### Today: OptBWTR

<table>
<thead>
<tr>
<th></th>
<th>Time (locate)</th>
<th>Time (count)</th>
<th>Space (words)</th>
</tr>
</thead>
<tbody>
<tr>
<td>r-index [GNP20]</td>
<td>$O(</td>
<td>P</td>
<td>\log \log_w (\sigma + n/r) + occ)$</td>
</tr>
<tr>
<td></td>
<td>$O(</td>
<td>P</td>
<td>+ occ)$</td>
</tr>
<tr>
<td>OptBWTR [NT21]</td>
<td>$O(</td>
<td>P</td>
<td>\log \log_w \sigma + occ)$</td>
</tr>
</tbody>
</table>
Recap: Burrows-Wheeler Transform

- characters (w.r.t. text) preserve order in $L$ and $F$
- $LF$-mapping returns previous character in text

$T = ababcabcabba$

**Diagram:**
- $F$ and $L$ with $LF(i)$ and $LF(j)$ indicating the mapping.

**Table:**

<table>
<thead>
<tr>
<th>$L$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>$b$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>$c$</td>
<td>b</td>
</tr>
<tr>
<td>$c$</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>$b$</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>$c$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>$b$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>$a$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>$c$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>$b$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>$a$</td>
<td>$\alpha$</td>
</tr>
</tbody>
</table>

**Note:**
- $a$, $b$, and $c$ are characters from the text $T$.
- $\alpha$ denotes the leftmost character.
- $LF(i)$ and $LF(j)$ show the mapping from $F$ to $L$.

**Legend:**
- $T$ represents the text.
- $F$ and $L$ are mappings.
- $LF$ indicates the mapping.

**Institute:**
- Institute of Theoretical Informatics, Algorithm Engineering
Recap: Backwards Search in the BWT

Function \text{BackwardsSearch}(P[1..n], C, rank):

1. \hspace{0.5cm} s = 1, \hspace{0.5cm} e = n
2. \hspace{0.5cm} for \hspace{0.5cm} i = m, \ldots, 1 \hspace{0.5cm} do
3. \hspace{1.5cm} s = C[P[i]] + rank_{P[i]}(s - 1) + 1
4. \hspace{1.5cm} e = C[P[i]] + rank_{P[i]}(e)
5. \hspace{1.5cm} if \hspace{0.5cm} s > e \hspace{0.5cm} then
6. \hspace{2cm} return \emptyset
7. \hspace{1cm} return \{s, e\}

- no access to text or SA required
- no binary search
- existential queries are easy
- counting queries are easy
- reporting queries require additional information
- example on the board
Given a text $T$ of length $n$ over an alphabet $\Sigma$ and its BWT, the $r$-index of this text consists of the following data structures:

**Array $I$**
- $I[i]$ stores position of $i$-th run in BWT

**Array $L'$**
- $L'[i]$ stores character of $i$-th run in BWT
- build wavelet tree for $L'$

**Array $R$**
- lengths of BWT runs stably sorted by runs’ characters
- accumulate for each character by performing exclusive prefix sum over run lengths’

**Array $C'$**
- $C'[\alpha]$ stores the start of the run lengths in $R$ for each character $\alpha \in \Sigma$ starting at 0

**Bit Vector $B$**
- compressed bit vector of length $n$ containing ones at positions where BWT runs start and rank-support
Recap: The $r$-Index (2/3)

$	ext{rank}_\alpha(BWT, i)$ with $r$-Index

- compute number $j$ of run ($j = \text{rank}_1(B, i)$)
- compute position $k$ in $R$ ($k = C'[\alpha]$)
- compute number $\ell$ of $\alpha$ runs before the $j$-th run ($\ell = \text{rank}_\alpha(L', j - 1)$)
- compute number $k$ of $\alpha$s before the $j$-th run ($k = R[k + \ell]$)
- compute character $\beta$ of run ($\beta = L'[j]$)
- if $\alpha \neq \beta$ return $k$ ✗ $i$ is not in the run
- else return $k + i - l[j] + 1$ ✗ $i$ is in the run
Lemma: Space Requirements $r$-Index

Given a text $T$ of length $n$ over an alphabet of size $\sigma$ that has $r$ BWT runs, then its $r$-index requires

$$O(r \lg n) \text{ bits}$$

and can answer rank-queries on the BWT in $O(\lg \sigma)$. Given a pattern of length $m$, the $r$-index can answer pattern matching queries in time

$$O(m \lg \sigma)$$
Let $\delta$ be permutation of $[1, r]$ such that

$$LF(\ell_{\delta[1]}) < LF(\ell_{\delta[2]}) < \cdots < LF(\ell_{\delta[r]})$$
### Input and Output Intervals

**T = ababcabcabba$**

<table>
<thead>
<tr>
<th>BWT</th>
<th>a</th>
<th>b</th>
<th>$</th>
<th>c</th>
<th>c</th>
<th>b</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>$</td>
<td>c^2</td>
<td>b^2</td>
<td>a^4</td>
<td>b^2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LF</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>12</td>
<td>13</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>in</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>

- There are $r$ intervals
- Represent domain of $LF$ by intervals
- Solve $LF$ without predecessor queries (we did not use predecessor queries)
- Predecessor queries are bottleneck
Disjoint Interval Sequence & Move Query

**Definition: Disjoint Interval Sequence**

Let \( I = (p_1, q_1), (p_2, q_2), \ldots, (p_k, q_k) \) be a sequence of \( k \) pairs of integers. We introduce a permutation \( \pi \) of \([1, k]\) and sequence \( d_1, d_2, \ldots, d_k \) for \( I \). \( \pi \) satisfies \( q_{\pi[1]} \leq q_{\pi[2]} \leq \cdots \leq q_{\pi[k]} \), and \( d_i = p_{i+1} - p_i \) for \( i \in [1, k] \), where \( p_{k+1} = n + 1 \). We call the sequence \( I \) a disjoint interval sequence if it satisfies the following three conditions:

- \( p_1 = 1 < p_2 < \cdots < p_k \leq n \)
- \( q_{\pi[1]} = 1 \)
- \( q_{\pi[i]} = q_{\pi[i-1]} + d_{\pi[i-1]} \) for each \( i \in [2, k] \).

**Example:**

Let \( T = \text{ababcabcabba}$

<table>
<thead>
<tr>
<th></th>
<th>in</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

**Move Query**

\[ \text{move}(i, x) = (i', x') \]

- \( i \) position in input interval
- \( x \) input interval
- \( i' \) position in output interval
- \( x' \) input interval covering \( i' \)
Answering Move Query

- $D_{\text{pair}} = (p_i, q_i)$ for every interval
- $D_{\text{index}}[i]$ index of input interval containing $q_i$

**Lemma: LF and RLBWT**

- Let $\ell_x < i < \ell_{x+1}$ for some $i \in [1, n]$, then
  \[
  LF(i) = LF(\ell_x) + (i - \ell_x)
  \]
- $LF(\ell_\delta[1]) = 1$ and
  \[
  LF(\ell_\delta[i]) = LF(\ell_\delta[i-1]) + |L_\delta[i-1]|
  \]

**Example on the board**

- $Move(i, x) = (i', x')$
  - $i$ position in input sequence
  - $x$ index of interval containing $i$
  - $i' = q_x + (i - p_x)$
  - $x'$ initially $D_{\text{index}}[x]$
  - scan $D_{\text{pair}}$ from $x'$ until $p'_{x} \geq l'$
  - $x'$ index satisfying condition
Moving for LF

**LF Query**
- input: interval containing an integer $i$
- output: interval containing $LF(i)$

- 1. move to corresponding output interval
- 2. move to input interval containing position $j$
- 3. linear search on at most four intervals

- worst-case intervals
- balance intervals

**Diagram**

$$T = ababcabcabba$$

<table>
<thead>
<tr>
<th>BWT</th>
<th>a</th>
<th>b</th>
<th>$</th>
<th>c</th>
<th>c</th>
<th>b</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>LF</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>12</td>
<td>13</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

**Diagram**

1. move to corresponding output interval
2. move to input interval containing position $j$
3. linear search on at most four intervals
Balance the Move Data Structure (1/2)

Definition: Permutation Graph

- each interval in the input and output sequence is a node
- each input interval \([p_i, p_i + d_i - 1]\) has a single outgoing edge pointing to output interval that contains \(p_i\)
- resulting graph \(G(I)\) has \(k\) edges

\(G(I)\) is out-balanced if each output interval has at most three incoming edges

\[T = ababcabcabba\]

<table>
<thead>
<tr>
<th>BWT</th>
<th>a</th>
<th>b</th>
<th>$</th>
<th>c</th>
<th>c</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>$</td>
<td>c^2</td>
<td>b^2</td>
<td>a^4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LF</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>12</td>
<td>13</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>in</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>out</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>
Balance Move Data Structure (2/2)

- identify intervals with $\geq 5$ incoming edges
- split it “equally”
- each new interval covers at least two input intervals

- number $r'$ of balanced input intervals is $k + r$
- $k$ is number of split operations
- $r$ is number of runs in BWT

Lemma: Size of Out-Balanced Sequence

$k \leq r$ and $r' \leq 2r$

Proof

- output contains at least $k$ big intervals, therefore $r' \geq 2k$
- $r' = r + k$, therefore $2k \leq r + k$
- this gives us $k \leq r$
Data Structures for Backwards Search

- $r'$ balanced input & output intervals for LF queries
- rank & select data structure build on the BWT
  - rank in $O(\log \log_w \sigma)$ time
  - select in $O(1)$ time

- $O(r') = O(r)$ space
- $O(|P| \log \log_w \sigma)$ running time

- $F(I_{LF})$: move data structure for LF
- $L_{first}$: character of each run
- $R(L_{first})$: rank and select support on $L_{first}$

- current interval is $[b, e]$ for $P[i + 1..m]$
- look if $P[i]$ occurs in $[b, e]$
  - $\text{rank}(L_{first}, c, j) - \text{rank}(L_{first}) \geq 1$
- find $\hat{b}, \hat{e}$ marking first/last occurrence of $P[i]$ in $[b, e]$
  - $\hat{b} = \text{select}(L_{first}, c, \text{rank}(L_{first}, c, i - 1) + 1)$
  - $\hat{e} = \text{select}(L_{first}, c, \text{rank}(L_{first}, c, j))$
- use move data structure to find new $b, e$ for $P[i..m]$
**Φ and Its Inverse**

- Use $\Phi^{-1}$ to compute $\text{occs}$ of $SA[b..b + \text{occ} - 1]$
- $\Phi^{-1}(SA[i]) = SA[i + 1]$
- $SA[b..b + \text{occ} - 1] = SA[b], \Phi^{-1}(SA[b]), \Phi^{-1}(\Phi^{-1}(SA[b])), \Phi^{-1}(\Phi^{-1}(\Phi^{-1}(SA[b])))$, ...

- $\Phi^{-1}$ can be represented by $r$ input & output intervals [GNP20]
- Use move data structure on balanced intervals
- Keep track of $SA[b]$

---

<table>
<thead>
<tr>
<th>$T = ababcabcabba$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BWT</td>
</tr>
<tr>
<td>a b c c b a a a b b</td>
</tr>
<tr>
<td>LF</td>
</tr>
<tr>
<td>2 7 1 12 13 8 9 3 4 5 6 10 11</td>
</tr>
<tr>
<td>SA</td>
</tr>
<tr>
<td>13 12 1 9 6 3 11 2 10 7 4 8 5</td>
</tr>
<tr>
<td>$\Phi^{-1}$</td>
</tr>
<tr>
<td>9 10 11 8 13 3 4 5 6 7 2 1 12</td>
</tr>
</tbody>
</table>

| in  | 1 2 3 4 5 6 7 8 9 10 11 12 13 |
| out | 1 2 3 4 5 6 7 8 9 10 11 12 13 |
Conclusion and Outlook

This Lecture
- move data structure
- optimal $O(r)$ space full-text index

Next Lecture
- longest common extension queries
- BIG Recap

Project
- “RESULT” is a string literal in the output
- SA/LCP can be discarded, tests would be appreciated
Bibliography I
