Kaleb Kim
10/29/24

CSDS 310 Assignment 3

Note: Arrays are zero-indexed.

Problem 1

Optimal Substructure

Let $f[c, t]$ represent the current number of coins to that make the amount t for a given coin value c, such that $t \geq c$.

$$f[c, t] = \begin{cases} 0 & \text{if } t = 0 \\ \min(f[t], 1 + f[t - c]) & \text{if } t > 0 \end{cases}$$

Pseudocode

```
procedure COIN_CHANGE(coins, t):
    dp ← ∞ array of t + 1 elements
    dp[0] ← 0
    for 1 ≤ i < t + 1:
        for coin in coins:
            if i ≥ coin:
                r[i] ← min(dp[i], 1 + dp[i − coin])
    return dp[t]
```

Proof

Let S be the optimal solution for amount t such that S is the minimum number of coins to construct t. Let p be the last coin used in S. Let S' be the solution for the remaining amount $t - p$, meaning $S = 1 + S'$. Suppose S' is unoptimal for the sake of contradiction. This means that there exists a better solution, B', for $t - p$. We have $S = 1 + S' > 1 + B'$. However, this contradicts that S is the optimal solution since $1 + B'$ is more optimal, proving that any optimal solution for amount t must contain an optimal solution.

Runtime Analysis

Let c represent the length of the given coins array. We have an outer loop that runs for $\Theta(t)$ time and each iteration contains an inner loop that runs for $\Theta(c)$. We also have r to be an array of $\Theta(t)$ size. Thus:

$$\text{Time complexity: } O(c \cdot t)$$
$$\text{Space complexity: } O(t)$$

Problem 2

Optimal Substructure

Let $D[i, j]$ represent the minimum number of edits required to make the two strings equal. Let the remove operation $R = D[i - 1, j]$, insert operation $I = D[i, j - 1]$, and replace operation $P = D[i - 1, j - 1]$.

$$D[i, j] = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ D[i-1, j-1] & \text{if } x_i = y_j \\ 1 + \min(R, I, P) & \text{otherwise} \end{cases}$$

Pseudocode

```
procedure EDIT_DISTANCE(s1, s2):
  m ← length of s1
  n ← length of s2
  dp ← 0 array of length (m + 1)(n + 1)
  for 0 ≤ i ≤ m:
    dp[i][0] ← i
  for 0 ≤ j ≤ n:
    dp[0][j] ← j
  for 1 ≤ i ≤ m:
    for 1 ≤ j ≤ n:
      if s1[i-1] = s2[j-1]:
        dp[i][j] ← dp[i-1][j-1]
      else:
        dp[i][j] ← 1 + min(dp[i][j-1], dp[i-1][j], dp[i-1][j-1])
  return dp[m][n]
```

Runtime Analysis

Let m, n be the lengths of $s1, s2$, respectively. We have two nested loops, the outer running for $\Theta(m)$ time and the inner for $\Theta(n)$ time. We also have dp to be a matrix of size $\Theta(m \times n)$. Thus:

$$\text{Time complexity: } O(m \cdot n)$$

$$\text{Space complexity: } O(m \cdot n)$$

Problem 3

Pseudocode

```
procedure MAX_CURRENCY(R, s, t, f):
  dp ← 0 array of length n × n
  prev ← −1 array of length n × n
  dp[s][0] ← 1
  for 1 ≤ k < n:
    for 1 ≤ curr ≤ n:
      for 1 ≤ prev_curr ≤ n:
        a ← dp[prev_curr][k-1] × R[prev_curr][curr] - f(k)
        if a > dp[curr][k]:
          dp[curr][k] ← a
          prev[curr][k] ← prev_curr
  best_k ← 0
```

```
best_amount ← dp[t][0]
for 1 ≤ k < n:
    if dp[t][k] > best_amount:
        best_amount ← dp[t][k]
        best_k ← k
return dp[best_amount][best_k]
```

Runtime Analysis

We have three nested loops: first for k exchanges running $\Theta(n)$ time, second for current currency running $\Theta(n)$ time, and third for previous currency running $\Theta(n)$ time. We also have dp and prev arrays each of size $\Theta(n \times n)$. Thus:

$$\text{Time complexity: } O(n^3)$$

$$\text{Space complexity: } O(n^2)$$