

Kevin Gao

# Notes on String Algorithms

With Applications in Computational Biology

Copyright © 2022 Kevin Gao

TERRA-INCOGNITA.DEV

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

# Contents

## Part I Exact Matching

|   |                                     |    |
|---|-------------------------------------|----|
| 1 | <i>Naive Exact Matching</i>         | 9  |
| 2 | <i>Z Algorithm</i>                  | 11 |
| 3 | <i>Boyer-Moore</i>                  | 13 |
| 4 | <i>Knuth-Morris-Pratt Algorithm</i> | 15 |
|   | <i>Index</i>                        | 17 |



*These notes are primarily based on the book **Algorithms on Strings, Trees, and Sequences** by Dan Gusfield.*

*Thanks Professor Gusfield for his great book.*



## **Part I**

# **Exact Matching**





# 1

## Naive Exact Matching

**TERMINOLOGY CONFUSION.** Before starting the discussion of string matching algorithms, we should note the difference between a *substring* and a *subsequence*. Given a string  $S$ , characters in a substring of  $S$  must occur *contiguously* in  $S$ ; whereas characters in a subsequence may be interspersed gaps (or indels, as we call them in biology) and/or characters not in the original string.

### Exact String Matching Problem

Given a text string  $T$  and pattern  $P$ , the goal of the exact string matching problem is to find all occurrences of  $P$  in  $T$ .

### The “Naïve” Algorithm

NAIVE-MATCH( $P, T$ )

```
1  matches = []
2  for i = 1 to |T| - |P| + 1
3      match = TRUE
4      for j = 1 to |P|
5          if T[i + j] ≠ P[j]
6              match = FALSE
7              break
8      if match
9          matches.APPEND(i)
10 return matches
```

The naïve exact matching algorithm aligns the left end of  $P$  with the left end of  $T$  and compares the characters of  $P$  and  $T$  left to right

$T$  : xabxyabxyabxz  
 $P$  : abxyabxz  
    abxyabxz  
    abxyabxz  
    abxyabxz  
    abxyabxz  
    abxyabxz

Figure 1.1: Naive exact matching with  $P = abxyabxa$  and  $T = xabxyabxyabxz$ .

until a mismatch is found, or until it reaches the end of  $P$ , in which case we report the position of  $P$ . Then,  $P$  is shifted to the right by one place. We repeat this procedure until the right end of  $P$  passes the right end of  $T$ .

### *Runtime of the Naive Algorithm*

Let  $n = |P|$  and  $m = |T|$ . The worst-case comparisons made by the naive algorithm is  $\Theta(nm)$ . We have the lower bound  $\Omega(nm)$  when  $P$  and  $T$  contains the same repeated characters (e.g.  $P = aaa$ ,  $T = aaaaaaaaa$ ), in which case the algorithm makes  $n(m - n + 1) \in \Omega(nm)$  comparisons.

## 2

# Z Algorithm

### Speeding Up the Naive Algorithm

The naive exact matching algorithm is stupid. It always shifts  $P$  by only one even if it knows for sure that the next shift will not yield a match. This gives us some ideas on how to improve the algorithm. If we can shift  $P$  by more than one character, but never shift so far as to miss the next occurrence of  $P$  in  $T$ , we can improve the runtime of the naive algorithm.

Doing this, however, requires us to have some prior knowledge of the pattern  $P$  or the text  $T$ .

### Fundamental Preprocessing

A fundamental preprocessing is a generalized way to process the pattern  $P$  to gain knowledge of the pattern, independent of any particular algorithm.

**Definition 2.1.** Given a string  $S$  and a position  $i > 1$ , let  $Z_i(S)$  be the length of the longest substring of  $S$  that starts at  $i$  and matches a prefix of  $S$ .

In other words,  $Z_i(S)$  is the length of the longest prefix of  $S[i \dots |S|]$  that matches a prefix of  $S$ .

**Definition 2.2 (Z-box).** For any position  $i > 1$  where  $Z_i$  is greater than 0, the **Z-box** at  $i$  is the interval starting at  $i$  and ending at  $i + Z_i - 1$ .

**Definition 2.3.** For every  $i > 1$ ,  $r_i$  is the right endpoint of the Z-box that begins at or before position  $i$  (i.e. the closest Z-box to the left). More formally,  $r_i$  is the largest value of  $j + Z_j - 1$  over all  $1 < j \leq i$  such that  $Z_j > 0$ .



Figure 2.1: Relations between  $i$ ,  $l_i$ ,  $r_i$  and the Z-box at  $l_i$ .



3

*Boyer-Moore*



4

*Knuth-Morris-Pratt Algorithm*





# *Index*

Z-box, [11](#)