Assignment 8
Due: Dec 3 at 11:59PM

1 Assignment 8 Specification

1.1 Implementation

The skeleton code at `https://classroom.github.com/a/KmQKV3HS` is the start of a program to solve the 0-1 knapsack problem (p425 in CLRS, and lecture notes): Given n objects, each with integer weight w_i and price p_i, and a knapsack that can hold up to weight W, select the most valuable subset of objects that don't exceed the knapsack's capacity. All weights and prices will be positive integers. Implement a dynamic programming solution to the problem.

For this assignment, you will implement the **FindDynamic** method (and any helper methods you want) in **A8.java**. In addition to returning an array with the included items, you should also update the class variable `best_price` with the optimal price in a solution. Since the order of the items doesn't matter, you can return them in any order.

There are two algorithms already implemented:

findEnumerate Tests every possible subset for the weight limit and price of the knapsack.

findGreedy Greedily picks the best remaining item based on its price:weight ratio, which is sometimes suboptimal.

The item files are supplied in the **objects** directory. They are text files with a $\langle weight,\ price \rangle$ pair to describe an item on each line.

1.2 Testing

For this assignment, JUnit tests are provided in the **edu.wit.cs.comp2350.tests** package. The tests check if the knapsack has the optimal price and satisfies the weight limit. For grading this assignment, I will use some different item lists in addition to the ones supplied. Write some of your own tests to test corner cases and make sure your code is robust.

A **ChartMaker** class is included, which will create a chart of runtimes comparing dynamic, enumerative, and greedy algorithms. You can use this chart to verify that your dynamic programming solution runtime is what you expect.

2 Grading

`best_price`: 70%

Items array: 30%