Burrows-Wheeler Transform [BW, 1994]

Reversible permutation of the characters of a string, used originally for compression.

a b a a b a $ → (Sort) All rotations

T

$	a	b	a	a	b	a
a	$	a	b	a	a	b
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

Burrow-Wheeler matrix

Burrows Wheeler transform: a b b a $ a a

BWT(T)

Q: How is it useful for compression?

Q: How is it reversible?

Q: How is it an index?

1: BWT brings like characters together in a run.

→ we can use run-length encoding to compress BWT(T).

* BWM bears a resemblance to the suffix array.

BWM(T)						
$	a	b	a	a	b	a
a	$	a	b	a	a	b
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(T)	
6	$
5	a$
2	aaba$
3	aba$
0	abaaba$
4	ba$
1	baaba$

Sort order is the same whether rows are rotations or suffixes.

Alternate way of constructing BWT(T)

$$BWT[i] = \begin{cases} T[SA[i]-1] & \text{if } SA[i]>0 \\ \$ & \text{if } SA[i]=0 \end{cases}$$

"BWT = characters just to the left of the suffixes in the suffix array"

Q. How to reverse the BWT?

T-ranking.

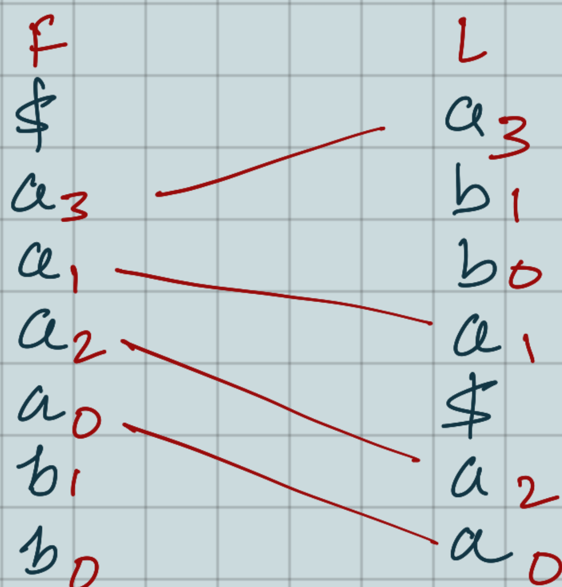
Given each character in T a rank, equal to # times the character occurred previously in T.

$a_0\, b_0\, a_1\, a_2\, b_1\, a_3\, \$$

* Ranks are not explicitely stored; they are just for illustration.

BWM with T-ranking.

Look at First & Last columns, called F, L



* as, bs occur in the same order in F & L

LF Mapping:-

The ith occurrence of a character c in L and the ith occurrence of c in F correspond to the same occurrence in T (i.e have the same Rank)

However we rank occurrences of c, ranks appear in the same order in F & L.

B-ranking.

We would like a different ranking so far so that for a given character, ranks are in ascending order as we look down the F/L columns.

	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

Ascending rank.

Q: Which BWM row begins with b_1?

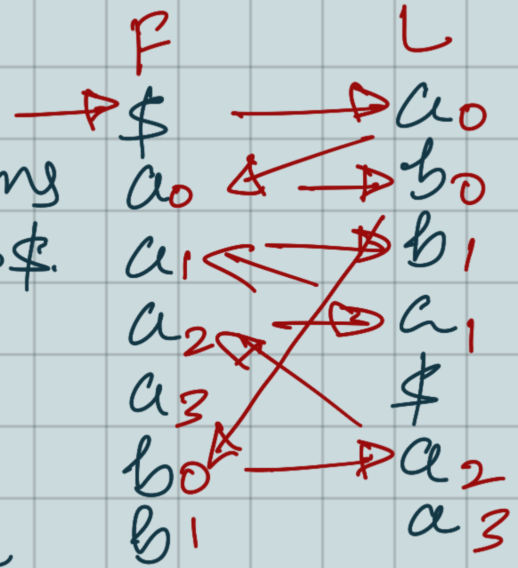
* F now has a very simple structure; a \$, a block of as with ascending ranks, a block of bs with ascending ranks.

Skip row starting \$ (1 row)
Skip rows starting a (4 rows)
Skip row starting b_0 (1 row)

Answer: row 6

BWT reversing:

Reverse BWT(T) starting at right-hand side of T and moving left.



Start in first row. F must have $. L contains character just prior to $.

a_0: LF mapping says this is the same occurrence of a as first a in F.
Jump to row begining with a_0. L contains character just prior to a_0: b_0

Repeat for b_0 : a_2
" a_2 : a_1
" a_1 : b_1
" b_1 : a_3
" a_3 : $: DONE

$T = a_3 b_1 a_1 a_2 b_0 a_0 \$$
original string.

FM-<u>Index</u> [Ferragina-Manzini FOCS 2000]

An index combining the BWT with a few small auxilliary data structures.

Core of the index consists of F & L from BWM

F can be represented very simply (1 integer per alphabet)

And L is compressible

Potentially very space economical.

For example: To represent human genome (3 G Bases)

ST	:	~47 GB
SA	:	~12 GB
FM-index	:	< 1.5 GB

FM-index: querying.

Though BWM is related to SA, we can't query it the same way.

* Binary search isn't possible

→ Look for range of rows of BWM(T) with P as prefix

→ Do this of P's shortest suffix, then extend to successively longer suffixes until range becomes empty or we have exhausted P.

P = aba

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

Rows that begin with a (rows 1–4)

Rows that begin with ba (rows 5–6)

Rows that begin with aba (rows 3–4)

Found the pattern

* Unlike SA, we don't immediately know where the matches are in T.

FM-index: Lingering issues.

1. If we scan characters in the last column, that can be very slow, $O(m)$
2. Storing ranks takes too much space.
3. Need way to find where matches occur in T.

Solutions:-

1. Is there an $O(1)$ way to determine which bs precede the as in our range?

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

F	L	Tally a	Tally 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

$O(1)$ time. but requires $m \times |\Sigma|$ integers.

Idea: Sparsify the Tally.

(2) With checkpoints, we greatly reduce # integers needed for ranks

But it's still $O(m)$ space - there's literature on how to improve this space bound

(3)

Idea: If SA were part of index, we could simply look up the offsets.

But SA requires O integers.

Idea: Store some, but no , entries of the suffi array.

[Sparsify]

FM- index : Small memory footprint.

Components of the FM-index.

First column (F) : $\sim|\Sigma|$ integers

Last column (L) : m characters

SA sample : m. a integers, where a is the f ion of rows kept

checkpoint : m x 1. b integers. where b is the ction of rows checkpointed.

Example: DNA alphabet (2 bits/Nucleotide)

T = Human genome, (3Bi) $a = \frac{1}{32}$, $b = \frac{1}{128}$

First column : 16 bytes
Last column : 2 bits x 3Bi = 750 MB
SA sample : $3Bi \times \frac{4B/char}{32} = \sim 400$ MB
checkpoints : $3Bi \times \frac{4B/char}{128} = \sim 100$ MB

Total : < 1.5GB