Text Indexing

Lecture 12: Optimal r-Index

Florian Kurpicz

The slides are licensed under a Creative Commons Attribution-ShareAlike 4.0 International License © Creative Commons: www.creativecommons.org/licenses/by-sa/4.0 | commit 0cd47f0 compiled at 2024-01-29-11:42
## 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</td>
<td>$O(</td>
<td>P</td>
<td>\log \log_w (\sigma + n/r) + \text{occ})$</td>
</tr>
<tr>
<td>[GNP20]</td>
<td>$O(</td>
<td>P</td>
<td>+ \text{occ})$</td>
</tr>
<tr>
<td>OptBWTR</td>
<td>$O(</td>
<td>P</td>
<td>\log \log_w \sigma + \text{occ})$</td>
</tr>
<tr>
<td>[NT21]</td>
<td></td>
<td></td>
<td></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
Recap: Burrows-Wheeler Transform

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

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

- Characters (w.r.t. text) preserve order in $L$ and $F$
- $LF$-mapping returns previous character in text
Recap: Burrows-Wheeler Transform

- characters (w.r.t. text) preserve order in $L$ and $F$
- $LF$-mapping returns previous character in text
Recap: Backwards Search in the BWT

Function `BackwardsSearch(P[1..n], C, rank)`:  

```plaintext
1  s = 1, e = n
2  for i = m, ..., 1 do
3      s = C[P[i]] + rank_{P[i]}(s - 1) + 1
4      e = C[P[i]] + rank_{P[i]}(e)
5      if s > e then
6          return ∅
7  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
Recap: The $r$-Index [GNP20] (1/3)

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 🎓
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 $BWT$
Recap: The $r$-Index [GNP20] (1/3)

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 $L'$
Recap: The $r$-Index [GNP20] (1/3)

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’

Recap: The \( r \)-Index [GNP20] (1/3)

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 \( L' \)**
- \( L'[i] \) stores character of \( i \)-th run in BWT
- build wavelet tree for \( L' \)

**Array \( I \)**
- \( I[i] \) stores position of \( i \)-th run in BWT

**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 [GNP20] (1/3)

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)

$\text{rank}_\alpha (\text{BWT}, i)$ 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 $k \uparrow$ $i$ is not in the run
- else return $k + i - I[j] + 1 \uparrow$ $i$ 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

$$O(r \lg n)$$

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

$$O(m \lg \sigma)$$
RLBWT

- partition \( BWT \) into \( r \) substrings
- \( BWT = L_1 L_2 \ldots L_r \)
- \( L_i \) is maximal repetition of same character
- \( \ell_1 = 1 \) and \( \ell_i = \ell_{i-1} + |L_{i-1}| \)
- \( RLBWT = (L_1[1], \ell_1)(L_2[1], \ell_2) \ldots (L_r[1], \ell_r) \)
partition $BWT$ into $r$ substrings

$BWT = L_1L_2 \ldots L_r$

$L_i$ is maximal repetition of same character

\[ \ell_1 = 1 \text{ and } \ell_i = \ell_{i-1} + |L_{i-1}| \]

$RLBWT = (L_1[1], \ell_1)(L_2[1], \ell_2) \ldots (L_r[1], \ell_r)$

let $\delta$ be permutation of $[1, r]$ such that

\[ LF(\ell_{\delta[1]}) < LF(\ell_{\delta[2]}) < \cdots < LF(\ell_{\delta[r]}) \]
partition BWT into r substrings

BWT = L_1 L_2 \ldots L_r

L_i is maximal repetition of same character

\ell_1 = 1 and \ell_i = \ell_{i-1} + |L_{i-1}|

RLBWT = (L_1[1], \ell_1)(L_2[1], \ell_2) \ldots (L_r[1], \ell_r)

let \delta be permutation of [1, r] such that

LF(\ell_{\delta[1]}) < LF(\ell_{\delta[2]}) < \cdots < LF(\ell_{\delta[r]})

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

\[ LF(\ell_{\delta[i-1]}) = LF(\ell_{\delta[i-1]}) + |L_{\delta[i-1]}| \]
## Input and Output Intervals

### Example:

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

### Notes:
- 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 \( i \in [2, k] \).
Disjoint Interval Sequence & Move Query

Definition: Disjoint Interval Sequence

Let \( l = (p_1, q_1), (p_2, q_2), \ldots, (p_k, q_k) \) 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 \( l \). \( \pi \) 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 \( l \) 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 \( i \in [2, k] \).

\[ T = ababcabcabba$ \]

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 $q_i$

example on the board
**Answering Move Query**

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

**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 \( D_{index}[x] \)
  - scan \( D_{pair} \) from \( x' \) until \( p_x' \geq l' \)
  - \( x' \) index satisfying condition
Answering Move Query

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

**Example on the board** 

**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
  \[
  LF(\ell_{\delta[i]}) = LF(\ell_{\delta[i-1]}) + |L_{\delta[i-1]}|
  \]

- $Move(i, x) = (i', x')$
  - $i'$ position in input sequence
  - $x'$ index of interval containing $i$
- $i' = q_x + (i - p_x)$
- $x'$ initially $D_{index}[x]$
- scan $D_{pair}$ 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 $LF(i)$

---

**Example**

$T = \text{ababcabcabba}$

<table>
<thead>
<tr>
<th>BWT</th>
<th>a</th>
<th>b</th>
<th>$</th>
<th>c</th>
<th>c</th>
<th>b</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

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

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

$T = ababcabcabba$

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

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

worst-case intervals
balance intervals
Moving for LF

**LF Query**

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

1. move to corresponding output interval
Moving for LF

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

1. move to corresponding output interval

$T = ababcabcabba$

<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>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>
</tr>
</tbody>
</table>

1. move to corresponding output interval

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
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 \( j \)

**T = ababcabcabba$**

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

**Diagram:**

- Input: \( 1 2 3 4 5 6 7 8 9 10 11 12 13 \)
- Output: \( 1 2 3 4 5 6 7 8 9 10 11 12 13 \)

1. Move to corresponding output interval.
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 $j$

### Example

**$T = ababcabcabba$**

<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>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>
</tr>
</tbody>
</table>

1. move to corresponding output interval
2. move to input interval containing position $j$
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

$T = \text{ababcabcabba}$

<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>

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

---

$T = \text{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>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>
</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

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

---

**Table:**

<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>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>
</tr>
</tbody>
</table>

**Diagram:**

$T = ababcabcabba$

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

**Table and Diagram**

<table>
<thead>
<tr>
<th>( T = \text{ababcabcbab}$</th>
<th>( \text{BWT} )</th>
<th>( \text{LF} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a \ b \ $ \ c \ c \ b \ a \ a \ a \ a \ b \ b )</td>
<td>( a \ b \ $ \ c^2 \ b^2 \ a^4 \ b^2 )</td>
<td>( 2 \ 7 \ 1 \ 12 \ 13 \ 8 \ 9 \ 3 \ 4 \ 5 \ 6 \ 10 \ 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
## 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

### Example
Let's consider the string $T = \text{ababcabcabba}\$.

The BWT of $T$ is:

<table>
<thead>
<tr>
<th>BWT</th>
<th></th>
<th>a</th>
<th>b</th>
<th>$</th>
<th>c</th>
<th>c</th>
<th>b</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>$</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

The LF array is:

| LF  | 2 | 7 | 1 | 12 | 13 | 8 | 9 | 3 | 4 | 5 | 6 | 10 | 11 |

**Diagram:**

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

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

<table>
<thead>
<tr>
<th>BWT</th>
<th>a</th>
<th>b</th>
<th>$</th>
<th>c</th>
<th>c</th>
<th>b</th>
<th>b</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>LF</td>
<td>a</td>
<td>b</td>
<td>$</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>in</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>
<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>
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
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

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 \( k \leq r \)
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
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 $r' \leq 2r$
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
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
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 $L_{first}$
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

- \( F(l_{LF}) \): 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 $\text{occ}$s of $SA[b..b + \text{occ} - 1]$
- $\Phi^{-1}(SA[i]) = SA[i + 1]$
- $SA[b..b + \text{occ} - 1] = SA[b], \Phi^{-1}(SA[b]), \Phi^{-1}(\Phi^{-1}(SA[b])), \Phi^{-1}(\Phi^{-1}(\Phi^{-1}(SA[b]))), \ldots$

### Table: $T = \text{ababcabcabba}$

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

### Table: Input and Output

<table>
<thead>
<tr>
<th></th>
<th>in</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{in}$</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13</td>
</tr>
</tbody>
</table>
**Φ and Its Inverse**

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

- $\Phi^{-1}$ can be represented by $r$ input & output intervals [GPN20]
- use move data structure on balanced intervals
- keep track of $SA[b]$

**T = ababcabcabba$**

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

Linear Time Construction

- ST
- SA
- WT
- LZ
- LCP
- BWT
- FM-Index
- r-Index

SA/LCP can be discarded, tests would be appreciated.

"RESULT" is a string literal in the output.

Next Lecture
- longest common extension queries

BIG Recap

This Lecture

This Lecture

Linear Time Construction
Conclusion and Outlook

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

Next Lecture
- longest common extension queries
- BIG Recap

Linear Time Construction

- ST
- SA
- WT
- LZ
- LCP
- BWT
- FM-Index
- $r$-Index
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

Linear Time Construction

Graph: ST → SA → LCP → WT → LZ → BWT → FM-Index → $r$-Index
Bibliography I
