* Greedy Method :- (is a Technique)

- Simplest & straight forward approach.
- The decision is taken on basis of current available infor.ⁿ. without worrying about the effect of current decision in future.
- Feasible Solⁿ that may or may not be optimal.

Feasible Solⁿ → any subset that satisfy the condition.

optimal Solⁿ ⇒ Best & most favorable Solⁿ.

Characteristic & Features :-

(1) To construct the Solⁿ in an optimal way, Algorithm maintains 2 sets :-
   - one contains chosen items &
   - other " rejected items.

(2) Greedy algorithm make good local choices.
   - an optimal Solⁿ
   - feasible Solⁿ

Scanned with CamScanner

Components of Greedy Algorithm :-

(1) **A Candidate Set :-**
A sol.^{n} is created from this set.

(2) **A selection fun.^{c} :-**
used to choose the best candidate to be added to the sol.^{n}.

(3) **A feasibility fun.^{c} :-**
Used to determine whether a candidate can be used to contribute to the sol.^{n}.

(4) **A objective fun.^{c} :-**
used to assign value to a solution or partial sol.^{n}

(5) **A solution fun.^{c} :-**
used to indicate whether a complete sol.^{n} has been reached.

Applications of Algos :-

(1) finding shortest Path
(2) finding Minimum Spanning Tree
(3) Job sequencing with deadline / activity-selection problem.
(4) fractional knapsack problem

Scanned with CamScanner

(5) Huffman coding.

Pseudocode for Greedy Algorithm;-

```
Algorithm Greedy (a, n)
{
    solution := 0;
    for i:=1 to n do
    {
        x := Select(a);
        if Feasible ( solution, x) then
            solution := Union(solution, x) ;
    }
    return solution;
}
```

Scanned with CamScanner

HUFFMAN CODE ALGORITHM :-

Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character.

This is how huffman coding makes sure that there is no ambiguity when decoding the generated bitstream.

Example :- Let there be four characters a, b, c & d and their corresponding variable length codes 00, 01, 0 & 1. This coding leads to ambiguity because code assigned to c is the prefix of codes assigned to 'a' and 'b'.

If the compressed bit stream is 0001

The de-compressed o/p may be → "cccd" ← 0001

or

"ccb" ← 0001

or

"acd" ← 0001

or

"ab" ← 001

See this for appl.n of Huffman Coding :-

There are mainly two major parts in Huffman Coding:

1) Build a Huffman Tree from i/p characters.
2) Traverse the Huffman Tree & assign codes to characters.

Scanned with CamScanner

→ Data Compression method.

→ lossless (without loss of information) compression.

Algorithm :-

Huffman (c) ← set of n character

1) n = |c|
2) Q = c // min. Priority Queue
3) for i=1 to n-1
4) do allocate a new node z
5) z.left ← x ← Extract.min(Q)
6) z.right ← y ← Extract.min(Q)
7) z.freq ← x.freq + y.freq
8) Insert (Q, z)
9) return Extract.min(Q) //return Root of the tree

Implementation →

To understand the concept of fixed & variable code :-

fixed

	a	b	c	d	e	f
freq. in Thousands →	45	13	12	16	9	5

Total freq. ⇒ 100 × 1000 = @ 1 lakh

If each character is represented by 3 bit ⇒

1 lakh × 3 = 3 lakh bit

Storage required

variable.

But in Variable Code ⇒

1×45 + 3×13 + 3×12 + 3×16 + 4×9 + 4×5

= 45 + 39 + 36 + 48 + 36 + 20

= 224

224 × 1000 = 224000 bit

Storage required

See Soln in next Page →

Scanned with CamScanner

Example:-

Character	frequency	fixed code	variable length
a	45	000	0
b	13	001	101
c	12	010	100
d	16	011	111
e	9	100	1101
f	5	101	1100

Solve using Huffman Code.

In fixed Code ⇒ we require more storage to store.

like $45 \times 3 = 135$ (3 bit)

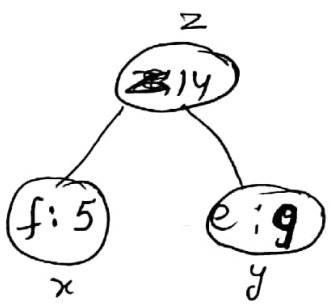
$13 \times 3 = 39$

$12 \times 3 = 36$

⋮ ⋮

For reducing storage we use variable length ⇒ by Huffman code.

1) $n = 6$
2) $Q = [a\ b\ c\ d\ e\ f]$ ← min. Priority Queue.
   $[45\ 13\ 12\ 16\ 9\ 6]$
3) for $i = 1 — 5$
4) do allocate a node (z)
5) z.left ← x ← f
6) z.right ← y ← e
7) z.freq ← x.freq + y.freq.
   z.freq ← 5 + 9
   z.freq ← 14



Scanned with CamScanner

8) Now, Insert (Q, Z)

Again go to step 3)

3) i = 2 – 5

4) allocate Z

5) Z.left ← x ← c

6) Z.right ← y ← b

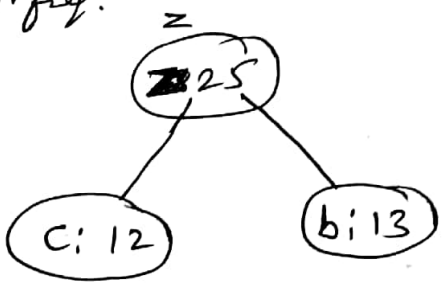
7) Z.freq. ← x.freq + y.freq.
← 12 + 13
← 25

8) Insert (Q, Z)

Repeat until 1 root node is found.

a	45	
b	13	
c	12	
d	16	
e	9	
f	5	

14



SHORT CUT :– a b c d e f

45 13 12 16 9 5

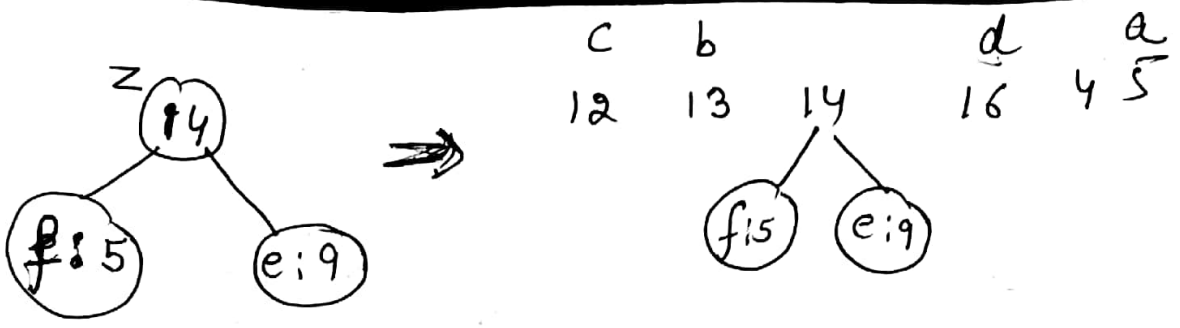
Arrange in ascending order :–

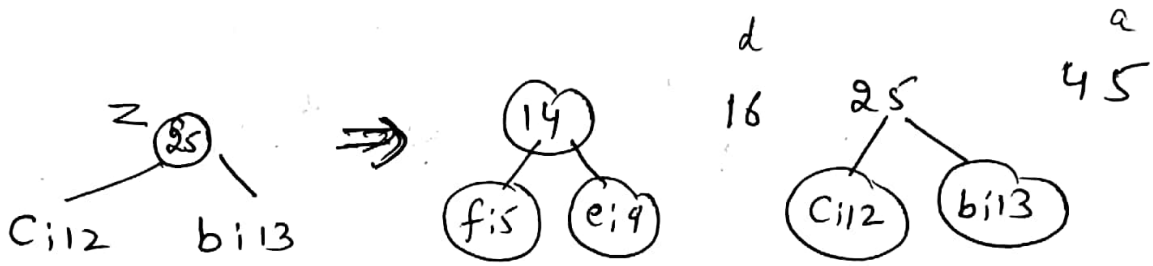
~~a b c d e f~~

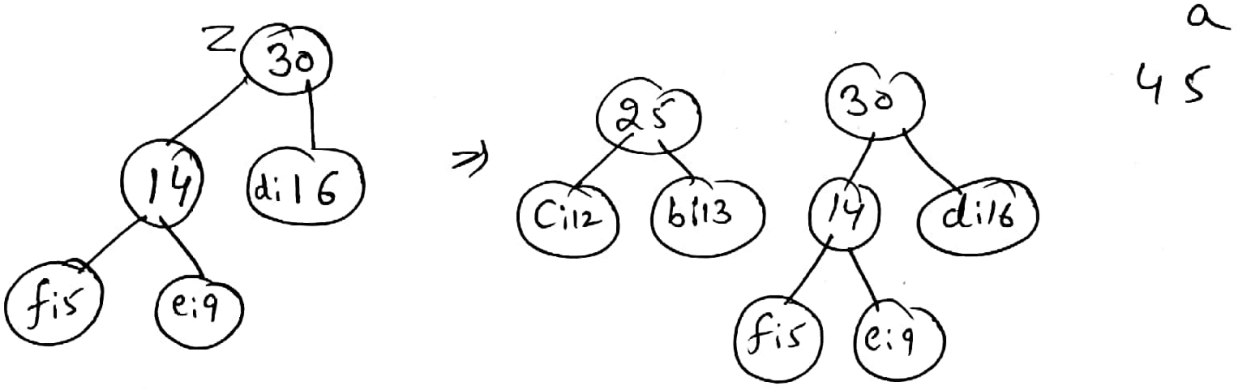
f e c b d a

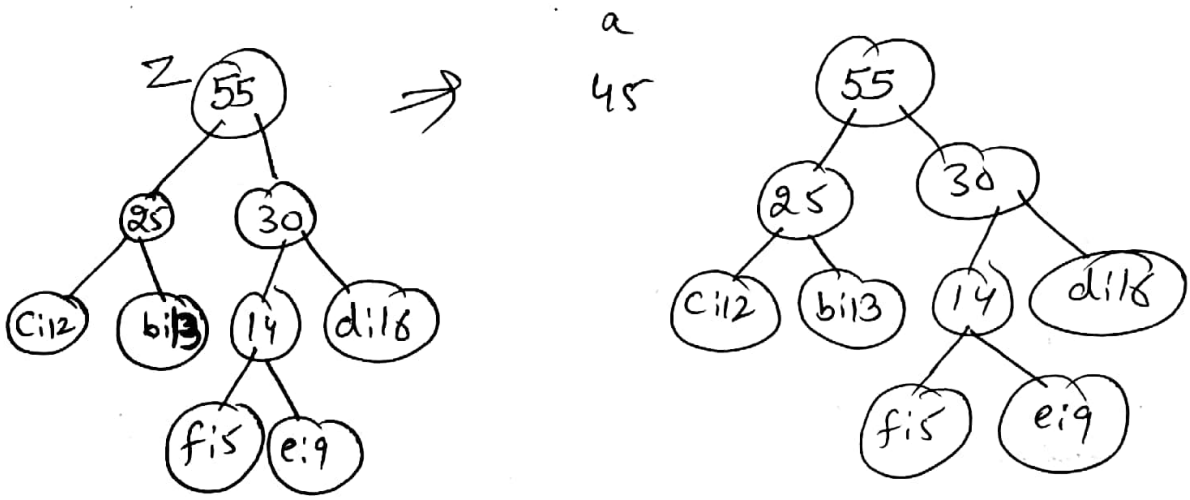
5 9 12 13 16 45

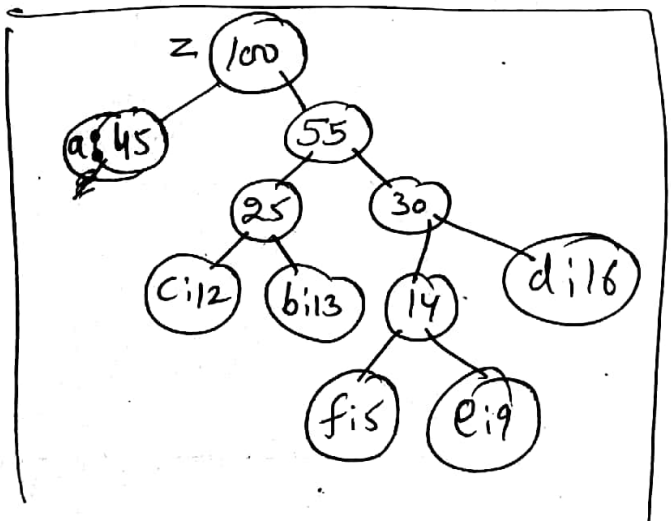
Scanned with CamScanner





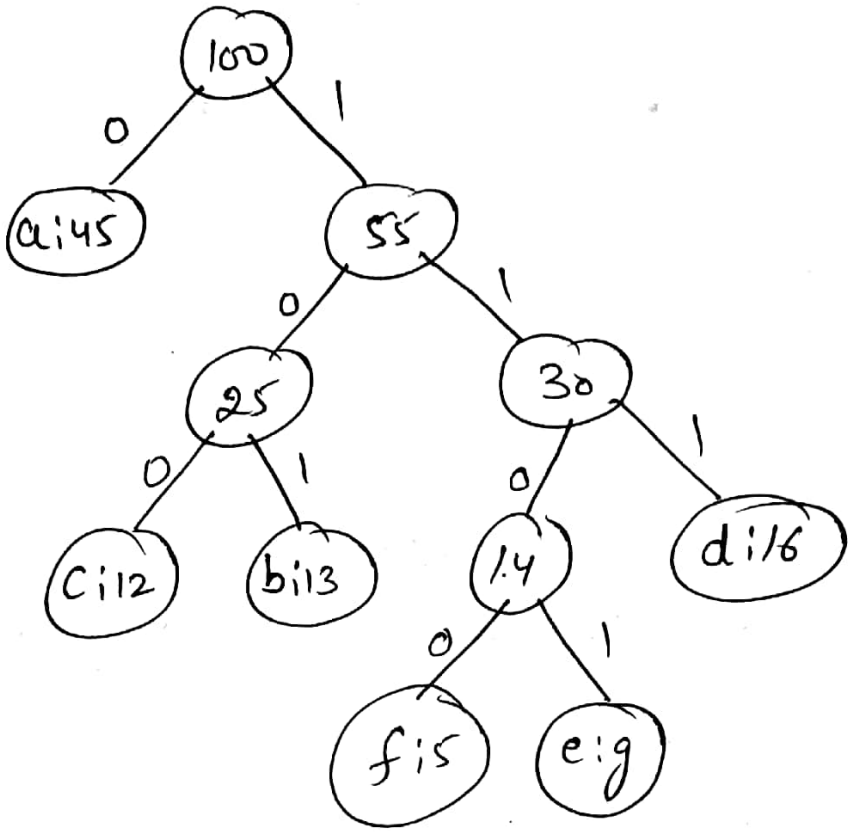






⟸ Now Assign (0) to left node & (1) to right node

Scanned with CamScanner



Now New Code :-

Variable Code

a	0
b	101
c	100
d	111
e	1101
f	1100

thus ⇒

Scanned with CamScanner

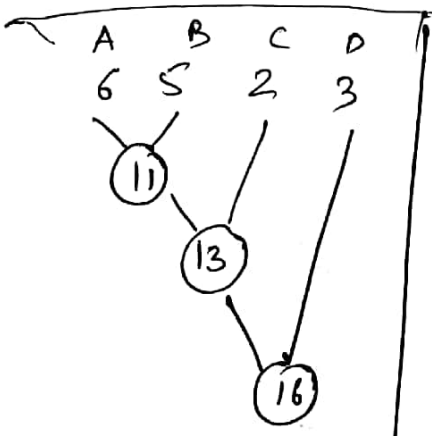
OPTICAL MERGE PATTERN

It relates to the merging of two or more sorted files in a single sorted file. This type of merging can be done by Two-way Merging method.

There are multiple ways to perform pairwise merge to get single sorted file, we have to select one which require minimum no. of comparisons.

List :- A B C D

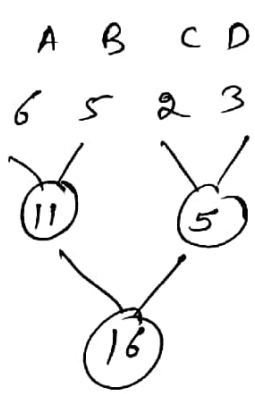
Sizes :- 6 5 2 3



11+13+16

Total Cost ⇒ = 40

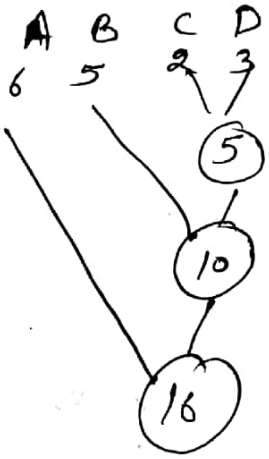
Pattern I



11+5+16

Total Cost ⇒ = 32

Pattern II



16+10+5

Total Cost = 31

Pattern III

In this we select Two Smaller list for Merging that Give Best Result i.e. Minimum result (Total Cost)

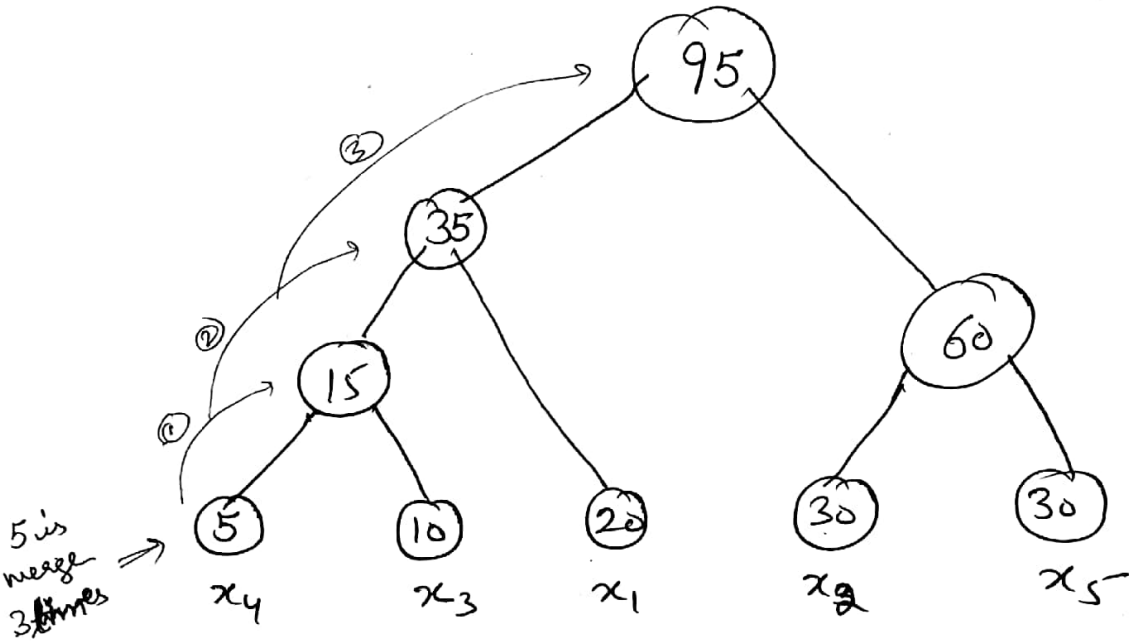
Scanned with CamScanner

Example :-1)

x_1	x_2	x_3	x_4	x_5
20	30	10	5	30

~~Select~~ Convert into increasing order →

x_4	x_3	x_1	x_2	x_5
5	10	20	30	30



Total Cost ⇒ 15 + 35 + 60 + 95 = 205
(Internal Node Size)

OR

Total Cost ⇒ How many time the No. of file being Merged that depends on the distance.

5 → is merge 3 times
10 → " " 3 "
20 → " " 2 "
30 ⇒ " " 2 "
30 → " " 2 "

⇒ 3x5 + 3x10 + 2x20 + 2x30 + 2x30

= 205

Scanned with CamScanner

Example :-

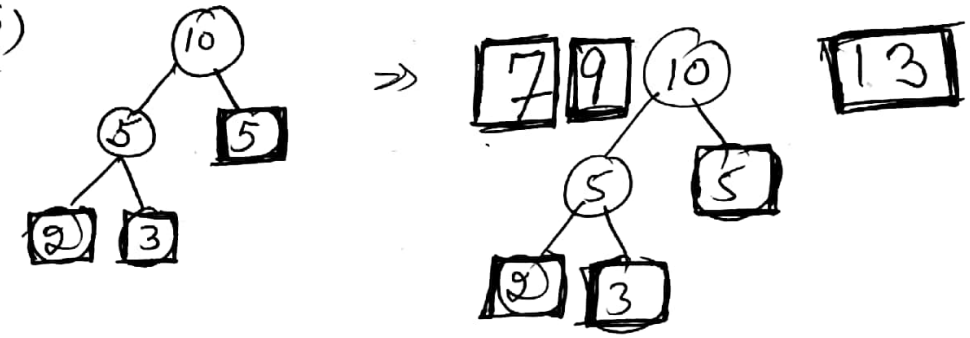
2)

Files →	x_1	x_2	x_3	x_4	x_5	x_6
Sizes →	2	3	5	7	9	13

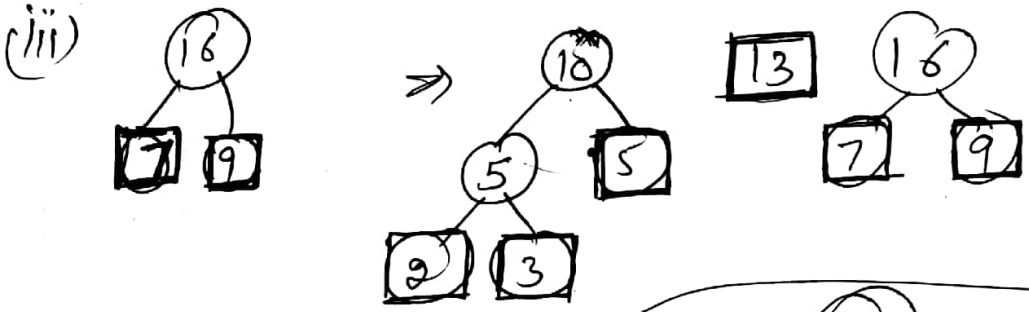
(Sorted if not first arrange in increasing order)

(i)

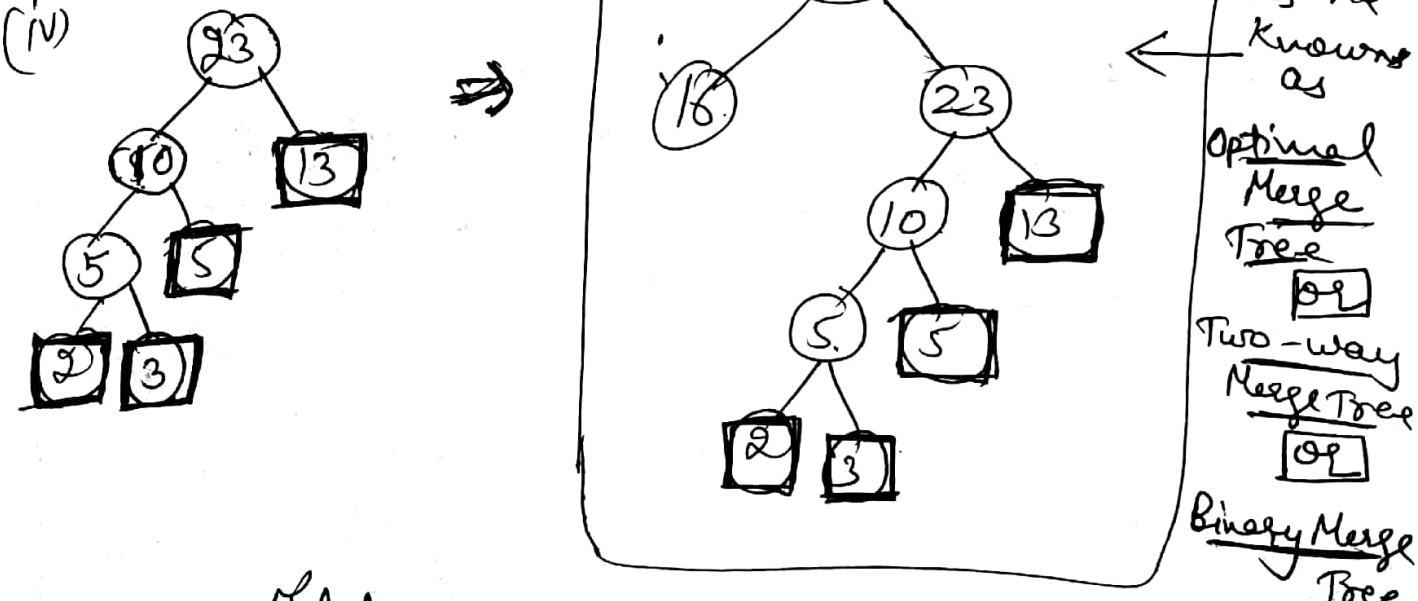
(ii)



(iii)



(iv)



This Tree Knowns as Optimal Merge Tree OR Two-way Merge Tree OR Binary Merge Tree

Total cost = 5 + 10 + 23 + 16 + 39

Optimal = 93

Scanned with CamScanner

OR

Total cost of Merging ⇒ $\boxed{\sum x_i * d_i}$

$= 4 \times 2 + 3 \times 4 + 5 \times 3 + 7 \times 2 + 9 \times 2 + 13 \times 2$

$= 8 + 12 + 15 + 14 + 13 + 26$

$= 93$ Ans.

Scanned with CamScanner

Algorithm for optimal Merge Pattern

Algorithm:- MergeTree(n) ← n single node

```
{
// list is a global list of
   n single nodes.
for i = 1 to n-1 do
{
   p = new treenode; // get a new treenode
   (p→lchild) = least(list); // least func finds node with least length of a list.
   (p→rchild) = least(list); //
   (p→weight) = ((p→lchild)→weight) + ((p→rchild)→weight);
                // Merge two trees with smallest weight.
   Insert(list, p); // Insert function insert p into list.
}
return list; // Tree left in list is the Merge Tree
}
```

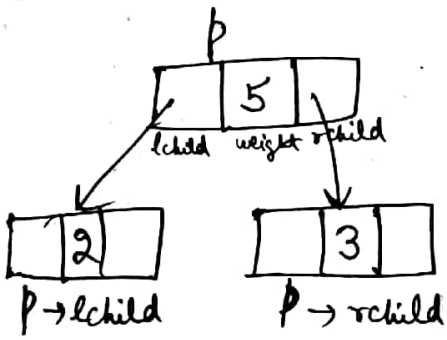
```
Struct treenode
{
  treenode *lchild;
  treenode *rchild;
  int weight;
};
```

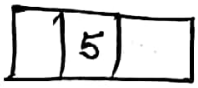
Example:- list → x_1 x_2 x_3 | n = 3

weight → 2 3 5

i = 1 to 2

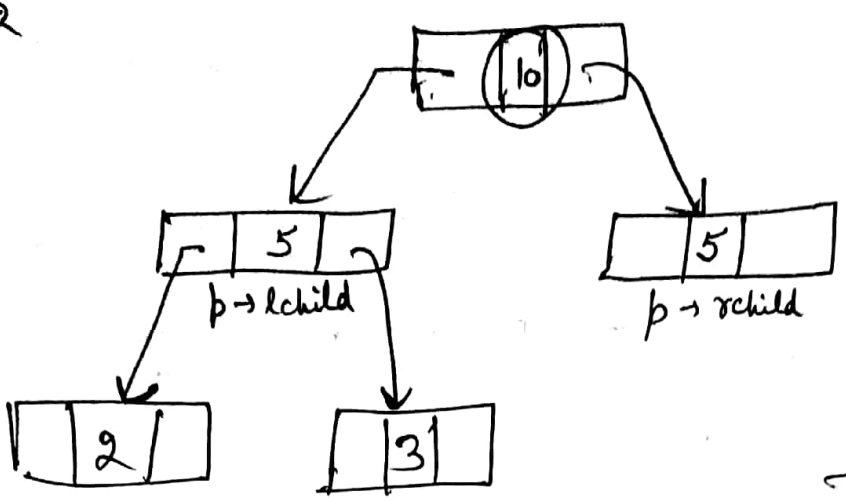
i=1





Scanned with CamScanner

i=2



optimal Merge Tree

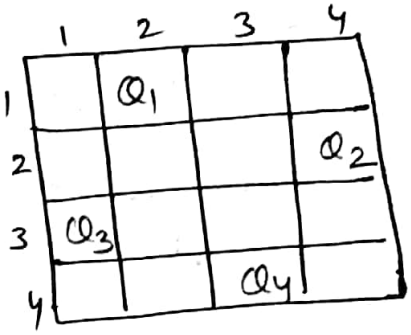
Scanned with CamScanner

N- Queens Problem :- (Backtracking)

No. of ways :- All Possible Soln

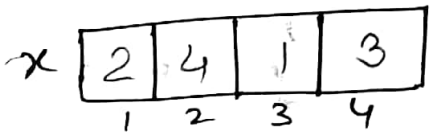
$^{16}C_4$

$Q_1\ Q_2\ Q_3\ Q_4$



Condition :-

A) 1st Queen will place in 1st row
2nd " " " " 2nd "
3rd " " " " 3rd "
4th " " " " 4th row

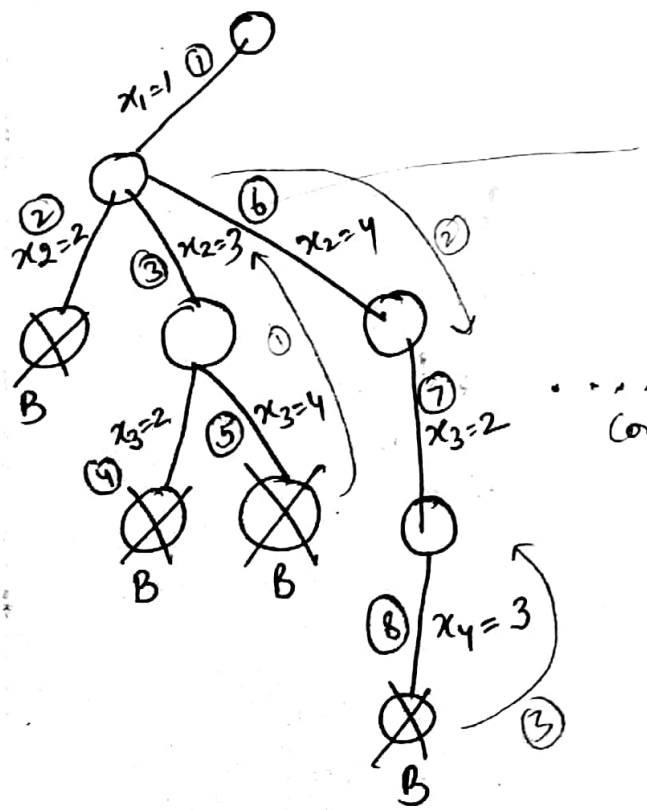


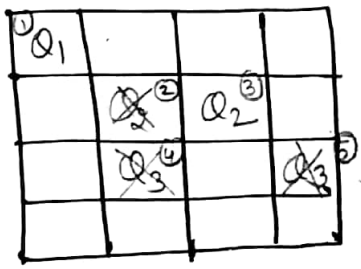
Column No. (∵ rows are fixed)

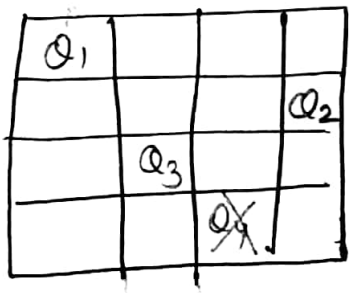
B) No two Queen are under attack.

i.e. No 2- Queen are under same Row
" " " " " Column
" " " " " Diagonal.

Sol.n ⇒ "Bounding fun.n" :- (No row/column/diagonal)

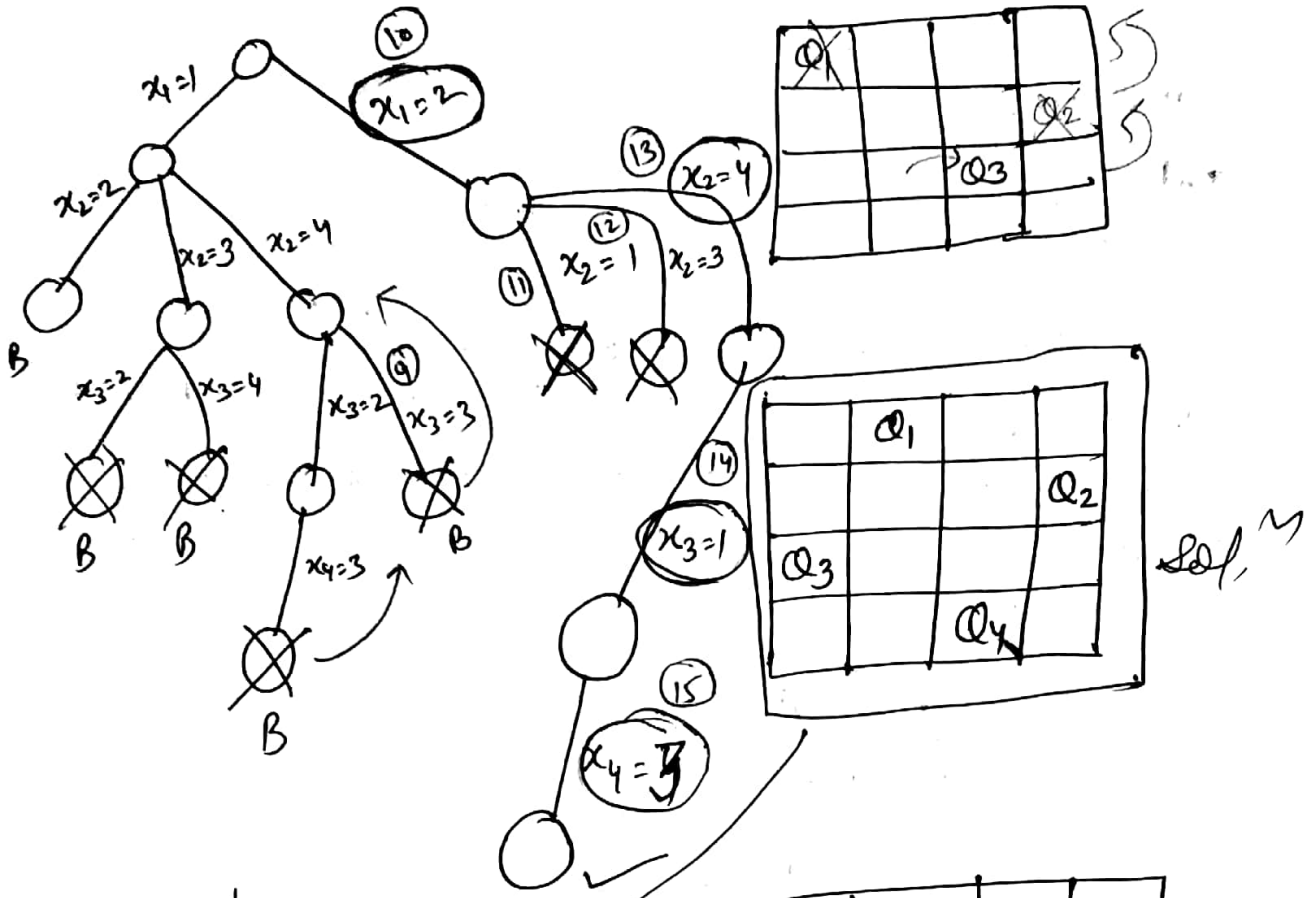






Backtrack

Scanned with CamScanner



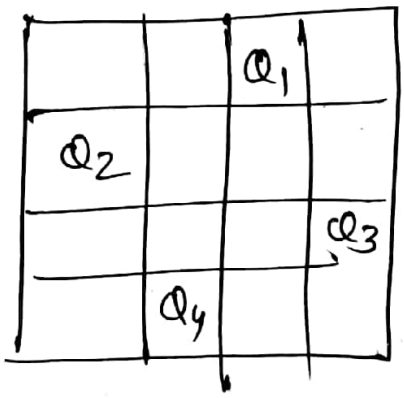
* Sequence to follow :-

(1) - (2) - (3) - (4) -- (15)

x	2	4	1	3
	1	2	3	4

Column No.

→ Mirror Image of this is also the Sol.ⁿ



x	3	1	4	2
	1	2	3	4

Column No.

& Sol.ⁿ ⇒ 2, 4, 1, 3
& 3, 1, 4, 2

Scanned with CamScanner

N-Queen:-

→ It's a classic combinational problem to place 'n' queens on n×n chessboard so that no two attack.

→ **Rules**:- No two of them should be on same row, column or diagonal.

→ All sol.^{n} to n-queens problem can therefore be represented as n-tuples $(x_1, x_2 \cdots x_n)$.

where 'x_i' is the column where queen 'i' is placed.

Scanned with CamScanner

Algorithm :- Backtracing N-Queen's Algorithm :-

Algorithm Place (k, i)

```
// return true if a is placed at kth row &
// ith column otherwise return false.
// x[ ] is a global array.
{
    for j=1 to (k-1) do
    {
        if ((x[j]=i)  // ith the same column
        or (Abs(x[j]-i) = Abs(j-k)))  // in the same diagonal
            return false
    }
    return true
}
```

Algorithm NQueens (k, n)

Let n=4
Starting Queen 1 ⇒ k=1

```
// using backtracking, this procedure prints all
   combinations of solutions.
{   for i=1 to n do
    {   if (place (k,i)) then
        {   x[k] = i
            if (k==n) then
                print (x[1:n])
            else
                NQueens (k+1, n)
        }
    }
}
```

Scanned with CamScanner

Backtracking:-

It is a general algorithmic technique that considers searching every possible combination in order to solve an optimization problem. Backtracking is also known as depth-first search or Branch & Bound.

Backtracking is an approach to problem solving which is usually applied to constraint satisfaction problems like Puzzles. In a backtracking solution, a search path is followed & the algorithm backtracks at a particular point (also known as decision point) in the path as soon as it realizes that this path won't lead to a valid solution & then it follows another path starting from previous decision point.

In this way, diff paths are repeatedly explored to arrive at the final solution.

Efficiency of Backtracking Algorithms:-

A) The Time required by a backtracking algorithm or Efficiency depends on 4 factors;-

1) The time to generate the next $X(k)$.

Scanned with CamScanner

2) The no. of $X(k)$ satisfying the explicit constraint.

3) The time for bounding fun.c "B_i"

4) The no. of $X(k)$ satisfying the "B_i" for all "i".

B) The first three are relatively independent of the problem instance being solved.

C) The Complexity for the first three is of polynomial complexity.

D) If the no. of nodes generated is 2^n, then the worst case complexity for a backtracking algorithm is $O(P(n)2^n)$ where $P(n)$ is a polynomial in 'n'.

Scanned with CamScanner

Fractional Knapsack (Array V, Array W, int W)

```
1) for i=1 to Size(v)
2)     do P[i] = v[i]/w[i]
3) Sort-Descending (P)
4) i ← 1
5) while (W>0)
6)     do amount = min (W, w[i])
7)     solution [i] = amount
8)     W = W - amount
9)     i ← i+1
10 return solution.
```

Q= Consider 5 items along their respective weights and values.

$I = (I_1, I_2, I_3, I_4, I_5)$

$w = (5, 10, 20, 30, 40)$

$v = (30, 20, 100, 90, 160)$

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

Sol^n ⇒

Scanned with CamScanner

→ Initially,

Item	Wi (kg) weight	Vi (Price)
I_1	5	30
I_2	10	20
I_3	20	100
I_4	30	90
I_5	40	160

Taking value per weight ratio i.e. $P_i = v_i/w_i$

Item	Wi	Vi	$P_i = V_i/W_i$
I_1	5	30	6.0
I_2	10	20	2.0
I_3	20	100	5.0
I_4	30	90	3.0
I_5	40	160	4.0

Now arrange the value of 'P_i' in descreasing order.

Item	Wi	Vi	$P_i = V_i/W_i$
I_1	5	30	6.0
I_3	20	100	5.0
I_5	40	160	4.0
I_6	30	90	3.0
I_2	10	20	2.0

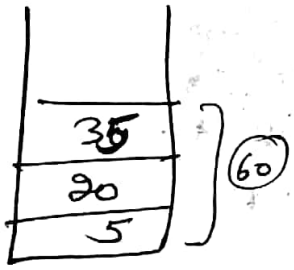
Now, fill the Knapsack acc. to the descreasing value of P_i.

first, we choose item I_1 whose weight is 5, then choose Item I_3 whose weight is 20. Now the total weight

Scanned with CamScanner

in knapsack is $5+20=25$.

Now, the next item is I_5 & its weight is 40, but we want only 35. So we choose fractional part of it i.e.



The value of fractional part of I_5 is

$$\frac{160}{40} \times 35 = 140$$

The max. value is $\Rightarrow 30+100+140$

$= 270$ Rs.

Scanned with CamScanner

Matrix Chain Multiplication

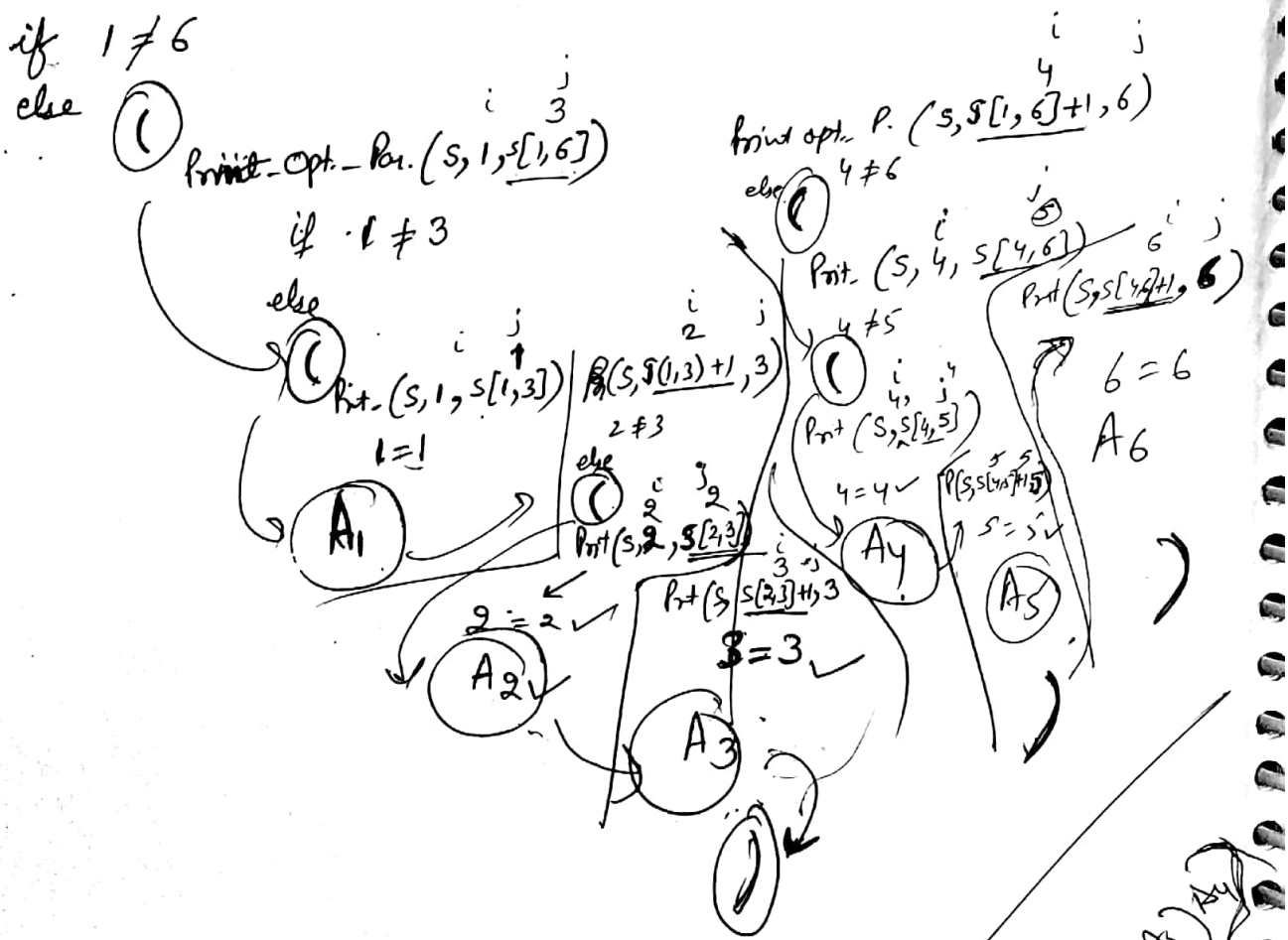
Print_optimal_Parens(s, i, j)

Print brackets

1) if i = j
2) then print 'Ai'
3) else print '('
4) PRINT_OPTIMAL_PARENS(s, i, s[i,j])
5) PRINT_OPTIMAL_PARENS(s, s[i,j]+1, j)
6) Print ")"

⇒ Example:- ~~(s(1,6))~~

Print_optimal_Parens(s, i=1, j=6)



((A1 (A2 A3)) ((A4 A5) A6))

((A1 (A2 A3)) A4)

Scanned with CamScanner

I	W	Values (V)
1	2	12
2	1	10
3	3	20
4	2	15

W=5

5.2 EIGRP (Enhanced Interior Gateway Routing Protocol)

Commands to configure EIGRP

Router#config

Router(config)#router eigrp<process no>

Router(config-router)#network <net address><wild mask>

Router(config-router)#network <net address><wild mask>

Router(config-router)#exit

5.3 OSPF (Open Shortest Path Factor)

Commands to configure OSPF

Router#config

Router(config)#router ospf<process no> Router(config-router)#network <net address><wild mask> area <area id> Router(config-router)#network <net address><wild mask> area <area id>

Router(config-router)#exit

Wild Mask – Complement of subnet mask

Example

255.255.255.255

- 255.255.192.0 subnet mask

0.0.63.255 wild mask

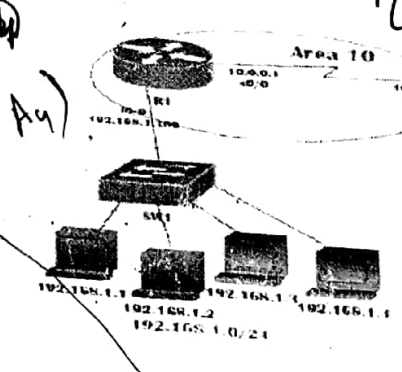


Fig.5.3.1

Scanned with CamScanner

0-1 Knapsack Problem by Dynamic Programing :-

Given n items of

integer weights :- $w_1, w_2 \dots w_n$

values :- $v_1, v_2 \dots v_n$

a knapsack of integer capacity W.

Find most valuable subset of items that fit into the knapsack.

Consider instance defined by first 'i' item and capacity j ($j \leq W$).

let V[i,j] be optimal value of such an instance. Then

$$V[i,j] = \begin{cases} \max\{V[i-1,j],\ v[i] + V[i-1, j-w[i]]\} & \text{, if } j - w[i] \geq 0 \\ V[i-1, j] & \text{, if } j - w[i] < 0 \end{cases}$$

Initial Conditions :-

$$V[0,j] = 0 \text{ and } V[i,0] = 0$$

Scanned with CamScanner

Algorithm:- DPknapsack ($w[1...n], v[1...n], W$)

```
var V[0...n, 0...W], P[1...n, 1...W] : int
for j = 0 to W do
        V[0,j] = 0
for i = 0 to n do
        V[i,0] = 0
for i = 1 to n do
    for j = 1 to W do
        if w[i] ≤ j and v[i] + V[i-1, j-w[i]] > V[i-1, j] then
        {
            V[i,j] = v[i] + V[i-1, j-w[i]]
            // P[i,j] = j - w[i]
        }
        else
            V[i,j] = V[i-1, j]
            // P[i,j] = j
return V[n, W] and the optimal subset by backtracking.
```

Running Time & Space Complexity :-

$O(nW)$

eg:- $V[4,5]$ → last row & last Column gives the max. Profit/Price in Bag.

Scanned with CamScanner

Example:-1) Knapsack of Capacity W=5

	item	(w) weight	(v) value
1	1	2	$12
2	2	1	$10
3	3	3	$20
4	4	2	$15

(Capacity) j →

i ↓	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	(12)	12	12	12
2	0	10	12	(22)	22	22
3	0	10	12	22	(30)	32
4	0	10	15	25	30	(37)

V

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

P

⇒ DPknapsack(w(1...4), v(1...4), 5)

V[0...4, 0...5], P[1...4, 1...5]

for j = 0 — 5 do
- V[0,0] = 0
- V[0,1] = 0
- V[0,2] = 0
- V[0,3] = 0
- V[0,5] = 0

for i = 0 to 4 do
- V[1,0] = 0
- V[2,0] = 0
- V[3,0] = 0
- V[4,0] = 0

for i = 1 to 4 do.

i=1 ⇒ for j = 1 to 5 do

→ j=1

if w[1] ≤ 1
2 ≤ 1 ✗

else
V[1,1] = V[0,1] | P[1,1] = 1
V[1,1] = 0

→ j=2

if w[1] ≤ 2
2 ≤ 2 ✓

then

and v[1] + V[0, 2-w[1]] > V[0,2]
12 + V[0, 2-2] > 0
12 + V[0,0] > 0
12 + 0 > 0
12 > 0 ✓

then
V[1,2] = v[1] + V[0, 2-w[1]]
= 12 + 0
= 12

P[1,2] = 2 - w[1]
= 2 - 2
= 0

Scanned with CamScanner

SHORTCUT;-

Bag / Capacity (j) →

ITEM i ↓	0	1	2	3	4	5
0	0	0	0	0	0	0
$I_1 \to 1$	0	0 (first ITEM NOT Pick ∵ 2kg>1)	12 (I_1 ✓)	12 (3Kg Bag में 2Kg आ सकता है)	12	12
$I_2 \to 2$	0	10 (I_2 ✓)	(12) (I_1 or I_2 ✓ But choose 10<12)	22 (Best I_1 ✓)	22	22
$I_3 \to 3$	0	10	12	22	30	32
$I_4 \to 4$	0	10	15	~~22~~ 25	~~30~~	(37)

(I_1+I_2 ✓ =3Kg)

$I_1 + I_2 + I_4$

$2 + 1 + 2$

$w = 5$

its value ⇒ 12+10+15

= (37)

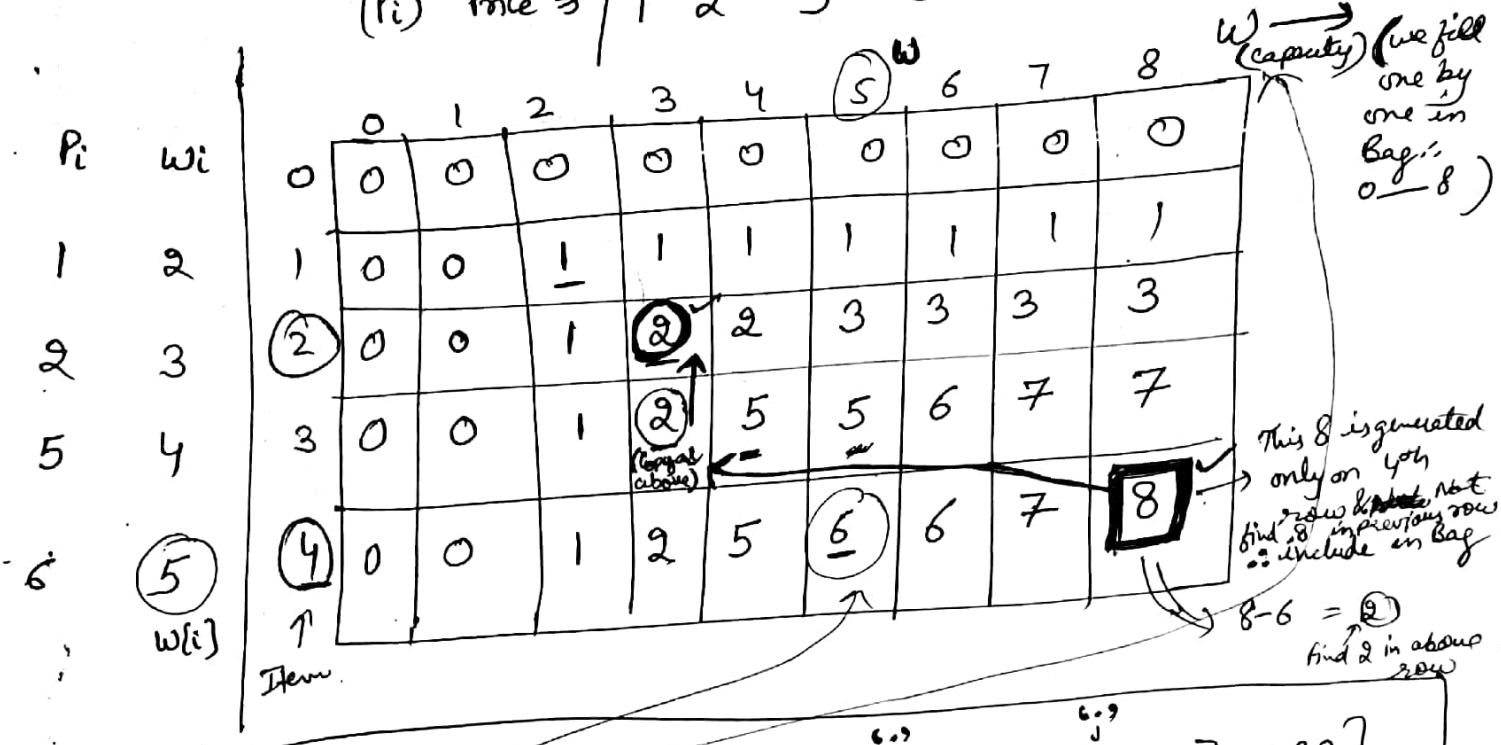
Scanned with CamScanner

Example 2):-

Item ⇒	1	2	3	4
(Wi) weight ⇒	2	3	4	5
(Pi) Price ⇒	1	2	5	6

(Capacity) $W = 8$

$n = 4$ (item)



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

Either fill directly Table or by using this formula

Formula:- $V[i, w] = \max\{V[i-1, w], V[i-1, w-w[i]] + P[i]\}$

Eg:- $V[4,5] = \max\{V[4-1,5], V[4-1, 5-5] + 6\}$

$= \max\{V[3,5], V[3,0] + 6\}$

$= \max\{3, 0+6\}$

$= 6$

Now,

I_1	I_2	I_3	I_4
0	1	0	1

⇒ In 4th row '8' is generated only in this row, that why 4th item must carry in Bag.

Now $8 - 6$ (Price)

$= 2$

Now check 2 is present in above row or Not, If Yes then check the above row also, If above row also same then again check upon... until this sequence break.

$2 - 2$

$= 0$

∴ I_2 & I_4 in Bag

& Price ⇒ (8)

Scanned with CamScanner

BELLMAN FORD :- Find min. distance of every vertex from the source vertex.

Algorithm:-

source vertex, No. of vertex.

Bellmanford(v, cost, dist, n)

```
for i=1 to n do
    dist[i] = ∞ ;
dist[v] = 0        // Initialize source vertex => v=0

for K=1 to n-1 do
for each vertex u ≠ v do
        distnew[u] = dist[u] ;
        for each edge (i,u) do
            distnew[u] = min{distnew[u], dist[i] + cost(i,u)} ;
for each vertex u ≠ v do
        dist[u] = distnew[u];
```

$O(n^3)$

General formula:-

$$dist^{K}[u] = \min\left\{ dist^{K-1}[u] , \min\{dist^{K-1}[i] + cost[i,u]\} \right\}$$

Scanned with CamScanner

Switch#vlan database
Switch(vlan)#vlan <no>
[name <word>]
Switch(vlan)#exit

Commands to configure ports for a Vlan

By default, all ports are member of single vlan that is Vlan1. we can change vlan membership according to our requirement.

Switch#conf ter
Switch(config)#interface <type> <no>
Switch(config-if)#switchport access vlan <no>
Switch(config-if)#exit

Commands to configure multiple ports in a vlan Switch#conf ter

Switch(config)#interface range <type> <slot/port no (space)–(space) port no> Switch(config-if)#switchport access vlan <no> Switch(config-if)#exit

To display mac address table

Switch#sh mac-address-table

Vlan	Mac address	type	ports
20	00-08-a16-ab-6a-7b	dynamic	fa0/7

Scanned with CamScanner

* Traveling Salesman (or Salesperson) Problem using Dynamic Programming:-

Dynamic Programming is that → try out all possible solution and pick up the best one.

→ Traveling problem consists of salesman and a set of cities. Here we have to find out the person who is travelling all the cities & the final ~~that~~ the cost of the cities is Minimum.

→ The salesman has to visit each city starting from Home and returning to the same city (i.e. Home)

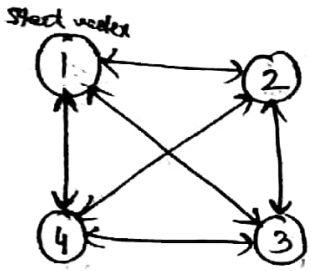
Main challenge :-

→ The person wants to minimize the total length of the trip. (or we say that minimum cost of travelling)

→ If visiting own city → the travelling cost is "Zero"
i.e. starting and ending place ⇒ cost is "Zero"

Scanned with CamScanner

Example:- Weighted Adjacency Graph is Given



$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix} \end{array}$$

Cost of either direction may or may not be same. To avoid many parallel directed uni-directional lines, we use bi-directional edges. Eg:- 1 —10→ 2 , 1 ←5— 2

⇒

Formula:-

$$g(i,s) = \min_{K \in S} \left\{ C_{iK} + g(K, S-\{K\}) \right\}$$

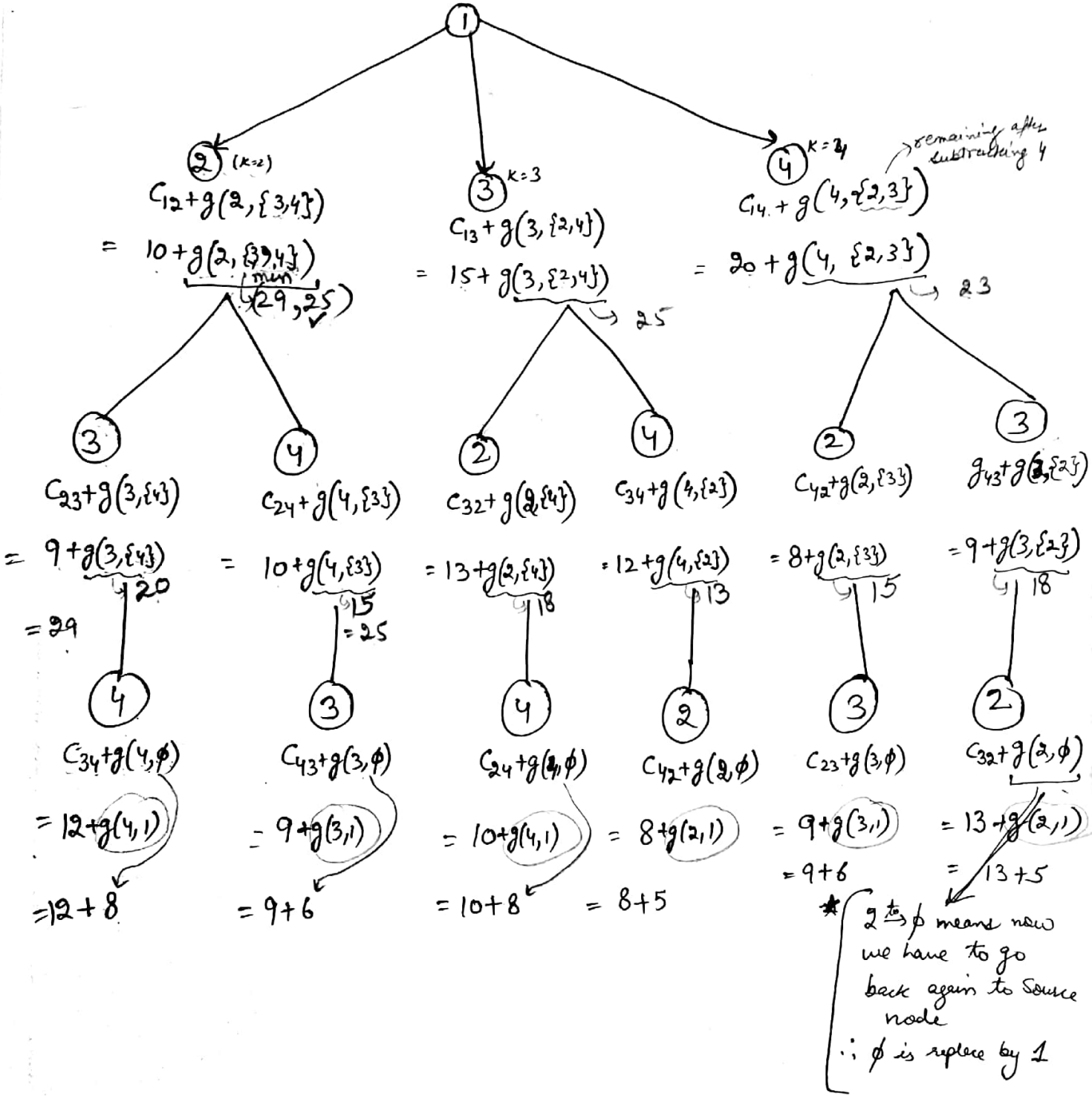
Now Convert General formula into given Graph form:-

$$g(1,\{2,3,4\}) = \min_{K \in \{2,3,4\}} \left\{ C_{1K} + g(K, \{2,3,4\} - \{K\}) \right\}$$

K=2. $C_{12} + g(2,\{3,4\})$.

Scanned with CamScanner

Recursive Tree



Scanned with CamScanner

Last level Nodes

$g(2, \phi) = 5$

$g(3, \phi) = 6$

$g(4, \phi) = 8$

2nd Last level Nodes

$g(2, \{3\}) = 15$

$g\{2, \{4\}) = 18$

$g(3, \{2\}) = 18$

$g(3, \{4\}) = 20$

$g(4, \{2\}) = 13$

$g(4, \{3\}) = 15$

2nd level Nodes

$g(2, \{3,4\}) = 25$

$g(3, \{2,4\}) = 25$

$g(4, \{2,3\}) = 23$

root node

$g(1, \{2,3,4\}) = 35$

So, The shortest route for Travelling Salesperson problem is = 35 Ans.

Scanned with CamScanner

CHANGE MAKING PROBLEM :-

COIN CHANGE PROBLEM :-

(i) find out the ways you can make the change of the given amount using the given coins.

(ii) find minimum no. of coins to get the given value.

Condition :- Infinite supply of coins.

Eg :- Coins $\Rightarrow \{1,2,3\}$ $w = 5$

(1,1,1,1,1) (1,1,1,2) (1,2,2) (1,1,3) (2,3)

Soln (i) $\Rightarrow$ No. of ways $\Rightarrow$ (5)

(ii) $\Rightarrow$ Minimum no. of coin to get given value = (2)

(2,3)

Rule

① Exclude the coin (Previous value)

② Include " " ([6-3]=③ → go to 3rd column within same row & take that value)

③ Add ① & ②

Example :- Coins $\Rightarrow \{2,3,5,10\}$, $w = 15$

(i) No. of ways

[3-3]=0 ⇒ goto 0th column within same row

(i) Coins \ (j) w	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
3	1	0	1	0+1=1	1+0=1	0+1=1	1+1=2	0+1	2	2	2	2	3	2	3	3
5	1	0	1	1	1	2	2	2	3	3	4	4	5	5	6	7
10	1	0	ϕ	1	1	2	2	2	3	3	5	4	6	6	7	7+2=9

1 way to make sum zero & this if we don't consider respective coin

① Exclude the coin ⇒ Pick above value. i.e. 1

② Include " " ⇒ [i,j] = [3,6] ⇒ 6-3 = ③ → go to 3rd column within same row & Pick that value = 1

③ Add ① & ② ⇒ 1+1 = (2)

Ily all

(i) No. of ways = (9) ways.

(ii) Algorithm :- (finding No. of ways)

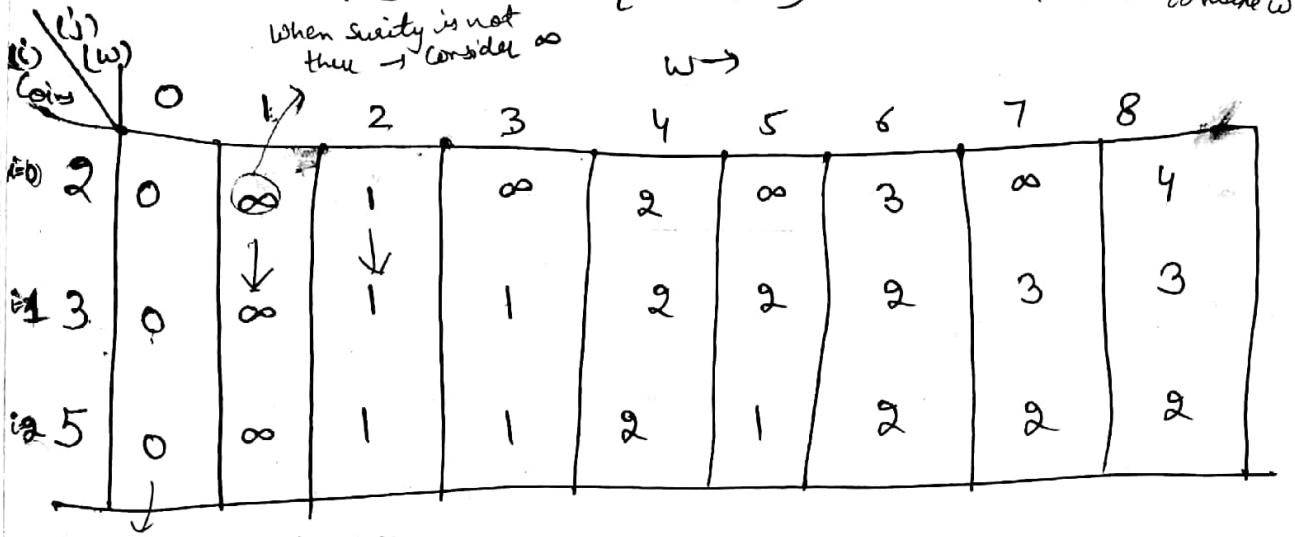
// if coin value > W, then just copy the above value

```
for (i=0; i<=coins.length ; i++)
{
  for (j=0; j<= amount ; j++)
  {
      if (coins[i] > j)
        a[i][j] = a[i-1][j];
      else
      a[i][j] = a[i-1][j] + a[i][j - coins[i]]
  }
}
```

O(nw)

Scanned with CamScanner

(ii) Example:- Coins = {2, 3, 5} $\boxed{w=8}$ (Find Minimum No. of Coins? to make w=8)



Algorithm:- Minimum No. of Coins

// if coins[i] > w just copy above value

```
for (i=0; i<=coins.length, i++)
{
   for j=0 ; j<=w ; j++)
   {
      if (coins[i] > j)
         { a[i][j] = a[i-1][j]
      else
         a[i][j] = min{ a[i-1][j] , 1+a[i][j-coins[i]] }
   }
}
```

$\boxed{O(nw)}$

Q:-

$a[i,j] \Rightarrow a[1,3] \Rightarrow$ $j \le w$

$3 \le 3$ ✓

if coin[1] > j

$3 > 3$ ✗

else $a[1][3] = \min\{ a[1-1][3] , 1 + a[1][3-coin[1]] \}$

$a[0][3] , 1 + a[1][3-3]$

$\infty , 1 + a[1][0]$ → $\min(\infty , 1+0)$ → ① ✓

Scanned with CamScanner

* Longest Common Subsequence :-

```
LCS_LENGTH (X,Y)
1)  m = X.length
2)  n = Y.length
3)  let b[1....m, 1....n] and c[0...m, 0...n] be new tables
4)  for i=1 to m
5)        c[i,0] = 0
6)  for j = 0 to n
7)        c[0,j] = 0
8)  for i=1 to m
9)      for j=1 to n
10)         if xi == yj
11)             c[i,j] = c[i-1,j-1] + 1
12)             b[i,j] = "↖"
13)         else if c[i-1,j] >= c[i,j-1]
14)               c[i,j] = c[i-1,j]
15)               b[i,j] = "↑"
16)           else c[i,j] = c[i,j-1]
17)                b[i,j] = "←"
18) return c and b
```

O(mn)

```
PRINT_LCS(b,X,i,j)        → Y.length
                          → X.length
1)  if i==0 or j==0
2)        return
3)  if b[i,j] == "↖"
4)        PRINT_LCS(b, X, i-1, j-1)
5)        print xi
6)  else if b[i,j] == "↑"
7)        PRINT_LCS(b, X, i-1, j)
8)     else PRINT_LCS(b, X, i, j-1)
```

O(m+n)

Scanned with CamScanner

Example:

$X = A\ B\ C\ B\ D\ A\ B$ ($x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7$)

$Y = B\ D\ C\ A\ B\ A$ ($y_1\ y_2\ y_3\ y_4\ y_5\ y_6$)

1) $m = 7$
2) $n = 6$
3) let $b[7,6]$, $c[7,6]$
4) $i = 1 \text{---} 7$
5) $c[i,0] = 0$
6) $j = 0 \text{---} 6$
7) $c[0,j] = 0$
8) for $i = 1 \text{---} 7$
9) $j = 1 \text{---} 6$

$i=1, j=1 \Rightarrow$

if $x_1 == y_1$

$A \neq B$

else if $c[0,1] \geq c[1,0]$

$0 \geq 0$ ✓

$c[1,1] = c[0,1]$

$= 0$

$b[1,1] =$ "↑"

$j=2 \Rightarrow$

if $x_1 == y_2$

$A \neq D$

else if $c[0,2] \geq c[1,1]$

$0 \geq 0$ ✓

$c[1,2] = c[0,2]$

$= 0$

$b[1,2] =$ "↑"

//by all this...

$y_j \rightarrow$

x_i ↓	0	1 B	2 D	3 C	4 A	5 B	6 A
0	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↑

Print_LCS (b, X, 7, 6)

1) $i \neq 0$ $j \neq 0$ ($7 \neq 0$, $6 \neq 0$)
3) if $b[7,6] \neq$ ↖
else if $b[7,6] ==$ ↑ ✓
PRINT_LCS(b, X, 6, 6)

~~1) 6 ≠ 0 6 ≠ 0~~
~~3) if b[6,6] == '↖'~~

1) $6 \neq 0$ $6 \neq 0$
3) if $b[6,6] ==$ '↖' ✓
Print_LCS(b, X, 5, 5)

$x_6 = A$ — (1)

1) i — x
3) if $b(5,5) \neq$ ~~'↖'~~
else if $b(5,5) ==$ ↑ ✓
PRINT_LCS(b, X, 4, 5)

1) i — x
3) if $b(4,5) ==$ ↖ ✓
PRINT_LCS(b, X, 3, 4)

$x_3 = B$ — (2)

Longest Common Sequence →

BCBA

Scanned with CamScanner

the transport protocol and can be used for carrying many different passenger protocols. The tunnels behave as virtual point-to-point links that have two endpoints identified by the tunnel source and tunnel destination addresses at each endpoint.

It creates vpn on router. VPN tunneling involves establishing and maintaining a logical network connection. It will count on starting and ending addresses and it will not show other addresses through which data passed. It shows only temporary IP not permanent. Though VPN we can access any site if it is block/banned in India. So it will help to change the ip to other countries ip.

COMMANDS AT ROUTER

```
interface Tunnel1
ip address 50.0.0.1 255.0.0.0
mtu 1476
tunnel source Serial0/2/0
tunnel destination 13.0.0.2
interface Serial0/2/0
ip address 10.0.0.1 255.0.0.0
!
interface Serial0/2/1
no ip address
clock rate 2000000
shutdown
!
interface Vlan1
no ip address
shutdown
!
router rip
network 10.0.0.0
Router# traceroute 50.0.0.2
Type escape sequence to abort.
Tracing the route to 50.0.0.2

1 50.0.0.2 12 msec 10 msec 16 msec
Router#
```

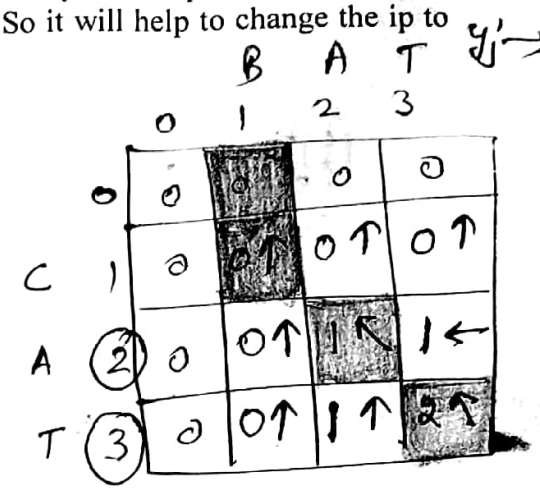
Always check through traceroute because all networks are not going to shown.

COMMANDS AT ANOTHRT ROUETR

```
interface Tunnel2
ip address 50.0.0.2 255.0.0.0
mtu 1476
tunnel source Serial0/0/0
tunnel destination 10.0.0.1
!
!
interface FastEthernet0/0
no ip address
duplex auto
```

X = CAT
Y = BAT
$y_1\ y_2\ y_3$

for i = 1 — 3
for j = 1 — 3



i=1 j=1
if $x_1 == y_1$
C ≠ B
else if c[0,1] ≥ c[1,0]
0 ≥ 0 ✓
c[1,1] = c[0,1] = 0
b[1,1] = ↑

j=2 $x_1 == y_2$
C ≠ A ✗
else if c[0,2] ≥ c[1,1]
0 ≥ 0 ✓
b[1,2] = ↑

j=3 $x_1 \neq y_3$
C ≠ T
elseif c[0,3] ≥ c[1,2]
0 ≥ 0 ✓
c[1,3] = c[0,3] = 0
b[1,3] = ↑

i=2 j=1 if $x_2 == y_1$
A ≠ B
else if c[1,1] ≥ c[1,0]
0 ≥ 0 ✓
c[2,1] = c[1,1] = 0
b[2,1] = ↑

j=2 if $x_2 == y_2$
A = A ✓
c[2,2] = c[1,1] + 1
= 0 + 1 = 1

j=3 if $x_2 == y_3$
A ≠ T
else if c[1,3] ≥ c[2,2]
0 ≥ 1 ✗
else c[2,3] = c[2,2] = 1
b[2,3] = ←

i=3 j=1 $x_3 = y_1$
T ≠ B
else if c[2,1] ≥ c[3,0]
0 ≥ 0 ✓
c[3,1] = c[2,1] = 0
b[3,1] = ↑

j=2 $x_3 = y_2$
T ≠ A
else if c[2,2] ≥ c[3,1]
1 ≥ 0 ✓
c[3,2] = c[2,2]
b[3,2] = ↑

j=3 $x_3 = y_3$
T = T ✓
c[3,3] = c[2,2] + 1
= 1 + 1
= 2
b[3,3] = ↖

LCS ⇒ AT

Scanned with CamScanner