Lightweight LCP Construction for Next-Generation Sequencing Datasets

M. J. Bauer[1], A. J. Cox[1], G. Rosone[2], M. Sciortino[2]

[1] Computational Biology Group, Illumina Cambridge Ltd., United Kingdom
[2] Dipartimento di Matematica e Informatica, University of Palermo, Italy

Ljubljana, 10-12 September 2012
WABI 2012

Whole human genome sequencing

- Modern DNA sequencing machines produce a lot of data! e.g. Illumina HiSeq 2000: > 40Gbases of sequence per day (paired 100-mers).
- Datasets of 100 Gbases or more are common.

- Many bioinformatics applications, e.g. the rapid search for maximal exact matches, shortest unique substrings and shortest absent words, use the SA (Suffix Array) and/or BWT (Burrows-Wheeler Transform) together with an additional table: the **LCP** (Longest Common Prefix) array.
- Together, SA/BWT and LCP can replace the larger suffix tree.

Goal: **Lightweight LCP Construction for Next-Generation Sequencing Datasets, i.e. for a large collection of short sequences**.

Let v a sequence on an alphabet of σ letters of length k.

- SA[i]: The starting position of the ith smallest suffix of v.
- BWT[i]: The symbol that (circularly) precedes the first symbol of the suffix.
- LCP[i]: The length of longest common prefix with preceding suffix in the list of sorted suffix.

Example

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$v=$	G	C	A	C	T	G	T	A	C	C	A	A	C	$

	SA	LCP	BWT	Sorted Suffixes of v
0	13	0	C	$
1	10	0	C	$AAC\$$
2	11	1	A	$AC\$$
3	7	2	T	$ACCAAC\$$
4	2	2	C	$ACTGTACCAAC\$$
5	12	0	A	$C\$$
6	9	1	C	$CAAC\$$
7	1	2	G	$CACTGTACCAAC\$$
8	8	1	A	$CCAAC\$$
9	3	1	A	$CTGTACCAAC\$$
10	0	0	$	$GCACTGTACCAAC\$$
11	5	1	T	$GTACCAAC\$$
12	6	0	G	$TACCAAC\$$
13	4	1	C	$TGTACCAAC\$$

For instance, the suffix $ACCAAC\$$ is the 6-suffix of v and the symbol T in the BWT precedes such suffix.

Definition

j-suffix of v is the last j non-$ symbols of that string and 0-suffix of v is $.

Helpful to think of BWT and LCP as being in $\sigma+1$ "segments" labelled according to first symbol of "associated" suffix.

Let v a sequence on an alphabet of σ letters of length k.

- SA[i]: The starting position of the ith smallest suffix of v.
- BWT[i]: The symbol that (circularly) precedes the first symbol of the suffix.
- LCP[i]: The length of longest common prefix with preceding suffix in the list of sorted suffix.

Example

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$v=$	G	C	A	C	T	G	T	A	C	C	A	A	C	\$

	SA	LCP	BWT	Sorted Suffixes of v
0	13	0	C	\$
1	10	0	C	$AAC\$$
2	11	1	A	$AC\$$
3	7	2	T	$ACCAAC\$$
4	2	2	C	$ACTGTACCAAC\$$
5	12	0	A	$C\$$
6	9	1	C	$CAAC\$$
7	1	2	G	$CACTGTACCAAC\$$
8	8	1	A	$CCAAC\$$
9	3	1	A	$CTGTACCAAC\$$
10	0	0	\$	$GCACTGTACCAAC\$$
11	5	1	T	$GTACCAAC\$$
12	6	0	G	$TACCAAC\$$
13	4	1	C	$TGTACCAAC\$$

For instance, the suffix $ACCAAC\$$ is the 6-suffix of v and the symbol T in the BWT precedes such suffix.

Definition

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

Helpful to think of BWT and LCP as being in $\sigma + 1$ "segments" labelled according to first symbol of "associated" suffix.

Let v a sequence on an alphabet of σ letters of length k.

- SA[i]: The starting position of the ith smallest suffix of v.
- BWT[i]: The symbol that (circularly) precedes the first symbol of the suffix.
- LCP[i]: The length of longest common prefix with preceding suffix in the list of sorted suffix.

Example

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$v=$	G	C	A	C	T	G	T	A	C	C	A	A	C	\$

	SA	LCP	BWT	Sorted Suffixes of v
0	13	0	C	\$
1	10	0	C	$AAC\$$
2	11	1	A	$AC\$$
3	7	2	T	$ACCAAC\$$
4	2	2	C	$ACTGTACCAAC\$$
5	12	0	A	$C\$$
6	9	1	C	$CAAC\$$
7	1	2	G	$CACTGTACCAAC\$$
8	8	1	A	$CCAAC\$$
9	3	1	A	$CTGTACCAAC\$$
10	0	0	\$	$GCACTGTACCAAC\$$
11	5	1	T	$GTACCAAC\$$
12	6	0	G	$TACCAAC\$$
13	4	1	C	$TGTACCAAC\$$

For instance, the suffix $ACCAAC\$$ is the 6-suffix of v and the symbol T in the BWT precedes such suffix.

Definition

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

Helpful to think of BWT and LCP as being in $\sigma + 1$ "segments" labelled according to first symbol of "associated" suffix.

Let v a sequence on an alphabet of σ letters of length k.

- SA[i]: The starting position of the ith smallest suffix of v.
- BWT[i]: The symbol that (circularly) precedes the first symbol of the suffix.
- LCP[i]: The length of longest common prefix with preceding suffix in the list of sorted suffix.

Example

$v=$	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	G	C	A	C	T	G	T	A	C	C	A	A	C	$

	SA	LCP	BWT	Sorted Suffixes of v	
0	13	0	C	$\$$	$-segment
1	10	0	C	$AAC\$$	A-segment
2	11	1	A	$AC\$$	
3	7	2	T	$ACCAAC\$$	
4	2	2	C	$ACTGTACCAAC\$$	
5	12	0	A	$C\$$	C-segment
6	9	1	C	$CAAC\$$	
7	1	2	G	$CACTGTACCAAC\$$	
8	8	1	A	$CCAAC\$$	
9	3	1	A	$CTGTACCAAC\$$	
10	0	0	$\$$	$GCACTGTACCAAC\$$	G-segment
11	5	1	T	$GTACCAAC\$$	
12	6	0	G	$TACCAAC\$$	T-segment
13	4	1	C	$TGTACCAAC\$$	

For instance, the suffix $ACCAAC\$$ is the 6-suffix of v and the symbol T in the BWT precedes such suffix.

Definition

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

Helpful to think of BWT and LCP as being in $\sigma + 1$ "segments" labelled according to first symbol of "associated" suffix.

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of strings on an alphabet of σ letters. The sum of lengths of S_i is N.

- GSA[i]: The i-th smallest suffix of the strings in S. If GSA[i]=(t,h), then it corresponds to the suffix starting at the position t of the string S_h.
- BWT[i]: The symbol that (circularly) precedes the first symbol of the suffix of S_h.
- LCP[i]: The length of longest common prefix with preceding suffix in the sorted list of the suffixes of S.

Example

	0	1	2	3	4	5	6
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	$\$_3$

GSA	LCP	BWT	Sorted Suffixes of S	
$(6,1)$	0	C	$\$_1$	$-segment
$(6,2)$	0	C	$\$_2$	
$(6,3)$	0	T	$\$_3$	
$(3,1)$	0	C	$AAC\$_1$	A-segment
$(4,1)$	1	A	$AC\$_1$	
$(1,2)$	1	G	$AGCTC\$_2$	
$(5,1)$	0	A	$C\$_1$	C-segment
$(5,2)$	1	T	$C\$_2$	
$(2,1)$	1	C	$CAAC\$_1$	
$(1,1)$	1	G	$CCAAC\$_1$	
$(1,3)$	1	T	$CGCTT\$_3$	
$(3,2)$	1	G	$CTC\$_2$	
$(3,3)$	2	G	$CTT\$_3$	
$(0,2)$	0	$\$_2$	$GAGCTC\$_2$	G-segment
$(0,1)$	1	$\$_1$	$GCCAAC\$_1$	
$(2,2)$	2	A	$GCTC\$_2$	
$(2,3)$	3	C	$GCTT\$_3$	
$(5,3)$	0	T	$T\$_3$	T-segment
$(4,2)$	1	C	$TC\$_2$	
$(0,3)$	2	$\$_3$	$TCGCTT\$_3$	
$(4,3)$	1	C	$TT\$_3$	

For instance, the suffixes $CAAC\$_1$, $GCTC\$_2$,$GCTT\$_3$ are the 4-suffixes of S.

In general, j-suffix of $S_i \in$ S is the last j non-\$ symbols of that string and 0-suffix of S_i is $\$_i$.

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of strings on an alphabet of σ letters. The sum of lengths of S_i is N.

- GSA[i]: The i-th smallest suffix of the strings in S. If GSA[i]=(t,h), then it corresponds to the suffix starting at the position t of the string S_h.
- BWT[i]: The symbol that (circularly) precedes the first symbol of the suffix of S_h.
- LCP[i]: The length of longest common prefix with preceding suffix in the sorted list of the suffixes of S.

Example

	0	1	2	3	4	5	6
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	$\$_3$

GSA	LCP	BWT	Sorted Suffixes of S	
$(6,1)$	0	C	$\$_1$	$-segment
$(6,2)$	0	C	$\$_2$	
$(6,3)$	0	T	$\$_3$	
$(3,1)$	0	C	$AAC\$_1$	A-segment
$(4,1)$	1	A	$AC\$_1$	
$(1,2)$	1	G	$AGCTC\$_2$	
$(5,1)$	0	A	$C\$_1$	C-segment
$(5,2)$	1	T	$C\$_2$	
$(2,1)$	1	C	$CAAC\$_1$	
$(1,1)$	1	G	$CCAAC\$_1$	
$(1,3)$	1	T	$CGCTT\$_3$	
$(3,2)$	1	G	$CTC\$_2$	
$(3,3)$	2	G	$CTT\$_3$	
$(0,2)$	0	$\$_2$	$GAGCTC\$_2$	G-segment
$(0,1)$	1	$\$_1$	$GCCAAC\$_1$	
$(2,2)$	2	A	$GCTC\$_2$	
$(2,3)$	3	C	$GCTT\$_3$	
$(5,3)$	0	T	$T\$_3$	T-segment
$(4,2)$	1	C	$TC\$_2$	
$(0,3)$	2	$\$_3$	$TCGCTT\$_3$	
$(4,3)$	1	C	$TT\$_3$	

For instance, the suffixes $CAAC\$_1$, $GCTC\$_2$,$GCTT\$_3$ are the 4-suffixes of S.

In general, j-suffix of $S_i \in$ S is the last j non-\$ symbols of that string and 0-suffix of S_i is $\$_i$.

A massive collection of sequences

Input:
A massive collection S of m strings on an alphabet of σ letters.
Output:
The LCP array of S (mainly) working in external memory.

- Usual algorithms do not fit to handle collections of sequences. So they concatenate sequences of S in order to obtain a single sequence.
- These algorithms
  - compute the LCP from Suffix Array (Kasai et. al. 2001, Kärkkäinen et. al. 2009, and so on).
    But: (often) need to hold SA in RAM (Simpson and Durbin estimate 700Gbytes RAM for SA of 60 Gbases of data).
  - compute the LCP **acting** directly on the BWT of the string and does not need its suffix array (Beller et. al. 2011).
    But: they mainly work in internal memory.

Our idea

Our algorithm computes the LCP of the collection S of sequences mostly in external memory, storing some tables in internal memory:

- without concatenating the strings belonging to S.
- without pre-computing either the BWT or the (G)SA.

It computes the LCP and the BWT at the same time.

Building upon the method (called BCR) of BWT computation (in external memory) introduced in Bauer et al., our algorithm adds some lightweight data structures and allows the LCP and BWT of a collection of strings to be computed simultaneously.

For further details on building of the BWT in external memory:

M. J. Bauer, A. J. Cox, G. R., Lightweight algorithms for constructing and inverting the BWT of string collections, Theoretical Computer Science, Available online 10 February 2012.

Our idea

Our algorithm computes the LCP of the collection S of sequences mostly in external memory, storing some tables in internal memory:

- without concatenating the strings belonging to S.
- without pre-computing either the BWT or the (G)SA.

It computes the LCP and the BWT at the same time.

Building upon the method (called BCR) of BWT computation (in external memory) introduced in Bauer et al., our algorithm adds some lightweight data structures and allows the LCP and BWT of a collection of strings to be computed simultaneously.

For further details on building of the BWT in external memory:

M. J. Bauer, A. J. Cox, G. R., Lightweight algorithms for constructing and inverting the BWT of string collections, Theoretical Computer Science, Available online 10 February 2012.

Our idea

Our algorithm computes the LCP of the collection S of sequences mostly in external memory, storing some tables in internal memory:

- without concatenating the strings belonging to S.
- without pre-computing either the BWT or the (G)SA.

It computes the LCP and the BWT at the same time.

Building upon the method (called BCR) of BWT computation (in external memory) introduced in Bauer et al., our algorithm adds some lightweight data structures and allows the LCP and BWT of a collection of strings to be computed simultaneously.

For further details on building of the BWT in external memory:

M. J. Bauer, A. J. Cox, G. R., Lightweight algorithms for constructing and inverting the BWT of string collections, Theoretical Computer Science, Available online 10 February 2012.

Our idea

Our algorithm computes the LCP of the collection S of sequences mostly in external memory, storing some tables in internal memory:

- without concatenating the strings belonging to S.
- without pre-computing either the BWT or the (G)SA.

It computes the LCP and the BWT at the same time.

Building upon the method (called `BCR`) of BWT computation (in external memory) introduced in Bauer et al., our algorithm adds some lightweight data structures and allows the LCP and BWT of a collection of strings to be computed simultaneously.

For further details on building of the BWT in external memory:

M. J. Bauer, A. J. Cox, G. R., Lightweight algorithms for constructing and inverting the BWT of string collections, Theoretical Computer Science, Available online 10 February 2012.

Our idea

Our algorithm computes the LCP of the collection S of sequences mostly in external memory, storing some tables in internal memory:

- without concatenating the strings belonging to S.
- without pre-computing either the BWT or the (G)SA.

It computes the LCP and the BWT at the same time.

Building upon the method (called `BCR`) of BWT computation (in external memory) introduced in Bauer et al., our algorithm adds some lightweight data structures and allows the LCP and BWT of a collection of strings to be computed simultaneously.

For further details on building of the BWT in external memory:

M. J. Bauer, A. J. Cox, G. R., Lightweight algorithms for constructing and inverting the BWT of string collections, Theoretical Computer Science, Available online 10 February 2012.

Our algorithm extLCP

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of strings.

Definition

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

Our algorithm

- works incrementally via K iterations, where K is the maximal length of the strings in S. At each of the iterations $j = 1, 2, \ldots, K-1$, the algorithm considers all the j-suffixes of S and computes
  - a partial BWT string $\mathsf{bwt}_j(\mathsf{S})$ by inserting the symbols preceding the j-suffixes of S at their correct positions into $\mathsf{bwt}_{j-1}(\mathsf{S})$
  - a partial LCP array $\mathsf{lcp}_j(\mathsf{S})$ by inserting the LCP-values of j-suffixes of S and updating the LCP-values of suffixes already inserted.

  Each iteration j simulates the insertion of the j-suffixes in the suffix array.
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

Our algorithm extLCP

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of strings.

Definition

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

Our algorithm

- works incrementally via K iterations, where K is the maximal length of the strings in S. At each of the iterations $j = 1, 2, \ldots, K-1$, the algorithm considers all the j-suffixes of S and computes
  - a partial BWT string bwt$_j$(S) by inserting the symbols preceding the j-suffixes of S at their correct positions into bwt$_{j-1}$(S)
  - a partial LCP array lcp$_j$(S) by inserting the LCP-values of j-suffixes of S and updating the LCP-values of suffixes already inserted.

  Each iteration j simulates the insertion of the j-suffixes in the suffix array.
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

Our algorithm extLCP

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of strings.

Definition

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

Our algorithm

- works incrementally via K iterations, where K is the maximal length of the strings in S. At each of the iterations $j = 1, 2, \ldots, K-1$, the algorithm considers all the j-suffixes of S and computes
  - a partial BWT string bwt$_j$(S) by inserting the symbols preceding the j-suffixes of S at their correct positions into bwt$_{j-1}$(S)
  - a partial LCP array lcp$_j$(S) by inserting the LCP-values of j-suffixes of S and updating the LCP-values of suffixes already inserted.

  Each iteration j simulates the insertion of the j-suffixes in the suffix array.
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

Our algorithm extLCP

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of strings.

Definition

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

Our algorithm

- works incrementally via K iterations, where K is the maximal length of the strings in S. At each of the iterations $j = 1, 2, \ldots, K-1$, the algorithm considers all the j-suffixes of S and computes
  - a partial BWT string $\mathsf{bwt}_j(\mathsf{S})$ by inserting the symbols preceding the j-suffixes of S at their correct positions into $\mathsf{bwt}_{j-1}(\mathsf{S})$
  - a partial LCP array $\mathsf{lcp}_j(\mathsf{S})$ by inserting the LCP-values of j-suffixes of S and updating the LCP-values of suffixes already inserted.

  Each iteration j simulates the insertion of the j-suffixes in the suffix array.
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

Our algorithm extLCP

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of strings.

Definition

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

Our algorithm

- works incrementally via K iterations, where K is the maximal length of the strings in S. At each of the iterations $j = 1, 2, \ldots, K-1$, the algorithm considers all the j-suffixes of S and 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 $\mathrm{bwt}_{j-1}(\mathrm{S})$
  - a partial LCP array $\mathrm{lcp}_j(\mathrm{S})$ by inserting the LCP-values of j-suffixes of S and updating the LCP-values of suffixes already inserted.

  Each iteration j simulates the insertion of the j-suffixes in the suffix array.
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1														$\$_1$
S_2														$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1													C	$\$_1$
S_2													C	$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1												A	C	$\$_1$
S_2												T	C	$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1											A	A	C	$\$_1$
S_2											C	T	C	$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1										C	A	A	C	$\$_1$
S_2										G	C	T	C	$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1									C	C	A	A	C	$\$_1$
S_2									A	G	C	T	C	$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1								A	C	C	A	A	C	$\$_1$
S_2								A	A	G	C	T	C	$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1							T	A	C	C	A	A	C	$\$_1$
S_2							A	A	A	G	C	T	C	$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1						G	T	A	C	C	A	A	C	$\$_1$
S_2						G	A	A	A	G	C	T	C	$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1					T	G	T	A	C	C	A	A	C	$\$_1$
S_2					A	G	A	A	A	G	C	T	C	$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1				C	T	G	T	A	C	C	A	A	C	$\$_1$
S_2				C	A	G	A	A	A	G	C	T	C	$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1			A	C	T	G	T	A	C	C	A	A	C	$\$_1$
S_2			A	C	A	G	A	A	A	G	C	T	C	$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1		C	A	C	T	G	T	A	C	C	A	A	C	$\$_1$
S_2		A	A	C	A	G	A	A	A	G	C	T	C	$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1	A	C	A	C	T	G	T	A	C	C	A	A	C	$\$_1$
S_2	G	A	A	C	A	G	A	A	A	G	C	T	C	$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1	A	C	A	C	T	G	T	A	C	C	A	A	C	$\$_1$
S_2	G	A	A	C	A	G	A	A	A	G	C	T	C	$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1	A	C	A	C	T	G	T	A	C	C	A	A	C	$\$_1$
S_2	G	A	A	C	A	G	A	A	A	G	C	T	C	$\$_2$

At step j, we insert the symbols circularly preceding the j-suffixes into the partial BWT and insert/update the LCP-values in the partial LCP.

We do not need to keep the entire collection in internal memory. It is enough to keep the symbols that we have to insert at the iteration j (red symbols).

We assume that $\$_1 = \$_2 = \$$ and $\$ < A < C < G < T$ and the longest common prefix between two end-markers is 0.

$S_i[|S_i|] = S_j[|S_j|] = \$$, and we define $S_i[|S_i|] < S_j[|S_j|]$, if $i < j$.

Example: Iteration 0

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1														$\$_1$
S_2														$\$_2$

We assume that the first element of LCP array is 0. Recall that $lcp(\$_1, \$_2) = 0$. We obtain:

$L_0(\$)$	$B_0(\$)$
0	C
0	C

Sorted Suffixes
$\$_1$
$\$_2$

Helpful to think of BWT and LCP as being in $\sigma + 1$ "segments" labelled according to the first symbol of associated suffix:
bwt$_0$(S) $= B_0(\$)$ and lcp$_0$(S) $= L_0(\$)$.

Example: Iteration 0

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1													C	$\$_1$
S_2													C	$\$_2$

We assume that the first element of LCP array is 0. Recall that $lcp(\$_1, \$_2) = 0$. We obtain:

$L_0(\$)$	$B_0(\$)$
0	C
0	C

Sorted Suffixes
$\$_1$
$\$_2$

Helpful to think of BWT and LCP as being in $\sigma + 1$ "segments" labelled according to the first symbol of associated suffix: $\mathrm{bwt}_0(\mathrm{S}) = B_0(\$)$ and $\mathrm{lcp}_0(\mathrm{S}) = L_0(\$)$.

Example: Iteration 0

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1													C	$\$_1$
S_2													C	$\$_2$

We assume that the first element of LCP array is 0. Recall that $lcp(\$_1, \$_2) = 0$. We obtain:

$L_0(\$)$	$B_0(\$)$
0	C
0	C

Sorted Suffixes
$\$_1$
$\$_2$

Helpful to think of BWT and LCP as being in $\sigma + 1$ "segments" labelled according to the first symbol of associated suffix: bwt$_0$(S) $= B_0(\$)$ and lcp$_0$(S) $= L_0(\$)$.

Example: Iteration 0

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1													C	$\$_1$
S_2													C	$\$_2$

We assume that the first element of LCP array is 0. Recall that $lcp(\$_1, \$_2) = 0$. We obtain:

$L_0(\$)$	$B_0(\$)$
0	C
0	C

Sorted Suffixes
$\$_1$
$\$_2$

Helpful to think of BWT and LCP as being in $\sigma + 1$ "segments" labelled according to the first symbol of associated suffix: $\text{bwt}_0(\text{S}) = B_0(\$)$ and $\text{lcp}_0(\text{S}) = L_0(\$)$.

Example: Iteration 0

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1													C	$\$_1$
S_2													C	$\$_2$

We assume that the first element of LCP array is 0. Recall that $lcp(\$_1, \$_2) = 0$. We obtain:

$L_0(\$)$	$B_0(\$)$
0	C
0	C

Sorted Suffixes
$\$_1$
$\$_2$

Helpful to think of BWT and LCP as being in $\sigma + 1$ "segments" labelled according to the first symbol of associated suffix:
bwt$_0$(S) $= B_0(\$)$ and lcp$_0$(S) $= L_0(\$)$.

Example: Iteration 0

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1													C	$\$_1$
S_2													C	$\$_2$

We assume that the first element of LCP array is 0. Recall that $lcp(\$_1, \$_2) = 0$. We obtain:

$L_0(\$)$	$B_0(\$)$
0	C
0	C

Sorted Suffixes
$\$_1$
$\$_2$

Helpful to think of BWT and LCP as being in $\sigma + 1$ "segments" labelled according to the first symbol of associated suffix:
$\mathrm{bwt}_0(\mathrm{S}) = B_0(\$)$ and $\mathrm{lcp}_0(\mathrm{S}) = L_0(\$)$.

Example: Iteration 1

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1													C	$\$_1$
S_2													C	$\$_2$

$L_0(\$)$	$B_0(\$)$
0	C
0	C

Sorted Suffixes
$\$_1$
$\$_2$

$L_1(C)$	$B_1(C)$
0	A
1	T

Sorted Suffixes
$C\$_1$
$C\$_2$

We recall that $\text{bwt}_1(\text{S}) = B_1(\$)B_1(A)\cdots B_1(T)$ and $\text{lcp}_1(\text{S}) = L_1(\$)L_1(A)\cdots L_1(T)$.

Example: Iteration 1

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1													C	$\$_1$
S_2													C	$\$_2$

$L_0(\$)$	$B_0(\$)$
0	C
0	C

Sorted Suffixes
$\$_1$
$\$_2$

$L_1(C)$	$B_1(C)$
0	A
1	T

Sorted Suffixes
$C\$_1$
$C\$_2$

We recall that $\text{bwt}_1(\text{S}) = B_1(\$)B_1(A)\cdots B_1(T)$ and $\text{lcp}_1(\text{S}) = L_1(\$)L_1(A)\cdots L_1(T)$.

Example: Iteration 1

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1												A	C	$\$_1$
S_2												T	C	$\$_2$

$L_0(\$)$	$B_0(\$)$
0	C
0	C

Sorted Suffixes
$\$_1$
$\$_2$

$L_1(C)$	$B_1(C)$
0	A
1	T

Sorted Suffixes
$C\$_1$
$C\$_2$

We recall that $\mathrm{bwt}_1(S) = B_1(\$)B_1(A)\cdots B_1(T)$ and $\mathrm{lcp}_1(S) = L_1(\$)L_1(A)\cdots L_1(T)$.

Example: Iteration 1

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1												A	C	$\$_1$
S_2												T	C	$\$_2$

$L_1(\$)$	$B_1(\$)$
0	C
0	C

Sorted Suffixes
$\$_1$
$\$_2$

$L_1(C)$	$B_1(C)$
0	A
1	T

Sorted Suffixes
$C\$_1$
$C\$_2$

We recall that $\text{bwt}_1(\text{S}) = B_1(\$)B_1(A)\cdots B_1(T)$ and $\text{lcp}_1(\text{S}) = L_1(\$)L_1(A)\cdots L_1(T)$.

Example: Iteration 1

Let S $= \{S_1, S_2\} = \{ACACTGTACCAAC, GAACAGAAAGCTC\}$ be a collection of $m = 2$ strings of length $k = 13$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1											A	A	C	$\$_1$
S_2											C	T	C	$\$_2$

$L_1(\$)$	$B_1(\$)$
0	C
0	C

Sorted Suffixes
$\$_1$
$\$_2$

$L_1(C)$	$B_1(C)$
0	A
1	T

Sorted Suffixes
$C\$_1$
$C\$_2$

We recall that $\mathsf{bwt}_1(\mathsf{S}) = B_1(\$)B_1(A)\cdots B_1(T)$ and $\mathsf{lcp}_1(\mathsf{S}) = L_1(\$)L_1(A)\cdots L_1(T)$.

Example: Looking in detail at iteration 13

	$L_{12}(\$)$	$B_{12}(\$)$	Sorted Suffixes
	0	C	$\$_1$
	0	C	$\$_2$

	$L_{12}(A)$	$B_{12}(A)$	Sorted Suffixes
	0	G	$AAAGCTC\$_2$
	2	C	$AAC\$_1$
→	**3**	$\mathbf{G}$	$\mathbf{AACAGAAAGCTC\$_2}$
	2	A	$AAGCTC\$_2$
	1	A	$AC\$_1$
	2	A	$ACAGAAAGCTC\$_2$
	2	T	$ACCAAC\$_1$
	2	C	$ACTGTACCAAC\$_1$
	1	C	$AGAAAGCTC\$_2$
	2	A	$AGCTC\$_2$

	$L_{12}(C)$	$B_{12}(C)$	Sorted Suffixes
	0	A	$C\$_1$
	1	T	$C\$_2$
	1	C	$CAAC\$_1$
→	**2**	$\mathbf{A}$	$\mathbf{CACTGTACCAAC\$_1}$
	2	$\mathbf{A}$	$CAGAAAGCTC\$_2$
	1	A	$CCAAC\$_1$
	1	G	$CTC\$_2$
	2	A	$CTGTACCAAC\$_1$

	$L_{12}(G)$	$B_{12}(G)$	Sorted Suffixes
	0	A	$GAAAGCTC\$_2$
	1	A	$GCTC\$_2$
	1	T	$GTACCAAC\$_1$

	$L_{12}(T)$	$B_{12}(T)$	Sorted Suffixes
	0	G	$TACCAAC\$_1$
	1	C	$TC\$_2$
	1	C	$TGTACCAAC\$_1$

$L_{13}(\$)$	$B_{13}(\$)$	Sorted Suffixes
0	C	$\$_1$
0	C	$\$_2$

$L_{13}(A)$	$B_{13}(A)$	Sorted Suffixes
0	G	$AAAGCTC\$_2$
2	C	$AAC\$_1$
3	G	$AACAGAAAGCTC\$_2$
2	A	$AAGCTC\$_2$
1	A	$AC\$_1$
$\underline{2}$	A	$ACAGAAAGCTC\$_2$
2	T	$ACCAAC\$_1$
2	C	$ACTGTACCAAC\$_1$
1	C	$AGAAAGCTC\$_2$
2	A	$AGCTC\$_2$

$L_{13}(C)$	$B_{13}(C)$	Sorted Suffixes
0	A	$C\$_1$
1	T	$C\$_2$
1	C	$CAAC\$_1$
2	G	$CACTGTACCAAC\$_1$
2	A	$CAGAAAGCTC\$_2$
1	A	$CCAAC\$_1$
1	G	$CTC\$_2$
2	A	$CTGTACCAAC\$_1$

$L_{13}(G)$	$B_{13}(G)$	Sorted Suffixes
0	A	$GAAAGCTC\$_2$
$\underline{1}$	A	$GCTC\$_2$
1	T	$GTACCAAC\$_1$

$L_{13}(T)$	$B_{13}(T)$	Sorted Suffixes
0	G	$TACCAAC\$_1$
1	C	$TC\$_2$
1	C	$TGTACCAAC\$_1$

Example: Looking in detail at iteration 13

	$L_{12}(\$)$	$B_{12}(\$)$	Sorted Suffixes	$L_{13}(\$)$	$B_{13}(\$)$	Sorted Suffixes
	0	C	$\$_1$	0	C	$\$_1$
	0	C	$\$_2$	0	C	$\$_2$
	$L_{12}(A)$	$B_{12}(A)$	Sorted Suffixes	$L_{13}(A)$	$B_{13}(A)$	Sorted Suffixes
	0	G	$AAAGCTC\$_2$	0	G	$AAAGCTC\$_2$
	2	C	$AAC\$_1$	2	C	$AAC\$_1$
$\rightarrow$	**3**	**G**	$\mathbf{AACAGAAAGCTC\$_2}$	3	G	$AACAGAAAGCTC\$_2$
	2	A	$AAGCTC\$_2$	2	A	$AAGCTC\$_2$
	1	A	$AC\$_1$	1	A	$AC\$_1$
	2	A	$ACAGAAAGCTC\$_2$			
	2	T	$ACCAAC\$_1$	$\underline{2}$	A	$ACAGAAAGCTC\$_2$
	2	C	$ACTGTACCAAC\$_1$	2	T	$ACCAAC\$_1$
	1	C	$AGAAAGCTC\$_2$	2	C	$ACTGTACCAAC\$_1$
	2	A	$AGCTC\$_2$	1	C	$AGAAAGCTC\$_2$
				2	A	$AGCTC\$_2$
	$L_{12}(C)$	$B_{12}(C)$	Sorted Suffixes	$L_{13}(C)$	$B_{13}(C)$	Sorted Suffixes
	0	A	$C\$_1$	0	A	$C\$_1$
	1	T	$C\$_2$	1	T	$C\$_2$
	1	C	$CAAC\$_1$	1	C	$CAAC\$_1$
$\rightarrow$	**2**	**A**	$\mathbf{CACTGTACCAAC\$_1}$	2	G	$CACTGTACCAAC\$_1$
	2	**A**	$CAGAAAGCTC\$_2$	2	A	$CAGAAAGCTC\$_2$
	1	A	$CCAAC\$_1$	1	A	$CCAAC\$_1$
	1	G	$CTC\$_2$	1	G	$CTC\$_2$
	2	A	$CTGTACCAAC\$_1$	2	A	$CTGTACCAAC\$_1$
	$L_{12}(G)$	$B_{12}(G)$	Sorted Suffixes	$L_{13}(G)$	$B_{13}(G)$	Sorted Suffixes
	0	A	$GAAAGCTC\$_2$	0	A	$GAAAGCTC\$_2$
	1	A	$GCTC\$_2$			
	1	T	$GTACCAAC\$_1$	$\underline{1}$	A	$GCTC\$_2$
				1	T	$GTACCAAC\$_1$
	$L_{12}(T)$	$B_{12}(T)$	Sorted Suffixes	$L_{13}(T)$	$B_{13}(T)$	Sorted Suffixes
	0	G	$TACCAAC\$_1$	0	G	$TACCAAC\$_1$
	1	C	$TC\$_2$	1	C	$TC\$_2$
	1	C	$TGTACCAAC\$_1$	1	C	$TGTACCAAC\$_1$

Example: Looking in detail at iteration 13

	$L_{12}(\$)$	$B_{12}(\$)$	Sorted Suffixes	$L_{13}(\$)$	$B_{13}(\$)$	Sorted Suffixes
	0	C	$\$_1$	0	C	$\$_1$
	0	C	$\$_2$	0	C	$\$_2$
	$L_{12}(A)$	$B_{12}(A)$	Sorted Suffixes	$L_{13}(A)$	$B_{13}(A)$	Sorted Suffixes
	0	G	$AAAGCTC\$_2$	0	G	$AAAGCTC\$_2$
	2	C	$AAC\$_1$	2	C	$AAC\$_1$
$\rightarrow$	**3**	$\mathbf{G}$	$\mathbf{AACAGAAAGCTC\$_2}$	3	G	$AACAGAAAGCTC\$_2$
	2	A	$AAGCTC\$_2$	2	A	$AAGCTC\$_2$
	1	A	$AC\$_1$	1	A	$AC\$_1$
	2	A	$ACAGAAAGCTC\$_2 \rightarrow$	1+1=2	$\$_1$	$\mathbf{ACACTGTACCAAC\$_1}$
	2	T	$ACCAAC\$_1$	$\underline{2}$	A	$ACAGAAAGCTC\$_2$
	2	C	$ACTGTACCAAC\$_1$	2	T	$ACCAAC\$_1$
	1	C	$AGAAAGCTC\$_2$	2	C	$ACTGTACCAAC\$_1$
	2	A	$AGCTC\$_2$	1	C	$AGAAAGCTC\$_2$
				2	A	$AGCTC\$_2$
	$L_{12}(C)$	$B_{12}(C)$	Sorted Suffixes	$L_{13}(C)$	$B_{13}(C)$	Sorted Suffixes
	0	A	$C\$_1$	0	A	$C\$_1$
$min = 1$	1	T	$C\$_2$	1	T	$C\$_2$
	1	C	$CAAC\$_1$	1	C	$CAAC\$_1$
$\rightarrow$	**2**	$\mathbf{A}$	$\mathbf{CACTGTACCAAC\$_1}$	2	G	$CACTGTACCAAC\$_1$
	2	$\mathbf{A}$	$CAGAAAGCTC\$_2$	2	A	$CAGAAAGCTC\$_2$
	1	A	$CCAAC\$_1$	1	A	$CCAAC\$_1$
	1	G	$CTC\$_2$	1	G	$CTC\$_2$
	2	A	$CTGTACCAAC\$_1$	2	A	$CTGTACCAAC\$_1$
	$L_{12}(G)$	$B_{12}(G)$	Sorted Suffixes	$L_{13}(G)$	$B_{13}(G)$	Sorted Suffixes
	0	A	$GAAAGCTC\$_2$	0	A	$GAAAGCTC\$_2$
	1	A	$GCTC\$_2$			
	1	T	$GTACCAAC\$_1$	$\underline{\mathbf{1}}$	A	$GCTC\$_2$
				1	T	$GTACCAAC\$_1$
	$L_{12}(T)$	$B_{12}(T)$	Sorted Suffixes	$L_{13}(T)$	$B_{13}(T)$	Sorted Suffixes
	0	G	$TACCAAC\$_1$	0	G	$TACCAAC\$_1$
	1	C	$TC\$_2$	1	C	$TC\$_2$
	1	C	$TGTACCAAC\$_1$	1	C	$TGTACCAAC\$_1$

Example: Looking in detail at iteration 13

	$L_{12}(\$)$	$B_{12}(\$)$	Sorted Suffixes	$L_{13}(\$)$	$B_{13}(\$)$	Sorted Suffixes
	0	C	$\$_1$	0	C	$\$_1$
	0	C	$\$_2$	0	C	$\$_2$

	$L_{12}(A)$	$B_{12}(A)$	Sorted Suffixes	$L_{13}(A)$	$B_{13}(A)$	Sorted Suffixes
	0	G	$AAAGCTC\$_2$	0	G	$AAAGCTC\$_2$
	2	C	$AAC\$_1$	2	C	$AAC\$_1$
$\rightarrow$	**3**	**G**	$\mathbf{AACAGAAAGCTC\$_2}$	3	G	$AACAGAAAGCTC\$_2$
	2	A	$AAGCTC\$_2$	2	A	$AAGCTC\$_2$
	1	A	$AC\$_1$	1	A	$AC\$_1$
	2	A	$ACAGAAAGCTC\$_2 \rightarrow$	**1+1=2**	$\$_1$	$\mathbf{ACACTGTACCAAC\$_1}$
	2	T	$ACCAAC\$_1$	**2+1=<u>3</u>**	A	$ACAGAAAGCTC\$_2$
	2	C	$ACTGTACCAAC\$_1$	2	T	$ACCAAC\$_1$
	1	C	$AGAAAGCTC\$_2$	2	C	$ACTGTACCAAC\$_1$
	2	A	$AGCTC\$_2$	1	C	$AGAAAGCTC\$_2$
				2	A	$AGCTC\$_2$

	$L_{12}(C)$	$B_{12}(C)$	Sorted Suffixes	$L_{13}(C)$	$B_{13}(C)$	Sorted Suffixes
	0	A	$C\$_1$	0	A	$C\$_1$
$min = 1$	1	T	$C\$_2$	1	T	$C\$_2$
	1	C	$CAAC\$_1$	1	C	$CAAC\$_1$
$\rightarrow$	**2**	**A**	$\mathbf{CACTGTACCAAC\$_1}$	2	G	$CACTGTACCAAC\$_1$
$min = 2$	**2**	**A**	$CAGAAAGCTC\$_2$	2	A	$CAGAAAGCTC\$_2$
	1	A	$CCAAC\$_1$	1	A	$CCAAC\$_1$
	1	G	$CTC\$_2$	1	G	$CTC\$_2$
	2	A	$CTGTACCAAC\$_1$	2	A	$CTGTACCAAC\$_1$

	$L_{12}(G)$	$B_{12}(G)$	Sorted Suffixes	$L_{13}(G)$	$B_{13}(G)$	Sorted Suffixes
	0	A	$GAAAGCTC\$_2$	0	A	$GAAAGCTC\$_2$
	1	A	$GCTC\$_2$			
	1	T	$GTACCAAC\$_1$	**<u>1</u>**	A	$GCTC\$_2$
				1	T	$GTACCAAC\$_1$

	$L_{12}(T)$	$B_{12}(T)$	Sorted Suffixes	$L_{13}(T)$	$B_{13}(T)$	Sorted Suffixes
	0	G	$TACCAAC\$_1$	0	G	$TACCAAC\$_1$
	1	C	$TC\$_2$	1	C	$TC\$_2$
	1	C	$TGTACCAAC\$_1$	1	C	$TGTACCAAC\$_1$

Example: Looking in detail at iteration 13

	$L_{12}(\$)$	$B_{12}(\$)$	Sorted Suffixes		$L_{13}(\$)$	$B_{13}(\$)$	Sorted Suffixes
	0	C	$\$_1$		0	C	$\$_1$
	0	C	$\$_2$		0	C	$\$_2$

	$L_{12}(A)$	$B_{12}(A)$	Sorted Suffixes		$L_{13}(A)$	$B_{13}(A)$	Sorted Suffixes
	0	G	$AAAGCTC\$_2$		0	G	$AAAGCTC\$_2$
$min = 2$	2	C	$AAC\$_1$		2	C	$AAC\$_1$
$\rightarrow$	**3**	**G**	**AACAGAAAGCTC$\$_2$**		3	G	$AACAGAAAGCTC\$_2$
	2	A	$AAGCTC\$_2$		2	A	$AAGCTC\$_2$
	1	A	$AC\$_1$		1	A	$AC\$_1$
	2	A	$ACAGAAAGCTC\$_2$	$\rightarrow$	**1+1=2**	**$\$_1$**	**ACACTGTACCAAC$\$_1$**
	2	T	$ACCAAC\$_1$		**2+1=3**	A	$ACAGAAAGCTC\$_2$
	2	C	$ACTGTACCAAC\$_1$		2	T	$ACCAAC\$_1$
	1	C	$AGAAAGCTC\$_2$		2	C	$ACTGTACCAAC\$_1$
	2	A	$AGCTC\$_2$		1	C	$AGAAAGCTC\$_2$
					2	A	$AGCTC\$_2$

	$L_{12}(C)$	$B_{12}(C)$	Sorted Suffixes		$L_{13}(C)$	$B_{13}(C)$	Sorted Suffixes
	0	A	$C\$_1$		0	A	$C\$_1$
$min = 1$	1	T	$C\$_2$		1	T	$C\$_2$
	1	C	$CAAC\$_1$		1	C	$CAAC\$_1$
$\rightarrow$	**2**	**A**	**CACTGTACCAAC$\$_1$**		2	G	$CACTGTACCAAC\$_1$
$min = 2$	**2**	**A**	$CAGAAAGCTC\$_2$		2	A	$CAGAAAGCTC\$_2$
	1	A	$CCAAC\$_1$		1	A	$CCAAC\$_1$
	1	G	$CTC\$_2$		1	G	$CTC\$_2$
	2	A	$CTGTACCAAC\$_1$		2	A	$CTGTACCAAC\$_1$

	$L_{12}(G)$	$B_{12}(G)$	Sorted Suffixes		$L_{13}(G)$	$B_{13}(G)$	Sorted Suffixes
	0	A	$GAAAGCTC\$_2$		0	A	$GAAAGCTC\$_2$
	1	A	$GCTC\$_2$	$\rightarrow$	**2+1=3**	**$\$_2$**	**GAACAGAAAGCTC$\$_2$**
	1	T	$GTACCAAC\$_1$		**1**	A	$GCTC\$_2$
					1	T	$GTACCAAC\$_1$

	$L_{12}(T)$	$B_{12}(T)$	Sorted Suffixes		$L_{13}(T)$	$B_{13}(T)$	Sorted Suffixes
	0	G	$TACCAAC\$_1$		0	G	$TACCAAC\$_1$
	1	C	$TC\$_2$		1	C	$TC\$_2$
	1	C	$TGTACCAAC\$_1$		1	C	$TGTACCAAC\$_1$

Example: Looking in detail at iteration 13

	$L_{12}(\$)$	$B_{12}(\$)$	Sorted Suffixes		$L_{13}(\$)$	$B_{13}(\$)$	Sorted Suffixes
	0	C	$\$_1$		0	C	$\$_1$
	0	C	$\$_2$		0	C	$\$_2$

	$L_{12}(A)$	$B_{12}(A)$	Sorted Suffixes		$L_{13}(A)$	$B_{13}(A)$	Sorted Suffixes
	0	G	$AAAGCTC\$_2$		0	G	$AAAGCTC\$_2$
$min = 2$	2	C	$AAC\$_1$		2	C	$AAC\$_1$
$\rightarrow$	**3**	**G**	**AACAGAAAGCTC$\$_2$**		3	G	$AACAGAAAGCTC\$_2$
	2	A	$AAGCTC\$_2$		2	A	$AAGCTC\$_2$
	1	A	$AC\$_1$		1	A	$AC\$_1$
	2	A	$ACAGAAAGCTC\$_2$	$\rightarrow$	**1+1=2**	$\$_1$	**ACACTGTACCAAC$\$_1$**
$min = 0$	2	T	$ACCAAC\$_1$		**2+1=3**	A	$ACAGAAAGCTC\$_2$
	2	C	$ACTGTACCAAC\$_1$		2	T	$ACCAAC\$_1$
	1	C	$AGAAAGCTC\$_2$		2	C	$ACTGTACCAAC\$_1$
	2	A	$AGCTC\$_2$		1	C	$AGAAAGCTC\$_2$
					2	A	$AGCTC\$_2$

	$L_{12}(C)$	$B_{12}(C)$	Sorted Suffixes		$L_{13}(C)$	$B_{13}(C)$	Sorted Suffixes
	0	A	$C\$_1$		0	A	$C\$_1$
$min = 1$	1	T	$C\$_2$		1	T	$C\$_2$
	1	C	$CAAC\$_1$		1	C	$CAAC\$_1$
$\rightarrow$	**2**	**A**	**CACTGTACCAAC$\$_1$**		2	G	$CACTGTACCAAC\$_1$
$min = 2$	**2**	**A**	$CAGAAAGCTC\$_2$		2	A	$CAGAAAGCTC\$_2$
	1	A	$CCAAC\$_1$		1	A	$CCAAC\$_1$
	1	G	$CTC\$_2$		1	G	$CTC\$_2$
	2	A	$CTGTACCAAC\$_1$		2	A	$CTGTACCAAC\$_1$

	$L_{12}(G)$	$B_{12}(G)$	Sorted Suffixes		$L_{13}(G)$	$B_{13}(G)$	Sorted Suffixes
	0	A	$GAAAGCTC\$_2$		0	A	$GAAAGCTC\$_2$
	1	A	$GCTC\$_2$	$\rightarrow$	**2+1=3**	$\$_2$	**GAACAGAAAGCTC$\$_2$**
	1	T	$GTACCAAC\$_1$		**0+1=1**	A	$GCTC\$_2$
					1	T	$GTACCAAC\$_1$

	$L_{12}(T)$	$B_{12}(T)$	Sorted Suffixes		$L_{13}(T)$	$B_{13}(T)$	Sorted Suffixes
	0	G	$TACCAAC\$_1$		0	G	$TACCAAC\$_1$
	1	C	$TC\$_2$		1	C	$TC\$_2$
	1	C	$TGTACCAAC\$_1$		1	C	$TGTACCAAC\$_1$

When are the minimum values computed?

We can compute the minimum values useful for the iteration j while we are inserting the new elements in the partial BWT and in the partial LCP in a sequential way during the iteration $j-1$.

In the example, while we build $B_{(12)}$ and $L_{(12)}$ segments, we can compute the minimum values useful for the computation of LCP-values corresponding to 13-suffixes.

We can compute minimum values at the same time for all j-suffixes.

$L_{12}(\$)$	$B_{12}(\$)$	Sorted Suffixes
0	C	$\$_1$
0	C	$\$_2$

$L_{12}(A)$	$B_{12}(A)$	Sorted Suffixes
0	G	$AAAGCTC\$_2$
2	C	$AAC\$_1$
3	**G**	**AACAGAAAGCTC$\$_2$**
2	A	$AAGCTC\$_2$
1	A	$AC\$_1$
2	A	$ACAGAAAGCTC\$_2$
2	T	$ACCAAC\$_1$
2	C	$ACTGTACCAAC\$_1$
1	C	$AGAAAGCTC\$_2$
2	A	$AGCTC\$_2$

$L_{12}(C)$	$B_{12}(C)$	Sorted Suffixes
0	A	$C\$_1$
1	T	$C\$_2$
1	C	$CAAC\$_1$
2	**A**	**CACTGTACCAAC$\$_1$**
2	A	$CAGAAAGCTC\$_2$
1	A	$CCAAC\$_1$
1	G	$CTC\$_2$
2	A	$CTGTACCAAC\$_1$

$L_{12}(G)$	$B_{12}(G)$	Sorted Suffixes
0	A	$GAAAGCTC\$_2$
1	A	$GCTC\$_2$
1	T	$GTACCAAC\$_1$

$L_{12}(T)$	$B_{12}(T)$	Sorted Suffixes
0	G	$TACCAAC\$_1$
1	C	$TC\$_2$
1	C	$TGTACCAAC\$_1$

When are the minimum values computed?

We can compute the minimum values useful for the iteration j while we are inserting the new elements in the partial BWT and in the partial LCP in a sequential way during the iteration $j-1$.

In the example, while we build $B_{(12)}$ and $L_{(12)}$ segments, we can compute the minimum values useful for the computation of LCP-values corresponding to 13-suffixes.

We can compute minimum values at the same time for all j-suffixes.

	$L_{12}(\$)$	$B_{12}(\$)$	Sorted Suffixes
	0	C	$\$_1$
	0	C	$\$_2$

	$L_{12}(A)$	$B_{12}(A)$	Sorted Suffixes
	0	G	$AAAGCTC\$_2$
$min = 2$	2	C	$AAC\$_1$
$\rightarrow$	**3**	**G**	**AACAGAAAGCTC$_2$**
	2	A	$AAGCTC\$_2$
	1	A	$AC\$_1$
	2	A	$ACAGAAAGCTC\$_2$
$min = 0$	2	T	$ACCAAC\$_1$
	2	C	$ACTGTACCAAC\$_1$
	1	C	$AGAAAGCTC\$_2$
	2	A	$AGCTC\$_2$

	$L_{12}(C)$	$B_{12}(C)$	Sorted Suffixes
	0	A	$C\$_1$
$min = 1$	1	T	$C\$_2$
	1	C	$CAAC\$_1$
$\rightarrow$	**2**	**A**	**CACTGTACCAAC$_1$**
$min = 2$	**2**	**A**	$CAGAAAGCTC\$_2$
	1	A	$CCAAC\$_1$
	1	G	$CTC\$_2$
	2	A	$CTGTACCAAC\$_1$

$L_{12}(G)$	$B_{12}(G)$	Sorted Suffixes
0	A	$GAAAGCTC\$_2$
1	A	$GCTC\$_2$
1	T	$GTACCAAC\$_1$

$L_{12}(T)$	$B_{12}(T)$	Sorted Suffixes
0	G	$TACCAAC\$_1$
1	C	$TC\$_2$
1	C	$TGTACCAAC\$_1$

When are the minimum values computed?

We can compute the minimum values useful for the iteration j while we are inserting the new elements in the partial BWT and in the partial LCP in a sequential way during the iteration $j-1$.

In the example, while we build $B_{(12)}$ and $L_{(12)}$ segments, we can compute the minimum values useful for the computation of LCP-values corresponding to 13-suffixes.

We can compute minimum values at the same time for all j-suffixes.

	$L_{12}(\$)$	$B_{12}(\$)$	Sorted Suffixes
	0	C	$\$_1$
	0	C	$\$_2$

	$L_{12}(A)$	$B_{12}(A)$	Sorted Suffixes
	0	G	$AAAGCTC\$_2$
$min = 2$	2	C	$AAC\$_1$
$\rightarrow$	**3**	**G**	**AACAGAAAGCTC$\$_2$**
	2	A	$AAGCTC\$_2$
	1	A	$AC\$_1$
	2	A	$ACAGAAAGCTC\$_2$
$min = 0$	2	T	$ACCAAC\$_1$
	2	C	$ACTGTACCAAC\$_1$
	1	C	$AGAAAGCTC\$_2$
	2	A	$AGCTC\$_2$

	$L_{12}(C)$	$B_{12}(C)$	Sorted Suffixes
	0	A	$C\$_1$
$min = 1$	1	T	$C\$_2$
	1	C	$CAAC\$_1$
$\rightarrow$	**2**	**A**	**CACTGTACCAAC$\$_1$**
$min = 2$	**2**	**A**	$CAGAAAGCTC\$_2$
	1	A	$CCAAC\$_1$
	1	G	$CTC\$_2$
	2	A	$CTGTACCAAC\$_1$

	$L_{12}(G)$	$B_{12}(G)$	Sorted Suffixes
	0	A	$GAAAGCTC\$_2$
	1	A	$GCTC\$_2$
	1	T	$GTACCAAC\$_1$

	$L_{12}(T)$	$B_{12}(T)$	Sorted Suffixes
	0	G	$TACCAAC\$_1$
	1	C	$TC\$_2$
	1	C	$TGTACCAAC\$_1$

Advantages

- The BWT and the LCP are split and kept in σ files.
- Sequentially reading.
  Each iteration $j > 0$ can be divided into two consecutive phases:
  - First phase: we read only the segments B_{j-1} in order to find the positions where we must insert the elements associated with the j-suffixes.
  - Second phase: we read the segments B_{j-1} and L_{j-1} in sequential way for the construction of new segments B_j and L_j and compute the minimum LCP-values for the next iteration.
- Inserting/updating simultaneously of m symbols in the partial BWT and $2m$ values in the partial LCP and computing simultaneously the $2m$ minimum LCP-values for the next iteration.

Moreover, at any iteration j, we can stop the running and by adding the elements corresponding to the end-markers we can obtain the BWT and LCP of the collection of the j-suffixes of S.

Experiments

Notice that an entirely like-for-like comparison between our implementation and the existing implementation requires the concatenation of the strings of the collection, but

- The use of many millions of different end markers could be not practicable.
- The use of the same end marker could lead to values in the LCP array that may exceed the lengths of the strings.

However, preliminary comparisons have shown that our algorithm uses less internal memory than these algorithms.

Experiments

Notice that an entirely like-for-like comparison between our implementation and the existing implementation requires the concatenation of the strings of the collection, but

- The use of many millions of different end markers could be not practicable.
- The use of the same end marker could lead to values in the LCP array that may exceed the lengths of the strings.

However, preliminary comparisons have shown that our algorithm uses less internal memory than these algorithms.

In particular

- bwt_based_laca2[1] computes the LCP of a single string and needs the pre-computed BWT of the string.
- In order to adapt this algorithm for a collection, we have computed the BWT of a collection by using BCR . Such output needs slight modifications because, in general, the BWT of a collection does not coincide with the BWT of a single string.
- Result
  - BCR requires about 5 hours of wallclock time taking only 4Gb of RAM $+$ LCP (computed by bwt_based_laca2) requires 18Gb of RAM to create the LCP in about 2 hours.
  - Our new method extLCP needs 4.7Gb of RAM to create both BWT and LCP in just under 18 hours.
- Attempting to use bwt_based_laca2 to compute the LCP of 800 million of sequences 100 bases long exceeds our available RAM on the 64Gb RAM machine.

[1] T. Beller, S. Gog, E. Ohlebusch, and T. Schnattinger. Computing the longest common prefix array based on the Burrows-Wheeler transform. Journal of Discrete Algorithms. To appear.

Experiments

instance	size in Gb	program	wall clock	efficiency	memory
0043M	4.00	BCR	0.99	0.84	0.57
	4.00	extLCP	3.29	0.98	1.00
0085M	8.00	BCR	1.01	0.83	1.10
	8.00	extLCP	3.81	0.87	2.00
0100M	9.31	BCR	1.05	0.81	1.35
	9.31	extLCP	4.03	0.83	2.30
0200M	18.62	BCR	1.63	0.58	4.00
	18.62	extLCP	4.28	0.79	4.70
0800M	74.51	BCR	3.23	0.43	10.40
	74.51	extLCP	6.68	0.67	18.00

- All reads are 100 bases long.
- wall clock time (the amount of time that elapsed from the start to the completion of the instance) is given as microseconds per input base.
- memory denotes the maximal amount of memory (in gigabytes) used during execution.
- The efficiency column states the CPU efficiency values, i.e. the proportion of time for which the CPU was occupied and not waiting for I/O operations to finish, as taken from the output of the `/usr/bin/time` command.

The `extLCP` algorithm:

- uses $O(mk^2 \log \sigma)$ disk I/O and $O((m + \sigma^2) \log(mk))$ bits of memory.
- takes $O(k(m + \mathrm{sort}(m)))$ CPU time, where $\mathrm{sort}(m)$ is the time taken to sort m integers.

Ongoing works

- Ongoing work:
  - Further optimizations for the construction, for instance by using the parallelization or by using different strategies for I/O operations.
  - Integrate the construction of LCP in the BEETL software library. BEETL for construction/querying of BWT of large string collections can be downloaded from

    http://beetl.github.com/BEETL
  - Bioinformatics applications based on BWT and LCP and by using `extLCP`.

Many thanks for your attention!

Ongoing works

- Ongoing work:
  - Further optimizations for the construction, for instance by using the parallelization or by using different strategies for I/O operations.
  - Integrate the construction of LCP in the BEETL software library. BEETL for construction/querying of BWT of large string collections can be downloaded from

    http://beetl.github.com/BEETL

  - Bioinformatics applications based on BWT and LCP and by using `extLCP`.

Many thanks for your attention!