

**Group A****Assignment No: 4**

**Title of the Assignment:** Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

**Objective of the Assignment:** Students should be able to understand and solve 0-1 Knapsack problem using dynamic programming

**Prerequisite:**

1. Basic of Python or Java Programming
  2. Concept of Dynamic Programming
  3. 0/1 Knapsack problem
- 

**Contents for Theory:**

1. Greedy Method
  2. 0/1 Knapsack problem
  3. Example solved using 0/1 Knapsack problem
-

## What is Dynamic Programming?

- Dynamic Programming is also used in optimization problems. Like divide-and-conquer method, Dynamic Programming solves problems by combining the solutions of subproblems.
- Dynamic Programming algorithm solves each sub-problem just once and then saves its answer in a table, thereby avoiding the work of re-computing the answer every time.
- Two main properties of a problem suggest that the given problem can be solved using Dynamic Programming. These properties are **overlapping sub-problems and optimal substructure**.
- Dynamic Programming also combines solutions to sub-problems. It is mainly used where the solution of one sub-problem is needed repeatedly. The computed solutions are stored in a table, so that these don't have to be re-computed. Hence, this technique is needed where overlapping sub-problem exists.
- For example, Binary Search does not have overlapping sub-problem. Whereas recursive program of Fibonacci numbers have many overlapping sub-problems.

## Steps of Dynamic Programming Approach

Dynamic Programming algorithm is designed using the following four steps –

- Characterize the structure of an optimal solution.
- Recursively define the value of an optimal solution.
- Compute the value of an optimal solution, typically in a bottom-up fashion.
- Construct an optimal solution from the computed information.

## Applications of Dynamic Programming Approach

- Matrix Chain Multiplication
- Longest Common Subsequence
- Travelling Salesman Problem

## Knapsack Problem

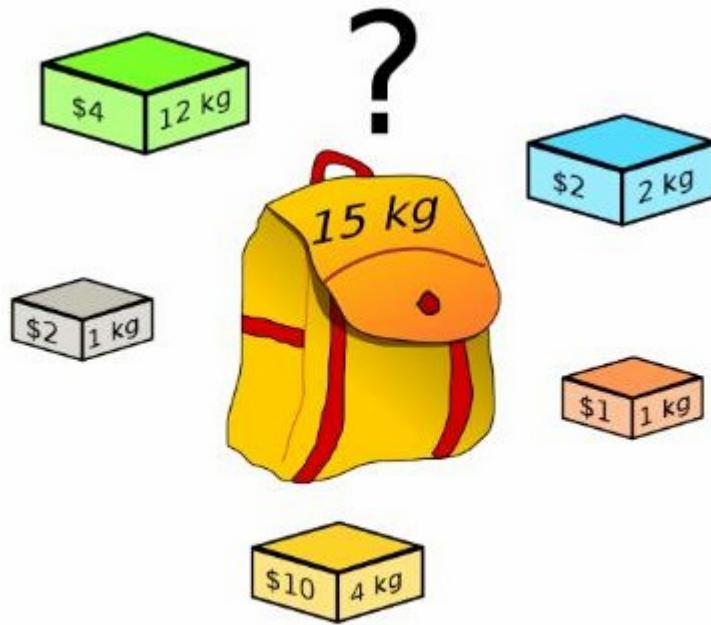
You are given the following-

- A knapsack (kind of shoulder bag) with limited weight capacity.
- Few items each having some weight and value.

The problem states-

Which items should be placed into the knapsack such that-

- The value or profit obtained by putting the items into the knapsack is maximum.
- And the weight limit of the knapsack does not exceed.



**Knapsack Problem**

## Knapsack Problem Variants

Knapsack problem has the following two variants-

1. Fractional Knapsack Problem
2. 0/1 Knapsack Problem

## 0/1 Knapsack Problem-

In 0/1 Knapsack Problem,

- As the name suggests, items are indivisible here.
- We can not take a fraction of any item.
- We have to either take an item completely or leave it completely.
- It is solved using a dynamic programming approach.

## 0/1 Knapsack Problem Using Greedy Method-

Consider-

- Knapsack weight capacity =  $w$
- Number of items each having some weight and value =  $n$

**0/1 knapsack problem is solved using dynamic programming in the following steps-**

### Step-01:

- Draw a table say „T“ with  $(n+1)$  number of rows and  $(w+1)$  number of columns.
- Fill all the boxes of  $0^{\text{th}}$  row and  $0^{\text{th}}$  column with zeroes as shown-

	0	1	2	3	.....	W
0	0	0	0	0	.....	0
1	0					
2	0					
.....						
n	0					

**T-Table**

**Step-02:**

Start filling the table row wise top to bottom from left to right.

Use the following formula-

$$T(i, j) = \max \{ T(i-1, j), \text{value}_i + T(i-1, j - \text{weight}_i) \}$$

Here,  $T(i, j)$  = maximum value of the selected items if we can take items 1 to i and have weight restrictions of j.

- This step leads to completely filling the table.
- Then, value of the last box represents the maximum possible value that can be put into the knapsack.

**Step-03:**

- To identify the items that must be put into the knapsack to obtain that maximum profit,
- Consider the last column of the table.
- Start scanning the entries from bottom to top.
- On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.
- After all the entries are scanned, the marked labels represent the items that must be put into the knapsack

Problem-.

For the given set of items and knapsack capacity = 5 kg, find the optimal solution for the 0/1 knapsack problem making use of a dynamic programming approach.

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

$$n = 4$$

$$w = 5 \text{ kg}$$

$$(w_1, w_2, w_3, w_4) = (2, 3, 4, 5)$$

$$(b_1, b_2, b_3, b_4) = (3, 4, 5, 6)$$

## Solution-

### Given

- Knapsack capacity ( $w$ ) = 5 kg
- Number of items ( $n$ ) = 4

### Step-01:

- Draw a table say „T“ with  $(n+1) = 4 + 1 = 5$  number of rows and  $(w+1) = 5 + 1 = 6$  number of columns.
- Fill all the boxes of 0<sup>th</sup> row and 0<sup>th</sup> column with 0.

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

**T-Table**

### Step-02:

Start filling the table row wise top to bottom from left to right using the formula-

$$T(i, j) = \max \{ T(i-1, j), \text{value}_i + T(i-1, j - \text{weight}_i) \}$$

#### Finding $T(1,1)$ :

We have,

- $i = 1$
- $j = 1$
- $(\text{value})_i = (\text{value})_1 = 3$
- $(\text{weight})_i = (\text{weight})_1 = 2$

Substituting the values, we get-

$$T(1,1) = \max \{ T(1-1, 1), 3 + T(1-1, 1-2) \}$$

$$T(1,1) = \max \{ T(0,1), 3 + T(0,-1) \}$$

$$T(1,1) = T(0,1) \{ \text{Ignore } T(0,-1) \}$$

$$T(1,1) = 0$$

### Finding T(1,2)-

We have,

- $i = 1$
- $j = 2$
- $(\text{value})_i = (\text{value})_1 = 3$
- $(\text{weight})_i = (\text{weight})_1 = 2$

Substituting the values, we get-

$$T(1,2) = \max \{ T(1-1, 2), 3 + T(1-1, 2-2) \}$$

$$T(1,2) = \max \{ T(0,2), 3 + T(0,0) \}$$

$$T(1,2) = \max \{ 0, 3+0 \}$$

$$T(1,2) = 3$$

### Finding T(1,3)-

We have,

- $i = 1$
- $j = 3$
- $(\text{value})_i = (\text{value})_1 = 3$
- $(\text{weight})_i = (\text{weight})_1 = 2$

Substituting the values, we get-

$$T(1,3) = \max \{ T(1-1, 3), 3 + T(1-1, 3-2) \}$$

$$T(1,3) = \max \{ T(0,3), 3 + T(0,1) \}$$

$$T(1,3) = \max \{ 0, 3+0 \}$$

$$T(1,3) = 3$$

- $i = 1$
- $j = 4$
- $(\text{value})_i = (\text{value})_1 = 3$
- $(\text{weight})_i = (\text{weight})_1 = 2$

$$T(1,4) = \max \{ T(1-1, 4), 3 + T(1-1, 4-2) \}$$

$$T(1,4) = \max \{ T(0,4), 3 + T(0,2) \}$$

$$T(1,4) = \max \{ 0, 3+0 \}$$

$$T(1,4) = 3$$

**Finding T(1,5)-**

We have,

- $i = 1$
- $j = 5$
- $(\text{value})_i = (\text{value})_1 = 3$
- $(\text{weight})_i = (\text{weight})_1 = 2$

Substituting the values, we get-

$$T(1,5) = \max \{ T(1-1, 5), 3 + T(1-1, 5-2) \}$$

$$T(1,5) = \max \{ T(0,5), 3 + T(0,3) \}$$

$$T(1,5) = \max \{ 0, 3+0 \}$$

$$T(1,5) = 3$$

**Finding T(2,1)-**

We have,

- $i = 2$
- $j = 1$
- $(\text{value})_i = (\text{value})_2 = 4$
- $(\text{weight})_i = (\text{weight})_2 = 3$

Substituting the values, we get-

$$T(2,1) = \max \{ T(2-1, 1), 4 + T(2-1, 1-3) \}$$

$$T(2,1) = \max \{ T(1,1), 4 + T(1,-2) \}$$

$$T(2,1) = T(1,1) \{ \text{Ignore } T(1,-2) \}$$

$$T(2,1) = 0$$

- $i = 2$
- $j = 2$
- $(\text{value})_i = (\text{value})_2 = 4$
- $(\text{weight})_i = (\text{weight})_2 = 3$

Substituting the values, we get-

$$T(2,2) = \max \{ T(2-1, 2), 4 + T(2-1, 2-3) \}$$

$$T(2,2) = \max \{ T(1,2), 4 + T(1,-1) \}$$

$$T(2,2) = T(1,2) \{ \text{Ignore } T(1,-1) \}$$

$$T(2,2) = 3$$

- $i = 2$
- $j = 3$
- $(\text{value})_i = (\text{value})_2 = 4$
- $(\text{weight})_i = (\text{weight})_2 = 3$

Substituting the values, we get-

$$T(2,3) = \max \{ T(2-1, 3), 4 + T(2-1, 3-3) \}$$

$$T(2,3) = \max \{ T(1,3), 4 + T(1,0) \}$$

$$T(2,3) = \max \{ 3, 4+0 \}$$

$$T(2,3) = 4$$

Similarly, compute all the entries.

After all the entries are computed and filled in the table, we get the following 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

### T-Table

- The last entry represents the maximum possible value that can be put into the knapsack.
- So, maximum possible value that can be put into the knapsack = 7.

### IdentifyingItemsToBePutIntoKnapsack

Following Step-04,

- We mark the rows labelled “1” and “2”.
- Thus, items that must be put into the knapsack to obtain the maximum value 7 are-

### Item-1 and Item-2

### TimeComplexity-

- Each entry of the table requires constant time  $\theta(1)$  for its computation.
- It takes  $\theta(nw)$  time to fill  $(n+1)(w+1)$  table entries.
- It takes  $\theta(n)$  time for tracing the solution since tracing process traces the n rows.
- Thus, overall  $\theta(nw)$  time is taken to solve 0/1 knapsack problem using dynamic programming

**Conclusion-**In this way we have explored Concept of 0/1 Knapsack using Dynamic approach

### **Assignment Question**

- 1. What is Dynamic Approach?**
- 2. Explain concept of 0/1 knapsack**
- 3. Difference between Dynamic and Branch and Bound Approach.Which is best?**
- 4. Solve one example based on 0/1 knapsack(Other than Manual)**

### **Reference link**

- <https://www.gatevidyalay.com/o-1-knapsack-problem-using-dynamic-programming-apprach/>
- <https://www.youtube.com/watch?v=mMhC9vuA-70>
- [https://www.tutorialspoint.com/design\\_and\\_analysis\\_of\\_algorithms/design\\_and\\_analysis\\_of\\_algorithms\\_fractional\\_knapsack.htm](https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_fractional_knapsack.htm)