CS325: Group Assignment 2

Taylor Dinkins

Due: Friday, July 19, 11:59 P.M.

Your report must be typeset. *Each team member's name must be included. I am providing a file for this assignment as a Jupyter notebook with skeletons of functions, as well as most of the plotting code. You will be responsible for the report and filling in the functions in the notebook. I expect more in terms of the report details this time, since I provided the functions to calculate the runtimes and plots.*

Install Python and Jupyter

Jupyter notebook is a browser-based web application that lets you run Python code in cells. The notebook files themselves are shareable "documents". I understand you may not be familiar with Python or Jupyter, but it will hopefully not take you much effort to get it up and running. To install Jupyter Notebook: `https://jupyter.org/install`. See the sections for Jupyter Notebook with Anaconda (which you should install if you don't want to use *pip*), or the Jupyter Notebook with *pip* section. If you want to use Anaconda, which is probably easiest, visit `https://www.anaconda.com/distribution/`. There are versions for all major OS. This is mostly so I can grade your assignments in one platform, and so your plots are displayed in one place. Again, since I'm providing you most of the plotting code, hopefully the extra time of the install is offset by not having to write those yourself. You'll still need to type up the report section separately, since I won't make you learn how to use Markdown in the notebook. If you have major problems getting anything set up, please let me know and I'll absolutely talk with you about alternatives.

Coin Change Problem

For this project, you will investigate the **coin-change** problem:

> Given a cash register, how does one make change such that the fewest coins possible are returned to the customer? In this assignment, we explore two algorithmic solutions to this problem: a greedy algorithm and a dynamic programming algorithm.

Formally, an algorithm for this problem should take as input:

- A vector *coins* where $coins[i]$ is the value of the coin of the i^{th} denomination.
- *value* which is the amount of change we are asked to make.

The algorithm should return the number of coins used to make change for *value*, and a list of the coins used to make this value. The objective is to minimize the number of coins returned. In other words we want to minimize $len(coins\_used)$. You will implement a greedy algorithm and a dynamic programming algorithm for this.

Greedy Algorithm

One approach to coin change problem is a greedy approach:

- Return the largest value coin possible.
- Subtract the value of this coin from the amount of change to be made.
- Repeat.

This implementation is called `change_greedy`.

Dynamic Programming

We can use a dynamic programming table T indexed by $0, 1, 2, \ldots, value$ where $T[i]$ is the minimum number of coins needed to make change for i. Here, i is an integer such that $i \geq 0$, for the total value of currency we are making change for.

$$T[i] = \min_{j:coins[j] \leq i} \{T[i - coins[j]] + 1\}$$

We initialize $T[0] = 0$, and $T[i] = \infty$, for $i = 1, \ldots, value$. How do you store and return the coins that make the change for $value$? This implementation is called `change_dp`.

Example

Suppose $coins = [1, 10, 25, 50]$ and $value = 40$. `change_greedy` should return number of coins equal to 7 and $coins\_used = [1, 1, 1, 1, 1, 10, 25]$. `change_dp` should return number of coins equal to 4 and $coins\_used = [10, 10, 10, 10]$. If $value$ is changed to 76, both algorithms should return number of coins equal to 3 and $coins\_used = [50, 25, 1]$. For $value = 40$, the dynamic programming solution outperforms the greedy solution: it finds a solution requiring fewer coins. Thus, the greedy solution is not the optimal solution here.

Your report

Your team's report should include answers to the following questions.

1. Describe, in words, how you fill in the dynamic programming table in `change_dp`.
2. Give pseudocode for each algorithm, `change_greedy` and `change_dp`.
3. Give the theoretical run-time analysis for both the approaches.
4. Prove that the dynamic programming approach is correct by induction. That is, prove that

   $$T[i] = \min_{j:coins[j] \leq i} \{T[i - coins[j]] + 1\}, T[0] = 0$$

   is the minimum number of coins possible to make change for value i.
5. Suppose $coins = [1, 10, 25, 50]$. For each integer value of $value$ in $[2000, 2300]$ determine the number of coins that `change_greedy` and `change_dp` requires. Plot this number of coins (two lines, one for each `change_greedy` and `change_dp`) as a function of $value$. How do the two approaches compare?
6. For the above situation, determine (experimentally) the running time required by both approaches. Use $value = [1000, 1500, 2000, \ldots, 20000]$, and get the running times for each algorithm. How do the running times of the two approaches compare? Now, fix $value = 127$ and compare different $coins$ sizes. These are randomly chosen coin values from $[0, 100]$ of sizes $0, 2, \ldots, 100$. What is the *asymptotic* running time in terms of the number of $coins$ and the $value$ for each approach? Note that these are *log-log* plots. That is, I take the *log* of both the x values and y values (see `https:`

`//statisticsbyjim.com/regression/log-log-plots/`). I incorporate the *print* statements to print the slope of a best-fit line to this *log-log* plot for you. What this slope represents is the power of a power-relationship between the two axes. In other words, a straight line with a slope of 2 implies that $y = c * x^2$, for some constant c. You need to figure out how the slope I'm printing out relates to the run-time as we increase *value* and the number of *coins*. Hint: think of rounding the slope as helpful.

7. Suppose you are asked to calculate the number of possible ways (sets of coins) to make change for a given *value*, rather than the minimum number of coins. How would you modify the dynamic programming approach to solve this problem? What would you store in the table? How would it be indexed? Think of the cases: solutions including coin j, solutions not including coin j; for some i value ranging from $0, \ldots, value$. Describe this in words, pseudocode is not necessary. This should take some thought, so think of it as preparation for the midterm and approach it like a new dynamic programming problem.

.