Algorithms for Data Science
Lab 7
NCoins Dynamic Programming

Given the starting code below, fill in the missing parts of the DPcoins function to compute the optimal number of coins using dynamic programming. Also, for DPcoins add print statements to display the actual coins needed to produce the optimal number. For example, a sample run might look like the following:

```
Enter desired amount of change: 29
DAC:
optimal: 3  in time:  13.3 ms

DP:
5
12
12
optimal: 3  in time:  0.1 ms
```

```
# Below are two algorithms (DAC and DP) to compute the
# minimum number of coins required to produce A cents worth of change
# The DP version also prints out the coins needed to produce this min

from time import time

# Algorithm 1: Divide-and-Conquer
def DACcoins(coins, amount):
    if amount == 0: # The base case
        return 0
    else:           # The recursive case
        minCoins = float("inf")
        for currentCoin in coins:  # Check all coins
            # If we can give change
            if (amount - currentCoin) >= 0:
                # Calculate the optimal for currentCoin
                currentMin = DACcoins(coins, amount-currentCoin) + 1
                # Keep the best
                minCoins = min(minCoins, currentMin)
        return minCoins

# Algorithm 2: Dynamic Programming with Traceback
def DPcoins(coins, amount):
    # Create the initial tables
```

```
    # Fill in the base case(s)

    # Fill in the rest of the table

    # Perform the traceback to print result

    return -1  # return optimal number of coins

C = [1,5,10,12,25] # coin denominations (must include a penny)

A = int(input('Enter desired amount of change: '))
assert A>=0

print("DAC:")
t1 = time()
numCoins = DACcoins(C,A)
t2 = time()
print("optimal:",numCoins," in time: ",round((t2-t1)*1000,1),"ms")

print()
print("DP:")
t1 = time()
numCoins = DPcoins(C,A)
t2 = time()
print("optimal:",numCoins," in time: ",round((t2-t1)*1000,1),"ms")
```
