DAA-2

dynamic programming

→ steps of dP algorithm

→ characterise the structure of an optimal solution
→ Recursively define the value of an optimal solution
→ compute the value of an optimal solution in a bottom-up fashion
→ construct an optimal solution from compute information

⇒ using dP method to find the sequence in which following chain of matrixes should be Multiply to minimize the computation time.

step-1 → the structure of an optimal parenthesization.
step-2 → Recursively define the value of an optimal solution

$A_i = A_{(i+1)} \cdots A_j$

$$m[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min\{m[i,k] + m[k+1,j] + p_{i-1} \cdot p_k \cdot p_j\} & \text{if } i<j \\ i \le k < j \end{cases}$$

step-3 → compute the value of an optimal solution
$m[1,n] \rightarrow$ min ~~cost~~ for $A_1 A_2 \cdots A_n$.

step-4 → construct an optimal solution.

$A_1 = (10 \times 20)$
$A_2 = (20 \times 50)$
$A_3 = (50 \times 1)$
$A_4 = (1 \times 100)$

$\Rightarrow A_1 (A_2 (A_3 A_4))$

$\Rightarrow A_3 A_4 = (50 \times 1)(1 \times 100) = 5000$
$= (50 \times 100)$

$A_2 (A_3 A_4) = (20 \times 50)(50 \times 100) = (20 \times 100)$ order
$= 20 \times 50 \times 100$
$= 100000$

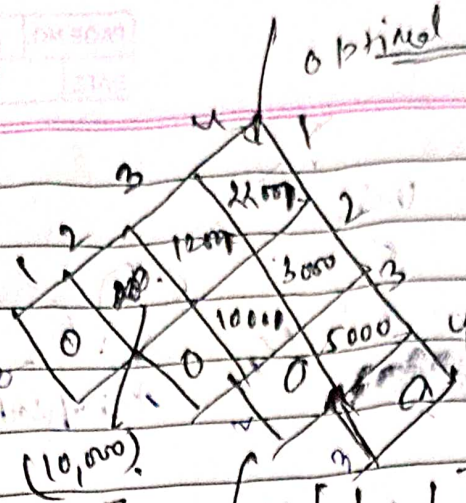
$A_1 (A_2 (A_3 A_4))$
$= (10 \times 20)(20 \times 100)$
$= 10 \times 20 \times 100$
$= 20000$
order $= 10 \times 100$

Total $= 5000 + 10,00,000 + 20,000$
$= 125000.$

$\Rightarrow (A_1 (A_2 A_3)) A_4$
$10 \times 20 \quad (20 \times 1)$
$= (10 \times 1)(1 \times 100) = (10 \times 100)$

$\Rightarrow 20 \times 50 + 10 \times 20 + 10 \times 100$
$= 1000 + 200 + 1000$
$= 2200$

$P_0 = 10, P_1 = 20, P_2 = 50, P_3 = 1, P_4 = 100$



$i \leq k < j$

$M[1,2] = (M[1,1] + M[2,2] + p_0 p_1 p_2)$

$= [0 + 0 + 10 \times 20 \times 50]$

$= 10,000$

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

$(A_1 (A_2 A_3)(A_4))$

$((A_1 A_2)(A_3 A_4))$

Ky bol raha yes Java

item no.	wt	value
1	2	5
2	2	6
3	6	10

Knapsack = 5

		0	1	2	3	4	5
→	0	0	0	0	0	0	0
1		0	(2)	(5)	5	5	5
2		0	0	5	6	6	11
3		0	0	5	6	6	11

or [i-1] [j],
~~or for -1~~
or [i-1] [

3X

(1+2)

1 (2 3)
2 (6,5) ✓
(5) - 3 : (5, 5+6)
(2)

0/1 Knapsack problem
DP Approach

Step 1 - Identify the smaller sub problems. If items are labelled as 1, 2 ... n, then, a subproblem would be to find an optimal solution for S_k.

S_k = { item labelled 1, 2 ... k }

Step 2 - Recursively define the value of an optimal solution in terms of solutions to subproblems.

initial conditions-

$V[0, j] = 0$ for all $j \geq 0$

$V[i, 0] = 0$ for all $i \geq 0$

V is 2-d array denoting value.

Recursive step:

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

(value)

step-3 → Bottom up computation using iteration.

1) The goal is to find V and w, the maximal value of a subset of n given items, that fit into the knapsack of capacity w and an optimal subset itself.

Apply DP Algo to solve the following instance of Knapsack problem with capacity w=5.

Item	weight	value
1	2	3
2	3	4
3	4	5
4	5	6

[0, 1]
3+0

Knapsack w=5 (capacity).

i \ V	0	1	2	3	4	5	
0	0	0	0	0	0	0	optimal value = V[4, 5] = 7
1	0	0	3	3	3	3	
2	0	0	3	4	4	7	
3	0	0	3	4	5	7	
4	0	0	3	4	5	7	

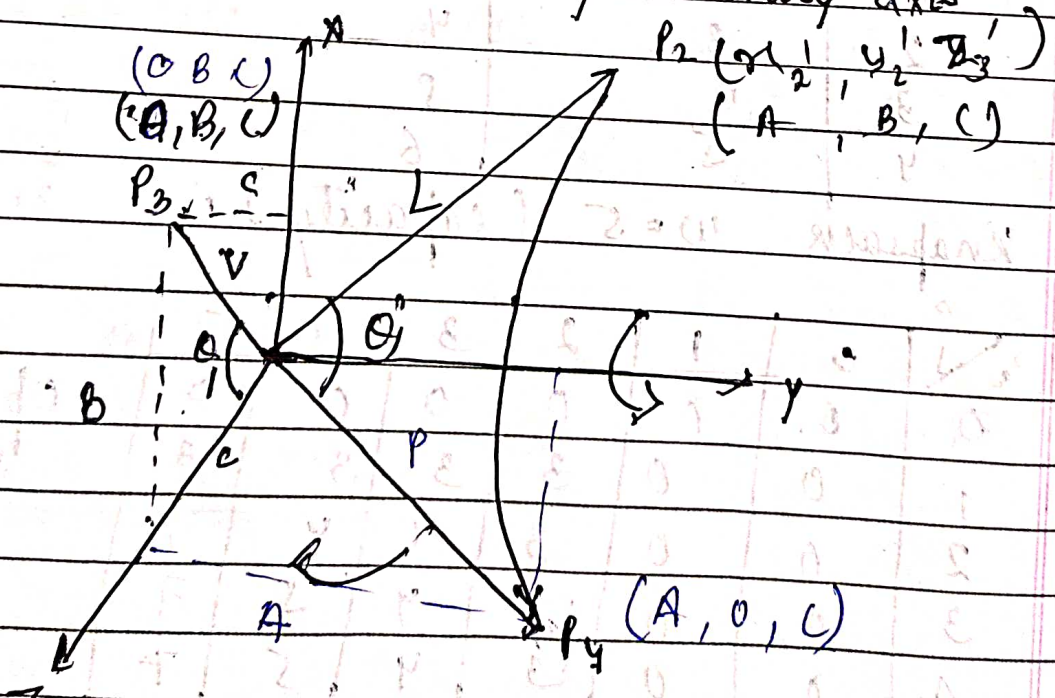
$V[4,5] = V[3,5]$
item 4 not included in list
$V[3,5] = V[2,5]$
item 3 not included in list
$V[2,5] \neq V[1,5]$
so item 2 is included in list.
Remaining capacity of knapsack = 5-3-2 kg
$V[1,2] \neq V[0,2]$
so, item 1 is included in list.
optimal subset = {1, 2} and its value = 7.

Solve the following knapsack problem using DP

# item	weight	value
1	1	1
2	2	6
3	4	4

Q.4 (Row-major order) →

Rotation about any arbitary axis



① Translation

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_1 & -y_1 & -z_1 & 1 \end{bmatrix}$$

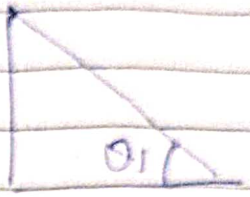
$A = (x_2 - x_1)$

$B = (y_2 - y_1)$

$C = (z_2 - z_1)$

$L = \sqrt{A^2 + B^2 + C^2}$

$V = \sqrt{B^2 + C^2}$



$\cos\theta_1 = \frac{C}{V}$

$\sin\theta_1 = \frac{B}{V}$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

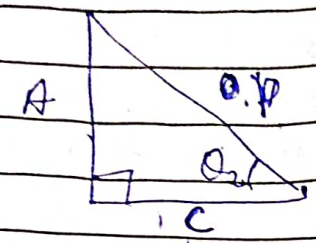
$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about Y-axis.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C/V & B/V & 0 \\ 0 & B/V & C/V & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation θ_2 about y-axis (this rotation is clockwise



$\cos\theta_2 = \frac{BC}{B}$ $\sin\theta_2 = \frac{A}{B}$

(3) $R_y(\theta) = \begin{bmatrix} \cos\theta_2 & 0 & \sin\theta_2 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_2 & 0 & \cos\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$T = T \cdot R_x(\theta_1) \cdot R_y(\theta_2) \cdot R_z(\theta_3) R_y(\theta_2)^{-1} [R_x(\theta_1)]^{-1} \cdot T$

Q. DAA

Solve the following 0/1 knapsack problem using dp.

item	weight	value
1	1	1
2	2	6
3	4	4

Capacity = 3.

i \ j	0	1	2	3	4
0	0	0	0	0	0
1	0	1			
2	0				
3	0				

.1 + V[0][0]

V[1][1] = V[0][1] = 0 1 - 1 ≥ 0 - 2 ≥ 0.
V[1][2] = 2=8 1 - 3 < 0 ✓

Shortest Path Problem

→ All pairs shortest path Problem (floyd's Algorithm).

→

Underlying idea of floyd's algorithm.

→ Let W denote the initial weight matrix.
Let D(K)[i, j] denote cost of shortest path from i to j whose intermediate vertices are a subset of {1, 2, ... K}.

Recursive definiation.

Case-1 - A shortest path from v_i to v_j restricted to using only vertices from $[v_1, v_2 \cdots v_K]$ as intermediate vertices doesn't use v_K. then

$$D(K)[i,j] = D[K-1][i,j].$$

case-2 - A shortest path from v_i to v_j restricted to using only vertices from $[v_1, v_2, \cdots v_K]$ as intermediate vertices do use v_K. then.

$$D(K)[i,j] = D(K-1)[i,K] + D(K-1)[K,j].$$

We conclude

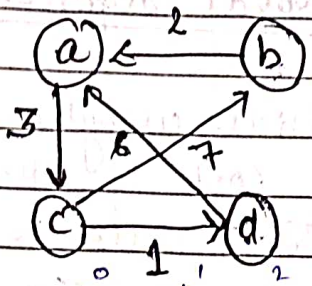
$$D^{(k)}[i,j] = \min\{D^{(k-1)}[i,j],\ D^{(k-1)}[i,k] + D^{(k-1)}[k,j]\}$$

// Algorithm

Input → weight matrix W
output → Distance matrix of shortest path's len

```
D ← W
for k ← 1 to n do
    for i ← 1 to n do
    for j ← 1 to n do
        D[i,j] ← min {D[i,j], D[i,k] + D[k,j]}
return D.
```

Example –



D^0 =

	a	b	c	d
a				
b				
c				
d				

W =

	a	b	c	d
0 a	0	∞	3	∞
1 b	2	0	∞	∞
2 c	∞	7	0	1
3 d	6	∞	∞	0

D^1 =

	a	b	c	d
a	0			
b				
c				
d				

~~$D[0,1] = \min[D[0,1], D[0,0] + D[0,1]$~~
~~$0 +$~~

Use previous matrix two times two nodes

$D^0 = W$

$D^1 =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	6
c	∞	7	0	1
d	6	∞	9	0

$P^1 =$

a	b	c	d
a	b	a	d
a	b	c	d
a	b	a	d

$D^1(b,c) = \min[D(b,c), D^0(b,a) + D^0(a,c)]$

$= \min(\infty, 5) = 5$

$\min[D^0(b,d), D(b,a) + D^0(a,d)]$

$= \min(\infty, \infty) = \infty$

$\min[D(c,b), D(c,d) + D(d,b)]$

$= \min(7, 1+\infty)$

$= 7$

$D^2 \longrightarrow$

	a	b	c	d
a	0	∞	3	4
b	2	0	5	∞
c	9	7	0	1
d	6	∞	9	0

$P^2 =$

a	b	c	d
a	b	a	d
b	b	c	d
a	b	a	d

$D^3 =$

	a	b	c	d
a	0	10	3	4
b	2	0	5	∞
c	9	7	0	1
d	6	∞	9	0

$P^3 =$

a	c	c	c
a	b	a	d
b	b	c	d
a	b	a	d

$D^4 =$

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	7	7	0	1
d	6	16	9	0

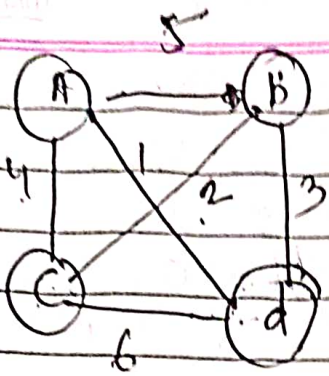
$P^4 =$

a	c	c	c
a	b	a	c
a	b	c	d
a	a	a	d

→ Path $d \to b$

$d \to a \to c \to b$

Q =



undirected graph.

D^0

$W =$

	a	b	c	d
a	0	5	4	1
b	5	0	2	3
c	4	2	0	6
d	1	3	6	0

$P =$

	a	b	c	d
a	a	b	c	d
b	a	b	c	d
c	a	b	c	d
d	a	b	c	d

D^1

$W =$

	a	b	c	d
a	0	5	4	1
b	5	0	2	3
c	4	2	0	(5)
d	1	3	(5)	0

$P =$

	a	b	c	d
a	a	b	c	d
b	a	b	c	d
c	a	b	c	a
d	a	b	a	d

D^2 = $W =$

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

$P =$

	a	b	c	d
a	a	b	c	d
b	a	b	c	d
c	a	b	c	a
d	a	b	a	d

no update

D^3

$W =$

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

$P =$

	a	b	c	d
a	a	b	c	d
b	a	b	c	d
c	a	b	c	a
d	a	b	a	d

no update.

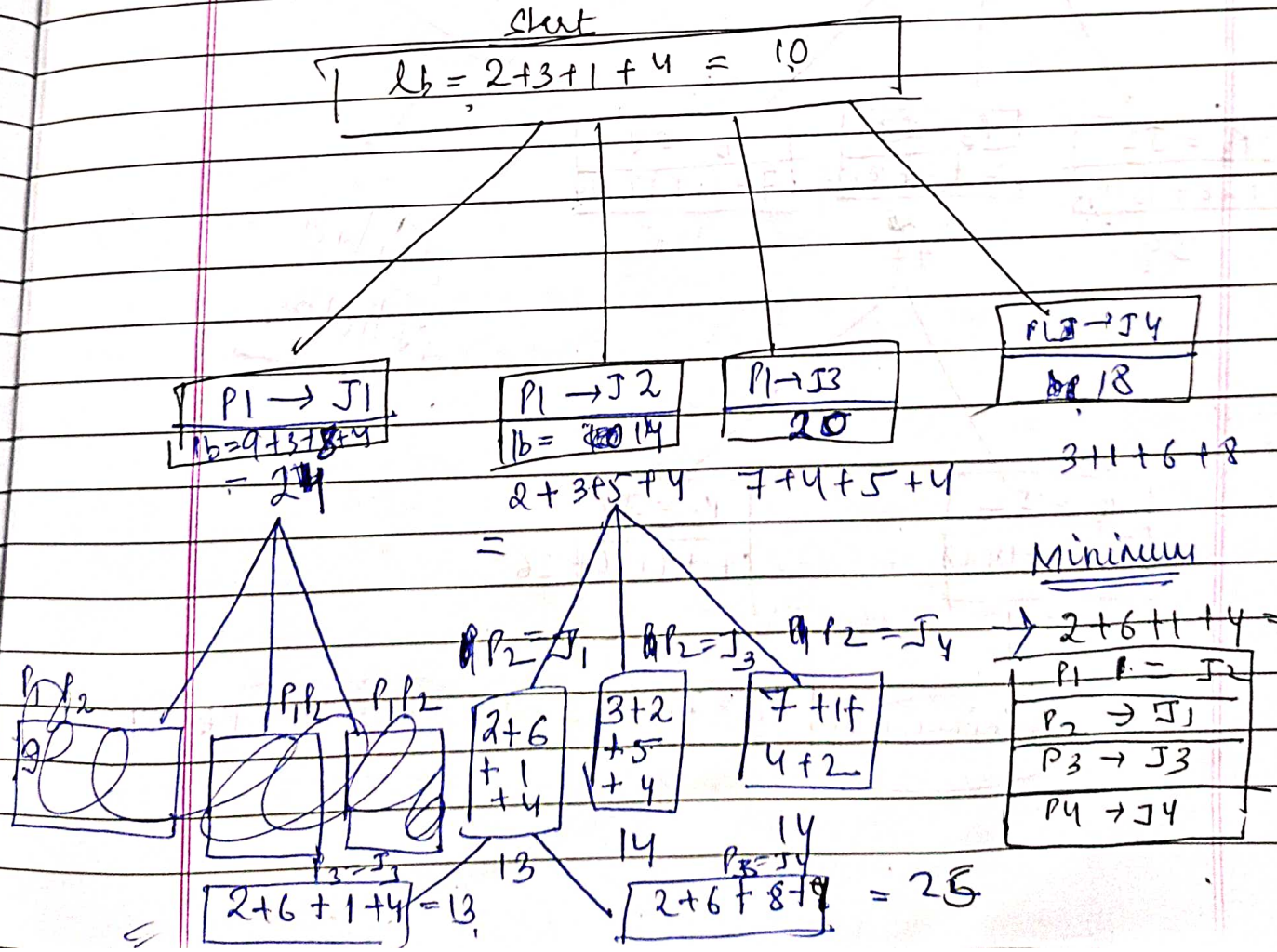
hence we can ~~conclude~~ conclude this is minimum path and weight matrix.

Branch and Bound.

Branch and bound is an algorithm design paradigm which is generally used for solving combinatorial optimization problems.

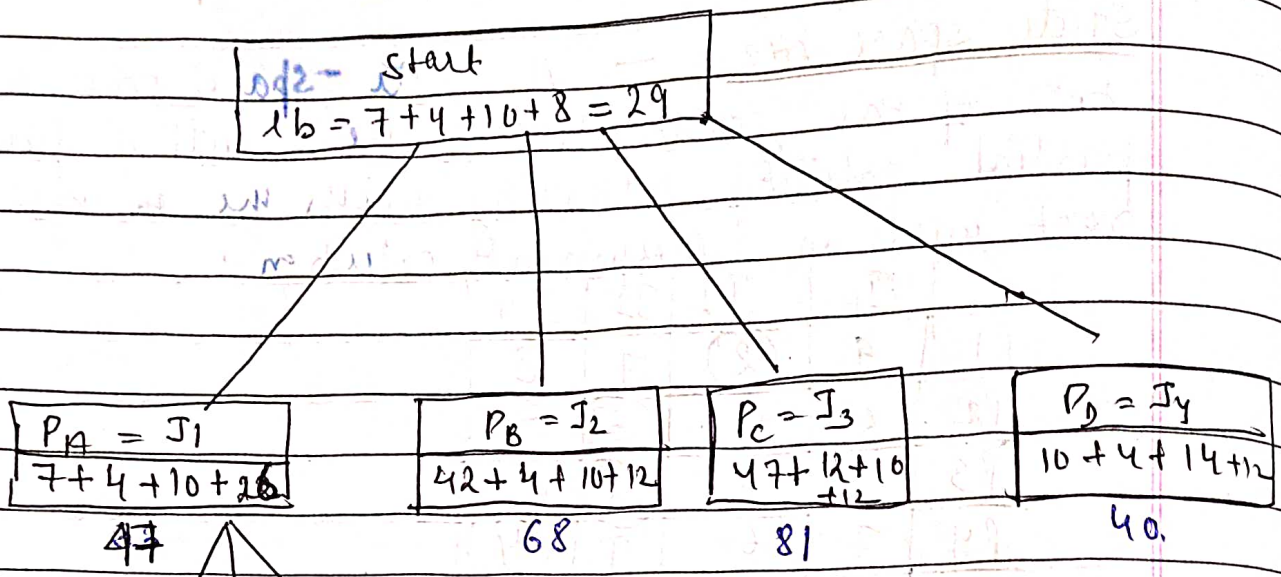
State space tree — A state-space tree is the tree of the construction of the solution from partial solution starting with the root with no component solution.

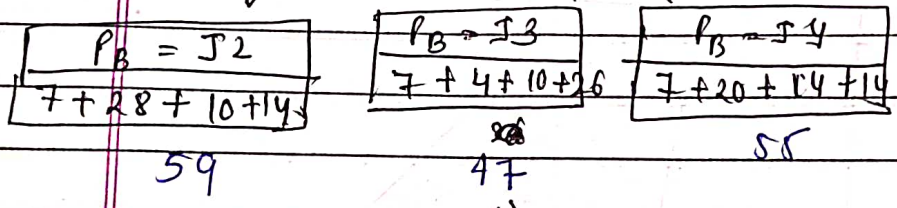
	J_1	J_2	J_3	J_4
P_1	9	(2)	7	8
P_2	6	4	(3)	7
P_3	5	8	(1)	8
P_4	7	6	9	(4)

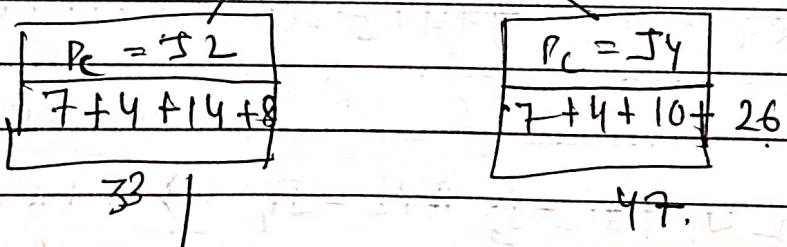


Q

	Job1	Job2	Job3	Job4
PA	7	42	47	10
PB	12	28	4	20
PC	34	14	15	10
PD	12	26	14	8







Minimum

PA = J1 PB = J3 PC = J2 PD = J4

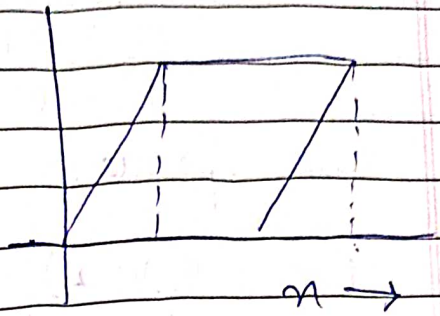
Tuesday
17/10/22

Shear Transformation

Here $y' = y$

$x' = x + S_x \cdot y$

	x'	y'	1
	1	0	0
	S_x	1	0
	0	0	1



* Shear along x-direction.

$x' = x$

$y' = y + SH_y \cdot x$

$z' = z + SH_z \cdot x$

	x'	y'	z'
x	1	SH_y	SH_z
y	0	1	0
z	0	0	1
1	0	0	0

* shear along y-direction

$y' = y$

$x' = x + SH_x \cdot y$

$z' = z + SH_z \cdot y$

* shear along z-direction.

$z' = z$

$x' = x + SH_x \cdot z$

$y' = y + SH_y \cdot z$

Q perform shearing transformation in OABCDEFG cuboid and shearing along z-direction $S_x = 2$, $S_y = 3$

O → origin.

A (0, 0, 4) B(0, 4, 2) C(2, 4, 0) d(2, 2, 4)

E (2, 0, 0) f (0, 0, 2) G(2, 0, 2).

A = (0, 0, 4)

$x' = 0 + 2 \times 4 = 8$

$y' = 0 + 3 \times 4 = 12$

new point = (8, 12, 4)

B = (0, 4, 2)

$x' = 0 + 2 \times 2 = 4$

$y' = 4 + 3 \times 2 = 10.$

New point = (4, 10, 2)

C = (2, 4, 0) new point = (2, 4, 0)

can easily be solve using matrix multiplication

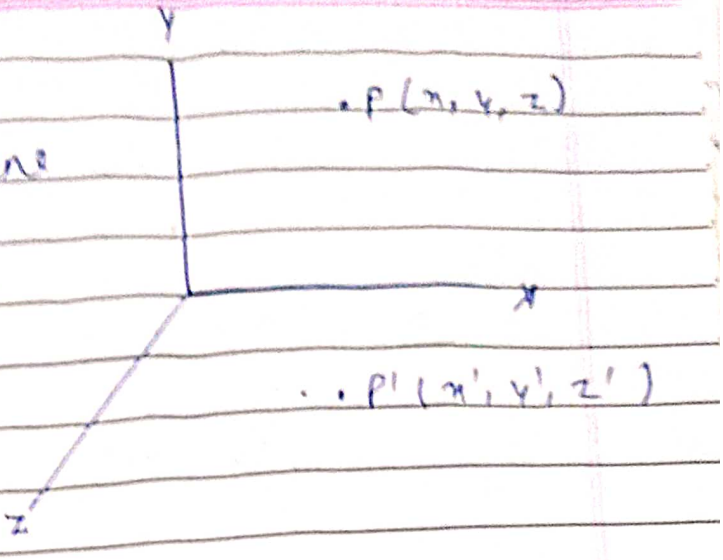
$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 4 & 1 \\ 0 & 4 & 2 & 1 \\ 2 & 4 & 0 & 1 \\ 2 & 2 & 4 & 1 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 2 & 1 \\ 2 & 0 & 2 & 1 \end{bmatrix}_{8 \times 4} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 2 & 3 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_{4 \times 4}$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 8 & 12 & 4 & 1 \\ 4 & 10 & 2 & 1 \\ 2 & 4 & 0 & 1 \\ 2 & 6 & 4 & 1 \\ 2 & 0 & 0 & 1 \\ 4 & 6 & 2 & 1 \\ 6 & 6 & 2 & 1 \end{bmatrix} \rightarrow \text{new points}$$

Reflection in 3D.

Reflection about x z plane

$x' = x, \; y' = -y, \; z' = z$



	x'	y'	z'	1
x	1	0	0	0
y	0	-1	0	0
z	0	0	1	0
1	0	0	0	1

DAA — Branch and Bound

② 0/1 Knapsack problem-

→ step-1

item	weight	value	value/weight ratio
1.	4	40	10
2.	7	42	6
3.	5	25	5
4	3	~~10~~ 12	~~3~~ 4

Knapsack capacity = 10.

→ step-2 arrange item in decreasing order. (according to weight.)

→ step-3 upperbound = $v + (W - w)(v_{i+1} / w_{i+1})$

↳ capacity of knapsack.

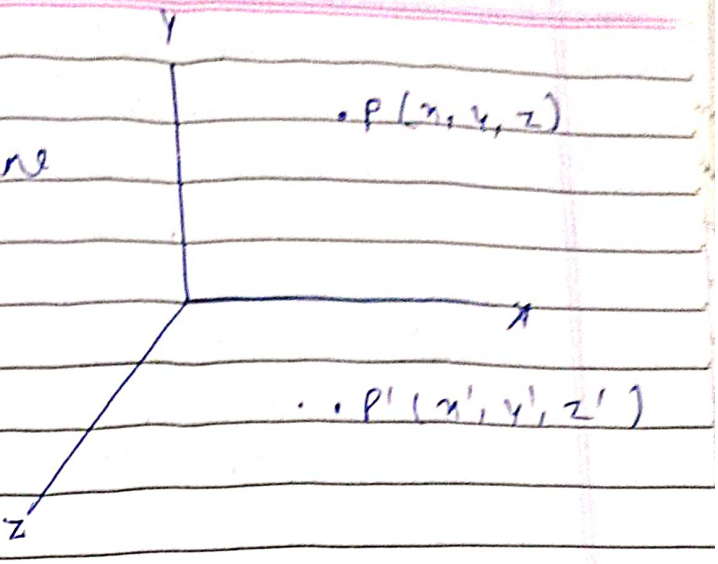
Now search path for optimal solution in state space tree.

Start node = $0 + (10 - 0)\ 10 = 100$

Reflection in 3D.

Reflection about x-z plane

$x' = x$, $y' = -y$, $z' = z$



	X'	Y'	Z'	1
X	1	0	0	0
Y	0	-1	0	0
Z	0	0	1	0
1	0	0	0	1

DAA — Branch and Bound

(2) 0/1 knapsack problem-

→ step-1

item	weight	value	value/weight ratio
1.	4	40	10
2.	7	42	6
3.	5	25	5
4	3	~~0~~ 12	~~5~~ 4

knapsack capacity = 10.

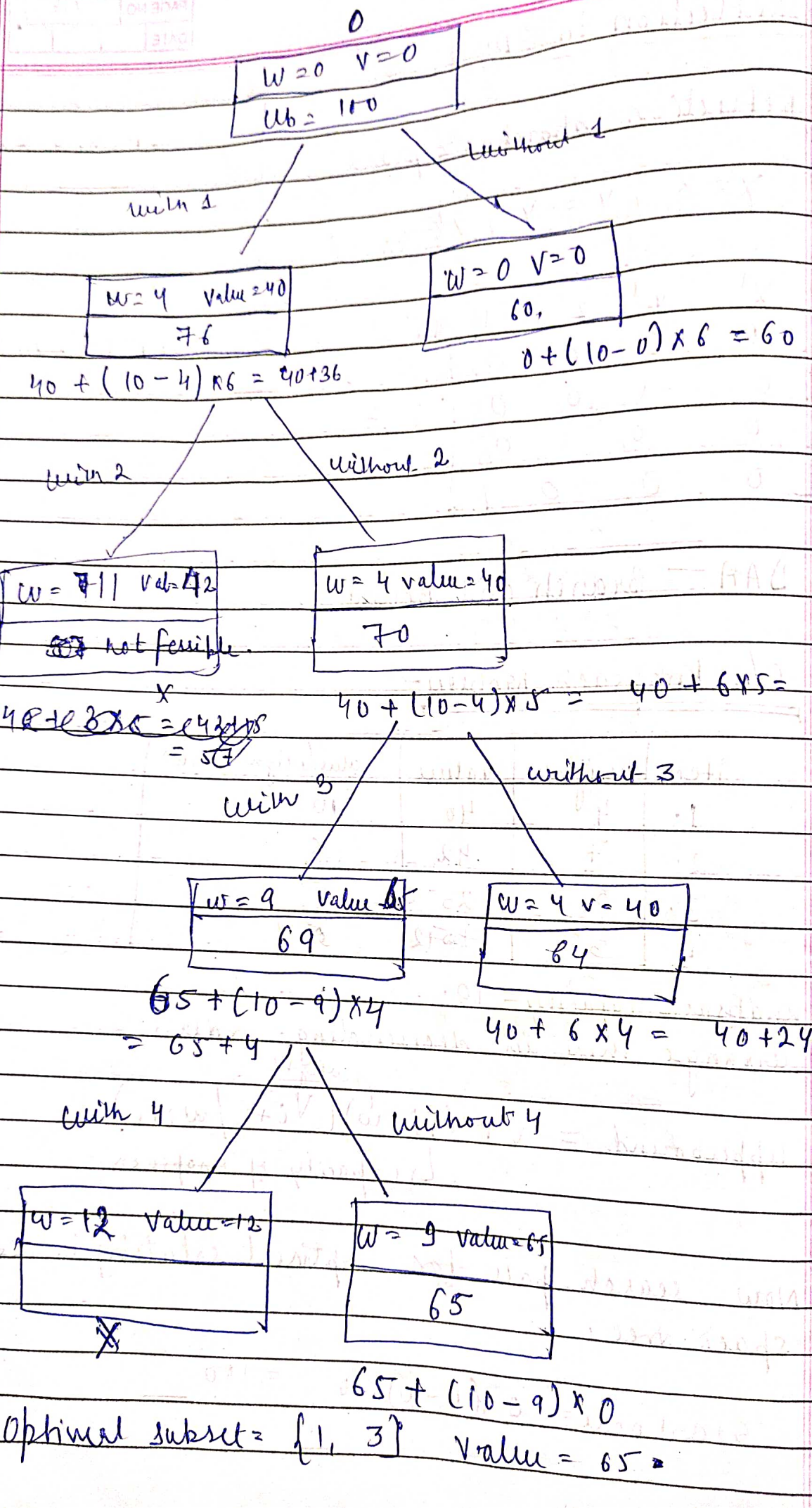
→ step-2 arrange them in decreasing order.

→ step-3 upperbound = $v + (W - w)(v_{i+1} / w_{i+1})$

(w: weight; W: capacity of knapsack.)

Now search path for optimal solution in state space tree.

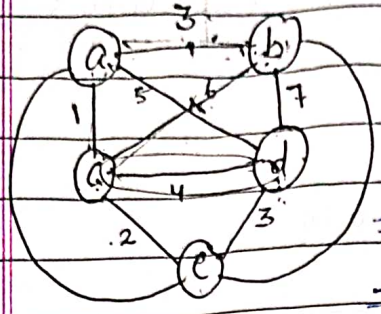
Start node = $0 + (10 - 0)\ 10 = 100$



Optimal subset = {1, 3} Value = 65

$\{(3+1)+(3+6)+(6+4)+(4+3)+(2+3)\}$

Travelling salesman problem.



(a)

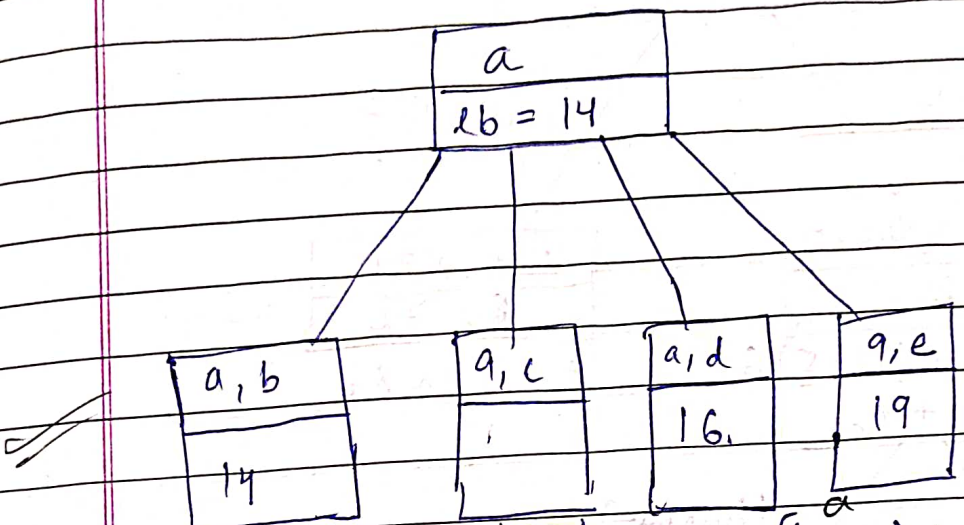
$lb = [s/2]$ where s = sum of two closest neighbour having minimum weight.

start node

$= (1+3)/2 = 2$.

$= [(1+3)+(3+6)+(1+2)+(3+4)+(2+3)]/2$

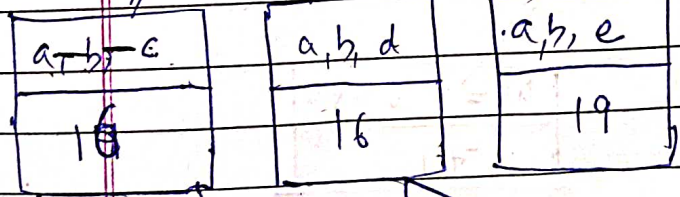
$= \{4+9+3+7+5\}/2 = 14$



$lb = \{(3+1)+(3+6)+(1+2)+(3+4)+(2+3)\}/2$ b is not before c.

$= [(1+5)+(3+6)+(5+3)+(1+2)+(2+3)]/2$

$= (6+9+8+3+5) = 31/2 = 15.5 \simeq 16.$



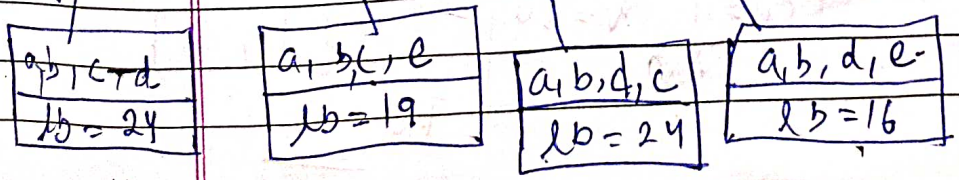
$\rightarrow [[3+1]+[3+6]+[1+6]+[4+3]+[2+3]] = 32/2.$

$\rightarrow [[3+1]+[3+7]+[1+4]+[7+3]+[2+3]] = 32/2$

$\rightarrow$

Sol = a → b → d → e → c → a

3 + 7 + 3 + 2 + 1 = 16

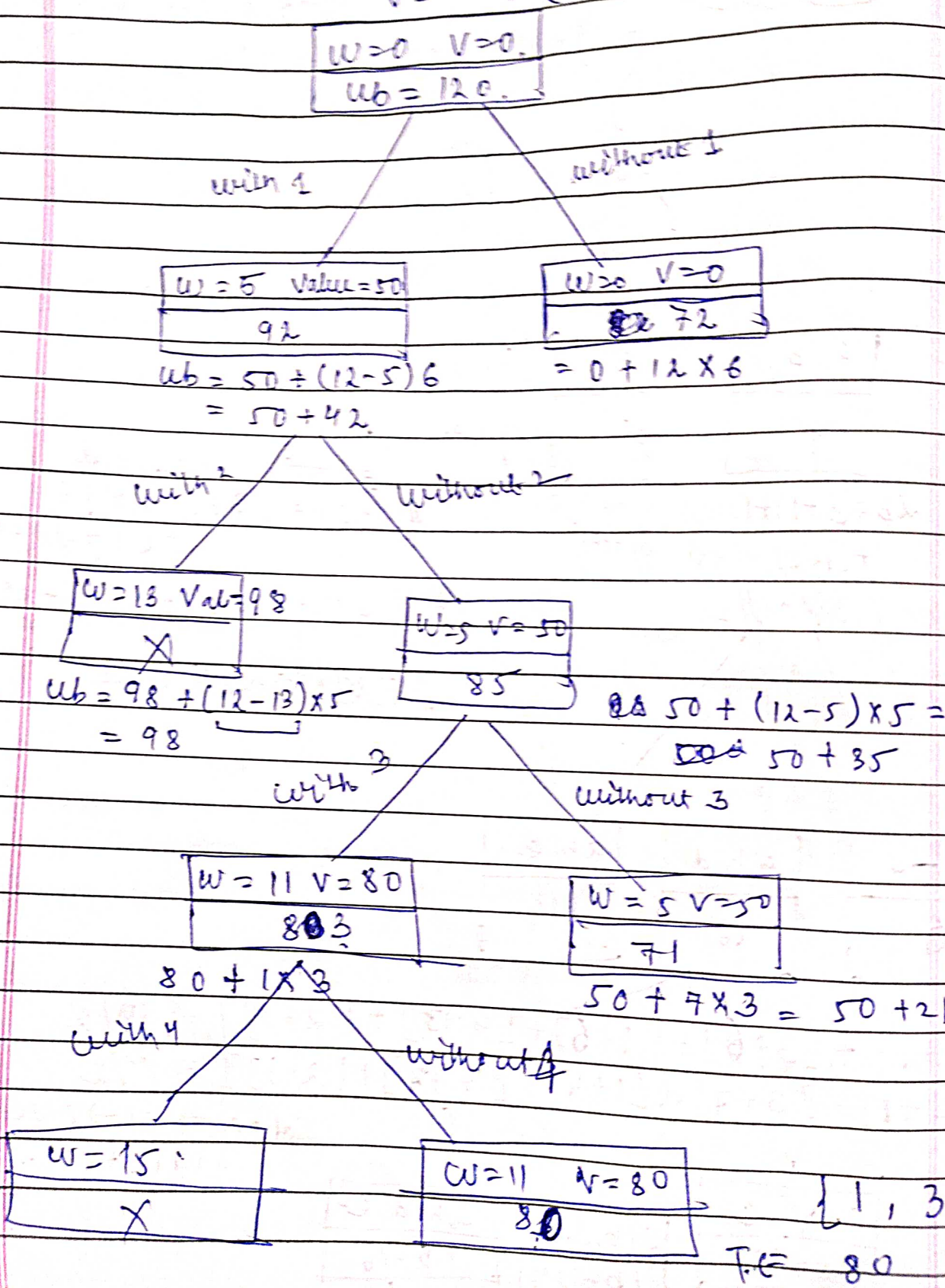


Q

item	v_i	w_i	v_i/w_i
1	50	5	10
2	48	8	6
3	30	6	5
4	12	4	3

Capacity = 12

$V = 0 + (12 - 0)(10)$



Travelling salesman problem (TSP)

	1	2	3	4	5
1	∞	17	7	35	18
2	9	∞	5	14	19
3	29	24	∞	30	12
4	27	21	25	∞	48
5	15	16	28	18	∞

$1 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow 2 \rightarrow 1$ T.C = 67.

DAA

Computer Graphics. Backtracking

N Queen problem.

N = 4 4 Queen problem.

Sum of subset problem:

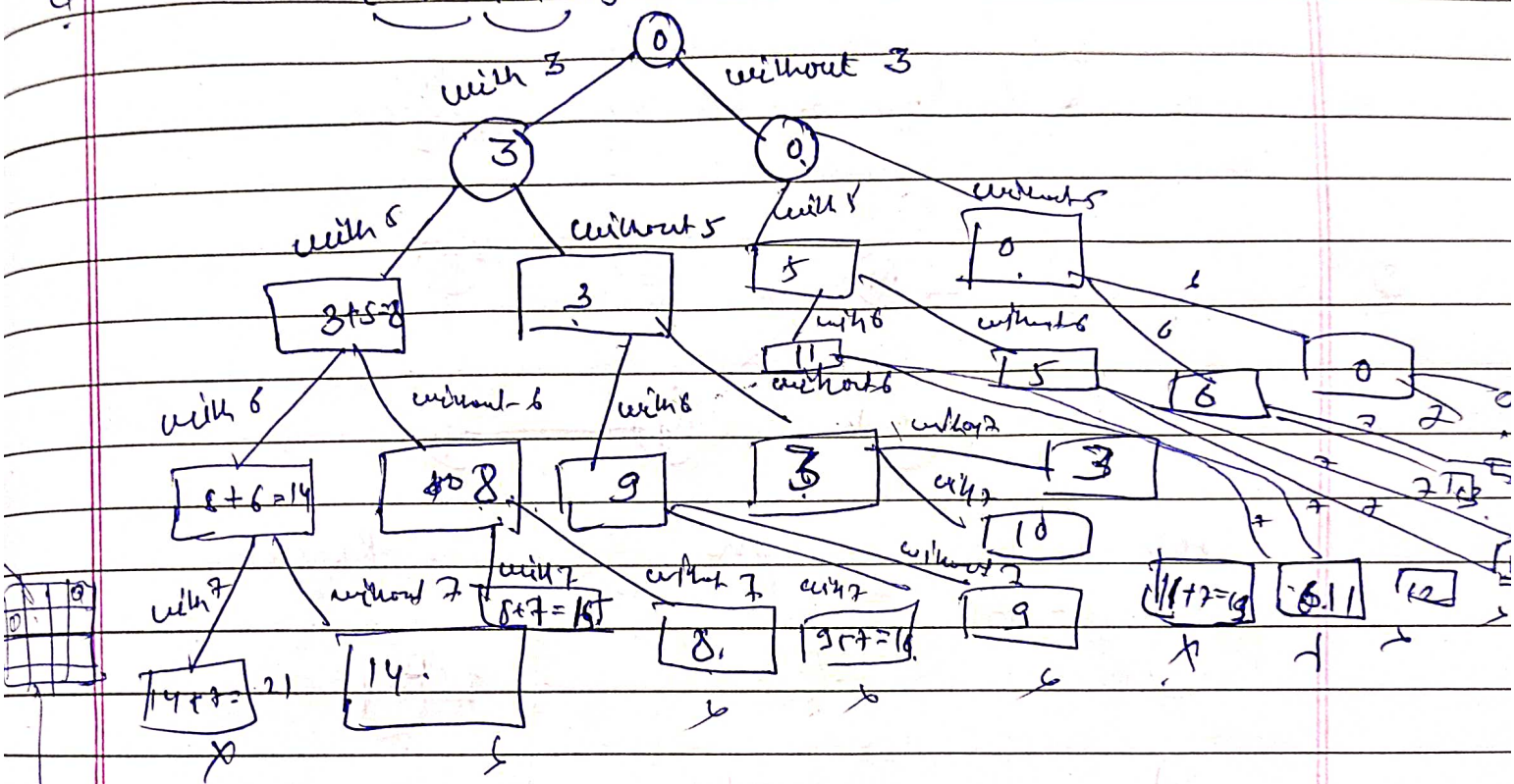
$A = \{a_1, a_2 \ldots a_n\}$ of n +ve integers. whose sum is equal to a given +ve integer d.

$A = \{1, 2, 5, 6, 8\}$

$d = 9$

Subsets are $\{1, 2, 6\}$ and $\{1, 8\}$.

Q. $A = \{3, 5, 6, 7\}$ $d = 15$



(1) → ~~8+7=15~~

3+5+7=15

Decrease & conquer

a) Decrease by constant (1)

n
(n-1) → factorial
(n-1)

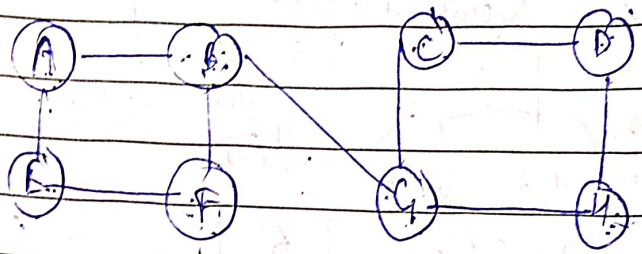
b) Decrease by a constant factor n/2

(1) DFS (Depth first search)

$G = (V, E)$

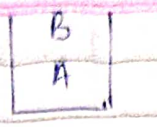
V

Q.

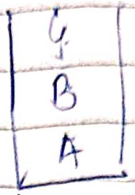


Step	Graph	Remarks
1	A	A
2	A — B	B, A
3	A — B, B — F	F, B, A
4	A — B, B — F, E — F	E, F, B, A

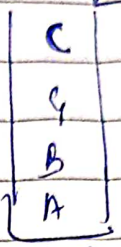
Backtrack



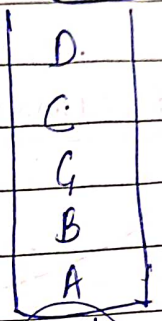
(5)



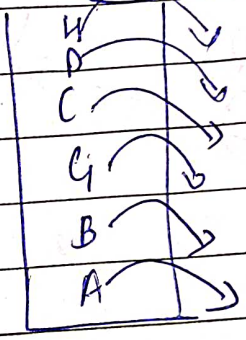
(6)



(7)

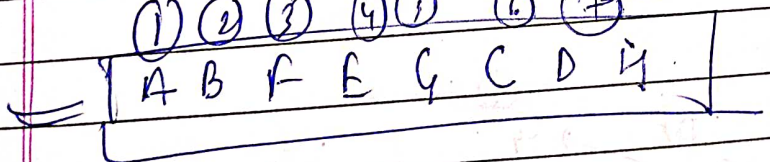


(8)



Stack is empty

(1) (2) (3) (4) (5) (6) (7)



~~A(1, 8) B(2, 7) C(6, 5) D(7, 4) E(4, 1) F(3, 2)~~

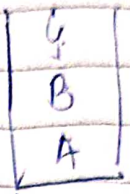
A(1, 8) B(2, 7) C(6, 5) D(7, 4) E(4, 1) F(3, 2)
G(5, 6) H(8, 3)

↑
POP at
which position.

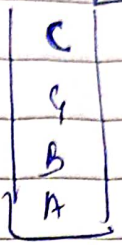
T.C → Using Adjacent Matrix $T(n) = O(n^2)$
Adjacent list $T(n) = O(n+E)$.

Backtrack

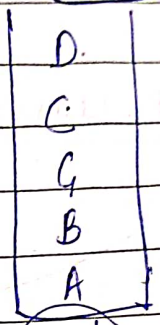
(5)



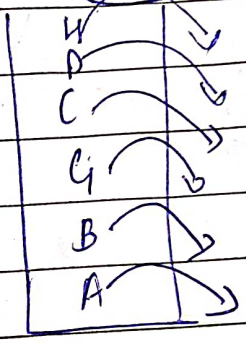
(6)



(7)

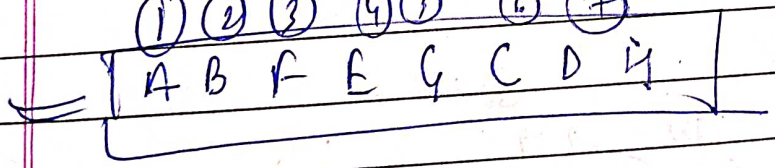


(8)



Stack is empty

(1) (2) (3) (4) (5) (6) (7)



~~A(0, 8) (2~~

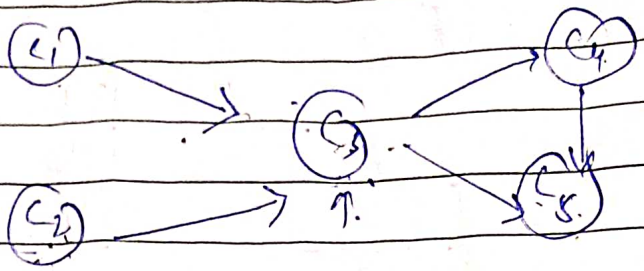
A(1, 8) B(2, 7) C(6, 5) D(7, 4) E(4, 1) F(3, 2)
G(5, 6) H(8, 3)

↑
Pop at
which position.

T.C → Using Adjacent Matrix $T(n) = O(n^2)$
Adjacent list $T(n) = O(n+E)$.

PAGE NO.
DATE

Topological sorting { ~~unique~~ Acyclic, directed graph }

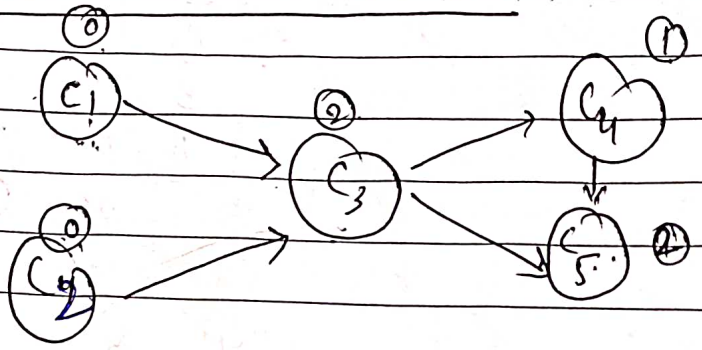


→ C1 C2 C3 C4 C5

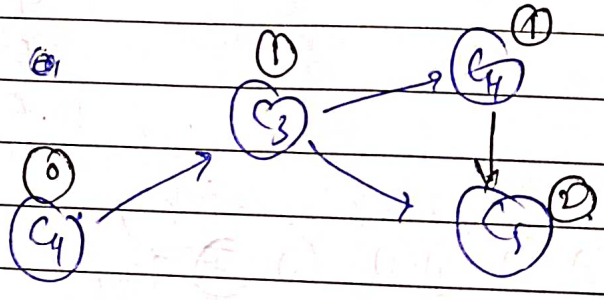
Methods

→ DFS X

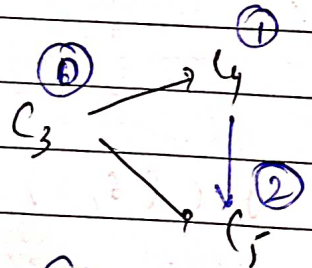
→ Source Removal Method.



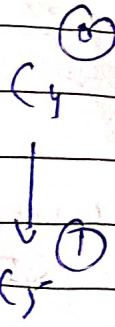
visit → C4



visit C3



visit C4



Vist → c_4 c_5

Mid → c_5

→ c_1 c_2 c_3 c_4 c_5.

Greedy Technique.

spanning tree → subgraph of a given graph, which contain all vertices of the given graph.

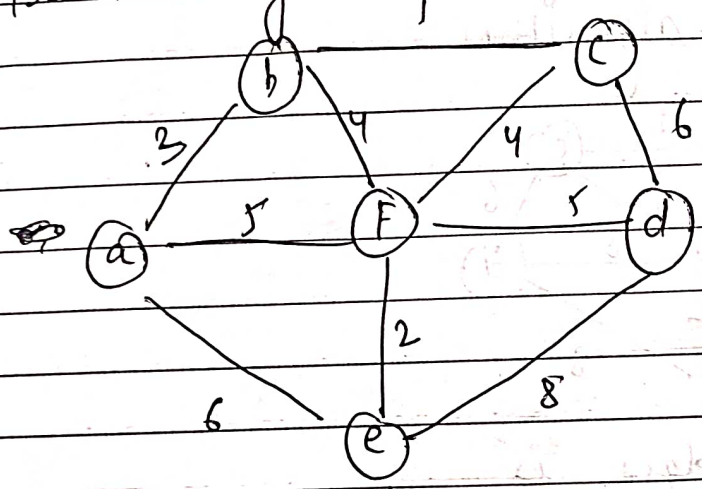
(weighted graph)

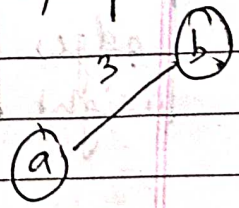
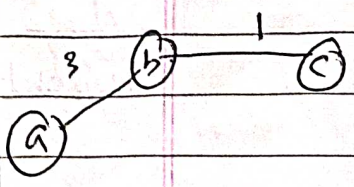
(1) Minimum spanning Tree:

Kruskal's algo

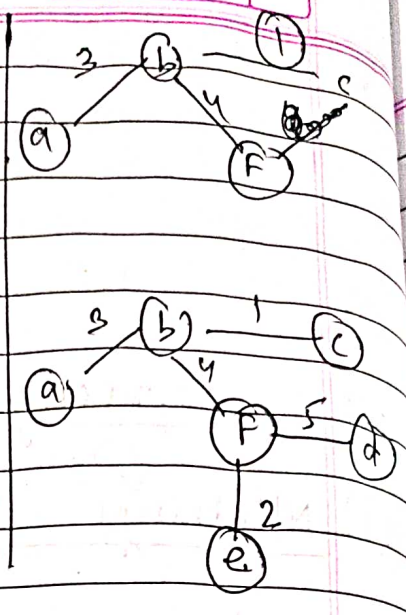
prim's algorithm.

(1) Prim's Algorithm.



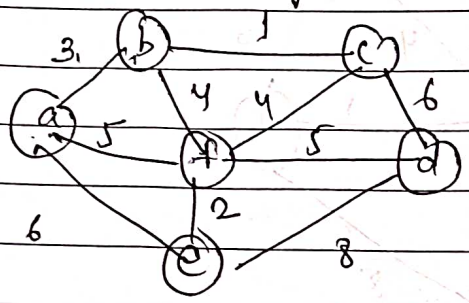
Tree Vertices	Remaining Vertices	Graph
a(-,-)	b(a, 3) c(a, ∞) d(a, ∞) e(a, 6) f(a, 5)	
b(a, 3)	c(b, 1) d(b, ∞) e(a, 6) f(b, 4)	

c(b, 1)	d(c, 6) f(b, 4) e(a, 6)
f(b, 4)	d(f, 5) e(f, 2)
e(f, 2)	d(f, 5)



Minimum cost → 15

(b) Kruskal's algorithm

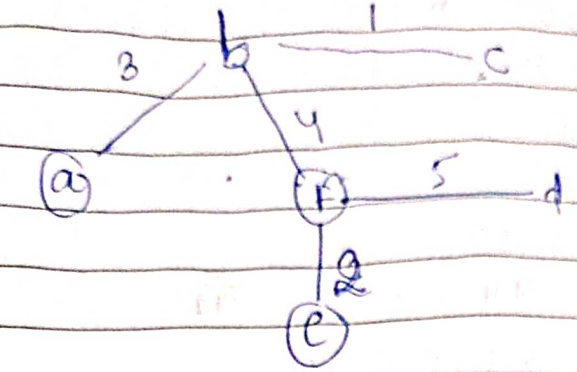


The list of edges is

edges	ab	af	ae	bc	bf	cf	cd	fd	de	ef
weight	3	5	6	1	4	4	6	5	8	2

Sort in ascending order-

edges	bc	ef	ab	bf	cf	af	fd	ae	cd	de
weight	1	2	3	4	4	5	5	6	6	8



weight → 15

T.C → E log E edge based algorithm

→ Prism is faster than Kruskal's because Prism uses ∯ heap to sort.

Prim's Algorithm Analysis → Priority queue

T.C →

array → $O(n^2)$

Binary heap → $O(M \log n)$

→ Fibonacci heap → $(n+M)\log n$.

Fractional knapsack algorithm. {solve by greedy only}

T.C → $O(n \log n)$.

$W = 15$

weight →	2	3	5	7	1	4	1
cost →	10	5	15	7	6	18	3
v/w →	5	1.67	3	1	6	4.5	3

decreasing order sort.

v/w →	6	5	4.5	3	3	1.67	1
we →	1	2	4	5	1	3	7
c →	6	10	18	15	3	5	7

value → 6 + 10 + ... = 55.34.

PAGE NO.
DATE

→ W = 1 V_5 = 6

Update wt = 15 − 1 = 14

value = 6

→ W_1 = 2 V = 10.

up = 14 − 2 = 12

valu = 6 + 10 = 16.

→ W_1 = 4 V = 18.

12 − 4 = 8

valu = 16 + 18 = 34

→ W_1 = 5 V = 15

8 − 5 = 3

= 34 + 15 = 49

→ W_1 = 1 V = 3

3 − 1 = (2) .

= 49 + 3 = 52

→ W_1 = ~~0~~ 3 V = 5

valu = (2) × 5 = 3.34.

3

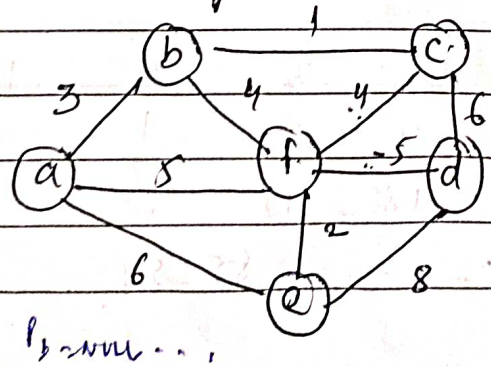
value = 52 + 3.34 = 55.34

Shortest path problem. T.C → O(n²) (array)

min → (M log n)

Dijkstra's Algorithm. {weighted connected graph}

Single source shortest path problem.



negative weight

X

Pa = NULL Pb = NULL . . .

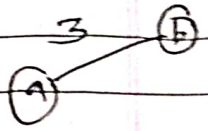
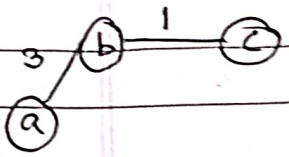
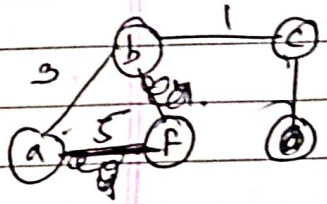
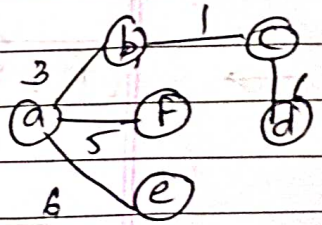
Initially Da = 0 Db = ∞

Dc = ∞ . . .

D(x) Distance from source to x | P(r, x) Path from r to x

Initially $D(x) = w(r, x)$ where rx is an edge
$D(x) = \infty$ where rx is not an edge
for each temporary vertex y different from x

$D(y) = \min\{D(y), D(x) + w(x,y)\}$ set

Tree vertices	Remaining vertices	Distance & Path values		Graph
		$D_a = 0$	$P_a = a$	
a(–, 0)	b(a, 3)	$D_b = 3$	$P_b = [a, b]$	
	c(–, ∞)	$D_c = \infty$	P_c = NULL	
	d(–, ∞)	$D_d = \infty$	P_d = NULL	
	e(a, 6)	$D_e = 6$	$P_e = [a, e]$	
	f(a, 5)	$D_f = 5$	$P_f = [a, f]$	
		$D_a = 0$	$P_a = a$	
b(a, 3)	c(b, 4)	$D_b = 3$	$P_b = [a, b]$	
	f(a, 5)	$D_c = 4$	$P_c = [a, b, c]$	
	e(a, 6)	$D_d = \infty$	P_d = NULL	
	d(–, ∞)	$D_e = 6$	$P_e = [a, e]$	
		$D_f = 5$	$P_f = [a, f]$	
		$D_a = 0$	$P_a = a$	
c(b, 4)	d(c, 10)	$D_b = 3$	$P_b = [a, b]$	
	f(a, 5)	$D_c = 4$	$P_c = [a, b, c]$	
	e(a, 6)	$D_d = 10$	$P_d = [a, b, c, d]$	
		$D_e = 6$	$P_e = [a, e]$	
		$D_f = 5$	$P_f = [a, f]$	
		$D_a = 0$	$P_a = a$	
f(a, 5)	d(c, 10)	$D_b = 3$	$P_b = [a, b]$	
	e(a, 6)	$D_c = 4$	$P_c = [a, b, c]$	
		$D_d = 10$	$P_d = [a, b, c, d]$	
		$D_e = 6$	$P_e = [a, e]$	
		$D_f = 5$	$P_f = [a, f]$	

PAGE NO.
DATE

Huffman code.

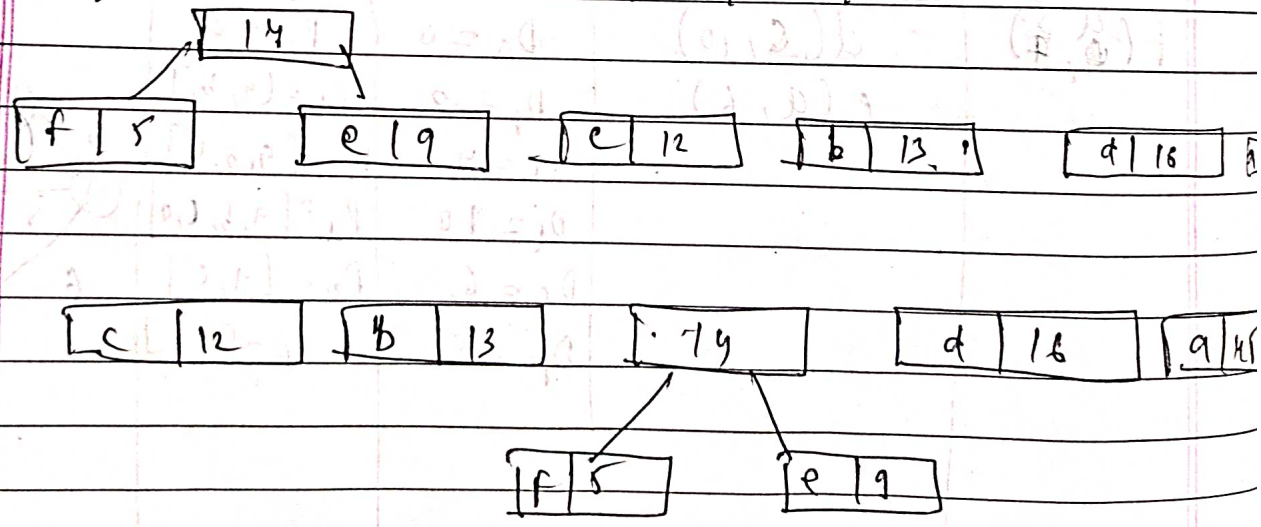
It is technique for data compression, invented by David A Huffman in 1981.
It uses a table of frequencies of occurances of the characters to build of an optimal way of representing each characters as a binary string.

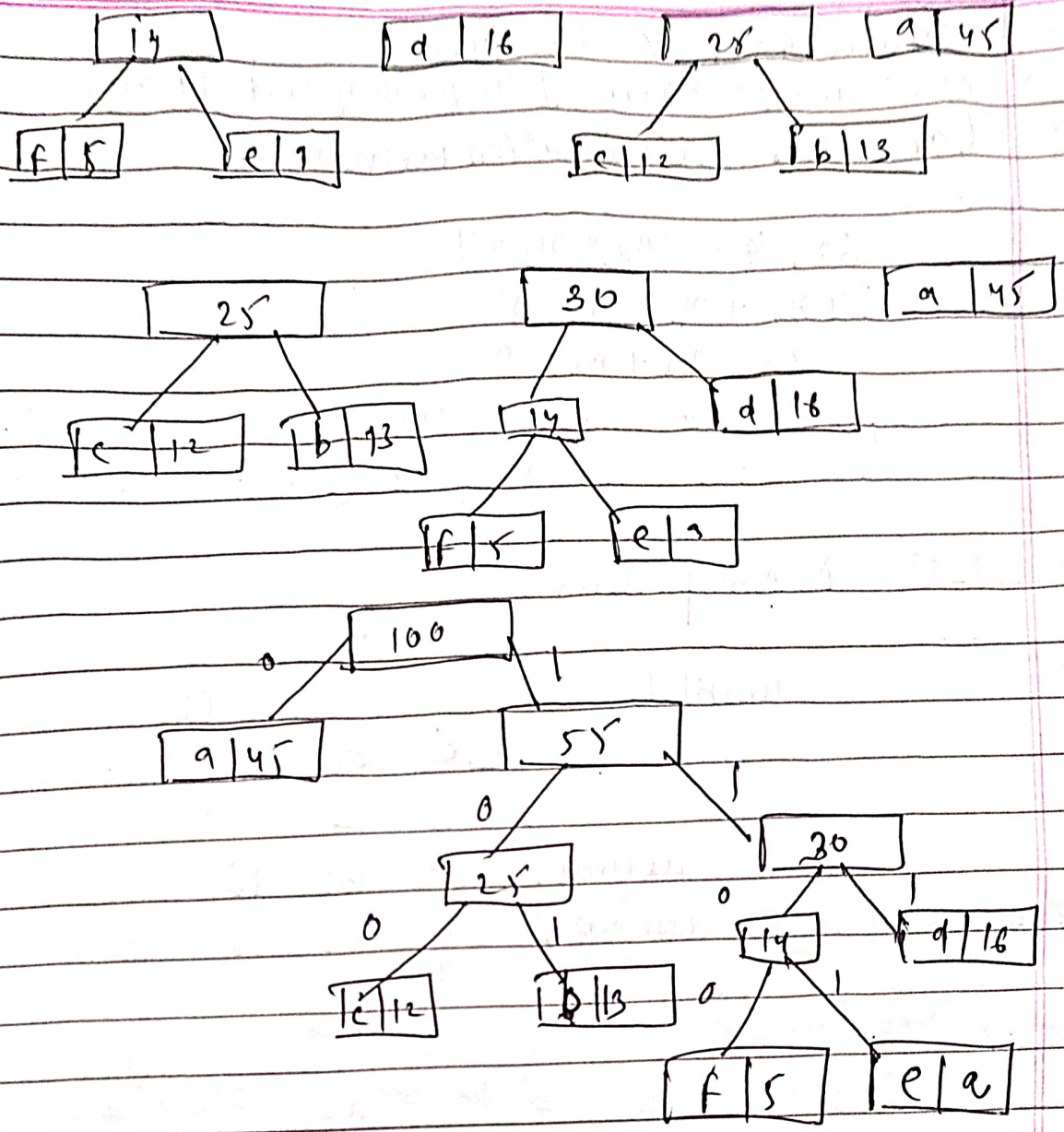
→ It achieve a data compression by finding the best variable length binary encoding scheme for the symbols that occur in the file to be compressed.

← HC for data compression achieve 20-90% compression.

Huffman Tree.

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





a → 0

c → 1 0 0

b → 1 0 1

f → 1 1 0 0

e → 1 1 0 1

d → 1 1 1

Decoding → 1 0 0 1 1 1 0 1 1 0 1

→ c d a e

decoding → c d a e

1 0 0 1 1 1 0 1 1 0 1

T.C — $O(n \log n)$

PAGE NO.
DATE

Transform & Conquer.

→ AVL Tree. / 2-3 Tree / Heap & Heap sort. / B Tree.

① Gaussian elimination method

$$2x_1 + x_2 + x_3 = 1$$
$$4x_1 + x_2 - x_3 = 5$$
$$x_1 + x_2 + x_3 = 0$$

→ Upper triangular matrix

→ solve for x_1, x_2 & x_3.

② LU Decomposition.

$$Ax = B \quad -(1)$$

Substitute $A = LU \quad -(2)$

$$\therefore LUx = B$$

assume $Ly = B$ &

assume $Ux = y$

Q. $L = \begin{bmatrix} 1 & 1 & -1 \\ 1 & -2 & 3 \\ 2 & 3 & 1 \end{bmatrix} \quad \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -6 \\ 7 \end{bmatrix}$

A X B

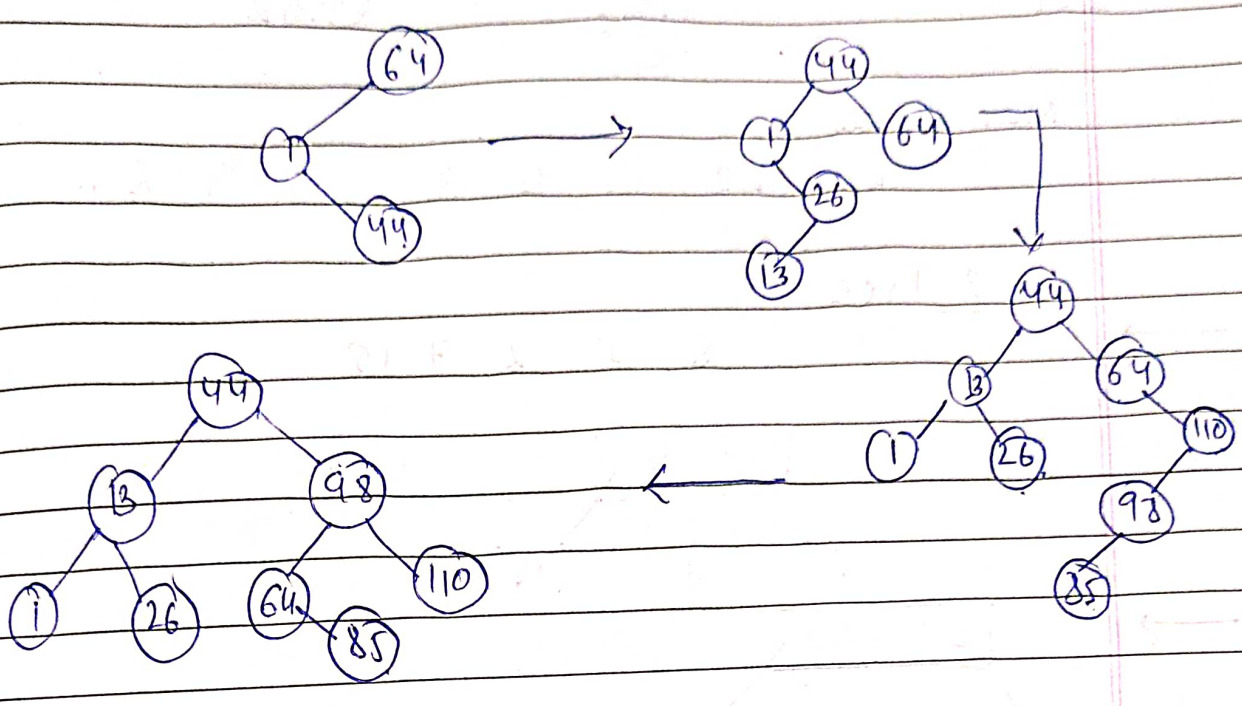
Lower triangular Matrix = $\begin{bmatrix} 3 & 0 & 0 \\ -5 & -4/3 & 0 \\ 2 & 1/3 & 1 \end{bmatrix}$

$Ly = B$

$$\begin{bmatrix} 3 & 0 & 0 \\ -5 & -4/3 & 0 \\ 2 & 1/3 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -6 \\ 7 \end{bmatrix}$$

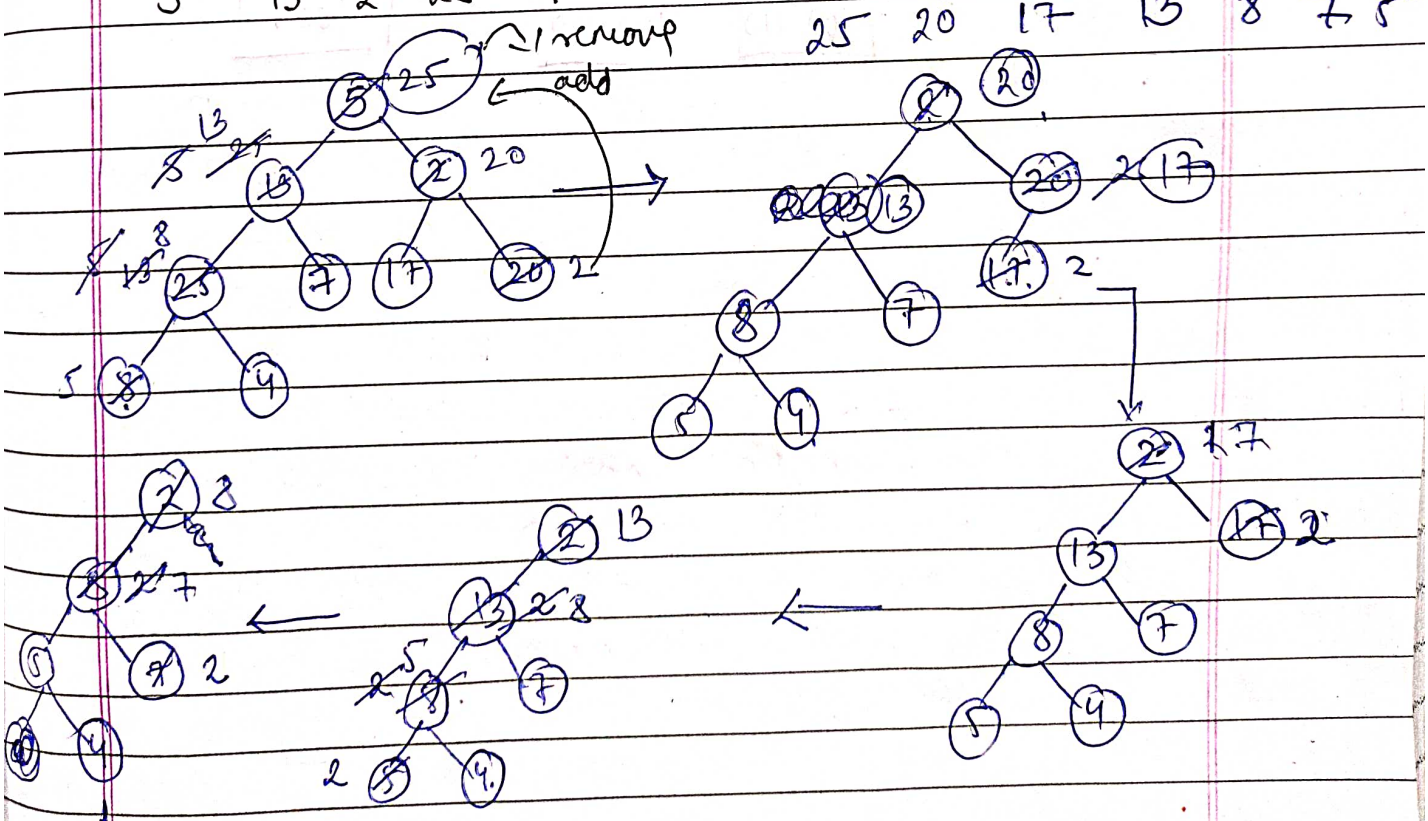
Construct AVL tree from the following data.

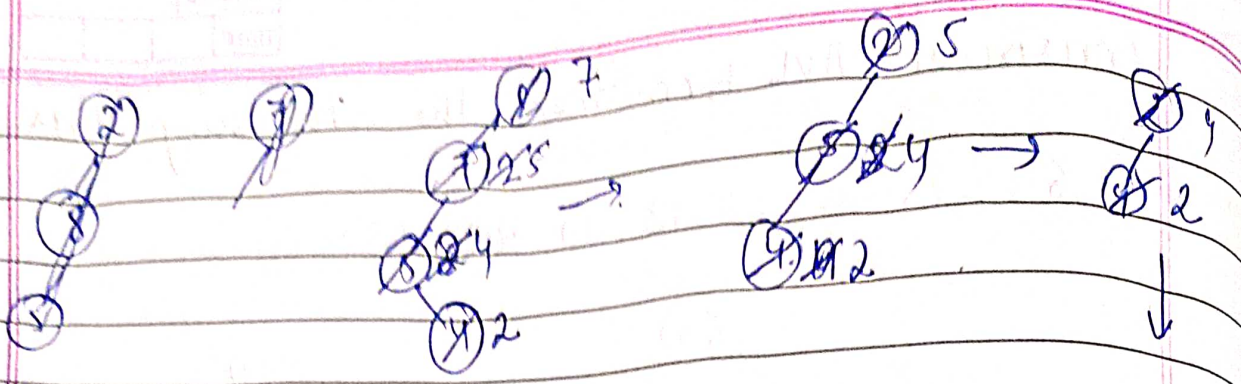
64, 1, 44, 26, 13, 110, 98, 85.



Sort the following elements using heap sort-

5 13 2 25 7 17 20 8 4 25 20 17 13



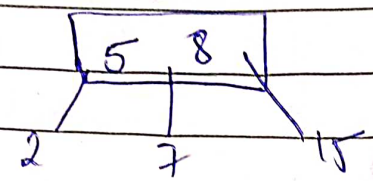


Max heap

25 20 17 13 8 7 5 4 2.

2-3 Tree

→ 8 5 2 7 15



→