Coin Change Problem

Note: This problem has multiple solutions and is a classic problem in showing issues with basic recursion. There are better solutions involving memoization and simple iterative solutions.If you are having trouble with this problem (or it seems to be taking a long time to run in some cases) check out the Solution Notebook and fully read the conclusion link for a detailed description of the various ways to solve this problem!

This problem is common enough that is actually has its own Wikipedia Entry! Let's check the problem statement again:

This is a classic recursion problem: Given a target amount **n** and a list (array) of distinct coin values, what's the fewest coins needed to make the change amount.

For example:

If n = 10 and coins = [1,5,10]. Then there are 4 possible ways to make change:

- 1+1+1+1+1+1+1+1+1+1
- 5 + 1+1+1+1+1
- 5+5
- 10

With 1 coin being the minimum amount.

Solution

This is a classic problem to show the value of dynamic programming. We'll show a basic recursive example and show why it's actually not the best way to solve this problem.

Make sure to read the comments in the code below to fully understand the basic logic!

In [ ]:
```
def rec_coin(target,coins):
    '''
    INPUT: Target change amount and list of coin values
    OUTPUT: Minimum coins needed to make change

    Note, this solution is not optimized.
    '''

    # Default to target value
    min_coins = target

    # Check to see if we have a single coin match (BASE CASE)
    if target in coins:
        return 1

    else:
        # for every coin value that is <= than target
        for i in [c for c in coins if c <= target]:

            # Recursive Call (add a count coin and subtract from the target)
            num_coins = 1 + rec_coin(target-i,coins)

            # Reset Minimum if we have a new minimum
            if num_coins < min_coins:

                min_coins = num_coins

    return min_coins
```

Let's see it in action.

In [ ]:
```
rec_coin(63,[1,5,10,25])
```

The problem with this approach is that it is very inefficient! It can take many, many recursive calls to finish this problem and its also inaccurate for non standard coin values (coin values that are not 1,5,10, etc.)

We can see the problem with this approach in the figure below:

In [ ]:
```
from IPython.display import Image
Image(url='http://interactivepython.org/runestone/static/pythonds/_images/callTree.png')
```

Each node here corresponds to a call to the **rec_coin** function. The label on the node indicated the amount of change for which we are now computng the number of coins for. Note how we are recalculating values we've already solved! For instance 15 is called 3 times. It would be much better if we could keep track of function calls we've already made.

Dynamic Programming Solution

This is the key to reducing the work time for the function. The better solution is to remember past results, that way before computing a new minimum we can check to see if we already know a result.

Let's implement this:

In [ ]:
```
def rec_coin_dynam(target,coins,known_results):
    '''
    INPUT: This funciton takes in a target amount and a list of possible coins to use.
    It also takes a third parameter, known_results, indicating previously calculated results.
    The known_results parameter shoud be started with [0] * (target+1)

    OUTPUT: Minimum number of coins needed to make the target.
    '''

    # Default output to target
    min_coins = target

    # Base Case
    if target in coins:
        known_results[target] = 1
        return 1
    # Return a known result if it happens to be greater than 1
    elif known_results[target] > 0:
        return known_results[target]

    else:
        # for every coin value that is <= than target
        for i in [c for c in coins if c <= target]:

            # Recursive call, note how we include the known results!
            num_coins = 1 + rec_coin_dynam(target-i,coins,known_results)

            # Reset Minimum if we have a new minimum
            if num_coins < min_coins:
                min_coins = num_coins

                # Reset the known result
                known_results[target] = min_coins

    return min_coins
```

Let's test it!

In [ ]:
```
target = 74
coins = [1,5,10,25]
known_results = [0]*(target+1)

rec_coin_dynam(target,coins,known_results)
```

Test Your Solution

Run the cell below to test your function against some test cases.

Note that the TestCoins class only test functions with two parameter inputs, the list of coins and the target

In [ ]:
```
"""
RUN THIS CELL TO TEST YOUR FUNCTION.
NOTE: NON-DYNAMIC FUNCTIONS WILL TAKE A LONG TIME TO TEST. IF YOU BELIEVE YOU HAVE A SOLUTION
"""

from nose.tools import assert_equal

class TestCoins(object):

    def check(self,solution):
        coins = [1,5,10,25]
        assert_equal(solution(45,coins),3)
        assert_equal(solution(23,coins),5)
        assert_equal(solution(74,coins),8)

        print('Passed all tests.')

# Run Test

test = TestCoins()
test.check(rec_coin)
```

Conclusion and Extra Resources

For homework, read the link below and also implement the non-recursive solution described in the link!

For another great resource on a variation of this problem, check out this link: Dynamic Programming Coin Change Problem