The Homework

Make a class that represents all the moves and allows one to query for the shortest path between two states.

The `CoinPuzzle` class.

- `__init__(self, pennies, dimes)` - takes in the number of pennies and dimes for the puzzle. There will always be one space.
- `solve(a,b)` - Find the shortest number of moves to get from configuration `a` to configuration `b`. You will need to search through your graph to find the path. It will return a list of configurations, starting from `a` and ending with `b`, that represents the sequence of configurations seen in the shortest sequence of moves from `a` to `b`.

Tips for generating all the configurations

One way to generate the set of configurations is to use either recursion or dynamic programming. Recall that the key to finding an algorithm with recursion or DP is to look for smaller instances of the same problem. So, for example, the configurations with two pennies, two dimes, and one space can be found by combining three sets:

- Add a penny to each configuration with one penny, two dimes, and one space.
- Add a dime to each configuration with two pennies, one dime, and one space.
- Add a space to each configuration with two pennies, two dimes, and no spaces.

You will want to make sets of such configurations and avoid enumerating the configurations multiple times for the same number of pennies, dimes, and spaces.

Use a generic graph class

This is a great opportunity to make a graph class that's independent of this particular puzzle problem. Just like we use a generic list inside our Stack class, this is a nice opportunity to use a graph data structure to represent the puzzle structure. The configurations are vertices and the moves are edges.

The Speed Challenge

A standard graph search as in the book will maybe be too slow. Consider augmenting it to stop when it reaches the desired vertex.

Summary

You will write code that will output the shortest path between any two configurations of a set amount of pennies and dimes. To do this, you will need to create a graph representing all possible configurations. The neighbors of a given configuration are configurations that can be reached by making any valid move. It is your choice on how to represent this graph.

Make sure to name your class `CoinPuzzle` and save it in a file called `coinpuzzle.py`. You will have to submit your graph data structure as well as your `configuration.py` file from the lab. The tests will use your lab solution, so make sure your Configuration class works correctly. Othewise, it can hide bugs in your `CoinPuzzle` class. Also submit any other files that are necessary to create a `CoinPuzzle` instance.