 Marwadi University Marwadi Chandarana Group	Marwadi University Faculty of Engineering and Technology Department of Information and Communication Technology	
Subject: DAA (01CT0512)	AIM: Implementing 0/1 Knapsack Problem using Dynamic Programming Approach	
Experiment No: 9	Date: 08/09/2025	Enrolment No: 92301733046

Implementing 0/1 Knapsack Problem using Dynamic Programming Approach

1. 0/1 Knapsack Problem Using Dynamic Programming

Theory: The **0/1 Knapsack Problem** is a classic **Dynamic Programming** problem where we are given a set of items with their respective weights and profits. The goal is to maximize profit while ensuring the total weight does not exceed the given capacity W . A DP table $dp[i][w]$ is used, where each entry represents the maximum profit achievable using the first i items with a knapsack capacity w . For each item, we decide whether to include it (if weight allows) or exclude it, and take the better of the two options.

- Subproblem Overlapping & Optimal Substructure


Programming Language: Python

Code:

```
def knapSack(W, wt, val, n):
    dp = [[0 for x in range(W+1)] for y in range(n+1)]
    for i in range(1, n+1):
        for w in range(1, W+1):
            if wt[i-1] <= w:
                dp[i][w] = max(val[i-1] + dp[i-1][w-wt[i-1]], dp[i-1][w])
            else:
                dp[i][w] = dp[i-1][w]
    return dp[n][W]
profit = [10, 5, 3, 2, 8, 7, 11]
weight = [2, 3, 1, 4, 3, 2, 7]
W = 5
n = len(profit)
print(knapSack(W, weight, profit, n))
```

Output:

```
PS D:\DAA\Lab and Lecture Codes> python -u "d:\DAA\Lab and Lecture Codes\knapsack_using_dp.py"
20
PS D:\DAA\Lab and Lecture Codes> |
```

 Marwadi University Marwadi Chandarana Group	Marwadi University Faculty of Engineering and Technology Department of Information and Communication Technology	
Subject: DAA (01CT0512)	AIM: Implementing 0/1 Knapsack Problem using Dynamic Programming Approach	
Experiment No: 9	Date: 08/09/2025	Enrolment No: 92301733046

Space complexity: $O(n.W)$

- **Justification:** A DP table of size $(n+1) \times (W+1)$ is used to store profits for different item and weight combinations.

Time complexity: $O(n.W)$

Best case time complexity: $O(n.W)$

- **Justification:** Even if items fit perfectly, the algorithm still fills the entire DP table.

Worst case time complexity: $O(n.W)$

- **Justification:** Regardless of item values and weights, the DP table of size $(n \times W)$ is filled completely.