CS 7800/4810: Advanced Algorithms Spring 2026

Lecture 8 — Feb 04, 2026

Prof. Prashant Pandey *Scribe: Abhishek Pujara*

1 Overview

In the last lecture we discussed **Suffix Arrays** and **Burrows-Wheeler Transform (BWT)**.

Goal

The goal of this lecture is to be able to **store the human genome** efficiently.

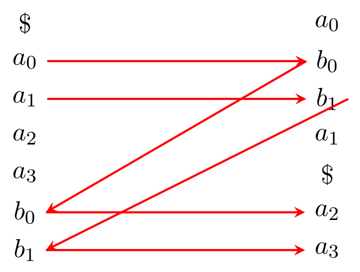
- Human genome: 3 billion base pairs
- Alphabet size: $|\Sigma| = 4$ (DNA bases)
- Memory requirements:
  - Suffix tree: 47 GB
  - Suffix array: 12 GB
  - FM-index: 1.5 GB

2 BWT Pattern Search Review

Consider the string $T =$ `abaaba$` and the pattern $P =$ `aba`.

The BWT produces first and last columns F and L (with subscripts to distinguish repeated letters):

F	\$	a_0	a_1	a_2	a_3	b_0	b_1
L	a_0	b_0	b_1	a_1	\$	a_2	a_3



Explanation: The red arrows show the **LF-mapping steps** as we search for the pattern `aba`. This corresponds to tracking the suffixes that match the pattern in reverse order via the BWT.

3 FM-Index

Ferragina and Manzini [2000]

The FM-index is an index that **combines the BWT with a few small auxiliary data structures**.

- Core of the index: **First (F) + Last (L) columns** from BWT
- F can be represented very simply (1 integer per character)
- L is highly compressible

3.1 Issues with BWT Pattern Search

1. Scanning characters in the last column L can be very slow.
2. Need a way to find where matches occur in T.

3.2 Solution 1: Tally Table

Idea: Precompute number of occurrences of each character up to each row (cumulative counts).

Without the tally table, we need to check all of a_0, a_1, a_2, a_3 to see which has a corresponding b in the right column. With the tally table, we can narrow the search window to between a_1 and a_2, significantly reducing the number of checks.

F	\$	a_0	a_1	a_2	a_3	b_0	b_1
L	a_0	b_0	b_1	a_1	\$	a_2	a_3

Tally:	1	2	3	4	5	6	7
a	1	1	1	2	2	3	4
b	0	1	2	2	2	2	2

Space trade-off: Storing every entry in the tally table is costly. Instead, we store every x-th tally and perform a few extra scans between stored entries (sparsifying), balancing space and speed.

3.3 Solution 2: Sampled Suffix Array

Idea: Tally tables solve the problem of speeding up the search, but we still need a way to locate the matching substrings. The suffix array directly corresponds to the ranking of the permuted strings in the BWT and lets us retrieve the offsets of the matches.

- We sparsify and store only some entries of the suffix array.

4 FM-Index Memory Footprint

- **First Column (F):** $\sim |\Sigma|$ integers
- **Last Column (L):** T characters
- **Sampled Suffix Array:** $T \cdot a$ integers, where a is the fraction of rows sampled
- **Sampled Tally:** $T \cdot |\Sigma| \cdot b$ integers, where b is the fraction of rows sampled

4.1 Example: Human Genome

- DNA alphabet: $|\Sigma| = 4$
- Human genome: 3 billion base pairs
- F: $4 \times 4 = 16$ bytes
- L: 2 bits per base $\times$3B bases $\approx$ 750 MB
- Sampled SA: 3B $\times 4 \times 1/32 \approx$ 400 MB
- Sampled Tally: 3B $\times 4 \times 4 \times 1/128 \approx$ 100 MB
- **Total:** $\lesssim$ 1.5 GB

References

P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS '00, page 390, USA, 2000. IEEE Computer Society. ISBN 0769508502.