RESEARCH

Introduction to Focus Areas in Bioinformatics Project Week 13

Sina Glöckner[*], Christina Kirschbaum, Swetha Rose Maliyakal Sebastian and Gokul Thothathri

[*]Correspondence:
sina.gloeckner@fu-berlin.de
Institute for Informatics, Freie Universität Berlin, Takustr. 9, Berlin, DE
Full list of author information is available at the end of the article

Abstract

Goal of the project: To benchmark the FM Index and compare the results with the naive approach and the suffix array approach.

Main results of the project: The suffix array approach got the overall best results in the benchmarking.

Personal key learnings:
We all got a deeper understanding of the FM-Index.

Estimation of the time:
Sina & Christina: 6h
Gokul & Swetha: 4h

Project evaluation: 1

Number of words: 992

1 Scientific Background

In bioinformatics, the problem of finding a query with m characters in a sequence of length n, both of which use the same alphabet, is a prevalent one [1]. Each character of the pattern is compared to a substring of the text which is the length of the pattern until there is a mismatch or a match. This results in a runtime of $O((n-m+1)m)$. [2]

Suffix trees are one method for reducing the time-consuming nature of this operation. Manber and Myers created suffix arrays in 1990 as a more efficient approach to save suffix trees, and they also detailed the first algorithms for constructing and using suffix arrays [3].

The Ferragina-Manzini index (FM-Index) is a compressed full-text substring index based on the Burrows-Wheeler transform that resembles the suffix array. [4] A FM-Index reduces the time complexity to $O(m + occ \log_2(u))$ [5]. In the field of bioinformatics, it is mostly used for the short read alignment problem [6]. That problem describes the mapping of sequencing reads on a reference sequence. It is commonly used for benchmarking exact string matching [7].

2 Methods

The naive approach and the suffix array approach we used, were explained in the project report of week 12. The FM-Index approach was used as it is implemented in the C++ library SeqAn [8, 9].

For searching the number of occurrences of a pattern P with length m in a text T with length n with the FM-Index, we first need a Burrows-Wheeler transform

(BWT). To build a BWT, a conceptual matrix M is needed first. M contains in every row one on the n cyclic shifts of T. In the next step, the rows of M are sorted lexicographically. After M is sorted, we only need to take the last column to get $T^{BWT} = L$. Through the reverse transform, we can build the first column LF, which allows us to access the first column F.

The pattern P can now be found through a backward search, which means we search the growing suffix of P from right to left. This search takes place in an interval of matches, in the end, the length of this interval equals the number of occurrences. The backward search is initialized with an interval from 1 to n. Afterwards, in every step the new borders of the interval can be computed with the following formulas, where c is the searched character, C is the first occurrence of c, and Occ is the occurrences of c in the given interval. It terminates when the first letter is reached or $a < b$ with a being the lower and b the upper border, so P does not occur in T.

$$a_{j-1} = C(c) + Occ(c, a_j - 1) + 1$$
$$b_{j-1} = C(c) + Occ(c, b_j)$$

3 Results

Similar to last week, the memory consumption and runtime for the FM-Index algorithm were measured. These results can be compared to the suffix array approach and the naive approach.

First, the influence of a change in the number of searches was examined. For this purpose, we executed searches with 100; 1,000; 10,000; 100,000 and 1,000,000 queries. As Table 1 shows, the runtime rises in unison with the number of queries, while the used memory remains constant. These observations are similar to last week. In the case of the FM-Index, the runtime is about twice as long as the suffix array. However, the memory consumption is much lower.

The search for 1,000,000 queries resulted in an error after about eight minutes, even after multiple tries. Since the error occurred during the FM-Index search of the SeqAn-library, this value was excluded from the results. A possible reason could be memory issues, as this run needed much more space than the other runs with lower query numbers.

Table 1: Results for the benchmarking with 100; 1,000; 10,000; 100,000; and 1,000,000 queries of length 100.

Queries	Naive approach		Suffix Array approach		FM-Index approach	
	Runtime	Memory	Runtime	Memory	Runtime	Memory
100	4m23s	217004 kB	2ms	607952 kB	4 ms	82120 kB
1,000	41m36s	217000 kB	16ms	607952 kB	41 ms	82124 kB
10,000	6h52m55s	217000 kB	159ms	607952 kB	410 ms	82124 kB
100,000	-	-	1558ms	607948 kB	3994 ms	82124 kB
1,000,000	-	-	1655ms	631392 kB	-	-

Additionally, the influence of different query lengths was examined. 100,000 queries of the lengths 40, 60, 80 and 100 were searched (Table 2). The results are inconclusive. The queries of length 40 took the most time. A steady increase can only be seen, when comparing the memory. Once again the memory is lower than the used space in the suffix array computations, but the runtime is inferior.

Table 2: Results for the benchmarking of queries of the length 40, 60, 80, and 100 with 100 queries for naive approach and 100,000 queries for suffixarray approach and FM-Index.

Query-Length	Naive approach		Suffix Array approach		FM-Index	
	Runtime	Memory	Runtime	Memory	Runtime	Memory
40	248s	210756 kB	1427ms	601704 kB	5894 ms	76024 kB
60	249s	213880 kB	1481ms	604824 kB	3551 ms	79024 kB
80	248s	215440 kB	1539ms	606392 kB	3702 ms	80568 kB
100	249s	217004 kB	1558ms	607948 kB	3994 ms	82124 kB

4 Discussion

In the benchmarking, only parts of the first chromosomes were used, which contained overall 100 million bases. If a whole human genome with 6.4 billion bases would be searched with our algorithms, the runtime would increase about the factor the sequence increases. The human genome is about 64 times bigger than the sequence we tested.

In Table 3, we extrapolated the runtime for 1,000; 10,000; and 100,000 queries on the whole human genome. However, search queries from different lengths still would be relatively constant.

Table 3: Results for the extrapolation on the runtime with 1,000; 10,000; and 100,000 queries of length 100.

Queries	Naive approach	Suffix Array approach	FM-Index approach
1,000	1T20h	1s	3s
10,000	18T8h	10s	26s
100,000	-	1m39s	4m16s

5 Conclusion

In general, the suffix array seems to be the most optimal approach. It was superior in every run (Figure 1). This is unexpected, as the FM Index has a better theoretical runtime. There are several possible reasons for this, such as our implementation not working correctly, errors in the time calculation, or inefficient memory management.

It is notable, that the printing of the results takes up a significant amount of time. In our benchmarking of the FM Index, we included a separate for-loop that prints all results, since the benchmarking of the suffix array approach also included printing all results. Without printing the results, the FM Index search took less than a millisecond.

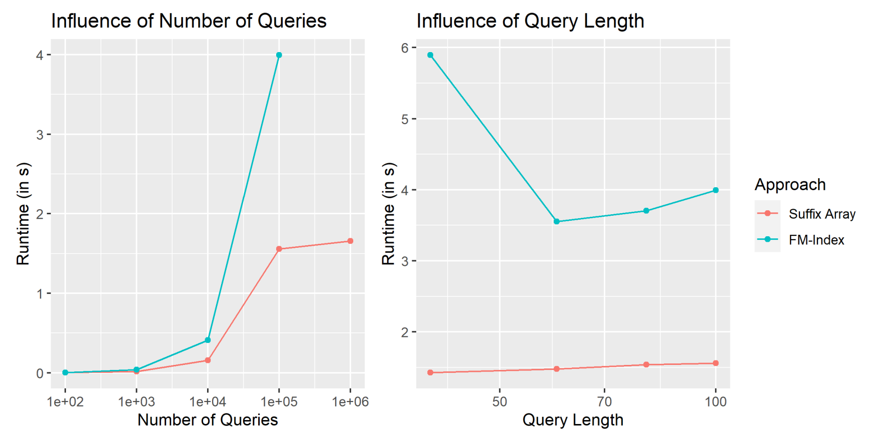

Figure 1: **Influence of Changes in Queries on Runtime** The figure shows how changes in query length or the amount of queries influence the runtime of the suffix array approach and the FM-Index approach.

Competing interests
The authors declare that they have no competing interests.

Author's contributions
Gokul Thothathri and Swetha Rose Maliyakal Sebastian informed about the scientific background. Sina Glöckner and Christina Kirschbaum implemented and benchmarked the search and wrote methods, results, discussion and conclusion in the report.

References

1. Nemytykh A. On Specialization of a Program Model of Naive Pattern Matching in Strings (Extended Abstract). 2021 08;.
2. Diwate R. Study of Different Algorithms for Pattern Matching. 2013 03;.
3. Simon J Puglisi WFS, Turpin A. A taxonomy of suffix array construction algorithms, ACM Computing Surveys, Vol. 39, Issue 2,. 2007 01;p. 2141–2144.
4. Ferragina P, Manzini G. Opportunistic data structures with applications. In: Proceedings 41st annual symposium on foundations of computer science. IEEE; 2000. p. 390–398.
5. Ferragina P, Manzini G. Opportunistic data structures with applications. In: Proceedings 41st Annual Symposium on Foundations of Computer Science; 2000. p. 390–398.
6. Simpson JT, Durbin R. Efficient construction of an assembly string graph using the FM-index. Bioinformatics. 2010 Jun;26(12):i367–i373. Available from: https://doi.org/10.1093/bioinformatics/btq217.
7. Fernandez E, Najjar W, Lonardi S. String Matching in Hardware Using the FM-Index. In: 2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines; 2011. p. 218–225.
8. Reinert K, Dadi TH, Ehrhardt M, Hauswedell H, Mehringer S, Rahn R, et al. The SeqAn C++ template library for efficient sequence analysis: A resource for programmers. Journal of Biotechnology. 2017;261:157–168. Bioinformatics Solutions for Big Data Analysis in Life Sciences presented by the German Network for Bioinformatics Infrastructure. Available from: https://www.sciencedirect.com/science/article/pii/S0168165617315420.
9. Gog S, Beller T, Moffat A, Petri M. From Theory to Practice: Plug and Play with Succinct Data Structures. In: Gudmundsson J, Katajainen J, editors. Experimental Algorithms. Cham: Springer International Publishing; 2014. p. 326–337.