Tyler Technologies Challenge WriteUp

Tyler LaBerge

First Steps

The first step I took to complete this challenge was to define the domain of the problem. While reading the description of the problem I wrote down any keywords that sounded like they could be modeled in a useful way in my program. Initially I determined the domain to contain Shopper, Cart, FoodItem and Inventory objects, with the idea that each of these would become a class in my program. Very quickly though I decided not to include an Inventory object and instead keep track of each FoodItem's stock within each FoodItem object. The reason I chose to do this is because I didn't want to have to add each individual food item to the cart one at a time, such as adding one million milk items. Instead it would be much faster and simpler to just add one milk item to the cart and tell the cart that the quantity of milk items I'm buying is one million.

Outside of the domain I realized early on that this problem could take advantage of the Strategy design pattern. Each task being it's own Strategy. I initially did this by creating a Task interface but pretty soon I realized I was solving many of the Tasks by following similar steps which could be abstracted into a base class. With that in mind I switched from the Strategy pattern to the very similar Template pattern, changing the Task interface into an abstract class with template methods. This turned out to work really well and combined with my domain objects the code for solving each of the tasks remained quite clean.

The Algorithms

Determining the algorithms to use for each task started with a bit of research. I discovered what is known as the Knapsack problem which is fairly similar to this problem, however with the multiple constraints of this problem I had a difficult time translating that algorithm to work for this particular problem, and so I decided to just do my own algorithm for each of the tasks.

Originally my algorithms for tasks 1, 2, and 4 involved assigning weights to the budget, weight limit, and volume limit constraints depending on which applied to each task, then assigning a value to each food item based on those weights, sorting the food items by smallest value, and going through each food item one at a time and purchasing as many of them as possible. While this worked for many inputs, I found that there were still some inputs that it failed on. I realized the main problem was in the way I was assigning weights, but even after fiddling with the weights in many different ways I couldn't get it to work for all inputs. I decided to scrap that algorithm and instead just have the algorithm simply sort each food item based on how much of each I can buy while remaining within the constraints, and without taking their stock into account, thus sorting by the most optimal items. I would then go through each item in the sorted list and buy as many as I could. This ended up being my final solution for task 1, 2, and 4 and works for all the inputs I tested

against. The order complexity for this algorithm came out to be O(nlogn), with n being each unique type of food item(i.e 30 milk food items = 1N, 30 milk and 20 bread food items = 2N, etc.).

Task 3 was by far the most difficult in my opinion, and required a different algorithm than the others. I spent a fair amount of time brainstorming what the algorithm could be for this task and eventually came up with one that seems to work well. Essentially the algorithm I came up with fills the cart in the worst way possible to begin with, making the worse items be the most distributed items, then the algorithm starts chipping away at the amount of each of these bad items in the cart and replaces them with better items. The reason we chip away from worse items is because it is guaranteed that a better item can be added to the cart after a worse item is removed (a better item being defined as one in which I can purchase more of). At this point it is likely that the most distributed items are actually better items because by definition you can purchase more of a better item than a worse item, and so if you remove one bad item then it is likely you can add at least two good items. Now that the good items are the most distributed you start chipping away at those instead. Basically this will eventually balance out to have the least amount of bad items while still maintaining the most amount of items that can purchased. I believe the order complexity for this algorithm in the worst case is O(n^k) where k is the stock of the most in stock item. In most cases k is much greater than n however in the average case the order complexity is probably more like O(n^2). The order complexity varies quite a bit between inputs and so it is tricky to determine the precise order complexity. While this may not be a great algorithm based on its order complexity it seems to work well enough. I'm not sure how it would do on very large input files but for reasonably sized inputs it works well.

Difficulties

The main difficulties I had during this challenge were determining the algorithm for task 3, and also writing thorough tests. I wrote over 100 tests for this application and I'm still not sure if they are thorough enough. Writing tests with complex inputs required me to manually go through and find the optimal output myself, which could take a decent amount of time. Also, coming up with tests for the edge cases was quite difficult because you need to get the math lined up correctly which also takes a while. If I had more time I would have written more tests, especially for task 3 which I don't feel I tested enough considering its complexity.

Conclusion

Overall this was an enjoyable challenge. I had fun brainstorming the algorithms and it was nice finding a place to take advantage of the Template/Strategy pattern to write clean code. I learned that writing tests can sometimes be just as hard and take just as long to write as the actual application itself. Simply by having large input files it can take a very long time to manually solve each task and write a test for it. Still, I had fun and I appreciate the opportunity to compete in this challenge.