1. Description:

**Brute Force:**

The idea is if we have two strings s1 and s2 where s1 ends at i and s2 ends at j, then the LCS is:

* if either string is empty, then the longest common subsequence is 0.
* If the last character (index i) of string 1 is the same as the last one in string 2 (index j), then the answer is 1 plus the LCS of s1 and s2 ending at i-1 and j-1, respectively. Because it's obvious that those two indices contribute to the LCS, so it's optimal to count them.
* If the last characters don't match, then we try to remove one of the characters. So we try finding LCS between s1 (ending at i-1) and s2 (ending at j) and the LCS between s1 (ending at i) and s2 (ending at j-1), then take the max of both.

**Example:**

S1=**1**21334 S2= **1**346

same so we mark them and move to next letters

[1]**2**1334 [1]**3**46

Different so we move to next letter

[1]2**1**334 [1]**3**46

Different so we move to next letter

[1]21**3**34 [1]**3**46

same so we mark them and move to next letters.

[1]21[3]**3**4 [1][3]**4**6

different so we move to next letter

[1]21[3]**3**4 [1][3]**4**6

same so we mark them

[1]21[3]3[**4**] [1][3][**4**]6

Since we reached the end of string s1,

[1]21[3]3[4] [1][3][4]6

Longest Subsequence is 134.

**Dynamic Programming:**

We use memorization in brute force technique to apply dynamic programming approach.

Let S1 = 1236

Let S2 = 136

S1 Length = 4

S2 Length = 3

Then we will make an array

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| L | 0 | 1 | 2 | 3 |
| 0 |  |  |  |  |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |

Now we fill row 0 and column 0 with 0’s

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| L | 0 | 1 | 2 | 3 |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 |  |  |  |
| 2 | 0 |  |  |  |
| 3 | 0 |  |  |  |
| 4 | 0 |  |  |  |

We use the Pseudo code (provided in the pseudo code section to fil the array)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| L | 0 | 1 | 2 | 3 |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 2 | 0 | 1 | 1 | 1 |
| 3 | 0 | 1 | 2 | 2 |
| 4 | 0 | 1 | 2 | 3 |

To know the length of the longest common subsequence for X and Y we have to look at the value L[XLen][YLen],  
i.e., L[4][3] = 3

So, Length of LCS = L[4][3] = 3

Since we know the length of longest common subsequence. We now will find the longest subsequence.

LCS Array Filling

(Pseudo Code In Pseudo Code Section)

|  |  |  |  |
| --- | --- | --- | --- |
| Array Index | 0 | 1 | 2 |
| Sequence Characters | 1 | 3 | 6 |

2. Pseudo Code

**Brute Force:**

LCS(s1, s2, i, j):

if(i == -1 || j == -1)

return 0

if(s1[i] == s2[j])

return 1 + LCS(s1, s2, i-1, j-1)

return max(LCS(s1, s2, i-1, j), LCS(s1, s2, i, j-1))

**Dynamic Programming Approach:**

if (S1[r-1] == S2 [c-1]) then

L[r][c] = L[r-1][c-1] + 1

else

L[r][c] = max { L[r-1][c], L[r][c-1] }

i.e., if current character in S1 and S2 are same

then they are in the resulting LCS.

To Fill LCS Array:

Set r = XLen, c = YLen, i = L[r][c]

while ( r > 0 && c > 0) do

if (X[r-1] == Y[c-1]) then

LCS[i-1] = X[r-1]

i = i - 1, r = r - 1, c = c - 1

else if (L[r-1][c] > L[r][c-1]) then

r = r - 1

else

c = c - 1

endwhile

3. Time Complexity

**Brute Force:**

Finding subsequences : 2n

Finding max subsequence: m

Total Complexity= m\*(2^n)

Iteration**:**

T(n) = 2\*T(n-1) + m

T(n-1) = 2\* T(n-2) + m

T(n) = 4T \* (n-2) + 3m

T(n-2) = 2\*T(n-3) + m

T(n) = 8T \* (n-3) + 7m

----

T(n)= 2^k \* T(n-k) + (2^ k -1) \* m

n-k=0 => k=n

T(n)= 2^n \* T(0) + (2^n -1)\* m

T(n)= 2^n +(2^n -1)m

T(n)=(2\*2^n)\*m-m

T(n)=O(m\*2^n)

4. Simulation Result

Note:

* Simulation Code and Output file for one RUN for each data size is saved in the Question’s Folder separately.
* Folder of **LCSDynamicProgramming** has a java file which contains both **BRUTEFORCE** and **DYNAMIC PROGRAMMING** CODES in separate functions because DATA SET had to be same for comparison purpose.
* Code Running Note: Kindly **Change File Path Of Output** before running the code.

**Brute Force:**

|  |  |  |
| --- | --- | --- |
| **Size of String 1** | **Size Of String 2** | **Average Execution To Find Longest Subsequence (ns)** |
| 10 | 10 | 30584 |
| 50 | 50 | 306595 |
| 80 | 80 | 798583 |
| 100 | 100 | 880140 |

**Dynamic Programming Approach:**

|  |  |  |
| --- | --- | --- |
| **Size of String 1** | **Size Of String 2** | **Average Execution To Find Longest Subsequence (ns)** |
| 10 | 10 | 24543 |
| 50 | 50 | 247315 |
| 80 | 80 | 393439 |
| 100 | 100 | 591669 |