Dynamic Programming

Longest Common Substring

A substring is a subset of consecutive elements of a string. For example, 'abc' has the substrings: {'a', 'b', 'c', 'ab', 'bc', 'abc'}. Notice 'ac' is NOT a substring because 'a' and 'c' do not appear consecutively in the original string.

The longest common substring algorithm takes two strings as input and outputs the longest substring 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). Leave the value as zero if they are not.

The tables below demonstrate the progress of the algorithm.

Out[ ]:

-	-	a	b	d	a	b	c
-	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

-	-	**a**	b	d	a	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	0	0	0
b	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

-	-	a	**b**	d	a	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	0	0	0
b	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

-	-	a	b	**d**	a	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	0	0	0
b	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

-	-	a	b	d	**a**	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

-	-	a	b	d	a	**b**	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

-	-	a	b	d	a	b	**c**
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

-	-	**a**	b	d	a	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

-	-	a	**b**	d	a	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	0	0
d	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

-	-	a	b	**d**	a	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	0	0
d	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

-	-	a	b	d	**a**	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	0	0
d	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

-	-	a	b	d	a	**b**	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	2	0
d	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

-	-	a	b	d	a	b	**c**
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	2	0
d	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

-	-	**a**	b	d	a	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	2	0
d	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

-	-	a	**b**	d	a	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	2	0
d	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0

-	-	a	b	**d**	a	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	2	0
d	0	0	0	3	0	0	0
a	0	0	0	0	0	0	0

-	-	a	b	d	**a**	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	2	0
d	0	0	0	3	0	0	0
a	0	0	0	0	0	0	0

-	-	a	b	d	a	**b**	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	2	0
d	0	0	0	3	0	0	0
a	0	0	0	0	0	0	0

-	-	a	b	d	a	b	**c**
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	2	0
d	0	0	0	3	0	0	0
a	0	0	0	0	0	0	0

-	-	**a**	b	d	a	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	2	0
d	0	0	0	3	0	0	0
a	0	1	0	0	0	0	0

-	-	a	**b**	d	a	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	2	0
d	0	0	0	3	0	0	0
a	0	1	0	0	0	0	0

-	-	a	b	**d**	a	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	2	0
d	0	0	0	3	0	0	0
a	0	1	0	0	0	0	0

-	-	a	b	d	**a**	b	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	2	0
d	0	0	0	3	0	0	0
a	0	1	0	0	4	0	0

-	-	a	b	d	a	**b**	c
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	2	0
d	0	0	0	3	0	0	0
a	0	1	0	0	4	0	0

-	-	a	b	d	a	b	**c**
-	0	0	0	0	0	0	0
a	0	1	0	0	1	0	0
b	0	0	2	0	0	2	0
d	0	0	0	3	0	0	0
a	0	1	0	0	4	0	0

Out[ ]:

-	-	**a**	**b**	**d**	**a**	b	c
-	0	0	0	0	0	0	0
a	0	**1**	0	0	1	0	0
b	0	0	**2**	0	0	2	0
d	0	0	0	**3**	0	0	0
a	0	1	0	0	4	0	0

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

1. Find the maximum value in the table and select the corresponding character.
2. Select each character going towards the top left until the value on the diagonal (i-1, j-1) is zero.

The final solution for this problem is: 'abda'

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: $O(n * m)$