RLPBWT

Davide Cozzi

Dipartimento di Informatica, Sistemistica e Comunicazione (DISCo)
Università degli Studi di Milano Bicocca

Outline

1. RLPBWT
2. Example

Outline

1. RLPBWT
2. Example

Some definitions

The permutation, panel M, $n \times m$

In *RLPBWT* we have a permutation π_j, $\forall\ 1 \leq j \leq m$ that stably sorts the bits of the j-th column of the PBWT.
This permutation can be stored in space proportional to the number of runs in the j-th column of the PBWT

Some definitions

The permutation, panel M, $n \times m$

In *RLPBWT* we have a permutation π_j, $\forall\ 1 \leq j \leq m$ that stably sorts the bits of the j-th column of the PBWT.
This permutation can be stored in space proportional to the number of runs in the j-th column of the PBWT

The positions in the columns of the PBWT of the bits in the i-th row of M are:

Some definitions

The permutation, panel M, $n \times m$

In *RLPBWT* we have a permutation π_j, $\forall\ 1 \leq j \leq m$ that stably sorts the bits of the j-th column of the PBWT.
This permutation can be stored in space proportional to the number of runs in the j-th column of the PBWT

The positions in the columns of the PBWT of the bits in the i-th row of M are:

$$i, \pi_1(i), \pi_2(\pi_1(i)), \ldots, \pi_{m-1}(\cdots(\pi_2(\pi_1(i)))\cdots)$$

Extracting the bits of the i-th row of M reduces to iteratively applying the π_{m-1} permutations, corresponding to iteratively apply LF in a standard BWT

The permutation

Computing the permutation

$$\pi_j(p) = \begin{cases} p - count_1 & \text{if } column[pref[p]] = 0 \\ count_0 + count_1 - 1 & \text{if } column[pref[p]] = 1 \end{cases}$$

- $count_0$: total number of zeros in the PBWT column
- $count_1$: number of ones in the PBWT column as far as index p

"LF-mapping" in Durbin's algorithm

$$w(i, \sigma) = \begin{cases} u[i] & \text{if } \sigma = 0 \\ c + v[i] & \text{if } \sigma = 1 \end{cases}$$

- c: total number of zeros in the column
- $u[i]$: number of zeros in the column as far as index i
- $v[i]$: number of ones in the column as far as index i

Travis's example

	1	2	3	4	5	6	7	8	9	10	11	12
0	0 1	0 1	0 0	0 0	0 0	0 1	0 0	0 0	0 1	0 1	0 1	0 1
1	0 1	0 1	0 0	0 0	0 0	0 1	0 0	0 0	0 1	0 1	0 **1**	0 1
2	0 1	0 1	0 1	0 0	0 0	0 0	0 1	0 1	0 1	0 0	0 1	0 1
3	0 1	0 1	0 0	0 **0**	0 **0**	0 **1**	0 0	0 0	0 1	0 1	0 0	0 1
4	0 1	0 0	0 1	0 0	0 0	0 1	0 0	0 0	0 1	0 1	0 0	0 1
5	0 1	0 0	0 1	0 0	0 0	0 0	0 0	0 0	0 1	0 **0**	0 0	0 1
6	0 1	0 0	0 1	0 0	0 0	0 0	0 0	0 0	0 1	0 0	0 0	0 0
7	0 1	0 1	0 1	0 0	0 0	0 0	0 0	0 0	0 0	1 1	0 0	0 1
8	0 0	0 1	0 **0**	0 0	0 0	0 1	0 0	0 0	0 0	1 1	0 0	0 1
9	0 **1**	1 0	0 0	0 0	0 0	0 1	0 0	0 0	0 0	1 0	0 0	0 1
10	0 1	1 1	0 0	0 0	0 0	0 0	0 0	0 0	0 1	1 1	0 0	0 1
11	0 0	1 1	1 0	0 1	0 1	0 0	0 0	0 0	0 1	1 0	0 0	0 1
12	0 0	1 1	1 0	0 0	0 1	0 0	0 0	0 0	0 0	1 0	0 0	0 1
13	0 0	1 0	1 0	0 0	0 1	0 0	0 0	0 0	0 0	1 0	1 0	0 1
14	0 0	1 0	1 0	0 0	0 0	0 0	1 0	0 0	0 **0**	1 0	1 0	0 1
15	0 0	1 0	1 0	1 0	0 0	0 0	1 0	0 **0**	0 0	1 0	1 0	0 1
16	0 1	1 0	1 0	1 0	0 0	0 0	1 **0**	0 0	1 1	1 0	1 0	0 1
17	0 0	1 **0**	1 0	1 0	0 0	1 0	1 0	0 1	1 1	1 0	1 0	1 1
18	1 0	1 0	1 0	1 0	0 0	1 0	1 0	0 1	1 1	1 0	1 0	1 **1**
19	1 0	1 0	1 0	1 0	1 0	1 0	1 0	1 1	1 1	1 0	1 0	1 1

The compressed data structure

The tables

- a set of m tables in which the m-th table stores only the positions of the run-heads in the m-th column and a bool to check the first symbol: 0 or 1
- the i-th row of the j-th table stores a quadruple

The compressed data structure

The tables

- a set of m tables in which the m-th table stores only the positions of the run-heads in the m-th column and a bool to check the first symbol: 0 or 1
- the i-th row of the j-th table stores a quadruple

The quadruple

1. the position p of the i-th run-head in the j-th column of the PBWT
2. the permutation $\pi_j(p)$
3. the index of the run containing bit $\pi_j(p)$ in the $(j+1)$-st column of the PBWT
4. the threshold, that's the index of the minimum *LCP value* (current column minus divergence array value) in the run

Row extraction

First step

We start by finding the row of the first table that starts with the position p of the head of the run containing bit i in first column of the PBWT, computing:

Row extraction

First step

We start by finding the row of the first table that starts with the position p of the head of the run containing bit i in first column of the PBWT, computing:

$$\pi_1(i) = \pi_1(p) + i - p$$

Row extraction

First step

We start by finding the row of the first table that starts with the position p of the head of the run containing bit i in first column of the PBWT, computing:

$$\pi_1(i) = \pi_1(p) + i - p$$

looking up the row for the run containing bit $\pi_1(p)$ in the the second table and scanning down the table until we find the row for the run containing bit $\pi_1(i)$

Next step

We continue repeating this procedure for each column

Travis's example I

	table 1			table 2			table 3			table 4			table 5			table 6		
0	0	9	3	0	11	4	0	0	0	0	0	0	0	0	0	0	14	2
1	8	0	0	4	0	0	2	15	2	11	19	2	11	17	5	2	0	0
2	9	17	5	7	15	4	3	2	0	12	11	1	14	11	5	3	16	2
3	11	1	0	9	3	2	4	16	2							5	1	0
4	16	19	5	10	17	4	8	3	0							8	18	2
5	17	6	1	13	4	3										10	4	2

	table 7			table 8			table 9			table 10			table 11			table 12
0	0	0	0	0	0	0	0	7	4	0	13	1	0	17	2	0
1	2	19	3	2	16	4	7	0	0	2	0	0	3	0	0	6
2	3	2	1	3	2	0	10	14	7	3	15	1				7
3				17	17	4	12	3	2	5	1	0				
4							16	16	7	7	17	1				
5										9	3	1				
6										10	19	1				
7										11	4	1				

Travis's example II

Extraction of row 9, $\pi_j(i) = \pi_j(p) + i - p$

- $\pi_1(9) = 17 + 9 - 9 = 17$
- $\pi_2(17) = 4 + 17 - 13 = 8$
- $\pi_3(8) = 4 + 8 - 8 = 3$
- $\pi_4(3) = 0 + 3 - 0 = 3$
- $\pi_5(3) = 0 + 3 - 0 = 3$
- $\pi_6(3) = 16 + 3 - 3 = 16$
- $\pi_7(16) = 2 + 16 - 3 = 15$
- $\pi_8(15) = 2 + 15 - 3 = 14$
- $\pi_9(14) = 3 + 14 - 12 = 5$
- $\pi_{10}(5) = 1 + 5 - 5 = 1$
- $\pi_{11}(1) = 17 + 1 - 0 = 18$

Outline

The matrixes

Panel and query

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	1	0	1	0	0	1	1	1	1	1	0	0	1	0	0	1	0	0	1
0	1	0	0	0	0	1	1	1	1	1	0	0	1	1	1	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	1	0	1	0
1	0	0	1	1	0	1	0	1	0	0	0	1	1	1	0	0	0	1	0
0	1	1	0	1	1	1	1	1	0	0	1	0	0	1	1	1	1	0	0
1	1	0	0	1	0	1	0	1	0	1	0	1	0	0	0	1	1	1	1
0	0	0	1	0	1	1	1	1	1	1	1	0	0	1	0	0	0	1	1
0	0	1	0	1	1	1	1	1	1	1	0	0	1	0	0	0	1	1	1

PBWT Matrix

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	1	0	1	0	0	1	0	0	1	1	0	1	1	1	0	1	0	1	1
0	0	0	1	1	0	0	1	1	0	0	1	1	1	1	0	1	0	1	0
0	1	0	1	1	1	1	1	1	0	0	0	0	0	0	0	1	0	1	1
1	0	0	0	0	0	1	0	1	1	1	1	0	0	0	1	0	1	1	0
0	1	1	1	0	0	1	0	1	1	1	0	0	1	1	0	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1	0	1	0	0	1	1	0	1	0
0	1	0	0	0	0	1	1	1	0	1	1	0	0	1	0	0	1	0	1

Prefix and Divergence Arrays

Prefix Arrays

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	2	2	1	1	1	2	2	2	5	3	3	1	4	5	5	6	6	0
1	2	6	6	5	2	2	1	5	5	3	4	5	0	6	2	2	3	3	4
2	4	3	3	4	6	0	0	3	3	4	5	1	4	5	0	0	1	1	6
3	6	1	1	2	0	5	5	1	1	2	2	0	6	2	4	6	5	2	3
4	0	4	0	6	5	3	3	0	0	1	1	4	3	1	6	3	2	0	1
5	3	0	5	3	4	6	6	6	6	0	0	2	5	0	1	4	0	5	2
6	5	5	4	0	3	4	4	4	4	6	6	6	2	3	3	1	4	4	5

LCP Arrays: current *k* minus the original Durbin's divergence arrays

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	3	1	2	0	1	0	6	3	1	9	3	3	4	3	4	1
0	1	1	2	1	5	4	3	4	5	2	0	2	1	1	1	2	1	2	0
0	1	0	1	0	3	1	2	0	1	0	1	8	2	2	0	1	0	1	5
0	0	2	2	4	0	2	3	4	5	1	2	0	0	0	4	2	5	4	3
0	1	1	3	3	2	0	1	2	3	6	7	1	2	10	1	0	3	0	2
0	1	2	0	2	1	1	2	3	4	4	5	3	1	1	2	2	1	2	1

Run-Length PBWT I, *p*, *perm*, *next perm*, *threshold*

[0, 1, 2, 3, 4]

$$
\begin{matrix} 0 & 4 & 4 & 0 \\ 1 & 0 & 0 & 1 \\ 3 & 5 & 5 & 3 \\ 4 & 2 & 2 & 4 \\ 5 & 6 & 6 & 5 \\ 6 & 3 & 3 & 6 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 3 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 2 & 4 & 1 & 2 \\ 3 & 1 & 0 & 3 \\ 4 & 5 & 2 & 4 \\ 5 & 2 & 0 & 5 \\ 6 & 6 & 2 & 6 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 0 & 0 & 0 \\ 4 & 6 & 3 & 4 \\ 5 & 4 & 2 & 5 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 3 & 2 & 0 \\ 3 & 0 & 0 & 3 \\ 4 & 6 & 4 & 4 \\ 5 & 1 & 1 & 6 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 0 & 0 & 0 \\ 1 & 4 & 2 & 2 \\ 3 & 1 & 0 & 3 \\ 5 & 6 & 4 & 5 \\ 6 & 3 & 2 & 6 \end{matrix}
$$

[5, 6, 7, 8, 9]

$$
\begin{matrix} 0 & 0 & 0 & 0 \\ 2 & 5 & 2 & 2 \\ 3 & 2 & 2 & 4 \\ 5 & 6 & 2 & 5 \\ 6 & 4 & 2 & 6 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 2 & 2 & 1 & 5 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 0 & 0 & 0 \\ 1 & 3 & 1 & 1 \\ 3 & 1 & 1 & 3 \\ 5 & 5 & 1 & 5 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 3 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 3 & 2 & 0 \\ 1 & 0 & 0 & 1 \\ 3 & 4 & 2 & 3 \\ 6 & 2 & 1 & 6 \end{matrix}
$$

Run-Length PBWT II

[10, 11, 12, 13, 14]

$$
\begin{matrix} 0 & 2 & 2 & 0 \\ 1 & 0 & 0 & 2 \\ 3 & 3 & 3 & 3 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 1 \\ 2 & 1 & 0 & 2 \\ 3 & 5 & 2 & 3 \\ 4 & 2 & 1 & 4 \\ 6 & 6 & 3 & 6 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 4 & 2 & 0 \\ 2 & 0 & 0 & 4 \\ 5 & 6 & 3 & 5 \\ 6 & 3 & 1 & 6 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 4 & 2 & 0 \\ 2 & 0 & 0 & 2 \\ 4 & 6 & 4 & 4 \\ 5 & 2 & 1 & 6 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 3 & 1 & 0 \\ 2 & 0 & 0 & 2 \\ 4 & 5 & 3 & 4 \\ 5 & 2 & 0 & 5 \\ 6 & 6 & 4 & 6 \end{matrix}
$$

[15, 16, 17, 18, 19]

$$
\begin{matrix} 0 & 0 & 0 & 0 \\ 3 & 5 & 2 & 3 \\ 4 & 3 & 1 & 4 \\ 5 & 6 & 3 & 5 \\ 6 & 4 & 1 & 6 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 3 & 1 & 0 \\ 3 & 0 & 0 & 3 \\ 5 & 6 & 3 & 5 \\ 6 & 2 & 0 & 6 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 0 & 0 & 0 \\ 3 & 5 & 2 & 3 \\ 4 & 3 & 0 & 5 \\ 6 & 6 & 3 & 6 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 2 & 2 & 0 \\ 4 & 0 & 0 & 4 \\ 5 & 6 & 4 & 5 \\ 6 & 1 & 1 & 6 \end{matrix}
\Longrightarrow
\begin{matrix} 0 & 4 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 2 & 5 & 0 & 2 \\ 3 & 1 & 0 & 5 \\ 6 & 6 & 0 & 6 \end{matrix}
$$

Match with external haplotype I

First case, bits matches at column j-th

- we are looking at d-th bit of the k-th run, that come from the i-th row of the panel
- if this bit match the next bit of the pattern we can go to column $j+1$ and we figure out which bit to look at in that column
- the next bit we look at is still from row i-th

Match with external haplotype II

Second case, bits doesn't matches at column j-th

- we are looking at d-th bit of the k-th run and that bit doesn't match the next bit in the pattern
- we look at the threshold for the k-th run:
  - if d is at most the threshold (check this "at most") than we move to the last bit of the $(k-1)$-st run in the j-th column and then we proceed as in *case 1*
  - if d is greater than the threshold than we move to the first bit of the $(k-1)$-st run in the j-th column and then we proceed as in *case 1*