DYANAMIC INVENTORY MANAGEMENT

PROJECT MEMBERS:
Vickkash J
Sudharsan V
Shanmuga Patel Kani C

OBJECTIVE

This project aims to provide an efficient solution to inventory management and resource allocation using the Knapsack Problem. By dynamically selecting the most valuable set of items for packing or shipping while adhering to weight constraints, the system helps maximize efficiency in real-world applications such as logistics and warehouse management.

PROBLEM STATEMENT

Given a list of items, each with a specific weight and value, and a shipping container (knapsack) with a limited weight capacity, the task is to find the most valuable subset of items that can fit into the container. The solution should maximize the total value of the selected items while ensuring that the total weight does not exceed the available capacity of the container.

ALGORITHM

Input:

A list of items, each with a value and weight.

The knapsack capacity.

Initialize DP Table:

Create a 2D array dp of size (n+1) x (capacity+1), where dp[i][w] stores the maximum value achievable using the first i items and weight w.

Fill DP Table:

```
For each item i (from 1 to n):
    For each weight w (from 1 to capacity):
        If item i's weight ≤ w, update:
            dp[i][w] = max(dp[i-1][w], dp[i-1][w - item[i-1].weight] + item[i-1].value)
        Else: dp[i][w] = dp[i-1][w].
```

Backtrack to find selected items:

Start from dp[n][capacity] and trace back to find the items that contributed to the optimal value.

Return:

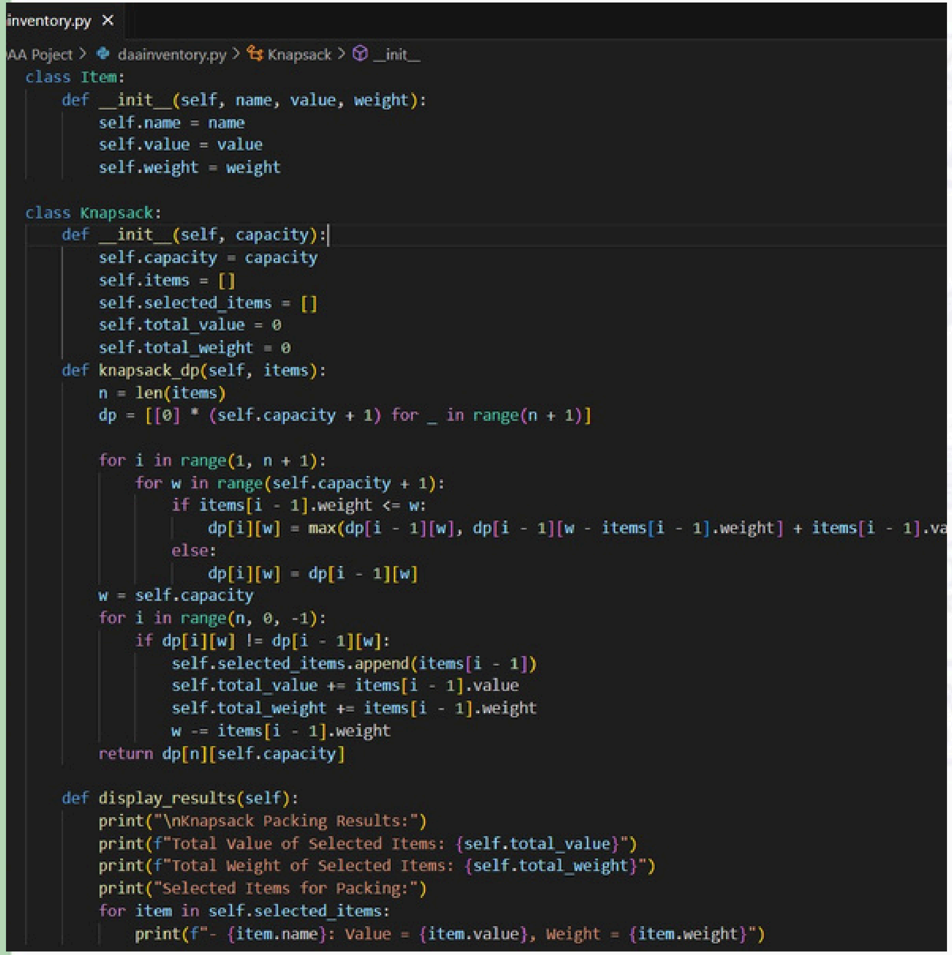
Maximum value and the list of selected items.

This algorithm uses dynamic programming to find the optimal subset of items that maximizes value without exceeding the weight capacity.

PSEUDOCODE

```
KnapsackProblem(items, capacity):
    n = length of items
    for i from 1 to n:
        for w from 1 to capacity:
            if items[i-1].weight <= w:
                dp[i][w] = max(dp[i-1][w], dp[i-1][w - items[i-1].weight] + items[i-1].value)
            else:
                dp[i][w] = dp[i-1][w]
    max_value = dp[n][capacity]
    selected_items = []
    w = capacity
    for i from n down to 1:
        if dp[i][w] != dp[i-1][w]:
            selected_items.append(items[i-1])
            w -= items[i-1].weight
    return max_value, selected_items
```

CODING



inventory.py ✕

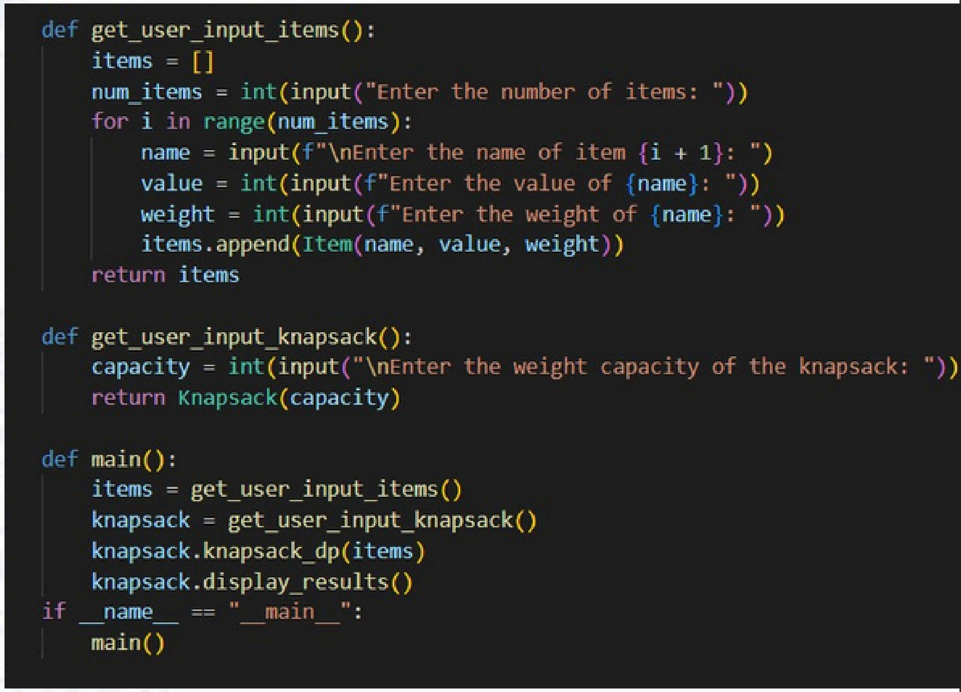
AA Poject > daainventory.py > Knapsack > __init__

```python
class Item:
    def __init__(self, name, value, weight):
        self.name = name
        self.value = value
        self.weight = weight

class Knapsack:
    def __init__(self, capacity):
        self.capacity = capacity
        self.items = []
        self.selected_items = []
        self.total_value = 0
        self.total_weight = 0
    def knapsack_dp(self, items):
        n = len(items)
        dp = [[0] * (self.capacity + 1) for _ in range(n + 1)]

        for i in range(1, n + 1):
            for w in range(self.capacity + 1):
                if items[i - 1].weight <= w:
                    dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - items[i - 1].weight] + items[i - 1].va
                else:
                    dp[i][w] = dp[i - 1][w]
        w = self.capacity
        for i in range(n, 0, -1):
            if dp[i][w] != dp[i - 1][w]:
                self.selected_items.append(items[i - 1])
                self.total_value += items[i - 1].value
                self.total_weight += items[i - 1].weight
                w -= items[i - 1].weight
        return dp[n][self.capacity]

    def display_results(self):
        print("\nKnapsack Packing Results:")
        print(f"Total Value of Selected Items: {self.total_value}")
        print(f"Total Weight of Selected Items: {self.total_weight}")
        print("Selected Items for Packing:")
        for item in self.selected_items:
            print(f"- {item.name}: Value = {item.value}, Weight = {item.weight}")
```



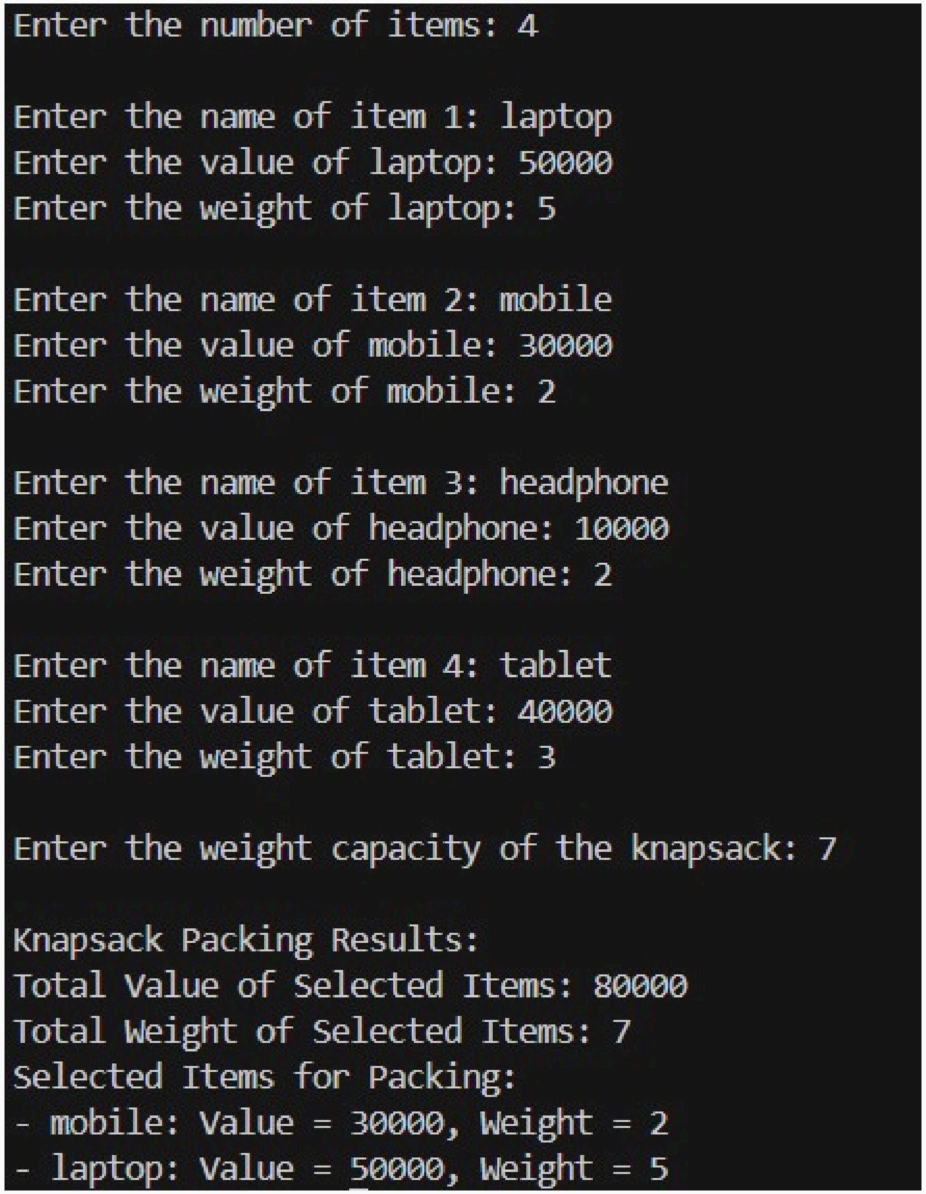
```python
def get_user_input_items():
    items = []
    num_items = int(input("Enter the number of items: "))
    for i in range(num_items):
        name = input(f"\nEnter the name of item {i + 1}: ")
        value = int(input(f"Enter the value of {name}: "))
        weight = int(input(f"Enter the weight of {name}: "))
        items.append(Item(name, value, weight))
    return items

def get_user_input_knapsack():
    capacity = int(input("\nEnter the weight capacity of the knapsack: "))
    return Knapsack(capacity)

def main():
    items = get_user_input_items()
    knapsack = get_user_input_knapsack()
    knapsack.knapsack_dp(items)
    knapsack.display_results()
if __name__ == "__main__":
    main()
```

KONGU ENGINEERING COLLEGE

OUTPUT



```
Enter the number of items: 4

Enter the name of item 1: laptop
Enter the value of laptop: 50000
Enter the weight of laptop: 5

Enter the name of item 2: mobile
Enter the value of mobile: 30000
Enter the weight of mobile: 2

Enter the name of item 3: headphone
Enter the value of headphone: 10000
Enter the weight of headphone: 2

Enter the name of item 4: tablet
Enter the value of tablet: 40000
Enter the weight of tablet: 3

Enter the weight capacity of the knapsack: 7

Knapsack Packing Results:
Total Value of Selected Items: 80000
Total Weight of Selected Items: 7
Selected Items for Packing:
- mobile: Value = 30000, Weight = 2
- laptop: Value = 50000, Weight = 5
```

CONCLUSION

The Dynamic Inventory Management System efficiently solves the Knapsack Problem by selecting the highest-value items within weight constraints, optimizing resource use in logistics and inventory management. Using dynamic programming, it offers scalable, practical solutions for real-world applications, with potential for future enhancements like additional constraints and real-time updates. This project demonstrates the impactful role of algorithms in improving decision-making and operational efficiency.

THANK YOU