 Assignment 0423

For our second-to-last paradigm, we revisit the old change-making problem with a new approach, known as dynamic programming. Of note: the “programming” in “dynamic programming” does not refer to the code that you write, but to the optimal solution (i.e., the “plan” or “program”) that is sought by the problem.

Outcomes

This assignment will affect your proficiency measures for outcomes 1a–1c, 2a–2c, and 3a–3f.

For Submission

Finish the Java program MakeOptimalChange by implementing its core method:

```java
public static Tally makeOptimalChange(
    int[] denominations, amount)
```

The returned Tally object should consist of the optimal way for making the given amount using the given denominations. When no such way exists, the returned value should be the special constant Tally.IMPOSSIBLE (also already defined for you).

To demonstrate the correctness of your implementation, add test cases to the supplied test harness. That test harness includes only the trivial example of standard change making using USA currency.

The rest of the program has been written for you, both to decrease the time required to finish the assignment and to provide a demonstration of “how the teacher would have done it.” As with previously supplied code, feel free to study what’s in there.

Invoke MakeOptimalChange like this:
```
java MakeOptimalChange denominations amount
```

The denominations argument is a comma-separated list of integers without spaces between them; amount is the integer amount for which to make change. Sample runs are included below (to save space, the usage message is included only if it is the only output shown by the program):

$ java MakeOptimalChange 9,10 -4
Change cannot be made for negative amounts.

$ java MakeOptimalChange 0,5,9 32
Denominations must all be greater than zero.

$ java MakeOptimalChange 0, 5, 9 32
Usage: java MakeOptimalChange <denominations> <amount>
- <denominations> is a comma-separated list of denominations (no spaces)
- <amount> is the amount for which to make change

$ java MakeOptimalChange 10,1,10,14 28
Duplicate denominations are not allowed.

$ java MakeOptimalChange 10,1,14 28
28 cents can be made with 2 coins as follows:
- 0 10-cent coins
- 0 1-cent coins
- 2 14-cent coins

$ java MakeOptimalChange 2,16,8 5
It is impossible to make 5 cents with those denominations.

$ java MakeOptimalChange 4,1,9 12
12 cents can be made with 3 coins as follows:
- 3 4-cent coins
- 0 1-cent coins
- 0 9-cent coins

$ java MakeOptimalChange 25,10,5,1 99
99 cents can be made with 9 coins as follows:
- 3 25-cent coins
- 2 10-cent coins
- 0 5-cent coins
- 4 1-cent coins

Implementation Notes

- It’s all about the mantra given in class, given in class, given in class (no shortcuts). Make sure you have the mantra down cold.
- The denominations do not have to be sorted (and your code should not sort them, either).
- A one-cent denomination is not required; thus, the “no answer” case is certainly possible.
- There may be more than one optimal solution (i.e., a tie); in this case, the program may display any optimal solution.

How to Turn It In

Submit your code to your GitHub repository under the folder optimal-change. As always, don’t forget to commit as you go.