Laboratory practice No. 1: Algorithm complexity

Sebastian Herrera Beleño
Universidad Eafit
Medellín, Colombia
sherrerab@eafit.edu.co

Jairo Miguel Marulanda Giraldo
Universidad Eafit
Medellín, Colombia
jmmarulang@eafit.edu.co

3) Practice for final project defense presentation

Note:
For the exercise 1.1 we implemented two different algorithms, one designed by ourselves and another designed by a fellow student. For most of the analysis we will use the later since it proved more consistent and faster (for most cases). However, we believe our algorithm may have an advantage for certain cases that will be studied in what follows. Our algorithm was denominated subCadenaMax(s1,s2), where s1 and s2 are strings. The other algorithm was denominated lcs(I,j,X,Y), were X and Y are strings, i and j are integers.

3.1

For lcs():
Worst case:
T(p) = T(p-1) + T(p-1) + c3 = c3((2**p)-1) + c4*2**p-1

With ci constant

Then
T(p) = O(2**p) with p being the sum of the length of the two strings

For subCadenaMax ():
Worst case:
T(n,m) = c3*2**n-1 + c4 + c3*2**m-1 + c4 + nlog(n*(n+1)/2) + c5 + mlog(m*(m+1)/2) + c5 + c6*n + c7*n*m + c8*n

With ci constant

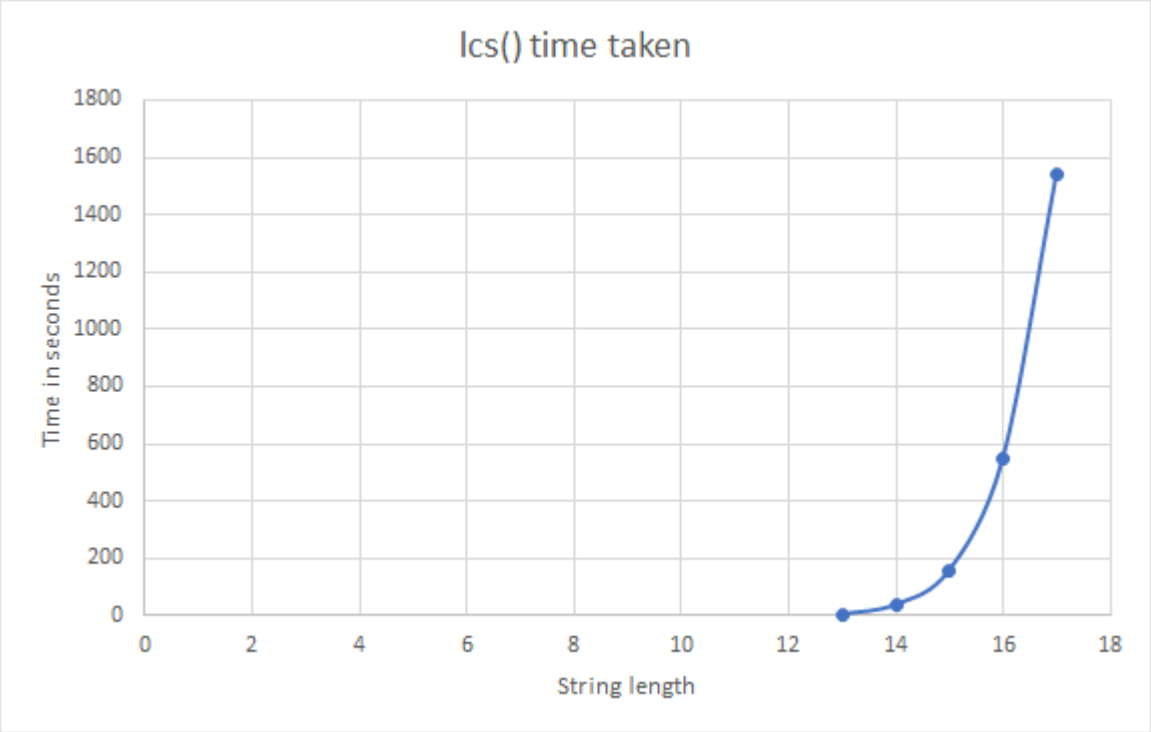
Then
T(n,m) = O(2**n + 2**m) with n being the lenght of s1 and m the lenght of s2
If n = m
Then
T(n,m) = O(2**n)

Notice how the big O notation applies a greater approximation in this case compared to the lather. For smaller values of n and m this may affect the accuracy of the prediction.

3.2

For lcs():

We can see the exponential function that we calculate in the previous step. As the length of the string increases, we see a large increase in the time take for the algorithm to be executed.



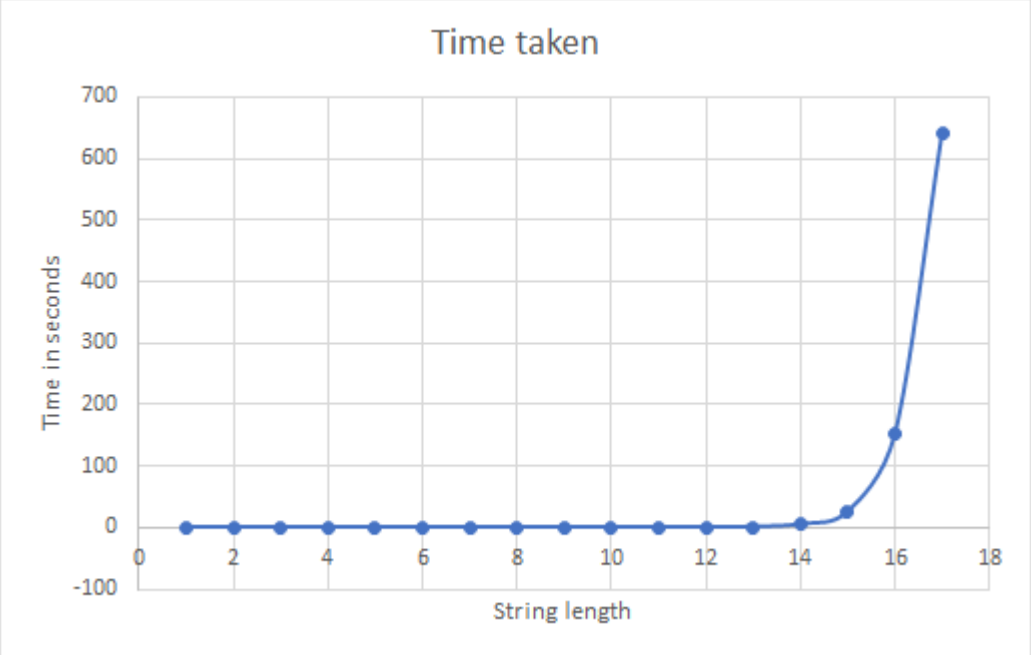
We cannot execute the algorithm with a string with a length of 20 for the long waiting time required, with a length of 17 the algorithm takes 1539.56955 seconds or 25 minutes to be executed.

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

With this scenario we can conclude that this type of algorithm is not suitable for the type of necessity. A DNA strand with a length of 300,000 is extremely inefficient apply recursion.

For subCadenaMax ():

For this calculation we used n = m for simplicity.
Similarly, we can observe the exponential behavior of the function as expected.



We cannot execute the algorithm with a string with a length of 20 for the long waiting time required, with a length of 17 the algorithm takes 642.13 seconds or around 10 minutes to be executed.

With this scenario we can conclude that this type of algorithm is not suitable for the type of necessity. A DNA strand with a length of 300,000 is extremely inefficient apply recursion.

In conclusion, subCadenaMax() appears to be faster. However, it is wort noting that is hard to predict how this observation may stand as the size of the strings increases, since the data used was relatively small.

3.3

For both algorithms. Since the algorithm appears to grow exponentially, such an enormous data set, such a large string, would take an even more enormous amount of time to process. In addition, one does not wish to compare just two strings of DNA, but several between each other. This makes the algorithm impractical for this application.

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

3.4

The method groupSum5(int start, int[] nums, int target) tells us if it is possible to generate a given int number by adding up different combinations of numbers in an int array, with the additional constraint that all multiples of five contained in the array must be used and that, if the value immediately following a multiple of five is a one, it must not be chosen.
The method requires three parameters: an int denominated start, an int list denominated nums and another int denominated target, in that order.
start represents the position of nums where the search will begin. nums is the array of values to be added up. target is the number we wish to determine if it is possible to generate. The method evaluates every possible combination of numbers of nums until it finds one which addition equals target, then it returns Boolean True. If it can't find any combination that fits the requirements, it returns false.
The method functions recursively by using start to represent the position of a number in nums, that may be subtracted or not from target. If by the end of the process target equals cero, it means that there is a combination of numbers in nums that fits the requirements. It goes as follows:
if the current number is a multiple of five, such number is subtracted from the target number, then groupSum() is called again but evaluating the next number in nums, and finally the value returned by this new call is returned as well. This guarantees that multiples of five are always used in the calculation. Similarly, if the number in the current position is a one, and the previous number was a 5, the current number is not subtracted and everything else goes as described. This ensures the second condition is satisfied.
On the other hand, if the number is not as the above, two different calls to groupSum() are generated, one where the current number is subtracted and another where it is not. In this case, the method returns true if any of the paths also returns true, since that means there exist a fitting combination.
Finally, when we reach the end of the array (that is, when target equals the length of nums), we evaluate if target equals cero. If it does, the method returns true, false is returned otherwise.

3.5

Let T(n) be the complexity function of each exercise, then

3.5.1 triangle exercise

We get
$T(n) = c1 + c2 + c0$ if $n = 0$
$T(n) = T(n-1) + c4 = c4n + c5$ else

with ci being a constant

then
$T(n) = O(n)$ with n being the number of rows of the triangle

3.5.2 bunnyEars2 exercise

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

We get
T(n) = c1 + c2 + c0 if n = 0
T(n) = T(n-1) + T(n-1) + c5 = c5*((2**n) -1) + c6*2**n-1 else

with ci being a constant

then
T(n) = O(2**n) with n being the number of bunnies

3.5.3 sumDigits exercise

We get
T(m) = c1 + c2 + c0 if m <= 1
T(m) = T(m-1) + c4 = c4m + c5 else

with ci being a constant

then
T(m) = O(m) with m being the number of digits of the evaluated number n

3.5.4 count7 exercise

We get
T(m) = c1 + c2 + c3 + c0 if m = 1
T(m) = T(m-1) + T(1) + c5 else
However T(1) = c3 + c2 + c1
Then
T(m) = T(m-1) + c7 = c7m + c8 else

with ci being a constant

Then
T(m) = O(m) with m being the number of digits of the evaluated number n

3.5.5 fibonacci exercise

We get
T(m) = c0 + c1 + c2, if n <=1
T(m) = c3 + T(m-1) + T(m-2) = T(n) = c3*2**m + c4 else

with ci being a constant

then
T(m) = O(2**m) with m being the number of digits of the evaluated number n

3.5.6 groupSum6 exercise

We get

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

T(n) = c0 + c1 + c2 + c3 + c4 if n = 0
Worst case is
T(n) = T(n-1) + T(n-1) + C = C*((2**n)-1) + c11*2**n-1

with ci being a constant

then
T(n) = O(2**n) with n being the length of the array evaluated nums

3.5.7 groupNoAdj exercise

We get
T(n) = c0 + c1 + c2 + c3 + c4 if n = 0
Worst case is
T(n) = T(n-2) + T(n-1) + C = Fibbonachi´s recursive ecuation = c8*2**n + c9

with ci being a constant

then
T(n) = O(2**n) with n being the length of the array evaluated nums

3.5.8 groupSum5 exercise

We get
T(n) = c0 + c1 + c2 + c3 + c4 if n = 0
Worst case is
T(n) = T(n-1) + T(n-1) + C = C*((2**n)-1) + c9*2**n-1

with ci being a constant

then
T(n) = O(2**n) with n being the length of the array evaluated nums

3.5.9 groupSumClump exercise

We get
T(n) = c0 + c1 + c2 + c3 + c4 if n = 0
Worst case is
T(n) = T(n-1) + T(n-1) + C = C*((2**n)-1) + c11*2**n-1

with ci being a constant

then
T(n) = O(2**n) with n being the length of the array evaluated nums

3.5.10 splitArray exercise

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

We get
T(n) = c0 + c1 + c2 + c3 + c4 if n = 0
Worst case is
T(n) = c8*n + c9 + c1 + C*((2**n)-1) + c18*2**n-1 + c18 + c4 + C

with ci and C being constants

Note that split array is depended of two auxiliar functions whose complexities were calculated in order to calculate the lather.

then
T(n) = O(2**n) with n being the length of the array evaluated nums

Note: A step by step process for all the calculations is shown as comments inside the code.

4) Practice for midterms

4.1 *1.c, 2.c, 3.b.*
4.2 Line 9:
floodFillUtil(screen, x+1, y+1, prevC, newC, N, M)

Line 10:
floodFillUtil(screen, x-1, y-1, prevC, newC, N, M)

Line 11:
floodFillUtil(screen, x-1, y+1, prevC, newC, N, M)

Line 12:
floodFillUtil(screen, x-1, y-1, prevC, newC, N, M)

4.3 *b.T(n,m) = C * n * m**2*
4.4 c. T(n) = T(n-1) + T(n-2) + c, que es O(2**n)
4.5 Line 3:
return True

Line 4:
b. If(s.charAt(0) == s.charAt(s.length() -1))

References

Ramirez Lopera, J. M. (2021). lcs method source code (Version 1) [source code].

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473