Assignment on the 0/1 Knapsack Problem:

Introduction

Once upon a time in a faraway land, there existed a wealthy merchant named Alaric who owned a grand knapsack. This knapsack had a special property: it could hold items of a certain weight but nothing more. Alaric, with his wealth and wisdom, wanted to find the optimal way to fill his knapsack with precious items, maximizing their value while not exceeding the weight limit.

The Problem

The task at hand is to help Alaric find the best way to fill his knapsack. Given a set of items, each with a weight and value, Alaric must decide which items to include in his knapsack so that the total weight does not exceed the knapsack's capacity, and the total value is maximized.

Approach 1: Recursive Solution

In the ancient town, a wise old sage named Sage Recursive proposed an initial solution. He said:

1. **Base Case**: If there are no items left or the knapsack's capacity is zero, then the maximum value will be zero.
2. **Recursive Case**: For each item, there are two options:
    - **Include the item**: If the item's weight is less than or equal to the knapsack's capacity, take the value of the item plus the maximum value that can be obtained from the remaining items with the reduced capacity.
    - **Exclude the item**: Consider only the remaining items without including the current one.

Hint: Use a recursive function to explore both options and return the maximum value.

Approach 2: Memoization Solution

The sage's apprentice, Memo, saw that Sage Recursive's solution involved a lot of repeated calculations. He proposed to keep track of the solutions to sub problems, storing them in a table to avoid recomputation. Thus, Memoization was born.

Hint: Use a memoization table to store intermediate results and avoid recalculating the same subproblems.

Approach 3: Dynamic Programming Solution

Finally, Alaric's other apprentice, Dyno, suggested a more efficient way. He proposed a bottom-up approach called Dynamic Programming, where all the subproblems would be solved first and their results used to build up the solution to the overall problem.

Hint: Create a 2D array to store the maximum value for different weights and items, and fill it using a nested loop.

Sample Input for 0/1 Knapsack

Values: [60, 100, 120]
Weights: [10, 20, 30]
Knapsack Capacity: 55

Sample Output for 0/1 Knapsack

Maximum value in Knapsack = 220

Extending the Story to Fractional Knapsack

Alaric, now an experienced merchant with a perfectly filled knapsack, faced a new challenge. On one of his trade journeys, he encountered a mystical merchant named Frax, who introduced him to the concept of fractional items. Frax explained that in this new land, Alaric could take fractions of items, not just whole ones, allowing him to make even better use of his knapsack's capacity.

Approach: Fractional Knapsack

Frax shared his wisdom with Alaric:

1. **Greedy Approach**: To maximize the value of the knapsack, first, calculate the value-to-weight ratio for each item.
2. **Sort Items**: Arrange items in descending order of their value-to-weight ratio.
3. **Pick Items**: Start by picking items with the highest value-to-weight ratio, and add as much as possible of each item until the knapsack is full.

Hint: Use a greedy algorithm to take the most valuable fractions of items first.

Sample Input for Fractional Knapsack

Values: [60, 100, 120]
Weights: [10, 20, 30]
Knapsack Capacity: 55

Sample Output for Fractional Knapsack

Maximum value in Knapsack = 260

Conclusion

With the help of Frax, Alaric was able to maximize the value of his knapsack even further by incorporating fractional items. He realized that while the 0/1 knapsack problem required careful consideration of whole items, the fractional knapsack allowed for even greater efficiency through the use of a greedy approach.

Assignment Tasks:

1. Implement the recursive, memoization, and dynamic programming solutions for the 0/1 knapsack problem.
2. Implement the greedy solution for the fractional knapsack problem.
3. Compare the time complexity and space complexity of these different approaches.
4. Provide examples with different sets of items and capacities to demonstrate how each approach works.