cse 3500 hw 8

Casey Provitera

April 2024

1 Question 1(40 points)

In class, we studied the problem of finding the longest common subsequence (LCS) of two sequences X and Y (with n and m elements respectively).
Recall that we have a two-dimensional array LCS[i, j] for X and Y where 0 <= i <= n, 0 <= j <= m. Here, LCS[i, j]is equal to the length of LCS of X[1..i] and Y [1..j]. Now we let S1 = ABAABBA and S2 = BAAABAB. Here are your tasks.

1. (20 points)
   First, create and fill in the dynamic programming table LCS[n, m].

X	{}	A	B	A	A	B	B	A
{}	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
A	0	1	1	2	2	2	2	2
A	0	1	1	2	3	3	3	3
A	0	1	1	2	3	3	3	4
B	0	1	2	2	3	4	4	4
A	0	1	2	3	3	4	4	5
B	0	1	2	3	3	4	5	5

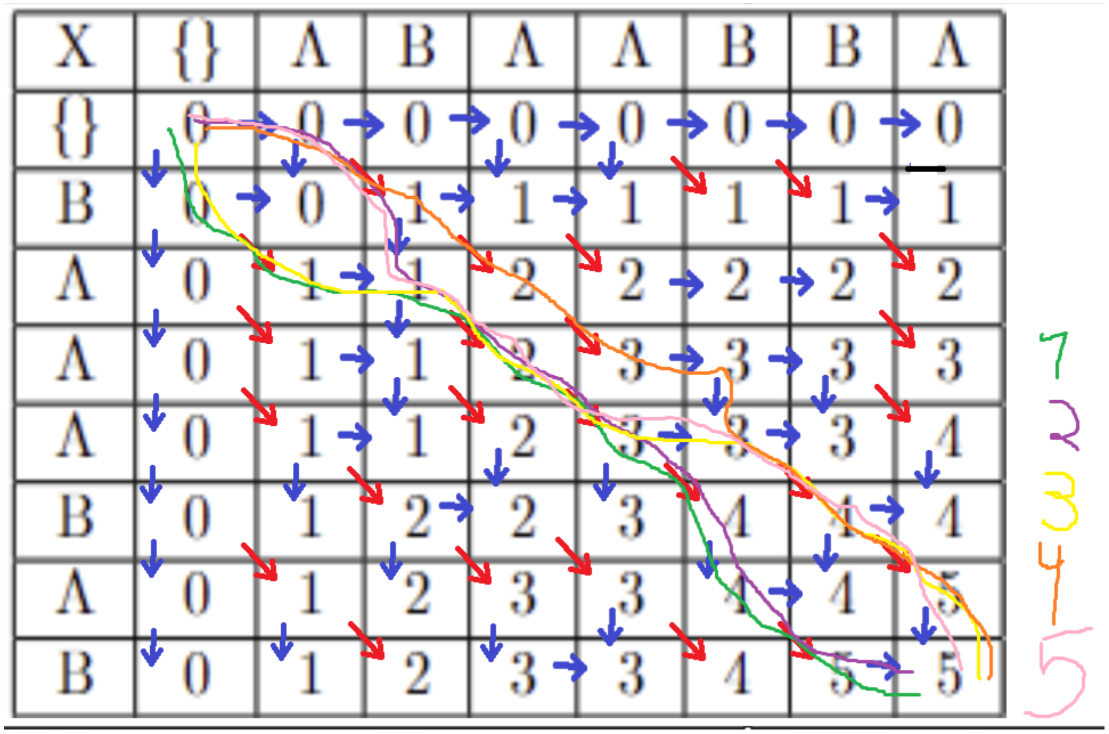
2. (10 points)
   Then, use the table to find the length of LCS between the two sequences.

   To use the table to find the length of the longest common subsequence all we have to do is look at the bottom right corner cell. This cell has the value of 5 so this is the length of our longest common subsequence.

3. (10 points)
   Finally, use the table to find the LCS itself (instead of just the length). Note: you need to show how you get the LCS: you need to mark the letters in S1 and S2 in the LCS while doing the trace back.
   In this example there are multiple subsequences that satisfy the max

length that we have found. We will add letters when we get to spots where we increased the because of matches.

Here is the chart showing how we find the subsequence:
Red denotes a match and blue denotes a non-match



X	{}	A	B	A	A	B	B	A
{}	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
A	0	1	1	2	2	2	2	2
A	0	1	1	2	3	3	3	3
A	0	1	1	2	3	3	3	4
B	0	1	2	2	3	4	4	4
A	0	1	2	3	3	4	4	5
B	0	1	2	3	3	4	5	5

Now to find the actual strings that do satisfy the subsequences we start at the bottom right corner cell and trace our way back to show how we got to the final result.
The result are as follows:
Result 1 - AAABB
Result 2 - BAABB
Result 3 - AAABA
Result 4 - BAABA
∗ Result 5 - BAABA ∗
As we can see there are 5 different ways to get from the initial having both strings be empty to the end square but we only get 4 distinct answers. Since results 4 and 5 give the same subsequence.

2 Question 2 (30 points)

Palindrome is a nonempty string over some alphabet that reads the same forward and backward. Examples of palindromes are civic, racecar, and aibohphobia (fear of palindromes).
Write an algorithm using Dynamic programming approach to find the longest palindrome subsequence. For example, given the input character, your algorithm should return 5, because the longest palindrome subsequence is carac.

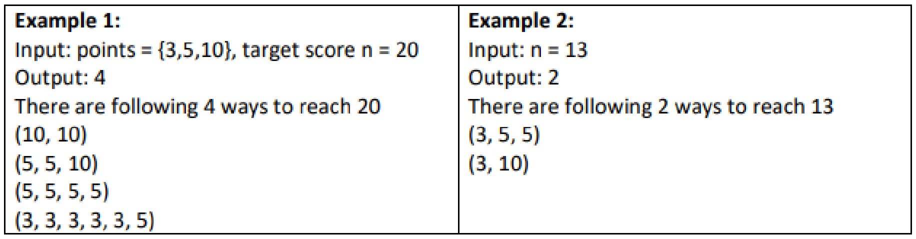
1. (5 points) First explain your high-level idea.
   Use LCS for strings with the input string. Since LCS takes two inputs we will have to change it so the second input is simply our original input string reversed. This will work because a palindrome is a word or phrase that is the same forwards and backwards so by comparing our word normally against the same word backwards we are testing for palindromes.

2. (10 points) Write the recurrence relation for your algorithm?
   Since this algorithm is going to mirror LCS almost exactly the recurrence relation is going to be the same as the LCS algorithm.
   Pal[i,j] = Pal[i-1,j-1] if $x_i = y_i$ else = max(Pal[i,j-1], Pal[i-1,j]

3. (10 points) Write the algorithm

```
• def PSL(input):
•     n = len(input)
•     Rinput = input[::-1]
•     table = [[0]*(n+1)]*(n+1)
•     for i in range(1,n+1):
•         for j in range(1,n+1):
•             if(input[i-1] == Rinput[j-1]):
•                 table[i][j] = table[i-1][j-1]+1
•             else:
•                 table[i][j] = max(table[i-1][j], table[i][j-1])
•     return table [n][n]
```

4. (5 points) What is the time complexity of your algorithm
   The time complexity is n^2 where n is the number of characters in the input string. Since as the string increases by n we need to do n^2 more comparisons.



Example 1:	**Example 2:**
Input: points = {3,5,10}, target score n = 20	Input: n = 13
Output: 4	Output: 2
There are following 4 ways to reach 20	There are following 2 ways to reach 13
(10, 10)	(3, 5, 5)
(5, 5, 10)	(3, 10)
(5, 5, 5, 5)	
(3, 3, 3, 3, 3, 5)	

3 Question 3 (30 points)

Imagine a game where players earn different points with each move. There's also a target score to reach. Our goal is to determine the number of possible ways to reach that target score using the given points.

1. (10 points) First explain your high-level idea.
   The general idea for this problem is to find the number of ways to get to every value with each move possible. We use one move at a time to see if there are any new ways to get to the values leading up to and including the target. This means once we have looped through the numbers 0 through the target for each move we will have found the total number of way to get to the target using the moves given.

2. (10 points) Write the algorithm

```
• def numCombos(target, values):
•     table = [0 for i in range(target+1)]
•     table[0] = 1
•     for move in values:
•         for i in range(move, target+1):
•             table[i] += table[i-move]
•     return table[target]
```

3. (10 points) What is the time complexity of your algorithm
   The time complexity of this algorithm is co-dependent on the number of moves available (m) and then size of the target (t) since there is a nested for loop. Which ends up with an expected time complexity of $\theta(m * t)$