Andrew DeMaria~MariaDeslis~Tri Nguyen

CSCI406-FALL2012

TSP PROJECT

Professor Mehta

1. Theory: Devise a dynamic programming algorithm that solves the problem.  
   Demonstrate each of the following dynamic programming steps:  
   1. Describe the main idea of your approach including how you break the problem into smaller recursive problems.

The first step is to determine the recursive relationship for finding the next largest palindromic subsequence from a given string of characters as well as the base case for a palindromic subsequence.

There are two main base cases. The first is when the entire string (palindrome) is one character. If the string is one character, that character is returned. The second is when the string has two characters, but both characters are the same (i.e. aa). If this is the case, then the entire string is also returned.

Otherwise, the recursive relationship can be defined as follows: If the outer two characters are equivalent then the length of the longest palindromic subsequence is equal to two plus the longest palindromic subsequence of the inner string. If the outer two characters are not equivalent, then the palindromic subsequence will be the largest of either the string with the leftmost character removed or the rightmost character removed. This ensures that all possibilities are covered.

The issue with the above recursive function is that repeated calls to find the largest palindromic subsequence will occur. To mitigate this, a lookup table is created from the base case up. The largest subsequence of the entire string is then found by starting at the upper right cell in the table and tracing back the decisions made until a case is found.

* 1. Write pseudocode for your dynamic programming algorithm.

First step: Initialize the table

Table dimensions = length of string x length of string (rows x columns)

Initialize the main diagonal as the base case, a single character

row - start character

column - end character

FOR each diagonal starting at the main diagonal and propagating to the upper right:

FOR each cell In the diagonal:

Look at whether the two characters represented by the indices are equivalent

IF not equivalent

compare the cell to left and to the cell underneath

IF the left is larger in length

Store a reference to the left cell and its size

ELSE

Store a reference to lower cell and its size

IF they are equivalent

Store a reference to the cell diagonal (row + 1, col - 1) and its size

RETURN the longest subsequence at Table[0][n-1].

Example:

What the table looks like with just characters:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | G | O | O | D | S | T | U | F | F |
| G | G | O | OO | OO | OO | OO | OO | OO | FF |
| O |  | O | OO | OO | OO | OO | U | F | FF |
| O |  |  | O | D | S | T | U | F | FF |
| D |  |  |  | D | S | T | U | F | FF |
| S |  |  |  |  | S | T | U | F | FF |
| T |  |  |  |  |  | T | U | F | FF |
| U |  |  |  |  |  |  | U | F | FF |
| F |  |  |  |  |  |  |  | F | FF |
| F |  |  |  |  |  |  |  |  | F |

What the table looks like after generating it using our custom data structure that holds a reference to which cell it referenced, as well as the size of palindrome:

BC: Base Case

L: Left

DG: Diagonal

D: Down

(referenced cell, size of palindrome)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | G | O | O | D | S | T | U | F | F |
| G | BC, 1 | D, 1 | D, 2 | D, 2 | D, 2 | D, 2 | D, 2 | D, 2 | D, 2 |
| O |  | BC, 1 | DG, 2 | L, 2 | L, 2 | L, 2 | L, 2 | L, 2 | D, 2 |
| O |  | 0 | BC, 1 | D, 1 | D, 1 | D, 1 | D, 1 | D, 1 | D, 2 |
| D |  |  |  | BC, 1 | D, 1 | D, 1 | D, 1 | D, 1 | D, 2 |
| S |  |  |  |  | BC, 1 | D, 1 | D, 1 | D, 1 | D, 2 |
| T |  |  |  |  |  | BC, 1 | D, 1 | D, 1 | D, 2 |
| U |  |  |  |  |  |  | BC, 1 | D, 1 | D, 2 |
| F |  |  |  |  |  |  |  | BC, 1 | DG, 2 |
| F |  |  |  |  |  |  |  |  | BC, 1 |

* 1. Develop a traceback algorithm that returns a longest **subsequence**.

Psuedocode:

String traceback:

Start at the top right of the table.

IF the cell is the base case:

RETURN the character at that index

IF the cell is of reference type diagonal:

RETURN the character of the string at index row plus traceback( cell at diagonal) plus the character of the string at index col.

IF the cell is of reference type left:

RETURN traceback( cell at the left)

IF the cell is of reference type down:

RETURN traceback( cell at the right)

Please see attached code for implementation.

1. Theory: Derive the complexity of your algorithm in terms of n.

The bulk of the work done is inside the nested for-loops. Since the comparisons for each cell is done in constant time, we can formulate the nested for-loops as the following:



Therefore, the time complexity of the algorithm is .

1. Implementation: Implement your algorithm and submit the printout of yourcode. If you were unable to get your code to compile/run, please state this clearly.Although we don't plan to run everybody's code, we might choose a few groups randomlyand ask them to demonstrate that their code works. (It would look pretty bad if youclaim your code runs, but it doesn't!)

Please see attached paper(s).

1. Implementation: Demonstrate that your code works correctly by showing itsresults on a small example. (Provide the example you used so that we can verify thiswhile grading.)Small example of our implementation is as follows:  
   Say we have the string, “maria”.  
     
   The code would will produce a table as follows:  
   This is what table looks like with respect to the characters:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | m | a | r | i | a |
| m | m | a | r | i | aia |
| a |  | a | r | i | aia |
| r |  |  | r | i | a |
| i |  |  |  | i | a |
| a |  |  |  |  | a |

What the table looks like after generating it using our custom data structure that holds a reference to which cell it referenced:

BC: Base Case

L: Left

DG: Diagonal

D: Down

(referenced cell, size of palindrome)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | m | a | r | i | a |
| m | BC, 1 | D, 1 | D, 1 | D, 1 | D, 3 |
| a |  | BC, 1 | D, 1 | D, 1 | DG, 3 |
| r |  |  | BC, 1 | D, 1 | D, 1 |
| i |  |  |  | BC, 1 | D, 1 |
| a |  |  |  |  | BC, 1 |

The longest subsequence in the table is: aia

1. Submit group effort percentages and indicate who did what. These must be agreed uponby the group. See syllabus to determine how I will use this to compute your individualgrade.

Group scores as decided and discussed by group as a whole:

* + Andrew DeMaria : 33.33%
  + Maria Deslis : 33.33%
  + Tri Nguyen : 33.33%