CCPS616 Lab 5 – Dynamic 0/1 Knapsack

Preamble

In this lab you will write a C++ program that finds an optimal solution to the 0/1 knapsack problem discussed in class. Your implementation must be from the ground up without linking any external libraries. Your lab should be completed in a single cpp file called **lab4.cpp**, and should compile at the command line very simply as follows:

```
g++ -o lab5 lab5.cpp
```

Please do not submit labs that don't even compile. We're past that. I'm not going to fix your syntax errors. If you can't make any progress on the lab whatsoever, submit a simple program that prints a message saying as much. For this, I will award you a single mark.

Lab Description

In this week's lab you will implement two algorithms for solving the 0/1 knapsack problem. The first algorithm will be a simple $O(2^n)$ brute force search. For the small data sets you'll be testing on, this won't be too costly. You can use the optimal result yielded from the brute force search to test your second solution, which will use dynamic programming. Refer to the example covered in class to see how this can be approached.

Input to your program will come in the form of a plain text file. The format of this file is identical to lab 4, so you should be able to re-use your file I/O code directly. The input file will have three lines. The first will indicate the maximum weight you can carry, the second will be a list of item weights, and the third will be a list of item values. A very simple example can be seen below. The max weight is 5, the first item has a weight of 2 and a value of 1, the second has a weight of 3 and a value of 3, the third has a weight of 4 and a value of 5.

```
5
2 3 4
1 3 5
```

Your task, of course, is to choose those items that maximize the value in your knapsack without going over the weight limit. In this case, the optimal choice is taking the third item alone.

Testing

Test your program on several different inputs. The brute force approach is easy to implement and can be used to validate the optimality of your dynamic programming approach. It might be handy to simply execute both implementations and do a comparison between results directly in your program, rather than comparing output files manually.

Results

Your results should be produced in the form of an output file. Consider the input below:

```
5
2 2 4
2 4 5
```

You output file should look as follow:

```
Items:  (2, 2), (2, 4)
Value:  6
Weight: 4
```

The “Items” line consists of a list of (weight, value) pairs of the items taken. The “Value” line is the total value of the knapsack, and the “Weight” line is the total weight (must be <= capacity, of course).

Submission

Lab 5 may be done in **pairs**! Submit your source file (**lab5.cpp**) to D2L. If you worked in pairs, Indicate the names of both partners in a comment at the top of your source code.

Marking Rubric: This lab is out of 5 marks

1 mark – Brute force implementation is correct
3 marks – Dynamic programming implementation is correct.
1 mark – Results produced correctly, file output format is correct.