Vidyavardhini's College of Engineering & Technology

Department of Computer Science and Engineering (Data Science)

ACADEMIC YEAR: 2024-25

Course: Analysis of Algorithm Lab

Course code: CSL401

Year/Sem: SE/IV

Experiment No.: 07
Aim: To implement 0/1 knapsack using dynamic programming approach
Name: SOHAM HEMENDRA RAUT
Roll Number: 24
Date of Performance: 13/03/2025
Date of Submission: 13/03/2025

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission.	10	
Total	**20**	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	5	3	2
Understanding	5	3	2
Journal work and timely submission.	10	8	4

Checked by

Name of Faculty : Mrs. Komal Champanerkar

Signature :

Date :

- **Aim: To implement 0/1 knapsack using dynamic programming approach**
- **Theory:**

  We are given N items where each item has some weight and profit associated with it. We are also given a bag with capacity W, [i.e., the bag can hold at most W weight in it]. The target is to put the items into the bag such that the sum of profits associated with them is the maximum possible. Initialize the solution matrix same as the input graph matrix as a first step.

  The constraint here is we can either put an item completely into the bag or cannot put it at all [It is not possible to put a part of an item into the bag].
- **Algorithm:**

Step 1: Define the problem

- Let n be the number of items.
- Let W be the maximum capacity of the knapsack.
- Each item has a weight and a value, arrange all the weights in the ascending order.

Step 2: Create a table

Define a table using the tabulation method, where V[i] [w] represents the maximum value that can be obtained by selecting from the first i items and having a knapsack capacity w.

Step 3: Finding max value

- If the max value is less than or equal to the previous value then,

  $V[i][w] = V[i-1][w]$

- If the max value is greater than the previous value, then update the value by-

  $V[i][w] = (V[i-1][w], V[i-1, w- w[i] + p[i])$

❖ **Program:**

```
def knapsack(weights, values, W, n):
    dp = [[0] * (W + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        for w in range(1, W + 1):
            # If the weight of the current item is greater than the current capacity, don't
update it
            if weights[i - 1] > w:
                dp[i][w] = dp[i - 1][w]
            else:
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1])
    return dp[n][W]

weights = [2, 3, 5, 4]   # Weights of items
values = [2, 3, 7, 5]    # Profit of items
W = 9  # Maximum capacity
n = len(weights)         # Number of items

max_value = knapsack(weights, values, W, n)
print(f"Maximum value in Knapsack = {max_value}")
```

❖ **Output:**

```
PS C:\Users\SOHAM> & C:/Users/SOHAM/anaconda3/python.exe "c:/Users/SOHAM/.conda/dynamic kanpsack.py"
Maximum value in Knapsack = 12
PS C:\Users\SOHAM>
```

❖ **Conclusion:** The 0/1 Knapsack problem has a time complexity of O(n×W) and a space complexity of O(n×W), where n is the number of items and W is the knapsack's weight capacity. The dynamic programming approach efficiently solves the problem by breaking it into smaller subproblems.