Advanced Algorithms Project Report

Rodrigo Bernardo
ist178942

Instituto Superior Técnico

May 19, 2017

1 Text Searching

1.1 Overview

In this section we discuss the implementation and profiling of the naive, Knuth-Morris-Pratt and Boyer-Moore algorithms for string matching. The implementation for the Knuth-Morris-Pratt algorithm was taken from the third edition of the book *Introduction to Algorithms*, while the implementation for the Boyer-Moore algorithm was taken from *Algorithms on Strings, Trees, and Sequences.*

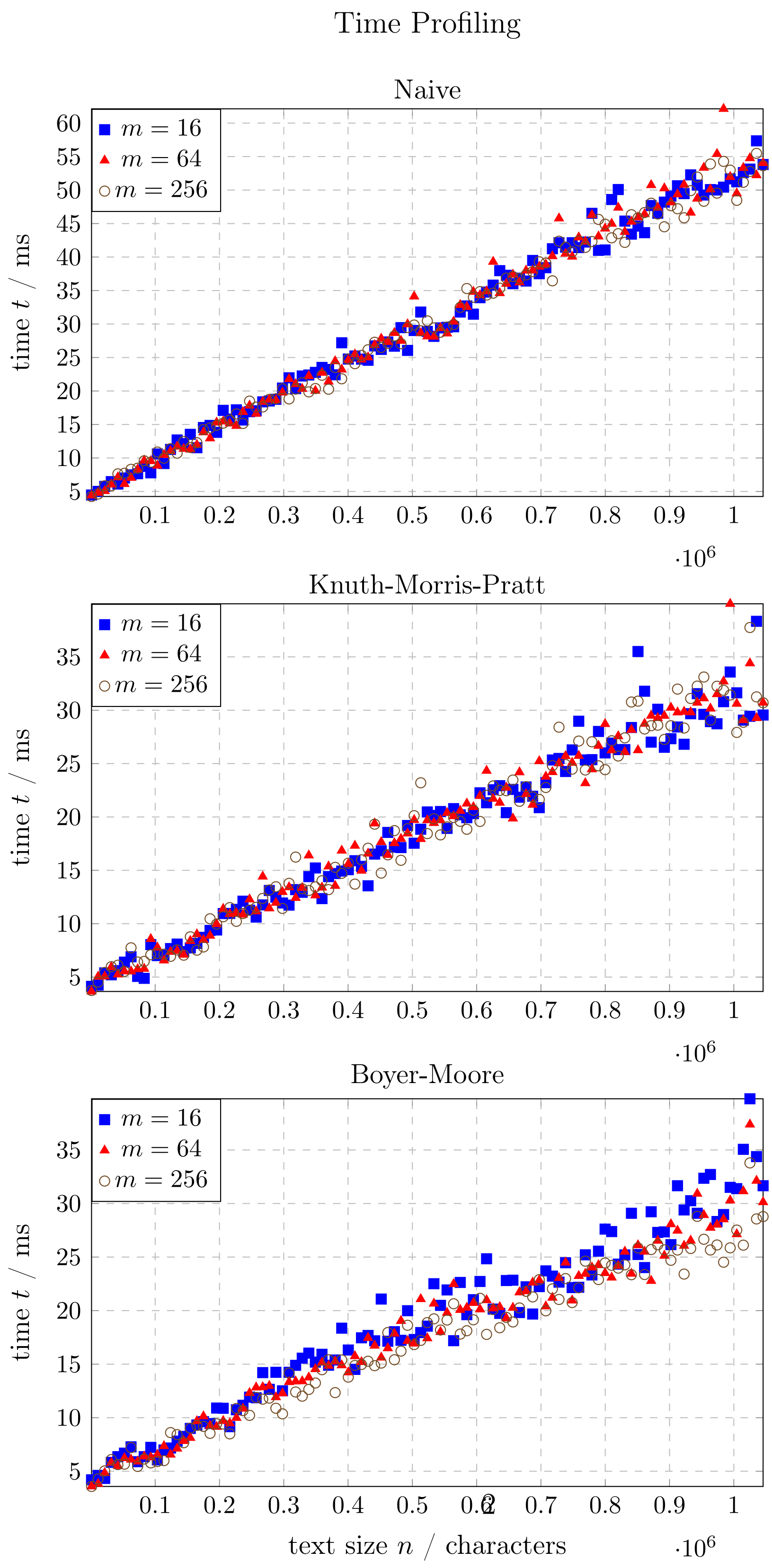
1.2 Tests and Expectations

In this project the naive algorithm serves as a baseline for testing the other two algorithms, which we expect to be much faster in all situations.

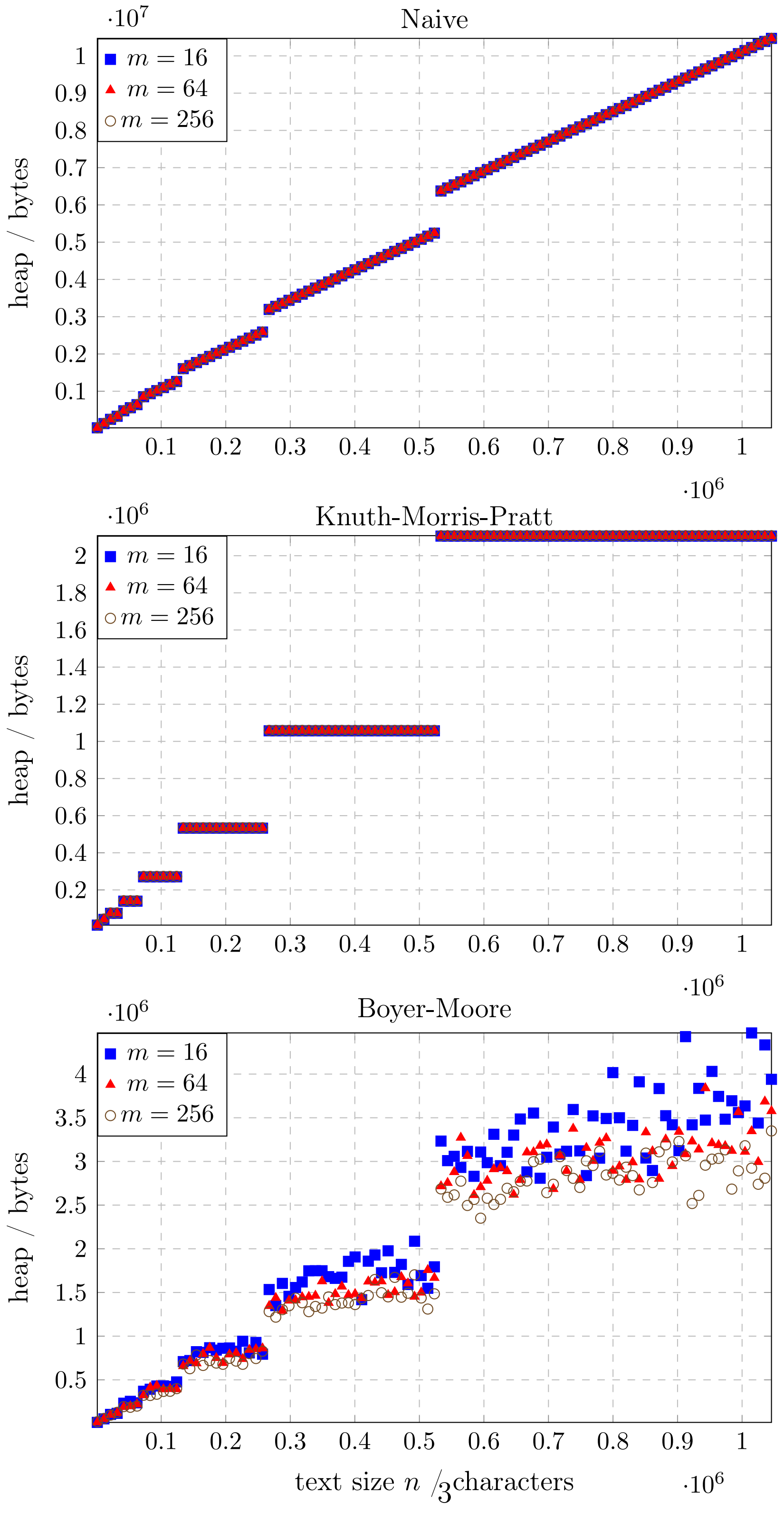
We profile time and heap usage, as well as the number of comparisons each algorithm performs. We test the algorithms on randomly generated text and pattern strings consisting of 'A', 'T', 'C' and 'G' characters only.

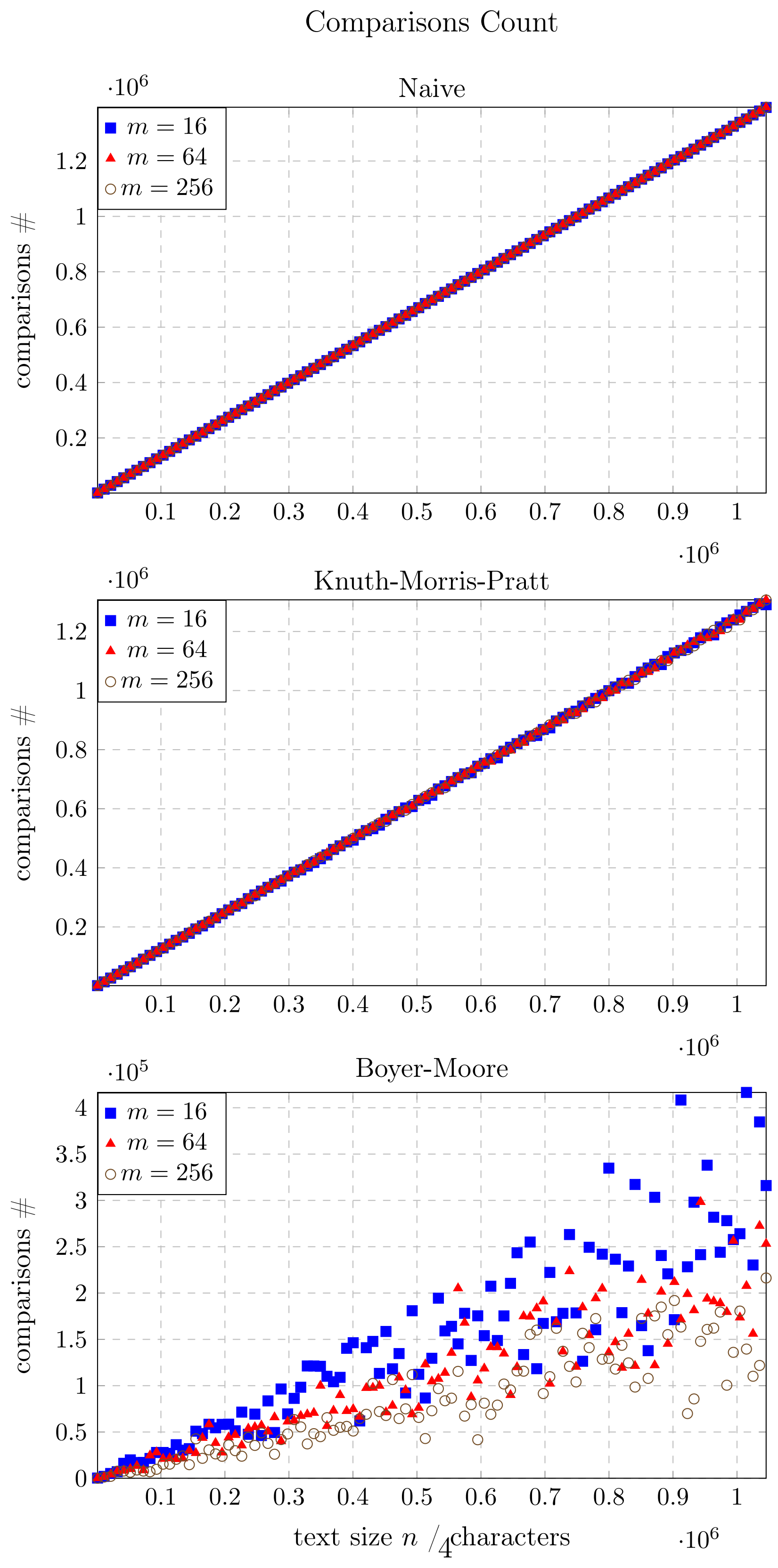
We expect the Knuth-Morris-Pratt and Boyer-Moore to run on $O(n + m)$ both on time and space, where n is the length of the text and m is the length of pattern. The complexity of the naive algorithm is also $O(n + m)$ in the average case, but it decays to $O(nm)$ in the worst case.

1.3 Results

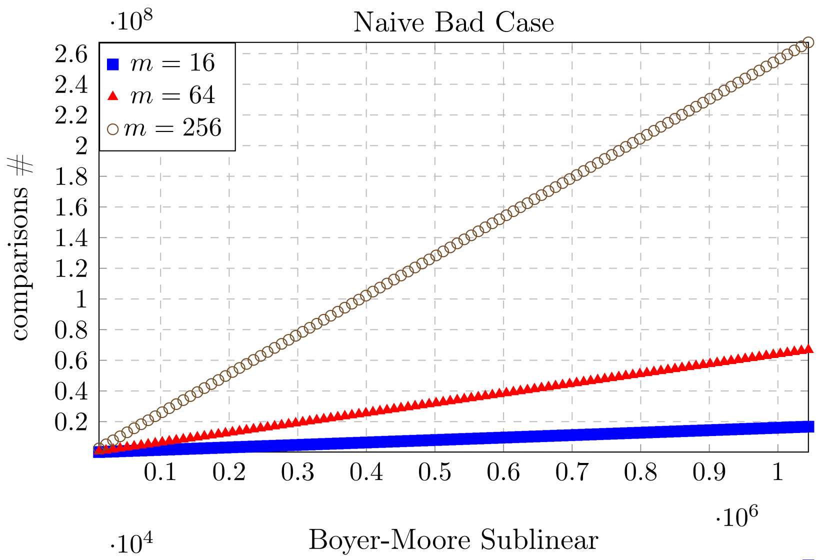

Time Profiling
Naive
Knuth-Morris-Pratt
Boyer-Moore
$m = 16$
$m = 64$
$m = 256$
time t / ms
text size n / characters
$\cdot 10^6$

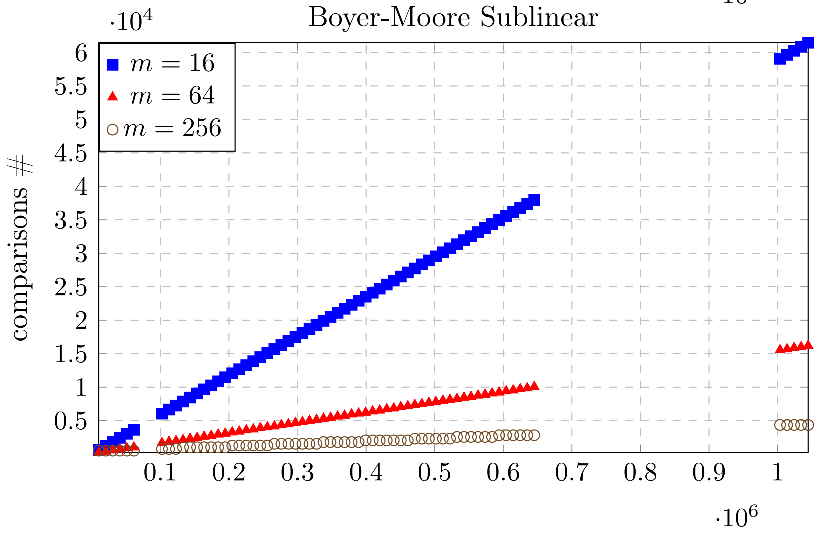
Heap Profiling



Comparisons Count
Naive
Knuth-Morris-Pratt
Boyer-Moore
$m = 16$
$m = 64$
$m = 256$
comparisons #
$\cdot 10^6$
$\cdot 10^5$
0.2 0.4 0.6 0.8 1 1.2
0 0.5 1 1.5 2 2.5 3 3.5 4
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
text size n / characters

4





1.4 Results Analysis and Conclusions

1.4.1 Time and Heap Profiling

We can check that all the algorithms run in linear time and space, and that Knuth-Morris-Pratt and Boyer-Moore have much better performance than naive. The naive and Knuth-Morris-Pratt running times don't seem to be much affected by the increase in size of the pattern. Boyer-Moore actually runs faster and in less space with bigger patterns.

1.4.2 Answers to the Questions in the Project Specification

The naive algorithm worst-case happens when the text has many similar substrings to the pattern. In the penultimate plot we can see the result of an experience were we applied the naive algorithm to texts and patterns with these caracteristics. We see, as expected, that the algorithm makes many more comparisons the bigger the pattern.

In the last plot we find the answer to why the Boyer-Moore is claimed to be sublinear. We applied this algorithm against texts and patterns where the matching process failed quickly. By use of the bad character, the algorithm manages to skip a lot of characters, thus managing to perform fewer comparisons. By abusing this rule the algorithm may reach the end of the text without making $O(n + m)$ comparisons, as we can see in the plot.

2 Connectivity in Forests

2.1 Overview

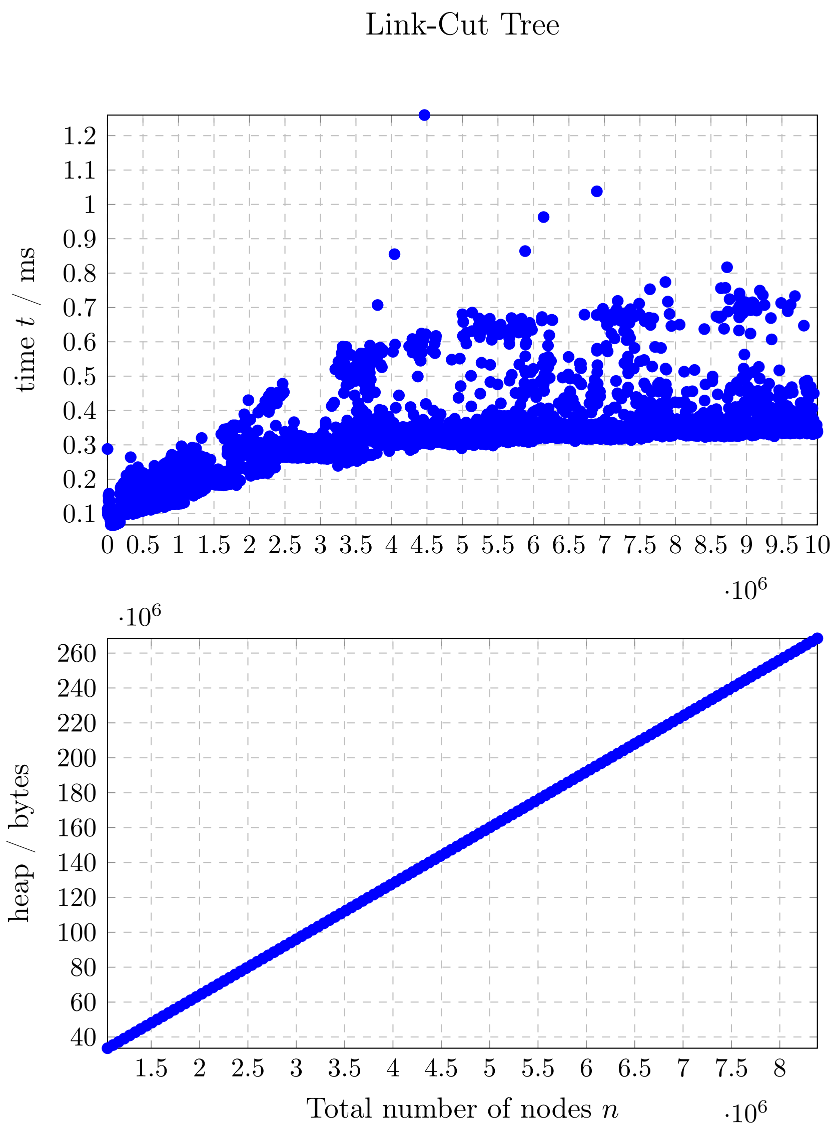
In this problem we aim to maintain the connectivity information of a forest. For that we use an implementation of Link-Cut trees with Splay Trees, based on the paper *Self-adjusting binary trees*, by Sleator and Tarjan.

2.2 Tests and Expectations

We test the implementation time and space performance. For each n (the number of nodes), we start by building a linear tree. A cut then follows, cutting the tree in two halves. Then we apply 2000 random operations (a link, a cut or a connected query), and measure the time those operations take.

We're expecting linear increases in space usage with the increase of the number of nodes, as the algorithm only does one explicit memory allocation per node, and doesn't use recursion. Also, we're expecting the tree's operations to run all in $log(n)$ amortized time.

2.3 Results



2.4 Results Analysis

The time plot shows a curve that resembles a logarithm. While we cannot assert that each operation runs, by itself, in $log(n)$, we can assert that in an amortized sense.

The space usage is completely linear on the number of nodes, as expected.