Projekt Sequenzanalyse

Uwe Baier

Institut für theoretische Informatik
Universität Ulm

14.4.2016

Inhalt

Einführung

Sequenzanalyse wird eingesetzt in...

- Genomanalyse
- Datenbanken
- Information Retrieval
- Datenkompression
- ...

Sequenzanalyse ermöglicht effiziente Operationen auf sehr großen Sequenzen $\Rightarrow$ unverzichtbar bei großen Datenmengen

In diesem Projekt

- Datenstrukturen zur Textindizierung (effiziente Suche und Textextraktion)
- Konstruktion der Datenstrukturen / benötigten Komponenten
- Überprüfung theoretischer Resultate per experimenteller Evaluation
- Auch eigene Themenvorschläge sind möglich

Formales

- Projekt: 8/16 LP
- Treffen nach Vereinbarung
- Voraussichtliches Projektende: Ende SS16, Anfang bis Mitte WS16/17
- Umfang je nach Gruppengröße

Aufgaben

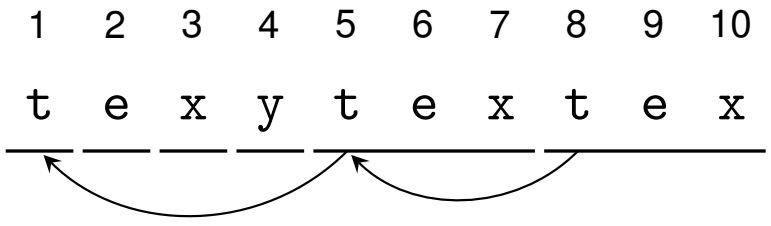
- Lauffähige Implementierung erstellen
- Verfahren mittels Experimenten testen
- Kleinen Report erstellen, in dem Verfahren, eigene Strategien und experimentelle Resultate enthalten sind
- Präsentieren der Ergebnisse (Termin je nach Projektstand und Vereinbarung)

Lempel-Ziv-Index

- LZ77: verlustfreies Kompressionsverfahren, z.B. in `gzip`
- LZ-Index: Textindizierung basierend auf LZ77
- Unterstütze Operationen: Textsuche und Textextraktion
- Speicherverbrauch: **asymptotisch fast gleich** zur Größe der LZ77-Kodierung

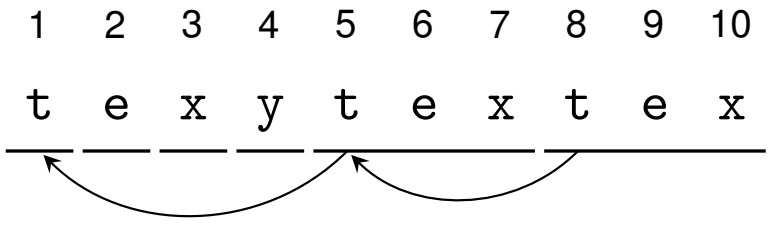
Lempel – Ziv 77

Gegeben: Text $S =$ `texitextex`
Idee: Stelle Text als einzelne Zeichen oder als Wiederholung bereits betrachteter Textstücke dar



Lempel – Ziv 77

Gegeben: Text $S =$ `texitextex`
Idee: Stelle Text als einzelne Zeichen oder als Wiederholung bereits betrachteter Textstücke dar

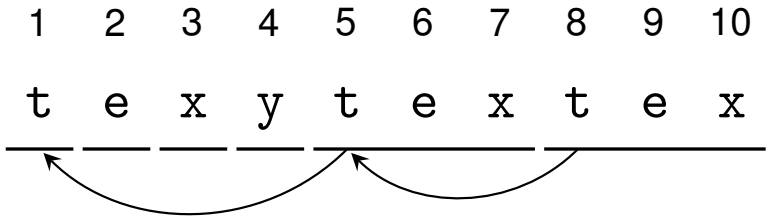


Zu den Teilstücken (*LZ – Faktoren*):

- *src*: vorherige Position der Wiederholung, oder Buchstabe bei *length* = 0
- *length*: Länge eines Faktors (bei 0 enthält *src* einen Buchstaben)

Lempel – Ziv 77

Gegeben: Text $S = \texttt{texitextex}$
Idee: Stelle Text als einzelne Zeichen oder als Wiederholung bereits betrachteter Textstücke dar



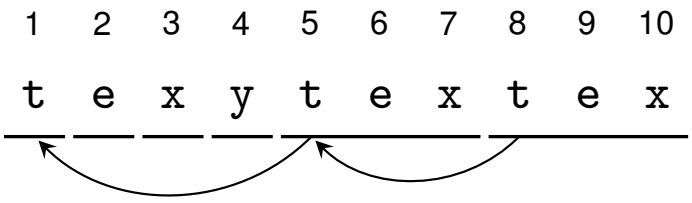
Zu den Teilstücken (*LZ – Faktoren*):

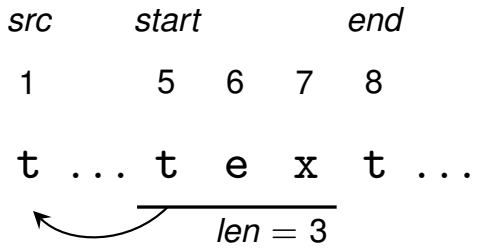
- *src*: vorherige Position der Wiederholung, oder Buchstabe bei *length* = 0
- *length*: Länge eines Faktors (bei 0 enthält *src* einen Buchstaben)

LZ – Faktoren für $S = \texttt{texitextex}$ im Format (*src*, *length*):

(‘t‘, 0) (‘e‘, 0) (‘x‘, 0) (‘y‘, 0) (1, 3) (5, 3)

LZ77 – Kodierung





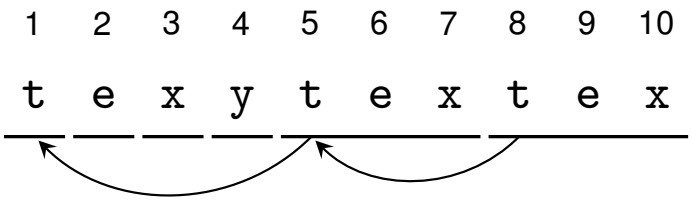
konventionell: (*src*, *len*) – Paare

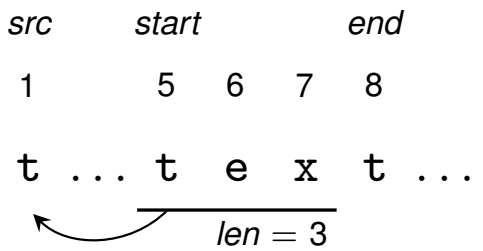
(‘t‘, 0)
(‘e‘, 0)
(‘x‘, 0)
(‘y‘, 0)
(1 , 3)
(5 , 3)

Für einen Faktor F mit $F > 1$ gilt:

- $len(F) = end(F) - end(F-1)$
- $start(F) = end(F-1)$

LZ77 – Kodierung





konventionell: (*src*, *len*) – Paare

(‘t‘, 0)
(‘e‘, 0)
(‘x‘, 0)
(‘y‘, 0)
(1 , 3)
(5 , 3)

alternativ: (*src*, *end*) – Paare

(‘t‘, 2)
(‘e‘, 3)
(‘x‘, 4)
(‘y‘, 5)
(1 , 8)
(5 , 11)

Für einen Faktor F mit $F > 1$ gilt:

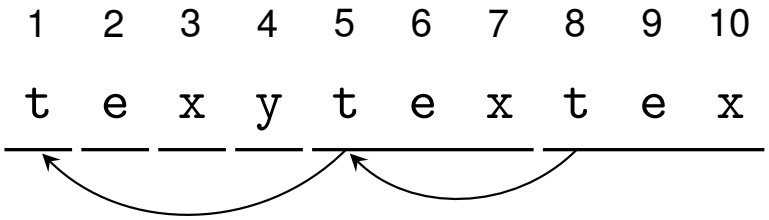
- $len(F) = end(F) - end(F-1)$
- $start(F) = end(F-1)$

$\Rightarrow$ durch alternative Kodierung alle Kenngrößen in $O(1)$ berechenbar

LZ77 - Textextraktion

Gegeben: Position i und Länge l
Idee: Folge "Grammatik"

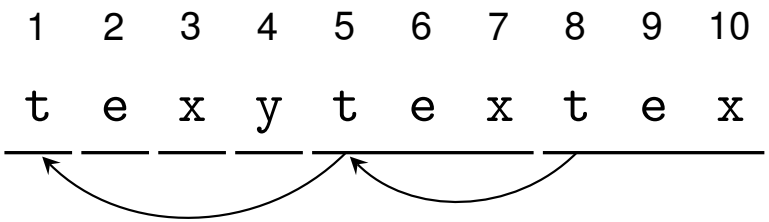
Gesucht: $S[i..i+l)$



LZ77 - Textextraktion

Gegeben: Position i und Länge l
Idee: Folge "Grammatik"

Gesucht: $S[i..i+l)$



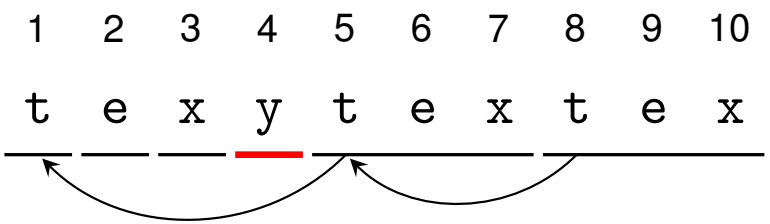
LZ-Faktorisierung:
(*src*, *end*) - Paare
(‘t‘, 2)
(‘e‘, 3)
(‘x‘, 4)
(‘y‘, 5)
(1 , 8)
(5 , 11)

$\Rightarrow$ Zugehörigen Faktor einer Position finden z.B. per binärer Suche

LZ77 - Textextraktion

Gegeben: Position i und Länge l
Idee: Folge "Grammatik"

Gesucht: $S[i..i+l)$



Beispiel $S[4..9) = \mathtt{y}$

$i = 4$, $l = 5$

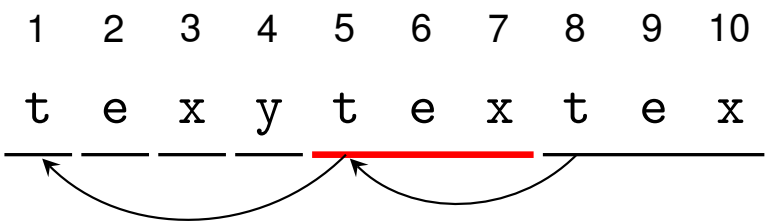
LZ-Faktorisierung:
(src, end) - Paare
(‘t‘, 2)
(‘e‘, 3)
(‘x‘, 4)
⇒(‘y‘, 5)
(1 , 8)
(5 , 11)

⇒ Zugehörigen Faktor einer Position finden z.B. per binärer Suche

LZ77 - Textextraktion

Gegeben: Position i und Länge l
Idee: Folge "Grammatik"

Gesucht: $S[i..i+l)$



Beispiel $S[4..9) = \mathtt{y}$

$i = 5$, $l = 4$

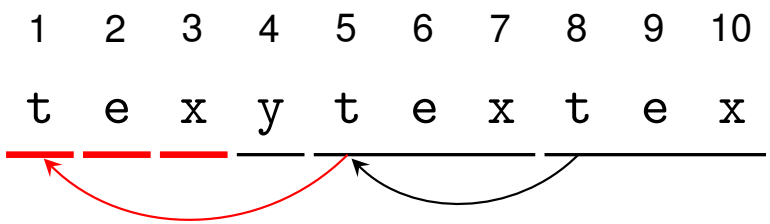
LZ-Faktorisierung:
(src, end) - Paare
(‘t‘, 2)
(‘e‘, 3)
(‘x‘, 4)
(‘y‘, 5)
⇒(1 , 8)
(5 , 11)

⇒ Zugehörigen Faktor einer Position finden z.B. per binärer Suche

LZ77 - Textextraktion

Gegeben: Position i und Länge l Gesucht: $S[i..i+l)$

Idee: Folge "Grammatik"



Beispiel $S[4..9) =$ `ytex`

$i = 5$, $l = 4$
$\rightarrow i = 1$, $l = 3$

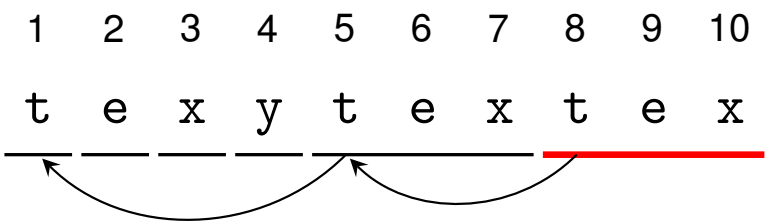
LZ-Faktorisierung:
(src, end) - Paare
$\Rightarrow$(‘t‘, 2)
$\Rightarrow$(‘e‘, 3)
$\Rightarrow$(‘x‘, 4)
(‘y‘, 5)
(1 , 8)
(5 , 11)

$\Rightarrow$ Zugehörigen Faktor einer Position finden z.B. per binärer Suche

LZ77 - Textextraktion

Gegeben: Position i und Länge l
Idee: Folge "Grammatik"

Gesucht: $S[i..i+l)$



Beispiel $S[4..9) =$ `ytex`

$i = 8$, $l = 1$

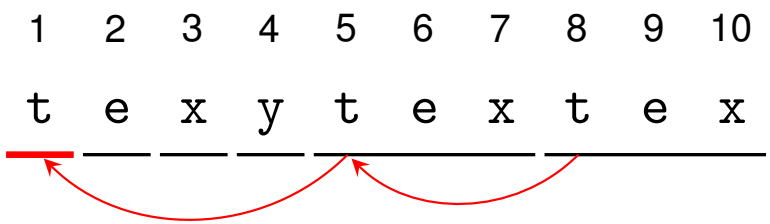
LZ-Faktorisierung:
(src, end) - Paare
(‘t‘, 2)
(‘e‘, 3)
(‘x‘, 4)
(‘y‘, 5)
(1 , 8)
$\Rightarrow$(5 , 11)

$\Rightarrow$ Zugehörigen Faktor einer Position finden z.B. per binärer Suche

LZ77 - Textextraktion

Gegeben: Position i und Länge l
Idee: Folge "Grammatik"

Gesucht: $S[i..i+l)$



Beispiel $S[4..9) =$ `ytext`

$i = 8\ ,\ l = 1$
$\rightarrow i = 5\ ,\ l = 1$
$\rightarrow i = 1\ ,\ l = 1$

LZ-Faktorisierung:
(src, end) - Paare
$\Rightarrow$(‘t‘, 2)
(‘e‘, 3)
(‘x‘, 4)
(‘y‘, 5)
(1 , 8)
(5 , 11)

$\Rightarrow$ Zugehörigen Faktor einer Position finden z.B. per binärer Suche

LZ77 - Textextraktion

Laufzeit hängt von der Höhe des Grammatikbaumes ab.
Laufzeitverbesserungen möglich durch

- Speicherung von Pfaden innerhalb der Grammatik
- Benutzung anderer zusätzlicher Textindizierungen (z.B. run-length-coded BWT)

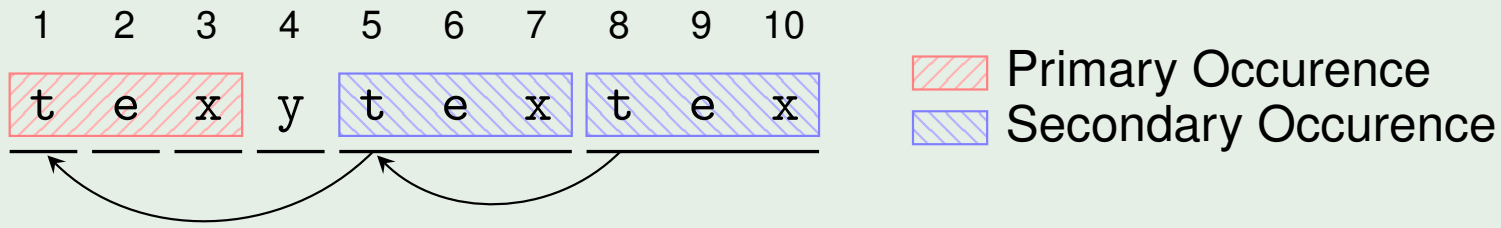
Textsuche

Gegeben: Pattern P, $|P| = m$.
Gesucht: Alle Positionen $p \in \{1, \ldots, n\}$ mit $S[p..p+m) = P$.

Idee bei LZ77: unterteile Suchtreffer in zwei Klassen:

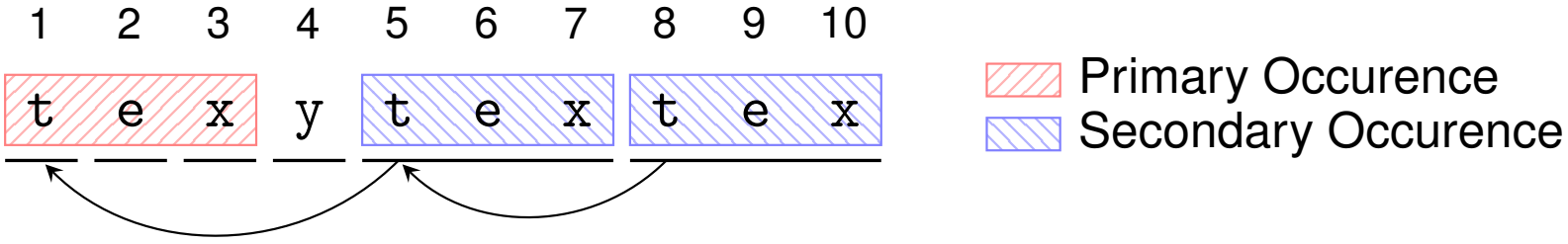
- Primäres Auftreten (*Primary Occurence*):
  Auftreten des Pattern im Text verläuft über Grenzen von mindestens zwei LZ-Faktoren.
- Sekundäres Auftreten (*Secondary Occurence*):
  Auftreten ist innerhalb des Textes vollständig in einem Faktor enthalten.

Suchtreffer für Pattern $P =$ `tex`



Beobachtung Textsuche

- Jedes sekundäre Auftreten setzt ein vorhergehendes primäres Auftreten voraus
- Bestimme erst primäre, dann (rekursiv) sekundäre Auftreten



Idee zu primären Auftreten

Splitte Pattern P an jeder möglichen Position in einen Präfixteil P_p und einen Suffixteil P_s. Für jeden möglichen Split $P = P_p P_s$, berechne

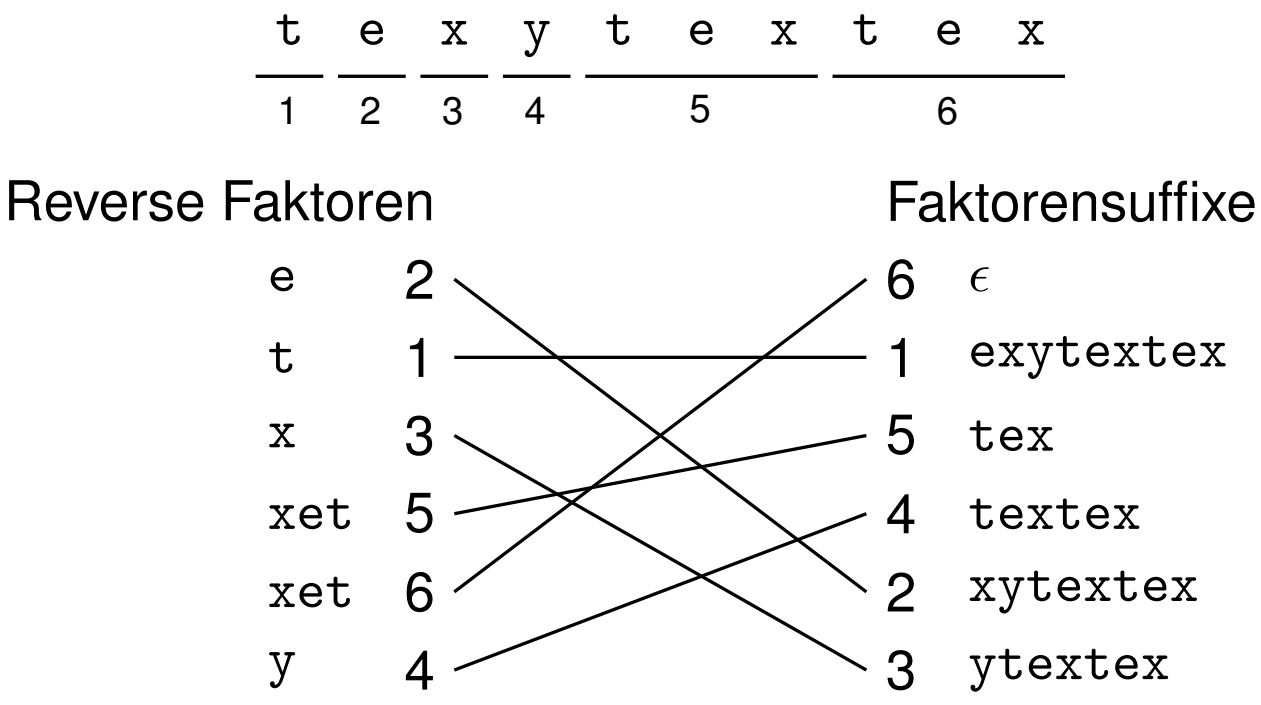
- Menge der LZ-Faktoren F, die mit Präfixteil P_p enden
- Menge der LZ-Faktoren F, deren darauf folgende Suffixe $S_{end(F)}$ mit Suffixteil P_s beginnen
- Bilde die Schnittmenge beider Mengen

Sei $\mathcal{P}$ die Vereinigung der so berechneten Mengen pro Split. Dann gilt für jeden LZ-Faktor $F \in \mathcal{P}$:

$$S[end(F) - |P_p| \ldots end(F) + |P_s|) = P$$

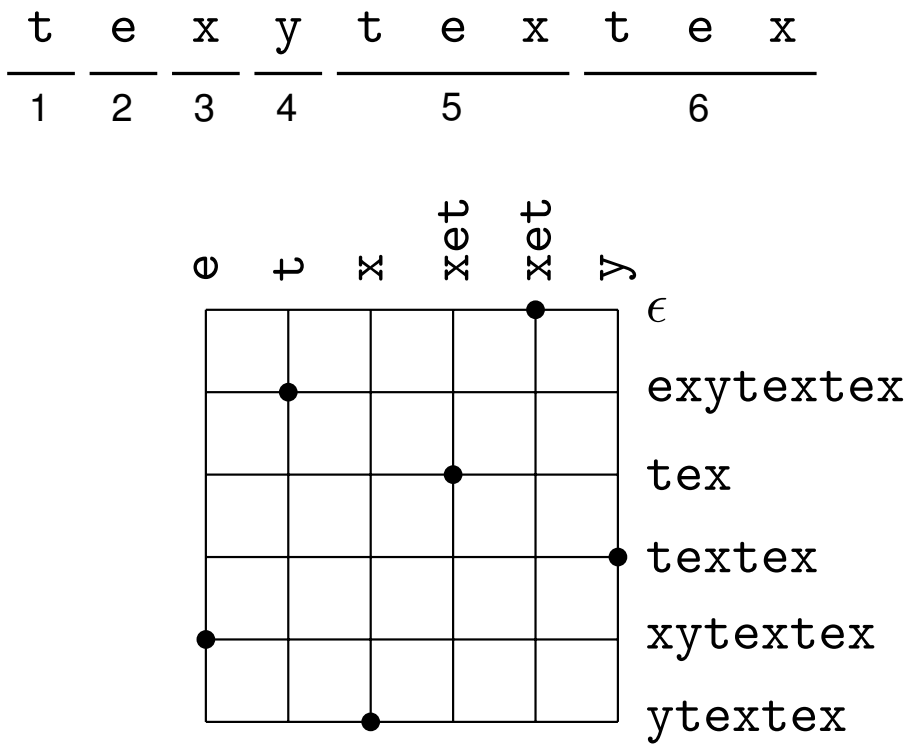
Berechnung primärer Auftreten

- Berechnung von Faktoren, die mit Präfixteil enden: Bereichssuche in lexikographisch sortierten reversen Faktoren
- Berechnung von Faktoren, deren Suffixe mit Suffixteil beginnen: Bereichssuche in lexikographisch sortierten Suffixen



Berechnung primärer Auftreten

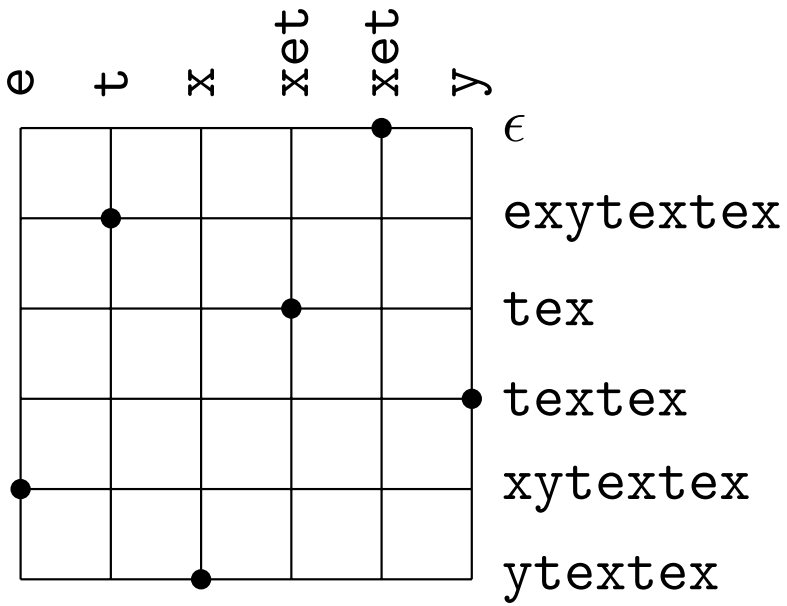
- Berechnung der Schnittpunkte: trage Assoziation der lexikographisch sortierten reversierten Phrasen mit lexikographisch sortierten Suffixen auf Raster ein
- Schnittmenge entspricht Punkten im aufgespannten Bereich



Berechnungsbeispiel

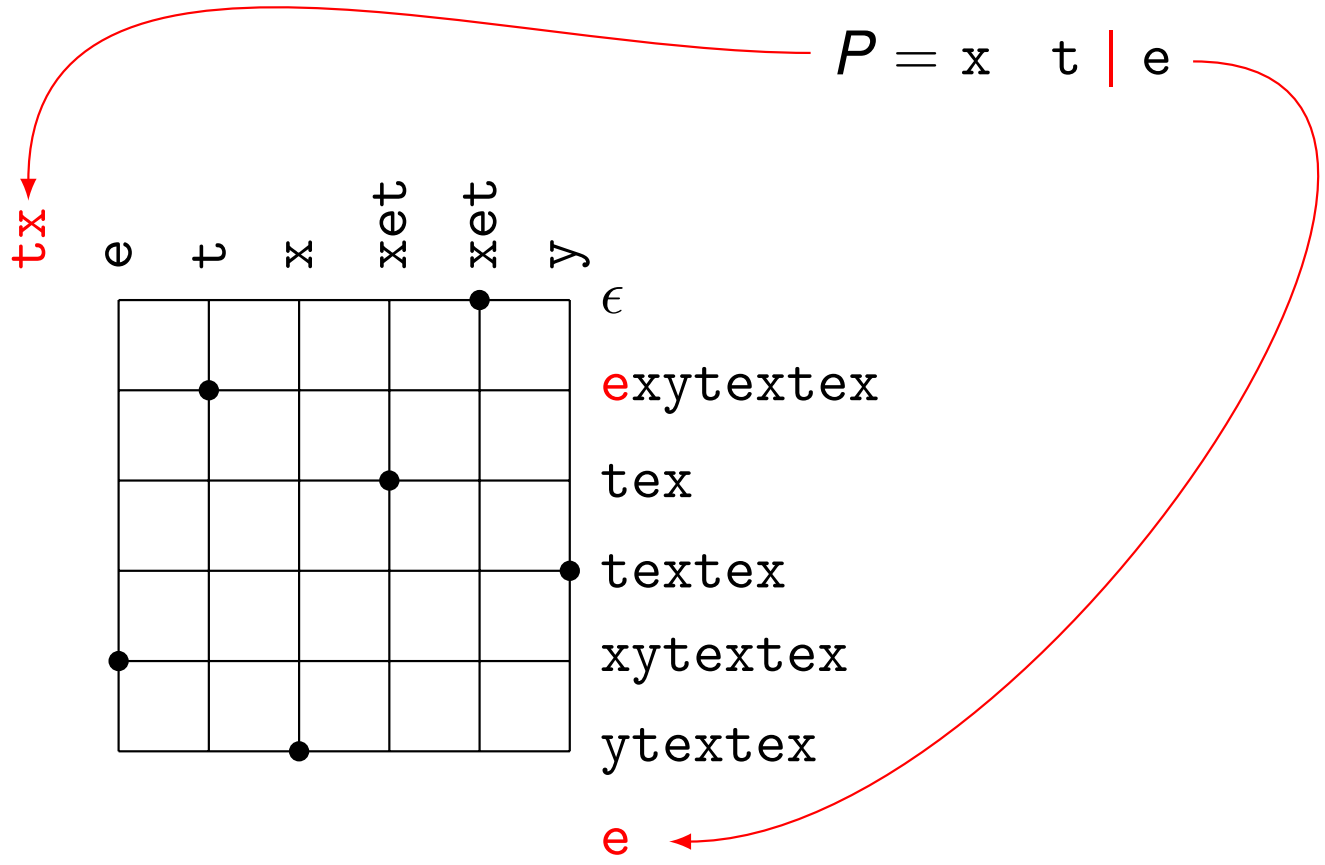
t e x y t e x t e x

$P =$ x t e



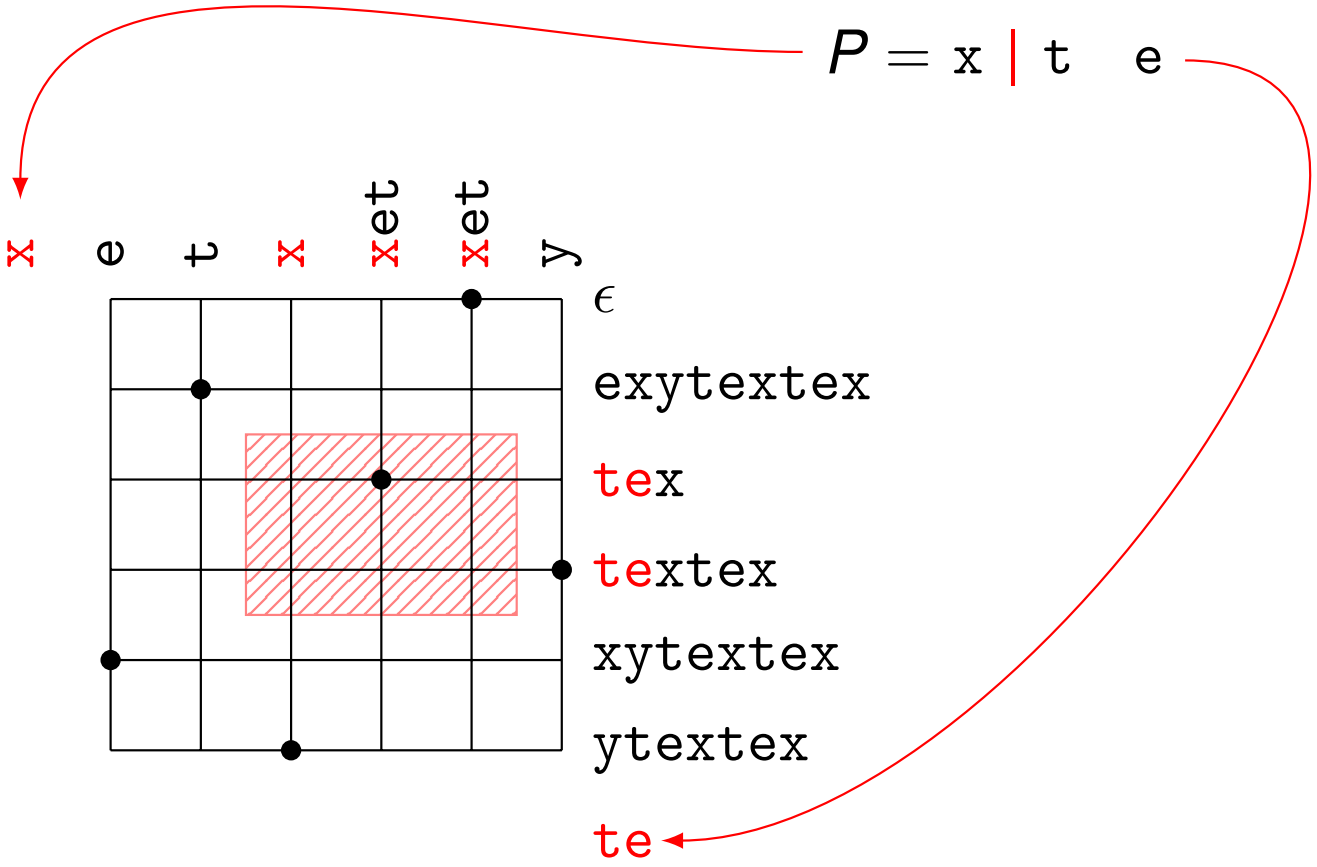
Berechnungsbeispiel

t e x y t e x t e x



Berechnungsbeispiel

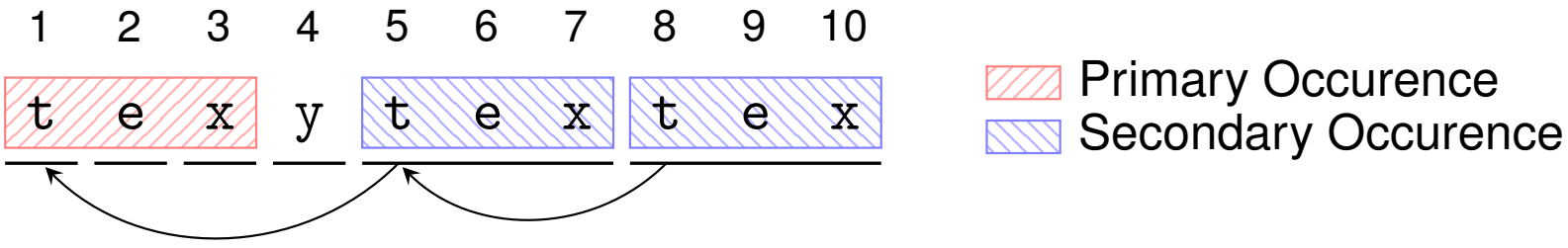
t e x y t e x t e x



Weitere Bemerkungen

- Suche nach lexikographischen Bereichen: z.B. per Trie
- Rasterimplementierung: z.B. per Wavelet Tree / Wavelet Matrix, unterstützen Bereichssuche
- Punkte auf Raster in Faktorindizes umwandeln: z.B. per "Faktorarray", Verweis von lexikographischer auf tatsächliche Faktornummer
- Berechnung von Positionen: subtrahiere Anzahl Zeichen in Präfixteil von Endposition eines entsprechenden Faktors

Sekundäre Auftreten



Idee analog zur Suche nach primärem Auftreten: pro gefundener Position p

- bestimme Faktoren F mit $src(F) \leq p$
- bestimme Faktoren F mit $src(F) + len(F) \geq p + m$
- bilde Schnittmenge

Neue Position des Patterns in einem Faktor F:

$$\tilde{p} = start(F) + p - src(F)$$

Berechnung sekundärer Auftreten

„Rekursive“ Berechnung von sekundären Auftreten

1: let $\mathcal{O}$ be a list of all primary occurences of pattern P
2: $it \leftarrow firstelement(\mathcal{O})$
3: **while** $it \neq$ `nil` **do**
4: $p \leftarrow value(it)$
5: let $\mathcal{F}$ be the set of all factors F with
$src(F) \leq p$ and
$src(F) + len(F) \geq p + m$
6: **for all** $F \in \mathcal{F}$ **do**
7: add $start(F) + p - src(F)$ to the end of list $\mathcal{O}$
8: **end for**
9: $it \leftarrow nextelement(it)$
10: **end while**
11: output $\mathcal{O}$

Weitere Bemerkungen zu sekundären Auftreten

- Bereichssuche nach Faktoren F mit $src(F) \leq p$: Speichere Faktoren nach $src(F)$ geordnet in Array, verwende binäre Suche für entsprechenden Bereich
- Bereichssuche nach Faktoren F mit $src(F) + len(F) \geq p + m$: analog
- Schnittmenge: verwende Raster mit geordneten Indizes, analog zur Suche nach primären Auftreten

Lempel-Ziv-Index Variante 1 – Grammatikbasiert

- **Textextraktion**
  Speichere Pfade in der LZ-Grammatik für schnellere Textextraktion
- **Textsuche**
  - **lexikographische Bereichssuche**
    per *Patricia Trees* (komprimierte Tries) und Textextraktion
  - **Suche im Raster**
    per *Wavelet Tree*, oder (je nach Umfang) mittels besseren Datenstrukturen
  - **Sekundäre Auftreten**
    per *Y-Fast-Trie* und *Range Maximum Queries*

Lempel-Ziv-Index Variante 2 – BWT-supported

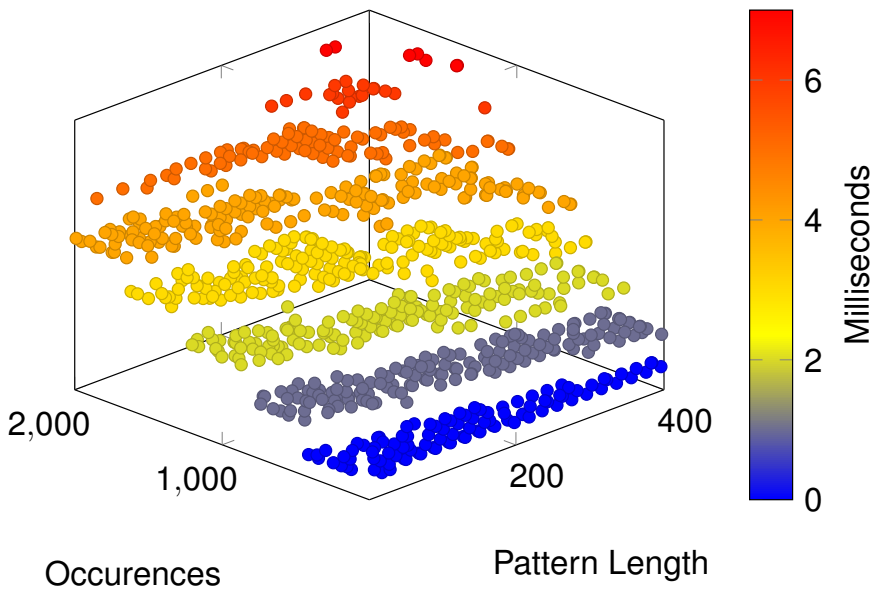
Speichere zusätzlich zu LZ77 - Faktorisierung run-length-coded BWT von S und $rev(S)$ ab.

- **Textextraktion**
  Mittels Intervallen und Rückwärtssuche in BWT($rev(S)$)
- **Textsuche**
  - **lexikographische Bereichssuche**
    per Rückwärtssuche in BWT(S) und BWT($rev(S)$)
  - **Suche im Raster**
    per *Wavelet Tree*/*Wavelet Matrix*
  - **Sekundäre Auftreten**
    z.B. per *Wavelet Tree* und binärer Suche

  Durch Rückwärtssuche auch Möglichkeit zu approximativer Textsuche

Lempel-Ziv-Index – Ziele

- Überprüfung theoretische Resultate per Implementierung und Experimenten
- Darstellung der Ergebnisse, z.B. für Patternsuche



Lempel-Ziv-Index – Anmerkungen

Thema recht komplex, aber

- Vorarbeit ist bereits gemacht (lauffähige Implementierung mit Tests existiert bereits)
- Fokus auf Entwicklung der Komponenten, keine Entwicklung "from scratch"
- Betreuer steht euch stets mit Rat und Tat zur Seite

BWT

Burrows–Wheeler–Transformation findet Einsatz in

- Sequenzanalyse
  - Textindizierung (z.B. per FM-Index)
  - Sequencing
  - Konstruktion anderer Strukturen (z.B. Suffixarray, LCP–Array, ...)
- Datenkompression
  - block–sorting compression
  - Einsatz in vielen Kompressionstools (z.B. `bzip2`)

BWT Definition

Gegeben: Text $S = \texttt{ctaataatg\$}$

Suffixe von S:

i	$S[i-1]$	S_i
0	$	ctaataatg$
1	c	taataatg$
2	t	aataatg$
3	a	ataatg$
4	a	taatg$
5	t	aatg$
6	a	atg$
7	a	tg$
8	t	g$
9	g	$

BWT Definition

Gegeben: Text $S = \texttt{ctaataatg\$}$

Suffixe von S:

i	$S[i-1]$	S_i
0	$	ctaataatg$
1	c	taataatg$
2	t	aataatg$
3	a	ataatg$
4	a	taatg$
5	t	aatg$
6	a	atg$
7	a	tg$
8	t	g$
9	g	$

lexikographisch sortiert:

i	SA[i]	$S[\text{SA}[i]-1]$	$S_{\text{SA}[i]}$
0	9	g	$
1	2	t	aataatg$
2	5	t	aatg$
3	3	a	ataatg$
4	6	a	atg$
5	0	$	ctaataatg$
6	8	t	g$
7	1	c	taataatg$
8	4	a	taatg$
9	7	a	tg$

BWT Definition

Gegeben: Text $S =$ `ctaataatg$`

Suffixe von S:

i	$S[i-1]$	S_i
0	$	ctaataatg$
1	c	taataatg$
2	t	aataatg$
3	a	ataatg$
4	a	taatg$
5	t	aatg$
6	a	atg$
7	a	tg$
8	t	g$
9	g	$

lexikographisch sortiert:

i	SA[i]	$S[\text{SA}[i]-1]$	$S_{\text{SA}[i]}$
0	9	g	$
1	2	t	aataatg$
2	5	t	aatg$
3	3	a	ataatg$
4	6	a	atg$
5	0	$	ctaataatg$
6	8	t	g$
7	1	c	taataatg$
8	4	a	taatg$
9	7	a	tg$

- Suffixarray (SA): Startpositionen der Suffixe von S in lexikographisch sortierter Reihenfolge
- Burrows – Wheeler – Transformation (BWT): zyklisch vorheriger Buchstabe eines Suffixes in SA, $\text{BWT}[i] = S[\text{SA}[i] - 1 \mod n]$

BWT Konstruktion

Konstruktion der BWT

- per Suffixarray (BWT$[i] = S[\text{SA}[i] - 1 \mod n]$)
  - \+ schnelle Konstruktion, lineare Laufzeit
  - − Speicherplatz für Suffixarray
- per partiellem Suffixarray
  - \+ schnelle Konstruktion, lineare Laufzeit
  - − Speicherplatz
- direkt
  - \+ inplace Konstruktion möglich
  - − asymptotisch quadratische Laufzeit
- hybride Verfahren

  $\Rightarrow$ Trade-Off zwischen Speicherplatz und Laufzeit möglich

BWT Grundlagen

BWT: Permutation der Buchstaben des Originaltexts S
Frage: Kann S aus BWT wiederhergestellt werden?

BWT Rückwärtsschritt

Sei $i \in [0, n)$ ein Index, $c \in \Sigma$ ein Buchstabe aus der BWT, C ein Array und $rank_{\text{BWT}}(c, i)$ eine Funktion, die wie folgt definiert sind:

$$C[c] := |\{\, j \in [0, n) \mid \text{BWT}[j] < c \,\}|$$
$$rank_{\text{BWT}}(c, i) := |\{\, j \in [0, i\,) \mid \text{BWT}[j] = c \,\}|$$

Seien weiter $j, k \in [0, n)$ zwei Indizes mit $\text{SA}[j] = \text{SA}[k] + 1$.
Man kann zeigen:

$$j = C[\text{BWT}[k]] + rank_{\text{BWT}}(\text{BWT}[k], k)$$

$\Rightarrow$ mit BWT kann der Originaltext rückwärts durchlaufen werden

Inverse BW – Transformation

Berechnung von S aus BWT

1: compute the C array from BWT
2: compute the $rank_{\mathrm{BWT}}$ - function
3: $S[n-1] \leftarrow \$$
4: $j \leftarrow 0$
5: **for** $i \leftarrow n-2$ **down to** 0 **do**
6: $\quad c \leftarrow \mathrm{BWT}[j]$
7: $\quad j \leftarrow C[c] + rank_{\mathrm{BWT}}(c, j)$
8: $\quad S[i] \leftarrow c$
9: **end for**
10: output S

Inverse BW – Transformation

Berechnung von S aus BWT

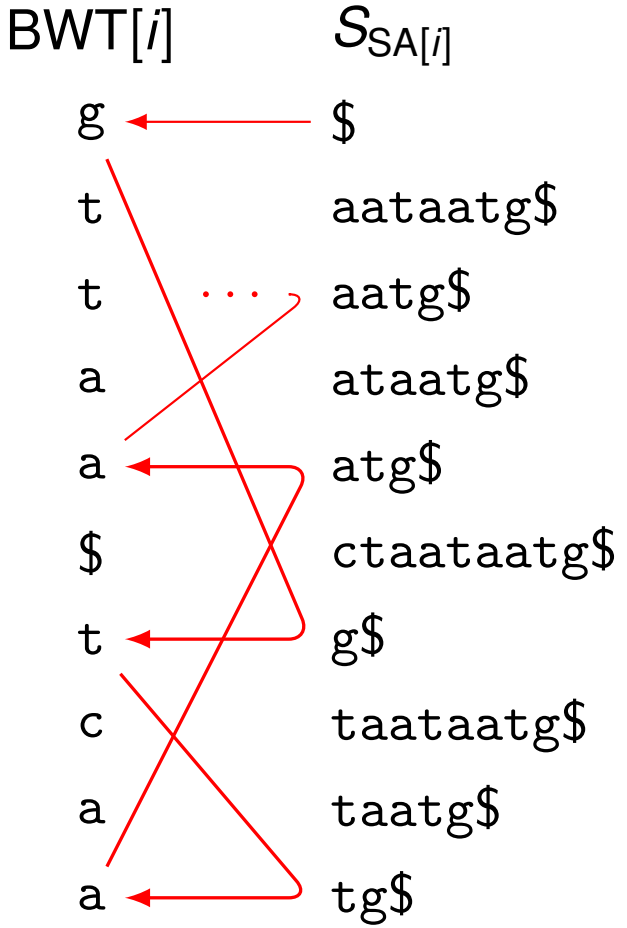
1: compute the C array from BWT
2: compute the $rank_{BWT}$ - function
3: $S[n-1] \leftarrow \$$
4: $j \leftarrow 0$
5: **for** $i \leftarrow n-2$ **down to** 0 **do**
6: $\quad c \leftarrow \text{BWT}[j]$
7: $\quad j \leftarrow C[c] + rank_{BWT}(c, j)$
8: $\quad S[i] \leftarrow c$
9: **end for**
10: output S

BWT[i]	$S_{SA[i]}$
g	$
t	aataatg$
t	aatg$
a	ataatg$
a	atg$
$	ctaataatg$
t	g$
c	taataatg$
a	taatg$
a	tg$

Inverse BW – Transformation

Berechnung von S aus BWT

1: compute the C array from BWT
2: compute the $rank_{BWT}$ - function
3: $S[n-1] \leftarrow \$$
4: $j \leftarrow 0$
5: **for** $i \leftarrow n-2$ **down to** 0 **do**
6: $\quad c \leftarrow BWT[j]$
7: $\quad j \leftarrow C[c] + rank_{BWT}(c, j)$
8: $\quad S[i] \leftarrow c$
9: **end for**
10: output S



BWT inplace

1. Generiere BWT der letzten beiden Suffixe des Textes
2. Laufe von rechts nach links, erweitere BWT vom letzten Schritt um neuen Buchstaben
   $\Rightarrow$ BWT kann am Ende des Textes gespeichert werden

Position des neuen Buchstabens

- Ersetze zyklisch letzten Buchstaben ($) durch neuen vordersten Buchstaben
- Neue Position des zyklisch letzten Buchstabens $ kann per Rückwärtssuche bestimmt werden

BWT inplace – Beispiel

$S =$ `ctaataat`g$

i	BWT[i]	S[SA[i]]
0	g	$
1	$	g

BWT inplace – Beispiel

$S =$ ctaataatg$

	i	BWT[i]	S[SA[i]]
	0	g	$
$\Rightarrow$	1	~~$~~ t	g

BWT inplace – Beispiel

$S =$ ctaataatg$

	i	BWT[i]	S[SA[i]]
	0	g	$
$\Rightarrow$	1	~~$~~ t	g
		$	t

Position des neuen Eintrags

$$C[\mathtt{t}] + rank_{\mathrm{BWT}}(\mathtt{t}, 1) = 2$$

BWT inplace – Beispiel

$S =$ ctaataatg$

Position des neuen Eintrags

$C[\mathrm{a}] + \mathit{rank}_{\mathrm{BWT}}(\mathrm{a}, 2) = 1$

	i	BWT[i]	S[SA[i]]
	0	g	$
		$	a
	1	t	g
$\Rightarrow$	2	~~$~~ a	t

BWT inplace – Beispiel

$S =$ ctaataatg$

Position des neuen Eintrags

$C[\mathtt{a}] + rank_{\mathrm{BWT}}(\mathtt{a}, 1) = 1$

	i	BWT[i]	S[SA[i]]
	0	g	$
		$	a
$\Rightarrow$	1	~~$~~ a	a
	2	t	g
	3	a	t

BWT inplace – Beispiel

$S = \texttt{ctaataatg\$}$

Position des neuen Eintrags

$C[\texttt{t}] + rank_{\text{BWT}}(\texttt{t}, 1) = 4$

	i	BWT[i]	S[SA[i]]
	0	g	$
$\Rightarrow$	1	~~$~~ t	a
	2	a	a
	3	t	g
		$	t
	4	a	t

BWT inplace – Beispiel

$S =$ ctaataatg$

Position des neuen Eintrags

$C[\mathtt{a}] + rank_{\mathrm{BWT}}(\mathtt{a}, 4) = 2$

	i	BWT[i]	S[SA[i]]
	0	g	$
	1	t	a
		$	a
	2	a	a
	3	t	g
$\Rightarrow$	4	~~$~~ a	t
	5	a	t

BWT inplace – Beispiel

$S =$ ctaataatg$

Position des neuen Eintrags

$C[\mathtt{a}] + rank_{\mathrm{BWT}}(\mathtt{a}, 2) = 1$

	i	BWT[i]	S[SA[i]]
	0	g	$
		$	a
	1	t	a
$\Rightarrow$	2	~~$~~ a	a
	3	a	a
	4	t	g
	5	a	t
	6	a	t

BWT inplace – Beispiel

$S = \texttt{ctaataatg\$}$

Position des neuen Eintrags

$C[\texttt{t}] + rank_{\text{BWT}}(\texttt{t}, 1) = 6$

	i	BWT[i]	S[SA[i]]
	0	g	$
$\Rightarrow$	1	~~$~~ t	a
	2	t	a
	3	a	a
	4	a	a
	5	t	g
		$	t
	6	a	t
	7	a	t

BWT inplace – Beispiel

$S = \texttt{ctaataatg\$}$

Position des neuen Eintrags

$C[\texttt{c}] + rank_{\text{BWT}}(\texttt{c}, 6) = 5$

	i	BWT[i]	S[SA[i]]
	0	g	$
	1	t	a
	2	t	a
	3	a	a
	4	a	a
		$	c
	5	t	g
$\Rightarrow$	6	~~$~~ c	t
	7	a	t
	8	a	t

BWT inplace – Beispiel

$S =$ `ctaataatg$`

i	BWT[i]	S[SA[i]]
0	g	$
1	t	a
2	t	a
3	a	a
4	a	a
5	$	c
6	t	g
7	c	t
8	a	t
9	a	t

BWT Konstruktion – Ziele

Zu implementierende Konstruktionsalgorithmen:

- BWT inplace
- BWT inplace mit zusätzlichem Speicherplatz
- BWT mittels anderem Ansatz (ebenfalls Zeichen einfügen)

Vergleich mit

- konventioneller Konstruktion per Suffixarray
- Konstruktion per partiellem Suffixarray
- sonstigen hybriden Verfahren (z.B. `SAScan`)

Fokus auf Time–Space–Tradeoff

Abschließende Anmerkungen

- Hilfestellung wird wie benötigt gegeben
- Experimentelle Resultate in Eigenregie oder per vorhandener Benchmarks
- Falls Wunschthema gefunden: Mail an `uwe.baier@uni-ulm.de`, mit Teampartnern und deren E–mail – Adressen
- Auch eigene Themenvorschläge per Mail an `uwe.baier@uni-ulm.de`