

MSc in Computer Science and Engineering

**Advanced Algorithms 2016-2017**

Text Searching:

In this project students had to implement and compare several online algorithms for matching

small patterns against a larger text. This problem is recurrent in computer science specially in bioinformatics applications, therefore we will use DNA sequences to explore this problem.

1. INTRODUCTION:

The string matching problem consists in finding one, several or all occurrences of a pattern P in a larger text T.

There are several solutions that solve this problem and in this project we explore 3 of them. The first one is the *Naive algorithm,* this is the simplest one and consists in trying to match P with every substring\* of T with the same size as P. As the name indicates this is a naïve approach and in 1977 Donald Knuth, Vaughan Pratt and James H. Morris published a better algorithm called *Knuth-Morris-Pratt algorithm* which we will examine in second. The third algorithm we are going to examine is *Boyer-Moore algorithm* developed by Robert S. Boyer and J. Strother Moore also in 1977. Bothe *Knuth-Morris-Pratt* and *Boyer-Moore* use preprocesses of the pattern to obtain a better performance than what we had with the naïve approach.

In this project we present the three algorithms mentioned before and also a data structure for reading string dynamically. The final goal is to be able to say which one is better and in what situations for the DNA sequence problem. To achieve this, we implement, test and compare. The comparison will be executed with randomly generated strings T, highly repetitive strings T and the pattern might appear or not. To measure the effectiveness of the algorithm we are going to count the number of comparisons and the execution time. In section 2 we introduce the problem and introduce some nomenclature. In section 3 we devote a section to our structure to read string dynamically and we devote a section to each algorithm. Finally, we compare the three algorithms in Section 4 and conclude in Section 5.

1. BACKGROUND:

We formalize the string matching problem as follow. We assume that a text T is an array T [0..n] of length n and a pattern P is and array P [0..m] of length m where n ≥ m. We also assume that every element presented in P and T belongs to a finite alphabet . For example, in the DNA sequence = {A, T, C, G}.

In figure 1 have an alphabet = {a, b, c} and as we can see the pattern P occurs in T starting in position T [3], we say that P occur in T with a shift 3. If P appears in T with shift s we say that s is a valid shift, otherwise we call s an invalid shift. Our goal is to find every valid shift in T for a pattern P.

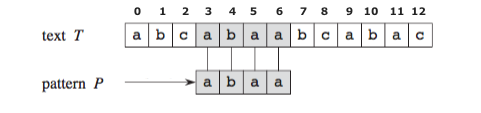
****

Figure 1 - example of a string matching problem.

Also the concept of suffix and prefix are important in this context. We say that a string x with size m is a prefix of another string y with size n and n ≥ m, if y starts with x. In the example of figure 1 the string ab is a prefix of P. A suffix is the opposite. We say that a string x with size m is a suffix of another string y with size n and n ≥ m, if y ends with x. In the example of figure 1 the string aa is a suffix o P.

1. DATA STRUCTURE AND ALGORITHMS:
   1. Dynamic Array:
   2. Naive Approach:

The solution provided by the naive approach consists in starting at the shift s = 0 check, character by character, if s is a valid shift. For each shift s 1 of 3 things can happen:

1. We find a character in P that differ from a character in the substring T [s ... s + m], which means that shift s is invalid and we move to the next shift = s+1.
2. We reach the end of P, then we have found a match between T [s … s + m] and P. We print s and move to the next.
3. Shift s reach the value n-m+1, which means s is not a valid shift anymore and the algorithm terminates.

FIGURA PARA OS CASOS??

The problem with this approach is the complexity. The naive approach is O (N x M) in worst case.

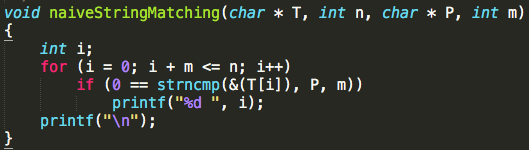


Figure 2 - Our implementation of the Naive Algorithm in C