UNIT - I

Divide & Conquer Method:

Strassen's Matrix Multiplication:

$$C_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$C_{11} = A_{11} B_{11} + A_{12} B_{21}$$
$$C_{12} = A_{11} B_{12} + A_{12} B_{22}$$
$$C_{21} = A_{21} B_{11} + A_{22} B_{21}$$
$$C_{22} = A_{21} B_{12} + A_{22} B_{22}$$

There are 8 multiplication & 4 additions.
Multiplication can be more expansive than addition in terms of computations. So, we use strassen's matrix Multi-plication.

7 Multiplication is their

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$

$O(n^3)$
$O(n^{2.81})$

$$Q = (A_{21} + A_{22}) B_{11}$$
$$R = A_{11} (B_{12} - B_{22})$$
$$S = A_{22} (B_{21} - B_{11})$$
$$T = (A_{11} + A_{12}) B_{22}$$
$$U = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$V = (A_{12} - A_{22})(B_{21} + B_{22})$$

These matrices P, Q, R, S, T, U and V will be used to calculate C_{ij}'s using 8 additions or subtractions as follows:

$$C_{11} = P + S - T + V$$

$$= 40 + 18 - 20 + 24$$

$$= 58 - 44 = 14$$

$$C_{12} = R + T$$

$$= -4 + 20 = 16$$

$$C_{21} = Q + S$$

$$= 11 + 18 = 29$$

$$C_{22} = P + R - Q + U$$

$$= 40 - 4 - 11 + 9$$

$$= 49 - 15 = 34$$

$$C = \begin{bmatrix} 14 & 16 \\ 29 & 34 \end{bmatrix}$$

Q:

$$A = \begin{bmatrix} \overbrace{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}}^{A_{11}} & \overbrace{\begin{bmatrix} A_{13} & A_{14} \\ A_{23} & A_{24} \end{bmatrix}}^{A_{12}} \\ \underbrace{\begin{bmatrix} A_{31} & A_{32} \\ A_{41} & A_{42} \end{bmatrix}}_{A_{21}} & \underbrace{\begin{bmatrix} A_{33} & A_{34} \\ A_{43} & A_{44} \end{bmatrix}}_{A_{22}} \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

$$B = \begin{bmatrix} \overbrace{\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}}^{B_{11}} & \overbrace{\begin{bmatrix} B_{13} & B_{14} \\ B_{23} & B_{24} \end{bmatrix}}^{B_{12}} \\ \underbrace{\begin{bmatrix} B_{31} & B_{32} \\ B_{41} & B_{42} \end{bmatrix}}_{B_{21}} & \underbrace{\begin{bmatrix} B_{33} & B_{34} \\ B_{43} & B_{44} \end{bmatrix}}_{B_{22}} \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

2

$C_{11} = P + S - T + V$

$C_{12} = R + T.$

$C_{21} = Q + S$

$C_{22} = P + R - Q + U$

$A = \begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix}$ $\quad B = \begin{bmatrix} 1 & 2 \\ 4 & 4 \end{bmatrix}$

$m = 2$

$P = (A_{11} + A_{22})(B_{11} + B_{22})$

$= (2+6)(1+4) = 40$

$Q = (A_{21} + A_{22}) B_{11}$

$= (5+6) 1 = 11$

$R = A_{11}(B_{12} - B_{22})$

$= 2(2-4) = -4$

$S = A_{22}(B_{21} - B_{11})$

$= 6(4-1) = 18$

$T = (A_{11} + A_{12}) B_{22}$

$= (2+3) 4 = 20$

$U = (A_{21} - A_{11})(B_{11} + B_{12})$

$= (5-2)(1+2) = 9$

$V = (A_{12} - A_{22})(B_{21} + B_{22})$

$= (3-6)(4+4) = -24$

Greedy - Method

Greedy method is used for optimization problems i.e. from the available alternations it makes the choice that looks best, optimum at the moment.

Greedy algorithm consists of 2 parts:

a) Optimal substructure :- If a problem has an optimal soln which includes optimal soln to the subproblem it means that the problem has optimal substructure.

b) Greedy choice property:- A locally optimal choice is made to yield at globally optimal choice i.e. the choice which is best at the moment is chosen, which solves the subproblem.

Knapsack problem:-

* It states that there is a knapsack which has capacity 'W' units. It has to be filled with few items which are 'n' in nos. Each item has a weight 'w_i' and 'v_i' value. Knapsack has to be filled till the capacity of W is not reached. There are 2 variants of knapsack problem:

1. 0/1 knapsack problem:

It says that each item can be put in knapsack either completely or not at all i.e. any item can not be broken & put into knapsack.

A 100 2pd
B 10 2pd
C 120 3pd.

C = 4pd.

A 100 1pd
B 10 2pd
C 120 3pd.

2. Fractional knapsack problem :-

* The fractions of item can taken and put onto knapsack i.e. the items can be broken and filled into knapsack.
* Their are n objects and each object has a weight w_i and profit value v_i. Capacity of knapsack is "W" units.
* If a fraction "x_i" of an object "i" is placed into the knapsack where $0 \le x_i \le i$, then profit of $v_i x_i$ is added.
* It is required that total weight of all the chosen objects should not exceed W and at the same time maximize the profit value.

$$\text{maximize} \sum_{i=1}^{n} v_i x_i \text{ subject to}$$

$$\sum_{i=1}^{n} w_i x_i \le W$$

and $0 \le x_i \le 1, \quad 1 \le i \le n$

Their are few methods to fill the knapsack :-

① Method I :- To fill the knapsack choose the objects with largest profit value. If that object doesn't fit into knapsack wholly, take a fraction of it to ~~attain~~ maximize the profit. Though it is possible that adding another object might be better than adding a fraction of some object with higher profit. But this method does not yield to the optimal sol.ⁿ

Ex:- Knapsack Capacity = 5

item 1 → W1 = 6, V1 = 8.5 → Pf = 42.5

item 2 = W1 = 5, V2 = 8.9 → Pf = 45

W3 = 3, V3 = 5.5 → 16.5

W4 = 2, V4 = 3.5

(2) Method II:- Now we try to be greedy with capacity i.e. choose objects with minimum weights & keep on increasing it slowly but this has disadvantage of low profits.

(3) Method III:- We have to achieve a balance b/w the profit & capacity. Profits needed to maximize & capacity reduce i.e. ratio of v_i/w_i is considered while selecting an object.

Algorithm

Knapsack_greedy

An array v[n] contains profit values of n items, w[n] contains weight of n items.
These are arranged in decreasing order of the v[i]/w[i]
W is a capacity of knapsack and x[n] is the sol.n vector.

Step1: Repeat step 2 for n items.
Step2: [Initialize vector x]
Set x[i] = 0.0 for $1 \le i \le n$
Step3: Repeat step 4 for n items.
tep4: for i = 1 to n
if w[i] ≤ W then:
Set x[i] = 1.0
Set W = W − w[i]
else
exit

Step5: If all items have not been chosen or knapsack is still empty i.e.
If $i \leq n$ & $W \neq 0$, Then:
Set $x[i] = W/w[i]$
Set $W = W - w[i] \times x[i]$

Step6: Exit

Example:- Find an optimal solution to the knapsack instance

$n = 7$, $W = 15$,

$(v_1, v_2, \ldots v_7) = (10, 5, 15, 7, 6, 18, 3)$

and $(w_1, w_2, \ldots w_7) = (2, 3, 5, 7, 1, 4, 1)$

Solution: v_i/w_i ratio for all $1 \leq i \leq 7$ and arranging in decreasing order.

$I_1 = v_1/w_1 = 10/2 = 5$

$I_2 = v_2/w_2 = 5/3 = 1.6$

$I_3 = v_3/w_3 = 15/5 = 3$

$I_4 = v_4/w_4 = 7/7 = 1$

$I_5 = v_5/w_5 = 6/1 = 6$

$I_6 = v_6/w_6 = 18/4 = 4.5$

$I_7 = v_7/w_7 = 3/1 = 3$

ITEMS	v_i/w_i	w_i	v_i	$x[i]$	W (Capacity of knapsack)
I_5	6	1	6	1	14
I_1	5	2	10	1	12
I_6	4.5	4	18	1	8
I_3	3	5	15	1	3
I_7	3	1	3	1	2
I_2	1.6	3	5	2/3	0
I_4	1	7	7	0	

8

Step 1: I_5 : $w[i] \le W$ then: Set $x[i] = 1.0$

$1 \le 15$ True so

$x[5] = 1$ &

$W = W - w[i]$

$W = 15 - 1 = 14$ $W = 14$

Step 2: I_1 : $2 \le 14$ T

so $x[1] = 1$ &

$W = 14 - 2 = 12$ $W = 12$

Step 3: I_6 : $4 \le 12$ T

so $x[6] = 1$ &

$W = 12 - 4$

$W = 8$

Step 4: I_3 : $5 \le 8$ T so $x[3] = 1$

$W = 8 - 5$

$W = 3$

P5: I_7 : $1 \le 3$ T so $x[7] = 1$

$W = 3 - 1$

$W = 2$

P6: I_2 : $3 \not\le 2$ f

P7: But knapsack can still accommodate 2 units. So will get a fraction of I_2 & it is equivalent to

$$x[2] = W / w[2] = 2/3$$

and $W = W - w[2] \times 2/3$

$W = 2 - 3 \times 2/3 = 0$

So, knapsack is full

Thus $x[4] = 0$. It means we will not add item I_4 in the knapsack.

Optimal solution for this problem is $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) =$

$(1, 1, 1, 1, 1, 2/3, 0)$ &

$$V = \sum_{i=1}^{n} x_i v_i$$

profit $V = (1 \times 6) + (1 \times 10) + (1 \times 18) + (1 \times 15)$

$+ (1 \times 3) + (2/3 \times 5) + (0 \times 7)$

$V = 7 + 10 + 18 + 15 + 3 + \frac{10}{3} + 0$

$V = 55.3$

Complexity = $O(n)$

Job Sequencing:

→ Job scheduling consist of problem of scheduling the jobs on a single m/c and to find out the optimal way.

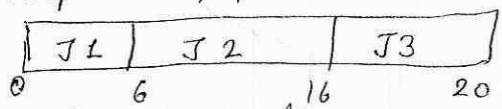
→ In this we have to minimize the average time for a job so that it remains for a least amt. of time in the system.

→ And in job scheduling with deadlines & profits. a certain profit is attainable only when the corresponding job is completed by deadline.

10

$t1 = 6,\ t2 = 10,\ t3 = 4$

1.) Job sequence : 1, 2, 3



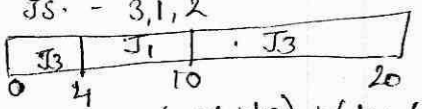
Total time = Waiting Time + Exec. Time
= (0+6+16) + (6+10+4)
= 42

Job scheduling without deadlines :

$P_1 = 24$
$P_2 = 3$
$P_3 = 3$

The Best possible way is SJF

2. JS. - 3, 1, 2



T.T. = (0+4+10) + (4+6+10)
= 14 + 20 = 34

Job Scheduling with deadlines :-

* There are 'n' jobs and with job 'i' there is an Integer deadline $d_i \geq 0$ and a profit $p_i \geq 0$
* for any job 'i' profit 'p_i' is earned iff the job is completed by its deadline 'd_i'
* For one unit of time the process p_i is executed.
* A feasible sol^n for this problem is a subset 'J' of jobs such that each job in this subset can be completed by deadline.
* Optimal sol^n is the max^m profitable out of feasible sol^n

ex:

Let $m = 4$, $(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$ and $(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$ the feasible sol^n and their values are.

	feasible sol^n	Processing seq.	value
1.	(1,2)	(2,1)	110
2.	(1,3)	(1,3) or (3,1)	115
3.	(1,4)	(4,1)	127

4.	(2,3)	(2,3)	25
5.	(3,4)	(4,3)	42
6.	(1)	(1)	100
7	(2)	(2)	10
8.	(3)	(3)	15
9.	(4)	(4)	27

* Now here soln (3) is optimal b'coz of maximum profit value.

* Now here we arrange the jobs in the decreasing order of there profit to find out the optimal soln

	Value	deadline
P_1	100	2
P_4	27	1
P_3	15	2
P_2	10	1

Now we consider job $J=0$ ~~and~~ and $p_i = 0$.

→ Now job J_1 i.e. P_1 is added $J = \{1\}$

→ Next P_4 is considered $J = \{1,4\}$ is feasible soln

→ Now no other job is scheduled b'coz P_3 and P_2's deadline is crossed so optimal soln is $J = \{1,4\}$

Algorithm Greedy Job (d, J, n)

Step 1: Set J := {1}

Step 2: Repeat steps for i := 2 to n do:

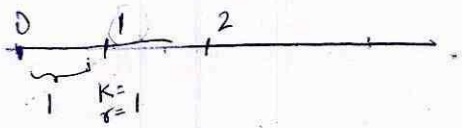
{ If (all jobs in J ∪ {i} can be completed by their deadlines) then:

Set J := J ∪ {i};

[End of If stmt]

[End of while loop]

Step 3: Exit.



Algorithm

JS (d, J, n)

~~Step 1:~~ [d[i] ≥ 1, 1 ≤ i ≤ n are the deadlines, n ≥ 1. the jobs are ordered such that P[1] ≥ P[2] ≥ ... ≥ P[n]. J[i] is the ith job in the optimal soln. 1 ≤ i ≤ K also at termination d[J[i]] ≤ d[J[i+1]], 1 ≤ i < K]

Step 1: Set d[0] := J[0] := 0 // Initialize.

Step 2: Set J[1] := 1 // Include job [1]

Step 3: Set K := 1

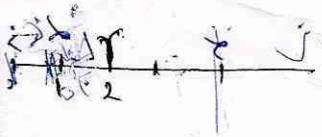
Step 4: Repeat steps 5 to 7 for i := 2 to n do:

Step 5: Set r := K [> can a[i] be included]

Step 6: Repeat steps while ((d[J[r]] > d[i]) and (d[J[r]] ≠ r)) do:

Set r := r − 1

[End of while loop].



↳ Shifting old jobs or not

Step 7: If ((d[J[r]] ≤ d[i]) and (d[i] > r)) then:
for (q = k; q ≥ (r+1); q--) then: // Insert i into J[]
Set J[q+1] := J[q]
[End of for loop]
Set J[r+1] := i; k := k+1
[End of If Str]
[End of step 4 loop]

Step 8: Return k.

q ≥ r+1 : we will schedule the new job at r+1 (q → r+1) & total jobs to be shifted will be k to r+1.

r here tells that if new job has deadline 2 & r is at 3 that means for new job there is no space to be scheduled already 3 spaces (r) are occupied. (time slot)

r = Job can be shifted or not.

K = No of Jobs Scheduled.

Ex

	P_i	d_i
J_1	50	2
J_2	25	2
J_3	15	1
J_4	10	3
J_5	5	3

J: 0 1 2 3 4 5 6 — 0 | 1 | 2 | 4 | | | |

d: 0 1 2 3 4 5 6 — 0 | | | | | | |

k := 1, i = 2
r = 1

Step 6: While ((d[J[1]] > d[2]) & (d[J[1]] ≠ r))
⇒ If (d[1] > d[2]) & (d[1] ≠ 1)
⇒ 2 > 2 False

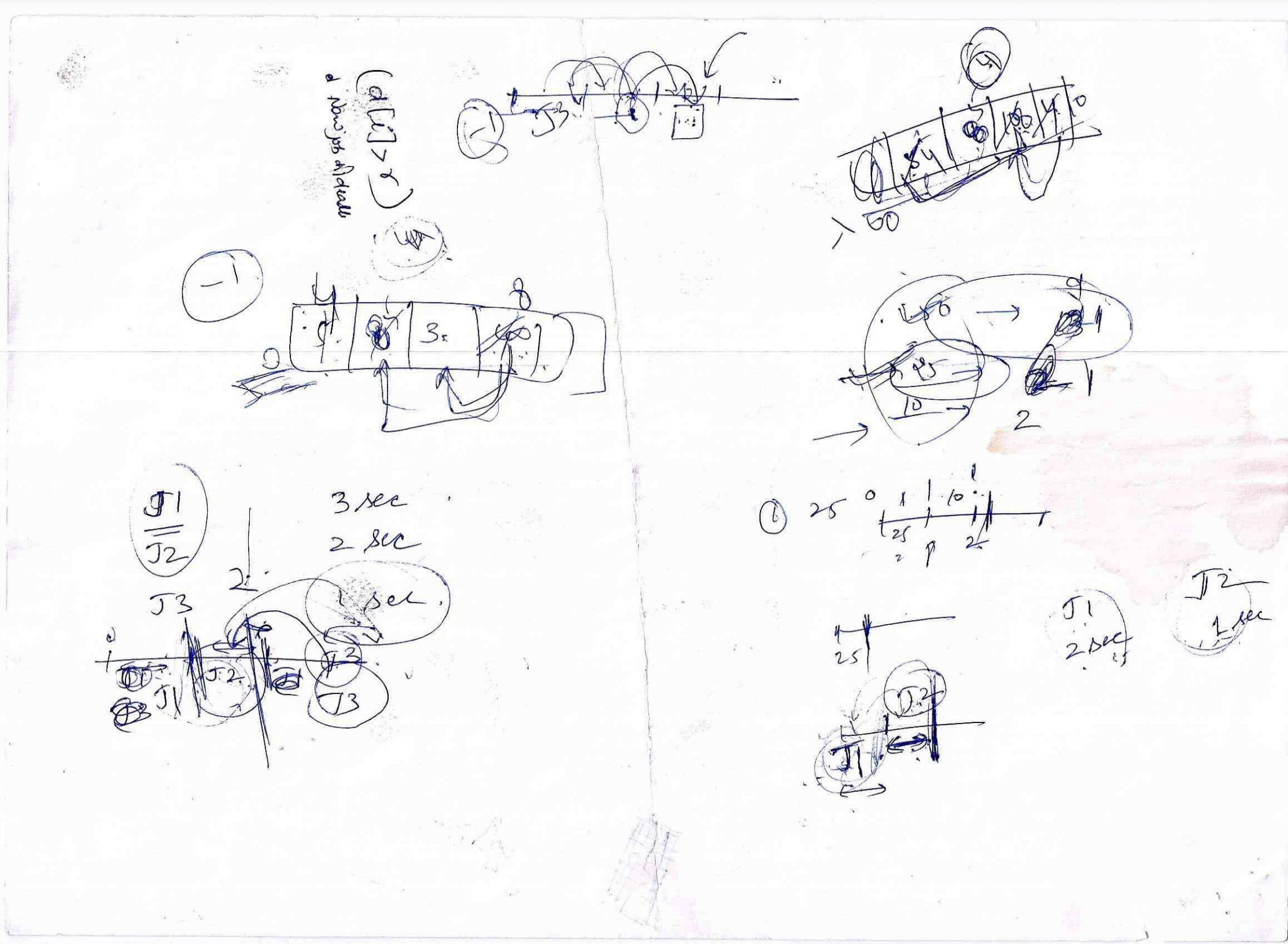
0 1 2 3 4 5 6 — 0 | 1 | | | | | |

4 > 3 & 4 ≠ 1 True
r = r-1 = 0
0 ≤ 3 & 3 > 0
q = 1, q ≥ 1; q--

Step 7: If (2 ≤ 2) & (2 ≥ 1) True
for (q = 1; q ≥ 1; q--) false
Set J[2] = i = 2 ; k = 2

⇒ i = 3 r = 2

(6) d[2] > d[3] & d[2] ≠ 2


J1
J2
3 sec
2 sec
1 sec
J3
J1
2 sec
J2
1 sec

14

Step (7) if $2 \leq 1$ false

$\Rightarrow$ $i = 4$, $r = 2$

(6) while $(d[2] > d[4]) \;\&\; (d[2] \neq 2)$

$2 > 3$ false

(7) if $2 \leq 3$ & $3 > 2$ True

for $(q = 2 ; q \geq 3 ; q--)$ false

set $J[2+1] = i$

$J[3] = 4$, $K = 3$

$\Rightarrow$ $i = 3$, $r = 3$

(6) while $(d[4] > d[3]) \;\&\; (d[4] \neq 3)$

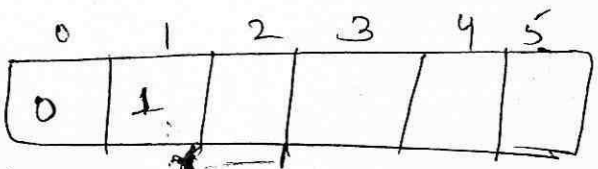
$3 > 3$ false

(7) if $3 \leq 3$ & $3 > 3$ false ✓

Job 1, 2 & 4 scheduled (J_1, J_2, J_4) Ans

J_1	30	4
J_2	25	3
J_3	15	1
J_4	10	2
J_5	5	3

Job



r = K = 1

(1) i = 2

$\rightarrow (d[J[1]] \geq d[2])$ & $(d[J[1]] \neq 1)$

4 > 3 & 4 ≠ 1 T

r = 0

$\rightarrow d[J[0]] > d[2]$ f

0 > 3

$\rightarrow$ if $((d[J[0]] \leq d[2])$ & $(d[2] > 0)$

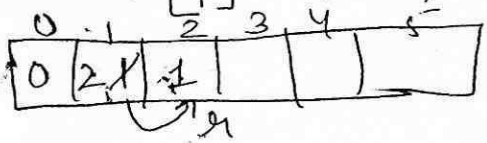
0 ≤ 3 & 3 > 0 T

for (q = 1 ; q ≥ 1 ; T

J[2] = J[1]

q = 0 f

J[1] = 2 , K = 2.



(2) i = 3

r = K = 2

$d[J[2]] > d[3]$ & $d[J[2]] \neq 2$ T

4 > 1 & 4 ≠ 2 T

r = 1

$d[J[1]] > d[3]$ & $d[J[1]] \neq 1$ T

$d[J[0]] > d[3]$ f

if $(d[J[0]] \leq d[3])$ & $d[3] > 0$

T

for (q = 2 ; q ≥ 1 ; T

J[3] = J[2]

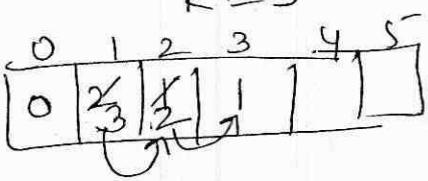
for q = 1 ; q ≥ 1 T

J[2] = J[1]

q = 0 f

J[1] = 3

K = 3



(3) i = 4

r = K = 3

$d[J[3]] > d[4]$ &

4 > 2 $d[J[3]] \neq 3$

4 ≠ 3 T

r = 2

$d[J[2]] > d[4]$ &

$d[J[2]] \neq 2$

3 > 2 & 3 ≠ 2 T

r = 1

$d[J[1]] > d[4]$ & $d[J[1]] \neq 1$

1 > 2 f

if $d[J[1]] \leq d[4]$ & $d[4] > 1$

1 ≤ 2 & 2 > 1 T

for (q = 3 ; q ≥ 2 ;

J_1	P_1 =	100	2
J_4	P_4	27	1
J_3	P_3	15	2
J_2	P_2	10	1

$J[]$ =

0	1	2	3	4
0	J_1			

$k = 1$

$i = 2$

$r = 1$

1 $k = 1$

$i = 2$, $r = 1$

$\Rightarrow d[J[1]] > d[2]$ && $d[J[1]] \neq 1$

$2 > 1$ T && $2 \neq 1$ T

$r = r - 1 = 0$

$d[J[0]] > d[2]$

$0 > 1$ F

$\Rightarrow d[J[0]] \leq d[2]$ && $d[2] > 0$

$0 \leq 1$ T && $1 > 0$ T

&& $q = 1$; $q \geq r + 1$

$1 \geq 0 + 1$ T

$J[2] = J[1]$

$q-- = 0$

$0 \geq 1$ F

3

$J[1] := 2$ $k = 2$

0	1	2	3	4
0	J_2	J_1		

) $i = 3$, $k = 2$

$r = 2$

$d[J[2]] > d[3]$ F

$2 > 2$

$d[J[2]] \leq d[3]$ && $2 > 2$ (F)

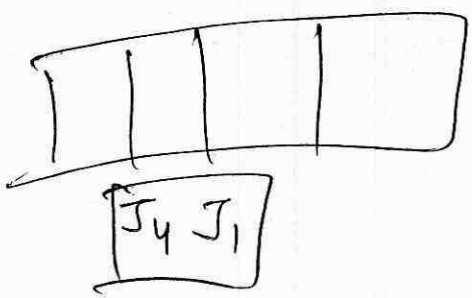
3) $i = 4$, $k = 2$

$r = 2$

$\Rightarrow d[J[2]] > d[4]$ && $2 \neq 2$ (F)

$2 > 1$ T

$\Rightarrow 2 \leq 1$ (F)



Optimal merge Patterns : (app^n – zipping files)

* It is the process of merging two sorted files.
* If more than 2 records are required to be merged. It can be done in pairs.
* Lets there are four files A, B, C, D.
  (i) first merge A & B to give X_1,
  (ii) then X_1 and C or C and D to give X_2
  (iii) finally X_1 & X_2 or other left.
* The optimum solution will be the one which has minimum Computing time.
* We are merging two files at a time so it is also called 2-way merge pattern and therefore represented using binary trees.
* Suppose there are two files with m and n records. it will take (m+n) moves.
* Internal nodes are known as moves, represented by Circles.
* and the two merge files are represented by external nodes, represented by square.
* Level is one more than depth for the leaf nodes, i.e. If level is i, then depth would be i – 1.

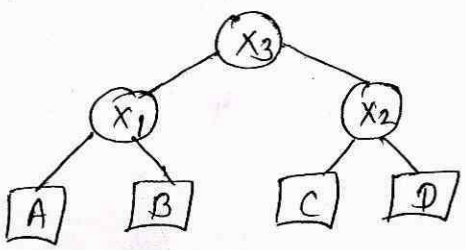
So total no. of moves merging will b

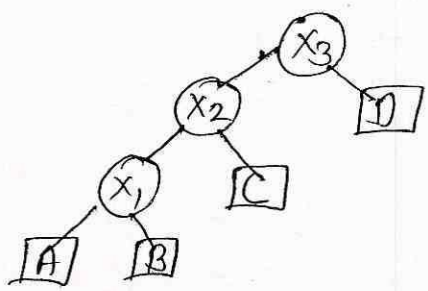
$$\sum_{i=1}^{n} d_i q_i$$

Where 'n' is the no. of files, d_i is the depth of each leaf node from root & q_i is the length of the file.

⇒ * The algorithm has as i/p a list list of n trees.
* Each node in a tree has 3 fields ① lchild, ② rchild & ③ weight.
* Initially, each tree in list has exactly one node.
* This node is an external node and has lchild & rchild field zero whereas weight is the length of one of the n files to be merged.
* t → weight ⇒ length of the merged file it represents (t → weight equals the sum of the lengths of the external nodes in tree t).
* function Tree uses two fun^ns a) Least(list) &
b) Insert(list, t)
* Least(list): finds a tree in list, whose root has least weight and returns a pointer to this tree. This tree is removed from list.
* Insert(list, t): insert the tree with root t into list.

example: A, B, C, D





Algorithm:

```
treenode = record {
    treenode *lchild;
    treenode *rchild;
    integer weight;
};
```

Tree(n)
[List is a global list of n single node]

Step 1: Repeat steps 2 to 5 for i := 1 to n-1 do:
Step 2: Set PTR := new treenode [get a new treenode]
Step 3: Set (PTR → lchild) := Least(list) [Merge two trees
Step 4: Set (PTR → rchild) := Least(list) with smallest leng
Step 5: Set (PTR → weight) := ((PTR → lchild) → weight)
+ ((PTR → rchild) → weight)

Insert(list, PTR)
[End of for loop]

Step 6: Return Least(list)

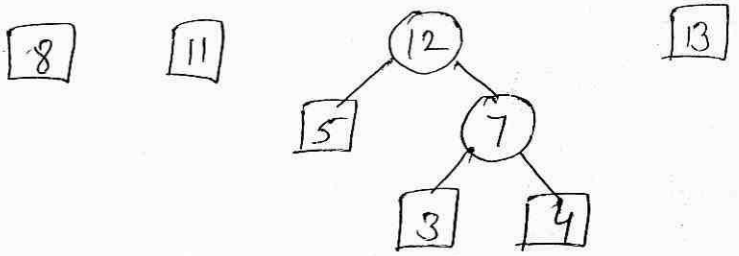
example:

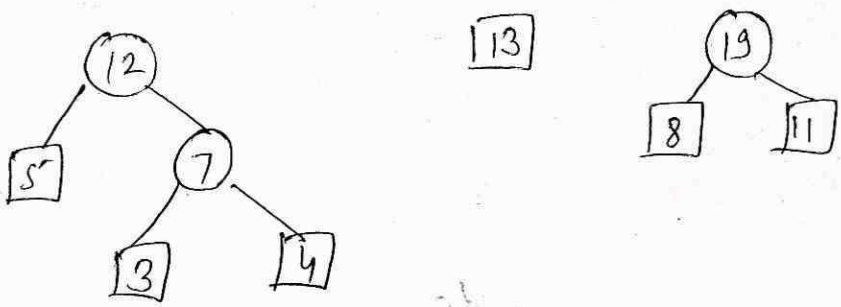
4, 3, 5, 8, 11, 13

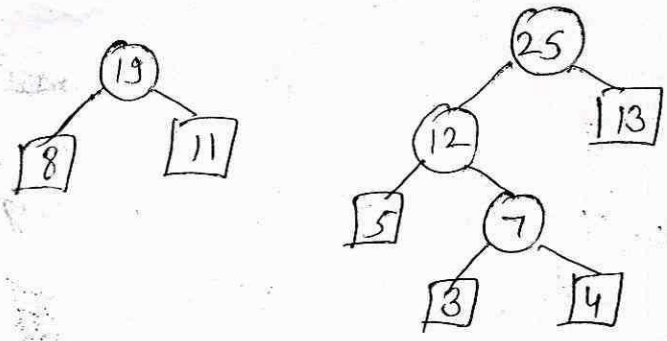
[3] [4] [5] [8] [11] [13]

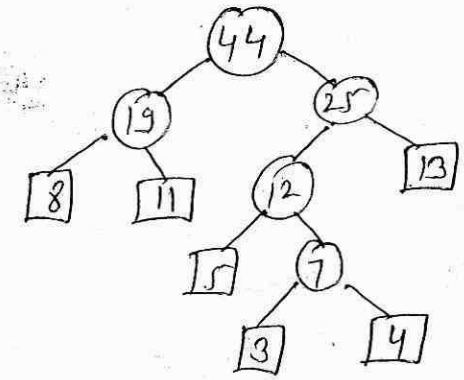
18

1) 

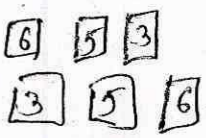
2.) 

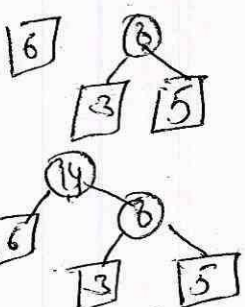
3.) 

4) 

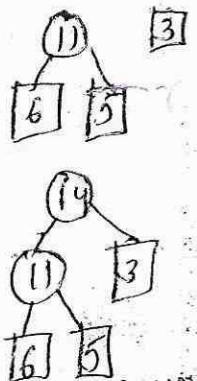
5.) 

Ex: 6, 5, 3





No. of moves = 22



No. of moves = 25

No. of moves $= \sum_{i=1}^{n} d_i q_i$ (Consider external nodes only)

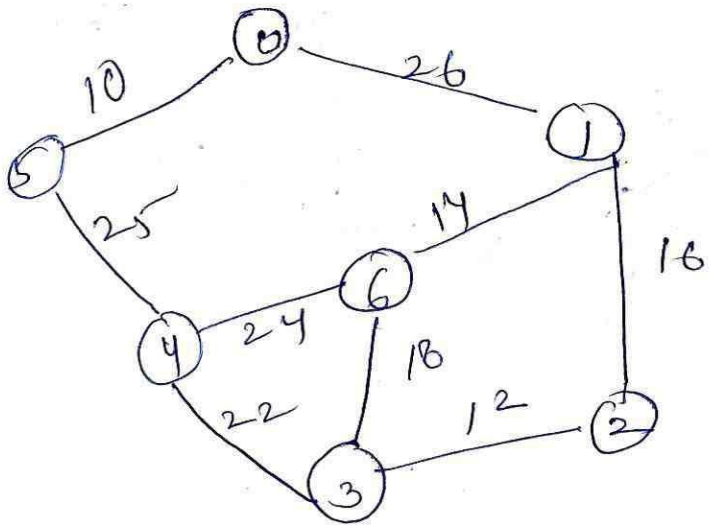
$= (2\times 8) + (2\times 11) + (4\times 3) + (4\times 4) + (3\times 5) + (2\times 13)$

$= 16 + 22 + 12 + 16 + 15 + 26$

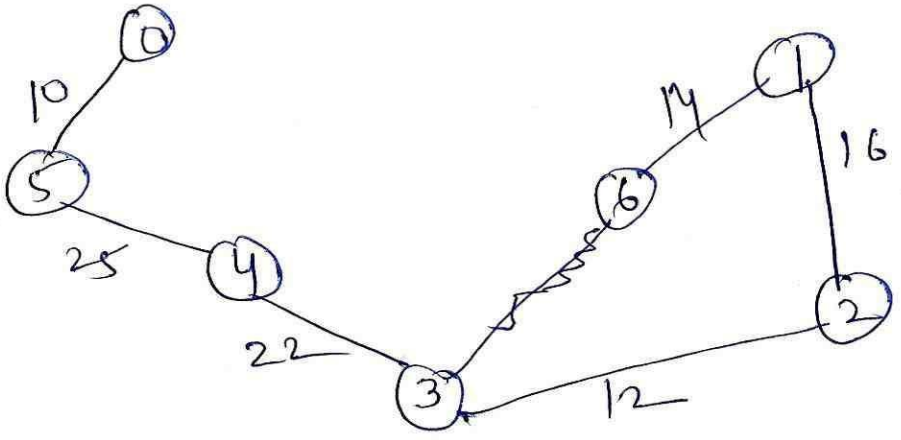
$= 107$

* In the prim's algorithm we again draw a skeleton and the source vertex is given lets say it is 'A'.

Prim's Algorithm:

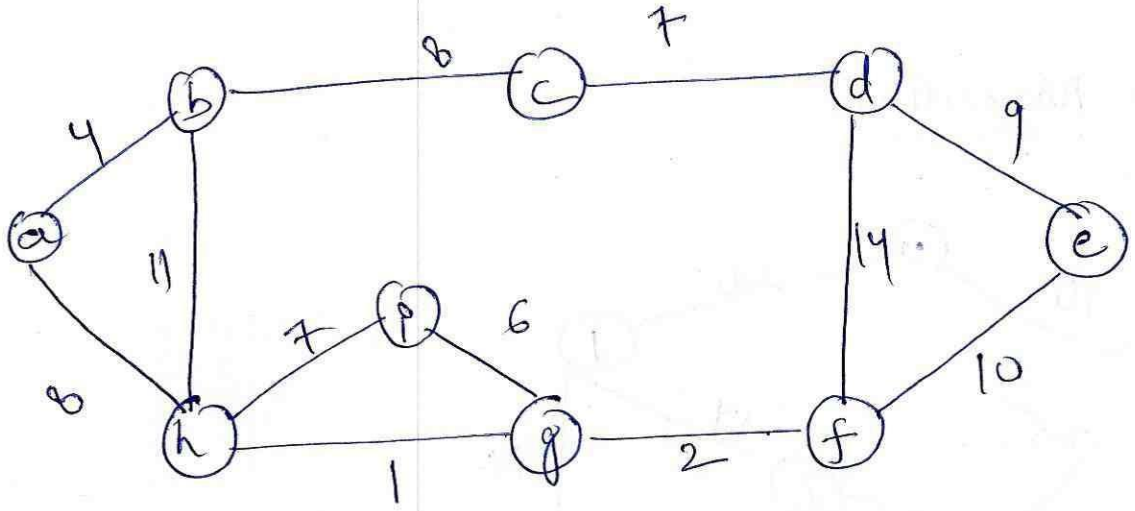


Let us suppose we consider (0) node as a root and then find the MST.



$14+16+12+22+25+10$

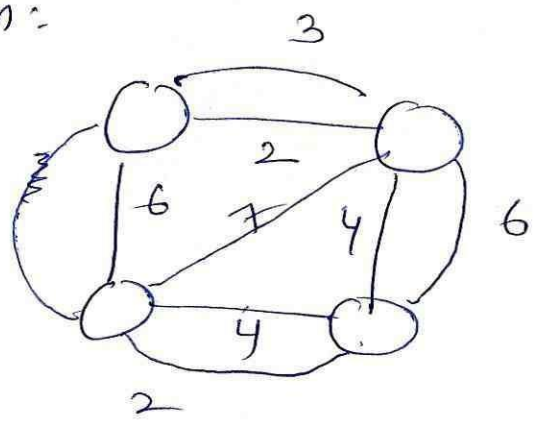
$= 99$

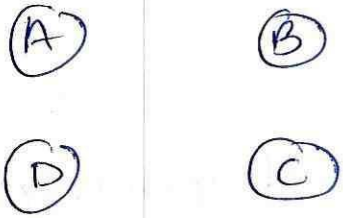

a
b
c
d
e
f
g
h
i
4
8
7
9
11
14
7
6
10
8
1
2

Kruskal's Algorithm :

1) Begin
2) create the edge list of given graph with their wts.
3) sort the edge list according to their wts in ascending order
4) Draw all the nodes to create skeleton for spanning tree.
5) Pick up the edge at the top of the edge list. (i.e. edge with minimum wts.)
6) Remove this edge from the edge list.
7) Connect the vertices in the skeleton with given edge. If by connecting vertices, a cycle is created in the skeleton then discard this edge.
8) Repeat steps (5) to (7) until (n-1) edges are added or list of edges is over.
9) Return.

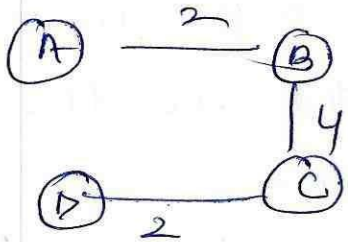
Prim's Algorithm:





* Now taking source vertex (A) we see all the edges that is minimum.

* Now we can Backtrack here also we have two (A) & (B) we find minimum but the cycle should not be there likewise we draw the graph.



MST = 2+2+4 = 8

Counting sort (Non-comparison type sorting)

K → max^m element in array A

1. ~~Array C for len~~

$O(n+K)$ (m)

$\Omega(n)$

1. for $i \leftarrow 0$ to K
   do $C[i] \leftarrow 0$
2. $n \leftarrow$ length[A]
3. for $j \leftarrow 1$ to n //C will contain repetitive value of each element in array A
   do $C[A[j]] \leftarrow C[A[j]] + 1$
   //for each occurrence of a value in A, C value is incremented for that element
4. for $i \leftarrow 1$ to K
   do $C[i] \leftarrow C[i] + C[i-1]$ Count index pos^n of each array A element
5. for $j \leftarrow n$ downto 1
   do $B[C[A[j]]] \leftarrow A[j]$
   Set $C[A[j]] \leftarrow C[A[j]] - 1$
6. Return B & exit

B[8] B[C[B[C[B[C[(3)

A[]:

1	2	3	4	5	6	7	8
6	0	2	0	1	3	4	6

max^m element = 6
size of array C[] = K = 6
n ← length[A] = 8

C[]:

0	1	2	3	4	5	6
0	0	0	0	0	0	0

C[] : =

0	1	2	3	4	5	6
2	1	1	1	1	0	2

← Count repetitive value (Occurrence of each element)

C[] =

0	1	2	3	4	5	6 ← element in array[A]
2	3	4	5	6	6	8

index pos^n of element 6 is 8 (2nd 6)

B[]:

1	2	3	4	5	6	7	8
				3	4		6

Sorted array

$j \leftarrow 8$ to 1

$B[C[A[j]]] \leftarrow A[j]$

$B[C[A[8]]] \leftarrow A[8]$

$B[C[6]] \leftarrow 6$

$B[8] \leftarrow 6$

$C[A[8]] \leftarrow C[A[8]]-1$

$C[6] \leftarrow C[6]-1 = 8-1 = 7$

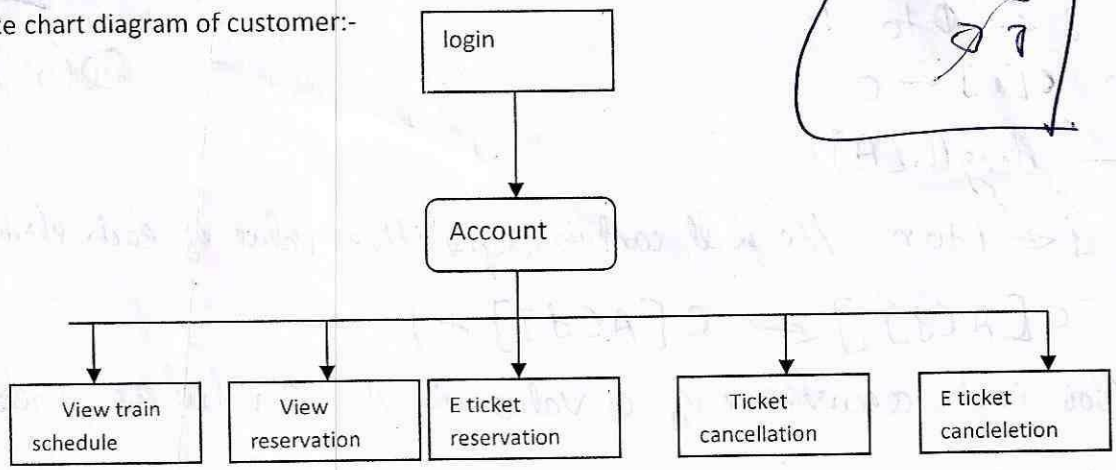
$B[C[4]] \leftarrow 4$

$B[6] \leftarrow 4$

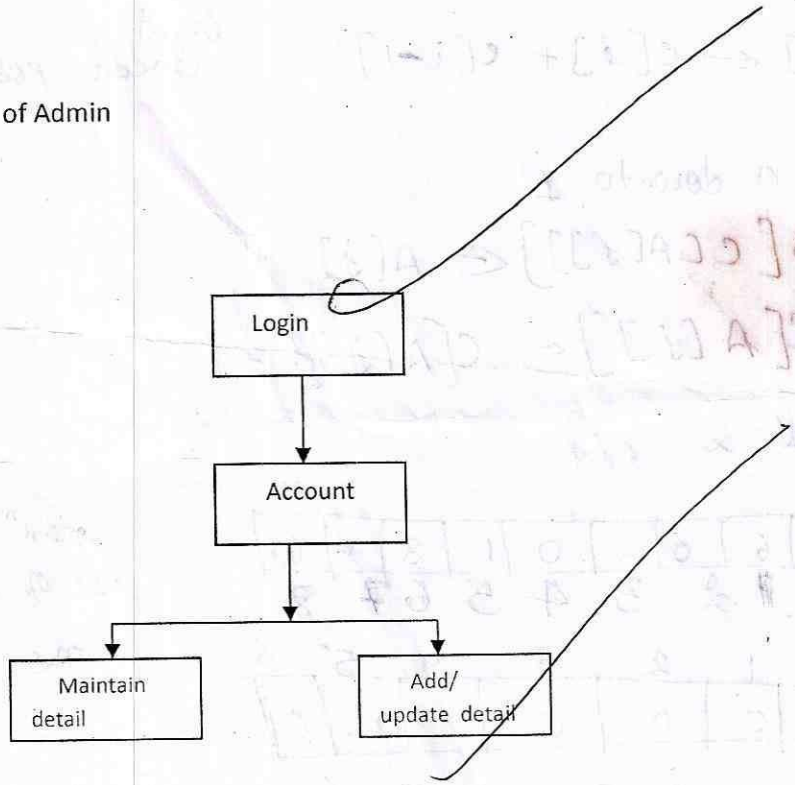
$B[C[3]] \leftarrow 3$

$B[5] \leftarrow 3$

State chart diagram of customer:-



State chart diagram of Admin



NITTTR CHANDIGARH
ICT based STC on
"Wireless Networks" from 02.11.2015 to 06.11.2015
Organized by Department of Computer Science, NITTTR, Chandigarh

DATE	09:30 a.m. to 11:00 a.m.		11:30 a.m. to 1:00 p.m.		2:00 p.m. to 4:00 p.m.
02.11.2015 (Monday)	Registration & Inauguration (RK/AS)	TEA BREAK	Overview of Wireless Communication (RK)	LUNCH BREAK	IP Addressing (AD)
03.11.2015 (Tuesday)	Configuring WiFi (VG)		Cellular Technologies (SBLS)		WSN 802.15.4 standard-Zigbee (SS)
04.11.2015 (Wednesday)	Live Demonstration of WSN Network (AN)		Simulation using Omnet++ - I (AN)		Simulation using Omnet++ - II (AN)
05.11.2015 (Thursday)	Wireless Security –I (VG)		Wireless Security-II (VG)		Challenges in Routing protocols (Saurabh)
06.11.2015 (Friday)	3G Technologies (KB)		4G/LTE Network (KB)		Feedback/ Evaluation (RK/AS)

Course Coordinators: Dr. Rakesh Kumar/ Er. Amrendra Sharan

RK: Dr. Rakesh Kumar, NITTTR Chandigarh

AD: Mr. Amit Doegar, NITTTR Chandigarh

AS: Er. Amrendra Sharan, NITTTR Chandigarh

SS: Dr. Sandeep Singhai, Scientist, CSIO, Chandigarh

AN: Er. Anand Nayyar, KCL Institute of Management and Technology, Jalandhar, Punjab

KB: Mr. Kapil Bhutani, Founder, Telcocrats Technologies, Mohali

VG: Mr. Vipin Gupta, Director, U-Net Solution Moga

SBLS : Prof. SBL Sachan, NITTTR Chandigarh

Branch and Bound

* Branch & Bound functions are used to help for avoiding the generations of subtrees that do not contain an answer node.
* A Branch & Bound method searches a state space tree using any search mechanism in which all the children of the current node are generated before another node is expanded.
* Each answer node has a cost associated with it and minimum cost answer node is to be found.
* At each stage bound for a particular node is calculated and checked if this bound will be able to generate the solution, thus bound indicates that how far we are from the sol.^{n}. If at any node we see that further expansion of node will give worse solution we kill that node.

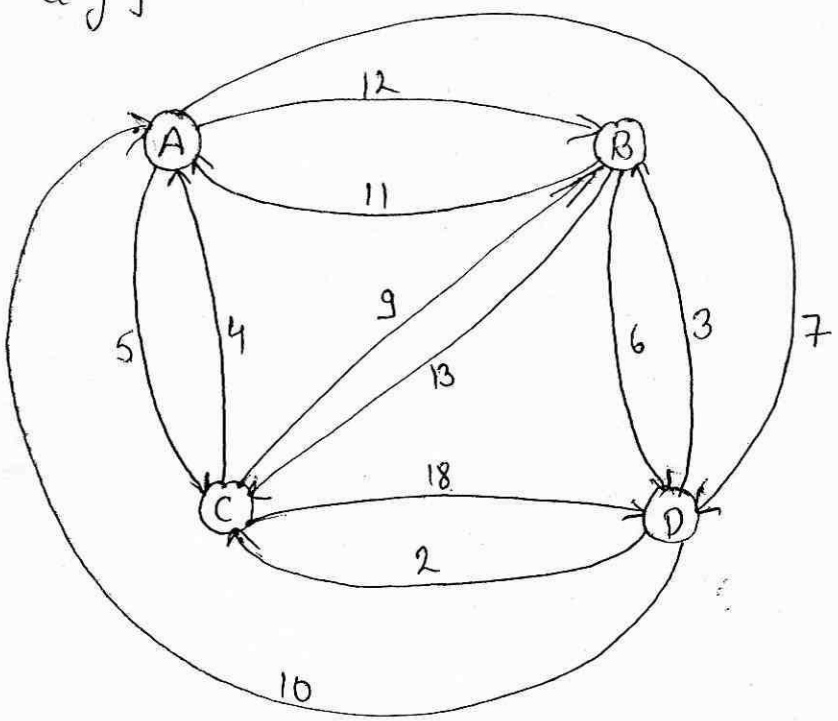
Travelling Salesperson:

* Graph $G(V,E)$ is given the problem states that a salesperson wants to visit a certain no. of cities.
* He knows the distance or cost of or time of journey b/w every pair of cities.
* Problem is to select such a route that starts from his home city passes through each city once and return to

...me city in the shortest possible distance or least cost time.

* Cost is shown by C_{ij} for edge (i, j)

* If $i = j$ then $C_{ij} = \infty$ means he cannot go from city i to city j



Step 1: Construct the cost matrix for the graph

→ To city

from city	A	B	C	D
A	∞	12	5	7
B	11	∞	13	6
C	4	9	∞	18
D	10	3	2	∞

Step 2: Find Reduced cost matrix "R". This is done by selecting the smallest element from each row and column and subtracting it from all other elements & the aim is to have at least one zero in each row & column.

	A	B	C	D	
A	∞	7	0	2	-5
B	5	∞	7	0	-6
C	0	5	∞	14	-4
D	8	1	0	∞	-2

Smallest element from each row is selected & subtracted from other elements

But still column 2 not have a single "0" value so smallest value of col 2 is "1" and subtracting it from other elements gives the reduced cost matrix as:

	A	B	C	D
A	∞	6	0	2
B	5	∞	7	0
C	0	4	∞	14
D	8	0	0	∞
		-1		

So total amount subtracted in the process of finding reduced cost matrix =

$$5 + 6 + 4 + 2 + 1 = \boxed{18}$$

This total amount will become the root of the search tree

Step 3: Let vertex "A" is the root node, so for every node of A we check that the child node is leaf node or not. If not we have to find the reduced cost matrix by following steps.

a) If A is the root node or parent node and B is child node. there is an edge b/w (A,B) so change all entries in row A and column B of reduced cost matrix to "∞".

b) Set (row B, column 1) to ∞

c) Reduce all rows and column in resulting matrix except the rows & columns containing only ∞.

* The total amt. subtracted is denoted by "T"
  cost of child node = cost of parent node + R(A,B) + T
* For edge A → B find reduced cost matrix for B.
* Make row A and column B to ∞
* Also position (B, 1) to ∞

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	7	0
C	0	∞	∞	14
D	8	∞	0	∞

all rows & column have at least "0" So T = 0.

Cost of node B = 18 + R(A,B) + T
= 18 + 6 + 0
= 24

for $A \to C$

	A	B	C	D	
A	∞	∞	∞	∞	
B	5	∞	∞	0	
C	∞	4	∞	14	-4
D	8	0	∞	∞	
	-5				

So, the find reduced cost matrix for C is

	A	B	C	D
A	∞	∞	∞	∞
B	0	∞	∞	0
C	∞	0	∞	10
D	3	0	∞	∞

$T = 5+4 = 9$

So cost of child node $C = 18 + R(A,C) + T$

$= 18 + 0 + 9 = \boxed{27}$

for edge $A \to D$

	A	B	C	D	
A	∞	∞	∞	∞	
B	5	∞	7	∞	-5
C	0	4	∞	∞	
D	∞	0	0	∞	

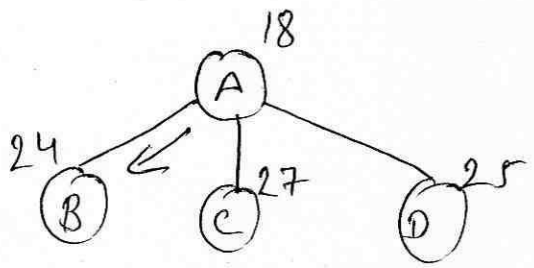
6

	A	B	C	D
A	∞	∞	∞	∞
B	0	∞	2	∞
C	0	4	∞	∞
D	∞	0	0	∞

$T = 5$

cost of D = 18 + R(A, D) + T

= 18 + 2 + 5 = **25**



In this tree node B has minimum value in others words path (A, B) is the most promising with cost 24. So we discard all other nodes & expand node B further.

From B there is path to C and D :

a) for edge A → B → C change all entries of row A, row B and column C to ∞. Also change R(B,1) & R(C,1) to ∞.

	A	B	C	D	
A	∞	∞	∞	∞	
B	∞	∞	∞	∞	
C	∞	4	∞	14	-4
D	8	0	∞	∞	
	-8				

Reduced cost matrix will be

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	∞	0	∞	10
D	0	0	∞	∞

$\Rightarrow$

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	∞	0	∞	0
D	0	0	∞	∞

-10

$$T = 4 + 10 + 8 = 22$$

Cost of child c with parent B

$$= 24 + R(B,C) + T$$

$$= 24 + 7 + 22$$

$$= \boxed{53}$$

for edge $A \rightarrow B \rightarrow D$

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	0	4	∞	∞
D	∞	0	0	∞

Row A, Row B & column D $= \infty$

$R(B,1)$ & $R(D,1) = \infty$

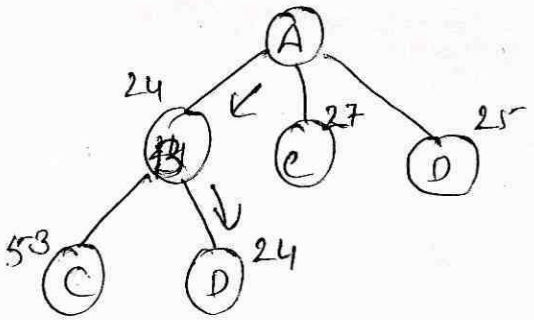
for this $T = 0$

So cost of D with parent B

$$= \text{cost of } B + R(B,D) + T$$

$$= 24 + 0 + 0$$

$$= \boxed{24}$$



Path $A \to B \to D$ is selected. discard all other path.

For verten D the only path left is towards verten C all other path are already traversed.

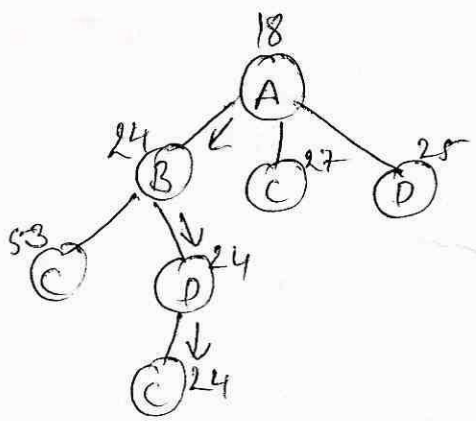
* Now C verten is the leaf node. Each leaf node defines a unique tour.

$$R(D, C) = 0$$

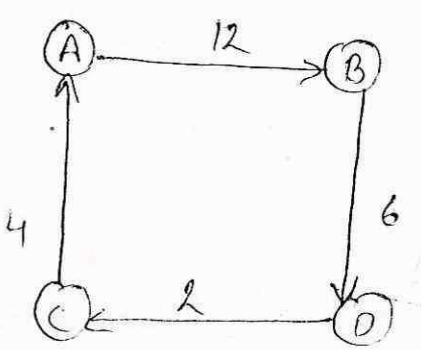
Cost of value of node C = cost of parent node + $R(D, C)$

$$= 24 + 0 + 4$$

$$= \boxed{24}$$



So path is $A \to B \to D \to C \to A$



Optimal cost = 12+6+2+4

= 24 Ans

Backtracking:

* Backtracking algorithm test for a solution, if sol.n is found in the first step the algorithm is solved, if not it recurs once and tests again, continuing until a sol.n is found.

Backtracking can be used to solved.

1) Queen's problem
2) Sum of subset problem
3) Graph coloring problem

1) Queen's Problem:

It is a famous combinational problem. that says "place n-queens on a n×n chess board. such that no queens attacks any other, we have to arrange n-queens on a different rows. Keeping in mind that they are in different columns and diff diagonals at the same time.

for n = 1

	1
1	q_1

for n = 2

	1	2
1	q_1	x
2	x	x

not possible to place 2 & 3 queens on 2×2 & 3×3 chess boards ~~resp~~ resp. such that they donot attack each other.

for n = 3

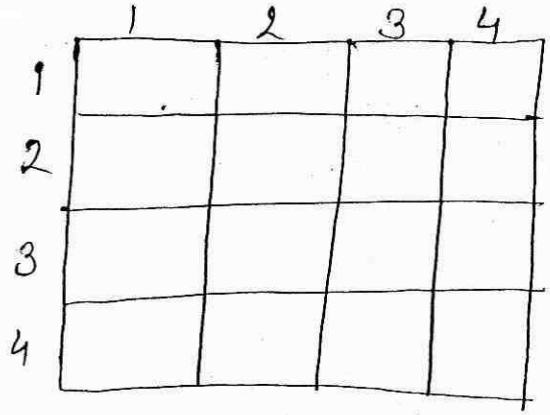
	1	2	3
1	q_1	x	x
2	x	x	q_2

or

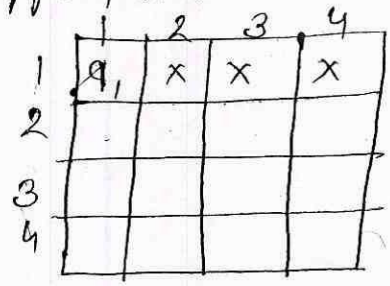
	1	2	3
1	x	q_1	x
2	x	x	x

4 - Queen's Problem:

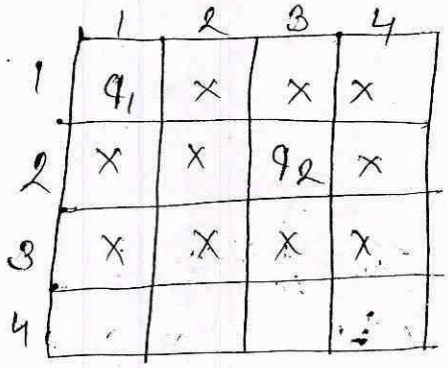
We are given a 4×4 chess board and we have to place the 4 - queens on this chess board such that they are in different rows, columns & diagonals.



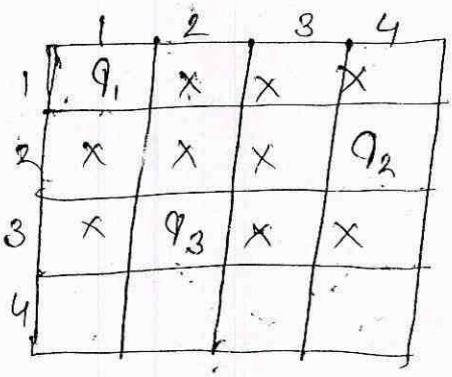
→ Starting from queen q_1, select appropriate column in row1, col1 is the first valid pos.n for queen 'q_1'



→ Queen q_2 is to be placed in row2 but we cannot place queen q_2 in col 1 and col 2 b'coz they are the attacking positions. So we place q_2 in col 3'



→ Queen q_3: Now this q_3 is to be placed in row '3' but it cannot be placed in col 1, col 2, col 3, col 4 b'coz of the attacking, so we have to backtrack one step. So if we place q_2 in col '4' the we can put q_3 in col 2



→ Now we have to place q_4 (queen) but we are at dead end bcoz their is not position for q_4 so we need to backtrack to find the other posⁿ of q_3. but when q_2 is placed in col4 q_3 should be placed in col2. so we need to see the position of 'q_2' but q_2 cannot be placed in any other col bcoz of q_1 is placed in col1, so we need to backtrack q_1

	1	2	3	4
1	x	q_1	x	x
2	x	x	x	q_2
3	q_3	x	x	x
4	x	x	q_4	x

So this is the solⁿ for 4-queens problem.

8-queens problem:

	1	2	3	4	5	6	7	8
1			q_1					
2					q_2			
3		q_3						
4								q_4
5	q_5							
6							q_6	
7				q_7				
8						q_8		

Sum of Subset problem:

This problem can be stated as follows:-

⇒ Given a set of n positive nos w_i, $1 \le i \le n$ and variable m, problem calls for all the subsets of w_i, such that their sum is m for eg. $(w_1, w_2, w_3, w_4) = (11, 13, 24, 7)$ & $m = 31$

The subsets of w which makes the sum 31 are (11, 13, 7) & (24, 7)

Solution vector can be represented by the indices of set n for above example. The sol^n vector is (1, 2, 4) & (3, 4) this gives us the variable sized soln vector.

→ Another way of representing the sol.^{n} vector is in binary form. The soln vector x_i can be either 0 or 1. If w_i is chosen then $x_i = 1$ otherwise $x_i = 0$ for the above example the sol.^{n} vector is (1, 1, 0, 1) & (0, 0, 1, 1). This gives the sol.^{n} vector of fixed size.

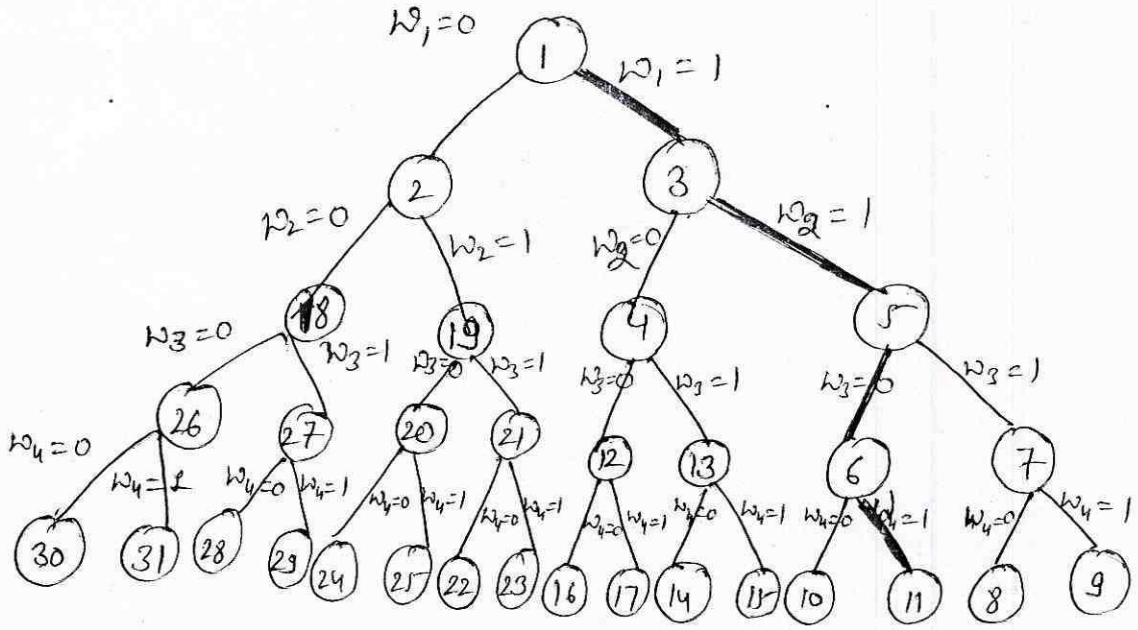
→ To find the soln of a given problem we make use of backtracking algor by means of three representations. For this purpose we follow fixed-size binary soln approach.

Sum of subset problem :

The maximum subset sum problem has a wide range of important applications. Example fields are material, transportation, television and radio industries and computer system resource allocation. In addition, the problem also models static job scheduling in a multiprogrammed parallel system. In such a system, there are m identical processors. Assume that we are given a set S of n jobs $J_1, J_2, \ldots J_n$, where job j_i requires x_i processors for parallel execution. The problem here is to find a subset S' of jobs to execute simultaneously in such a way that system utilization is maximized. Furthermore, we try to schedule as many jobs as possible to maximize system throughput.

A vendor car has capacity K kg. There are some bundles having respective weights $c_1, c_2, \ldots c_n$ kg. which are to be transported by that vendor car. The problem is to pick up those bundles and load them in the car so that the car capacity is maximum utilized, if not fully.

In a restaurant, where a person has to choose K course, without surpassing the amount of C calories, his diet prescribes. Assuming that here are N_i dishes to choose among for each course $i = 1, \ldots K$, and w_{ij} is the nutritive value while P_{ij} is a rating saying how well each dish tastes. Then an optimal meal may be found

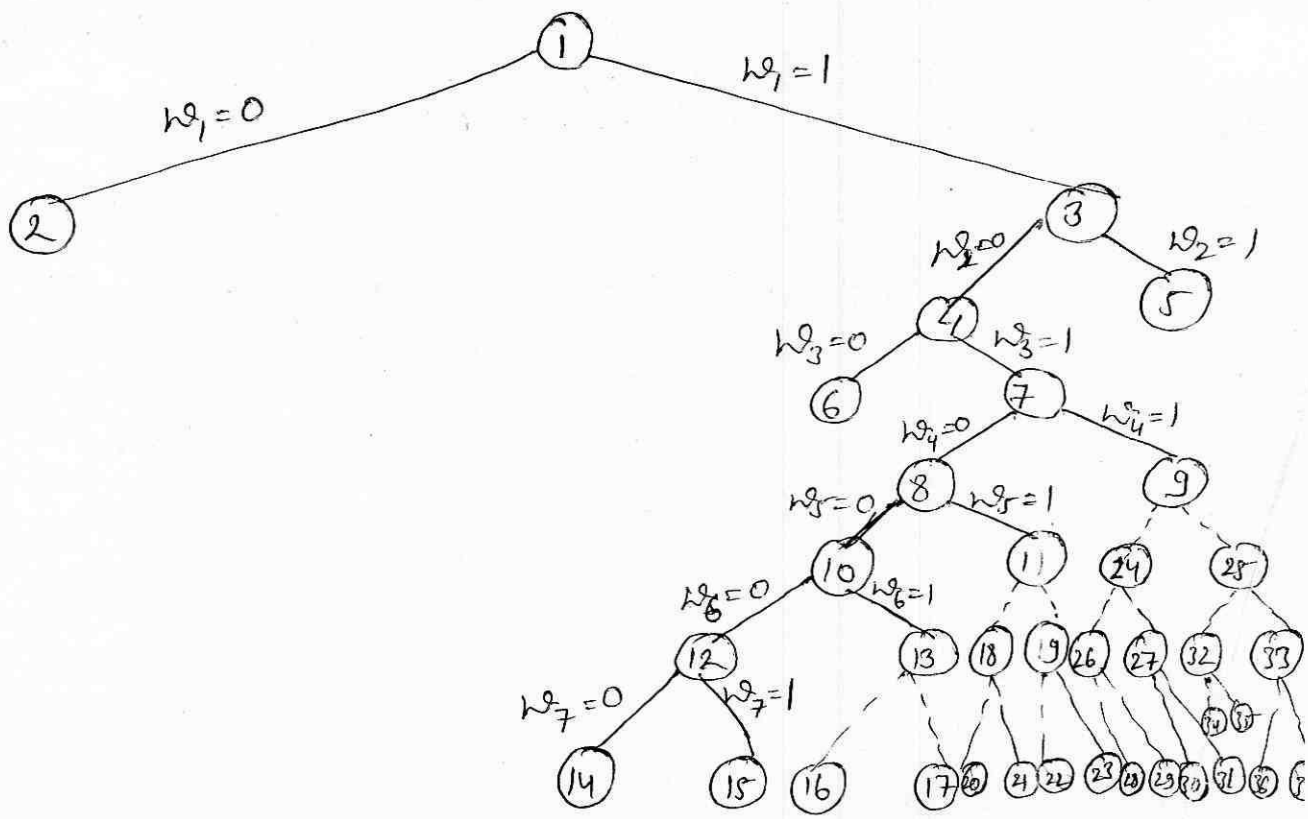


ex $W = (5, 7, 10, 12, 15, 18, 20)$ $km = 35$

Subsets of W which makes the sum 35 are

$(5, 10, 20)$, $(5, 12, 18)$ and $(7, 10, 18)$

fixed size : $(1,0,1,0,0,0,1)$, $(1,0,0,1,0,1,0)$ & $(0,1,1,0,0,1,0)$



14

We initiate from node 1 & expand it to nodes 2 & 3. If we select this node then we will move to node 3 of tree. with wt. of first node that is 11. Now, if we select the second node also by making $w_2 = 1$ then we move to node 5 with weight = 24. Now continuing in the same manner, If we select the node 5 by making $w_3 = 1$ then we have the total wt. 48, which is greater than 31.
So, we discard this edge & move ~~backward~~ backward to node 5, so we skip the third item by making $w_3 = 0$ & move to forth item i.e. node 6. At this point we have total wt. of 24. Now if we ~~sel~~ seled this forth item by making $w_4 = 1$ then we have total wt = 31 so this is one of the valid sol^n.

In this tree representation the sol^n vector of the above problem includes the nodes 1, 3, 5, 6, 11. This sol^n vector is shown by bold lines in the search tree. Similarly other sol^n can be found out by tracing the search tree.

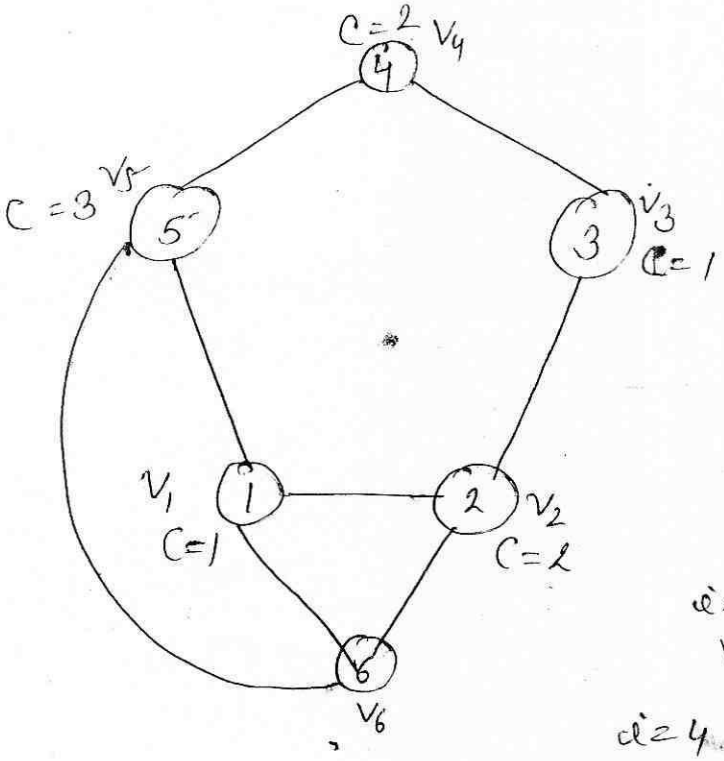
Graph Coloring Problem:

* A vertex coloring of an undirected graph $G(V,E)$ is a labeling of the vertices such that no edge in E has two end points with same color
* Optimization problem is coloring the vertices with min^m no. of coloring.
* Coloring is to be ~~chosen~~ ~~in~~ done in such a way that no two adjacent nodes are colored with same color with only 'm' colors given.
* Also called - m - colorability decision problem

Algorithm:

```
Vertex color ( G(V,E))
{
   for (i=1 to |V|)
   {
      c=1 ;
      while ( ∃ a vertex adjacent to vi with color c) do
      {
         c = c+1 ;
      }
      Label vi with color c ;
   }
}
```

16



$C = 1$ for $i = 1$

$V_1 = C = 1$

$i = 2$

$V_2 = C = C+1 = 2$

$i = 3$

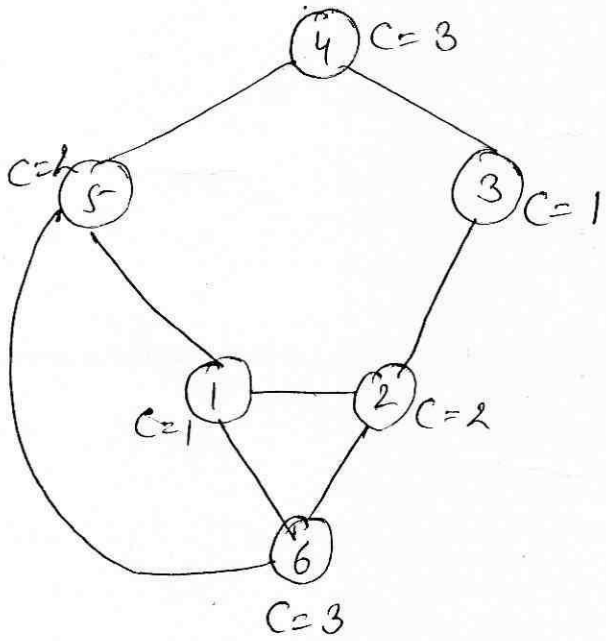
$V_3 = C = 1$

$i = 4$

$C = 1, \quad C = C+1 = C = 2$

~~$i = 5$~~

~~$C = 1, \quad C = 3$~~



$m = 3$ colors

$\exists$ → there exists

UNIT : 4

Randomized Algorithms

So far we have seen only deterministic algorithms. They produce same output on same input always.

But there are some algorithms that rely on chances.

"A randomized Algorithm is an algorithms that is allowed to access independent, unbiased random bits, and then allowed to use these random bits to influence its computation."

So we can say that randomized Algorithms belongs to the category of Probabilistic Algos.

Why to use Randomized Algorithms

Some Algos have good best case & Average Case complexity but very bad worst case comp

for eg - Quicksort

~~Best case~~ Its Avg case = $O(n \log n)$

Worst case $O(n^2)$

which is very bad as compared to Avg Case

So to improve worst case like in Quicksort in randomized Algos we will choose random value of pivot instead of first or last element which can lead us very close to average case.

- Thus we use randomized Algorithms.

Randomized Algorithm can be of two types —

① Las Vegas Algorithms

② Monte Carlo Algorithms.

Las Vegas Algorithms

→ The randomized Algorithm that Always produce the correct output for the same input [random bits can be chosen for every run] then these are known as Las Vegas Algorithms.

→ The runtime of such Algorithms is random as it depends on the output of a randomizer.

If we are lucky, the algorithm might terminate fast, else it can run for a longer period of time.

→ It does not gamble with the correctness of the result, it only gambles with the resources used for the computation.

→ ~~for~~ all randomized Algorithm we characterize them by $O(\cdot)$ notation but for Las Vegas algorithm we characterize it by $\tilde{O}(\cdot)$ notation.

→ ~~It uses~~ for Quick sort, it uses a random function which is used to choose pivot.

Random(x, y) [no to be choosen as pivot b/w x & y]

if x < y

return (x + (rand() * (x − y)));

JECRC Foundation

JECRC Foundation

Give Blood. Save Life. Be a Hero

8 October, 2015

Blood Donation 2015 Registration Slip

JECRC

Name of Donor...

Branch .... Section.......................................

Semester...............................

Email ID...

Mobile No...

Blood Group .... Signature

JECRC Jaipur Engineering College and Research Centre | JECRC University | JECRC UDML College of Engineering

JECRC Foundation

JECRC Foundation

Give Blood. Save Life. Be a Hero

8 October, 2015

Blood Donation 2015 Registration Slip

JECRC

Name of Donor...

Branch .... Section.......................................

Semester...............................

Email ID...

Mobile No...

Blood Group .... Signature

JECRC Jaipur Engineering College and Research Centre | JECRC University | JECRC UDML College of Engineering

JECRC Foundation

JECRC Foundation

Give Blood. Save Life. Be a Hero

8 October, 2015

Blood Donation 2015 Registration Slip

JECRC

Name of Donor...

Branch .... Section.......................................

Semester...............................

Email ID...

Mobile No...

Blood Group .... Signature

JECRC Jaipur Engineering College and Research Centre | JECRC University | JECRC UDML College of Engineering

JECRC Foundation

JECRC Foundation

Give Blood. Save Life. Be a Hero

8 October, 2015

Blood Donation 2015 Registration Slip

JECRC

Name of Donor...

Branch .... Section.......................................

Semester...............................

Email ID...

Mobile No...

Blood Group .... Signature

JECRC Jaipur Engineering College and Research Centre | JECRC University | JECRC UDML College of Engineering

JECRC Foundation

JECRC Foundation

Give Blood. Save Life. Be a Hero

8 October, 2015

Blood Donation 2015 Registration Slip

JECRC

Name of Donor...

Branch .... Section.......................................

Semester...............................

Email ID...

Mobile No...

Blood Group .... Signature

JECRC Jaipur Engineering College and Research Centre | JECRC University | JECRC UDML College of Engineering

JECRC Foundation

JECRC Foundation

Give Blood. Save Life. Be a Hero

8 October, 2015

Blood Donation 2015 Registration Slip

JECRC

Name of Donor...

Branch .... Section.......................................

Semester...............................

Email ID...

Mobile No...

Blood Group .... Signature

JECRC Jaipur Engineering College and Research Centre | JECRC University | JECRC UDML College of Engineering

JECRC Foundation

JECRC Foundation

Give Blood. Save Life. Be a Hero

8 October, 2015

Blood Donation 2015 Registration Slip

JECRC

Name of Donor...

Branch .... Section.......................................

Semester...............................

Email ID...

Mobile No...

Blood Group .... Signature

JECRC Jaipur Engineering College and Research Centre | JECRC University | JECRC UDML College of Engineering

JECRC Foundation

JECRC Foundation

Give Blood. Save Life. Be a Hero

8 October, 2015

Blood Donation 2015 Registration Slip

JECRC

Name of Donor...

Branch .... Section.......................................

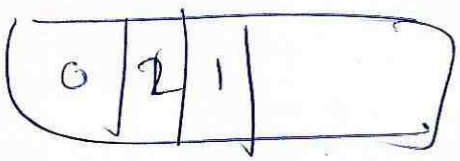
Semester...............................

Email ID...

Mobile No...

Blood Group .... Signature

JECRC Jaipur Engineering College and Research Centre | JECRC University | JECRC UDML College of Engineering



k=1

i=2 to n

r=1

@ 2>2 x

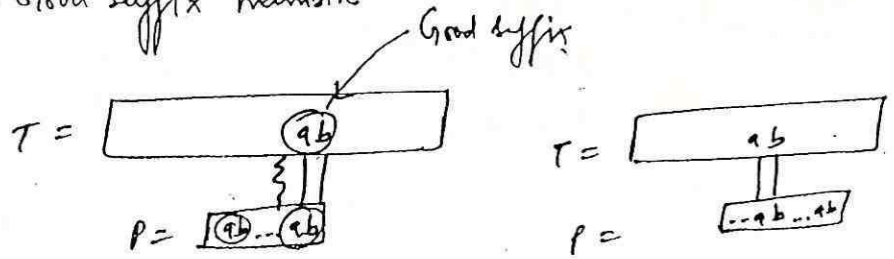
2 ≤ 2 2>1

1≥2

q=1 ; q≥2 ; q--

J[2]=J[1]

* Good Suffix heuristic



GSH(P, m)

1. π = Prefix(P)
   P' = Reverse(P)
   π' = Prefix(P')
2. for j = 0 to m
   $\gamma[j] = m - \pi[m]$
3. for k = 1 to m
   (a) j = m − π'[k]
   (b) if $\gamma[j] > k - \pi'[k]$
   $\gamma[j] = k - \pi'[k]$
4. return γ

Boyer Moore PMA (T, P, Σ)

1. n = length[T]
   m = length[P]
   bad character latest occurrence → λ = BCH (P, m)
   γ = GSH (P, m)
2. for i = 0 to n − m
   (a) j = m
   (b) while j > 0 and P[j] == T[i+j]
   j--
   (c) if j == 0
   print "pattern" i
   i = i + γ[0]
   else
   i = i + max{γ[j], j − λ[T[i+j]]}
3. Exit.

* Out of BCH and GSH, whichever provides higher shift is selected

abcdea

d[a] = 5
d[d] = 4
d[b] = 2

Brute force algorithm

Naive PMA (T, P)

Text size → n = length[T]

Pattern size → m = length[P]

2. for i = 0 to n-m

   if P[1 ... m] = T[i+1 ... i+m]

   print "Pattern occurs at shift" i

3. Exit.

* Time complexity. O (n-m+1) m

** Preprocessing of string.
** Comparison with shifting pattern

· Mismatch occurs.
· Pattern moved one position to the Right .... with Respect to text.

Left to Right

T = abcd 4
P = ab 2

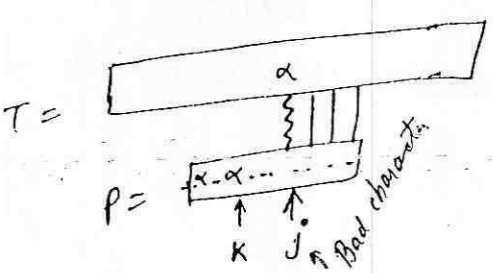
P[1 ... 2] ? T[1 ... 3]

a	b	c	d

Boyer Moore P.M.A →

* Boyer-Moore PMA works on two heuristics –
  (i) Bad character heuristic
  (ii) Good suffix heuristic.
* B.M. algorithm runs on average linear time.
* B.M. algo^r is good when Pattern is Long.
* Bad character heuristic

Right to left
Pattern match.



* Bad character provides a shift of j-K
* if j-K is -ve then ignore BCH
* if K=0 (no bad character present in pattern) a safe shift of j can be given.

BCH (m ← length[P]

1. for every character c in the alphabet set |Σ|

   λ[c] = 0 (last occurrence of each character)

2. for i = 1 to m

   λ[P[i]] = i

3. Exit

Dynamic Programming :-

* Dynamic programming method work like 'Divide & Conquer' method i.e. solves the problem by combining the sol^n to subproblems.
* Divide & Conquer work in the top down manner, it devide the problem into subproblems, find sol^n to all sub-problems and then combine them to find the sol^n for oregenal problem i.e. works on subproblems independently.
* While dynamic programming work on Bottom up approach, subproblems are dependent.
* It solves subproblems once, store their sol^n in some table so that it can be reused by other subproblems.

Matrix chain multiplication :

* In this problem, given a chain of n matrices $(A_1, A_2 \ldots A_n)$ to be multiplied. the matrices required to be fully parenthesized to resolve ambiguities.
* The products of the parenthesization is same where many ways of parenthesization are possible. we have to find the optimal one.
* Let there are 3 matrices (A_1, A_2, A_3) : 2 ways of parenthesizing $(A_1 (A_2, A_3))$ and

$((A_1, A_2) A_3)$

2

* The different ways of parenthesizing chain of matrices have different costs involved.
* One with minimum cost is the optimal.
* for ex: 3 matrices A_1, A_2, A_3 with dimensions 2×3, 3×4 & 4×3.

→ To multiply 2 matrix they should be compatible, i.e. columns of first matrix should be same as the row of second.

If $A_1 = P \times q$

$A_2 = q \times r$

then product = $P \times r$ (dimension of resultant product matrix)

and computation time will be no. of scalar multiplication which is $P \times q \times r = Pqr$

→ for above ex, if matrices are parenthesized as

$(A_1 (A_2 A_3))$

Scalar multiplications for $A_2 A_3 = 3\times4\times3 = \boxed{36}$

And the dimensions of resultant matrix B will be 3×3 & Scalar multiplications for $A_1 B$ is $2\times3\times3 = \boxed{18}$.

So total no. of Scalar multiplication = $36+18 = \boxed{54}$

If matrices are parenthesized as

$((A_1, A_2)\ A_3)$

Scalar Mul. for $A_1 A_2 = 2\times3\times4 = 24$

dimension of resultant matrix $B = 2\times4$

Scalar mul. for $B A_3 = 2\times4\times3 = 24$

Total scalar mul. = $24+24 = \boxed{48}$

Algorithm:

Matrix_chain_order

Step 1: The number of matrices to be multiplied is n where

$n \leftarrow$ length[P] − 1

Step 2: For single matrix, scalar multiplications are zero, so,

For $i \leftarrow 1$ to n

Set $m[i, i] := 0$

Step 3: For sub-problems of matrix chain length l, do following steps

For $l \leftarrow 2$ to n

do for $i \leftarrow 1$ to $n-l+1$

Set $j \leftarrow i+l-1$

$m[i, j] \leftarrow \infty$ [End of inner for loop]

[End of outer for loop]

for $k \leftarrow i$ to $j-1$

Set $q \leftarrow m[i, k] + m[k+1, j] + P_{i-1} P_k P_j$

if $q < m[i, j]$ Then:

$m[i, j] \leftarrow q$

$s[i, j] \leftarrow k$

[End of if stn] [End of innermost for loop] [End of Outer j loop]

[End of Outer l loop]

Step 4: Return the auxiliary-table (arrays) m & s

Step 5: Exit.

$m[i, j] \leftarrow$ Cost of multiplying $A_1 \dots A_k$ + Cost of multiplying $A_{k+1} \dots A_j$ + Cost of multiplying 2 resultant matrices.

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k)\, P(n-k) & n \geq 2 \end{cases}$$

{ To calculate the no. of Paranthesis

eg:- $A_1 \begin{bmatrix} A_1 & A_2 & A_3 \\ 0 & \infty & \infty \\ \infty & 0 & \infty \\ \infty & \infty & 0 \end{bmatrix}$ Let $n = 3$ then $P(3) = P(1)P(2) + P(2)P(1)$

$= 1 + 1 = \boxed{2}$ Ans

4

Algorithm

PRINT_OPTIMAL_PARENS (S, i, j)

[Given the table s with indices i, j]

Step1: If there is a single matrix, i.e.

i = j Then:

print "A_i"

Step2: If $i \neq j$ Then:

print "("

recursively call PRINT_OPTIMAL_PARENS(S, i, S[i, j])

PRINT_OPTIMAL_PARENS (S, S[i, j] + 1, j)

Print ")"

Step3: Exit

Example 1: Find an optimal parenthesization of a matrix chain product whose sequence of dimensions is

$\langle 5, 10, 3, 12, 5, 50, 6 \rangle$

$\langle 5\times10, 10\times3, 3\times12, 12\times5, 5\times50, 50\times6 \rangle$

Soln

(1) $P_0 = 5$, $P_1 = 10$, $P_2 = 3$, $P_3 = 12$, $P_4 = 5$, $P_5 = 50$, $P_6 = 6$

So the length of matrix chain is length[P] = 7

So n = length[P] − 1 = 6

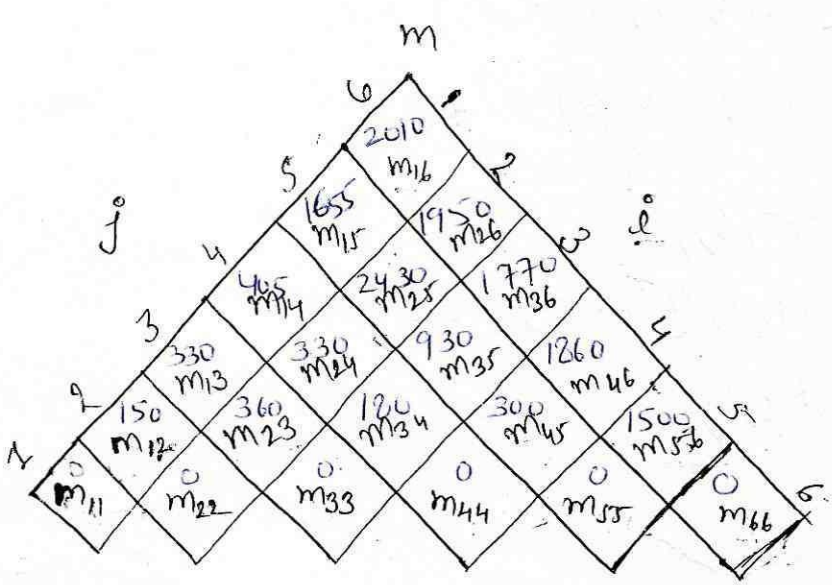
So there are 6 matrix present.

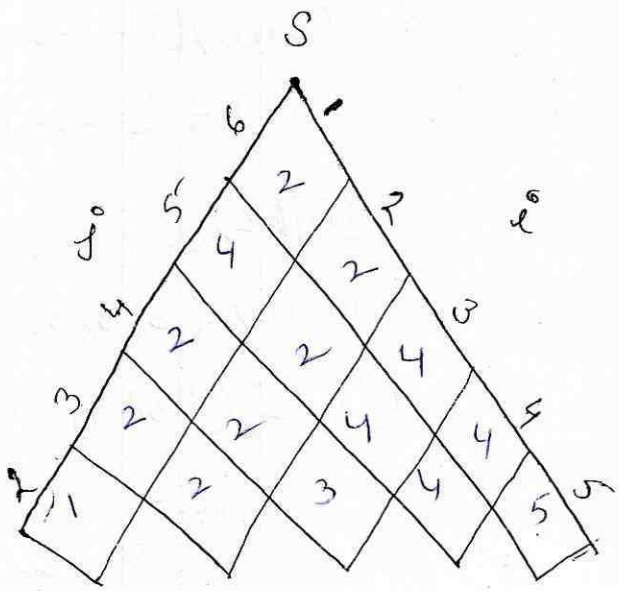
(2) Now we have to construct the table m and S.

(3) We are considering the table only above the main diagonal for ($i \le j$)

If $i = j$ then: $m[i,j] = 0$

So $m_{11} = m_{22} = m_{33} = m_{44} = m_{55} = m_{66} = 0$





(4) In all other cases in table we can see $i < j$

So it initially $m[i,j] = \infty$ for $i < j$

Now we have to calculate minimum value of $m[i,j]$ and put that value in table m.

$$m_{12} = \min_{\substack{i \le k < j \\ 1 \le k < 2}} \{ m[i,k] + m[k+1, j] + P_{i-1}, P_k, P_j \}$$

So $1 \le k < 2$ for only $k = 1$

$m_{12} = m[1,1] + m[2,2] + P_0, P_1, P_2$

$= 0 + 0 + (5 \times 10 \times 3)$

$m_{12} = 150$ for $k = 1$

So $S[1,2] = k = 1$

$i = 1, j = 6$

So print "("

P_O_P(S, 1, S[1,6])

P_O_P(S, S[1,6]+1, 6)

print ")"

⇒ ((S,1,2)(S,3,6))

⇒ (((S,1,1)(S,2,2)) ((S,3,4)), (S,5,6))

(S,1,1) & (S,2,2), $i = j$ so print "A_1"

⇒ $((A_1A_2)((S,3,4)(S,5,6))$

⇒ $((A_1A_2)((S,3,3)(S,4,4)((S,5,5)(S,6,6))$

⇒ $((A_1A_2)((A_3A_4)(A_5A_6)))$

Similarly

m_{23} = for $2 \le k < 3$ for $k = 2$ only one time

$m_{23} = m[2,2] + m[3,3] + P_1 P_2 P_3$

$m_{23} = 0 + 0 + (10 \times 3 \times 12)$

$\boxed{m_{23} = 360}$ for $k = 2$

so $\boxed{S[2,3] = 2}$

like this.

$m[3,4] = 180$, $k = 3$, $S[3,4] = 3$

$m[4,5] = 3000$, $k = 4$, $S[4,5] = 4$

$m[5,6] = 1500$, $k = 5$, $S[5,6] = 5$

Now for $m[1,3]$

for $m[1,3]$ we have to select minimum of 2 values.

for $i \le k < j$

$1 \le k < 3$ so k will be for $k = 1$, $k = 2$

Similarly for $k = 1$, $m[1,3] = 960$

$k = 2$, $m[1,3] = 330$

$\boxed{\text{so } m[1,3] \text{ is min for } k = 2}$

for $m[1,4]$ we have to select min of 3 values

for $i \le k < j$

$1 \le k < 4$ so k will be for $k = 1$, $k = 2$, $k = 3$

for $k = 1$, $m[1,4] = 580$

$k = 2$, $m[1,4] = 405$

$k = 3$, $m[1,4] = 630$

so $\boxed{m[1,4] \text{ is min for } k = 2}$

Longest Common Subsequence (LCS) :-

* LCS deals with strings with the comparison of strings.
* Let two string S1 and S2 each consisting of sequence of characters.

S1 = ABCAABBAD

S2 = DBDABCAD

S1 =

A	B	C	A	A	B	D	A	D

S2 =

D	B	D	A	B	C	A	D

S3 =

B	A	B	A	D

is LCS ABCAD

* Application of LCS is in biological issues like comparing the DNA of two organisms.

Algorithm LCS print :-

Table 'b' is the o/p, X is one of the sequence. Initially i = m and j = n where m and n are lengths of sequence X and Y.

Step 1: set $i := m$ or $j := n$ ~~there is no LCS~~

Step 2: If b[i,j] = "↖" then

recursively call print LCS with

$i \leftarrow i - 1$ and

$j \leftarrow j - 1$

print x_i

elseif b[i,j] = "↑" then

recursively call print_LCS with

$i \leftarrow i-1$

else

recursively call print LCS with

$j \leftarrow j-1$

Step3: Exit

Algorithm: LCS length

Given the sequences X[1...m] and Y[1....n]

Step1: Assign the lengths of x and y to variables.

set $m \leftarrow$ length [X]

$n \leftarrow$ length [Y]

Step2: If the length of one of the sequence is 0, then

C[i,j] = 0

for $i \leftarrow 0$ to m

set $c[i,0] \leftarrow 0$

for $j \leftarrow 0$ to n

~~Set c[i,0]~~

set $c[0,j] \leftarrow 0$

Step3: for $i \leftarrow 1$ to m

do for $j \leftarrow 1$ to n

do if $x_i = y_i$ then:

Set $c[i,j] \leftarrow c[i-1, j-1]+1$

M N O M

else if $c[i-1, j] \geq c[i, j-1]$ then:

set $c[i,j] \leftarrow c[i-1,j]$

and $b[i,j] \leftarrow$ "↑"

else

set $c[i,j] \leftarrow c[i,j-1]$

And $b[i,j] \leftarrow$ "←"

Step 4: Return tables c and b and Exit.

Example:

$X = < M\ L\ N\ O\ M >$

$Y = < M\ N\ O\ M >$

$m = 5$, $n = 4$

Soln: putting one of the sequence is of 0 length

$c[i,j] = 0$ at Index 0 of both of i and j.

		0	1	2	3	4
	i / j	y_j	M	N	O	M
0	x_i	0	0	0	0	0
1	M	0	①	←1	←1	↖1
2	L	0	↑1	↑1	↑1	↑1
3	N	0	↑1	②	←2	←2
4	O	0	↑1	↑2	③	←3
5	M	0	↖1	↑2	↑3	④

~~M O~~ MNOM

(1) $x_1 = y_1 = m$

so : $C[1,1] = C[0,0] + 1 = 1$

and $b[1,1] =$ "↖"

(2) $x_2 \neq y_2$, $x = m$, $y = n$

$C[1,2] = \max\{ C[1,1], C[0,2] \}$

$= C[1,1] = 1$

$b[1,2] =$ "←"

Similarly other values of $C[i,j]$ and $b[i,j]$ can be computed. The last value of $C[i,j]$ i.e.

$C[5,4] = 4$ which means LCS is of length (4)

---

1) Divide & Conquer Algorithm :- It devides the problem into subproblems.

2) Greedy Algorithm :- Greedy algo attempt not only to find a sol^n but to find the ideal sol^n to any given problem.

3) Branch & Bound Algorithm : Branch & Bound algo form a tree of subproblems to the primary problem, following each branch until it is either solved or ~~dumped~~ dumped in with another branch.

4) Dynamic Programming Algo :- This remembers older results and attempt to use this to speed the process of finding

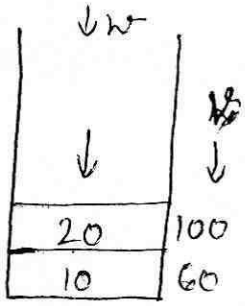
new result

0/1 Knapsack problem:

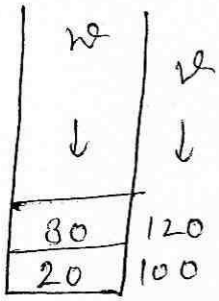
* We are given items with their weights and profit. pairs are (10, 60), (20, 100) & (30, 120) respectively.

Capacity of knapsack $W = 50$

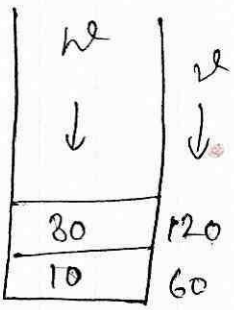
We have 3 options to fill the knapsack



Knapsack capacity = 30
" value = 160



50
220



40
180

So, It is clear that by filling the knapsack with items I_2 & I_3, profit is maximum & knapsack is full. Using Dynamic programming method we can solve this 0/1 knapsack problem optimally.

To solve the knapsack problem (0/1) we use the following algorithm which takes as I/P the max^m weight W, no. of items n and their value of weighted sequences as

$V = \langle v_1, v_2, v_3, \ldots v_4 \rangle$ and $w = \langle w_1, w_2 \ldots w_4 \rangle$

and given as O/P the 2-D matrix $C[i, j]$ which contains (item, weight) pairs in row major order. At the end of Computation $C[n, w]$ contains the max^m value that can be put into the knapsack.

n → item
w → wt.

Algorithm 0/1 _ Knapsack

[Take as input the maximum weight W, n items and profit value v and weights w.]

Step1: If i:=0 or w:=0, solution array contains '0'.

```
for w ← 0 to W
    Set c[0,w] ← 0

for i ← 1 to n
    Set c[i,0] ← 0
```

Step2: Check if item i can be added to the knapsack or not.

```
for i ← 1 to n
    dofor w ← 1 to W
        if wi ≤ w then:
            if (vi + c[i-1, w-wi] > c[i-1, w]) then:
                c[i,w] ← vi + c[i-1, w-wi]
            else:
                c[i,w] ← c[i-1, w]
        [End of Inner If structure]
    else:
        c[i,w] ← c[i-1, w]          //(wi > w)
    [End of outer if str]
```

Step3: Return table c and Exit

Now the optimal solution is computed by backtracking the table c starting from c[n, w].

Algorithm: Find_knapsack

Step1: Set $i \leftarrow n$

Set $w \leftarrow W$

Step 2: while ($i > 0$ and $w > 0$)

do if $c[i, w] = c[i-1, w]$ then:

item i not the part of knapsack

Set $i \leftarrow i-1$

$w \leftarrow W$

else:

item i part of knapsack

Set $i \leftarrow i-1$

$w \leftarrow w - w_i$

[End of while loop]

Step3: Exit

Example:

Consider a knapsack having weight capacity $W = 6$ and no. of items $n = 3$ such that

$$(w_1, w_2, w_3) = (2, 3, 3)$$

& $(v_1, v_2, v_3) = (1, 2, 4)$

Soln $W = 6$, $n = 3$

cost matrix

i \ w	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1
2	0	0	1	2	2	3	3
3	0	0	1	4	4	5	6

(1) $i = 1$, $w = 1$, $w_i = 2$

if $2 \leq 1$ f if $(w_i \leq w)$

so $c[1,1] \leftarrow c[0,1] = 0$

$w = 2$

if $2 \leq 2$ T

if $(w_i + c[0,0]) > c[0,2]$

if $2 > 0$ T

$c[1,2] \leftarrow 2$

$w = 3$

if $2 \leq 3$ T

if $(1 + c[0,1]) > c[0,3]$

if $1 > 0$ T

$c[1,3] \leftarrow 1$

$w = 4$

if $2 \leq 4$ T

if $(1 + c[0,2]) > c[0,4]$

$1 > 0$ T

$c[1,4] \leftarrow 1$

16

$w = 5$

if $2 \leq 5$ T

if $(1 + C[0,3]) > C[0,5]$

if $1 > 0$ T

$C[1,5] \leftarrow 1$

$w = 6$

if $2 \leq 6$ T

if $(1 + C[0,4]) > C[0,6]$

if $1 > 0$ T

$C[1,6] \leftarrow 1$

(2) ⇒ Now $\boxed{i = 2}$, $w_2 = 3$

$w = 1$

if $w_2 \leq w$

$3 \leq 1$ F

$C[2,1] \leftarrow C[1,1]$

$C[2,1] \leftarrow 0$

$w = 2$

if $3 \leq 2$ F

$C[2,2] \leftarrow C[1,2]$

$C[2,2] \leftarrow 1$

$w = 3$

if $3 \leq 3$ T $\boxed{v_2 = 2}$

if $(v_2 + C[2-1, 3-3]) > C[2-1,3]$

if $(2 + C[1,0]) > C[1,3]$

if $2 > 1$ T

$C[2,3] \leftarrow 2$

$w=4$

if $3 \leq 4$ T

if $(2 + C[1,1]) > C[1,4]$

if $2 > 1$ T

$C[2,4] \leftarrow 2$

$w=5$

if $3 \leq 5$ T

if $(2 + C[1,2]) > C[1,5]$

if $2+1 > 1$ T

$C[2,5] \leftarrow 3$

$w=6$

if $3 \leq 6$ T

if $(2 + C[1,3]) > C[1,6]$

if $2+1 > 1$ T

$C[2,6] \leftarrow 3$

Same steps will be repeated for $i=3$

⇒ Now, find out the optimal sol^n, $C[n,w]$

Starting from $C[3,6]$

if $C[3,6] \neq C[2,6]$

$i=3$ So item 3 is part of knapsack

$w_3=3$ $i \leftarrow i-1 = 2$

$w \leftarrow w - w_i = 6-3=3$

~~Next~~ ~~C[3,3] ← C[~~

18:

$\Rightarrow$ Now $C[2,3] \neq C[1,3]$

$2 \neq 1$ T

item 2 part of knapsack

$i = 2$
$w_2 = 3$

$i \leftarrow 1$

$w \leftarrow 3 - 3 = 0$

Thus $w = 0$ and $i = 0$ and algorithm ends.

Finally items 2 & 3 in the knapsack with profit value $= 2 + 4 =$ 6 Ans.

example $n = 4, \quad W = 10$

$(w_1, w_2, w_3, w_4) = (5, 4, 6, 3)$

$(v_1, v_2, v_3, v_4) = (10, 40, 30, 50)$

Boyer_Moore_Matcher (T, P, Σ)

1. n ← length[T]
2. m ← length[P]
3. λ ← compute_Last_occurrence_func (P, m, Σ)
4. γ ← compute_Good_Suffix_func(P, m)
5. S ← 0
6. while S ≤ n-m
7. do j ← m
8. while j > 0 and P[j] = T[S+j]
9. do j ← j-1
10. if j = 0
11. then print "Pattern occurs at shift S".
12. S ← S + γ[0]
13. else S ← S + max(γ[j], j - λ[T[S+j]])

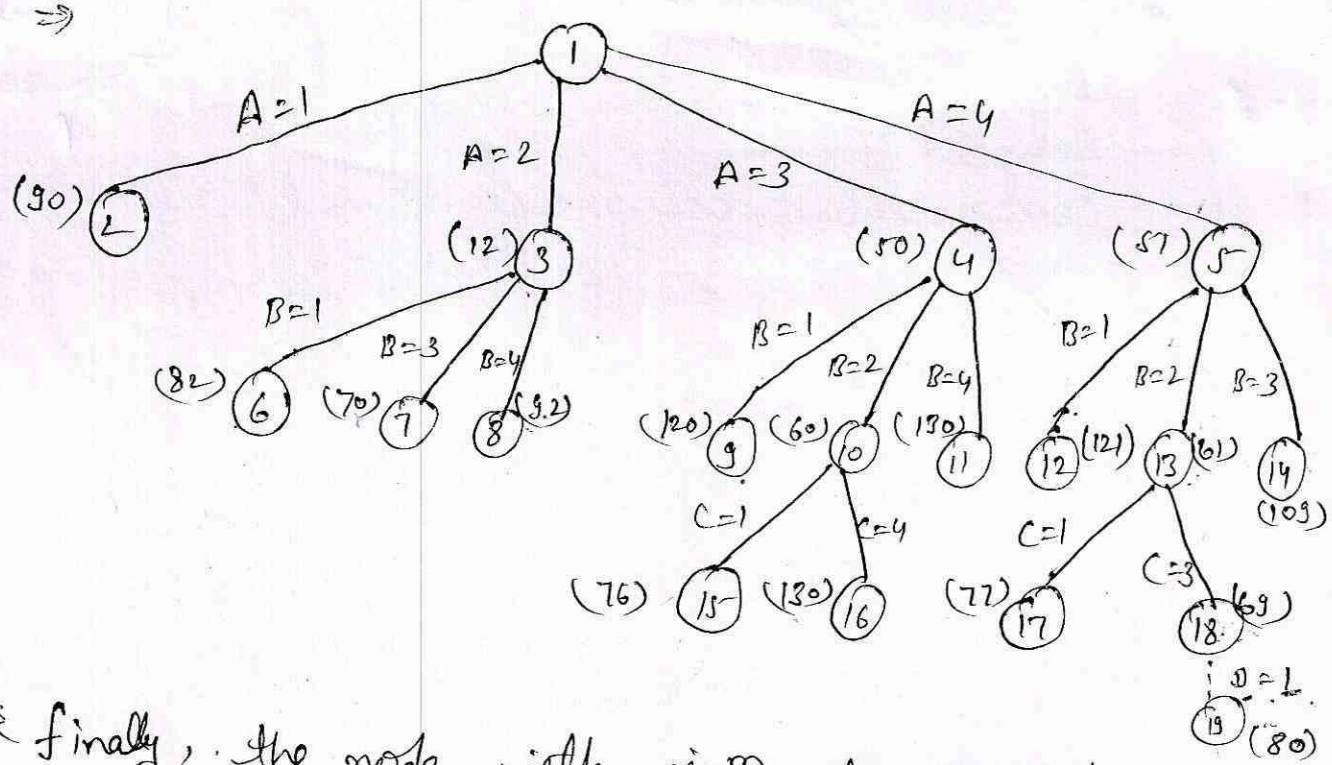
compute_last_occ

1. for each character a ∈ Σ
2. do λ[a] = 0
3. for j ← 1 to m
4. do λ[P[j]] ← j
5. return λ

Compute_Good_Suffix

1. π ← compute_prefix_func (P)
2. P' ← reverse(P)
3. π' ← compute (P')
4. for j ← 0 to m
5. do γ[j] ← m - π[m] [End of for loop]
6. for l ← 1 to m
7. do j ← m - π'[l]
8. if γ[j] > l - π'[l]
9. then γ[j] ← l - π'[l] [End of for loop]
10. return γ

⇒



* Finally, the node with min value is node 18. Exploring it we get node with person D assigned to job 1

Optimal cost is = 51 + 10 + 8 + 11

= 80

(path) assignment is :

$A \leftarrow 4, \quad B \leftarrow 2, \quad C \leftarrow 3, \quad D \leftarrow 1$

job 4 is assigned to agent A

job 2 " " " agent B

job 3 " " " agent C

job 1 " " " agent D

⇒ Quadratic Assignment Problem (QAP) :- describe a location problem.

Monte Carlo Algorithms

→ The algorithm whose output may differ from run to run is known as Monte Carlo Algorithms

→ These algorithms are used when it is infeasible or impossible to compute an exact result with a deterministic Algorithm

→ There can be diff. approaches but Genral steps are

⇒ Define ~~input~~ domain of inputs

⇒ Using probability distribution generate random inputs.

⇒ Perform a deterministic computation using the inputs

⇒ Aggregate the result of the individual component into a final result.

Primality Testing

To check whether the given no. n is prime or not

Fermet's theorem states that "If p is prime, then for every $1 \leq a < p$, $a^{p-1} \equiv 1 \pmod{p}$

This theorem will take lot of time as it will check for every no.

So we choose random no to increase prob.

$p=7$, $a=3$; $p=8$

$a^{p-1} \equiv 1 \pmod{p}$ — Pass — prime; fail — Composite

$a^{p-1} - 1 = pk$ $\quad k \in I$

$\frac{3^6 - 1}{7}$ $\quad 104$ $\quad 3^1$

M.C. Algo has two sub Types

↳ One Side Error :— If the Input is prime then it will always give correct answer but if not then there is a prob. that it can give wrong answer. So this is one sided error 50% chances Error on one side of decision

↳ Two Sided Error :— It gives error on both side of the decision.

Randomized Algorithm for Min-Cut Problem

To understand the problem, we need some terms

→ A cut of a connected graph is obtained by dividing the vertex set V of a graph G into two sets V_1 & V_2 such that

→ no common vertices in V_1 & V_2

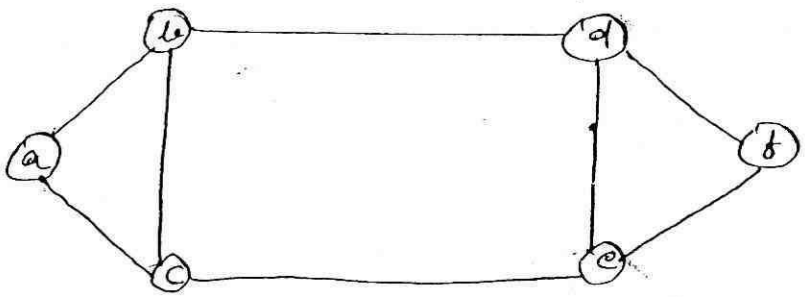
→ $V_1 \cup V_2 = V$

→ Any edge from V_1 to V_2 or V_2 to V_1 is said to be crossing the cut.

→ All the edges which cross the cut collectively form the cut-set of the graph.

This is for unweighted graph. For weighted graph we have different Algorithm.

Q1 - Find a minimal cut for the given graph using a randomized algorithm.



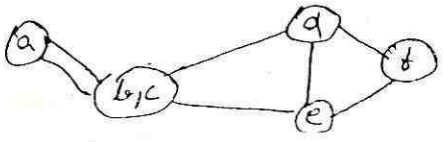
Sol - $V = \{a\}\{b\}\{c\}\{d\}\{e\}\{\}\}$

$G = \{(a,b), (b,c), (a,c), (b,d), (c,e), (d,e), (d,f)(e,f)\}$

Step 1 :- Select random edge

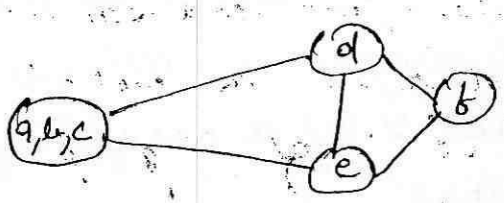
(b, c)

$V = \{a, \{b, c\}, d, e, f\}$

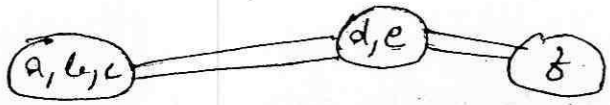


(a, c)

$V = [\{a, b, c\}, d, e, f\}$

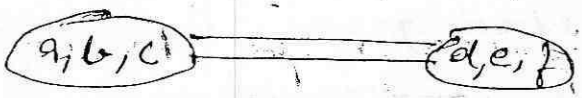


$V = [\{a, e, c\}, \{d, e\}, f\}$



(d, f)

$V = \{\{a, e, c\}, \{d, e, f\}$



Defination of Min cut of Graph:-

A min cut is the ~~minimum~~ division of graph into minimum no. of components making two vertices only.

Multicommodity flow

In practical Application we need to send more than one commodity (goods) from one place to other through same intermediate stations or routes. For such a representation, we use a multicommodity flow network.

Formal Definition

A multicommodity flow network is a directed graph $G(V, E)$ such that

⇒ For every edge $(u, v) \in E$, a capacity function $c(u, v)$ is associated.

⇒ There is a distinct set S of vertices called sources. $S = \{s_1, s_2, \dots s_n\}$

⇒ There is a distinct set T of vertices called sink. $T = \{t_1, t_2, \dots t_n\}$

⇒ There are n commodities $K_1, K_2, \dots K_n$ defined by $K_i = (s_i, t_i, d_i)$ where s_i is the source, t_i is the sink of commodity and d_i is the demand.

Properties of multicommodity flow n/w are :-

→ **Capacity constraints** :- Through each edge the total amount of various commodities flowing should not exceed its capacity i.e.

$$\sum_{i=1}^{n} f_i(u, v) \leq c(u, v)$$

→ Flow conservation :- For all $u \in V - \{s, t\}$ we require

$$\sum_{v \in V} f_i(u,v) = 0$$

and

Skew symmetry :-

$$u, v \in V, f_i(u,v) = -f_i(v,u)$$

→ Demand Satisfaction :- The objective of the multicommodity flow network is to flow each commodity from its source to its destination, as per the demand. Thus the amount of commodity i, leaving source s_i, reaching destination t_i is equal to the demand d_i, i.e.

$$\sum_{v \in V} f_i(s_i, v) = \sum_{v \in V} f_i(v, t_i) = d_i$$

Multicommodity Problems

① Minimum cost Multicommodity flow problem

There is a cost 'a' for sending flow on any edge (u,v). We need to minimize the total cost of flow of every commodity

$$\text{minimize} \sum_{(u,v) \in E} \left[a(u,v) \cdot \sum_{i=1}^{n} f_i(u,v) \right]$$

(2) Maximum Multicommodity flow problem

Total throughput is to be maximized.
i.e. flow the maximum for each commodity

$$\text{maximize} \sum_{i=1}^{} \sum_{v \in V} f_i(s_i, v).$$

(3) Maximum Concurrent flow problem

The task is to maximize the minimal fraction of the flow of each commodity to its demand.

Objective is $\min_{1 \le i \le n} \frac{\sum_{v \in V} f_i(s, v)}{d_i}$

Flow Shop Scheduling

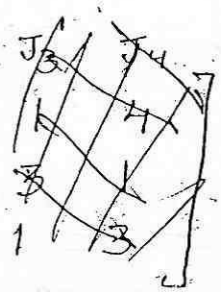
* Let say we have n jobs each requiring m tasks ~~each requiring m machines~~ $T_{1i}, T_{2i}, T_{3i} \cdots T_{mi}$ for $i = 1$ to n
* Task T_{ji} is to be performed on processor P_j for $j = 1$ to m
* ~~Tim~~ No processor may have more than one task at any time.
* Task T_{ji} cannot be started until task $T_{j-1,i}$ has been completed.

Two types

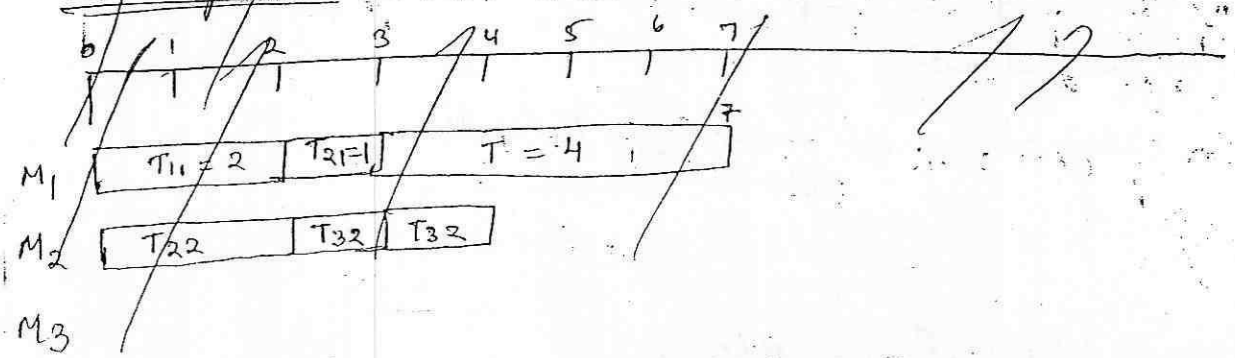
(1) Preemptive Scheduling :- A schedule in which the processing of a task on any processor is allowed to terminate before task is completed.

Non preemptive Scheduling :- A schedule in which the processing of a task on any processor is not terminated until the task is complete.

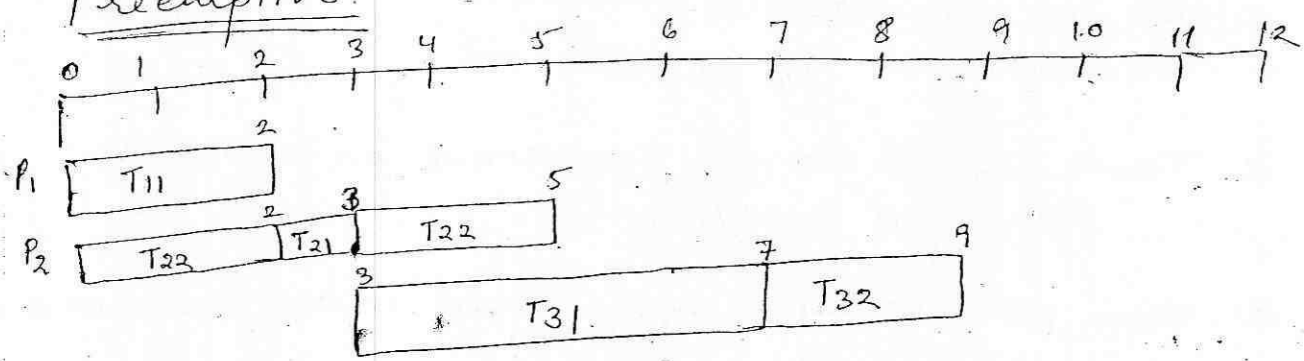
$$J = \begin{array}{c} \\ M_1 \\ M_2 \\ M_3 \end{array} \begin{bmatrix} J_1 & J_2 \\ 2 & 0 \\ 1 & 4 \\ 4 & 2 \end{bmatrix}$$



Preemptive

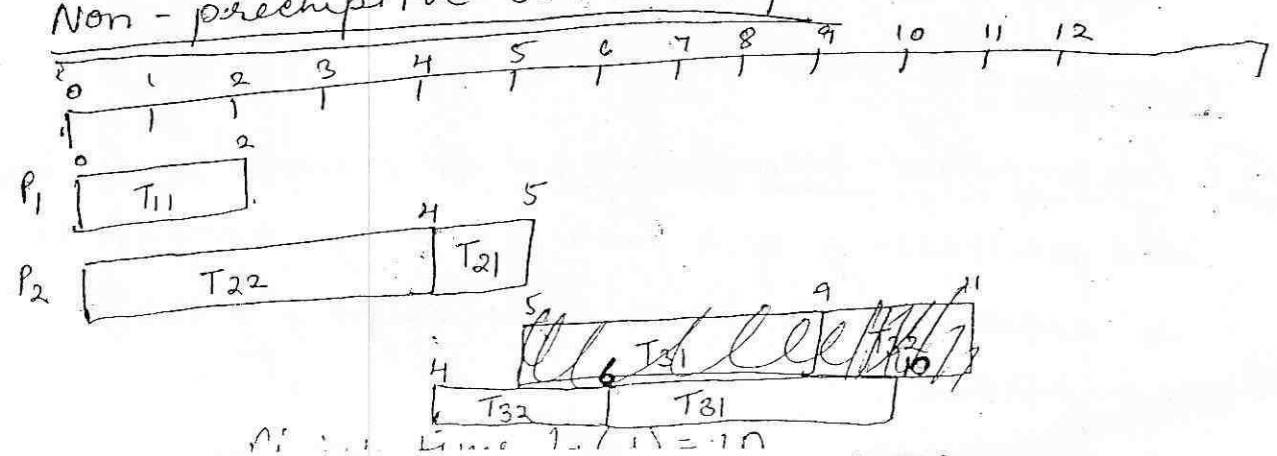


Preemptive



Finish Time $f_i(S) = 9$

Non-preemptive Scheduling



Max-flow min-cut theorem.

In a flow network, following conditions are equivalent -

1. f is a maximum flow in G.
2. The residual network contains no augmenting path.
3. $|f| = c(S,T)$ for some cut (S,T) of G

i.e. flow = cut capacity.

Algo

Ford-Fulkerson (G, s, t)

Step 1 for each edge $(u,v) \in E$

do $f(u,v) \leftarrow 0$

$f(v,u) \leftarrow 0$

Step 2 while there exists path p from s to t in Residual n/w

1) do $c_f(p) \leftarrow \min \{ c_f(u,v) \mid (u,v) \text{ is in } p \}$

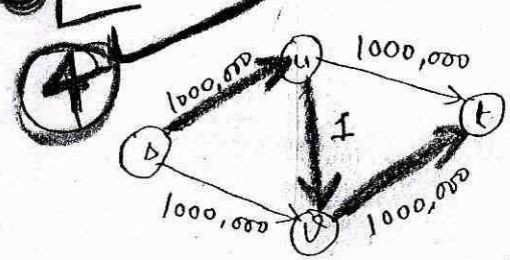
2) for each edge (u,v) in p

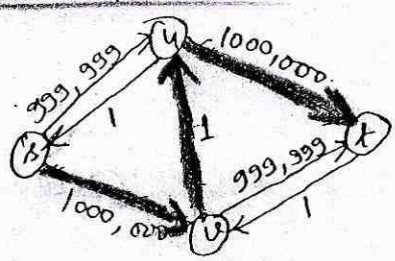
a) $f(u,v) \leftarrow f(u,v) + c_f(p)$

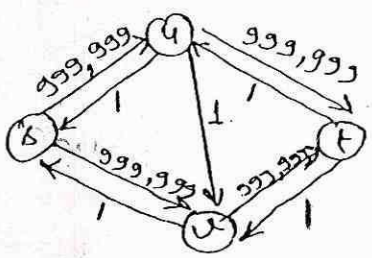
b) $f(v,u) \leftarrow -f(u,v)$

Step 3 End

T.C O [If ... net] is the maximum flow.







* The above situation can be avoid using Edmonds-karp algor.
* Edmond karp runs in $O(VE^2)$ time

Analysis → Book

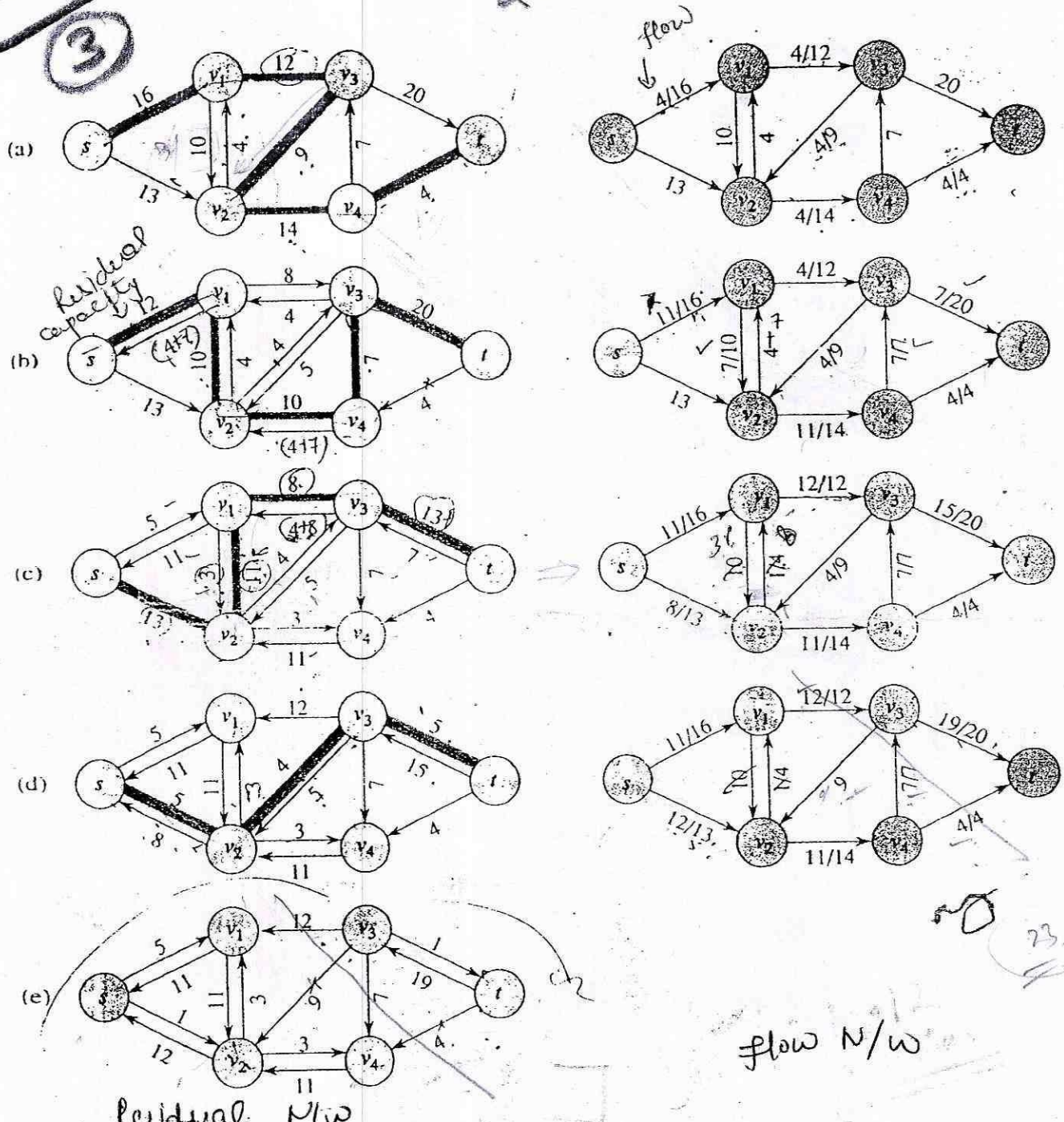


Figure 26.5 The execution of the basic Ford-Fulkerson algorithm. (a)–(d) Successive iterations of the while loop. The left side of each part shows the residual network G_f from line 4 with a shaded augmenting path p. The right side of each part shows the new flow f that results from adding f_p to f. The residual network in (a) is the input network G. (e) The residual network at the last while loop test. It has no augmenting paths, and the flow f shown in (d) is therefore a maximum flow.

Pattern Matching Algorithms:

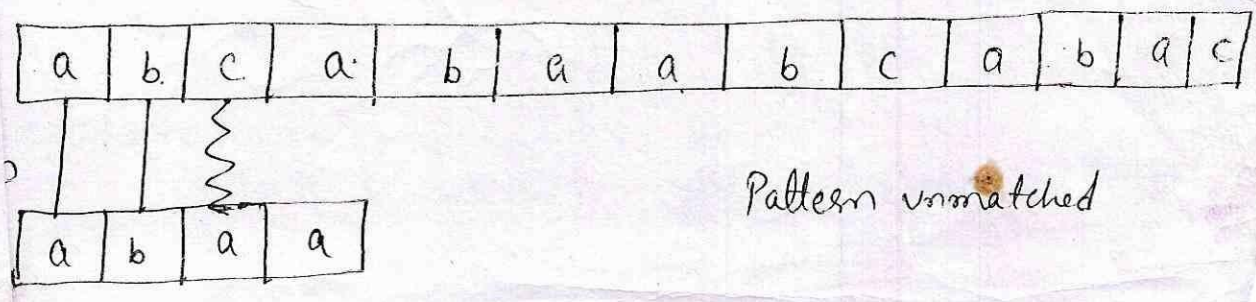
* Pattern matching is the way of comparing 2 or more text or array of characters. It is also called string matching.

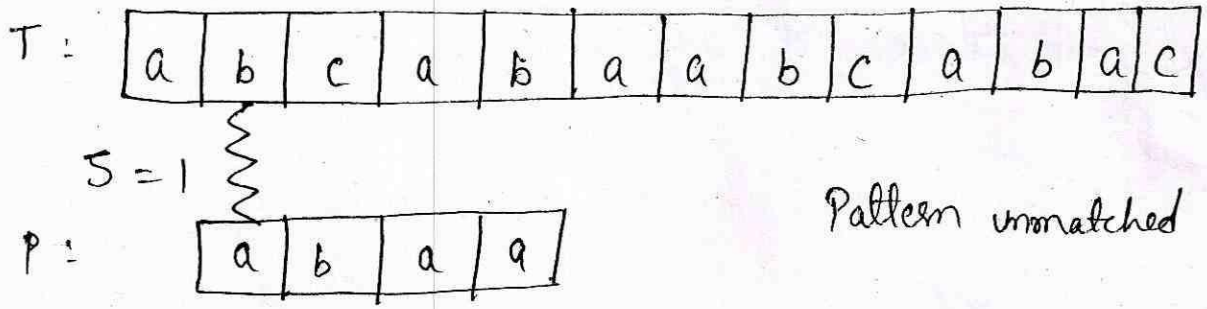
applications :- match the strands of DNA of 2 organisms & find how the similar the 2 organism are?

* Thus, the pattern matching problem can be formally defined as follows: Given the text and pattern to be matched where text is an array T[1...n] containing n characters and pattern is an array P[1...m] containing m characters, where $m \leq n$. These characters may contain $\Sigma = \{0,1\}$ or $\Sigma = \{a,b,\ldots,z\}$

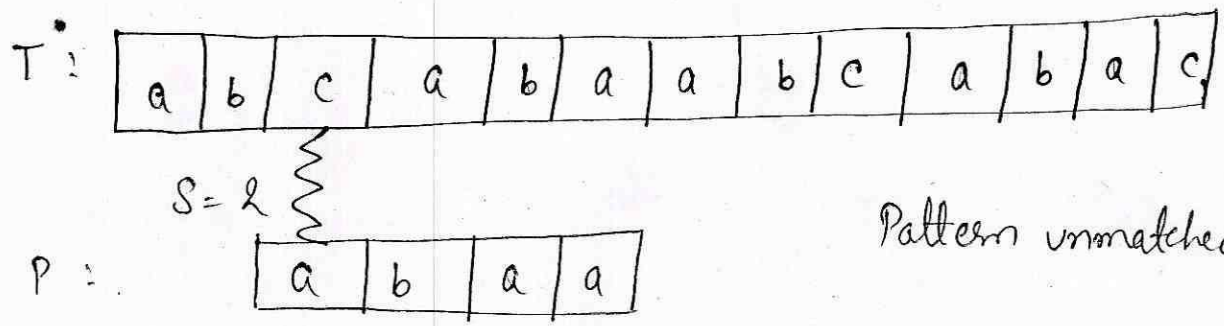
Starting from the first character, pattern P is matched with text T and is keep on shifted till either the pattern is found, or text is reached the end. A shift variable S is initially 0 and is keep on creasing till $S = n-m$. So, $0 \leq S \leq n-m$.

: n=13, m=4 $0 \leq S \leq 9$

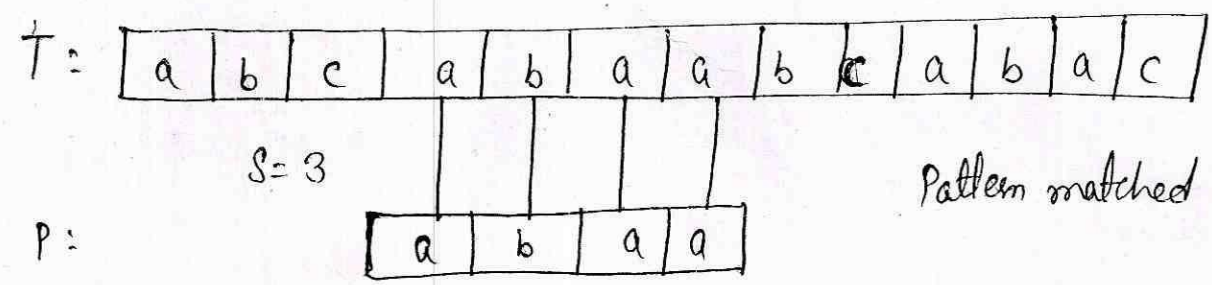




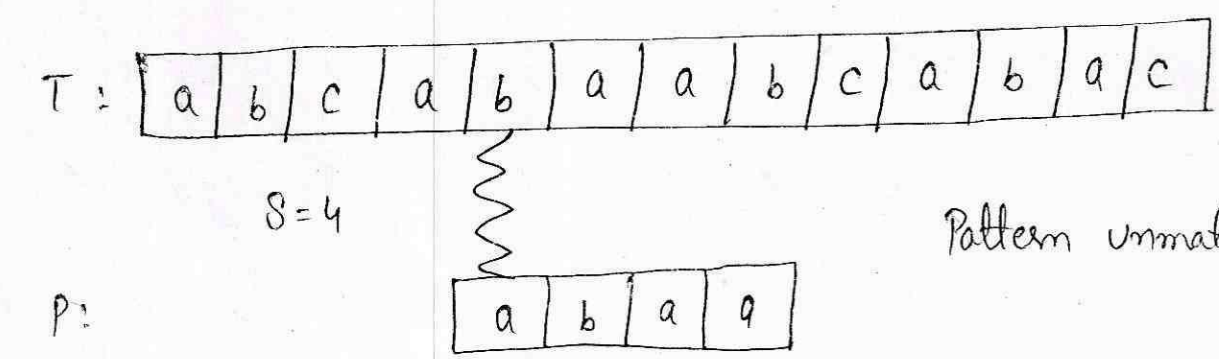
Pattern unmatched



Pattern unmatched



Pattern matched



Pattern unmatched

We will keep on shifting till we get next match or the text string becomes smaller than the pattern.

Thus, we see that pattern P matches the given text T for S = 3. This S = 3 is valid shift. In all other cases, where pattern doesn't match, S is invalid shift.

1) Naive Pattern Matching Algorithm: (also called brute-force algorithm

* Pattern matching algorithms generally follow 2 steps:
  ① Pre-processing of the string
  ② Comparison with Shifting of pattern (i.e. matching)

* In all other pattern matching algorithms, some preprocessi is done & then valid shift is found. But in naive algorithm, no preprocessing is done.

* Comparison → from left to right.
  When mismatch occurs, pattern is moved one position to the right with respect to the text and again comparison is done from left to right.

Algorithm:

Naive_String_Matches

[Inputs to the algorithm are the text T and pattern to be matched, P]

Step 1: Find length of text
  Set n ← length[T]

Step 2: Find length of pattern
  Set m ← length[P]

Step 3: For every possible value of shift S compare the characters in both arrays and print shift value on successful match.

For $S \leftarrow 0$ to $n-m$

If $P[1 \ldots m] = T[S+1 \ldots S+m]$; then:

"pattern matched" and

return S ;

Step 4 : Exit

Analysis: For loop of the algorithm at that running time proportional to $(n-m+1)$ &

worst case compaison $O(m)$

So total runningtime for Naive algorithm is

$$O((n-m+1)m)$$

Since, there is no preprocessing done so, this running time is the time for second Step, i.e. matching the string.

Advantage :

→ It is simple

disadvantage :

Cannot avoid rescanning the input text string T

ex:

P = 0001

T = 000010001010001

So, n ← length [T] = 15

m ← length [P] = 4

S neill vary from 0 to 11 (n−m)

* Same as simple method S = 1, 5, 11 Ans

2) Rabin Karp Algorithm :-

* The Rabin Karp algorithm involves both the steps of pattern matching : ① Preprocessing
② matching

* ~~we assume that String process starts with the calculation of dec~~

* we assume that string characters contain $\Sigma = \{0, 1, \ldots 9\}$ thus, each character is a decimal digit. Our process starts with the * calculation of decimal value for the pattern & * the substring of given text.

* for pattern P[1...m] → p denotes the decimal value for text T[1...n] → t_s " the " " for length m substring T[S+1... S+m] where, $0 \le S \le n-m$.

* for the given pattern, S is valid shift if & only if $p = t_s$, means

$$P[1...m] = T[S+1...S+m]$$

Now p & t_{s+1} are calculated as:

$$p = P \bmod q$$

$$t_{s+1} = \left(d(t_s - T[s+1]h) + T[s+m+1]\right) \bmod q$$

(old high-order digit) (new low-order digit)

where, $d = 10$ for decimals

$$h = d^{m-1} \bmod q$$

* If $t_s = p$ it does not assure that its a valid shift s. But, it will definitely rule out invalid shifts s. So, it is required to further test if it is really valid & not fake hit.

But if $t_s \neq p$ (Invalid hit)

Algorithm:

RABIN_KARP_MATCHER (T, P, d, q)

[T → Text, P → pattern, d → ~~base~~ decimal value of text and pattern, q → modulo]

Step 1: Find the length of text T & pattern P

$n \leftarrow$ length [T]

$m \leftarrow$ length [P]

Step 2: set $h \leftarrow d^{m-1} \bmod q$

Step 3: Initialize decimal values p & t_0

Set $p \leftarrow 0$

Set $t_0 \leftarrow 0$

Step 4: for $i \leftarrow 1$ to m // preprocessing

do $p \leftarrow (dp + P[i]) \bmod q$ (after Lem $\Rightarrow p \bmod q$

$t_0 \leftarrow (dt_0 + T[i]) \bmod q \Rightarrow t_0 \bmod q$

Step 5: for $s \leftarrow 0$ to $n-m$ // Matching

do if $p = t_s$

then if $P[1 \ldots m] = T[s+1 \ldots s+m]$

then: print "pattern occurs with shift" s

if $s < n-m$ then:

$t_{s+1} \leftarrow (d(t_s - T[s+1]h) + T[s+m+1]) \bmod q$

Step 6: Exit

Example:

$T: \langle 2,3,5,9,0,2,3,1,4,1,5,2,6,7,3,9,9,2,1 \rangle$

$P: \langle 3,1,4,1,5 \rangle$

modulo $q = 13$

$n = 19, \quad m = 5, \quad 0 \leq s \leq 14$

Step 1:

T:	2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1

$s = 0$

P:	3	1	4	1	5

8

$$p = P \bmod 9$$

$$= 31415 \bmod 13$$

$$= 7.$$

There is no shift initially so,

$$t_s = 23590 \bmod 13$$

$$= 8$$

$8 \neq 7$ Unmatched

Step 2:

T:	2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1

$s = 1$

P:	3	1	4	1	5

$$h = d^{m-1} \bmod q$$

$$h = 10^4 \bmod 13 = 3$$

$$t_{s+1} = \left(d(t_s - T[s+1]\,h) + T[s+m+1]\right) \bmod 13$$

$$t_{s+1} = t_1 = \left(10(8 - 2 \times 3) + 2\right) \bmod 13$$

$$= (20 + 2) \bmod 13$$

$$= 22 \bmod 13$$

$t_1 = 9 \neq 7$ Unmatched

Step 3:

T:	2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1

$s = 2$

P:	3	1	4	1	5

$$t_{s+1} = t_2 = (10(9 - 3 \times 3) + 3) \bmod 13$$

$t_2 = 3 \neq 7$ unmatched.

Step 4:

T:	2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1

S=3

P:	3	1	4	1	5

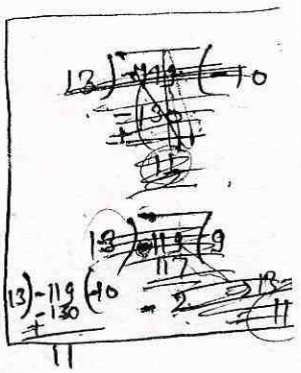
$t(s+1)$

$2+5+1$

$$t_{s+1} = t_3 = (10(3 - 5 \times 3) + 1) \bmod 13$$

$t_3 = -119 \bmod 13$

$t_3 = 11 \neq 7$ unmatched



Step 5:

T:	2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1

S=4

P:	3	1	4	1	5

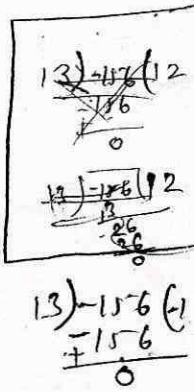
$$t_{s+1} = t_4 = (10(11 - 9 \times 3) + 4) \bmod 13$$

$t_4 = (10 \times -16 + 4) \bmod 13$

$t_4 = -156 \bmod 13$

$t_4 = 0 \neq 7$

pattern unmatched



Step 6:

T:

2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1

$S = 5$

P:

3	1	4	1	5

$$t_{s+1} = t_6 = (10(0 - 0 \times 3) + 1) \bmod 13$$

$$t_6 = 1 \bmod 13$$

$$t_6 = 1 \neq 7 \quad \text{unmatched}$$

Step 7:

T:

2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1

$S = 6$

P:

3	1	4	1	5

$$t_{s+1} = t_7 = (10(1 - 2 \times 3) + 5) \bmod 13$$

$$t_7 = (-50 + 5) \bmod 13$$

$$t_7 = -45 \bmod 13$$

$$t_7 = 7 = 7$$

13) -45 (-4
+52
7

Here $p = t_{s+1} = t_7 = p$ p:

but, we have to still check if pattern P & text T match

$P = <3\ 1\ 4\ 1\ 5>$

$T = <3\ 1\ 4\ 1\ 5>$

pattern matched for $\boxed{S = 6}$

Step 8: We can go further in text T & find if more pattern exists further. Here only pattern match is at $\boxed{S = 6}$. Ans

Example:

$T : \langle 3141592653589793 \rangle$

$P : \langle 26 \rangle$

modulo $q = 11$

$m = \text{length}[P] = 2, \quad n = \text{length}[T] = 16$

$0 \le S \le 14$

$S = 6$ Ans

2) Knuth Morris Pratt Algorithm:

* In Naive algorithm for pattern matching, every time a mismatch occurs, there is a shift by a single character.
* Knuth Morris Pratt algorithms avoid testing for such useless shifts. For this purpose, an auxiliary function called the prefix function for a pattern, denoted by 'π' is computed.

Algorithm:

KMP_Matcher (T, P)

Step1: Compute the length of text T & pattern P.

$n \leftarrow \text{length}[T]$

$m \leftarrow \text{length}[P]$

Step2: Compute the prefix function for P. So call the procedure

Compute_prefix_function.

$\pi \leftarrow$ Compute_prefix_function (P)

Step3: Set $q \leftarrow 0$ // Initially the no. of characters matched is 0.

Step4: Now, we have to compare the text with pattern P character by character from left to right.

```
for i ← 1 to n
{
    Do while q>0 and P[q+1] ≠ T[i]
    {
    } do q ← Π[q].          // next character does not much so decrementing value of q

    If P[q+1] = T[i] ; then:
    {
        q ← q+1             // next character ~~does not match~~ matches, so increment q
    }

    If q = m then:
    {
        print "pattern occurs with shift" i-m

        q ← Π[q]            // find out for the next pattern match in rest of the text.

    }
}
```

Step5: Exit

⇒ Compute_Prefix_Function (P)

Step1: Compute the length of P

m ← length[P]

Step2: First value of prefix function would be definitely 0. No comparison initially.

Π[1] ← 0

K ← 0

Step3: First element of array π is 0. So looping for the purpose of Comparing pattern with itself will start from element 2 as:

```
For q ← 2 to m
{
    do while k>0 and P[k+1] ≠ P[q]
    {
    }   do k ← π[k]        // mismatch so k is decremented
    If P[k+1] = P[q] then:   ⟶ matching again begin from starting & store index pos. in another next data set
    {
        set k ← k+1
    }
    π[q] ← k
}
```

Step4: Return function π and Exit

Example:

$$P = \langle a\ a\ b\ a\ b \rangle \quad (q: 1\ 2\ 3\ 4\ 5)$$

$$T = \langle a\ a\ a\ b\ a\ b\ a\ a\ b\ a\ a\ b\ a\ b\ a\ a\ b \rangle \quad (i: 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16\ 17)$$

$n \leftarrow$ length $[T] = 17$

$m \leftarrow$ length $[P] = 5$

for $q = 1$, $\pi[1] = 0$

Set $\boxed{k \leftarrow 0}$

for $q = 2$

$P[k+1] = P[1] = a$

$P[q] = P[2] = a$

If $P[k+1] = P[q]$

$a = a$ T so $k = k+1 \Rightarrow \boxed{k = 1}$

4

$$\boxed{\pi[2] = 1}$$

for $q = 3$ $\quad k > 0$

$$P[k+1] = P[2] = a$$
$$P[q] = P[3] = b$$

$\Rightarrow \quad P[k+1] \neq P[q]$

so $\quad k = \pi[1] = 0$

$$\boxed{k = 0}$$

so $\quad P[k+1] = P[1] = a$
$\quad P[q] = P[3] = b$

$\because \quad P[k+1] = P[q]$
$\quad a \neq b \quad F$

so $\boxed{\pi[3] = 0}$

for $q = 4$

$$P[k+1] = P[1] = a$$
$$P[q] = P[4] = a$$

$\because \quad P[k+1] = P[q]$
$\quad a = a \quad T$

$$\boxed{k = 1}$$

$$\boxed{\pi[4] = 1}$$

for $q = 5$

$$P[k+1] = P[2] = a$$
$$P[q] = P[5] = b$$

$k > 0$ & $\quad P[k+1] \neq P[q]$

$$a \neq b$$

so $k = \pi[1] \Rightarrow \boxed{k = 0}$

Now $P[k+1] = P[1] = a$

$P[q] = P[5] = b$

If $a = b$ F

$\Rightarrow \boxed{\pi[5] = 0}$

Thus, prefix function for the pattern as follows:

q	1	2	3	4	5
P[q]	a	a	b	a	b
π[q]	0	1	0	1	0

π[k]

// Now perform the matching of text T & pattern P.

Set $q = 0$

for $i = 1$ to 17

$\Rightarrow \underline{i = 1}$

If $P[q+1] = P[1] = a$

$T[i] = T[1] = a$

If $P[q+1] = T[i]$

$\Rightarrow a = a$

So $q = q+1 \Rightarrow \boxed{q = 1}$

If $q = m \Rightarrow 1 \neq 5$

$\Rightarrow \underline{i = 2}$ while $q > 0$ & $P[q+1] \neq T[i]$

$P[2] \neq T[2] \Rightarrow a \neq a$ F

16

if $P[q+1] = T[i]$

$a = a$ T

$\boxed{q = 2}$

$\Rightarrow q \neq m$.

$i = 3$

$P[q+1] = P[3] = b$

$T[i] = T[3] = a$

while $q > 0$ && $P[q+1] \neq T[i]$

$b \neq a$

do $q \leftarrow \pi[2] \Rightarrow \boxed{q = 1}$

→ which represents that atleast 1 character is already matched & now, compare to next values i.e. $[q+1]$ & $T[i]$.

now $q > 0$ && $P[q+1] \neq T[i] \Rightarrow P[2] \neq T[3]$

$\Rightarrow a \neq a$ F

if $P[2] = T[3]$

$a = a$ T

$\boxed{q = 2}$

$i = 4$

$P[q+1] = P[3] = b$

$T[i] = T[4] = b$

$P[q+1] = T[i]$

$\Rightarrow b = b$ T $\Rightarrow \boxed{q = 3}$

$i = 5$

$P[q+1] = P[4] = a$

$T[i] = T[5] = a$

$P[q+1] = T[i]$

$\Rightarrow \quad a = a \quad T \Rightarrow$ $\boxed{q = 4}$

$i = 6$

$P[q+1] = P[5] = b$

$T[i] = T[6] = b$

$P[q+1] = T[i]$

$b = b \Rightarrow$ $\boxed{q = 5}$

if $q = m \Rightarrow 5 = 5$ T

pattern occurs with shift $i - m = 6 - 5 = 1$

$\boxed{S = 1}$

$q = \pi[5] = \boxed{q = 0}$

Continue until $i = 17$ $\boxed{S = 1, S = 9}$ Ans.

	1	2	3	4	5	6	7	8	9	10
P[q]	a	b	a	b	a	b	a	b	c	a
π[q]	0	0	1	2	3	4	5	6	0	1

$\Rightarrow P = \langle a\,b\,a\,b\,a\,b\,a\,b\,c\,a \rangle$

Boyer - Moore Algorithm :

In this algorithm we start the matching of characters from right end of the pattern. If a match is occured we proceed in left direction of the pattern and try to match second last character of the pattern and so on. The character in the text that is mismatched is known as the bad character. Once a mismatch occurs the bad character is searched in the remaining part of the pattern.

Rule 1 : If the bad character exists in the remaining part of the pattern, then the ~~whole pattern~~

bad character in the text is aligned with the same character in the pattern and the matching process is moved on in the left direction.

rule II If the bad character does not exist in the remaining part of the pattern, then the whole pattern is shifted right after the bad character and then matching process is carried on in the left direction.

Boyer-Moore-Matcher (T, P, Σ)

1. $n \leftarrow$ length$[T]$
2. $m \leftarrow$ length$[P]$
3. $\lambda \leftarrow$ compute-last-occurrence-function(P, m, Σ)
4. $\gamma \leftarrow$ compute-good-suffix-func(P, m)
5. $s \leftarrow 0$
6. while $s \leq n - m$ (17 − 10 = 7)
7. do $j \leftarrow m$
8. while $j \geq 0$ & $P[j] = T[s+j]$
9. do $j \leftarrow j - 1$ ✓
10. if $j = 0$
11. then print "pattern occurs at shift s"
12. $s \leftarrow s + \gamma[0]$
13. else $s \leftarrow s + \max(\gamma[j], j - \lambda[T[s+j]])$

Good suffix — bad character

Compute-Last-occurrence-function (P, m, Σ) (a, b, c)

1. for each character $a \in \Sigma$
2. do $\lambda[a] = 0$
3. for $j \leftarrow 1$ to m
4. do $\lambda[P[j]] \leftarrow j$
5. return λ

Compute-good-suffix-function (P, m)

1. $\pi \leftarrow$ compute-prefix-func.(P)
2. $P' \leftarrow$ reverse(P)
3. $\pi' \leftarrow$ compute-prefix-func.(P')
4. for $j \leftarrow 0$ to m
5. do $\gamma[j] \leftarrow m - \pi[m]$
6. for $l \leftarrow 1$ to m
7. do $j \leftarrow m - \pi'[l]$
8. if $\gamma[j] > l - \pi'[l]$
9. then $\gamma[j] \leftarrow l - \pi'[l]$
10. return γ

JECRC

I Mid Term Test-2013

Branch: CSE **Semester: VI**

Subject: Design and Analysis of Algorithms **Max Marks: 20**

Time: 1hrs 15min

Attempt all questions.

Q1. Show the strassen's matrix multiplication process on the matrix A and B given below: (5)

$$A = \begin{bmatrix} 4 & 2 \\ 3 & 3 \end{bmatrix} \qquad B = \begin{bmatrix} 5 & 4 \\ 1 & 4 \end{bmatrix}$$

Q2. Consider 5 items along their respective weights and values (5)

I = <I1, I2, I3, I4, I5>

W= <5, 10, 20, 30, 40>

V= <30, 20, 100, 90, 160>

The capacity of knapsack W=60. Find the solution to the fractional knapsack problem.

Q3. Use Huffman algorithm (Optimal merge pattern algorithm) to construct a tree for the frequencies given in figure: (5)

A	B	C	D	E	F
30	15	25	8	2	20

Q4. Find an optimal parenthesization of a matrix chain product whose sequence of dimension is: (5)

<5, 4, 6, 2, 7>

P = ababa Ka ababa

π = 0 0 1 2 0 0 1 2 3 4

P' = ababa Kabab

π' = 0 0 1 2 3 0 1 2 3 4

j ← 0 to 10

γ[j] ← m − π[m] = 10 − π[10]
= 6

γ

0	1	2	3	4	5	6	7	8	9	10
6	6	6	6	6	6	6	6	6	6	6

l ← 1 to m(10)

l = 1 j ← m − π'[l]
10 − π'[1]
10 − 0 = 10

if γ[10] > l − 0
6 > 1 T

γ[10] ← 1

γ

0	1	2	3	4	5	6	7	8	9	10
6	6	6	6	6	6	4	3	2	2	1

l = 2 j ← m − π'[l]
10 − π'[2]

same.

l = 3 j ← m − π'[3]
10 − 1
9

if γ[9] > l − π'[3]
3 − 1 = 2
6 > 2
γ[9] ← 2

l = 4 j ← 10 − 2 = 8
if γ[8] > 2
γ[8] ← 2

l = 5 j ← 10 − 3 = 7
γ[7] > 3
γ[7] ← 2

l − π'[l]
5 − 3 2

l = 6 j ← 10 − 0 same as (j, 2)

l = 7 j ← 10 − 1 = 9
if γ[9] > 1 f

l = 8 j ← 10 − 2 = 8
if γ[8] > 2 f

l = 9 j ← 10 − 3 = 7
if γ[7] > 3 F

l = 10 j ← 10 − 4 = 6
if γ[6] > 4
γ[6] ← 4

γ

0	1	2	3	4	5	6	7	8	9	10
6	6	6	6	6	6	4	3	2	2	1

6 2 2 2 1

Bad character :

λ

a	b	K
10	9	5

Boyer Moore - Matcher

T ababakababakababa

P ababakababa

n ← 16 , m ← 10

λ , γ.

S ← 0

while S ≤ (16 − 10) = 6

j ← 10

while j > 0 & P[10] = T[0+10]

j ← 9

while j > 0 & P[9] = T[9]

j ← 8,

j ← 7. P[7] = T[7] f.

S ← S + max(γ[j], j − λ[T[S+j]]

= 0 + max(γ[7], 7 − λ[T[7]])

= 0 + max(3, 7 − λ[K])

= 0 + max(3, 7 − 5)

= 0 + max(3, 2)

= 3

if T = ababakababakababa
P = ababakababa j = 10
j = 0
Now, it matches with shift S = 0
S = S + γ[0] = 6
T = ababakababakababa
ababakababa

* So, in γ[0] we put the value of minimum shift.

MA	8:30 1	9:30 2	10:30 3	11:30 3	12:00 4	1:00 5	2:00 6
Mon	SEMINAR 8CSA3			B			
Tue	SEMINAR 8CSA2			R			
Wed				E			
Thu				A		PROJECT-8CS-A1	
Fri				K		PROJECT-8CS-A2	
Sat						PROJECT-8CS-A3	

SC	8:30 1	9:30 2	10:30 3	11:30 3	12:00 4	1:00 5	2:00 6
Mon				B	AVA-6CS		
Tue				R	AVA-6CS	SEMINAR IT 8CSB1	
Wed				E		SEMINAR 8CSB1	
Thu	JAVA LAB-6CS-A1			A		SEMINAR 8CSA2	
Fri	JAVA LAB-6CS-A2			K		AVA-6CS	
Sat	JAVA LAB-6CS-A3					PROJECT-8CS-A3	

RJB	8:30 1	9:30 2	10:30 3	11:30 3	12:00 4	1:00 5	2:00 6
Mon	PPL-4CS			B	SHELL LAB-6CS-A1		
Tue		PPL-4CS		R	SHELL LAB-6CS-A2		
Wed			PPL-4CS	E	SHELL LAB-6CS-A3		
Thu	PPL-4CSA			A		SEMINAR IT 8CSA3	
Fri			PPL-4CSA	K		SEMINAR IT 8CSA1	
Sat	SEMINAR 8CSB1		PPL-4CSA				

NC	8:30 1	9:30 2	10:30 3	11:30 3	12:00 4	1:00 5	2:00 6
Mon		AI 6CSA		B		PROJ 8CSB1	
Tue				R	AI 6CSB	PROJ 8CSB2	
Wed	AI 6CSA			E	AI 6CSB	PROJ 8CSB3	
Thu				A	AI 6CSA	SEMINAR 8CSA2	
Fri				K		SEMINAR 8CSA3	
Sat			AI 6CSA				

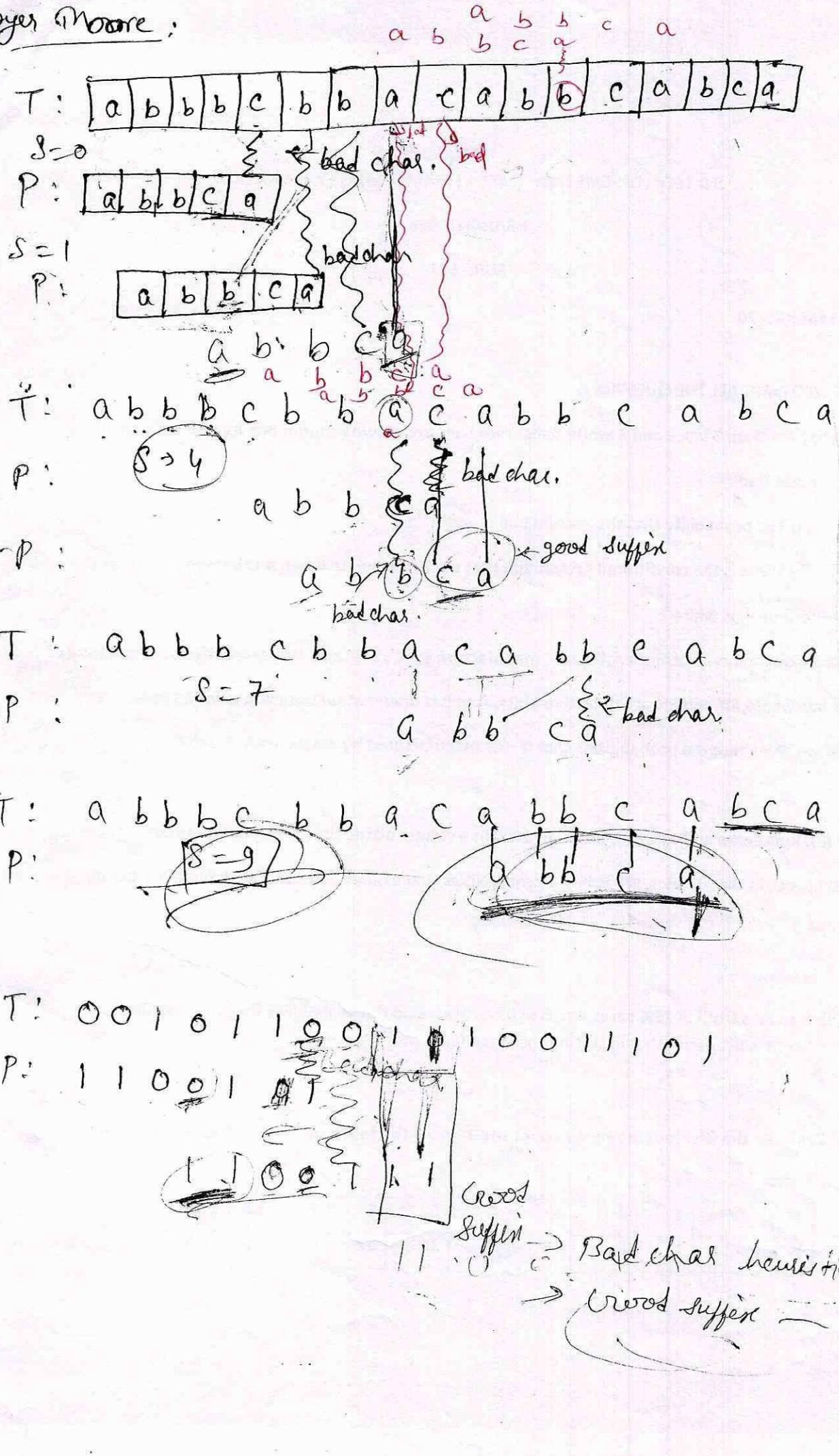
AMIT	8:30 1	9:30 2	10:30 3	11:30 3	12:00 4	1:00 5	2:00 6
Mon	JAVA LAB-6CS-B1			B			
Tue	JAVA LAB-6CS-B2			R		AVA-6CS	
Wed	JAVA LAB-6CS-B3			E		VA-6CS	
Thu	SEMINAR 8CSB3			A	AVA-6CS		
Fri	SEMINAR 8CSB2			K		SEMINAR IT 8CSA1	
Sat	SEMINAR 8CSB1						

3-89

1-90
2-180
3-270
4-360
5-450

according to prefix function computed.
in which at last we have value
4. Now 10-4=6

Boyer Moore:



Q: T: 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 0 1

P: 1 1 0 0 1

1 1 : 0 → Bad char heuristic

→ Good suffix

Assignment Problem

* Assignment problem is one of the fundamental combinatorial optimization.

* In the assignment problem we have given n numbers of persons (or agents) and n numbers of jobs (or tasks). Our aim is to assign n jobs to n persons so as to minimize the total cost of performing n jobs.

Eg: A manager has 4 persons (i.e. facilities) available for 4 separate jobs and the cost of assigning each job to each person is given. His objective is to assign each person only one job in such a way that the total cost of assignment is minimized.

* If j^{th} job is assigned to i^{th} person, where $1 \le i \le n$ and $1 \le j \le n$. Then the cost of performing this particular job is c_{ij}.

So it can be formulated as:

minimize the cost $\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$

such that

$$x_{ij} = \begin{cases} 1, & \text{if } i^{th} \text{ person is assigned } j^{th} \text{ job} \\ 0, & \text{otherwise} \end{cases}$$

$\sum_{j=1}^{n} x_{ij} = 1$ (one job is done by i^{th} person)

$\sum_{i=1}^{n} x_{ij} = 1$ (one person is assigned j^{th} job)

$x_{ij} \Rightarrow j^{th}$ job is to be assigned to the i^{th} person

Algorithm

Assignment_problem

Step1: Check whether the no. of rows are equal to the no. of columns, i.e. no. of jobs should be same as the no. of persons. If not, add a dummy row or column to the matrix. This can be done by adding 0's in the dummy row or column.

Step2: Subtract minimum element of each row from other elements of that row such that there is a zero in each row. If already there is a zero in any row, no need of subtraction.

Step3: Subtract minimum element of each column from other elements of that column such that there is a zero in each column. If already there is a zero in some column no need of subtraction.

Step4: Select a row containing exactly one uncovered zero & draw a vertical line through the column containing this zero. select a column containing exactly one uncovered zero & draw a horizontal line through the row containing the zero.

Step5: The total lines should be equal to the size of the matrix which signifies optimal sol.n. If it is not so, select the minimum element which is uncovered & subtract it from other uncovered elements & add it to the elements at the intersection of lines covering zeros. Again start the process of drawing horizontal & vertical lines that cover

all zeros.

Step6: for assignment of jobs, select each row with exactly one zero, mark the column corresponding to the zero & circle that zero. Repeat this for each row.

Step7: Circled cells are the allocation which forms the optimal solution. If C_{ij} is the circled cell, i is the person & j is the job assigned to person i.

Step8: Add up the values of selected circled cells which form the minimum value of the solution.

<u>example</u>

Given the cost matrix

1)

Person i \ Job j	1	2	3	4	
1	33	40	43	32	-32
2	45	28	31	23	-23
3	42	29	36	29	-29
4	27	42	44	38	-27

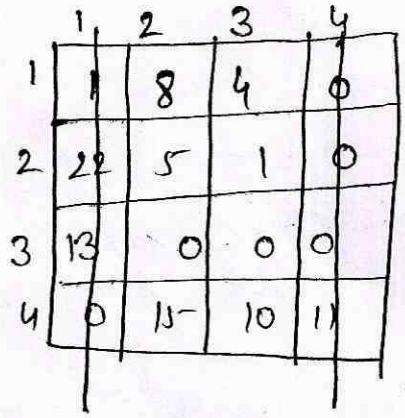
2)

	1	2	3	4	
1	1	8	11	0	
2	22	5	8	0	
3	13	0	7	0	
4	0	15	17	11	
					-7

3)

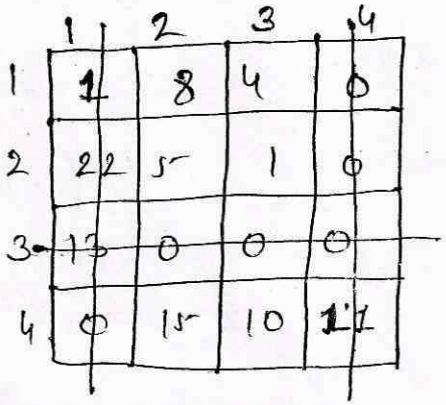
Person i \ Job j	1	2	3	4
1	1	8	4	0
2	22	5	1	0
3	13	0	0	0
4	0	15	10	11

Select a row that contains exactly 1 zero & draw a vertical line through the column containing that zero.

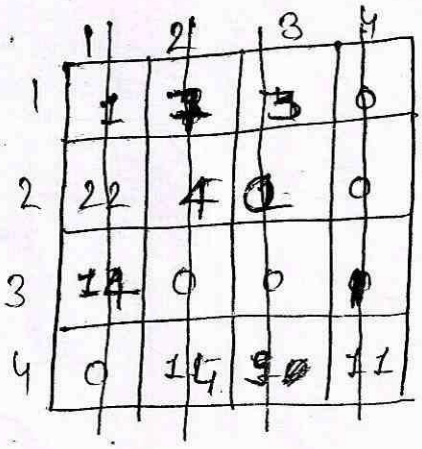


Third row contains more than one zero, so it is ignored. Now select a col. containing exactly 1 zero & draw a horizontal line through the row containing that zero.

Column 2 contains exactly one 0, so cross the row 3 which contains this 0. No more columns required to be worked



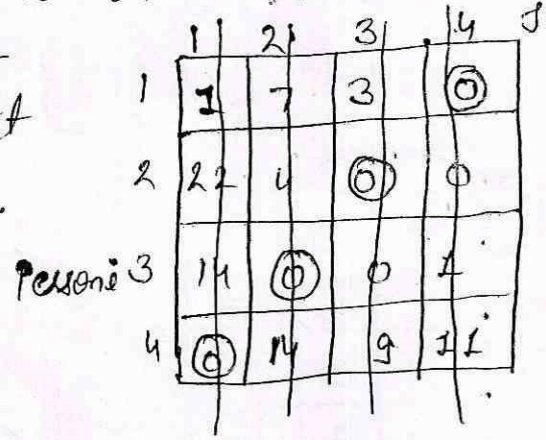
:) Total 3 lines are there, but the size of matrix 4. so it is not an optimal sol.n. Now select the min.m element of the unmarked elements (which is 1 in this example) & subtract it from all unmarked elements & add it to elements at intersection of lines.



→ again check zero in each row & col.

→ process of drawing lines is repeated again. Total 4 lines are there equal to the size of matrix. So it is an optimal sol.n.

6) Next step is the assignment process. select the row with exactly one zero, circle it & mark the col. containing that zero. Do this for every row.



1.) C_{14}, C_{23}, C_{32}, C_{41} are the circled zeros & give us the optimal soln. This solves that

Person 1 is assigned job 4
Person 2 is assigned job 3
Person 3 is assigned job 2
Person 4 is assigned job 1

Total cost = 32 + 31 + 29 + 27
= 119

eg:-

Cost matrix = $\begin{bmatrix} 4 & 7 & 5 \\ 2 & 6 & 1 \\ 3 & 9 & 8 \end{bmatrix}$ $\begin{matrix} -4 \\ -1 \\ -3 \end{matrix}$

1) $\begin{bmatrix} 0 & 3 & 1 \\ 1 & 5 & 0 \\ 0 & 6 & 5 \end{bmatrix}$ $\begin{matrix} \\ \\ -3 \end{matrix}$ $\begin{bmatrix} 0 & 0 & 1 \\ 1 & 2 & 0 \\ 0 & 3 & 5 \end{bmatrix}$

2.) $\begin{matrix} & 1 & 2 & 3 \\ 1 & & & \\ 2 & & & \\ 3 & & & \end{matrix}$ $\begin{bmatrix} 0 & 0 & 1 \\ 1 & 2 & 0 \\ 0 & 3 & 5 \end{bmatrix}$

3) $\begin{matrix} & 1 & 2 & 3 \\ 1 & & & \\ 2 & & & \\ 3 & & & \end{matrix}$ $\begin{bmatrix} 0 & 0 & 1 \\ 1 & 2 & 0 \\ 0 & 3 & 5 \end{bmatrix}$

C_{12}, C_{23}, C_{31}

7 + 1 + 3 ⇒ 11

Assignment Problem

* The assignment problem is one of the fundamental Combinatorial optimization.

* In the assignment problem we have given n number of persons (or agents) and n number of jobs (or tasks). Our aim is to assign n jobs to n persons so as to minimize the total cost of performing n jobs.

* There are $\underline{n}$ combinations but we have to obtain an optimal solution that minimizes the total cost of performing n jobs.

* If j^{th} job is assigned to i^{th} person where $1 \le i \le n$ and $1 \le j \le n$. Then the cost of performing this particular job is c_{ij}

So it can be formulated as:-

$$\text{minimum cost} = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

Subject to

$$x_{ij} = \begin{cases} 1 & \text{if each } i^{th} \text{ person is assigned } j^{th} \text{ job} \\ 0 & \text{otherwise} \end{cases}$$

* By applying Branch-and-Bound, we illustrate how to solve the assignment problem.

Example:

C_{ij} =

i ↓ \ j →	Jobs 1	2	3	4
A	90	12	50	51
B	70	10	58	80
Person C	16	85	8	70
D	11	37	80	21

i ← persons , j ← jobs

A, B, C denote the persons
1, 2, 3 denote the jobs.

* In the given problem we have 4 agents & 4 jobs. Now we have to assign 4 jobs to 4 persons. Our aim is to find out which job has been assigned to which person so that the cost will be minimum.

* In the given problem agents are denoted by rows & jobs are denoted by columns. and the value represents the cost to perform that task.

⇒ Lowerbound :

$C[LB] = 11 + 10 + 8 + 21$
$= 50$

// pick minm values from each col & compute their sum

$R[LB] = 12 + 10 + 8 + 11$
$= 41$

// pick minm values from each row & compute their sum

$\max\{C[LB], R[LB]\} = \max\{50, 41\}$

⇒ $\boxed{LB = 50}$

Upper Bound : (Sum of the digonal elements of the cost matri

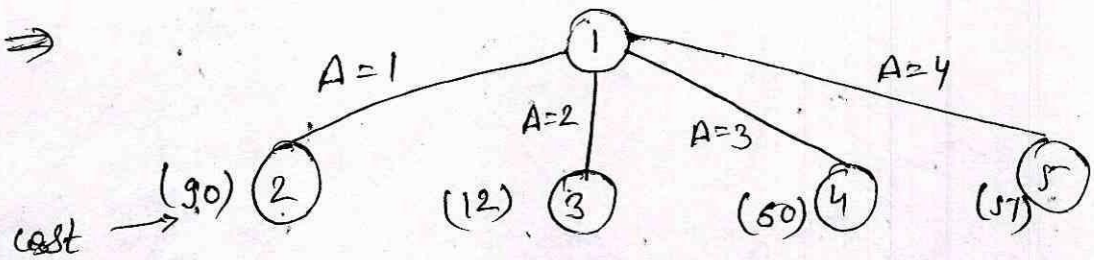
$$\text{UB}_1 = 90 + 10 + 8 + 21$$
$$= 129$$

$$\text{UB}_2 = 51 + 58 + 85 + 11$$
$$= 205$$

$$\text{UB} = \min\{\text{UB}_1, \text{UB}_2\} = \min\{129, 205\}$$

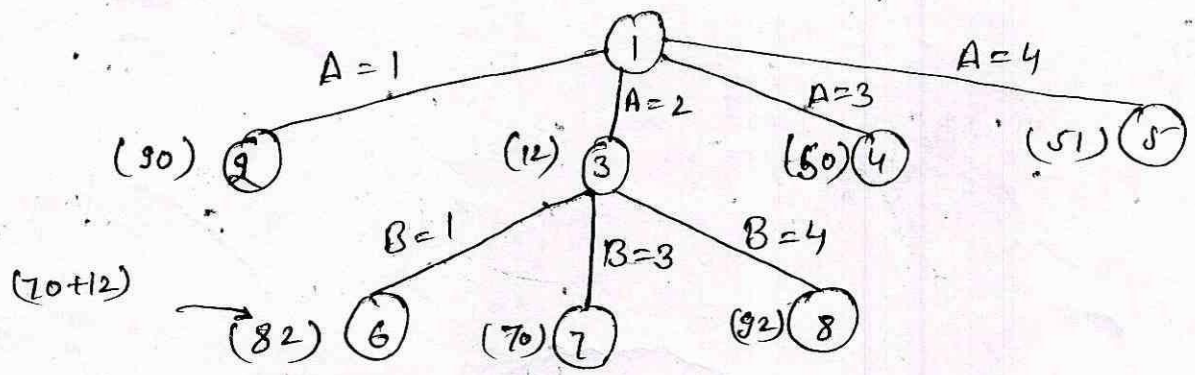
$$\Rightarrow \boxed{\text{UB} = 129}$$

$\Rightarrow$



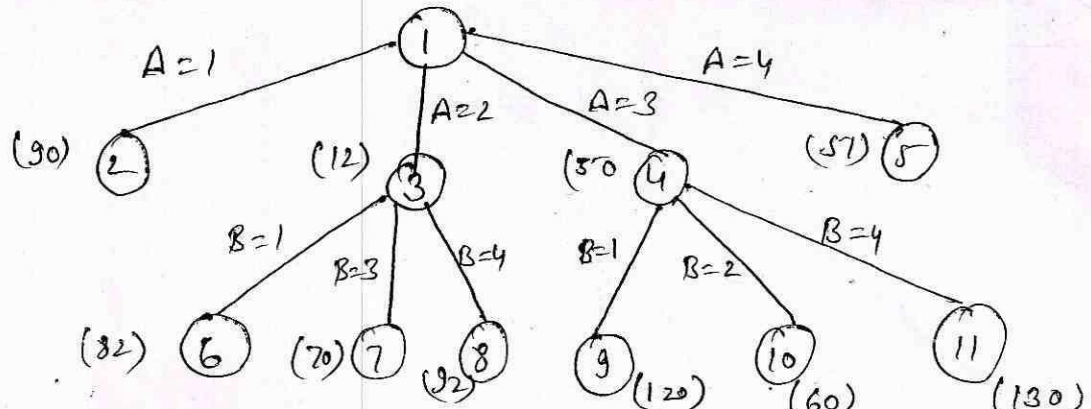
[assigning jobs 1, 2, 3 & 4 to person A]

$\Rightarrow$ At this stage min^m cost node currently is node 3.
So explore it further:



* At present, not with min^m value is node 4

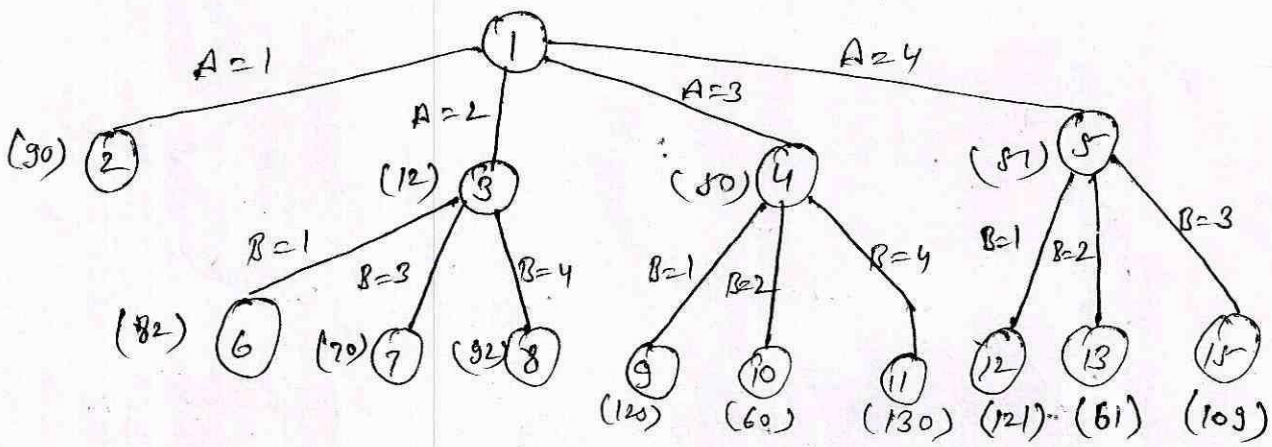
⇒



(Crossed the upper bound so there is no need of exploring it further)

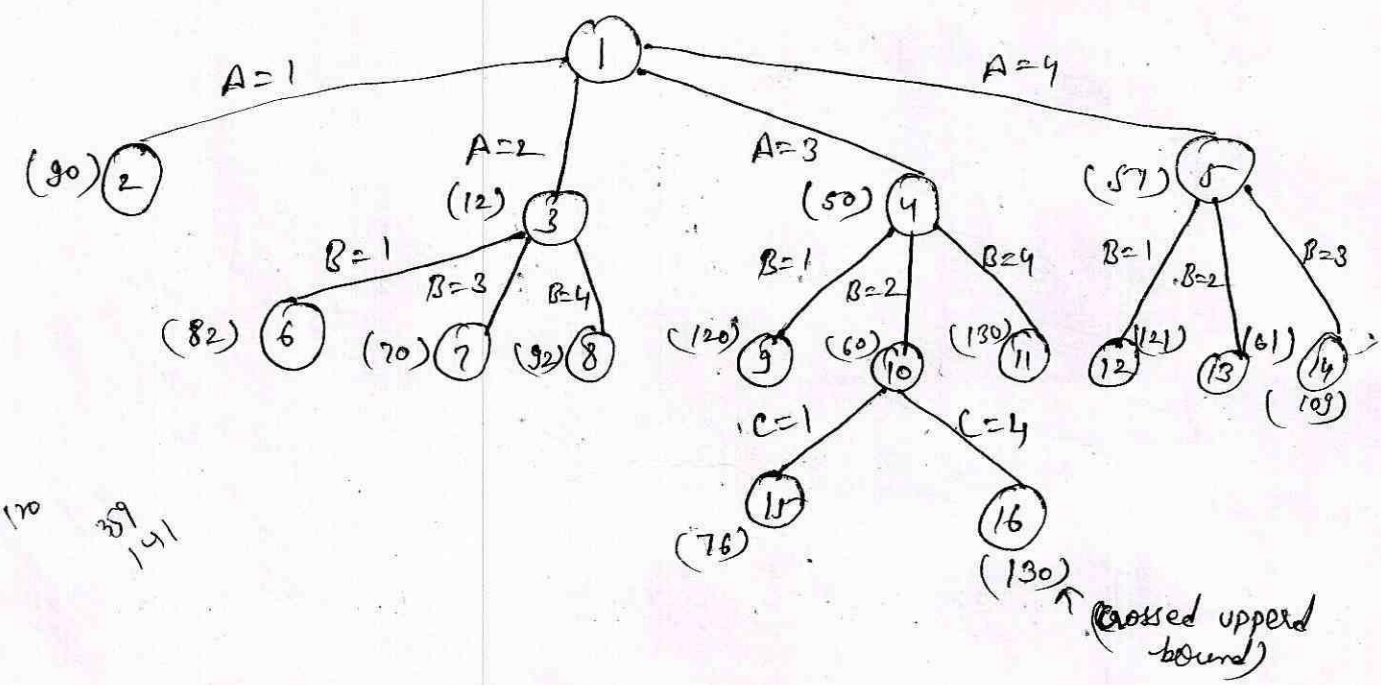
* Node 5 is the min cost node

⇒



* Node 10 is min

⇒



(Crossed upper bound)

* min is node 13

UNIT : 4

Randomized Algorithms

(1) So far we have seen only deterministic algorithms. (2) They produces same output on same input always.

(3) But there are some algorithms that rely on chances. (4) These are of a Randomized Algo.

(5) "A randomized Algorithm is an algorithm that is allowed to access independent, unbiased, random bits, and then allowed to use these random bits, to influence its computation."

(6) So we can say that randomized Algorithms belongs to the category of Probabilistic Algos.

Why to use Randomized Algorithms?

(7) Some Algos have good best case & Average Case complexity but very bad worst case complexity

(8) for eg - QuickSort

Best case || Avg case = $O(n \log n)$

Worst case $O(n^2)$

which is very bad as compared to Avg case

So to improve worst case like in Quicksort in randomized Algos, we will choose random value of pivot, instead of first or last element which can lead us very close to average case.

(9) Thus we use randomized Algorithms.

Randomized Algorithm can be of two types —

① Las Vegas Algorithms

② Monte Carlo Algorithms.

II Las Vegas Algorithms

→ The randomized Algorithm that Always produce the correct output for the same input [random bits can be chosen for every run], then these are known as Las Vegas Algorithms.

→ The runtime of such Algorithms is random as it depends on the output of a randomizer.

→ If we are lucky, the algorithm might terminate fast, else it can run for a longer period of time

→ It does not gamble with the correctness of the result, it only gambles with the resources used for the computation.

NK

we characterize

(5) ~~for~~ all randomized Algorithm ~~we do~~ by $O(\cdot)$ notation but for Las Vegas algorithm we characterize it by $\tilde{O}(\cdot)$ notation.

(6) → ~~It uses~~ For Quick sort, it uses a random function which is used to choose pivot.

Random(x, y) [no to be chosen as pivot b/w x & y]

if x < y

return (x + (rand() * (x - y)));

IV

Monte Carlo Algorithms

1. The algorithm whose output may differ from run to run is known as Monte Carlo Algorithms
2. These algorithms are used when it is infeasible or impossible to compute an exact result with a deterministic Algorithm
3. There can be diff. approaches but General steps are
   - ⇒ Define ~~input~~ domain of inputs.
   - ⇒ Using probability distribution generate random inputs.
   - ⇒ Perform a deterministic computation using the inputs.
   - ⇒ Aggregate the result of the individual component into a final result.

Primality Testing

To check whether the given no. n is prime or not

Fermet's theorem states that "If p is prime, then for every $1 \leq a < p$, $a^{p-1} \equiv 1 \pmod{p}$"

This theorem will take lot of time as it will check for every no.

So we choose random no, to increase prob.

$p = 7$, $a = 3$

$p = 8$

$a^{p-1} \equiv 1 \pmod{p}$ — Pass — Prime

— fail — Composite

$a^{p-1} - 1 = pk$ $\quad k \in I$

$\frac{3^6 - 1}{7}$ $\quad 104$ $\quad 3^7$

(5) M.C. Algo has two sub types

↳ One sided Error :- If the input is prime then it will always give correct answer but if not then there is a prob. that it can give wrong answer. So this is one sided error. 50% chance of Error on one side of decision

↳ Two sided Error :- It gives error on both side of the decision.

(6) Book

V

Randomized Algorithm for Min-Cut Problem

To understand the problem, we need some terms –

→ A cut of a connected graph is obtained by

(1) dividing the vertex set V of a graph G into two sets V_1 & V_2 such that

→ no common vertices in V_1 & V_2 $(V_1 \cap V_2 = \phi)$

→ $V_1 \cup V_2 = V$

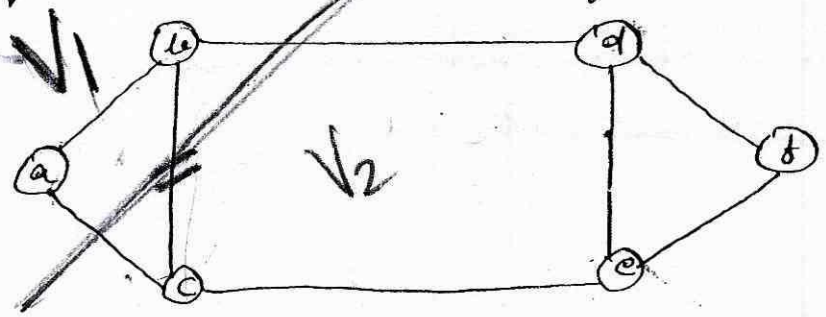
(2) Any edge from V_1 to V_2 or V_2 to V_1 is said to be crossing the cut.

(3) All the edges which cross the cut collectively form the cut-set of the graph.

(5) This is for unweighted graph. For weighted graph we have different Algorithms.

Q1 Find a minimal cut for the given graph using a randomized algorithm.

(6)



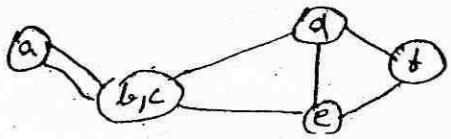
Sol – $V = \{a, b, c, d, e, \}$

$G = \{(a,b), (b,c), (a,c), (b,d), (c,e), (d,e), (a,f), (e,f)\}$

Step 1 – Select random edge

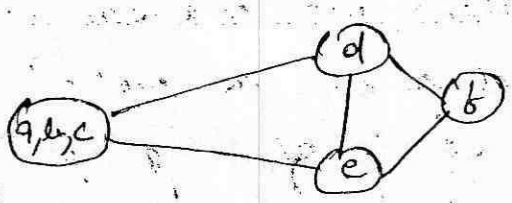
(b, c)

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



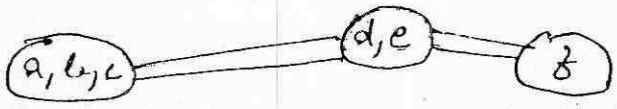
2) (a,c)

$V = [\{a, b, c\}, d, e, f\}$



3) (d,e)

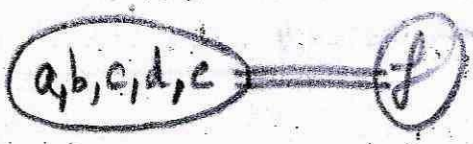
$V = [\{a, b, c\}, \{d, e\}, f\}$



4) (d,f)

$V = \{\{a, b, c\}, \{d, e, f\}$

or



7) Defination of Min cut of Graph:-

A min cut is the ~~minimum cut~~ division of graph into minimum no. of components making two vertices only.

VI

Multicommodity flow

(1) In practical Application, we need to send more than one commodity (goods) from one place to other through same intermediate stations or routes. For such a representation, we use a multicommodity flow network.

Formal Definition

(2) A multicommodity flow network is a directed graph $G(V, E)$ such that

⇒ for every edge $(u,v) \in E$, a capacity function $c(u,v)$ is associated.

⇒ There is a distinct set S of vertices called sources. $S = \{s_1, s_2, \dots s_n\}$

⇒ There is a distinct set T of vertices called sink. $T = \{t_1, t_2, \dots t_n\}$

⇒ There are n commodities $K_1, K_2, \dots K_n$ defined by $K_i = (s_i, t_i, d_i)$ where s_i is the source, t_i is the sink of commodity and d_i is the demand.

(3) **Properties** of multicommodity flow n/w are;-

→ **Capacity constraints** - Through each edge the total amount of ~~various~~ commodities flowing should not exceed its capacity i.e.

$$\sum_{i=1}^{h} f_i(u,v) \leq c(u,v)$$

⇒

→ **Flow conservation :-** for all $u \in V - \{s, t\}$ we require

$$\sum_{v \in V} f_i(u, v) = 0$$

and

Skew Symmetry :-

$$u, v \in V, \; f_i(u, v) = -f_i(v, u)$$

→ **Demand Satisfaction :-** The objective of the multicommodity flow network is to flow each commodity from its source to its destination, as per the demand. Thus the amount of commodity i, leaving source s_i, reaching destination t_i is equal to the demand d_i, i.e.

$$\sum_{v \in V} f_i(s_i, v) = \sum_{v \in V} f_i(v, t_i) = d_i$$

(4) Multicommodity Problems

(1) Minimum cost Multicommodity flow problem

There is a cost 'a' for sending flow on any edge (u, v). We need to minimize the total cost of flow of [redacted] commodity

$$\text{minimize} \sum_{(u,v) \in E} \left[a(u, v) \cdot \sum_{i=1}^{n} f_i(u, v) \right]$$

(2) Maximum Multicommodity flow problem

Total throughput is to be maximized.

i.e. flow the maximum for each commodity

maximize $\sum_{v \in V} f_i(s_i, v)$.

(3) Maximum Concurrent flow problem

The task is to maximize the minimal fraction of the flow of each commodity to its demand.

Objective is $\max_{1 \le i \le n} \frac{\sum_{v \in V} f_i(s_i, v)}{d_i}$

VII

Flow Shop Scheduling

(1) * Let say we have n jobs each requiring m tasks ~~each requiring m machines~~ $T_{1i}, T_{2i}, T_{3i} \dots T_{mi}$ for $i = 1$ to n

* Task T_{ji} is to be performed on processor P_j for $j = 1$ to m

* ~~Tim~~ No processor may have more than one task at any time.

* Task T_{ji} cannot be started until task $T_{j-1,i}$ has been completed.

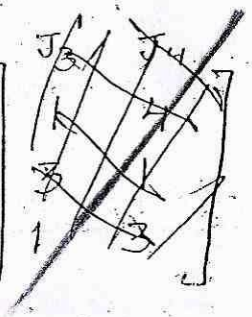
(2) Two types

(1) Preemptive Scheduling :- A schedule in which the processing of a task on any processor is allowed to terminate before task is completed.

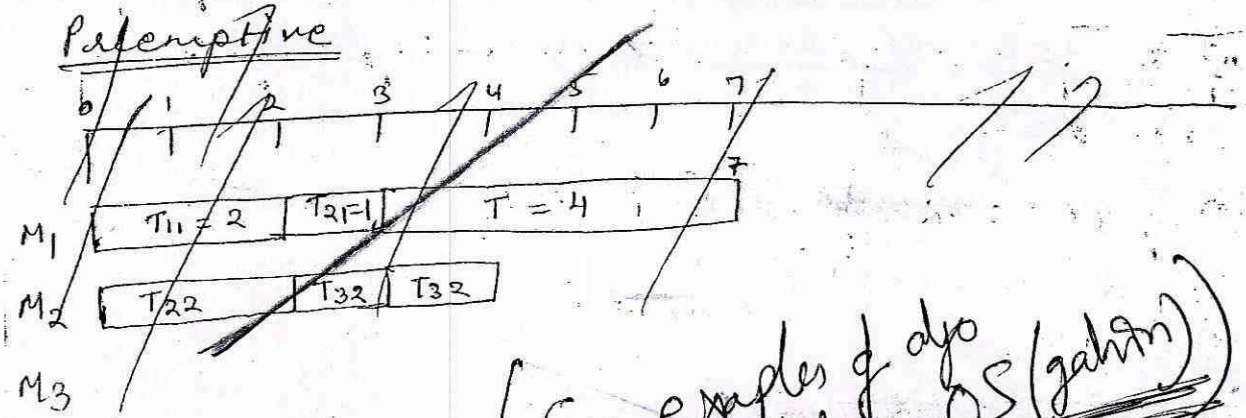
(2) **Non preemptive** Scheduling :- A schedule in which the processing of a task on any processor is not terminated until the task is completed

eg

$$T = \begin{matrix} & T_1 & T_2 \\ M_1 & 2 & 0 \\ M_2 & 1 & 4 \\ M_3 & 4 & 2 \end{matrix}$$

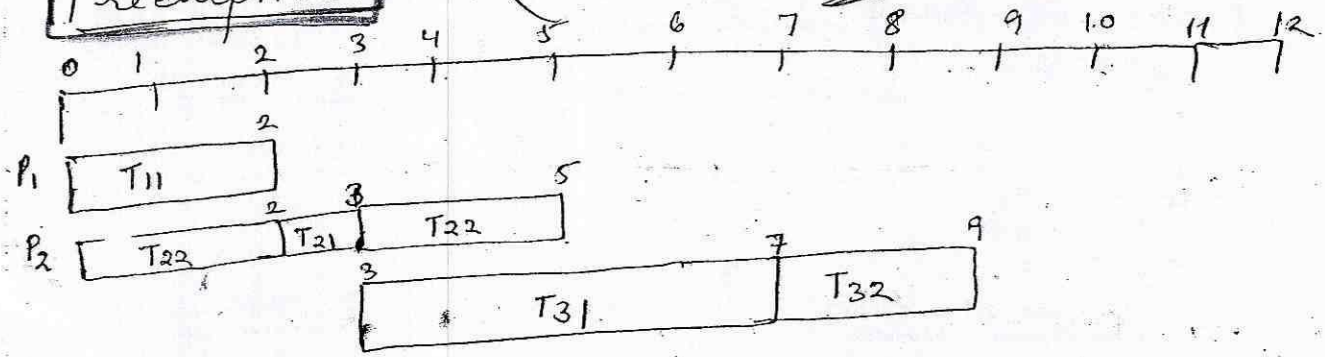


Preemptive



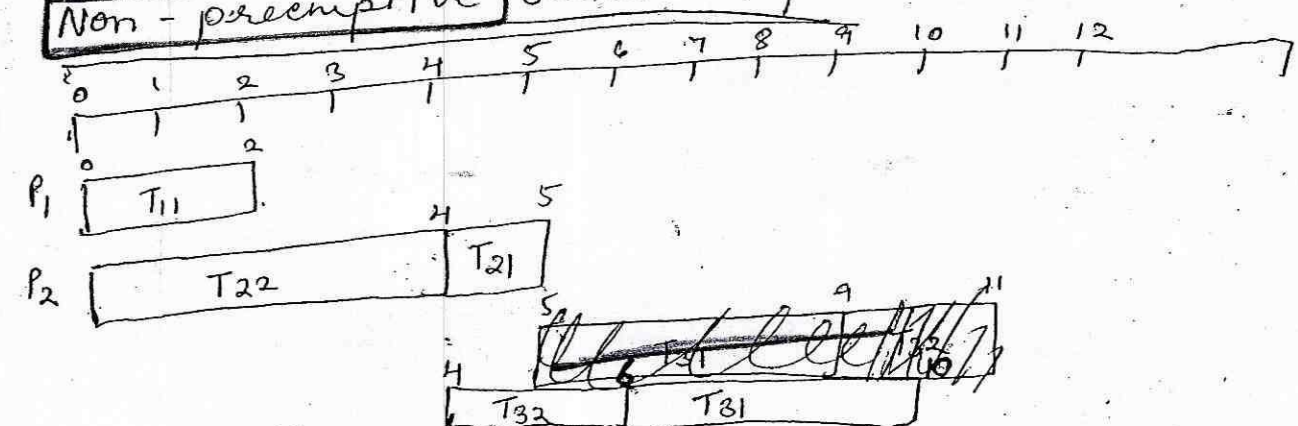
(See examples & dyo from OS (gahan))

(3) **Preemptive**



Finish Time $f_i(S) = 9$

Non - preemptive Scheduling



Finish Time $f_i(S) = 10$

Flow Networks

Just as we can model a road map as a directed graph in order to find the shortest path from one point to another, we can also interpret a directed graph as a "flow network"

The flow network means flow of material from source to sink.

Definition :- A flow network $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$.

If $(u, v) \notin E$, we assume that $c(u, v) = 0$

Formal Definition

Let $G = (V, E)$ be a flow network with a capacity function c. Let s be the source of the n/w, and let t be the sink. A flow in G is a real valued function $f : V \times V \to R$ that satisfies three properties :-

(1) **Capacity Constraint** :- for all $u, v \in V$, we require $f(u, v) \leq c(u, v)$.

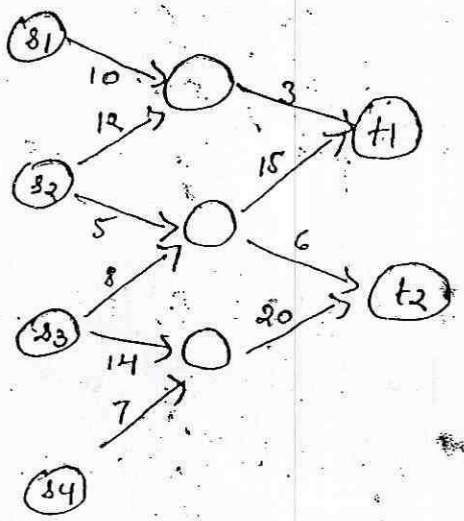
(2) **Skew Symmetry** :- for all $u, v \in V$, we require $f(u, v) = -f(v, u)$

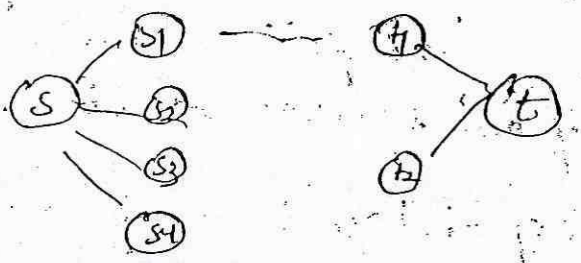
(3) **Flow Conservation** :- for all $u \in V - \{s, t\}$ we require

$$\sum_{v \in V} f(u, v) = 0$$

Networks with Multiple sources and sinks

A flow network can have several sources & sinks, rather than just one of each.





This can be converted into single source and single sink. i.e. ordinary max. flow n/w.

The Ford-Fulkerson Method

Some terms

① **Residual capacity** :- The amount of additional flow we can push from u to v before exceeding the capacity $c(u,v)$ ~~is the~~

$$c_f(u,v) = c(u,v) - f(u,v)$$

Residual n/w :- Given a flow n/w $G = (V, E)$ and a flow f, the residual n/w of G induced by f is $G_f = (V, E_f)$, where

$$E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$$

② Augmenting paths

Given a flow network $G = (V, E)$ and a flow f, an augmenting path p is a simple path from source s to sink t in the residual network G_f.

$$c_f(p) = \min \{ c_f(u,v) : (u,v) \text{ is on } p \}$$

③ Cuts of flow networks

A cut (S, T) of flow network $G = (V, E)$ is a partition of V into S and $T = V - S$ such that $s \in S$ and $t \in T$.

If f is a flow, then the net flow across the cut (S, T) is defined to be $f(S, T)$. The capacity of the cut (S, T) is $c(S, T)$

Max-flow min-cut theorem

If f is a flow in a flow network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent

1. f is a maximum flow in G.
2. The residual network G_f contains no augmenting paths
3. $|f| = c(S, T)$ for some cut (S, T) of G

Ford Fulkerson Algorithm (G, s, t)

(1) for each edge $(u,v) \in E[G]$
(2) do $f[u,v] \leftarrow 0$
(3) $f[v,u] \leftarrow 0$
(4) while there exists a path p from s to t in the residual network G_f
(5) do $c_f(p) \leftarrow \min\{c_f(u,v) : (u,v) \text{ is in } p\}$
(6) do each edge (u,v) in p
(7) do $f[u,v] \leftarrow f[u,v] + c_f(p)$
(8) $f[v,u] \leftarrow -f[u,v]$

Analysis

F-F Algo runs in $O(E|f^*|)$
where f^* is the maximum flow found by the algorithm i.e. from line 4 to 8
and line 1 to 3 takes $O(E)$ runtime

Q - Solve the following flow network for max. flow.

Multicommodity flow problem

Flow Networks. Ruby Garg

A flow network $G = (V, E)$ is a directed graph in which edge $(u, v) \in E$ has a non-ve capacity $c(u, v) \geq 0$

→ flow of material from source to sink.

A flow in G satisfies the following three properties —

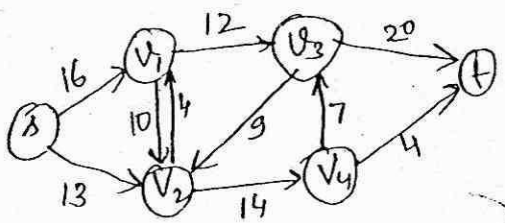
(i) Capacity Constraints → $f(u, v) \leq c(u, v)$

(ii) Skew Symmetry → $f(u, v) = -f(v, u)$

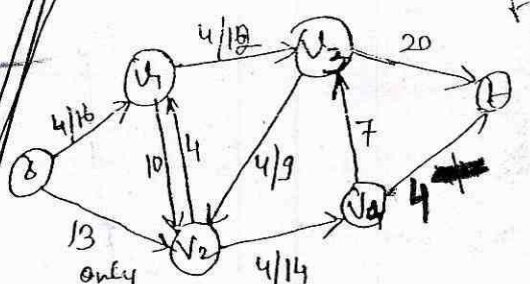
(iii) Flow conservation → $\sum_{v \in V} f(u, v) = 0$ means incoming flow may be +ive but except source and destination capacity can't.

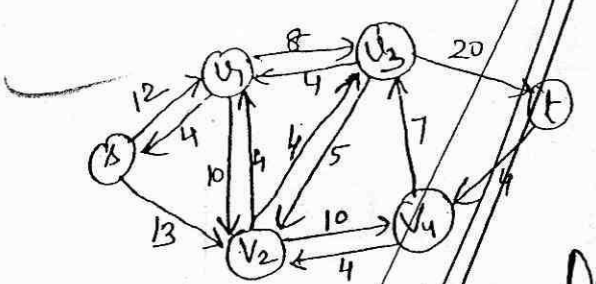
except source and sink.

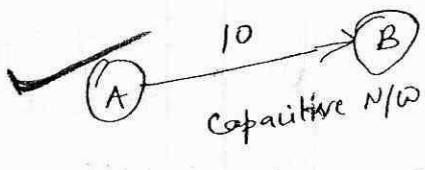
In the maximum flow problem we are given a flow network G with source s and sink t, and we wish to find a flow of maximum value.



Capacitive N/W







Ford-Fulkerson method.

* Ford-Fulkerson method is used to solve maximum flow problem.
* Ford fulkerson method is based on three ideas-
  - (i) Residual N/w.
  - (ii) Augmenting Paths
  - (iii) Cuts of flow N/w

(i) **Residual N/w** – The amount of addition flow that can be pushed from u to v before exceeding the capacity

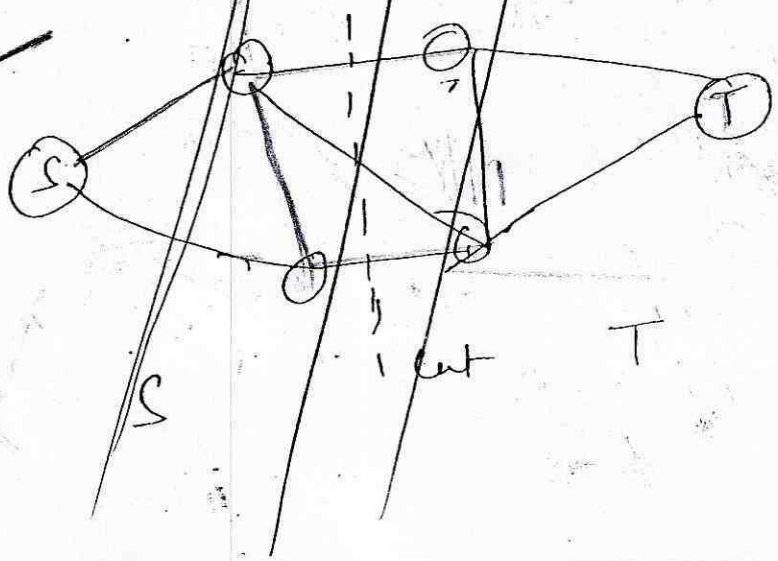
$$c_f(u,v) = c(u,v) - f(u,v)$$

Residual Capacity = Capacity − flow

Book

(ii) **Augmenting path** – A path from source to sink in the residue Network is called augmenting path

(iii) **Cuts of flow networks** – A cuts (S,T) of flow network $G=(V,E$ is a partition of network into two parts such that source s lies in the first part and sink T lies in the different part.



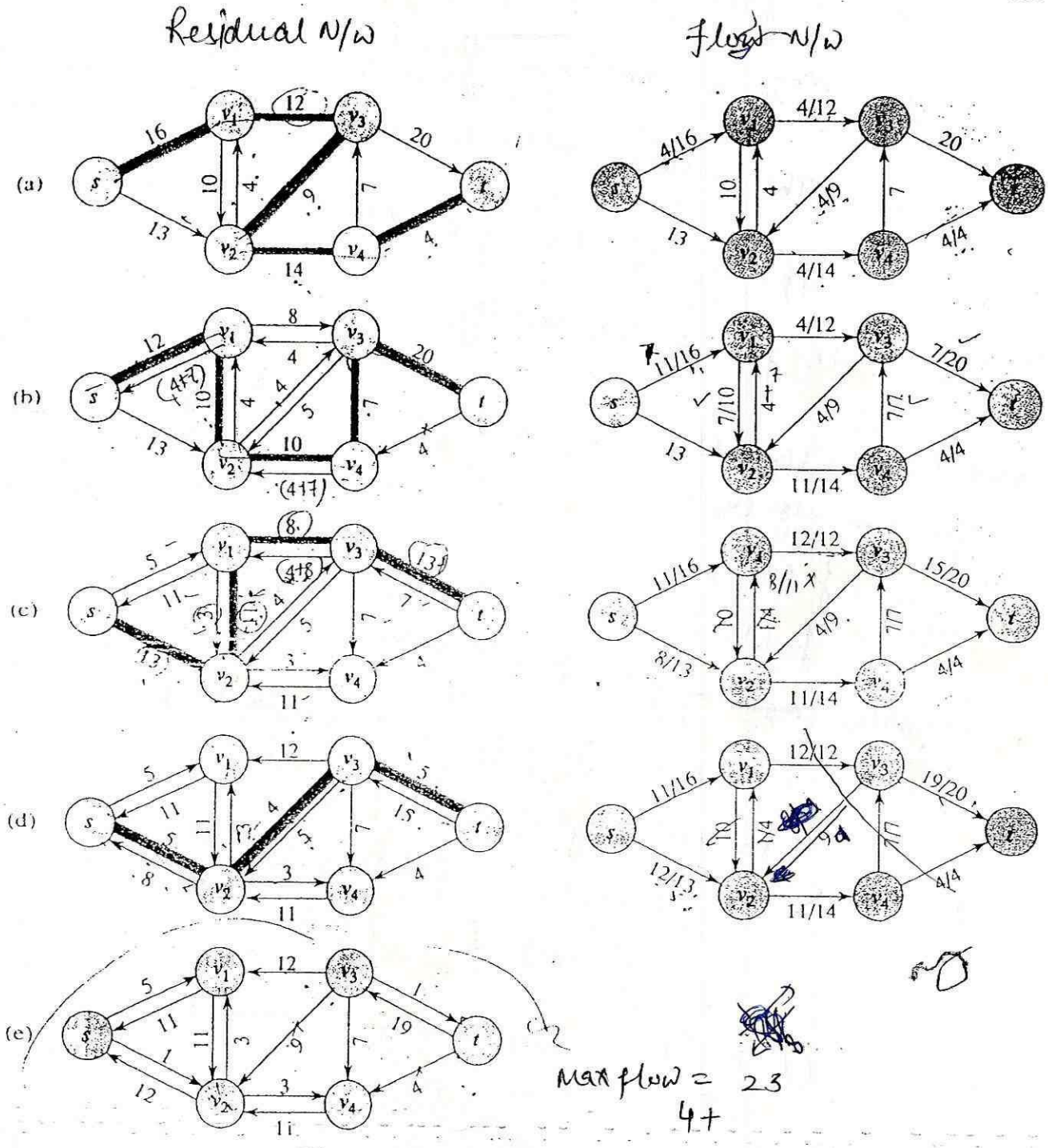


Figure 26.5 The execution of the basic Ford-Fulkerson algorithm. (a)–(d) Successive iterations of the while loop. The left side of each part shows the residual network G_f from line 4 with a shaded augmenting path p. The right side of each part shows the new flow f that results from adding f_p to f. The residual network in (a) is the input network G. (e) The residual network at the last while loop test. It has no augmenting paths, and the flow f shown in (d) is therefore a maximum flow.

Ruby garg

Max-Flow min-cut theorem.

In a flow network following conditions are equivalent -

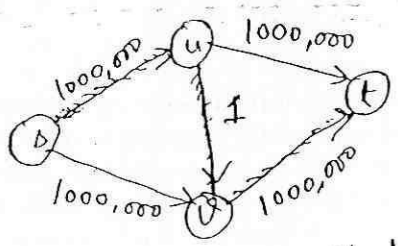
1. f is a maximum flow in G.
2. The residual network contains no augmenting path.
3. $|f| = c(S,T)$ for some cut (S,T) of G

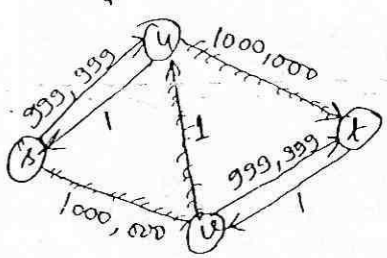
i.e. flow = cut capacity.

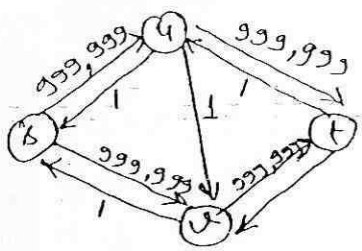
Ford-Fulkerson (G, s, t)

1. for each edge $(u,v) \in E$
   $f(u,v) = 0$
   $f(v,u) = 0$
2. while augmenting path p exist from s to t
   $c_f(p) = \min \{ c_f(u,v) : (u,v) \text{ is in } p \}$
   for each edge (u,v) in p
   $f(u,v) = f(u,v) + c_f(p)$
   $f(u,v) = -f(u,v)$

*T.C $O(E|f|)$ here f is the maximum flow.







* The above situation can be avoid using Edmonds-karp algor.
* Edmond karp runs in $O(VE^2)$ time.

Ford - Fulkerson method.

* Ford - Fulkerson method is used to solve maximum flow problem
* Ford fulkerson method is based on three ideas -
  (i) Residual N/w.
  (ii) Augmenting Paths
  (iii) Cuts of flow N/w

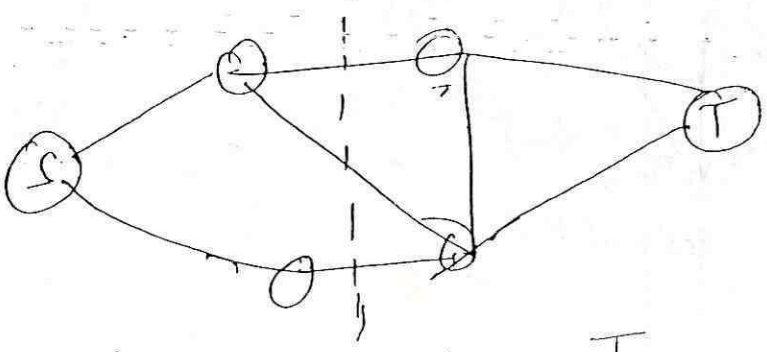
(i) Residual N/w - The amount of addition flow that can be pushed from u to v before exceeding the capacity

$$c_f(u,v) = c(u,v) - f(u,v)$$

Residual Capacity = Capacity - flow

(ii) Augmenting path - A path from source to sink in the residue Network is called augmenting path

(iii) Cuts of flow networks - A cuts (S,T) of flow network $G=(V,E$ is a partition of network into two parts such that source s lies in the first part and sink T lies in the different part.



Flow Networks.

Ruby Garg

A flow network $G = (V, E)$ is a directed graph in which edge $(u, v) \in E$ has a non-ve capacity $c(u, v) \geq 0$

→ flow of material from source to sink.

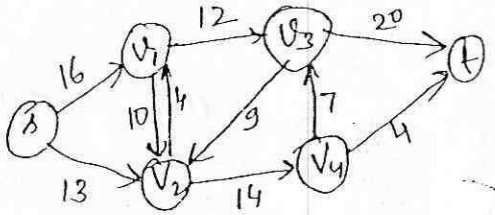
A flow in G satisfies the following three properties –

(i) Capacity Constraints – $f(u, v) \leq c(u, v)$

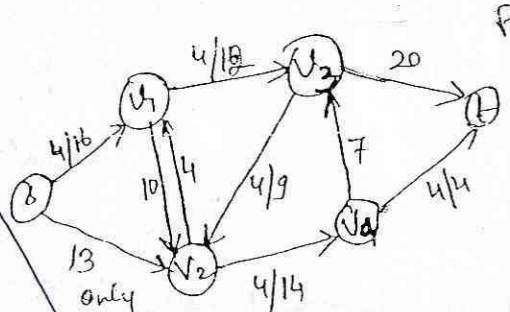
(ii) Skew Symmetry – $f(u, v) = -f(v, u)$

(iii) Flow conservation – $\sum_{v \in V} f(u, v) = 0$ means incoming flow may be -ive but except source and sink. capacity can't. except source and destination

In the maximum flow problem we are given a flow network G with source s and sink t, and we wish to find a flow of maximum value.



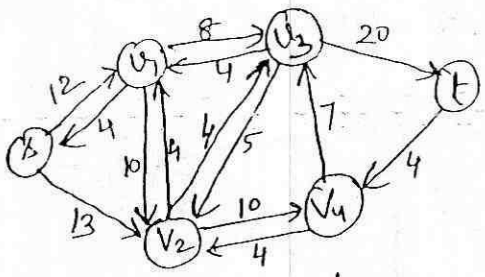
Capacitive N/w



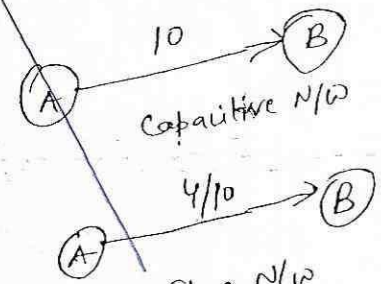
F/C

only Capacity

Flow Network

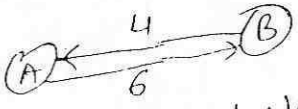


Residual N/w



Capacitive N/w

Flow N/w



Residual N/w

Unit-5.

Problem Classes NP, NP-Hard & NP Complete

Definitions of P, NP-Hard and NP Complete Problems

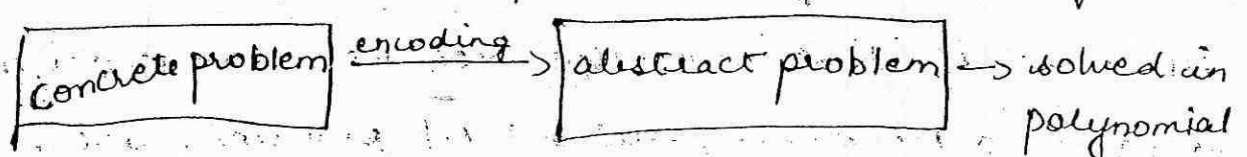
Class P :- It consists of those problems that are solvable in polynomial time. They are problems that can be solved in time $O(n^k)$ for some constant k, where n is the size of the input to the problem.

These problems are regarded as tractable, i.e. generally feasible i.e. it can be actually solved by some computational means.

To understand the class of polynomial Time solvable problem (P Class), we must have a formal notion of what a 'problem' is. We define it as abstract problem Q to be a binary relation on a set I of problem instances and a set S of problem solutions.

Now, to make an abstract problem Q to be a binary relation, we do encoding where we convert input into a binary pattern.

Thus we call a problem whose instance set is the set of binary pattern, a concrete problem



encoding act as a bridge between concrete problem and abstract problem.

NP Class :- The class NP consists of those problem that are verifiable in polynomial time.

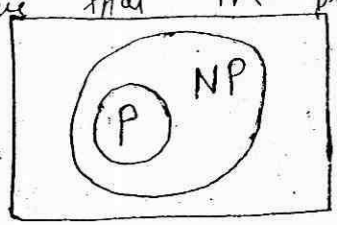
NP class means Non Deterministic Polynomial class

Suppose a computer program could guess a solution to a problem and then could check if the guessed solution is correct or not

~~This~~ An NP problem is the one which is not necessarily solvable in polynomial time by computational means but its verification is done in polynomi time.

Any problem in P is also in NP. This indicate that $P \subseteq NP$, but there is not a single problem in NP for it has been true that the problem in P as well as no polynomial relation between P and NP

Since NP is much larger set than P



It is yet undiscovered that $P = NP$ or $P \neq NP$ is true or not.

eg of NP Class are :- Hamiltonian Cycles, Traveling Salesman Problem etc. graph coloring, job scheduling, knapsack

NP-Hard :- The term NP-Hard refers to a problem as hard as any NP Problem. A problem is called NP-hard if it cannot be decided, or does not belong to NP class, but all NP problems are reducible to it in polynomial time.

Suppose there is a problem Q, which may or may not belong to NP i.e. it cannot be decide

But since an NP-complete problem (L) is the hardest problem in NP class, and L can be reduced to Q, we say that Q is as hard as L. Thus Q is NP-Hard.

NP complete :- Informal definition states that a problem is in the class NP complete if it is in NP and is as hard as any problem in NP

Formal definition states

A Language $L \subseteq \{0,1\}^*$ is NP complete if

1.) $L \in NP$ and

2.) $L' \leq_p L$ for every $L' \in NP$

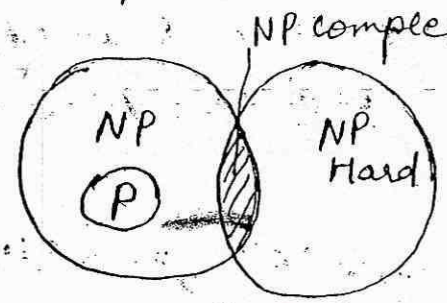
Second property states that 'pL' is the hardest of any problem in NP and L' are rest NP problem which are not as hard as pL

Thus $L' \leq_p L$

Only

If Second property is satisfied and not necessarily first property then we say that L is NP-hard.

Relationship among P, NP, NP-Hard and NP-Comp. can be defined with this Venn Diagram as.



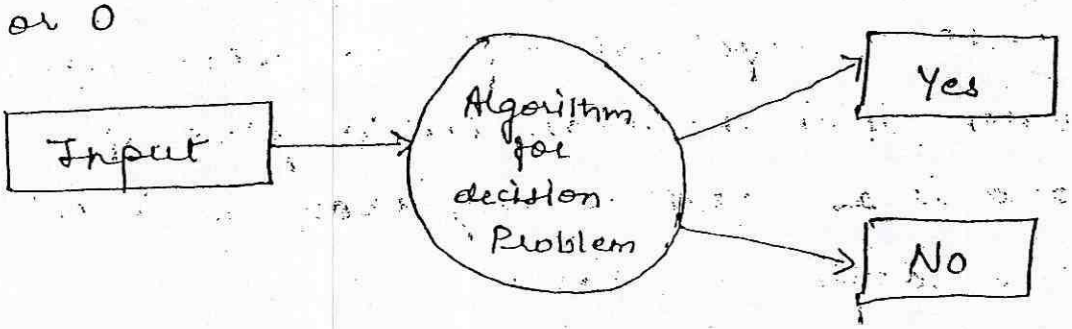
Hamilton cycle

cliques

travelling salesman problem.

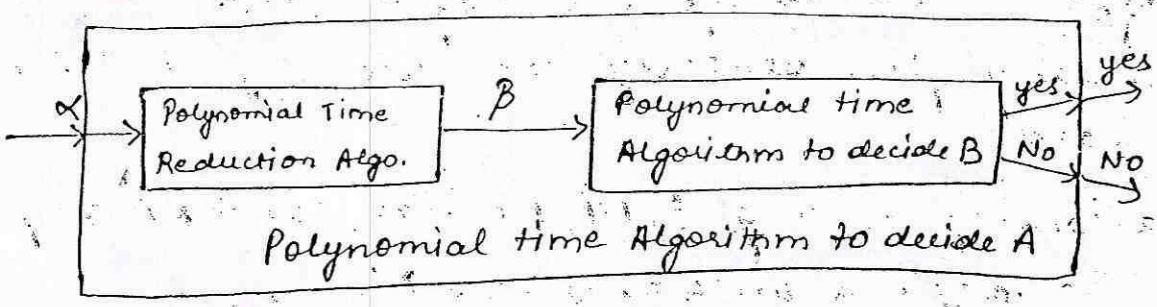
Decision Problem

Decision Problem is used to decide whether the output of an Algorithm is yes or no i.e. 1 or 0

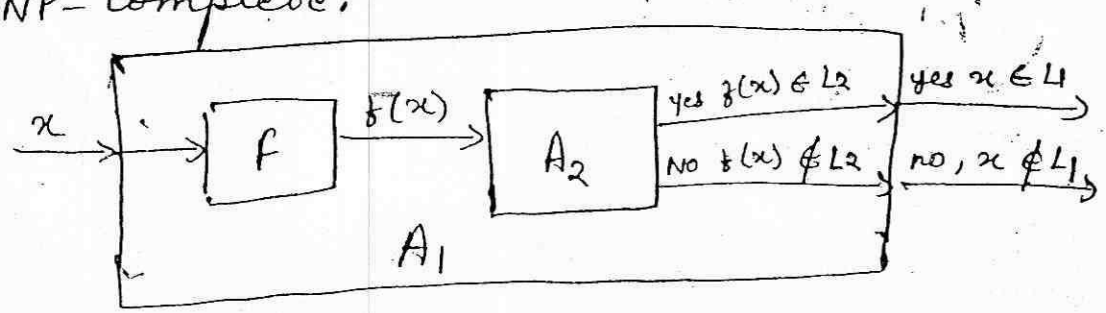


Now Suppose we have to decide the output of problem A which may be solved in polynomial time and we are given only an instance α of it

Now using a polynomial time reduction Algorithm we transform its instance α to β for problem B. Now using decision problem Algorithm we decide calculate the output for Problem B and accordingly decides the output for Problem A



With this we prove that problem B is NP complete on the assumption that problem A is also NP-complete.



The Vertex Cover problem

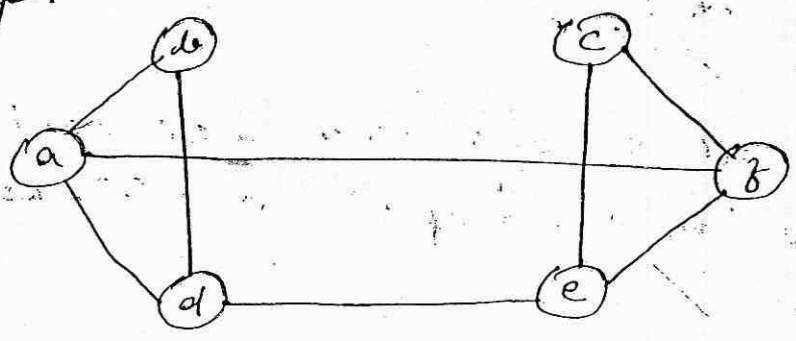
A Vertex cover of an undirected graph $G=(V,E)$ is a subset $V' \subseteq V$ such that if (u,v) is an edge of G, then either $u \in V'$ or $v \in V'$ (or both). The size of a vertex cover is the number of vertices in it.

We use an approximation Algorithm that takes as input an undirected graph G and returns a vertex cover that is near optimal. This problem is the optimization version of an NP complete decision problem.

APPROX-VERTEX-COVER(G)

1. $C \leftarrow \phi$
2. $E' \leftarrow E[G]$
3. while $E' \neq \phi$
4. do let (u,v) be an arbitrary edge of E'
5. $C \leftarrow C \cup \{u,v\}$
6. remove from E' every edge incident on either u or v
7. Return C.

Q- Find the vertex-cover of the following graph



Approximation Algorithms

→ Vertex Cover
→ Set Cover Problem

Approximation Algorithm :- If a problem is NP-Comp., then it is not necessary that we may find a polynomial-time algorithm for solving it exactly but still we have hope that we can solve it to some near-optimal solution. Thus an Algorithm that returns near optimal solutions is called 'Approximation Algorithm'

Performance Ratios for Approximation Algorithms :-

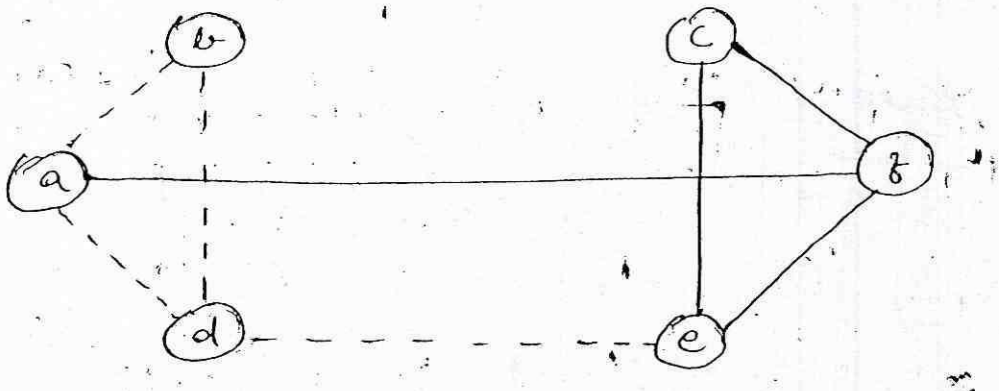
Suppose we are working on a problem in which each possible solution has some positive cost. Now the near optimal solution can be maximum cost or minimum cost depending upon the problem.

We say that an algorithm for a problem has an approximation ratio $\rho(n)$ if, for any input of size n, the cost C of the solution produced by the Algorithm is within a factor of $\rho(n)$ of the cost C^* of an optimal solution.

$$\max\left[\frac{C}{C^*}, \frac{C^*}{C}\right] \leq \rho(n)$$

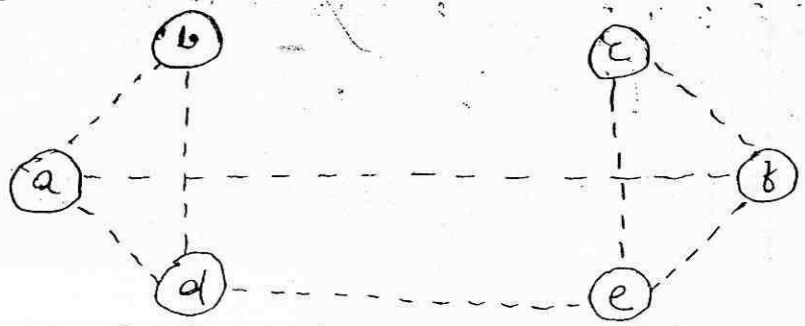
We also call an algorithm that achieves an approximation ratio of $\rho(n)$ a $\rho(n)$-Approximation algorithm.

1. Initially $C \leftarrow \phi$
   i.e. $C = \phi$
2. $E' \leftarrow E[G]$
   $\therefore E' = \{ab, ad, bd, af, de, ce, cf, ef\}$
   i.e. we have 8 edges & 6 vertices
3. choose randomly any edge
   let say 'bd'
   move these vertices b & d to C
   i.e. $C = \{b, d\}$
   and remove edges associated with vertices b & d and show them with the dotted lines
   these edges are ab, ad, bd & de
   Now $E = \{af, cf, ce, ef\}$



4. Now again pick random edge, let say cf
   associated edges are cf, ef, ce & af
   $C = \{b, d, c, f\}$
   & $E = \phi$

Graph becomes



Runtime is
~~O(VE)~~
$O(V+E)$

The Set Covering Problem

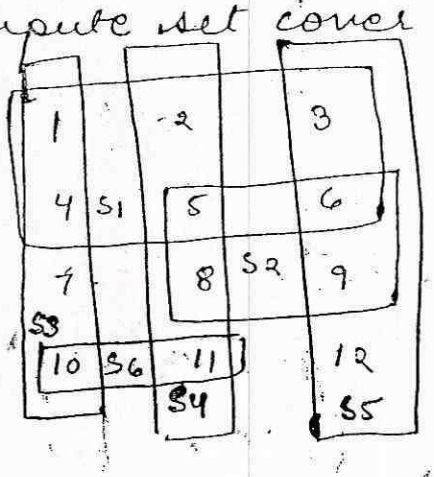
An instance (X, F) of the set-covering problem consists of a finite set X and a family F of subsets of X, such that every element of X belongs to at least one subset in F.

GREEDY-SET-COVER(X, F)

1. $U \leftarrow X$ — set of U
2. $C \leftarrow \phi$
3. while $U \neq \phi$
4. do select an $S \in F$ that maximizes $|S \cap U|$
5. $U \leftarrow U - S$
6. $C \leftarrow C \cup \{S\}$
7. Return C.

Runtime is $O(|X||F|)$

Q- Compute set cover for the given instance



$S = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12$

$S_1 = 1, 2, 3, 4, 5, 6$

$S_2 = 5, 6, 8, 9$

$S_3 = 1, 4, 7, 10$

$S_4 = 2, 5, 8, 11, 7$

$S_5 = 3, 6, 9, 12$

$S_6 = 10, 11$

minimum set cover size is 3 = S_3, S_4, S_5

Greedy set cover approach gives us

S_1, S_4, S_5, S_3

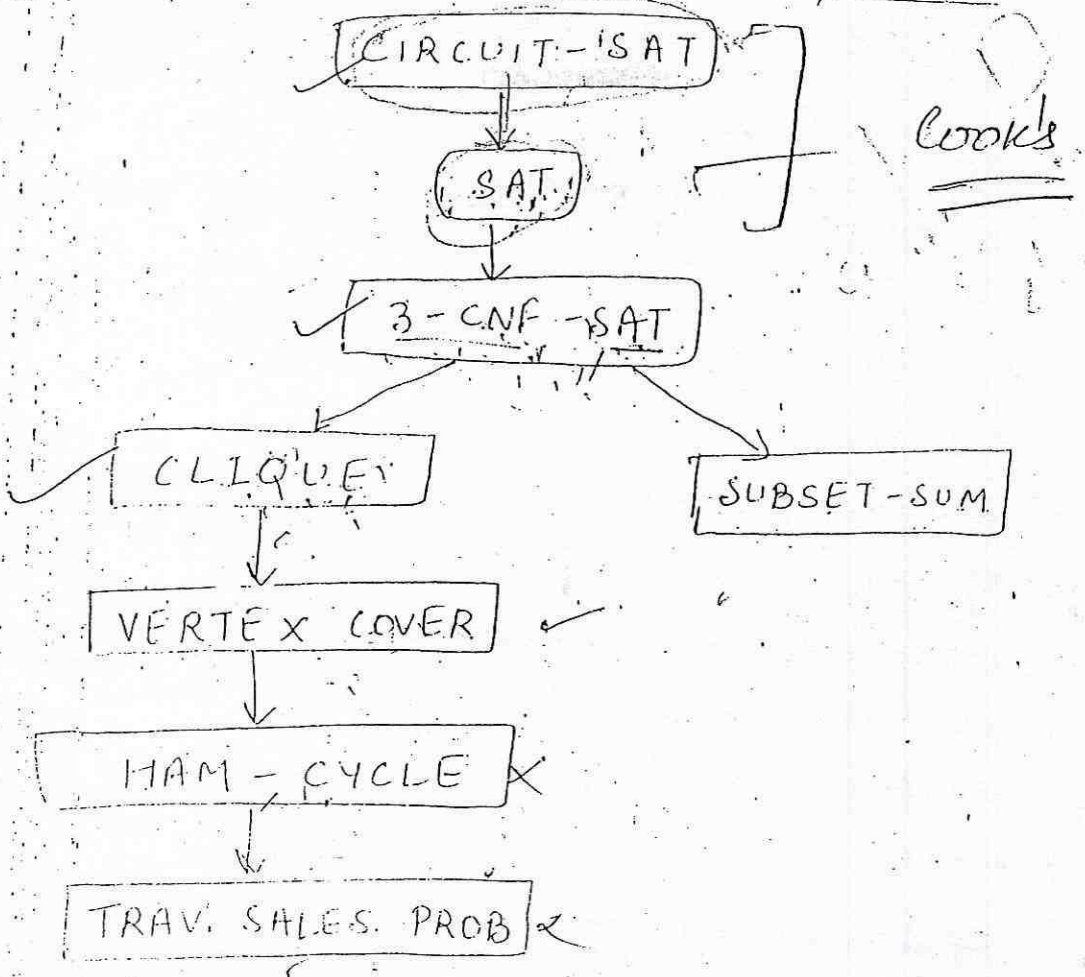
$U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$

max S_i with $|S_i \cap U| = S_1$

$U - S_1 = \{7, 8, 9, 10, 11, 12\}$

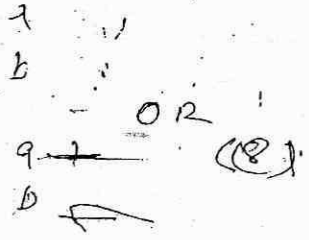
and so on.

Reducible Structure for NP-complete



Now as we have proved that Circuit-Sat is NP Complete. Thus, we can easily prove that other problems are also NP complete by reducing them

Circuit-Satisfiability is reduced to Satisfiability. Thus making it NP complete and so on.



$\{(x_1 \to x_2) \vee ((\neg x_1 \leftrightarrow x_3) \vee x_4)\} \wedge \neg x_2$

Thus SAT is NP complete.
Now we will reduce it to 3-CNF-SAT to make it NP complete

To prove that 3-CNF-SAT is NP complete

$SAT \leq_p$ 3-CNF SAT

To convert or reduce SAT to 3-CNF SAT we follow these steps.

(1) Construct a binary parse tree for the input with literals as leaves and connectives as internal nodes. Apply associativity such that each internal node has at most two children. Now we associate a variable y_i to each internal node to hold its output.

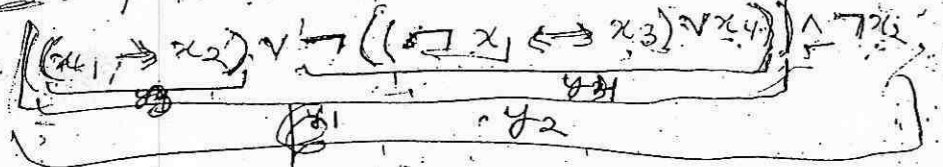
(2) Rewrite the original formula as the AND of root and a conjunction of clauses describing the operation of each node. Call it f'.

(3) Convert each clause into CNF.

(4) Transform the formula so that each clause contains exactly three literals.

Hence SAT $\leq_p$ 3-CNF SAT

for eg :- $f = \{(x_1 \to x_2) \vee ((\neg x_1 \leftrightarrow x_3) \vee x_4)\} \wedge \neg x_2$



$\{(x_1 \to x_3) \vee ((\neg x_1 \leftrightarrow x_3) \vee x_4)\} \wedge \neg x_2$

- truth table wherever we get 0, we consider formula in Disjunctive Normal form (DNF)

→ OR of AND's which is equivalent to $\neg\phi_i'$

$$\phi_1' = (y_1 \leftrightarrow (y_2 \wedge \neg x_2))$$

$$\neg\phi_1' = (y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2)$$

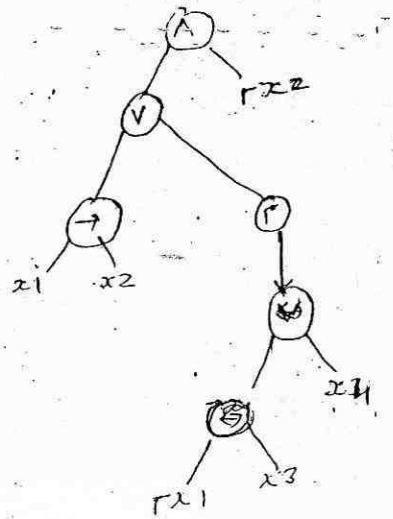
Applying DeMorgan's Law we get the formula in CNF

$$\phi_1'' = (\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2)$$

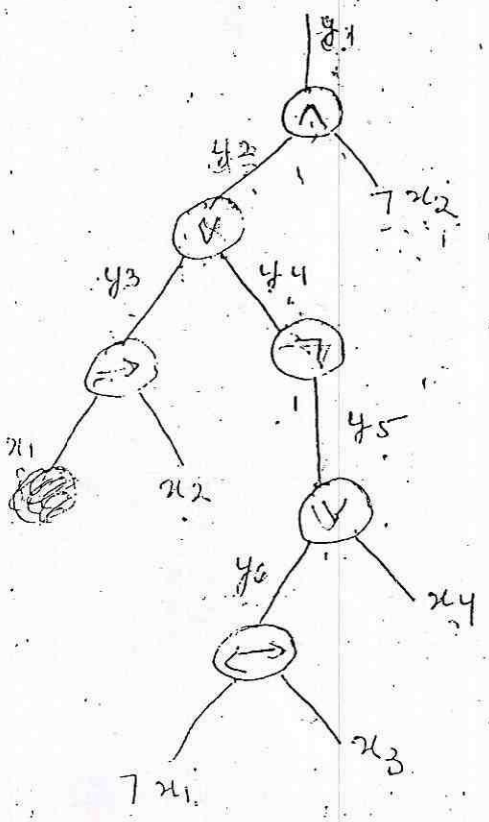
Each clause has exactly 3 literals.

3 CNF

$$\left[(x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)\right] \wedge \neg x_2$$



SPARSY



NOT

x	$\bar{x}$
0	1
1	0

$x \wedge y$ → AND

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

$$f = y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \wedge (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \wedge (y_4 \leftrightarrow \neg y_5) \wedge (y_5 \leftrightarrow (y_6 \vee x_4)) \wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))$$

0100 1011 HB

→ To convert each clause into CNF, we will make a possible truth table for it.

For, eg -

y_1	y_2	x_2	~~$y_1 \leftrightarrow$ (illegible)~~	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	1	1	0 ✓ 1	1 0
1	1	0	1	0
1	0	1	0 ✓	1 ↔ (1 ∧ 0
1	0	0	0 ✓	1 ↔ 0 = 0
0	1	1	1	
0	1	0	0 ✓	1 ↔ (1 ∧ 1)
0	0	1	1	1 ↔ 1 = 1
0	0	0	1	

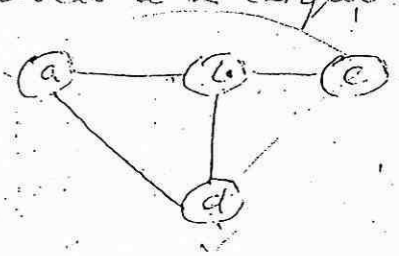
HB

HB

1011

Clique Problem

Definition of Clique :- In an undirected graph, a subgraph wherein each node is connected to each other node in the subgraph is a clique. Given an integer K and a graph, the problem is to determine that there is a clique of size K anywhere in the graph. A clique of size K is called a K clique.



A graph with a 3-clique

Proof that clique is NP-complete

(1) clique $\in$ NP

As it can be verified in polynomial time that the clique which we have formed is correct or not, we can say that clique $\in$ NP.

(2) To prove that

3-CNF SAT $\leq_p$ CLIQUE

To reduce 3-CNF SAT to pCLIQUE we will be given a combination of clauses, with 3 literals in each clause as it is in 3-CNF SAT.

→ We will construct a graph G based on the combinational clause ϕ.

→ Let K be the number of clauses in ϕ

→ Graph G will have a clique of size K if ϕ is satisfiable.

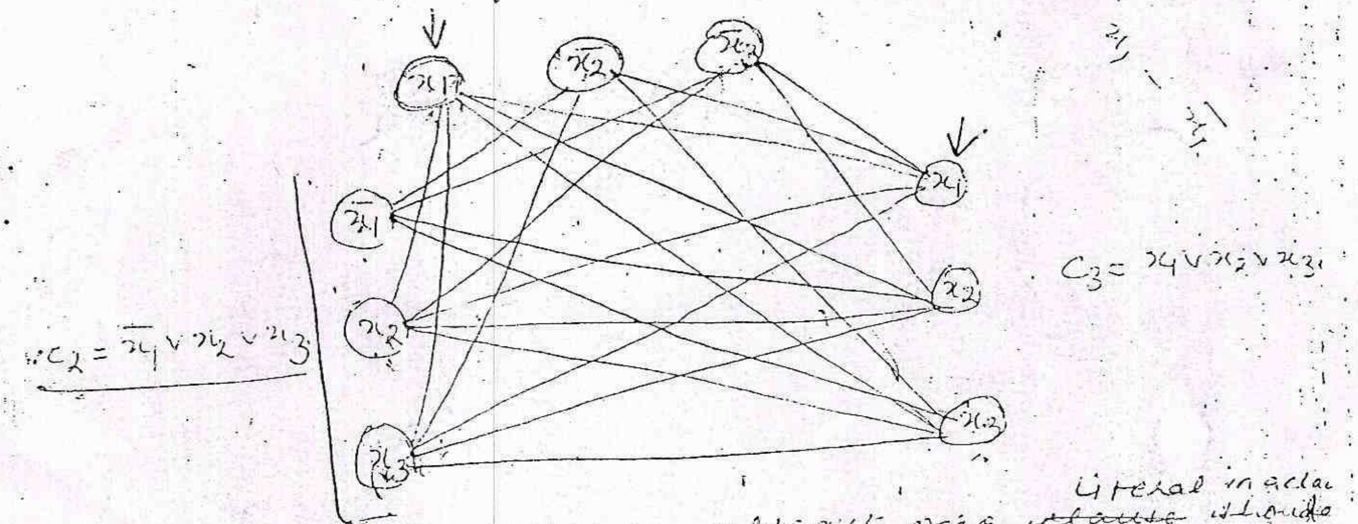
Construction of G

Point to be Noted*:- The nodes will all be connected, except that no nodes from the same clause will be connected and no two nodes with contradictory labels will be connected i.e. $(x_1 \& \bar{x}_1)$

For e.g. C_1 C_2 C_3

$\phi = (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$

$C_1 = x_1 \vee \bar{x}_2 \vee \bar{x}_3$



3CNF to be satisfiable atleast one literal in a clause should have truth assignment to make each clause true. Picking such literals from C_1, C_2 & C_3 we make a set V' of k vertices. We claim that V' is a clique.