Ingredients of Dynamic Programming

Dynamic programming solves problems into subproblems and these subproblems are not independent. It solves each problem once & then stores the result in a table so that it can be rapidly retrieved when needed again.

Recursion — Dynamic Programming

Top-down Mechanism. — Button-Up Mechanism

The main ingredients of dynamic programming are:-

1) Overlapping subproblems
2) Optimal substructure

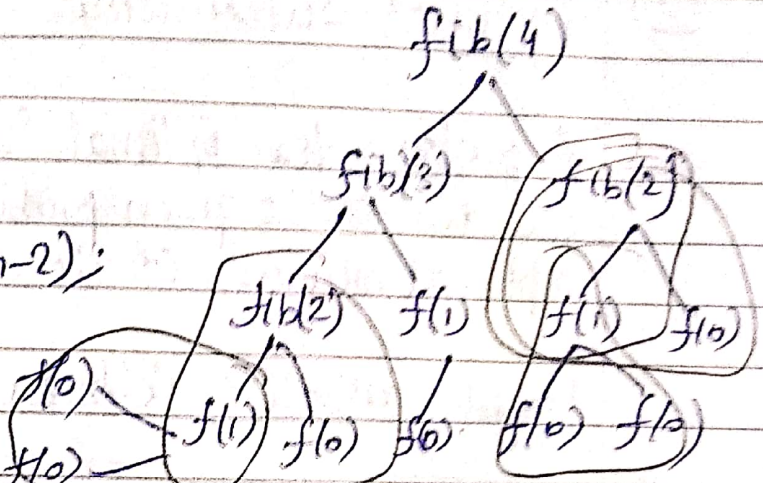
1) Overlapping subproblems

→ Divide problems into smaller subproblems and then the solutions of these are stored in a table.

→ So, dp is not useful when there are no common (overlapping) subproblems because there is no point of storing the solutions if they are not needed again.

Eg

```
int fib(int n)
{
  if (n <= 1)
     return n;
  return fib(n-1) + fib(n-2);
}
```



Value of fib(2) could have stored and then used again. To do this, there are two different ways

a) Memorization (Top-down) b) Tabulation (Bottom-up)

a) Same as recursive version with small modification that it looks into a lookup table before computing solutions. Lookup array is initialized and whenever we need to find the solution, we first look into the lookup table.

If the precomputed value is there then we return the value, otherwise we calculate the value and put the result in lookup table for later use.

b) The tabulated program builds in bottom-up fashion and returns the last entry from table. So, basically we are building the solutions of subproblems bottom up. Starting from first entry, all entries all filled one by one.

2) Optimal Substructure

A problem has Optimal Substructure Property if Optimal solution of the given problem can be obtained by using optimal solutions of its sub-problems. Seen in

Floyd-Warshall & Bellman Ford algorithms.

Greedy Method

→ Used for solving optimization problem, that is a problem that either requires minimum result or maximum result.

→ They solve problems by making the choice that seems best at the particular moment, so there can be cases when a greedy algo cannot provide efficient solution and may provide solution close to optimal. Works only when the problem has the 2 properties:-

1) Greedy Choice Property

A global optimal solution can be arrived at by making a locally optimal solution, or an optimal solution can be made by making "greedy" choices.

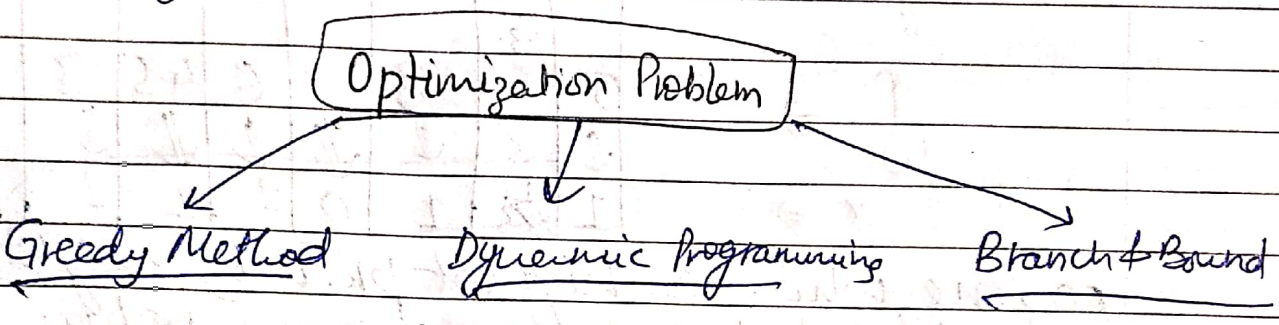
2) Optimal Substructure

Same condition as required in dynamic programming, as optimal solution contain optimal sub-solutions, or solutions to sub-problems of an optimal solution are also optimal.

→ Man the locally optimal choice at each stage.

DELTA Page No. Date / /

So, a problem that follows the constraints as mentioned in statement, then that answer is feasible and out of those feasible, the minimized or maximized answers are the optimized answers or optimization.



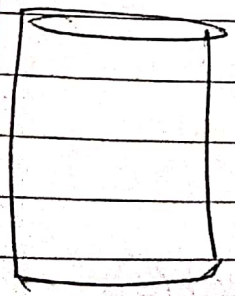
Fractional Knapsack Problem

(x) No. of objects ⇒ 0 1 2 3 4 5 6 7

Profits ⇒ P = 10, 5, 15, 7, 6, 18, 3

weight ⇒ W = 2, 3, 5, 7, 1, 4, 1

& m = 15 (Capacity in weight)



Profit should be maximized

Total weight ≤ 15

or divided

Now here, the objects can be broken down and hence some fractions from their part can be taken to maximize the solution, like $\frac{1}{3}$, $\frac{1}{2}$ or etc.

hence, $0 \leq x_i \leq 1$

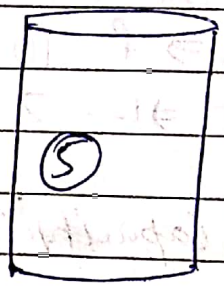
So, $m = 15$

No. of objects ⇒	0	1	2	3	4	5	6	7
Profits ⇒ P	10	85	15	7	6	18	3	
Weight ⇒ W	2	3	5	7	1	4	1	
$\frac{P}{W}$ ⇒	5	5	3	1	6	4.5	3	
x ⇒	1	0	1	0	1	1	1	

So, one who has highest profit by weight, we should select that to maximize profits.

Hence, highest $\frac{P}{W}$ is for object ⑤ so it should be included first,

So,



Weight left ⇒ 15 − 1 = 14.

Next, select object ①,

So ⑤① 14 − 2 ⇒ 12

Next, select object ⑥,

So ⑤①⑥ 12 − 4 ⇒ 8.

Next select ③,

So 8 − 5 ⇒ 3

Next, select ⑦,

So 3 − 1 = 2

Next select ②

2 − 3 X

2 − 2 = 0 (Only take 2 kg)

Hence only 2/3.

Ans

$$x = \begin{pmatrix} 1 & 2/3 & 1 & 0 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \end{pmatrix}$$

Verify, $\sum x_i w_i \leq 15$

$\Rightarrow (1 \times wt) + (2/3 \times wt) + (1 \times wt) + (0 \times wt) + (1 \times wt) + (1 \times wt) + (1 \times wt)$

$\Rightarrow (1 \times 2) + (2/3 \times 3) + (1 \times 5) + (0 \times 7) + (1 \times 1) + (1 \times 4) + (1 \times 1)$

$\Rightarrow 2 + 2 + 5 + 0 + 1 + 4 + 1 \Rightarrow 15$

$\sum p_i x_i$

$\Rightarrow (1 \times profit) + (2/3 \times prof) + (1 \times prof) + (0 \times prof) + (1 \times prof) + (1 \times prof) + (1 \times prof)$

$\Rightarrow (1 \times 10) + (2/3 \times 5) + (1 \times 15) + (0 \times 7) + (1 \times 6) + (1 \times 18) + (1 \times 3)$

$10 + 2 \times 1.3 + 15 + 0 + 6 + 18 + 3 \Rightarrow 54.6$

profit

Algo

```
Knapsack(Array v, Array w, int W)
{
for i = 1 to size(V)
    do p[i] = v[i]/w[i]
sort-descending (P)
    i ← 1
while(W > 0)
    do amount = min(W, w[i])
        solution[i] = amount
        W = W - amount
        i ← i+1
return solution
```

HUFFMAN CODING

Message ⇒ BCCABBDDAECCBBAEDDCC

Length ⇒ 20

ASCII ⇒ 8 bits

So A ⇒ 65 ⇒ 01000001
B ⇒ 66 ⇒ 01000010

20 × 8 = 160 bits
we don't need it

Fixed Size Code

$2^1 \Rightarrow 2$
$2^2 \Rightarrow 4$
$2^3 \Rightarrow 8$

Now, msg has 20 characters & 3 bits each ⇒ 60 bits msg.

Character	Count / Frequency		Code
A	3	3/20	000
B	5	5/20	001
C	6	6/20	010
D	4	4/20	111
E	2	2/20	100
	20		

5 characters & 8 bit ASCII

So 5 × 8 bits characters

5 characters × 3 bits

So 5 × 3 bits code

So Table ⇒ characters + Codes

(5 × 8) + (5 × 3)

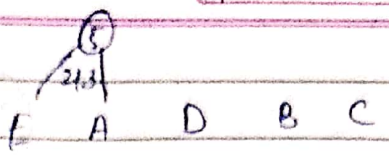
5 × 11

55 bits

So Total ⇒ 60 + 55

⇒ 115 bits

Variable Size Code

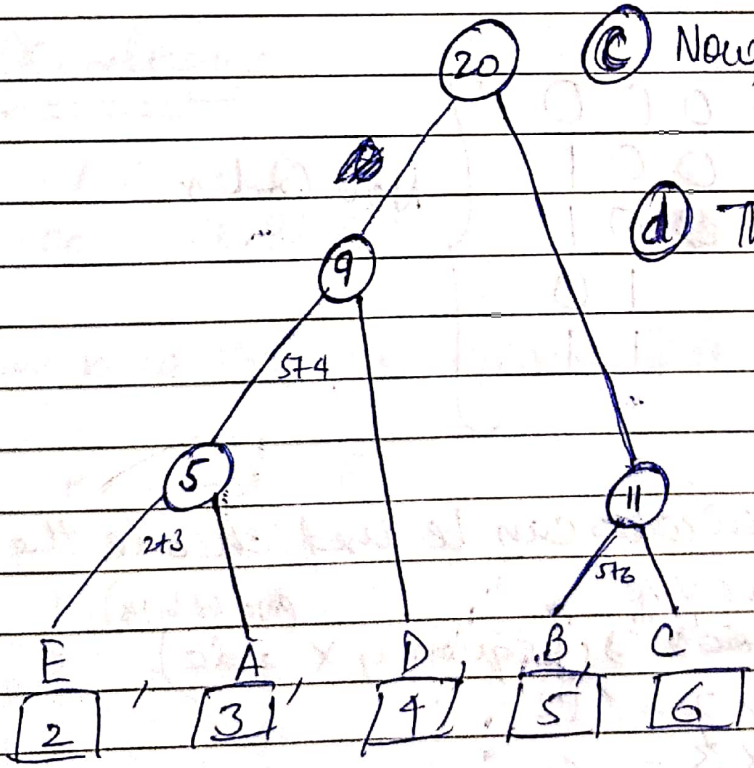


char	count/v	Code
A	3	
B	5	
C	6	
D	4	
E	2	
	20	

(a) Arrange in increasing order

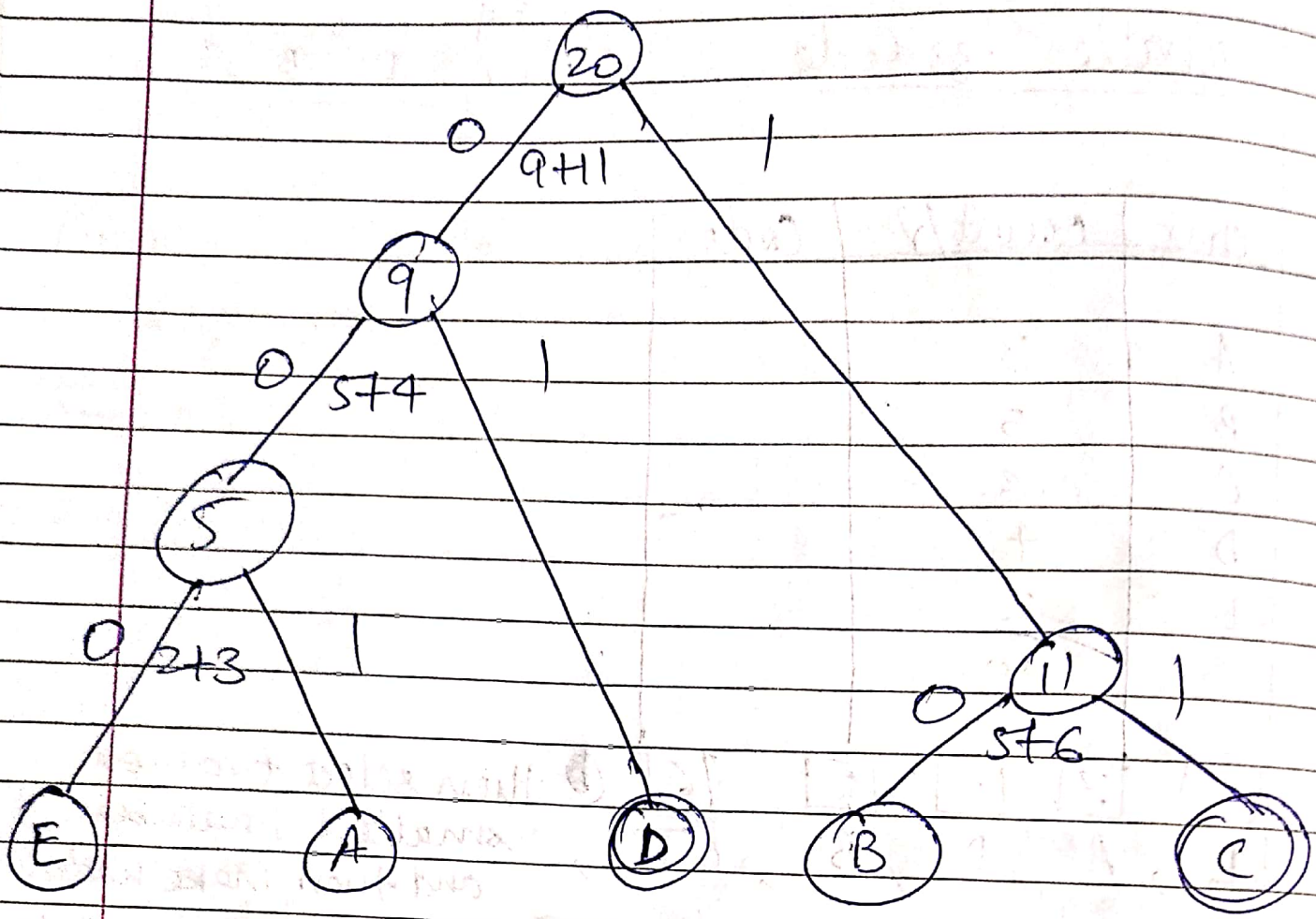
2	3	4	5	6
E	A	D	B	C

(b) Then select two most smallest numbers and then make node.



(c) Now, allocate 0 to left and 1 to right

(d) Then make 0 and 1 code from starting



∞ E ⇒ 0 0 0
A ⇒ 0 0 1
D ⇒ 0 1
B ⇒ 1 0
C ⇒ 1 1

New Codes.

Now, these new codes can be used encode the message.

∴ size of ~~code~~ message ⇒ (frequency × code) (No. of bits)

For C ⇒ 6 × 2 = 12
For E ⇒ 2 × 3 = 6
For A ⇒ 3 × 3 = 9
For D ⇒ 4 × 2 = 8
For B ⇒ 5 × 2 = 10

45 bits

Now, for table

We have 5 character & ASCII (8 bits) code

$5 \times 8 \Rightarrow 40$ bits and

Code

001
10
11
101
000
12 bits

Total ~~45~~ bits

$\therefore 40 + 12 \Rightarrow 52$ bits
↳ for Table/Tree

Total $\Rightarrow 52 + 45$
$\Rightarrow 97$ bits

and Decoding

B C C D A C -----
001 11 11 01 10 11 . . .

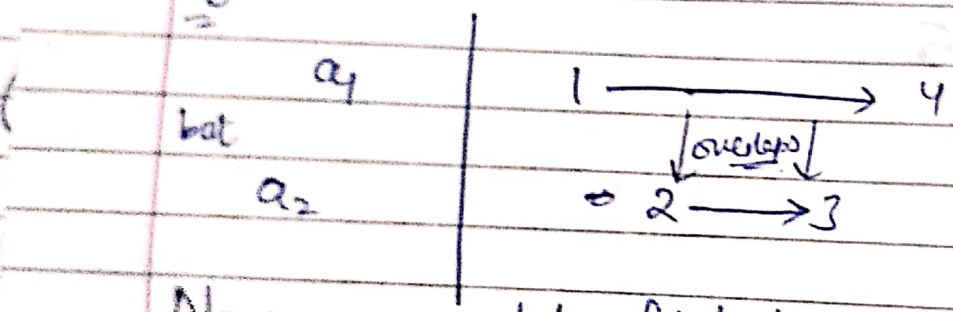
we need the tree or table for it, then

001 → B
11 → C
11 → C
01 → D
↓
↓
↓
etc

ACTIVITY SELECTION PROBLEM

Activities	a_1	a_2	a_3	a_4	a_5	a_6
Starting Time	1	2	3	4	5	6
Finishing Time	4	3	5	6	9	8

The activities should be chosen in such a way that no of activities should be maximised and the activities should satisfy the condition of compatibility that is there shouldn't be any overlapping of timings.



Now, we need to find the maximum nos of compatible activities for these given durations.

Now to do question, re-order the question by taking finishing time in increasing order

	a_0	a_4	a_2	a_3	a_4	a_5	a_6	a_7
S_i		2	1	3	4	6	5	
f_i		3	4	5	6	8	9	

Now, we are going to add a dummy activity at a_0 with S_i & $f_i \Rightarrow 0$ and a " " " a_7 with S_i & $f_i \Rightarrow \infty$.

Video - 106 (Gatebook lecture)

	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
s_i	0	2	1	3	1	8	5	∞
f_i	0	3	4	5	6	8	9	∞

Now start looking from a_4 finishing time.

Here it is 3 so now you have to find a starting time that starts with 3 or more than 3 on right side and then start collecting activities.

a_4	a_0	a_2	a_3	a_5	a_6
2	1	3	4	6	5
(3)	4	(5)	6	8	9

a_5 → After previous activity finishes, it starts.
a_3 → After previous activity finishes it starts.
a_4 → As it has least finishing time

Eg

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

→ Its already arranged in ascending order of finishing time.

→ Next, take the first activity and move on with the procedure.

a_{11}, a_8, a_4, a_1 → All these properties are found and the process is complete.

Algo Greedy (s, f)

```
{ n ← length [s]
  A ← {1}
  j ← 1
  for i ← 2 to n
      do if si ≥ fi
          then A ← A ∪ {1}
               j ← i
  return A.
```

Task Scheduling Problem

$n = 5$

Jobs	J_1	J_2	J_3	J_4	J_5
Profits	20	15	10	5	1
Deadlines	2	2	1	3	3

→ It is problem of optically scheduling unit time task on single processor such that profit is maximized.

→ A unit time task is a job, such as program that has to be run on computer that requires exactly one unit of time to complete.

Suppose, here we assume 1 Unit ⇒ 1 hour. so on m/c each job needs 1 hour for completion

n = 5

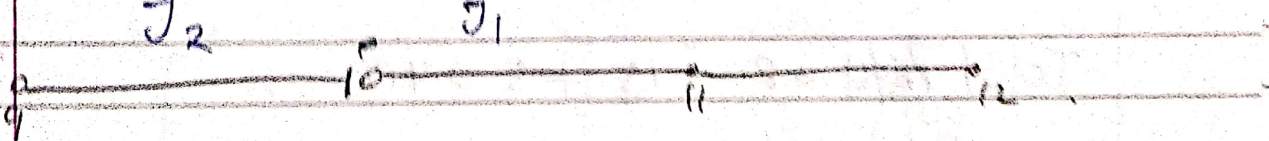
Jobs	J_1	J_2	J_3	J_4	J_5
Profits	20	15	10	5	1
Deadline	2	2	1	3	3

Now suppose that the jobs start at 9 am

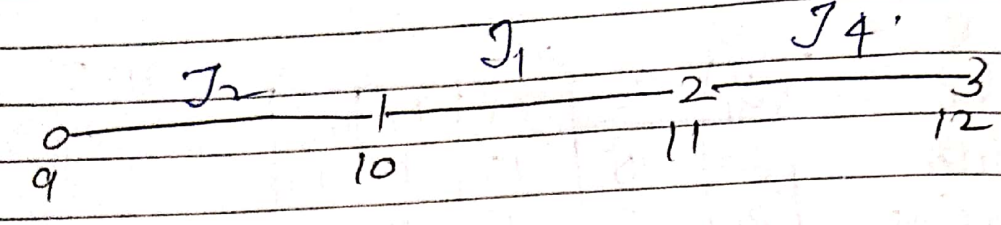
As we have total deadlines of 3 hrs hence 9 + 3 = 12 pm.

Next, as each Job needs 1 hr but their deadlines differ, hence take the job with highest profit.

So, J_1 has highest profit and deadline of 2 hours hence if we put the job in 2nd part of 2 hours, we can also accomodate jobs with deadline of 1 hr.

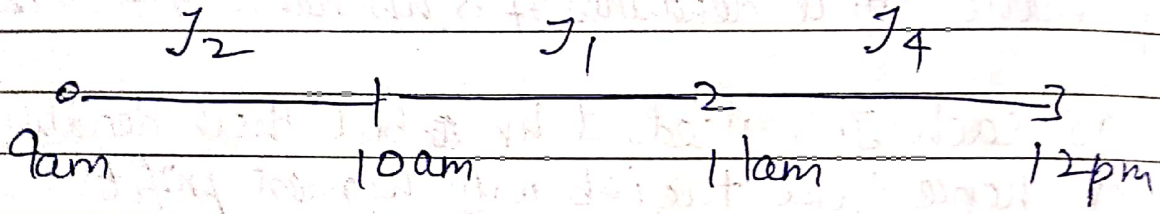


So, As J_1 has next highest profit and also a deadline of 2 hrs i.e. of 9 to 11 am. So as 10 to 11 am is occupied by J_1 so 9 to 10 am can be given to J_2



As, J_3 is next highest in profit but as deadline of 1 hr ie. 9-10am which is already occupied so we neglect it and choose next job, i.e. J_4 which has 3 hr deadline and can be accomodated b/w 9-12 hrs

so final



hence,

$$\{J_2 \rightarrow J_1 \rightarrow J_4\}$$

OR

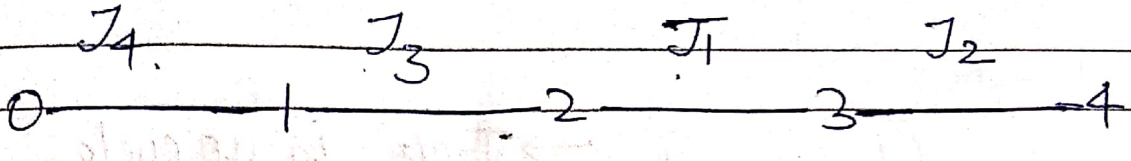
$$\{J_1 \rightarrow J_2 \rightarrow J_4\}$$

Any sequence can be done as J_1 & J_2 have same deadlines

Total Profit $\Rightarrow 20 + 15 + 5 \Rightarrow 40$

Eg 2

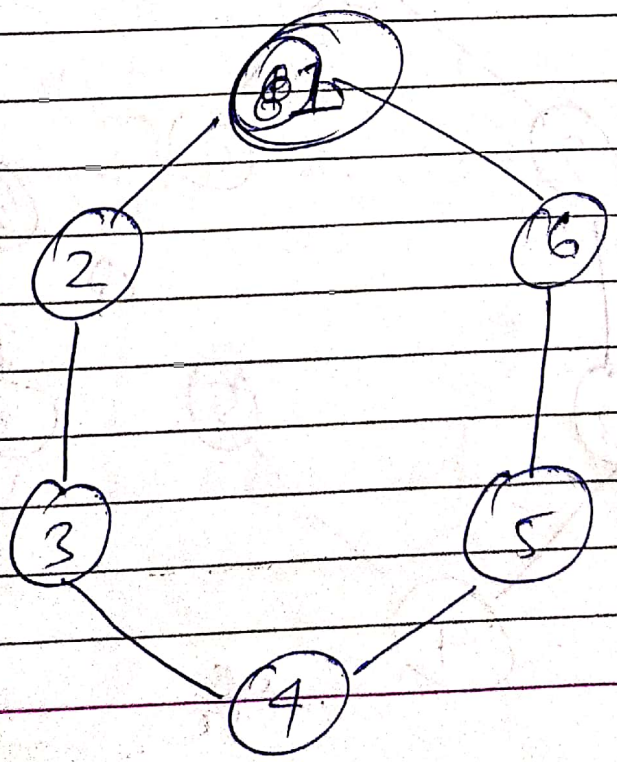
Jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7
Profits	35	30	25	20	15	12	5
Deadlines	3	4	4	2	3	1	2



profit ⇒ [20] [25] [35] [30]

Total profit ⇒ 110

MINIMUM SPANNING TREES



Graph is made up of set of vertices and edges.

$G = (V, E)$

$V = \{1, 2, 3, 4, 5, 6\}$

$E = \{(1,2)(2,3)(3,4) \dots\}$

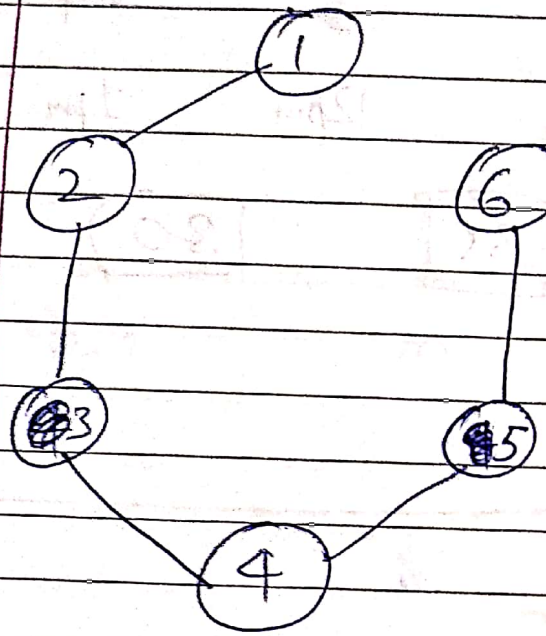
Date / /

Spanning tree – It is subset or subgraph of a graph that is made up all the vertices and only one less edge.

so if $|V| = n = 6$

so edges $\Rightarrow 6 + (n-1)$

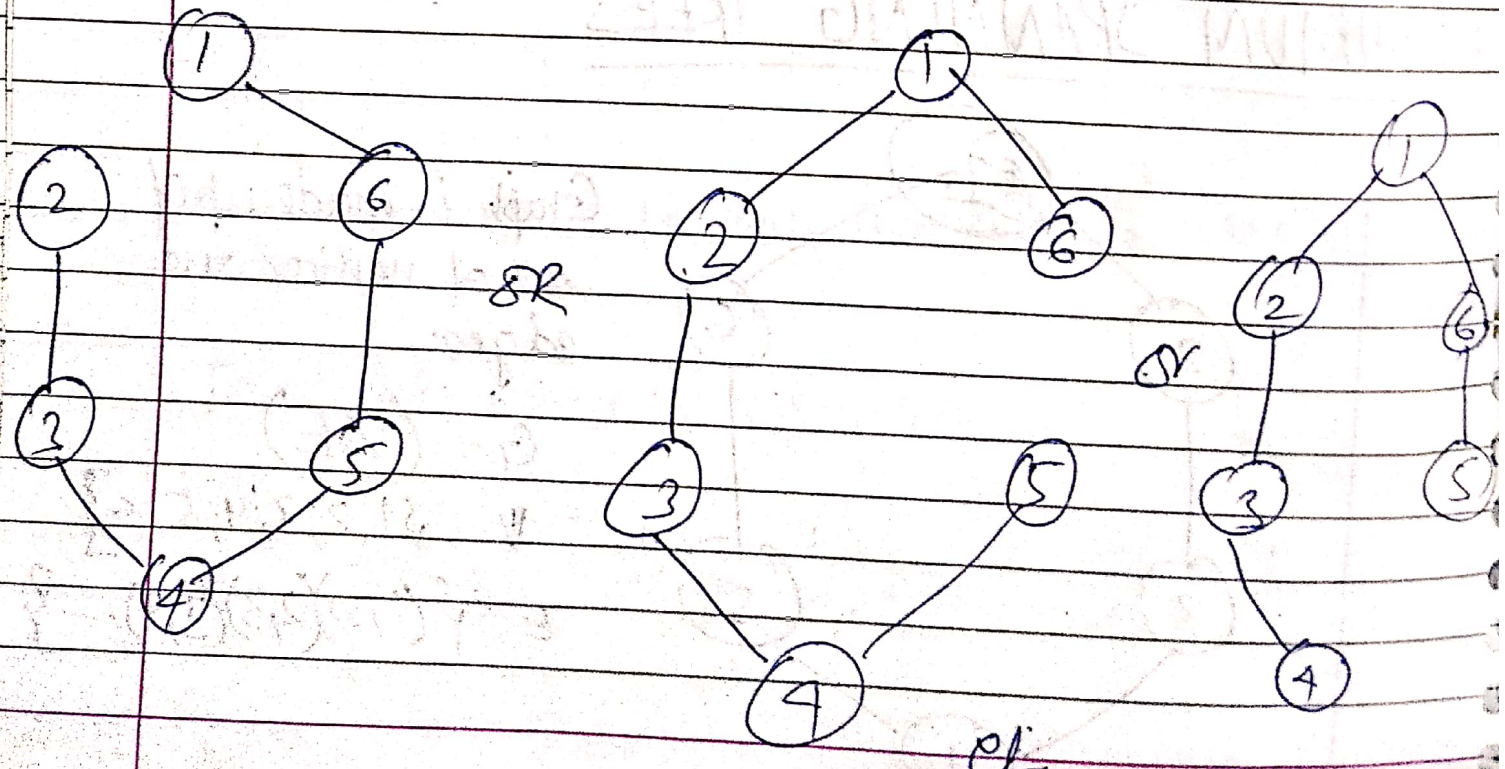
$\Rightarrow 5$



→ There is no cycle,

→ There can be many cases of graphs (spanning trees) that suffice the condition of $V = 6$

$E = 5$



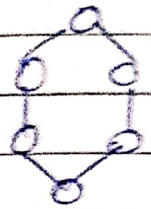
So S(subgraph) ⊆ G

$S = (V, E')$

where $V = V$

$|E'| = V - 1$

No of spanning trees ⇒ Total no of edges $C_{\text{Total edges} - 1}$

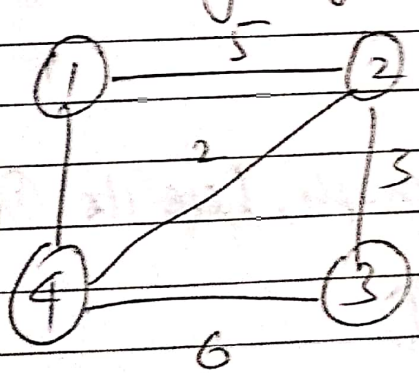


$\Rightarrow {}^{6}C_{5} = 6$

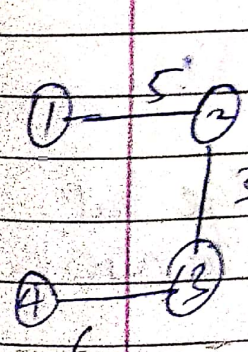
To know how many different spanning trees are possible

Formula ⇒ ${}^{\text{Total no of edges}}C_{\text{Total edges}-1}$ − No of cycles

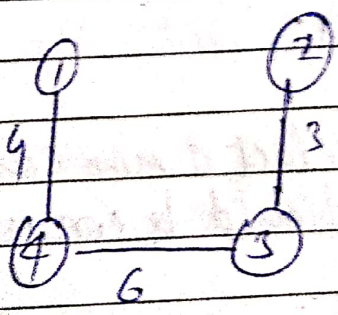
Now, if you have weighted graph,



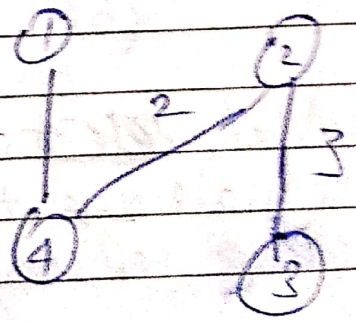
Various spanning trees are possible



Cost = 14



Cost = 13

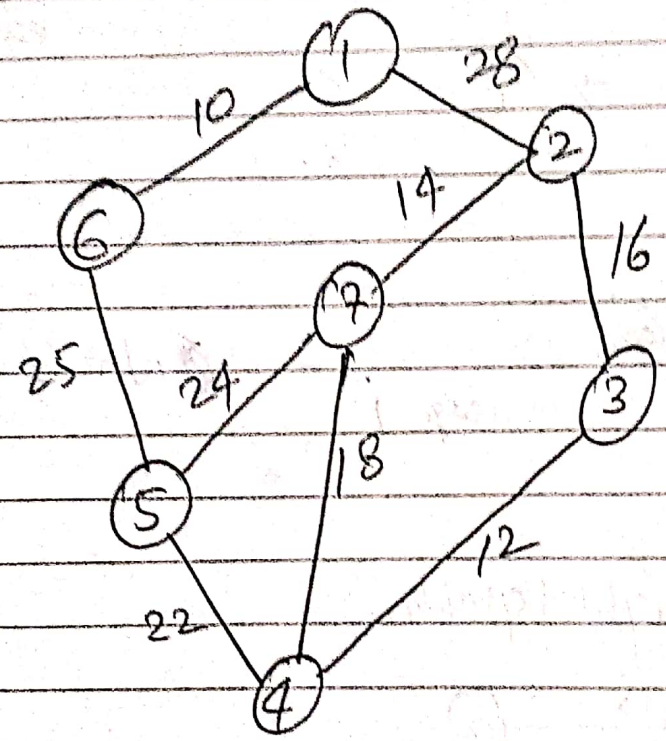


Cost = 9

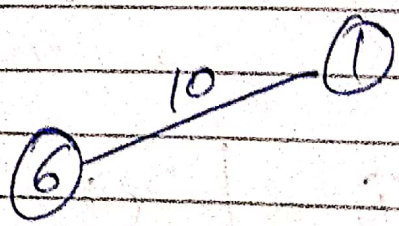
So, we need to find the minimum cost spanning tree without finding all spanning trees. Ways are

① Prim's Algo ② Krushkal's algo

① Prims Algorithm

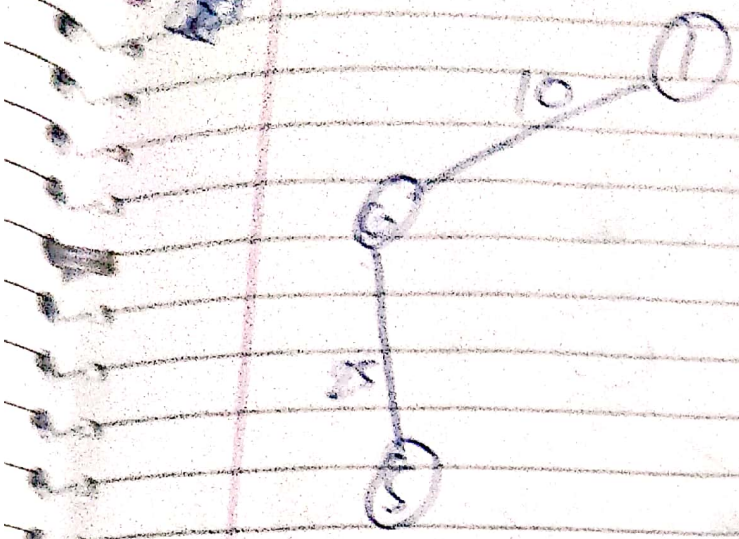


a) Select smallest edge graph, here its ① to ⑥

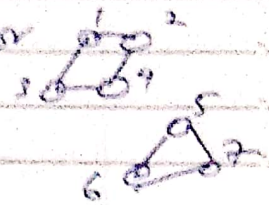


b) Next, ~~select~~ always select a min cost edge but make sure that it should be connected to already selected vertices.

③ to ④ is minimum, but it is not connected to either (7) or ⑥

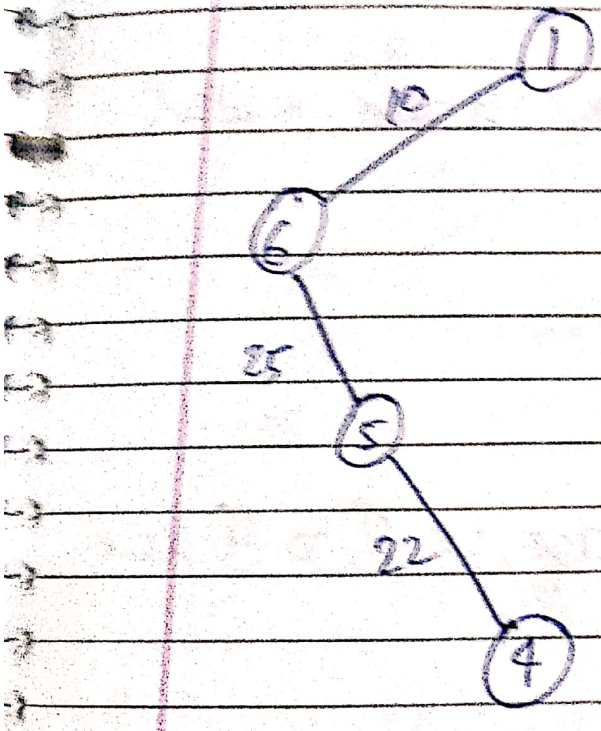


* As the graph is connected so we can find the min spanning tree, but we cannot find mst for

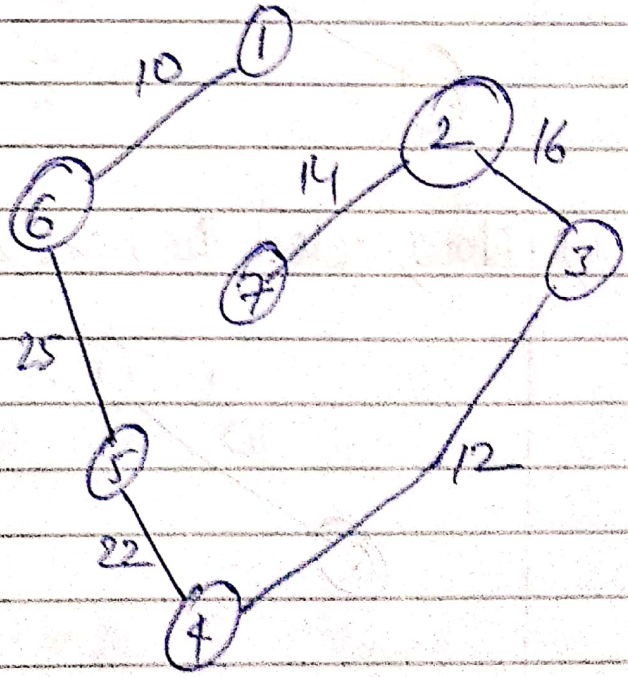


there.

c) Continuing same step as (b)



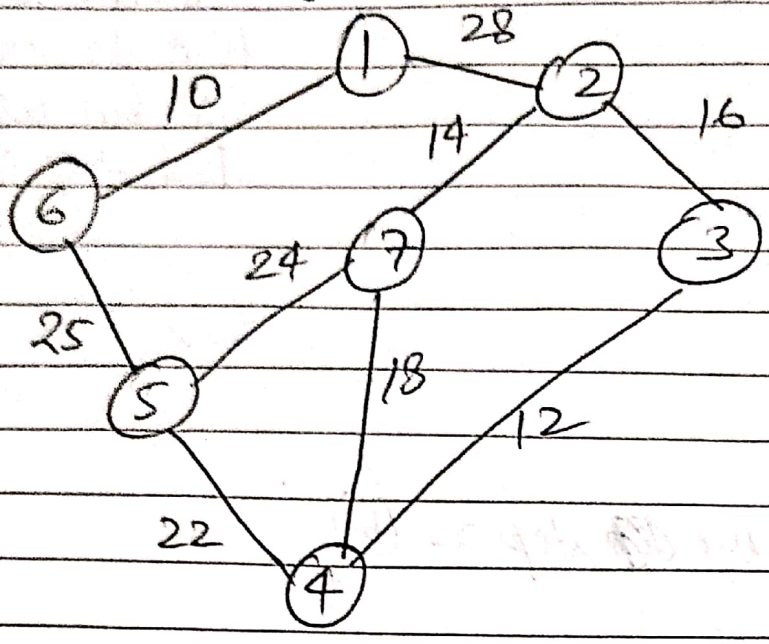
d) Again continuing



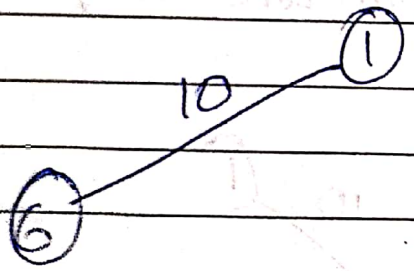
And now, all edges are connected.
So, this is how we get the minimum cost spanning tree.

cost = 99

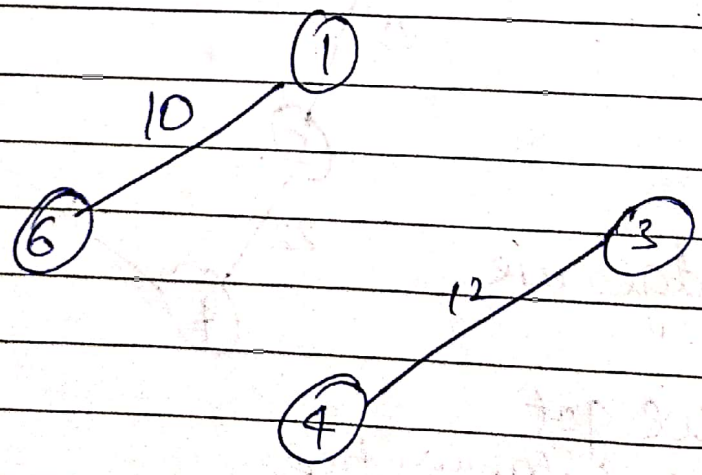
2) Kruskal's Algorithm



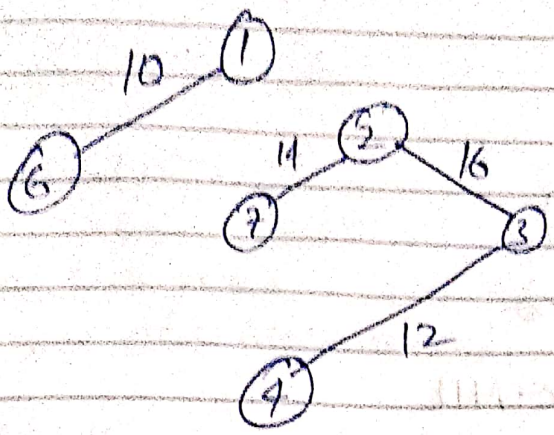
a) Select the most smallest edge or cost wise smallest.



b) Now, select the next smallest edge, i.e. (3) to (4) here

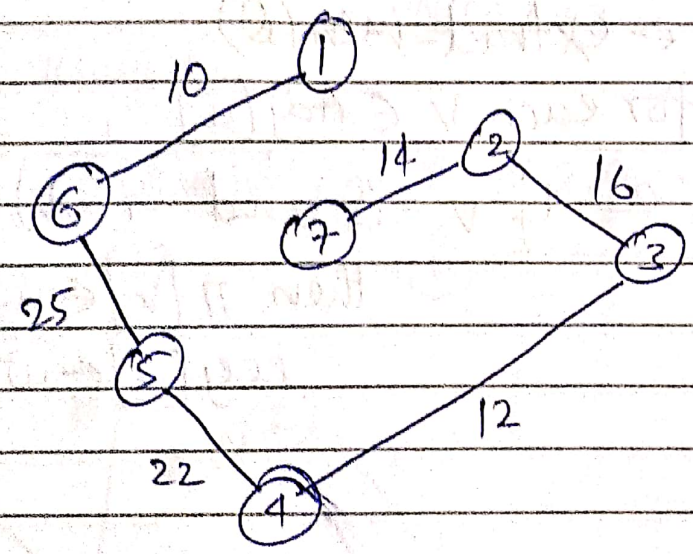


c) Similarly, select next minimum and continue with it.



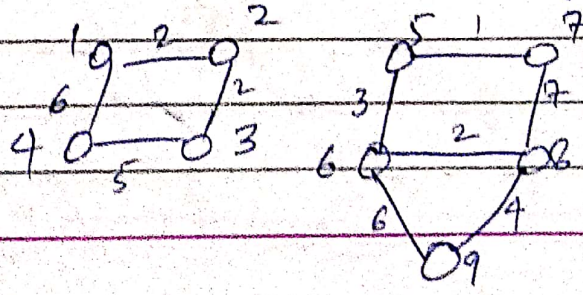
d) But, now next smallest edge is (7) to (4) with weight 18, and if we join them, it forms a cycle.

But Kruskal's method doesn't allow it so don't select it/discard it, and continue with (c)



Now adding all weights ⇒ cost = 99

* May work for non-connected graphs OR May not



DELTA Page No. Date / /

Algo (Prims)

Prim (G, w, r)

{ for each $u \in V[G]$

do $key[u] \leftarrow \infty$

$\pi[u] \leftarrow NIL$

$key[r] \leftarrow 0$

$Q \leftarrow V[G]$

while $Q \neq \phi$

do $u \leftarrow$ extract-min (Q)

for each $v \in Adj[u]$

do if $v \in Q$ and $w(u,v) < key[v]$

then $\pi[v] \leftarrow u$

$key[v] \leftarrow w[u,v]$

Algo (Kruskal's)

Kruskal(G, w)
A ← ∅
for each vertex v ∈ V[G]
do make-set(v)

Sort the edges of E into increasing order by weight w

for each edge (u,v) ∈ E, taken in increasing order

do if Find-Set(u) ≠ Find-Set(v)
then A ← A ∪ {(u,w)}
UNION(u,v)

Return A

Eg Ref