CS532: Homework 4 (version 3)

Version 1 includes correction to the output of question 2.

Version 2 includes the suggestion to comment out the print statement in question 3.

Version 3 added question 4 and 5.

Brute Force String Alignment

Question 1: Asymptotic Behavior

In the class lecture notes 04, I discussed viewing an alignment as finding paths through an array where you start in the upper left corner and can move left, down, or diagnonally left-down until the lower right corner is reached.

I claimed that this brute force method was $(n+m)^3$. This is a mistake. Prove that it is actually $O(3^{n+m})$.

Question 2: Tracing Paths

Make a function that will trace out all paths through the 2 dimensional array. You do not actually have to create the two dimensional array for this problem, instead you will have the two one-dimenional arrays xArray and yArray that contain the two strings that need to be aligned.

The function will be passed the current location, x and y, from there it can go down, left, or diagonally down and left, making sure to not go outside the bounds of the array. When you reach the lower-right hand corner you should end.

Also, keep track of the alignment as you go, '-' for going left, '|' for going down, and '\' for going diagonally left-down. To use '\' in python, you might need to escape it with a '\'.

Call your routine `version1`, and it should take 5 parameters, x_array, y_array, two integers, x and y, and a string with the current alignment. You should call it with:

```
version1(["A","B","C","C","D"],["A","E","B","F","C","D"],-1,-1,"")
```

When the function reaches the lower-right hand corner, it should print the alignment and return.

Note that your routine will essentially be doing a depth-first search through paths throug the array.

Below are the first few lines that you should output.

```
-----||||||
----|-|||||
----||-||||
----|||-|||
----||||-||
----|||||-|
----||||||-
----|||||\
----||||\|
----|||\||
----||\|||
----|\||||
```

```
----\|||||
---|--|||||
---|-|-||||
---|-||-|||
---|-|||-||
---|-||||-|
```

Hand in your code for the function version1 (and the code for any other functions that it might call).

Question 3: Exponential Size

Show that this routine runs in exponential time. Do this by having x_array and y_array the same length, and varying this length from size 1 to 10, and timing how long the code takes to run. There is python module timeit that you should use (or you can use the time module and the command time.clock()). Feel free to comment out the print statement so that your code will run faster.

```
from timeit import default_time as timer

start = timer()

# do stuff

end = timer()
```

Print out a table of your results, showing size of the arrays versus running time. Roughly argue why your results indicate that this is exponential time.

Question 4: Scoring Alignments

Create the function version2 so that it also computes the score of the alignment. Do this by adding a 6th parameter to the function, to get the score so far. When version2 is initially called, this parameter should be given 0. For each down or left, 1 should be added to the score for the next recursive call. For the diagonal, 1 should be added only if the current character in the two arrays is not the same. When a full alignment is found, print out the score and the alignment before returning.

When called on the same arrays as in Question 2, the first part of my output looks as follows:

```
11 -----||||||
11 ----|-|||||
11 ----||-||||
11 ----|||-|||
11 ----||||-||
11 ----|||||-|
11 ----||||||-
9 ----|||||\
10 ----||||\|
10 ----|||\||
10 ----||\|||
10 ----|\||||
10 ----\|||||
11 ---|--|||||
```

```
11 ---|-|-||||
11 ---|-||-|||
11 ---|-|||-||
11 ---|-||||-|
11 ---|-|||||-
9 ---|-||||\
```

Hand in a copy of your code.

Question 5: Finding the best alignment

As the final part of doing a brute-force string alignment, create version3 that will return the best alignment. On the arrays from question 2, you should get the following result.

```
2 \|\\\\
```