Spring 2024

CS 412 (Algorithms: Design and Analysis)

Weekly Challenge 03: Dynamic Programming

Announced: Friday, March 29, 2024.
Deadline: Friday, April 12 , 2024 (11:59 pm PKT).
Total marks: 2.

Instructions: Submit **individually** your solution as a PDF with the file name as your *studentID.pdf*; typeset in LaTeX. You must submit your solution on Canvas.

---

1. (1 point) In compilers, the lexical analyzer scans the input source code character by character. When it encounters whitespace characters, it discards them without creating tokens. Whitespace characters do not contribute to the code's meaning and are often used for formatting and readability purposes only. After removing whitespace, the lexical analyzer continues to tokenize the remaining non-whitespace characters in the source code, identifying keywords, identifiers, literals, operators, and other language constructs.

   The lexical analyzer generates the sequence of tokens (without spaces) as an output. Consider the following pair of correct and incorrect programming language statements represented as a sequence of tokens, and devise a dynamic programming algorithm to identify the minimum number of edits required to transform an incorrect sequence to the correct sequence.

   Incorrect sequence: <int><int><;>
   Correct sequence: <int><var><.>

   1. Identify whether the problem has an optimal substructure property.
   2. Identify the smallest subproblem.
   3. Formulate a dynamic programming algorithm to solve the problem.
   4. Generate a dynamic programming table for the given problem.

Solution:

1. The problem does have an optimal substructure, because the minimum number of edits needed to transform the incorrect sequence to the correct one can be broken down into smaller subproblems, and the solution to the larger subproblem can be constructed from the smaller subproblems.

2. The smallest subproblem is transforming a single element of the incorrect sequence to the corresponding element of the correct sequence. The simplest subproblem would be when one of the strings is empty, so the number of edits will be the length of the non-empty sequence.

3. We can create a dp table where dp[i][j] represents the minimum number of edits required to transform the first i elements of the incorrect sequence to the first j elements of the correct sequence.

Code Listing 1: Dynamic Programming Solution

```
function minEdits(incorrect, correct):
    n = length(incorrect)
    m = length(correct)

    // Initialize dynamic programming table
    dp = 2D array of size (n+1) x (m+1)

    // Base case: Initialize the first row and column
    for i from 0 to n:
        dp[i][0] = i
    for j from 0 to m:
        dp[0][j] = j

    // Fill the dynamic programming table
    for i from 1 to n:
        for j from 1 to m:
            if incorrect[i] = correct[j]:
                dp[i][j] = dp[i-1][j-1]  // Characters match, no edit required
            else:
                dp[i][j] = 1 + min(dp[i-1][j-1], dp[i-1][j], dp[i][j-1])
                // Choose the minimum edit operation:
                // - Replace incorrect[i] with correct[j]
                // - Delete incorrect[i]
                // - Insert correct[j] after incorrect[i-1]

    // Return the minimum number of edits to transform the entire sequence
    return dp[n][m]
```

4. The following is the table:

Table 1: Dynamic Programming Table

		C	o	r	r	e	c	t
		1	2	3	4	5	6	7
	0	1	2	3	4	5	6	7
i	1	0	1	2	3	4	5	6
n	2	1	1	2	3	4	5	6
c	3	2	2	1	2	3	4	5
o	4	3	3	2	1	2	3	4
r	5	4	4	3	2	1	2	3
r	6	5	5	4	3	2	1	2
e	7	6	6	5	4	3	2	1
c	8	7	7	6	5	4	3	2
t	9	8	8	7	6	5	4	3
;	10	9	9	8	7	6	5	4

Here, the cell dp[i][j] represents the minimum number of edits required to transform the prefix of the incorrect sequence incorrect[1..i] to match the prefix of the correct sequence correct[1..j].