Q.S
R.S
Strassen. (1)

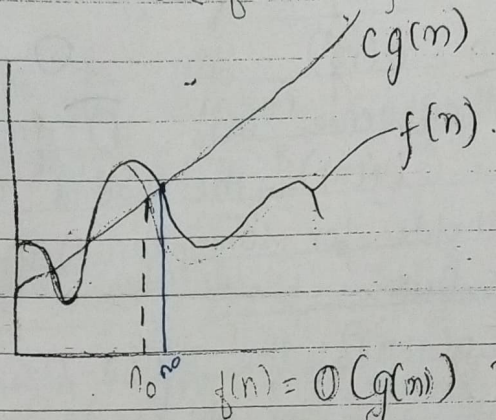
2) O-notation Worst case

- It is used for asymptotic upper bound.

For a given function $g(n)$ we denoted by $O(g(n))$ set of functions.

$O(g(n)) = f(n)$ such that there exist positive constants c and n_0 such that

$0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$.



→ For all values n to the right of n_0, the value of function $f(n)$ is on or below $cg(n)$.

→ $f(n) = O(g(n))$ indicates that a function $f(n)$ is a member of the set $O(g(n))$.

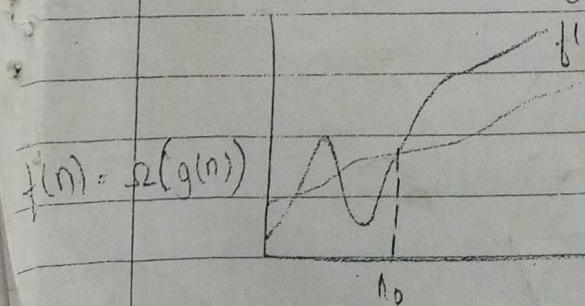
* $f(n) = O(g(n))$ implies $f(n) = O(g(n))$ since Θ notation is a stronger notation than $O(g(n))$

3) Ω - notation. Best case

For a given function $g(n)$ we denote by $\Omega(g(n))$ the set of functions.

$\Omega(g(n)) = f(n)$ such that there exists positive constants c & n_0 such that

$0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$



For all values n to the right of n_0, the value of $f(n)$ is on or above $cg(n)$.

4 o-notation :- (little-oh)

o-notation is used to denote an upper bound that is not asymptotically tight.

$o(g(n)) = f(n)$ for any positive constant c there exists a constant $n_0 > 0$ such that $0 \le f(n) < cg(n)$ for all $n \ge n_0$

For eg:- $2n = o(n^2)$ but $2n^2 \neq o(n^2)$?

The main difference between O and o is that :-

in $f(n) = O(g(n))$, the bound $0 \le f(n) \le cg(n)$ holds for some constant $c > 0$.

But

$f(n) = o(g(n))$, the bound $0 \le f(n) < cg(n)$ holds for all constants $c > 0$.

5 ω-notation (little-omega)

$\omega(g(n))$ is defined as the set

$\omega(g(n)) = f(n)$ for any positive constant $c > 0$, there exists a constant $n_0 > 0$ such that $0 \le cg(n) < f(n)$ for all $n \ge n_0$.

For eg:- $\frac{n^2}{2} = \omega(n)$ but $\frac{n^2}{2} \neq \omega(n^2)$

Also,

$f(n) \in \omega(g(n))$ iff $g(n) \in o(f(n))$

I SUBSTITUTION METHOD

→ The substitution method for solving recurrences entails 2 steps:

1) Guess the form of solution
2) Use mathematical induction to find the constants and show that the solution works.

→ The substitution method can be used to establish either upper or lower bounds on a recurrence.

Ex:- Determine an upperbound on the recurrence

$T(n) \Rightarrow 2T\left(\frac{n}{2}\right) + n$ *

Soln:- Guess that the solution is, $T(n) = O(n \log n)$

Acc. to definition; $f(n) = O(g(n))$ and $0 \le f(n) \le cg(n)$

Thus,

$f(n) = T(n)$

$g(n) = n \log n$

Hence prove that :- $0 \le T(n) \le cn \log n$

Assume that this bound holds for $\left(\frac{n}{2}\right)$

i.e $T\left(\frac{n}{2}\right) \le c\left(\frac{n}{2}\right) \log\left(\frac{n}{2}\right)$

Substituting into the recurrence yields; (i.e from eqn *)

$T(n) \le 2\left[c\left(\frac{n}{2}\right) \log\left(\frac{n}{2}\right)\right] + n$

$\le \frac{2cn}{2} \log\left(\frac{n}{2}\right) + n$

$\le cn \log\left(\frac{n}{2}\right) + n$

$= cn \log n - cn \log 2 + n$

$= cn \log n - cn + n$

$\le cn \log n$ for $(c \ge 1)$

Thus, $0 \le T(n) \le cn \log n$

(1)

2) Show that the solution of $T(n) = T(\frac{n}{2}) + 1$ is $O(\log n)$

ie $T(n) = O(\log n)$

According to the defn of Big-Oh;

$f(n) = O(g(n))$ where $f(n) \leq cg(n)$ $n \geq n_0$

Thus, here:-

$f(n) = T(n)$

$g(n) = \log n$

∴ We have to prove that :- $T(n) \leq c\log n$

Assume that this bound holds for $(\frac{n}{2})$

i.e $T(\frac{n}{2}) \leq c\log(\frac{n}{2})$

Substituting into the recurrence yields:-

$T(n) \leq c\log(\frac{n}{2}) + 1$

$= c\log n - c\log_2 2 + 1$

$= c\log n - c + 1$

∴ $T(n) \leq c\log n$ if $c \geq 1$?

$n_0 \geq 2$ ∴

3. Show that the solution to $T(n) = 2T(\frac{n}{2} + 17) + n$ is $O(n\log n)$

ie $T(n) = O(n\log n)$

According to the definition of Big-Oh;

$f(n) = O(g(n))$ where $f(n) \leq cg(n)$ for n

Thus;

$f(n) = T(n)$

$g(n) = n\log n$

∴ We have to prove that; $T(n) \leq cn\log n$ for n

Assume that this bound holds for $(\frac{n}{2})$

ie $T(\frac{n}{2}) \leq c\frac{n}{2}\log(\frac{n}{2})$

Substituting into the recurrence yields:-

$T(n) \leq 2\left(c\frac{n}{2}\log(\frac{n}{2}) + 17\right) + n$

$\leq \frac{2cn}{2}\log(\frac{n}{2}) + 34 + n$

$\leq cn\log(\frac{n}{2}) + 34 + n$

$= cn\log n - cn\log_2 2 + 34 + n$

$T(n) = cn\log n - cn + 34 + n$

∴ $T(n) \leq cn\log n$

for $c > 1$?

$n_0 = 2$ ∴

4) Solve the recurrence $T(n) = 2T(\sqrt{n}) + 1$ by making a change of variables. Your solution should be

$T(n) \leq c \log n$

RECURSION TREE METHOD:

- In a recursion tree, each node represents the cost of a single subproblem somewhere in the set of recurrence function invocation.
- We sum the costs within each level of the tree to obtain the set of per level costs.
- Sum up all per-level costs to determine the total cost of all levels of the recursion.
- Recursion trees are particularly useful when the recurrence describes the running time of a divide and conquer algorithm.
- A recursion tree is the best used to generate a good guess, which is then verified by the substitution method.

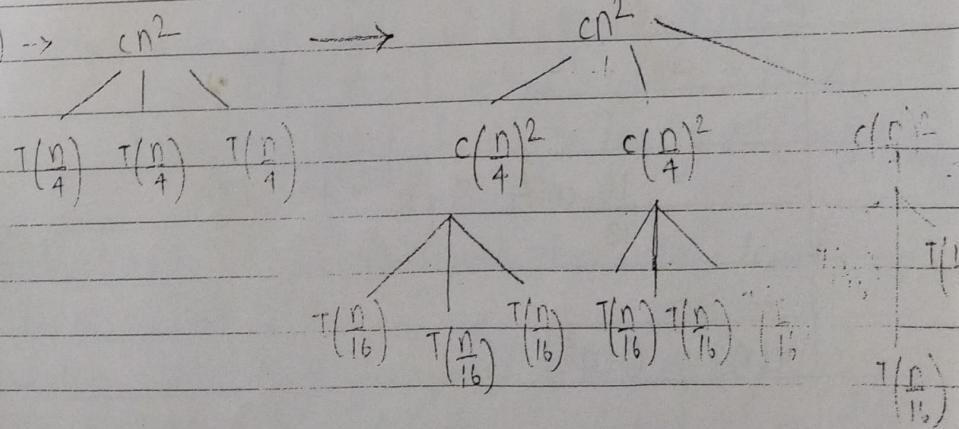
Example:-

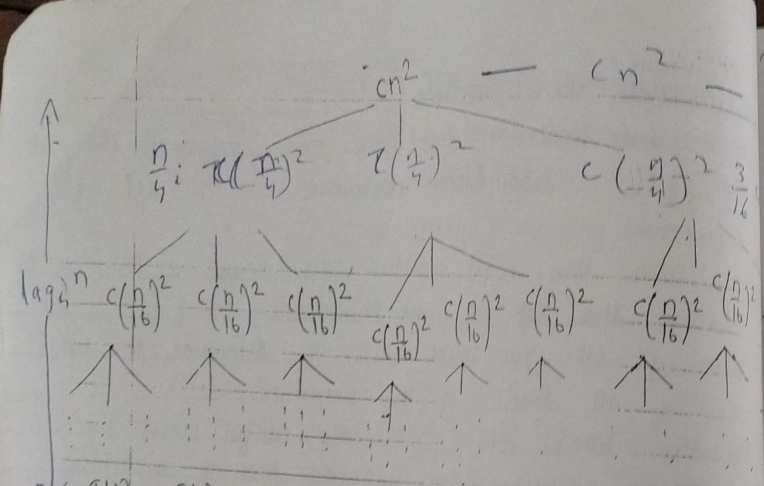
→ Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 3T\left(\frac{n}{4}\right) + cn^2$

Use substitution method to verify your answer.

Soln

) Recursion tree for the given recurrence.





$T(n) = cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 + \dots + (\frac{3}{16})^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3})$ Total

Add up the costs over all levels to determine cost of the entire tree.

$$T(n) = \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

level = $\frac{n}{4^i}$

$i = \log_4 n$

Eq (A.5)

$$< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

Eq (A.6)

$$= \frac{1}{1-\frac{3}{16}} cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3})$$

$T(n) = O(n^2)$

Substitution method.

$T(n) \leq 3T(\frac{n}{4}) + cn^2$ and solution ; $T(n) = O(n$

$g(n) = n^2$

since $f(n) \leq cg(n)$.

$(\frac{3}{16})^2 cn^2$

Thus, $T(n) \leq$ ~~d n~~ dn^2

Assume that this bound holds for $(\frac{n}{4})$

$\Theta(n^{\log_4 3}) \therefore T(\frac{n}{4}) = d(\frac{n}{4})^2$

Substituting into the recurrence yields;

$T(n) \leq 3d(\frac{n}{4})^2 + cn^2$

$\leq \frac{3dn^2}{16} + cn^2$

$= \frac{3}{16} dn^2 + cn^2$

$T(n) \leq dn^2$ when $d \geq \frac{16}{3}c$?

$T(\frac{n}{4^i})$

-> Subproblem size for node at depth i is $n/4^i$

-> Subproblem size hit $n = 1$ when $n/4^i = 1$ or when $i = \log_4 n$ total level => $\log_4 n + 1$ $(0, 1, 2 \dots \log_4 n)$

-> Each level has 3 times more node than above level at depth i is 3^i

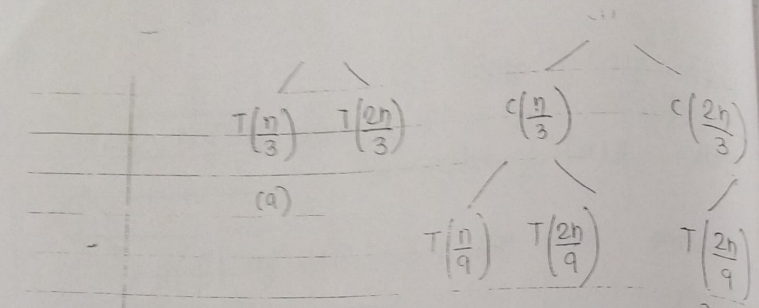
-> Subproblem reduce by 4 from root at each depth $i = 0, 1, 2 \dots$ Each node cost => $c(\frac{n}{4^i})^2$ Total $3^i c(\frac{n}{4^i})^2$

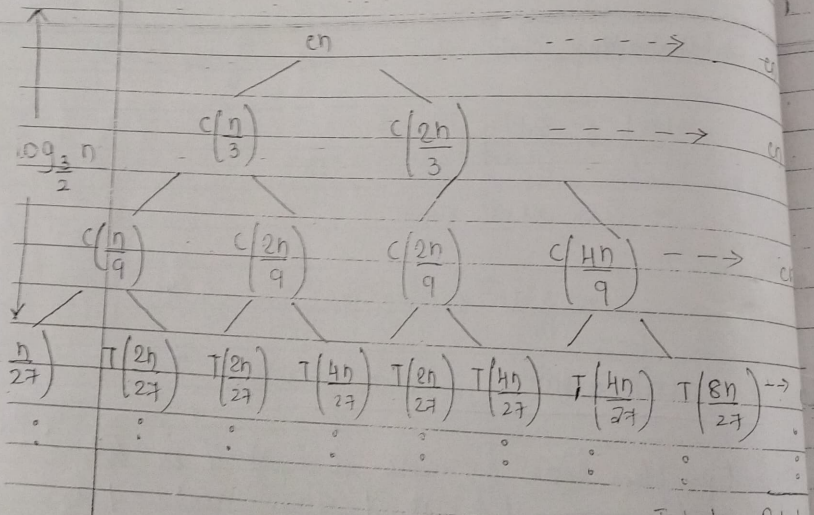
$= (\frac{3}{16})^i cn^2$ ∴ at bottom at depth $\log_4 n$

$3^{\log_4 n}$ -> cost $T(1)$ $n^{\log_4 3} T(1)$

$\Theta(n^{\log_4 3})$

Ex 2 $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n)$





Total : $O(n \log n)$

Assume that the solution is, $T(n) = O(n \log n)$
Thus, $f(n) = T(n)$

Hence we have to prove that; $f(n) \leq dg(n)$
ie $T(n) \leq dn \log n$

Assume that this bound holds for $\left(\frac{n}{3}\right)$ & $\left(\frac{2n}{3}\right)$

$\therefore T\left(\frac{n}{3}\right) \leq d\frac{n}{3}\log\left(\frac{n}{3}\right)$ & $T\left(\frac{2n}{3}\right) \leq d\frac{2n}{3}\log\left(\frac{2n}{3}\right)$

Substituting into the recurrence yields.

$$T(n) \leq d\frac{n}{3}\log\left(\frac{n}{3}\right) + d\frac{2n}{3}\log\left(\frac{2n}{3}\right) + cn$$

$$\leq \left[\frac{dn}{3}\log n - \frac{dn}{3}\log 3\right] + \left[\frac{d2n}{3}\log n - \frac{d2n}{3}\log\right]$$

$$\leq dn\log n - \frac{dn}{3}\log 3 +$$

Master Method

→ The master method provides a "cookbook"

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

where $a \geq 1$, $b > 1$ are constants

$f(n)$ is an asymptotically positive function

→ The above recurrence describes the running time of an algorithm that divides a problem of size n into a subproblems, each of size $\frac{n}{b}$.

The a subproblems are solved recursively each in time $T\left(\frac{n}{b}\right)$.

$f(n)$ is the cost of dividing the problem & combining the results of the subproblem.

MASTER THEOREM:-

Let $a \geq 1$ and $b > 1$ be constants, $f(n)$ is a function, and let $T(n)$ be defined on the non-negative integers by the recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + f(n).$$

$T(n)$ can be bounded asymptotically as follows:-

Case 1:-

$$T(n) = \Theta(n^{\log_b a}) \text{ if } f(n) = O(n^{\log_b a - \epsilon})$$

$\epsilon > 0$

For this case, $f(n)$ must be smaller than $n^{\log_b a}$

$f(n) < n^{\log_b a}$ $f(n)$ must be polynomially smaller

(means)

Case 2:-

$$T(n) = \Theta(n^{\log_b a} \log n) \text{ if } f(n) = \Theta(n^{\log_b a}).$$

This case is applied only when the two functions $f(n)$ & $n^{\log_b a}$ are of same size

Case 3:-

$$T(n) = \Theta(f(n)) \text{ if } f(n) = \Omega(n^{\log_b a + \epsilon}) \text{ for } \epsilon >$$

AND if $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ & sufficiently large n

$f(n)$ must be larger than $n^{\log_b a}$ and must be polynomially larger and should also satisfy the "regularity" condition i.e $af\left(\frac{n}{b}\right) \leq cf(n)$.

Example :-

1.

a) $T(n) = 9T\left(\frac{n}{3}\right) + n$

Comparing the given recurrence with Std we get =

$a = 9$

$b = 3$

$f(n) = n$

To find:- $n^{\log_b a}$

$\log_b a = \log_3 9$

$= \log_3 3^2$

$= 2 \log_3 3$

$\log_b a = 2$

$\therefore n^{\log_b a} = n^2$

$f(n) < (a) f(n)$

Thus $n \leq n^{\log_b a}$

~~Given~~ $f(n) \leq n$ Thus $n < n^2$

$\therefore f(n) < n^{\log_b a}$ and it is also polynomially smaller

Hence, case 1 is applied.

$\therefore T(n) = \Theta(n^2)$

b) $T(n) = T\left(\frac{2n}{3}\right) + 1$

$a = 1$ $\quad f(n) = 1$

$b = \frac{3}{2}$

To find:- $n^{\log_b a}$

$\therefore \log_b a = \log_{3/2} 1$

$\log_b a = 0$

Thus,

$n^{\log_b a} = n^0$

$n^{\log_b a} = 1$

Comparing $f(n)$ & $n^{\log_b a}$ we conclude that

$f(n) = n^{\log_b a}$

Thus case 2 applies. $\Rightarrow T(n) = \Theta(n^{\log_b a} \log n)$

$T(n) = \Theta(\log n)$

c) $T(n) = 3T\left(\frac{n}{4}\right) + n\log n$

Soln:- Comparing it:

$a = 3$

$b = 4$

$f(n) = n\log n$

To find:- $n^{\log_b a}$

$\therefore \log_b a = \log_4 3$

$= \frac{\log 3}{\log 4}$

$\log_b a = 0.793$

Thus,

$n^{\log_b a} = n^{0.793}$

Here, $f(n) > n^{\log_b a}$ $\therefore$ case 3 applies.

CHECK FOR REGULARITY CONDITION

$af\left(\frac{n}{b}\right) \leq cf(n)$

$\therefore 3\left(\frac{n}{4}\right)\log\left(\frac{n}{4}\right)$ *

$= \left(\frac{3}{4}\right) n\log n$

$= cf(n)$ for $c = \frac{3}{4}$

Thus using case 3, $T(n) = \Theta(n\log n)$

$T(n) = \Theta(f(n))$

d) $T(n) = 4T\left(\frac{n}{3}\right) + n^2$

Soln:- Comparing it:

$a = 4$

$b = 3$

$f(n) = n^2$

To find:- $n^{\log_b a}$

$\therefore \log_b a = \log_3 4$

$= \frac{\log 4}{\log 3}$

$\log_b a = 1.261$

Thus, $f(n) > n^{\log_b a}$ $\therefore$ case 3 applies

$\therefore n^{\log_b a} = n^{1.261}$

$\therefore n^2 > n^{1.261}$

CHECK FOR REGULARITY CONDITION

$af\left(\frac{n}{b}\right) \leq cf(n)$

$\therefore af\left(\frac{n}{b}\right) = 4\left(\frac{n}{3}\right)^2$

$= 4\left(\frac{n^2}{3^2}\right) = \frac{4}{9}(n^2)$

$= \frac{4}{9}(n^2)$ $\therefore \frac{4}{9}n^2 \leq cf(n^2)$ for $c \geq \frac{4}{9}$

Thus using case 3 :- $T(n) = \Theta(n^2)$

e) $T(n) = 3T\left(\frac{n}{4}\right) + n^2$

Soln:- Comparing the given equation

$a = 3$

$b = 4$

$f(n) = n^2$

To find:- $n^{\log_b a}$

$\therefore \log_b a = \log_4 3$

$= \frac{\log 3}{\log 4}$

$\log_b a = 0.793$

$\therefore n^{\log_b a} = n^{0.793}$

$\therefore n^2 > n^{0.793}$

Thus $f(n) > n^{\log_b a}$ $\therefore$ Case 3 applies.

CHECK FOR REGULARITY CONDITION.

$af\left(\frac{n}{b}\right) \leq cf(n)$

$\therefore af\left(\frac{n}{b}\right) = 3\left(\frac{n^2}{16}\right)$

$= \left(\frac{3}{16}\right) n^2$

$\therefore \left(\frac{3}{16}\right) n^2 \leq cn^2$ for $c = \frac{3}{16}$

Thus using case 3; $T(n) = \Theta(n^2)$

f) $T(n) = 16T\left(\frac{n}{4}\right) + n^2$

Soln:- Comparing the

$a = 16$

$b = 4$

$f(n) = n^2$

To find:- $n^{\log_b a}$

$\therefore \log_b a = \log_4 16$

$= \log_4 4^2$

$= 2 \log_4 4$

$\log_b a = 2$

$\therefore n^{\log_b a} = n^2$

Thus, $n^2 = n^2$

$\therefore f(n) = n^{\log_b a}$

Thus Case 2 is applied.

$\therefore T(n) = \Theta(n^2 \log n)$

Shortest Path Problem.

I All Pairs shortest path problem.

→ All pair shortest path

→ Uses a weight matrix $W = w_{ij}$ where

$w_{ij} = 0$ if $i = j$

the weight of the directed edge

∞ $i \neq j$ & $(i,j) \in E$

→ Compute a series of matrices $D^1, D^2, \ldots$ for $m = 1, 2, \ldots n-1$ where $D^m = d_{ij}^m$

$$d_{ij}^m = \min\left(d_{ij}^{m-1}, \min\left\{d_{ik}^{m-1} + w_{kj}\right\}\right)$$

→ The final matrix D^{n-1} contains the actual shortest path weights

→ $D^1 = W$.

→ Also find PREDECESSOR MATRIX $\Pi = (\pi_{ij})$ where

$\pi_{ij} = NIL$ if $i = j$ or there is no path from i to j

otherwise π_{ij} is the predecessor of j in some shortest path from i.

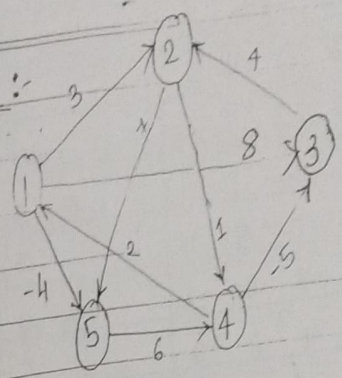
Thus

$$\pi_{ij}^0 = \begin{cases} NIL & \text{if } i = j \text{ or } w_{ij} = \infty \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty \end{cases}$$

And

$$\pi_{ij}^k = \begin{cases} \pi_{ij}^{k-1} & \text{if } d_{ij}^{k-1} \leq d_{ik}^{k-1} + d_{kj}^{k-1} \\ \pi_{kj}^{k-1} & \text{if } d_{ij}^{k-1} > d_{ik}^{k-1} + d_{kj}^{k-1} \end{cases}$$

Example :-



Step 1 :- Find W

W =	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

Step 2 :- $D^1 = W$ and Π^1

$\therefore D^1 =$

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

$\Pi^1 =$

	1	2	3	4	5
1	nil	1	1	nil	1
2	nil	nil	nil	2	2
3	nil	3	nil	nil	nil
4	4	nil	4	nil	nil
5	nil	nil	nil	5	nil

For D^2 select row 1 and row 1 of D^1 (1,1)

Step 3

$\therefore D^2 =$

	1	2	3	4	5
1	0	3	8	2	-4
2	3	0	-4	1	7
3	∞	4	0	5	11
4	2	-1	-5	0	-2
5	∞	∞	1	6	0

$\Pi^2 =$

	1	2	3	4	5
1	nil	1	1	5	1
2	4	nil	4	2	2
3	nil	3	nil	2	2
4	4	3	4	nil	1
5	4	nil	4	5	nil

Step 4:-

	1	2	3	4	5
1	0	3	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	11
4	2	-1	-5	0	-2
5	8	5	1	6	0

π^3 =

	1	2	3	4
1	nil	1	4	2
2	4	nil	4	2
3	4	3	nil	2
4	4	3	4	nil
5	4	3	4	5

Step 5:-

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

π^4 =

	1	2	3	4
1	nil	3	4	5
2	4	nil	4	2
3	4	3	nil	2
4	4	3	4	nil
5	4	3	4	5

⇒ Print path from (1,2)

(1,2)
↓
(1,3)
↓
(1,4)
↓
(1,5)
↓
(1,1)

∴ Path is
1 → 5 → 4 → 3 → 2
and cost is 1

* Print shortest path from (2,1)

(2,1)
↓
(2,4)
↓
(2,2)

∴ Path is
2 → 4 → 1
cost is 3

Algorithm :- $O(n^4)$

```
Slow-All-Pair-shortest path (W)
1.  n ← rows[W]
2.  D' = W.
3.  for i = 1 to n
4.      for j = 1 to n
5.          π'ij = i , if wij < ∞ AND i ≠ j
6.          else
7.              π'ij = nil
8.  for m = 2 to n-1
9.      do π^m, D^m ← Extend-SP(π^(m-1), D^(m-1), W)
10. return D^(n-1), π^(n-1)

Extend-SP (π, D, W),
  n ← rows(D)
  let D' = dij be nxn matrix
      π' = π'ij be nxn matrix
  for i = 1 to n do
      for j = 1 to n do
          π'ij = πij
          d'ij = dij
  for k = 1 to n
      if d'ij > dik + wkj
          d'ij = dik + wkj
          π'ij = πkj
  Return π', D'
```

```
Print-all-path-SP(π, i, j)
if (i == j)
    then print i
else
    if πij = nil
        then print "no path"
    else
        print-all-path-SP
            (π, i, πij)
        print j
```

11 THE FLOYD-WARSHALL ALGORITHM

→ The algorithm is based on the following observation. Let the vertices of directed graph G are $V = \{1, 2, 3, \ldots, n\}$. Consider a subset $\{1, 2, 3, \ldots, k\}$ for any pair of vertices i, j in V consider all paths from i to j whose intermediate vertices all taken from $\{1, 2, 3, \ldots, k\}$.

2. Compute A^k for $k = 1$ to n for n nodes.

3. While computing A^k the path from i to j is divided with an intermediate 'k'. Thus two possibilities arise :-
   1) Path going from i to j via k
   2) Path which is not going via k.
   Select only shortest path from two cases.

4. Shortest path is computed using bottom-up computation.

STEPS TO BE FOLLOWED.

Step 1:- $A^0 = W_{ij}$

Step 2:- Find $A^k_{ij} = \min\left(A^{k-1}_{ij}, A^{k-1}_{ik} + A^{k-1}_{kj}\right)$ for $k = 1$ to n where n is the no. of nodes.

Step 3:- D^n_{ij} gives the final answer.

Also find π^n_{ij}

Consider previous example

$$W = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 3 & 8 & \infty & -4 \\ 2 & \infty & 0 & \infty & 1 & 7 \\ 3 & \infty & 4 & 0 & \infty & \infty \\ 4 & 2 & \infty & -5 & 0 & \infty \\ 5 & \infty & \infty & \infty & 6 & 0 \end{array}$$

$$\pi^0 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ 1 & \text{nil} & 1 & \text{nil} & \text{nil} & 1 \\ 2 & \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ 3 & \text{nil} & 3 & \text{nil} & \text{nil} & \text{nil} \\ 4 & 4 & \text{nil} & 4 & \text{nil} & \text{nil} \\ 5 & \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{array}$$

For any D^k, select n^{th} row, that will be added to other rows.

Step 2

$$D^1 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 3 & 8 & \infty & -4 \\ 2 & \infty & 0 & \infty & 1 & 7 \\ 3 & \infty & 4 & 0 & \infty & \infty \\ 4 & 2 & 5 & -5 & 0 & -2 \\ 5 & \infty & \infty & \infty & 6 & 0 \end{array} \qquad \pi^1 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ 1 & \text{nil} & 1 & 1 & \text{nil} & 1 \\ 2 & \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ 3 & \text{nil} & 3 & \text{nil} & \text{nil} & \text{nil} \\ 4 & 4 & 1 & 4 & \text{nil} & 1 \\ 5 & \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{array}$$

Step 3

$$D^2 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 3 & 8 & 4 & -4 \\ 2 & \infty & 0 & \infty & 1 & 7 \\ 3 & \infty & 4 & 0 & 5 & 11 \\ 4 & 2 & 5 & -5 & 0 & -2 \\ 5 & \infty & \infty & \infty & 6 & 0 \end{array} \qquad \pi^2 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ 1 & \text{nil} & 1 & 1 & 2 & 1 \\ 2 & \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ 3 & \text{nil} & 3 & \text{nil} & 2 & 2 \\ 4 & 4 & 1 & 4 & \text{nil} & 1 \\ 5 & \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{array}$$

$(\infty+3)$, $(0+3)$, $(\infty+\infty)$ $(3+1)$ $(7+3)$ - 1st row

Step 4:-

$$D^3 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 3 & 8 & 4 & -4 \\ 2 & \infty & 0 & \infty & 1 & 7 \\ 3 & \infty & 4 & 0 & 5 & 11 \\ 4 & 2 & -1 & -5 & 0 & -2 \\ 5 & \infty & \infty & \infty & 6 & 0 \end{array} \qquad \pi^3 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ 1 & \text{nil} & 1 & 1 & 2 & 1 \\ 2 & \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ 3 & \text{nil} & 3 & \text{nil} & 2 & 2 \\ 4 & 4 & 3 & 4 & \text{nil} & 1 \\ 5 & \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{array}$$

D^4 =

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

π^4 =

	1	2	3	4	5
1	nil	3	4	5	4
2	4	nil	4	5	4
3	4	3	nil	5	4
4	4	3	4	nil	4
5	4	3	4	5	nil

value in π^k_{ij}

step 6:-

D^5 =

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

π^5 =

	1	2	3	4	5
1	nil	3	4	5	4
2	4	nil	4	5	4
3	4	3	nil	5	4
4	4	3	4	nil	4
5	4	3	4	5	nil

ALGORITHM:-

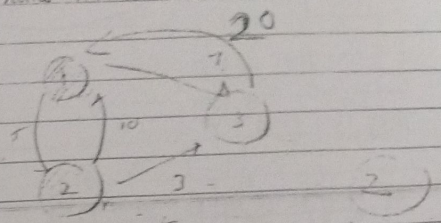
FLOYD-WARSHALL (W)

1. $n \leftarrow$ rows[W]
2. $\Theta^0 \leftarrow W$.
3. for i=1 to n do
4. for j=1 to n do
5. π^0_{ij} = NIL if i=j or $w_{ij} = \infty$.
6. else $\pi^0_{ij} = i$ if $i \neq j$ and $w_{ij} < \infty$.
7. for k ← 1 to n do
8. for i ← 1 to n do
9. for j ← 1 to n do
10. $d^k_{ij} \leftarrow \min(d^{k-1}_{ij}, d^{k-1}_{ik} + d^{k-1}_{kj})$
11. $\pi^k_{ij} \leftarrow \pi^{k-1}_{ij}$ if $d^{k-1}_{ij} \leq d^{k-1}_{ik} + d^{k-1}_{kj}$
12. else $\pi^k_{ij} \leftarrow \pi^{k-1}_{kj}$ if $d^{k-1}_{ij} > d^{k-1}_{ik} + d^{k-1}_{kj}$
13. return Θ^n, π^n

ANALYSIS

... time of the Floyd-Warshall ... is determined by the triply nested for loop in lines 7-12. Because each execution of line 10 takes O(1) time;

The algorithm runs in time $\Theta(n^3)$



$d^{(0)}$ =

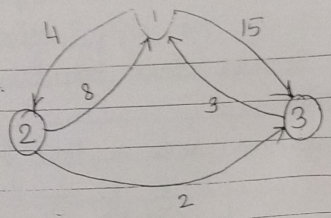
	1	2	3
1	0	7	
2	10	0	3
3	20		0

	1	2
1		1
2	2	
3	3	

	1	2
1		
2		
3		

	1	2
1		
2		
3		

Q: Compute all-pair shortest path for the following graph using Floyd-Warshall.



Step 1:- Find W

$$W = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 15 \\ 8 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix} \end{array}$$

Step 2:- $D^1 = W$

$$D^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 15 \\ 8 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix} \end{array}$$

$$\Pi^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} nil & 1 & 1 \\ 2 & nil & 2 \\ 3 & nil & nil \end{bmatrix} \end{array}$$

Step 2

$$D^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

$$\Pi^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \begin{bmatrix} nil & 1 \\ 3 & nil \\ 3 & 1 \end{bmatrix} \end{array}$$

D^2 gives the final solution

* Find shortest path from (3,1)

(3,1) → (3,3) ∴ Path is 3 → 1 Cost 3

* Path from (1,3) (1,3) → (1,2) (1,1) ∴ 1 → 2 → 3 cost = 6

Path from (3,2)

(3,2) → (3,1) ∴ Path is:- Cost = 7

(3,3)

$$D^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 15 \\ 8 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix} \end{array}$$

$$\Pi^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} nil & 1 & 1 \\ 2 & nil & 2 \\ 3 & nil & nil \end{bmatrix} \end{array}$$

Step 2:-

$$D^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 15 \\ 8 & 0 & 2 \\ 3 & 11 & 0 \end{bmatrix} \end{array}$$

$$\Pi^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} nil & 1 & 1 \\ 2 & nil & 2 \\ 3 & 1 & nil \end{bmatrix} \end{array}$$

Step 3:-

$$D^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 6 \\ 8 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

$$\Pi^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} nil & 1 & 2 \\ 2 & nil & 2 \\ 3 & 1 & nil \end{bmatrix} \end{array}$$

Step 4:-

$$D^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \end{array}$$

$$\Pi^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} nil & 1 & 2 \\ 3 & nil & 2 \\ 3 & 1 & nil \end{bmatrix} \end{array}$$

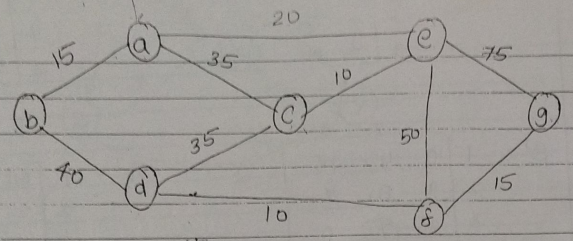
D^3 gives final solution

1 → 3 1 → 2 → 3

III SINGLE SOURCE - SHORTEST PATH

a) DIJKSTRA'S

①



write Queue !!

Queue :- (~~a~~, b, c, d, e, f, g) → choose min d Extract

Extract :- a → find adj nodes

d	a	b	c	d	e	f	g
	0	15	35	∞	20	∞	∞

π	b	c	d	e
	a	a	Nil	a

Queue (~~b~~, c, d, e, f, g) → choose min d extract

Extract :- b → find adj nodes d change value if adj node value is minimum.

d	a	b	c	d	e	f	g
	0	15	35	55	20	∞	∞

π	b	c	d	e	f
	a	a	b	a	Nil

Queue (c, d, ~~e~~, f, g)

Extract :- e

d	a	b	c	d	e	f	g
	0	15	30	55	20	70	95

π	b	c	d	e	f
	a	e	b	a	e

Queue (~~c~~, d, f, g)

Extract :- c

d	a	b	c	d	e	f	g
	0	15	30	55	20	70	95

π	b	c	d	e	f	g
	a	e	b	a	e	e

Queue (~~a~~, f, g)

Extract :- d

0	15	30	55	20	65	95

π	b	c	d	e	f	g
	a	e	b	a	d	e

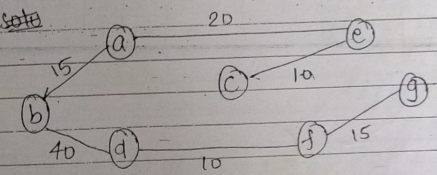
Queue (~~f~~, g)

Extract :- f

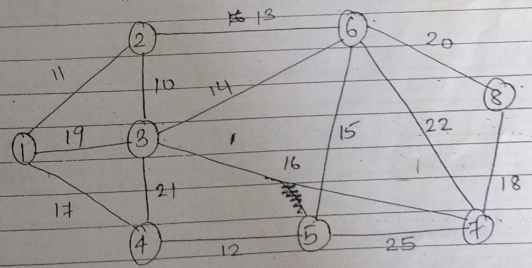
d	a	b	c	d	e	f	g
	0	15	30	55	20	65	80

π	b	c	d	e	f	g
	a	e	b	a	d	f

Sol:



②



1. Queue (~~1~~, 2, 3, 4, 5, 6, 7, 8)

Extract :- 1

d	1	2	3	4	5	6	7	8
	0	11	19	17	∞	∞	∞	∞

π	2	3	4	5	6	7	8
	1	1	1	Nil	Nil	Nil	Nil

2. Queue (~~2~~, 3, 4, 5, 6, 7, 8)

Extract :- 2

d	1	2	3	4	5	6	7	8
	0	11	19	17	∞	24	∞	∞

Π	2	3	4	5
	1	1	1	Nil

Extract :- 4

d	1	2	3	4	5	6	7	8
	0	11	29	17	29	24	∞	∞

Π	2	3	4	5	6
	1	1	1	4	2

4. Queue (5, 6, 7, 8)

Extract :- 3

d	1	2	3	4	5	6	7	8
	0	11	19	17	29	24	35	∞

Π	2	3	4	5	6
	1	1	1	4	2

5 Queue (5, 6, 7, 8)

Extract :- 6

d	1	2	3	4	5	6	7	8
	0	11	19	17	29	24	35	44

Π	2	3	4	5	6
	1	1	1	4	2

6. Queue (5, 7, 8)

Extract :- 5

d	1	2	3	4	5	6	7	8
	0	11	19	17	29	24	35	44

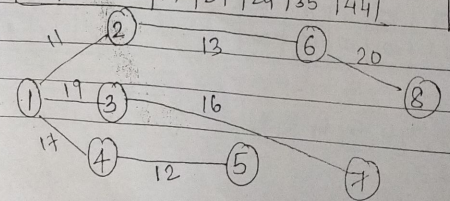
Π	2	3	4	5	6	7
	1	1	1	4	2	3

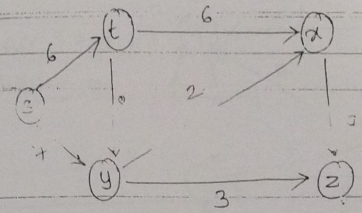
7- Queue (7, 8)

Extract :- 7, 8

d	1	2	3	4	5	6	7	8
	0	11	19	17	29	24	35	44

Π	2	3	4	5	6
	1	1	1	4	2





Queue :- (s, t, x, y, z) Extract :- s

1.	d	s	t	x	y	z
	0	6	∞	7	∞

Π	t	x	y	z
	s	Nil	s	Nil

2. Queue :- (t, x, y, z)

Extract :- t

d	s	t	x	y	z
	0	6	12	7	∞

Π	t	x	y	z
	s	t	s	Nil

3. Queue :- (x, y, z)

Extract :- y

d	s	t	x	y	z
	0	6	9	7	10

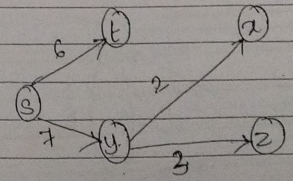
Π	t	x	y	z
	s	y	s	y

4. Queue :- (x, z)

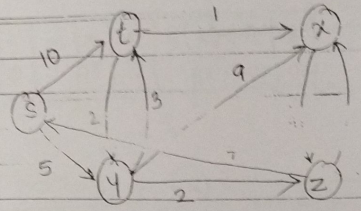
Extract :- x

d	s	t	x	y	z
	0	6	9	7	10

Π	t	x	y	z
	s	y	s	y



(4)

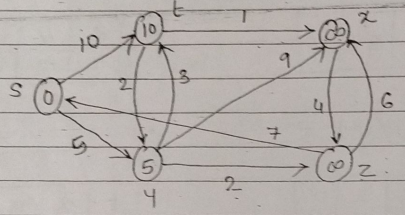


Step 1:- Queue (s, t, x, y, z)

Extract :- s

d	s	t	x	y	z
	0	10	∞	5	∞

Π	t	x	y	z
	s	Nil	s	Nil

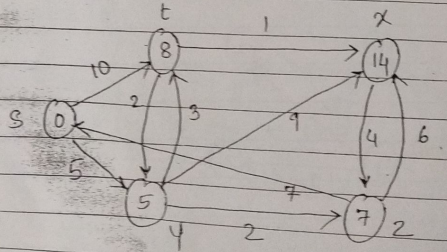


Step 2:- Queue (t, x, y, z)

Extract :- y

d	s	t	x	y	z
	0	8	14	5	7

Π	t	x	y	z
	y	y	s	y

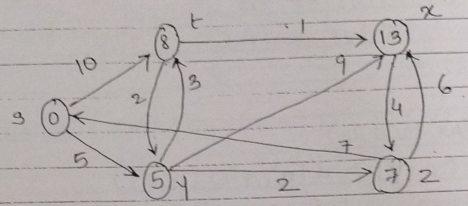


Step 3:- Queue (t, x, z)

Extract :- z

d	s	t	x	y	z
	0	8	10	∞	∞

Π	t	x	y	z
	y	t	s	y

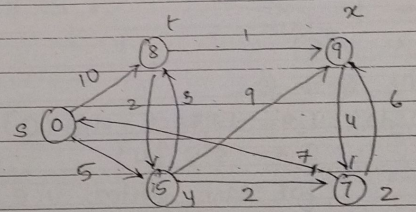


Step 4:- Queue (x, z)

Extract :- t

d	s	t	x	y	z
	0	8	9	5	7

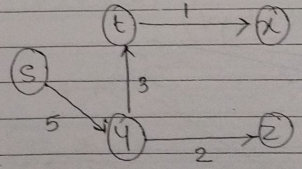
Π	t	x	y	z
	y	t	s	y



Step 5:- ~~Queue (x)~~

~~Extract z~~

Soln:-



ALGORITHM:-

INITIALIZE - SINGLE - SOURCE (G, s)

```
1  for each vertex v ∈ V[G]
2     do d[v] ← ∞
3        π[v] ← NIL
4  d[s] ← 0
```

RELAX (u, v, w)

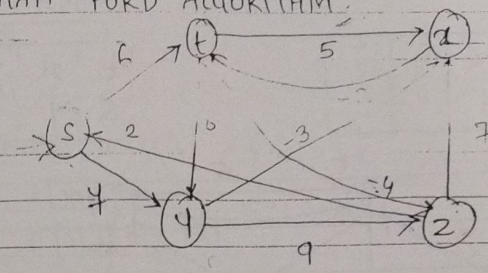
```
1  if (d[v] > d[u] + w[u,v])
2  then d[v] ← d[u] + w[u,v]
3       π[v] ← u
```

DIJKSTRA (G, w, s)

```
1  INITIALIZE - SINGLE - SOURCE (G, s)
2  S ← ϕ
3  Q ← V[G]
   while Q ≠ ϕ
      do u ← EXTRACT - MIN (Q)
         S ← S ∪ {u}
         for each vertex v ∈ Adj[u]
            do RELAX (u, v, w)
```

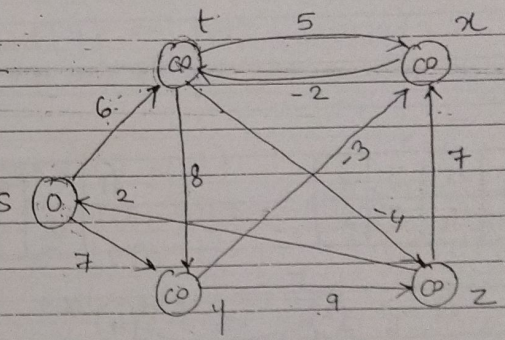
ANALYSIS $O(E \log V)$

BELLMAN - FORD ALGORITHM



Edge - list :- (s,t) (s,y) (t,x) (t,y) (t,z) (x,t) (z,x) (z,s) (y,x) (y,z).

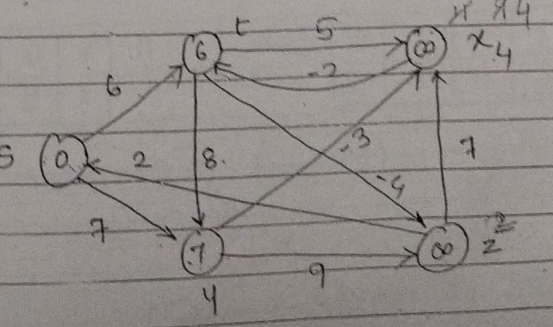
Initialize :-



Pass 1 :- Relax all the edges, if (Node dist + edge dist < Node dist) then change node dist. value. (outgoing edge) (incoming edge)

d	s	t	x	y	z
	0	6	∞	7	∞

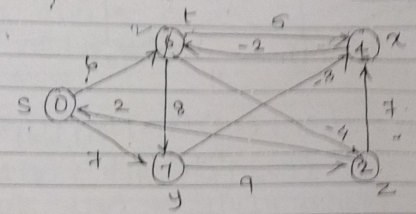
π	t	x	y	z
	s	Nil	s	Nil



Pass 2 :-

d	s	t	x	y	z
	0	6	4	7	2

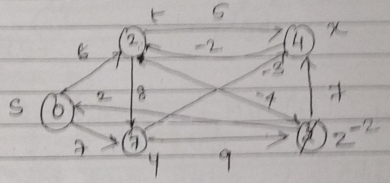
π	t	x	y	z
	s	y	s	t



Pass 3 :-

d	s	t	x	y	z
	0	2	4	7	2

π	t	x	y	z
	x	y	s	t

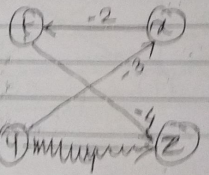


Pass 4 :-

d	s	t	x	y	z
	0	2	4	7	-2

π	t	x	y	z
	x	y	s	t

Final soln.



ALGORITHM :-

INITIALIZE - SINGLE - SOURCE (G, s)

1. for each vertex $v \in V[G]$
2. do $d[v] \leftarrow \infty$
3. $\pi[v] \leftarrow$ NIL
4. $d[s] \leftarrow 0$

RELAX (u, v, w)

1. if $d[v] > d[u] + w(u, v)$
2. then $d[v] \leftarrow d[u] + w(u, v)$
3. $\pi[v] \leftarrow u$

BELLMAN - FORD (G, w, s)

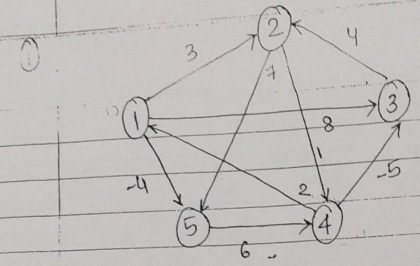
1. INITIALIZE - SINGLE - SOURCE (G, s) → $O(V)$
2. for $i \leftarrow 1$ to $|V[G]| - 1$
3. do for each edge $(u, v) \in E[G]$ } → $O(E)$
4. do RELAX (u, v, w)
5. for each edge $(u, v) \in E[G]$
6. do if $d[v] > d[u] + w(u, v)$ } → $O(E)$
7. then return FALSE
8. return TRUE

Total $O(VE)$

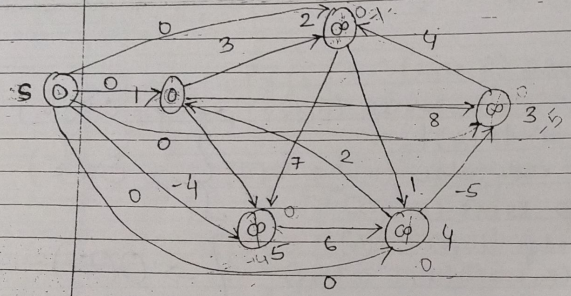
ANALYSIS :-

The Bellman - ford algorithm runs in time $O(VE)$ since the initialization in line 1 takes $O(V)$ time. Each of the $|V| - 1$ passes over the edges in lines 2-4 takes $O(E)$ time and the for loop of lines 5-7 takes $O(E)$ time.

JOHNSON'S ALGORITHM FOR SPARSE GRAPH



Step 1:- Find G' (Add extra node s' connected to all other nodes with edge weight 0)



Pass 1:-

d	1	2	3	4	5
	0	3	-1	4	-1

Pass 1:-

d	1	2	3	4	5	s
	0	0	-5	0	0	0

Pass 2:-

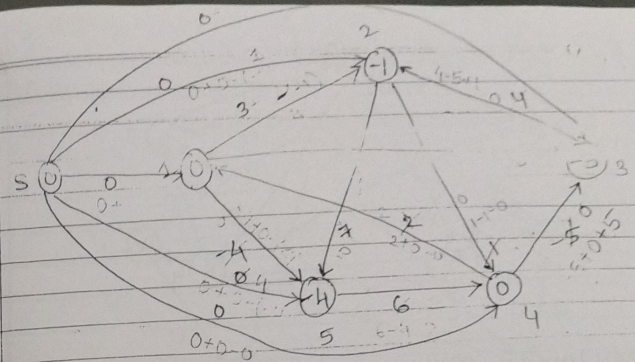
d	1	2	3	4	5	s
	0	-1	-5	0	-1	0

Pass 3:- d

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

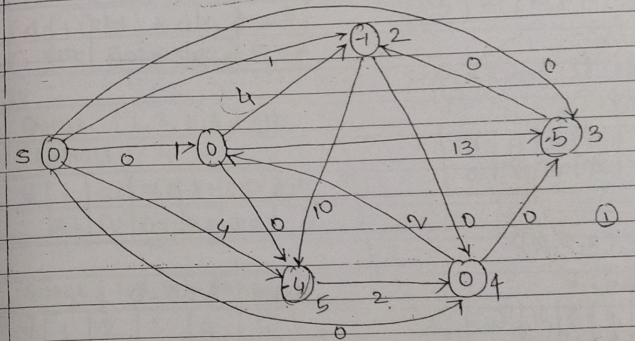
Pass 3:-

d	1	2	3	4	5
	0	-1	-5	0	-4

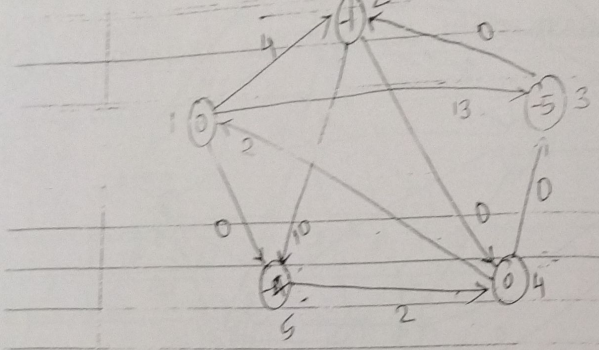


Step 2:- Relax edge weight using the following formula

$$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$$



Step 3:- Remove the extra node that was added (ie s) for next step consider only original graph edge with new weight. ~~and node~~



Step 4:- Considering source vertex as 1 runs Dijkstra

1) Queue (1,2,3,4,5)

Extract (1)

d	1	2	3	4	5
	0	4	13	∞	∅

$\delta(u,v) = \delta(u,v) + h(u) - h(v)$

$d_{12} = d_{12} + h(2) - h(1)$

$= 2 + (-1)$

$= 2-1$

$d_{12} = 1$

$d_{13} = 2 + (-5) - 0$

$= -3$

$d_{14} = 2 + 0 - 0 = 2$

$d_{15} = 0 + (-4) - 0$

$= -4$

2) Queue (2,3,4,5)

Extract : 5

d	1	2	3	4	5
	0	4	13	2	0

3) Queue (2,3,4)

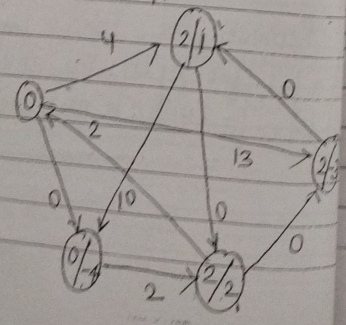
Extract : 4

d	1	2	3	4	5
	0	4	0a	2	0

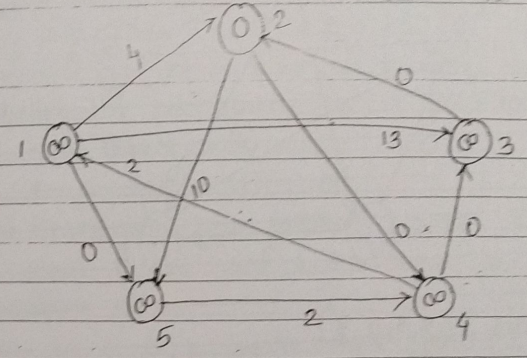
4) Queue (2,3)

Extract : 3

d	1	2	3	4	5
	0	0a	0a	2	0



Consider vertex 2 as source.



1) Queue (1,2,3,4,5)

Extract : 2

d	1	2	3	4	5
	∞	0	∞	0,2	10,2

Queue (1,3,4,5)

Extract :- 4

d	1	2	3	4	5
	2,4	0,3	0,4	0,2	10,2

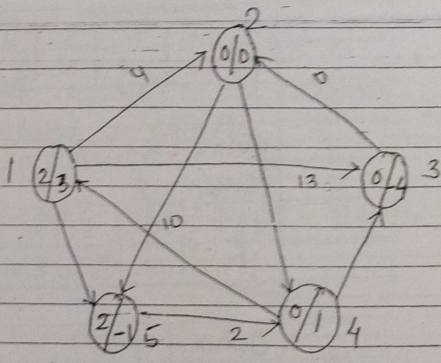
Queue (1,3,5) Extract : 3

d	1	2	3	4	5
	2,4	0,3	0,4	0,2	10,2

Queue (1,5)

Extract :- 1

d	1	2	3	4	5
	2,4	0,3	0,4	0,2	8,1



$d_{21} = 2 + 0 - (-1) = 3$

$d_{23} = 0 + (-5) - (-1)$

$= -5 + 1 = -4$

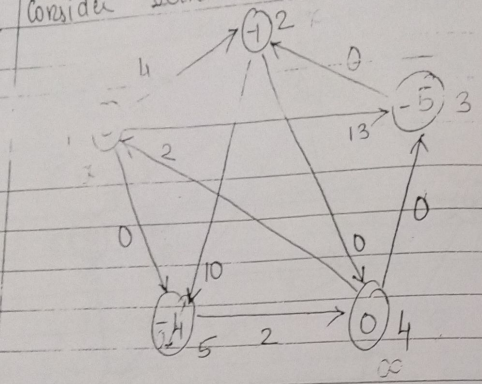
$d_{24} = 0 + 0 - (-1) = 1$

$d_{25} = 2 + (-4) + (-1)$

$= 2 - 4 + 1 = -3 + 1$

Similarly for vertex 3 & 4.

step 6 Consider vertex 5 as source



Queue (1,2,3,4,~~5~~)

Extract :- 5

d	1	2	3	4	5
	∞	∞	∞	(2,5)	0

Queue (1,2,3,~~4~~)

Extract :- 4

d	1	2	3	4	5
	4,4	∞	(2,4)	2,5	0

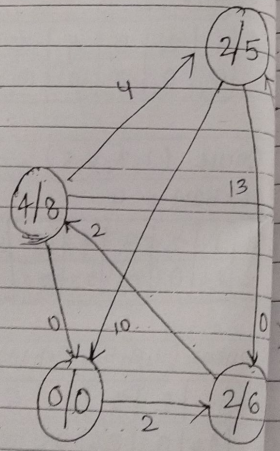
Queue (1,2,~~3~~)

Extract :- 3

d	1	2	3	4	5
	4,4	(2,3)	2,4	2,5	0

Queue (1,~~2~~) Extract :- 2

d	1	2	3	4	5
	4,4	2,3	2,4	2,5	0



IV FLOW NETWORKS

- A flow network $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has a non-negative capacity $c(u,v) \geq 0$. If $(u,v) \notin E$, we assume that $c(u,v) = 0$.
- We distinguish two vertices in a flow network
  - a source → (s)
  - a sink → (t)
- In the maximum flow problem, given a flow network with source s and sink t we have to find a flow of maximum value.
- A flow in G is a real-valued function $f : V \times V \to R$ that satisfies the following 3 properties:

CAPACITY CONSTRAINT :- For all $u, v \in V$, we require,

$$f(u,v) \leq c(u,v)$$

FLOW CONSERVATION :- For all $u \in V - \{s,t\}$ we require

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

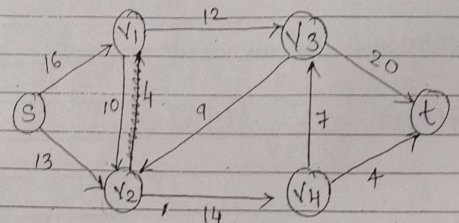
RESIDUAL CAPACITY :- $c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E. \\ f(v,u) \end{cases}$

a FORD - FULKERSON METHOD

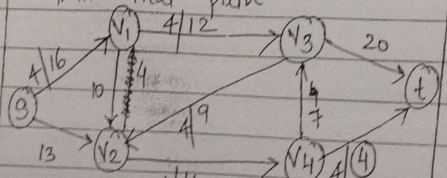
1. Arbitrarily any path is taken from source
2. Edge with lowest capacity is taken
3. Allow that capacity to flow from that
4. At each ~~int~~ iteration, we increase the flow value by finding an "augmenting path" ↓ is any (every path is selected once) path from source
5. This process is repeated until no augmenting is found.

* A flow is maximum iff its residual graph network contains no augmenting path.

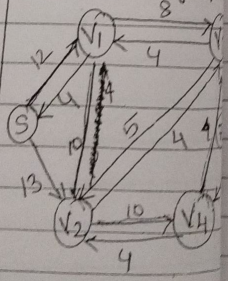
(1)



Step 1:- Select a path from source to sink ($s-v_1-v_3-v_2-v_4-t$) Find min cut and Allow that capacity to flow from that path

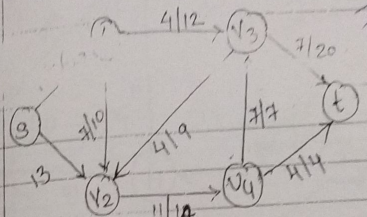


Residual graph

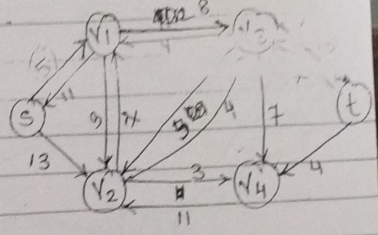


4+7-5-3+4

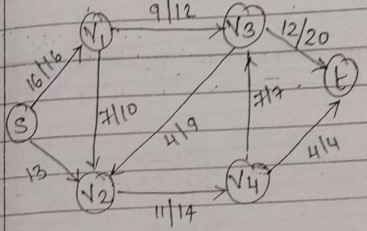
Path ($s-v_1-v_2-v_4-v_3-t$)



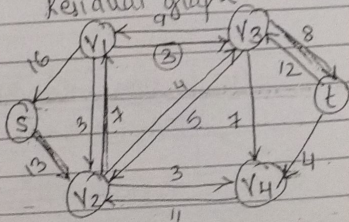
Residual graph



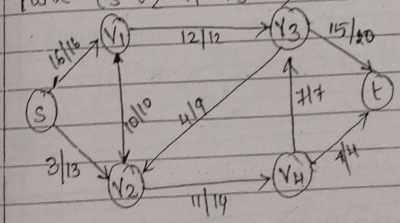
Path ($s-v_1-v_3-t$)



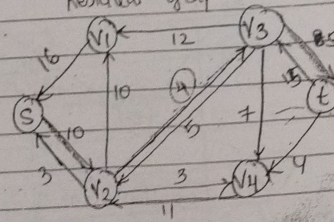
Residual graph



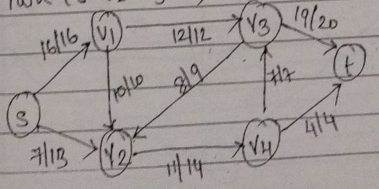
Path ($s-v_2-v_1-v_3-t$)



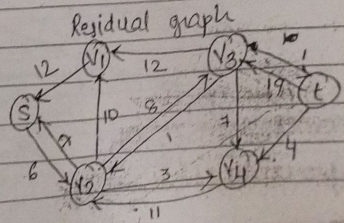
Residual graph

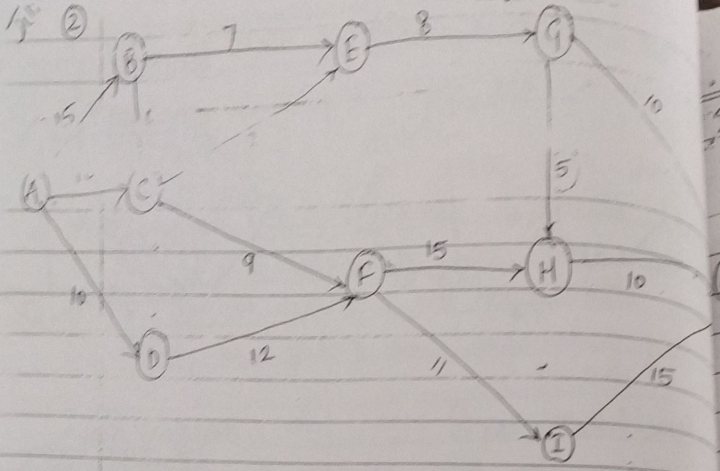


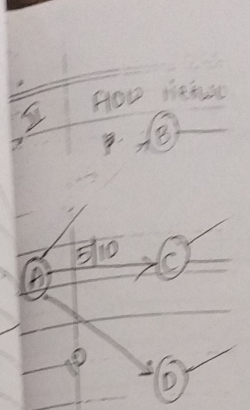
Path ($s-v_2-v_3-t$)

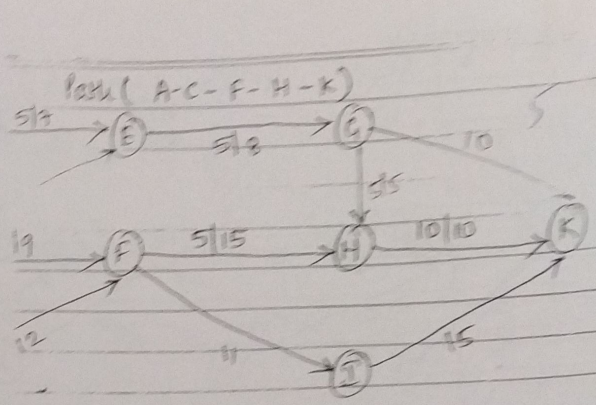


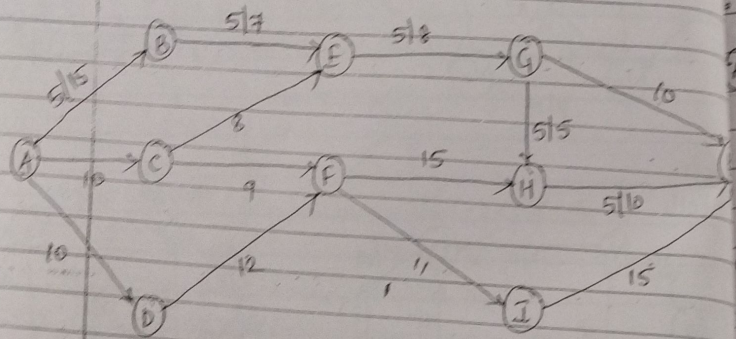
Residual graph

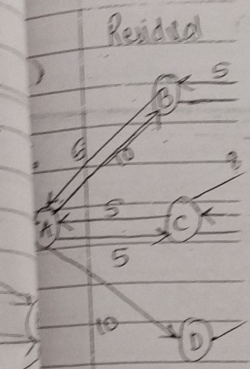


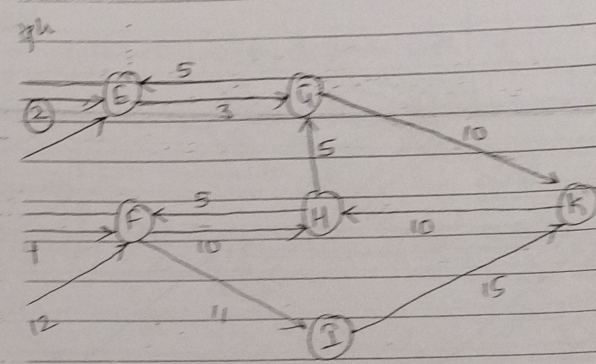

A
B
C
D
E
F
G
H
I
15
7
8
10
5
9
15
10
12
11
15

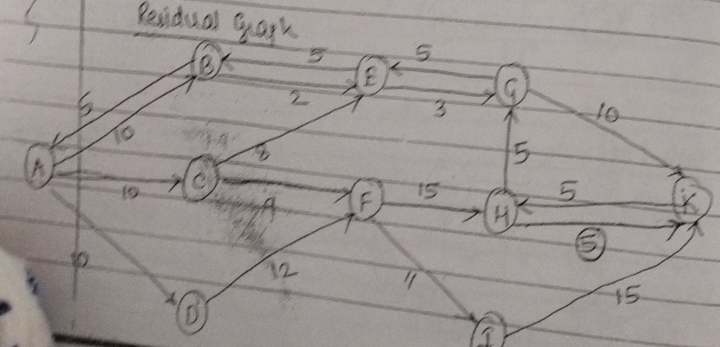

Flow network
A
B
C
D
5|10
10

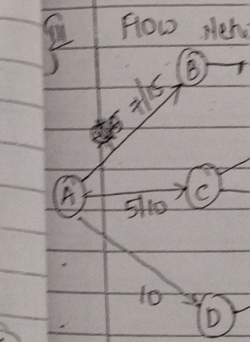

Path (A-C-F-H-K)
5|5
E
5|8
G
10
5|5
19
F
5|15
H
10|10
K
12
11
I
15

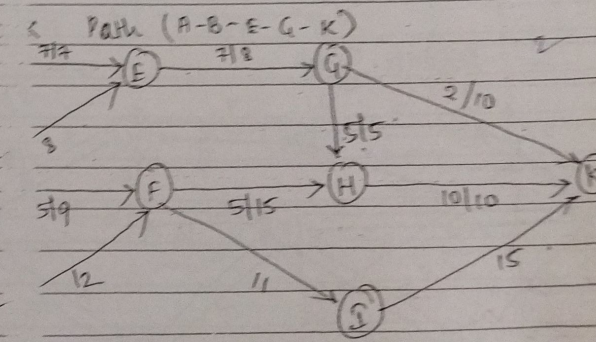

Flow network
Path (A-B-E-G-H-K)
5|7
5|8
5|15
10
5|5
15
5|10
10
12
11
15


Residual
5
5
10
5
5
10

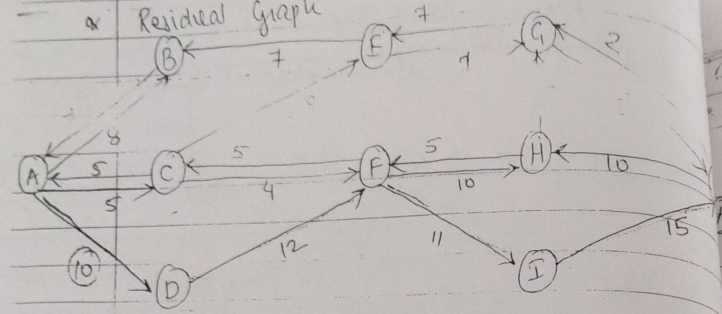

5
3
5
10
5
10
7
10
12
11
15


Residual Graph
5
5
5
2
3
10
10
5
10
15
5
5
10
12
11
15

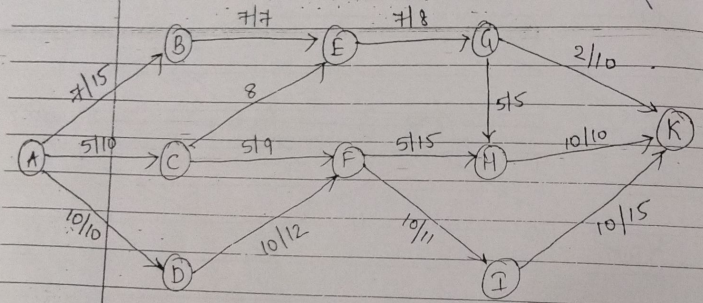

Flow network
7|15
5|10
10


Path (A-B-E-G-K)
7|7
7|8
2|10
3
5|5
19
5|15
10|10
12
11
15

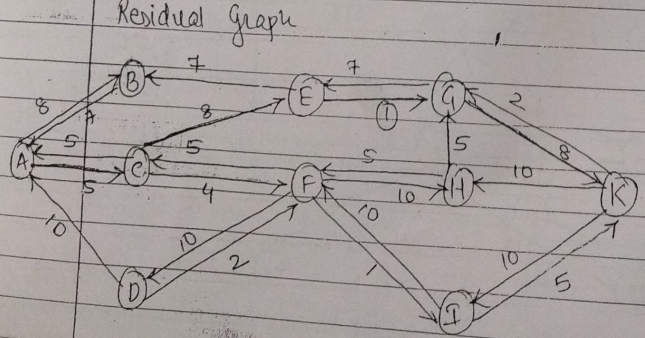
Residual Graph



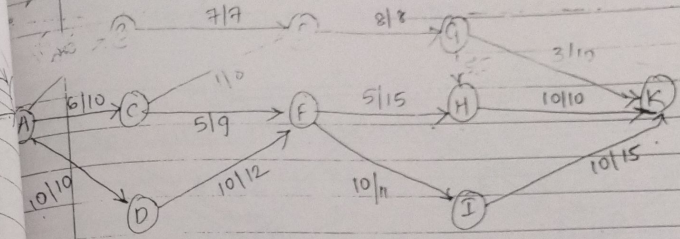
Flow Network (A-D-F-I-K)



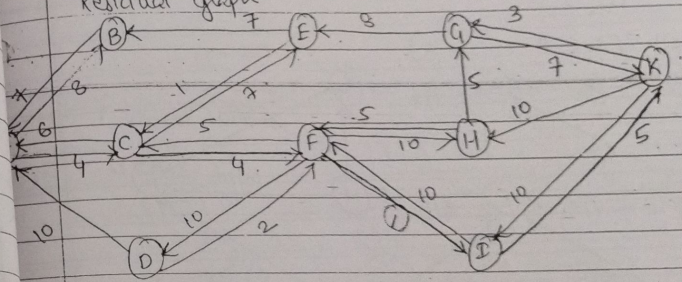
Residual Graph



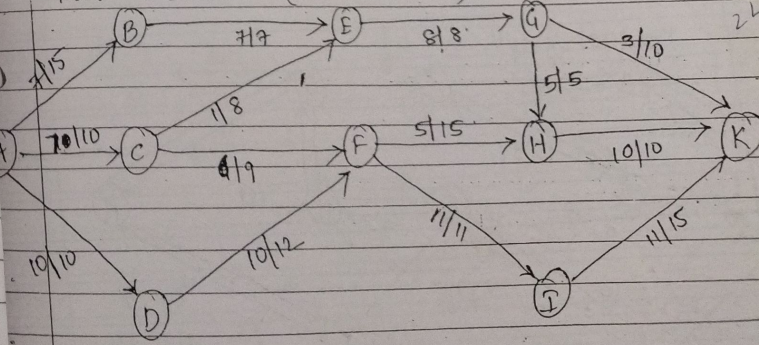
Flow Network (A-C-E-G-K)

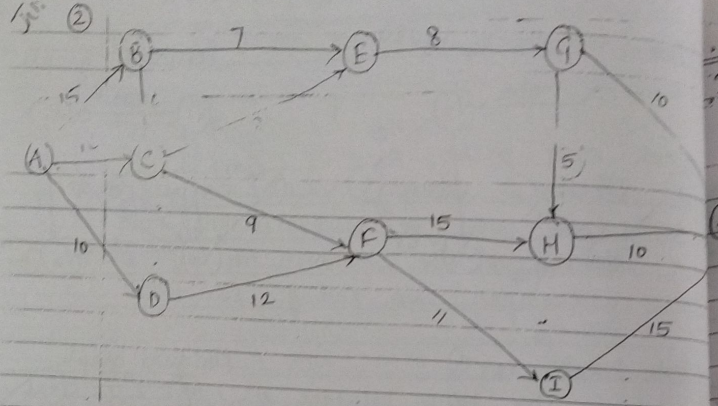


Residual Graph



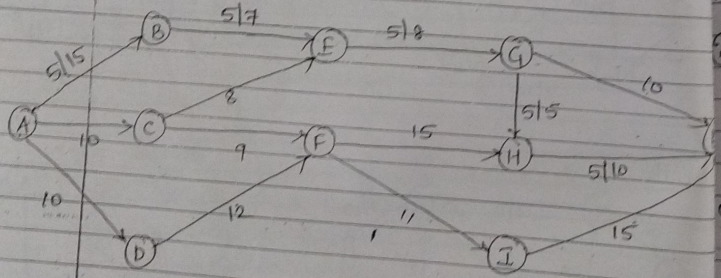
Flow Network (A-C-F-I-K)



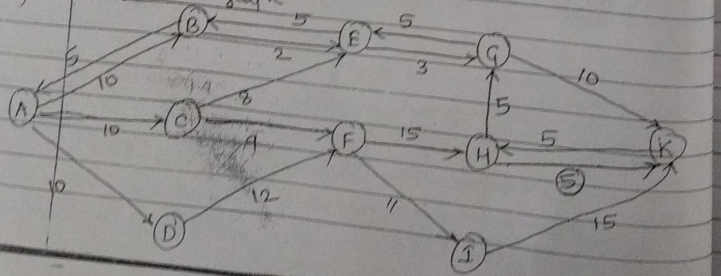


I Flow Network

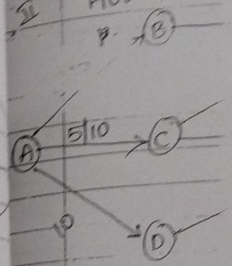
Path (A-B-E-G-H-K)



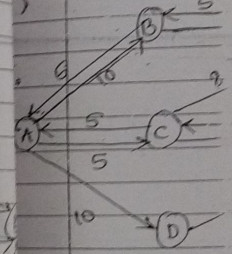
Residual Graph



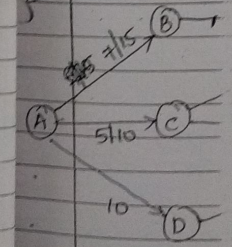
II Flow Netwo



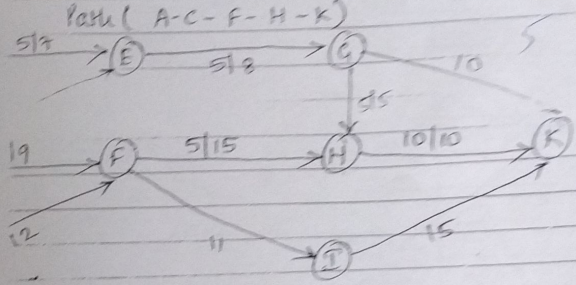
Residual

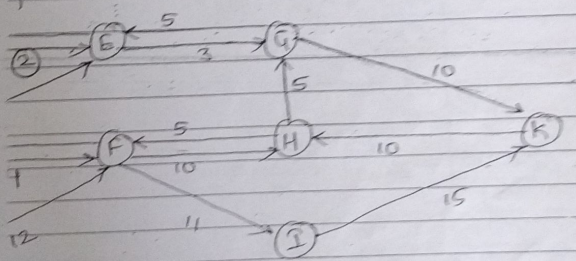


III Flow Netw

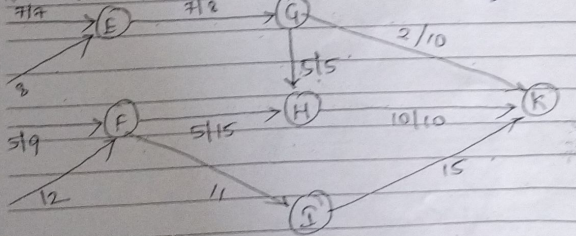


Path (A-C-F-H-K)

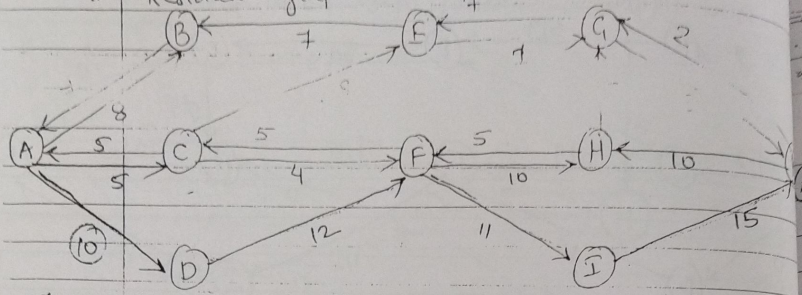




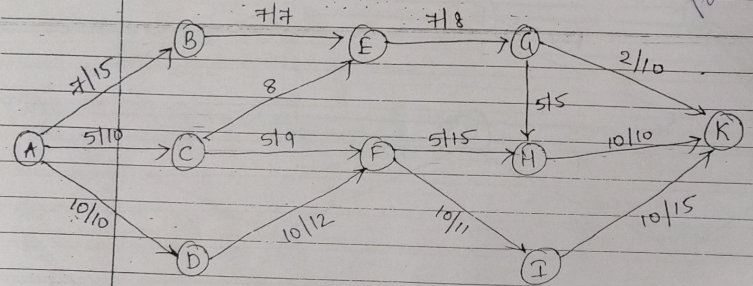
Path (A-B-E-G-K)



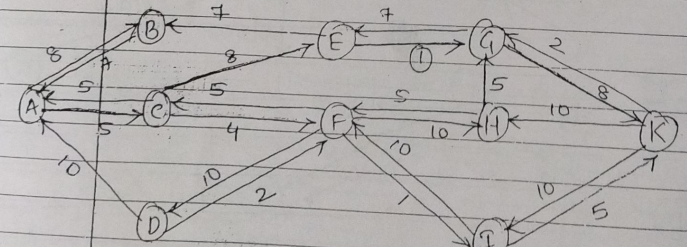
x Residual Graph



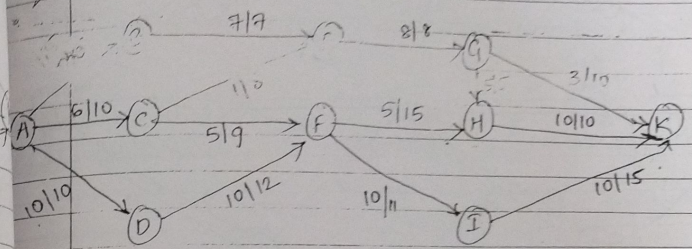
(iv) Flow Network (A-D-F-I-K)



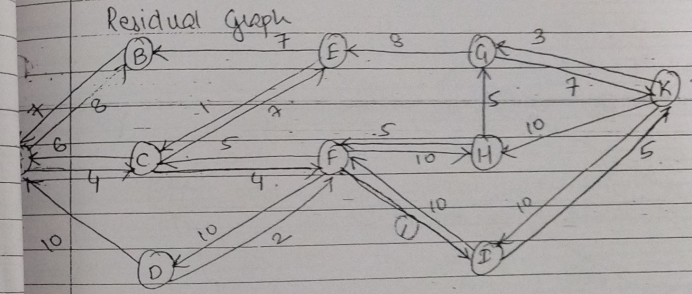
Residual Graph



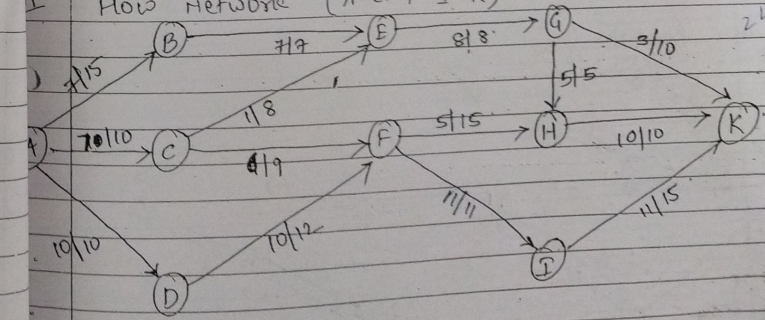
(v) Flow Network (A-C-E-G-K)



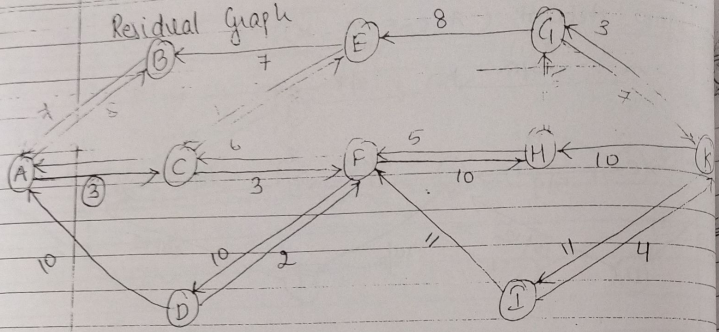
Residual Graph



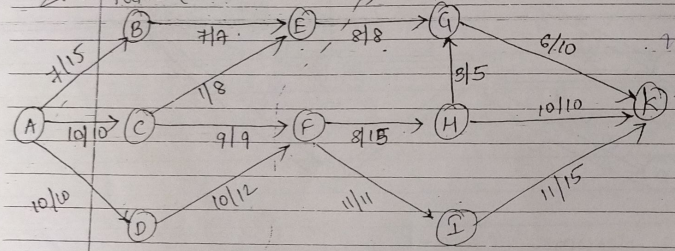
Flow Network (A-C-F-I-K)



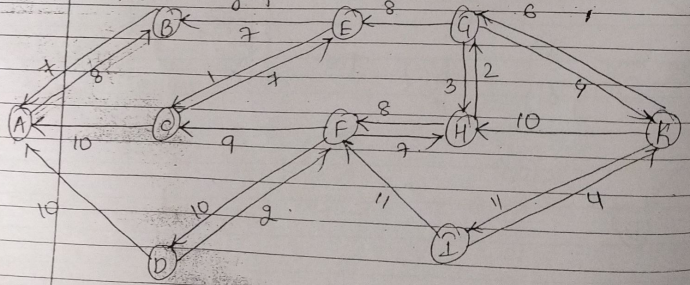
Residual Graph



viii. Path (A-C-F-H-G-K)



Residual Graph



Max flow = 10+10+7 = 27

8+9+10 = 27 ✓

Ans 02.

* VARIABLE LENGTH HUFFMAN CODE

f = 00 c = 05 g = 06 d = 08

Step 1:- Arranging the frequencies in ascending order.

f = 00 ✓
c = 05 ✓
g = 06
d = 08
a = 20
e = 25

Total = 64.

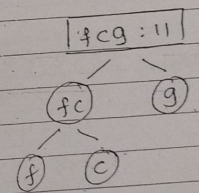
fc = 05 ✓
g = 06 ✓
d = 08
a = 20
e = 25

fc : 0

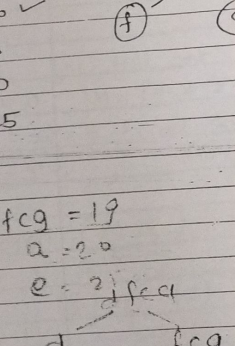
Step 2:-

d = 08 ✓
fcg = 11 ✓
a = 20
e = 25

fcg : 11



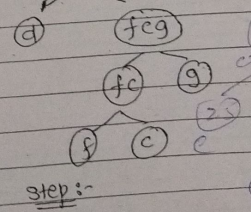
dfcg = 19
a = 20
e = 25



Step 3:-

dfcg : 19 ✓
a : 20 ✓
e : 25

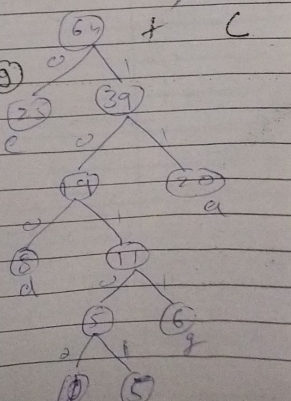
dfcg : 19



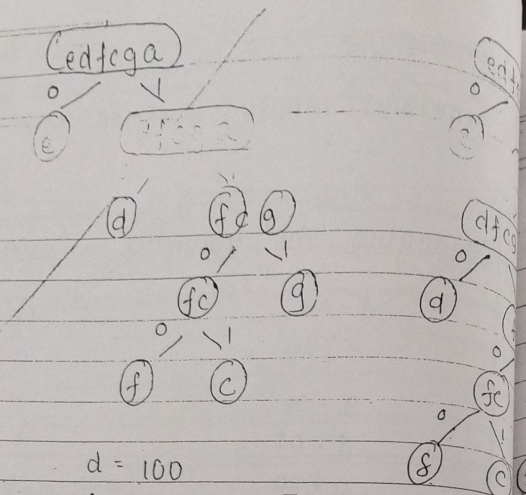
Step 4:-

e : 25 ✓
dfcga : 39 ✓

edfcga :- 64



e 0
a 11
d 100
g 101



a = 11 d = 100
e = 0 f = 10100
c = 10101 g = 1011

Total No. of bit = 20×2 + 25×1 + 5×5 + 8×3 + 5×5
= 40 + 25 + 25 + 24 + 0 + 24
= 90 + 48
= 138.

Probability = Total No. of bits / Total frequency

= 138 / 64

= 2.15

FIXED LENGTH CODE.

a = 000
b = 010
c = 011
d = 001
f = 100
g = 101

Total No. of bit
= 20×3 + 25×3 + 5×3 + 8×3 + 6×3
= 192

Probability = 192/64 = 3.

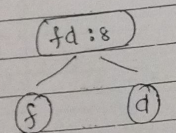
(2). Generate variable length Huffman code for following set of frequencies.

a = 20, b = 10, c = 15, d = 5, e = 22, f = 3. Find: 3.

Step1:- Arrange in ascending order

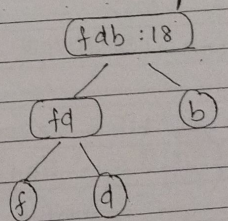
f = 3 ✓
d = 5 ✓
b = 10
c = 15
a = 20
e = 22

fd = 8 ✓
b = 10 ✓
c = 15
a = 20
e = 22



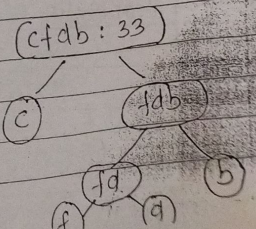
Step2:-

c = 15 ✓
fdb = 18 ✓
a = 20
e = 22



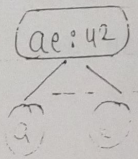
Step3:-

a = 20 ✓
e = 22 ✓
cfdb = 33

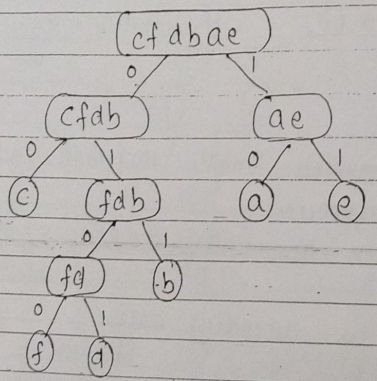


cfdb = 33

Step 4:- ae = 42



Step 5:- cfdbae = 75



a = 10 (20)
b = 011 (11)
c = 00 (15)
d = 0101 (5)
e = 11 (22)
f = 0100 (3)

Total No of bits = 2×20 + 3×10 + 2×15 + 4×5 + 2×22
= 40 + 30 + 30 + 20 + 44 + 12
= 176

Probability = Total No. of bits / Total frequency
= 176 / 75
= 2.34

* Use master method to solve the recurrence
$T(n) = 9T(\frac{n}{3}) + n^3$

Soln: Comparing the given recurrence with eqn recurrence we get,

$a = 9$
$b = 3$
$f(n) = n^3$

To find:- $n^{\log_b a}$

$\therefore \log_b a = \log_3 9$

$= \log_3 3^2$

$\therefore \log_b a = 2$

Thus, $f(n) > n^{\log_b a}$

$\therefore$ Case 3 applies.

CHECK FOR REGULARITY CONDITION.

$af(\frac{n}{b}) \leq cf(n)$

$\therefore = 9f(\frac{n^3}{3^3}) \quad 0 = 9(\frac{n^3}{3^3})$

$\therefore = 3f(n^3) \quad = (\frac{9}{27})n^3$

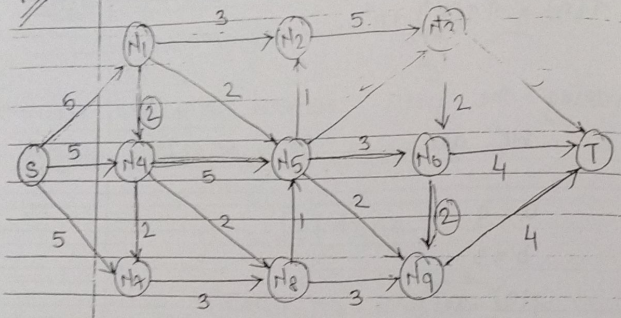
$\therefore 3(n^3) \leq cn^3$ for $c \geq 3$.

$\therefore (\frac{9}{27})n^3 \leq cn^3$ for $c \geq \frac{9}{27}$

Thus using case 3 we get $T(n) = \Theta(n^2)$

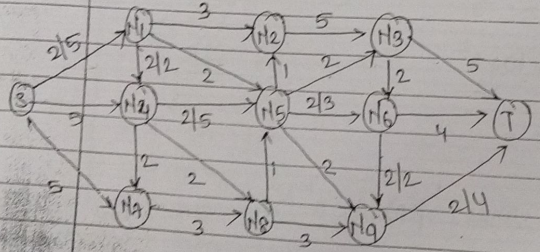
Sec 10.4

Evaluate the maximum flow from node 1

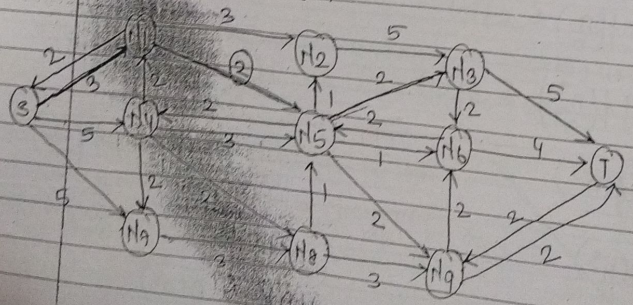


Step 1 Path :- S-n1-n4-n5-n6-n9-T

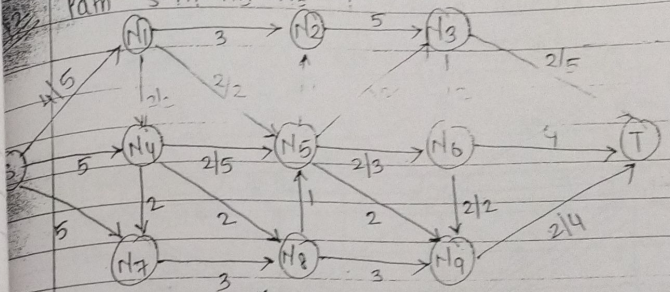
Flow Network



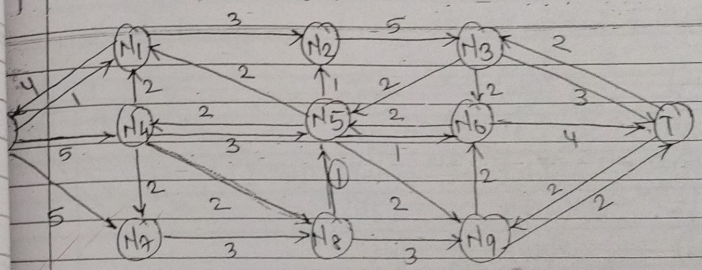
Residual graph



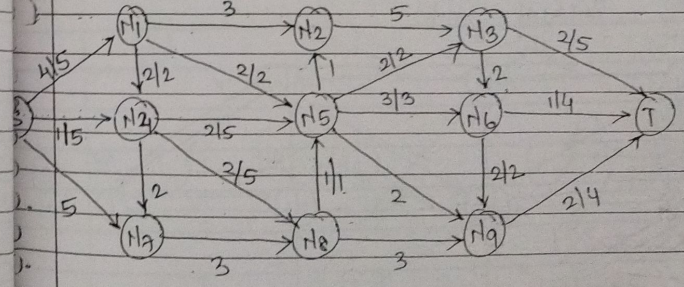
Path S-n1-n5-n2-T



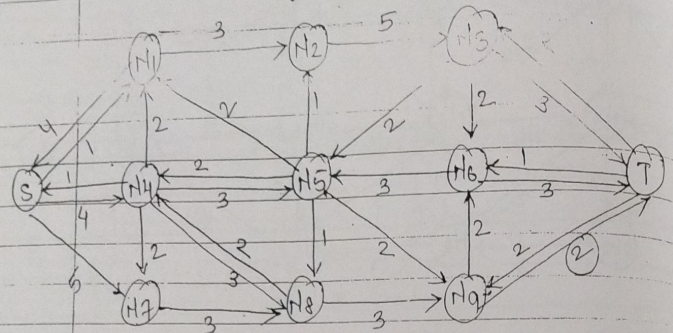
Residual graph



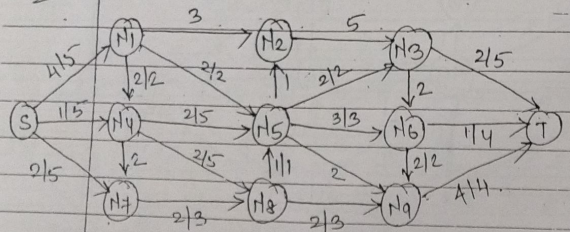
Path S-n4-n8-n5-n6-T



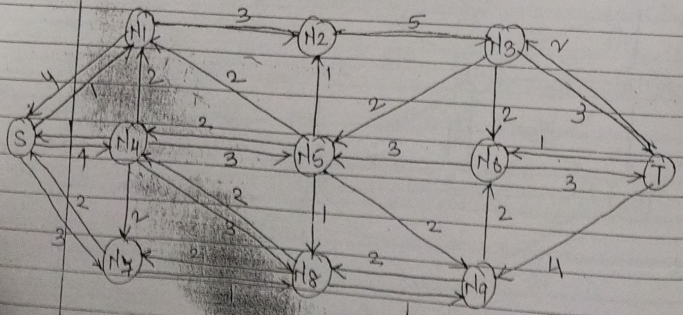
Residual graph



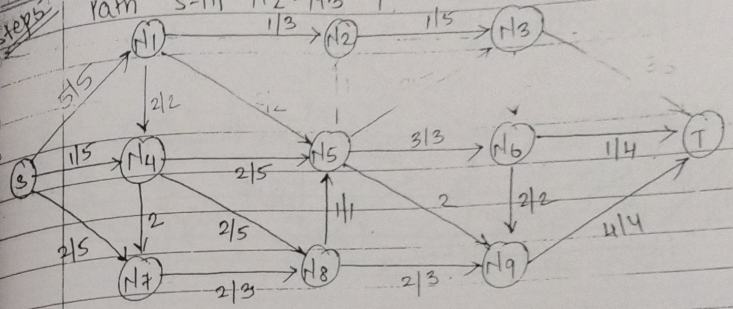
Step4 Path S-N7-N8-N9-T



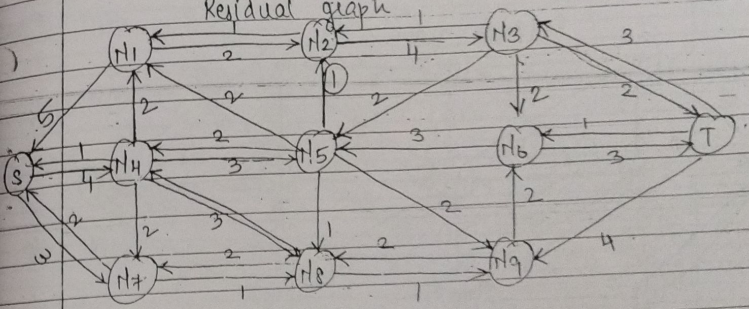
Residual graph



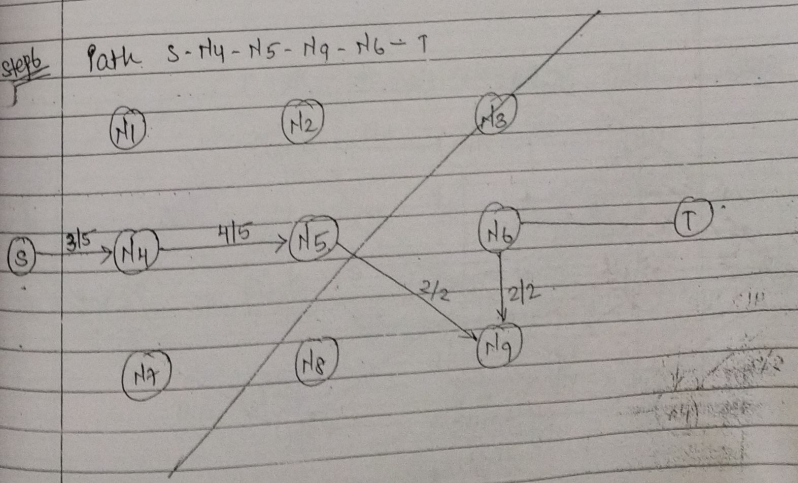
Steps Path S-N1-N2-N3-T



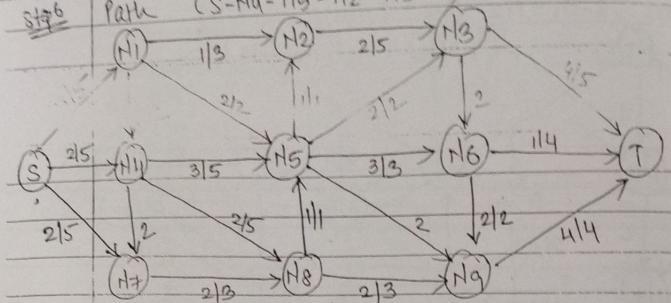
Residual graph



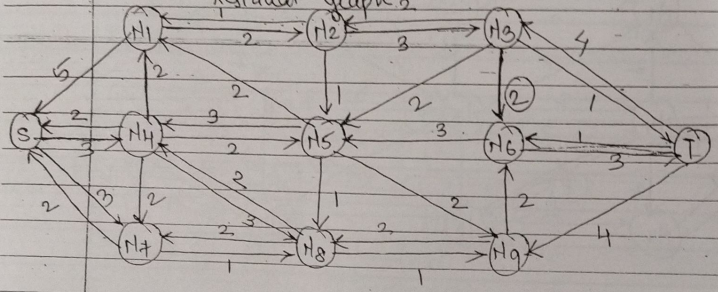
Steps Path S-N4-N5-N9-N6-T



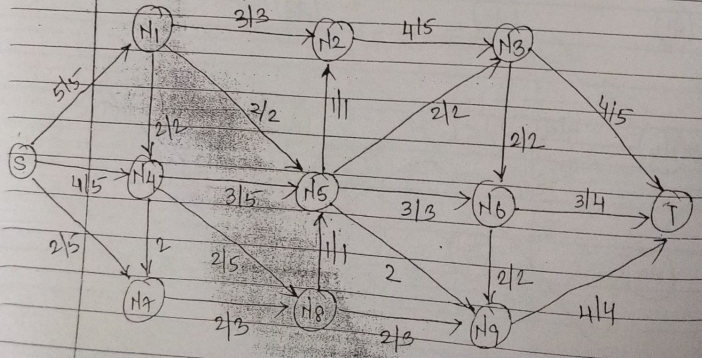
Step 6 Path CS-N4-N5-N2-N3-T



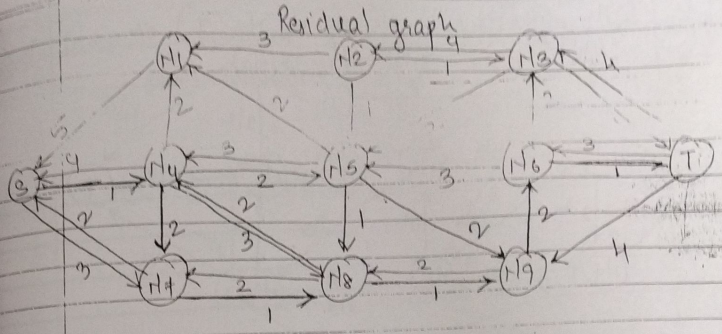
Residual graph 2



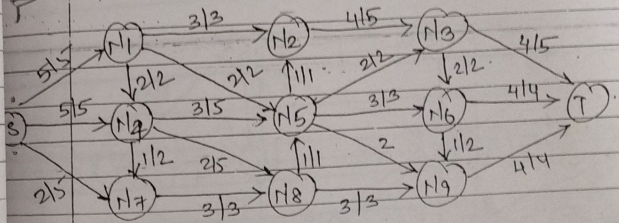
Step 7 Path S-N4-N1-N2-N3-~~N6~~-T



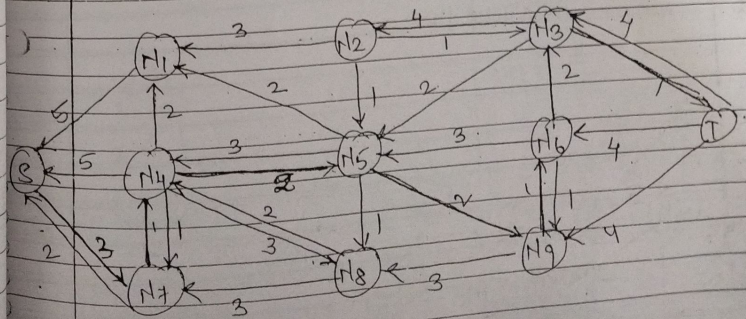
Residual graph



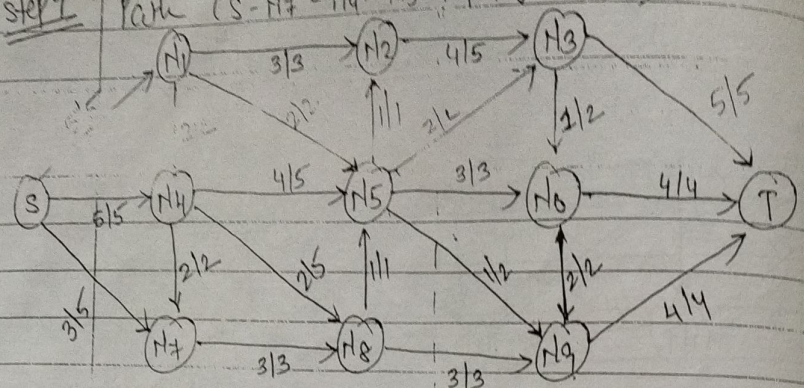
Step 8 Path S-N4-N7-N8-N9-N6-T



Residual graph

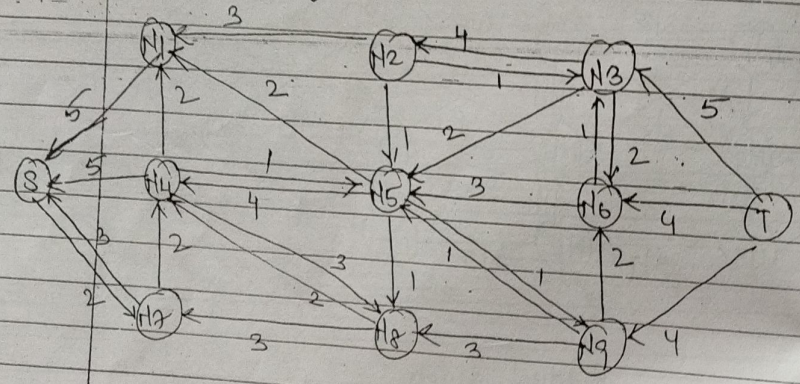


step 1 Path (S-N7-N4-N5-N9-N6-N3-T)



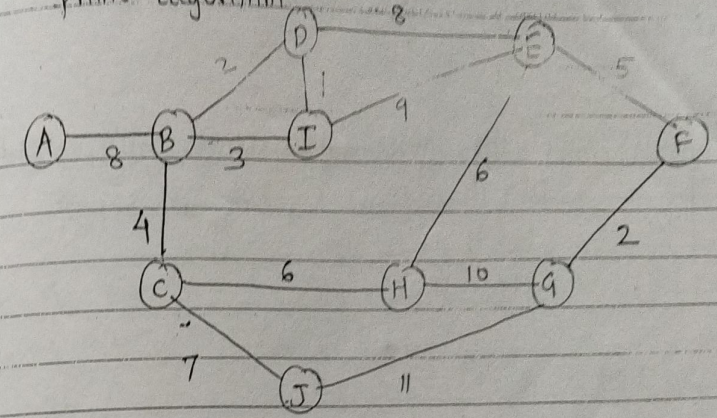
4+2+3+1+3 : 13

Residual graph

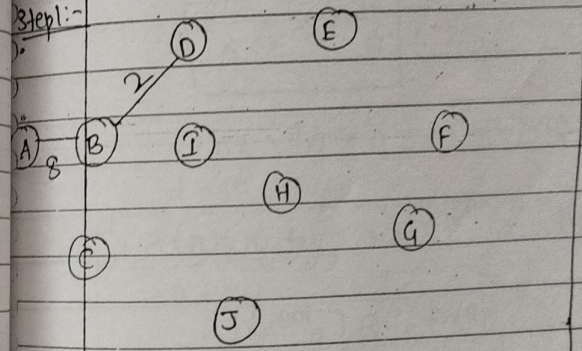


Maximum flow =

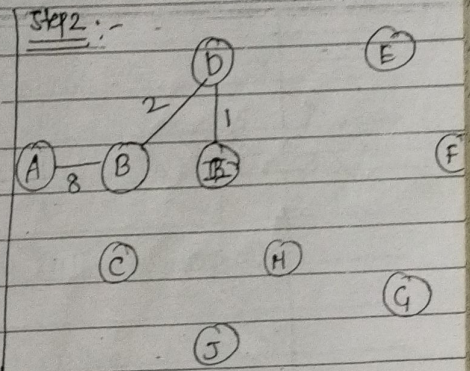
Or Compute minimum spanning tree for foll graph using prim's algorithm



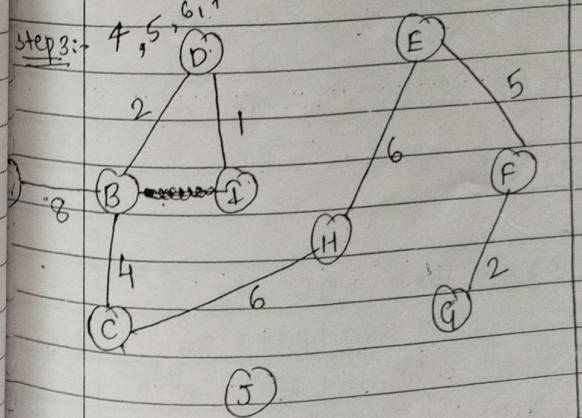
Step 1:-



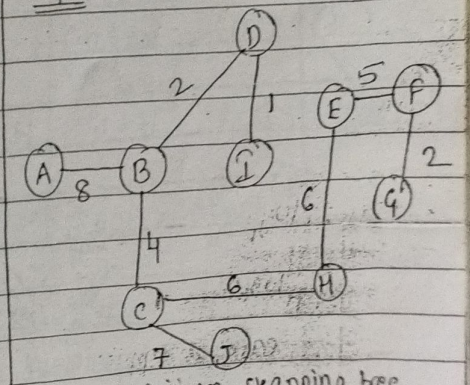
Step 2:-



Step 3:- 4, 5, 6, 7



Step 8:-



Cost of minimum spanning tree = 41

Dec 07

* Solve the recurrence using Master Method

a) $T(n) = 16T(n/4) + n$

Soln:- Comparing the given recurrence with std recurrence

$a = 16$

$b = 4$

$f(n) = n$

To find:- $n^{\log_b a}$

$\therefore \log_b a = \log_4 16$

$= \log_4 4^2$

$\therefore \boxed{\log_b a = 2}$

$\therefore \boxed{n^{\log_b a} = n^2}$

$\therefore n < n^2$

Comparing $f(n)$ & $n^{\log_b a}$ we get; $f(n) < n^{\log_b a}$

$\therefore$ Case 1 applies $\therefore T(n) = \Theta(n^{\log_b a})$

Thus, $T(n) = \Theta(n^2)$

b) $T(n) = 2T(\frac{n}{4}) + \sqrt{n}$

Soln $a = 2$, $b = 4$, $f(n) = \sqrt{n}$

$\therefore \log_b a = \log_4 2 = \log 2 \div \log 4 = 0.5 = 1/2$

$\therefore n^{\log_b a} = n^{1/2} \therefore n^{1/2} = n^{1/2} \therefore f(n) = n^{\log_b a}$

$\therefore$ case 2 applies $\therefore T(n) = (\sqrt{n} \log n)$

Aug 08

Evaluate variable length Huffman code for:-

$a = 20$, $b = 06$, $c = 25$, $f = 04$, $g = 08$, $e = 0$, $d = 10$

Step 1:- Arrange in ascending order, select min two, add the frequencies and draw tree

$e = 0$ ✓

$f = 04$ ✓

$b = 06$

$g = 08$

$d = 10$

$a = 20$

$c = 25$

$ef = 04$ ✓

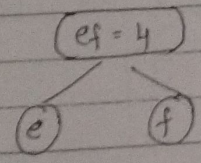
$b = 06$ ✓

$g = 08$

$d = 10$

$a = 20$

$c = 25$

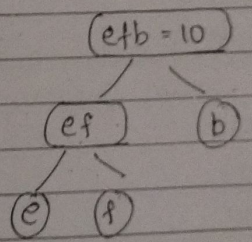


Step 2:- $g = 08$ ✓, $d = 10$ ✓

$efb = 10$

$a = 20$

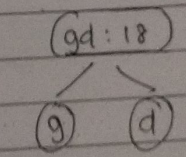
$c = 25$



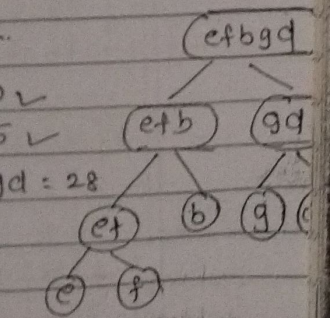
Step 3:- $efb = 10$ ✓, $gd = 18$ ✓

$a = 20$

$c = 25$

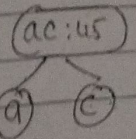


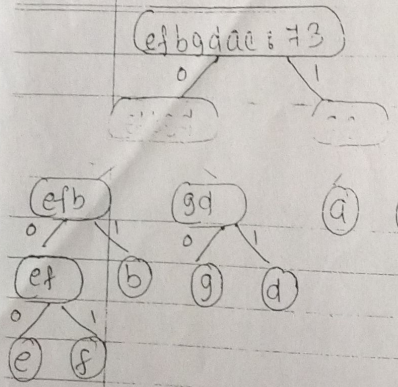
Step 4:- $a = 20$ ✓, $c = 25$ ✓, $cfbgd = 28$



Step 5:- ~~ac~~ $efbgd = 28$ ✓

$ac = 45$ ✓





e = 0000 (0)
f = 0001 (1)
b = 001
g = 010
d = 011 (10)
a = 10 (20)
c = 11 (25)

Total Number of bits = 4×0 + 4×4 + 3×6 + 3×8 + 3×10 + 2×25
= 0 + 16 + 18 + 24 + 30 + 40 + 50
= 178

Probability = Total number of bits / Total frequency

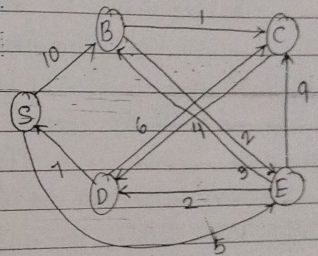
= 178 / 73

= 2.43

Ques 10 Given below is the weight matrix W for the graph consisting of 5 nodes {S, B, C, D, E}. Find the shortest path from node S to all other nodes

$$W = \begin{array}{c|ccccc} & S & B & C & D & E \\ S & 0 & 10 & \infty & \infty & 5 \\ B & \infty & 0 & 1 & \infty & 2 \\ C & \infty & \infty & 0 & 4 & \infty \\ D & 7 & \infty & 6 & 0 & \infty \\ E & \infty & 3 & 9 & 2 & 0 \end{array}$$

Sol. Resultant Graph =



1. d

S	B	C	D	E
0	10	∞	∞	5

Π

S	C	D	E
S	n	n	S

2. d

S	B	C	D	E
0	8	14	7	5

Π

B	C	D	E
E	E	E	S

3. d

S	B	C	D	E
0	8	13	7	5

Π

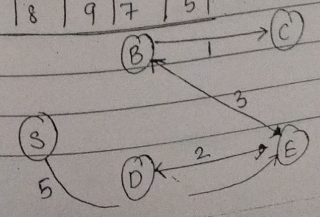
B	C	D	E
E	D	E	S

4 d

S	B	C	D	E
0	8	9	7	5

Π

B	C	D	E
E	B	E	S



ALGORITHM

* JOHSON(G)

1. compute G' where $V[G'] = V[G] \cup \{s\}$
   $E(G') = E[G] \cup \{(s,v) : v \in V[G]\}$
   $w(s,v) = 0$ for all $v \in V[G]$.
2. if BELLMAN-FORD (G', w, s) = FALSE
3. then print " The input graph contains a -ve
   else
4. for each vertex $v \in V[G']$
5. do set $h(v)$ to the value of $\delta(s,v)$
   computed by Bellman-Ford algo
6. for each edge $(u,v) \in E[G']$.
7. do $\hat{w}(u,v) \leftarrow w(u,v) + h(u) - h(v)$
8. for each vertex $u \in V[G]$
9. do run DIJKSTRA $(G, \hat{w}, u)$ to compute
   $\hat{\delta}(u,v)$ for all $v \in V[G]$.
10. for each vertex $v \in V[G]$.
11. do $d_{uv} \leftarrow \hat{\delta}(u,v) + h(v) - h(u)$.
12. return D.

ANALYSIS :-

If the min-priority queue in Dijkstra's algorithm is implemented by Fibonacci heap, the running time of Johnson's algorithm is $O(V^2 \log V + VE)$.

The simpler binary min-heap implementation yields a running time of $O(VE \log V)$.

* FORD - FULKERSON (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. for each edge (u,v) in p
7. do $f[u,v] = f[u,v] + c_f(p)$
8. $f[v,u] = -f[u,v]$

$|f^*|$ since the flow value increases by atleast one unit in each iteration

ANALYSIS :-

The implementation of FORD-FULKERSON runs in time $O(E|f^*|)$ where f^* is the maximum flow found by the algorithm.

Lines 1-3 takes time $\Theta(E)$

Lines 4-8 is executed atmost $|f^*|$ times, since the flow value increases by atleast one unit in each iteration.

29/8/12

DYNAMIC AND GREEDY ALGORITHM

* DYNAMIC PROGRAMMING.

→ Dynamic programming like the divide and conq method, solves problems by combining the solutio to the subproblems.

→ The difference between the two is that IN DYNAM PROGRAMMING the result obtained from solving smaller sub-problems are reused in the calculatio of larger sub-problems.

→ Dynamic programming is a BOTTOM-UP technique that usually begins by solving the smaller sub saving these results and then reusing them to larger sub-problems until the solution to original problem is obtained.

→ Dynamic programming is typically applied to optimization problems. In such problems there can many possible solutions. Each solution has a valu and we wish to find a solution with the optimal value. Such a solution is called an OPTIMAL SOLUTION.

→ Dynamic programming works on the principle of OPTIMALITY. The principle states that :- In an optimal sequence of decision or choices each sub-sequence must also be optimal.

→ The development of dynamic programming can be broken into a sequence of 4 steps :-

a) Characterize the structure of an optimal solution.
b) Recursively define the value of an optimal solution.
c) Compute the value of an optimal solution in a bottom-up fashion.
d) Construct an optimal solution from computed inform

→ Following are the two key characteristics that a problem must have for dynamic programming :-

a) OPTIMAL SUBSTRUCTURES

The first step in solving an optimization proble by dynamic programming is to characterize the structure of an optimal solution.

→ A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to subproblems.

- It is used in bottom-up fashion.

b) OVERLAPPING SUBPROBLEMS :-

- The second ingredient that an optimization problem must have for dynamic programming is that the space of the subproblems must be "small".
- When an recursive algo. revisits the same problems over and over again, it is said that optimization problem has overlapping subproblems.

* GREEDY APPROACH

- A greedy algorithm obtains an optimal solution to a problem by making sequence of choices.
- For each decision point in the algorithm, the choice that seems best at that moment is chosen.
- This heuristic strategy does not always produce an optimal solution but sometimes it does.
- The two key ingredients of greedy Approach
  1) GREEDY - CHOICE PROPERTY
  2) OPTIMAL - SUBSTRUCTURE.

a) GREEDY CHOICE PROPERTY:

- A globally optimal solution can be arrived making a locally optimal (greedy) choice.

we make the choice that looks best in the current problem without considering results from subproblems.

b) OPTIMAL SUBSTRUCTURE.

A problem exhibits optimal substructure if an solution to the problem contains within it optim solutions to subproblems.

GREEDY	DYNAMIC
1. Solves the sub problems using top-down technique	Solve the sub-problems using bottom-up approach.
2. Best choice is made at the moment & then solve the subproblems arising after the choice is made.	Choice is made at e step which usually depends on the solu to subproblems.
3. Key ingredient :- Greedy choice property + optimal substructure	Key ingredient :- optimal substructure, overlapping subproblems.
No guarantee of getting optimal solution	guaranteed optimal solution us the principle of optimality is used in bottom up fashion
4. Optimal substructure is used in top-down fashion	
5 Many decision sequences are made	only one decision sequenc is made

```
*  HUFFMAN (C)
1.  n ← |C|
2.  Q ← C
3.
4.  do allocate a new node z
5.     left[z] ← x ← EXTRACT-MIN(Q)
6.     right[z] ← y ← EXTRACT-MIN(Q)
7.     f(z) ← f(x) + f(y)
8.     INSERT (Q, z)
9.  return EXTRACT-MIN (Q)
```

ANALYSIS :-

- Q is implemented as a binary min-heap.

For set C of n characters, initialization of Q in lin is performed in O(n) time.

for loop in lines 3-8 is executed n-1 times + sinc each heap operation requires time O(logn), the loop contributes O(nlogn).

∴ Total running time = O(nlogn)

```
*  MST-KRUSKAL (G, w)                          - O(1)
1.  A ← ϕ
2.  for each vertex v ∈ V[G]
3.     do MAKE-SET (v)                          - O(E lo
4.  sort the edges of E into non-increasing order by wt
5.  for each edge (u,v) ∈ E, taken in "  "  "  "
6.     do if FIND-SET(u) ≠ FIND-SET(v)          } O((V+E)
7.        then A ← A ∪ {(u,v)}
8.             UNION (u,v)
9.  return A
```

Running time = O(ElogV)

```
* MST-Prim (G, w, s)
1  for each u ∈ V[G]
2      do key[u] ← ∞
3         π[u] ← NIL
4  key[r] ← 0
5  Q ← V[G]
6  while Q ≠ ∅                          } = O(V)
7      do u ← EXTRACT-MIN(Q)            }
8         for each v ∈ Adj[u]
9             do if v ∈ Q and w(u,v) < key[v]   }
10                then π[v] ← u                 }
11                     key[v] ← w(u,v)          }
```

Thus the total time for Prim's algorithm
$= O(V \log V + E \log V)$

$= O(E \log V)$.

If Fibonacci heap is used to implement the min-priority queue Q then the running time improves to $O(E + V \log V)$.

$m[i,j] = \min\left(m_{ik} + m_{k+1,j} + p_{i-1} p_i p_j\right)$

MATRIX CHAIN MULTIPLICATION

Q. $d[6] = [5, 10, 3, 12, 5, 50, 6]$

$p_0 = 5$, $p_1 = 10$, $p_2 = 3$, $p_3 = 12$, $p_4 = 5$, $p_5 = 50$, $p_6 = 6$.

$\min\{m(i,k) + m(k+1, j) + p_{i-1} p_k p_j\}; i \le k < j$

$m[1,2] = m[1,1] + m[2,2] + p_0 p_1 p_2$
$= 0 + 0 + 5 \times 10 \times 3$
$= 150$

$m[2,3] = m[2,2] + m[3,3] + p_1 p_2 p_3$
$= 0 + 0 + 10 \times 3 \times 12$
$= 360$

$m[3,4] = m[3,3] + m[4,4] + p_2 p_3 p_4$
$= 0 + 0 + 3 \times 12 \times 5$
$= 180$

$m[4,5] = m[4,4] + m[5,5] + p_3 p_4 p_5$
$= 0 + 0 + 12 \times 5 \times 50$
$= 3000$

$m[5,6] = m[5,5] + m[6,6] + p_4 p_5 p_6$
$= 0 + 0 + 5 \times 50 \times 6$
$= 1500$

$m[6,X] = m[6,6] + m[X,X] + p_5 p_6 p_X$
$= 0 + 0 +$
$=$

II) $m[1,3] = \min \begin{cases} m[1,1] + m[2,3] + p_0 p_1 p_3 \\ = 0 + 360 + 5 \times 10 \times 12 = 960 \\ m[1,2] + m[3,3] + p_0 p_2 p_3 \\ = 150 + 0 + 5 \times 2 \times 12 \end{cases}$

$m[2,4] = \min \begin{cases} m[2,2] + m[3,4] + p_1 p_2 p_4 \\ = 0 + 180 + 10 \times 3 \times 5 = 330 \\ m[2,3] + m[4,4] + p_1 p_3 p_4 \\ = 360 + 0 + 10 \times 12 \times 5 = 960 \end{cases}$

$m[3,5] = \min \begin{cases} m[3,3] + m[4,5] + p_2 p_3 p_5 \\ = 0 + 3000 + 3 \times 12 \times 50 = 4800 \\ m[3,4] + m[5,5] + p_2 p_4 p_5 \\ = 180 + 0 + 3 \times 5 \times 50 = 930 \end{cases}$

$m[4,6] = \min \begin{cases} m[4,4] + m[5,6] + p_3 p_4 p_6 \\ = 0 + 1500 + 12 \times 5 \times 6 = 1860 \\ m[4,5] + m[6,6] + p_3 p_5 p_6 \\ = 3000 + 0 + 12 \times 50 \times 6 = 6600 \end{cases}$

III) $m[1,4] = \min \begin{cases} m[1,1] + m[2,4] + p_0 p_1 p_4 \\ = 0 + 330 + 5 \times 10 \times 5 = 580 \\ m[1,2] + m[3,4] + p_0 p_2 p_4 \\ = 150 + 180 + 5 \times 3 \times 5 = 405 \\ m[1,3] + m[4,4] + p_0 p_3 p_4 \\ = 330 + 0 + 5 \times 12 \times 5 = 630 \end{cases}$

$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 \\ = 0 + 930 + 10 \times 3 \times 50 = 2430 \\ m[2,3] + m[4,5] + p_1 p_3 p_5 \\ = 360 + 3000 + 10 \times 12 \times 50 = 9360 \\ m[2,4] + m[5,5] + p_1 p_4 p_5 \\ = 330 + 0 + 10 \times 5 \times 50 = 2830 \end{cases}$

$m[3,6] = \min \begin{cases} m[3,3] + m[4,6] + p_2 p_3 p_6 \\ = 0 + 1860 + 3 \times 12 \times 6 = 2076 \\ m[3,4] + m[5,6] + p_2 p_4 p_6 \\ = 180 + 1500 + 3 \times 5 \times 6 = 1770 \\ m[3,5] + m[6,6] + p_2 p_5 p_6 \\ = 930 + 0 + 3 \times 50 \times 6 = 1880 \end{cases}$

$m[1,5] = \min$

$m[1,1] + m[2,5] + P_0 P_1 P_5$
$= 0 + 2430 + 5 \times 10 \times 50 = 4930$

$m[1,2] + m[3,5] + P_0 P_2 P_5$
$= 150 + 930 + 5 \times 3 \times 50 = 1830$

$m[1,3] + m[4,5] + P_0 P_3 P_5$
$= 330 + 3000 + 5 \times 12 \times 50 = 6330$

$m[1,4] + m[5,5] + P_0 P_4 P_5$
$= 405 + 0 + 5 \times 5 \times 50 = \boxed{1655}$

$m[2,6] = \min$

$m[2,2] + m[3,6] + P_1 P_2 P_6$
$= 0 + 1770 + 10 \times 3 \times 6 = 1950$

$m[2,3] + m[4,6] + P_1 P_3 P_6$
$= 360 + 1860 + 10 \times 12 \times 6 = 2940$

$m[2,4] + m[5,6] + P_1 P_4 P_6$
$= 330 + 1500 + 10 \times 5 \times 6 = \boxed{2130}$

$m[2,5] + m[6,6] + P_1 P_5 P_6$
$= 2430 + 0 + 10 \times 50 \times 6 = 5430$

$m[1,6] = \min$

$m[1,1] + m[2,6] + P_0 P_1 P_6$
$= 0 + 2130 + 5 \times 10 \times 6 = 2430$

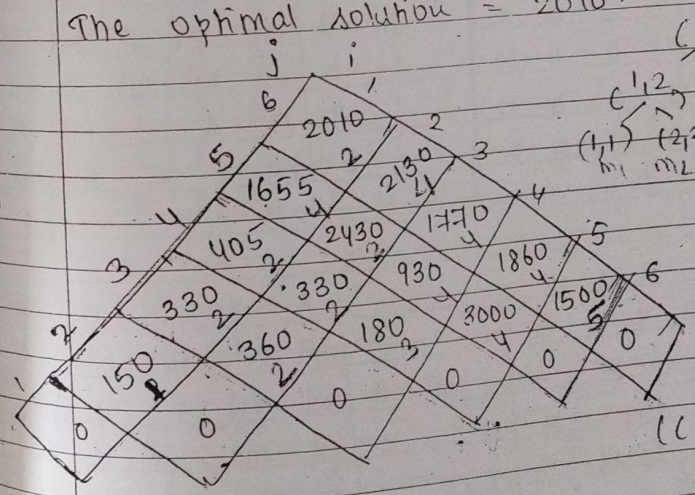
$m[1,2] + m[3,6] + P_0 P_2 P_6$
$= 150 + 1770 + 5 \times 3 \times 6 = \boxed{2010}$

$m[1,3] + m[4,6] + P_0 P_3 P_6$
$= 330 + 1860 + 5 \times 12 \times 6 = 2550$

$m[1,4] + m[5,6] + P_0 P_4 P_6$
$= 405 + 1500 + 5 \times 5 \times 6 = 2055$

$m[1,5] + m[6,6] + P_0 P_5 P_6$
$= 1655 + 0 + 5 \times 50 \times 6 = 3155$

The optimal solution = 2010.



Cost Matrix.

$((m_1 \times m_2)((m_3 \times m_4)(m_5 \times m_6))$

$((1,2)((3,4)(5,6)))$

(*)

$W =$

	A	B	C	D	E
A	0	10	∞	∞	5
B	∞	0	1	∞	2
C	∞	∞	0	-4	∞
D	7	∞	6	0	∞
E	∞	-3	9	2	0

Step 1:-

$D^0 =$

	A	B	C	D	E
A	0,	10	∞	∞	5
B	∞,	0	1	∞	2
C	∞,	∞	0	-4	∞
D	7,	∞	6	0	∞
E	∞,	-3	9	2	0

$\Pi^0 =$

	A	B	C	D	E
A	nil	A	nil	nil	A
B	nil	nil	B	nil	B
C	nil	nil	nil	C	nil
D	D	nil	D	nil	nil
E	nil	E	E	E	nil

Step 2:-

$D^1 =$

	A	B	C	D	E
A	0	10,	∞	∞	5
B	∞	0,	1	∞	2
C	∞	∞	0	-4	∞
D	7	17,	6	0	12
E	∞	-3,	9	2	0

$\Pi^1 =$

	A	B	C	D	E
A	nil	A	nil	nil	A
B	nil	nil	B	nil	B
C	nil	nil	nil	C	nil
D	D	A	D	nil	A
E	nil	E	E	E	nil

Step 3:-

$D^2 =$

	A	B	C	D	E
A	0	10	11,	∞	5
B	∞	0	1,	∞	2
C	∞	∞	0,	-4	∞
D	7	17	6,	0	12
E	∞	-3	-2,	2	0

-1

$\Pi^2 =$

	A	B	C	D	E
A	nil	A	B	nil	A
B	nil	nil	B	nil	B
C	nil	nil	nil	C	nil
D	D	A	D	nil	A
E	nil	E	B	E	nil

Step 4:-

$D^3 =$

	A	B	C	D	E
A	0	10	11	7	5
B	∞	0	1	-3	2
C	∞	∞	0	-4	∞
D	7	17	6	0	12
E	∞	-3	-2	-6	0

-1

$\Pi^3 =$

	A	B	C	D	E
A	nil	A	B	C	A
B	nil	nil	B	C	B
C					
D					
E	nil	E	B	E	nil

Step 5:-

$D^4 =$

	A	B	C	D	E
A	0	10	11	7	5,
B	4	0	1	-3	2,
C	3	13	0	-4	8,
D	7	17	6	0	12
E	1	-3	0,	-6	0

-1

$\Pi^4 =$

	A	B	C	D	E
A	nil	A	B	C	A
B	D	nil	B	C	B
C	D	A	nil	~~C~~	A
D	D	A	D	nil	A
E	D	E	D	E	nil

Step 6:-

$D^5 =$

	A	B	C	D	E
A	0	2	5	-1	5
B	3	0	1	-4	-1
C	3	5	0	-4	8
D	7	9	6	0	11
E	0	-2	-1	-7	0

$\Pi^5 =$

	A	B	C	D	E
A	nil	E	D	E	A
B	D	nil	B	C	
C			nil		
D				nil	
E	D				nil

PATH

(A,D)	D	(A,B)	B	(A,C)	C
↓	↑	↓	↑	↓	↑
(A,E)	E	(A,E)	E	(A,D)	D
↓	↑	↓	↑	↓	↑
(A,A)	A	(A,A)	A	(A,E)	E
				↓	↑
				(A,A)	A

10/12

String Matching

→ Finding all occurences of a pattern in a text is a problem that arises frequently in text-editing.

→ Typically, the text is a document being edited and the pattern searched for, is a particular word supplied by the user.

→ Efficient algorithms for this problem can greatly aid the responsiveness of the text-editing program.

→ These algorithms are also used to search for a particular patterns in DNA sequences.

* STRING MATCHING PROBLEM.

It can be formulized as follows:-

1. We assume that a) the text is an array $T[1 \ldots n]$ of length n.
   b) The pattern is an array $P[1 \ldots m]$ of length $m \leq n$.
2. The elements of P and T are characters drawn from a finite alphabet Σ.
   for eg $\Sigma = \{0, 1\}$
   $\Sigma = \{a, b, c, \ldots, z\}$.
3. Pattern P occurs with shift s in text T if $0 \leq s \leq n-m$ and $T[s+1 \ldots s+m] = P[1 \ldots m]$. If P occurs with shift s in T, then we call s a "valid shift" otherwise we call s as an "invalid shift".
4. The string matching problem is to find all valid shifts with which a given pattern P occurs in a given text T.

The following figure illustrates the definition.

text T

1	2	3	4	5	6	7	8	9	10	11	12	13
a	b	c	a	b	a	a	b	c	a	b	a	c

pattern P →

a	b	a	a
1	2	3	4

$s = 3$

Here the shift $s=3$ is a valid shift.

5. Every string matching algorithm except the (BRUTE FORCE) NAIVE STRING MATCHING performs some preprocessing on the pattern & then finds all valid shifts. The latter phase is known as "MATCHING PHASE".

TOTAL RUNNING TIME = PREPROCESSING + MATCHING TIME.

I THE NAIVE STRING MATCHING ALGORITHM. (BRUTE-FORCE)

1. In this algorithm, pattern P is compared character by character in the given text T.
2. The naive algorithm finds all valid shifts using a loop that checks the condition
   $P[1 \ldots m] = T[s+1 \ldots s+m]$
   for each of the $n-m+1$ (13-4+1) possible values of s.
   * In the above eq the possible values of s are 1 to 10
3. It can be interpreted graphically as SLIDING a "TEMPLATE" containing the pattern over the text, noting, for which shifts all of the characters on the template equal the corr. characters in the text.

Example

1	2	3	4	5	6	7	8	9	10	11
a	b	a	b	a	b	a	a	b		

P = | a | b | a | a |
(1 2 3 4)

∴ n-m+1 = 11-4+1 = 8

Iteration 1:-

T = | a | b | b | b | a | b | a | b | a | a | b |

P ↓ | a | b | a | a |

→ No match.

→ Match found, shift s & P by 1

Since no match is found, shift pattern by one position to right of T.

⋮

Continue till search is successful.

& Match is found at shift = 6.

ALGORITHM NAIVE-STRING-MATCHER (T, P)

1. n ← length [T]
2. m ← length [P]
3. for s ← 0 to n-m (Considers each shift explicitly)
4. do if P[1..m] = T[s+1..s+m] (check whether shift is valid or not)
5. then print "Pattern occurs with shift" s.
   ↳ Prints each valid shift s

ANALYSIS:-

→ The worst-case running time is $\Theta((n-m+1)m)$ which is $\Theta(n^2)$ if $m = \lfloor n/2 \rfloor$.

→ The running time of N.S.M algo is equal to its matching time since there is no preprocessing.

→ It is inefficient because info. gained about the text for one value of s is entirely ignored in considering other values of s. Such info. is however valuable.

as other original work

RABIN-KARP ALGORITHM

- The Rabin-Karp algorithm is a string searching algorithm created by Michael O. Rabin and Richard M. Karp that seeks a pattern (i.e. a substring) within a text by using hashing.
- It is not widely used for single pattern matching, but is of considerable importance and is very effective for multiple pattern matching.

PRACTICAL APPLICATION OF RABIN-KARP is PLAGIARISM*

WORKING:-

1. Given a pattern P[1..m] let p denote its corresponding decimal value.
2. Similarly given a text T[1..n] let t_s denote the decimal value of length m substring.
3. Certainly $t_s = p$ if and only if T[s+1..s+m] = P[1..m] thus s is a valid shift if and only if $t_s = p$.
4. Using HORNER'S RULE we can compute p in time $\Theta(m)$

$$p = P[m] + 10\,[P[m-1]) + 10^2 P[m-2]$$

t_s can be computed as follows:-

$$t_{s+1} = 10\,(t_s - 10^{m-1} T[s+1]) + T[s+m+1].$$

Thus p and $t_0, t_1, \ldots, t_{n-m}$ can all be computed in $O(n+m)$ time. All occurences of the pattern P[1..m] in the text T[1..n] can be found in $O(n+m)$ time.

5. Since p and ts may be too large to work modulo q is used where q is any prime

Recurrence eqn is

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

where $h = d^{m-1} \pmod q$.

Example

Text T = 2359023141526739921

Pattern P = 31415

$q = 13$, $m = 5$.

1. To find:- $P \bmod q = 31415 \bmod 13 = 7$

$t_0 = 23590 = t_s$.

$\therefore 23590 \bmod 13 = 8$.

Not valid shift since $7 \neq 8$.

2. $t_{s+1} = ?$

$s = 0$, $m = 5$, $t_s = 23590$.

$\therefore t_{s+1} = 10(t_s - 10^{m-1} T[s+1]) + T[s+m+1]$

$= 10(23590 - 10^4 T[1]) + T[0+5+1]$

$= 10(23590 - 2 \times 10^4) + T[6]$

$= 10(23590 - 20000) + 2$

$= 35900 + 2$

$t_{s+1} = 35902$ $\quad \therefore t_1 = 0.35902 = t_s$

Thus; 35902 mod 13 = 9.

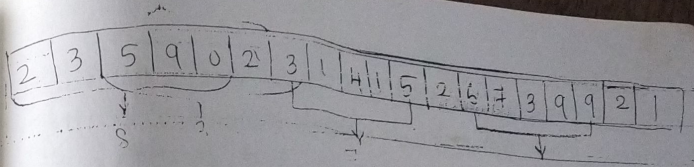
3. $t_{s+1} = ?$ $\quad s = 1$, $m = 5$, $t_s = 35902$

$t_{s+1} = 10(t_s - h T[s+1]) + T[s+m+1]$

$= 10(35902 - 30000) + 3$

$= 59020 + 3$ $\quad t_2 = 59023 = t_s$

$= 59023$ $\quad 59023 \bmod 13 = 3$



ALGORITHM :-

RABIN - KARP - MATCHER (T, P, d, q)

```
1  n ← length[T]
2  m ← length[P]
3  h ← d^(m-1) mod q
4  p ← 0
5  to ← 0
6  for i = 1 to m
7     do p ← (dp + P[i]) mod q
8        to ← (dto + T[i]) mod q
9  for s ← 0 to n-m
10    do if p = ts
11          then if P[1..m] = T[s+1..s+m]
12                  then print "Pattern occurs with shift" s
13               if s < n-m
14                  then ts+1 ← (d(ts - T[s+1]h) + T[s+m+1]) mod q
```

ANALYSIS :-

Preprocessing time = $\theta(m)$

Matching time = $O((n-m+1)m)$ ⇒ Worst case

since Rabin-Karp algorithm explicitly verifies every valid shift.

In many applications we expect few valid shifts. In these applications the expected matching time of the algorithm is $O((n-m+1)+cm) = O(n+m)$ plus the time required to process spurious hits.

* Working modulo $q = 11$ how many spurious hits does the Rabin-Karp matcher encounter in the

Text T = 3 1 4 1 5 2 6 5 3 5 8 9 7 9 3 looking

Patter P = 26

Here $q = 11$

$m = 2$

$s = n - m + 1$

$s = 16 - 2 + 1$

$s = 15$

$p = p \bmod q$

$= 26 \bmod 11$

$\boxed{p = 4}$

$t_0 = 31 = t_s$

$\Rightarrow 31 \bmod 11 = 9$

t_{s+1} ; $s=0$, $m=2$, $q=11$, $t_s = 31$

$t_{s+1} = 10\,(t_s - 10\,T[s+1]) + T[s+m+1]$

$= 10\,(31 - 10 \times 3) + T[0+2+1]$

$= 10\,(31-30) + T[3]$

$= 10 + 4$

$t_{s+1} = 14$

$\therefore t_1 = 14 \bmod 13 = t_s$

$\Rightarrow 14 \bmod 11 = 3$

Similarly

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
9	3	8	4	4	4	10	10	9	2	3	1	9	2	5

→ valid match.

No. of spurious hits = 3

III LONGEST COMMON SUBSEQUENCE

→ In biological applications, we often want to compare the DNA of two different organisms. A strand of DNA consists of a string of molecules called BASES. where possible bases are adenine, guanine, cytosine, thymine. Thus a strand of DNA can be expressed as a string over the finite set {A, C, G, T}.

For eg:- DNA of one organism may be

S_1 = ACCGGTCGAGTGCGCGG....

DNA of another organism may be

S_2 = GTCGTTCGGAATGC....

A C T G

One goal of comparing two strands of DNA is TO DETERMINE HOW SIMILAR" THE 2 strands are

Possible ways To determine similarity.

→ 1) Two DNA strands are similar if one is a substring of the other

2) If the no. of changes needed To turn one into other is small.

3) Find a third strand S_3, for which the bases in S_3 appear in each S_1 and S_2 * THESE BASES MUST APPEAR IN THE SAME ORDER *

The longer the strand S_3, the more similar it is To S_1 and S_2.

LONGEST common SUBSEQUENCE uses the last concept of similarity.

Example :-

X = {A, B, C, B, D, A, B}

Y = {B, D, C, A, B, A}

i		j	0	1	2	3	4	5	6
			yj	B	D	C	A	B	A
0	xi		0	0	0	0	0	0	0
1	A		0	0↑	0↑	0↑	1↖	1←	1↖
2	B		0	(1)↖	1←	1←	1←	2↖	2←
3	C		0	1↑	1↑	(2)↖	2←	2↑	2↑
4	B		0	1↖	1↑	2↑	2↑	(3)↖	3←
5	D		0	1↑	2↖	2↑	2↑	3↑	3↑
6	A		0	1↑	2↑	2↑	3↖	3↑	(4)↖
7	B		0	1↖	2↑	2↑	3↑	4↖	(4)↑

BDAB
CBA

The length of the LCS is 4

LCS = BCBA

Dec 11

2) X = {A B B A B A B A} = {1,0,0,1,0,1,0,1}

Y = {B C A B B A A B A} = {0,1,0,1,1,0,1,1,0}

i		j	0	1	2	3	4	5	6	7	8	9
			yj	0	1	0	1	1	0	1	1	0
0	xi		0	0	0	0	0	0	0	0	0	0
1	1		0	0↑	(1)↖	1←	1↖	1↖	1←	1↖	1↖	1←
2	0		0	1↖	1↑	(2)↑	2←	2←	2↖	2←	2←	2↖
3	0		0	1↖	1↑	2↖	2↑	2↑	3↖	3←	3←	3↖
4	1		0	1↑	2↖	2↑	3↖	3↖	3↑	(4)↖	4↖	4←
5	0		0	1↖	2↑	3↖	3↑	3↑	4↖	4↑	4↑	5↖
6	1		0	1↑	2↖	3↑	4↖	4↖	4↑	5↖	(5)↖	5↑
7	0		0	1↖	2↑	3↖	4↑	4↑	5↖	5↑	5↑	(6)↖
8	1		0	1↑	2↖	3↑	4↖	5↖	5↑	6↖	6↖	6↑

0 1 0 1 0 1 (5) ✗
0 1 0 1 0 (6) ✓

1 0 1 0 1 1 (3)
1 0 1 1 0 1 (3)
0 1 0 1 0 1 (5)

Length = 6

LCS = 100110 (1)

ALGORITHM :-

```
*  LCS - LENGTH (X, Y)
1  m ← length [X]
2  n ← length [Y]
3  for i ← 0 to m
4      do c[i,0] ← 0
5  for j ← 0 to n
6      do c[0,j] ← 0
7  for i ← 1 to m
8      do for j ← 1 to n
9          do if xi = yj
10             then c[i,j] ← c[i-1, j-1] + 1
11                  b[i,j] ← "↖"
12         else if c[i-1,j] ≥ c[i,j-1]
13              then c[i,j] ← c[i-1,j]
14                   b[i,j] ← "↑"
15         else c[i,j] ← c[i,j-1]
16              b[i,j] ← "←"
17 return c and b
```

```
*  PRINT - LCS (b, X, i, j)
1  if i = 0 or j = 0
2      then return
3  if b[i,j] = "↖"
4      then PRINT - LCS (b, X, i-1, j-1)
5           print xi
6  else if [b[i,j] = "↑"
7      then PRINT - LCS (b, X, i-1, j)
8      else PRINT - LCS (b, X, i, j-1)
```

ANALYSIS :-

The running time of the procedure is O(m

MQ) TO

3) A = 0001101100

B = 1110010010

i		j	0	1	2	3	4	5	6	7	8
		yj	1	1	1	0	0	1	0	0	
0	xi		0	0	0	0	0	0	0	0	0
1	0		0	0↑	0↑	0↑	1↖	1↖	1←	1↖	1↑
2	0		0	0↑	0↑	0↑	1↖	(2)↖	2←	2↖	2↑
3	0		0	0↑	0↑	0↑	1↖	2↖	2↑	(3)↖	3↑
4	1		0	1↖	1↖	1↖	1↑	2↑	3↖	3↑	3↑
5	1		0	1↖	2↖	2↖	2←	2↑	3↖	3↑	3↑
6	0		0	1↑	2↑	2↑	3↖	3↖	3↑	4↖	(4)↖
7	1		0	1↖	2↖	3↖	3↑	3↑	4↖	4↑	4↑
8	1		0	1↖	2↖	3↖	3↑	3↑	4↖	4↑	4↑
9	0		0	1↑	2↑	3↑	4↖	4↖	4↑	5↖	5↖
10	0		0	1↑	2↑	3↑	4↖	5↖	5←	5↖	6↖

Length of LCS = 6

LCS = 000010

4) A = aaabbcada

B = acabbdbdbda

i		j	0	1	2	3	4	5	6	7	8	9	10	11
		yj	a	c	a	b	b	d	b	d	b	d	a	
0	xi		0	0	0	0	0	0	0	0	0	0	0	0
1	a		0	1↖	1←	1↖	1←	1←	1←	1←	1←	1←	1←	1↖
2	a		0	1↖	1↑	2↖	2←	2←	2←	2←	2←	2←	2←	2↖
3	a		0	1↖	1↑	2↖	2↑	2↑	2↑	2↑	2↑	2↑	2↑	3↖
4	b		0	1↑	1↑	1↑	3↖	3↑	3←	3↖	3←	3↖	3←	3↑
5	b		0	1↑	1↑	1↑	2↑	4↖	4←	4↖	4←	(4)↖	4←	4←
6	c		0	1↑	2↖	2←	2↑	4↑	4↑	4↑	4↑	4↑	4↑	4↑
7	a		0	1↖	2↑	3↖	3←	4↑	4↑	4↑	4↑	4↑	4↑	5↖
8	d		0	1↑	2↑	3↑	3↑	4↑	5↖	5←	5↖	5←	(5)↖	5↑
9	a		0	1↖	2↑	3↖	3↑	4↑	5↑	5↑	5↑	5↑	5↑	(6)↖

Length of LCS = 6

LCS = aabbda

5) X = ABCBDAB

Y = BDCABA

i		j	0	1	2	3	4	5	6
		yj	B	D	C	A	B	A	
0	xi		0	0	0	0	0	0	0
1	A		0	0↑	0↑	0↑	(1↖)	1←	(1↖)
2	B		0	(1↖)	1←	1←	1↑	(2↖)	2←
3	C		0	1↑	1↑	(2↖)	2←	2↑	2↑
4	B		0	(1↖)	1↑	2↑	2↑	(3↖)	3←
5	D		0	1↑	(2↖)	2↑	2↑	3↑	3↑
6	A		0	1↑	2↑	2↑	3↖	3↑	(4↖)
7	B		0	(1↖)	2↑	2↑	3↑	4↑	4↑

D BCBD ✓

2) BCAB

3) BDAB

BDAB

CBA

Length of LCS = 4

LCS = BCBA

BCAB

IV STRING MATCHING WITH FINITE AUTOMATA

Non-deterministic Algorithms

1. Non-deterministic Algorithms
Non-deterministic algorithm involves some kind of

2. the algorithms we use to solve problems in reality using computers belong to the deterministic category, every step of a deterministic algorithm is uniquely determined.

3. On the other hand in the NDA, athose corresponding to one or more steps of the algorithm is dependent on a number of alternatives

4. The algorithm is assumed to have the power to make correct choice. We assume that each time a choice is to be made in a NDA, several copies of the algorithm are brought into action

5. Each copy lives with a separate choice

6. The copy that succeeds first terminates all other copies

7. A non-successful copy does not affect other copies.

8. Thus, a NDA always makes a correct choice whenever it is asked to make selection out of a number of alternatives, provided a correct choice exists in the alternatives

* Non-deterministic Searching

```
Proredure NDS.
Array A(n)
j = select one element from the set {1,2, ..., n};
if A[j] = e then
    print (" found at index = ", j);
else
Endif, print (" Not present in the array.");
End NDS.
```

Analysis

This algorithm requires constant time, thus complexity is O(1)

* **Non-deterministic Satisfiability**

1. Problem is to determine if a Boolean expression in CNF is true for assignment of truth to the variables
2. A Boolean expression in CNF is the [illegible] of number of OR terms
3. It is easy to obtain a non-deterministic polynomial time algorithm that terminates successfully if the given Boolean expression $F(x_1, x_2, \ldots, x_n)$ is satisfiable

Procedure NDSAT

```
for i=1 to n do
    xi = select one element from the set {True, False}
End for
If f(x1, x2, x3, ..., xn) is true then
    print ("satisfied");
else
    print ("unsatisfied");
End if
End NDSAT
```

The complexity of this algorithm is $O(n)$

PROBLEM TAXONOMY

Problems may be broadly divided into [illegible] types

UNDECIDABLE

eg:- 1) Halting problem of Turing machine

2) Tiling a rectangular area with given tiles with certain conditions

3) Word correspondence problems

INTRACTABLE PROBLEMS:-

i) Finding Hamiltonian cycle in a given graph

ii) Finding clique of a given maximum size

iii) The set of partitioning problem etc.

TRACTABLE PROBLEMS:-

Good algorithms irrespective of the degree of polynomial.

* **P-Problems**

This is the class of problems for which we have deterministic polynomial time algorithms

* **NP-Problems**

This is the class of problems solvable by a NDA in polynomial time

P-Problem in terms of language

$P = \{ L \mid L = L(M)$ for some Turing machine M which runs in polynomial time $\}$

The language accepted by M, which is denoted by $L(M)$

has an associated alphabet Σ and is defined
$L(M) = \{\omega \in \Sigma^* \mid M \text{ accepts } \omega\}$.

NP :-

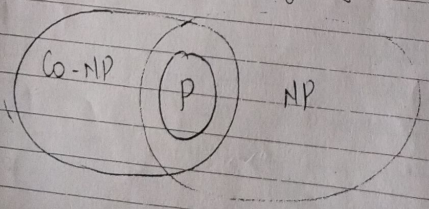
A language over Σ is in NP iff there is
and a polynomial time checking relation R such
for all $w \in \Sigma^*$, $w \in L \leftrightarrow$ there exists
$y \{|y| \leq |w|^k$ and $R(w,y)\}$
where $|y|$ = length of y
$|w|$ = length of w.

A checking relation is simply a binary relation
$R \subseteq \Sigma^* \times \Sigma_1^*$ for some finite alphabet Σ and

* Co-NP

A language L belongs to this class if L^c, the
complement of L belongs to NP. The complement
of language L is defined as

$L^c = \{\Sigma^* - x \mid x \text{ belongs } L\}$.



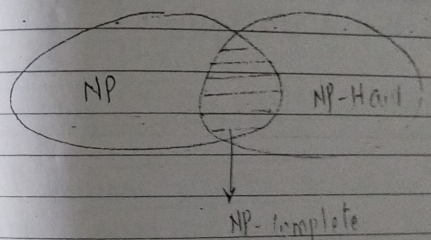
* NP-Hard :-

A problem L is NP-Hard if and only if satisfiability ~~reduces~~ reduces to L or any other NP-hard ~~problem~~ problem reduces to L (this means that L and satisfiability problems are polynomially equivalent

* NP-Complete

A problem L is NP-complete if L is NP-Hard and L belongs to NP

A decision problem D is NP-complete if D belongs to NP and for every D' belongs to NP, $D' \leq D$



HOW DO WE PROVE A PROBLEM IS NP-COMPLETE

Given a problem U, the steps involved in proving that it is NP-complete are the following

1. Show that U is in NP
2. Select a known NP-complete problem V
3. Construct a reduction from V to U.
4. Show that the reduction requires polynomial time

Following methods are adopted to cope with NP-complete problems

1. Use dynamic programming, backtracking or branch and bound technique to reduce the computational cost. This might work only if the problem size is not too large.
2. Find sub-problems of the original problem that have polynomial time solution
3. Use approximation algo to find approximate solution in polynomial time.
4. Use randomized algo. to find solution in affordable time with high probability of correctness of solution.

eg Vertex-cover is NP-complete.

THE SUBSET-SUM PROBLEM is NP-COMPLETE

Proof:-

To show that SUBSET-SUM is in NP, for an instance (S, t) of the problem, let S' be the certificate.

To show that :- 3-CNF-SAT $\leq_p$ SUBSET-SUM

1. Given a 3-CNF formula ϕ over variables $x_1, x_2, \ldots, x_n$ with clauses $C_1, C_2, \ldots, C_k$ each containing exactly 3 literals.

$$\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$$

where $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$

$C_2 = (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$C_3 = (\neg x_1 \vee \neg x_2 \vee x_3)$

$C_4 = (x_1 \vee x_2 \vee x_3)$

2. The reduction algorithm constructs an instance (S, t) of the subset-sum problem such that ϕ is satisfiable if and only if there is a subset of S whose sum is exactly t.

3. Two simplifying assumptions are made about the formula ϕ
   i) No clause contains both the variable & its negation
   ii) Each variable appears in atleast one clause.

4. The reduction creates two numbers in set S for each variable x_i (y_i and v_i) and two numbers in S for each clause C_j (s_j and s_j')

MSB ← Reduction of 3-CNF-SAT → LSB

	x_1	x_2	x_3	c_1	c_2	c_3	c_4	
v_1	1	0	0	1	0	0	1	
v_1'	1	0	0	0	1	1	0	
v_2	0	1	0	0	0	0	1	
* v_2'	0	1	0	1	1	1	0	
* v_3	0	0	1	0	0	1	0	c_1 (1)
v_3'	0	0	1	1	1	0	1	
* s_1	0	0	0	1	0	0	0	
* s_1'	0	0	0	2	0	0	0	c_2 (1)
s_2	0	0	0	0	1	0	0	
* s_2'	0	0	0	0	2	0	0	
* s_3	0	0	0	0	0	1	0	c_3 (1)
s_3'	0	0	0	0	0	2	0	
* s_4	0	0	0	0	0	0	1	
* s_4'	0	0	0	0	0	0	2	c_4 (1)
t	1	1	1	4	4	4	4	

Selected ← v_1' v_2'

⇒ A satisfying assignment of ϕ is $(x_1 = 0, x_2 = 0, x$

⇒ The target t has 1 in each digit labeled by a variable & a 4 in each digit labeled by a clause

⇒ For each variable x_i there are two integers v_i and v_i' in S. Each has 1 in the digit labeled by x_i and 0's in other variable digit. If literal x_i appears in c_j then the digit labeled by c_j in v_i contains 1.
If literal $\neg x_i$ appears in c_j then the digit labeled by c_j in v_i contains 1 rest all are

⇒ for each clause c_j there are two integers, s_j and s_j' in S. For s_j, there is 1 in the c_j digit and s_j' has 2 in that digit, rest

These are SLACK VARIABLES.

* $S' \subseteq S$ is marked with "*" and contains v_1', v_2', v_3 to the satisfying assignment $s_1, s_1', s_2', s_3, s_4, s_4'$ to achieve the target value 4.

$S'(v_1, v_1', v_2, v_2', v_3, v_3', s_1, s_1', s_2, s_2', s_3, s_3', s_4, s_4')$

6. The reduction can be performed in polynomial time value
The set S contains $2n + 2k$ values $(n=3, k=11)$ $(\because 14$
each of which has $n+k$ digits and the time to produce each digit is polynomial in $n+k$.
The target t has $n+k$ digits and the reduction produces each in constant time

— To show that [3-CNF formula ϕ is satisfiable is satisfiable if and only if there is a subset $S' \subseteq S$ whose sum is t.

Now,

→ Suppose that there is a subset $S' \subseteq S$ that sums to t. The subset S' must include exactly one of v_i and v_i' for each $i = 1, 2, \ldots, n$

→ We claim that every clause c_j for $j = 1, 2, \ldots, k$ is satisfied by this assignment. For this the subset S' must include atleast one v_i and v_i' value that has a 1 in the digit labeled by c_j

→ If S' includes a v_i that has a 1 in that position, then the literal x_i appears in clause c_j. since we have set $x_i = 1$ when $v_i \in S'$, clause c_j is satisfied.

→ If s' includes a v_i' that has 1 in that position then the literal $\neg x_i$ appears in C_j. Since we set $x_i = 0$ when $v_i' \in s'$, clause C_j is again satisfied.

Thus all clauses of ϕ are satisfied, which completes the proof.

* APPROXIMATION PROBLEM.

→ The subset-sum problem.

1. A fully polynomial-time approximation scheme for the subset-sum problem can be derived by "trimming" each list L_i after it is created.
2. A trimming parameter δ is used such that $0 < \delta < 1$
3. To trim a list L by δ, means to remove as many elements from L as possible, in such a way that if L' is the result of trimming L then for every element y that was removed from L there is an element z still in L' that approximates y i.e $\frac{y}{1+\delta} \leq z \leq y$.

The following procedure trims list $L = \{y_1, y_2, \ldots, y_n\}$ in time $\Theta(m)$, given L and δ, and assuming that L is sorted in monotonically ascending order.

```
TRIM (L, δ)
1  m ← |L|
2  L' ← {y1}
3  last ← y1
4  for i ← 2 to m
5      do if yi > last · (1+δ)
6          then append yi to the end of L'
7               last ← yi
8  return L'
```

```
APPROX-SUBSET-SUM(S, t, ε)
   // input is a set S = {x1, x2, ... xn}
   // target integer, t
   // approximation parameter ε, where 0<ε<1.

1  n ← |S|
2  L0 ← (0)
3  for i ← 1 to n
4      do Li ← MERGE-LISTS (Li-1, Li-1 + xi)
5         Li ← TRIM (Li, ε/2n)
6         remove from Li every element > than t.
7  let z* be the largest value in Ln
8  return z*
```

The above procedure returns a correct approximation if one exists.

Example.

S = (104, 102, 201, 101)

t = 308

ε = 0.40

$\delta = \frac{\varepsilon}{2 \times 4} = \frac{\varepsilon}{8} = 0.05$

line 2: $L_0 = (0)$

line 4: $L_0 = (0, 104)$

line 5: $L_1 = (0, 104)$

line 6: $L_1 = (0, 104)$ ✓

line 4: $L_2 = (0, 102, 104, 206)$ ✓ (y_1, y_2, y_3, y_4)

line 5: $L_2 = (0, 102, 206)$

line 6: $L_2 = (0, 102, 206)$

line 4: $L_3 = (0, 102, 201, 206, 303, 407)$

line 5: $L_3 = (0, 102, 201, 303, 407)$

line 6: $L_3 = (0, 102, 201, 303)$ ✓

line 4: $L_4 = (0, 101, 102, 201, 203, 206, 302, 303, 307, 404)$

line 5: $L_4 = (0, 101, 201, 302, 404)$

line 6: $L_4 = (0, 101, 201, 302)$ ✓

The algorithm return $z^* = 302$ as its answer which is well within ε = 40% of the optimal answer

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 the vertex cover is the number of vertices in it.

→ The vertex cover problem is to find a vertex cover of minimum size in a given undirected graph. Such a vertex covers is an optimal vertex cover.

→ The following approximation algorithm takes as input an undirected graph G and returns a vertex cover whose size is guaranteed to be no more than twice the size of an optimal vertex cover.

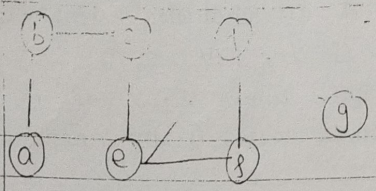
ALGORITHM.

```
APPROX - VERTEX - COVER (G)
1. C ← ϕ
2. E' ← E[G]
3. while E' ≠ ϕ
4.     do let (u,v) be an arbitrary edge in E'
5.        C ← C ∪ {u,v}
6.        remove from E' every edge incident on either u or v
7. return C
```

ANALYSIS :-

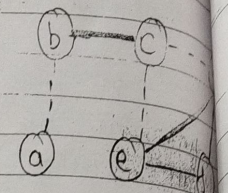
The running time of this algorithm is O(V+E) using adjacency list to represent E'

OPERATION OF APPROX - VERTEX - COVER



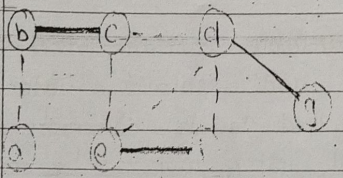
Input graph with 7 edges and 8 vertices.

ii)



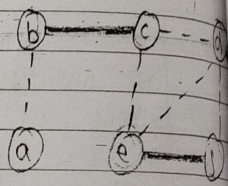
Vertices b and c are to the set C. Dashed are removed since they now covered by some vertex

iii)



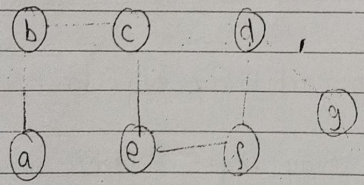
Vertices e, f are added to set C

iv)



Vertices d and g a added to set C

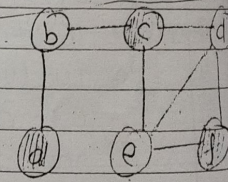
v)



Set C, which is the vertex cover produced by APPROX - VERTEX - COVER contains 6 vertices

b, c, d, e, f, g.

vi



Optimal vertex cov which contains onl 3 vertices b, d, e

The travelling salesman problem

In TSP, we are given a complete undirected graph $G = (V, E)$ that has a non-negative integer cost $c(u,v)$ associated with each edge $(u,v) \in E$, and we must find a ~~hamilt~~ hamiltonian cycle (a tour) of G with minimum cost.

→ let $c(A)$ denote the total cost of the edges in the subset $A \subseteq E$

$$c(A) = \sum_{(u,v) \in A} c(u,v)$$

→ In many practical situations, it is always cheapest to go directly from a place u to a place w, going by way of any intermediate stop or can't be (cheap) less expensive. In other words, cutting out an intermediate stop never increases the cost.

→ This notion can be formulated with the cost function c which statisfies is "The triangle inequality" if for all vertices $u, v, w \in V$,

$$c(u,w) \leq c(u,v) + c(v,w).$$

APPROX - TSP - TOUR (G, c)

1 select a vertex $r \in V[G]$ to be a "root" vertex.
2 compute a MST T for G from root r using
MST - PRIM (G, c, r)
3 let L be the list of vertices visited in a preorder tree walk of T.
4 return Hamiltonian cycle H that visits the vertices in the order L

Linear Programming

1 The SIMPLEX ALGORITHM

→ The simplex algorithm is the classical method for solving linear programs.

→ The solution proceeds iteratively.

→ Associated with each iteration will be "basic solution" that we can easily obtain from the slack form of the linear program.

Set each non-basic variable to 0 and compute the values of the basic variables from the equality constraints. An iteration converts one slack form into equivalent slack form.

→ The objective value of the associated basic feasible solution will be no less than that at the previous iteration & usually greater.

→ To achieve this increase in the objective value, we choose a non basic variable such that if we were to increase that variable's value from 0, then the objective value would also increase. (non-basic variable)

→ We raise it until some basic variable becomes 0.

→ We then rewrite the slack form, exchanging the roles of that basic variable & the chosen non-basic variable.

→ It simply rewrites the linear program until an optimal solution becomes "obvious".

Example:-

subject to : $x_1 + x_2 + 3x_3 \leq 30$ --(1) x_4

$2x_1 + 2x_2 + 5x_3 \leq 24$ --(2) x_5

$4x_1 + x_2 + 2x_3 \leq 36$ --(3) x_6

$x_1, x_2, x_3 \geq 0$ $x_1 + x_2 + 3x_3 - x_4 \leq 0$

→ Non-basic variable.

Introduce the slack variables x_4, x_5, x_6 in the above eqn (1,2,3) and rewrite the linear program as

$z = 3x_1 + x_2 + 2x_3$ - (A)

$x_4 = 30 - x_1 - x_2 - 3x_3$ - (4)

$x_5 = 24 - 2x_1 - 2x_2 - 5x_3$ - (5)

$x_6 = 36 - 4x_1 - x_2 - 2x_3$ - (6) ✗

BASIC SOLUTION

set all the (non-basic) variables on RHS to 0 & then compute the values of (basic) variables on LHS.

Thus, Basic solution is $(\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4, \bar{x}_5, \bar{x}_6)$

$= (0, 0, 0, 30, 24, 36)$.

objective value,

$z = (3 \times 0) + (0 \times 1) + (2 \times 0)$

$z = 0$.

If basic solution is also feasible, it is known as BASIC FEASIBLE SOLUTION.

✗ Our goal in each iteration, is to reformulate the linear program so that the basic solution has greater objective value.

ITERATION 1:- Consider x_1

As we increase x_1, the values of x_4, x_5, x_6 will decrease.

Put x_2, $x_3 = 0$

Now what should be the value of x_1 such that $x_5 = 0$, $x_6 = 0$ for eqn (4,5,6)

Thus, For $x_4 = 0$, x_1 should be (30).

$x_5 = 0$, $x_1 = \frac{24}{2} = (12)$ | $0 = 24 - 2x_1$

$x_6 = 0$, $x_1 = \frac{36}{4} = 9$)

Select the minimum value of x_1 and switch the roles of x_1 & x_6.

∴ Eqn (6) is modified as;

$$4x_1 = 36 - x_2 - 2x_3 - x_6$$

$$\therefore \boxed{x_1 = 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4}} \quad -- (7)$$

Substitute "x_1" in eqn (A) (4,5)

$$\therefore z = 3\left(9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4}\right) + x_2 + 2x_3$$

$$\therefore \boxed{z = \frac{27}{-} + \frac{x_2}{4} + \frac{x_3}{2} - \frac{3x_6}{4}} \quad -(8)$$

$$\boxed{x_1 = 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4}} \quad \cdot (8)$$

$$x_4 = 30 - \left(9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4}\right) - x_2 - 3x_3.$$

$$\boxed{x_4 = 21 - \frac{3x_2}{4} - \frac{5x_3}{2} + \frac{x_6}{4}} \quad \cdot (9)$$

$$x_5 = 24 - 18 + \frac{x_2}{2} + x_3 + \frac{x_6}{2} - 2x_2 - 5x_3$$

$$\boxed{x_5 = 6 - \frac{3x_2}{2} - 4x_3 + \frac{x_6}{2}} \quad \cdot (10)$$

BASIC SOLUTION = (9, 0, 0, 21, 6, 0)

ITERATION 2 :- Consider x_3. Any one (from x_2, x_3 can be chosen)

* Put x_2, $x_6 = 0$ & find x_3 for (8,9,10)

For $x_1 = 0$, $x_3 = 18$,

$x_4 = 0$, $x_3 = 42/5$

$x_5 = 0$, $x_3 = (3/2)$

$3x_3 = 21$ | $4x_3 = 6$

$x_3 = \frac{42}{5}$ | $x_3 = \frac{6}{4}$

~~Switch~~ Switch the roles of x_5 & x_3.

∴ Eqn 10 will be now;

$$4x_3 = 6 - \frac{3x_2}{2} + \frac{x_6}{2} - x_5$$

$$\therefore \boxed{x_3 = \frac{3}{2} - \frac{3x_2}{8} + \frac{x_6}{8} - \frac{x_5}{4}} \quad \frac{3}{8}x_2 = \frac{3}{2}$$

$x_2 = \frac{3 \times 8}{2 \times 3}$

Substitute "x_3" in (B, 8, 9)

$$z = 27 + \frac{x_2}{4} + \frac{1}{2}\left(\frac{3}{2} - \frac{3x_2}{8} + \frac{x_6}{8} - \frac{x_5}{4}\right) - \frac{3x_6}{4}$$

$$= 27 + \frac{x_2}{4} + \frac{3}{4} - \frac{3x_2}{16} + \frac{x_6}{16} - \frac{x_5}{8} - \frac{3x_6}{4}$$

$$\boxed{z = \frac{111}{4} + \frac{x_2}{16} - \frac{x_5}{8} - \frac{11x_6}{16}} \quad - C$$

$$x_1 = 9 - \frac{x_2}{4} - \frac{1}{2}\left(\frac{3}{2} - \frac{3}{8}x_2 + \frac{x_6}{8} - \frac{x_5}{4}\right) - \frac{x_6}{4}$$

$$= 9 - \frac{x_2}{4} - \frac{3}{4} + \frac{3x_2}{16} - \frac{x_6}{16} + \frac{x_5}{8} - \frac{x_6}{4}$$

$$\boxed{x_1 = \frac{33}{4} - \frac{x_2}{16} + \frac{x_5}{8} - \frac{5x_6}{16}} \quad -11 \quad \frac{x_2}{16} = \frac{33}{4}$$

$$x_4 = 21 - \frac{3x_2}{4} - \frac{5}{2}\left(\frac{3}{2} - \frac{3}{8}x_2 + \frac{x_6}{8} - \frac{x_5}{4}\right) + \frac{x_6}{4}$$

$$= 21 - \frac{3x_2}{4} - \frac{15}{4} + \frac{15x_2}{16} - \frac{5x_6}{16} + \frac{5x_5}{8} + \frac{x_6}{4}$$

$$\boxed{x_4 = \frac{69}{4} + \frac{3x_2}{16} + \frac{5x_5}{8} - \frac{x_6}{16}} \quad -12 \quad \frac{-3x_2}{16} = 6$$

Basic solutions:- $\left(\frac{33}{4}, 0, \frac{3}{2}, \frac{69}{4}, 0, 0\right)$

OBJECTIVE VALUE = $\frac{111}{4}$

ITERATION 3:- Consider x_2

For $x_3 = 0$, $x_2 = 4$

For $x_1 = 0$, $x_2 = 132$

For $x_4 = 0$, $x_2 = -92 \rightarrow$ co ?

Switch the roles of x_3 & x_2

$$\frac{3}{8}x_2 = \frac{3}{2} + \frac{x_6}{8} - \frac{x_5}{4} - x_3$$

$$x_2 = \frac{8}{3}\left(\frac{3}{2} + \frac{x_6}{8} - \frac{x_5}{4} - x_3\right)$$

$$\boxed{x_2 = 4 + \frac{x_6}{3} - \frac{2x_5}{3} - \frac{8x_3}{3}}$$

Substitute "x_2" in C, 11, 12

$$z = \frac{111}{4} - \frac{1}{4}x_3 - \frac{1}{16}\left(4 + \frac{x_6}{3} - \frac{2x_5}{3} - \frac{8x_3}{3}\right) - \frac{x_5}{8} - \frac{x_6}{4}$$

$$= \frac{111}{4} + \frac{4}{16} + \frac{x_6}{48} - \frac{2x_5}{48} - \frac{x_3}{6} - \frac{x_5}{8} - \frac{11x_6}{16}$$

$$\boxed{z = 28 - \frac{x_3}{6} - \frac{x_5}{6} - \frac{2x_6}{3}}$$

11y:-

$$x_1 = 8 + \frac{x_3}{6} + \frac{x_5}{6} - \frac{x_6}{3}$$

$$x_4 = 18 - \frac{x_3}{2} + \frac{x_5}{2}$$

BASIC SOLUTION $(8, 4, 0, 18, 0, 0)$

Objective value = 28.

Our solution is;

$x_1 = 8$

$x_2 = 4$

$x_3 = 0$

objective value :- $= 3 \times 8 + 4 + 2 \times 0$

$= 24 + 4 + 0$

$= 28$

ALGORITHM

```
SIMPLEX (A, b, c)
1  (N, B, A, b, c, v) = INITIALIZE-SIMPLEX (A, b, c)
2  let Δ be a new vector of length n
3  while some index j ∈ N has cj > 0
4      choose an index e ∈ N for which ce > 0
5      for each index i ∈ B
6          if aie > 0
7              Δi = bi / aie
8          else
9              Δi = ∞
10     choose an index l ∈ B that minimizes Δi
11     if Δl == ∞
12         return "unbounded"
13     else (N, B, A, b, c, v) = PIVOT (N, B, A, b, c, v, l, e)
14 for i = 1 to n
15     if i ∈ B
16         x̄i = bi
17     else
18         x̄i = 0
19 return (x̄1, x̄2, ..., x̄n)
```

```
PIVOT (N, B, A, b, c, v, l, e)
1.  // Compute the coefficients of the equation for new
        basic variable xe.
2.  let Â be a new m×n matrix
3.  b̂e = bl / ale
4.  for each j ∈ N - {e}.
5.      âej = alj / ale
6.  âle = 1 / ale
7.  // Compute the coefficients of the remaining constraints
8.  for each i ∈ B - {l}
9.      b̂i = bi - aie b̂e
10.     for each j ∈ N - {e}.
11.         âij = aij - aie âej
12.     âil = - aie âel
13. // Compute the objective function
14. v̂ = v + ce b̂e
15. for each j ∈ N - {e}
16.     ĉj = cj - ce âej
17.     ĉl = - ce âel
18. // Compute new sets of basic & non basic variable
19. N̂ = N - {e} ∪ {l}.
20. B̂ = B - {l} ∪ {e}.
21. return (N̂, B̂, Â, b̂, ĉ, v̂)
```

11 DUALITY.

= Duality enables us to prove that a solution is indeed optimal

Example :- The max-flow min-cut theorem

Suppose that, given an instance of maximum problem, we find a flow f with value $|f|$. To know whether f is a max flow, by the max-flow min-cut theorem, if we can find a cut whose value is also $|f|$, then we have verified that f is indeed a maximum flow

→ This relationship provides an example of duality.

Given a maximization problem, we define a related minimization problem such that the two problems have the same optimal objective values.

→ Given a linear program in which the objective is to maximize, we shall describe how to formulate a dual linear program in which the objective is to minimize whose optimal value is identical to that of the original problem.

→ When refering to dual linear programs, we call the original linear program the "PRIMAL"

Given a primal linear program in standard form as follows.

We define the dual linear program as;

$$\text{Minimize} \quad \sum_{i=1}^{m} b_i y_i$$

$$\text{Subject to} \quad \sum_{i=1}^{m} a_{ij} y_i \geq c_j \quad \text{for } j = 1, 2, \cdots, n$$

$$y_i \geq 0 \quad \text{for } i = 1, 2, \cdots, m$$

To form the dual,

→ We change the maximization to a minimization.

→ Exchange the roles of coefficients on RHS & the objective function

→ Replace each ~~less than~~ $\leq$ by $\geq$.

→ Each of the m constraints in the primal has an associated variable y_i in the dual & Each of the n constraints in the dual has an associated variable x_j in the primal.

Example :-

→ The optimal value of the dual linear program always equal to the optimal value of the primal linear program.

→ The simplex algorithm actually implicitly solves both the primal & the dual linear programs simultaneously, thereby providing a proof of optimality.

Maximum Bipartite Matching

* Motivating example:

J: jobs C: candidates

L_i = List of jobs candidate i can do

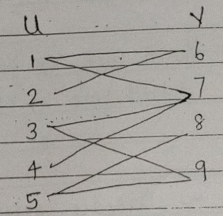
Constraint:- Each candidate must be given almost 1 job.
Each job must be assigned to almost 1 candidate

Goal:- Assign candidates to jobs such that maximum numbes of jobs are filled.

* Bipartite Maximum Matching

Input:- Bipartite graph $G = (U, V, E)$ where U & V are vertex sets with $|U| = n_1$, $|V| = n_2$ (edges go only between U & V) E = set of edges

eg:-

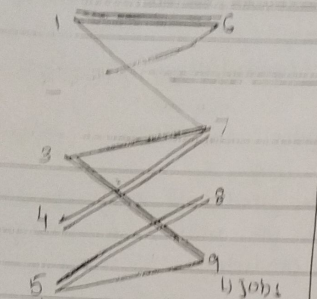


No edges within U and V within V

Matching:- $M \subseteq E$ is said to be matching if almost one edge from M is incident on any vertex (in U or in V)

1, 6, 7, 8, 9 are not eligible

eg:-



5 candidates

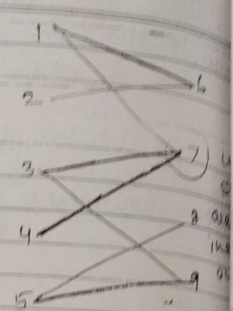
4 jobs

size: 3

(not maximum)

max matching with size = 4

Set of Matching edges are
(1,6), (3,7), (5,9)

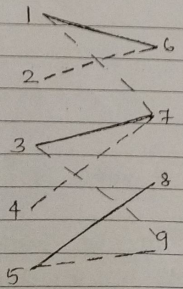


Goal:- Matching of maximum possible size.
→ job assignment

* candidate 1 can do job 6, 7.

I greedy idea :- Keep on adding edges into M till no more edges can be added (Make sure there are no conflicts)

eg:-



solid line ⇒ included in M
dotted line ⇒ conflicts are there so not considered

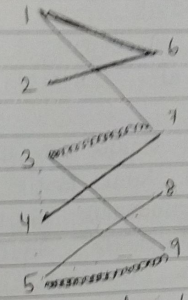
⇒ greedy approach

I

Free vertex for M: not an endpoint of any edge in M

Free vertices are :- 2, 4, 8

(red) M+1 3 edges
(blue) Removed 2 edges
Still have 1
4-7-3-9 : 2



14,000
7000

T a b c d e f g h / a b c d a b c a b c d f
P d e f / a b c d f

a b c d a b c

prefix a, ab, abc, abcd
suffix c, bc, abc, dabc

T a b a b c a b c a b a b a b d
P a b a b d

~~Push - Relabel Algorithm~~

KNAPSACK PROBLEM

① Greedy approach.

Fractional knapsack

- If a fraction x_i of object i is placed into the knapsack such that $0 \le x_i \le 1$ then the profit earned is $p_i x_i$

→ The objective is to obtain a filling of the knapsack that maximizes the total profit earned.

→ Since capacity of knapsack is M, we require the total weights of all chosen objects to be at most M.

Thus, the problem is stated as follows :-

$$\text{Maximize} \sum_{1 \le i \le n} p_i x_i \quad -(1)$$

$$\text{Subject to} \sum_{1 \le i \le n} w_i x_i \le M \quad -(2)$$

$$\text{and} \quad 0 \le x_i \le 1 \; ; \; 1 \le i \le n \quad -(3)$$

The profits and weights are positive numbers.

Feasible solution is any set $(x_1, x_2, \ldots, x_n)$ satisfying equation (2) & (3).

While an optimal solution is a feasible solution for which eq(1) is maximized.

ALGORITHM :-

```
Greedy knapsack (M, n, W, P)
for i ← 1 to n
    x[i] = 0.0
    U = M;
for i ← 1 to n do
    if (W[i] > U)
        then break;
    x[i] = 1.0
    U = U - W[i];
if (i ≤ n)
    x[i] = U / W[i];
```

Example :-

$n = 3, \quad M = 20$

$(P_1, P_2, P_3) = (25, 24, 15)$

$(W_1, W_2, W_3) = (18, 15, 10)$

I. Calculate P_i / W_i

n	P_i	W_i	P_i / W_i
1	25	18	1.3
2	24	15	1.6
3	15	10	1.5

II. Sort in descending order of P_i / W_i

2	24	15	1.6
3	15	10	1.5
1	25	18	1.3

(iii) Fill Knapsack.

i		Item wt	Profit	Sol
2	24	15	20-15 = 5	24
3	15	10	rem. wt / wt. = 5/10 = 1/2	15 × 1/2
			(10 × 1/2) − 5 = 0	= 7.5
				31.5

Max. Profit = 31.5

optimal solution = $(0, 1, \frac{1}{2})$

Time complexity = O(n log n)

n = 4 MW = 5

	P	W
1	3	2
2	4	3
3	5	4
4	6	5

M = 25

W	0	1	2	3	4	P	W
						25	16
						23	10
						26	6

S^0 (0,0)

S^0_1 (3,2)

S^1 (0,0) (3,2)

S^1_1 (4,3) (7,5)

S^2 (0,0) (3,2) (4,3) (7,5)

S^2_1 (5,4) (8,6) (9,7) (12,9)

S^3 (0,0) (3,2) (4,3) (7,5) (5,4) (8,6) (9,7) (12,9)

S^3_1 (6,5) (9,7) (10,8) (13,10) (11,9) (14,11) (15,12) (18,14)

S^0 (0,0) (1,0)

S^0_1 (25,16)

S^1 (0,0) (25,16)

S^1_1 (23,10) (48,26)

S^2 (0,0) (25,16) (23,10) (48,26)

S^2_1 (16,5) (41,21) (39,15) (64,31)

S^3 (1,1,0,0)

(ii) (i) KNAPSACK PROBLEM

→ We are given n-objects and a Knapsack or a bag in which the object i that has weight w_i is to be placed. The Knapsack has capacity W. Then the profit that can be earned is $P_i x_i$. The objective is to obtain filling of Knapsack with maximum profit earned.

Maximize :- $\sum_{n}^{1} P_i x_i$

Subject to :- $\sum_{n}^{1} w_i x_i \leq W$.

where $1 \leq i \leq n$ and n is total no. of objects

$x_i = 0$ or $x_i = 1$.

STEPS used in Dynamic Programming

→ Notations used are

$f_i(y_j)$ ⇒ value of optimal solution.

S^i is a pair (P, W) where $P = f_i(y_j)$

$W = y_j$

→ Initially $S^0 = \{(0,0)\}$.

Compute S^{i+1} from S^i.

$S^i_1 = \{(P, W) \mid (P - P_i, W - w_i) \in S^i\}$

S^{i+1}_1 can be computed by merging $S^i + S^i_1$

2) n=4, M=21

	P_i	W_i
1	2	10
2	5	15
3	8	6
4	1	9

P W
60 10
~~100 20~~ M=50
120 30

$S^0 = \{(0,0)\}$.

$S_1^0 = \{(2,10)\}$

I S^1 = Merge S^0 & S_1^0
$= \{(0,0), (2,10)\}$.

$S_1^1 = \{(5,15), (7,25)\}$.

II S^2 = Merge S^1 & S_1^1
$= \{(0,0), (2,10), (5,15), (7,25)\}$

$S_1^2 = \{(8,6), (10,16), (13,21), (15,31)\}$.

III S^3 = Merge S^2 & S_1^2
$= \{(0,0), (2,10) (5,15) (7,25) (8,6) (10,16) (13,21) (\ldots$

$S_1^3 = \{(1,9) (3,19) (6,24) (8,24) (9,15) (11,25) (14,30)$
$(16,40)\}$

IV ~~S~~ S^4 = Merge S^3 & S_1^3
$= \{(0,0), (2,10), (5,15) (7,25) (8,6) (10,16) (13,21)$
$(15,31), (1,9) (3,19) (6,24) (8,24) (9,15$
$(11,25) (14,30) (16,40)\}$. ↑

M=21
And (13,21) pair is in
$\therefore x_3 = 1$

Remaining $(P, W) = (5,15)$

$(5,15) \in S^2 \therefore x_2 =$

Thus final solution = (0

$x_1 x_2 x_3 x_4$
0 1 1 1

* TRAVELLING SALESMAN PROBLEM

Problem:-

Let $G = (V, E)$ be a directed graph with edge cost c_{ij}. The variable c_{ij} is defined such that,

$c_{ij} > 0$ for all $i \neq j$

$c_{ij} = \infty$ if $(i,j) \notin E$

Let $|V| = n$ & assume $n > 1$

A tour of G is a directed simple cycle that includes every vertex in V. The cost of tour is the sum of the cost of the edges on the tour. The TSP is to find a tour of minimum cost.

Application:-

- To use a robot arm to tighten the nuts on some piece of machinery on an assembly line.
- The arm will start from its initial position, successively move to each of the remaining nuts & return to the initial position.
- The path of the arm is a tour on a graph in which vertices represent the nuts.
- A MINIMUM COST tour will minimize the time needed for the arm to complete its task.

PROBLEM SOLUTION:

- Every tour consists of an edge $(1,k)$ for some $k \in V - \{1\}$ and a path from vertex k to vertex 1.

- The path from vertex k to vertex 1 goes through each vertex in V - {1, k} exactly once.
- If the tour is optimal then the path from
must be shortest path going through all v in V - {1, k}. Hence the principle of optimality holds.

Let g(i, s) be the length of a shortest path starting at vertex i, going through all vertices in S & terminating at vertex 1.

The function g(1, V - {1}) is the length of an optimal salesperson tour.

From the principle of optimality it follows that

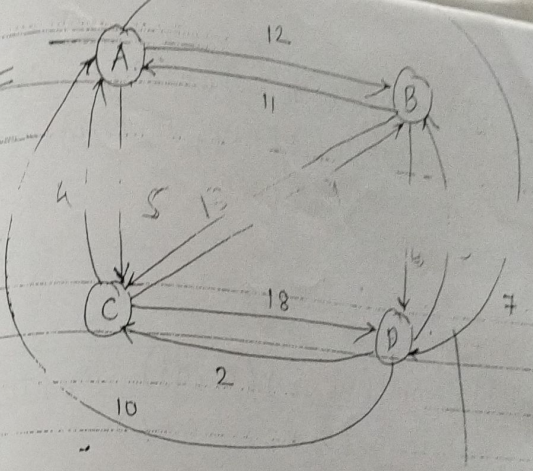
$$g(1, V-\{1\}) = \min_{2 \le k \le n} \{ C_{1k} + g(k, V-\{1,k\}) \}$$

$$g(i, \phi) = C_{i1}, \quad 1 \le i \le n$$

	A	B	C	D	E
ϕ	∞	4	10	14	2
B	12	∞	6	10	4
C	16	14	∞	8	14
D	24	8	12	∞	10
E	2	6	4	16	∞

$$g(i, S) = \min_{j \in S} [w(i, j) + g(j, \{S - j\})]$$

Example



Edge length matrix

	A	B	C	D
A	0	12	5	7
B	11	0	13	6
C	4	9	0	18
D	10	3	2	0

I) $g(B, \phi) = C_{ba} = 11$
$g(C, \phi) = C_{ca} = 4$
$g(D, \phi) = C_{da} = 10$

II) $g(B, \{C\}, A) = C_{bc} + g(C, \phi) = 13 + 4 = 17$
$g(B, \{D\}, A) = C_{bd} + g(d, \phi) = 6 + 10 = 16$

$g(C, \{B\}, A) = C_{cb} + g(b, \phi) = 9 + 11 = 20$
$g(C, \{D\}, A) = C_{cd} + g(d, \phi) = 18 + 10 = 28$

$g(D, \{C\}, A) = C_{dc} + g(C, \phi) = 2 + 4 = 6$
$g(D, \{B\}, A) = C_{db} + g(b, \phi) = 3 + 11 = 14$

III $g(A,\{B,C\}) = \min(C_{ab} + g(b,\{c\}), C_{ac} + g(c,$

$= \min(12+17, 5+20)$

$= \min(29, 25)$

$g(A,\{B,C\}) = 25$

$g(B,\{C,D\}) = \min(C_{bc} + g(c,\{d\}), C_{bd} + g(d,\{c\}))$

$= \min(13+28, 6+6)$

$g(B,\{C,D\}) = 12$

B } C, D

$C_{bc} + g(cd$) C_{bd}

$g(C,\{B,D\})$

III $g(B,\{C,D\},A) = \min(C_{bc} + g(c,d,a), C_{bd} + g(d,c,a))$

$= \min(13+28, 6+6)$

$g(B,\{C,D\},A) = 12$

$g(C,\{B,D\},A) = \min(C_{cb} + g(b,d,a), C_{cd} + g(d,b,$

$= \min(9+16, 18+14)$

$= 25$

$g(D,\{B,C\},A) = \min(C_{db} + g(b,c,a), C_{dc} + g(c,b,a))$

$= \min(3+17, 2+20)$

$= 20$

IV $g(A,\{B,C,D\},A) = \min(C_{ab} + g(b,cd,a), C_{ac} + g(c$ dbl

$+ C_{ad} + g(d$ bc a))

$= \min(12+12, 5+25, 7+20)$

$= 24$

Optimal tour = $A \to B \to D \to C \to A$

II Obtain optimum tour for the following graph

	A	B	C	D
B	5	0	9	10
C	6	13	0	12
D	8	8	9	0

A - B - D - C - A

$10 + 12 + 9 + 6$

$2a + 6$

35

I $g(B,\phi) = C_{ba} = 5$

$g(C,\phi) = C_{ca} = 6$

$g(D,\phi) = C_{da} = 8$

II $g(B,\{c\}) = C_{bc} + g(c,\phi) = 9+6 = 15$

$g(B,\{D\}) = C_{bd} + g(d,\phi) = 10+8 = 18$

$g(C,\{D\}) = C_{cd} + g(d,\phi) = 12+8 = 20$

$g(C,\{B\}) = C_{cb} + g(b,\phi) = 13+5 = 18$

$g(D,\{B\}) = C_{db} + g(b,\phi) = 8+5 = 13$

$g(D,\{C\}) = C_{dc} + g(c,\phi) = 9+6 = 15$

III ~~$g(A,\{B,C\},A) = \min$~~

$g(B,\{C,D\},A) = \min(C_{bc} + g(c,\{d\},a), C_{bd} + g(d$

$= \min(9+20, 10+15)$

$= 25$

$g(C,\{B,D\},A) = \min(C_{cb} + g(b,\{d\},a), C_{cd} + g(d,\{b\},a)$

$= \min(13+18, 12+13)$

$= 25$

Last

$g(D, \{B, C\}, A) = \min(C_{db} + g(b, \{c\}, a), C_{dc} + g(c, \{b\}, a)$

$= \min(8 + 15, 9 + 18)$

IV $g(A, \{B, C, D\}, A) = \min(C_{ab} + g(B, \{C, D\}, A), C_{ac} + g(C, \{B, D\}, A), C_{ad} + g(d, \{B, C\}, A)$

$= \min(10 + 25, 15 + 25, 20 + 23)$

$g(A, \{B, C, D\}, A) = 35$

Optimal cost = 35

Optimal Tour = $A \to B \to D \to C \to A$

Bellmand	$O(VE)$.
dijk	$O(E \log V)$.
floyd - warshal	$O(n^3)$
ford - fullerson	$O(E \mid F^* \mid)$
Johnson	$O(V^2 \log V + VE)$.
TSP	
Knapsack	$(n \log n)$.
LCS	$(m * n)$.
MCM	$O(n^3)$
rabin karp	$((n-m+1)) O(n+m)$
Naive string.	(N^2) $((n-m+1)m)$