# Lightweight BWT Construction for Very Large String Collections

Markus J. Bauer, Anthony J. Cox and Giovanna Rosone

Computational Biology Group, Illumina Cambridge Ltd., United Kingdom
Dipartimento di Matematica e Informatica, University of Palermo, Palermo, ITALY

Workshop PRIN, 5-7 September 2011

## Whole human genome sequencing

- Modern DNA sequencing machines produce a lot of data!
  e.g. Illumina HiSeq 2000: $> 40$Gbases of sequence per day (paired 100-mers)
- Whole human genome sequencing: about 3Gbase genome typically sampled to  20 to 30-fold redundancy to ensure adequate coverage of both copies (i.e. each position in the genome sampled 30 times, on average)
- Datasets of 100 Gbases or more are common
- Applications: Comparing of genomes, assembl, alignment, $\cdots$

## The BWT

- The BWT is a reversible transformation that produces a permutation $bwt(v)$ of an input sequence $v$, defined over an ordered alphabet $\Sigma$, so that occurrences of a given symbol tend to occur in clusters in the output sequence.
- Traditionally the major application of the Burrows-Wheeler Transform has been for Data Compression. The BWT represents for instance the heart of the BZIP2 algorithm.
- Today, there are reports of the application of the BWT in bio-informatics, full-text compressed indexes, prediction and entropy estimation, and shape analysis in computer vision, etc.

## The BWT

- The BWT is a reversible transformation that produces a permutation $bwt(v)$ of an input sequence $v$, defined over an ordered alphabet $\Sigma$, so that occurrences of a given symbol tend to occur in clusters in the output sequence.
- Traditionally the major application of the Burrows-Wheeler Transform has been for Data Compression. The BWT represents for instance the heart of the BZIP2 algorithm.
- Today, there are reports of the application of the BWT in bio-informatics, full-text compressed indexes, prediction and entropy estimation, and shape analysis in computer vision, etc.

## The BWT

- The BWT is a reversible transformation that produces a permutation $bwt(v)$ of an input sequence $v$, defined over an ordered alphabet $\Sigma$, so that occurrences of a given symbol tend to occur in clusters in the output sequence.
- Traditionally the major application of the Burrows-Wheeler Transform has been for Data Compression. The BWT represents for instance the heart of the BZIP2 algorithm.
- Today, there are reports of the application of the BWT in bio-informatics, full-text compressed indexes, prediction and entropy estimation, and shape analysis in computer vision, etc.

## How does BWT work?

- BWT takes as input a text $v$, append $\$$ to the end of $v$ ($\$$ is unique and smaller then any other character) and produces:
  - a permutation $bwt(v)$ of the letters of $v\$$.
  - the index $I$, that is useful in order to recover the original word $v$.
- Example: $v = abraca$

- Each row of $M$ is a conjugate of $v\$$ in lexicographic order.
- $bwt(v)$ coincides with the last column $L$ of the BW-matrix $M$.
- The index $I$ is the row of $M$ containing the original sequence followed by $\$$.

$$
\begin{array}{cccccccc}
 & & & & M & & & \\
 & F & & & & & & L \\
 & \downarrow & & & & & & \downarrow \\
0 & \$ & a & b & r & a & c & a \\
1 & a & \$ & a & b & r & a & c \\
I \rightarrow 2 & a & b & r & a & c & a & \$ \\
3 & a & c & a & \$ & a & b & r \\
4 & b & r & a & c & a & \$ & a \\
5 & c & a & \$ & a & b & r & a \\
6 & r & a & c & a & \$ & a & b
\end{array}
$$

## How does BWT work?

- BWT takes as input a text $v$, append $\$$ to the end of $v$ ($\$$ is unique and smaller then any other character) and produces:
  - a permutation $bwt(v)$ of the letters of $v\$$.
  - the index $I$, that is useful in order to recover the original word $v$.
- Example: $v = abraca$

- Each row of $M$ is a conjugate of $v\$$ in lexicographic order.
- $bwt(v)$ coincides with the last column $L$ of the BW-matrix $M$.
- The index $I$ is the row of $M$ containing the original sequence followed by $\$$.

$$M$$

$$\begin{array}{cccccccc}
 & F & & & & & & L \\
 & \downarrow & & & & & & \downarrow \\
0 & \$ & a & b & r & a & c & a \\
1 & a & \$ & a & b & r & a & c \\
I \rightarrow 2 & a & b & r & a & c & a & \$ \\
3 & a & c & a & \$ & a & b & r \\
4 & b & r & a & c & a & \$ & a \\
5 & c & a & \$ & a & b & r & a \\
6 & r & a & c & a & \$ & a & b
\end{array}$$

## How does BWT work?

- BWT takes as input a text $v$, append $\$$ to the end of $v$ ($\$$ is unique and smaller then any other character) and produces:
  - a permutation $bwt(v)$ of the letters of $v\$$.
  - the index $I$, that is useful in order to recover the original word $v$.
- Example: $v = abraca$

- Each row of $M$ is a conjugate of $v\$$ in lexicographic order.
- $bwt(v)$ coincides with the last column $L$ of the BW-matrix $M$.
- The index $I$ is the row of $M$ containing the original sequence followed by $\$$.

$$
\begin{array}{rccccccc}
 & & & & M & & & \\
 & F & & & & & & L \\
 & \downarrow & & & & & & \downarrow \\
0 & \$ & a & b & r & a & c & a \\
1 & a & \$ & a & b & r & a & c \\
I \rightarrow 2 & a & b & r & a & c & a & \$ \\
3 & a & c & a & \$ & a & b & r \\
4 & b & r & a & c & a & \$ & a \\
5 & c & a & \$ & a & b & r & a \\
6 & r & a & c & a & \$ & a & b
\end{array}
$$

## Properties

The following properties hold:

1. For all $i = 0, \ldots, |v|$, $i \neq I$, the character $F[i]$ follows $L[i]$ in the original string;
2. for each character $c$, the $r$-th occurrence of $c$ in $F$ corresponds to the $r$-th occurrence of $c$ in $L$.

Ferragina and Manzini (2000) noticed the following connection:

$$LF[i] = C[L[i]] + rank(L[i], i-1)$$

For instance:
if $i = 5$ then $L[i] = a$ and
$LF[5] = C[a] + rank(a, 4) = 1 + 2 = 3$

$$
M \qquad
\begin{array}{rccccccc}
 & F & & & & & & L \\
 & \downarrow & & & & & & \downarrow \\
0 & \$ & a & b & r & a & c & a \\
1 & a & \$ & a & b & r & a & c \\
I \rightarrow 2 & a & b & r & a & c & a & \$ \\
3 & a & c & a & \$ & a & b & r \\
4 & b & r & a & c & a & \$ & a \\
5 & c & a & \$ & a & b & r & a \\
6 & r & a & c & a & \$ & a & b
\end{array}
$$

## Properties

The following properties hold:

1. For all $i = 0, \ldots, |v|$, $i \neq I$, the character $F[i]$ follows $L[i]$ in the original string;
2. for each character $c$, the $r$-th occurrence of $c$ in $F$ corresponds to the $r$-th occurrence of $c$ in $L$.

Ferragina and Manzini (2000) noticed the following connection:

$$LF[i] = C[L[i]] + rank(L[i], i-1)$$

For instance:
if $i = 5$ then $L[i] = a$ and
$LF[5] = C[a] + rank(a, 4) = 1 + 2 = 3$

$$\begin{array}{rccccccc}
 & & & & M & & & \\
 & F & & & & & & L \\
 & \downarrow & & & & & & \downarrow \\
0 & \$ & a & b & r & a & c & a \\
1 & a & \$ & a & b & r & a & c \\
I \to 2 & a & b & r & a & c & a & \$ \\
3 & a & c & a & \$ & a & b & r \\
4 & b & r & a & c & a & \$ & a \\
5 & c & a & \$ & a & b & r & a \\
6 & r & a & c & a & \$ & a & b
\end{array}$$

## Properties

The following properties hold:

1. For all $i = 0, \ldots, |v|$, $i \neq I$, the character $F[i]$ follows $L[i]$ in the original string;
2. for each character $c$, the $r$-th occurrence of $c$ in $F$ corresponds to the $r$-th occurrence of $c$ in $L$.

Ferragina and Manzini (2000) noticed the following connection:

$$LF[i] = C[L[i]] + rank(L[i], i-1)$$

For instance:
if $i = 5$ then $L[i] = a$ and
$LF[5] = C[a] + rank(a, 4) = 1 + 2 = 3$

$$M$$

$$\begin{array}{ccccccccc}
 & & F & & & & & & L \\
 & & \downarrow & & & & & & \downarrow \\
 & 0 & \$ & a & b & r & a & c & a \\
 & 1 & a & \$ & a & b & r & a & c \\
I \rightarrow & 2 & a & b & r & a & c & a & \$ \\
 & 3 & a & c & a & \$ & a & b & r \\
 & 4 & b & r & a & c & a & \$ & a \\
 & 5 & c & a & \$ & a & b & r & a \\
 & 6 & r & a & c & a & \$ & a & b
\end{array}$$

## Properties

The following properties hold:

1. For all $i = 0, \ldots, |v|$, $i \neq I$, the character $F[i]$ follows $L[i]$ in the original string;
2. for each character $c$, the $r$-th occurrence of $c$ in $F$ corresponds to the $r$-th occurrence of $c$ in $L$.

Ferragina and Manzini (2000) noticed the following connection:

$$LF[i] = C[L[i]] + rank(L[i], i-1)$$

For instance:
if $i = 5$ then $L[i] = a$ and
$LF[5] = C[a] + rank(a, 4) = 1 + 2 = 3$

$$M$$

$$\begin{array}{cc|ccccccc}
 & & F & & & & & & L \\
 & & \downarrow & & & & & & \downarrow \\
 & 0 & \mathbf{\$} & a & b & r & a & c & a \\
 & 1 & \mathbf{a} & \mathbf{\$} & a & b & r & a & c \\
I \rightarrow & 2 & \mathbf{a} & \mathbf{b} & \mathbf{r} & \mathbf{a} & \mathbf{c} & \mathbf{a} & \mathbf{\$} \\
 & 3 & \mathbf{a} & \mathbf{c} & \mathbf{a} & \mathbf{\$} & a & b & r \\
 & 4 & \mathbf{b} & \mathbf{r} & \mathbf{a} & \mathbf{c} & \mathbf{a} & \mathbf{\$} & a \\
 & 5 & \mathbf{c} & \mathbf{a} & \mathbf{\$} & a & b & r & a \\
 & 6 & \mathbf{r} & \mathbf{a} & \mathbf{c} & \mathbf{a} & \mathbf{\$} & a & b
\end{array}$$

## The BWT in bioinformatics

- BWT-based text indexes are the core of popular mapping programs
  1. Bowtie (Langmead et al.,Genome Biology 2009)
  2. BWA (Li and Durbin, Bioinformatics 2009, 2010)
  3. SOAP2 (Li et al., Bioinformatics 2009)
- Create index from reference genome (e.g. human)
  create once, use many times
- Simpson and Durbin, Bioinformatics 2010: FM-index of a set of DNA sequences for overlap detection stage of de novo assembly
  See also Valimaki et al., CPM 2010

## The BWT in bioinformatics

- BWT-based text indexes are the core of popular mapping programs
  1. Bowtie (Langmead et al.,Genome Biology 2009)
  2. BWA (Li and Durbin, Bioinformatics 2009, 2010)
  3. SOAP2 (Li et al., Bioinformatics 2009)
- Create index from reference genome (e.g. human)
  create once, use many times
- Simpson and Durbin, Bioinformatics 2010: FM-index of a set of DNA sequences for overlap detection stage of de novo assembly
  See also Valimaki et al., CPM 2010

## The BWT in bioinformatics

- BWT-based text indexes are the core of popular mapping programs
  1. Bowtie (Langmead et al.,Genome Biology 2009)
  2. BWA (Li and Durbin, Bioinformatics 2009, 2010)
  3. SOAP2 (Li et al., Bioinformatics 2009)
- Create index from reference genome (e.g. human)
  create once, use many times
- Simpson and Durbin, Bioinformatics 2010: FM-index of a set of DNA sequences for overlap detection stage of de novo assembly
  See also Valimaki et al., CPM 2010

## BWT of a collection of strings

- BWT extended to set of strings by S. Mantaci et al. (CPM 2005, TCS 2007) by using a different ordering of the conjugates of the strings.
- original BWT of concatenated strings
  - Straightforward to compute BWT from suffix array.
  - Lots of work on efficient linear time SA generation methods.
  - But: need to hold SA in RAM (Simpson et al. estimate 700Gbytes RAM for SA of 60 Gbases of data)
  - Other options:
    - Siren, SPIRE 2009: divide collection into batches, compute BWT of each then merge
    - Ferragina et al., Latin 2010: partition string $T$ into blocks $T_r, \ldots, T_1$, create SA of each in turn

## BWT of a collection of strings

- BWT extended to set of strings by S. Mantaci et al. (CPM 2005, TCS 2007) by using a different ordering of the conjugates of the strings.
- original BWT of concatenated strings
  - Straightforward to compute BWT from suffix array.
  - Lots of work on efficient linear time SA generation methods.
  - **But**: need to hold SA in RAM (Simpson et al. estimate 700Gbytes RAM for SA of 60 Gbases of data)
  - Other options:
    - Siren, SPIRE 2009: divide collection into batches, compute BWT of each then merge
    - Ferragina et al., Latin 2010: partition string $T$ into blocks $T_r \cdots T_1$, create SA of each in turn

## BWT of a collection of strings

- BWT extended to set of strings by S. Mantaci et al. (CPM 2005, TCS 2007) by using a different ordering of the conjugates of the strings.
- original BWT of concatenated strings
  - Straightforward to compute BWT from suffix array.
  - Lots of work on efficient linear time SA generation methods.
  - **But**: need to hold SA in RAM (Simpson et al. estimate 700Gbytes RAM for SA of 60 Gbases of data)
  - Other options:
    - Siren, SPIRE 2009: divide collection into batches, compute BWT of each then merge
    - Ferragina et al., Latin 2010: partition string $T$ into blocks $T_r \cdots T_1$, create SA of each in turn

## BWT of a collection of strings

- BWT extended to set of strings by S. Mantaci et al. (CPM 2005, TCS 2007) by using a different ordering of the conjugates of the strings.
- original BWT of concatenated strings
  - Straightforward to compute BWT from suffix array.
  - Lots of work on efficient linear time SA generation methods.
  - **But**: need to hold SA in RAM (Simpson et al. estimate 700Gbytes RAM for SA of 60 Gbases of data)
  - Other options:
    - Siren, SPIRE 2009: divide collection into batches, compute BWT of each then merge
    - Ferragina et al., Latin 2010: partition string $T$ into blocks $T_r \cdots T_1$, create SA of each in turn

## BWT of a collection of strings

- BWT extended to set of strings by S. Mantaci et al. (CPM 2005, TCS 2007) by using a different ordering of the conjugates of the strings.
- original BWT of concatenated strings
  - Straightforward to compute BWT from suffix array.
  - Lots of work on efficient linear time SA generation methods.
  - **But**: need to hold SA in RAM (Simpson et al. estimate 700Gbytes RAM for SA of 60 Gbases of data)
  - Other options:
    - Siren, SPIRE 2009: divide collection into batches, compute BWT of each then merge
    - Ferragina et al., Latin 2010: partition string $T$ into blocks $T_r \cdots T_1$, create SA of each in turn

## Observations

Let S be a collection of $m$ strings of length $k$ on an alphabet of $\sigma$ letters.
Our algorithm computes the BWT of S

- without concatenating the strings belonging to S and without needing to compute their suffix array.
- incrementally via $k$ iterations. At each of the iterations $j = 1, 2, \ldots, k$, the algorithm computes a partial BWT string $\text{bwt}_j(\text{S})$ by inserting the symbols preceding the $j$-suffixes of S at their correct positions into $\text{bwt}_{j-1}(\text{S})$. Each iteration $j$ simulates the insertion of the $j$-suffixes in the suffix array.
- The string $\text{bwt}_j(\text{S})$ is a 'partial BWT' in the sense that the addition of $m$ end markers in their correct positions would make it the BWT of the collection $\{S_1[k-j-1,k], S_2[k-j-1,k], \ldots, S_m[k-j-1,k]\}$.
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

## Observations

Let S be a collection of $m$ strings of length $k$ on an alphabet of $\sigma$ letters. Our algorithm computes the BWT of S

- without concatenating the strings belonging to S and without needing to compute their suffix array.
- incrementally via $k$ iterations. At each of the iterations $j = 1, 2, \ldots, k$, the algorithm computes a partial BWT string $\mathrm{bwt}_j(\mathrm{S})$ by inserting the symbols preceding the $j$-suffixes of S at their correct positions into $\mathrm{bwt}_{j-1}(\mathrm{S})$. Each iteration $j$ simulates the insertion of the $j$-suffixes in the suffix array.
- The string $\mathrm{bwt}_j(\mathrm{S})$ is a 'partial BWT' in the sense that the addition of $m$ end markers in their correct positions would make it the BWT of the collection $\{S_1[k-j-1,k], S_2[k-j-1,k], \ldots, S_m[k-j-1,k]\}$.
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

## Observations

Let S be a collection of $m$ strings of length $k$ on an alphabet of $\sigma$ letters. Our algorithm computes the BWT of S

- without concatenating the strings belonging to S and without needing to compute their suffix array.
- incrementally via $k$ iterations. At each of the iterations $j = 1, 2, \ldots, k$, the algorithm computes a partial BWT string $\mathrm{bwt}_j(\mathrm{S})$ by inserting the symbols preceding the $j$-suffixes of S at their correct positions into $\mathrm{bwt}_{j-1}(\mathrm{S})$. Each iteration $j$ simulates the insertion of the $j$-suffixes in the suffix array.
- The string $\mathrm{bwt}_j(\mathrm{S})$ is a 'partial BWT' in the sense that the addition of $m$ end markers in their correct positions would make it the BWT of the collection $\{S_1[k-j-1,k], S_2[k-j-1,k], \ldots, S_m[k-j-1,k]\}$.
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

## Observations

Let S be a collection of $m$ strings of length $k$ on an alphabet of $\sigma$ letters. Our algorithm computes the BWT of S

- without concatenating the strings belonging to S and without needing to compute their suffix array.
- incrementally via $k$ iterations. At each of the iterations $j = 1, 2, \ldots, k$, the algorithm computes a partial BWT string $\mathrm{bwt}_j(\mathrm{S})$ by inserting the symbols preceding the $j$-suffixes of S at their correct positions into $\mathrm{bwt}_{j-1}(\mathrm{S})$. Each iteration $j$ simulates the insertion of the $j$-suffixes in the suffix array.
- The string $\mathrm{bwt}_j(\mathrm{S})$ is a 'partial BWT' in the sense that the addition of $m$ end markers in their correct positions would make it the BWT of the collection $\{S_1[k-j-1,k], S_2[k-j-1,k], \ldots, S_m[k-j-1,k]\}$.
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

## Observations

Let S be a collection of $m$ strings of length $k$ on an alphabet of $\sigma$ letters. Our algorithm computes the BWT of S

- without concatenating the strings belonging to S and without needing to compute their suffix array.
- incrementally via $k$ iterations. At each of the iterations $j = 1, 2, \ldots, k$, the algorithm computes a partial BWT string $\text{bwt}_j(\text{S})$ by inserting the symbols preceding the $j$-suffixes of S at their correct positions into $\text{bwt}_{j-1}(\text{S})$. Each iteration $j$ simulates the insertion of the $j$-suffixes in the suffix array.
- The string $\text{bwt}_j(\text{S})$ is a 'partial BWT' in the sense that the addition of $m$ end markers in their correct positions would make it the BWT of the collection $\{S_1[k-j-1, k], S_2[k-j-1, k], \ldots, S_m[k-j-1, k]\}$.
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

## Observations

Let S be a collection of $m$ strings of length $k$ on an alphabet of $\sigma$ letters. Our algorithm computes the BWT of S

- without concatenating the strings belonging to S and without needing to compute their suffix array.
- incrementally via $k$ iterations. At each of the iterations $j = 1, 2, \ldots, k$, the algorithm computes a partial BWT string $\text{bwt}_j(\text{S})$ by inserting the symbols preceding the $j$-suffixes of S at their correct positions into $\text{bwt}_{j-1}(\text{S})$. Each iteration $j$ simulates the insertion of the $j$-suffixes in the suffix array.
- The string $\text{bwt}_j(\text{S})$ is a 'partial BWT' in the sense that the addition of $m$ end markers in their correct positions would make it the BWT of the collection $\{S_1[k-j-1,k], S_2[k-j-1,k], \ldots, S_m[k-j-1,k]\}$.
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

## Example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $S_1$ | | | | | | | | $\$_1$ |
| $S_2$ | | | | | | | | $\$_2$ |
| $S_3$ | | | | | | | | $\$_3$ |

We suppose that $\$_1 < \$_2 < \$_3 < A < C < G < T$.
$j$-suffix of $S_i$ is the last $j$ non-\$ symbols of that string and 0-suffix of $S_i$ is $\$_i$.
At stage $j$, insert the characters associated with the $j$-suffixes into the partial BWT.

## Example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

|  | $0$ | $1$ | $2$ | $3$ | $4$ | $5$ | $6$ | $7$ |
|---|---|---|---|---|---|---|---|---|
| $S_1$ |  |  |  |  |  |  | $C$ | $\$_1$ |
| $S_2$ |  |  |  |  |  |  | $C$ | $\$_2$ |
| $S_3$ |  |  |  |  |  |  | $T$ | $\$_3$ |

We suppose that $\$_1 < \$_2 < \$_3 < A < C < G < T$.
$j$-suffix of $S_i$ is the last $j$ non-\$ symbols of that string and 0-suffix of $S_i$ is $\$_i$.
At stage $j$, insert the characters associated with the $j$-suffixes into the partial BWT.

## Example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

| | $0$ | $1$ | $2$ | $3$ | $4$ | $5$ | $6$ | $7$ |
|---|---|---|---|---|---|---|---|---|
| $S_1$ | | | | | | $A$ | $C$ | $\$_1$ |
| $S_2$ | | | | | | $T$ | $C$ | $\$_2$ |
| $S_3$ | | | | | | $T$ | $T$ | $\$_3$ |

We suppose that $\$_1 < \$_2 < \$_3 < A < C < G < T$.
$j$-suffix of $S_i$ is the last $j$ non-\$ symbols of that string and 0-suffix of $S_i$ is $\$_i$.
At stage $j$, insert the characters associated with the $j$-suffixes into the partial BWT.

## Example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $S_1$ | | | | | $A$ | $A$ | $C$ | $\$_1$ |
| $S_2$ | | | | | $C$ | $T$ | $C$ | $\$_2$ |
| $S_3$ | | | | | $C$ | $T$ | $T$ | $\$_3$ |

We suppose that $\$_1 < \$_2 < \$_3 < A < C < G < T$.
$j$-suffix of $S_i$ is the last $j$ non-\$ symbols of that string and 0-suffix of $S_i$ is $\$_i$.
At stage $j$, insert the characters associated with the $j$-suffixes into the partial BWT.

## Example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $S_1$ | | | | $C$ | $A$ | $A$ | $C$ | $\$_1$ |
| $S_2$ | | | | $G$ | $C$ | $T$ | $C$ | $\$_2$ |
| $S_3$ | | | | $G$ | $C$ | $T$ | $T$ | $\$_3$ |

We suppose that $\$_1 < \$_2 < \$_3 < A < C < G < T$.
$j$-suffix of $S_i$ is the last $j$ non-\$ symbols of that string and 0-suffix of $S_i$ is $\$_i$.
At stage $j$, insert the characters associated with the $j$-suffixes into the partial BWT.

## Example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $S_1$ | | | $C$ | $C$ | $A$ | $A$ | $C$ | $\$_1$ |
| $S_2$ | | | $A$ | $G$ | $C$ | $T$ | $C$ | $\$_2$ |
| $S_3$ | | | $C$ | $G$ | $C$ | $T$ | $T$ | $\$_3$ |

We suppose that $\$_1 < \$_2 < \$_3 < A < C < G < T$.
$j$-suffix of $S_i$ is the last $j$ non-\$ symbols of that string and 0-suffix of $S_i$ is $\$_i$.
At stage $j$, insert the characters associated with the $j$-suffixes into the partial BWT.

## Example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $S_1$ | | $G$ | $C$ | $C$ | $A$ | $A$ | $C$ | $\$_1$ |
| $S_2$ | | $G$ | $A$ | $G$ | $C$ | $T$ | $C$ | $\$_2$ |
| $S_3$ | | $T$ | $C$ | $G$ | $C$ | $T$ | $T$ | $\$_3$ |

We suppose that $\$_1 < \$_2 < \$_3 < A < C < G < T$.
$j$-suffix of $S_i$ is the last $j$ non-\$ symbols of that string and 0-suffix of $S_i$ is $\$_i$.
At stage $j$, insert the characters associated with the $j$-suffixes into the partial BWT.

## Example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

| | $0$ | $1$ | $2$ | $3$ | $4$ | $5$ | $6$ | $7$ |
|---|---|---|---|---|---|---|---|---|
| $S_1$ | $T$ | $G$ | $C$ | $C$ | $A$ | $A$ | $C$ | $\$_1$ |
| $S_2$ | $A$ | $G$ | $A$ | $G$ | $C$ | $T$ | $C$ | $\$_2$ |
| $S_3$ | $G$ | $T$ | $C$ | $G$ | $C$ | $T$ | $T$ | $\$_3$ |

We suppose that $\$_1 < \$_2 < \$_3 < A < C < G < T$.
$j$-suffix of $S_i$ is the last $j$ non-\$ symbols of that string and 0-suffix of $S_i$ is $\$_i$.
At stage $j$, insert the characters associated with the $j$-suffixes into the partial BWT.

## Example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $S_1$ | $T$ | $G$ | $C$ | $C$ | $A$ | $A$ | $C$ | $\$_1$ |
| $S_2$ | $A$ | $G$ | $A$ | $G$ | $C$ | $T$ | $C$ | $\$_2$ |
| $S_3$ | $G$ | $T$ | $C$ | $G$ | $C$ | $T$ | $T$ | $\$_3$ |

We suppose that $\$_1 < \$_2 < \$_3 < A < C < G < T$.

$j$-suffix of $S_i$ is the last $j$ non-\$ symbols of that string and 0-suffix of $S_i$ is $\$_i$.

At stage $j$, insert the characters associated with the $j$-suffixes into the partial BWT.

## Iteration 0

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $S_1$ | | | | | | | | $\$_1$ |
| $S_2$ | | | | | | | | $\$_2$ |
| $S_3$ | | | | | | | | $\$_3$ |

We obtain:

| |
|---|
| $C$ |
| $C$ |
| $T$ |

## Iteration 0

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $S_1$ | | | | | | | $C$ | $\$_1$ |
| $S_2$ | | | | | | | $C$ | $\$_2$ |
| $S_3$ | | | | | | | $T$ | $\$_3$ |

We obtain:

| |
|---|
| $C$ |
| $C$ |
| $T$ |

## Iteration 0

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

|  | $0$ | $1$ | $2$ | $3$ | $4$ | $5$ | $6$ | $7$ |
|---|---|---|---|---|---|---|---|---|
| $S_1$ |  |  |  |  |  |  | $C$ | $\$_1$ |
| $S_2$ |  |  |  |  |  |  | $C$ | $\$_2$ |
| $S_3$ |  |  |  |  |  |  | $T$ | $\$_3$ |

We obtain:

|  |
|---|
| $C$ |
| $C$ |
| $T$ |

## Iteration 0

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $S_1$ | | | | | | $A$ | $C$ | $\$_1$ |
| $S_2$ | | | | | | $T$ | $C$ | $\$_2$ |
| $S_3$ | | | | | | $T$ | $T$ | $\$_3$ |

We obtain:

| |
|---|
| $C$ |
| $C$ |
| $T$ |

## Iteration 0

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $S_1$ | | | | | | $A$ | $C$ | $\$_1$ |
| $S_2$ | | | | | | $T$ | $C$ | $\$_2$ |
| $S_3$ | | | | | | $T$ | $T$ | $\$_3$ |

We obtain:

| |
|---|
| $C$ |
| $C$ |
| $T$ |

## Iteration 0

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

| | $0$ | $1$ | $2$ | $3$ | $4$ | $5$ | $6$ | $7$ |
|---|---|---|---|---|---|---|---|---|
| $S_1$ | | | | | | $A$ | $C$ | $\$_1$ |
| $S_2$ | | | | | | $T$ | $C$ | $\$_2$ |
| $S_3$ | | | | | | $T$ | $T$ | $\$_3$ |

We obtain:

| |
|---|
| $C$ |
| $C$ |
| $T$ |

## Iteration 0

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

| | $0$ | $1$ | $2$ | $3$ | $4$ | $5$ | $6$ | $7$ |
|---|---|---|---|---|---|---|---|---|
| $S_1$ | | | | | | $A$ | $C$ | $\$_1$ |
| $S_2$ | | | | | | $T$ | $C$ | $\$_2$ |
| $S_3$ | | | | | | $T$ | $T$ | $\$_3$ |

We obtain:

| |
|---|
| $C$ |
| $C$ |
| $T$ |

## Iteration 0

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

|  | $0$ | $1$ | $2$ | $3$ | $4$ | $5$ | $6$ | $7$ |
|---|---|---|---|---|---|---|---|---|
| $S_1$ |  |  |  |  |  | $A$ | $C$ | $\$_1$ |
| $S_2$ |  |  |  |  |  | $T$ | $C$ | $\$_2$ |
| $S_3$ |  |  |  |  |  | $T$ | $T$ | $\$_3$ |

We obtain:

|  |
|---|
| $C$ |
| $C$ |
| $T$ |

## Iteration 0

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

|  | $0$ | $1$ | $2$ | $3$ | $4$ | $5$ | $6$ | $7$ |
|---|---|---|---|---|---|---|---|---|
| $S_1$ |  |  |  |  |  | $A$ | $C$ | $\$_1$ |
| $S_2$ |  |  |  |  |  | $T$ | $C$ | $\$_2$ |
| $S_3$ |  |  |  |  |  | $T$ | $T$ | $\$_3$ |

We obtain:

|  |
|---|
| $C$ |
| $C$ |
| $T$ |

## Observation

$$LF[i] = C[L[i]] + rank(L[i], i-1)$$

We can think of $\text{bwt}_j(\text{S})$ as being partitioned into $\sigma + 1$ strings $B_j(0), B_j(1), \ldots, B_j(\sigma)$, with the symbols in $B_j(h)$ being those that are associated with the suffixes of S that are of length $j$ or less and begin with $c_0 = \$$ and $c_h \in \Sigma$, for $h = 1, \ldots, \sigma$.

| | | $F$ ↓ | | | | | | $L$ ↓ |
|---|---|---|---|---|---|---|---|---|
| $B(0)$ | 0 | **$** | a | b | r | a | c | a |
| $B(1)$ | 1 | **a** | **$** | a | b | r | a | c |
| | 2 | **a** | **b** | **r** | **a** | **c** | **a** | **$** |
| | 3 | **a** | **c** | **a** | **$** | a | b | r |
| $B(2)$ | 4 | **b** | **r** | **a** | **c** | **a** | **$** | a |
| $B(3)$ | 5 | **c** | **a** | **$** | a | b | r | a |
| $B(4)$ | 6 | **r** | **a** | **c** | **a** | **$** | a | b |

We do not need the array $C$. We only need the rank function.

We note that $B_j(0)$ is constant for all $j$ and, at each iteration $j$, we store $B_j(h)$ in $\sigma + 1$ external files that are sequentially read one-by-one.

## Observation

$$LF[i] = C[L[i]] + rank(L[i], i-1)$$

We can think of bwt$_j$(S) as being partitioned into $\sigma + 1$ strings $B_j(0), B_j(1), \ldots, B_j(\sigma)$, with the symbols in $B_j(h)$ being those that are associated with the suffixes of S that are of length $j$ or less and begin with $c_0 = \$$ and $c_h \in \Sigma$, for $h = 1, \ldots, \sigma$.

| | | $F \downarrow$ | | | | | | $L \downarrow$ |
|---|---|---|---|---|---|---|---|---|
| $B(0)$ | 0 | **$\$$** | *a* | *b* | *r* | *a* | *c* | *a* |
| $B(1)$ | 1 | **a** | **$\$$** | *a* | *b* | *r* | *a* | *c* |
| | 2 | **a** | **b** | **r** | **a** | **c** | **a** | **$\$$** |
| | 3 | **a** | **c** | **a** | **$\$$** | *a* | *b* | *r* |
| $B(2)$ | 4 | **b** | **r** | **a** | **c** | **a** | **$\$$** | *a* |
| $B(3)$ | 5 | **c** | **a** | **$\$$** | *a* | *b* | *r* | *a* |
| $B(4)$ | 6 | **r** | **a** | **c** | **a** | **$\$$** | *a* | *b* |

We do not need the array $C$. We only need the rank function.

We note that $B_j(0)$ is constant for all $j$ and, at each iteration $j$, we store $B_j(h)$ in $\sigma + 1$ external files that are sequentially read one-by-one.

## Observation

$$LF[i] = C[L[i]] + rank(L[i], i-1)$$

We can think of bwt$_j$(S) as being partitioned into $\sigma + 1$ strings $B_j(0), B_j(1), \ldots, B_j(\sigma)$, with the symbols in $B_j(h)$ being those that are associated with the suffixes of S that are of length $j$ or less and begin with $c_0 = \$$ and $c_h \in \Sigma$, for $h = 1, \ldots, \sigma$.

| | | $F$ $\downarrow$ | | | | | | $L$ $\downarrow$ |
|---|---|---|---|---|---|---|---|---|
| $B(0)$ | 0 | **$** | $a$ | $b$ | $r$ | $a$ | $c$ | $a$ |
| $B(1)$ | 1 | **a** | **$** | $a$ | $b$ | $r$ | $a$ | $c$ |
| | 2 | **a** | **b** | **r** | **a** | **c** | **a** | **$** |
| | 3 | **a** | **c** | **a** | **$** | $a$ | $b$ | $r$ |
| $B(2)$ | 4 | **b** | **r** | **a** | **c** | **a** | **$** | $a$ |
| $B(3)$ | 5 | **c** | **a** | **$** | $a$ | $b$ | $r$ | $a$ |
| $B(4)$ | 6 | **r** | **a** | **c** | **a** | **$** | $a$ | $b$ |

We do not need the array $C$. We only need the rank function.

We note that $B_j(0)$ is constant for all $j$ and, at each iteration $j$, we store $B_j(h)$ in $\sigma + 1$ external files that are sequentially read one-by-one.

## Observation

$$LF[i] = C[L[i]] + rank(L[i], i-1)$$

We can think of bwt$_j$(S) as being partitioned into $\sigma + 1$ strings $B_j(0), B_j(1), \ldots, B_j(\sigma)$, with the symbols in $B_j(h)$ being those that are associated with the suffixes of S that are of length $j$ or less and begin with $c_0 = \$$ and $c_h \in \Sigma$, for $h = 1, \ldots, \sigma$.

| | | $F$ $\downarrow$ | | | | | | $L$ $\downarrow$ |
|---|---|---|---|---|---|---|---|---|
| $B(0)$ | 0 | **\$** | a | b | r | a | c | a |
| $B(1)$ | 1 | **a** | **\$** | a | b | r | a | c |
| | 2 | **a** | **b** | **r** | **a** | **c** | **a** | **\$** |
| | 3 | **a** | **c** | **a** | **\$** | a | b | r |
| $B(2)$ | 4 | **b** | **r** | **a** | **c** | **a** | **\$** | a |
| $B(3)$ | 5 | **c** | **a** | **\$** | a | b | r | a |
| $B(4)$ | 6 | **r** | **a** | **c** | **a** | **\$** | a | b |

We do not need the array $C$. We only need the rank function.

We note that $B_j(0)$ is constant for all $j$ and, at each iteration $j$, we store $B_j(h)$ in $\sigma + 1$ external files that are sequentially read one-by-one.

## Looking in detail at iteration 6

| | $B_5(0)$ | Associated Suffixes |
|---|---|---|
| 0 | $C$ | $\$_1$ |
| 1 | $C$ | $\$_2$ |
| 2 | $T$ | $\$_3$ |

| | $B_5(1)$ | Associated Suffixes |
|---|---|---|
| 0 | $C$ | $AAC\$_1$ |
| 1 | $A$ | $AC\$_1$ |
| 2 | **G** | **AGCTC**$\$_2$ |

| | $B_5(2)$ | Associated Suffixes |
|---|---|---|
| 0 | $A$ | $C\$_1$ |
| 1 | $T$ | $C\$_2$ |
| 2 | $C$ | $CAAC\$_1$ |
| 3 | **G** | **CCAAC**$\$_1$ |
| 4 | **T** | **CGCTT**$\$_3$ |
| 5 | $G$ | $CTC\$_2$ |
| 6 | $G$ | $CTT\$_3$ |

| | $B_5(3)$ | Associated Suffixes |
|---|---|---|
| 0 | $A$ | $GCTC\$_2$ |
| 1 | $C$ | $GCTT\$_3$ |

| | $B_5(4)$ | Associated Suffixes |
|---|---|---|
| 0 | $T$ | $T\$_3$ |
| 1 | $C$ | $TC\$_2$ |
| 2 | $C$ | $TT\$_3$ |

$TG$**CCAAC**$\$_1$,
$AG$**AGCTC**$\$_2$,
$GT$**CGCTT**$\$_3$.

$P_5(0) = []$, $N_5(0) = []$(empty array)
$P_5(1) = [2]$, $N_5(1) = [2]$
$P_5(2) = [3, 4]$, $N_5(2) = [1, 3]$
$P_5(3) = []$, $N_5(3) = []$
$P_5(4) = []$, $N_5(4) = []$

$\Downarrow$

For $h = 0, 3, 4$: nothing
For $h = 1$:
$rank(G, 2) = 0(sequence = 2)$
For $h = 2$:
$rank(G, 3) = 1(sequence = 1)$
$rank(T, 4) = 2(sequence = 3)$

$T$**GCCAAC**$\$_1$,
$A$**GAGCTC**$\$_2$,
$G$**TCGCTT**$\$_3$.

$\Downarrow$

$P_6(0) = []$, $N_6(0) = []$
$P_6(1) = []$, $N_6(1) = []$
$P_6(2) = []$, $N_6(2) = []$
$P_6(3) = [0, 1]$ and $N_6(3) = [2, 1]$
$P_6(4) = [2]$ and $N_6(4) = [3]$

| | $B_6(0)$ | Associated Suffixes |
|---|---|---|
| 0 | $C$ | $\$_1$ |
| 1 | $C$ | $\$_2$ |
| 2 | $T$ | $\$_3$ |

| | $B_6(1)$ | Associated Suffixes |
|---|---|---|
| 0 | $C$ | $AAC\$_1$ |
| 1 | $A$ | $AC\$_1$ |
| 2 | $G$ | $AGCTC\$_2$ |

| | $B_6(2)$ | Associated Suffixes |
|---|---|---|
| 0 | $A$ | $C\$_1$ |
| 1 | $T$ | $C\$_2$ |
| 2 | $C$ | $CAAC\$_1$ |
| 3 | $G$ | $CCAAC\$_1$ |
| 4 | $T$ | $CGCTT\$_3$ |
| 5 | $G$ | $CTC\$_2$ |
| 6 | $G$ | $CTT\$_3$ |

| | $B_6(3)$ | Associated Suffixes |
|---|---|---|
| 0 | A | **GAGCTC**$\$_2$ |
| 1 | T | **GCCAAC**$\$_1$ |
| 2 | $A$ | $GCTC\$_2$ |
| 3 | $C$ | $GCTT\$_3$ |

| | $B_6(4)$ | Associated Suffixes |
|---|---|---|
| 0 | $T$ | $T\$_3$ |
| 1 | $C$ | $TC\$_2$ |
| 2 | G | **TCGCTT**$\$_3$ |
| 3 | $C$ | $TT\$_3$ |

Position of $GCCAAC\$_1$ in $G$ segment = # of $G$ before $CCAAC\$_1$ in partial BWT = # of $G$ in \$-segment + # of $G$ in $A$-segment + # of $G$ before $CCAAC\$_1$ in $C$-segment

## Looking in detail at iteration 6

| | $B_5(0)$ | Associated Suffixes |
|---|---|---|
| 0 | $C$ | $\$_1$ |
| 1 | $C$ | $\$_2$ |
| 2 | $T$ | $\$_3$ |

| | $B_5(1)$ | Associated Suffixes |
|---|---|---|
| 0 | $C$ | $AAC\$_1$ |
| 1 | $A$ | $AC\$_1$ |
| 2 | **G** | **AGCTC**$\$_2$ |

| | $B_5(2)$ | Associated Suffixes |
|---|---|---|
| 0 | $A$ | $C\$_1$ |
| 1 | $T$ | $C\$_2$ |
| 2 | $C$ | $CAAC\$_1$ |
| 3 | **G** | **CCAAC**$\$_1$ |
| 4 | **T** | **CGCTT**$\$_3$ |
| 5 | $G$ | $CTC\$_2$ |
| 6 | $G$ | $CTT\$_3$ |

| | $B_5(3)$ | Associated Suffixes |
|---|---|---|
| 0 | $A$ | $GCTC\$_2$ |
| 1 | $C$ | $GCTT\$_3$ |

| | $B_5(4)$ | Associated Suffixes |
|---|---|---|
| 0 | $T$ | $T\$_3$ |
| 1 | $C$ | $TC\$_2$ |
| 2 | $C$ | $TT\$_3$ |

$TG$**CCAAC**$\$_1$,
$AG$**AGCTC**$\$_2$,
$GT$**CGCTT**$\$_3$.

$P_5(0) = [], N_5(0) = []$(empty array)
$P_5(1) = [2], N_5(1) = [2]$
$P_5(2) = [3, 4], N_5(2) = [1, 3]$
$P_5(3) = [], N_5(3) = []$
$P_5(4) = [], N_5(4) = []$

$\Downarrow$

For $h = 0, 3, 4$: nothing
For $h = 1$:
$rank(G, 2) = 0(sequence = 2)$
For $h = 2$:
$rank(G, 3) = 1(sequence = 1)$
$rank(T, 4) = 2(sequence = 3)$

$T$**GCCAAC**$\$_1$,
$A$**GAGCTC**$\$_2$,
$G$**TCGCTT**$\$_3$.

$\Downarrow$

$P_6(0) = [], N_6(0) = []$
$P_6(1) = [], N_6(1) = []$
$P_6(2) = [], N_6(2) = []$
$P_6(3) = [0, 1]$ and $N_6(3) = [2, 1]$
$P_6(4) = [2]$ and $N_6(4) = [3]$

| | $B_6(0)$ | Associated Suffixes |
|---|---|---|
| 0 | $C$ | $\$_1$ |
| 1 | $C$ | $\$_2$ |
| 2 | $T$ | $\$_3$ |

| | $B_6(1)$ | Associated Suffixes |
|---|---|---|
| 0 | $C$ | $AAC\$_1$ |
| 1 | $A$ | $AC\$_1$ |
| 2 | $G$ | $AGCTC\$_2$ |

| | $B_6(2)$ | Associated Suffixes |
|---|---|---|
| 0 | $A$ | $C\$_1$ |
| 1 | $T$ | $C\$_2$ |
| 2 | $C$ | $CAAC\$_1$ |
| 3 | $G$ | $CCAAC\$_1$ |
| 4 | $T$ | $CGCTT\$_3$ |
| 5 | $G$ | $CTC\$_2$ |
| 6 | $G$ | $CTT\$_3$ |

| | $B_6(3)$ | Associated Suffixes |
|---|---|---|
| 0 | **A** | **GAGCTC**$\$_2$ |
| 1 | **T** | **GCCAAC**$\$_1$ |
| 2 | $A$ | $GCTC\$_2$ |
| 3 | $C$ | $GCTT\$_3$ |

| | $B_6(4)$ | Associated Suffixes |
|---|---|---|
| 0 | $T$ | $T\$_3$ |
| 1 | $C$ | $TC\$_2$ |
| 2 | **G** | **TCGCTT**$\$_3$ |
| 3 | $C$ | $TT\$_3$ |

Position of $GCCAAC\$_1$ in $G$ segment $=$ # of $G$ before $CCAAC\$_1$ in partial BWT $=$ # of $G$ in \$-segment $+$# of $G$ in $A$-segment $+$# of $G$ before $CCAAC\$_1$ in $C$-segment

## Two versions of our algorithm: BCR vs. BCRext

| | **BCR** | **BCRext** |
|---|---|---|
| CPU time | $O(ksort(m))$ | $O(km)$ |
| RAM usage (bits) | $O((m+\sigma^2)log(mk))$ | $O(\sigma^2 log(mk))$ |
| I/O (bits) | $O(mk^2 log(s))$ | $O(mk^2 log(\sigma))$ |
| | (partial BWT) | (partial BWT) |
| | $O(mklog(\sigma))$ | $O(mk^2 log(\sigma))$ |
| | (sequence slices) | $(sequences)$ |
| | | $O(mklog(mk))$ |
| | | $(P - array)$ |
| | | $O(mklog(m))$ |
| | | $(N - array)$ |

## Performance on human DNA sequence data

| *Dataset size (millions of 100-mers)* | *Program Program* | *Wallclock time ($\mu$s per input base)* | *CPU efficiency (%)* | *Max RAM (Gbyte)* |
|---|---|---|---|---|
| 85 | bwte | 7.99 | 99 | 4.00 |
| | rlcsa | 2.44 | 99 | 13.40 |
| | BCR | 1.01 | 83 | 1.10 |
| | BCRext | 4.75 | 27 | negligible |
| 1000 | BCR | 5.74 | 19 | 13.00 |
| | BCRext | 5.89 | 21 | negligible |

## Further works

- Able to compute BWT of 1 billion 100-mers in under 24 hours
- Ongoing work:
  - Further optimizations to construction, parallelization
  - Software library for construction/querying of BWT of large string collections
  - Algorithm can be adapted to allow sets of strings to be added/removed from collection
  - Applications of BWT of string collection to bioinformatics