String Algorithms and Data Structures

FM Index

CS 199-225
Brad Solomon

April 3, 2023

Department of Computer Science
Burrows-Wheeler Transform

Reversible permutation of the characters of a string

Burrows M, Wheeler DJ: A block sorting lossless data compression algorithm. *Digital Equipment Corporation, Palo Alto, CA* 1994, Technical Report 124; 1994
Burrows-Wheeler Transform: LF Mapping

The $i^{th}$ occurrence of a character $c$ in $L$ and the $i^{th}$ occurrence of $c$ in $F$ correspond to the *same* occurrence in $T$ (i.e. have same rank)

Any ranking we give to characters in $T$ will match in $F$ and $L$
Burrows-Wheeler Transform: LF Mapping

Another way to visualize:

\[
\begin{array}{cccccccccccccc}
F & L & F & L & F & L & F & L & F & L & F & L & F & L \\
\$ & a_3 & $ & a_3 & $ & a_3 & $ & a_3 & $ & a_3 & $ & a_3 & $ & a_3 \\
a_3 & b_1 & a_3 & b_1 & a_3 & b_1 & a_3 & b_1 & a_3 & b_1 & a_3 & b_1 & a_3 & b_1 \\
a_1 & b_0 & a_1 & b_0 & a_1 & b_0 & a_1 & b_0 & a_1 & b_0 & a_1 & b_0 & a_1 & b_0 \\
a_2 & a_1 & a_2 & a_1 & a_2 & a_1 & a_2 & a_1 & a_2 & a_1 & a_2 & a_1 & a_2 & a_1 \\
a_0 & $ & a_0 & $ & a_0 & $ & a_0 & $ & a_0 & $ & a_0 & $ & a_0 & $ \\
b_1 & a_2 & b_1 & a_2 & b_1 & a_2 & b_1 & a_2 & b_1 & a_2 & b_1 & a_2 & b_1 & a_2 \\
b_0 & a_0 & b_0 & a_0 & b_0 & a_0 & b_0 & a_0 & b_0 & a_0 & b_0 & a_0 & b_0 & a_0 \\
\end{array}
\]

\[T: \ a_0 \ b_0 \ a_1 \ a_2 \ b_1 \ a_3 \ \$\]
A review of ‘F’ and ‘L’

$L = \text{CGGGCC}$  \hspace{1cm} \Sigma = “ACGT”

How can we represent $F$?
A review of ‘F’ and ‘L’

$L = \text{CGGGCC}$ $\Sigma = \text{“ACGT”}$

How can we represent $F$?

As a full text string: $F = \$\text{CCCGGG}$

As a map<string, int>: $F = \{\$: 1, \text{C}: 3, \text{G}: 3\}$

As a vector<int>: $F = [0, 3, 3, 0]$
A review of ‘F' and ‘L’

BWT(T) = e$1ppa

What row index in $F$ contains ‘e’?

What row index in $L$ contains ‘e’?

What row index in $F$ contains the second ‘p’?
FM Index

An index combining the BWT with a few small auxiliary data structures

Core of index is **first (F)** and **last (L)** rows from BWM:

$L$ is the same size as $T$

$F$ can be represented as array of $|\Sigma|$ integers (or not stored at all!)

We’re discarding $T$ — *we can recover it from $L*$!
FM Index: Querying

\[ P = \text{A A A} \]

\[
\begin{array}{cccccccc}
\$ & B & B & B & A & A & A_0 \\
A_0 & $ & B & B & B & A & A_1 \\
A_1 & A & $ & B & B & B & A_2 \\
A_2 & A & A & $ & B & B & B_0 \\
B_0 & A & A & A & $ & B & B_1 \\
B_1 & B & A & A & A & $ & B_2 \\
B_2 & B & B & A & A & A & $ \\
\end{array}
\]
FM Index: Querying

\[ P = B \ A \ B \]

\[
\begin{align*}
\$ & B \ B \ B \ A \ A \ A_0 \\
A_0 & \$ B B B A A_1 \\
A_1 & A \$ B B B A_2 \\
A_2 & A A \$ B B B_0 \\
B_0 & A A A \$ B B_1 \\
B_1 & B A A A \$ B_2 \\
B_2 & B B A A A A \$
\end{align*}
\]
FM Index: Lingering Issues
FM Index: Lingering Issues

(1) Scanning for preceding character in \( L \) is slow

| $ | a | b | a | a | b | a_0 |
|---|---|---|---|---|---|-----|
| a_0 | $ | a | b | a | a | b_0 |
| a_1 | a | b | a | $ | a | b_1 |
| a_2 | b | a | $ | a | b | a_1 |
| a_3 | b | a | a | b | a | $ |
| b_0 | a | $ | a | b | a | a_2 |
| b_1 | a | a | b | a | $ | a_3 |

Need way to find where matches occur in \( T \):

| $ | a | b | a | a | b | a_0 |
|---|---|---|---|---|---|-----|
| a_0 | $ | a | b | a | a | b_0 |
| a_1 | a | b | a | $ | a | b_1 |
| a_2 | b | a | $ | a | b | a_1 |
| a_3 | b | a | a | b | a | $ |
| b_0 | a | $ | a | b | a | a_2 |
| b_1 | a | a | b | a | $ | a_3 |

We don’t store ranks!

Current output: \([3,4]\)

Location in \( T \): \([0,3]\)

This is where our auxiliary data structures come in...
FM Index: Fast rank calculations

Is there a fast way to determine which *specific* bs precede the as in our range?

More generally, given a range in $L$ and a character to search, how can we quickly find all matches (and their ranks)?
FM Index: Occurrence Table

Idea: pre-calculate cumulative # a's, b's in L up to every row:

| L | a | b |
|---|---|---|
| a |   |   |
| b |   |   |
| b |   |   |
| a |   |   |
| $ |   |   |
| a |   |   |
| a |   |   |
| a |   |   |
# FM Index: Occurrence Table

Idea: pre-calculate cumulative # as, bs in L up to every row:

| $L$ | a | b |
|-----|---|---|
| a   | 1 | 0 |
| b   | 1 | 1 |
| b   | 1 | 2 |
| a   | 2 | 2 |
| $\$|$ 2 | 2 |
| a   | 3 | 2 |
| a   | 4 | 2 |
**FM Index: Occurrence Table**

Idea: pre-calculate cumulative # as, bs in L up to every row:

| $ | a | b |
|---|---|---|
| 1 | 0 |
| a | b | 1 | 1 |
| a | b | 1 | 2 |
| a | a | 2 | 2 |
| a | $ | 2 | 2 |
| b | a | 3 | 2 |
| b | a | 4 | 2 |
**FM Index: Occurrence Table**

Query: ‘aba’

Idea: pre-calculate cumulative # of as, bs in L up to every row:

|   | L    | F | a | b |
|---|------|---|---|---|
| $ | a    |   | 1 | 0 |
| a | b    |   | 1 | 1 |
| a | b    |   | 1 | 2 |
| a | a    |   | 2 | 2 |
| a | $    |   | 2 | 2 |
| b | a    |   | 3 | 2 |
| b | a    |   | 4 | 2 |

- 0 bs up to & including this row
- 2 bs up to & including this row
FM Index: Occurrence Table

Query: ‘aba’

Idea: pre-calculate cumulative # a's, b's in L up to every row:

|   | F | L | a | b |
|---|---|---|---|---|
| $ | a | 1 | 0 |
| a | b | 1 | 1 |
| a | b | 1 | 2 |
| a | a | 2 | 2 |
| a | $ | 2 | 2 |
| b | a | 3 | 2 |
| b | a | 4 | 2 |
What two indices should I look up? What ranks did we find?

| $F$ | $L$ | a | b |
|-----|-----|---|---|
| $ $ | a   | 1 | 0 |
| a   | b   | 1 | 1 |
| a   | $ $ | 1 | 1 |
| b   | b   | 1 | 2 |
| b   | b   | 1 | 3 |
| b   | b   | 1 | 4 |
| b   | a   | 2 | 4 |
FM Index: Occurrence Table

An index combining the BWT with a few small auxiliary data structures

Occurrence table speeds up $L$ lookup by implicitly storing ranks

Table is $m \times |\Sigma|$ integers — that’s worse than a suffix array!
Next idea: pre-calculate \# a's, b's in L up to some rows, e.g. every 5\textsuperscript{th} row. Call pre-calculated rows *checkpoints*.

| $F$ | $L$ | a | b |
|-----|-----|---|---|
| $\$$ | a | 1 | 0 |
| a   | b |   |   |
| a   | b |   |   |
| a   | b |   |   |
| a   | a |   |   |
| a   | $\$$|   |   |
| b   | a | 3 | 2 |
| b   | a |   |   |
FM Index: Occurrence Table

To resolve a lookup for a non-checkpoint row, walk to nearest checkpoint. Use value at that checkpoint, \textit{adjusted for characters we saw along the way}.

| $F$ | $L$ | $a$ | $b$ |
|-----|-----|-----|-----|
| $\$ | $a$ | 1   | 0   |
| $a$ | $b$ |     |     |
| $a$ | $b$ |     |     |
| $a$ | $b$ |     |     |
| $a$ | $a$ |     |     |
| $a$ | $a$ |     |     |
| $a$ | $\$ |     |     |
| $b$ | $a$ | 3   | 2   |
| $b$ | $a$ |     |     |
If checkpoints are $O(1)$ distance apart, lookups are $O(1)$.
FM Index: Occurrence Table

An index combining the BWT with *a few small auxiliary data structures*

Occurrence table speeds up $L$ lookup by implicitly storing **ranks**

Checkpoints reduce the storage costs (Still $O(m)$ but better than SA)
### FM Index: Querying

**Problem 2:** We don’t know *where* the matches are in $T$...

Let $P = \text{aba}$

Got the same range, $[3, 4]$, we would have got from suffix array

| $F$       | $L$       |
|-----------|-----------|
| $\$ a b a a b a_0$ | 6 $\$       |
| $a_0 \$ a b a a b_0$ | 5 a $\$       |
| $a_1 a b a \$ a b_1$ | 2 a a b a $\$       |
| $a_2 b a \$ a b a_1$ | 3 a b a $\$       |
| $a_3 b a a b a \$     | 0 a b a a b a $\$       |
| $b_0 a \$ a b a a_2$ | 4 b a $\$       |
| $b_1 a a b a \$ a_3$ | 1 b a a b a $\$       |

Where are these?

Where are these?
**FM Index: Suffix Array Sampling**

Idea: store some suffix array elements, but not all

| $ | a | b | a | a | b | a | \( F \) |
|---|---|---|---|---|---|---|---|
| a | $ | a | b | a | a | b | \( L \) |
| a | a | b | a | $ | a | b | \( \) |
| a | b | a | $ | a | b | a | \( \) |
| a | b | a | a | b | a | $ | \( \) |
| b | a | $ | a | b | a | a | \( \) |
| b | a | a | b | a | $ | a |

\[ SA'(\text{evens only}) \]

| 6 |
|---|
| 2 |
| 0 |
| 4 |

Lookup for row 4 succeeds

Lookup for row 3 fails - SA entry was discarded
FM Index: Suffix Array Sampling

LF Mapping tells us that “a” at the end of row 3 corresponds to...

...“a” at the beginning of row 2

If saved SA values are $O(1)$ positions apart in $T$, resolving index is $O(1)$ time
FM Index: Suffix Array Sampling

Many LF-mapping steps may be required to get to a sampled row:

Starting here

|   |   |   |   |   |
|---|---|---|---|---|
| $ | a | b | a | a |
| a | $ | a | b | a |
| a | a | b | a | $ |
| a | $ | b | a | b |
| a | a | b | a | $ |
| a | b | a | b | a |
| b | a | $ | a | b |
| b | a | a | b | a |
| b | a | a | b | $ |

SA' (every 4th)

0
4
FM Index: Suffix Array Sampling

Many LF-mapping steps may be required to get to a sampled row:

\[
\begin{array}{cccc}
F & L & \text{SA'} (\text{every 4th}) \\
$ & a & b & a \\
a & $ & a & b \\
a & a & b & a \\
& a & b & a \\
& a & b & a \\
& a & b & a \\
& a & b & a \\
& a & b & a \\
\end{array}
\]
FM Index: Suffix Array Sampling

Many LF-mapping steps may be required to get to a sampled row:

Starting here →

| F  | L  | SA' (every 4th) |
|----|----|-----------------|
| $  | a b a b a b     | 0              |
| a  | a $ a b a b     | 4              |
| a  | b a $ a b       |                |
| a  | b a a b a       |                |
| b  | a $ b a a b     |                |
| b  | a a b a $       |                |
FM Index: Suffix Array Sampling

Many LF-mapping steps may be required to get to a sampled row:

Starting here

Missing value = 0 (SA val at destination) + 3 (# steps to destination) = 3
FM Index: Suffix Array Sampling

An index combining the BWT with a few small auxiliary data structures

Stores all index positions in T with O(1) extra work to calculate

\[
\begin{align*}
\text{Index: 0} & \quad \text{Index: 0 + 3 = 3} \\
\end{align*}
\]

Lets put all these pieces together…
FM Index: Querying

\[ P = \text{aba} \]

\[
\begin{array}{cccccccc}
F & & & & & & & \\
L & & & & & & & \\
$ & a & b & a & a & b & a_0 & \\
a_0 & $ & a & b & a & a & b & \\
a_1 & a & b & a & $ & a & b & \\
a_2 & b & a & $ & a & b & a_1 & \\
a_3 & b & a & a & b & a & $ & \\
b & a & $ & a & b & a & a_2 & \\
b & a & a & b & a & $ & a_3 & \\
\end{array}
\]

get_frange()
pair<int, int> get_frange(string c, int s, int e)

Input:

string c: The char we are looking for in F
int s: The starting rank value
int e: The ending rank value

Output:

A pair of values (index start, index end)

What are c, s, and e?

What are the output values?
FM Index: Querying

\[ P = \texttt{aba} \]

\[ F \quad L \]

\[
\begin{array}{cccccccc}
\$ & a & b & a & a & b & a_0 \\
{a_0} & $ & a & b & a & a & b \\
{a_1} & a & b & a & $ & a & b \\
{a_2} & b & a & $ & a & b & a_1 \\
{a_3} & b & a & a & b & a & $ \\
{b} & a & $ & a & b & a & a_2 \\
{b} & a & a & b & a & $ & a_3 \\
\end{array}
\]

get\_frange()  get\_lrange()
pair<int, int> get_lrange(string c, int s, int e)

Input:
- **string c**: The char we are looking for in $F$
- **int s**: The starting *index* of our range
- **int e**: The ending *index* of our range

Output:
- A pair of values (# occurrences start, end)

What are $c$, $s$, and $e$?

What are the output values?
FM Index: Querying

\[ P = \text{aba} \]

\[
\begin{array}{c|c|c}
F & L \\
$ & a & b \\
a_0 & $ & a \\
a_1 & a & b \\
a_2 & b & a \\
a_3 & b & a \\
\end{array}
\]

get_frange()  

get_lrange()
pair<int, int> get_frange(string c, int s, int e)

Input:

string c: The char we are looking for in F
int s: The starting rank value
int e: The ending rank value

Output:

A pair of values (index start, index end)

What are c, s, and e?

What are the output values?
FM Index: Querying

\[ P = aba \]

\[
\begin{array}{c|c}
F & L \\
\hline
\$ & a \ b \ a \ a \ b \ a_0 \\
a_0 & a \ b \ a \ a \ b \\
a_1 & a \ b \ a \ $ \ a \ b \\
a_2 & b \ a \ $ \ a \ b \ a_1 \\
a_3 & b \ a \ a \ b \ a \ $ \\
\end{array}
\]

get_frange() \quad get_lrange()
get_lrange(‘a’,5,6)->[2,4]  
$P = \text{aba}$  

\[ F \quad L \]
\[
\begin{array}{cccccc}
$ & a & b & a & a & b & a_0 \\
a_0 & $ & a & b & a & a & b_0 \\
a_1 & a & b & a & $ & a & b_1 \\
a_2 & b & a & $ & a & b & a_1 \\
a_3 & b & a & a & b & a & $ \\
b_0 & a & $ & a & b & a & [a_2] \\
b_1 & a & a & b & a & $ & [a_3] \\
\end{array}
\]

get_frange(‘a’,2,3)->[3,4]  

\[ F \quad L \]
\[
\begin{array}{cccccc}
$ & a & b & a & a & b & a_0 \\
a_0 & $ & a & b & a & a & b_0 \\
a_1 & a & b & a & $ & a & b_1 \\
a_2 & b & a & $ & a & b & a_1 \\
a_3 & b & a & a & b & a & $ \\
b_0 & a & $ & a & b & a & a_2 \\
b_1 & a & a & b & a & $ & a_3 \\
\end{array}
\]

SA[3] = 3, SA[4] = 0 --> Return {0, 3}
**FM Index**

\(|T| = m, \ |P| = n\)

Finding all matches of \(P\) occurs in \(T\) in FM Index is _____________ time
Assignment 9: a_fmi

Learning Objective:

Construct a full FM Index

Implement exact pattern matching on a FM Index

Consider: How would you modify the provided code to handle sub-sampling in the Occurrence Table (OT) or Suffix Array (SA)?
FM Index

Let $a =$ fraction of rows we keep

Let $b =$ fraction of SA elements we keep

FM Index consists of these, plus $L$ and $F$ columns

Note: suffix tree/array didn't have parameters like $a$ and $b$
FM Index

Components of FM Index:  (blue indicates what we can adjust by changing $a$ & $b$)

- First column ($F$): $\sim |\Sigma|$ integers
- Last column ($L$): $m$ characters
- SA sample: $m \cdot a$ integers, $a$ is fraction of SA elements kept
- OT Checkpoints: $m \cdot |\Sigma| \cdot b$ integers, $b$ is fraction of tallies kept

For DNA alphabet (2 bits / nt), $T =$ human genome, $a = \frac{1}{32}, b = \frac{1}{128}$:

- First column ($F$): 16 bytes
- Last column ($L$): 2 bits * 3 billion chars = 750 MB
- SA sample: 3 billion chars * 4 bytes / 32 = $\sim$ 400 MB
- OT Checkpoints: 3 billion * 4 alphabet chars * 4 bytes / 128 = $\sim$ 400 MB

Total $\approx 1.5$ GB  $\sim$0.5 bytes per input char
FM Index: Small Memory Footprint

- **Suffix tree**: $\geq 45 \text{ GB}$
- **Suffix array**: $\geq 12 \text{ GB}$
- **FM Index**: $\sim 1.5 \text{ GB}$
### Suffix-Based Index Bounds

|                        | Suffix tree | Suffix array | FM Index |
|------------------------|-------------|--------------|----------|
| Time: Does $P$ occur?  |             |              |          |
| Time: Count $k$        |             |              |          |
| occurrences of $P$     |             |              |          |
| Time: Report $k$       |             |              |          |
| locations of $P$       |             |              |          |
| Space                  |             |              |          |
| Needs $T$              |             |              |          |
| Bytes per input        |             |              |          |
| character              |             |              |          |

$m = |T|$, $n = |P|$, $k = \#$ occurrences of $P$ in $T$
# Suffix-Based Index Bounds

|                          | Suffix tree | Suffix array | FM Index |
|--------------------------|-------------|--------------|----------|
| Time: Does P occur?      | $O(n)$      | $O(n \log m)$ | $O(n)$   |
| Time: Count $k$          | $O(n + k)$  | $O(n \log m)$ | $O(n)$   |
| occurrences of P         |             |               |          |
| Time: Report $k$ locations of P | $O(n + k)$  | $O(n \log m + k)$ | $O(n + k)$ |
| Space                    | $O(m)$      | $O(m)$       | $O(m)$   |
| Needs T?                 | yes         | yes          | no       |
| Bytes per input character| $>15$       | $\sim4$     | $\sim0.5$ |

$m = |T|$, $n = |P|$, $k = \#$ occurrences of $P$ in $T$