MARMARA UNIVERSITY

FACULTY OF ENGINEERING DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

CSE 2046 – 0-1 MULTI-CONSTRAINT KNAPSACK PROBLEM

STUDENTS	NUMBERS
Sueda BİLEN	150117044
Özge SALTAN	150517059
Zehra KURU	150119841

Submitted To:	**Due Date:**
Asst. Prof. Ömer Korçak	16/06/2021

0-1 Multi-constraint Knapsack Problem

Multi-constraint Knapsack problem which has multiple constraints, such as *m* knapsacks and *n* items. When an item is selected, it puts the item into every knapsack but items' weights may be different for different knapsacks. Also knapsacks' capacities may differ.

General Variables:

Common used variables that we used for implementing the algorithm:

- *int knapsackNum*, to hold amount of knapsacks.
- *int itemNum*, to hold amount of items.
- *int counter*, it statically manages Knapsack method every iteration.
- *int totalValue*, holds calculated total value.
- *boolean whileBreaker*, to detect the end of the program.

- *arrayList<Integer> values*, holds values of knapsack items.
- *arrayList<Integer> capacityList*, holds capacities of knapsacks as a list.
- *arrayList<Integer> finalCommons*, holds only common elements.
- *arrayList<Integer> commonElements*, holds all common elements after Knapsack method executed.
- *arrayList<Integer> weights*, holds all knapsacks values in it.

Algorithm Explanation:

After our research and discussions, we decided to specify general 0/1 knapsack method to solve the multi constraint knapsack problem.

At first, we thought of filling all the knapsacks one by one and taking the common items for all knapsacks. Then, if any knapsack is not filled yet, their capacities are not 0 or less than remaining items' values, we take the common items for a few knapsacks and we fill the knapsacks with those items. We thought of continuing this algorithm until one of knapsacks' capacities is full or cannot be filled anymore because the remaining capacity is less than remaining items' values.

Step-by-Step Code Stream:

1. Taking values to arraylists and variables from the provided input file.
2. *While loop* executes if *capacityList* does not contain 0 value in it and *whileBreaker* is still false. That means there are still some common elements to add *totalValue* probably.
3. It sets *counter* to 1 for controlling knapsack method's arraylist assignments.
4. *For loop* executes for all knapsacks with *Knapsack() method[1]* to find common elements between them and calculates possible max value according to every knapsack. It converts values and weights arrayLists to an array and sends them to the *Knapsack() method*.
5. After the *for loop* is completed, we call the *countFrequencies method* with *commonElements* arrayList.

6. *countFrequencies method* has two *for loops* to add *commonElements* values to *finalCommons* arrayList. In first *for loop*, if the item is common for all knapsacks, and it does not exceed the capacity of the knapsack where it belongs we set capacity of knapsack with *capacityList*.get(k) - *weights*.get(k).get(val.getKey()) added that item to *finalCommons* arrayList. If we successfully add the item to *finalCommons* arrayList, *newElementAdded* boolean variable will turn to true.

7. Second *for loop* will execute and add element to *finalCommons* arrayList if no item is added to *finalCommons* arrayList in that iteration (if *newElementAdded* == false). In this loop we will add less common elements from *commonElements* arrayList with descending order to get maximum *totalValue*. If any item is added to *finalCommons* arrayList, *lessCommonAdded* boolean value will be true.

8. After the *countFrequencies* method, we remove items from weights that we added into *finalCommons*. If *whileBreaker* value turns to true, then *while loop* will stop.

9. After the *while loop* completes the execution, *totalValue* is calculated with getting values from values arrayList according to positions stored in *finalCommons* arrayList.

10. Finally, we printed *totalValue* and 0-1's to the output file.

[1]: We just add finding *commonElements* part to the Knapsack method.

Table 1. Outputs Table

FILE NAME	***KNAPSACK NUMBER***	***ITEM NUMBER***	***SOLUTION***
sample1.txt	2	28	138477
sample2.txt	5	40	5592
sample3.txt	30	60	7259
test1.txt	2	105	1090481
test2.txt	5	90	11165
test3.txt	30	60	8455
test4.txt	30	40	346