2020/2021

Advanced Algorithms

Algorithm Development Strategies

The String to String Correction Problem

Diogo Andrade 89265

7 December 2020

Index

Problem

The string-to-string correction problem is to find a minimal sequence of edit operations (insert, remove, replace) for changing a given string into another string, that is given two strings, determine your distance using the WagnerFischer Algorithm.

To resolve this problem, I developed two algorithms, one recursive and another dynamic.

Execution example

Usage:

main.py -s1 STR1 -s2 STR2

Execution:

python3 main.py -s1 saturday -s2 sunday

Result:

str1: saturday str2: sunday

func	distance	operations	execution time
recursive	3	3691	0.0029907227
memoized	3	3691	0.0040204525
dynamic	3	48	0.0000000000
improved	3	48	0.0000000000

Usage:

test.py [-m] [-d] -r REPETITIONS -s MAX_STR_SIZE

Execution:

python3 test.py -s 10 -r 15

Default execution uses the recursive algorithm. If used the -m flag will use the recursive memoization algorithm and if used the -d flag will use the dynamic algorithm.

This program runs multiple times the same algorithm, allowing us to compare the results and generate graphics.

Recursive Algorithm

Complexity

n = string 1 size

m = string 2 size

$O(n^m) \rightarrow$ exponential, as we can see in the image below.

The worst case happens when none of the characters of the two strings match.

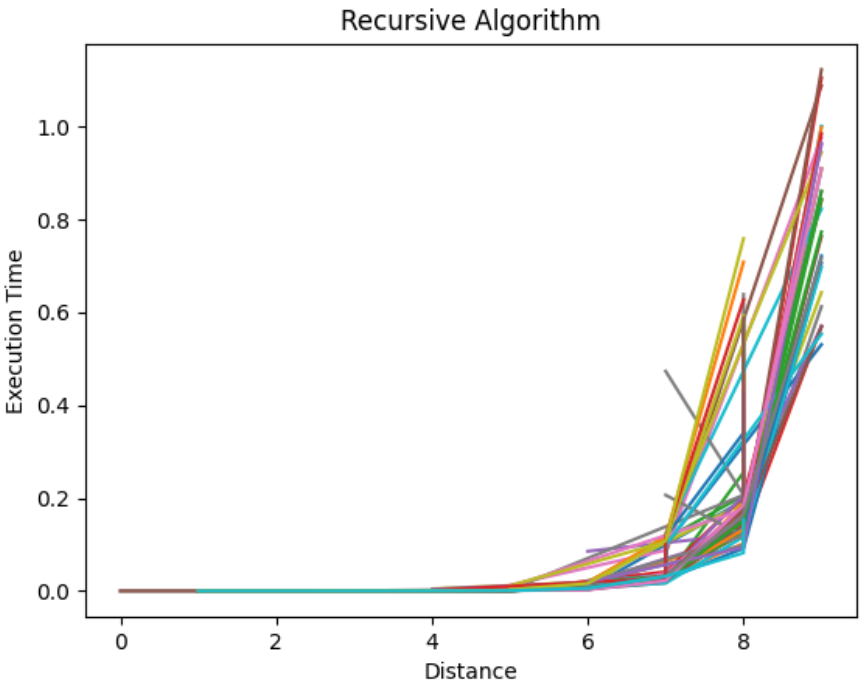

Figure 1: Complexity of Recursive Algorithm

Execution Test

I ran this algorithm 15 times up to a distance of 10, from a distance of 11 this algorithm start to be slow, more than 1 second.

The execution time grows linearly with the number of basic operations.

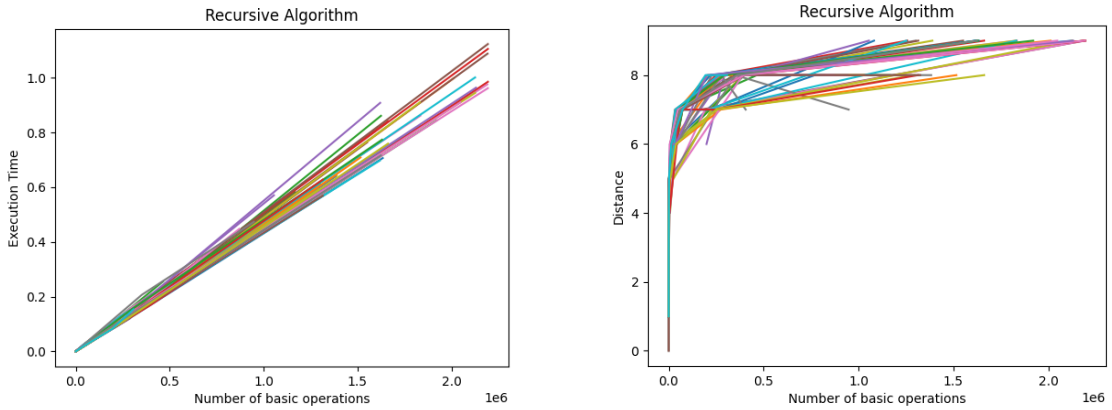

Figure 2: Recursive Algorithm execution test

Recursive Memoization Algorithm

This algorithm uses the same recursive algorithm, so the complexity will be the same. I couldn't see the execution time difference between this algorithm and the simple recursive algorithm, but from what I was able to observe the bigger the size of the strings the bigger the difference in execution time between the two algorithms.

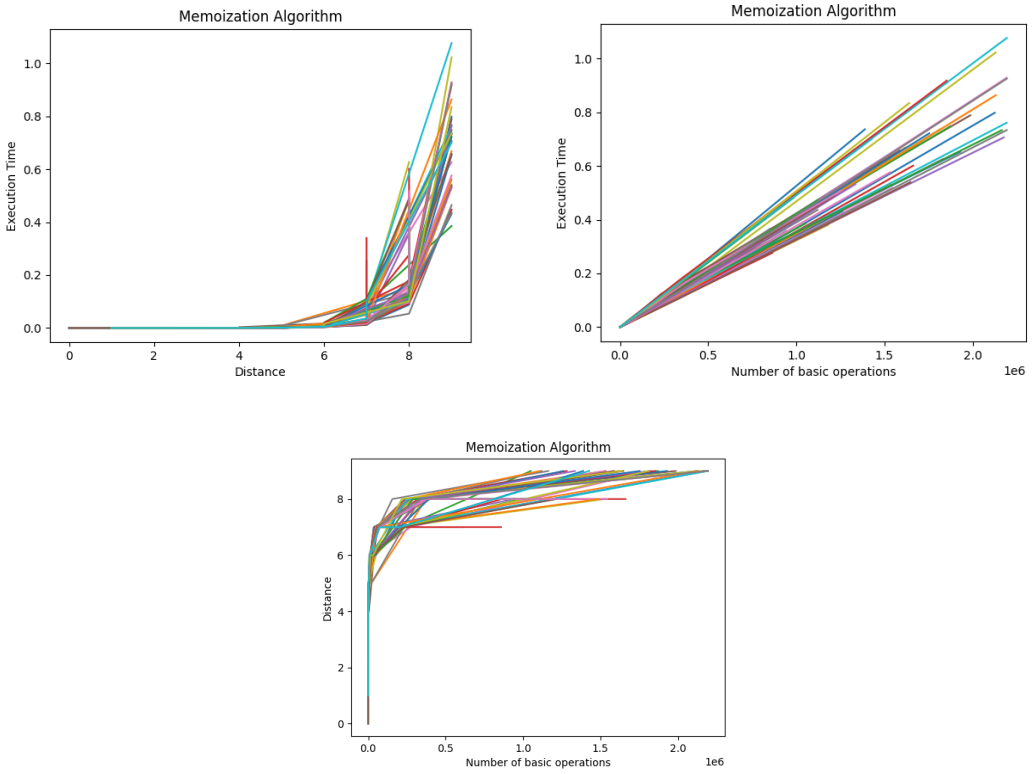

Figure 3: Recursive Memoized Algorithm execution test

Dynamic Algorithm

Complexity

n = string 1 size

m = string 2 size

$O(n * m) \rightarrow$ linear, as we can see in the image below.

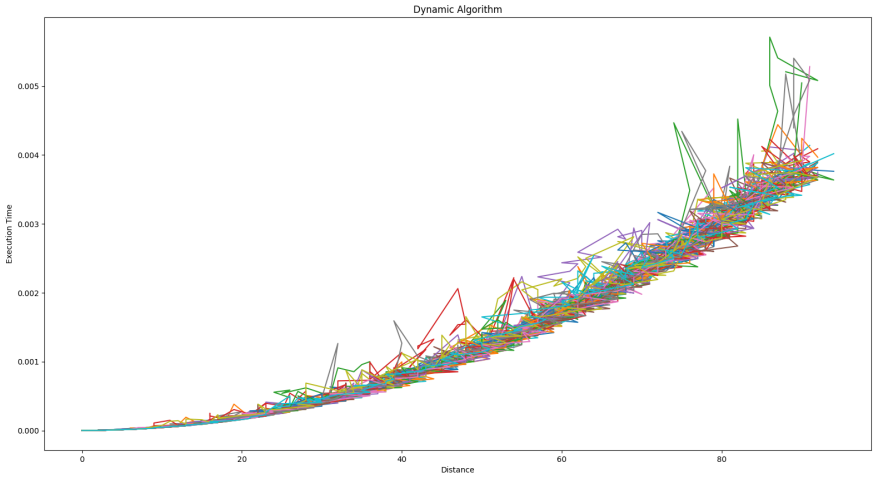

Figure 4: Complexity of Dynamic Algorithm

Execution Test

I ran this algorithm 50 times up to a distance of 100.

The execution time grows linearly with the number of basic operations.

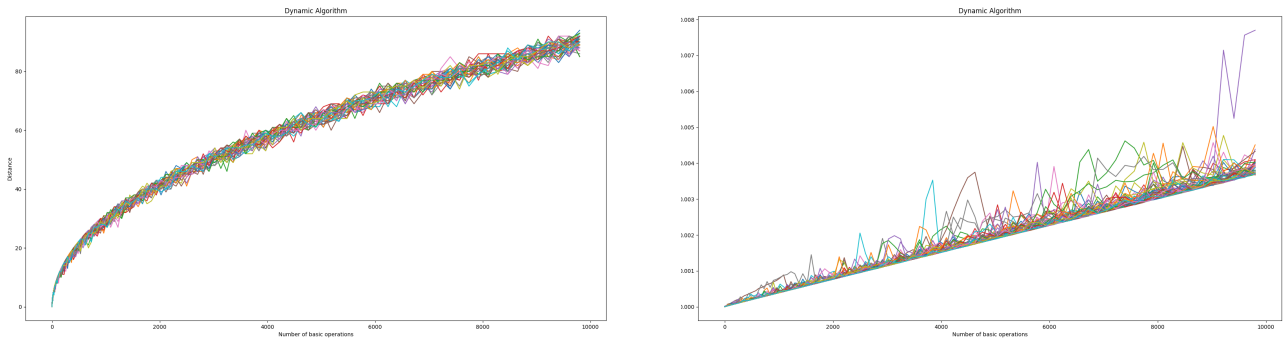

Figure 5: Dynamic Algorithm execution test

Conclusion

In conclusion, the recursive algorithm has the biggest complexity. However for small and similar strings the recursive algorithm can be more efficient then the dynamic algorithm.

For big strings the execution time of a recursive algorithm is too big, so not allow us to use that algorithm, even the recursive memoized algorithm has better execution times, but not enough, so in those cases it is better to use the dynamic algorithm.

The dynamic algorithm uses a list of lists, that list size depends on string sizes, so the program will have a limit of memory we can use, so I created an improved function which uses only a list with two lists. That is possible because I only need to access the previous list.

Comparing the algorithms, obviously the dynamic algorithm is the fastest algorithm, but uses more memory and the recursive memoized algorithm is faster than a simple recursive algorithm.