

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:

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

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 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

Outline

1 RLPBWT

2 Example

The Panel

Panel

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	1	0	1	0	0
1	0	0	1	1	0	1	0	1	0	0	0	1	1	1	0	0	1	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

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	1	0	1	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

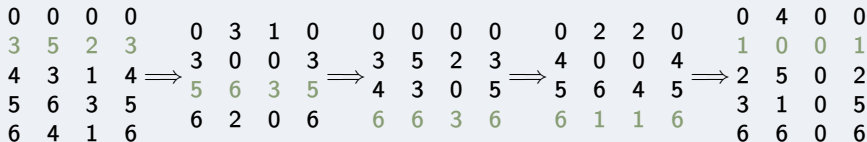
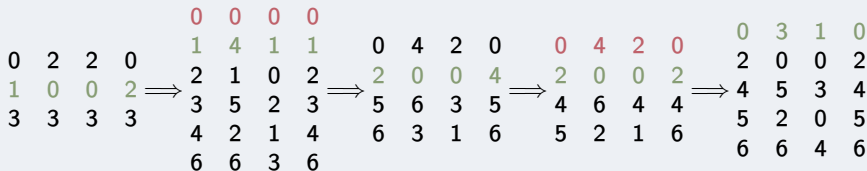
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	2	2	1	1	1	2	2	2	5	3	3	1	4	5	5	6	6	0
1	2	6	6	5	2	2	1	5	5	3	4	5	0	6	2	2	3	3	4
2	4	3	3	4	6	0	0	3	3	4	5	1	4	5	0	0	1	1	6
3	6	1	1	2	0	5	5	1	1	2	2	0	6	2	4	6	5	2	3
4	0	4	0	6	5	3	3	0	0	1	1	4	3	1	6	3	2	0	1
5	3	0	5	3	4	6	6	6	6	0	0	2	5	0	1	4	0	5	2
6	5	5	4	0	3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

0 4 4 0 0 3 0 0
1 0 0 1 1 0 0 1
3 5 5 3 2 4 1 2
4 2 2 4 ⇒ 3 1 0 3 ⇒ 0 0 0 0
5 6 6 5 4 6 3 4 ⇒ 3 0 0 3
6 3 3 6 5 4 2 5 ⇒ 4 6 4 4 ⇒ 1 4 2 2
0 3 0 0 5 4 2 5 ⇒ 5 1 1 6 ⇒ 0 0 0 0
1 0 0 1 6 6 2 6 6 3 2 6

$$\begin{array}{cccc}
 0 & 0 & 0 & 0 \\
 2 & 5 & 2 & 2 \\
 3 & 2 & 2 & 4 \\
 5 & 6 & 2 & 5 \\
 6 & 4 & 2 & 6
 \end{array}
 \Rightarrow
 \begin{array}{cccc}
 0 & 1 & 1 & 0 \\
 1 & 0 & 0 & 1 \\
 2 & 2 & 1 & 5 \\
 5 & 5 & 1 & 5
 \end{array}
 \Rightarrow
 \begin{array}{cccc}
 0 & 0 & 0 & 0 \\
 1 & 3 & 1 & 1 \\
 3 & 1 & 1 & 3 \\
 5 & 5 & 1 & 5
 \end{array}
 \Rightarrow
 \begin{array}{cccc}
 0 & 0 & 0 & 0 \\
 1 & 1 & 1 & 3 \\
 3 & 4 & 2 & 3 \\
 6 & 2 & 1 & 6
 \end{array}
 \Rightarrow
 \begin{array}{cccc}
 0 & 3 & 2 & 0 \\
 1 & 0 & 0 & 1 \\
 3 & 4 & 2 & 3 \\
 6 & 2 & 1 & 6
 \end{array}$$



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") then 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 then 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*