1 Coin change

We revisit the coin change problem. Here, we have four types of coins: half-dollar (50 cents), quarter, dime and penny. We assume each coin has unlimited supply. Recall the objective is to give the smallest number of coins to change for \( n \) cents.

1. In this problem, will the greedy algorithm give optimal solution? Justify your answer.

2. Give a dynamic programming algorithm for the more general problem, where we have coins worthing \( d_1, d_2, \ldots, d_k \) (where \( d_1 = 1 \), the penny). Use the following definition for your algorithm: \( C[i, w] \) is equal to the smallest number of coins for choosing coins of \( d_1, d_2, \ldots, d_i \) only for making change of \( w \) cents.

3. Now suppose the government just issues a new one dollar coin. You only have one coin of one-dollar (the other coin types are still of unlimited supply). You are lazy: you do not want to design a new algorithm, but rather you want to re-use the algorithm in the previous step. Now tell me how you can solve this slightly changed problem (where you can use the single $1 coin) with the algorithm in the previous step (i.e. algorithm without knowing about the one $1 coin).

2 LCS

Show the dynamic programming table of the longest common substring problem for two sequences: \( S_1 = ABAABBA \) and \( S_2 = BAAABAB \). Also show how to find the LCS itself from the table.

3 Comparing two sequences

This is the common task for spell check. You are given two sequences, \( S_1 \) and \( S_2 \), with \( n \) and \( m \) symbols each. You want to find out how “similar” these two sequences are. One way is to add spaces into \( S_1 \) and \( S_2 \) so that the two sequences become the same length. For example, let \( S_1 \) be ANNDREW and \( S_2 \) be AMDREWS. We can add one space to each:

ANNDREW-
A-MDREWS.

Here, - stands for the added space. Intuitively, adding spaces makes the two sequences look similar and then we can compare them. We then score the two changed sequences as follows: we compare the corresponding positions (i.e. \( S1[i] \) with \( S2[i] \)); if the two symbols match, the cost is 0; if the two mismatch (possibly one of them is -), the cost is 1. Note, both symbols are -, the cost is 0. In the above example, the total cost of this comparison is equal to 3. Our goal is to find the smallest cost over all feasible ways of adding spaces into \( S_1 \) and \( S_2 \).

Now, give a dynamic programming algorithm for this problem. Write the pseudo-code for your algorithm, then explain why it works. Also give an analysis of running time. This problem is related to the LCS problem. So try to define subproblems similar to those of the LCS problem.