Computing the BWT and the LCP Array in Constant Space

Felipe A. Louza and Guilherme P. Telles

Institute of Computing, University of Campinas, Brazil

Motivation

- Suffix arrays may be used to solve many string matching and analysis problems.
- In larger scales, suffix arrays may be replaced by other indexes based on the Burrows-Wheeler transform, such as the FM-index.
- The LCP may be used together with the FM-index to solve some approximate string matching problems.

Our article

- Shows how to build the BWT and the LCP array in constant space and $O(n^2)$.
- Builds on the beautiful algorithm by Crochemore *et al.* to build the BWT in-place.
- Points out that other paths may be followed to perform the same construction, but this one is very simple.
- Presents some additional relations between the BWT and the LCP array.

Burrows-Wheeler transform

- A reversible transformation that produces a permutation of a string S which tends to group the occurrences of a symbol into blocks.
- To build the BWT sort all the cyclic rotations of S and get the last column.

i	cyclic rotations	sorted rotations	BWT
0	BANANA$	$BANANA	A
1	$BANANA	A$BANAN	N
2	A$BANAN	ANA$BAN	N
3	NA$BANA	ANANA$B	B
4	ANA$BAN	BANANA$	$
5	NANA$BA	NA$BANA	A
6	ANANA$B	NANA$BA	A

Suffix Array

- An array of integers that stores the order of every suffix of a string in lexicographic order.

	0	1	2	3	4	5	6
T	B	A	N	A	N	A	\$

i	$SA[i]$	$T[SA[i], n-1]$
0	6	\$
1	5	A\$
2	3	ANA\$
3	1	ANANA\$
4	0	BANANA\$
5	4	NA\$
6	2	NANA\$

lcp, LCP array

- The lcp between two strings S and T is the length of their largest common prefix.
- The LCP array stores the lcp value of consecutive suffixes in the suffix array.

i	$SA[i]$	$LCP[i]$	$T[SA[i], n-1]$
0	6	0	\$
1	5	0	A\$
2	3	1	ANA\$
3	1	3	ANANA\$
4	0	0	BANANA\$
5	4	0	NA\$
6	2	2	NANA\$

Relations

i		F L	SA	LCP	BWT	$T[SA[i], n-1]$
0	BANANA$	$BANANA	6	0	A	$
1	$BANANA	A$BANAN	5	0	N	A$
2	A$BANAN	ANA$BAN	3	1	N	ANA$
3	NA$BANA	ANANA$B	1	3	B	ANANA$
4	ANA$BAN	BANANA$	0	0	$	BANANA$
5	NANA$BA	NA$BANA	4	0	A	NA$
6	ANANA$B	NANA$BA	2	2	A	NANA$

- $L[i] = BWT[i]$ and $F[i] = T[SA[i]]$.
- The j-th symbol in the BWT is the symbol that precedes the j-th suffix (in order).
  The 0-th symbol in the BWT precedes $T[n-1]$.
- $BWT[i] = T[SA[i]-1]$ if $SA[i] \neq 0$ or $BWT[i] = \$$ if $SA[i] = 0$.

Related results

	time	space (bits)
In-place suffix sorting:		
• **?**	$O(n \log n)$	$n \log n + n \log \sigma + O(\log n)$ SA + text + $O(1)$ words
In-place BWT:		
• **?**	$O(n^2)$	$n \log \sigma + O(\log n)$ BWT + $O(1)$ words
In-place LCP:		
• ??		
• This work	$O(n^2)$	$n \log n + n \log \sigma + O(\log n)$ LCP + BWT + $O(1)$ words

In place BWT (Crochemore *et al.*)

- Proceeds by induction from the right to the left in T.
- Adds $T[s]$, which corresponds to the suffix T_s.
- The position of $\$$ gives the rank of T_s in the solution already constructed.

In place BWT (Crochemore *et al.*)

Let $BWT(T_s)$ be the BWT for $T[s, n-1]$.

- proceed by induction from right to left in T
  - while maintaining the BWT of the current suffix T_s
  - Incremental step from $BWT(T_{s+1})$
    - Replace \$ with new first character
    - Insert the new suffix and its preceding character \$

N A N A \$

BWT	sorted suffixes
A	\$
N	A \$
N	A N A \$
A	N A \$
\$	N A N A \$

A N A N A \$

BWT	sorted suffixes
A	\$
N	A \$
N	A N A \$
\$	A N A N A \$
A	N A \$
A	N A N A \$

In place BWT (Crochemore *et al.*)

Let $BWT(T_s)$ be the BWT for $T[s, n-1]$.

- proceed by induction from right to left in T
  - while maintaining the BWT of the current suffix T_s
  - Incremental step from $BWT(T_{s+1})$
    - Replace $ with new first character
    - Insert the new suffix and its preceding character $

N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
A	N A $
$	N A N A $

A N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
$	A N A N A $
A	N A $
A	N A N A $

In place BWT (Crochemore *et al.*)

Let $BWT(T_s)$ be the BWT for $T[s, n-1]$.

- proceed by induction from right to left in T
  - while maintaining the BWT of the current suffix T_s
  - Incremental step from $BWT(T_{s+1})$
    - Replace \$ with new first character
    - Insert the new suffix and its preceding character \$

N A N A \$

BWT	sorted suffixes				
A	\$				
N	A	\$			
N	A	N	A	\$	
A	N	A	\$		
\$	N	A	N	A	\$

A N A N A \$

BWT	sorted suffixes					
A	\$					
N	A	\$				
N	A	N	A	\$		
\$		N	A	N	A	\$
A	N	A	\$			
A	N	A	N	A	\$	

In place BWT (Crochemore *et al.*)

Let $BWT(T_s)$ be the BWT for $T[s, n-1]$.

- proceed by induction from right to left in T
  - while maintaining the BWT of the current suffix T_s
  - Incremental step from $BWT(T_{s+1})$
    - ⋆ Replace \$ with new first character
    - ⋆ Insert the new suffix and its preceding character \$

N A N A \$

BWT	sorted suffixes
A	\$
N	A \$
N	A N A \$
A	N A \$
\$	N A N A \$

A N A N A \$

BWT	sorted suffixes
A	\$
N	A \$
N	A N A \$
\$	A N A N A \$
A	N A \$
A	N A N A \$

In place BWT (Crochemore *et al.*)

- The new suffix starts with the character that is being added.
- To determine its position, the algorithm counts:
  - smaller characters
  - preceding equal characters (`rank`)

A N A N A $

BWT	sorted suffixes
A	
N	
N	
$	
A	
A	

In place BWT (Crochemore *et al.*)

- The new suffix starts with the character that is being added.
- To determine its position, the algorithm counts:
  - smaller characters
  - preceding equal characters (`rank`)

A N A N A $

BWT	sorted suffixes
A	$
N	A
N	A
$	A
A	N
A	N

In place BWT (Crochemore *et al.*)

- The new suffix starts with the character that is being added.
- To determine its position, the algorithm counts:
  - smaller characters
  - preceding equal characters (`rank`)

A N A N A $

BWT	sorted suffixes
A	$
N	A
N	A
$	A
A	N
A	N

In place BWT (Crochemore *et al.*)

- The new suffix starts with the character that is being added.
- To determine its position, the algorithm counts:
  - smaller characters
  - preceding equal characters (`rank`)

A N A N A $

BWT	sorted suffixes
A	$
N	A
N	A
$	A
A	N
A	N

In place BWT (Crochemore *et al.*)

- The new suffix starts with the character that is being added.
- To determine its position, the algorithm counts:
  - smaller characters
  - preceding equal characters (`rank`)

A N A N A $

BWT	sorted suffixes
A	$
N	A
N	A
$	A
A	N
A	N

In place BWT (Crochemore *et al.*)

- The new suffix starts with the character that is being added.
- To determine its position, the algorithm counts:
  - smaller characters
  - preceding equal characters (`rank`)

A	N	A	N	A	$

BWT	sorted suffixes
A	$
N	A
N	A
$	A
A	N
A	N

BWT and LCP

- Adding a character to the solution will demand computing the values of the LCP for T_s and its neighbors only.
- The first LCP value involves T_s and the largest suffix T_a in $\{T_{s+1}, \ldots, T_{n-1}\}$ that is smaller than T_s.
- The second LCP value involves T_s and the smallest suffix T_b in $\{T_{s+1}, \ldots, T_{n-1}\}$ that is larger than T_s.

BWT and LCP

2' Find the position p_{a+1} of the suffix T_{a+1}, such that suffix T_a has rank r in $BWT(T_{s+1})$, and compute:

$$\ell_a = \begin{cases} RMQ(p_{a+1}, p) + 1 & \text{if } T[p_{a+1}] = T[s] \\ 0 & \text{otherwise.} \end{cases}$$

2'' Find the position p_{b+1} of the suffix T_{b+1}, such that suffix T_b has rank $r+1$ in $BWT(T_{s+1})$, and compute:

$$\ell_b = \begin{cases} RMQ(p, p_{b+1}) + 1 & \text{if } T[s] = T[p_{b+1}] \\ 0 & \text{otherwise.} \end{cases}$$

4' Shift $LCP[s+1, r]$ one position to the left, store ℓ_a in $LCP[r]$ and if $r+1 < n$ then store ℓ_b in $LCP[r+1]$.

RMQ

- The RMQ (range minimum query) with respect to the LCP is the largest lcp value in an interval of the suffix array.
- $RMQ(i, j) = \min_{i<k\leq j}\{LCP[k]\}$, for $0 \leq i < j < n$.
- Given a string T with length n and its LCP array it is easy to see that $LCP(T_{SA[i]}, T_{SA[j]}) = RMQ(i, j)$.

LCP and BWT

Let the current symbol be $c = T[s]$.

BWT:

1. $p =$ position of \$ in $BWT(T_{s+1})$
   - replace \$ by c
2. $r =$ rank of the suffix T_s using just symbol c

LCP:

- $a =$ select $(r-1)^{th}$ suffix in $BWT(T_s)$
  - $LCP[r] = min_{LCP}(a, p) + 1$ (if $T[a] = T[p]$)
- $b =$ select $(r+1)^{th}$ suffix in $BWT(T_s)$
  - $LCP[r+1] = min_{LCP}(p, b) + 1$ (if $T[p] = T[b]$)

A N A N A \$

		BWT	sorted suffixes	LCP
		A	\$	0
b	$\rightarrow$	N	A \$	0
		N	A N A \$	1
r	$\rightarrow$	\$	A N A N A \$	
a	$\rightarrow$	A	N A \$	0
p	$\rightarrow$	\$	N A N A \$	2

LCP and BWT

Let the current symbol be $c = T[s]$.

BWT:

1. $p =$ position of \$ in $BWT(T_{s+1})$
   - replace \$ by c
2. $r =$ rank of the suffix T_s using just symbol c

LCP:

- $a =$ select $(r-1)^{th}$ suffix in $BWT(T_s)$
  - $LCP[r] = min_{LCP}(a, p) + 1$ (if $T[a] = T[p]$)
- $b =$ select $(r+1)^{th}$ suffix in $BWT(T_s)$
  - $LCP[r+1] = min_{LCP}(p, b) + 1$ (if $T[p] = T[b]$)

A N A N A \$

		BWT	sorted suffixes	LCP
		A	\$	0
b	$\rightarrow$	N	A \$	0
		N	A N A \$	1
r	$\rightarrow$	\$	A N A N A \$	
a	$\rightarrow$	A	N A \$	0
p	$\rightarrow$	A	N A N A \$	2

LCP and BWT

Let the current symbol be $c = T[s]$.

BWT:

1. $p =$ position of \$ in $BWT(T_{s+1})$
   - replace \$ by c
2. $r =$ rank of the suffix T_s using just symbol c

LCP:

- $a =$ select $(r-1)^{th}$ suffix in $BWT(T_s)$
  - $LCP[r] = min_{LCP}(a, p) + 1$ (if $T[a] = T[p]$)
- $b =$ select $(r+1)^{th}$ suffix in $BWT(T_s)$
  - $LCP[r+1] = min_{LCP}(p, b) + 1$ (if $T[p] = T[b]$)

A N A N A \$

		BWT	sorted suffixes	LCP
		A	\$	0
b	$\rightarrow$	N	A \$	0
		N	A N A \$	1
r	$\rightarrow$	\$	A N A N A \$	??
a	$\rightarrow$	A	N A \$	0
p	$\rightarrow$	A	N A N A \$	2

LCP and BWT

Let the current symbol be $c = T[s]$.

BWT:

1. $p =$ position of \$ in $BWT(T_{s+1})$
   - replace \$ by c
2. $r =$ rank of the suffix T_s using just symbol c

LCP:

- $a =$ select $(r-1)^{th}$ suffix in $BWT(T_s)$
  - $LCP[r] = min_{LCP}(a, p) + 1$ (if $T[a] = T[p]$)
- $b =$ select $(r+1)^{th}$ suffix in $BWT(T_s)$
  - $LCP[r+1] = min_{LCP}(p, b) + 1$ (if $T[p] = T[b]$)

A N A N A $

		BWT	sorted suffixes	LCP
		A	$	0
b	$\rightarrow$	N	A $	0
		N	A N A $	1
r	$\rightarrow$	$	A N A N A $	3
a	$\rightarrow$	A	N A $	0
p	$\rightarrow$	A	N A N A $	2

LCP and BWT

Let the current symbol be $c = T[s]$.

BWT:

1. $p =$ position of \$ in $BWT(T_{s+1})$
   - replace \$ by c
2. $r =$ rank of the suffix T_s using just symbol c

LCP:

- $a =$ select $(r-1)^{th}$ suffix in $BWT(T_s)$
  - $LCP[r] = min_{LCP}(a, p) + 1$ (if $T[a] = T[p]$)
- $b =$ select $(r+1)^{th}$ suffix in $BWT(T_s)$
  - $LCP[r+1] = min_{LCP}(p, b) + 1$ (if $T[p] = T[b]$)

A N A N A \$

		BWT	sorted suffixes	LCP
		A	\$	0
b	$\rightarrow$	N	A \$	0
		N	A N A \$	1
r	$\rightarrow$	\$	A N A N A \$	3
a	$\rightarrow$	A	N A \$	??
p	$\rightarrow$	A	N A N A \$	2

LCP and BWT

Let the current symbol be $c = T[s]$.

BWT:

1. $p =$ position of \$ in $BWT(T_{s+1})$
   - replace \$ by c
2. $r =$ rank of the suffix T_s using just symbol c

LCP:

- $a =$ select $(r-1)^{th}$ suffix in $BWT(T_s)$
  - $LCP[r] = min_{LCP}(a, p) + 1$ (if $T[a] = T[p]$)
- $b =$ select $(r+1)^{th}$ suffix in $BWT(T_s)$
  - $LCP[r+1] = min_{LCP}(p, b) + 1$ (if $T[p] = T[b]$)

A N A N A \$

		BWT	sorted suffixes	LCP
		A	\$	0
b	$\rightarrow$	N	A \$	0
		N	A N A \$	1
r	$\rightarrow$	\$	A N A N A \$	3
a	$\rightarrow$	A	N A \$	0
p	$\rightarrow$	A	N A N A \$	2

Final remarks

- Although $O(n^2)$ these algorithms explore interesting relations among the structures.
- Only a constant number of additional variables is needed.
- The C code is quite short and clean.

```
void compute_bwt_lcp(unsigned char *T, int n, int *LCP){
int i, p, r=1, s, p_a1, p_b1, l_a, l_b;
LCP[n-1] = LCP[n-2] = 0; // base cases

for (s=n-3; s>=0; s--) {

   /*steps 1 and 2*/
   p=r+1;
   for (i=s+1, r=0; T[i]!=END_MARKER; i++)
     if(T[i]<=T[s]) r++;
   for (; i<n; i++)
     if (T[i]<T[s]) r++;

   /*step 2'*/
   p_a1=p+s-1;
   l_a=LCP[p_a1+1];
   while (T[p_a1]!=T[s]) // RMQ function
     if (LCP[p_a1--]<l_a)
       l_a=LCP[p_a1+1];
   if (p_a1==s) l_a=0;
   else l_a++;

   /*step 2''*/
   p_b1=p+s+1;
   l_b=LCP[p_b1];
   while (T[p_b1]!=T[s] && p_b1<n) // RMQ function
     if (LCP[++p_b1]<l_b)
       l_b=LCP[p_b1];
   if (p_b1==n) l_b=0;
   else l_b++;

   /*steps 3 and 4*/
   T[p+s]=T[s];
   for (i=s; i<s+r; i++) {
     T[i]=T[i+1];
     LCP[i]=LCP[i+1];
   }
   T[s+r]=END_MARKER;

   /*step 4'*/
   LCP[s+r]=l_a;
   if (s+r+1<n)  // If r+1 is not the last position
     LCP[s+r+1]=l_b;
  }
}
```

Thank you!

References I

The road not taken

IP-BWT

1. Find the position p of \$ in $T[s+1, n-1]$. Evaluating $p-s$ gives the local rank of T_{s+1} that originally was starting at position $s+1$.
2. Find the local rank r of the suffix T_s using just symbol $c = T[s]$. To this end, sum the number of symbols in $T[s+1, n-1]$ that are strictly smaller than c with the number of occurrences of c in $T[s+1, p]$ and with s, obtaining r.
3. Store c into $T[p]$, replacing \$.
4. Shift $T[s+1, r]$ one position to the left. Write \$ at $T[r]$.