Karlsruhe Institute of Technology

Text Indexing

Lecture 12: Optimal r-Index

Florian Kurpicz

Today: OptBWTR

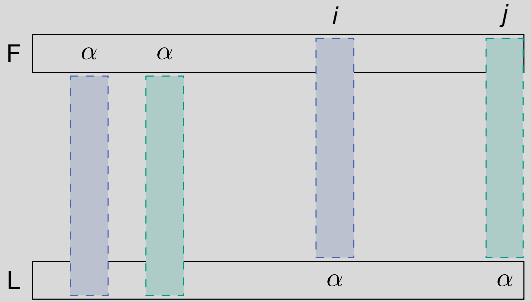
	Time (locate)	Time (count)	Space (words)
r-index [GNP20]	$O(\lvert P\rvert \log\log_w(\sigma + n/r) + occ)$ $O(\lvert P\rvert + occ)$	$O(\lvert P\rvert \log\log_w(\sigma + n/r))$ $O(\lvert P\rvert)$	$O(r)$ $O(r \log\log(\sigma + n/r))$
OptBWTR [NT21]	$O(\lvert P\rvert \log\log_w \sigma + occ)$	$O(\lvert P\rvert \log\log_w \sigma)$	$O(r)$

Recap: Burrows-Wheeler Transform

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

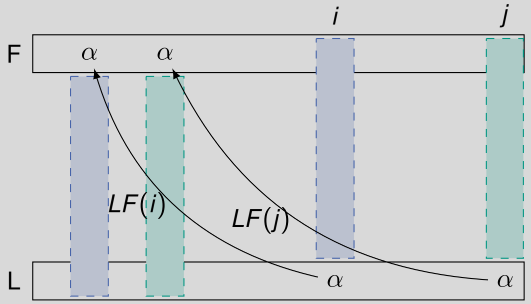
Recap: Burrows-Wheeler Transform

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



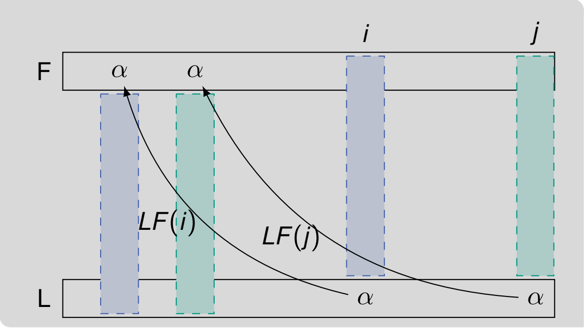
Recap: Burrows-Wheeler Transform

- 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

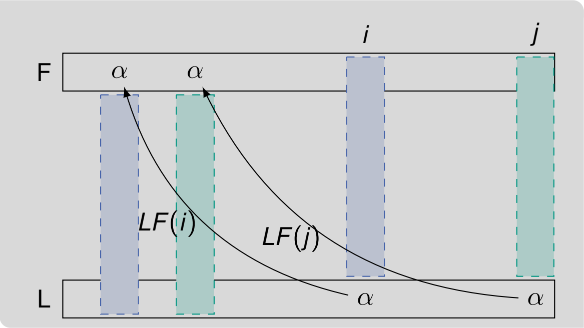


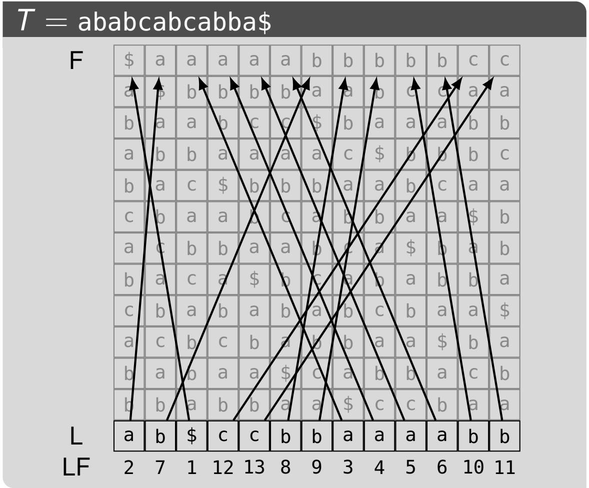
$T = \texttt{ababcabcabba\$}$

F	\$	a	a	a	a	a	b	b	b	b	b	c	c
	a	\$	b	b	b	b	a	a	b	c	c	a	a
	b	a	a	b	c	c	\$	b	a	a	a	b	b
	a	b	b	a	a	a	a	c	\$	b	b	b	c
	b	a	c	\$	b	b	b	a	a	b	c	a	a
	c	b	a	a	b	c	a	b	b	a	a	\$	b
	a	c	b	b	a	a	b	c	a	\$	b	a	b
	b	a	c	a	\$	b	c	a	b	a	b	b	a
	c	b	a	b	a	b	a	b	c	b	a	a	\$
	a	c	b	c	b	a	b	b	a	a	\$	b	a
	b	a	b	a	a	\$	c	a	b	b	a	c	b
	b	b	a	b	b	a	a	\$	c	c	b	a	a
L	a	b	\$	c	c	b	b	a	a	a	a	b	b
LF	2	7	1	12	13	8	9	3	4	5	6	10	11

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*$(P[1..n], C, rank)$**:**

$s = 1, e = n$
for $i = m, \ldots, 1$ **do**
  $s = C[P[i]] + rank_{P[i]}(s - 1) + 1$
  $e = C[P[i]] + rank_{P[i]}(e)$
  if $s > e$ **then**
    return $\emptyset$
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 Σ and its BWT, the r-index of this text consists of the following data structures

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

Given a text T of length n over an alphabet Σ 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 Σ 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 Σ 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 Σ 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

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

Given a text T of length n over an alphabet Σ 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)

$rank_\alpha(BWT, i)$ with *r*-Index

- compute number j of run ($j = rank_1(B, i)$)
- compute position k in R ($k = C'[\alpha]$)
- compute number ℓ of α runs before the j-th run ($\ell = rank_\alpha(L', j-1)$)
- compute number k of αs before the j-th run ($k = R[k + \ell]$)
- compute character β of run ($\beta = L'[j]$)
- if $\alpha \neq \beta$ return k ℹ i is not in the run
- else return $k + i - I[j] + 1$ ℹ i is in the run

Recap: The r-Index (3/3)

Lemma: Space Requirements r-Index

Given a text T of length n over an alphabet of size σ 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

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

RLBWT

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

RLBWT

Karlsruhe Institute of Technology

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

- let δ be permutation of $[1, r]$ such that

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

RLBWT

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

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

$T =$ ababcabcabba$

BWT	a	b	$	c	c	b	b	a	a	a	a	b	b
	a	b	$	c^2		b^2		a^4				b^2	
LF	2	7	1	12	13	8	9	3	4	5	6	10	11

Input and Output Intervals

T = ababcabcabba$

BWT	a	b	$	c	c	b	b	a	a	a	a	b	b
	a	b	$	c^2		b^2		a^4				b^2	
LF	2	7	1	12	13	8	9	3	4	5	6	10	11
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

- there are r intervals
- represent domain of *LF* by intervals

- solve LF without predecessor queries ℹ we did not use predecessor queries
- predecessor queries are bottleneck

Disjoint Interval Sequence & Move Query

Karlsruhe Institute of Technology

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 π of $[1, k]$ and sequence $d_1, d_2, \ldots, d_k$ for I. π 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 $I = (p_1, q_1), (p_2, q_2), \dots, (p_k, q_k)$ be a sequence of k pairs of integers. We introduce a permutation π of $[1, k]$ and sequence $d_1, d_2, \dots, d_k$ for I. π 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 $i \in [2, k]$.

$T =$ `ababcabcabba$`

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

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

- $D_{pair} = (p_i, q_i)$ for every interval
- $D_{index}[i]$ 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 $p'_x \geq I'$
- x' index satisfying condition

Answering Move Query

Karlsruhe Institute of Technology

- $D_{pair} = (p_i, q_i)$ 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 $p'_x \geq I'$
- x' index satisfying condition

Moving for LF

LF Query

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

T = ababcabcabba$

BWT	a	b	$	c	c	b	b	a	a	a	a	b	b
	a	b	$	c^2		b^2		a^4				b^2	
LF	2	7	1	12	13	8	9	3	4	5	6	10	11
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 $LF(i)$

T = ababcabcabba$

BWT	a	b	$	c	c	b	b	a	a	a	a	b	b
	a	b	$	c^2		b^2		a^4				b^2	
LF	2	7	1	12	13	8	9	3	4	5	6	10	11
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

Karlsruhe Institute of Technology

LF Query

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

- 1. move to corresponding output interval

T = ababcabcabba$

BWT	a	b	$	c	c	b	b	a	a	a	a	b	b
	a	b	$	c^2		b^2		a^4				b^2	
LF	2	7	1	12	13	8	9	3	4	5	6	10	11
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

T = ababcabcabba$

BWT	a	b	$	c	c	b	b	a	a	a	a	b	b
	a	b	$	c^2		b^2		a^4				b^2	
LF	2	7	1	12	13	8	9	3	4	5	6	10	11

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.

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$

BWT	a	b	$	c	c	b	b	a	a	a	a	b	b
	a	b	$	c^2		b^2		a^4				b^2	
LF	2	7	1	12	13	8	9	3	4	5	6	10	11
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

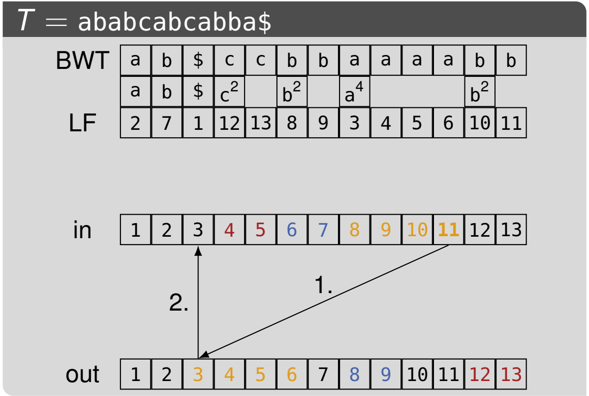
1.

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



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 = ababcabcabba$

BWT	a	b	$	c	c	b	b	a	a	a	a	b	b
	a	b	$	c^2		b^2		a^4				b^2	
LF	2	7	1	12	13	8	9	3	4	5	6	10	11

in	1	2	3	4	5	6	7	8	9	10	**11**	12	13

2. 1.

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

Moving for LF

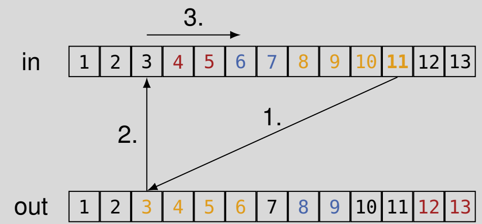
LF Query

- 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
- 3. linear search on at most four intervals

$T =$ ababcabcabba$

BWT	a	b	$	c	c	b	b	a	a	a	a	b	b
	a	b	$	c^2		b^2		a^4				b^2	
LF	2	7	1	12	13	8	9	3	4	5	6	10	11



Moving for LF

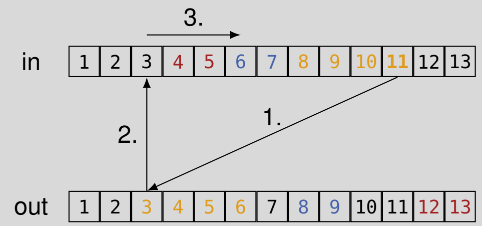
LF Query

- 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

$T =$ ababcabcabba$

BWT	a	b	$	c	c	b	b	a	a	a	a	b	b
	a	b	$	c^2		b^2		a^4				b^2	
LF	2	7	1	12	13	8	9	3	4	5	6	10	11



Moving for LF

LF Query

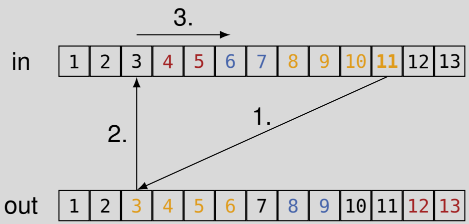
- input: interval containing an integer i
- output: interval containing $LF(i)$

- 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

T = ababcabcabba$

BWT	a	b	$	c	c	b	b	a	a	a	a	b	b
	a	b	$	c^2		b^2		a^4				b^2	
LF	2	7	1	12	13	8	9	3	4	5	6	10	11



Moving for LF

LF Query

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

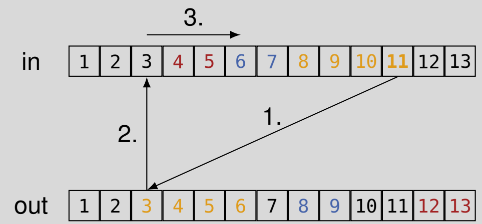
- 1. move to corresponding output interval
- 2. move to input interval containing position j
- 3. linear search on at most four intervals

- worst-case intervals

- balance intervals

$T = \texttt{ababcabcabba\$}$

BWT	a	b	\$	c	c	b	b	a	a	a	a	b	b
	a	b	\$	c^2		b^2		a^4				b^2	
LF	2	7	1	12	13	8	9	3	4	5	6	10	11



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$`

BWT	a	b	\$	c	c	b	b	a	a	a	a	b	b
	a	b	\$	c^2		b^2		a^4				b^2	
LF	2	7	1	12	13	8	9	3	4	5	6	10	11
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

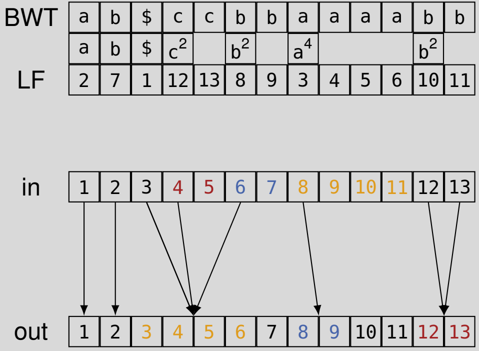
Balance the Move Data Structure (1/2)

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

$T = \texttt{ababcabcabba\$}$



Balance Move Data Structure (2/2)

- identify intervals with ≥ 5 incoming edges
- split it "equally"
- each new interval covers at least two input intervals

Balance Move Data Structure (2/2)

- identify intervals with ≥ 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 ≥ 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 ≥ 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

- $F(I_{LF})$: 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 $[b, e]$
  - $rank(L_{first}, c, j) - rank(L_{first}) \geq 1$
- find $\hat{b}, \hat{e}$ marking first/last occurrence of $P[i]$ in $[b, e]$
  - $\hat{b} = select(L_{first}, c, rank(L_{first}, c, i-1) + 1)$
  - $\hat{e} = select(L_{first}, c, rank(L_{first}, c, j))$
- use move data structure to find new b, e for $P[i..m]$

Φ and Its Inverse

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

$T =$ ababcabcabba$

BWT	a	b	$	c	c	b	b	a	a	a	a	b	b
	a	b	$	c^2		b^2		a^4				b^2	
LF	2	7	1	12	13	8	9	3	4	5	6	10	11
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
Φ^{-1}	9	10	11	8	13	3	4	5	6	7	2	1	12

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

Φ and Its Inverse

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

- Φ^{-1} can be represented by r input & output intervals [GNP20]
- use move data structure on balanced intervals
- keep track of $SA[b]$

$T =$ ababcabcabba$

BWT	a	b	$	c	c	b	b	a	a	a	a	b	b
	a	b	$	c^2		b^2		a^4				b^2	
LF	2	7	1	12	13	8	9	3	4	5	6	10	11
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
Φ^{-1}	9	10	11	8	13	3	4	5	6	7	2	1	12

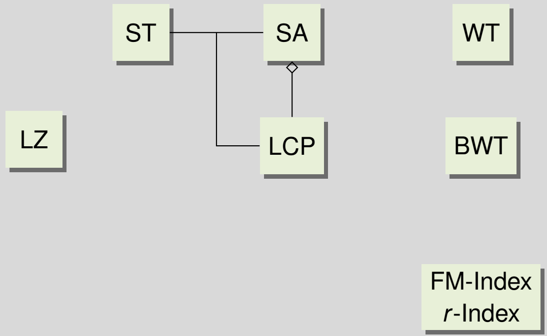
in	1	2	3	4	5	6	7	8	9	10	11	12	13
out	1	2	3	4	5	6	7	8	9	10	11	12	13

Conclusion and Outlook

This Lecture

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

Linear Time Construction



Conclusion and Outlook

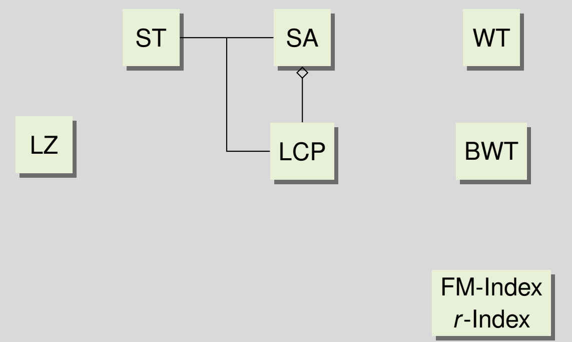
This Lecture

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

Next Lecture

- longest common extension queries
- BIG Recap

Linear Time Construction



Conclusion and Outlook

This Lecture

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

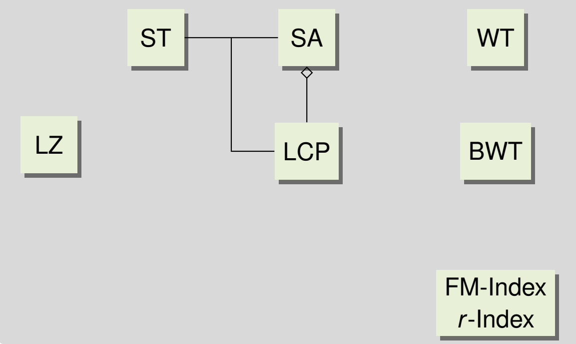
Next Lecture

- 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



Bibliography I

[GNP20] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. "Fully Functional Suffix Trees and Optimal Text Searching in BWT-Runs Bounded Space". In: *J. ACM* 67.1 (2020), 2:1–2:54. DOI: 10.1145/3375890.

[NT21] Takaaki Nishimoto and Yasuo Tabei. "Optimal-Time Queries on BWT-Runs Compressed Indexes". In: *ICALP*. Volume 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 101:1–101:15. DOI: 10.4230/LIPIcs.ICALP.2021.101.