Genome Assembly

Low-level approach with R

officialprofile

2021-08-15

Contents

Chapter 1

Introduction

This mini textbook describes (or perhaps one should say "will describe") selected algorithms that play a vital role in the *de novo* genome assembly or in some related areas.

The premise of this book is to construct these algorithms from the very bottom along with explaining their gists. Naturally, the applications are included as well.

By default the code is written in R, but python and shell can appear at some point as well.

1.1 Prerequisites

It is assumed that the reader

1. has a basic understanding of genetics,
2. has some experience with programming in R,
3. had some contact with higher mathematics, e.g. probability, statistics, graph theory.

Reader should also be familiar with the pipe %>% syntax that is derived from dplyr package.

Throughout the book the following libraries are being used and it is assumed that the reader has them loaded.

```
library(stringr)
library(dplyr)
```

Chapter 2

Burrows-Wheeler transform

2.1 Introduction

[Description]

2.2 Burrows-Wheeler matrix

Consider the following sequence:

```
sequence <- 'GATTACA'
```

In order to create the Burrows-Wheeler matrix for the given string we at first add the dollar sign $ at the end of the sequence.

```
sequence  <- str_c(sequence, '$')
```

Afterwards we perform a series of shifts

```
sequences <- c(sequence)
n         <- nchar(sequence)

for (i in 1:(n-1)){
  sequence <- str_c(str_sub(sequence, 2, n),
                    str_sub(sequence, 1, 1))

  sequences <- c(sequences, sequence)
}

cat(sequences, sep = '\n')
#> GATTACA$
```

```
#> ATTACA$G
#> TTACA$GA
#> TACA$GAT
#> ACA$GATT
#> CA$GATTA
#> A$GATTAC
#> $GATTACA
```

Then we sort these sequences with the assumption that the dollar sign precedes lexicographically every symbol.

```
sequences <- sort(sequences)
cat(sequences, sep = '\n')
#> $GATTACA
#> A$GATTAC
#> ACA$GATT
#> ATTACA$G
#> CA$GATTA
#> GATTACA$
#> TACA$GAT
#> TTACA$GA
```

By putting every letter

```
bw.matrix           <- data.frame(matrix(, n, n))
colnames(bw.matrix) <- 1:n

for (i in 1:n){
  bw.matrix[i, ] <- strsplit(sequences[i], split = '')[[1]]
}

knitr::kable(bw.matrix)
```

1	2	3	4	5	6	7	8
$	G	A	T	T	A	C	A
A	$	G	A	T	T	A	C
A	C	A	$	G	A	T	T
A	T	T	A	C	A	$	G
C	A	$	G	A	T	T	A
G	A	T	T	A	C	A	$
T	A	C	A	$	G	A	T
T	T	A	C	A	$	G	A

Sequence at the very bottom of the Burrows-Wheeler matrix is called **Burrows-Wheeler transform**.

```
transform <- paste(bw.matrix[,n], collapse = '')
```

```
cat('The Burrows-Wheeler transform of',
    str_sub(sequence, 2, n), 'is', transform)
#> The Burrows-Wheeler transform of GATTACA is ACTGA$TA
```

2.3 Inverse transform

We are going to reconstruct the Burrows-Wheeler matrix and initial sequence itself. In order to do so one as first has to sort the transformed sequence.

```
first.sequence <- strsplit(transform, split = '')[[1]] %>% sort
paste(first.sequence, collapse = '')
#> [1] "$AAACGTT"
```

Note that this string is equal to the first column of the Burrrows-Wheeler transform. Also keep in mind that the characters from last and the first column are adjacent.

```
bw.inverse           <- data.frame(matrix(, n, 2))
colnames(bw.inverse) <- c(n, 1)

bw.inverse[, 1] <- strsplit(transform, split = '')[[1]]
bw.inverse[ ,2] <- first.sequence

knitr::kable(bw.inverse)
```

8	1
A	$
C	A
T	A
G	A
A	C
$	G
T	T
A	T

In other words, a this point we have a set of 2-mers.

```
kmers <- apply(bw.inverse, 1,
               function(x) paste(x, collapse = ''))
kmers
#> [1] "A$" "CA" "TA" "GA" "AC" "$G" "TT" "AT"
```

Once again, we strictly rely on the fact that Burrows-Wheeler matrix is sorted lexicographically. This allows us to reconstruct the remaining columns.

```
kmers <- sort(kmers)
kmers
#> [1] "$G" "A$" "AC" "AT" "CA" "GA" "TA" "TT"
```

These 2-mers (k-mers in general) represent first two columns of the Burrows-Wheeler matrix.

We can derive last characters of the 2-mers in the following way:

```
sapply(kmers, function(x) str_sub(x, 2, 2),
       simplify = TRUE, USE.NAMES = FALSE)
#> [1] "G" "$" "C" "T" "A" "A" "A" "T"
```

```
for (i in 2:(n-1)){
  kmers             <- apply(bw.inverse, 1,
                             function(x) paste(x, collapse = ''))
  kmers             <- sort(kmers)
  bw.inverse[, i+1] <- sapply(kmers, function(x) str_sub(x, i, i),
                              simplify = TRUE, USE.NAMES = FALSE)
  colnames(bw.inverse)[i+1] = i
}
knitr::kable(bw.inverse)
```

8	1	2	3	4	5	6	7
A	$	G	A	T	T	A	C
C	A	$	G	A	T	T	A
T	A	C	A	$	G	A	T
G	A	T	T	A	C	A	$
A	C	A	$	G	A	T	T
$	G	A	T	T	A	C	A
T	T	A	C	A	$	G	A
A	T	T	A	C	A	$	G

Finally we move first column to the very end

```
bw.inverse[,n+1] <- bw.inverse[, 1]
bw.inverse       <- bw.inverse[,2:(n+1)]
colnames(bw.inverse)[n] = n

knitr::kable(bw.inverse)
```

1	2	3	4	5	6	7	8
$	G	A	T	T	A	C	A
A	$	G	A	T	T	A	C
A	C	A	$	G	A	T	T
A	T	T	A	C	A	$	G
C	A	$	G	A	T	T	A
G	A	T	T	A	C	A	$
T	A	C	A	$	G	A	T
T	T	A	C	A	$	G	A

One can also verify that bw.matrix and bw.inverse are in fact the same.

```
knitr::kable(bw.inverse == bw.matrix)
```

1	2	3	4	5	6	7	8
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE

Additionally we can encapsulate Burrows-Wheeler transform in a function

```
BWT <- function(sequence){
  sequence  <- str_c(sequence, '$')
  sequences <- c(sequence)
  n         <- nchar(sequence)

  for (i in 1:(n-1)){
    sequence <- str_c(str_sub(sequence, 2, n),
                      str_sub(sequence, 1, 1))
    sequences <- c(sequences, sequence)
  }
  sequences <- sort(sequences)

  bw.matrix           <- data.frame(matrix(, n, n))
  colnames(bw.matrix) <- 1:n

  for (i in 1:n){
    bw.matrix[i, ] <- strsplit(sequences[i], split = '')[[1]]
  }
  return(paste(bw.matrix[,n], collapse = ''))
}
```

```
BWT('GATTACA')
#> [1] "ACTGA$TA"
```

One can verify that this output is equal to result we obteined earlier.

Out of pure curiosity lets check the Burrows-Wheeler transform for a longer sequence.

```
BWT('ATGCTCGTGCCATCATATAGCGCGCGCGCGATCTCTACGCGCG')
#> [1] "GTTTCCG$TCGGGGGAGGGTTGTCCTCCCCCCATCCAAACCAGA"
```

Please note that the input string has no identical characters at adjacent positions whereas in the transformed sequence such situation appears quite often. These substrings of identical characters will allow us represent the sequence in a more condensed manner.

2.4 Applications