Kaleb Kim
11/10/24

CSDS 310 Assignment 3 v2

Note: Arrays are zero-indexed.

Problem 2

For this problem, let X be string 1 and Y be string 2.

Recurrence Relation

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 $P = D[i-1, j-1]$.

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

Optimal Substructure Proof

Let S be an optimal solution that has the minimum number of edits c required to convert $X[0...i]$ to $Y[0...j]$. For all S, if $i = 0$, i.e. $X[0...0]$ is an empty string, then the remaining edits required, D, is j to build Y. Similarly, if $j = 0$, i.e. $Y[0...0]$ is an empty string, then $E = i$. There exists no other edits to convert $X[0...i]$ to $Y[0...j]$, meaning S is optimal.

We have four remaining cases to prove:

- Case 1 $\left(X_i = Y_j\right)$: Let S' be optimal solution for S without the last match, i.e. the solution with the minimum edits a to convert $X[0...i-1]$ to $Y[0...j-1]$. Suppose S' is not optimal for the sake of contradiction. Let b represent the number of edits to convert $X[0...i-1]$ to $Y[0...j-1]$ in S, such that $b > a$. With the match operation requiring no additional edit, we have the total edits for S, c, be $c = 0 + a = a$. We also have $c = 0 + b = b$, but $b > a$ hence $b \neq a$. Then, $c \neq c$, a contradiction. Hence, S is optimal since S' must be optimal.
- Case 2 (deletion): Let S' be S without the last deletion, i.e. the solution with the minimum edits a to convert $X[0...i-1]$ to $Y[0...j-1]$. Suppose S' is not optimal for the sake of contradiction. Then, there exists b with greater edits to convert $X[0...i-1]$ to $Y[0...j-1]$, such that $b > a$. With the deletion operation adding one more edit to D' and D, we have $c = 1 + a$ for S. We also have $c = 1 + b$ for S, yet $b > a$ which means $b \neq a$. We have $c \neq c$, a contradiction. Thus, S is optimal since S' must be optimal.
- Case 3 (insertion): Similar contradiction for $X[0...i]$ to $Y[0...j]$.
- Case 4 (replacement): Similar contradiction for $X[0...i]$ to $Y[0...j]$.

Therefore, with all cases proven for the recurrence relation, S is an optimal solution that contains optimal solutions to its subproblems.

Runtime Analysis

Let m, n be the lengths of X, Y, 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

Recurrence Relation

$$M[j,k] = \begin{cases} 1 & \text{if } k = 0 \text{ and } i = s \\ 0 & \text{if } k = 0 \text{ and } i \neq s \\ \max_{i \in \text{ currencies}} \big(M[i,k-1] \times r_{ij} - f(k)\big) & \text{if } k > 0 \end{cases}$$

Pseudocode

```
procedure RECONSTRUCT_PATH(dp_i, t, k):
  if k = 0:
    return [s]
  S← k + 1 length array of 0′s
  S[0] ← s
  c ← t
  while k > 0:
    S[k] ← c
    c ← dp_i[c][k]
    k ← k − 1
  return S

procedure MAX_CURRENCY(rates, s, t, f):
  dp ← n × n matrix of 0′s
  dp_i ← n × n matrix of −1′s
  dp_j[s][0] ← 1
  for 0 ≤ k < n − 1:
    for 0 ≤ j < n:
      for 0 ≤ i < n:
        a ← dp[i][k] × rates[i][j]
        if a > dp[j][k + 1]:
          dp[j][k + 1] ← a
          dp_i[j][k + 1] ← i
  best_k ← 0
  best_a ← dp[t][0]
  for 1 ≤ k < n − 1:
    if dp[t][k + 1] − f(k + 1) > best_a:
      best_a ← dp[t][k+1]
      best_k ← k+1
  return RECONSTRUCT_PATH(dp_i, t, best_k)
```

Optimal Substructure Proof

Let S be an optimal solution for a sequence of k exchanges from source currency s to target t that gives the maximum profit. Let S' be the solution for a the sequence of $k-1$ exchanges to get to currency t for S, and let c be the currency we have after $k-1$ exchanges in S.

Suppose S' is not optimal for the sake of contradiction. Let c' be the currency we have after $k-1$ exchanges in S', such that $c' < c$. Then, we have a new sequence T that uses S'. Amount from S = $c \times r_{ij} - f(k)$. Amount from T = $c' \times r_{ij} - f(k)$.

Since the amount from $S > c' > c'$ > amount from S', new sequence gives more target currency.

This contradicts S being optimal. Therefore, S' must be optimal for getting to i in k-1 exchanges.

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:

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

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