Laboratory practice No. 1: Recursion

Andres Felipe Agudelo Ortega
Universidad EAFIT
Medellín, Colombia
afagudeloo@eafit.edu.co

Juan Camilo Gutiérrez Urrego
Universidad EAFIT
Medellín, Colombia
jcgutierru@eafit.edu.co

February 28, 2021

3) Practice for final project defense presentation

3.1. Calculate complexity for the worst case of exercise 1.1

```
def lcs_aux(cadenaA ,cadenaB ,m ,n ,T):
    for i in range(m):                                  # C1 * m
        for j in range(n):                              # C2 * (n*m)
            if cadenaA[i] == cadenaB[j]:                # C3 * (n*m)
                T[i][j] = T[i-1][j-1]+1                 # (C3+C4) * (n*m)
            else:                                       # C5 * (n*m)
                T[i][j] = max(T[i-1][j],T[i][j-1])      # (C5+C6) * (n*m)
    return (np.amax(T),T)                               # C7

Calculate T(n)
T(m,n) = C1*m + C2(n*m) + C3(n*m) + (C3+C4)(n*m) + C5(n*m) + (C5+C6)(n*m) + C7
T(m,n) = C1*m + (C2+C3+(C3+C4)+C5+(C5+C6))(n*m) + C7

Apply O() notation
T(m,n) = O(C1*m + (C2+C3+(C3+C4)+C5+(C5+C6))(n*m) + C7)

Apply addition rule
T(m,n) = O((C2+C3+(C3+C4)+C5+(C5+C6))(n*m))

Apply product rule
T(m,n) = O(n*m)
```

3.2. *Take times for 20 different problem sizes, generate a graph and discuss the results. Estimate how long this algorithm is going to delay on the longest common subsequence between two Mitochondrial DNAs (which have about 300,000 characters each).*

For this problem, we programmed a cycle which stored the time that it took to the **lcs** algorithm to be executed in 20 different length sizes for the two Mitocondrial DNAs sequences, from 1 to 10000 in steps of 500, and then we plotted the result to see the time behaviour against the size of n and m.

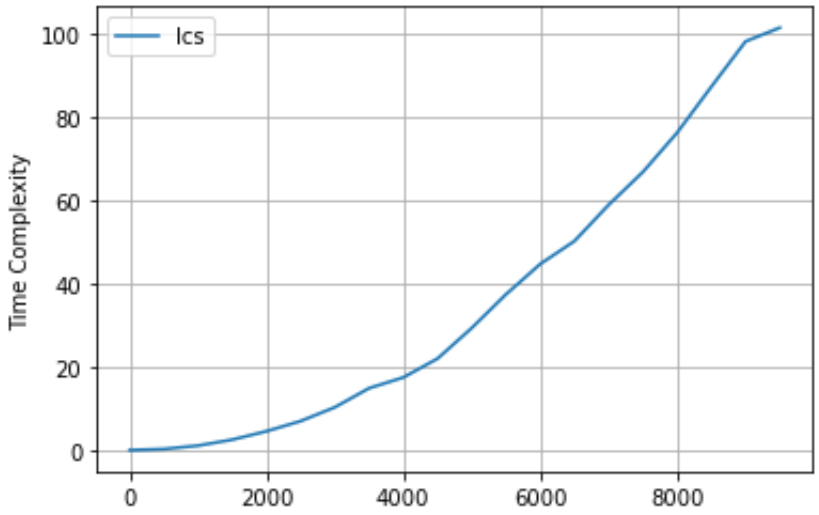

Figure 1: Mitocondrial DNAs sequences size vs Time Complexity $O(n * m)$

3.3. *¿Is the complexity of the algorithm in exercise 1.1 appropriate to find the longest common sub-sequence between Mitochondrial DNAs datasets?*

The complexity of the longest common sub-sequence algorithm is appropriate to be applied on the Mitochondrial DNAs datasets, because for a problem of 16000 elements the execution time takes around 6 minutes. However for bigger datasets, like the ones with sizes of 300000 each one, the algorithm would take too much time to process all the matrices, because the complexity of the matrices of the longest common sub-sequence between almost equal sized matrices or same sizes matrices is $O(n^2)$ as we saw in Figure 1, however, on the other hand, we can infer that if one of the arrays compared is significantly smaller than the other so that we can consider it even constant, the Complexity graphic will tend to be O(n), being n the longest array. See Figure 2

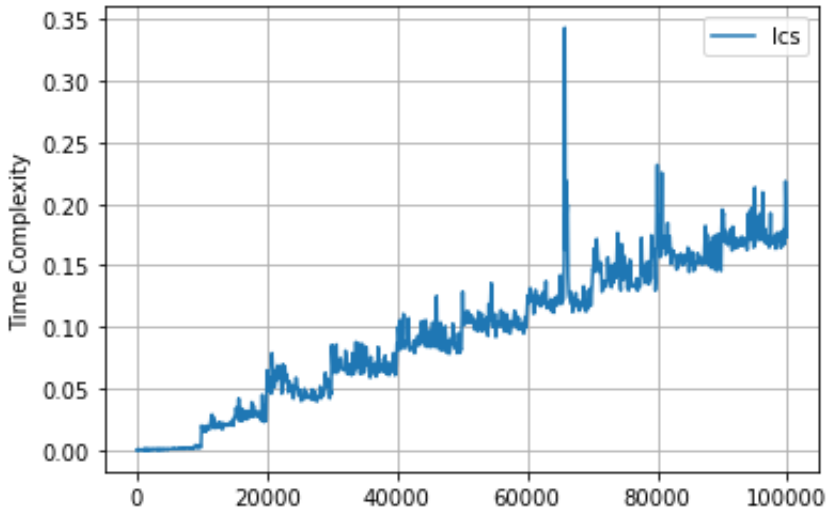

Figure 2: Mitocondrial DNAs sequences size vs Time Complexity, m being 10000 times shorter than n. Can be considered as $O(n)$

3.4. Explain with your on words how GroupSum5 works

GroupSum5 is an algorithm that takes elements from an array, sums them and see if it is possible to get to a target by summing a certain combination of them. But this algorithm has 2 conditions in order to reach this target, the first one says that if an element from the array is multiple of 5, that element must be added in order to complete the target. And the other condition is that if a number 1 is right after a multiple of 5, then that number 1 can not be added to the target.

3.5. Calculate the complexity of the exercises of the numerals 2.1 and 2.2

For this point we will only show the worst case of each algorithm after having divided every line of the code in all the constants and functions of its complexity. This division will be in the attached python files on the "ejercicioEnLinea" GitHub folder.

3.5.1 Recursion I

i. fibonacci

Worst case:

$$T(n) = c_6 + T(n-1) + T(n-2)$$

Recurrence equation solution:

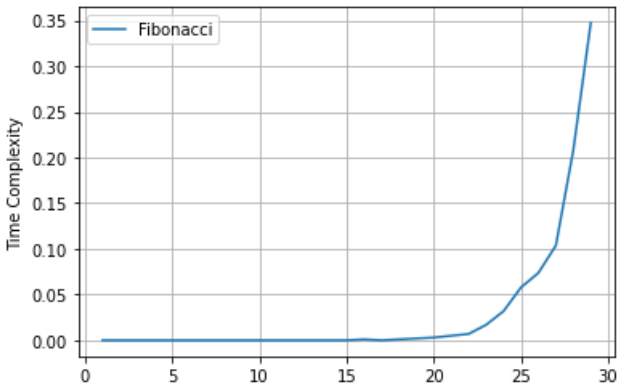

Figure 3: Fibonacci Complexity

Big O notation:

$$T(n) = O(2^n)$$

ii. PowerN

Worst case:

$$T(n) = T(n) = c_3 + T(n-1)$$

Recurrence equation solution:

$$T(n) = c_3 n + c_1$$

Big O notation:

$$T(n) = O(n)$$

iii. ChangeXY

Worst case:

$$T(n) = c_5 + T(n-1)$$

Recurrence equation solution:

$$T(n) = c_5 n + c_1$$

Big O notation:

$$T(n) = O(n)$$

Professor Mauricio Toro Bermudez
Phone: (+57)(4)2619500 Ext. 9473. Office: 19 – 627
E-mail: mtorobe@eafit.edu.co

iv. ChangePi

Worst case:

$$T(n) = c_5 + T(n-1)$$

Recurrence equation solution:

$$T(n) = c_5 n + c_1$$

Big O notation:

$$T(n) = O(n)$$

v. NoX

Worst case:

$$T(n) = c_3 + T(n-1)$$

Recurrence equation solution:

$$T(n) = c_3 n + c_1$$

Big O notation:

$$T(n) = O(n)$$

3.5.2 Recursion II

i. split53

Worst case:

$$T(n) = c_9 + T(n-1) + T(n-1)$$

Recurrence equation solution:

$$T(n) = c_9(2^n - 1) + c_1 2^{(n-1)}$$

(where c_1 is arbitrary)

Big O notation:

$$T(n) = O(n^2)$$

ii. splitArray

Worst case:

$$T(n) = c_3 + T(n-1) + T(n-1)$$

Recurrence equation solution:

$$T(n) = c_3(2^n - 1) + c_1 2^{(n-1)}$$

(where c_1 is arbitrary)

Big O notation:

$$T(n) = O(n^2)$$

iii. splitOdd10

Worst case:

$$T(n) = c_8 + T(n-1) + T(n-1)$$

Recurrence equation solution:

$$T(n) = c_8(2^n - 1) + c_1 2^{(n-1)}$$

(where c_1 is arbitrary)

Big O notation:

$$T(n) = O(n^2)$$

iv. GroupSum5

Worst case:

$$T(n) = c_9 + T(n-1) + T(n-1)$$

Recurrence equation solution:

$$T(n) = c_9(2^n - 1) + c_1 2^{(n-1)}$$

(where c_1 is arbitrary)

Big O notation:

$$T(n) = O(n^2)$$

v. GroupSum6

Worst case:

$$T(n) = T(n) = c7 + T(n-1) + T(n-1)$$

Recurrence equation solution:

$$T(n) = c_7(2^n - 1) + c_1 2^{(n-1)}$$

(where c_1 is arbitrary)

Big O notation:

$$T(n) = O(n^2)$$

3.6. Explain with your own words the variables from exercise 3.5

3.6.1 Recursion I

i. fibonacci

Big O notation:

$$T(n) = O(2^n)$$

- n is the input of the mathematical Fibonacci function, the n^{th} element of Fibonacci

ii. PowerN

Big O notation:

$$T(n) = O(n)$$

- n is the exponential part of the base number

iii. ChangeXY

Big O notation:

$$T(n) = O(n)$$

- n is the size of the string

iv. ChangePi

Big O notation:

$$T(n) = O(n)$$

- n is the size of the string

v. NoX

Big O notation:

$$T(n) = O(n)$$

- n is the size of the string

3.6.2 Recursion II

i. split53

Big O notation:

$$T(n) = O(n^2)$$

- n is the size of the string

ii. splitArray

Big O notation:

$$T(n) = O(n^2)$$

- n is the size of the string

iii. splitOdd10

Big O notation:

$$T(n) = O(n^2)$$

- n is the size of the string

iv. GroupSum5

Big O notation:

$$T(n) = O(n^2)$$

- n is the size of the string

v. GroupSum6

Big O notation:

$$T(n) = O(n^2)$$

- n is the size of the string

4) Practice for midterms

4.1.

4.1.1 a

4.1.2 c

4.1.3 a

4.2.

4.2.1 floodFillUtil(screen, x+1, y+1, prevC, newC, N, M)

4.2.2 floodFillUtil(screen, x-1, y-1, prevC, newC, N, M)

4.2.3 floodFillUtil(screen, x-1, y+1, prevC, newC, N, M)

4.2.4 floodFillUtil(screen, x+1, y-1, prevC, newC, N, M)

4.3.

4.3.1 b

4.4.

4.4.1 c

4.5.

4.5.1 b

4.5.2 b

Professor Mauricio Toro Bermudez
Phone: (+57)(4)2619500 Ext. 9473. Office: 19 – 627
E-mail: mtorobe@eafit.edu.co