Laboratory practice No. 1: Recursion and complexity

Isabella Pérez Serna
Universidad Eafit
Medellín, Colombia
iperezs2@eafit.edu.co

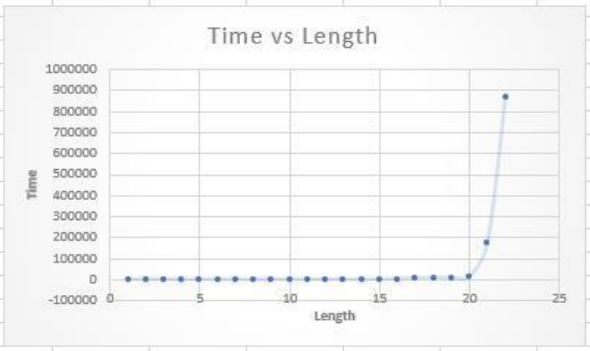
Mariana Quintero Montoya
Universidad Eafit
Medellín, Colombia
mquintero3@eafit.edu.co

3) Practice for final project defense presentation

3.1

3.2

Length	Time
1	0
2	1
3	0
4	0
5	0
6	0
7	1
8	2
9	2
10	5
11	16
12	47
13	49
14	123
15	516
16	1492
17	7778
18	7832
19	7921
20	12879
21	172108
22	869915



The graph of the data is exponential, in this one it is possible to be appreciated with greater clarity that from the 15 or 20 data the time passed of execution increases a remarkable amount, nevertheless for the first 7 data they are made in a time smaller than 1 millisecond

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

3.3 Algorithm's 1 complexity is exponential; this implies that as the number of characters increases, the time spent to give a result will also increase. By this means, is not viable to use it for.

3.4 Given an array of ***ints***, this algorithm tells us if it is possible to choose a group of those ***ints*** so that when we add them, we obtain the given objective, however, there are two restrictions: all multiples of five within the array must be included in the sum and if the value of the number following a multiple of five is one then it should not be chosen.

Then, the algorithm receives as parameters: an array of ***ints*** called ***nums***, an ***int*** called ***start*** and an ***int*** called ***target*** that will be the one that represents the objective to be reached. The process to be followed by the algorithm is as follows: First, It asks if **start** is a number greater than or equal to the length of the array ***nums*** or if that length is equal to zero, in case either of the two conditions is true, the algorithm is responsible for initializing ***target*** as zero, once this is done, it asks if the number found in the ***start*** position within the ***nums*** array is a multiple of 5 (this when executing the modulo operation) and once it knows if this is true, proceed to ask a second question that consists of whether the number following ***start*** is less than the length of the array and also whether the number in that position within the ***nums*** array is equal to one, in case both conditions are met, returns a recursive call of the function with ***start*** plus two, ***nums*** and ***target*** minus the number at the ***start*** position of the array.

In case both last conditions are not met, but it remains true that the number in the ***start*** position is a multiple of five, then a recursive call of the function is returned but with ***start*** plus one, ***nums*** and ***target*** minus the number in the ***start*** position, previously it was ***start*** plus two because with that instruction the algorithm understands that it cannot choose the number one (the one that meets both conditions) to do the addition, now only one is added because the algorithm understands that the number next to the multiple of five is different from one. Finally, the algorithm understands that in case none of the previous conditions are met, it must simply return a disjunction between a recursive call of the function with ***start*** plus one, ***nums*** and ***target*** minus the number in the ***start*** position or the recursive call with ***start*** plus one, ***nums*** and ***target***.

3.5

fibonacci

$T(n) = -c_2 + c_1 F_n + c_2 L_n$ (where c_1 and c_2 are arbitrary parameters)

F_n is the n^{th} Fibonacci number

bunnyEars

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

$T(n) = c_2\,n + c_1$ (where c_1 is an arbitrary parameter)

factorial

$T(n) = c_2\,n + c_1$ (where c_1 is an arbitrary parameter)

triangle

$T(n) = c_1\,n + c_1 + \frac{1}{2}\,n\,(n+1)$ (where c_1 is an arbitrary parameter)

powerN

$T(n) = c_1 - c_2\,n$ (where c_1 is an arbitrary parameter)

groupNoAdj

$T(n) = -c_2 + c_1\left(\frac{1}{2}\left(-1-\sqrt{5}\right)\right)^n + c_2\left(\frac{1}{2}\left(\sqrt{5}-1\right)\right)^n$

(where c_1 and c_2 are arbitrary parameters)

groupSum6

$T(n) = 2^{-n}\,(c_2 + 2\,c_1) - c_2$ (where c_1 is an arbitrary parameter)

splitArray

$T(n) = 2^{-n}\,(c_2 + 2\,c_1) - c_2$ (where c_1 is an arbitrary parameter)

split53

$T(n) = 2^{-n}\,(c_2 + 2\,c_1) - c_2$ (where c_1 is an arbitrary parameter)

groupSum5

$T(n) = 2^{-n}\,(c_2 + 2\,c_1) - c_2$ (where c_1 is an arbitrary parameter)

3.6 The variables n and m can be understood as those that represent the extension, length or size of the program that is performed, I mean, on them depends if the complexity of the algorithm that is being implemented increases or decreases.

4) Practice for midterms

4.1

1) A.
2) C.
3) A.

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

4.2

1) A.
2) A y C.

4.3

1) B.

4.4 return lucas(n - 1) + lucas(n - 2);

1) C.

4.5

1) A.
2) B.

4.6

1) return sumaAux(n, i + 2);
2) return (n.charAt(i) - '0') + sumaAux(n, i + 1);

OPTIONAL:

4.6 (The first one)

1) A

4.7 (The first one)

1) E

4.8 (The second one)

1) C.

4.9

1) C.

5) Recommended reading (optional)

Narasimha Karumanchi, Data Structures and Algorithms made
Chapter 3: Recursion and Backtracking
Summarize:
What is Recursion?

Function which calls itself is called recursive. A recursive method solves a problem by calling a copy of itself to work on a smaller problem. The recursion step can result in

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

many more such recursive calls.

Why Recursion?

Recursive code is generally shorter and easier to write than iterative code. Generally, loops are turned into recursive functions when they are compiled or interpreted.

Recursion and Memory

Each recursive call makes a new copy of that method in memory. Once a method ends , the copy of that returning method is removed from memory. The recursive solutions look simple but visualization and tracing takes time. A recursive approach makes it simpler to solve a problem that may not have the most obvious of answers.

But, recursion adds overhead for each recursive call .

Recursion

If we get infinite recursion, the program may run out of memory and result in stack overflow.

Iteration

Each iteration does not require extra space. An infinite loop could loop forever since there is no extra memory being created.

What is Backtracking?

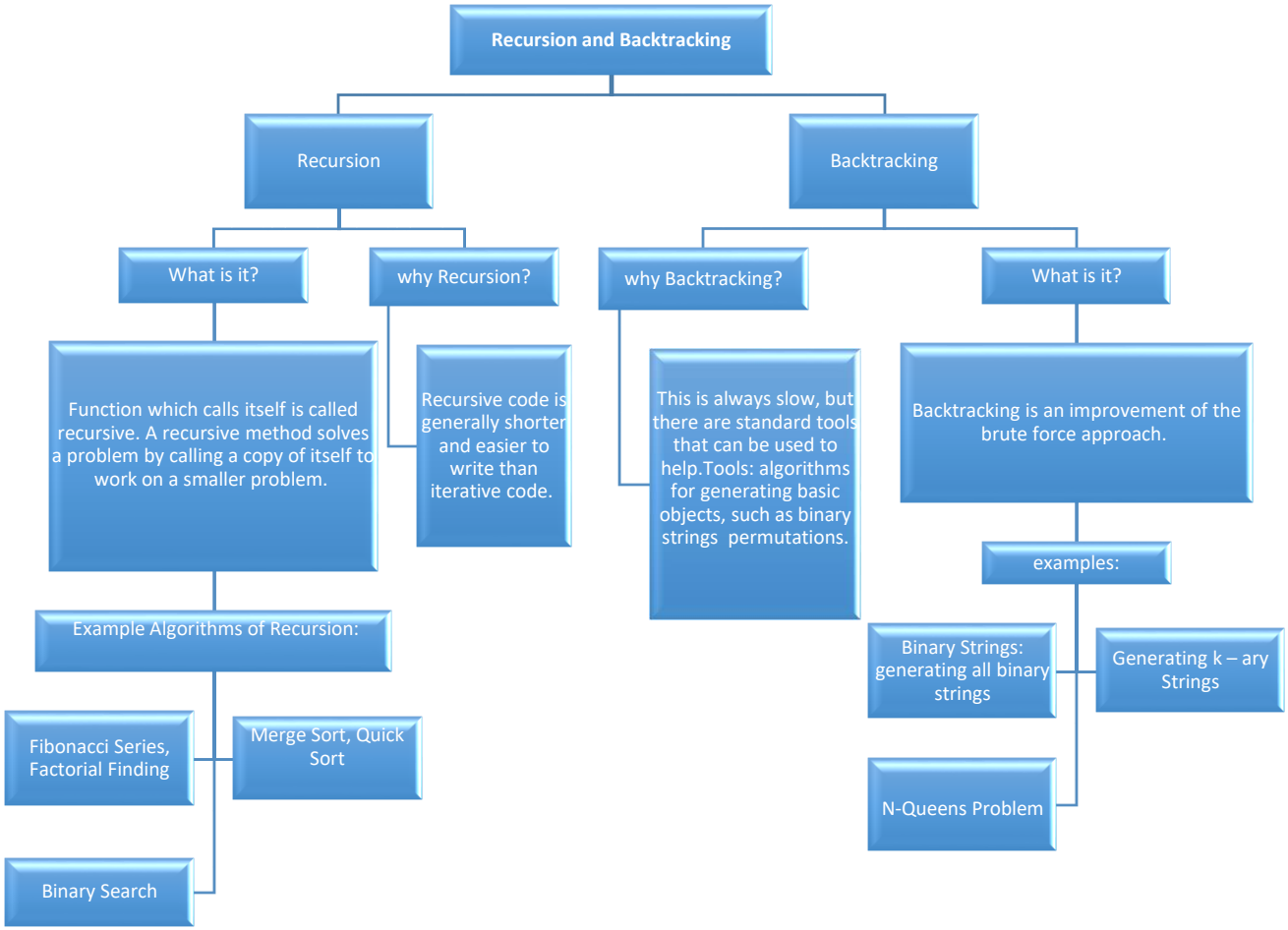
Backtracking is an improvement of the brute force approach. It systematically searches for a solution to a problem among all available options. In backtracking, we start with one possible option out of many available options and try to solve the problem if we are able to solve the problem with the selected move then we will print the solution else we will backtrack and select some other option and try to solve it. If none if the options work out we will claim that there is no solution for the problem.

Backtracking is a form of recursion. Backtracking can be thought of as a selective tree/graph traversal method. Backtracking allows us to deal with situations in which a raw brute-force approach would explode into an impossible number of options to consider. Backtracking is a sort of refined brute force.

What is interesting about backtracking is that we back up only as far as needed to reach a previous decision point with an as-yet-unexplored alternative. If we backtrack all the

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

way to our initial state and have explored all alternatives from there, we can conclude the problem is unsolvable.



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