Pseudocode Outline: Topic 6

By Joseph Abraham

Knapsack Problem — Greedy and Dynamic Programming Techniques

Purpose:

Solve the 0/1 Knapsack problem using both the **greedy algorithm** and **dynamic programming**.

Track and report the number of **steps** (comparisons/decisions) to compare the **complexity and effectiveness** of both techniques.

Handle the base case and the extended version with multiple copies of each item.

BEGIN

Initialize Inputs:

- Knapsack capacity = 280
- Item weights = {20, 30, 40, 60, 70, 90}
- Item values = {70, 80, 90, 110, 120, 200}
- Item quantities = {1, 2, 1, 3, 1, 2}

Greedy Algorithm (Single and Multi-item Cases):

FOR each item:

- Compute value-to-weight ratio

Sort items in descending order of ratio

Track greedySteps during sorting and selection

Initialize:

- totalWeight = 0

```
    totalValue = 0

FOR each item in sorted list:
    IF item fits in remaining capacity:
        Add to total weight and value
        Increment greedySteps

Output total greedy value and step count
```

Dynamic Programming Algorithm (Single and Multi-item Cases):

```
Initialize DP table of size (n+1) × (capacity+1)

Track dpSteps during filling of table

FOR i from 1 to n:
    FOR w from 0 to capacity:
        IF item fits:
            dp[i][w] = max(dp[i-1][w], value[i-1] + dp[i-1][w - weight[i-1]])
        ELSE:
            dp[i][w] = dp[i-1][w]
        Increment dpSteps

Output value at dp[n][capacity] and step count
```

Handle Multiple Copies of Items:

```
FOR each item:
    Replicate weight and value based on quantity
    Expand arrays

Pass expanded arrays into greedy and dynamic methods
```

END