CSC 201 Data Structures

Professor Burg

Fall 2020

Programming Assignment 4

Using Backtracking in the Knapsack Problem

Due Tuesday, November 24 at 11:59 PM

100 points

This is the knapsack problem:

- You have a knapsack that you want to fill with "things."
- The knapsack has a weight capacity c
- There are t things you can possibly put into the knapsack, numbered from 0 to t-1.
- For 0 <= i <= t-1, each thing has a weight of w_i.
- For 0 <= i <= t-1, each thing has a value of v_i.
- Your goal is to pack the knapsack so that you don't exceed its capacity. You will write two versions of a "fill" method.
    - **Greedy version.** At each step (going from index 0 to n-1), choose to put in the knapsack the thing with the largest value and a weight that will not exceed the knapsack's remaining capacity. As soon as you can't take the thing at the next index, stop. You do not backtrack in the greedy algorithm. This version does not have to be recursive. Call this method *greedyFill*.
    - **Optimal version.** Find the "packing" that gets the greatest total value possible in the knapsack without exceeding the weight capacity. This version should use a recursive backtracking algorithm. Call this version *optimalFill*.

Your program should read in the data for the problem from an external file called *knapsack.txt*. The data will be in this format:

- an integer representing the capacity of the knapsack
- the number of things available to pack
- a list of the weights of each of these things (integers)
- a list of the values of each of these things (integers)

in *main*

- Print out the data you read in from *knapsack.txt.* (You wouldn't want to do this if you had a very long list of things, but our input won't be that long. Don't print if number of things > 15)
- Call each of your fill methods from *main.*

In each of the two fill methods, print the following:

- The name of the method giving the output that follows.

- The final solution -- i.e., the list of things that you packed, with their weights and values, and the total weight and value. (Don't print the list of things you paced if number of things > 15.)

Advice on programming:

- Do NOT look for a solution online. We are going to use the code-checker on your programs.
- Think of your recursive method as executing in the form of a tree. Draw a picture for yourself to visualize where and when backtracking occurs.
- Do your own class design. It does not have to be complicated, but think in an object-oriented way that models your real-world problem. Be sure the data members and methods of a class make sense for that class.
- Use the debugger as you develop your program. If you come to me for help, I'm going to ask you to trace your program using the debugger.
- We will give you an input file to practice on, but we will check your program with other input. It is a good idea for you to make other input files on your own to check your program correctness. It's ok to share input files with other students to see if you get the same answers, but do not share code.

How to submit:

Zip up your entire IntelliJ folder for the project and submit it on Canvas.