Ali Asghar Momeni Vesalian
momeni.vesalian@gmail.com
Mobiquity code challenge
February, 2020

Package Challenge

Strategy

The main strategy is to find an accurate solution in minimum time span and memory usage for 0/1 knapsack problem with decimal weights. The traditional dynamic programming solution has been customized here and combined with a recursive method to remove unnecessary calculations.

Algorithm

The decimal weights made the problem little different from the traditional 0/1 knapsack problem. Due to this, the traditional approach of dynamic programming algorithm has been customized to produce the desired result. For the aforementioned traditional approach by removing the existing fractions we have to multiply them with the appropriate power of ten. This approach is not quite efficient because many unnecessary calculations make the process lengthy. For this reason, I combined the solution with a recursive method for removing these unnecessary calculations. This recursive method also handles the problem of fractions. Also for this recursion, we do not need to store the results of the sub- problems. Hence the solution is more rigorous.

Now for finding the result for the *problem(n, w)*, I calculated the maximum result of *problem(n-1, w)* and *problem(n-1, w - w(n))*. Therefore the maximum is obtained from getting the result with the maximum cost and then minimum weight. Hence the time and space complexity of the solution is O(nw).

Data Structure

The ***java.util.List*** is used in the solution to store the items of knapsack.

Concerns

The main concerns in designing and implementing the project are SOLID principles, separation of concerns, simplicity, maintainability, loosely coupling, and coherency. The TDD approach is also employed during the implementation of the project.

Design Pattern

Strategy pattern is used for having multiple implementations of KnapsackProblemReader and KnapsackSolver. Henceforth we can add our reader which reads the stream of data from a different data source. We can also add our solver which solves the problem with a different algorithm without altering the semantics of the project.

Constraints

- Given that we might need to choose up to 15 items, the recursive algorithm won't get StackOverflowError
- According to the algorithm, we won't have any limitations in maximum wight for packages or items.