

FM-Index: Error free search & backtracking for errors

Group 7: Abhinav Mishra, Jule Brenningmeyer, Maike Herkenrath

Department of Mathematics & Informatics, FU Berlin

Introduction

- For answering various biological questions it is necessary to find a specific pattern in a given text such as sequenced DNA or RNA. The FM-Index can be used to efficiently find patterns with a space complexity of $O(\log \sigma \cdot n) + o(\log \sigma \cdot \sigma \cdot n)$.
- It consists of the Burrows-Wheeler transform of the text and the additional data structures Occ , C and L .
- The FM-Index can be used to find exact matches of a pattern in the text as well as approximate string matches that include a defined number of errors.

FM-Index Data Structures

Let text $T = \text{mississippi\$}$ with length $n = 12$ and alphabet of size $\sigma = 5$.

- To obtain the Burrows-Wheeler transform L all cyclic shifts of the text T are sorted in lexicographic order to form the matrix M .
- The last column of M corresponds to the data structure L , which has a space complexity of $O(n)$, where n is the length of the text.

	F		L
mississippi\$	\$	mississippi	i
ississippi\$m	i	\$mississippi	p
ssissippi\$mi	i	ppi\$missis	s
sissippi\$mis	i	ssippi\$mis	s
issippi\$miss	i	ssissippi\$	m
ssippi\$missi	m	ississippi	\$
sippi\$missis	p	i\$mississi	p
ippi\$mississ	p	pi\$mississ	i
ppi\$mississi	s	ippi\$missi	s
pi\$mississip	s	issippi\$mi	s
i\$mississipp	s	sippi\$miss	i
\$mississippi	s	sissippi\$m	i

Table 1: Burrows-Wheeler transform

- The array C indicates for each character in the alphabet how often a lexicographically smaller character occurs in the text T . This is the index of the row in the matrix M above the first string starting with c . The array C has a space complexity of $O(\sigma \cdot \log(n))$.
- The number of occurrences of a character c in L until the index i is saved in the matrix Occ , with a space complexity of $O(\sigma \cdot n)$.

$\alpha \in \Sigma$	\$	i	m	p	s
n_α	1	4	1	2	4
$C(\alpha)$	0	1	5	6	8

i	1	2	3	4	5	6	7	8	9	10	11	12
$L[i]$	i	p	s	s	m	\$	p	i	s	s	i	i
$Occ(L[i], i)$	1	1	1	2	1	1	2	2	3	4	3	4

Table 2: Array C , Matrix Occ

Reverse Transform

- The original text can be reconstructed from L , C and Occ by applying L-to-F-mapping. The goal of L-to-F-mapping is to find the previous letter in L . Thereby, the data structures C and Occ are used.
- Formula for the L-to-F mapping:

$$LF(i) = C(L[i]) + Occ(L[i], i)$$

- Important to notice is that the order of the same alphabet-elements in L and F are the same. This is called rank preservation.

Backward Search

- To find an exact match of a pattern in the text using the FM-Index, a backward search is performed. The backward search returns an interval whose length is equal to the number of occurrences of the pattern.
- The backward search returns the positions of the occurrences of the pattern in F . To obtain the positions of the occurrences in the text, the matches are located in the text by applying L to F-mapping till an element is reached which location in the text is known.

F	L	F	L	F	L	F	L
\$ mississippi	i	\$ mississippi	i	\$ mississippi	i	\$ mississippi	i
i \$mississip	p	i \$mississip	p	i \$mississip	p	i \$mississip	p
i ppi\$missis	s	i ppi\$missis	s	i ppi\$missis	s	i ppi\$missis	s
i ssippi\$mis	s	i ssippi\$mis	s	i ssippi\$mis	s	i ssippi\$mis	s
i ssissippi\$	m	i ssissippi\$	m	i ssissippi\$	m	i ssissippi\$	m
m ississippi	\$	m ississippi	\$	m ississippi	\$	m ississippi	\$
p i\$mississi	p	p i\$mississi	p	p i\$mississi	p	p i\$mississi	p
p pi\$mississ	i	p pi\$mississ	i	p pi\$mississ	i	p pi\$mississ	i
s ippi\$missi	s	s ippi\$missi	s	s ippi\$missi	s	s ippi\$missi	s
s issippi\$mi	s	s issippi\$mi	s	s issippi\$mi	s	s issippi\$mi	s
s sippi\$miss	i	s sippi\$miss	i	s sippi\$miss	i	s sippi\$miss	i
s sissippi\$m	i	s sissippi\$m	i	s sissippi\$m	i	s sissippi\$m	i

Table 3: Backward search with zero errors for pattern "si".

Backtracking

Algorithm 7 Simple Backtracking

```

1: procedure BACKTRACKING(iterator,  $S, e$ )
2:   if  $S = \epsilon$  then                                     ▷ leaf reached
3:     report iterator
4:   else if  $e = 0$  then                                     ▷ no errors left
5:     if iterator'  $\leftarrow$  forward_search(iterator,  $S$ ) then
6:       report iterator'
7:   else                                                     ▷ errors left
8:     for  $c \in \Sigma$  do
9:       if iterator'  $\leftarrow$  forward_search(iterator,  $c$ ) then
10:         $e' \leftarrow e - (c \neq S[0])$ 
11:        BACKTRACKING(iterator',  $S[1..], e'$ )
12:
13: procedure BACKTRACKING( $S, e$ )
14:   iterator  $\leftarrow$  root                                     ▷ iterator stores suffix array ranges
15:   BACKTRACKING(iterator,  $S, e$ )

```

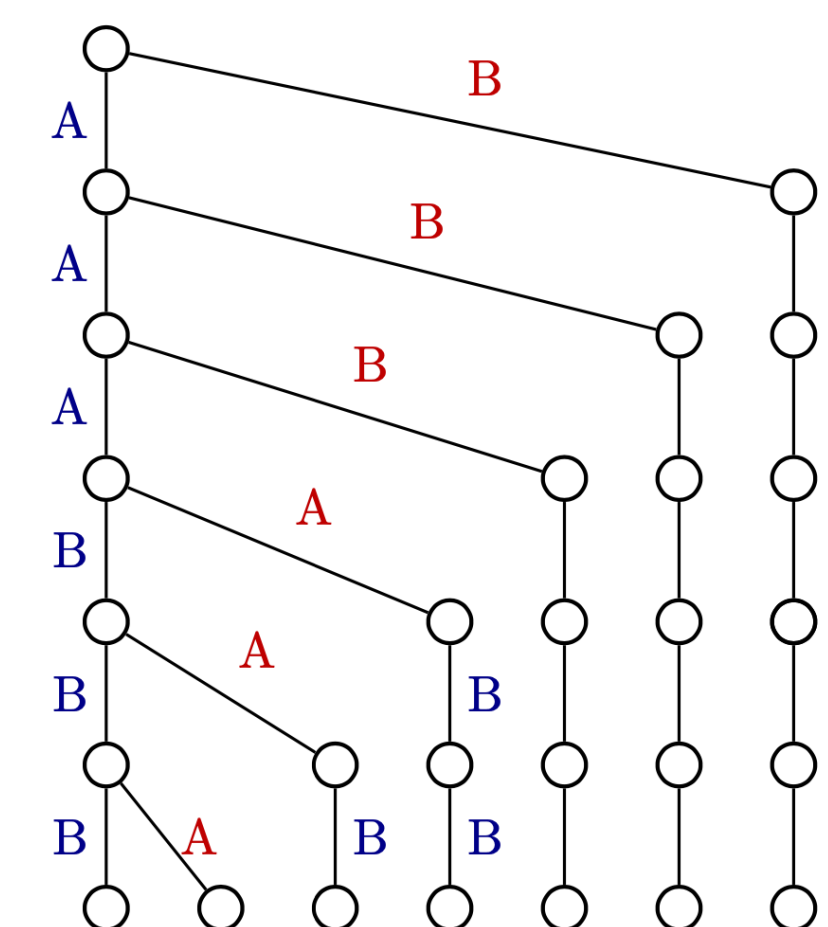


Figure 1: Search step: | = matches, \ = substitutions

Simple backtracking with up to e mismatches is one search with one piece allowing between 0 and e errors, whereas *pigeonhole search* with e errors can be represented by a search scheme with $e + 1$ searches and $e + 1$ pieces [1]. Original sequence S is searched in the index character by character from the root node (Figure 4).

A non-error-free search leads to σ edges going down from the current node, one for each character.

$$c_{k,e} = \begin{cases} \sigma + c_{k-1,e} + (\sigma - 1) \cdot c_{k-1,e-1}, & k > 0 \wedge e > 0 \\ k, & \text{otherwise} \end{cases}$$

where σ = size of the alphabet, e = error threshold, c = number of edges, and k = search steps [1] [2].

If sequence S is conceptually divided into p pieces, then search occurs in each of the p pieces separately with at most $\lfloor e/p \rfloor$ errors using the simple backtracking, and upto total e after successful search continues [2]. Choosing $p = e + 1$ is the pigeonhole search strategy, that requires bidirectional index over unidirectional one. [1]

References

- [1] Christopher Maximilian Pockrandt. "Approximate String Matching : Improving Data Structures and Algorithms". PhD thesis. Freie Universitaet Berlin, 2019.
- [2] Gregory Kucherov, Kamil Salikhov, and Dekel Tsur. "Approximate string matching using a bidirectional index". In: *Theoretical Computer Science* 638 (2016), pp. 145–158.
- [3] Knut Reinert. "L1,2 suffixarrays-slides". In: (2023).
- [4] Knut Reinert. "L4-7-FM_iindex". In: (2023).