String-searching Algorithms

Burrows Wheeler Transform

Sergio Peignier sergio.peignier@insa-lyon.fr

Associate Professor
INSA Lyon
Biosciences department

Alphabetical order

Characters conventional **ordering** system:

$A < B < C < .... < Z$

Lexicographic order

Alphabetical order **strings extension**:
Order depends on the **alphabetic order** between the **characters** in the **first position** where the **strings differ**

- **Same length strings comparison**
  Let $S = (s_1, ..., s_n)$ and $T = (t_1, ..., t_n)$ two strings. $S < T$ if $\exists i \in \{1, 2, ..., n\} \mid s_i < t_i$ and $s_j = t_j \; \forall j < i$

- **Different length strings comparison**
  - Let \$ be the "end of string" character, s.t. $\$ < A$
  - Let $S = (s_1, ..., s_n)$ and $T = (t_1, ..., t_m)$ two strings (s.t. $s_n = \$$ and $t_m = \$$) and $n < m$.
    $S < T$ if $\exists i \in \{1, 2, ..., n\} \mid s_i < t_i$ and $s_j = t_j \; \forall j < i$

Exercise: Order the strings $T[1, 30]$ and $T[1, 5]$

Circular Shift

Let $R : \mathcal{S}, k \rightarrow \mathcal{S}$ s.t. $R(T, k) = T'$ with:

$T'[1, |T| - k] = T[k + 1, |T|]$ and $T'[|T| - k + 1, |T|] = T[1, k]$

Example

T = ”dans l’herbe noire Les Kobolds vont$”

R(T,3) = ”s l’herbe noire Les Kobolds vont$dan”

R(T,6) = ?

Burrows Wheeler Transform

Let $T = (T_1, T_2, \ldots T_n)$ be a string of length n.

- Let r be the list of **all circular shifts** of T, sorted by their **lexical orders**:
  $r = (R(T, k_1), R(T, k_2), \ldots, R(T, k_n) \quad | \quad R(T, k_1) \leq R(T, k_2) \leq \ldots \leq R(T, k_n)\ )$

- Let the **Burrow Wheeler Transform** be a function $BWT : \mathcal{S} \to \mathcal{S}$ s.t.
  $BWT(T) = (R(T, k_1)[n], R(T, k_2)[n], \ldots, R(T, k_n)[n])$

Burrows Wheeler Transform | Example

Original String: $S = ACATACAGATG\$$

Sorted circular shifts matrix:

$ACATACAGAT**G**

ACAGATG$ACA**T**

ACATACAGATG**$**

AGATG$ACATA**C**

ATACAGATG$A**C**

ATG$ACATACA**G**

CAGATG$ACAT**A**

CATACAGATG$**A**

G$ACATACAGA**T**

GATG$ACATAC**A**

TACAGATG$AC**A**

TG$ACATACAG**A**

Burrow Wheeler Transform: $BWT(S) = \text{GT\$CCGAATAAA}$

Data Compression

- **BWT**: **pre-processing** for data compression
- In practice BWT rearranges a string into **pieces of repeated characters**
- Compression applied to **BWT** sequence.
  e.g. **run-length encoding algorithm** (sequence of repeated characters stored as a single character followed by its number of repeats).

Example

original string : AAAAAAATTTTTGGGGTGTTTTTT

Compressed string : A7T5G4TGT6

BWT and suffix table

Circular shifts and suffixes:

Let $T' = R(T, k)$ be the k-th circular shift of string T.

$T'[1, |T| - k] = T[k + 1, |T|]$ and $T'[|T| - k + 1, |T|] = T[1, k]$

$\implies T' = (T_{k+1}, \ldots T_{|T|-1}\$, T_1, \ldots, T_k)$

$\implies T' = Suffix^T_{k+1,.}, \$, Prefix^T_{.,k}$

$T'[1, |T| - k]$ is the suffix of $T'[|T| - k + 1, |T|]$ in string T

$\rightarrow$ **Suffix table** and **sorted circular shifts** have **same order**.

$\rightarrow$ BWT sequence:

characters $T[k]$ sorted by their right contexts $T[k + 1, |T|]$

BWT repeated sequences

- In real sequences some characters tend to **co-occur** with higher frequency
- **Sorting** characters according to their **suffixes** (write context) will **bring together similar characters**

Example

- T = long document about dogs.
- There are chances that many characters ”d” in *T* will have a suffixes ”og...”
- Sorting characters by their suffixes will bring together many ”d”, alternated with other characters (e.g., ”c” for ’cog’) in the BWT.

Naive Inverse BWT

$$T \xrightarrow{BWT} T_{BWT} \xrightarrow{\text{compression}} T_{comp} \xrightarrow{\text{decompression}} T_{BWT} \xrightarrow{BWT^{-1}} T$$

$CSM_i[j]$: j-th char. of i-th col. of **circular shifts matrix**

Goal: reconstruct *CSM*

- $CSM_{|T|} = T_{BWT}$; CSM_1: Sorted characters of T (or T_{BWT})
- Given the circular shift:
  $\forall i$ if $T[i] = T_{bwt}[i]$, then $T[i+1] = CSM_1[i]$
  $\{(T_{bwt}[i], CSM_1[i])\ \forall i\}$: successive characters pairs in T
- CSM_1 and CSM_2 : sorted pairs of successive characters
- Again, paste the characters in T_{bwt} to those of CSM_1 and CSM_2 to get the set of all triplets ...
- T is the row of *CSM* with character \$ at the end

Naive Inverse BWT | Example

$\rightarrow S_{bwt}$ = GT$CCGAATAAA

$\rightarrow$ Sort S_{bwt} to get CSM_1 = $AAAAACGGTT

$\rightarrow$ Get pairs $\{\forall i \quad (S_{bwt}[i], CSM_1[i])\}$

= {G$, TA, $A, CA, CA, GA, AC, AC, TG, AG, AT, AT}

$\rightarrow$ Sort the pairs to get CSM_1 and CSM_2:

=($A, AC, AC, AG, AT, AT, CA, CA, G$, GA, TA, TG)

CSM_2 = ACCGTTAA$AAG

$\rightarrow$ Get triplets: $\{\forall i \quad (S_{bwt}[i], CSM_1[i], CSM_2[i])\}$ =

{G$A, TAC, $A, CAG, CAT, GAT, ACA, ACA, TG$, AGA, ATA, ATG}

...

Efficient inverse BWT | First Last property

Let T_{BWT} characters index according to their order of appearance (the k-th char. A in T is index A_k)[1].

This characters indexing is preserved in CSM_1, given the lexical sorting in BWT: $xxxx < yyyy < zzzz$

$$\begin{array}{cc} \text{xxxxx} & G_1 \\ \dots & \dots \\ \text{yyyyy} & G_2 \\ \dots & \dots \\ \text{zzzzz} & G_3 \end{array} \Longrightarrow \begin{array}{cc} \dots & \dots \\ G_1 & \text{xxxxx} \\ G_2 & \text{yyyyy} \\ G_3 & \text{zzzzz} \\ \dots & \dots \end{array}$$

[1] Example: $(A_1, A_2, T_1, A_3, C_1, C_2)$

First Last property | Example

$ACATACAGATG_1$

$ACAGATG\$ACAT_1$

$ACATACAGATG\$_1$

$AGATG\$ACATAC_1$

$ATACAGATG\$AC_2$

$ATG\$ACATACAG_2$

$CAGATG\$ACATA_1$

$CATACAGATG\$A_2$

$G\$ACATACAGAT_2$

$GATG\$ACATACA_3$

$TACAGATG\$ACA_4$

$TG\$ACATACAGA_5$

Efficient inverse BWT | Intuition

Characters in CSM_1 are:

- **Sorted** according to the **lexicographic order**
- **Indexed** as in S_{bwt} (FL property)

Goal:

- Compute the location in CSM_1 of each char. from S_{bwt}
- Get its left context in the same location in S_{bwt}

Efficient inverse BWT | Algorithm

Let the concatenation of strings T' and T be:
$T' + T = (T'[1], \ldots, T'[|T'|], T[1], \ldots, T[|T|])$

Let $\mathcal{A}$ denote the alphabet and S_{bwt} the BWT of S

Initialization

- $\#X$ be the occurrences of character $X \in \mathcal{A}$ in S_{bwt}
- Index the chars of S_{bwt} by their order of appearance.
  K is the list of indexes, s.t. $K[i]$ is the index of $S_{bwt}[i]$
- By definition:
  $CSM_1[1] = \$$, $S[|S|] = \$$ and thus $S[|S| - 1] = S_{bwt}[1]$
- $X \leftarrow S_{bwt}[1]$ denote the current character
  And $k \leftarrow K[1]$ its corresponding index.
- $S \leftarrow \$$

Efficient inverse BWT | Algorithm

Repeat while $X \neq \$$:

- $S \leftarrow (X) + S$
- Let j be the location of X in the First column
  $j \leftarrow k + \sum_Y \#Y$ for all character $Y \in A$ such that $Y < X$
- $S_{bwt}[j]$ has X as right context, and thus it is its right context:
  - $X \leftarrow S_{bwt}[j]$
  - $k \leftarrow K[j]$

Exercise: Compute the complexities of the Naive and efficient inverse BWT

String search using the BWT

A similar procedure allows to find a sub-string T in S

initialization

- $L \leftarrow BWT(S),\ F \leftarrow CSM_1$
- $e \leftarrow 1,\ f \leftarrow |F|\ , i \leftarrow |T|$

while $e < f$ **and** $i > 0$**:**

- $X \leftarrow T[i]$
- $r \leftarrow$ first position of $T[i]$ in L[e,f]
- $s \leftarrow$ last position of $T[i]$ in L[e,f]
- $e \leftarrow$ index of the r-th occurrence of $T[i]$ in F
- $f \leftarrow$ index of the s-th occurrence of $T[i]$ in F
- $i \leftarrow i - 1$

Exercise: Compute the complexity