Name: Omar Ahmed Mohamed Refaat
ID: 1210106

Repetition Knapsack with Dynamic Programming

Introduction

The Knapsack Problem is a classic optimization problem in computer science and mathematics. The **Repetition Knapsack Problem (also called the Unbounded Knapsack Problem)** allows an unlimited number of each item to be included in the knapsack, making it distinct from the 0/1 Knapsack Problem.

Problem Statement

Given a set of items, each with a weight and a value, determine the maximum value that can be achieved by selecting items to fit within a knapsack of capacity WW, where each item can be selected any number of times.

- **Input:**
  - n: Number of items
  - W: Knapsack capacity
  - weights[i]: Weight of the ith item
  - values[i]: Value of the ith item
- **Output:**
  - Maximum achievable value without exceeding the capacity W.

Main Idea of the Dynamic Programming Solution

Dynamic Programming breaks the problem into smaller subproblems and solves each subproblem only once, storing the results for reuse.

1. **Define the Result State:**
   - Let result[w] represent the maximum value achievable for a knapsack with capacity w.
2. **Base Case:**
   - result[0]=0: If the capacity is 0, the maximum value is 0.
3. **Transition Relation:**
   - For each item i:
     - If the item can fit into the current capacity w, update: result[w]=max(result[w], result[w−weights[i]]+values[i])
4. **Final Answer:**
   - The result is stored in result[W], which gives the maximum value for the full capacity of the knapsack.

Example

Input:

- Capacity W=8
- Items:
  - Item 1: Weight = 2, Value = 3
  - Item 2: Weight = 3, Value = 4
  - Item 3: Weight = 4, Value = 5

Name: Omar Ahmed Mohamed Refaat
ID: 1210106

Step-by-Step Solution:

1. Initialize result[] array:

   result=[0,0,0,0,….,0]

2. Iterate through capacities 1 to 8:
   - For capacity w=2: result[2]=max(result[2],result2[2−2]+3)=3
   - For capacity w=3: result[3]=max(result[3],result[3−3]+4)=4
   - For capacity w=4: result[4]=max(result [4], result[4−2]+3, result[4−4]+5)=6
   - Repeat for all capacities.

Final result [] Array:
result=[0,0,3,4,6,7,9,10,12]
Output: The maximum value that can be achieved is **12**.

Complexity Analysis

- **Time Complexity:** O(n×W), where n is the number of items and W is the capacity.
- **Space Complexity:** O(W), as we use a 1D array result[].

Conclusion
The Repetition Knapsack Problem is efficiently solvable using dynamic programming. This method ensures that the solution is optimal and avoids redundant calculations through memoization. The example demonstrates how the approach works step-by-step to achieve the maximum value for a given capacity.
