Practical No. 4

Title:

Write a Program to Solve 0-1 Knapsack Problem Using Dynamic Programming

Aim:

To implement a program that solves the 0-1 Knapsack problem using the dynamic programming approach to maximize total profit without exceeding the knapsack capacity.

Objective:

To understand and apply the dynamic programming method to solve the 0-1 Knapsack problem by building a table that stores intermediate results to avoid redundant calculations.

Hardware Requirements:

- Processor: Intel Core i3 or higher
- RAM: Minimum 2 GB
- Storage: Minimum 100 MB free space
- Input Devices: Keyboard and Mouse
- Output Device: Monitor

Software Requirements:

- Operating System: Windows / Ubantu
- Programming Language: Python 3.x (or C/C++/Java)
- IDE/Text Editor: VS Code / PyCharm / Notepad++
- Terminal or Command Prompt

Theory:

The 0-1 Knapsack problem is a classical optimization problem where:

- There are n items, each with weight w[i] and profit p[i].
- The knapsack has a maximum capacity W.
- The objective is to select items (either whole — 0 or 1, no fractions) to maximize profit without exceeding capacity.

Dynamic Programming solves this by constructing a 2D table where rows represent items and columns represent weight capacities, building solutions for smaller subproblems and combining them.

Algorithm (Dynamic Programming):

1. Create a 2D array dp[n+1][W+1] where dp[i][j] represents max profit using first i items and capacity j.
2. Initialize dp[0][j] = 0 for all j, and dp[i][0] = 0 for all i.
3. For each item i from 1 to n:
    - For each capacity j from 1 to W:
        - If weight of ith item ≤ j:
          dp[i][j] = max(dp[i-1][j], profit[i-1] + dp[i-1][j - weight[i-1]])
        - Else:
          dp[i][j] = dp[i-1][j]
4. The answer is dp[n][W].

Conclusion:

Dynamic programming is an effective approach to solve the 0-1 Knapsack problem by solving overlapping subproblems and storing intermediate results, thus reducing time complexity from exponential to polynomial.