in-biosx000- Algorithms module
Sequence mapping

Autumn 2020

Gigsaw from University of Dundee

- Four people in each group

- Set up the reference sequence

- Notice that:
  – reads are reverse complemented on the back
  – some reads contain sequencing errors

- Map the reads to the reference as efficiently as possible:
  – how do you organise the work between team members?
  – which "side/strand" of read to use?

- In 10 mins, I want to know what you have found out

Gigsaw created by David Martin, University of Dundee, Scotland

Strategies and observations

- Strategies
  - Each team member gets their own pile of reads (parallelisation)
  - Good idea to use reverse complement
  - Try matching the first few bases first and then check rest of read

- Observations
  - Some areas with poor coverage (middle and ends)
  - Sequencing errors in first few bases can make matching more difficult

Results

Position   REF   Reads

8   T   1C/2T   <<< possibly variant

14   C   6C / 1G

24   C   2A / 8C   <<< possibly variant

33   C   7G / 3A   <<< variant

65   A   5A / 1C

74   A   11A / 1G

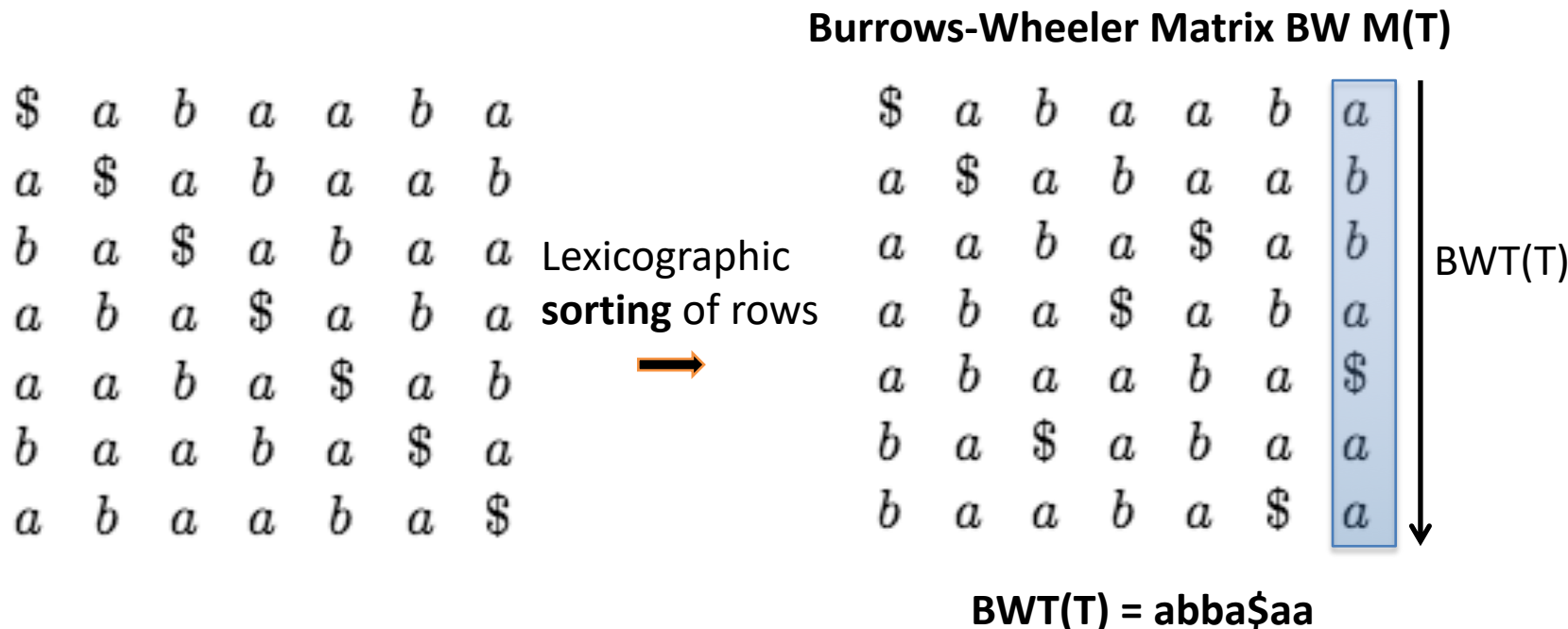77   T   10T / 1G

90   C   10C / 1A

Could this be a diploid organism?

FASTQ → Mapping (BWA) → SAM → HouseKeeping & refinement → BAM → Variant calling (gatk) → VCF file

MAPPING WITH BWA

Why mapping?

- The biggest difference with Sanger
  - we did not design and use primers for sequence amplification
  - we sonicated the DNA
  - >> we do not know where the reads "originate" from

- For each read
  - we need to determine its likely origin
  - how likely it is that we have correctly identified its origin

- The Burrows-Wheeler Transform (BWT) is a way of permuting the characters of a string T into another string BWT (T)

- For example for T = **abaaba$**

- $ defined to be a character that does **not** appear elsewhere in T and which is **lexicographically prior** to all other characters

- The rotations of T

Burrows-Wheeler Matrix BW M(T)

$ a b a a b a
a $ a b a a b
b a $ a b a a   Lexicographic
a b a $ a b a   **sorting** of rows
a a b a $ a b
b a a b a $ a
a b a a b a $

➡

$ a b a a b a
a $ a b a a b
a a b a $ a b
a b a $ a b a
a b a a b a $
b a $ a b a a
b a a b a $ a

BWT(T)

BWT(T) = abba$aa

Relationship to suffix array

- Suffix array = sorted array of all suffixes of a string

- T = **abaaba$**

All suffixes of T

```
0   a b a a b a $
1   b a a b a $
2   a a b a $
3   a b a $
4   b a $
5   a $
6   $
```

Sorted suffixes

```
6 $
5 a $
2 a a b a $
3 a b a $
0 a b a a b a $
4 b a $
1 b a a b a $
```

Matching order

```
BWM:        SA: Suffixes given by SA:
$abaaba      6  $
a$abaab      5  a$
aaba$ab      2  aaba$
aba$aba      3  aba$
abaaba$      0  abaaba$
ba$abaa      4  ba$
baaba$a      1  baaba$
```

FM index – key to searching a string

- We re-rank them:
  - Before we ranked according how many times occurred previously in T ie T ranking
  - Now we re-rank according to how many times previouly seen in BWT(T) ie B ranking
- <u>Ferragina and Manzini finding: BWT can be used as a space-efficient index of T using the LF mapping property</u>
- Assume we are searching for occurrences of P = aba
- The BWM is sorted, so any rows having P as a **prefix** will be **consecutive**

First find the rows beginning with the shortest **suffix** of P i.e. "a"
>> rows 1,2,3 and 4

1

2 Second find all rows begin with the **next** longest **suffix** of P i.e. "ba"
 >> these are rows 1 and 2 (**rows are rotations of T**)

3

F						L	$rank$
$\$$	a	b	a	a	b	a	0
a	$\$$	a	b	a	a	b	0
a	a	b	a	$\$$	a	b	1
a	b	a	$\$$	a	b	a	1
a	b	a	a	b	a	$\$$	0
b	a	$\$$	a	b	a	a	2
b	a	a	b	a	$\$$	a	3

The **LF mapping** and rank array tell us which rows have "ba" as a prefix: rows beginning with b0 and b1

The LF mapping states: the ith occurrence of a character c in the last column has the same rank as the ith occurrence of c in the first column

Searching T

The LF mapping and rank array tell us which rows have "ba" as a prefix: row begining with b0 and b1

Occurrences of "ba" are preceded by a2 and a3 (bcse rows are rotations of T)

Backward matching: apply LF mapping repeatedly to find range of rows **prefixed** by successively longer proper **suffixes of P until range is empty or run out of suffixes**

With only 3 backward matching (LF mapping and rotations) operations we have found the rows prefixed with occurrences of "aba"

So far we have discussed how to use the FM Index to determine <u>whether</u> and <u>how many times</u> a substring P occurs within T, but we have not discussed how to find <u>where</u> P occurrs, i.e. the substring's offset into T.

If our index included the suffix array $SA(T)$, we could simply look this up in $SA(T)$. For example, here was the range we ended up with after searching for $P = aba$ within $BWT(abaaba\$)$:

SA and BWM have a corresponding order

F							L	$SA(T)$	BWM:	SA: Suffixes given by SA:
$\$$	a	b	a	a	b		a	6	$\underline{\$}$abaaba	6 $\$$
a	$\$$	a	b	a	a		b	5	a$\underline{\$}$abaab	5 a$\$$
a	a	b	a	$\$$	a		b	2	aaba$\underline{\$}$ab	2 aaba$\$$
a	b	a	$\$$	a	b		a	3	aba$\underline{\$}$aba	3 aba$\$$
a	b	a	a	b	a		$\$$	0	abaaba$\underline{\$}$	0 abaaba$\$$
b	a	$\$$	a	b	a		a	4	ba$\underline{\$}$abaa	4 ba$\$$
b	a	a	b	a	$\$$		a	1	baaba$\underline{\$}$a	1 baaba$\$$

$SA(T)$ tells us these matches occurred at T offsets 0 and 3.

T=abaaba
012345

Maybe that did not seem impressive.....

F						L	rank
$	a	b	a	a	b	a	0
a	$	a	b	a	a	b	0
a	a	b	a	$	a	b	1
a	b	a	$	a	b	a	1
a	b	a	a	b	a	$	0
b	a	$	a	b	a	a	2
b	a	a	b	a	$	a	3

I could have found the offsets of "aba" in T.
Yes, BUT:
- **either** you then scan T with "aba" >> exhausitve search
- **or** you have to store the whole BWM matrix

When the string you are searching is 3 billion bases long that means:
- **either** you check for a match at every 3 bn positions
- **or** you need to store a matrix that is 3bn x 3bn

If you use the BWT and suffix array:
- You only need to store the first and last columns of the BWM (not the blue section)
- You need to do **X** backward matching operations to locate a string of **length X**

There are many optimisations to:
- Decrease space requirements
- Decrease time requirements
- Allow for mismatches

Application to read mapping

- You need to build an index of the genome you wish to search
- Reads are 100 bp long
- But, the longer the string searched for the longer the search time **(longer backtracking)**, thus an advantage to search with a shorter string.
- There are 4 different nucleotides and 3.0E09 bases of genome
  - 4**16 = 4.0E09
  - 4**17 = 17.0E09
- Assuming complex sequence, 16 to 17 bp should be enough for a unique match
- BUT
  - variation in sample relative to reference
  - base call errors
  - **THUS** 17 bp search string is not enough
- BWA searches with a "seed" of 32 and allows 2 differences in the seed
- **Reads with seed matches are aligned using Smith-Waterman**

Mapping exercise with BWA

- Log into the virtual machine

- Open the exercise file in a text editor:
  algorithms_mappingExercise.bash

- Open a terminal and carry out the exercise

APPENDIX

Skipped

We re-rank them:
* Before we ranked according how many times occurred previously in T ie T ranking
* Now we re-rank according to how many times previouly seen in BWT(T) ie B ranking

F	L	rank
$	a	0
a	b	0
a	b	1
a	a	1
a	$	0
b	a	2
b	a	3

Start with $

F	L	rank
$	a	0
a	b	0
a	b	1
a	a	1
a	$	0
b	a	2
b	a	3

Rows are rotations of T so last col of the first row contains the character to left of $ in T: a

⬇

Rank array tells us this is a with rank 0
⬇

How do we get the character to the left of a_0?
⬇

We can do this if we know the row that begins with a_0
⬇

The LF tells us which row begins with a_0 : the first row containing an "a"

The LF mapping states: the ith occurrence of a character c in the last column has the same rank as the ith occurrence of c in the first column

⟷

Assuming first row is 0, we visit rows: 0, 1, 5, 3, 2, 6, 4

⟶

$$a_3 b_1 a_1 a_2 b_0 a_0 \$$$

Reversible means useful for compression