CS 7280: Data Str and Algs for Scalable Computing	Spring 2025

Lecture 6 — January 29, 2025

Prof. Prashant Pandey *Scribe: Bada Kwon*

1 Overview

Recap:
Discussing string inputs which could vary in universe size depending on the alphabet. Our goal is to be able to find matches of a pattern P in a string T, using **very little space and very quickly**

String Matching:

- Text T and Pattern P = Strings over Σ alphabet
- Goal: Find all occurrences of P in T, with location and number of occurrences

Methods we've covered so far.

- Brute Force with rolling hash (slightly better): $O(T)$ time complexity.
- Tries: Space = Array, BST, Hash Table,... and efficient queries.
- Suffix Trees: $O(P)$ query time, $O(T)$ space.
- Suffix Arrays: $O(P \log T)$ query time, $O(T)$ space.
- FM-index: A compressed representation using the Burrows-Wheeler Transform (BWT).

Human Genome Example:

- $|T| = 3$ Billion
- $|\Sigma| = 4$ (A,C,T,G)
- Suffix Tree = about 47GB
- Suffix Array = about 12GB
- FM-index = about 1.5GB

2 FM-index

The FM-index ([1]) uses indexes which combine the BWT with some small auxiliary data structures. The core of the the index consists of F and L columns from the BW-Matrix.

2.1 Key Components

- F: First column (sorted characters of the text).
- L: Last column from the BWT matrix.

idx	F		L
0	\$	...	a_0
1	a_0	...	b_0
2	a_1	...	b_1
3	a_2	...	a_1
4	a_3	...	\$
5	b_0	...	a_2
6	b_1	...	a_3

Table 1: F and L

2.2 Pattern Matching with FM-Index

- T = abaaba\$
- P = aba
- Store F as $|\Sigma|$ indexes
- Try to find P within T by using F and L ...
  - Mapping from \$ $\rightarrow a_0$, then checking for a_0 in F to find the next predecessor in T
  - repeat the process...
  - With this, find the pattern match.

In order to find ALL occurrences of P, use index of CLUSTERS of similar characters in F

2.3 ISSUES

1. SLOW: if we scan characters in L, it is very slow.
   - $O(m)$ time
   - (m = $|L|$)
2. Storing ranks takes too much space
3. Doesn't find WHERE the matches occur in T

2.4 SOLUTIONS

1. IDEA 1: Is there $O(1)$ way to determine which b's preceded the a's in our range?
   - Pre-calculate number of a's and b's in L up to every row.
   - Space = m * $|\Sigma|$
idx	a	b
0	1	0
1	1	1
2	1	2
3	2	2
4	2	2
5	3	2
6	4	2
   - index 0 and 5: a precedes b
2. IDEA 2: If Suffix Array was part of the index, we can simplify, by looking up the offset.
   - Suffix Array takes $O(T)$ space.
3. IDEA 3: SPARSIFY
   - Only store SOME rows, every fraction.
   - EXAMPLE:
idx	a	b
0	1	0
3	2	2
6	4	2
   - The **size between each "Checkpoint"** row should be **small enough** for intermediate information to be accessed with a linear scan in efficient time.
   - FM-index Space: Human Genome Example
     - F: about $|\Sigma|$ integers $\rightarrow$ 16 Bytes
       - 16 Bytes = 4 Bytes (per integer) * 4 letters
     - L: m characters $\rightarrow$ 750 megabytes
       - 750 megabytes = 6 Billion bits (each letter as 2 bits, 2 * 3 billion) / 8 = 750k Bytes
     - Suffix Array Sample: m * a (fraction a = 1/32) $\rightarrow$ 400 megabytes
       - 400 megabytes $=\frac{3billion*4}{32}$
     - Checkpoint Tally: m * $|\Sigma|$ * b (fraction b = 1/128) $\rightarrow$ 100 megabytes
       - 100 megabytes $=\frac{3billion*4*4}{128}$
     - **TOTAL: about 1.5 GB**

3 Hashing: Balls and Bins preview

3.1 Probability Analysis

How many tosses of a 50/50 coin to have at least 1 heads result?

- **WITH HIGH PROBABILITY:**
- $P[E] = 1 - \frac{1}{n^c}$, n = number of events
- $\lim_{x \to \inf} f(x) \to 1$, $c \geq 1$
- What order of "n" will be needed for high probability?

C value for at least one head:

- $P[\text{no head}] = \frac{1}{2}$, for one toss
- $P[\text{no head in "i" tosses}] = \frac{1}{2^i}$
- $i = 3lg(n) \to \frac{1}{2^{3lgn}} \to \frac{1}{n^3}$ ... so
- $P[\text{at least one head in n}] = 1 - P[\text{no head in "i" tosses}] = 1 - \frac{1}{n^3}$

References

[1] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. *FOCS*, 20000.