Dynamic Programming

Q.1 Explain Dynamic Programming with example.

⇒ i) Dynamic programming is mainly an optimization over plain recurssion.

ii) Whenever we see a recursion solution that has repeated calls for same inputs, we can optimize it using DP.

iii) The idea is simply store the results of subproblems, so that we do not have to re-compute them when needed later.

iv) This simple optimization reduces the time complexity from exponential to polynomial.

v) DP is a bottom-up approach.

vi) It starts solving the smallest possible problem and uses a solution of smaller problem to build a solution of larger problem.

vii) Applications of DP -

a) TSP
b) Assembly line scheduling
c) Knapsack
d) Multi-stage graph, etc.

Q.2 Explain multistage graph with example.

⇒ i) Multistage graph is a directed graph in which the nodes can be divided into set of stages such as all edges from a stage to next stage only.

ii) In other words, there is no edge between vertices of same stage and from vertex of current stage to previous stage.

- Algorithm :

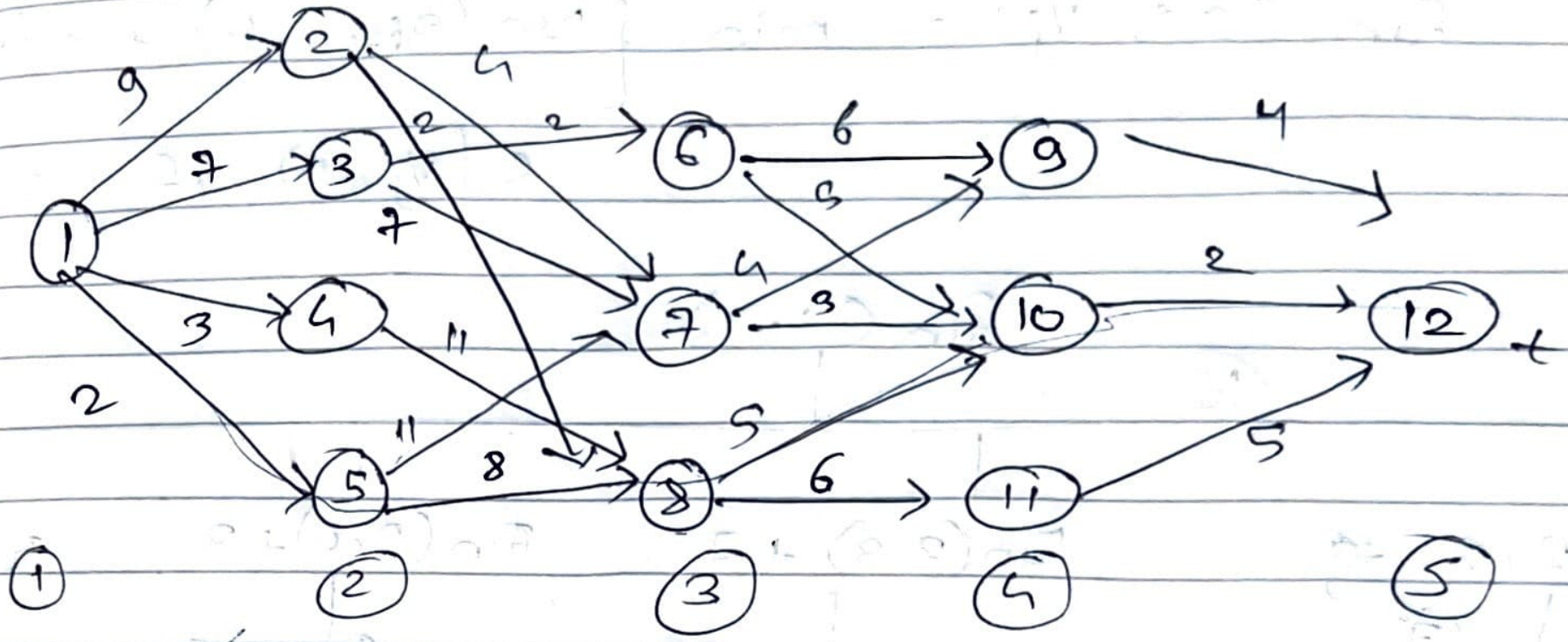
```
MULTI_STAGE (G, k, n, p)
{
  // c[i,j] : cost of edge (i, j)

  cost[n] ← 0

  for  j ← n-1 to 1 do

      cost[j] ← c[j, r] + cost[r]
      Π[j] ← r

  p[1] ← 1
  p[k] ← n

  for  j ← 2 to k-1 do
      p[j] ← Π[p[j-1]]

}
```

Time complexity : $O(n+|E|)$

8 1 9 1 2 1 2

17

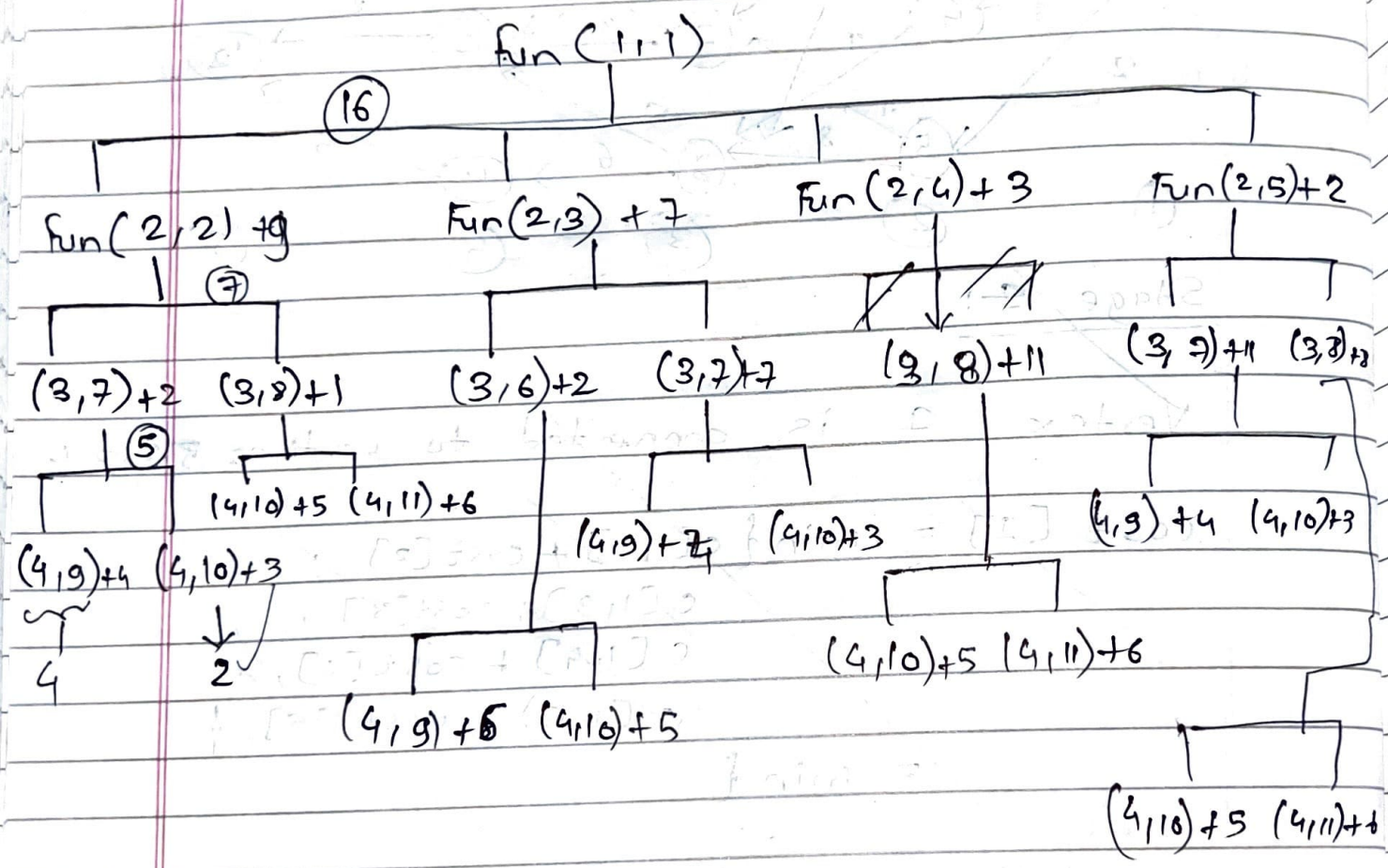
For e.g.:-



- Multistage graph

⇒ $Fun(s_i, v_j) = \min \{ Fun(s_{i+1}, k) + c(v_j, k) \}$

$c(v_j, D)$ if $s_i = s_{n-1}$



Cost of path = 16

path = 1 - 2 - 7 - 10 - 12

Q.3 All Pair Shortest Path

⇒ • Problem :

Let $G = \langle V, E \rangle$ be a directed graph, where V is set of vertices and E is set of edges with non-negative length.
L = matrix, which gives length of each ~~node~~ edge.

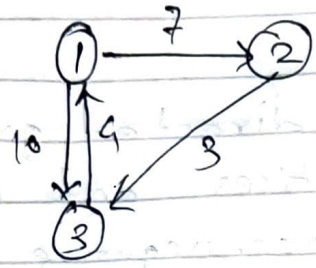
$L[i,j] = 0$, if $i = j$
$L[i,j] = \infty$, if $i \neq j$ and $(i,j) \notin E$
$L[i,j] = w(i,j)$, if $i \neq j$ and $(i,j) \in E$

• Algorithm :

```
FLOYD (L)
{
    D ← L

    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                D[i,j]^k ← min( D[i,j]^(k-1),
                                D[i,k]^(k-1) + D[k,j]^(k-1) )

    return D
}
```

• Complexity Analysis : $T(n) = \Theta(n^3)$

For e.g.,



$\Rightarrow$

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

$\therefore D^0 =$

	1	2	3
1	0	7	10
2	∞	0	3
3	4	∞	0

Iteration 1 ($k=1$) :

$$D^1[1,2] = \min\{ D^0[1,2],\ D^0[1,1] + D^0[1,2] \}$$
$$= \min\{ 7,\ 0+7 \}$$
$$= 7$$

$D^1 =$

0	7	10
∞	0	3
4	11	0

Iteration 2 ($k=2$) :

$D^2 =$

0	7	10
∞	0	3
4	11	0

$D^3 =$

0	7	10
7	0	3
4	11	0

//

Q.4 Assembly line Scheduling

i) Assembly line scheduling is a manufacturing problem.

ii) In automobile industry, assembly lines are used to transfer parts from one station to another station.

iii) Manufacturing of large items like cars, trucks, etc undergoes through multiple stations, where each station is responsible for assembling particular part only.

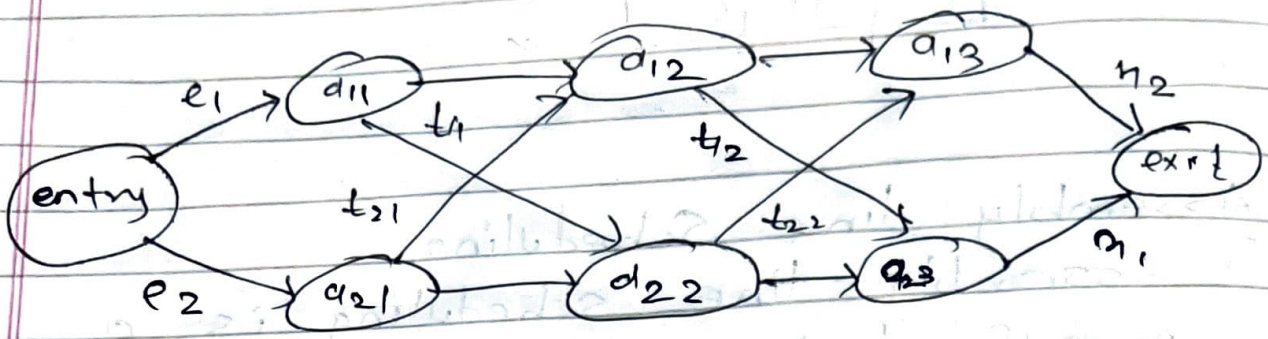
iv) Entire product be ready after it goes through predefined n stations in sequence.

v) Manufacturing of cars may be done through several stages like fitting of engine, coloring, light fitting, seats and many other things.

vi) The particular task is carried out at the station dedicated to that task only.

vii) Based on the requirement there may be more than one assembly lines.

- The architecture of assembly lines ⇒



Where,

e_i = Entry time

x_i = Exit time

t_{ij} = time required to transfer componer from one assembly to other.

Q.5 0/1 Knapsack

⇒

Fractional Knapsack	0/1 Knapsack
i) It is a part of greedy algorithm.	i) It is a part of dynamic programing
ii) It ensures the optimal solution	ii) It does not ensures optimal solution.
iii) In this problem, we take Fraction of item.	iii) This problem either takes on item ~~or~~ ~~not~~ completely but doesn't take Fraction.
iv) Items can be selected partially	iv) Breaking of item is not allowed.
v) In this, finds a most valuable subset item with a total value equal to weight	v) In this, finds most valuable subsal item with total less than equal to weight
vi) The Fractional knapsack do have greedy choice property.	vi) 0/1 Knapsack do not have greedy choice property.

- **Algorithm :**

```
0/1Knapsack (V, W, M)
{
    for i ← 1 to n do
        V[i,0] ← 0

    for i ← 1 to M do
        V[0,i] ← 0

    for i ← 1 to n do
        for j ← 0 to M do

            if W[i] ≤ j
                V[i,j] ← max{V[i-1,j] ,
                             V[i] + V[i-1, j-W[i]]}
            else
                V[i,j] ← V[i-1, j]

}
```

For e.g.; $N = 4, M = 8$

$(P_1, P_2, P_3, P_4) = (1, 2, 5, 6)$

$(W_1, W_2, W_3, W_4) = (2, 3, 4, 5)$

P	W	0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1	1
2	3	2	0	0	1	2	2	3	3	3
5	4	3	0	0	1	2	5	5	6	7
6	5	4	0	0	1	2	5	6	6	7

$$V[i, w] = \max \{ V[i-1, w], V[i-1, w-W_i] + P[i] \} \quad w \geq w_i$$

$$V[4, 1] = \max \{ V[3, 1], V[3, 1-5] + 6 \}$$
$$= 0$$

$$V[4, 5] = \max \{ V[3, 5], V[3, 0] + 6 \}$$
$$= \max \{ 5, 6 \}$$
$$= 6$$

$$P[4, 7] = \max \{ V[3, 7], V[3, 2] + 6 \}$$
$$= \max \{ 7, 1+6 \}$$
$$= 7$$

- x_1 x_2 x_3 x_4

  0 1 0 1

Earned Profit $= P_2 + P_4$

$= 2 + 6$

$= 8$ //

Q.6 LCS

⇒ • LCS problem defined as –

$$LCS[i,j] = \begin{cases} 0 & i=0 \mid j=0 \\ LCS[i-1, j-1]+1 & a_i = b_j \\ \max\{LCS[i,j-1], LCS[i-1,j]\} & a_i \neq b_j \end{cases}$$

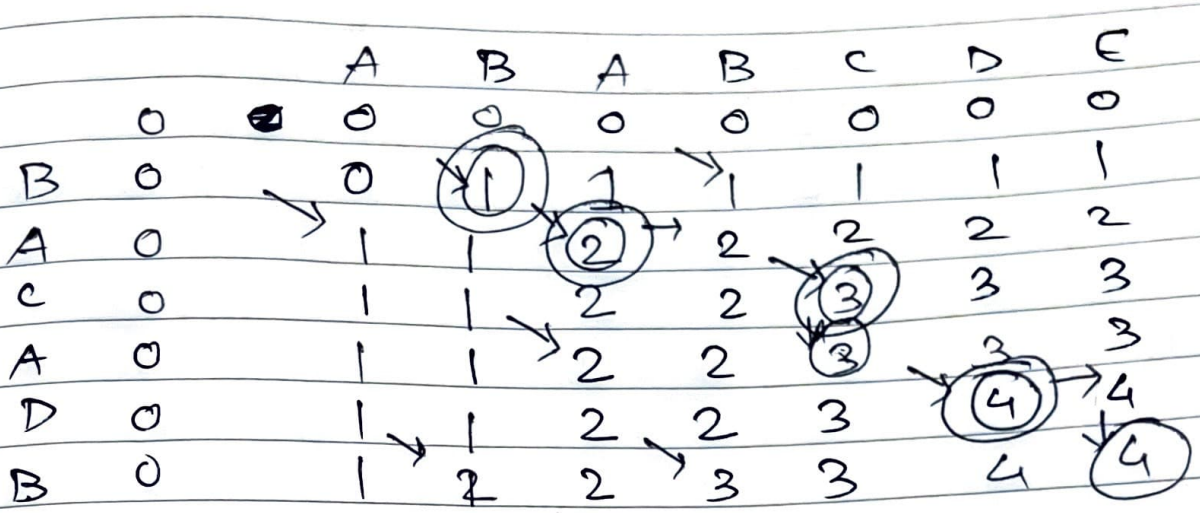
• $T(n) = O(mn)$

• Algorithm:

```
LCS (X, Y)
{
    for i ← 1 to m do
        LCS[i,0] ← 0

    for j ← 0 to n do
        LCS[0,j] ← 0

    for i ← 1 to m do
        for j ← 01 to n do

            if Xi == Yj then
                LCS[i,j] ← LCS[i-1, j-1] + 1

            else
                if LCS[i-1, j] ≥ LCS[i, j-1] then
                    LCS[i,j] ← LCS[i-1, j]
                else
                    LCS[i,j] ← LCS[i, j-1]

    return LCS
}
```

For e.g.:

X = ABABCDE
Y = BACADB



BACD