IT251 – DATA STRUCTURES & ALGORITHMS II

ASSIGNMENT 3

Name: **Sachin Prasanna**
Roll No.: **211IT058**

Question Statements:

1. Solve the knapsack problem (optimal solution derivation) using dynamic programming techniques (5 Marks) • Provide an implementation technique using tabulation method (4 Marks) • Provide a brief write-up on possible alternate methods of implementation (1 Mark)

2. Implement the knapsack problem using Greedy method technique (5 Marks)
• Show the optimization/maximization by supporting fractional support (implement as a user input option) (5 Marks)

Answers:

1)

Output (For the 6 different methods implemented, including tabulation):

```
ANSWERS FOR SAMPLE TEST CASES GIVEN:

Maximum value in the Knapsack through BRUTE FORCE RECURSION USING EXTRA VECTOR: 220
Maximum value in the Knapsack through SIMPLE RECURSION: 220
Maximum value in the Knapsack through MEMOIZATION: 220
Maximum value in the Knapsack through TABULATION: 220
Maximum value in the Knapsack through SPACE OPTIMIZED TABULATION (2 ARRAYS): 220
Maximum value in the Knapsack through SPACE OPTIMIZED BEST TABULATION (1 ARRAY): 220


ENTER VALUES FOR A USER INPUTTED SOLUTION!

Enter number of Objects: 7
Enter weight (Capacity) of Knapsack: 15
Enter Value and Weight of each object:
 5 1
10 3
15 5
7 4
8 1
9 3
4 2


ANSWERS FOR USER INPUTTED TEST CASE:

Maximum value in the Knapsack through BRUTE FORCE RECURSION USING EXTRA VECTOR: 51
Maximum value in the Knapsack through SIMPLE RECURSION: 51
Maximum value in the Knapsack through MEMOIZATION: 51
Maximum value in the Knapsack through TABULATION: 51
Maximum value in the Knapsack through SPACE OPTIMIZED TABULATION (2 ARRAYS): 51
Maximum value in the Knapsack through SPACE OPTIMIZED BEST TABULATION (1 ARRAY): 51


...Program finished with exit code 0
Press ENTER to exit console.
```

Please refer to the other document for the brief write up on ways of approaching a knapsack problem.

2)

Output (For different ways of Greedy algorithm by supporting fractional support):

```
input
Enter number of Objects: 7
Enter weight (Capacity) of Knapsack: 15
Enter Profit and Weight of Each object:
 5 1
10 3
15 5
7 4
8 1
9 3
4 2

***KNAPSACK PROBLEM BY GREEDY ALGORITHMS BY SUPPORTING FRACTIONAL SUPPORT***

1.Selecting items based on which item yields the highest profit initially -> 47.25
2.Selecting items based on which item yields the lowest weight initially -> 46
3.Selecting items based on which item yields the highest profit/weight ratio initially -> 51



...Program finished with exit code 0
Press ENTER to exit console.
```

Explanation of the methods:

The sortByFirst function sorts the profitandWeight vector in descending order of profits, breaking ties by sorting in ascending order of weights. The sortBySecond function sorts the profitandWeight vector in ascending order of weights, breaking ties by sorting in descending order of profits.

The **maxProfit method** calculates the maximum profit that can be obtained by selecting objects based on the highest profit initially. It first sorts the profitandWeight vector using the sortByFirst function, and then iterates through the sorted vector, selecting objects until the Knapsack's weight capacity is reached. If an object's weight is greater than the remaining capacity, the function selects a fraction of the object based on the remaining capacity and adds it to the total profit.

The **minWeight method** calculates the minimum weight that can be obtained by selecting objects based on the lowest weight initially. It first sorts the profitandWeight vector using the sortBySecond function, and then iterates through the sorted vector, selecting objects until the Knapsack's weight capacity is reached. If an object's weight is greater than the remaining capacity, the function selects a fraction of the object based on the remaining capacity and adds it to the total weight.

The **maxProfitWeight method** calculates the maximum profit that can be obtained by selecting objects based on the highest profit/weight ratio initially. It first sorts the profitWeightRatio vector in descending order of profit/weight ratio using the cmp function, and then iterates through the sorted vector, selecting objects until the Knapsack's weight capacity is reached. If an object's weight is greater than the remaining capacity, the function selects a fraction of the object based on the remaining capacity and adds it to the total profit.

Please refer to the code files within this folder of submission for the codes of both the questions.
OnlineGDB is preferred to run the codes if VSCode has any issues.

****