

Übung 9 – Algorithmen II

Florian Kurpicz kurpicz@kit.edu

https://algo2.iti.kit.edu/AlgorithmenII_WS23.php

Institut für Theoretische Informatik - Algorithm Engineering

```
    result = current_weight;
    return true;
}

for( EdgeID eid = graph.edgeBegin( current ); eid != graph.edgeEnd( current ); ++eid ){
    const Edge & edge = graph.getEdge( eid );
    COUNTING( statistic_data.inc( DijkstraStatisticData::TOUCHED_EDGES ); )
    if( edge.forward ){
        COUNTING( statistic_data.inc( DijkstraStatisticData::RELAXED_EDGES ); )
        weight new_weight = edge.weight + current_weight;
        GUARANTEE( new_weight >= current_weight, std::runtime_error, "Weight overflow detected." );
        if( !priority_queue.isReached( edge.target ) ){
            COUNTING( statistic_data.inc( DijkstraStatisticData::SUCCESSFULLY_RELAXED_EDGES ); )
            COUNTING( statistic_data.inc( DijkstraStatisticData::REACHED_NODES ); );
            priority_queue.push( edge.target, new_weight );
        } else {
            if( priority_queue.getCurrentKey( edge.target ) > new_weight ){
                COUNTING( statistic_data.inc( DijkstraStatisticData::SUCCESSFULLY_RELAXED_NODES ); )
                priority_queue.decreaseKey( edge.target, new_weight );
            }
        }
    }
}
```

- Stringology
 - Multikey Quicksort
 - Suche mit Suffix-Arrays

Bentley, Sedgewick (1997)

Three-way Radix Quicksort

- sortiert Elemente mit **mehreren Schlüsseln** wie *msd-Radixsort*
→ z.B. Stellen einer Zahl, Zeichen eines Strings
- für einen Schlüssel wird *Quicksort* mit **drei Fällen** ausgeführt
→ **kleiner als**, **gleich**, **größer als** das Pivotelement

Multikey Quicksort

Ablauf

Function mkqSort(S : Array of String, i : Integer) : Array of String

if $|S| \leq 1$ **then return** S

choose $p \in S$ uniformly at random

return concatenation of

mkqSort($\langle e \in S : e[i] < p[i] \rangle$, i),

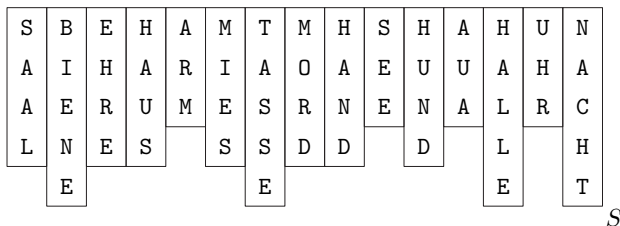
mkqSort($\langle e \in S : e[i] = p[i] \rangle$, $i + 1$),

mkqSort($\langle e \in S : e[i] > p[i] \rangle$, i)

Basisfall

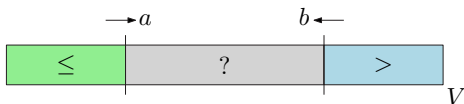
Pivotelement

Rekursion



in-place bei Quicksort für Integer

- teilt Elemente in **kleiner gleich** und **größer** als Pivotelement p
- zwei Zeiger a , b wandern von außen “in die Mitte”
→ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
- Wähle Pivot p und tausche mit erstem Element, setze $a = 2$, $b = n$
- $a \rightarrow a + 1$, solange $V[a] \leq p$,
 $b \rightarrow b - 1$, solange $V[b] > p$,
- Tausch, wenn $V[a] > p$ und $V[b] \leq p$
- Ende, wenn $a > b$

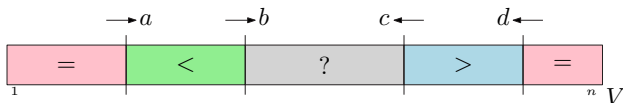


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort

- teilt Elemente in **kleiner**, **gleich** und **größer** als Pivotelement p
- zwei Zeiger b, c wandern von außen “in die Mitte”
- gleiche Elemente werden mit Zeiger a, d “außen” gesammelt
→ Invariante: $V[i \in [a, b) \wedge a \neq b] \leq p$, $V[i < a \vee i > d] = p$, $V[i \in (c, d) \wedge c \neq d] > p$

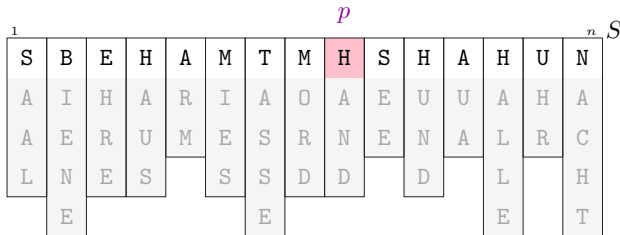


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Algorithmus

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2$, $c = d = n$
- $b \rightarrow b + 1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \rightarrow a + 1$, $c \rightarrow c - 1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \rightarrow d - 1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

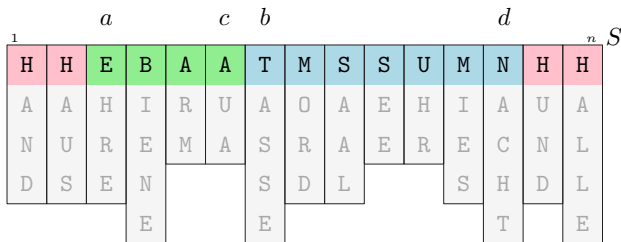


in-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort Umgruppierung

- $r = \min(a - 1, b - a)$
Tausch von r Zeichen zwischen $[1, r)$ und $[b - r, b)$
- $r = \min(d - c, n - d)$
Tausch von r Zeichen zwischen $[c + 1, c + r)$ und $[n - r + 1, n + 1)$



in-place Multikey Quicksort

Zusammenfassung

- *Three-way Radix Quicksort*

Partitionierung in **kleiner**, **gleich**, **größer** über alle Stellen analog zu *msd-Radixsort*

- effizient $\mathcal{O}(|S| \log |S| + d)$

$d \triangleq$ Summe der Länge der unterscheidenden Präfixe

- *in-place* Partitionierung möglich

durch geschicktes Speichern und Verschieben der gleichen Elemente

- sehr einfache Implementierung

Suffix-Arrays

Wiederholung

Suffix-Array SA von T
indiziert alle Suffixe
in sortierter Reihenfolge

Im Beispiel $\mathcal{O}(n^3)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$	
i	SA [i]															
1	15	\$														
2	8	a	b	a	r	b	e	r	\$							
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	\$	
4	10	a	r	b	e	r	\$									
5	5	a	r	h	a	b	a	r	b	e	r	\$				
6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
7	9	b	a	r	b	e	r	\$								
8	4	b	a	r	h	a	b	a	r	b	e	r	\$			
9	12	b	e	r	\$											
10	13	e	r	\$												
11	7	h	a	b	a	r	b	e	r	\$						
12	14	r	\$													
13	3	r	b	a	r	h	a	b	a	r	b	e	r	\$		
14	11	r	b	e	r	\$										
15	6	r	h	a	b	a	r	b	e	r	\$					

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

$n = \text{Textlänge}$, $m = \text{Pattern-Länge}$

- Naiv: $\mathcal{O}(n \cdot m)$
- KMP: $\mathcal{O}(n + m)$
- Mit Suffix-Arrays zunächst: $\mathcal{O}(m \cdot \log n)$
- Optimiert: $\mathcal{O}(m + \log n)$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

- (SA bestimmen)
- finde Start binäre Suche
 - $l = 1, r = n$
 - while** ($l < r$) **do**
 - $q = \lfloor \frac{l+r}{2} \rfloor$
 - if** ($P > T_{SA[q]..SA[q]+m-1}$)
 - then** $l = q + 1$
 - else** $r = q$
- $s = l$
- if** ($P \neq T_{SA[s]..SA[s]+m-1}$)
- then break**
- finde Ende binäre Suche
 - $l = s, r = n$
 - while** ($l < r$) **do**
 - $q = \lceil \frac{l+r}{2} \rceil$
 - if** ($P = T_{SA[q]..SA[q]+m-1}$)
 - then** $l = q$
 - else** $r = q$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
i	15	8	2	10	5	1	9	4	12	13	7	14	3	11	6
$SA[i]$	\$	a b a r b e r \$	a r b a r h a b a r b e r \$	a r b e r \$	a r h a b a r b e r \$	b a r b a r h a b a r b e r \$	b a r b e r \$	b a r h a b a r b e r \$	b e r \$	e r \$	h a b a r b e r \$	r \$	r b a r h a b a r b e r \$	r b e r \$	r h a b a r b e r \$
$q = 8$															

$s = 6, t = 8$

- Verlagerung des Aufwands von Anfrage in Vorverarbeitung
 - einmal Suffix-Array generieren in $\mathcal{O}(n)$,
 - danach **Anfragen** in $\mathcal{O}(m \log n)$ möglich, statt in $\mathcal{O}(m + n)$gut, wenn auf einem Text viele Anfragen stattfinden

- Ausnutzung der Eigenschaften des Suffix-Arrays
 - jeder Substring ist Präfix eines Suffix
 - **alle Substrings liegen "sortiert" vor**
 - mögliche Ausnahme: Substring ist Präfix von Substringdas Suffix-Array indiziert alle Suffixe in sortierter Reihenfolge

Definition:

- $LCP[i]$: Länge des längsten gemeinsamen Präfixes von je zwei lexikographisch benachbarten Suffixen $A[SA[i-1] \dots n]$ und $A[SA[i] \dots n]$

Erweiterung auf beliebige Suffixe

- $LCP[i][j]$: Länge des längsten gemeinsamen Präfix beliebiger lexikographischer Suffixe $A[SA[i] \dots n]$ und $A[SA[j] \dots n]$
- Konstruktion: $\mathcal{O}(n \log n)$ Zeit und Platz
- Zugriff: $\mathcal{O}(1)$

Schnelle Suche mit Suffix-Arrays

Erster Ansatz

Suche: $P = \text{bar}$

- Ziel: kein wiederholtes Vergleichen von Zeichen aus P
- Nutze LCP-Array um Suche zu beschleunigen
- Starte Suche bei mlr
 - $l := \text{LCP}(L, P)$
 - $r := \text{LCP}(R, P)$
 - $mlr := \min(l, r)$
 - Update von l, r , keine Neuberechnung
- Oft $\mathcal{O}(m + \log n)$
- Worst case $\mathcal{O}(m \log n)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$		
i SA[i]	1	15															
	1	8															
	2	8	a	b	a	r	b	e	r	\$							
	3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	\$	
	4	10	a	r	b	e	r	\$									
	5	5	a	r	h	a	b	a	r	b	e	r	\$				
$L = 6$	1		b	a	r	b	a	r	h	a	b	a	r	b	e	r	\$
$q = 7$	9		b	a	r	b	e	r	\$								
	8	4	b	a	r	h	a	b	a	r	b	e	r	\$			
$R = 9$	12		b	e	r	\$											
	10	13	e	r	\$												
	11	7	h	a	b	a	r	b	e	r	\$						
	12	14	r	\$													
	13	3	r	b	a	r	h	a	b	a	r	b	e	r	\$		
	14	11	r	b	e	r	\$										
	15	6	r	h	a	b	a	r	b	e	r	\$					

$$L = 6, R = 9$$

$$l = 3, r = 1$$

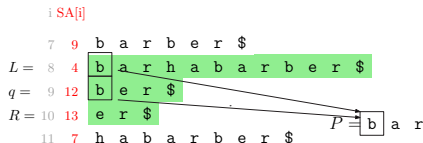
$$mlr := 1 = \min(l, r)$$

Schnelle Suche mit Suffix-Arrays

Redundante Vergleiche

Problem

- Falls $l \neq r \rightarrow$ wiederholtes Vergleichen



Definition

- Vergleich eines Zeichens aus P ist **redundant**, falls das Zeichen vorher schon einmal überprüft wurde.

Ziel

- Beschränke redundante Vergleiche auf $\mathcal{O}(1)$ pro Iteration
- Vergleiche bei $\max(l, r)$ beginnen

Ansatz

- **if** $(l = r)$
start at mlr
Update l, r, L, R
- **if** $(l > r \wedge \text{LCP}[L, q] > l)$
 $L := q + 1$
Update l
- **if** $(l > r \wedge \text{LCP}[L, q] < l)$
 $R := q$
 $r := \text{LCP}[L, q]$
- **if** $(l > r \wedge \text{LCP}[L, q] = l)$
start at l

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{barberac}$

- **if** ($l = r$)
start at mlr
Update l, r, L, R
- **if** ($l > r \wedge \text{LCP}[L, q] > l$)
 $L := q + 1$
Update l
- **if** ($l > r \wedge \text{LCP}[L, q] < l$)
 $R := q$
 $r := \text{LCP}[L, q]$
- **if** ($l > r \wedge \text{LCP}[L, q] = l$)
start at l

b a r b a r h a b a r b e r ...

b a r b e r a b a ...

b a r b e r a b c ...

b a r b e r a c c ...

b a r b e r c b c \$

Laufzeit

- LCP + SA: $\mathcal{O}(m + \log n)$ Vergleiche

b a r b i

$l = 4, r = 4$

Beweisidee

17

Florian Kurpicz:

Übung 9, Fall 1c

werden nur größer

Suche mit Suffix-Arrays

Zusammenfassung

■ Verlagerung des Aufwands von Anfrage in Vorverarbeitung

- einmal Suffix-Array generieren in $\mathcal{O}(n)$,
- danach **Anfragen in $\mathcal{O}(m \log n)$** möglich, statt in $\mathcal{O}(m + n)$

gut, wenn auf einem Text viele Anfragen stattfinden

■ Verhindern redundanter Vergleiche

- einmal Suffix-Array generieren in $\mathcal{O}(n)$,
- einmal LCP-Array generieren in $\mathcal{O}(n)$,
- einmal erweitertes LCP-Array generieren in $\mathcal{O}(n \log n)$,
- danach **Anfragen in $\mathcal{O}(m + \log n)$**

Algorithmen 2

Fortsetzung Übung Stringology

Florian Kurpicz

The slides are licensed under a Creative Commons Attribution-ShareAlike 4.0 International License [@i@: www.creativecommons.org/licenses/by-sa/4.0 | commit cbff5ce](https://creativecommons.org/licenses/by-sa/4.0/) compiled at 2024-01-16-14:11

Burrows-Wheeler-Transformation

Definition: Zyklische Rotation

Sei T ein Text der Länge n , dann ist die i -te **zyklische Rotation**

$$T^{(i)} = T[i..n]T[1..i]$$

- i -te zyklische Rotation ist die Konkatenation des i -ten Suffixes und des $(i - 1)$ -ten prefixes

$T = \text{ababcabcabba\$}$

$T^{(1)}$	$T^{(2)}$	$T^{(3)}$	$T^{(4)}$	$T^{(5)}$	$T^{(6)}$	$T^{(7)}$	$T^{(8)}$	$T^{(9)}$	$T^{(10)}$	$T^{(11)}$	$T^{(12)}$	$T^{(13)}$
a	b	a	b	c	a	b	c	a	b	b	a	\$
b	a	b	c	a	b	c	a	b	b	a	\$	a
a	b	c	a	b	c	a	b	b	a	\$	a	b
b	c	a	b	c	a	b	b	a	\$	a	b	a
c	a	b	c	a	b	b	a	\$	a	b	a	b
a	b	c	a	b	b	a	\$	a	b	a	b	c
b	c	a	b	b	a	\$	a	b	a	b	c	a
c	a	b	b	a	\$	a	b	a	b	c	a	b
a	b	b	a	\$	a	b	a	b	c	a	b	c
b	b	a	\$	a	b	a	b	c	a	b	c	a
b	a	\$	a	b	a	b	c	a	b	c	a	b
a	\$	a	b	a	b	c	a	b	c	a	b	b
\$	a	b	a	b	c	a	b	c	a	b	b	a

Burrows-Wheeler-Transformation

Definition: Zyklische Rotation

Sei T ein Text der Länge n , dann ist die i -te **zyklische Rotation**

$$T^{(i)} = T[i..n]T[1..i]$$

- i -te zyklische Rotation ist die Konkatenation des i -ten Suffixes und des $(i - 1)$ -ten prefixes

Definition: Burrows-Wheeler Transform (Alt.)

Sei T ein Text und M die Matrix, welche alle zyklischen Rotationen von T in lexikographischer Reihenfolge als **Spalten** enthält, dann ist die **BWT** die letzte **Zeile** von M

$T = \text{ababcabcabba\$}$

$T^{(1)}$	$T^{(2)}$	$T^{(3)}$	$T^{(4)}$	$T^{(5)}$	$T^{(6)}$	$T^{(7)}$	$T^{(8)}$	$T^{(9)}$	$T^{(10)}$	$T^{(11)}$	$T^{(12)}$	$T^{(13)}$
a	b	a	b	c	a	b	c	a	b	b	a	\$
b	a	b	c	a	b	c	a	b	b	a	\$	a
a	b	c	a	b	c	a	b	b	a	\$	a	b
b	c	a	b	c	a	b	b	a	\$	a	b	a
c	a	b	c	a	b	b	a	\$	a	b	a	b
a	b	c	a	b	b	a	\$	a	b	a	b	c
b	c	a	b	b	a	\$	a	b	a	b	c	a
c	a	b	b	a	\$	a	b	a	b	c	a	b
a	b	b	a	\$	a	b	a	b	c	a	b	c
b	b	a	\$	a	b	a	b	c	a	b	c	a
b	a	\$	a	b	a	b	c	a	b	c	a	b
a	\$	a	b	a	b	c	a	b	c	a	b	b
\$	a	b	a	b	c	a	b	c	a	b	b	a

Burrows-Wheeler-Transformation

Definition: Zyklische Rotation

Sei T ein Text der Länge n , dann ist die i -te **zyklische Rotation**

$$T^{(i)} = T[i..n]T[1..i]$$

- i -te zyklische Rotation ist die Konkatenation des i -ten Suffixes und des $(i - 1)$ -ten prefixes

Definition: Burrows-Wheeler Transform (Alt.)

Sei T ein Text und M die Matrix, welche alle zyklischen Rotationen von T in lexikographischer Reihenfolge als **Spalten** enthält, dann ist die **BWT** die letzte **Zeile** von M

$T = \text{ababcabcabba}\$$

	$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
\$	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	
a	b	\$	c	c	b	b	a	a	a	a	b	b	

Burrows-Wheeler-Transformation

Definition: Zyklische Rotation

Sei T ein Text der Länge n , dann ist die i -te **zyklische Rotation**

$$T^{(i)} = T[i..n]T[1..i]$$

- i -te zyklische Rotation ist die Konkatenation des i -ten Suffixes und des $(i - 1)$ -ten prefixes

Definition: Burrows-Wheeler Transform (Alt.)

Sei T ein Text und M die Matrix, welche alle zyklischen Rotationen von T in lexikographischer Reihenfolge als **Spalten** enthält, dann ist die **BWT** die letzte **Zeile** von M

$T = \text{ababcabcabba\$}$

	$T^{(13)}$	$T^{(12)}$	$T^{(1)}$	$T^{(9)}$	$T^{(6)}$	$T^{(3)}$	$T^{(11)}$	$T^{(2)}$	$T^{(10)}$	$T^{(7)}$	$T^{(4)}$	$T^{(8)}$	$T^{(5)}$
\$	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	
a	b	\$	c	c	b	b	a	a	a	b	b		

Erste und Letzte Zeile

- zwei wichtige Zeilen in der Matrix
- es gibt eine spezielle Beziehung zwischen den beiden Zeilen
- alle anderen Zeilen werden nicht benötigt

Erste Zeile F

- enthält alle Zeichen von T in sortierter Reihenfolge

Letzte Zeile L

- ist die *BWT*

$T = \text{ababcabcabba\$}$

	$\tau^{(13)}$	$\tau^{(12)}$	$\tau^{(1)}$	$\tau^{(9)}$	$\tau^{(6)}$	$\tau^{(3)}$	$\tau^{(11)}$	$\tau^{(2)}$	$\tau^{(10)}$	$\tau^{(7)}$	$\tau^{(4)}$	$\tau^{(8)}$	$\tau^{(5)}$
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

Ränge der Zeichen

Definition: Rang (im Text)

Sei T ein Text über dem Alphabet Σ , der **Rang** eines Zeichens an der Position $i \in [1, n]$ ist

$$\text{rang}(i) = |\{j \in [1, i]: T[i] = T[j]\}|$$

- der Rang entspricht der Anzahl gleicher Zeichen, die vorher im Text vorkommen

$T = \text{ababcabcabba\$}$

	$T(13)$	$T(12)$	$T(1)$	$T(9)$	$T(6)$	$T(3)$	$T(11)$	$T(2)$	$T(10)$	$T(7)$	$T(4)$	$T(8)$	$T(5)$
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

Ränge der Zeichen

Definition: Rang (im Text)

Sei T ein Text über dem Alphabet Σ , der **Rang** eines Zeichens an der Position $i \in [1, n]$ ist

$$\text{rang}(i) = |\{j \in [1, i]: T[i] = T[j]\}|$$

- der Rang entspricht der Anzahl gleicher Zeichen, die vorher im Text vorkommen

T a b a b c a b c a b b a \$
 rang 1 1 2 2 1 3 3 2 4 4 5 5 1

$T = \text{ababcabcabba\$}$

	$T(13)$	$T(12)$	$T(1)$	$T(9)$	$T(6)$	$T(3)$	$T(11)$	$T(2)$	$T(10)$	$T(7)$	$T(4)$	$T(8)$	$T(5)$
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

Ränge der Zeichen

Definition: Rang (im Text)

Sei T ein Text über dem Alphabet Σ , der **Rang** eines Zeichens an der Position $i \in [1, n]$ ist

$$\text{rang}(i) = |\{j \in [1, i]: T[i] = T[j]\}|$$

- der Rang entspricht der Anzahl gleicher Zeichen, die vorher im Text vorkommen

T a b a b c a b c a b b a \$
 rang 1 1 2 2 1 3 3 2 4 4 5 5 1

$T = \text{ababcabcabba\$}$

	$T(13)$	$T(12)$	$T(1)$	$T(9)$	$T(6)$	$T(3)$	$T(11)$	$T(2)$	$T(10)$	$T(7)$	$T(4)$	$T(8)$	$T(5)$
F	\$	a	a	a	a	a	b	b	b	b	c	c	
	1	5	1	4	3	2	5	1	4	3	2	1	2
	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

Ränge der Zeichen

Definition: Rang (im Text)

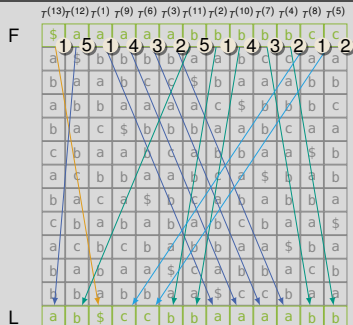
Sei T ein Text über dem Alphabet Σ , der **Rang** eines Zeichens an der Position $i \in [1, n]$ ist

$$\text{rang}(i) = |\{j \in [1, i] : T[i] = T[j]\}|$$

- der Rang entspricht der Anzahl gleicher Zeichen, die vorher im Text vorkommen

T a b a b c a b c a b b a \$
 rang 1 1 2 2 1 3 3 2 4 4 5 5 1

$T = \text{ababcabcabba\$}$



Ränge der Zeichen

Definition: Rang (im Text)

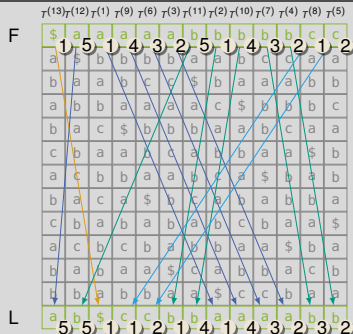
Sei T ein Text über dem Alphabet Σ , der **Rang** eines Zeichens an der Position $i \in [1, n]$ ist

$$\text{rang}(i) = |\{j \in [1, i] : T[i] = T[j]\}|$$

- der Rang entspricht der Anzahl gleicher Zeichen, die vorher im Text vorkommen

T a b a b c a b c a b b a \$
 rang 1 1 2 2 1 3 3 2 4 4 5 5 1

$T = \text{ababcabcabba\$}$



Ränge der Zeichen

Definition: Rang (im Text)

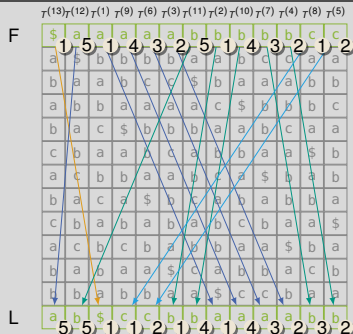
Sei T ein Text über dem Alphabet Σ , der **Rang** eines Zeichens an der Position $i \in [1, n]$ ist

$$\text{rang}(i) = |\{j \in [1, i] : T[i] = T[j]\}|$$

- der Rang entspricht der Anzahl gleicher Zeichen, die vorher im Text vorkommen
- für jedes Zeichen $\alpha \in \Sigma$ ist die Reihenfolge der Ränge in der ersten und letzten Zeile gleich

T	a	b	a	b	c	a	b	c	a	b	b	a	\$
rang	1	1	2	2	1	3	3	2	4	4	5	5	1

$T = \text{ababcabcabba\$}$



LF-Mapping (1/2)

- wir möchten die Zeichen aus der letzten Zeile auf die erste Zeile abbilden
- warum?
 - ermöglicht Mustersuche
 - Transformation *BWT* zurück zu *T*

Definition: *LF*-Mapping

Sei T ein Text der Länge n und SA das Suffix-Array von T , dann ist das *LF*-Mapping eine Permutation von $[1, n]$, so dass

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

$T = \text{ababcabcabba\$}$

	$\tau^{(13)}$	$\tau^{(12)}$	$\tau^{(1)}$	$\tau^{(9)}$	$\tau^{(6)}$	$\tau^{(3)}$	$\tau^{(11)}$	$\tau^{(2)}$	$\tau^{(10)}$	$\tau^{(7)}$	$\tau^{(4)}$	$\tau^{(8)}$	$\tau^{(5)}$
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-Mapping (1/2)

- wir möchten die Zeichen aus der letzten Zeile auf die erste Zeile abbilden
- warum?
 - ermöglicht Mustersuche
 - Transformation *BWT* zurück zu *T*

Definition: *LF*-Mapping

Sei T ein Text der Länge n und SA das Suffix-Array von T , dann ist das *LF*-Mapping eine Permutation von $[1, n]$, so dass

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

$T = \text{ababcabcabba\$}$

	$\tau(13)$	$\tau(12)$	$\tau(1)$	$\tau(9)$	$\tau(6)$	$\tau(3)$	$\tau(11)$	$\tau(2)$	$\tau(10)$	$\tau(7)$	$\tau(4)$	$\tau(8)$	$\tau(5)$
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-Mapping (1/2)

- wir möchten die Zeichen aus der letzten Zeile auf die erste Zeile abbilden
- warum?
 - ermöglicht Mustersuche
 - Transformation *BWT* zurück zu *T*

Definition: *LF*-Mapping

Sei T ein Text der Länge n und SA das Suffix-Array von T , dann ist das *LF*-Mapping eine Permutation von $[1, n]$, so dass

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

$T = \text{ababcabcabba\$}$

	$\tau(13)$	$\tau(12)$	$\tau(1)$	$\tau(9)$	$\tau(6)$	$\tau(3)$	$\tau(11)$	$\tau(2)$	$\tau(10)$	$\tau(7)$	$\tau(4)$	$\tau(8)$	$\tau(5)$
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-Mapping (1/2)

- wir möchten die Zeichen aus der letzten Zeile auf die erste Zeile abbilden
- warum?
 - ermöglicht Mustersuche
 - Transformation *BWT* zurück zu *T*

Definition: *LF*-Mapping

Sei T ein Text der Länge n und SA das Suffix-Array von T , dann ist das *LF*-Mapping eine Permutation von $[1, n]$, so dass

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

$T = \text{ababcabcabba\$}$

	$\tau(13)$	$\tau(12)$	$\tau(1)$	$\tau(9)$	$\tau(6)$	$\tau(3)$	$\tau(11)$	$\tau(2)$	$\tau(10)$	$\tau(7)$	$\tau(4)$	$\tau(8)$	$\tau(5)$
F	\$	a	a	a	a	a	b	b	b	b	b	c	c
	a	b	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-Mapping (1/2)

- wir möchten die Zeichen aus der letzten Zeile auf die erste Zeile abbilden
- warum?
 - ermöglicht Mustersuche
 - Transformation *BWT* zurück zu *T*

Definition: *LF*-Mapping

Sei T ein Text der Länge n und SA das Suffix-Array von T , dann ist das *LF*-Mapping eine Permutation von $[1, n]$, so dass

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

$T = \text{ababcabcabba\$}$

	$\tau(13)$	$\tau(12)$	$\tau(1)$	$\tau(9)$	$\tau(6)$	$\tau(3)$	$\tau(11)$	$\tau(2)$	$\tau(10)$	$\tau(7)$	$\tau(4)$	$\tau(8)$	$\tau(5)$
F	\$	a	a	a	a	a	b	b	b	b	b	c	c
	a	b	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	b	\$	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-Mapping (1/2)

- wir möchten die Zeichen aus der letzten Zeile auf die erste Zeile abbilden
- warum?
 - ermöglicht Mustersuche
 - Transformation *BWT* zurück zu *T*

Definition: *LF*-Mapping

Sei T ein Text der Länge n und SA das Suffix-Array von T , dann ist das *LF*-Mapping eine Permutation von $[1, n]$, so dass

$$LF(i) = j \iff SA[j] = SA[i] - 1$$

$T = \text{ababcabcabba}\$$



LF-Mapping (2/2)

Definition: *C*-Array und *Rang*-Function

Sei T ein Text der Länge n über dem Alphabet Σ ,
 $\alpha \in \Sigma$ und $i \in [1, n]$, dann gilt

$$C[\alpha] = |\{i \in [1, n] : T[i] < \alpha\}|$$

und

$$\text{rang}_\alpha(i) = |\{j \in [1, i] : T[j] = \alpha\}|$$

- C enthält die absolute Anzahl kleinerer Zeichen
- rang_α enthält die Anzahl α s in Präfix $T[1..i]$

LF-Mapping (2/2)

Definition: C -Array und $Rang$ -Function

Sei T ein Text der Länge n über dem Alphabet Σ ,
 $\alpha \in \Sigma$ und $i \in [1, n]$, dann gilt

$$C[\alpha] = |\{i \in [1, n] : T[i] < \alpha\}|$$

und

$$rang_{\alpha}(i) = |\{j \in [1, i] : T[j] = \alpha\}|$$

- C enthält die absolute Anzahl kleinerer Zeichen
- $rang_{\alpha}$ enthält die Anzahl α s in Präfix $T[1..i]$

T	a	b	a	b	c	a	b	c	a	b	b	a	\$
$rang$	1	1	2	2	1	3	3	2	4	4	5	5	1

- Ränge jetzt auf der BWT
- C ist exklusive Präfixsumme über Histogramm



LF-Mapping (2/2)

Definition: *C*-Array und *Rang*-Funktion

Sei T ein Text der Länge n über dem Alphabet Σ ,
 $\alpha \in \Sigma$ und $i \in [1, n]$, dann gilt

$$C[\alpha] = |\{i \in [1, n] : T[i] < \alpha\}|$$

und

$$\text{rang}_\alpha(i) = |\{j \in [1, i] : T[j] = \alpha\}|$$

- C enthält die absolute Anzahl kleinerer Zeichen
- rang_α enthält die Anzahl α s in Präfix $T[1..i]$

T	a	b	a	b	c	a	b	c	a	b	b	a	\$
rang	1	1	2	2	1	3	3	2	4	4	5	5	1

- Ränge jetzt auf der *BWT*
- C ist exklusive Präfixsumme über Histogramm



Definition: *LF*-Mapping (Alt.)

Gegeben die *BWT*, das *C*-array, und die *rang*-Funktion, dann gilt

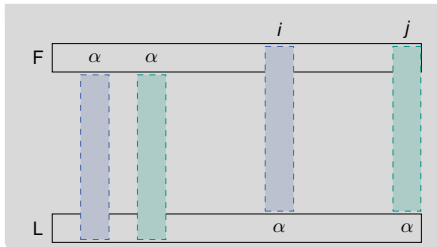
$$LF(i) = C[BWT[i]] + \text{rang}_{BWT[i]}(i)$$

Reversing the BWT (1/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF -Mapping gibt vorheriges Zeichen zurück

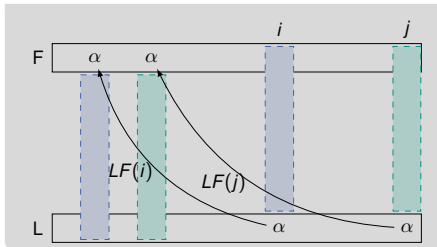
Reversing the BWT (1/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF -Mapping gibt vorheriges Zeichen zurück



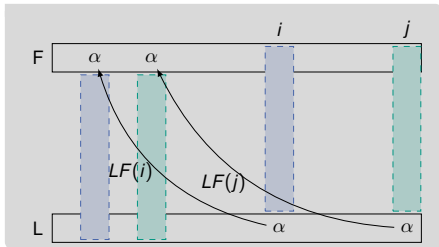
Reversing the BWT (1/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF -Mapping gibt vorheriges Zeichen zurück



Reversing the BWT (1/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF -Mapping gibt vorheriges Zeichen zurück

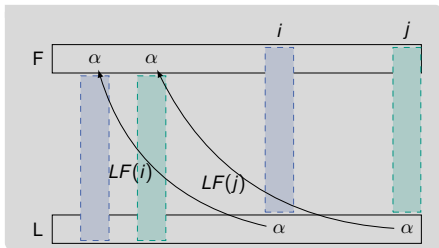


$T = \text{ababcabcabba\$}$

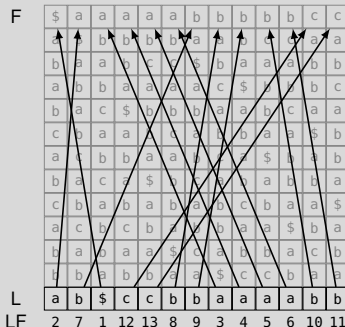
F	\$	a	a	a	a	a	b	b	b	b	b	c	c
a	\$	b	b	b	b	a	a	b	c	c	a	a	
b	a	a	b	c	c	\$	b	a	a	a	b	b	
a	b	b	a	a	a	a	c	\$	b	b	b	c	
b	a	c	\$	b	b	b	a	a	b	c	a	a	
c	b	a	a	b	c	a	b	b	a	a	\$	b	
a	c	b	b	a	a	b	c	a	\$	b	a	b	
b	a	c	a	\$	b	c	a	b	a	b	b	a	
c	b	a	b	a	b	a	b	c	b	a	a	\$	
a	c	b	c	b	a	b	b	a	a	\$	b	a	
b	a	b	a	a	\$	c	a	b	b	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

Reversing the BWT (1/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF -Mapping gibt vorheriges Zeichen zurück



$T = \text{ababcabcabba\$}$



Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF -Mapping gibt vorheriges Zeichen zurück

	1	2	3	4	5	6	7	8	9	0	11	12	13
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

Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF -Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende LF -Mapping auf Ergebnis an und erhalte Zeichen

$$T[n - i] = L[\underbrace{LF(LF(\dots(LF(1))\dots))}_{i-1 \text{ times}}]$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
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

Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF -Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende LF -Mapping auf Ergebnis an und erhalte Zeichen

$$T[n - i] = L[\underbrace{LF(LF(\dots(LF(1))\dots))}_{i-1 \text{ times}}]$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
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

- $T[13] = \$$ und $k = 1$

Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF -Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende LF -Mapping auf Ergebnis an und erhalte Zeichen

$$T[n - i] = L[\underbrace{LF(LF(\dots(LF(1))\dots))}_{i-1 \text{ times}}]$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
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

- $T[13] = \$$ und $k = 1$
- $T[12] = L[1] = a$ und $k = LF(1) = 2$

Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF -Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende LF -Mapping auf Ergebnis an und erhalte Zeichen

$$T[n - i] = L[\underbrace{LF(LF(\dots(LF(1))\dots))}_{i-1 \text{ times}}]$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
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

- $T[13] = \$$ und $k = 1$
- $T[12] = L[1] = a$ und $k = LF(1) = 2$
- $T[11] = L[2] = b$ und $k = LF(2) = 7$

Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF -Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende LF -Mapping auf Ergebnis an und erhalte Zeichen

$$T[n - i] = L[\underbrace{LF(LF(\dots(LF(1))\dots))}_{i-1 \text{ times}}]$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
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

- $T[13] = \$$ und $k = 1$
- $T[12] = L[1] = a$ und $k = LF(1) = 2$
- $T[11] = L[2] = b$ und $k = LF(2) = 7$
- $T[10] = L[7] = b$ und $k = LF(7) = 9$

Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF -Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende LF -Mapping auf Ergebnis an und erhalte Zeichen

$$T[n - i] = \underbrace{L[LF(LF(\dots(LF(1))\dots))]}_{i-1 \text{ times}}$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
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

- $T[13] = \$$ und $k = 1$
- $T[12] = L[1] = a$ und $k = LF(1) = 2$
- $T[11] = L[2] = b$ und $k = LF(2) = 7$
- $T[10] = L[7] = b$ und $k = LF(7) = 9$
- $T[9] = L[9] = a$ und $k = LF(9) = 4$

Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF -Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende LF -Mapping auf Ergebnis an und erhalte Zeichen

$$T[n - i] = L[\underbrace{LF(LF(\dots(LF(1))\dots))}_{i-1 \text{ times}}]$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
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

- $T[13] = \$$ und $k = 1$
- $T[12] = L[1] = a$ und $k = LF(1) = 2$
- $T[11] = L[2] = b$ und $k = LF(2) = 7$
- $T[10] = L[7] = b$ und $k = LF(7) = 9$
- $T[9] = L[9] = a$ und $k = LF(9) = 4$
- $T[8] = L[4] = c$ und $k = LF(4) = 12$

Umkehrung der BWT (2/2)

- Zeichen (bzgl. Text) behalten Reihenfolge in L und F
- LF -Mapping gibt vorheriges Zeichen zurück

- $T[n] = \$$ und $T^{(n)}$ in erster Reihe
- wende LF -Mapping auf Ergebnis an und erhalte Zeichen

$$T[n - i] = L[\underbrace{LF(LF(\dots(LF(1))\dots))}_{i-1 \text{ times}}]$$

	1	2	3	4	5	6	7	8	9	0	11	12	13
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

- $T[13] = \$$ und $k = 1$
- $T[12] = L[1] = a$ und $k = LF(1) = 2$
- $T[11] = L[2] = b$ und $k = LF(2) = 7$
- $T[10] = L[7] = b$ und $k = LF(7) = 9$
- $T[9] = L[9] = a$ und $k = LF(9) = 4$
- $T[9] = L[4] = c$ und $k = LF(4) = 12$
- ...

Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
- Alphabet darf keine Rolle spielen

Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
- Alphabet darf keine Rolle spielen
- $T = a^{n-1}$

Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
 - Alphabet darf keine Rolle spielen
 - $T = a^{n-1}$
- Wie sieht die LZ78-Faktorisierung davon aus?

Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
- Alphabet darf keine Rolle spielen
- $T = a^{n-1}$
- Wie sieht die LZ78-Faktorisierung davon aus?
- $T = a|aa|aaa|\dots$

Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
- Alphabet darf keine Rolle spielen
- $T = a^{n-1}$
- Wie sieht die LZ78-Faktorisierung davon aus?
- $T = a|aa|aaa|\dots$

- Können wir das genauer sagen?

Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
- Alphabet darf keine Rolle spielen
- $T = a^{n-1}$

- Wie sieht die LZ78-Faktorisierung davon aus?
- $T = a|aa|aaa| \dots$

- Können wir das genauer sagen?
- $T = a|aa|aaa| \dots |a^{k-1}|a^k|a^{n-k(k+1)/2}$

Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
- Alphabet darf keine Rolle spielen
- $T = a^{n-1}$

- Wie sieht die LZ78-Faktorisierung davon aus?
- $T = a|aa|aaa| \dots$

- Können wir das genauer sagen?
- $T = a|aa|aaa| \dots |a^{k-1}|a^k|a^{n-k(k+1)/2}$
- für $k(k+1)/2 < n$

Eigenschaften von LZ78

Alte Klausuraufgabe

Zeigen Sie, dass die LZ78-Faktorisierung für einen Text der Länge n über einem konstanten Alphabet bis zu $\Theta(\sqrt{n})$ Faktoren erzeugt.

- Bei welchen Eingaben erzeugen wir möglichst viele Faktoren?
- Alphabet darf keine Rolle spielen
- $T = a^{n-1}$

- Wie sieht die LZ78-Faktorisierung davon aus?
- $T = a|aa|aaa| \dots$

- Können wir das genauer sagen?
- $T = a|aa|aaa| \dots |a^{k-1}|a^k|a^{n-k(k+1)/2}$
- für $k(k+1)/2 < n$
- also für $k \in \Theta(\sqrt{n})$

Ende!



Feierabend!