Burrows-Wheeler Transform & FM Index

Ben Langmead

Please sign guestbook (www.langmead-lab.org/teaching-materials) to tell me briefly how you are using the slides. For original Keynote files, email me (ben.langmead@gmail.com).
Indexing with suffixes

Suffix Tree

Suffix Array

FM Index

Su

ffix Tree

Su

ffix Array

FM Index

A

ANA

BANANA$

3

1

$ A$ ANANA$

5

4

NA$

2
Burrows-Wheeler Transform

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

All rotations

(then they repeat)
Burrows-Wheeler Transform

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

How is it useful for compression?  How is it reversible?  How is it an index?

Burrows-Wheeler Transform

```python
def rotations(t):
    """ Return list of rotations of input string t ""
    tt = t * 2
    return [ tt[i:i+len(t)] for i in range(0, len(t)) ]

def bwm(t):
    """ Return lexicographically sorted list of t’s rotations ""
    return sorted(rotations(t))

def bwtViaBwm(t):
    """ Given T, returns BWT(T) by way of the BWM ""
    return ''.join(map(lambda x: x[-1], bwm(t)))
```

>>> bwtViaBwm("Tomorrow_and_tomorrow_and_tomorrow$")
'w$wwdd__nnoooaattTmmmrrrrrooo__ooo'

>>> bwtViaBwm("It_was_the_best_of_times_it_was_the_worst_of_times$")
's$esttssfftteww_hhmmbootttt_i__woeeaaressIi_______'

>>> bwtViaBwm('in_the_jingle_jangle_morning_I'll_come_following_you$')
'u_gleeengj_mlhl_nnnnt$nwj__lggIolo_iiiiarfcmylo_oo_'

http://j.mp/CG_BWT
Burrows-Wheeler Transform

$abaaba$

$abaababa$ $abaaba$ $abaababa$

BWT(T) orders T’s characters according to alphabetical order of their right contexts in T.
Right context

The right context of a position in $T$ consists of everything that comes after it with "wrap around"

$T: \text{a b a a b a}$

Right context: $\text{b a a b a}$

$T: \text{a b a a b a}$

Right context: $\text{b a a b a}$
Burrows-Wheeler Transform

Right context:

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

Right context:

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

a b a $ a b$

a b a $ a b$

a b a $ a b$

a b a $ a b$

a b a $ a b$

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

Right context:

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

Right context:

$ a b a a b a$
$a b a a b a$
$a b a a b a$
$b a a b a a$
$b a a b a a$
$b a a b a a$
Burrows-Wheeler Transform

Sorted by right-context

Gives “structure” to BWT(T), making it more compressible

---

Burrows-Wheeler Transform

BWM is related to the suffix array

<table>
<thead>
<tr>
<th>BWM(T)</th>
<th>SA(T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ a b a a b a</td>
<td>6 $</td>
</tr>
<tr>
<td>a $ a b a a b</td>
<td>5 a $</td>
</tr>
<tr>
<td>a a b a $ a b</td>
<td>2 a a b a $</td>
</tr>
<tr>
<td>a b a $ a b a</td>
<td>3 a b a $</td>
</tr>
<tr>
<td>a b a a b a $</td>
<td>0 a b a a b a $</td>
</tr>
<tr>
<td>b a $ a b a a</td>
<td>4 b a $</td>
</tr>
<tr>
<td>b a a b a $ a</td>
<td>1 b a a b a $</td>
</tr>
</tbody>
</table>

Same order whether rows are rotations or suffixes
Burrows-Wheeler Transform

In fact, this gives us a new definition / way to construct 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”
Burrows-Wheeler Transform

```python
def suffixArray(s):
    """ Given T return suffix array SA(T). We use Python's sorted function here for simplicity, but we can do better. ""
    satups = sorted(((s[i:], i) for i in xrange(0, len(s))))
    # Extract and return just the offsets
    return map(lambda x: x[1], satups)

def bwtViaSa(t):
    """ Given T, returns BWT(T) by way of the suffix array. ""
    bw = []
    for si in suffixArray(t):
        if si == 0: bw.append('$')
        else: bw.append(t[si-1])
    return ''.join(bw) # return string-ized version of list bw

>>> bwtViaSa("Tomorrow_and_tomorrow_and_tomorrow$")
'w$wwdd__nnoooaattTmmmmrrrrrooo__ooo'

>>> bwtViaSa("It_was_the_best_of_times_it_was_the_worst_of_times$")
's$esttssfftteww_hhmmbootttt_ii__woeeaaressIi_______'

>>> bwtViaSa('in_the_jingle_jangle_morning_Ill_come_following_you$')
'u_gleeengj_mlhl_nnnnt$nwj__lggIolo_iiiiarfcmylo_o_oo_'
```

How to reverse the BWT?

BWM has a key property called the *LF Mapping*...
Burrows-Wheeler Transform: T-ranking

Give each character in $T$ a rank, equal to # times the character occurred previously in $T$. Call this the $T$-ranking.

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

Ranks aren’t explicitly stored; they are just for illustration

Now let’s re-write the BWM including ranks...
Burrows-Wheeler Transform

BWM with T-ranking:

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$a_0b_0a_1a_2b_1a_3$</td>
<td>$a_3$</td>
</tr>
<tr>
<td>$a_3$</td>
<td>$a_0b_0a_1a_2b_1$</td>
<td>$a_0b_0$</td>
</tr>
<tr>
<td>$a_1a_2b_1a_3$</td>
<td>$a_0b_0a_1$</td>
<td>$a_0b_0a_1a_2$</td>
</tr>
<tr>
<td>$a_0b_0a_1a_2b_1a_3$</td>
<td>$a_0b_0a_1a_2b_1a_3$</td>
<td>$a_0b_0a_1a_2b_1a_3$</td>
</tr>
<tr>
<td>$b_1a_3$</td>
<td>$a_0b_0a_1a_2b_1a_3$</td>
<td>$a_0b_0a_1a_2b_1a_3$</td>
</tr>
<tr>
<td>$b_0a_1a_2b_1a_3$</td>
<td>$a_0b_0a_1a_2b_1a_3$</td>
<td>$a_0b_0a_1a_2b_1a_3$</td>
</tr>
</tbody>
</table>

Look at first and last columns, called $F$ and $L$

And look at just the $a$s

$a$s occur in the same order in $F$ and $L$. As we look down columns, in both cases we see: $a_3, a_1, a_2, a_0$
Burrows-Wheeler Transform

<p>| | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>a_0</td>
<td>b_0</td>
<td>a_1</td>
<td>a_2</td>
<td>b_1</td>
<td>a_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a_3</td>
<td>$</td>
<td>a_0</td>
<td>b_0</td>
<td>a_1</td>
<td>a_2</td>
<td>b_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a_1</td>
<td>a_2</td>
<td>b_1</td>
<td>a_3</td>
<td>$</td>
<td>a_0</td>
<td>b_0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a_2</td>
<td>b_1</td>
<td>a_3</td>
<td>$</td>
<td>a_0</td>
<td>b_0</td>
<td>a_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a_0</td>
<td>b_0</td>
<td>a_1</td>
<td>a_2</td>
<td>b_1</td>
<td>a_3</td>
<td>$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b_1</td>
<td>a_3</td>
<td>$</td>
<td>a_0</td>
<td>b_0</td>
<td>a_1</td>
<td>a_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b_0</td>
<td>a_1</td>
<td>a_2</td>
<td>b_1</td>
<td>a_3</td>
<td>$</td>
<td>a_0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Same with $b$s: $b_1, b_0$
Burrows-Wheeler Transform: LF Mapping

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).

However we rank occurrences of $c$, ranks appear in the same order in $F$ & $L$. 
Burrows-Wheeler Transform: LF Mapping

Why does the LF Mapping hold?

Why are these \(a\)s in this order relative to each other?

They’re sorted by right-context

Occurrences of \(c\) in \(F\) are sorted by right-context. Same for \(L\)!

Whatever ranking we give to characters in \(T\), rank orders in \(F\) and \(L\) will match
Burrows-Wheeler Transform: LF Mapping

BWM with T-ranking:

We’d like a different ranking so that for a given character, ranks are in ascending order as we look down the F / L columns...
**Burrows-Wheeler Transform: LF Mapping**

BWM with B-ranking:

<table>
<thead>
<tr>
<th>F</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>a0</td>
</tr>
<tr>
<td>a0</td>
<td>$</td>
</tr>
<tr>
<td>a1</td>
<td>$</td>
</tr>
<tr>
<td>a2</td>
<td>$</td>
</tr>
<tr>
<td>a3</td>
<td>$</td>
</tr>
<tr>
<td>b0</td>
<td>a0</td>
</tr>
<tr>
<td>b1</td>
<td>a0</td>
</tr>
<tr>
<td>a3</td>
<td>$</td>
</tr>
<tr>
<td>$</td>
<td>a3</td>
</tr>
</tbody>
</table>

Ascending rank

*F now has very simple structure: a $, a block of a's with ascending ranks, a block of b's with ascending ranks*
Burrows-Wheeler Transform

Say $T$ has 300 As, 400 Cs, 250 Gs and 700 Ts and $\$ < A < C < G < T$

Which BWM row (0-based) begins with $G_{100}$? (Ranks are B-ranks.)

Skip row starting with $\$$(1 row)
Skip rows starting with $A$ (300 rows)
Skip rows starting with $C$ (400 rows)
Skip first 100 rows starting with $G$ (100 rows)

Answer: row 1 + 300 + 400 + 100 = row 801
Burrows-Wheeler Transform: 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 $\$$: $a_0$

Jump to row *beginning* with $a_0$. $L$ contains character just prior to $a_0$: $b_0$.

Repeat for $b_0$, get $a_2$  
Repeat for $a_2$, get $a_1$  
Repeat for $a_1$, get $b_1$  
Repeat for $b_1$, get $a_3$  
Repeat for $a_3$, get $\$$(done)

In reverse order, we saw $= a_3 b_1 a_1 a_2 b_0 a_0 \$ = T$
Burrows-Wheeler Transform: reversing

Another way to visualize:

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

\[T: \quad a_3 \ b_1 \ a_1 \ a_2 \ b_0 \ a_0 \ \$\]
Burrows-Wheeler Transform: reversing


def rankBwt(bw):
    ''' Given BWT string bw, return parallel list of B-ranks. Also
    returns tots: map from character to # times it appears. '''
    tots = dict()
    ranks = []
    for c in bw:
        if c not in tots: tots[c] = 0
        ranks.append(tots[c])
        tots[c] += 1
    return ranks, tots

L

{ a: 4, b: 2, $: 1}

Like when we did it by eye, the code depends on knowing the ranks of all the characters in L

But ranks is big! We’ll fix this later
We’ve seen how BWT is useful for compression:

Sorts characters by right-context, making a more compressible string

And how it’s reversible:

Repeated applications of LF Mapping, recreating $T$ from right to left

How is it used as an index?
FM Index

FM Index: an index combining the BWT with a few small auxiliary data structures

Core of index is \( F \) and \( L \) from BWM:

\( L \) is the same size as \( T \)

\( F \) can be represented as array of \(|\Sigma|\) integers

\( L \) is compressible (but even uncompressed, it's small compared to suffix array)

We're discarding \( T \)

---

FM Index: querying

How to query?
FM Index: querying

Can we query like the suffix array?

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

We don’t have these columns, and we don’t have T. Binary search not possible.
FM Index: querying

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

Start with shortest suffix, then match successively longer suffixes

\[
P = \text{aba}
\]

Easy to find all the rows beginning with \( a \)
FM Index: querying

We have rows beginning with $a$, now we want rows beginning with $ba$

\[ P = aba \]

\[ F \]

\[ L \]

$ \quad a \quad b \quad a \quad a \quad b \quad a_0$
\[ a_0 \quad $ \quad a \quad b \quad a \quad a \quad b \quad b_0 \]
\[ a_1 \quad a \quad b \quad a \quad $ \quad a \quad b \quad b_1 \]
\[ a_2 \quad b \quad a \quad $ \quad a \quad b \quad a_1 \]
\[ a_3 \quad b \quad a \quad a \quad b \quad a \quad $ \]
\[ b_0 \quad a \quad $ \quad a \quad b \quad a \quad a_2 \]
\[ b_1 \quad a \quad a \quad b \quad a \quad $ \quad a_3 \]

\[ F \]

\[ L \]

$ \quad a \quad b \quad a \quad a \quad b \quad a_0$
\[ a_0 \quad $ \quad a \quad b \quad a \quad a \quad b \quad b_0 \]
\[ a_1 \quad a \quad b \quad a \quad $ \quad a \quad b \quad b_1 \]
\[ a_2 \quad b \quad a \quad $ \quad a \quad b \quad a_1 \]
\[ a_3 \quad b \quad a \quad a \quad b \quad a \quad $ \]
\[ b_0 \quad a \quad $ \quad a \quad b \quad a \quad a_2 \]
\[ b_1 \quad a \quad a \quad b \quad a \quad $ \quad a_3 \]

Look at those rows in $L$. $b_0, b_1$ are $b$s occuring just to left.

Use LF Mapping. Let new range delimit those $b$s

Now we have the rows with prefix $ba$
FM Index: querying

We have rows beginning with \textit{ba}, now we seek rows beginning with \textit{aba}

\begin{align*}
P &= \text{aba} \\
F &\quad L \\
\$ &\quad a \ b \ a \ a \ b \ a_0 \\
a_0 &\quad $ \ a \ b \ a \ a \ b_0 \\
a_1 &\quad a \ b \ a \ $ \ a \ b_1 \\
a_2 &\quad b \ a \ $ \ a \ b \ a_1 \\
a_3 &\quad b \ a \ a \ b \ a \ $ \\
b_0 &\quad a \ $ \ a \ b \ a \ a_2 \\
b_1 &\quad a \ a \ b \ a \ $ \ a_3 \\
\end{align*}

\begin{align*}
P &= \text{aba} \\
F &\quad L \\
\$ &\quad a \ b \ a \ a \ b \ a_0 \\
a_0 &\quad $ \ a \ b \ a \ a \ b_0 \\
a_1 &\quad a \ b \ a \ $ \ a \ b_1 \\
a_2 &\quad b \ a \ $ \ a \ b \ a_1 \\
a_3 &\quad b \ a \ a \ b \ a \ $ \\
b_0 &\quad a \ $ \ a \ b \ a \ a_2 \\
b_1 &\quad a \ a \ b \ a \ $ \ a_3 \\
\end{align*}

Use LF Mapping

\begin{align*}
\text{a}_2, \text{a}_3 \text{ occur just to left.} \\
\text{Now we have the rows with prefix } \textbf{aba}
\end{align*}
FM Index: querying

\[ P = \text{aba} \]

Got the same range, \([3, 5)\), we would have got from suffix array

\[
\begin{array}{c|c}
F & L \\
\$ & a b a a b a_0 \\
a_0 & a b a a b_0 \\
a_1 & a b a_1 a b \\
a_2 & b a_2 a b a_1 \\
a_3 & b a_3 a b a_2 \\
\end{array}
\]

\[
\begin{array}{c|c}
L & F \\
\$ & 6 \\
a & 5 \\
a b a & 2 \\
a b a a b & 3 \\
a b a b a & 0 \\
a b a a b a & 4 \\
b a a b a & 1 \\
\end{array}
\]

Unlike suffix array, we don’t immediately know *where* the matches are in T...

Where are these?
FM Index: querying

When $P$ does not occur in $T$, we eventually fail to find next character in $L$:

$$ P = \textbf{bba} $$

Rows with $\textbf{ba}$ prefix

$\textbf{b0} a \ a \ b \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \ a \ a \ a \ b \a
FM Index: querying

If we *scan* characters in the last column, that can be slow, $O(m)$

$$P = \texttt{aba}$$

<table>
<thead>
<tr>
<th></th>
<th>$\texttt{aba}$</th>
<th>$\texttt{aba}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$\texttt{aba}$</td>
<td>$\texttt{aba}$</td>
</tr>
<tr>
<td>$L$</td>
<td>$\texttt{aba}$</td>
<td>$\texttt{aba}$</td>
</tr>
<tr>
<td>$a_0$</td>
<td>$\texttt{aba}$</td>
<td>$\texttt{aba}$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$\texttt{aba}$</td>
<td>$\texttt{aba}$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$\texttt{aba}$</td>
<td>$\texttt{aba}$</td>
</tr>
<tr>
<td>$a_3$</td>
<td>$\texttt{aba}$</td>
<td>$\texttt{aba}$</td>
</tr>
<tr>
<td>$b_0$</td>
<td>$\texttt{aba}$</td>
<td>$\texttt{aba}$</td>
</tr>
<tr>
<td>$b_1$</td>
<td>$\texttt{aba}$</td>
<td>$\texttt{aba}$</td>
</tr>
</tbody>
</table>

Scan, looking for $\texttt{bs}$
**FM Index: lingering issues**

1. **Scanning for preceding character is slow**

   - Initial string: $a b a a b a_0$
   - After BWT: $a_0 a b b a a b b a_1$
   - After reverse BWT: $a_0 a b a a b a_1$

   - $O(m)$ scan

2. **Storing ranks takes too much space**

   ```python
def reverseBwt(bw):
    """ Make T from BWT(T) """
    ranks, tots = rankBwt(bw)
    first = firstCol(tots)
    rowi = 0
    t = "$"
    while bw[rowi] != '$':
        c = bw[rowi]
        t = c + t
        rowi = first[c][0] + ranks[rowi]
    return t
```

3. **Need way to find where matches occur in $T$:**

   - String: $a b a a b a_0$
   - BWT: $a_0 a b b a a b b a_1$
   - Reverse BWT: $a_0 a b a a b a_1$

   - Where?
FM Index: fast rank calculations

Is there an fast way to determine which \texttt{bs} precede the \texttt{as} in our range?

\begin{align*}
F & = \texttt{ $a b a a b$} \\
    \texttt{a}_0 & = \texttt{ $a b a a$} \\
    \texttt{a}_1 & = \texttt{a b a $ a$} \\
    \texttt{a}_2 & = \texttt{b a $ a b$} \\
    \texttt{a}_3 & = \texttt{b a a b} \\
L & = \texttt{ $a b a a”} \\
    \texttt{b}_0 & = \texttt{ a $ a b a$} \\
    \texttt{b}_1 & = \texttt{a a b a$}
\end{align*}
FM Index: fast rank calculations

Idea: pre-calculate cumulative # as, bs in L up to every row:
FM Index: fast rank calculations

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

<table>
<thead>
<tr>
<th>L</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
FM Index: fast rank calculations

<table>
<thead>
<tr>
<th>F</th>
<th>L</th>
<th>Tally</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>a</td>
<td>a 1 0</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>b 1 1</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>b 1 2</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a 2 2</td>
</tr>
<tr>
<td>a</td>
<td>$</td>
<td>b 2 2</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>a 3 2</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>b 4 2</td>
</tr>
</tbody>
</table>

$0$ bs up to & including this row

$2$ bs up to & including this row

So $b_0$ and $b_1$ must be in there!
FM Index: fast rank calculations

So $a_2$ and $a_3$ must be in there!

O(1) time; 2 lookups regardless of range size.
**FM Index: fast rank calculations**

<table>
<thead>
<tr>
<th>$F$</th>
<th>$L$</th>
<th>Tally</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$</td>
<td>a</td>
<td>$1 \ 0$</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>$1 \ 1$</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>$1 \ 2$</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>$2 \ 2$</td>
</tr>
<tr>
<td>a</td>
<td>$$</td>
<td>$2 \ 2$</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>$3 \ 2$</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>$4 \ 2$</td>
</tr>
</tbody>
</table>

$Tally$ is $m \times |\Sigma|$ integers

Too big!
Next idea: pre-calculate #a, bs in L up to some rows, e.g. every 5th row. Call pre-calculated rows checkpoints.

<table>
<thead>
<tr>
<th>F</th>
<th>L</th>
<th>Tally</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>1 0</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>3 2</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

Checkpoint 1
Checkpoint 2
Next idea: pre-calculate $\# a$s, $b$s in $L$ up to some rows, e.g. every 5$^{\text{th}}$ row. Call pre-calculated rows *checkpoints*.

To resolve a lookup for a non-checkpoint row, walk to nearest checkpoint. Use tally at that checkpoint, *adjusted for characters we saw along the way*. 

---

**Tally**

<table>
<thead>
<tr>
<th>$F$</th>
<th>$L$</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$</td>
<td>$a$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>$a$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>$$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>$a$</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$b$</td>
<td>$a$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FM Index: fast rank calculations

If checkpoints are $O(1)$ distance apart, lookups are $O(1)$.
FM Index: a few problems

Solved! At the expense of adding checkpoints \(O(m)\) integers to index.

\(O(1)\) with checkpoints

---

(1) This scan is \(O(m)\) work

(2) Ranking takes too much space

```python
def reverseBwt(bw):
    """ Make T from BWT(T) ""
    ranks, tots = rankBwt(bw)
    first = firstCol(tots)
    rowi = 0
    t = "$"
    while bw[rowi] != "$":
        c = bw[rowi]
        t = c + t
        rowi = first[c][0] + ranks[rowi]
    return t
```

Still \(O(m)\) space to store checkpoints, but we control the constant
FM Index: a few problems

Not yet solved: (3) Where are these occurrences in $T$?

If we had suffix array, we could look up offsets...

|$\$$|a|b|a|a|b|a|a|b|a|a|b|a|a|
|a|$\$$|a|b|a|a|b|a|b|
|a|a|b|a|a|b|a|b|
|a|b|a|a|b|a|b|
|b|a|$\$$|a|b|a|a|
|b|a|a|b|a|a|$\$$|a|

$F$ | $L$  | SA
--- | --- | ---
$6$ | $\$$ | 
$5$ | a | $\$$
$2$ | a | b | a | $\$$
$3$ | a | b | a | $\$$
$0$ | a | b | a | a | b | a | $\$$
$4$ | b | a | $\$$
$1$ | b | a | a | b | a | $\$$

Offsets: 0, 3

...but we don't; we are trying to avoid storing $m$ integers
# FM Index: resolving offsets

Idea: store some suffix array elements, but not all

<table>
<thead>
<tr>
<th>F</th>
<th>L</th>
<th>SA' (evens only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>a b a a b a</td>
<td>6</td>
</tr>
<tr>
<td>a</td>
<td>$ a b a a b</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>a b a $ a b</td>
<td></td>
</tr>
<tr>
<td>a b a</td>
<td>$ a b a</td>
<td>$</td>
</tr>
<tr>
<td>a</td>
<td>a b a a b a $</td>
<td>0</td>
</tr>
<tr>
<td>b a</td>
<td>$ a b a</td>
<td>4</td>
</tr>
<tr>
<td>b a</td>
<td>a a b a $ a</td>
<td></td>
</tr>
</tbody>
</table>

Lookup for row 4 succeeds

Lookup for row 3 fails - SA entry was discarded
FM Index: resolving offsets

LF Mapping tells us that “a” at the end of row 3 corresponds to...
...“a” at the beginning of row 2

Row 2 of suffix array = 2
Missing value in row 3 = 2 (row 2’s SA val) + 1 (# steps to row 2) = 3
If saved SA values are O(1) positions apart in T, resolving offset is O(1) time
FM Index: resolving offsets

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

```
F L SA' (every 4th)
$ a b a a b a
a $ a b a a b
a a b a $ a b
a a b a a b
a b a a b a
b a $ a b a
b a a b a
b a a b a $ a
```

Starting here

```
0
4
```

Missing value = 0 (SA elt at destination) + 3 (# steps to destination) = 3
Solved! At the expense of adding some SA values ($O(m)$ integers) to index Call this the “SA sample”

(3) Need a way to find where these occurrences are in $T$:

$$
\begin{array}{ccccccc}
\$ & a & b & a & a & b & a_0 \\
 a_0 & $ & a & b & a & a & b_0 \\
a_1 & a & b & a & $ & a & b_1 \\
a_2 & b & a & $ & a & b & a_1 \\
a_3 & b & a & a & b & a & $ \\
b_0 & a & $ & a & b & a & a_2 \\
b_1 & a & a & b & a & $ & a_3 \\
\end{array}
$$

With SA sample we can do this in $O(1)$ time per occurrence
FM Index

| T | = m

Reversing BWT(T) in FM Index is $O(m)$ time
FM Index

$P = \text{aba}$

$|T| = m$, $|P| = n$

Determining of $P$ occurs in $T$ in FM Index is $O(n)$ time

2 $O(1)$ rank calculations

2 $O(1)$ rank calculations
**FM Index**

Let $a =$ fraction of rows we keep  
Let $b =$ fraction of SA elements we keep

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>SA'</td>
</tr>
<tr>
<td>482</td>
<td>432</td>
<td></td>
</tr>
<tr>
<td>488</td>
<td>439</td>
<td></td>
</tr>
</tbody>
</table>

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:

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

Last column \((L)\): \(m\) characters

SA sample: \(m \cdot a\) integers, \(a\) is fraction of SA elements kept

Checkpoints: \(m \cdot |\Sigma| \cdot b\) integers, \(b\) is fraction of tallies kept

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

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

Checkpoints: 3 billion \(*\) 4 alphabet chars \(*\) 4 bytes / 128 = \(\sim 400\) MB

Total \(\approx 1.5\) GB

\(\sim 0.5\) bytes per input char

(blue indicates what we can adjust by changing \(a\) & \(b\))
FM Index: small memory footprint


FM Index described here is simplified version of what’s described in paper

Also discussed in paper: compressing $\text{BWT}(T)$ for further savings (and selectively decompression portions of it at query time)
FM Index: small memory footprint

SUFFIX TREE

SUFFIX ARRAY

FM INDEX

Su
ffix tree

$ \geq 45 \text{ GB}$

Suffix array

$ \geq 12 \text{ GB}$

FM Index

$ \sim 1.5 \text{ GB}$
## Suffix index bounds

<table>
<thead>
<tr>
<th></th>
<th>Suffix tree</th>
<th>Suffix array</th>
<th>FM Index</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time: Does P occur?</strong></td>
<td>$O(n)$</td>
<td>$O(n \log m)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><strong>Time: Count $k$ occurrences of P</strong></td>
<td>$O(n + k)$</td>
<td>$O(n \log m)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><strong>Time: Report $k$ locations of P</strong></td>
<td>$O(n + k)$</td>
<td>$O(n \log m + k)$</td>
<td>$O(n + k)$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(m)$</td>
<td>$O(m)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>Needs T?</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Bytes per input character</td>
<td>$&gt;15$</td>
<td>$\approx 4$</td>
<td>$\approx 0.5$</td>
</tr>
</tbody>
</table>

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