UNIT - III

5 Dynamic Programming

Syllabus

General strategy, Principle of optimality, 0/1 knapsack Problem, Coin change-making problem, Bellman - Ford Algorithm, Multistage Graph problem(using Forward computation), Travelling Salesman Problem

Contents

5.1 General Strategy ..... **Dec.-06, 07,08, 11, 18, May-19,** ..... Marks 6

5.2 Principle of Optimality ..... **Dec.-06, 07,13, May-15, April-18, March-20** ..... Marks 17

5.3 Elements of Dynamic Programming ..... **Dec.-06,07, May-07,08,10,** ..... Marks 6

5.4 Applications of Dynamic Programming

5.5 The 0/1 Knapsack Problem ..... **May-07,08,15,17,18, Dec.-07,11,12, April-18,** ..... Marks 10

5.6 Coin Change-making Problem ..... **April-18,** ..... Marks 5

5.7 Bellman - Ford Algorithm ..... **May-19, April-18,** ..... Marks 10

5.8 Multistage Graph Problem ..... **May-18, April-10,** ..... Marks 10

5.9 Travelling Salesman Problem ..... **May-08,12,13,15, 18, Dec.-11, 18,** ..... Marks 10

5.10 Multiple Choice Questions

5.1 General Strategy

SPPU : Dec.-06, 07,08, 11,18, End Sem, May-19, End Sem., Marks 6

Dynamic programming is typically applied to optimization problems.

For each given problem, we may get any number of solutions from which we seek for optimum solution (i.e. minimum value or maximum value solution). And such an optimal solution becomes the solution to the given problem.

5.1.1 Comparison with other Algorithmic Strategies

Divide and Conquer and Dynamic Programming

Sr. No.	Divide and Conquer	Dynamic Programming
1.	The problem is divided into small subproblems. These subproblems are solved independently. Finally all the solutions of sub-problems are collected together to get the solution to the given problem.	In dynamic programming many decision sequences are generated and all the overlapping sub-instances are considered.
2.	In this method duplications in sub-solutions are neglected. i.e., duplicate subsolutions may be obtained.	In dynamic computing duplications in solutions is avoided totally.
3.	Divide and conquer is less efficient because of rework on solutions.	Dynamic programming is efficient than divide and conquer strategy.
4.	The divide and conquer uses top down approach of problem solving (recursive methods).	Dynamic programming uses bottom up approach of problem solving (iterative method).
5.	Divide and conquer splits its input at specific deterministic points usually in the middle.	Dynamic programming splits its input at every possible split points rather than at a particular point. After trying all split points it determines which split point is optimal.

Greedy Algorithm Vs. Dynamic Programming

In this section we will discuss *"What are the differences and similarities between Greedy algorithm and dynamic programming ?"*

Sr. No.	Greedy method	Dynamic programming
1.	Greedy method is used for obtaining **optimum solution.**	Dynamic programming is also for obtaining **optimum solution.**
2.	In Greedy method a set of **feasible solutions** is obtained and the picks up the optimum solution.	There is **no** special set of **feasible solutions** in this method.
3.	In Greedy method the optimum selection is **without revising previously** generated solutions.	Dynamic programming considers **all possible sequences** in order to obtain the optimum solution.
4.	In Greedy method there is **no** as such **guarantee** of getting **optimum solution.**	It is guaranteed that the dynamic programming will generate optimal solution using **principle of optimality.**

5.1.2 Steps of Dynamic Programming

Dynamic programming design involves 4 major steps :

1. Characterize the structure of optimal solution. That means develop a mathematical notation that can express any solution and subsolution for the given problem.
2. Recursively define the value of an optimal solution.
3. By using bottom up technique compute value of optimal solution. For that you have to develop a recurrence relation that relates a solution to its subsolutions, using the mathematical notation of step 1.
4. Compute an optimal solution from computed information.

Ex. 5.1.1 *Comment on the statement : "In dynamic programming, many decision sequences may be generated".* **SPPU : Dec.-08, Marks 2**

Sol. : Dynamic programming is typically applied to optimization problems for a given problem we may get any number of solutions. From all those solutions we seek for optimum solution and such an optimum solution becomes the solution to the given problem. In this method subproblem is solved only once. The result of each subproblem is recorded in a table from which we can obtain a solution to the original problem. Hence in dynamic programming many decision sequences are generated and all the overlapping sub instances are considered.

Review Questions

Q.1 What is the major difference between greedy method and dynamic programming ?
SPPU : Dec.-06, 07, Marks 2, Dec.-18, End Sem, May-19, End Sem., Marks 3

Q.2 What are the common step in the dynamic programming to solve any problem ? Compare dynamic programming with greedy approach. **SPPU : Dec.-11, Marks 6**

5.2 Principle of Optimality

SPPU : Dec.-06, 07,13, May-15, Marks 17, April-18, March-20

The dynamic programming algorithm obtains the solution using principle of optimality.

The principle of optimality states that "in an optimal sequence of decisions or choices, each subsequence must also be optimal."

When it is not possible to apply the principle of optimality it is almost impossible to obtain the solution using the dynamic programming approach.

For example : Finding of shortest path in a given graph uses the principle of optimality.

Review Questions

Q.1 What is the "Principle of optimality" ?
SPPU : Dec.-06, 07, April 18, March-20, In Sem. Marks 2

Q.2 What is dynamic programming ? Is this the optimization technique ? Give reasons. What are its drawbacks ? Explain memory functions. **SPPU : Dec.-13, Marks 17**

Q.3 What is principle of optimality ? Differentiate between greedy and dynamic method.
SPPU : May-15 [End sem] Marks 5

5.3 Elements of Dynamic Programming

SPPU : Dec.-06,07, May-07,08,10, Marks 6

1) **Optimal substructure** - The dynamic programming technique makes use of principle of optimality to find the optimal solution from subproblems.

2) **Overlapping subproblems** - The dynamic programming is a technique in which the problem is divided into subproblems. The solutions of subproblems are shared to get the final solution to the problem. It avoids repetition of work and we can get the solution more efficiently.

Review Question

Q.1 Name the elements of dynamic programming. How does the dynamic programming solve the problem ? **SPPU : Dec.-06, 07, May-07, 08, Marks 6, May-10, Marks 4**

5.4 Applications of Dynamic Programming

Various applications that make use of dynamic programming strategy are -

1. The 0/1 Knapsack problems
2. Bellman Ford Algorithm
3. Multistage Graph Problem
4. Optimal Binary Search Trees
5. Traveling Salesman Problem

5.5 The 0/1 Knapsack Problem

SPPU : May - 07,08,15,17, 18, Dec.-07,11,12, April-18, Marks 10

As we know that dynamic programming is a technique for solving problems with overlapping subproblems. These subproblems are typically based on recurrence relation. In this section we will discuss the method of solving **knapsack problem** using dynamic programming approach. The knapsack problem can be defined as follows : If there are n items with the weights w_1, w_2, ... w_n and values (profit associated with each item) $v_1, v_2, \ldots v_n$ and capacity of knapsack to be W, then find the most valuable subset of the items that fit into the knapsack.

To solve this problem using dynamic programming we will write the recurrence relation as :

$$\text{table}[i, j] = \text{maximum}\ \{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\} \text{ if } j \geq w_i$$

$$\text{table}[i-1, j] \text{ if } j < w_i$$

That means, a table is constructed using above given formula. Initially, table [0, j] = 0 as well as table [i, 0] = 0 when $j \geq 0$ and $i \geq 0$.

The initial stage of the table can be

	0	1		$j - w_i$		j		W
0	0	0	...	0	...	0	...	0
	:			table [i – 1. j – w_i]				
i-1	0					table [i – 1, j]		
i	0					table [i, j]		
	:							
n	0							table [n, W]

Goal i.e., maximum value of items

The table [n, W] is a goal i.e., its gives the total items sum of all the selected items for the knapsack.

From this goal value the selected items can be traced out. Let us solve the knapsack problem using the above mentioned formula -

Ex. 5.5.1 *For the given instance of problem obtain the optimal solution for the knapsack problem.*

Item	Weight	Value
1	2	3
2	3	4
3	4	5
4	5	6

The capacity of knapsack is W = 5.

Sol.: Initially, table [0, j] = 0 and table [i, 0] = 0. There are 0 to n rows and 0 to W columns in the table.

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

Now we will fill up the table either row by row or column by column. Let us start filling the table row by row using following formula :

$$\text{table}[i, j] = \begin{cases} \text{maximum } \{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i] \text{ when } j \geq w_i\} \\ \text{or} \\ \text{table}[i-1, j] \text{ if } j < w_i \end{cases}$$

table [1, 1] With i = 1, j = 1, $w_i = 2$ and $v_i = 3$.

As $j < w_i$ we will obtain table [1, 1] as

$$\text{table}[1, 1] = \text{table}[i-1, j]$$
$$= \text{table}[0, 1]$$

$\therefore$ **table [1, 1] = 0**

table [1, 2] With i = 1, j = 2, $w_i = 2$ and $v_i = 3$

As $j \geq w_i$ we will obtain table [1, 2] as

$$\text{table}[1, 2] = \text{maximum } \{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \text{maximum } \{(\text{table}[0, 2]), (3 + \text{table}[0, 0]\}$$
$$= \text{maximum } \{0, 3 + 0\}$$

$\therefore$ **table [1, 2] = 3**

table [1, 3] With i = 1, j = 3, $w_i = 2$ and $v_i = 3$

As $j \geq w_i$ we will obtain table [1, 3] as

$$\text{table}[1, 3] = \text{maximum } \{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \text{maximum } \{\text{table}[0, 3], 3 + \text{table}[0, 1]\}$$
$$= \text{maximum } \{0, 3 + 0\}$$

$\therefore$ **table [1, 3] = 3**

table [1, 4] With i = 1, j = 4, $w_i = 2$ and $v_i = 3$

As $j \geq w_i$, we will obtain table [1, 4] as

$$\text{table}[1, 4] = \text{maximum } \{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \text{maximum } \{\text{table}[0, 4], 3 + \text{table}[0, 2]\}$$
$$= \text{maximum } \{0, 3 + 0\}$$

$\therefore$ **table [1, 4] = 3**

table [1, 5] With i = 1, j = 5, $w_i = 2$ and $v_i = 3$

As $j \geq w_i$ we will obtain table [1, 5] as

$$\text{table}[1, 5] = \text{maximum } \{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \text{maximum } \{\text{table}[0, 5], 3 + \text{table}[0, 3]\}$$
$$= \text{maximum } \{0, 3 + 0\}$$

$\therefore$ **table [1, 5] = 3**

The table with these values can be

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

Now let us fill up next row of the table.

table [2, 1] With i = 2, j = 1, $w_i = 3$ and $v_i = 4$

As $j < w_i$, we will obtain table [2, 1] as

$$\text{table}[2, 1] = \text{table}[i-1, j]$$
$$= \text{table}[1, 1]$$

$\therefore$ **table [2, 1] = 0**

table [2, 2] With $i = 2, j = 2, w_i = 3$ and $v_i = 4$

As $j < w_i$, we will obtain table [2, 2] as

$$\text{table } [2, 2] = \text{table } [i - 1, j]$$
$$= \text{table } [1, 2]$$

$\therefore$ **table [2, 2] = 3**

table [2, 3] With $i = 2, j = 3, w_i = 3$ and $v_i = 4$

As $j \geq w_i$, we will obtain table [2, 3] as

$$\text{table } [2, 3] = \text{maximum } \{\text{table } [i - 1, j], v_i + \text{table } [i - 1, j - w_i]\}$$
$$= \text{maximum } \{\text{table } [1, 3], 4 + \text{table } [1, 0]\}$$
$$= \text{maximum } \{3, 4 + 0\}$$

$\therefore$ **table [2, 3] = 4**

table [2, 4] With $i = 2, j = 4, w_i = 3$ and $v_i = 4$

As $j \geq w_i$, we will obtain table [2, 4] as

$$\text{table } [2, 4] = \text{maximum } \{\text{table } [i - 1, j], v_i + \text{table } [i - 1, j - w_i]\}$$
$$= \text{maximum } \{\text{table } [1, 4], 4 + \text{table } [1, 1]\}$$
$$= \text{maximum } \{3, 4 + 0\}$$

$\therefore$ **table [2, 4] = 4**

table [2, 5] With $i = 2, j = 5, w_i = 3$ and $v_i = 4$

As $j \geq w_i$, we will obtain table [2, 5] as

$$\text{table } [2, 5] = \text{maximum } \{\text{table } [i - 1, j], v_i + \text{table } [i - 1, j - w_i]\}$$
$$= \text{maximum } \{\text{table } [1, 5], 4 + \text{table } [1, 2]\}$$
$$= \text{maximum } \{3, 4 + 3\}$$

$\therefore$ **table [2, 5] = 7**

The table with these computed values will be

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0					
4	0					

table [3, 1] With $i = 3, j = 1, w_i = 4$ and $v_i = 5$

As $j < w_i$, we will obtain table [3, 1] as

$$\text{table } [3, 1] = \text{table } [i - 1, j]$$
$$= \text{table } [2, 1]$$

$\therefore$ **table [3, 1] = 0**

table [3, 2] With $i = 3, j = 2, w_i = 4$ and $v_i = 5$

As $j < w_i$, we will obtain table [3, 2] as

$$\text{table } [3, 2] = \text{table } [i - 1, j]$$
$$= \text{table } [2, 2]$$

$\therefore$ **table [3, 2] = 3**

table [3, 3] with $i = 3, j = 3, w_i = 4$ and $v_i = 5$

As $j < w_i$, we will obtain table [3, 3] as

$$\text{table } [3, 3] = \text{table } [i - 1, j]$$
$$= \text{table } [2, 3]$$

$\therefore$ **table [3, 3] = 4**

table [3, 4] With $i = 3, j = 4, w_i = 4$ and $v_i = 5$

As $j \leq w_i$, we will obtain table [3, 4] as

$$\text{table } [3, 4] = \text{maximum } \{\text{table } [i - 1, j], v_i + \text{table } [i - 1, j - w_i]\}$$
$$= \text{maximum } \{\text{table } [2, 4], 5 + \text{table } [2, 0]\}$$
$$= \text{maximum } \{4, 5 + 0\}$$

$\therefore$ **table [3, 4] = 5**

table [3, 5] With $i = 3, j = 5, w_i = 4$ and $v_i = 5$

As $j \geq w_i$, we will obtain table [3, 5] as

$$\text{table } [3, 5] = \text{maximum } \{\text{table } [i - 1, j], v_i + \text{table } [i - 1, j - w_i]\}$$
$$= \text{maximum } \{\text{table } [2, 5], 5 + \text{table } [2, 1]\}$$
$$= \text{maximum } \{7, 5 + 0\}$$

$\therefore$ **table [3, 5] = 7**

The table with these values can be

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0					

table [4, 1] With $i = 4, j = 1, w_i = 5, v_i = 6$

As $j < w_i$, we will obtain table [4, 1] as

$$\text{table } [4, 1] = \text{table } [i - 1, j]$$

$$= \text{table } [3, 1]$$

$\therefore$ table [4, 1] = 0

table [4, 2] With $i = 4, j = 2, w_i = 5$ and $v_i = 6$

As $j < w_i$, we will obtain table [4, 2] as

$$\text{table } [4, 2] = \text{table } [i - 1, j]$$

$$= \text{table } [3, 2]$$

$\therefore$ table [4, 2] = 3

table [4, 3] With $i = 4, j = 3, w_i = 5$ and $v_i = 6$

As $j < w_i$, we will obtain table [4, 3] as

$$\text{table } [4, 3] = \text{table } [i - 1, j]$$

$$= \text{table } [3, 3]$$

$\therefore$ table [4, 3] = 4

table [4, 4] with $i = 4, j = 4, w_i = 5$ and $v_i = 6$

As $j < w_i$, we will obtain table [4, 4] as

$$\text{table } [4, 4] = \text{table } [i - 1, j]$$

$$= \text{table } [3, 4]$$

$\therefore$ table [4, 4] = 5

table [4, 5] With $i = 4, j = 5, w_i = 5$ and $v_i = 6$

As $j \geq w_i$, we will obtain table [4, 5] as

$$\text{table } [4, 5] = \text{maximum } \{\text{table } [i - 1, j], v_i + \text{table } [i - 1, j - w_i]\}$$

$$= \text{maximum } \{\text{table } [3, 5], 6 + \text{table } [3, 0]\}$$

$$= \text{maximum } \{7, 6 + 0\}$$

$\therefore$ table [4, 5] = 7

Thus the table can be finally as given below

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

This is the total value of selected items

How to find actual knapsack items ?

Now, as we know that table [n, W] is the total value of selected items, that can be placed in the knapsack. Following steps are used repeatedly to select actual knapsack item

```
Let, i = n and k = W then
while (i>0 and k>0)
{
    if(table [i,k] ≠ table[i-1,k])then
    mark i^th item as in knapsack
    i = i-1 and k=k- w_i          //selection of i^th item
  else
    i = i-1  //do not select i^th item
}
```

Let us apply these steps to the problem given in example 5.5.1. As we have obtained the final table –

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

→ Start from here

i = 4 and k = 5

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

i.e., table [4, 5] = table [3, 5]

∴ do not select i^{th} i.e., 4^{th} item.

Now set $i = i - 1$

$i = 3$

Capacity →

items ↓	0	1	2	3	4	5
0	0	0	0	0	0	0
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

As table [i, k] = table [i – 1, k]

i.e., table [3, 5] = table [2, 5]

do not select i^{th} item i.e., 3^{rd} item.

Now set i = i – 1 = 2

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

As table [i, k] ≠ table [i – 1, k]

i.e. table [2, 5] ≠ table [1, 5]

select i^{th} item.

That is, select 2^{nd} item.

Set $i = i - 1$ and $k = k - w_i$

i.e. $i = 1$ and $k = 5 - 3 = 2$

	0	1	2	3	4	5
0	0	0	0	0	0	0
✓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

As table [i, k] ≠ table [i – 1, k]

i.e. table [1, 2] ≠ table [0, 2]

select i^{th} item.

That is select 1^{st} item.

Set $i = i - 1$ and $k = k - w_i$

i.e. $i = 0$ and $k = 2 - 2 = 0$

Thus we have selected **item 1** and **item 2** for the knapsack. This solution can also be represented by solution vector **(1, 1, 0, 0).**

Let us now discuss the algorithm for the knapsack problem using dynamic programming.

Ex. 5.5.2 *Using dynamic programming, solve the following knapsack instance :*
$n = 3$, $[\omega_1, \omega_2, \omega_3] = [1, 2, 2]$ *and* $[P_1, P_2, P_3] = [18, 16, 6]$ *and* $M = 4$.

Sol. :

Item	Weight	Value
1	1	18
2	2	16
3	2	6

The capacity of knapsak is M = 4. Initially table [0, j] and table [i, 0] = 0

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

$$\text{table } [i, j] = \begin{cases} \text{maximum } \{\text{table } [i-1, j], V_i + \text{table } [i-1, j-W_i]\} \\ \text{or} \\ \text{table } [i-1, j] \text{ if } j < W_i \end{cases}$$

table [1, 1] With i = 1, j = 1, W_i = 2 and V_i = 3.

As $j < W_i$ we will obtain table [1, 1] as

table [1, 1] = max {table [0, 1], 18 + table [0, 0]}

= max {0, 18}

∴ **table [1, 1] = 18**

table [1, 2] With i = 1, j = 2, W_i = 1 and V_i = 18.

table [1, 2] = max {table [0, 2], 18 + table [0, 1]

= max {0, 18 + 0}

∴ **table [1, 2] = 18**

table [1, 3] With i = 1, j = 3, W_i = 1 and V_i = 18.

table [1, 3] = max {table [0, 3], 18 + table [0, 2]

= max {0, 18 + 0}

∴ **table [1, 3] = 18**

table [1, 4] With i = 1, j = 4, W_i = 1 and V_i = 18

table [1, 4] = max {table [0, 4], 18 + table [0, 3]

= max {0, 18 + 0}

∴ **table [1, 4] = 18**

	0	1	2	3	4
0	0	0	0	0	0
1	0	18	18	18	18
2	0				
3	0				

Consider table [2, 1] with i = 2, j = 1, W_i = 2, V_i = 16

As $j < W_i$

table [i, j] = table [i – 1, j]

table [2, 1] = table [1, 1]

table [2, 1] = 18

table [2, 2] with i = 2, j = 2, W_i = 2, V_i = 16

As $j > W_i$

table [2, 2] = max {table [i – 1, j], V_i + table [i – 1, j – W_i]}

= max {table [1, 2], 16 + table [1, 0]}

= max [18, 16 + 0]

table [2, 2] = 18

table [2, 3] with i = 2, j = 3, W_i = 2, V_i = 16

table [2, 2] = 18 + 16 = 34

table [2, 4] with i = 2, j = 4, W_i = 2, V_i = 16

table [2, 4] = 34

	0	1	2	3	4
0	0	0	0	0	0
1	0	18	18	18	18
2	0	18	18	34	34
3	0				

table [3, 1] with i = 3, j = 1, W_i = 2, V_i = 6

As $j < W_i$

table [i, j] = table [i – 1, j]

table [3, 1] = table [2, 1]

table [3, 1] = 18

Thus we will fill up the table as

	0	1	2	3	4
0	0	0	0	0	0
1	0	18	18	18	18
2	0	18	18	34	34
3	0	18	18	24	34

table [3, 2] with i = 3, j = 2, $W_i = 2$, $V_i = 6$

table [3, 2] = max {table [i – 1, j], V_i + table [i – 1, j – W_i]}

= max {18, 6 + 0}

table [3, 2] = 18

table [3, 3] with i = 3, j = 3, $W_i = 2$, $V_i = 6$

table [3, 3] = max {18, 6 + 0} = 24

table [3, 4] with i = 3, j = 4, $W_i = 2$, $V_i = 6$

table [3, 4] = max {34, 6 + 18} = 34

Thus the maximal value is $ 34

As table [3, 4] = table [2, 4]

Do not select 3rd item.

Now set i = i = 3 – 1 = 2

Select item 2 and then item 1. The solution vector will be **(1, 1, 0)**

Ex. 5.5.3 *Apply the dynamic programming following instance of the knapsack problem and solve.*

SPPU : May-15, 18 [In Sem], Marks 5

Item	Weight	Value
1	2	$12
2	1	$10
3	3	$20
4	2	$15

Capacity W = 5

Sol. : Initially, table [0, j] and table [i, 0] = 0. There are 0 to n rows and 0 to W columns. in the table.

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

Now we will fill up the table row by row. Let us start filling the table row by row using following formula :

$$\text{table}[i, j] = \begin{cases} \text{maximum } \{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\} \\ \text{or} \qquad \text{when } j \geq w_i \\ \text{table}[i-1, j] \text{ if } j < w_i \end{cases}$$

table [1, 1] with i = 1, j = 1 and $w_i = 2$, $v_i = 12$

As $j < w_i$

table [1, 1] = table [i – 1, j]

= table [0, 1]

table [1, 1] = 0

table [1, 2] with i = 1, j = 2, $w_i = 2$, $v_i = 12$

As $j \geq w_i$

table [1, 2] = max {table [i – 1, j], v_i + table [i – 1, j – w_i]}

= max {0, 12 + 0}

∴ **table [1, 2] = 12**

table [1, 3] with i = 1, j = 3, $w_i = 2$, $v_i = 12$

As $j \geq w_i$

table [1, 3] = max {table [i – 1, j], v_i + table [i – 1, j – w_i]}

= max {0, 12 + 0}

∴ **table [1, 3] = 12**

table [1, 4] with i = 1, j = 4, $w_i = 2$ and $v_i = 12$

As $j \geq w_i$

table [1, 4] = max {table [i – 1, j], v_i + table [i – 1, j – w_i]}

= max {0, 12 + 0}

= 12

table [1, 5] with i = 1, j = 5, $w_i = 2$ and $v_i = 12$

As $j \geq w_i$

$$\text{table}[1,5] = \max\{\text{table}[i-1,j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \max\{0, 12+0\}$$

$\therefore$ **table [1, 5] = 12**

The table with these values will be.

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0					
3	0					
4	0					

table [2, 1] with $i=2, j=1, w_i=1, v_i=10$

As $j \geq w_i$

$$\text{table}[2,1] = \max\{\text{table}[i-1,j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \max\{0, 10+0\}$$

$\therefore$ **table [2, 1] = 10**

table [2, 2] with $i=2, j=2, w_i=1, v_i=10$

As $j \geq w_i$

$$\text{table}[2,2] = \max\{\text{table}[i-1,j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \max\{\text{table}[1,2], 10 + \text{table}[1,1]\}$$
$$= \max\{12, 10+0\}$$

table [2, 2] = 12

table [2, 3] with $i=2, j=3, w_i=1, v_i=10$

As $j \geq w_i$

$$\text{table}[2,3] = \max\{\text{table}[i-1,j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \max\{\text{table}[1,3], 10 + \text{table}[1,2]\}$$
$$= \max\{12, 10+12\}$$

$\therefore$ **table [2, 3] = 22**

table [2, 4] with $i=2, j=4, w_i=1, v_i=10$

As $j \geq w_i$

$$\text{table}[2,4] = \max\{\text{table}[i-1,j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \max\{\text{table}[1,4], 10 + \text{table}[1,3]\}$$

table [2, 4] = 22

table [2, 5] with $i=2, j=5, w_i=1, v_i=10$

As $j \geq w_i$

$$\text{table}[2,5] = \max\{\text{table}[i-1,j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \max\{\text{table}[1,5], 10 + \text{table}[1,4]\}$$

table [2, 5] = 22

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0					
4	0					

table [3, 1] with $i=3, j=1, w_i=3, v_i=20$

As $j \geq w_i$

$$\text{table}[3,1] = \text{table}[i-1,j]$$
$$= \text{table}[2,1]$$

$\therefore$ **table [3, 1] = 10**

table [3, 2] with $i=3, j=2, w_i=3, v_i=20$

As $j < w_i$

$$\text{table}[3,2] = \text{table}[i-1,j]$$
$$= \text{table}[2,2]$$

$\therefore$ **table [3, 2] = 12**

table [3, 3] with $i=3, j=3, w_i=3, v_i=20$

As $j \geq w_i$

$$\text{table}[3,3] = \max\{\text{table}[i-1,j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \max\{\text{table}[2,3], 20 + \text{table}[2,0]\}$$
$$= \max\{22, 20+0\}$$

$\therefore$ **table [3, 3] = 22**

table [3, 4] with $i=3, j=4, w_i=3, v_i=20$

As $j \geq w_i$

$$\text{table}[3,4] = \{\text{table}[i-1,j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \max\{\text{table}[2,4], 20 + \text{table}[2,1]\}$$
$$= \max\{22, 20+10\}$$

$\therefore$ **table [3, 4] = 30**

table [3, 5] with $i=3, j=5, w_i=3, v_i=20$

As $j \geq w_i$

$$\text{table}[3, 5] = \max\{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \max\{\text{table}[2, 5], 20 + \text{table}[2, 2]\}$$
$$= \max\{22, 20 + 12\}$$

$\therefore$ **table [3, 5] = 32**

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0					

table [4, 1] with $i = 4, j = 1, w_i = 2, v_i = 15$

As $j < w_i$

$$\text{table}[4, 1] = \text{table}[i-1, j]$$
$$= \text{table}[3, 1]$$

$\therefore$ **table [4, 1] = 10**

table [4, 2] with $i = 4, j = 2, w_i = 2, v_i = 15$

As $j \geq w_i$

$$\text{table}[4, 2] = \max\{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \max\{\text{table}[3, 2], 15 + \text{table}[3, 0]\}$$
$$= \max\{12, 15 + 0\}$$

$\therefore$ **table [4, 2] = 15**

table [4, 3] with $i = 4, j = 3, w_i = 2, v_i = 15$

As $j \geq w_i$

$$\text{table}[4, 3] = \max\{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \max\{\text{table}[3, 3], 15 + \text{table}[3, 1]\}$$
$$= \max\{22, 15 + 10\}$$

$\therefore$ **table [4, 3] = 25**

table [4, 4] with $i = 4, j = 4, w_i = 2, v_i = 15$

As $j \geq w_i$

$$\text{table}[4, 4] = \max\{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \max\{\text{table}[3, 4], 15 + \text{table}[3, 2]\}$$
$$= \max\{30, 15 + 12\}$$

$\therefore$ **table [4, 4] = 30**

table [4, 5] with $i = 4, j = 5, w_i = 2, v_i = 15$

As $j \; w_i$

$$\text{table}[4, 5] = \max\{\text{table}[i-1, j], v_i + \text{table}[i-1, j-w_i]\}$$
$$= \max\{\text{table}[3, 5], 15 + \text{table}[3, 3]\}$$
$$= \max\{32, 15 + 22\}$$

$\therefore$ **table [4, 5] = 37**

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

$i = 4, k = 4$

As table [i, j] $\neq$ table [i − 1, k]

i.e. table [4, 5] $\neq$ table [3, 5]

Select i^{th} i.e. **4th item.**

Now $i = i - 1$ and $k = k - w_i$

$\therefore$ $i = 4 - 1 = 3$ and $k = 5 - 2 = 3$

As table [i, k] = table [i − 1, k]

i.e. table [3, 3] = table [2, 3]

$\therefore$ Do not select i^{th} i.e. 3^{rd} item.

Now set $i = i - 1$

$\therefore$ $i = 2$ and $k = 3$.

As table [2, 3] $\neq$ table [1, 3]

$\therefore$ Select i^{th} i.e **2nd item.**

Set $i = i - 1$ and $k = k - w_i$

$\therefore$ $i = 1$ and $k = 3 - 1 = 2$.

As table [1, 2] $\neq$ table [0, 2]

Select i^{th} i.e. **1st item.**

Now set $i = i - 1$ and $k = k - w_i$

$\therefore$ $i = 0$ and $k = 0$

Thus solution to this Knapsack problem is **(item 1, item 2, item 4) with total profit = 37.**

Ex. 5.5.4 *Consider 0/1 knapsack problem N = 3, w = (4, 6, 8), p = (10, 12, 15) using dynamic programming devise the recurrence relations for the problem and solve the same. Determine the optimal profit for the knapsack of capacity 10.* **SPPU : Dec.-11, 12, Marks 10**

Sol. : The recurrence relation would be

$$\text{table}[i,j] = \begin{cases} \max\{\text{table}[i-1,j],\ p_i + \text{table}[i-1, j-w_i]\} \text{ when } j \geq w_i, \\ \text{or} \\ \text{table}[i-1,j], \text{ if } j < w_i) \end{cases}$$

initially table [0, j] = 0 and table [i, 0] = 0. There will be 0 to n rows and 0 to W columns in table.

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0										
2	0										
3	0										

Now we will fill up table row by row

table [1,1] with i = 1, j = 1, $w_i = 4$, $p_i = 10$

As $j < w_i$

table [1, 1] = table [i – 1, j] = table [0, 1]

∴ **table [1, 1] = 0**

table with i = 1, j = 2, $w_i = 4$, $p_i = 10$

As $j < w_i$

table [1, 2] = table [i – 1, j] = table [0, 2]

table [1, 2] = 0

Thus table [1, 3] = 0

Now table [1, 4] with i = 1, j = 4, $w_i = 4$, $p_i = 10$ is

As $j = w_i$.

table [i, j] = max {table [i – 1, j], p_i + table [i – 1, j – w_i]}

table [1, 4] = max {table [0, 4], 10 + table [0, 0]}

table [1, 4] = 10

table [1, 5] with i = 1, j = 5, $w_i = 4$, $p_i = 10$.

Table [i, j] = max {table [i – 1, j], pi + table [i – 1, – w_i]

= max (table [0, 5], 10 table [0, 1])

table [1, 5] = 10

Similarly table [1, 6] = table [1, 7] = table [1, 8] = table [1, 9] = table [1, 10] = 10

The table will then be

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	10	10	10	10	10	10	10
2											
3											

table [2, 1] with i = 2, j = 1, $w_2 = 6$, $p_2 = 12$

As $j < w_2$

table [2, 1] = table [1, 1]

table [2, 1] = 0

Similarly table [2, 2] = table [2, 3] = table [2, 4] = table [2, 5] = 0

Now table [2, 6] with i = 2, j = 6, $w_2 = 6$, $p_2 = 12$.

As $j = w_2 = 6$

table [2, 6] = max {table [i – 1, j], p_i + table [i – 1, j – w_i]}

= max {table [1, 6], 12 + table [1, 0]}

table [2, 6] = 12

Similarly **table [2, 7] = table [2, 8] = table [2, 9] = 12**

Now with table [2, 10] , i = 2, j = 10, $w_2 = 6$, $p_2 = 12$.

table [2, 10] = max {table [i – 1, j], p_i + table [i – 1, j – w_i]}

= max {table [1, 10], 12 + table [1, 4]}

table [2, 10] = 22

The table will then be

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	10	10	10	10	10	10	10
2	0	0	0	0	0	0	12	12	12	12	22
3	0										

table [3, 1] with i = 3, j = 1, $w_3 = 8$, $p_3 = 15$.

$j < w_i$ is true for j = 1 to 7.

Hence

table [3, 1] = table [2, 1] = 0
table [3, 2] = table [2, 2] = 0
table [3, 3] = table [2, 3] = 0
table [3, 4] = table [3, 5] = 0
table [3, 6] = table [2, 6] = 12
table [3, 7] = table [2, 7] = 12

Now with table [3, 8] i = 3, j = 8, $w_3 = 8$, $p_3 = 15$.

As $j = w_i$

table [3, 8] = max {table [i – 1, j], p_i + table [i – 1, j – w_i]}

= max {table [2, 8], 15 + table [2, 0]}

table [3, 8] = 15

Similarly table [3, 9] = 15

But with table [3, 10] i = 3, j = 10, $w_3 = 8$, $p_3 = 15$.

table [3, 10] = max {table [2, 10], 15 + table [2, 2]}

table [3, 10] = 22

The table will be

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	10	10	10	10	10	10	10	10
2	0	0	0	0	0	0	12	12	12	12	22
3	0	0	0	0	0	0	12	12	15	15	22

Now let us apply steps to identify selected items.

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	10	10	10	10	10	10	10	10
2	0	0	0	0	0	0	12	12	12	12	22
3	0	0	0	0	0	0	12	12	15	15	22

i = 3, k = 10

check table [i, k] $\stackrel{?}{=}$ table [i – 1, k]

check if table [3, 10] = table [2, 10]

i.e. 22 = 22

∴ *Do not select* i^{th} i.e. 3^{rd} item.

$i = i - 1$

Now set $i = 3 - 1 = 2$

∴ $i = 2,\ k = 10$

Now table [2, 10] $\stackrel{?}{=}$ table [1, 10]

∴ table [2, 10] ≠ table [1, 10]

As $10 \neq 0$

i.e. **Select i^{th} i.e. 2^{nd} item**

∴ $i = i - 1$ and $k = k - w_i$

Set $i = 2 - 1 = 1$ and $k = 10 - 6 = 4$

∴ $i = 1,\ k = 4$

Now table [1, 4] $\stackrel{?}{=}$ table [0, 4]

∴ $10 \neq 0$

As **Select i^{th} i.e. 1^{st} item**

Set $i = i - 1$ $k = k - w_i$

$i = i - 1$ $k = 4 - 4$

∴ $i = 0$ $k = 0$.

Terminate the process by selecting item 1 and item 2

∴ Solution is **(1, 1, 0).**

Ex. 5.5.5 *Solve the knapsack problem using Dynamic programming for number of objects n = 4, given capacity M = 8.* **SPPU : April 18, In Sem., Marks 5**

Items	*1*	*2*	*3*	*4*
Value	*15*	*10*	*9*	*5*
Weight	*1*	*5*	*3*	*4*

Sol. : Initially table[0, j] = table[i,0] = 0 in the table[n, M]

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

Now we will fill up the table row by row using following formula -

$$table[i, j] = \max\{table[i-1], v_i + table[i-1, j-w_i]\} \text{ when } j \geq w_i$$

or

$table[i-1, j]$ if $j < w_i$

Step 1 :

table[1, 1] with $i = 1, j = 1, w_i = 1$ $v_i = 15$ As $j \geq w_i$ **Formula Used :** $\max\{table[i-1, j], v_i + table[i-1, j-w_i]\}$ $= \max\{table[0, 1], v_i + table[0, 0]$ $= \max\{0, 15 + 0\}$ **∴ table[1, 1] = 15**	table[1, 2] with $i = 1, j = 2, w_i = 1$ $v_i = 15$ As $j \geq w_i$ **Formula Used :** $\max\{table[i-1, j], v_i + table[i-1, j-w_i]\}$ $= \max\{table[0, 2], v_i + table[0, 1]$ $= \max\{0, 15 + 0\}$ **∴ table[1, 2] = 15**
table[1, 3] with $i = 1, j = 3, w_i = 1$ $V_i = 15$ As $j \geq w_i$ **Formula Used :** $\max\{table[i-1, j], v_i + table[i-1, j-w_i]\}$ $= \max\{table[0, 3], v_i + table[0, 2]$ $= \max\{0, 15 + 0\}$ **∴ table[1, 3] = 15**	table[1, 4] with $i = 1, j = 4, w_i = 1$ $v_i = 15$ As $j \geq w_i$ **Formula Used :** $\max\{table[i-1, j], v_i + table[i-1, j-w_i]\}$ $= \max\{table[0, 4], v_i + table[0, 3]$ $= \max\{0, 15 + 0\}$ **∴ table[1, 4] = 15**
table[1, 5] with $i = 1, j = 5, w_i = 1$ $v_i = 15$ As $j \geq w_i$ **Formula Used :** $\max\{table[i-1, j], v_i + table[i-1, j-w_i]\}$ $= \max\{table[0, 5], v_i + table[0, 4]$ $= \max\{0, 15 + 0\}$ **∴ table[1, 5] = 15**	table [1, 6] with $i = 1, j = 6, w_i = 1$ $v_i = 15$ As $j \geq w_i$ **Formula Used :** $\max\{table[i-1, j], v_i + table[i-1, j-w_i]\}$ $= \max\{table[0, 6], v_i + table[0, 5]$ $= \max\{0, 15 + 0\}$ **∴ table[1, 6] = 15**
table[1, 7] with $i = 1, j = 7, w_i = 1$ $v_i = 15$ As $j \geq w_i$ **Formula Used :** $\max\{table[i-1, j], v_i + table[i-1, j-w_i]\}$ $= \max\{table[0, 7], v_i + table[0, 6]$ $= \max\{0, 15 + 0\}$ **∴ table[1, 7] = 15**	table[1, 8] with $i = 1, j = 8, w_i = 1$ $v_i = 15$ As $j \geq w_i$ **Formula Used :** $\max\{table[i-1, j], v_i + table[i-1, j-w_i]\}$ $= \max\{table[0, 8], V_i + table[0, 7]$ $= \max\{0, 15 + 0\}$ **∴ table[1, 8] = 15**

The partially filled table is

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	15	15	15	15	15	15	15	15
2	0								
3	0								
4	0								

Step 2 :

table[2, 1] with $i = 2, j = 1, w_i = 5$ $v_i = 10$ As $j \geq w_i$ **Formula Used :** table $[i-1, j]$ = table [1, 1] = 15 **∴ table[2, 1] = 15**	table[2, 2] with $i = 2, j = 2, w_i = 5$ $v_i = 10$ As $j \geq w_i$ **Formula Used :** table $[i-1, j]$ = table [1, 2] = 15 **∴ table[2, 2] = 15**
table[2, 3] with $i = 2, j = 3, w_i = 5$ $v_i = 10$ As $j \geq w_i$ **Formula Used :** table $[i-1, j]$ = table [1, 3] = 15 **∴ table[2, 3] = 15**	table[2, 4] with $i = 2, j = 4, w_i = 5$ $v_i = 10$ As $j \geq w_i$ **Formula Used :** table $[i-1, j]$ = table [1, 4] = 15 **∴ table[2, 4] = 15**
table[2, 5] with $i = 2, j = 5, w_i = 5$ $v_i = 10$ As $j \geq w_i$ **Formula Used :** max {table $[i-1, j]$ v_i + table $[i-1, j-w_i]$} = max {table [1, 5], v_i + table [1, 0] = max {15, 10 + 0} **∴ table[2, 5] = 15**	table[2, 6] with $i = 2, j = 6, w_i = 5$ $v_i = 10$ As $j \geq w_i$ **Formula Used :** max {table $[i-1, j]$ v_i + table $[i-1, j-w_i]$} = max {table [1, 6], v_i + table [1, 1] = max {15, 10 + 0} **∴ table[2, 6] = 25**
table[2, 7] with $i = 2, j = 7, w_i = 5$ $v_i = 10$ As $j \geq w_i$ **Formula Used :** max {table $[i-1, j]$ v_i + table $[i-1, j-w_i]$} = max {table [1, 7], v_i + table [1, 2] = max {15, 15 + 15} **∴ table[2, 7] = 25**	table[2, 8] with $i = 2, j = 8, w_i = 5$ $v_i = 10$ As $j \geq w_i$ **Formula Used :** max {table $[i-1, j]$ v_i + table $[i-1, j-w_i]$} = max {table [1, 8], v_i + table [1, 3] = max {15, 10 + 15} **∴ table[2, 8] = 25**

The partially filled table is

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	15	15	15	15	15	15	15	15
2	0	15	15	15	15	15	15	15	15
3	0								
4	0								

Step 3 :

table[3, 1] with i = 3, j = 1, $w_i = 3$
$v_i = 9$
As $j \geq w_i$
Formula Used : table[i – 1, j]
= table[2, 1]
= 15
∴ table[3, 1] = 15

table[3, 2] with i = 3, j = 2, $w_i = 3$
$v_i = 9$
As $j \geq w_i$
Formula Used : table[i – 1, j]
= table[2, 2]
= 15
∴ table[3, 2] = 15

table[3, 3] with i = 3, j = 3, $w_i = 3$
$v_i = 9$
As $j \geq w_i$
Formula Used : max{table[i – 1, j], v_i + table [i – 1, j – w_i]}
= max{table[2, 3], v_i + table [2, 0]
= max{15, 9 + 0}
∴ table[3, 3] = 15

table[3, 4] with i = 3, j = 4, $w_i = 3$
$v_i = 9$
As $j \geq w_i$
Formula Used : max{table[i – 1, j], v_i + table [i – 1, j – w_i]}
= max{table[2, 4], v_i + table [2, 1]
= max{15, 9 + 15}
∴ table[3, 4] = 24

table[3, 5] with i = 3, j = 5, $w_i = 3$
$v_i = 9$
As $j \geq w_i$
Formula Used : max{table[i – 1, j], v_i + table [i – 1, j – w_i]}
= max{table[2, 5], v_i + table [2, 2]
= max{15, 9 + 15}
∴ table[3, 5] = 24

table[3, 6] with i = 3, j = 6, $w_i = 3$
$v_i = 9$
As $j \geq w_i$
Formula Used : max{table[i – 1, j], v_i + table [i – 1, j – w_i]}
= max{table[2, 6], V_i + table [2, 3]
= max{25, 9 + 15}
∴ table[3, 6] = 25

table[3, 7] with i = 2, j = 7, $w_i = 3$
$v_i = 9$
As $j \geq w_i$
Formula Used : max{table[i – 1, j], v_i + table [i – 1, j – w_i]}
= max{table[2, 7], V_i + table [2, 4]
= max{25, 9 + 15}
∴ table[3, 7] = 25

table[3, 8] with i = 3, j = 8, $w_i = 3$
$v_i = 9$
As $j \geq W_i$
Formula Used : max{table[i – 1, j], v_i + table [i – 1, j – w_i]}
= max{table[2, 8], v_i + table [2, 5]
= max{25, 9 + 15}
∴ table[3, 8] = 25

The partially filled table is

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	15	15	15	15	15	15	15	15
2	0	15	15	15	15	15	15	15	15
3	0	15	15	15	15	15	15	15	15
4	0								

Step 4 :

table[4, 1] with i = 4, j = 1, $w_i = 4$
$v_i = 5$
As $j \geq w_i$
Formula Used : table[i – 1, j]
= table[3, 1]
= 15
∴ table[4, 1] = 15

table[4, 2] with i = 4, j = 2, $w_i = 4$
$v_i = 5$
As $j \geq w_i$
Formula Used : table[i – 1, j]
= table [3, 2]
= 15
∴ table[4, 2] = 15

table[4, 3] with i = 4, j = 3, $w_i = 4$
$v_i = 5$
As $j \geq w_i$
Formula Used : table[i – 1, j]
= table[3, 3]
= 15
∴ table[4, 3] = 15

table[4, 4] with i = 4, j = 4, $w_i = 4$
$v_i = 5$
As $j \geq w_i$
Formula Used : max{table[i – 1, j], v_i + table [i – 1, j – w_i]}
= max{table[3, 4], v_i + table [3, 0]
= max{24, 5 + 0}
∴ table[4, 4] = 24

table[4, 5] with i = 4, j = 5, $w_i = 4$
$v_i = 5$
As $j \geq w_i$
Formula Used : max{table[i – 1, j], v_i + table [i – 1, j – w_i]}
= max{table[3, 5], v_i + table [3, 1]
= max{24, 5 + 15}
∴ table[4, 5] = 24

table[4, 6] with i = 4, j = 6, $w_i = 4$
$v_i = 5$
As $j \geq w_i$
Formula Used : max{table[i – 1, j], v_i + table [i – 1, j – w_i]}
= max{table[3, 6], v_i + table [3, 2]
= max{25, 5 + 15}
∴ table[4, 6] = 25

table[4, 7] with i = 4, j = 7, $w_i = 4$
$v_i = 5$
As $j \geq w_i$
Formula Used : max{table[i – 1, j], v_i + table [i – 1, j – w_i]}
= max{table[3, 7], v_i + table [3, 3]
= max{25, 5 + 15}
∴ table[4, 7] = 25

table[4, 8] with i = 4, j = 8, $w_i = 4$
$v_i = 5$
As $j \geq w_i$
Formula Used : max{table[i – 1, j], v_i + table [i – 1, j – w_i]}
= max{table[3, 8], v_i + table [3, 4]
= max{25, 5 + 24}
∴ table[4, 8] = 29

The partially filled table is

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	15	15	15	15	15	15	15	15
2	0	15	15	15	15	15	15	15	15
3	0	15	15	15	15	15	15	15	15
4	0	15	15	15	15	15	15	15	15

Selection of items

Step 1 : i = 4, k = 8 As table[i, k] ≠ table[i – 1, k] i.e. table[4, 8] ≠ table[3, 8] **Select i^{th} i.e. 4^{th} item**	**Step 2 :** Now i = i – 1 and k = w_i ∴ i = 3 and k = 8 – 4 = 4 As table[i, k] ≠ table[i – 1, k] i.e. table[3, 4] ≠ table[2, 4] **Select i^{th} i.e. 3^{rd} item**
Step 3 : Now i = i – 1 and k = k – w_i ∴ i = 2 and k = 4 – 3 = 1 As table[i, k] ≠ table[i – 1, k] i.e. table[2, 1] = table[1, 1] **Do not Select i^{th} i.e. 2^{nd} item**	**Step 4 :** Now i = i – 1 ∴ i = 1 and k = 1 As table[i, k] ≠ table[i – 1, k] i.e. table[1, 1] ≠ table[0, 1] **Select i^{th} i.e. 1^{st} item**

Thus the solution to Knapsack item is **(1, 0, 1, 1)**

5.5.1 Algorithm

```
Algorithm Dynamic_Knapsack(n,W,w[ ],v[ ])
//Problem Description:This algorithm is for obtaining knapsack
//solution using dynamic programming
//Input: n is total number of items, W is the capacity of
//knapsack, w[ ] stores weights of each item and v[ ] stores
//the values of each item.
//Output: Returns the total value of selected items for the
//knapsack.
for (i ←0 to n) do
{
for (j←0 to W) do
{
table[i,0]=0  // table initialization
table[0,j]=0
}
}
for (i←0 to n) do
{
for (j←0 to W) do
{
if(j<w[i]) then
  table[i,j]← table[i-1,j]
else if(j>=w[i]) then
  table[i,j]←max (table[i-1,j],(v[i]+table[i-1,j-w[i]]) )
}
}
return table[n,W]
```

5.5.2 Analysis

In this algorithm the basic operation is **if ... else if** statement within two nested for loops.

Hence

$$c(n) = \sum_{i=0}^{n} \sum_{j=0}^{W} 1 = \sum_{i=0}^{n} W - 0 + 1$$

$$= \sum_{i=0}^{n} (W+1) = \sum_{i=0}^{n} W + \sum_{i=0}^{n} 1$$

$$= W \cdot \sum_{i=0}^{n} 1 + \sum_{i=0}^{n} = W(n-0+1) + (n-0+1)$$

$$= W_n + W + n + 1$$

Thus $c(n) \approx W_n$ **The time complexity of this algorithm is Θ (nW).**

Review Question

Q.1 Write an algorithm for 0/1 knapsack problem using dynamic programming approach. **SPPU : May-07, 08, Dec.-07, Marks 8**

5.6 Coin Change-making Problem

SPPU : April-18, Marks 5

Suppose there are some coins available. The values of these coins are as follows.

1 dollar = 100 cents

1 quarter = 25 cents
1 dimes = 10 cents
1 nickels = 5 cents
1 pennies = 1 cent

Now the making change problem is to pay the desired amount of money by using as few coins as possible.

For example : If a customer wants to pay 3.37 S then he should pay 3 dollars (= 300 cents) ÷ 1 quarter (= 25 cents) ÷ 1 dime (= 10 cents) + 2 pennies (= 2 cents) = 3.37 S.

This method uses **Greedy** approach because at every step it chooses the largest coin it can. Furthermore, once the coin is selected then that is the final. This approach does not allow to change the decision. Unfortunately although **Greedy** approach is very efficient, it works only for limited number of instances. For example if there exists coins of 1, 4 and 6 units and we have to make change for 9 units. Then there are two ways :

1. Greedy Method : Select one coin of 6 unit and 3 coins of 1 unit = **4 coins.**

2. Better Method : Select two coins of 4 units and 1 coin of 1 unit = **3 coins.**

This better method is devised by dynamic programming approach.

For solving this problem using dynamic programming approach we need to build a table, in which rows correspond to denomination and column corresponds to 0 to N units.

Ex. 5.6.1 *Solve making change problem for $d_1 = 1$, $d_2 = 4$, $d_3 = 6$, $n = 3$ and $N = 8$ units.*

Sol. : Given that :

$d_1 = 1, d_2 = 4, d_3 = 6, n = 3, N = 8$ units.

Hence we will create a table with rows ranging from 1 to 3 and columns ranging from 0 to 8.

Initially c [i, 0] = 0 ∴ c [1, 0] = c [2, 0] = c [3, 0] = 0

j →

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

We can perform computations row-by-row from left to right, or column-by-column from top to bottom or diagonally.

Here we will perform computation **column-by-column** and fill up the table accordingly.

We will use following formula for computations.

1. If $i = 1$ then $c [i, j] = 1 + c [1, j - d_1]$
2. If $j < d_i$ then $c [i, j] = c [i - 1, j]$
3. Otherwise $c [i] [j] = \min (c [i - 1, j], 1 + c [i, j - d_i])$

c [1, 1] with i = 1, j = 1 $d_1 = 1$.

As i = 1

Formula used : $1 + c [1, j - d_1]$
$= 1 + c [1, 1 - 1]]$
$= 1 + c [1, 0]$
$= 1 + 0$
$= 1$

c [1, 1] = 1

∴ **c [2, 1] with i = 2, j = 1, $d_2 = 4$**

As $j < d_2$ i.e. $1 < 4$

Formula used : $c [i - 1, j]$
$= c [1, 1]$
$= 1$

∴ **c [2, 1] = 1**

c [3, 1] with i = 3, j = 1, $d_3 = 6$

As $j < d_3$ i.e. $1 < 6$

Formula used : $c [i - 1, j]$
$= c [3 - 1, 1]$
$= c [2, 1]$
$= 1$

∴ **c [3, 1] = 1**

c [1, 2] with i = 1, j = 2, $d_1 = 1$

As i = 1

Formula used : $1 + c [1, j - d_1]$
$= 1 + c [1, 2 - 1]$
$= 1 + c [1, 1]$
$= 1 + 1$

$= 2$

$\therefore$ **c [1, 2] = 2**

c [2, 2] with i = 2, j = 2, $d_2 = 4$

As $j < d_2$ i.e. $2 < 4$

Formula used : $c[i-1, j]$

$= c[1, 2]$

$= 2$

$\therefore$ **c [2, 2] = 2**

c [3, 2] with i = 3, j = 2, $d_3 = 6$

As $j < d_3$ i.e. $2 < 6$

Formula used : $c[i-1, j]$

$= c[2, 2]$

$= 2$

$\therefore$ **c [2, 2] = 2**

c [1, 3] with i = 1, j = 3, $d_1 = 1$

As $i = 1$

Formula used : $1 + c[1, j - d_1]$

$= 1 + c[1, 3-1]$

$= 1 + c[1, 2]$

$= 1 + 2$

$= 3$

$\therefore$ c [1, 3] = **3**

c [2,3] with i = 2, j = 3, $d_2 = 4$

As $j < d_3$ i.e. $3 < 4$

Formula used : $c[i-1, j]$

$= c[1,3]$

$= 3$

$\therefore$ **c [2, 3] = 3**

c [3, 3] with i = 3, j = 3, $d_3 = 6$

As $j < d_3$ i.e. $3 < 6$

Formula used : $c[i-1, j]$

$= c[2, 3]$

$= 3$

$\therefore$ **c [3, 3] = 3**

c [1, 4] with i = 1, j = 4, $d_1 = 1$

As $i = 1$

Formula used : $1 + c[1, j - d_1]$

$= 1 + c[1, 4-1]$

$= 1 + c[1, 3]$

$= 1 + 3$

$= 4$

$\therefore$ **c [1, 4] = 4**

c [2, 4] with i = 2, j = 4, $d_2 = 4$

As $i \neq 1$ and $j > d_2$

Formula used : $\min(c[i-1, j], 1 + c[i, j - d[i]])$

$= \min(c[2-1, 4], 1 + c[2, 4-4])$

$= \min(c[1, 4]\ 1 + c[2, 0])$

$= \min(4, 1 + 0)$

$= 1$

$\therefore$ **c [2, 4] = 1**

c [3, 4] with i = 3, j = 4, $d_3 = 6$

As $j < d_3$ i.e. $4 < 6$

Formula used : $c[i-1, j]$

$= c[2, 4]$

$= 1$

$\therefore$ **c [3, 4] = 1**

c [1, 5] with i = 1, j = 5, $d_1 = 1$

As $i = 1$

Formula used : $1 + c[1, j - d_1]$

$= 1 + c[1, 5-1]$

$= 1 + c[1, 4]$

$= 1 + 4$

$= 5$

$\therefore$ **c [1, 5] = 5**

c[2, 5] with i = 2, j = 5, $d_2 = 4$

As $i \neq 1$ and $j > d_2$

Formula used : $\min (c [i - 1, j], 1 + c [i, j - d_2])$

$= \min (c [1, 5], 1 + c [2, 5 - 4]$

$= \min (c [1, 5], 1 + c \{2, 1])$

$= \min (5, 1 + 1)$

$= 2$

$\therefore$ **c [2, 5] = 2**

c [3, 5] with i = 3, j = 5, $d_3 = 6$

As $j < d_3$ i.e. $5 < 6$

Formula used : $c [i - 1, j]$

$= c [3 - 1, 5]$

$= c [2, 5]$

$\therefore$ **c [3, 5] = 2**

c [1, 6] with i = 1, j = 6, $d_1 = 1$

As i = 1

Formula used : $1 + c [1, j - d_1]$

$= 1 + c [1, 6 - 1]$

$= 1 + c [1, 5]$

$= 6$

$\therefore$ **c [1, 6] = 6**

c [2, 6] with i = 2, j = 6, $d_2 = 4$

As $i \neq 2$ and $j > d_2$

Formula used : $\min (c [i - 1, j], 1 + c [i, j - d_2]$

$= \min (c [1, 6], 1 + c [2, 2])$

$= \min (6, 1 + 2)$

$= 3$

$\therefore$ **c [2, 6] = 3**

c [3, 6] with i = 3, j = 6, $d_3 = 6$

As $i \neq 3$ and $j > d_3$

Formula used : $\min (c [i - 1, j], 1 + c [i, j - d_3])$

$= \min (c [2, 6], 1 + c [3, 0]$

$= \min (3, 1 + 0)$

$= 1$

$\therefore$ **c [3, 6] = 1**

c [1, 7] with i = 1, j = 7 $d_1 = 1$

As i = 1

Formula used : $1 + c [1, j - d_1]$

$= 1 + c [1, 6]$

$= 1 + 6$

$\therefore$ **c [1, 7] = 7**

c [2, 7] with i = 2, j = 7, $d_2 = 4$

As $i \neq 1$ and $j > d_2$

Formula used : $\min (c [i - 1, j], 1 + c [i, j - d_2])$

$= \min (c [1, 7], 1 + c [2, 3])$

$= \min (7, 1 + 3)$

$= 4$

$\therefore$ **c [2, 7] = 4**

c [3, 7] with i = 3, j = 7, $d_3 = 6$

As $i \neq 1$ and $j > d_3$

Formula used : $\min (c [i - 1, j], 1 + c [i, j - d_3]$

$= \min (c [2, 7], 1 + c [3, 1])$

$= \min (4, 1 + 1)$

$= 2$

$\therefore$ **c [3, 7] = 2**

c [1, 8] with i = 1, j = 8, $d_1 = 1$

As i = 1

Formula used : $1 + c [1, j - d_1]$

$= 1 + c [1, 7]$

$= 1 + 7$

$= 8$

$\therefore$ **c [1, 8] = 8**

c [2,8] with i = 2, j = 8, $d_2 = 4$

As $i \neq 1$ and $j > d_2$

Formula used : $\min (c [i - 1, j], 1 + c [i, j - d_2])$

$= \min (c [1, 8], 1 + c[2, 4])$

$= \min (8, 1 + 1)$

$= 2$

$\therefore$ **c [2, 8] = 2**

c [3, 8] with i = 3, j = 8, $d_3 = 6$

As $i \neq 1$ and $j > d_3$

Formula used : $\min (c [i - 1, j], 1 + c [i, j - d_3]$

$= \min (c [2, 8], 1 + c [3, 2])$

$= \min (2, 1 + 2)$

$= 2$

$\therefore$ **c [3, 8] = 2**

Thus we can represent the table filled up by above computations will be

		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

↑ Total number of coins

The value c [n, N] i.e. c [3, 8] = 2 represents the minimum number of coins required to get the sum for 8 units. Hence we require 2 coins for getting 8 units. The coins are :

4 units = 1 coin

\+ 4 units = 1 coin

8 units = 2 coins

Ex. 5.6.2 *Given coins of denominations 2, 4 and 5 with amount to be pay is 7. Find optimal number of coins and sequence of coins used to pay given amount using dynamic method.*

Sol.: $d_1 = 2$, $d_2 = 4$, $d_3 = 5$. N = 7

We will create a table with rows ranging from 1 to 3 and column ranging from 0 to 7.

Initially C [i, 0] = 0

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

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

Computations can be performed column by column; and fill up the table accordingly.

C [1, 1] with i = 1, j = 1, $d_1 = 2$

As i = 1

$C [1,1] = 1 + C[1, j - d1]$

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

C [1, 1] = ∞

C [2, 1] with i = 2, j = 1, $d_2 = 4$

$C [2, 1] = C [i - 1, j]$

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

C [2, 1] = ∞

C [3, 1] with i = 3, j = 1, $d_3 = 5$

As $j < d_3$

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

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

$\therefore$ **C [3, 1] = ∞**

C [1, 2] with i = 1, j = 2, $d_1 = 2$

As i = 1

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

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

$\therefore$ **C [1, 2] = 1**

C [2, 2] with i = 2, j = 2, $d_2 = 4$

As $j < d_2$

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

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

$\mathbf{C[2, 2] = 1}$

C[3, 2] with $i = 3, j = 2, d_3 = 5$

As $j < d_3$

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

$\therefore \quad C[3, 2] = C[2, 2]$

$\therefore \quad \mathbf{C[3, 2] = 1}$

C [1, 3] with $i = 1, j = 3\ d_1 = 2$

As $i = 1$

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

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

$\mathbf{C[1, 3] = \infty}$

C [2, 3] with $i = 2, j = 3, d_2 = 4$

As $j < d_2$

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

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

$\therefore \quad \mathbf{C[2, 3] = \infty}$

C [3, 3] with $i = 3, j = 3, d_3 = 5$

As $j < d_3$

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

$\therefore \quad C[3, 3] = C[2, 3]$

$\therefore \quad \mathbf{C[3, 3] = \infty}$

C [1, 4] with $i = 1, j = 4, d_1 = 2$

As $i = 1$

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

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

$\therefore \quad \mathbf{C[1, 4] = 2}$

C [2, 4] with $i = 2, j = 4, d_2 = 4$

As $d_2 = j$ and $i \neq 1$

$C[i, j] = \min(C[i-1, j], 1 + C[i, j - d[i]])$

$= \min(C[1, 4], 1 + C[2, 4 - 4])$

$= \min(C[1, 4], 1 + 0)$

$\mathbf{C[2, 4] = \min(2, 1)}$

C [3, 4] with $i = 3, j = 4$ and $d_3 = 5$

As $j < d_3$

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

$C[3, 4] = C[2, 4]$

$\mathbf{C[3, 4] = 1}$

$\therefore$

C [1, 5] with $i = 1, j = 5$ and $d_1 = 2$

As $i = 1$

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

$= 1 + C[1, 3]$

$\mathbf{C[1, 5] = \infty}$

C [2, 5] with $i = 2, j = 5$ and $d_2 = 4$

$C[i, j] = \min(C[i-1, j]\ 1 + C[i, j - d[i]])$

$= \min(C[1, 5], 1 + C[2, 1])$

$C[2, 5] = \min(\infty, 1 + \infty)$

$\mathbf{C[2, 5] = \infty}$

C[3, 5] with $i = 3, j = 5$ and $d_3 = 5$

$C[i, j] = \min(C[i-1, j], 1 + C[i, j - d[i]])$

$= \min(C[2, 5], 1 + C[3, 0])$

$C[3, 5] = \min(\infty, 1 + 0)$

$\mathbf{C[3, 5] = 1}$

C[1, 6] with $i = 1, j = 6, d_1 = 2$

As $i = 1$

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

$C[1, 6] = 1 + C[1, 4] = 1 + 2$

$\mathbf{C[1, 6] = 3}$

C [2, 6] with $i = 2, j = 6, d_2 = 4$

$C[i, j] = \min(C[i-1, j], 1 + C[i, j - d[i]])$

$= \min(C[1, 6], 1 + C[2, 2])$

$C[2, 6] = \min(3, 1+1)$

C [2, 6] = 2

C [3, 6] with i = 3, j = 6, $d_3 = 5$

C[i, j] = min (C [i – 1, j], 1 + C [i, j – d [i]])

= min (C [2, 6], 1 + C [3, 1])

= min (2, 1 + ∞)

C[3, 6] = 2

C [1, 7] with i = 1, j = 7 and $d_1 = 2$

As i = 1

C [i, j] = 1+ C [i, j – d_1]

= 1 + C [1, 5] = 1 + ∞

C [1, 7] = ∞

C [2,7] with i = 2, j = 7 and $d_2 = 4$

C [i, j] = min (C [i – 1, j], 1 + C [i, j – d [i]])

= min (C [1, 7], 1 + C [2, 3])

= min (∞, 1 + ∞)

C [2, 7] = ∞

C [3, 7] with i = 3, j = 7 and $d_3 = 5$

C [i, j] = min (C [i – 1, j], 1 + C [i, j – d [i]])

= min (C [2, 7], 1 + C [3, 2])

= min (∞, 1 + 1)

C [3, 7] = 2

The table can be

		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

↑ Total number of coins

That means 2 coins are required for sum as 7. The coins are,

$d_1 = 2 \rightarrow$ 1 coin

$d_3 = 5 \rightarrow$ 1 coin

Total 7 with 2 coins

Ex. 5.6.3 *Solve making change problem using Dynamic Programming. (Denominations : d1 = 1, d2 = 4, d3 = 6). Give your answer for making change of ₹ 9.*

Sol. : $d_1 = 1$, $d_2 = 4$, $d_3 = 6$, N = 9 ₹, n = 3. Hence we will create a table with rows ranging from 1 to 3 and columns ranging from 0 to 9.

Initially C[i, 0] = 0 C[1, 0] = C [2, 0] = C [3, 0] = 0

j →

	0	1	2	3	4	5	6	7	8	9
$d_1 = 1$	0									
$d_2 = 4$	0									
$d_3 = 6$	0									

We will complete the table **column-by-column :** Refer example 5.6.1.

	0	1	2	3	4	5	6	7	8	9
$d_1 = 1$	0	1	2	3	4	5	6	7	8	
$d_2 = 4$	0	1	2	3	1	2	3	4	2	
$d_3 = 6$	0	1	2	3	1	2	1	2	2	

c[1, 9] with i = 1, j = 9, $d_1 = 1$

As i = 1

Formula used : 1 + c [1, j – d_1]

= 1 + c [1, 8]

= 1 + 8

= 9

∴ **c [1, 9] = 9**

c [2, 9] with i = 2, j = 9, $d_2 = 4$

As i ≠ 1 and j > d_2

Formula used : min (c [i – 1, j], 1 + c [i, j – d_2])

= min (c [1, 9], 1 + c [2, 5])

= min (9, 1 + 2)

= 3

∴ **c [2, 9] = 3**

c [3, 9] with i = 3, j = 9, $d_3 = 6$

As i ≠ 1 and j > d_3

Formula used : min (c [i – 1, j], 1 + c [i, j – d_3])

= min (c [2, 9], 1 + c [3, 3])

= min (3, 1 + 3)

= 3

∴ **c [3, 9] = 3**

Thus the complete table filled up by all the computations will be

	0	1	2	3	4	5	6	7	8	9
$d_1 = 1$	0	1	2	3	4	5	6	7	8	9
$d_2 = 4$	0	1	2	3	1	2	3	4	2	3
$d_3 = 6$	0	1	2	3	1	2	1	2	2	3

Total number of coins

The value c [n, N] i.e. c [3, 9] = 3 represents the minimum number of coins required to get the sum for 9 units. Hence coins will be

$d_2 = 4$	1 coin
$d_2 = +4$	1 coin
$d_1 = +1$	1 coin
₹ 9	= 3 coins

Ex. 5.6.4 *Solve making change problem using dynamic technique. d1 = 1, d2 = 3, d3 = 5, d4 = 6. Calculate for making change of ₹ 8.*

Sol. : Let,

d1 = 1, d2 = 3, d3 = 5, d4 = 6

We will create a table with rows ranging from 1 to 4 and columns ranging from 0 to 8.

Initially c [i, 0] = 0

		0	1	2	3	4	5	6	7	8
i = 1	d1 = 1	0								
i = 2	d2 = 3	0								
i = 3	d3 = 5	0								
i = 4	d4 = 6	0								

Computations can be performed column by column and fill up the table.

c [1, 1] with i = 1, j = 1, d1 = 1.

As i = 1

c [1, 1] = 1 + c [1, j – d1]

= 1 + c [1, 0]

= 1 + 0

c [1, 1] = 1.

c [2, 1] with i = 2, j = 1, d2 = 3.

Here j < di i.e. 1 < 3

c [i, j] = c [i – 1, j]

∴ c [2, 1] = c [2 – 1, 1]

= c [1, 1]

c [2, 1] = 1

c [3, 1] with i = 3, j = 1, d3 = 5

j < di i.e. 1 < 5

c [i, j] = c [i – 1, j]

∴ c [3, 1] = c [3 – 1, 1] = c [2, 1]

c [3, 1] = 1

c [4, 1] with i = 4, j = 1, d4 = 6

j < d4 i.e. 1 < 6

c [i, j] = c [i – 1, j]

∴ c [4, 1] = c [4 – 1, 1]

= c [3, 1]

c [4, 1] = 1

c [1, 2] with i = 1, j = 2, d1 = 1

Here As i = 1

c [i, j] = 1 + c [1, j – d1]

∴ = 1 + c [1, 1] = 1 + 1

c [1, 2] = 2

c [2, 2] with i = 2, j = 2, d2 = 3

As j < di i.e. 2 < 3

c [i, j] = c [i – 1, j]

∴ = c [2 – 1, 2]

$= c[1, 2]$

$c[2, 2] = 2$

$c[3, 2]$ with $i = 3, j = 2, d3 = 5$

As $j < di$ i.e. $2 < 5$

$\therefore$ $c[i, j] = c[i - 1, j]$

$= c[3 - 1, 2]$

$= c[2, 2]$

$\mathbf{c[3, 2] = 2}$

$c[4, 2]$ with $i = 4, j = 2, d4 = 6$

As $j < di$ i.e. $2 < 6$

$\therefore$ $c[i, j] = c[i - 1, j]$

$= c[4 - 1, 2]$

$= c[3, 2]$

$\mathbf{c[4, 2] = 2}$

$c[1, 3]$ with $i = 1, j = 3, d1 = 1$

As $i = 1$

$\therefore$ $c[i, j] = 1 + c[1, j - d1]$

$= 1 + c[1, 3 - 1]$

$= 1 + c[1, 2]$

$\mathbf{c[1, 3] = 3}$

$c[2, 3]$ with $i = 2, j = 3, d2 = 3$

$i \neq 1$ and $j = d_i$

$\therefore$ Formula used $c[i, j] = \min(c[i - 1, j], 1 + c[i, j - di])$

$\therefore$ $c[2, 3] = \min(c[2 - 1, 3], 1 + c[2, 3 - 3])$

$= \min(c[1, 3], 1 + c[2, 0])$

$= \min(3, 1 + 0)$

$\mathbf{c[2, 3] = 1}$

$c[3, 3]$ with $i = 3, j = 3, d3 = 5$

$j < di$ i.e. $3 < 5$

$\therefore$ $c[i, j] = c[i - 1, j]$

$= c[3 - 1, 3]$

$= c[2, 3]$

$\mathbf{c[3, 3] = 1}$

$c[4, 3]$ with $i = 4, j = 3, d4 = 6$

As $j < d4$

$\therefore$ $c[i, j] = c[i - 1, j]$

$= c[4 - 1, 3]$

$= c[3, 3]$

$\mathbf{c[4, 3] = 1}$

Partially the table will be -

	0	1	2	3	4	5	6	7	8
d1 = 1	0	1	2	3					
d2 = 3	0	1	2	1					
d3 = 5	0	1	2	1					
d4 = 6	0	1	2	1					

$c[1, 4]$ with $i = 1, j = 4, d1 = 1$

As $i = 1$

$\therefore$ $c[i, j] = 1 + c[1, j - d1]$

$= 1 + c[1, 4 - 1]$

$= 1 + c[1, 3]$

$\mathbf{c[1, 4] = 4}$

$c[2, 4]$ with $i = 2, j = 4, d2 = 3$

As $i \neq 1$ and $j > d2$

$\therefore$ $c[i, j] = \min(c[i - 1, j], 1 + c[i, j - d_i])$

$= \min(c[2 - 1, 4], 1 + c[2, 4 - 3])$

$= \min(c[1, 4], 1 + c[2, 1])$

$= \min(4, 1 + 1)$

$\mathbf{c[2, 4] = 2}$

$c[3, 4]$ with $i = 3, j = 4, d3 = 5$

As $j < d_i$ i.e. $4 < 5$

$\therefore$ $c[i, j] = c[i - 1, j]$

$= c[3 - 1, 4]$

$= c[2, 4]$

$$c[3, 4] = 2$$

c [4, 4] with i = 4, j = 4, d4 = 6

As j < d4

$$\therefore \quad c[i, j] = c[i - 1, j] = c[4 - 1, 4]$$
$$= c[3, 4]$$
$$c[4, 4] = 2$$

c [1, 5] with i = 1, j = 5, d1 = 1.

As i = 1

$$c[i, j] = 1 + c[i, j - d1]$$
$$= 1 + c[1, 5 - 1]$$
$$= 1 + c[1, 4]$$
$$c[1, 5] = 5.$$

c [2, 5] with i = 2, j = 5, d2 = 3

Here i ≠ 1 and j > di

∴ Formula used

$$c[i, j] = \min(c[i - 1, j], 1 + c[i, j - di])$$
$$c[2, 5] = \min(c[2 - 1, 5], 1 + c[2, 5 - 3])$$
$$= \min(c[1, 5], 1 + c[2, 2])$$
$$= \min(5, 1 + 2)$$
$$c[2, 5] = 3$$

c [3, 5] with i = 3, j = 5, d3 = 5

$$c[3, 5] = \min(c[i - 1, j], 1 + c[i, j - di])$$
$$= \min(c[3 - 1, 5], 1 + c[3, 5 - 5])$$
$$= \min(c[2, 5], 1 + c[3, 0])$$
$$= \min(3, 1 + 0)$$
$$c[3, 5] = 1$$

c [4, 5] with i = 4, j = 5, d4 = 6.

As j < d4

$$\therefore \quad c[i, j] = c[i - 1, j] = c[4 - 1, 5]$$
$$= c[3, 5]$$
$$c[4, 5] = 1$$

c [1, 6] with i = 1, j = 6, d1 = 1

As i = 1

$$c[i, j] = 1 + c[i, j - d1]$$
$$= 1 + c[1, 6 - 1]$$
$$= 1 + c[1, 5]$$
$$c[1, 6] = 6$$

c [2, 6] with i = 2, j = 6, d2 = 3

Here i ≠ 1 and j > d2

$$c[i, j] = \min(c[i - 1, j], 1 + c[i, j - di])$$
$$= \min(c[2 - 1, 6], 1 + c[2, 6 - 3])$$
$$= \min(c[1, 6], 1 + c[2, 3])$$
$$= \min(6, 1 + 1)$$
$$c[2, 6] = 2$$

c [3, 6] with i = 3, j = 6, d3 = 5

∴

$$c[i, j] = \min(c[i - 1, j], 1 + c[i, j - di])$$
$$= \min(c[3 - 1, 6], 1 + c[3, 6 - 5])$$
$$= \min(c[2, 6], 1 + c[3, 1])$$
$$= \min(2, 1 + 1)$$
$$c[3, 6] = 2$$

c [4, 6] with i = 4, j = 6, d4 = 6

$$c[i, j] = \min(c[i - 1, j], 1 + c[i, j - di])$$
$$= \min(c[3, 6], 1 + c[4, 0])$$
$$= \min(2, 1 + 0)$$
$$c[4, 6] = 1$$

Continuing in this fashion, we get -

	0	1	2	3	4	5	6	7	8
d1 = 1	0	1	2	3	4	5	6	7	8
d2 = 3	0	1	2	1	2	3	2	3	4
d3 = 5	0	1	2	1	2	1	2	3	2
d4 = 6	0	1	2	1	2	1	1	2	2

↑

Total number of coins

The value c [4, 8] = 2 represents total number of coins to get sum of ₹. 8. The coins are,

d2 i.e. 3 units = 1 coin
+ d3 i.e. 5 units = 1 coin

8 units (₹) = 2 coins

Ex. 5.6.5 *Solve the following making change problem using dynamic programming method : Amount ₹ 7 and Denominations : (₹ 1, ₹ 2 and ₹ 4).*

Sol. : d1 = 1, d2 = 2 and d3 = 4 and N = 7

We will create a table with rows ranging from 1 to 3 and column ranging from 0 to 7.

Initially C[i,0]=0

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

		0	1	2	3	4	5	6	7
i = 1	d1 = 1	0							
i = 2	d2 = 2	0							
i = 3	d3 = 4	0							

Computations can be performed column by column and fill up the table accordingly.

We will use following formula

1. If i = 1 then C[i, j] = 1 + C[1, j – d1]
2. If j < di then C[i, j] = 1 + C[i – 1, j]
3. Otherwise C[i, j] = min(C[i –1, j], 1 + C[i, j – d1])

Step 1	C[1,1] with i = 1, j = 1,d1 = 1 As i = 1, **Formula Used :** 1 + C[1,j – d1] = 1 + C[1,1 – 1] = 1 + C[1,0] = 1 + 0 = 1 **C[1,1] = 1**	C[2,1] with i = 2, j = 1, d2 = 2 As j < d2 **Formula Used :** C[i – 1,j] = C[1,1] = 1 **C[2,1] = 1**	C[3,1] with i = 3, j = 1, d3 = 4 As j < d3 **Formula Used :** C[i – 1,j] = C[3 – 1,1] = C[2,1] **C[3,1] = 1**
Step 2	C[1,2] with i = 1, j = 2, d1 = 1 As i = 1 **Formula Used:** 1+C[1,j – d1] = 1+C[1,1] = 1 + 1 **C[1,2] = 2**	C[2,2] with i = 2,j = 2, d2 = 2 As j = d2 and i 1 **Formula Used:** min(C[i 1,j],1 + C[i,j di]) = min(C[1,2] , 1 + C[2,0] = min (2, 1 + 0) **C[2,2] = 1**	C[3,2] with i = 3, j = 2, d 3 = 4 As j < d3 **Formula Used :** C[i – 1,j] = C[3 – 1,2] = C[2,2] **C[3,2] = 1**

Step 3	C[1,3] with i = 1, j = 3,d1 = 1 As i = 1 **Formula Used :** 1+C[1,j – d1] = 1+C[1,2] = 1 + 2 **C[1,3] = 3**	C[2,3] with i = 2, j = 3, d2 = 2 As j > d3 **Formula Used :** = min(C[i – 1,j], 1+C[i,j di]) = min(C[1,3], 1+C[2,1]) = min (3, 2) **C[2,3] = 2**	C[3,3] with i = 3, j = 3, d 3 = 4 As j<d3 **Formula Used :** C[i – 1,j] = C[3 – 1,3] = C[2,3] **C[3,3] = 2**
Step 4	C[1,4] with i = 1, j = 4, d1 = 1 As i = 1 **Formula Used:** 1+C[1,j – d1] = 1+C[1,4 – 1] = 1 + C[1,3] **C[1,4] = 4**	C[2,4] with i = 2, j = 4, d2 = 2 As j > d2 **Formula Used :** = min(C[i – 1,j], 1+C[i,j – di]) = min(C[1,4], 1+C[2,2]) = min (4, 1+1) **C[2,4] = 2**	C[3,4] with i = 3, j = 4,d 3 = 4 As j = d3 and i **Formula Used :** = min(C[i – 1,j], 1+C[i,j – di]) = min(C[2,4], 1+C[3,0] = min (2,1+0) **C[3,4] = 1**
Step 5	C[1,5] with i = 1, j = 5, d1 = 1 As i = 1 Formula Used : 1+C[1,j – d1] = 1+C[1,5 – 1] = 1+C[1,4] **C[1,5] = 5**	C[2,5] with i = 2, j = 5, d2 = 2 As j > d2 **Formula Used :** = min(C[i – 1,j], 1+C[i,j-di]) = min(C[1,5], 1+C[2,3]) = min (5,1+2) **C[2,5] = 3**	C[3,5] with i = 3, j = 5, d3 = 4 As j > d2 **Formula Used :** = min(C[i – 1,j], 1+C[i, j – di]) = min(C[2,5], 1+C[3,1]) = min (3, 1+1) **C[3,5] = 2**
Step 6	C[1,6] with i = 1, j = 6,d1 = 1 As i = 1 **Formula Used :** 1+C[1,j – d1] = 1+C[1,6 – 1] = 1+C[1,5] **C[1,6] = 6**	C[2,6] with i = 2, j = 6,d2 = 2 As j > d2 **Formula Used :** = min(C[i – 1,j], 1+C[i,j – di]) = min(C[1,6], 1+C[2,4]) = min (6, 1+2) **C[2,6] = 3**	C[3,6] with i = 3, j = 6, d3 = 4 As j > d2 **Formula Used :** = min(C[i – 1,j], 1+C[i,j – di]) = min(C[2,6], 1+C[3,2]) = min (3, 1+2) **C[3,6] = 3**
Step 7	C[1,7] with i = 1, j = 7, d1 = 1 As i = 1 **Formula Used :** 1+C[1,j – d1] = 1+C[1,7 – 1] = 1+C[1,6] **C[1,7] = 7**	C[2,7] with i = 2, j = 7,d2 = 2 As j > d2 **Formula Used:** = min(C[i – 1,j], 1+C[i,j – di]) = min(C[1,7], 1+C[2,5]) = min (7, 1+3) **C[2,7] = 4**	C[3,7] with i = 3, j = 7,d3 = 4 As j > d2 **Formula Used :** = min(C[i – 1,j], 1+C[i,j – di]) = min(C[2,7], 1+C[3,3]) = min (4, 1+2) **C[3,7] = 3**

Thus we can represent the table filled up with above computations will be -

		0	1	2	3	4	5	6	7
i = 1	d1 = 1	0	1	2	3	4	5	6	7
i = 2	d2 = 2	0	1	1	2	2	3	3	4
i = 3	d3 = 4	0	1	2	2	1	2	3	3

↑

Total number of coins

The value C[n,N] i.e C[3,7] = 3 which represents total number of coins required to get the sum 7 are 3.

Thus we get

₹ 1 -> 1 coin

₹ 2-> 1 coin

₹ 3 -> 1 coin

Sum=7

```
Algorithm Number_of_Coins (N)
{
// Problem Description : This algorithm computes
//minimum number of coins required to make
// change for N units.
    for ( i ← 1 to n) do
    initialize array d[i] with the values of coins.
for ( i ← 1 to n ) do
        c [i, 0] ← 0          Initialising first column by 0
For ( i ← 1 to n ) do
{
    For ( i  1 to N ) do      Initialising first column by 0
    {
        if (i = 1 & & j < d [ i] ) then
            c [ i ] [ j ] = infinity ;
        else if ( i = 1 ) then
            c [ i ] [ j ] =  1+ c [ 1, j – d [1] ];
        else if ( j <d [ i ] ) then
```



```
            c [ i ] [ j ] =  c [j – 1, j];
        else
            c [ i ] [ j ] =  min ( c [i – 1, j], 1+ c [i, j – d [i]);
    }
}
return c [ n, N]          Total number of coins in the change
```

Analysis - As the basic operation in above algorithms is to fill up the table, which is performed within nested for loops. Hence the time complexity of this algorithm is Θ (nN).

Review Questions

Q.1 Explain coin change problem with suitable example.

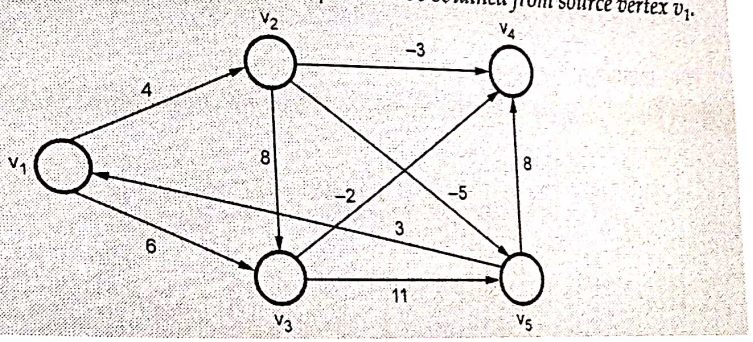
Q.2 Give an algorithm for coin change problem. Analyze it.

5.7 Bellman-Ford Algorithm

SPPU : May-19, April-18, Marks 10

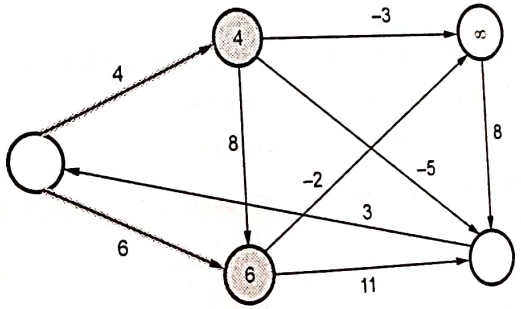
- The Bellman-Ford algorithm works for finding single source shortest path. It works for **negative** edge weights.
- This is a single source shortest path algorithm.
- Let us understand this algorithm with the help of some example.

Ex. 5.7.1 *Consider following for which the shortest path can be obtained from source vertex v_1.*

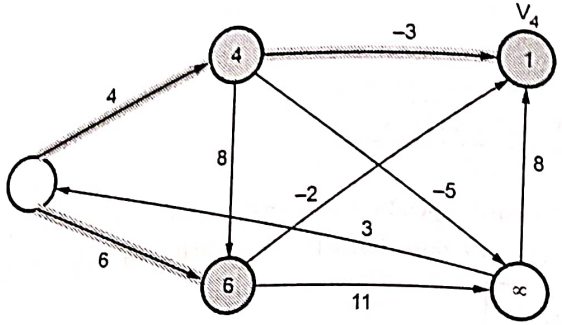


Sol. :

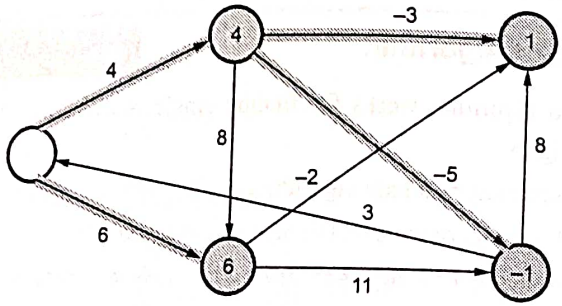
Step 1 : From vertex v_1 we find the distances to v_2 and v_3. The direct edges corresponds to the weights of destination vertices.



Step 2 : Modify vertex v_4 by its minimum weight i.e. 1. The path is shown by shaded area.

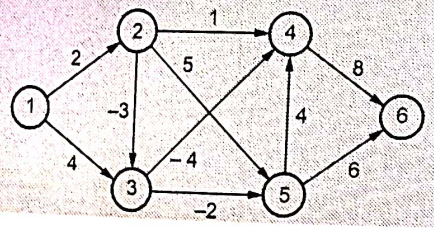


Step 3 : Process vertex v_5 by its minimum weight −1. The path is shown by shaded area.



Thus minimum distance of each vertex is obtained.

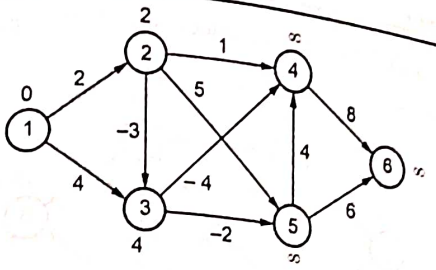
Ex. 5.7.2 *Find shortest path from node 1 to every other node in the graph as given below using Bellman and Ford Algorithm.*



Sol. : We have a source vertex 1. Only the direct edges to other v ertices are consider. The distances to remaining vertices are ∞.

We will write weights above each node.

Step 1 : The vertex 2 gives minimum distance so choose **vertex 2**



It can be represented as

1	2	3	4	5	6
0	2	4	∞	∞	∞

Step 2 : Now we will consider 2 as parent node and find distance to all other vertices

1	2	3	4	5	6
0	2	4	∞	∞	∞
0	2	−1	3	7	∞

For vertex 3
Dist (1, 2)
+ dist (2, 3)
= 2 + (− 3) = − 1

For vertex 4
dist (1, 2) + dist (3 4)
= 2 + 1 = 3

For vertex 5
dist (1, 2) + dist (2, 5)
= 2 + 5 = 7

Step 3 : Now consider 3 as parent node

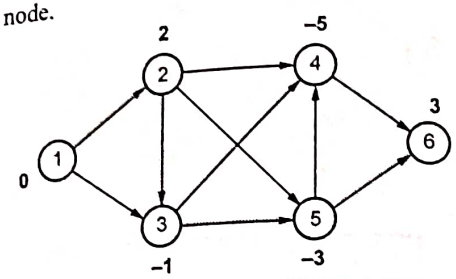
1	2	3	4	5	6
0	2	4	∞	∞	∞
0	2	−1	3	7	∞
0	2	−1	**−5**	**−3**	∞

The distance to vertex 4 and vertex 5 get updated, by considering node 3 as parent node

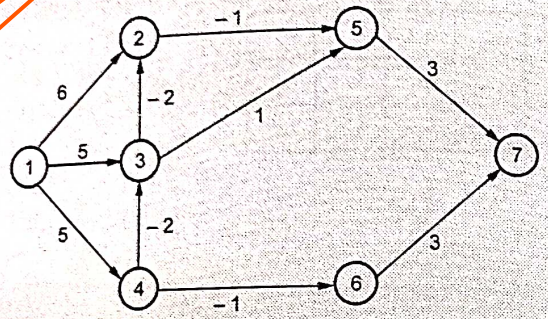
Step 4 : Consider node 4 as parent node shortest distance are then

1	2	3	4	5	6
0	2	4	∞	∞	∞
0	2	−1	3	7	∞
0	2	−1	−5	−3	∞
0	2	−1	−5	−3	3

Thus we have obtained all the shortest distances. The shortest path distances are shown in boldface above each node.

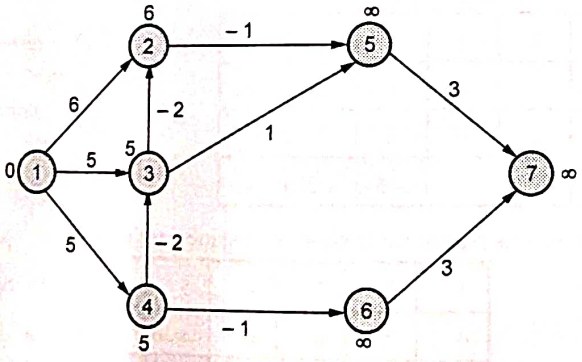


Ex. 5.7.3 Use Bellman Ford algorithm to find shortest path for the following graph. **SPPU : May-19, Marks 10**



Sol. : We have source vertex 1. Only direct edges of vertices are considered as distances. The distance to remaining vertices are ∞. We will write weights above each node.

Step 1 :



It can be represented as

1	2	3	4	5	6	7
0	6	5	5	∞	∞	∞

Now vertex 3 gives minimum distance 5. Hence choose vertex 3.

Step 2 : We will consider vertex 3 as parent node and find distance to all other vertices.

1	2	3*	4	5	6	7
0	6	5	5	∞	∞	∞
0	3	5	5	6	∞	∞

Step 3 : Here vertex 2 gives minimum distance

1	2*	3*	4	5	6	7
0	6	5	5	∞	∞	∞
0	3	5	5	6	∞	∞
0	3	5	5	5	∞	∞

Step 4 : Now vertex 4 gives minimum distance. Hence choose vertex 4.

1	2*	3*	4*	5	6	7
0	6	5	5	∞	∞	∞
0	3	5	5	6	∞	∞
0	3	5	5	5	∞	∞
0	3	3	5	4	4	∞

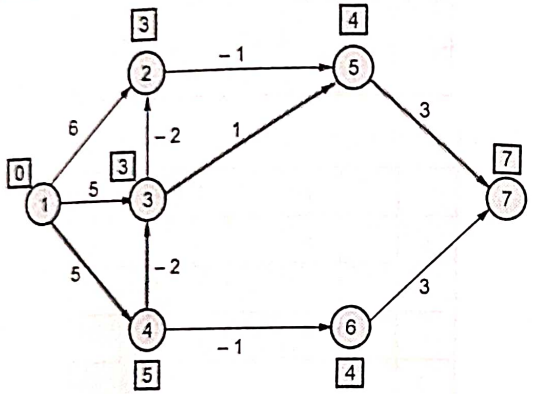
Step 5 : Here vertex 5 gives minimum distance. Hence choose vertex 5.

1	2*	3*	4*	5*	6	7
0	6	5	5	∞	∞	∞
0	3	5	5	6	∞	∞
0	3	5	5	5	∞	∞
0	3	3	5	4	4	∞
0	3	3	5	4	4	7

Step 6 : Here vertex 6 gives minimum distance. Hence choose vertex 6.

1	2*	3*	4*	5*	6*	7
0	6	5	5	∞	∞	∞
0	3	5	5	6	∞	∞
0	3	5	5	5	∞	∞
0	3	3	5	4	4	∞
0	3	3	5	4	4	7
0	3	3	5	4	4	7

Thus we have obtained all the shortest distances. The shortest path distances are shown in boldface above each node.



Algorithm

```
Algorithm Bellman Ford (vertices, edges, source)
{
// Problem Description : This algorithm finds
// the shortest path using Bellman Ford method
for (each vertex v)
{
    if (v is source) then
    v distance ← 0
    else
    v. distance ← infinity
    v. prede ← Null
```

Graphical initialization


```
}
for (i ← 1 to toal_ vertices – 1)
{
    for (each edge uv)
    {
    U ← uv. source
    V ← uv. desination
    if (v. distance > u. distance + uv. weight ) then
    {
        v. distance ← u. distance + uv. weight
        v. prede ← u
    }
}
for (each edge uv)
{
    u ← uv. source
    v ← uv. destination
    if (v. distance > u.distance + uv. weight) then
    {
        Write ("Graph has negative edges")
        return False
    }
}
} // end of for return True
} // end of algorithm
```

Newly obtained minimum distance

Relaxing edges

In above algorithm, we have used a term "**relaxing edges**". The process of relaxing edge (u, v) means testing whether we can get more shorter distance to vertex v from u. The relaxation step **decrease** the value of the shortest path estimated earlier and modify the predecessors of vertex v. The running time of Bellman-Ford algorithm is O(nm).

Review Question

Q.1 Write Bellman ford algorithm to find shortest path and analyze it.

SPPU : April-18, In Sem. Marks 5

5.8 Multistage Graph Problem

SPPU : May-18, April-19, Marks 10

A multistage graph G = (V, E) which is a directed graph. In this graph all the vertices are partitioned into the k stages where k ≥ 2. In multistage graph problem we have to find the

shortest path from source to sink. The cost of each path is calculated by using the weight given along that edge. The cost of a path from source (denoted by S) to sink (denoted by T) is the sum of the costs of edges on the path. In multistage graph problem we have to find the path from S to T. There is set of vertices in each stage. The multistage graph can be solved using forward and backward approach.

Let us solve multistage problem for both the approaches with the help of some example.

Consider the graph G as shown in the Fig. 5.8.1.

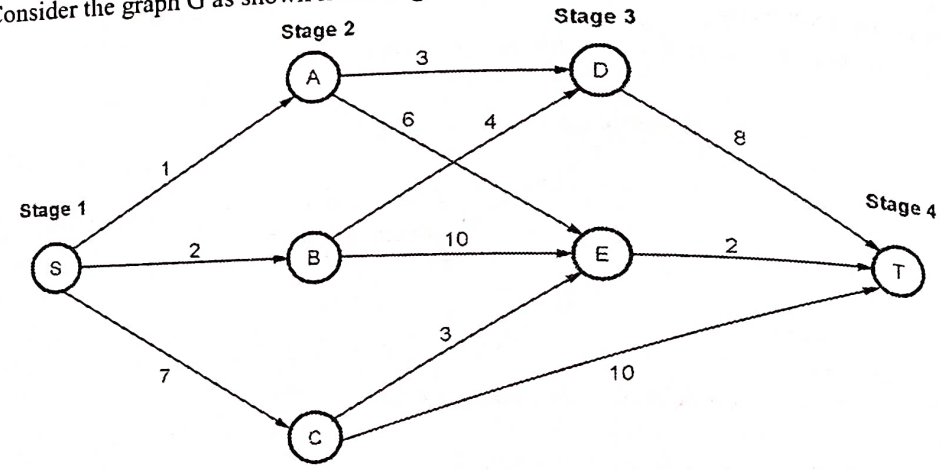


Fig. 5.8.1 Multistage graph

There is single vertex in stage 1, then 3 vertices in stage 2, then 2 vertices in stage 3 and only one vertex in stage 4 (this is a target stage).

Backward approach

$$d(S, T) = \min \{1 + d(A, T), 2 + d(B, T), 7 + d(C, T)\} \quad \ldots (1)$$

We will now compute d(A, T), d(B, T) and d(C,T).

$$d(A, T) = \min \{3 + d(D, T), 6 + d(E, T)\} \quad \ldots (2)$$

$$d(B, T) = \min \{4 + d(D, T), 10 + d(E, T)\} \quad \ldots (3)$$

$$d(C, T) = \min \{3 + d(E, T), d(C, T)\} \quad \ldots (4)$$

Now let us compute d(D, T) and d(E, T).

$$\mathbf{d(D, T) = 8}$$

$$\mathbf{d(E, T) = 2} \qquad \textbf{backward vertex = E}$$

Let us put these values in equations (2), (3) and (4).

$$d(A, T) = \min \{3 + 8, 6 + 2\}$$

$$\mathbf{d(A, T) = 8} \qquad \textbf{A - E - T}$$

$$d(B, T) = \min \{4 + 8, 10 + 2\}$$

$$\mathbf{d(B, T) = 12} \qquad \textbf{A - D - T}$$

$$d(C, T) = \min \{3 + 2, 10\}$$

$$\mathbf{d(C, T) = 5} \qquad \textbf{C - E - T}$$

$$\therefore \quad d(S, T) = \min \{1 + d(A, T), 2 + d(B, T), 7 + d(C, T)\}$$

$$= \min \{1 + 8, 2 + 12, 7 + 5\}$$

$$= \min \{9, 14, 12\}$$

$$d(S, T) = 9 \qquad \textbf{S - A - E - T}$$

Forward approach

$$d(S, A) = 1$$

$$d(S, B) = 2$$

$$d(S, C) = 7$$

$$d(S, D) = \min \{1 + d(A, D), 2 + d(B, D)\}$$

$$= \min \{1 + 3, 2 + 4\}$$

$$\mathbf{d(S, D) = 4}$$

$$d(S, E) = \min \{1 + d(A, E), 2 + d(B, E), 7 + d(C, E)\}$$

$$= \min \{1 + 6, 2 + 10, 7 + 3\}$$

$$= \min \{7, 12, 10\}$$

$$\mathbf{d(S, E) = 7} \qquad \textbf{i.e. Path S - A - E is chosen.}$$

$$d(S, T) = \min \{d(S, D) + d(D, T), d(S, E) + d(E, T), d(S, C) + d(C, T)\}$$

$$= \min \{4 + 8, 7 + 2, 7 + 10\}$$

$$d(S, T) = 9 \qquad \textbf{i.e. Path S - E, E - T is chosen.}$$

∴ The minimum cost = 9 with the path **S - A - E - T.**

This method is called **forward reasoning.**

The algorithm for forward approach is –

```
Algorithm Forward_Gr (G, stages, n, p)
//  Problem description : This algorithm is for
//  forward  approach of multistage graph
//  Input : The multistage graph G = (V, E),
//  'Stages' is the variable representing number of stages
//  n is total number of vertices of G
//  p is an array for restoring path
```

```
//  Output : The path with minimum cost
        cost[i] ← 0
    for (i ← n – 2 downto 0)
    {
        r ← Get_min (i,n) //      r is an edge with min cost
        cost[i] ← c[i][r] + cost[r]
            d[i] ← r
    }
    //  finding minimum cost path
        p[0] ← 0
        p[stages – 1] ← n – 1
    for (i ← 1 to stages–1) do
        p[i] ← d[p[i+1]] ;
```

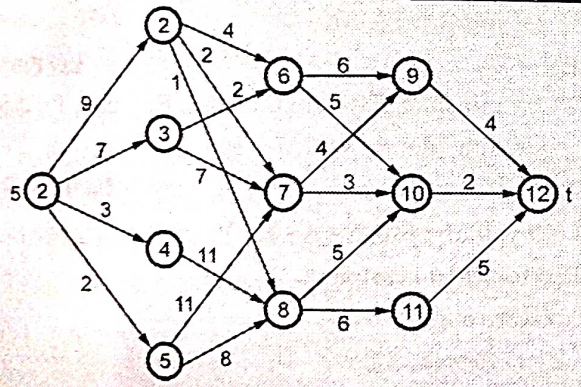
Analysis : The algorithm has Θ (|V|+|E|) time complexity.

Using dynamic programming strategy, the multistage graph problem is solved. This is because in multistage graph problem we obtain the minimum path at each current stage by considering the path length of each vertex obtained in earlier stage. Thus the sequence of decisions are taken by considering overlapped solution.

In dynamic programming, we may get any number of solutions for given problem. From all these solutions we seek for optimal solution, finally optimal solution becomes the solution to given problem. Multistage graph problem is solved using this same approach.

Ex. 5.8.1 Find the minimum cost path from source (s) to sink (t) of the following multistage graph.

SPPU : May-18, End Sem Marks 10



Sol. : Step 1 :

$d(1,2) = 9$

$d(1,3) = 7$

$d(1,4) = 3$

$d(1,5) = 2$

Step 2 :

$d(1,6) = \min\{(9 + d(2,6)), (7 + d(3,6))\}$

$= \min\{(9 + 4), (7+2)\}$

$\boxed{d(1,6) = 9}$ Here we choose **vertex 3** as minimum distance source.

$d(1,7) = \min\{(9 + d(2,7)), (7 + d(3,7)), (2 + d(5,7))\}$

$= \min\{(9 + 2), (7 + 7), (2 + 11)\}$

$\boxed{d(1,7) = 11}$ Here we choose **vertex 2** as minimum distance source.

$d(1,8) = \min\{(9 + d(2,8)), (3 + d(4,8)), (2 + d(5,8))\}$

$= \min\{(9 + 1), (3 + 11), (2 + 8)\}$

$\boxed{d(1,8) = 10}$

Here we choose either **vertex 2** or **vertex 5** as minimum distance source.

Step 3 :

$d(1,9) = \min\{((1,6) + d(6,9)), ((1,7) + d(7,9))\}$

$= \min\{(9 + 6), (11 + 4)\}$

$\boxed{d(1,9) = 15}$

Here we choose **vertex 6** or **vertex 7** as minimum distance source.

$d(1,10) = \min\{((d(1,6) + d(6, 10)), (d(1,7) + d(7,10)), (d(1,8) + d(8,10))\}$

$= \min\{(9 + 5), (11 + 3), (10 + 5)\}$

$\boxed{d(1,10) = 14}$

Here we choose **vertex 6** or **vertex 7** as minimum distance source.

$d(1,11) = (d(1,8) + d(8,11))$

$= (10 + 6)$

$\boxed{d(1,11) = 16}$

Step 4 :

$d(1,12) = \min\{(d(1,9) + d(9,12)), (d(1,10) + d(10,12)),$

$(d(1,11) + d(11,12))\}$

$= \min\{(15 + 4), (14 + 2), (16 + 5)\}$

$\boxed{d(1,12) = 16}$

Here we choose **vertex 10** as minimum distance vertex.

Step 5 : From step 1 to step 4 we can obtain minimum distance as -

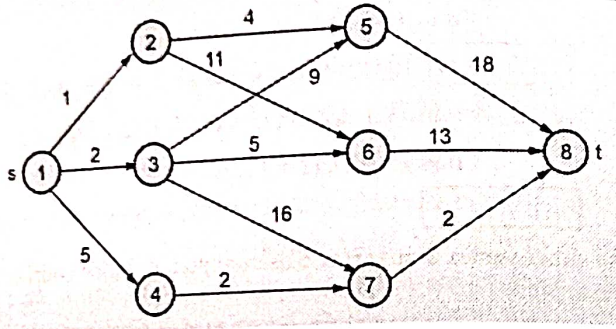
1 – 3 – 6 – 10 – 12

1 – 2 – 7 – 10 – 12

The **minimum cost = 16**

Ex. 5.8.2 Find minimum cost path from source (s) to sink (t) of the following multistage graph.

SPPU : April-19, In Sem, Marks 5



Solution :

At stage 1, we will consider vertex 1 as source 's'

At stage 2, we will consider vertex 2, 3 and 4 as source.

At stage 3, we will consider vertex 5, 6 and 7 as source.

At stage 4, we will consider vertex 8 as target 't'

Now, using forward approach we will find **d(s,t)**

Step 1 : We will first obtain the direct distances from source 's'.

$\therefore$ $d(1,2) = 1$

$d(1,3) = 2$

$d(1,4) = 5$

Step 2 :

$d(1,5) = \min\{(1 + d(2,5)), (2 + d(3,5))\}$

$= \min\{(1 + 4), (2 + 9)\}$

$\boxed{d(1,5) = 5}$ Here **vertex 2** gives us minimum diatance.

$d(1,6) = \min\{(1 + d(2,6)), (2 + d(3,6))\}$

$= \min\{(1 + 11), (2 + 5)\}$

$= \min\{12,7\}$

$\boxed{d(1,6) = 7}$ Here **vertex 3** gives us minimum diatance.

$d(1,7) = \min\{(2 + d(3,7)), (5 + d(4,7))\}$

$= \min\{(2 + 16), (5 + 2)\}$

$\boxed{d(1,7) = 7}$ Here **vertex 4** gives us minimum diatance.

Step 3 :

$d(1,8) = \min\{(5 + d(5,8)), (7 + d(6,8)), (7 + d(7,8))\}$

$= \min\{(5 + 18), (7 + 13), (7 + 2)\}$

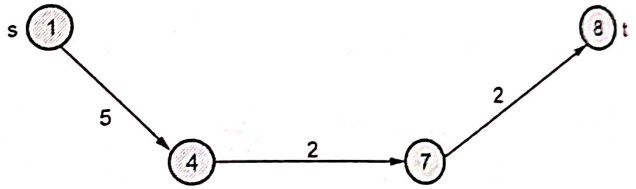
$\boxed{d(1,8) = 9}$ Here **vertex 7** gives us minimum diatance.

Step 4 :

To reach to target node 8 (i.e. t) we choose **vertex 7** which gives us minimum distance.

From step 2, we observe the d(1,7). For minimum distance of this path, we have chosen vertex 4.

Hence we can conclude, the minimum path as 1 – 4 – 7 – 8 i.e. s – 4 – 7 – t



Minimum cost = 9

5.9 Travelling Salesman Problem

SPPU : May-08,12,13,15, 18, Dec.-11, 18, Marks 10

- **Problem Description**

  Let G be directed graph denoted by (V, E) where V denotes set of vertices and E denotes set of edges. The edges are given along with their cost C_{ij}. The cost $C_{ij} > 0$ for all i and j. If there is no edge between i and j then $C_{ij} = \infty$.

- A tour for the graph should be such that all the vertices should be visited only once and cost of the tour is sum of cost of edges on the tour.
- The traveling salesperson problem is to find the tour of minimum cost.
- Dynamic programming is used to solve this problem.

Step 1 : Let the function C(1, V–{1}) is the total length of the tour terminating at 1. The objective of TSP problem is that the cost of this tour should be minimum.

Let d[i, j] be the shortest path between two vertices i and j.

Step 2 : Let V_1, $V_2 \ldots V_n$ be the sequence of vertices followed in optimal tour. Then $(V_1, V_2, \ldots V_n)$ must be a shortest path from V_1 to V_n which passes through each vertex exactly once.

Here the principle of optimality is used. The path V_i, $V_{i+1}, \ldots V_j$ must be optimal for all paths beginning at V(i), ending at V(j), and passing through all the intermediate vertices $\{V_{(i+1)} .. V_{(j-1)}\}$ once.

Step 3 : Following formula can be used to obtain the optimum cost tour.

$Cost(i, S) = \min \{d[i, j] + Cost(j, S-\{j\})\}$ where $j \in S$ and $i \notin S$.

Consider one example to understand solving of TSP using dynamic programming approach.

Ex. 5.9.1 *For the given diagraph, obtain optimum cost tour.* **SPPU : May-08, Marks 10, May-13, May-18 End. Sem., Dec.-18, End Sem., Marks 8**

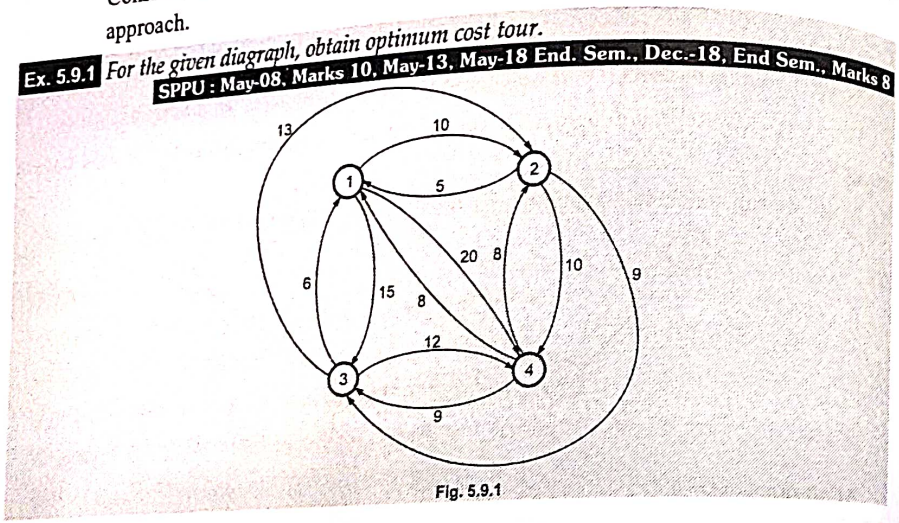


Fig. 5.9.1

Sol. : The distance matrix can be given by,

to → / from ↓	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

First we will select any arbitrary vertex say select 1.

Now process for intermediate sets with increasing size.

Step 1 : Let $S = \phi$ then,

$$Cost(2, \phi, 1) = d(2, 1) = 5$$

$$Cost(3, \phi, 1) = d(3, 1) = 6$$

$$Cost(4, \phi, 1) = d(4, 1) = 8$$

That means we have obtained dist (2, 1), dist (3, 1) and dist (4, 1).

Candidate (S) = 1

Step 2 :

Apply the formula, $Cost(i, S) = \min \{d[i, j] + Cost(j, S - \{j\})\}$

Hence from vertex 2 to 1, vertex 3 to 1 and vertex 4 to 1 by considering intermediate path lengths we will calculate total optimum cost.

$$Cost(2, \{3\}, 1) = d(2, 3) + Cost(3, \phi, 1)$$
$$= 9 + 6$$
$$Cost(2, \{3\}, 1) = 15$$
$$Cost(2, \{4\}, 1) = d(2, 4) + Cost(4, \phi, 1)$$
$$= 10 + 8$$
$$Cost(2, \{4\}, 1) = 18$$
$$Cost(3, \{2\}, 1) = d(3, 2) + Cost(2, \phi, 1)$$
$$= 13 + 5$$
$$Cost(3, \{2\}, 1) = 18$$
$$Cost(3, \{4\}, 1) = d(3, 4) + Cost(4, \phi, 1)$$
$$= 12 + 8$$
$$Cost(3, \{4\}, 1) = 20$$
$$Cost(4, \{2\}, 1) = d(4, 2) + Cost(2, \phi, 1)$$
$$= 8 + 5$$
$$Cost(4, \{2\}, 1) = 13$$
$$Cost(4, \{3\}, 1) = d(4, 3) + Cost(3, \phi, 1)$$
$$= 9 + 6$$
$$Cost(4, \{3\}, 1) = 15$$

Step 3 : Consider candidate (S) = 2

$$Cost(2, \{3, 4\}, 1) = \min \{[d(2, 3) + Cost(3, \{4\}, 1)], [d(2, 4) + Cost(4, \{3\}, 1)]\}$$
$$= \min \{[9 + 20], [10+15]\}$$
$$Cost(2, \{3, 4\}, 1) = 25$$
$$Cost(3, \{2, 4\}, 1) = \min \{[d(3, 2) + Cost(2, \{4\}, 1)], [d(3, 4) + Cost(4, \{2\}, 1)]\}$$
$$= \min \{[13+18], [12+13]\}$$

$$\text{Cost}(3, \{2, 4\}, 1) = 25$$

$$\text{Cost}(4, \{2, 3\}, 1) = \min \{[d(4, 2) + \text{Cost}(2, \{3\}, 1)], [d(4, 3) + \text{Cost}(3, \{2\}, 1)]\}$$
$$= \min \{[8 + 15], [9 + 18]\}$$

$$\text{Cost}(4, \{2, 3\}, 1) = 23$$

Step 4 : Consider candidate (S) = 3. i.e. Cost (1, {2, 3, 4}) but as we have chosen vertex 1 initially the cycle should be completed i.e. starting and ending vertex should be 1.

∴ We will compute,

$$\text{Cost}(1, \{2, 3, 4\}, 1) = \left\{ \begin{matrix} [d(1, 2) + \text{Cost}(2, \{3, 4\}, 1)], [d(1, 3) + \text{Cost}(3, \{2, 4\}, 1] \\ [d(1, 4) + \text{Cost}(4, \{2, 3\}, 1)] \end{matrix} \right\}$$
$$= \min([10 + 25], [15 + 25], [20 + 23])$$
$$= 35$$

Thus the optimal tour is of path length 35.

Consider step 4 now, from vertex 1 we obtain the optimum path as d(1, 2). Hence select vertex 2. Now consider step 3, in which from vertex 2 we obtain optimum cost from d(2, 4). Hence select vertex 4. Now in step 2 we get remaining vertex 3 as d(4, 3) is optimum. Hence optimal tour is 1, 2, 4, 3, 1.

Ex. 5.9.2 *For a directed graph the edge length matrix is given below. Solve the travelling salesperson problem, for this 4 city graph using dynamic programming method. What will be the time complexity for n city TSP problem solved using this method.*

	1	2	3	4
1	0	9	8	8
2	12	0	13	6
3	10	9	0	5
4	20	15	10	0

SPPU : May-12, Marks 7

Sol. : We will start with vertex 1.

We will now process for intermediate sets with increasing size.

Step 1 :

Let $S = \phi$ then

$$\text{cost}(2, \phi, 1) = d(2, 1) = 12$$
$$\text{cost}(3, \phi, 1) = d(3, 1) = 10$$
$$\text{cost}(4, \phi, 1) = d(4, 1) = 20$$

That means we have obtained distance (2, 1), (3, 1) and distance (4, 1).

Step 2 :

Candidate (S) = 1

Now we will apply formula,

$$\text{cost} = \min$$

Hence from vertex 2 to 1, vertex 3 to 1 and vertex 4 to 1 by considering intermediate path lengths we will calculate total optimal cost.

$$\text{cost}(2, \{3\}, 1) = d(2, 3) + \text{cost}(3, \phi, 1)$$
$$= 13 + 10$$
$$\textbf{cost (2, \{3\}, 1) = 23}$$

$$\text{cost} - (2, \{4\}, 1) = d(2, 4) + \text{cost}(4, \phi, 1)$$
$$= 6 + 20$$
$$\textbf{cost (2, \{4\}, 1) = 26}$$

$$\text{cost}(3, \{2\}, 1) = d(3, 2) + \text{cost}(2, \phi, 1)$$
$$= 9 + 12$$
$$\textbf{cost (3, \{2\}, 1) = 21}$$

$$\text{cost}(3\{4\}, 1) = d(3, 4) + \text{cost}(4, \phi, 1)$$
$$= 5 + 20$$
$$\textbf{Cost (3,\{4\}, 1) = 25}$$

$$\text{cost}(4, \{2\}, 1) = d(4, 2) + \text{cost}(2, \phi, 1)$$
$$= 15 + 12$$
$$\textbf{cost (4, \{2\}, 1) = 27}$$

$$\text{cost}(4, \{3\}, 1) = d(4, 3) + \text{cost}(3, \phi, 1)$$
$$= 10 + 10$$
$$\textbf{Cost (4, \{3\}, 1) = 20}$$

Step 3 : Consider candidate (S) = 2

$$\text{cost}(2, \{3, 4\}, 1) = \min \{d[2, 3] + \text{cost}(3, \{4\}, 1), d[2,4] + \text{cost}(4, \{3\}, 1)\}$$
$$= \min \{13 + 25, 6 + 20\}$$
$$= 26$$

$$\text{Cost}(3, \{2, 4\}, 1) = \min \{d[3,2] + \text{cost}(2, \{4\}, 1), d[3,4] + \text{cost}(4, \{2\}, 1)\}$$
$$= \min \{9 + 26, 5 + 27\}$$

$$\text{Cost}(3, \{2, 4\}, 1) = 32$$

$$\text{Cost}(4, \{2, 3\}, 1) = \min \{d[4, 2] + \text{cost}(2, \{3\}, 1), d[4, 3] + \text{cost}(3, \{2\}, 1)\}$$
$$= \min \{15 + 23, 10 + 21\}$$

$$\text{Cost}(4, \{2, 3\}, 1) = 31$$

Step 4 : Consider candidate (S) = 3. Hence the cost will be cost (1, {2, 3, 4}). But we have chosen vertex 1 initially the cycle should be completed.

$\therefore$ we will compute

$$\text{cost}(1, \{2,3,4\}, 1) = \min \begin{Bmatrix} d[1,2] + \text{cost}(2, \{3,4\}, 1), \\ d[1,3] + \text{cost}(3, \{2,4\}, 1), \\ d[1,4] + \text{cost}(4, \{2,3\}, 1) \end{Bmatrix} = \min(9+26,\ 8+32,\ 8+31)$$

$$= \min(9+26,\ 8+32,\ 8+31)$$

$$= \min(35, 40, 39) = 35$$

Thus the optimal tour is of cost 35.

Consider step 4 now, from vertex 1 we obtain the optimum path as d (1, 2). Hence select vertex **2.**

Now consider step 3, in which from vertex 2 we obtain optimum cost from d [2, 4] hence select vertex **4.**

In step 2, we get remaining vertex as 3 as d(4, 3) is optimum. Hence optimal tour sequence is **1-2-4-3-1.**

The total time required by the algorithm to execute is O ($nm - m^2$) which turns out to be O (n^3)

Review Questions

Q.1 Explain how dynamic programming is used to obtain optimal solution for travelling salesperson problem. Also explain why this technique is not used to solve TSP for large number of cities. **SPPU : Dec.-11, Marks 6**

Q.2 What is travelling salesperson problem ? How it can be solved by dynamic programming strategy ? **SPPU : May-15 In Sem, Marks 5**

5.10 Multiple Choice Questions

Q. 1 Which method can be used when the solution to a problem can not be viewed as the result of a sequence of decisions?

[a] Greedy method. [b] Divide and conquer.

[c] Dynamic Programming. [d] Backtracking.

Q. 2 The _____ states that whenever the initial state and decision are given, the remaining decisions must constitute an optimal decision sequence.

[a] principle of locality [b] principle of optimality

[c] principle of backtracking [d] none of the above

Q. 3 The principle of optimality is used by _____.

[a] brute force [b] divide and conquer

[c] dynamic programming [d] backtracking

Q. 4 The goal is obtained at _____ in 0/1 Knapsack using dynamic programming

[a] table[0,0] [b] table[0,n]

[c] table[n,w] [d] table[n,n]

Q. 5 The solution for the given of Knapsack is _____.

M=6 n=3

(p1, p2, p3)=(1,2,5)

(w1, w2, w3)=(2,3,4)

[a] 011 [b] 101 [c] 001 [d] 010

Answer Keys for Multiple Choice Questions

Q.1	c	Q.2	b	Q.3	c	Q.4	c	Q.5	b