Ch-4 Dynamic Programming.

Q-1] Comparision of Devide & Conqure and dynamic Programming.

Ans)

Devide & Conqure	dynamic Programming
→ The Problems are devided into small subProblems. These subProblems are solved independently. finally all the solution of subProblems are collected together to get the solution of given Problem.	In dynamic Programming many decision sequences are generated and all the subinstance are considered.
→ In this method the duplication of subsolution is neglected. i.e duplicate subsolution may be obtained.	In dynamic Programming duplication of solution is avoided totally.
→ Devide & Conqure method is less efficient because reworks on solution.	dynamic Programming is efficient then devide & Conqure method.
→ Devide & Conqure method uses top down approach of Problem solving (Recursive method)	→ dynamic Programming uses Bottom up approach of Problem solving. (Iterative method)
→ Devide & Conqure splits its input at specific deterministic point usually in the middle.	dynamic Programming split its input at all the possible split point rather then a particular split point. After trying all split point it determines which split point is more optimum.

Q-2] What are the advantages of dynamic Programming method over devide-&-conqure method?

Ans) In dynamic Programming duplication of solution is totally avoided.
→ dynamic Programming method is efficient then devide-&-conqure method.
→ Devide & conqure method splits its input at the middle however, in dynamic Programming split it's input at every possible split point rather then a particular split point. After trying every split point it determines which split point is optimum.

Q-3] What are the disadvantages of greedy algorithm over dynamic Programming method?

Ans)
1) In greedy algorithm there are no guarantee of getting optimum solution.
2) The optimum selection is without revising previously generated solution.
3) Designing greedy algorithm with right approach is hard.
4) Knowing greedy algorithm correct is hard.

Q-4] Principle of optimality [m. imp].

Ans) The dynamic Programming obtain the solution using Principle of optimality.
→ The Principle of optimality states that "In an optimal sequence of desisions and choices, each subsequence must also be optimal".
→ When it is impossible to apply Principle of optimality it is almost impossible to obtain solution using dynamic Programming approach.
→ Fort to find the shortest path of given graph ~~using~~ using Principle of optimality.

Note

$C(n,k) \Rightarrow {}^{n}C_{k} \Rightarrow \binom{n}{k} \Rightarrow \frac{n!}{k!(n-k)!}$

Calculating the Binomial coefficient.

eqn. $C[i,j] = C[i-1,j] + C[i-1,j-1]$.

Ex-1] Compute $C(4,2)$ using dynamic Programming.

Ans Here $C(n,k) = C(4,2)$

$\therefore \boxed{n=4}$, $\boxed{k=2}$

→ We draw table for $n=4$ and $k=2$

k → / n ↓	0	1	2
0	1	X	
1	1	1	X
2	1	2	1
3	1	3	3
4	1	4	(6)

(j →, i ↓)

→ Here $k=0$ to 2 and $n=0$ to 4.

→ Put $C[i,j]=1$ where $j=0$

→ Put cross where $j>i$ in each row.

→ Now we compute remaining cells using this equation $C[i,j] = C[i-1,j] + C[i-1,j-1]$.

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

② $C[2,1] = C[1,1] + C[1,0]$
$= 1 + 1$
$= 2$

③ $C[2,2] = C[1,2] + C[1,1]$
$= 0 + 1$
$= 1$

④ $C[3,1] = C[2,1] + C[2,0]$
$= 2 + 1$
$= 3$

⑤ $C[3,2] = C[2,2] + C[2,1]$
$= 1 + 2$
$= 3$

⑥ $C[4,1] = C[3,1] + C[3,0]$
$= 3 + 1$
$= 4$

⑦ $C[4,2] = C[3,2] + C[3,1]$
$= 3 + 3$
$\boxed{C[4,2] = 6}$

$\therefore \boxed{C(4,2) = 6}$

Ex-2] Compute $NCR\binom{5}{3}$ using dynamic method.

Ans Here ${}^{n}C_{k} = {}^{5}C_{3}$

$\therefore \boxed{n=5}$ and $\boxed{k=3}$

→ We draw table for $n=5$ and $k=3$

k → / n ↓	0	1	2	3
0	1	X		
1	1	1	X	
2	1	2	1	X
3	1	3	3	1
4	1	4	6	4
5	1	5	10	(10)

(j →, i ↓)

→ Here $k=0$ to 3 and $n=0$ to 5.

→ We Put $C[i,j]=1$ where $j=0$.

→ Put $C[i,j]=0$ (or cross x) where $j>i$ in each row.

→ Now we compute remaining cells using these equation $C[i,j] = C[i-1,j] + C[i-1,j-1]$.

① C[1,1] = C[0,1] + C[0,0]
= 0 + 1
= 1

② C[2,1] = C[1,1] + C[1,0]
= 1 + 1
= 2

③ C[2,2] = C[1,2] + C[1,1]
= 0 + 1
= 1

④ C[3,1] = C[2,1] + C[2,0]
= 2 + 1
= 3

⑤ C[3,2] = C[2,2] + C[2,1]
= 1 + 2
= 3

⑥ C[3,3] = C[2,3] + C[2,2]
= 0 + 1
= 1

⑦ C[4,1] = C[3,1] + C[3,0]
= 3 + 1
= 4

⑧ C[4,2] = C[3,2] + C[3,1]
= 3 + 3
= 6

⑨ C[4,3] = C[3,3] + C[3,2]
= 1 + 3
= 4

⑩ C[5,1] = C[4,1] + C[4,0]
= 4 + 1
= 5

⑪ C[5,2] = C[4,2] + C[4,1]
= 6 + 4
= 10

⑫ C[5,3] = C[4,3] + C[4,2]
= 4 + 6
C[5,3] = 10

∴ NCR $\binom{5}{3}$ = 10

Algorithm of Binomial coefficient

```
Algorithm Binomial(n,k)
for(i=0 to n) do
{
    for(j=0 to k) do
    {
        if(j=0 or (i=j))
        {
            C[i,j] = 1
        }
        else
        {
            C[i,j] = C[i-1,j] + C[i-1,j-1]
        }
    }
}
return [n,k];
```

Time complexity = O(nk).

Making change Problem :-

Ex-1) Solve making change problem for $d_1=1$, $d_2=4$, $d_3=6$, $n=3$ and $N=8$ units.

Ans)

Note

Three formula is important for making change problem

- if $i=1$ then $C[i,j] = 1 + C[1, j-d_1]$
- if $i \neq 1$ and $j < d_i$ then $C[i,j] = C[i-1, j]$
- Otherwise $C[i][j] = \min(C[i-1,j], 1 + C[i, j-d_i])$.

- Here $d_1=1$, $d_2=4$, $d_3=6$ and $N=8$ units
- Initially our table is empty $C[i,j]=0$

$C[1,0] = C[2,0] = C[3,0] = 0$

- In our table row is staging from 1 to 3 and columns is staging from 0 to 8.

↓ i / j →		0	1	2	3	4	5	6	7	8
i=1	$d_1=1$	0	1	2	3	4	5	6	7	8
i=2	$d_2=4$	0	1	2	3	1	2	3	4	2
i=3	$d_3=6$	0	1	2	3	1	2	1	2	

① $C[1,1]$ with $i=1, j=1, d_1=1$

Here $i=1$ Hence,

$C[i,j] = 1 + C[1, j-d_1]$

$= 1 + C[1, 1-1]$

$= 1 + C[1,0]$

$= 1 + 0$

$= 1$

② $C[2,1]$ with $i=2, j=1, d_2=4$

Here $i \neq 1$ so we check for $j < d_i \Rightarrow 1 < d_2 \Rightarrow 1 < 4$ is True Hence.

$C[i,j] = C[i-1, j]$

$= C[1,1]$

$= 1$

Note :- If we do short method of finding $\min(\text{above value}, 1 + [j - d_i])$

③ $C[3,1]$ with $i=3, j=1, d_3=6$

Here $i \neq 1$ so we check for $j < d_i \Rightarrow 1 < d_3 \Rightarrow 1 < 6$ is True then

$C[i,j] = C[i-1, j]$

$= C[2,1]$

$= 1$

④ $C[1,2]$ with $i=1, j=2, d_1=1$

Here $i=1$ Hence

$C[i,j] = 1 + C[1, j-d_1]$

$= 1 + C[1, 1]$

$= 1 + 0 1$

$= 2$

⑤ $C[2,2]$ with $i=2, j=2, d_2=4$

Here $i \neq 1$ so we check for $j < d_i \Rightarrow 2 < d_2 \Rightarrow 2 < 4$ is true then

$C[i,j] = C[i-1, j]$

$= C[1,2]$

$= 2$

⑥ $C[2,3]$ with $i=2, j=3, d_3=6$

Here $i \neq 1$ so we check for $j < d_i \Rightarrow 3 < d_2 \Rightarrow 3 < 6$ is true then

$C[i,j] = C[i-1, j]$

$= C[1,3]$

⑥ $C[3,2]$ with $i=3, j=2, d_3=6$

Here $i \neq 1$ so we check for $j < d_i \Rightarrow 2 < 6$ is true then

$C[i,j] = C[i-1, j]$

$= C[2,2]$

$= 2$

⑦ $C[1,3]$ with $i=1, j=3, d_1=1$

Here $i=1$ Hence

$C[i,j] = 1 + C[1, j-d_1]$

$= 1 + C[1,2]$

$= 1 + 2$

$= 3$

(8) C[2,3] with i=2, j=3 and $d_2 = 4$
Here i≠1 Hence we check $j < d_i \Rightarrow 3 < 4$ is true then
C[i,j] = C[i-1, j]
= C[1, 3]
= 3

(9) C[3,3] with i=3, j=3 and $d_3 = 6$
Here i≠1 Hence we check $j < d_i \Rightarrow 3 < 6$ true then
C[i,j] = C[i-1, j]
= C[2, 3]
= 3

(10) C[1,4] with i=1, j=4 and $d_1 = 1$
Here i=1 Hence
C[i,j] = 1 + C[1, j-d1]
= 1 + C[1, 3]
= 1 + 3
= 4

(11) C[2,4] with i=2, j=4 and $d_2 = 4$
Here i≠1 Hence we check $j < d_i \Rightarrow 4 < 4$ is also false
Hence,
C[i,j] = min(C[i-1, j], 1 + C[i, j-di])
= min(C[1, 4], 1 + C[2, 0])
= min(4, 1 + 0)
= 1

(12) C[3,4] with i=3, j=4 and $d_3 = 6$
Here i≠1 Hence we check $j < d_i \Rightarrow 4 < 6$ is true then
C[i,j] = C[i-1, j]
= C[2, 4]
= 1

→ Now we continue this process and till the end we get this table

	N→	0	1	2	3	4	5	6	7	8
i=1	$d_1 = 1$	0	1	2	3	4	5	6	7	8
i=2	$d_2 = 4$	0	1	2	3	1	2	3	4	2
i=3	$d_3 = 6$	0	1	2	3	1	2	1	2	2

(8-4) Here is difference we take 1 4 coin
← we need 2 coins

Note
Here at C[2,5] we do ghost method
→ min(5, 1+1) → min(5,2) → 2
@ At C[2,6]
→ min(6, 1+(6-4))
→ min(6, 1+C[2]) → min(6, 1+2)
→ 3

→ Answer: (4, 4)

Ex-2) given coins of denominations 2, 4 and 5 with amount to be pay is 7. find out optimal number of coins and sequence of coins used to pay given amount using dynamic method.

Ans) Here $d_1 = 2$, $d_2 = 4$ and $d_3 = 5$
N = 7.
→ Initially our table is empty C[i,j] = 0
→ for C[1,0] = C[2,0] = C[3,0] = 0.

j →

↓	N→	0	1	2	3	4	5	6	7
i=1	$d_1 = 2$	0	∞	1	∞	2			
i=2	$d_2 = 4$	0							
i=3	$d_3 = 5$	0							

① C[1,1] with i=1, j=1, d_1 = 2

Here i=1 Hence

C[i,0] ~~C[i,0]~~ C[i,j] = 1 + C[1, j-d_1]

= 1 + C[1, -1]

= 1 + ∞

= ∞

② C[1,2] with i=1, j=2 and d_1 = 2

Here i=1 Hence

C[i,j] = 1 + C[1, j-d_1]

= 1 + C[1, 0]

= 1 + 0

= 1

③ C[1,3] with i=1, j=3 and d_1 = 2

Here i=1 Hence

C[i,j] = 1 + C[1, j-d_1]

= 1 + C[1, 1]

= 1 + ∞

= ∞

④ C[1,4] with i=1, j=4 and d_1 = 2

Here i=1 Hence

C[i,j] = 1 + C[1, j-d_1]

= 1 + C[1, 2]

= 1 + 1

= 2

→ We continue this process till the end and we get this type of table

	W→	0	1	2	3	4	5	6	7
i=1	d_1=2	0	∞	1	∞	2	∞	3	∞
i=2	d_2=4	0	∞	1	∞	1	∞	2	∞
i=3	d_3=5	0	∞	1	∞	1	1	2	2

Answer:

{2, 5}

Knapsack Problem: [0/1 knapsack].

- Equations:

If j ≥ w_i then

m[i,j] = max(m[i-1, j] , v_i + m[i-1, j-w_i])

If j < w_i then

m[i,j] = m[i-1, j]

Ex-1) Solve the following 0/1 knapsack Problem using dynamic Programming. There are five items whose weights and values are given in following order.

Weight W[] = {1, 2, 5, 6, 7}

Value V[] = {1, 6, 18, 22, 28}

Show your equation and findout the optimal knapsack items for weight capacity of 11 units.

Ans:

Here W=11 units

Let,

item	Weight	Profit
1	1	1
2	2	6
3	5	18
4	6	22
5	7	28

→ Now we draw one dp table for following knapsack.

→ Create m[i,j] table with

rows → 0 to 11

columns → 0 to 5

m

Vi	Wi	i \ J	0	1	2	3	4	5	6	7	8	9	10	11
		0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
6	2	2	0	1	6	7	7	7						
18	5	3	0											
22	6	4	0											
28	7	5	0											

max(1, 6+1) max(2, 6+2) max(2, 6+2) max(29, 28+6)

→ Initially
m[i,j] = 0 if i = j = 0

① m[1,1] with i=1, j=1, Vi=1, Wi=1
Here j ≥ wi ⇒ 1 ≥ 1 then
m[i,j] = max(m[i-1,j], Vi + m[i-1, j-wi])
m[1,1] = max(m[0,1], 1 + m[0,0])
m[1,1] = max(0, 1+0)
m[1,1] = max(0,1)
= 1

② m[1,2] with i=1, j=2, Vi=1, Wi=1
Here j ≥ wi ⇒ 2 ≥ 1 then
m[i,j] = max(m[i-1,j], Vi + m[i-1, j-wi])
= max(m[0,2], 1 + m[0,1])
= max(0, 1+0)
= 1

③ m[1,3] with i=1, j=3, Vi=1, Wi=1
Here j ≥ wi ⇒ 3 ≥ 1 then
m[i,j] = max(m[i-1,j], Vi + m[i-1, j-wi])
= max(m[0,3], 1 + m[0,2])
= max(0, 1+0)
= 1

continue.

④ m[2,1] with i=2, j=1, Vi=6, Wi=2
Here j < wi ⇒ 1 < 2 then
m[i,j] = m[i-1,j]
= m[1,1]
= 1

⑤ m[2,2] with i=2, j=2, Vi=6, Wi=2
Here j ≥ wi ⇒ 2 ≥ 2 then
m[i,j] = max[m[i-1,j], Vi + m[i-1, j-wi])
= max[m[1,2], 6 + m[1,0])
= max[1, 6+0]
= 6

We continue to fill all this cells using this method and till end we get below table

m

Vi	Wi	i \ J	0	1	2	3	4	5	6	7	8	9	10	11
		0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
6	2	2	0	1	6	7	7	7	7	7	7	7	7	7
18	5	3	0	1	6	7	7	18	19	24	25	25	25	25
22	6	4	0	1	6	7	7	18	22	24	28	29	29	40
28	7	5	0	1	6	7	7	18	22	28	29	34	35	40

Note (above method to solve it direct)
m[2,3] = max(6+1, 1) = 7
m[2,4] = max(6+1, 1) = 7
m[3,5] = max(18+0, 7) = 18
m[4,11] = max(22+18, 25) = 40

Answer:-
→ selected item 3 and item 4 that is {5, 6}
of it = 18+22 = 40

(7-2) Given a knapsack having maximum weight capacity W=4 and number of items available are three, such that

S=3

$W_i = \langle 1, 3, 4 \rangle$

$V_i = \langle 3, 4, 5 \rangle$

fill the knapsack using DP such that knapsack should not exceed to maximum capacity and it should have maximum Profit value.

Ans 2 Here W=4 and n=3

let,

Items	Weight	Profit
1	1	3
2	3	4
3	4	5

→ we draw DP table for following knapsack

→ m[i,j]=0 if i=j=0

m

		j / i	0	1	2	3	4
Vi	Wi		0	0	0	0	0	0
3	1		1	0	3	3	3	3
4	3		2	0	3			
5	4		3	0				

(1) m[1,1] with i=1, j=1, V_i=3, W_i=1

Here j ≥ W_i then

$m[i,j] = \max(m[i-1,j], V_i + m[i-1, j-W_i])$

$= \max(m[0,1], 3 + m[0,0])$

$= \max(0, 3+0)$

$= 3$

(2) m[1,2] with i=1, j=2, V_i=3, W_i=1

Here j ≥ W_i then

$m[i,j] = \max\{m[i-1,j], V_i + m[i-1, j-W_i]\}$

$= \max[m[0,2], 3 + m[0,1]]$

$= \max(0, 3+0)$

$= 3$

(3) m[2,1] with i=2, j=1, V_i=4, W_i=3

Here j < W_i then

$m[i,j] = m[i-1, j]$

$= m[1,1]$

$= 3$

By following this method we get table like below

		i / j	0	1	2	3	4
Vi	Wi		0	0	0	0	0	0
3	1		1	0	3	3	3	3
4	3		2	0	3	3	4	7
5	4		3	0	3	3	4	7

Answer :-

∴ item 1 and item 2 are selected that is <1,3>

with Profit <3,4> = 7

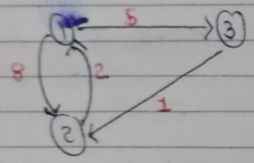
The Floyd-Warshall algorithm :
[to find shortest path]

Eqn for D^k

~~$D_k[i,j] = D_{k-1}$~~

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

Ex-1] Obtain the all pair shortest path using floyd's algorithm for the following weighted graph.



Ans

→ First we find all-pair shortest path using no intermediate vertex and denoted as D_0

$$D_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 8 & 5 \\ 2 & 0 & \infty \\ \infty & 1 & 0 \end{bmatrix} \end{array}$$

→ Now we find all pair shortest path using Node 1 as intermediate vertex and denoted it as D_1

$\therefore D_1 \Rightarrow D_k \quad \therefore k=1$

① $D_1[1,1] = 0$ with $k=1, i=1, j=1$

$\therefore D_1[1,1] = \{D_{(k-1)}[i,j], D_{(k-1)}[i,k] + D_{(k-1)}[k,j]\}$
$\therefore D_1[1,1] = \min\{D_0[1,1], D_0[1,1] + D_0[1,1]\}$
$= \min\{0, 0+0\}$
$= 0$

② $D_1[1,2]$ with $k=1, i=1, j=2$

$\therefore D_1[1,2] = \min\{D_{(k-1)}[i,j], D_{(k-1)}[i,k] + D_{(k-1)}[k,j]\}$
$= \min\{D_0[1,2], D_0[1,1] + D_0[1,2]\}$
$= \min\{8, 0+8\}$
$= 8$

③ $D_1[1,3]$ with $k=1, i=1, j=3$

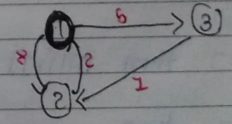
$\therefore D_1[1,3] = \min\{D_{(k-1)}[i,j], D_{(k-1)}[i,k] + D_{(k-1)}[k,j]\}$
$= \min\{D_0[1,3], D_0[1,1] + D_0[1,3]\}$
$= \min\{5, 0+5\}$
$= 5$

④ $D_1[2,1]$ with $k=1, i=2, j=1$

$\therefore D_1[2,1] = \min\{D_{(k-1)}[i,j], D_{(k-1)}[i,k] + D_{(k-1)}[k,j]\}$
$= \min\{D_0[2,1], D_0[2,1] + D_0[1,1]\}$
$= \min\{2, 2+0\}$
$= 2$

continue till end and we get D_1 as

$$D_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 8 & 5 \\ 2 & 0 & (7) \\ \infty & 1 & 0 \end{bmatrix} \end{array} \Rightarrow$$



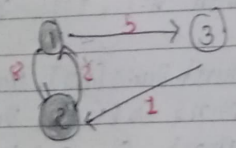
⇒ Now we find all pair shortest path using Node 1 and Node 2 As intermediate vertex and denote it as D_2

→ From the eqn $D_k[i,j] = \min\{D_{(k-1)}[i,j], D_{(k-1)}[i,k] + D_{(k-1)}[k,j]\}$

We get following shortest path for D_2

D_2 =

	1	2	3
1	0	8	5
2	2	0	7
3	3	1	0



→ Now we find all Pair Shortest Path using Node 1, Node 2 and Node 3 as intermediate vertex and denoted it as D_3

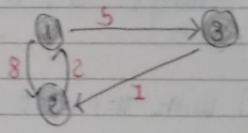
From the following eqn

$D_3[i,j] = \min\{D_{3-1}[i,j],\ D_{3-1}[i,3] + D_{3-1}[3,j]\}$

we get the D_3

D_3 =

	1	2	3
1	0	6	5
2	2	0	7
3	3	1	0



→ So D_3 is our All Pair Shortest Path for following weighted graph.

Matrix Chain Multiplication

Conditions

(i) if $i=j$ then $m[i,j]=0$

(ii) if $i<j$ then

$$m[i,j] = m[i,k] + m[k+1,j] + P_{i-1}P_kP_j$$

for $i \le k \le j-1$

if k is different so

→ $\min\begin{cases} m[i,k] + m[k+1,j] + P_{i-1}P_kP_j & \text{for } k=k_2 \\ m[i,k] + m[k+1,j] + P_{i-1}P_kP_j & \text{for } k=k_3 \end{cases}$

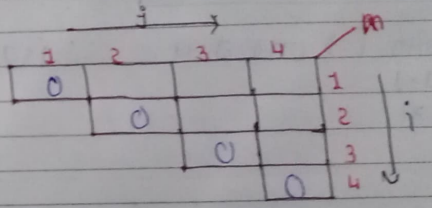
Ex:1) Explain chained matrix multiplication for following example sequence for multiplication $D = \langle 15, 5, 10, 20, 25 \rangle$

Ans Here

$P_0=15$, $P_1=5$, $P_2=10$, $P_3=20$, $P_4=25$

$N=5$

→ we draw one matrix of size $(N-1) \times (5-1) \Rightarrow 4$



→ if $i=j$ then $m[i,j]=0$, so put 0 in above matrix where $[i=j]$

$\therefore m[1,1] = m[2,2] = m[3,3] = m[4,4] = 0$

→ if $i<j$ then

$m[i,j] = m[i,k] + m[k+1,j] + P_{i-1}P_kP_j$ for $i \le k \le j-1$

→ We start calculation with following eqn

(i) $m[1,2]$ with $i=1$, $j=2$ $\quad \because i<j$

$\therefore i \le k \le j-1$

$\therefore 1 \le k \le 1$

$\therefore \boxed{k=1}$

$\therefore m[i,j] = m[i,k] + m[k+1,j] + P_{i-1}P_kP_j$

$\therefore m[1,2] = m[1,1] + m[2,2] + P_0P_1P_2$

$= 0 + 0 + (15)(5)(10)$

$\boxed{m[1,2] = 750 \text{ with } k=1}$

② $m[2,3]$ with $i=2$ and $j=3$ $\because i<j$

$\therefore i \le u < j-1$

$\therefore 2 \le u < 2$

$\therefore \boxed{u=2}$

$\therefore m[i,j] = m[i,u] + m[u+1,j] + p_{i-1} p_u p_j$

$= m[2,2] + m[3,3] + p_1 p_2 p_3$

$= 0 + 0 + (25)(10)(20)$

$\boxed{m[2,3] = 1000 \text{ with } u=2}$

③ $m[3,4]$ with $i=3$ and $j=4$ $\because i<j$

$\therefore i \le u < j-1$

$\therefore 3 \le u < 3$

$\therefore \boxed{u=3}$

$\therefore m[i,j] = m[i,u] + m[u+1,j] + p_{i-1} p_u p_j$

$= m[3,3] + m[4,4] + p_2 p_3 p_4$

$= 0 + 0 + (10)(20)(25)$

$\boxed{m[3,4] = 5000 \text{ with } u=3}$

④ $m[1,3]$ with $i=1$, $j=3$ $\because i<j$

$\therefore i \le u < j-1$

$\therefore 1 \le u \le 2$

$\therefore \boxed{u=1,2}$

$\therefore m[i,j] = \begin{cases} m[i,u] + m[u+1,j] + p_{i-1} p_u p_j & \text{for } u=1 \\ m[i,u] + m[u+1,j] + p_{i-1} p_u p_j & \text{for } u=2 \end{cases}$

$= \begin{cases} m[1,1] + m[2,3] + p_0 p_1 p_3 \\ m[1,2] + m[3,3] + p_0 p_2 p_3 \end{cases}$

$= \min \begin{cases} 0 + 1000 + (15)(5)(20) \\ 750 + 0 + (15)(10)(20) \end{cases}$

$= \min \begin{cases} 2500 \text{ for } u=1 \\ 3750 \text{ for } u=2 \end{cases}$

$\boxed{m[i,j] = 2500 \text{ with } u=1}$

⑤ $m[2,4]$ with $i=2$, $j=4$ $\because i<j$

$\therefore i \le u < j-1$

$\therefore 2 \le u < 3$

$\therefore \boxed{u=2,3}$

$\therefore m[i,j] = \min \begin{cases} m[i,u] + m[u+1,j] + p_{i-1} p_u p_j & \because u=2 \\ m[i,u] + m[u+1,j] + p_{i-1} p_u p_j & \because u=3 \end{cases}$

$= \min \begin{cases} m[2,2] + m[3,4] + p_1 p_2 p_4 \\ m[2,3] + m[4,4] + p_1 p_3 p_4 \end{cases}$

$= \min \begin{cases} 0 + 5000 + (5)(10)(25) \\ 1000 + 0 + (5)(20)(25) \end{cases}$

$= \min \begin{cases} 8250 \text{ with } u=2 \\ 3500 \text{ with } u=3 \end{cases}$

$\boxed{m[2,4] = 3500 \text{ with } u=3}$

⑥ $m[1,4]$ with $i=1$, $j=4$ $\because i<j$

$i \le u < j-1$

$1 \le u < 3$

$\therefore \boxed{u=1,3}$

$\therefore m[i,j] = \min \begin{cases} m[i,u] + m[u+1,j] + p_{i-1} p_u p_j & \because u=1 \\ m[i,u] + m[u+1,j] + p_{i-1} p_u p_j & \because u=3 \end{cases}$

$= \min \begin{cases} m[1,1] + m[2,4] + p_0 p_1 p_4 \\ m[1,3] + m[4,4] + p_0 p_3 p_4 \end{cases}$

$= \min \begin{cases} 0 + 3500 + (15)(5)(25) \\ 2500 + 0 + (15)(20)(25) \end{cases}$

$= \min \begin{cases} 5375 \text{ with } u=1 \\ 10,000 \text{ with } u=3 \end{cases}$

$\boxed{m[1,4] = 5375 \text{ with } u=1}$

$\therefore$

1	2	3	4	m
0	750	2500	5375	1
	0	1000	3500	2
		0	5000	3
			0	4

[m-table]

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

Ex-2] consider

matrix	Dimension
A_1	5×4
A_2	4×6
A_3	6×2
A_4	2×7

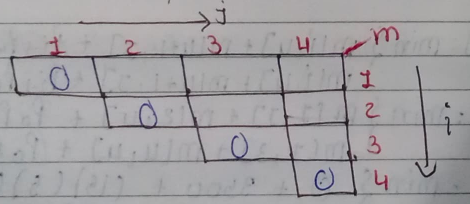
Compute the matrix chain order

Ans Let

$P_0 = 5$, $P_1 = 4$, $P_2 = 6$, $P_3 = 2$, $P_4 = 7$

$N = 5$

→ Now we draw matrix for size $(N-1) \Rightarrow (5-1) \Rightarrow 4$



→ Now when $i = j$ then $m[i,j] = 0$

$\therefore m[1,1] = m[2,2] = m[3,3] = m[4,4] = 0$

→ Now we compute the other cells using eqn $m[i,j] = m[i,u] + m[u+1,j] + P_{i-1} P_u P_j$ for $i \le u < j-1$

① $m[1,2]$ with $i=1, j=2$ $\because i<j$

$\therefore i \le u < j-1$

$\therefore 1 \le u < 1$

$\therefore \boxed{u=1}$

$\therefore m[i,j] = m[i,u] + m[u+1,j] + P_{i-1} P_u P_j$

$= m[1,1] + m[2,2] + P_0 P_1 P_2$

$= 0 + 0 + (5)(4)(6)$

$\boxed{m[i,j] = 120 \text{ with } u=1}$

② $m[2,3]$ with $i=2, j=3$ $\because i<j$

$\therefore i \le u < j-1$

$\therefore 2 \le u < 2$

$\therefore \boxed{u=2}$

$\therefore m[i,j] = m[i,u] + m[u+1,j] + P_{i-1} P_u P_j$

$= m[2,2] + m[3,3] + P_1 P_2 P_3$

$= 0 + 0 + (4)(6)(2)$

$\boxed{m[2,3] = 48 \text{ with } u=2}$

③ $m[3,4]$ with $i=3, j=4$ $\because i<j$

$\therefore i \le u < j-1$

$3 \le u < 3$

$\therefore \boxed{u=3}$

$\therefore m[i,j] = m[i,u] + m[u+1,j] + P_{i-1} P_u P_j$

$= m[3,3] + m[4,4] + P_2 P_3 P_4$

$= 0 + 0 + (6)(2)(7)$

$\boxed{m[3,4] = 84 \text{ with } u=3}$

④ $m[2,3]$ with $i=2, j=3$ $\because i<j$

$\therefore i \le u < j-1$

$\therefore 2 \le u \le 2$

$\therefore \boxed{u=2}$

⑤ m[1,3] with i=1, j=3 ∵ i<j

~~∴ i ≤ k ≤ j-1~~

∴ $1 \le k < 3-1$

$1 \le k < 2$

∴ k = 1, 2

$m[1,3] = \min \begin{cases} m[1,1] + m[2,3] + p_0 p_1 p_3 & \because k=1 \\ m[1,2] + m[3,3] + p_0 p_2 p_3 & \because k=2 \end{cases}$

$= \min \begin{cases} 0 + 48 + (5)(4)(2) \\ 120 + 0 + (5)(6)(2) \end{cases}$

$= \min \begin{cases} 88 \text{ with } k=1 \\ 180 \text{ with } k=2 \end{cases}$

$m[1,3] = 88$ with $k=1$

⑥ m[2,4] with i=2, j=4 ∵ i<j

∴ $i \le k < j-1$

∴ $2 \le k \le 3$

∴ k = 2, 3

$m[2,4] = \min \begin{cases} m[2,2] + m[3,4] + p_1 p_2 p_4 & \because k=2 \\ m[2,3] + m[4,4] + p_1 p_3 p_4 & k=3 \end{cases}$

$= \min \begin{cases} 0 + 84 + (4)(6)(7) \\ 48 + 0 + (4)(2)(7) \end{cases}$

$= \min \begin{cases} 252 \text{ with } k=2 \\ 104 \text{ with } k=3 \end{cases}$

$m[2,4] = 104$ with $k=3$

⑦ m[1,4] with i=1, j=4 ∵ i<j

∴ $i \le k < j-1$

∴ $1 \le k < 3$

∴ k = 1, 3

$m[i,j] = \min \begin{cases} m[i,k] + m[k+1,j] + p_{i-1} p_k p_j & \because k=1 \\ m[i,k] + m[k+1,j] + p_{i-1} p_k p_j & \because k=3 \end{cases}$

$= \min \begin{cases} m[1,1] + m[2,4] + p_0 p_1 p_4 \\ m[1,3] + m[4,4] + p_0 p_3 p_4 \end{cases}$

$= \min \begin{cases} 0 + 104 + (5)(4)(7) \\ 88 + 0 + (5)(2)(7) \end{cases}$

$= \min \begin{cases} 244 \text{ with } k=1 \\ 158 \text{ with } k=3 \end{cases}$

$m[1,4] = 158$ with $k=3$

∴ → m table

m	1	2	3	4
1	0	120	88	158
2		0	48	~~104~~
3			0	84
4				0

→ k table

m	1	2	3	4
1	1	1	1	3
2		2	2	3
3			3	3
4				4

Longest common subsequence.

Ex-1) find out lcs of A = {K, A, N, D, L, A, P} and B = {A, N, D, L}

Ans) lcs,

A[i]	K	A	N	D	L	A	P
B[j]	A	N	D	L			

→ Now we build C[i,j] table where C[i,0] ← 0 where i represents the rows and C[0,j] ← 0 where j represents the columns.

• Step-1

		0	1 (A)	2 (N)	3 (D)	4 (L)
0	0	0	0	0	0	0
1	K	0				
2	A	0				
3	N	0				
4	D	0				
5	L	0				
6	A	0				
7	P	0				

(i ↓, j →)

• Step-2

A[i]	K	A	N	D	L	A	P
B[j]	A	N	D	L			

AS A[i] ≠ B[j]
∴ if [C[0,1] ≥ C[1,0]) is true
i.e 0 ≥ 0

∴ C[1,1] = C[i-1, j] = C[0,1] = 0
∴ d[1,1] = ↑

• Step-3

A[i]	K	A	N	D	L	A	P
B[j]	A	N	D	L			

AS A[i] ≠ B[j]
∴ if (C[1,2] ≥ C[0,2]) is true
i.e 0 ≥ 0
∴ C[1,2] = C[i-1, j] = C[0,2] = 0
∴ d[1,2] = ↑

• Step-4

→ continue the following process till we get the result
→ After all the calculation the C[i,j] table is as follows.

		0	1 (A)	2 (N)	3 (D)	4 (L)
0	0	0	0	0	0	0
1	K	0	↑0	↑0	↑0	↑0
2	A	0	↖1	←1	←1	←1
3	N	0	↑1	↖2	←2	←2
4	D	0	↑1	↑2	↖3	←3
5	L	0	↑1	↑2	↑3	↖4
6	A	0	↖1	↑2	↑3	↑4
7	P	0	↑1	↑2	↑3	↑4

∴ Longest common subsequence = <A, N, D, L>

Ch-5 Greedy Algorithm:

Q-1) What are the difference b/w greedy approach and dynamic programming

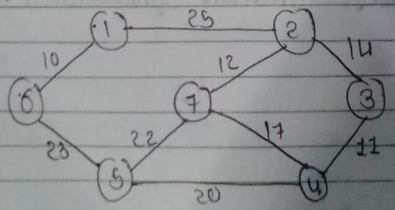
Ans

Greedy Approach	Dynamic Programming
→ Greedy method is used to get optimum solution	Dynamic method is also used to get optimum solution
→ In greedy method a set of feasible solutions and picks up the optimum solution.	there is no special set of feasible solution in this method.
→ In greedy method the optimum selection is without revising the previously generated result	→ Dynamic Programming considers all the possible sequences in order to get the optimum result
→ In greedy method there is no such guarantee to getting optimum solution	→ In Dynamic method there is such guarantee of getting optimum solution using Principle of optimality.

Minimum Spanning tree

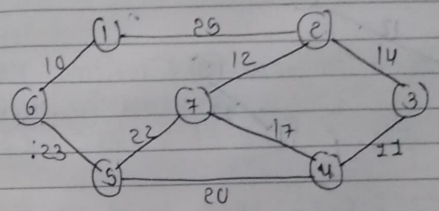
Prim's Algorithm — Kruskal's Algorithm

Ex-1 What is minimum spanning tree. Consider the given graph below. obtain minimum spanning tree using Prim's Algorithm



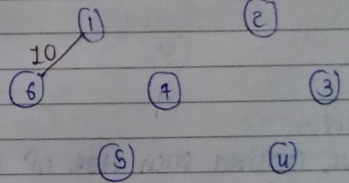
Ans A minimum spanning tree of a weighted graph G is a minimum spanning tree with minimum or smallest weight

→ We have following graph

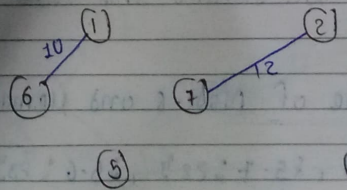


• We start from Node-1

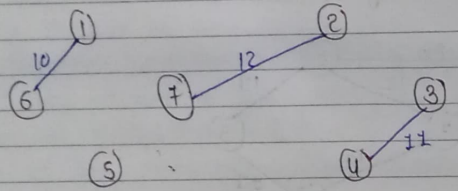
• Step-1 :-
We consider each & edge of Node-1.
{1-6 : 10}, {1-2 : 25}
Here the smallest edge is {1-6 : 10}, so we draw this Node first



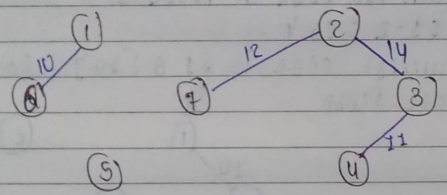
• Step-2 :-
We consider each edge of Node-2 and remaining edge of Node-1.
{1-2 : 25}, {2-3 : 14}, {2-7 : 12}



• Step-3
We consider each edge of Node-3 and remaining edge of Node-2
{1-2 : 25}, {2-3 : 14}, {3-4 : 11}



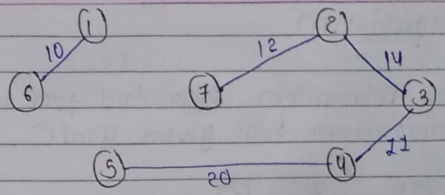
• Step-4
We consider each edge of Node-4 and remaining edge of Node-3
{1-2 : 25}, {2-3 : 14}, {4-7 : 17}, {4-5 : 20}



• Step-5
We consider each edge of Node-5 and remaining edge of Node-4
{1-2 : 25}, {4-7 : 17}, {4-5 : 20}, {5-7 : 22}, {5-6 : 23}

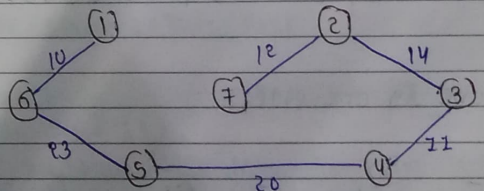
→ If we take {4-7 : 17} edge so there are cycle created in graph so we discard this edge

• Step-6
We consider each edge of Node-6 and remaining edge of Node-5
{1-2 : 25}, {4-5 : 20}, {5-7 : 22}, {5-6 : 23},



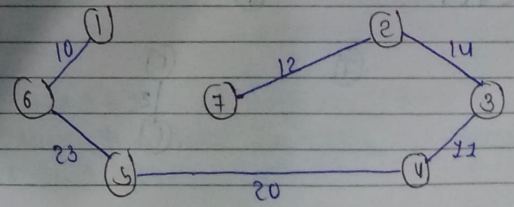
• Step-7
We consider each edge of Node-7 and remaining edge of Node-6
{1-2 : 25}, {5-7 : 22}, {5-6 : 23}

→ If we take edge {5-7 : 22}, so cycle is generated so we discard this edge
Now we have remaining edge,
{1-2 : 25}, {5-6 : 23}



• Step-8.
Remaining edge is {1-2 : 25}
If we take this edge so cycle is generated so we discard this edge
→ We have final minimum spanning tree as follows

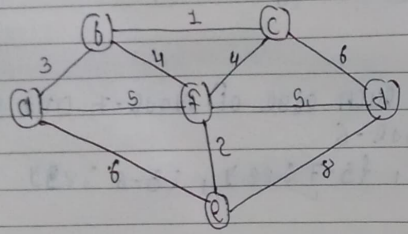
Ans ⇒



Kruskal's Algorithm

Pt-2) find out the minimum spanning tree using kruskal's algorithm for given graph

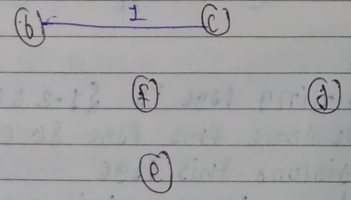
Ans



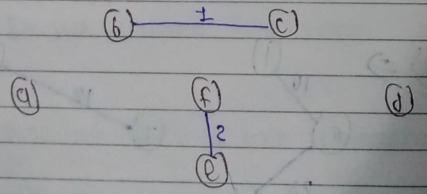
• Step-1:-
first we arrange all the edge in ascending order
{b-c : 1}, {f-e : 2}, {a-b : 3}, {b-f : 4}, {c-f : 4}, {a-f : 5}, {f-d : 5}, {a-e : 6}, {d-e : 8}

→ Now we take one by one edge

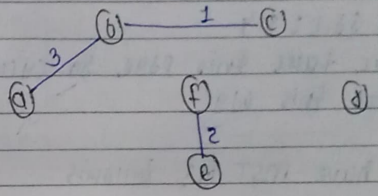
• Step-2 :-
{b-c : 1}



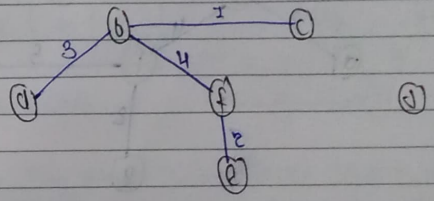
• Step-3 :-
take {f-e : 2}



• Step-4 :-
take {a-b : 3}



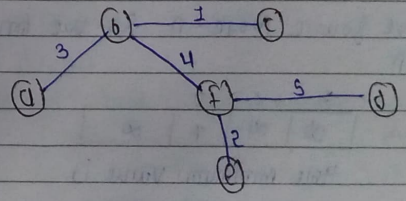
• Step-5 :-
take {b-f : 4}



• Step-6
take {c-f : 4},
if we take this edge so cycle is generated, so we discard this edge

• Step-7
take {a-f : 5},
if we take this edge so cycle is generated so we discard this edge

• Step-8
take {f-d : 5}

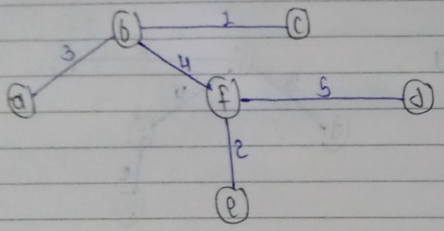


• Step-9.
take {a-e : 6},
if we take this edge so cycle is generated so we discard this edge

• Step-10
take $d-e$ i.e. 8

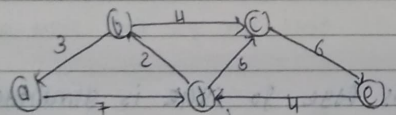
⇒ If we take this edge. So cycle is generated so we discard this edge.

⇒ We have MST as follows



Dijkstra's Algorithm:-

Ex-1) Solve the following instance of single-source shortest Path path Problem with vertex a as source

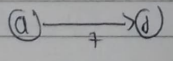


Ans 2

→ We have source Node-A. So we find shortest Path from A

A to →	B	C	D	E
	∞	∞	7	∞

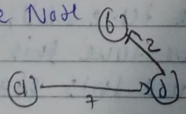
Here minimum value is 7 so we select Node-d.



→ Now we have Node-d as intermediate Node

A-d to →	B	C	D	E
	9	12	7	∞

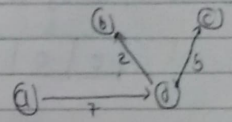
Here minimum value is 9 so we select Node B



→ Now we have Node d and b as intermediate Node

A-d-b to →	B	C	D	E
	9	12	7	∞

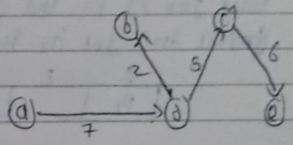
minimum value is 12 so we select Node-C



→ Now we have Node d, b and c as intermediate Node

A-d-b-C to →	B	C	D	E
	9	12	7	18

minimum value is 18 so we select Node E



- Thus from source a all other remaining vertices are:-

a to b	(a-d-b)	9
a to c	(a-d-c)	12
a to d	(a-d)	7
a to e	(a-d-c-e)	18

Knapsack Problem:-

→ Select object with maximum Profit
→ Select object with minimum weight
→ Select object with maximum P/w ratio

Job Sequencing Problem

Ex-2) using greedy algorithm find an optimal sequence for following job with n=7

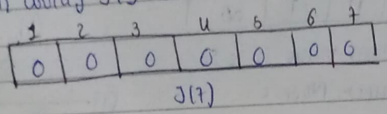
Profits: $(P_1, P_2, P_3, P_4, P_5, P_6, P_7) = (3, 5, 18, 20, 6, 1, 38)$

deadline: $(d_1, d_2, d_3, d_4, d_5, d_6, d_7) = (1, 3, 3, 4, 1, 2, 1)$

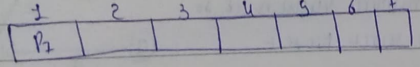
Ans Arrange the Profits in descending order

Profits	38	20	18	6	5	3	1
Job	P_7	P_4	P_3	P_5	P_2	P_1	P_6
deadline	1	4	3	1	3	1	2

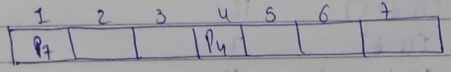
→ Step-2
- Create an array J[] which stores the job.



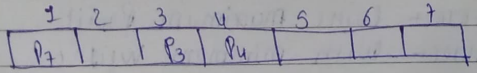
- Step-3
First job is P_7 The deadline for this job is 1. hence insert this job in the array J[] at 1st index



→ Step-4
Second job is P_4. The deadline for this job is 4. hence insert this job in the array J[] at 4th index

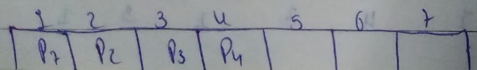


→ Step-5
Third job is P_3. The deadline for this job is 3 hence insert this job in the array J[] at 3rd index



→ Step-6
Fourth job is P_5. The deadline for this job is 1. But there at index 1 in the array J[] already one job is exists and there are no place at J[] < index so discard this job.

→ Step-7
Fifth job is P_2. The deadline for this job is 3. But at index 3 already one job is exists but at index J(3) one place is remaining so insert this job in the array J[] at 2nd position



② Net module :-

```
var net = required('net');

var server = net.createServer(function(connection) {
    console.log('client connected');
    connection.on('end', function() {
        console.log('client disconnected');
    });
    connection.write("Hello World");
    connection.pipe(connection);
});

server.listen(8080, function() {
    console.log('Server listining');
});
```

Node JS Global object.

- Node JS is a open source project and it is used as server-side scripting
- Node JS global objects are objects that are available in all modules

① Buffer class :- Buffer class object is used to represent binary data as a sequence of byte

Example

```
> const buffer = new Buffer.alloc(5, "abcde");
> console.log(buffer)
> <Buffer 61 62 63 64 65>
```

② console :- It is an inbuilt global object used to print stdout and stderr

```
> console.log("Hello World")
> Hello World
> console.error("An error occured")
> An error occured
```

③ Node JS Utility module

Briefly discuss various methods provided by OS Utility module

Node JS Utility module

```
var util = require('util');
```

os module :-

```
var os = require('os');
```

Example

```
var os = require('os');
console.log('Operating System type' + os.type());
console.log('Platform:' + os.platform());
console.log('total memory:' + os.totalmem());
console.log('Available memory:' + os.freemem());
```

④ Write a Node JS code to print a list of files present in the current directory

Ans

```
// Import file system module
const fs = require("fs");

fs.readdirsync(__dirname);
console.log("current directory file names");
filenames.forEach(file => {
    console.log(file)
});

fileobjs = fs.readdirsync(__dirname, {withFileType: true});
console.log("current directory file Names");
fileobjs.forEach(file => {
    console.log(file);
});
```

Callback function Example :-

```
var fs = require("fs");

console.log("serving user1");
fs.readFile('myfile.txt', function(err, data) {
    if(err) return console.error(err);
    console.log("In content of file is");
    console.log(data.toString);
});
console.log("serving user2");
console.log("serving user3");
```

① use of NPM

→ NPM is a package manager for Node.js package.
→ NPM stands for Node package manager

* Installing modules using NPM

→ for installing any Node.js module we have to use install command.
→ In command prompt write
[> npm install express.
→ for using this module in our program we have to use following command.

```
var ex = require('express');
```

→ Uninstalling module using NPM

→ for Uninstalling Node.js module we have to use Uninstall command in command prompt

> npm Uninstall express

→ var ex = require('express');

→ Uninstalling module using NPM

→ > npm Uninstall express.

② Custom filter.

```
<!DOCTYPE html>
<html>
<head>
<title> Demo </title>
<script src = "_____"></script>
</head>
<body>
<div ng-app = "myApp" ng-controller = "myController">
<ul>
<li ng-repeat = "x in names"> {{ x | myFormat }} </li>
</ul>
</div>
<script>
var app = angular.module('myApp', []);
app.filter('myFormat', function() {
    return function(x) {
        var i, c, txt = "";
        for (i=0 ; i < x.length ; i++)
        {
            c = x[i];
            if(i%2 == 0) {
                c = c.toUpperCase();
            }
            txt += c;
        }
        return txt;
    };
});
app.controller('myController', function($scope) {
    $scope.names = ['Yani', 'Rani', 'Ashraf'];
});
</script>
</body>
</html>
```

```
app.controller('myController', function ($scope) {
    $scope.price = 50;
});
</script>
</body>
</html>
```

⑤ filter filter

```
<!DOCTYPE html>
<html>
<head>
<title>Demo</title>
<script src = "_____"></script>
</head>
<body>
<div ng-app="myApp" ng-controller="myController">
<li ng-repeat="x in names | filter:'j'">
{{x}}
</li>
</ul>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myController', function ($scope) {
    $scope.names = ['Jani', 'Rima', 'Ritul', 'Rahul',
                    'Rishu'];
});
</script>
</body>
</html>
```

```
<title>Demo</title>
<script src = "_____"></script>
</head>
<body>
<div ng-app="myApp" ng-controller="myController">
<ul>
<li ng-repeat="x in names | orderBy:'country'">
{{x.name + ', ' + x.country}}
</li>
</ul>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myController', function ($scope) {
    $scope.names = [
        {name:'John', country:'America'},
        {name:'Rishu', country:'India'},
    ];
});
</script>
</body>
</html>
```

⑥ Currency filter

```
<!DOCTYPE html>
<html>
<head>
<title>Demo</title>
<script src="_____"></script>
</head>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
<p>Price: {{price | currency}}</p>
</div>
<script>
var app = angular.module('myApp', []);
```

```
<p> your name is : {{ lastname | uppercase }} </p>
</div>
<script>
var app = angular.module('myAPP', []);
app.controller('myController', function($scope) {
    $scope.lastname = "Darji";
    $scope.firstname = "Akshat";
</script>
</body>
</html>
```

② lowercase

```
<!DOCTYPE html>
<html>
<head>
<title> Demo </title>
<script src: "_____"></script>
</head>
<body>
<div ng-app: "myApp" ng-controller: "myController">
<p> your name is : {{ firstname | lowercase }} </p>
</div>
</body>
<script>
var app = angular.module('myAPP', []);
app.controller('myController', function($scope) {
    $scope.firstname = "AKSHAT";
    $scope.lastname = "DARJI";
</script>
</body>
</html>
```

③ OrderBy filter

```
<!DOCTYPE html>
<html>
<head>
<body>
<div ng-app="myAPP" ng-controller="myController">
<p> Search here </p>
<input type="text" ng-model="test">
<ul>
  <li ng-repeat="x in names | filter:'test'">{{x}}</li>
</ul>
</div>
<script>
var app = angular.module('myAPP', []);
app.controller('myController', function($scope) {
  $scope.names = [
      'Jani'
      'Amun'
      'Sachul'
  ];
});
</script>
</body>
</html>
```

Filters in Angular JS

① Uppercase filter

```
<!DOCTYPE html>
<html>
<head>
<title> Demo </title>
<script src="_____"></script>
</head>
<body>
<div ng-app:"myAPP" ng-controller="myController">
```

(1) Create a single page application for searching and requesting books from library

```
<!  ---------- adding books ---------->
<!DOCTYPE html>
<html>
<head>
<title>Demo</title>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/
        1.3.14/angular.min.js"></script>
</head>
<body>
<div ng-app="myApp" ng-controller="myController">
  <ul>
    <li ng-repeat="x in books">{{x}}</li>
  </ul>
  <input ng-model="addme">
  <button ng-click="addBooks()">addBooks</button>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myController', function($scope) {
    $scope.books = ["Java", "Python"];
    $scope.addBooks = function() {
        $scope.books.push($scope.addme);
    }
});
</script>
</body>
</html>

<!  ---------- Searching books ---------->
<!DOCTYPE html>
<html>
<head>
<title>Demo</title>
<script src="  ---------- "></script>
</head>
```

(2) Algorithm for Quick Search

```
quicksearch(A[0 — n-1], low, high)
{
    if (low < high) then
    m ← Partition(A[low — high]);
    Quick(A[low — m-1]);
    Quick(A[m+1 — high]);
}
Algorithm Partition(A[low — high])
{
    Pivot ← A[low];
    i ← A[low];
    j ← high+1;
    while(i<=j) do
    {
        while(A[i] <= Pivot) do
        i = i+1;
        while(A[j] >= Pivot) do
        j = j-1;
        if(i<=j) then
        swap(A[i], A[j]);
    }
    swap(A[low], A[j]);
    return j;
}
```

Best-case : $O(n \log_2 n)$
Worst-case : $O(n^2)$
Average case : $O(n \log n)$

```
while ( A[i] <= mid AND A[j] <= high )
{
    if ( A[i] <= A[j] ) then
    {
        temp[k] ← A[i];
        i ← i+1;
        k ← k+1;
    }
    else
    {
        temp[k] ← A[j];
        j ← j+1;
        k ← k+1;
    }
}
while ( i <= mid ) do
{
    temp[k] = A[i];
    i ← i+1;
    k ← k+1;
}
while ( j <= high ) do
{
    temp[k] = A[j];
    j ← j+1
    k ← k+1;
}
}
```

- Best case : $O(n \log_2 n)$
- Worst case : $O(n \log_2 n)$
- Average case : $O(n \log_2 n)$