# Homework 3
# Colorado CSCI 5454

Alex Book

September 14, 2021

People I studied with for this homework: Cole Sturza
Other external resources used: N/A

## Problem 1

Consider using a matrix $A$ with dimensions $(|X| + 1) \times (|Y| + 1)$ to tabulate the optimal answers of previous subproblems (+1 to accommodate the base cases, which use the a null/empty character).

### Part a

### Subproblem

Let subproblem $(i, j)$ represent the longest common substring between $X[0 \dots i]$ and $Y[0 \dots j]$, necessarily including $X[i]$ and $Y[j]$.

### Recurrence

$$A[i,j] = \begin{cases} A[i-1, j-1] + 1 & (\text{if } X[i] = Y[j]) \\ 0 \end{cases}$$

### Correctness

Let there be a row and column of index $-1$ that aligns an empty character before the start of the characters of $X$ and $Y$ (to allow the indexing to remain simple and readable, suggested/approved by Bo during office hours).

### Base Case

The row and column with index $-1$ will all have values of zero, as the empty/null character will not match any other character (we make the assertion that the two empty characters (subproblem $(-1,-1)$) also are not evaluated as equal).

### Inductive step

Assume that all subproblems $(-1\ldots i-1, -1\ldots j-1)$ are correct. For subproblem $(i,j)$, if $X[i] = Y[j]$, we look at subproblem $(i-1, j-1)$ and add 1. That subproblem is correct by our inductive assumption, so adding 1 to that solution makes subproblem $(i,j)$ correct (extending the longest common substring that must include $X[i-1]$ and $Y[j-1]$ by 1 since the current characters are equal). If $X[i] \neq Y[j]$, the optimal solution is 0 (since we must include $X[i]$ and $Y[j]$, the longest common substring must be the empty string, which has length 0). Therefore subproblem $(i,j)$ is correct.

## Full Algorithm

Consider the following algorithm:

**Algorithm 1** Make Change DP

```
Input: Strings $X$ and $Y$
Set $A = 0$ matrix of dimension $(|X|+1) \times (|Y|+1)$ with the initial row having index $-1$
for i in 0 ... $|X|$ do
    for j in 0 ... $|Y|$ do
        if $X[i] == Y[j]$ then
            $A[i][j] = A[i-1][j-1] + 1$
        else
            $A[i][j] = 0$
        end if
    end for
end for
return $A$
```

## Running Time and Space Use

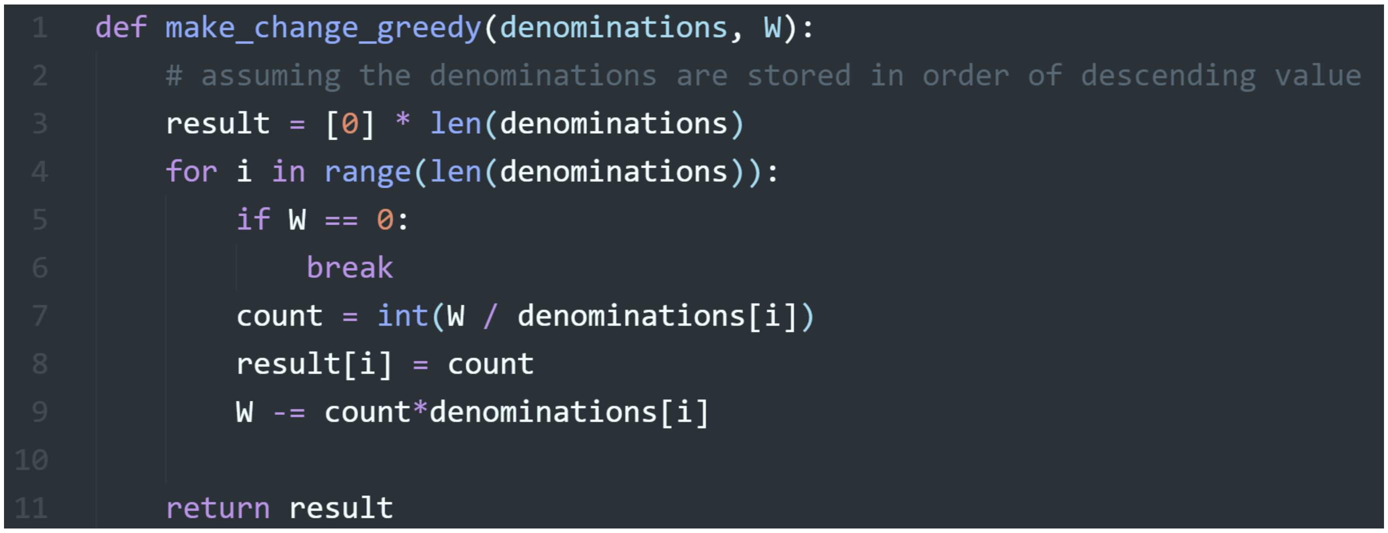
The running time for this algorithm is $O(|X| \times |Y|)$, as initialization and the operations in the nest **for** loops run in constant time, and the loops themselves run $|X|$ and $|Y|$ times, respectively. The space use is also $O(|X| \times |Y|)$, as it is dominated by matrix $A$, which is of dimensions $|X| \times |Y|$

## Part b

In order to return the substring itself, we find the maximum value in $A$. From there, we will trace back along the upward-left diagonal until reaching a value of zero, appending characters to the beginning of the string along the way (we will not append the character from the cell with value 0). This will give us the entire longest common substring between $X$ and $Y$.

## Problem 2

My implementation of the greedy change-making problem:



```python
def make_change_greedy(denominations, W):
    # assuming the denominations are stored in order of descending value
    result = [0] * len(denominations)
    for i in range(len(denominations)):
        if W == 0:
            break
        count = int(W / denominations[i])
        result[i] = count
        W -= count*denominations[i]

    return result
```

For the given examples:

USA coin denominations with $W = 72$ :
Total coins used: 5 (one 50 cent coin, two 10 cent coins, and two 1 cent coins)

USA coin denominations with $W = 2919$ :
Total coins used: 35 (twenty-nine 100 cent coins, one 10 cent coin, one 5 cent coin, and four 1 cent coins)

Coin denominations 1, 7, 15, 33, 51, and 99 with $W = 72$ :
Total coins used: 8 (one 51 cent coin, one 15 cent coin, and six 1 cent coins)

Coin denominations 1, 7, 15, 33, 51, and 99 with $W = 2919$ :
Total coins used: 31 (twenty-nine 99 cent coins, one 33 cent coin, and one 15 cent coin)

### Part b

Consider the following example:
$W = 6$, denominations $= [4, 3, 1]$.
The greedy solution would use three coins (one coin worth 4 and two coins worth 1 each).
The optimal solution, however, would use only two coins (two coins worth 3 each).

## Part c

Consider the following algorithm:

**Algorithm 2** Make Change DP

```
Input: Array $A$ with $n$ elements, integer $W$
Set $B = [0, \infty, \infty, \dots, \infty]$ // total length $W + 1$
for i in $1 \dots W + 1$ do
    for j in $0 \dots n$ do
        if $A[j] \leq i$ then
            if $B[i - A[j]] \neq \infty$ then
                $B[i] = \max(B[i], B[i - A[j]] + 1)$
            end if
        end if
    end for
end for
Set $C = [0, 0, \dots, 0]$ // total length $n$
curr = $W$
while curr $> 0$ do
    for j in $0 \dots n$ do
        temp = curr $-A[j]$
        if temp $\geq 0$ and $B[$ temp $] == B[$ curr $] - 1$ then
            curr = curr $-A[j]$
            $C[j] = C[j] + 1$
            break
        end if
    end for
end while
return $C$
```

### Subproblem Definition

The subproblem $B[i]$ represents the minimum number of coins needed to make change for $i$ cents.

### Computing the Final Answer

Following the idea presented in the subproblem definition, we can find our final answer at $B[W]$, which will hold the minimum number of coins needed to make change for $W$ cents.

**Recurrence**

$$B[0] = 0$$

$$B[i] = \underset{w_i \in A}{\operatorname{argmax}} B[i - w_i]$$

**Correctness of Recurrence - Base Case**

The minimum number of coins needed to make change for 0 sense is 0. Base case is correct.

**Correctness of Recurrence - Inductive Step**

Assume all subproblems held in $B[0 \ldots k-1]$ are correct. We look through the coin denominations and their corresponding subproblems ($B[i - w_i]$) and use whichever coin will give the minimum value for the current subproblem. Since all previous subproblems are correct by our assumption, simply picking which change-making method for the current cent value takes the least coins will leave us with the correct answer for the current subproblem.

**Reconstructing the Solution**

We set some variable **curr** equal to $W$ to start at the end of the array. We then try the denominations of each coin and find which value $B[\mathbf{curr} - w_i] = B[\mathbf{curr}] - 1$. Once found, we know that we must have used a coin of value $w_i$ and make a note as such, then set $\mathbf{curr} = \mathbf{curr} - w_i$. We do this until we arrive back to $B[0]$. This will give us the number of coins of each denomination used to make change for $W$ cents.

**Correctness of algorithm**

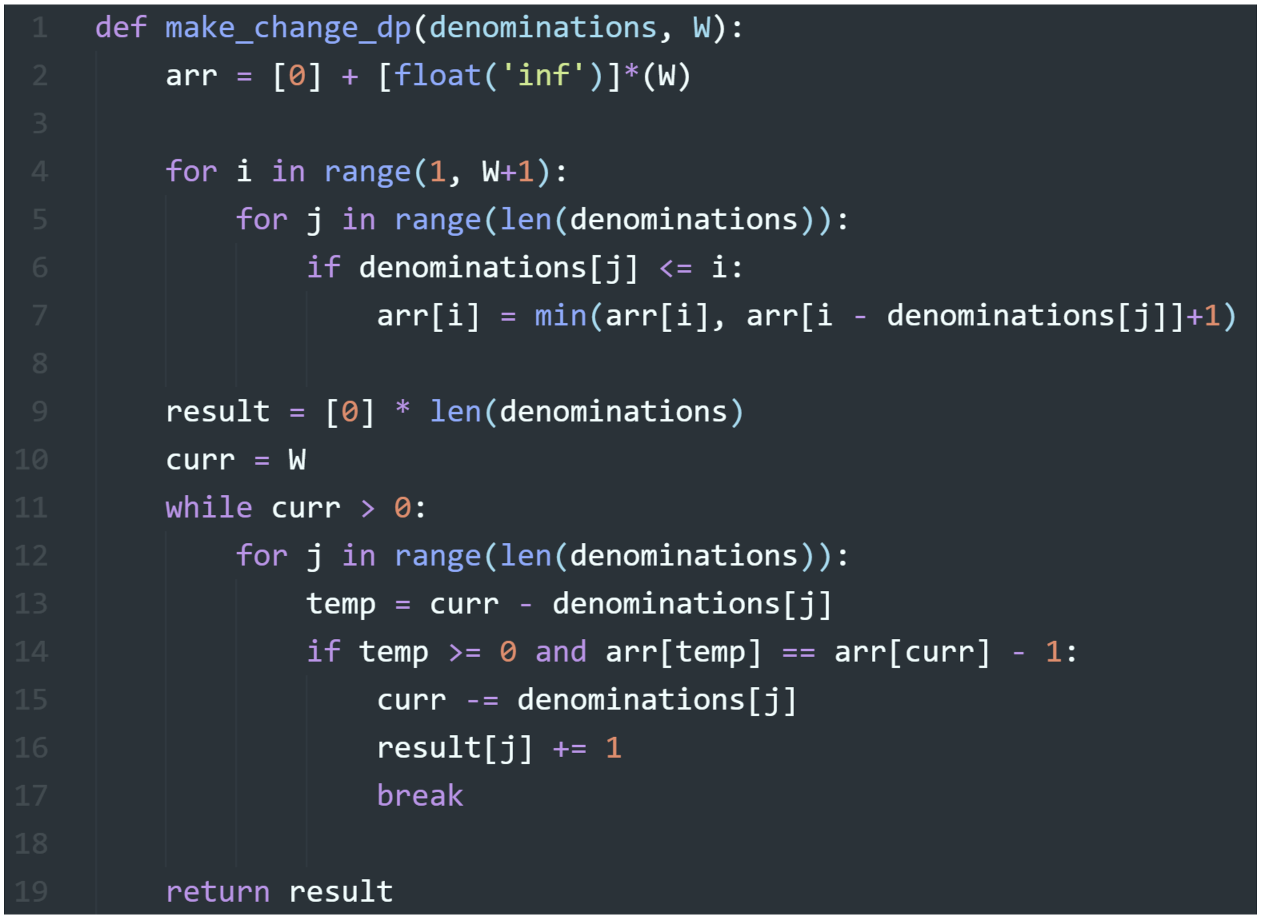
The algorithm given above implements the recurrence described above. Note the correctness proof of the recurrence. The correctness of the algorithm follows.

**Running Time and Space Use**

The running time of this algorithm is $O(W \times n)$, as initialization and the operations in the nested **for** loops run in constant time, while the **for** loops run $W$ and $n$ times, respectively. The **while** loop runs an amount of times equal to the minimum number of coins needed to make change for $W$ (which is at worst $W$ times), while the **for** loop runs at worst $n$ times. Space use is $O(\max(W, n))$, as it is dominated by the arrays $B$ (which is of length $W + 1 \approx W$) and $C$ (which is of length $n$).

## Part d

My implementation of the DP change-making problem:



```python
def make_change_dp(denominations, W):
    arr = [0] + [float('inf')]*(W)

    for i in range(1, W+1):
        for j in range(len(denominations)):
            if denominations[j] <= i:
                arr[i] = min(arr[i], arr[i - denominations[j]]+1)

    result = [0] * len(denominations)
    curr = W
    while curr > 0:
        for j in range(len(denominations)):
            temp = curr - denominations[j]
            if temp >= 0 and arr[temp] == arr[curr] - 1:
                curr -= denominations[j]
                result[j] += 1
                break

    return result
```

For the given examples:

USA coin denominations with $W = 72$ :
Total coins used: 5 (one 50 cent coin, two 10 cent coins, and two 1 cent coins)

USA coin denominations with $W = 2919$ :
Total coins used: 35 (twenty-nine 100 cent coins, one 10 cent coin, one 5 cent coin, and four 1 cent coins)

Coin denominations 1, 7, 15, 33, 51, and 99 with $W = 72$ :
Total coins used: 4 (one 51 cent coin and three 7 cent coins)

Coin denominations 1, 7, 15, 33, 51, and 99 with $W = 2919$ :
Total coins used: 31 (twenty-nine 99 cent coins, one 33 cent coin, and one 15 cent coin)