Short (and Long) Read Alignment

Simon Rasmussen
Next Generation Sequencing analysis
DTU Bioinformatics
Generalized NGS analysis

Question

Raw reads

Pre-processing

Assembly: Alignment / de novo

Application specific: Variant calling, count matrix, ...

Compare samples / methods

Answer?
Generalized NGS analysis

Question
Raw reads
Pre-processing
Assembly: Alignment / de novo
Application specific:
Variant calling, count matrix, ...
Compare samples / methods
Answer?
• Assemble your reads by aligning them to a closely related reference genome

• High sequence similarity between individuals makes this possible
Sounds easy?

• Some pitfalls:
  • Divergence between sample and reference genome
  • Repeats in the genome
  • Recombination and re-arrangements
  • Poor reference genome quality
  • Read errors
  • Regions not in the ref. genome
Alignment approaches

- Short reads: global read alignment (or glocal for little longer reads)
- Long reads: local alignments (more likely to have gap/indels)
- Exact string match
- Creating hash tables (maybe you know these from Perl/Python/Java/C, ...)
- BWT + suffix arrays
Simplest solution

- Exact string matches:
  
  Reference: ACGTGCGrACGCTGAACGTGACG
  Read: GTG

- We need to allow mismatches/indels (Smith-Waterman, Needleman-Wunsch)

- One of the worlds fastest computer (K computer - RIKEN)

- 20 mill reads 100 nt reads vs. human genome
  ~ 1 month

- We search each read vs. the entire reference

Complexity: $O(n \times m)$
Simplest solution

- Exact string matches:

  Reference: ACGTGCAGGACGCTGAACGTGACG
  Read: GTG GTG GTG

- We need to allow mismatches/indels (Smith-Waterman, Needleman-Wunsch)

- One of the worlds fastest computer (K computer - RIKEN)

- 20 mill reads 100 nt reads vs. human genome
  ~ 1 month

- We search each read vs. the entire reference

Complexity: $O(n \times m)$
Simplest solution

- Exact string matches:

  Reference: ACGTGC GGACGCTGAACGTGACG
  Read: GTG    GTG    G–TG    GTG

- We need to allow mismatches/indels (Smith-Waterman, Needleman-Wunsch)

- One of the world's fastest computer (K computer - RIKEN)

- 20 mill reads 100 nt reads vs. human genome ~ 1 month

- We search each read vs. the entire reference

Complexity: $O(n^2)$
How about BLAST?

- Everybody uses BLAST
- Everybody will believe your BLAST hits (pun intended)

- However BLAST
  - finds local alignments - not always what we want for short reads
  - and other stuff (alignment scores, output format, speed)
  - Not practical for short reads!
How about BLAST?

- Everybody uses BLAST
- Everybody will believe your BLAST hits (pun intended)

What we can learn: Reducing the search space

- However BLAST
  - finds local alignments - not always what we want for short reads
  - and other stuff (alignment scores, output format, speed)
  - Not practical for short reads!
Smart solution

1. Use algorithm to quickly find possible matches

Drastically reduced search space

2. Allow us to perform slow/precise alignment for possible matches (Smith-Waterman)
Smart solution

1. Use algorithm to quickly find *possible* matches

Drastically reduced search space

2. Allow us to perform slow/precise alignment for possible matches (Smith-Waterman)

$3.2\text{Gb} \times \text{possible matches} \rightarrow 1 \text{ best match}$
## Hash based algorithms

**Lookups in hashes are fast!**

1. Index the reference using *k*-mers.
2. Search reads vs. hash *k*-mers
3. Perform alignment of entire read around seed
4. Report best alignment

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTGCGTGTGA</td>
<td>Chr1_pos1234; Chr2_pos567</td>
</tr>
<tr>
<td>ACTGCGTGTGC</td>
<td>Chr7_posX</td>
</tr>
<tr>
<td>ACTGCGTGTGT</td>
<td>Chr7_posZ; ...</td>
</tr>
</tbody>
</table>

...
Hash based algorithms

Lookups in hashes are *fast!*

1. Index the reference using *k*-mers.
2. Search reads vs. hash *k*-mers
3. Perform alignment of entire read around seed
4. Report best alignment

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTGCGTGTGA</td>
<td>Chr1_pos1234; Chr2_pos567</td>
</tr>
<tr>
<td>ACTGCGTGTGC</td>
<td>Chr7_posX</td>
</tr>
<tr>
<td>ACTGCGTGTGT</td>
<td>Chr7_posZ; ...</td>
</tr>
</tbody>
</table>

Also known as *Seed and extend*
Spaced seeds

• Key/k-mer is called a seed
• BLAST uses $k=11$ and all must be matches

• Smarter: Spaced seeds (only care about “1” in seed)
  • Higher sensitivity
  • One can use several seeds

$L = 11$, 11 matches

$L = 18$, 11 matches
Multiple seeds & drawbacks

• One could require multiple short seeds
  • Instead of extending around each seed, extend around positions with several seed matches

• Drawbacks of hash-based approaches:
  • Lots(!) of RAM to keep index in memory (hg ~48Gb!)
  • Poor hashing may lead to slow alignment
Burrows-Wheeler Transform

- Hash based aligners require lots of memory and are only reasonable fast
- Can we make it better/faster?
- Burrows Wheeler Transformation (BWT), Suffix Arrays and FM-index
- BWT originally created for compression (implemented in bzip2)
The concepts

- Burrows-Wheeler Transform (BWT)
  - A reversible transformation of the genome
- Suffix Array is “array of integers giving the starting positions of suffixes of a string in lexicographical order”
- Ferragina and Manzini (FM) index
  - Allows us to recreate parts of the Suffix Array on the fly
- First create BWT of the genome (=index)
BWT: Create index

Genome

T = AGGAGC$

Marks end-of-string, lexicographically smallest
BWT: Create index

\[ T = \text{AGGAGC}$ \]
BWT: Create index

1. Create all possible transformations of the string
   (move first base to end)

\[ T = \text{AGGAGC}\$ \]

\[ \text{AGGAGC}\$ \]
BWT: Create index

1. Create all possible transformations of the string
   (move first base to end)

\[ T = \text{AGGAGC}\$ \]

- AGGAGC\$
- GGAGC\$A
BWT: Create index

1. Create all possible transformations of the string (move first base to end)

\[ T = \text{AGGAGC} $ \]

\[ \text{AGGAGC}$, \text{GGAGC}$A, \text{GAGC}$A\text{G} \]
BWT: Create index

1. Create all possible transformations of the string (move first base to end)

\[ T = \text{AGGAGC}\$ \]

- AGGAGC$
- GGAGC$A
- GAGC$AG
- AGC$AGG
- GC$AGGA
- C$AGGAG
- $AGGAGC
BWT: Create index

1. Create all possible transformations of the string (move first base to end)

\[ T = \text{AGGAGC}$ \]

\[
\begin{align*}
0 & \quad \text{AGGAGC}$ \\
1 & \quad \text{GGAGC}$A \\
2 & \quad \text{GAGC}$AG \\
3 & \quad \text{AGC}$AGG \\
4 & \quad \text{GC}$AGGA \\
5 & \quad \text{C}$AGGAG \\
6 & \quad \text{$AGGAGC} \\
\end{align*}
\]
### BWT: Create Index

**T = AGGAGC$**

2. Sort the strings lexicographically to create BWT matrix and Suffix Array

<table>
<thead>
<tr>
<th>0</th>
<th>AGGAGC$</th>
<th>6</th>
<th>$</th>
<th>A</th>
<th>G</th>
<th>G</th>
<th>A</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GGAGC$A</td>
<td>3</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>$</td>
<td>A</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>2</td>
<td>GAGC$AG</td>
<td>0</td>
<td>A</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>$</td>
</tr>
<tr>
<td>3</td>
<td>AGC$AGG</td>
<td>5</td>
<td>C</td>
<td>$</td>
<td>A</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>4</td>
<td>GC$AGGA</td>
<td>2</td>
<td>G</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>$</td>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>5</td>
<td>C$AGGAG</td>
<td>4</td>
<td>G</td>
<td>C</td>
<td>$</td>
<td>A</td>
<td>G</td>
<td>G</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>$AGGAGC</td>
<td>1</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>$</td>
<td>A</td>
</tr>
</tbody>
</table>
BWT: Create index

$T = \text{AGGAGC}$

2. Sort the strings lexicographically to create BWT matrix and Suffix Array

<table>
<thead>
<tr>
<th>SA</th>
<th>BWT matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AGGAGC$</td>
<td>$A$ $G$ $G$ $A$ $G$ $C$</td>
</tr>
<tr>
<td>AGAGC$A</td>
<td>$A$ $G$ $C$ $A$ $G$ $G$</td>
</tr>
<tr>
<td>GAGC$AG</td>
<td>$A$ $G$ $G$ $A$ $G$ $C$ $</td>
</tr>
<tr>
<td>AGC$AGG</td>
<td>$C$ $A$ $G$ $G$ $A$ $G$</td>
</tr>
<tr>
<td>GC$AGGA</td>
<td>$G$ $A$ $G$ $C$ $A$ $G$</td>
</tr>
<tr>
<td>CG$AGGAG</td>
<td>$G$ $C$ $A$ $G$ $G$ $A$</td>
</tr>
<tr>
<td>$AGGAGC$</td>
<td>$G$ $G$ $A$ $G$ $C$ $A$ $</td>
</tr>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>6</td>
<td>$</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>G</td>
</tr>
<tr>
<td>4</td>
<td>G</td>
</tr>
<tr>
<td>1</td>
<td>G</td>
</tr>
</tbody>
</table>

SA  BWT matrix
BWT

This one we need later

<table>
<thead>
<tr>
<th></th>
<th>$</th>
<th>A</th>
<th>G</th>
<th>G</th>
<th>A</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>$</td>
<td>A</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>0</td>
<td>A</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>$</td>
</tr>
<tr>
<td>5</td>
<td>C</td>
<td>$</td>
<td>A</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>2</td>
<td>G</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>$</td>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>4</td>
<td>G</td>
<td>C</td>
<td>$</td>
<td>A</td>
<td>G</td>
<td>G</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>G</td>
<td>G</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>$</td>
<td>A</td>
</tr>
</tbody>
</table>

SA    BWT matrix
BWT

This one we need later

SA  BWT matrix
This one we need later

<table>
<thead>
<tr>
<th>F</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ A G G A G C</td>
<td>$ A G G A G C</td>
</tr>
<tr>
<td>A G C $ A G G</td>
<td>A G C $ A G G</td>
</tr>
<tr>
<td>A G G A G C $</td>
<td>A G G A G C $</td>
</tr>
<tr>
<td>C $ A G G A G</td>
<td>C $ A G G A G</td>
</tr>
<tr>
<td>G A G C $ A G</td>
<td>G A G C $ A G</td>
</tr>
<tr>
<td>G C $ A G G A</td>
<td>G C $ A G G A</td>
</tr>
<tr>
<td>G G A G C $ A</td>
<td>G G A G C $ A</td>
</tr>
</tbody>
</table>

$F =\text{First column}$

$L =\text{Last column}$
BWT: $T$-rank

$T = \text{AGGAGC}\$

*T-ranking:* # of times the base occurred previously in $T$

$$
\begin{array}{ccccccc}
\text{A}_0 & \text{G}_0 & \text{G}_1 & \text{A}_1 & \text{G}_2 & \text{C}_0 & \$
\end{array}
$$
**BWT: T-rank**

\[ T = \text{AGGAGC}$ \]

*T-ranking*: # of times the base occurred previously in \( T \)

\[
\begin{align*}
\text{A}_0 & \quad \text{G}_0 & \quad \text{G}_1 & \quad \text{A}_1 & \quad \text{G}_2 & \quad \text{C}_0 & \quad \$ \\
\text{G}_1 & \quad \text{A}_1 & \quad \text{G}_2 & \quad \text{C}_0 & \quad \$ & \quad \text{A}_0 & \quad \text{G}_0 & \quad \text{G}_1 \\
\text{G}_0 & \quad \text{C}_0 & \quad \$ & \quad \text{A}_0 & \quad \text{G}_0 & \quad \text{G}_1 & \quad \text{A}_1 & \quad \text{G}_2 \\
\text{G}_2 & \quad \text{C}_0 & \quad \$ & \quad \text{A}_0 & \quad \text{G}_0 & \quad \text{G}_1 & \quad \text{A}_1 & \quad \text{G}_0 \\
\text{C}_0 & \quad \$ & \quad \text{A}_0 & \quad \text{G}_0 & \quad \text{G}_1 & \quad \text{A}_1 & \quad \text{G}_2 & \quad \text{C}_0 & \quad \$ \\
\end{align*}
\]
**BWT: T-rank**

\[ T = \text{AGGAGC}\$ \]

**T-ranking:** # of times the base occurred previously in \( T \)

\[ \begin{array}{cccccc}
A_0 & G_0 & G_1 & A_1 & G_2 & C_0 \$
\end{array} \]

Notice that individual base-rank is the same in \( F \) and \( L \)

\[ \begin{array}{cccccc}
$ & A_0 & G_0 & G_1 & A_1 & G_2 \\
A_1 & G_2 & C_0 & $ & A_0 & G_0 \\
A_0 & G_0 & G_1 & A_1 & G_2 & C_0 \\
C_0 & $ & A_0 & G_0 & G_1 & A_1 \\
G_1 & A_1 & G_2 & C_0 & $ & A_0 \\
G_2 & C_0 & $ & A_0 & G_0 & G_1 \\
G_0 & G_1 & A_1 & G_2 & C_0 & $ \\
\end{array} \]

Rank will always be the same in \( F \) and \( L \)
BWT: \textit{B-rank}

\textit{B-ranking}: Ranked based on occurrence in $F/L$

\begin{align*}
F & \quad \text{B-rank} & \quad T \quad \text{T-rank} \\
\$ & A_1 & G_2 & G_0 & A_0 & G_1 & C_0 & G_0 & A_0 & G_1 & C_0 & \$ \\
A_0 & G_1 & C_0 & \$ & A_1 & G_2 & G_0 & A_0 & \$ & G_1 & G_2 & A_0 & \$ \\
A_1 & G_2 & G_0 & A_0 & G_1 & C_0 & \$ & G_1 & G_2 & A_0 & \$ & A_1 & \$ \\
C_0 & \$ & A_1 & G_2 & G_0 & A_0 & \$ & G_1 & G_2 & A_0 & \$ & A_1 & \$ \\
G_0 & A_0 & G_1 & C_0 & \$ & A_1 & \$ & G_1 & G_2 & A_0 & \$ & A_1 & \$ \\
G_1 & C_0 & \$ & A_1 & G_2 & G_0 & \$ & G_1 & G_2 & A_0 & \$ & A_1 & \$ \\
G_2 & G_0 & A_0 & G_1 & C_0 & \$ & A_1 & \$ & G_1 & G_2 & A_0 & \$ & A_1 & \$
\end{align*}

$T = \text{AGGAGC}\$

$T$-rank: $A_0 \ G_0 \ G_1 \ A_1 \ G_2 \ C_0 \ \$
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

\[
\begin{array}{cccccc}
F & \$ & A_1 & G_2 & G_0 & A_0 & G_1 \\
A_0 & G_1 & C_0 & \$ & A_1 & G_2 \\
A_1 & G_2 & G_0 & A_0 & G_1 & C_0 \\
C_0 & \$ & A_1 & G_2 & G_0 & A_0 \\
G_0 & A_0 & G_1 & C_0 & \$ & A_1 \\
G_1 & C_0 & \$ & A_1 & G_2 & G_0 \\
G_2 & G_0 & A_0 & G_1 & C_0 & \$
\end{array}
\]

\[
\begin{array}{cccc}
L & C_0 \\
G_0 & $ \\
G_1 & G_2 \\
A_0 & A_1 \\
A_1 & \\
\end{array}
\]
**BWT is reversible**

LF-mapping: LF can be used to recreate the original genome

<table>
<thead>
<tr>
<th>$</th>
<th>A_0</th>
<th>A_1</th>
<th>C_0</th>
<th>G_0</th>
<th>G_1</th>
<th>G_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td></td>
</tr>
<tr>
<td>A_0</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td></td>
</tr>
<tr>
<td>A_1</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td></td>
</tr>
<tr>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td></td>
</tr>
<tr>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td></td>
</tr>
<tr>
<td>G_1</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td></td>
</tr>
<tr>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>C_0</td>
</tr>
<tr>
<td>A_0</td>
<td>G_0</td>
</tr>
<tr>
<td>A_1</td>
<td>$</td>
</tr>
<tr>
<td>C_0</td>
<td>G_1</td>
</tr>
<tr>
<td>G_0</td>
<td>G_2</td>
</tr>
<tr>
<td>G_1</td>
<td>A_0</td>
</tr>
<tr>
<td>G_2</td>
<td>A_1</td>
</tr>
</tbody>
</table>
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

\[ F \]

\[
\begin{array}{cccc}
\$ & A_0 & A_1 & C_0 \\
A_0 & G_1 & C_0 & $ \\
A_1 & G_2 & G_0 & A_0 \\
C_0 & $ & A_1 & G_2 \\
G_0 & A_0 & G_1 & C_0 \\
G_1 & C_0 & $ & A_1 \\
G_2 & $ & A_1 & G_2 \\
G_0 & A_0 & G_1 & C_0 \\
\end{array}
\]

\[ L \]

\[
\begin{array}{cccc}
C_0 & G_0 & $ & C_0 \\
G_0 & $ & G_1 & G_0 \\
G_1 & G_2 & A_0 & A_1 \\
\end{array}
\]
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

\[
F \quad \begin{array}{cccccc}
$ & A_1 & G_2 & G_0 & A_0 & G_1 \\
A_0 & G_1 & C_0 & $ & A_1 & G_2 \\
A_1 & G_2 & G_0 & A_0 & G_1 & C_0 \\
C_0 & $ & A_1 & G_2 & G_0 & A_0 \\
G_0 & A_0 & G_1 & C_0 & $ & A_1 \\
G_1 & C_0 & $ & A_1 & G_2 & G_0 \\
G_2 & G_0 & A_0 & G_1 & C_0 & $ \\
\end{array}
\quad L \quad \begin{array}{cccccc}
C_0 \\
G_0 \\
$ \\
G_1 \\
G_2 \\
A_0 \\
A_1 \\
\end{array}
\]

$C_0 G_1$
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

\[ \text{C}_0 \text{G}_1 \]
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

C_0 G_1 A_0
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

\[ C_0 G_1 A_0 \]
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

$A_1G_2G_0A_0G_1$
$G_1C_0A_1G_2$
$G_2G_0A_0G_1C_0$
$C_0A_1G_2G_0A_0$
$A_0G_1C_0A_1G_2$
$G_0A_0G_1C_0A_1$
$G_0A_0G_1C_0A_1$

$C_0G_1A_0G_0$

36626 - Next Generation Sequencing Analysis
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

\[ C_0 G_1 A_0 G_0 \]
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

\[ \text{C}_0 \text{G}_1 \text{A}_0 \text{G}_0 \text{G}_2 \]
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

\[ C_0 G_1 A_0 G_0 G_2 \]
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

$C_0G_1A_0G_0G_2A_1$
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

C₀G₁A₀G₀G₂A₁
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

\[ C_0 G_1 A_0 G_0 G_2 A_1 \]$
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

\[ C_0 G_1 A_0 G_0 G_2 A_1 \]

Reversed:
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

$C_0 G_1 A_0 G_0 G_2 A_1 \$, Reversed:

$A_1 G_2 G_0 A_0 G_1 C_0$

$T = AGGAGGC$
BWT is reversible

LF-mapping: LF can be used to recreate the original genome

\[ \text{F} = \begin{array}{c}
\$ \\
A_0 \\
A_1 \\
C_0 \\
G_0 \\
G_1 \\
G_2 \\
\end{array} \quad \Rightarrow \quad \begin{array}{c}
A_1 \\
C_2 \\
G_0 \\
A_0 \\
C_0 \\
G_1 \\
G_2 \\
\end{array} \]

\[ \text{L} = \begin{array}{c}
C_0 \\
G_0 \\
A_1 \\
\$ \\
G_1 \\
G_2 \\
A_0 \\
A_1 \\
\end{array} \]

Reversed:

\[ \text{T} = \text{AGGAGC}\$ \]

\[ \text{F} \text{ can be computed } = 2x A, 1x C, 3x G \]

We therefore only need to store \( L \)
BWT: Lookups

We can look up where a read matches in our genome

Read = “AGG”

Start from last base:
A G G

F
$
A_0
A_1
C_0
G_0
G_1
G_2

L
C_0
G_0
$
G_0
$
G_1
G_2
A_0
A_1

$
BWT: Lookups

We can look up where a read matches in our genome

<table>
<thead>
<tr>
<th>F</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>C_0</td>
</tr>
<tr>
<td>A_0</td>
<td>G_0</td>
</tr>
<tr>
<td>A_1</td>
<td>$</td>
</tr>
<tr>
<td>C_0</td>
<td>G_0</td>
</tr>
<tr>
<td>G_0</td>
<td>$</td>
</tr>
<tr>
<td>G_1</td>
<td>G_1</td>
</tr>
<tr>
<td>G_2</td>
<td>A_0</td>
</tr>
</tbody>
</table>

Read = “AGG”

Start from last base:

A G G

Find all rows that starts with “G”
BWT: Lookups

We can look up where a read matches in our genome

\[
\begin{array}{ccccccc}
F & A_1 & G_2 & G_0 & A_0 & G_1 \\
\$ & A_1 & G_2 & G_0 & A_0 & G_1 \\
A_0 & G_1 & C_0 & \$ & A_1 & G_2 \\
A_1 & G_2 & G_0 & A_0 & G_1 & C_0 \\
C_0 & \$ & A_1 & G_2 & G_0 & A_0 \\
G_0 & A_0 & G_1 & C_0 & \$ & A_1 \\
G_1 & C_0 & \$ & A_1 & G_2 & G_0 \\
G_2 & G_0 & A_0 & G_1 & C_0 & \$ \\
\end{array}
\]

\[
L & C_0 \\
G_0 \\
G_0 \\
G_0 \\
G_1 \\
G_2 \\
G_1 \\
A_1 \\
\]

Read = “AGG”

Start from last base:

A G G

Find all rows that starts with “G”

Simple because F is sorted
## BWT: Lookups

We can look up where a read matches in our genome.

### Table

<table>
<thead>
<tr>
<th>$</th>
<th>A_1</th>
<th>G_2</th>
<th>G_0</th>
<th>A_0</th>
<th>G_1</th>
<th>C_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
</tr>
<tr>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
</tr>
<tr>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
</tr>
<tr>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
</tr>
<tr>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
</tr>
<tr>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
</tr>
</tbody>
</table>

Read = “AGG”

Start from last base:

A  G  G

In this range, match second last base in $L$
We can look up where a read matches in our genome

<table>
<thead>
<tr>
<th></th>
<th>$</th>
<th>A_1</th>
<th>G_2</th>
<th>G_0</th>
<th>A_0</th>
<th>G_1</th>
<th>C_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td></td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C_0</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>G_0</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>G_1</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
</tr>
</tbody>
</table>

Read = “AGG”

Start from last base:

A   G   G

In this range, match second last base in $L$
BWT: Lookups

We can look up where a read matches in our genome

Read = “AGG”

Start from last base:
A G G

Use LF mapping to get to F
We can look up where a read matches in our genome

Read = “AGG”

Start from last base:
A  G  G

Use LF mapping to get to F
We can look up where a read matches in our genome.

Read = “AGG”

Start from last base:

A  G  G

Match next base (A) in $L$

$F$

<table>
<thead>
<tr>
<th></th>
<th>A₀</th>
<th>G₂</th>
<th>G₀</th>
<th>A₀</th>
<th>G₁</th>
<th>C₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>A₁</td>
<td>G₂</td>
<td>G₀</td>
<td>A₀</td>
<td>G₁</td>
<td>C₀</td>
</tr>
<tr>
<td>A₀</td>
<td>G₁</td>
<td>C₀</td>
<td>$</td>
<td>A₁</td>
<td>G₂</td>
<td>G₀</td>
</tr>
<tr>
<td>A₁</td>
<td>G₂</td>
<td>G₀</td>
<td>A₀</td>
<td>G₁</td>
<td>C₀</td>
<td>$</td>
</tr>
<tr>
<td>C₀</td>
<td>$</td>
<td>A₁</td>
<td>G₂</td>
<td>G₀</td>
<td>A₀</td>
<td>G₁</td>
</tr>
<tr>
<td>G₀</td>
<td>A₀</td>
<td>G₁</td>
<td>C₀</td>
<td>$</td>
<td>A₁</td>
<td>G₂</td>
</tr>
<tr>
<td>G₁</td>
<td>C₀</td>
<td>$</td>
<td>A₁</td>
<td>G₂</td>
<td>G₀</td>
<td>A₀</td>
</tr>
<tr>
<td>G₂</td>
<td>G₀</td>
<td>A₀</td>
<td>G₁</td>
<td>C₀</td>
<td>$</td>
<td>A₁</td>
</tr>
</tbody>
</table>
BWT: Lookups

We can look up where a read matches in our genome

<table>
<thead>
<tr>
<th>$</th>
<th>A_1</th>
<th>G_2</th>
<th>G_0</th>
<th>A_0</th>
<th>G_1</th>
<th>C_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
</tr>
<tr>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
</tr>
<tr>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
</tr>
<tr>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
</tr>
<tr>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
</tr>
<tr>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
</tr>
</tbody>
</table>

Read = “AGG”

Start from last base:

A G G

Luckily there was a match - else no alignment
BWT: Lookups

We can look up where a read matches in our genome

\[
\begin{array}{ccccccc}
F & A_1 & G_2 & G_0 & A_0 & G_1 & C_0 \\
A_0 & G_1 & C_0 & $ & A_1 & G_2 & G_0 \\
A_1 & G_2 & G_0 & A_0 & G_1 & C_0 & $ \\
C_0 & $ & A_1 & G_2 & G_0 & A_0 & G_1 \\
G_0 & A_0 & G_1 & C_0 & $ & A_1 & G_2 \\
G_1 & C_0 & $ & A_1 & G_2 & G_0 & A_0 \\
G_2 & G_0 & A_0 & G_1 & C_0 & $ & A_1 \\
\end{array}
\]

Read = “AGG”

Start from last base:

\[\text{A G G}\]

Use LF mapping to get to \( F \)
BWT: Lookups

We can look up where a read matches in our genome

Read = “AGG”

Start from last base:

A G G

Use LF mapping to get to \( F \)
**BWT: Lookups**

We can look up where a read matches in our genome

<table>
<thead>
<tr>
<th>F</th>
<th>A_1</th>
<th>G_2</th>
<th>G_0</th>
<th>A_0</th>
<th>G_1</th>
<th>C_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
</tr>
<tr>
<td>A_0</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
</tr>
<tr>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
</tr>
<tr>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
</tr>
<tr>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
</tr>
<tr>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
</tr>
</tbody>
</table>

Read = “AGG”

Start from last base:

A G G

This is our match!
# BWT: Lookups

We can look up where a read matches in our genome

<table>
<thead>
<tr>
<th>$</th>
<th>A_1</th>
<th>G_2</th>
<th>G_0</th>
<th>A_0</th>
<th>G_1</th>
<th>C_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
</tr>
<tr>
<td><strong>A_1</strong></td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
</tr>
<tr>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
</tr>
<tr>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
</tr>
<tr>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
</tr>
<tr>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
</tr>
</tbody>
</table>

Read = “AGG”

Start from last base:

```
A G G
```

This is our match!
## BWT: Lookups

We can look up where a read matches in our genome

<table>
<thead>
<tr>
<th>$</th>
<th>A_1</th>
<th>G_2</th>
<th>G_0</th>
<th>A_0</th>
<th>G_1</th>
<th>C_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
</tr>
<tr>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
</tr>
<tr>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
</tr>
<tr>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
</tr>
<tr>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
</tr>
<tr>
<td>G_2</td>
<td>G_0</td>
<td>A_0</td>
<td>G_1</td>
<td>C_0</td>
<td>$</td>
<td>A_1</td>
</tr>
</tbody>
</table>

Read = “AGG”

Start from last base:

```
A G G
```

This is our match!

But where is this in our genome?
BWT: Lookups

We can look up where a read matches in our genome

Read = “AGG”

<table>
<thead>
<tr>
<th>SA</th>
<th>F</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>$ A_1 G_2 G_0 A_0 G_1 C_0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>A_0 G_1 C_0 $ A_1 G_2 G_0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>A_1 G_2 G_0 A_0 G_1 C_0 $</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>C_0 $ A_1 G_2 G_0 A_0 G_1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>G_0 A_0 G_1 C_0 $ A_1 G_2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>G_1 C_0 $ A_1 G_2 G_0 A_0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>G_2 G_0 A_0 G_1 C_0 $ A_1</td>
<td></td>
</tr>
</tbody>
</table>
### BWT: Lookups

We can look up where a read matches in our genome

<table>
<thead>
<tr>
<th>SA</th>
<th>F</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>$</td>
<td>A_1 G_2 G_0 A_0 G_1 C_0</td>
</tr>
<tr>
<td>3</td>
<td>A_0 G_1 C_0 $ A_1 G_2 G_0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>A_1 G_2 G_0 A_0 G_1 C_0 $</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>C_0 $ A_1 G_2 G_0 A_0 G_1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>G_0 A_0 G_1 C_0 $ A_1 G_2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>G_1 C_0 $ A_1 G_2 G_0 A_0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>G_2 G_0 A_0 G_1 C_0 $ A_1</td>
<td></td>
</tr>
</tbody>
</table>

Read = “AGG”

Our read aligns at pos 0

T = AGGAGC$

pos 0

---

36626 - Next Generation Sequencing Analysis
BWT for alignment

- Entire SA is 12Gb for human genome
- FM-index
  - We only store certain parts of the array
  - We can calculate missing parts on the fly

- Human genome can be effectively indexed and searched using 3Gb RAM!
Implementation in BWA

- Burrows Wheeler Aligner (BWA) can use:
  - `bwa aln`: First ~30nt of read as seed
    - Extend around positions with seed match
  - `bwa mem`: Multiple short seeds across the read
    - Extend around positions with several seed matches
Genome: GTAC$

All possible transformations

Lexicographically sorted

The BWT is: __________
Single vs. Paired alignment

- Always get paired end reads (if possible)
- Can map across repeats
- Less mapping errors
Single vs. Paired alignment

- Always get paired end reads (if possible)
- Can map across repeats
- Less mapping errors
Single vs. Paired alignment

- Always get paired end reads (if possible)
- Can map across repeats
- Less mapping errors
Single vs. Paired alignment

- Always get paired end reads (if possible)
- Can map across repeats
- Less mapping errors

Unmapped read can be “rescued” by a good aligning mate
• Coverage/depth is how many times that your data covers the genome (on average)

• Example:
  • N: Number of reads: 5 mill
  • L: Read length: 100
  • G: Genome size: 5 Mbases
  • \( C = \frac{5 \times 100}{5} = 100 \times \)
  • On average there are 100 reads covering each position in the genome
Actual depth

- We aligned reads to the genome - how much do we actually cover?
- Avg. depth ~ 90X
- Range from 0-250X
- Only 50% of the genome was covered with reads
SAM/BAM format

• Sequence Alignment / Map format
• BAM = Binary SAM and zipped - always convert to BAM
• Two sections
  • Header: All lines start with “@”
  • Alignments: All other lines
SAM - Example

Header section

@HD     VN:1.0  SO:coordinate
@SQ     SN:CHROM1_NAME LN:CHROM1_LENGTH
@SQ     SN:CHROM2_NAME LN:CHROM2_LENGTH
@RG     ID:LANE_NAME       PL:PLATFORM     LB:LIBRARY    SM:SAMPLE
@PG     ID:bwa   PN:bwa   VN:0.5.9-r16

Alignment section

ReadName   FLAG CONTIG POS MAPQ CIGAR MATE_CHROM MATE_POS INSERT_SIZE READ_SEQ READ_QUAL TAGS...

Lots(!) of different tags, two important:

XT:A:__    U = Unique, R = Repeat, M = Mate-rescued
NM:I:__    Alignment edit distance