointer to the new table. The argument *previous* points to a previously created symbol table, presumably that for the enclosing procedure.

2. *enter(table, name, type, offset)* creates a new entry for name *name* in the symbol table pointed to by *table.* Again, *enter* places type *type* and relative address *offset* in fields within the entry.

3. *addwidth(table, width)* records the cumulative width of all the entries in table in the header associated with this symbol table.

4. *enterproc(table, name, newtable)* creates a new entry for procedure *name* in the symbol table pointed to by *table*. The argument *newtable* points to the symbol table for this procedure *name*.

Syntax directed translation scheme for nested procedures

P → M D	***{ addwidth (top(tblptr) , top (offset)); pop (tblptr); pop (offset) }***
M → ε	***{ t : = mktable (nil); push (t,tblptr); push (0,offset) }***
D → D₁ ; D₂	
D → proc id ; N D₁ ; S	***{ t : = top (tblptr); addwidth (t, top (offset)); pop (tblptr); pop (offset); enterproc (top (tblptr), id.name, t) }***
D → id : T	***{ enter (top (tblptr), id.name, T.type, top (offset)); top (offset) := top (offset) + T.width }***
N → ε	***{ t := mktable (top (tblptr)); push (t, tblptr); push (0,offset) }***

- The stack *tblptr* is used to contain pointers to the tables for **sort, quicksort,** and **partition** when the declarations in **partition** are considered.

- The top element of stack *offset* is the next available relative address for a local of the current procedure.

- All semantic actions in the subtrees for B and C in

  $A \rightarrow BC$ {$action_A$}

  are done before $action_A$ at the end of the production occurs. Hence, the action associated with the marker M is the first to be done.

- The action for nonterminal M initializes stack *tblptr* with a symbol table for the outermost scope, created by operation *mktable(nil).* The action also pushes relative address 0 onto stack offset.
- Similarly, the nonterminal N uses the operation *mktable(top(tblptr))* to create a new symbol table. The argument *top(tblptr)* gives the enclosing scope for the new table.
- For each variable declaration **id:** T, an entry is created for **id** in the current symbol table. The top of stack offset is incremented by T.width.
- When the action on the right side of $D \rightarrow proc\ id;\ ND_1;\ S$ occurs, the width of all declarations generated by D_1 is on the top of stack offset; it is recorded using *addwidth.* Stacks *tblptr* and *offset* are then popped.
  At this point, the name of the enclosed procedure is entered into the symbol table of its enclosing procedure.

ASSIGNMENT STATEMENTS

Suppose that the context in which an assignment appears is given by the following grammar.

$P \rightarrow M\ D$

$M \rightarrow \varepsilon$

$D \rightarrow D\ ;\ D\ \mid$ **id** : T $\mid$ **proc id** ; N D ; S

$N \rightarrow \varepsilon$

Nonterminal P becomes the new start symbol when these productions are added to those in the translation scheme shown below.

Translation scheme to produce three-address code for assignments

```
S → id : = E        { p : = lookup ( id.name);
                      if  p ≠ nil then
                      emit( p ‘ : =’ E.place)
                      else error }

E → E1 + E2         { E.place : = newtemp;
                      emit( E.place ‘: =’ E1.place ‘ + ‘ E2.place ) }

E → E1 * E2         { E.place : = newtemp;
                      emit( E.place ‘: =’ E1.place ‘ * ‘ E2.place ) }

E → - E1            { E.place : = newtemp;
                      emit ( E.place ‘: =’  ‘uminus’ E1.place ) }

E → ( E1 )          { E.place : = E1.place }
```

```
E → id                { p : = lookup ( **id**.name);
                        if p ≠ nil then
                            E.place : = p
                        else error }
```

Reusing Temporary Names

- The temporaries used to hold intermediate values in expression calculations tend to clutter up the symbol table, and space has to be allocated to hold their values.
- Temporaries can be reused by changing *newtemp*. The code generated by the rules for $E \rightarrow E_1 + E_2$ has the general form:

  evaluate E_1 into t_1
  evaluate E_2 into t_2
  $t := t_1 + t_2$

- The lifetimes of these temporaries are nested like matching pairs of balanced parentheses.
- Keep a count c , initialized to zero. Whenever a temporary name is used as an operand, decrement c by 1. Whenever a new temporary name is generated, use $c and increase c by 1.
- For example, consider the assignment x := a * b + c * d – e * f

Three-address code with stack temporaries

statement	*value of c*
	0
$0 := a * b	1
$1 := c * d	2
$0 := $0 + $1	1
$1 := e * f	2
$0 := $0 - $1	1
x := $0	0

Addressing Array Elements:

Elements of an array can be accessed quickly if the elements are stored in a block of consecutive locations. If the width of each array element is *w*, then the *i*th element of array A begins in location

base* + (*i* – *low*)** x ***w

where *low* is the lower bound on the subscript and *base* is the relative address of the storage allocated for the array. That is, *base* is the relative address of A*[low]*.

The expression can be partially evaluated at compile time if it is rewritten as

$$i \text{ x } w + (base - low \text{ x } w)$$

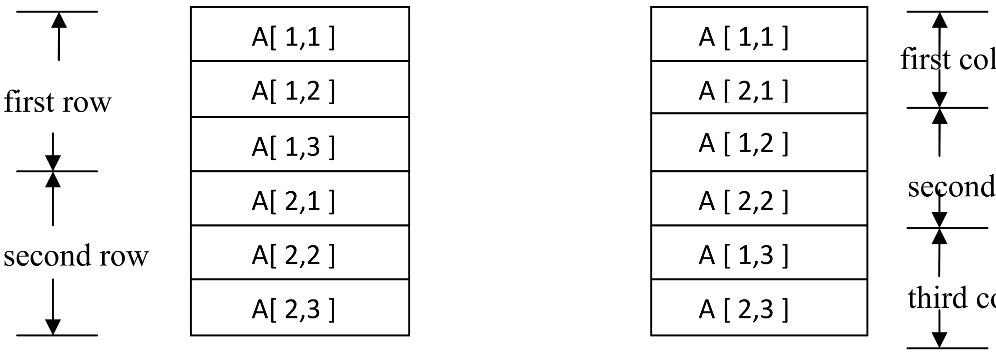
The subexpression $c = base - low$ x w can be evaluated when the declaration of the array is seen. We assume that c is saved in the symbol table entry for A , so the relative address of A[i] is obtained by simply adding i x w to c.

Address calculation of multi-dimensional arrays:

A two-dimensional array is stored in of the two forms :

- Row-major (row-by-row)
- Column-major (column-by-column)

Layouts for a 2 x 3 array



(a) ROW-MAJOR **(b) COLUMN-MAJOR**

In the case of row-major form, the relative address of A[i_1 , i_2] can be calculated by the formula

$$base + ((i_1 - low_1) \text{ x } n_2 + i_2 - low_2) \text{ x } w$$

where, low_1 and low_2 are the lower bounds on the values of i_1 and i_2 and n_2 is the number of values that i_2 can take. That is, if $high_2$ is the upper bound on the value of i_2, then $n_2 = high_2 - low_2 + 1$.

Assuming that i_1 and i_2 are the only values that are known at compile time, we can rewrite the above expression as

$$((i_1 \text{ x } n_2) + i_2) \text{ x } w + (base - ((low_1 \text{ x } n_2) + low_2) \text{ x } w)$$

Generalized formula:

The expression generalizes to the following expression for the relative address of A[$i_1,i_2,...,i_k$]

$$((\ldots ((i_1n_2 + i_2) n_3 + i_3) \ldots) n_k + i_k) \text{ x } w + base - ((\ldots((low_1n_2 + low_2)n_3 + low_3) \ldots) n_k + low_k) \text{ x } w$$

for all j, $n_j = high_j - low_j + 1$

The Translation Scheme for Addressing Array Elements :

Semantic actions will be added to the grammar :

(1) $S \rightarrow L := E$
(2) $E \rightarrow E + E$
(3) $E \rightarrow (E)$
(4) $E \rightarrow L$
(5) $L \rightarrow Elist$]
(6) $L \rightarrow$ **id**
(7) $Elist \rightarrow Elist , E$
(8) $Elist \rightarrow$ **id** [E

We generate a normal assignment if *L* is a simple name, and an indexed assignment into the location denoted by *L* otherwise :

```
(1)  S → L : = E          {  if L.offset = null then  /* L is a simple id  */
                               emit ( L.place ': =' E.place ) ;
                             else
                               emit ( L.place '[' L.offset ']'  ': =' E.place) }

(2)  E → $E_1$ + $E_2$            { E.place : = newtemp;
                            emit ( E.place ': =' $E_1$.place '+' $E_2$.place ) }

(3)  E → ( $E_1$ )             { E.place : = $E_1$.place }
```

When an array reference *L* is reduced to *E* , we want the *r*-value of *L*. Therefore we use indexing to obtain the contents of the location *L.place* [*L.offset*] :

```
(4)  E → L                { if L.offset = null  then   /* L is a simple id */
                                E.place : = L.place
                            else begin
                                E.place : = newtemp;
                                emit ( E.place  ': =' L.place '[' L.offset ']')
                            end }

(5)  L → Elist ]          {  L.place : = newtemp;
                             L.offset : = newtemp;
                             emit (L.place ': =' c( Elist.array ));
                             emit (L.offset ': =' Elist.place '*' width (Elist.array)) }

(6)  L → id                {  L.place := id.place;
                              L.offset := null  }

(7)  Elist → $Elist_1$ , E       {  t := newtemp;
                             m : = $Elist_1$.ndim + 1;
                             emit ( t ': ='  $Elist_1$.place '*' limit ($Elist_1$.array,m));
                             emit ( t ': =' t '+' E.place);
                             Elist.array : = $Elist_1$.array;
```

```
                              Elist.place : = t;
                              Elist.ndim : = m }

(8)  Elist → id [ E          { Elist.array : = id.place;

                              Elist.place : = E.place;
                              Elist.ndim : = 1 }
```

Type conversion within Assignments :

Consider the grammar for assignment statements as above, but suppose there are two types – real and integer , with integers converted to reals when necessary. We have another attribute *E.type*, whose value is either *real* or *integer*. The semantic rule for *E.type* associated with the production $E \rightarrow E + E$ is :

```
          E → E + E            { E.type : =
                                   if E_1.type = integer and
                                      E_2.type = integer then integer
                                   else real  }
```

The entire semantic rule for $E \rightarrow E + E$ and most of the other productions must be modified to generate, when necessary, three-address statements of the form x : = inttoreal y, whose effect is to convert integer y to a real of equal value, called x.

Semantic action for $E \rightarrow E_1 + E_2$

```
E.place := newtemp;
if E_1.type = integer and E_2.type = integer then begin
     emit( E.place ': =' E_1.place 'int +' E_2.place);
     E.type : = integer
end
else  if E_1.type = real and E_2.type = real then begin
       emit( E.place ': =' E_1.place 'real +' E_2.place);
       E.type : = real
end
else if E_1.type = integer and E_2.type = real then begin
      u : = newtemp;
      emit( u ': =' 'inttoreal' E_1.place);
      emit( E.place ': =' u ' real +' E_2.place);
      E.type : = real
end
else if E_1.type = real and E_2.type =integer then begin
      u : = newtemp;
      emit( u ': =' 'inttoreal' E_2.place);
      emit( E.place ': =' E_1.place ' real +' u);
      E.type : = real
end
else
      E.type : = type_error;
```

For example, for the input x : = y + i * j
assuming *x* and *y* have type *real*, and i and j have type *integer*, the output would look like

t_1 : = i int* j
t_3 : = inttoreal t_1
t_2 : = y real+ t_3
x : = t_2

BOOLEAN EXPRESSIONS

Boolean expressions have two primary purposes. They are used to compute logical values, but more often they are used as conditional expressions in statements that alter the flow of control, such as if-then-else, or while-do statements.

Boolean expressions are composed of the boolean operators (**and, or,** and **not**) applied to elements that are boolean variables or relational expressions. Relational expressions are of the form E_1 **relop** E_2, where E_1 and E_2 are arithmetic expressions.

Here we consider boolean expressions generated by the following grammar :

E → E **or** E | E **and** E | **not** E | (E) | **id relop id | true | false**

Methods of Translating Boolean Expressions:

There are two principal methods of representing the value of a boolean expression. They are :

- To encode true and false ***numerically*** and to evaluate a boolean expression analogously to an arithmetic expression. Often, 1 is used to denote true and 0 to denote false.
- To implement boolean expressions by ***flow of control***, that is, representing the value of a boolean expression by a position reached in a program. This method is particularly convenient in implementing the boolean expressions in flow-of-control statements, such as the if-then and while-do statements.

Numerical Representation

Here, 1 denotes true and 0 denotes false. Expressions will be evaluated completely from left to right, in a manner similar to arithmetic expressions.

For example :

- The translation for
  a **or** b **and** **not** c
  is the three-address sequence
  t_1 : = **not** c
  t_2 : = b **and** t_1
  t_3 : = a **or** t_2
- A relational expression such as a < b is equivalent to the conditional statement
  if a < b then 1 else 0

which can be translated into the three-address code sequence (again, we arbitrarily start statement numbers at 100) :

```
100 :   if a < b goto 103
101 :   t : = 0
102 :   goto 104
103 :   t : = 1
104 :
```

Translation scheme using a numerical representation for booleans

$E \rightarrow E_1$ **or** E_2	{ *E.place : = newtemp;* *emit(E.place ': =' E_1.place* '**or**' *E_2.place) }*
$E \rightarrow E_1$ **and** E_2	{ *E.place : = newtemp;* *emit(E.place ': =' E_1.place* '**and**' *E_2.place) }*
$E \rightarrow$ **not** E_1	{ *E.place : = newtemp;* *emit(E.place ': ='* '**not**' *E_1.place) }*
$E \rightarrow (E_1)$	{ *E.place : = E_1.place* }
$E \rightarrow$ **id$_1$ relop id$_2$**	{ *E.place : = newtemp;* *emit('if'* **id$_1$**.*place* **relop**.*op* **id$_2$**.*place* '**goto**' *nextstat* + **3**); *emit(E.place ': ='* '**0**'); *emit(*'**goto**' *nextstat* +**2**); *emit(E.place ': ='* '**1**') }
$E \rightarrow$ **true**	{ *E.place : = newtemp;* *emit(E.place ': ='* '**1**') }
$E \rightarrow$ **false**	{ *E.place : = newtemp;* *emit(E.place ': ='* '**0**') }

Short-Circuit Code:

We can also translate a boolean expression into three-address code without generating code for any of the boolean operators and without having the code necessarily evaluate the entire expression. This style of evaluation is sometimes called "**short-circuit**" or "**jumping**" code. It is possible to evaluate boolean expressions without generating code for the boolean operators **and, or,** and **not** if we represent the value of an expression by a position in the code sequence.

Translation of a < b or c < d and e < f

100 : if a < b goto 103	107 : t_2 : = 1
101 : t_1 : = 0	108 : if e < f goto 111
102 : goto 104	109 : t_3 : = 0
103 : t_1 : = 1	110 : goto 112
104 : if c < d goto 107	111 : t_3 : = 1
105 : t_2 : = 0	112 : t_4 : = t_2 and t_3
106 : goto 108	113 : t_5 : = t_1 or t_4

Flow-of-Control Statements

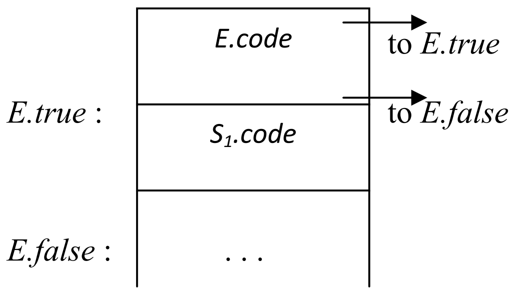
We now consider the translation of boolean expressions into three-address code in the context of if-then, if-then-else, and while-do statements such as those generated by the following grammar:

S → **if** E **then** S_1
| **if** E **then** S_1 **else** S_2
| **while** E **do** S_1

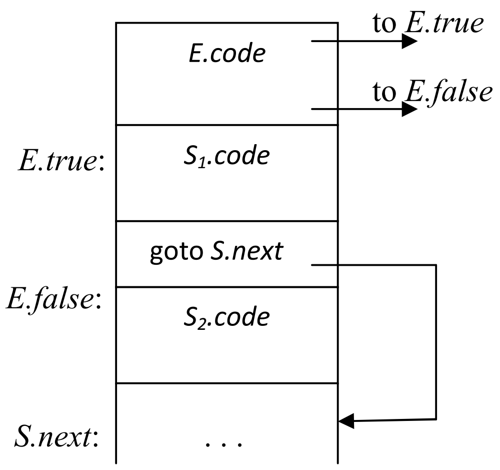
In each of these productions, *E* is the Boolean expression to be translated. In the translation, we assume that a three-address statement can be symbolically labeled, and that the function *newlabel* returns a new symbolic label each time it is called.

- E.true is the label to which control flows if E is true, and E.false is the label to which control flows if E is false.
- The semantic rules for translating a flow-of-control statement S allow control to flow from the translation S.code to the three-address instruction immediately following S.code.
- S.next is a label that is attached to the first three-address instruction to be executed after the code for S.

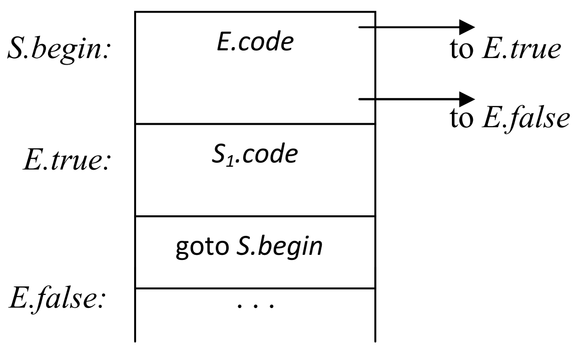
Code for if-then , if-then-else, and while-do statements



(a) if-then



(b) if-then-else



(c) while-do

Syntax-directed definition for flow-of-control statements

PRODUCTION	SEMANTIC RULES
$S \rightarrow$ **if** E **then** S_1	$E.true := newlabel;$ $E.false := S.next;$ $S_1.next := S.next;$ $S.code := E.code \mid\mid gen(E.true\ `:') \mid\mid S_1.code$
$S \rightarrow$ **if** E **then** S_1 **else** S_2	$E.true := newlabel;$ $E.false := newlabel;$ $S_1.next := S.next;$ $S_2.next := S.next;$ $S.code := E.code \mid\mid gen(E.true\ `:') \mid\mid S_1.code \mid\mid$ $gen(`$**goto**$'\ S.next) \mid\mid$ $gen(E.false\ `:') \mid\mid S_2.code$
$S \rightarrow$ **while** E **do** S_1	$S.begin := newlabel;$ $E.true := newlabel;$ $E.false := S.next;$ $S_1.next := S.begin;$ $S.code := gen(S.begin\ `:') \mid\mid E.code \mid\mid$ $gen(E.true\ `:') \mid\mid S_1.code \mid\mid$ $gen(`$**goto**$'\ S.begin)$

Control-Flow Translation of Boolean Expressions:

Syntax-directed definition to produce three-address code for booleans

PRODUCTION	SEMANTIC RULES
$E \rightarrow E_1$ **or** E_2	$E_1.true := E.true;$ $E_1.false := newlabel;$ $E_2.true := E.true;$ $E_2.false := E.false;$ $E.code := E_1.code \mid\mid gen(E_1.false\ `:') \mid\mid E_2.code$
$E \rightarrow E_1$ **and** E_2	$E.true := newlabel;$ $E_1.false := E.false;$ $E_2.true := E.true;$ $E_2.false := E.false;$ $E.code := E_1.code \mid\mid gen(E_1.true\ `:') \mid\mid E_2.code$
$E \rightarrow$ **not** E_1	$E_1.true := E.false;$ $E_1.false := E.true;$ $E.code := E_1.code$
$E \rightarrow (E1)$	$E_1.true := E.true;$

	E_1.false : = E.false; *E.code : = E_1.code*
E → **id$_1$ relop id$_2$**	*E.code : = gen*('**if**' **id$_1$**.*place* **relop.***op* **id$_2$**.*place* '**goto**' *E.true)* || *gen*('**goto**' *E.false*)
E → **true**	*E.code : = gen(*'**goto**' *E.true)*
E → **false**	*E.code : = gen(*'**goto**' *E.false)*

CASE STATEMENTS

The "switch" or "case" statement is available in a variety of languages. The switch-statement syntax is as shown below :

Switch-statement syntax

```
switch expression
      begin
          case value :   statement
          case value :   statement
                . . .
          case value :   statement
          default :      statement
      end
```

There is a selector expression, which is to be evaluated, followed by *n* constant values that the expression might take, including a default "value" which always matches the expression if no other value does. The intended translation of a switch is code to:

1. Evaluate the expression.
2. Find which value in the list of cases is the same as the value of the expression.
3. Execute the statement associated with the value found.

Step (2) can be implemented in one of several ways :

- By a sequence of conditional **goto** statements, if the number of cases is small.
- By creating a table of pairs, with each pair consisting of a value and a label for the code of the corresponding statement. Compiler generates a loop to compare the value of the expression with each value in the table. If no match is found, the default (last) entry is sure to match.
- If the number of cases s large, it is efficient to construct a hash table.
- There is a common special case in which an efficient implementation of the n-way branch exists. If the values all lie in some small range, say i_{min} to i_{max}, and the number of different values is a reasonable fraction of i_{max} - i_{min}, then we can construct an array of labels, with the label of the statement for value j in the entry of the table with offset j - i_{min} and the label for the default in entries not filled otherwise. To perform switch,

evaluate the expression to obtain the value of j , check the value is within range and transfer to the table entry at offset $j\text{-}i_{min}$.

Syntax-Directed Translation of Case Statements:

Consider the following switch statement:

```
switch E
    begin
        case V1 :     S1
        case V2 :     S2
                ...
        case Vn-1 :   Sn-1
        default :     Sn
    end
```

This case statement is translated into intermediate code that has the following form :

Translation of a case statement

```
                code to evaluate E into t
                goto test
L1 :            code for S1
                goto next
L2 :            code for S2
                goto next
                  . . .
Ln-1 :          code for Sn-1
                goto next
Ln :            code for Sn
                goto next
test :           if  t = V1 goto L1
                 if  t = V2 goto L2
                   . . .
                 if  t = Vn-1  goto Ln-1
                 goto Ln
next :
```

To translate into above form :

- When keyword **switch** is seen, two new labels **test** and **next,** and a new temporary **t** are generated.

- As expression E is parsed, the code to evaluate E into **t** is generated. After processing E , the jump **goto test** is generated.

- As each **case** keyword occurs, a new label L_i is created and entered into the symbol table. A pointer to this symbol-table entry and the value V_i of case constant are placed on a stack (used only to store cases).

- Each statement **case** $V_i : S_i$ is processed by emitting the newly created label L_i, followed by the code for S_i , followed by the jump **goto next**.
- Then when the keyword **end** terminating the body of the switch is found, the code can be generated for the n-way branch. Reading the pointer-value pairs on the case stack from the bottom to the top, we can generate a sequence of three-address statements of the form

```
case  V1  L1
case  V2  L2
      . . .
case  Vn-1 Ln-1
case  t  Ln
label  next
```

where t is the name holding the value of the selector expression *E*, and L_n is the label for the default statement.

BACKPATCHING

The easiest way to implement the syntax-directed definitions for boolean expressions is to use two passes. First, construct a syntax tree for the input, and then walk the tree in depth-first order, computing the translations. The main problem with generating code for boolean expressions and flow-of-control statements in a single pass is that during one single pass we may not know the labels that control must go to at the time the jump statements are generated. Hence, a series of branching statements with the targets of the jumps left unspecified is generated. Each statement will be put on a list of goto statements whose labels will be filled in when the proper label can be determined. We call this subsequent filling in of labels ***backpatching***.

To manipulate lists of labels, we use three functions :

1. *makelist*(*i*) creates a new list containing only *i*, an index into the array of quadruples; *makelist* returns a pointer to the list it has made.
2. *merge*(p_1,p_2) concatenates the lists pointed to by p_1 and p_2, and returns a pointer to the concatenated list.
3. *backpatch(p,i)* inserts i as the target label for each of the statements on the list pointed to by *p.*

Boolean Expressions:

We now construct a translation scheme suitable for producing quadruples for boolean expressions during bottom-up parsing. The grammar we use is the following:

(1) $E \rightarrow E_1$ **or** $M\ E_2$
(2) | E_1 **and** $M\ E_2$
(3) | **not** E_1
(4) | (E_1)
(5) | $\mathbf{id_1}$ **relop** $\mathbf{id_2}$
(6) | **true**
(7) | **false**
(8) $M \rightarrow \varepsilon$

Synthesized attributes *truelist* and *falselist* of nonterminal E are used to generate jumping code for boolean expressions. Incomplete jumps with unfilled labels are placed on lists pointed to by *E.truelist* and *E.falselist*.

Consider production $E \rightarrow E_1$ **and** $M\ E_2$. If E_1 is false, then E is also false, so the statements on E_1*.falselist* become part of *E.falselist*. If E_1 is true, then we must next test E_2, so the target for the statements E_1*.truelist* must be the beginning of the code generated for E_2. This target is obtained using marker nonterminal M.

Attribute *M.quad* records the number of the first statement of E_2*.code*. With the production $M \rightarrow \varepsilon$ we associate the semantic action

{ *M.quad* : = *nextquad* }

The variable *nextquad* holds the index of the next quadruple to follow. This value will be backpatched onto the E_1*.truelist* when we have seen the remainder of the production $E \rightarrow E_1$ **and** $M\ E_2$. The translation scheme is as follows:

(1) $E \rightarrow E_1$ **or** $M\ E_2$	{ *backpatch* (E_1*.falselist*, *M.quad*);
	E.truelist : = *merge*(E_1*.truelist*, E_2*.truelist*);
	E.falselist : = E_2*.falselist* }

(2) $E \rightarrow E_1$ **and** $M\ E_2$	{ *backpatch* (E_1*.truelist*, *M.quad*);
	E.truelist : = E_2*.truelist*;
	E.falselist : = *merge*(E_1*.falselist*, E_2*.falselist*) }

(3) $E \rightarrow$ **not** E_1	{ *E.truelist* : = E_1*.falselist*; *E.falselist* : = E_1*.truelist*; }
(4) $E \rightarrow (E_1)$	{ *E.truelist* : = E_1*.truelist*; *E.falselist* : = E_1*.falselist*; }

(5) $E \rightarrow$ **id₁ relop id₂**	{ *E.truelist* : = *makelist* (*nextquad*);
	E.falselist : = *makelist*(*nextquad* + *1*);
	emit('if' **id**$_1$*.place* **relop**.*op* **id**$_2$.place '**goto_**')
	emit('**goto_**') }

(6) $E \rightarrow$ **true**	{ *E.truelist* : = *makelist*(*nextquad*);
	emit('**goto_**') }

(7) $E \rightarrow$ **false**	{ *E.falselist* : = *makelist*(*nextquad*);
	emit('**goto_**') }

(8) $M \rightarrow \varepsilon$	{ *M.quad* : = *nextquad* }

Flow-of-Control Statements:

A translation scheme is developed for statements generated by the following grammar :

(1) $S \rightarrow$ **if** E **then** S
(2) $\quad |$ **if** E **then** S **else** S
(3) $\quad |$ **while** E **do** S
(4) $\quad |$ **begin** L **end**
(5) $\quad |$ A
(6) $L \rightarrow L \, ; S$
(7) $\quad |$ S

Here S denotes a statement, L a statement list, A an assignment statement, and E a boolean expression. We make the tacit assumption that the code that follows a given statement in execution also follows it physically in the quadruple array. Else, an explicit jump must be provided.

Scheme to implement the Translation:

The nonterminal E has two attributes *E.truelist* and *E.falselist*. *L* and *S* also need a list of unfilled quadruples that must eventually be completed by backpatching. These lists are pointed to by the attributes *L..nextlist* and *S.nextlist*. *S.nextlist* is a pointer to a list of all conditional and unconditional jumps to the quadruple following the statement S in execution order, and *L.nextlist* is defined similarly.

The semantic rules for the revised grammar are as follows:

(1) $S \rightarrow$ **if** E **then** $M_1\, S_1\, N$ **else** $M_2\, S_2$

```
        { backpatch (E.truelist, M1.quad);
          backpatch (E.falselist, M2.quad);
          S.nextlist : = merge (S1.nextlist,  merge (N.nextlist, S2.nextlist)) }
```

We backpatch the jumps when E is true to the quadruple $M_1.quad$, which is the beginning of the code for S_1. Similarly, we backpatch jumps when E is false to go to the beginning of the code for S_2. The list *S.nextlist* includes all jumps out of S_1 and S_2, as well as the jump generated by *N*.

(2)	$N \rightarrow \varepsilon$	{ *N.nextlist* : = *makelist*(*nextquad*); *emit*('**goto** _') }
(3)	$M \rightarrow \varepsilon$	{ *M.quad* : = *nextquad* }
(4)	$S \rightarrow$ **if** E **then** $M\, S_1$	{ *backpatch*(*E.truelist*, *M.quad*); *S.nextlist* : = *merge*(*E.falselist*, S_1*.nextlist*) }
(5)	$S \rightarrow$ **while** $M_1\, E$ **do** $M_2\, S_1$	{ *backpatch*(S_1*.nextlist*, M_1*.quad*); *backpatch*(*E.truelist*, M_2*.quad*); *S.nextlist* : = *E.falselist* *emit*('**goto**' M_1*.quad*) }
(6)	$S \rightarrow$ **begin** L **end**	{ *S.nextlist* : = *L.nextlist* }

(7) $S \rightarrow A$ { *S.nextlist* : = **nil** }

The assignment *S.nextlist* : = **nil** initializes *S.nextlist* to an empty list.

(8) $L \rightarrow L1$; $M\ S$ { *backpatch*(L_1*.nextlist*, *M.quad*);
L.nextlist : = *S.nextlist* }

The statement following L_1 in order of execution is the beginning of *S*. Thus the *L1.nextlist* list is backpatched to the beginning of the code for *S*, which is given by *M.quad*.

(9) $L \rightarrow S$ { *L.nextlist* : = *S.nextlist* }

PROCEDURE CALLS

The procedure is such an important and frequently used programming construct that it is imperative for a compiler to generate good code for procedure calls and returns. The run-time routines that handle procedure argument passing, calls and returns are part of the run-time support package.

Let us consider a grammar for a simple procedure call statement

(1) $S \rightarrow$ **call id** (*Elist*)
(2) *Elist* $\rightarrow$ *Elist* , *E*
(3) *Elist* $\rightarrow$ *E*

Calling Sequences:

The translation for a call includes a calling sequence, a sequence of actions taken on entry to and exit from each procedure. The falling are the actions that take place in a calling sequence :

- When a procedure call occurs, space must be allocated for the activation record of the called procedure.
- The arguments of the called procedure must be evaluated and made available to the called procedure in a known place.
- Environment pointers must be established to enable the called procedure to access data in enclosing blocks.
- The state of the calling procedure must be saved so it can resume execution after the call.
- Also saved in a known place is the return address, the location to which the called routine must transfer after it is finished.
- Finally a jump to the beginning of the code for the called procedure must be generated.

For example, consider the following syntax-directed translation

```
(1) S → call id ( Elist )
            {  for each item p on queue do
                    emit (' param' p );
```

emit (‘call*’* **id.***place*) }

(2) *Elist* → *Elist* , *E*

{ append *E.place* to the end of *queue* }

(3) *Elist* → *E*

{ initialize *queue* to contain only *E.place* }

- Here, the code for S is the code for *Elist*, which evaluates the arguments, followed by a **param** *p* statement for each argument, followed by a **call** statement.

- *queue* is emptied and then gets a single pointer to the symbol table location for the name that denotes the value of E.

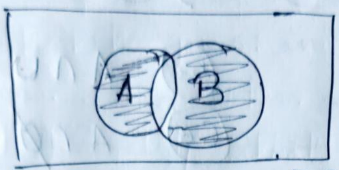
Symmetric Differences :

The symmetric difference of sets A and B, denoted by $A \oplus B$, consists of those elements which belong to A or B but not both.

$$A \oplus B = (A \cup B) \setminus (A \cap B)$$

or

$$A \oplus B = (A \setminus B) \cup (B \setminus A) .$$



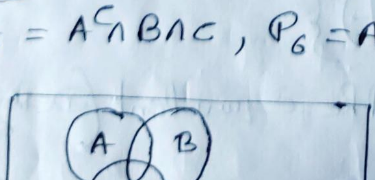
$A \oplus B$ is shaded.

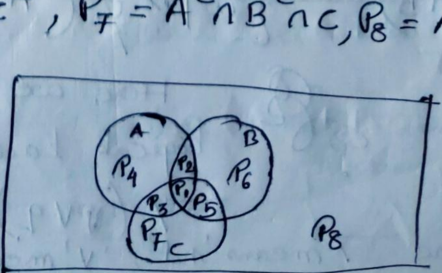
Fundamental Products :

Consider n distinct sets $A_1, A_2, \ldots A_n$.
A fundamental product of the sets is a set of the form
$A_1^* \cap A_2^* \cap A_3^* \cap \ldots \ldots \cap A_n^*$ where $A_i^* = A$ or $A_i^* = A^c$.
We note that :
i) There are $m = 2^n$ such fundamental products.
ii) Any two such fundamental products are disjoint.
iii) The universal set U is the union of all fundamental products.

Eg: There are 3 sets A, B, C. The following lists the $m = 2^3 = 8$ fundamental products of the set A, B, C :

$P_1 = A \cap B \cap C$, $P_2 = A \cap B \cap C^c$, $P_3 = A \cap B^c \cap C$, $P_4 = A \cap B^c \cap C^c$
$P_5 = A^c \cap B \cap C$, $P_6 = A^c \cap B \cap C^c$, $P_7 = A^c \cap B^c \cap C$, $P_8 = A^c \cap B^c \cap C^c$





Algebra of Sets.

Laws of the algebra of sets.

Idempotent laws : (1a) $A \cup A = A$ (1b) $A \cap A = A$

Associative " : (2a) $(A \cup B) \cup C = A \cup (B \cup C)$ (2b) $(A \cap B) \cap C = A \cap (B \cap C)$

Commutative : (3a) $A \cup B = B \cup A$ (3b) $A \cap B = B \cap A$.

Distributive : (4a) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ (4b) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

Identity : (5a) $A \cup \emptyset = A$ (5b) $A \cap U = A$
(6a) $A \cup U = U$ (6b) $A \cap \emptyset = \emptyset$.

Involution : (7) $(A^C)^C = A$

Complement : (8a) $A \cup A^C = U$ (8b) $A \cap A^C = \emptyset$
(9a) $U^C = \emptyset$ (9b) $\emptyset^C = U$.

DeMorgan's law : (10a) $(A \cup B)^C = A^C \cap B^C$ (10b) $(A \cap B)^C = A^C \cup B^C$.

Proof of DeMorgan's law :

$$(A \cup B)^C = \{x \mid x \notin (A \text{ or } B)\}$$
$$= \{x \mid x \notin A \text{ and } x \notin B\}$$
$$= A^C \cap B^C.$$

~~$(A \cap B)^C = \{x \mid x \notin (A \text{ and } B)\}$~~

Here we use the equivalent (DeMorgan's) logical law :

$$\neg (p \vee q) = \neg p \wedge \neg q$$

where $\neg$ means 'not', '$\vee$' means 'or' and '$\wedge$' means 'and'.

Duality :

Suppose, E is an equation of set algebra. The dual E^* of E is the equation obtained by replacing each occurrence of $\cup, \cap, \mathbb{U}$ & $\emptyset$ in E by $\cap, \cup, \emptyset$ and $\mathbb{U}$ respectively.

eg : $(\mathbb{U} \cap A) \cup (B \cap A) = A$

The dual of A is

$$A^* = (\emptyset \cup A) \cap (B \cup A)$$

Observe ~~that the~~ pairs in laws in table are dual of each other. ~~The~~. It is a fact of set algebra, called the Principle of duality, that if any equation E is an identity then its dual E^* is also an identity.

Qn

(1) Let $\mathbb{U} = \{1, 2, 3, \ldots\ldots 9\}$ be the universal set

$A = \{1, 2, 3, 4, 5\}$, $B = \{4, 5, 6, 7\}$, $C = \{5, 6, 7, 8, 9\}$,

$D = \{1, 3, 5, 7, 9\}$, $E = \{2, 4, 6, 8\}$, $F = \{1, 5, 9\}$.

Find i) $A \cup B$, ii) $A \cap B$, iii) $A \cup C$, iv) $A \cap C$, v) $D \cup F$, vi) $D \cap F$

vii) A^C, B^C, C^C, D^C, E^C

viii) $A \setminus B$, $B \setminus A$, $D \setminus E$.

ix) $A \oplus B$, $C \oplus D$, $E \oplus F$.

(2) ~~Prove $B \setminus A = B \cap A^c$~~

In a survey of 120 people, it was found that

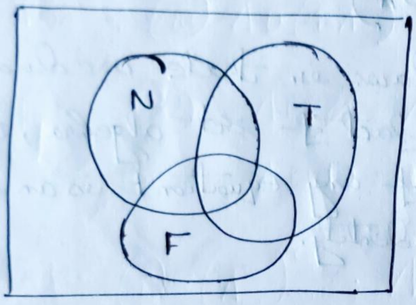
65 read Newsweek Magazine.	20 read both Newsweek & Time
45 " Times.	25 read both Newsweek & Fortune.
42 " Fortune.	15 read both Time & Fortune.

8 reads all three magazines.

a) Find the no. of people who read at least one of the three magazines

b) Fill the correct no. of people in ~~[illegible]~~ each of the eight regions of venn diagram where N, T & F denotes the set of people who read Newsweek, Time & Fortune respectively.

c) Find the no. of people who read exactly one magazine.



ⓐ $n(N \cup T \cup F) = n(N) + n(T) + n(F) - n(N \cap T)$

$- n(N \cap F) - n(T \cap F) + n(N \cap T \cap F)$

$= 65 + 45 + 42 - 20 - 25 - 15 + 8 = 100$.

ⓑ 8 read all three magazines.

20-8=12 read Newsweek and Times but not all three magazin..

25-8 = 17 " Newsweek & Fortune " " "

15-8 = 7 " Times & Fortune " " "

65 - 12 - 8 - 17 = 28 read only Newsweek.

45 - 12 - 8 - 7 = 18 read only Times.

42 - 17 - 8 - 7 = 10 read only Fortune.

120 - 100 = 20 read no magazine at all.

c) (28 + 18 + 0) = 56 read exactly one of the magazines.

Notes: Relational Algebra

Relational Query Languages

Relational query languages use relational algebra to break the user requests and instruct the DBMS to execute the requests. It is the language by which user communicates with the database. These relational query languages can be procedural or non-procedural.

Procedural Query Language

A procedural query language will have set of queries instructing the DBMS to perform various transactions in the sequence to meet the user request.
For example, *get_CGPA* procedure will have various queries to get the marks of student in each subject, calculate the total marks, and then decide the CGPA based on his total marks. This procedural query language tells the database what is required from the database and how to get them from the database. Relational algebra is a procedural query language.

Non-Procedural Query Language

Non-procedural queries will have single query on one or more tables to get result from the database. For example, get the name and address of the student with particular ID will have single query on STUDENT table. Relational Calculus is a non-procedural language which informs what to do with the tables, but doesn't inform how to accomplish this.
These query languages basically will have queries on tables in the database. In the relational database, a table is known as relation. Records / rows of the table are referred as tuples. Columns of the table are also known as attributes. All these names are used interchangeably in relational database.

Relational Algebra

Relational algebra is a procedural query language. It takes one or more relations / tables and performs the operation and produce the result. This result is also considered as a new table or relation. Suppose we have to retrieve student name, address and class for the given ID. What a relational algebra will do in this case is, it filters the name, address and class from the STUDENT table for the input ID. In mathematical terms, relational algebra has produced a subset of STUDENT table for the given ID.

Relational algebra will have operators to indicate the operations. This algebra can be applied on single relation – called **unary** or can be applied on two tables – called **binary**. While applying the operations on the relation, the resulting subset of relation is also known as new relation. There can be multiple steps involved in some of the operations. The subsets of relations at the intermediary level are also known as relation. We will understand it better when we see different operations below.
Relational Algebra in DBMS has 6 fundamental operations. There are several other operations defined upon these fundamental operations.

Select (σ) – This is a unary relational operation. This operation pulls the horizontal subset (subset of rows) of the relation that satisfies the conditions. This can use operators like <, >, <=, >=, = and != to filter the data from the relation. It can also use logical AND, OR and NOT operators to combine the various filtering conditions. This operation can be represented as below:

$$\sigma_{p}(r)$$

Where σ is the symbol for select operation, r represents the relation/table, and p is the logical formula or the filtering conditions to get the subset. Let us see an example as below:

$$\sigma_{STD\_NAME = \text{“James”}}(STUDENT)$$

What does above relation algebra do? It selects the record/tuple from the STUDENT table with Student name as ‘James’

$$\sigma_{dept\_id = 20 \text{ AND } salary>=10000}(EMPLOYEE)$$

– Selects the records from EMPLOYEE table with department ID = 20 and employees whose salary is more than 10000.

Project (∏) – This is a unary operator and is similar to select operation above. It creates the subset of relation based on the conditions specified. Here, it selects only selected columns/attributes from the relation- vertical subset of relation. The select operation above creates subset of relation but for all the attributes in the relation. It is denoted as below:

$$\prod_{a1, a2, a3}(r)$$

Where ∏ is the operator for projection, r is the relation and a1, a2, a3 are the attributes of the relations which will be shown in the resultant subset.

$$\prod_{std\_name, address, course}(STUDENT) \text{ –}$$

This will select all the records from STUDENT table but only selected columns – std_name, address and course. Suppose we have to select only these 3 columns for particular student then we have to combine both project and select operations.

$$\prod_{STD\_ID, address, course}(\sigma_{STD\_NAME = \text{“James”}}(STUDENT))$$

– this selects the record for ‘James’ and displays only std_ID, address and his course columns. Here we can see two unary operators are combined, and it has two operations performing. First it selects the tuple from STUDENT table for ‘James’. The resultant subset of STUDENT is also considered as intermediary relation. But it is temporary and exists till the end of this operation. It then filters the 3 columns from this temporary relation.

Rename (ρ) – This is a unary operator used to rename the tables and columns of a relation. When we perform self join operation, we have to differentiate two same tables. In such case rename operator on tables comes into picture. When we join two or more tables and if those tables have same column names, then it is always better to rename the columns to differentiate them. This occurs when we perform Cartesian product operation.

$$\rho_{R}(E)$$

Where ρ is the rename operator, E is the existing relation name, and R is the new relation name.

$\rho_{STUDENT}(STD\_TABLE)$ – Renames STD_TABLE table to STUDENT

Programming in JAVA

Lecture on

“JAVA Inheritance”

Prof. Mithun Roy

Inheritance in Java

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviours of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a parent-child relationship.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

The syntax of Java Inheritance

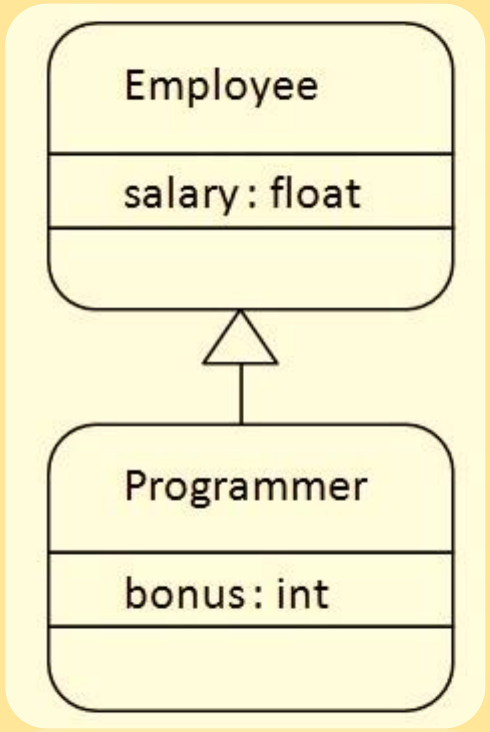
```
1.class Subclass-name extends Superclass-name
2.{
3.   //methods and fields
4.}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

Java Inheritance Example

As displayed in the above figure, Programmer is the subclass and Employee is the superclass. The relationship between the two classes is **Programmer IS-A Employee**. It means that Programmer is a type of Employee.

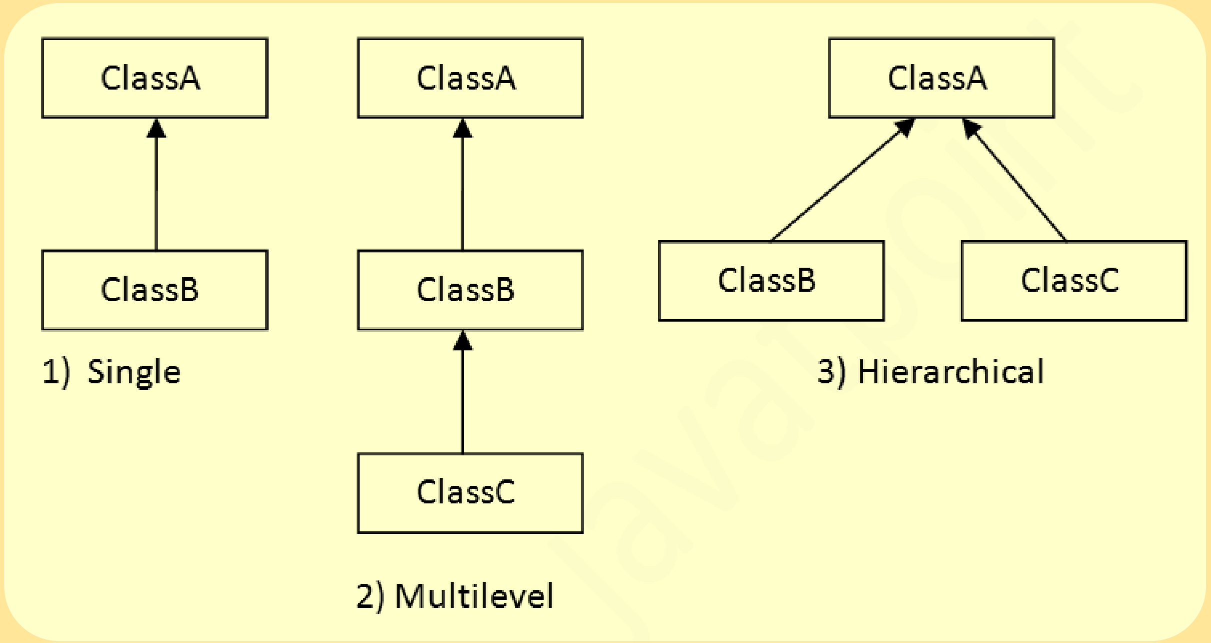


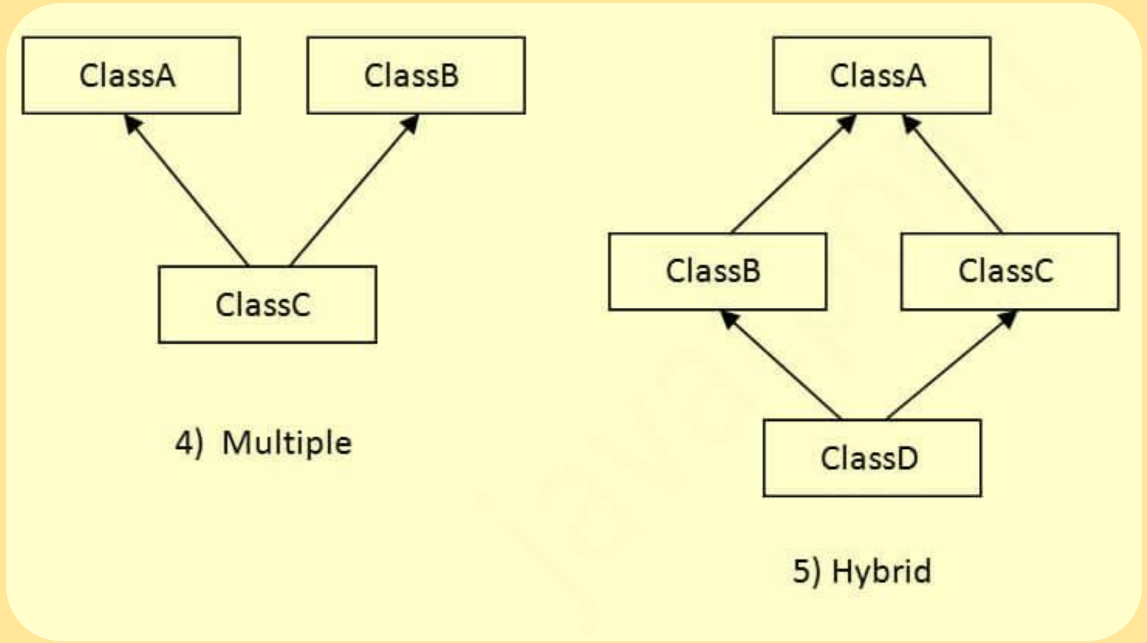
```java
class Employee{
float salary=40000;
}
class Programmer extends Employee{
int bonus=10000;
public static void main(String args[]){
  Programmer p=new Programmer();
  System.out.println("Programmer salary is:"+p.salary);
  System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.





Single Inheritance Example

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

Output:

```
barking...
eating...
```

Multilevel Inheritance Example

When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

```java
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:

```
weeping...
barking...
eating...
```

Hierarchical Inheritance Example

When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:

```
meowing...
eating...
```

Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java. Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.
Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

```java
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 public static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}
}
```

Super Keyword in Java

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

1) super is used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

```java
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
```

In the above example, Animal and Dog both classes have a common property color. If we print color property, it will print the color of current class by default. To access the parent property, we need to use super keyword.

2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat();
bark();
}
}
class TestSuper2{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}
```

In the above example Animal and Dog both classes have eat() method if we call eat() method from Dog class, it will call the eat() method of Dog class by default because priority is given to local.
To call the parent class method, we need to use super keyword.

3) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}}
```

As we know well that default constructor is provided by compiler automatically if there is no constructor. But, it also adds super() as the first statement.

Another example of super keyword where super() is provided by the compiler implicitly.

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
System.out.println("dog is created");
}
}
class TestSuper4{
public static void main(String args[]){
Dog d=new Dog();
}}
```

Aggregation in Java

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Consider a situation, Employee object contains many information such as id, name, emailId etc. It contains one more object named address, which contains its own information such as city, state, country, zipcode etc. as given below.

```
class Employee{
int id;
String name;
Address address;//Address is a class
...
}
```

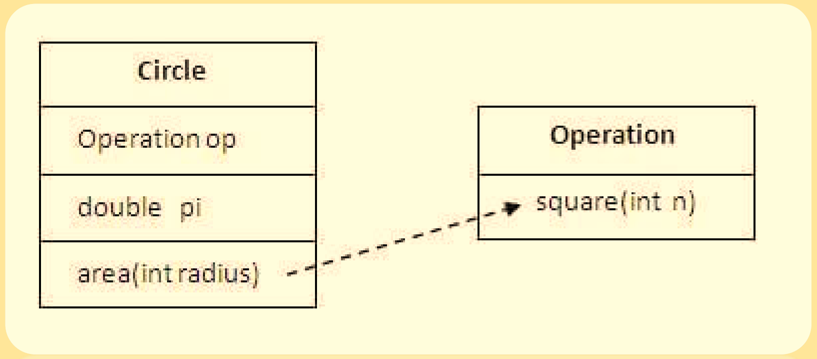
In such case, Employee has an entity reference address, so relationship is Employee HAS-A address.

Why use Aggregation?

For Code Reusability.

Simple Example of Aggregation

In this example, we have created the reference of Operation class in the Circle class.



```
class Operation{
int square(int n){
 return n*n;
}
}
class Circle{
Operation op;//aggregation
double pi=3.14;

double area(int radius){
  op=new Operation();
  int rsquare=op.square(radius);//code reusability (i.e. delegates the method call).
  return pi*rsquare;
}

public static void main(String args[]){
  Circle c=new Circle();
  double result=c.area(5);
  System.out.println(result);
}
}
```

When use Aggregation?

Code reuse is also best achieved by aggregation **when there is no is-a relationship**.

Inheritance should be used **only if the relationship is-a** is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

Understanding meaningful example of Aggregation

In this example, Employee has an object of Address, address object contains its own information such as city, state, country etc. In such case relationship is Employee HAS-A address.

Thank You

Lecture 21: Single Source Shortest Paths - Bellman-Ford Algorithm

CS 360 → Lecture Notes → Lecture 21

MST solves the problem of finding a minimum total weight subset of edges that spans all the vertices. Another common graph problem is to find the shortest paths to all reachable vertices from a given source. We have already seen how to solve this problem in the case where all the edges have the *same* weight (in which case the shortest path is simply the minimum *number* of edges) using BFS. Now we will examine two algorithms for finding *single source shortest paths* for directed graphs when the edges have *different* weights - Bellman-Ford and Dijkstra's algorithms. Several related problems are:

- Single destination shortest path - find the transpose graph (i.e. reverse the edge directions) and use single source shortest path
- Single pair shortest path (i.e. a specific destination) - asymptotically this problem can be solved no faster than simply using single source shortest path algorithms to all the vertices
- All pair shortest paths - one technique is to use single source shortest path for each vertex, but later we will see a more efficient algorithm

Single Source Shortest Path

Problem

Given a directed graph *G(V,E)* with *weighted edges w(u,v)*, define the *path weight* of a path *p* as

$$w(p)=\sum_{i=1}^{k} w(v_{i-1},v_i)$$

For a given source vertex *s*, find the *minimum weight paths* to every vertex reachable from *s* denoted

$$\delta(s,v)=\begin{cases}\min\{w(p) \quad s\rightarrow v\} \\ \infty \text{ otherwise}\end{cases}$$

The final solution will satisfy certain caveats:

- The graph cannot contain any *negative weight cycles* (otherwise there would be no minimum path since we could simply continue to follow the negative weight cycle producing a path weight of -∞).
- The solution cannot have any *positive weight cycles* (since the cycle could simply be removed giving a lower weight path).
- The solution can be assumed to have no zero weight cycles (since they would not affect the minimum value).

Therefore given these caveats, we know the shortest paths must be *acyclic* (with ≤ |*V*| distinct vertices) ⇒ ≤ |*V*| - 1 edges in each path.

Generic Algorithm

The single source shortest path algorithms use the same notation as BFS (see lecture 17) with predecessor π and distance *d* fields for each vertex. The optimal solution will have $v.d = \delta(s,v)$ for all $v \in V$.

The solutions utilize the concept of *edge relaxation* which is a test to determine whether going through edge (*u*,*v*) reduces the distance to *v* and if so update *v.π* and *v.d*. This is accomplished using the condition

$$\text{if } v.d > u.d + w(u,v)$$

$$v.d = u.d + w(u,v)$$

$$v.\pi = u$$

Bellman-Ford Algorithm

The *Bellman-Ford algorithm* uses relaxation to find single source shortest paths on directed graphs that may contain *negative weight edges*. The algorithm will also detect if there are any *negative weight cycles* (such that there is no solution).

```
BELLMAN-FORD(G,w,s)
1.  INITIALIZE-SINGLE-SOURCE(G,s)
2.  for i = 1 to |G.V|-1
3.     for each edge (u,v) ∈ G.E
4.        RELAX(u,v,w)
5.  for each edge (u,v) ∈ G.E
6.     if v.d > u.d + w(u,v)
7.        return FALSE
8.  return TRUE

INITIALIZE-SINGLE-SOURCE(G,s)
1.  for each vertex v ∈ G.V
2.     v.d = ∞
3.     v.pi = NIL
4.  s.d = 0

RELAX(u,v,w)
1.  if v.d > u.d + w(u,v)
2.     v.d = u.d + w(u,v)
3.     v.pi = u
```

Basically the algorithm works as follows:

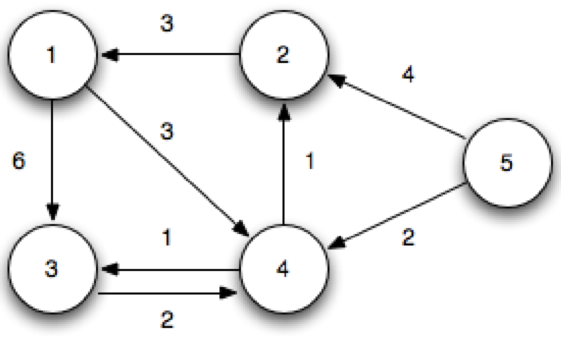
1. Initialize *d*'s, π's, and set $s.d = 0 \Rightarrow O(V)$
2. Loop $|V|$-1 times through all edges checking the relaxation condition to compute minimum distances $\Rightarrow (|V|-1)\,O(E) = O(VE)$
3. Loop through all edges checking for negative weight cycles which occurs if any of the relaxation conditions fail $\Rightarrow O(E)$

The run time of the Bellman-Ford algorithm is O($V + VE + E$) = O(VE).

Note that if the graph is a DAG (and thus is known to not have any cycles), we can make Bellman-Ford more efficient by first *topologically sorting G* (O(V+E)), performing the same initialization (O(V)), and then simply looping through each vertex *u in topological order* relaxing only the edges in Adj[u] (O(E)). This method only takes O($V + E$) time. This procedure (with a few slight modifications) is useful for finding *critical paths* for PERT charts.

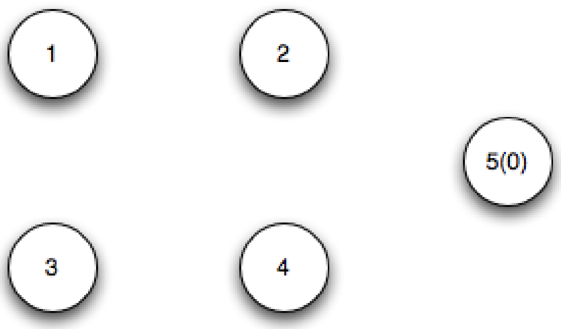
Example

Given the following directed graph



(1,3) = 6
(1.4) = 3
(2,1) = 3
(3,4) = 2
(4,2) = 1
(4,3) = 1
(5,2) = 4
(5,4) = 2

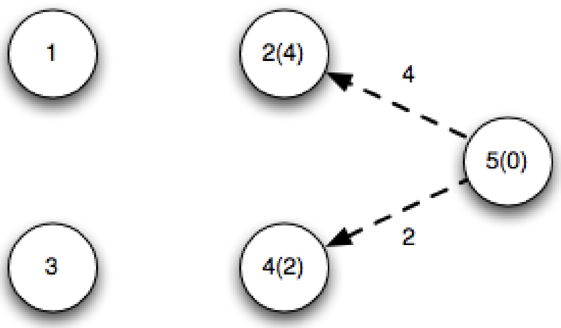
Using vertex 5 as the source (setting its distance to 0), we initialize all the other distances to ∞.



(1,3) = 6
(1.4) = 3
(2,1) = 3
(3,4) = 2
(4,2) = 1
(4,3) = 1
(5,2) = 4
(5,4) = 2

	1	2	3	4	5
d	∞	∞	∞	∞	**0**
π	/	/	/	/	/

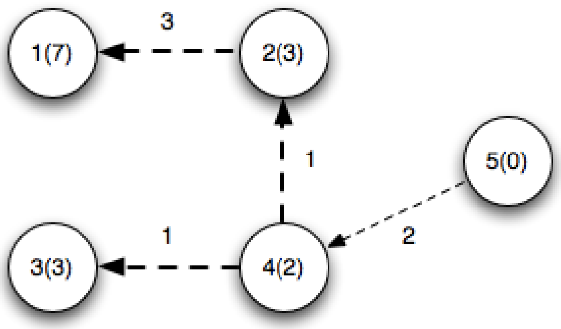
Iteration 1: Edges (u_5,u_2) and (u_5,u_4) relax updating the distances to 2 and 4



(1,3) = 6
(1.4) = 3
(2,1) = 3
(3,4) = 2
(4,2) = 1
(4,3) = 1
(5,2) = 4
(5,4) = 2

	1	2	3	4	5
d	∞	**4**	∞	**2**	0
π	/	**5**	/	**5**	/

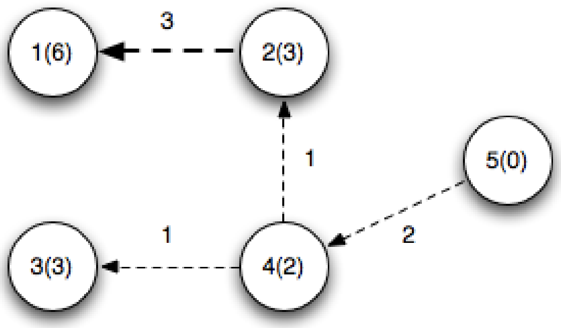
Iteration 2: Edges (u_2,u_1), (u_4,u_2) and (u_4,u_3) relax updating the distances to 1, 2, and 4 respectively. Note edge (u_4,u_2) finds a shorter path to vertex 2 by going through vertex 4



(1,3) = 6
(1.4) = 3
(2,1) = 3
(3,4) = 2
(4,2) = 1
(4,3) = 1
(5,2) = 4
(5,4) = 2

	1	2	3	4	5
d	**7**	**3**	**3**	2	0
π	**2**	**4**	**4**	5	/

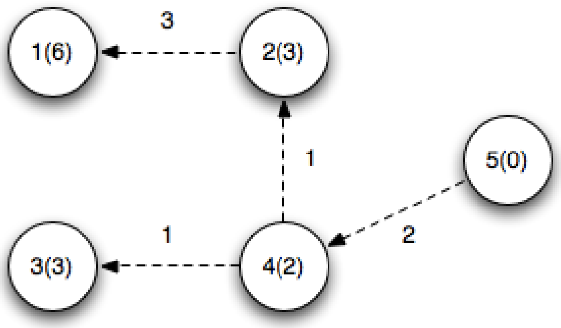
Iteration 3: Edge (u_2,u_1) relaxes (since a shorter path to vertex 2 was found in the previous iteration) updating the distance to 1



(1,3) = 6
(1.4) = 3
(2,1) = 3
(3,4) = 2
(4,2) = 1
(4,3) = 1
(5,2) = 4
(5,4) = 2

	1	2	3	4	5
d	**6**	3	3	2	0
π	**2**	4	4	5	/

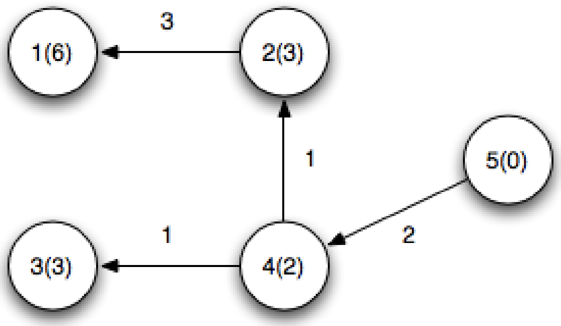
Iteration 4: No edges relax



(1,3) = 6
(1.4) = 3
(2,1) = 3
(3,4) = 2
(4,2) = 1
(4,3) = 1
(5,2) = 4
(5,4) = 2

	1	2	3	4	5
d	6	3	3	2	0
π	2	4	4	5	/

The final shortest paths from vertex 5 with corresponding distances is



	1	2	3	4	5
d	6	3	3	2	0
π	2	4	4	5	/

Negative cycle checks: We now check the relaxation condition one additional time for each edge. If any of the checks pass then there exists a negative weight cycle in the graph.

$v_3.d > u_1.d + w(1,3) \Rightarrow 4 \not> 6 + 6 = 12$ ✓

$v_4.d > u_1.d + w(1,4) \Rightarrow 2 \not> 6 + 3 = 9$ ✓

$v_1.d > u_2.d + w(2,1) \Rightarrow 6 \not> 3 + 3 = 6$ ✓

$v_4.d > u_3.d + w(3,4) \Rightarrow 2 \not> 3 + 2 = 5$ ✓

$v_2.d > u_4.d + w(4,2) \Rightarrow 3 \not> 2 + 1 = 3$ ✓

$v_3.d > u_4.d + w(4,3) \Rightarrow 3 \not> 2 + 1 = 3$ ✓

$v_2.d > u_5.d + w(5,2) \Rightarrow 3 \not> 0 + 4 = 4$ ✓

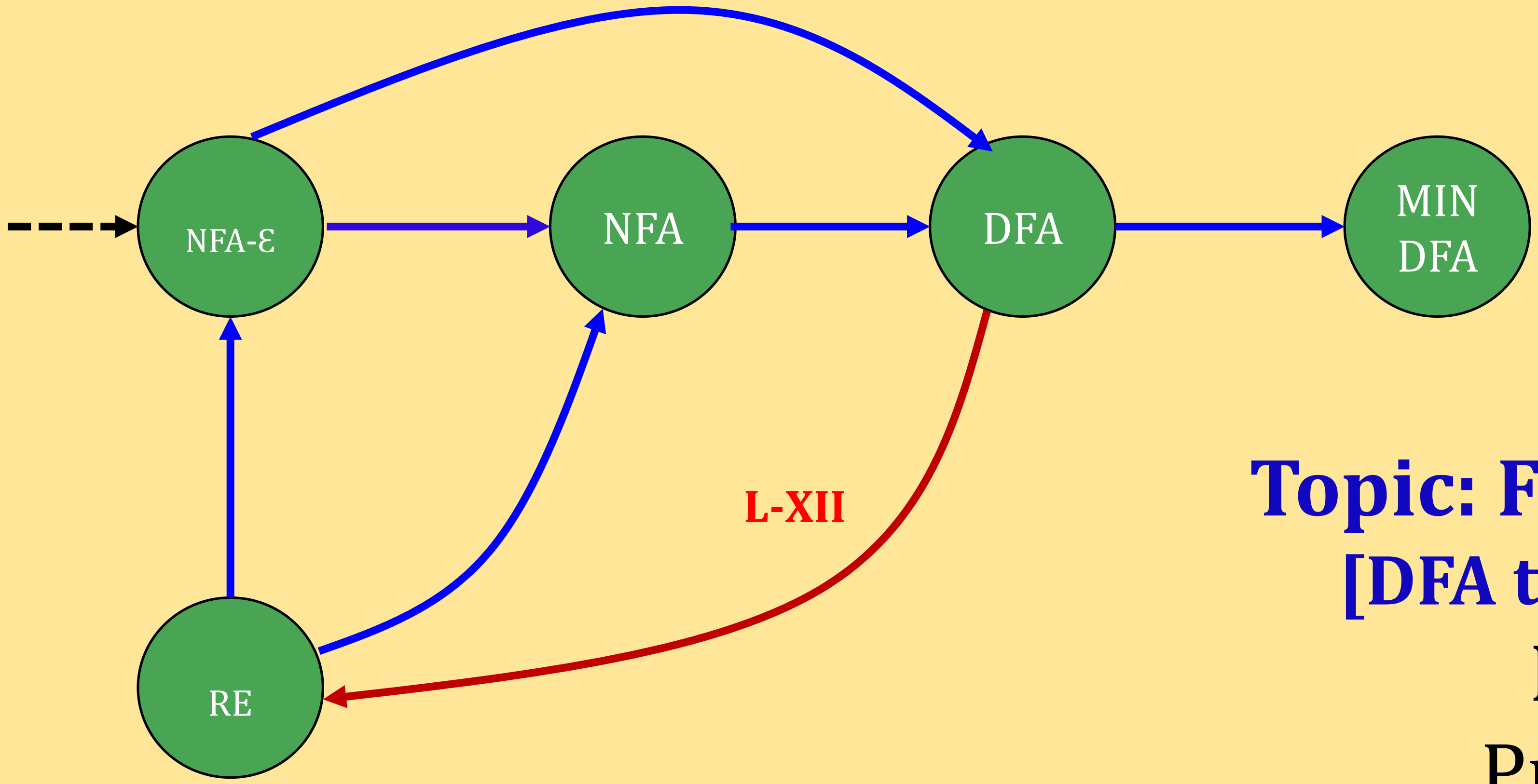
$v_4.d > u_5.d + w(5,4) \Rightarrow 2 \not> 0 + 2 = 2$ ✓

Note that for the edges *on the shortest paths* the relaxation criteria gives equalities.

Additionally, the path to any reachable vertex can be found by starting at the vertex and following the π's back to the source. For example, starting at vertex 1, $u_1.\pi = 2$, $u_2.\pi = 4$, $u_4.\pi = 5 \Rightarrow$ the shortest path to vertex 1 is $\{5,4,2,1\}$.

Formal Language & Automata Theory

PCC-CS 403



Topic: Finite Automata [FA]

[DFA to RE (Arden's Theorem)]

Lecture – XII

Prof. Mithun Roy

Arden's Theorem

Statement –

Let **P** and **Q** be two regular expressions.

If **P** does not contain null string, then **R = Q + RP** has a unique solution that is **R = QP***

Proof –

R = Q + (Q + RP)P [After putting the value R = Q + RP]

= Q + QP + RPP

When we put the value of **R** recursively again and again, we get the following equation –

$R = Q + QP + QP^2 + QP^3 \ldots\ldots$

$R = Q\ (\varepsilon + P + P^2 + P^3 + \ldots.)$

R = QP* [As P* represents (ε + P + P2 + P3 + ….)]

Hence, proved.

Assumptions for Applying Arden's Theorem

- The transition diagram must not have NULL transitions
- It must have only one initial state

Method

Step 1 – Create equations as the following form for all the states of the DFA having n states with initial state q_1.

$$q_1 = q_1R_{11} + q_2R_{21} + \dots + q_nR_{n1} + \varepsilon$$

$$q_2 = q_1R_{12} + q_2R_{22} + \dots + q_nR_{n2}$$

…………………………

…………………………

…………………………

…………………………

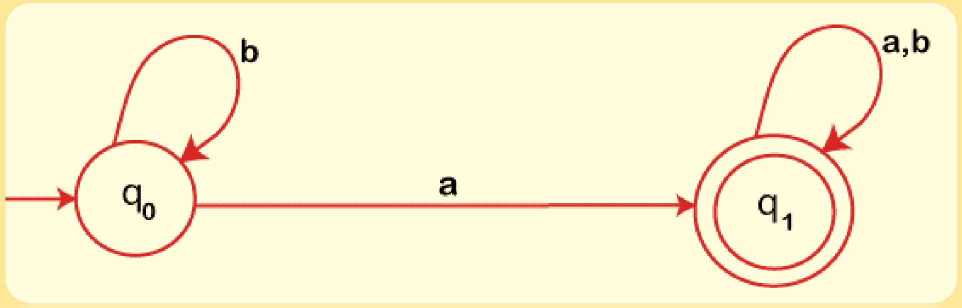
$$q_n = q_1R_{1n} + q_2R_{2n} + \dots + q_nR_{nn}$$

$\mathbf{R_{ij}}$ represents the set of labels of edges from $\mathbf{q_i}$ to $\mathbf{q_j}$, if no such edge exists, then $\mathbf{R_{ij} = \emptyset}$

Step 2 – Solve these equations to get the equation for the final state in terms of $\mathbf{R_{ij}}$

Example - I

Construct a regular expression corresponding to the automata given below –



Here the initial state and final state is $\mathbf{q_0}$.
The equations for the tow states q0 and q1 are as follows –

$$q_0 = q_0 b + \in \quad (i)$$
$$q_1 = q_0 a + q_1(a + b) – (ii)$$

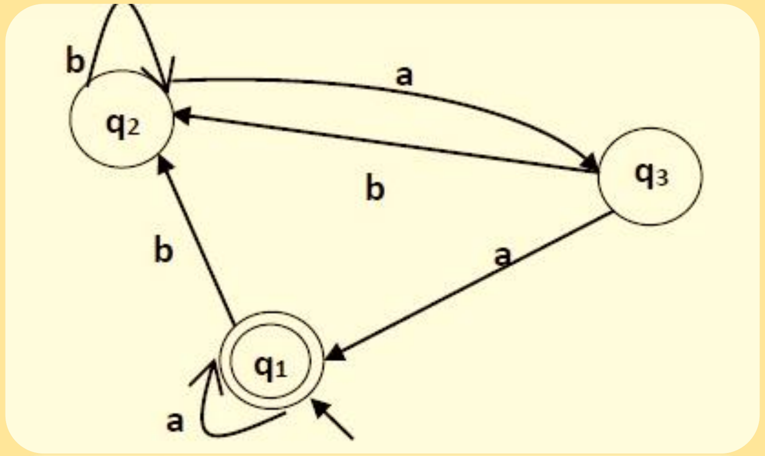
From equation (i) $R = q_0, Q = \in, P = b \Rightarrow q_0 = \in \ + q_0 b \Rightarrow q_0 = \in b^* = b^*$

From (ii), $q_1 = b^* a + q_1(a + b) \Rightarrow q_1 = b^* a(a + b)^*$

So, The RE is $\mathbf{b^* a(a + b)^*}$

Example - II

Construct a regular expression corresponding to the automata given below –



Here the initial state and final state is **q_1**.

The equations for the three states q1, q2, and q3 are as follows –

$q_1 = q_1a + q_3a + \varepsilon$

(ε move is because q1 is the initial state)

$q_2 = q_1b + q_2b + q_3b$

$q_3 = q_2a$

$q_2 = q_1b + q_2b + q_2ab$

$= q_1b + q_2(b + ab)$

$R = q_2, Q = q_1b\,, P = (b + ab)$

Using the Arden's

$q_2 = q_1b(b + ab)^*$

$q_3 = q_2a = q_1b(b + ab)^*a$

Finally,

$q_1 = q_1a + q_1b(b + ab)^*aa + \in$

$q_1 = \in\ + q_1(a + b(b + ab)^*aa)$

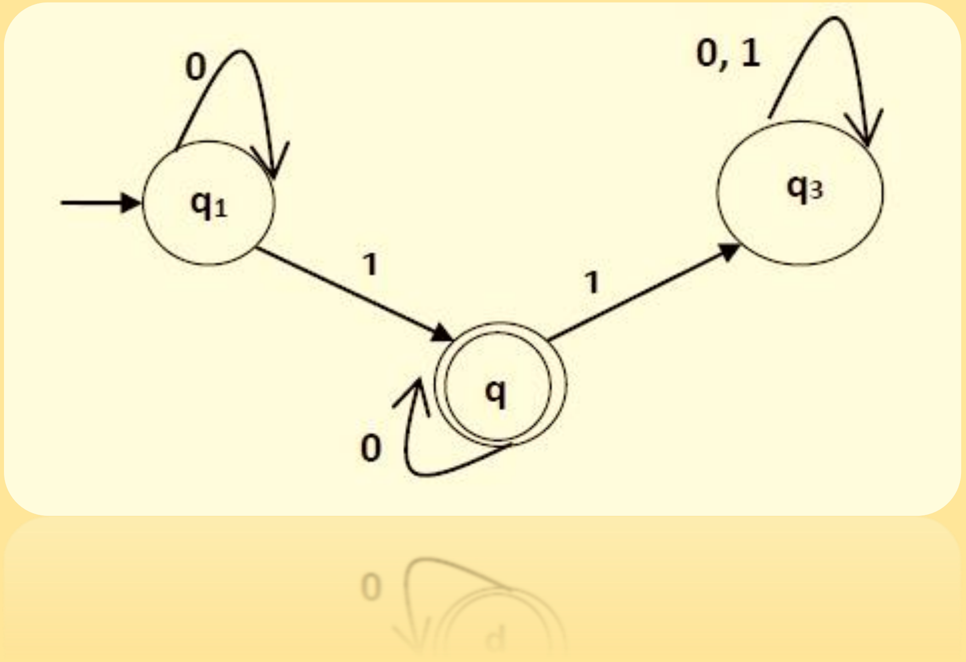
Using the Arden's

$q_1 = (a + b(b + ab)^*aa)^*$

So, $R = (a + b(b + ab)^*aa)^*$

Example - III

Construct a regular expression corresponding to the automata given below –



Here the initial state and final state is $\mathbf{q_1}$.
The equations for the three states q1, q2, and q3 are as follows –

$$q_1 = \in + q_1 0 \quad - (i)$$

$$q_2 = q_1 1 + q_2 0 \quad - (ii)$$

$$q_3 = q_2 0 + q_3 (0 + 1) - (iii)$$

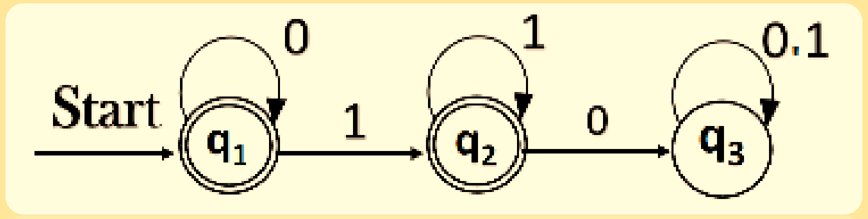
$$\text{From (i)}, q_1 = 0^*, (\text{Using Arden's})$$

$$\text{So}, q_2 = 0^* 1 + q_2 0 = 0^* 1 0^*$$

$$\mathbf{R = 0^* 1 0^*}$$

Example - IV

Construct a regular expression corresponding to the automata given below –



Here the initial state and final state is **q_1**.
The equations for the three states q1, q2, and q3 are as follows –

$$q_1 = \in \; + q_1 0 \; - (\mathrm{i})$$

$$q_2 = q_1 1 + q_2 1 \; - (\mathrm{ii})$$

$$q_3 = q_2 0 + q_3 (0 + 1) - (\mathrm{iii})$$

From (i) $q_1 = 0^*$

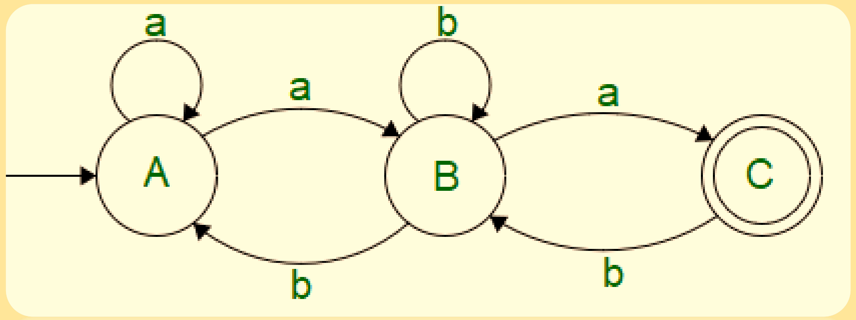
So, $q_2 = 0^*1 + q_2 1 = 0^*11^*$

Here, q_2 is not depends on q_3

So, $R = 0^*1\,1^*$

Example - V

Construct a regular expression corresponding to the automata given below –



$A = Bb + Aa + \in \ -(\text{i})$
$B = Aa + Cb + Bb \ -(\text{ii})$
$C = Ba - (\text{iii})$

$\text{From (i)}, A = Bba^*$

$\text{From (ii), we put } A = Bba^*, \text{So, we get, } B$
$= Cb + Bba^*a + \ Bb \Rightarrow Cb + B\,(ba^*a + b)$
$\text{So}, B = Cb(ba^*a + b)^*$

$\text{From (iii), we put, } B = Cb(ba^*a + b),$
$\text{So we get, } C = \in + Cb(ba^*a + b)a$

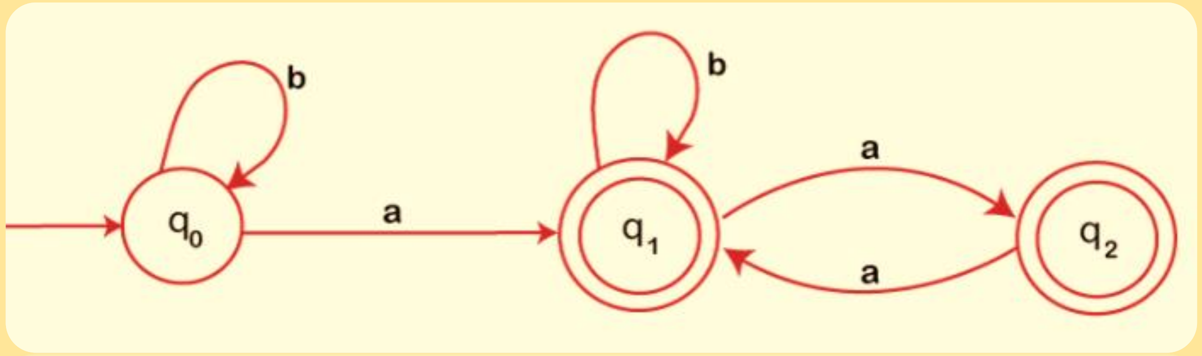
$\text{Finally}, C = (b(ba^*a + b)a)^*$

Regular Expression

R = $\mathbf{(b(ba^*a + b)a)^*}$

Example - VI

Construct a regular expression corresponding to the automata given below –



Finally,

$q_1 + q_2 = b^*a\ (b + aa)^* + b^*a\ (b + aa)^*a$

$= b^*a(b + aa)^*(\in\ + a) = b^*a\ (b + aa)^*a$

So, The Regular Expression is

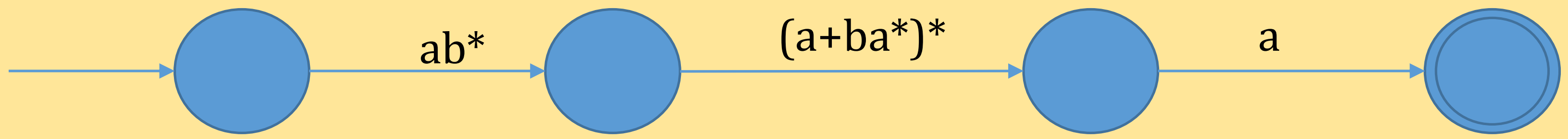
$r = \ b^*a\ (b + aa)^*a$

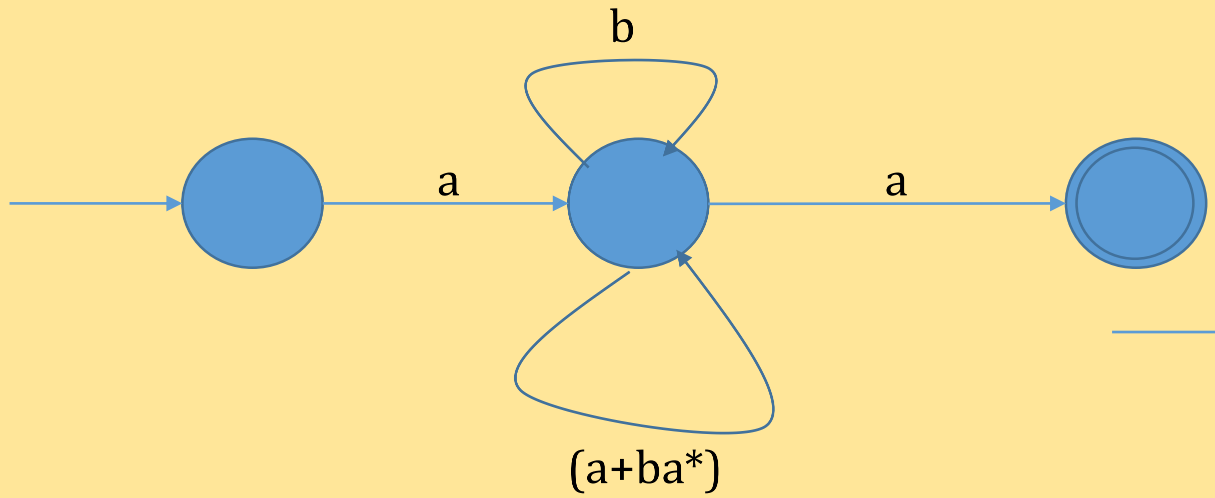
$q_0 = \in\ + q_0b\ \ - (i)$
$q_1 = q_0a + q_1b + q_2a\ \ - (ii)$
$q_2 = q_1a\ \ - (iii)$

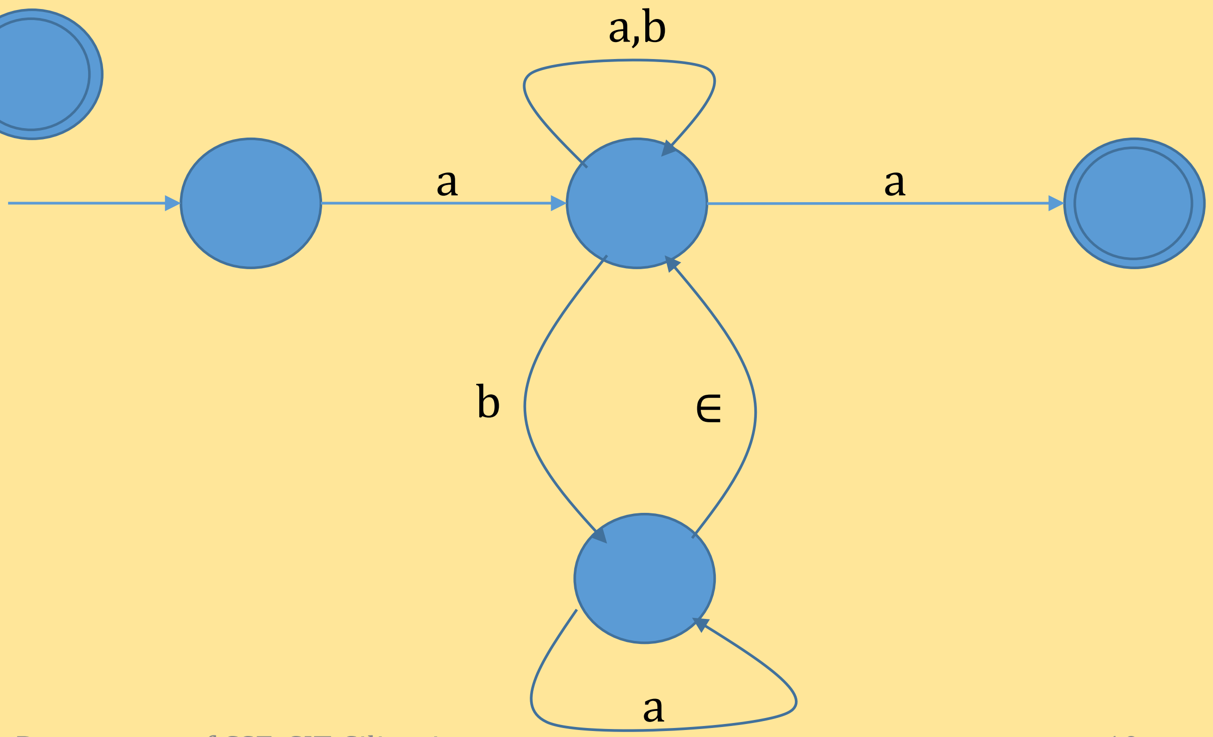
From (i) $q_0 = b^*$, (Using Arden's)
From (ii) $q_1 = b^*a + q_1b + q_1aa = b^*a + q_1(b + aa) = b^*a(b + aa)^*$
From (iii) $q_2 = b^*a\ (b + aa)^*a$

Homework – X

Draw a FA from the RE = ab*(a + ba*)* a

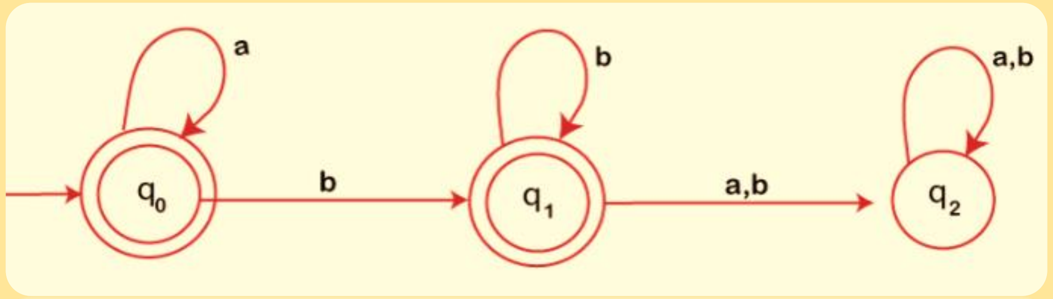






Homework – XI

Construct a regular expression corresponding to the automata given below –



Thank You

Binary Search Tree

Paper Name : *Data Structure and Algorithm*
Paper Code : *CS*302

Department of Information Technology
Siliguri Institute of Technology

September 4, 2019

Outlines

- Definition
- Representation
- Operation
- Complexity
- Application

Definition

- Definition : A Binary search Tree is a Binary Tree in which every node value grater than of its right child and less then of its left child.
- Example :

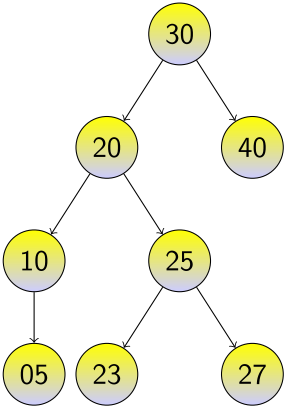


Figure: Example of Binary Search Tree.

Representation

A Binary search Tree can represent in two way: array representation and Linked list representation

- Array Representation:

30	20	40	10	25	-	-	5	-	23	27

Table: Array Representation of BST

- Linked list Representation:

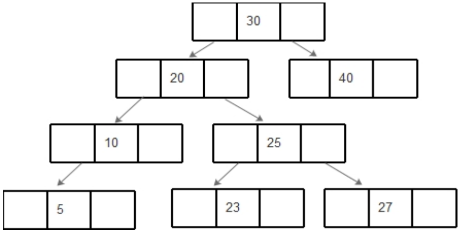


Figure: linked list Representation

Operation of BST

In BST there are four basic operation: Traversal, searching, Insert a node, Delete a node

- Traversal:
- Searching:
- Insert a node:
- Delete a node:

Traversal

Traversal means visiting each node exactly once.

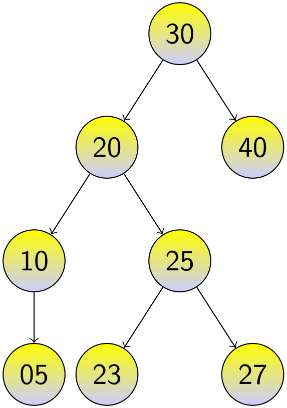


Figure: Example of Binary Search Tree.

- In Order Traversal:The traversing sequence is 5,10,20,23,25,27,30,40
- Pre Order Traversal:The traversing sequence is

Search a node from BST

Search a node from a Tree means the desired node exist or not in a BST. Two way to search a node : recursive way and Non recursive way.

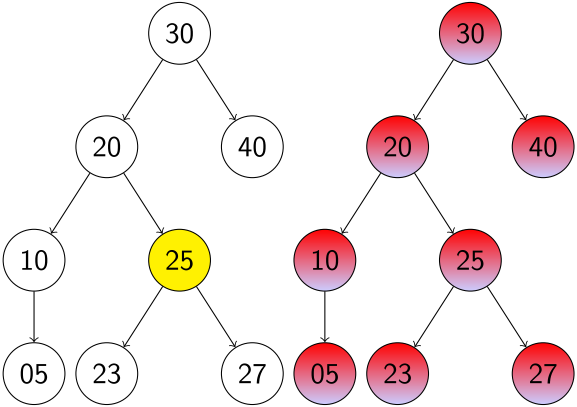


Figure: $SearchNode25and21$

Algorithm for Searching

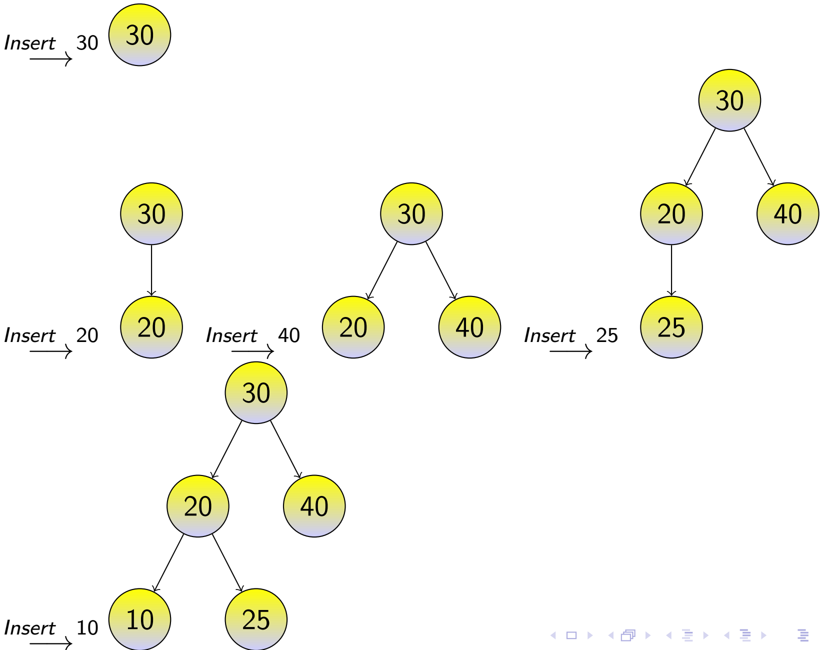
Algorithm 1 Recursive Search algorithm

INPUT: Binary search Tree (T) and current node i.e present node PN under scanning . Searched item/key is K

OUTPUT: KEY ELEMENT FOUND if the function return 1 i.e K in T other wise KEY ELEMENT NOT FOUND the function return 0.

```
Recursive Searching($TNode * PN$,$K$)
**if** $PN == NULL$ **then**
    Return 0
**else if** $K == PN \rightarrow Data$ **then**
    Return 1
**else if** $K \leq PN \rightarrow Data$ **then**
    Return Recursive Searching($PN \rightarrow Lchild, K$)
**else**
    Return Recursive Searching($PN \rightarrow Rchild, K$)
**end if**
```

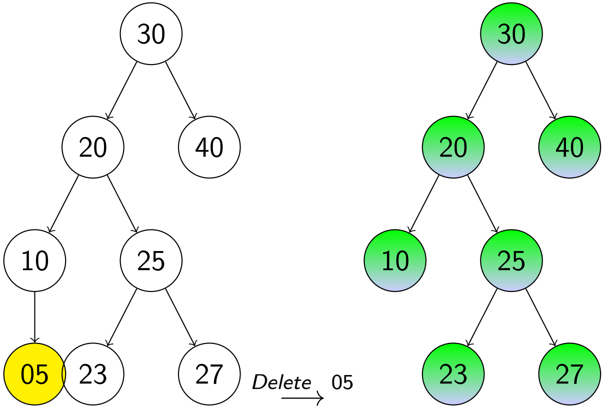
Insert a node in a BST



Delete a node from a BST

During delete a node there are three possibility :
i) Deleted node does not have any child
ii) Deleted node have only one child
iii)Deleted node have two child

- No child:



Delete One child

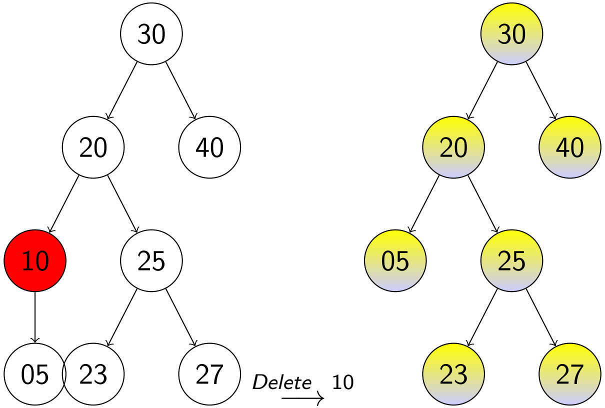


Figure: *Detete* 10 from Binary Search Tree.

Delete Two child

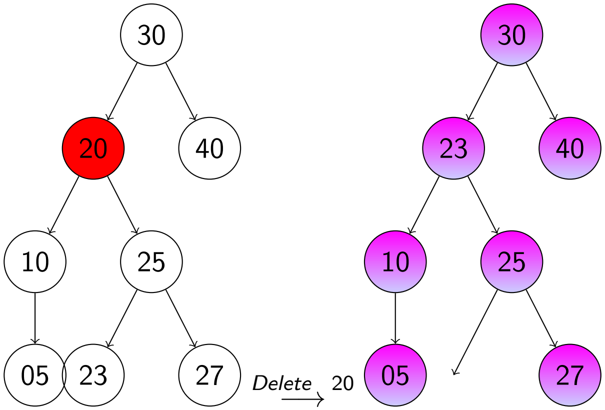


Figure: Detete 20 from Binary Search Tree.

Complexity of BST

Table: Complexity of BST operations

Operations	Best Case	Average Case	Worst Case
Traversal	O(N)	O(N)	O(N)
Searching	O(1)	O(log N)	O(N)
Insert	O(1)	O(log N)	O(N)
Delete	O(1)	O(log N)	O(N)

Applications of BST

(i) BST Used in many searching application where data is constantly entering or leaving such as map and set object in many language library
(ii) Storing a set of Names and being able to look up based on a prefix of the name.
(iii) BST is Used to express arithmetic expressions
(iv) To implement Huffman Coding Algorithm Binary search tree is used

Thank You

Chapter-7-Macromolecular Analysis.

Difference between α-helix and beta plated sheets →

α helix	β Plated Sheet
i) Right-handed coiled rod like structure.	i) Sheet like structure.
ii) Hydrogen bonds form within the polypeptide chain	ii) Beta sheets are formed by linking two or more beta strands by H-bonds.
iii) H-bonds form between N-H group of one amino residue with C-O group of another amino acid.	iii) Hydrogen bonds are formed in between the neighboring N-H and C=O groups of adjacent peptide chains.
iv) -R groups of the amino acids are oriented outside of the helix	iv) R-groups are directed to both inside and outside of the sheet.
v) Alpha-helix can be a single chain.	v) Beta sheet cannot exist as a single beta strand.
vi) Alpha-helix has only one type.	vi) Beta sheet can be parallel anti-parallel or mixed.

RNA is multifunctional → RNA is multifunctional

it's primary function is to encode protein, according ~~DA~~ to the instructions within a cell's DNA. They control and regulate may aspects of protein synthesis in eukaryotes.

Types of RNA in a Eukaryotic cell →

Type of RNA	Functions
messenger RNA (mRNA)	carries information specifying amino acid sequences of proteins from DNA to ribosomes.

Type of RNA	Functions.
ii) Transfer RNA (tRNA)	Plays catalytic (ribozyme) roles and structural roles in ribosome.
iii) Ribosomal RNA (rRNA)	Plays structural and catalytic (ribozyme) roles in ribosomes
iv) primary transcript	Serve as a precursor to mRNA, rRNA, or tRNA and may be processed by splicing or cleavage. In eukaryotes, pre-mRNA commonly contains introns. non-coding segments that are spliced out as the primary transcript is processed, some catalyzing it's own splicing.
v) Small nuclear RNA (snRNA)	Plays structural and catalytic roles in spliceosomes, the complexes of protein and RNA that splice pre-mRNA in the eukaryotic nucleus.
vi) SRP RNA (signal-recognition particle - RNA)	Is a component of the signal recognition particle (SRP), the protein - RNA complex that recognizes the signal peptides of polypeptides targeted to the ER (endoplasmic-reticulum)
vii) SiRNA (small interfering RNA).	control gene expression.

Scanned with CamScanner

Write in detail the function of proteins?

⇒ They do most of the work in cell and are required for the structure, function, and regulation of the body's tissues and organs. Proteins are made up of hundreds or thousands of smaller units called amino acids, which are attached to one another in long chains. These proteins provide structure and support for cells.

	Function	Description	Example
1)	Antibody	Antibodies bind to specific foreign particles, such as viruses and bacteria, to help protect the body	Immunoglobulin G (IgG)
11)	Enzyme	Enzyme carry out almost all of the thousands of chemical reactions that take place in cells. They also assist with the formation of new molecules by reading the genetic information stored in DNA.	Phenylalanine. hydroxylase
3)	Messenger	Messenger proteins, such as some types of hormones transmit signals to coordinate biological processes between different cells, tissues and organs.	Growth hormone.

Scanned with CamScanner

Function	Destription	Example
4) Structural Component	These proteins provide structure and support for cells. on a larger scale, they also allow the body to move.	Actin.
5) Transport storage	These proteins bind and carry atoms and molecules within cells and throughout the body.	Ferritin.

Scanned with CamScanner