Dynamic Programming

Longest Common Subsequence

A sequence is a subset of elements of a string if any number of elements from that string are removed, but the original order is preserved. For example, 'abc' has the substrings: {'a', 'b', 'c', 'ab', 'bc', 'ac', 'abc'}. Notice 'ca' is NOT a subsequence because 'a' and 'c' do not appear in the order that they appear in the original string.

The longest common subsequence algorithm takes two strings as input and outputs the longest subsequence that belongs to both strings.

Given string1 and string2 lengths n and m, use the following method:

1. Create a n+1 x m+1 matrix that has the characters of string1 as rows and the characters of string2 as columns with a column of padding on the far left and at the top.
2. Initialize all values to zeros.
3. Starting at (2, 2) if you are indexed by 1, compare the row and column characters. If they are the same, add 1 to the value in (i-1, j-1). If they are not, take the max value from the call one to the left left or one above: max((i-1, j), (i, j-1)).

Note: If there are multiple options, then it means the solution is non-unique.

The tables below demonstrate the progress of the algorithm.

Out[ ]:

-	-	a	m	d	f	b
-	0	0	0	0	0	0
a	0	0	0	0	0	0
d	0	0	0	0	0	0
a	0	0	0	0	0	0
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	**a**	m	d	f	b
-	0	0	0	0	0	0
a	0	1	0	0	0	0
d	0	0	0	0	0	0
a	0	0	0	0	0	0
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	a	**m**	d	f	b
-	0	0	0	0	0	0
a	0	1	1	0	0	0
d	0	0	0	0	0	0
a	0	0	0	0	0	0
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	a	m	**d**	f	b
-	0	0	0	0	0	0
a	0	1	1	1	0	0
d	0	0	0	0	0	0
a	0	0	0	0	0	0
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	a	m	d	**f**	b
-	0	0	0	0	0	0
a	0	1	1	1	1	0
d	0	0	0	0	0	0
a	0	0	0	0	0	0
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	a	m	d	f	**b**
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	0	0	0	0	0
a	0	0	0	0	0	0
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	**a**	m	d	f	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	0	0	0	0
a	0	0	0	0	0	0
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	a	**m**	d	f	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	0	0	0
a	0	0	0	0	0	0
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	a	m	**d**	f	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	0	0
a	0	0	0	0	0	0
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	a	m	d	**f**	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	0
a	0	0	0	0	0	0
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	a	m	d	f	**b**
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	0	0	0	0	0
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	**a**	m	d	f	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	1	0	0	0	0
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	a	**m**	d	f	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	1	1	0	0	0
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	a	m	**d**	f	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	1	1	2	0	0
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	a	m	d	**f**	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	1	1	2	2	0
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	a	m	d	f	**b**
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	1	1	2	2	2
b	0	0	0	0	0	0
a	0	0	0	0	0	0

-	-	**a**	m	d	f	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	1	1	2	2	2
b	0	1	0	0	0	0
a	0	0	0	0	0	0

-	-	a	**m**	d	f	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	1	1	2	2	2
b	0	1	1	0	0	0
a	0	0	0	0	0	0

-	-	a	m	**d**	f	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	1	1	2	2	2
b	0	1	1	2	0	0
a	0	0	0	0	0	0

-	-	a	m	d	**f**	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	1	1	2	2	2
b	0	1	1	2	2	0
a	0	0	0	0	0	0

-	-	a	m	d	f	**b**
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	1	1	2	2	2
b	0	1	1	2	2	3
a	0	0	0	0	0	0

-	-	**a**	m	d	f	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	1	1	2	2	2
b	0	1	1	2	2	3
a	0	1	0	0	0	0

-	-	a	**m**	d	f	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	1	1	2	2	2
b	0	1	1	2	2	3
a	0	1	1	0	0	0

-	-	a	m	**d**	f	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	1	1	2	2	2
b	0	1	1	2	2	3
a	0	1	1	2	0	0

-	-	a	m	d	**f**	b
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	1	1	2	2	2
b	0	1	1	2	2	3
a	0	1	1	2	2	0

-	-	a	m	d	f	**b**
-	0	0	0	0	0	0
a	0	1	1	1	1	1
d	0	1	1	2	2	2
a	0	1	1	2	2	2
b	0	1	1	2	2	3
a	0	1	1	2	2	3

Out[ ]:

-	-	**a**	m	**d**	f	**b**
-	0	0	0	0	0	0
a	0	**1**	**1**	1	1	1
d	0	1	1	**2**	2	2
a	0	1	1	**2**	**2**	2
b	0	1	1	2	2	**3**
a	0	1	1	2	2	**3**

The final table above shows the method of obtaining the final answer:

1. Find the last cell of the table.
2. If the value to the left or top is equivalent to the current value, move to that value, but do not select any characters.
3. If step 2 doesn't apply, move to the value to the top left corner (i-1, j-1) and select the corresponding character of the value you are moving from.

If there is a choice between left, top, and diagonal, it means there is more than one correct answer. This implementation prioritizes in this order: go left if possible, go up if possible, go diagonal.

The final solution for this problem is: 'adb'

Complexity

Time complexity: $O(n * m)$ where n and m are the lengths of the strings

- You must fill $O(n * m)$ cells which takes $O(1)$ work per cell

Space complexity: $O(n * m)$