Karlsruher Institut für Technologie

Burrows Wheeler Transformation – Algorithmen II

Simon Gog – *gog@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS17.php`

Institut für Theoretische Informatik - Algorithmik II

Burrows-Wheeler-Transformation

Einführung

Karlsruher Institut für Technologie

Burrows, Wheeler (1983, 1994)

- "nur" eine Umordnung des Textes,
  $\rightarrow$ gruppiert Zeichen mit ähnlichem Kontext, reversibel
- ursprünglich eingeführt zur Textkompression
  $\rightarrow$ Teilschritt von bzip2
- später auch für Textindizierung verwendet
  $\rightarrow$ z.B. Rückwärtssuche, Berechnung des LCP-Array

Burrows-Wheeler-Transformation

Wiederholung: Suffix-Arrays (und Suffixbäume)

- Textindizierung
  $\rightarrow$ schnelle Suche in Texten
- kann in $O(n)$ erzeugt werden
  (DC3-Algorithmus)

	1	2	3	4	5	6	7	8	9
$T =$	l	a	l	a	n	g	n	g	$

Definition

- $\mathrm{SA}[i] \mathrel{\hat{=}}$ Index des i-ten sortierten Suffix
  (einer Zeichenkette T)

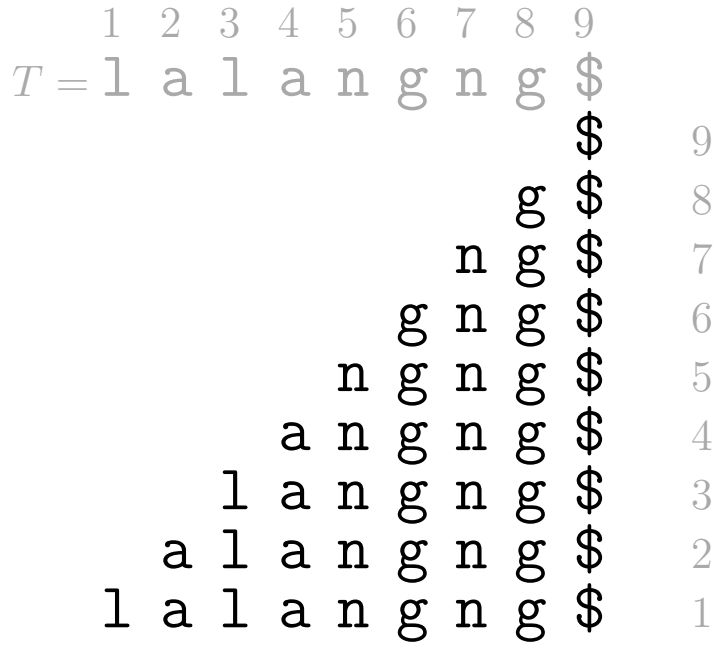
Burrows-Wheeler-Transformation

Wiederholung: Suffix-Arrays (und Suffixbäume)

- Textindizierung
  $\rightarrow$ schnelle Suche in Texten
- kann in $O(n)$ erzeugt werden
  (DC3-Algorithmus)

Definition

- $\mathrm{SA}[i] \mathrel{\hat{=}}$ Index des i-ten sortierten Suffix
  (einer Zeichenkette T)



Burrows-Wheeler-Transformation

Wiederholung: Suffix-Arrays (und Suffixbäume)

- Textindizierung
  $\rightarrow$ schnelle Suche in Texten
- kann in $O(n)$ erzeugt werden
  (DC3-Algorithmus)

Definition

- $\mathrm{SA}[i] \mathrel{\hat{=}}$ Index des i-ten sortierten Suffix
  (einer Zeichenkette T)

	1	2	3	4	5	6	7	8	9	
$T =$	l	a	l	a	n	g	n	g	$	
									$	9
		a	l	a	n	g	n	g	$	2
			a	n	g	n	g	$		4
							g	$		8
						g	n	g	$	6
	l	a	l	a	n	g	n	g	$	1
			l	a	n	g	n	g	$	3
							n	g	$	7
					n	g	n	g	$	5

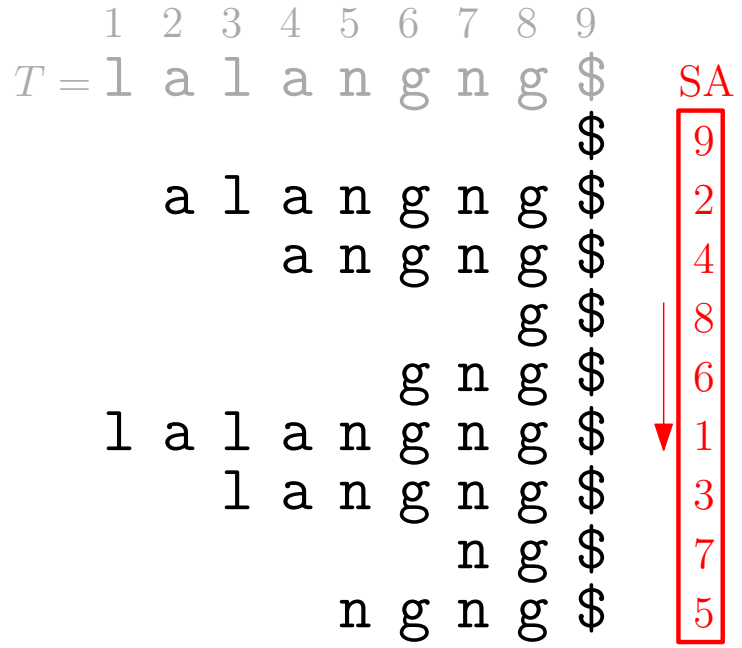
Burrows-Wheeler-Transformation

Wiederholung: Suffix-Arrays (und Suffixbäume)

- Textindizierung
  $\rightarrow$ schnelle Suche in Texten
- kann in $O(n)$ erzeugt werden
  (DC3-Algorithmus)

Definition

- $\mathrm{SA}[i] \mathrel{\hat{=}}$ Index des i-ten sortierten Suffix
  (einer Zeichenkette T)



Burrows-Wheeler-Transformation

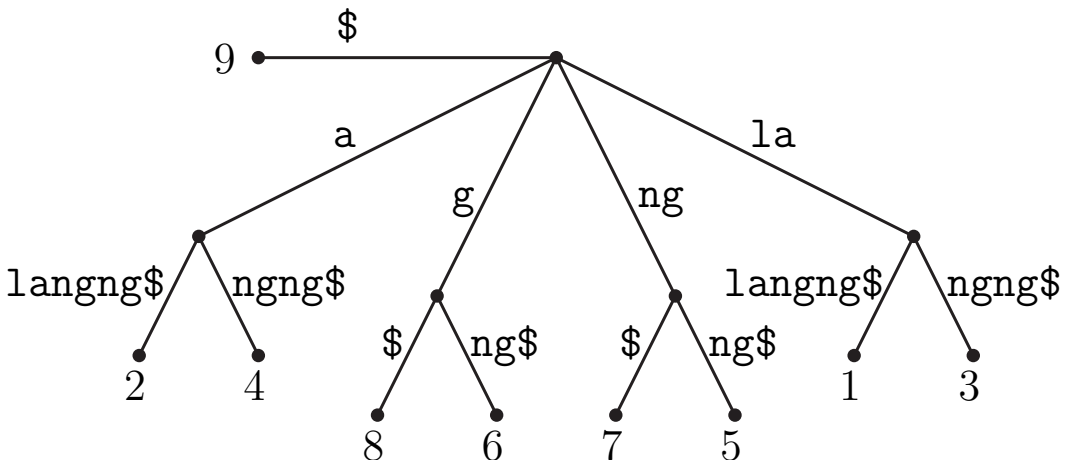
Wiederholung: Suffix-Arrays (und Suffixbäume)

- Textindizierung
  $\rightarrow$ schnelle Suche in Texten
- kann in $O(n)$ erzeugt werden
  (DC3-Algorithmus)

	1	2	3	4	5	6	7	8	9	
$T =$	l	a	l	a	n	g	n	g	$	SA
									$	9
		a	l	a	n	g	n	g	$	2
			a	n	g	n	g	$		4
							g	$		8
					g	n	g	$		6
	l	a	l	a	n	g	n	g	$	1
		l	a	n	g	n	g	$		3
						n	g	$		7
				n	g	n	g	$		5

Definition

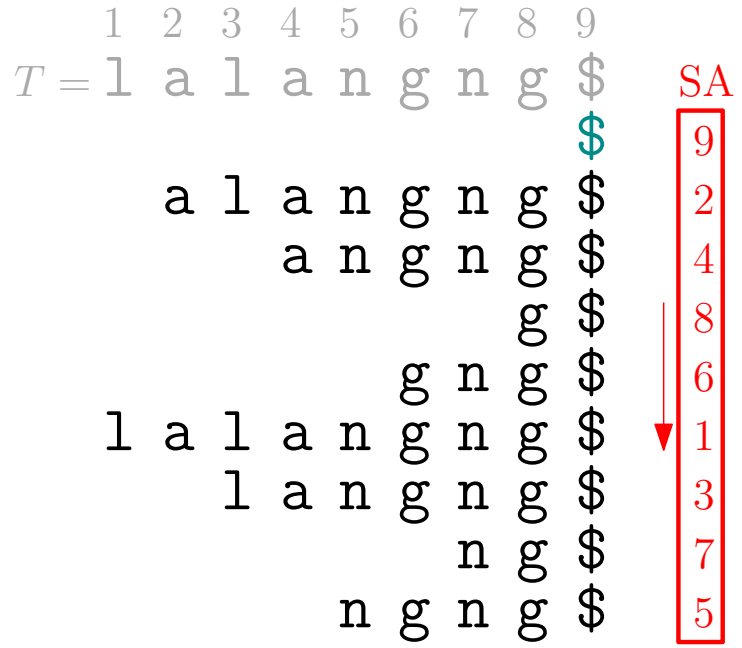
- $\mathrm{SA}[i] \triangleq$ Index des i-ten sortierten Suffix
  (einer Zeichenkette T)



Burrows-Wheeler-Transformation

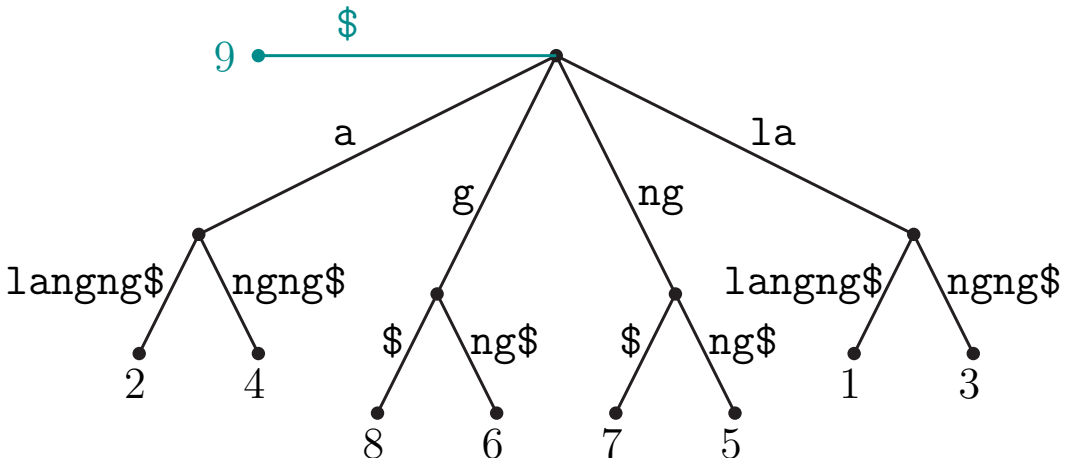
Wiederholung: Suffix-Arrays (und Suffixbäume)

- Textindizierung
  $\rightarrow$ schnelle Suche in Texten
- kann in $O(n)$ erzeugt werden
  (DC3-Algorithmus)



Definition

- SA$[i] \triangleq$ Index des i-ten sortierten Suffix
  (einer Zeichenkette T)



Burrows-Wheeler-Transformation

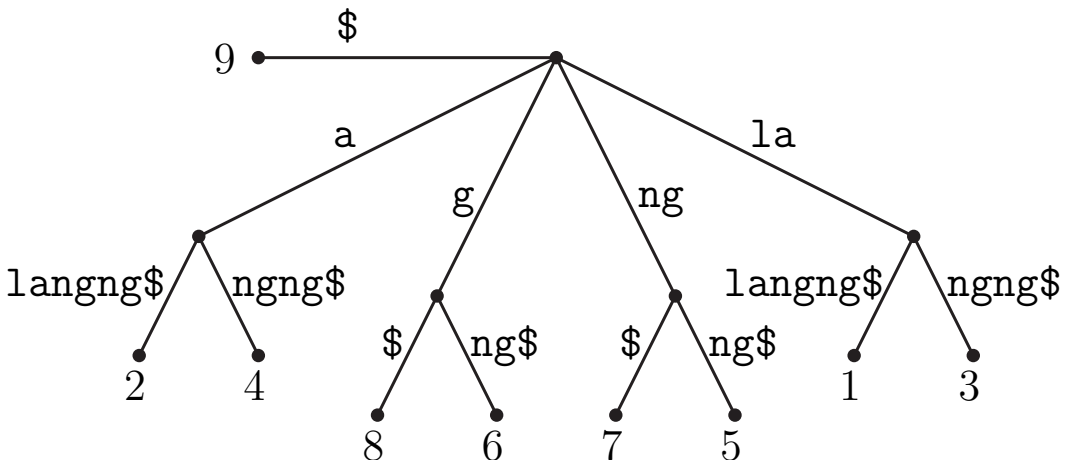
Wiederholung: Suffix-Arrays (und Suffixbäume)

- Textindizierung
  $\rightarrow$ schnelle Suche in Texten
- kann in $O(n)$ erzeugt werden
  (DC3-Algorithmus)

	1	2	3	4	5	6	7	8	9	
$T =$	l	a	l	a	n	g	n	g	$	SA
									$	9
				a	l	a	n	g	n	g	$	2
					a	n	g	n	g	$	4
								g	$	8
							g	n	g	$	6
	l	a	l	a	n	g	n	g	$	1
			l	a	n	g	n	g	$	3
							n	g	$	7
					n	g	n	g	$	5

Definition

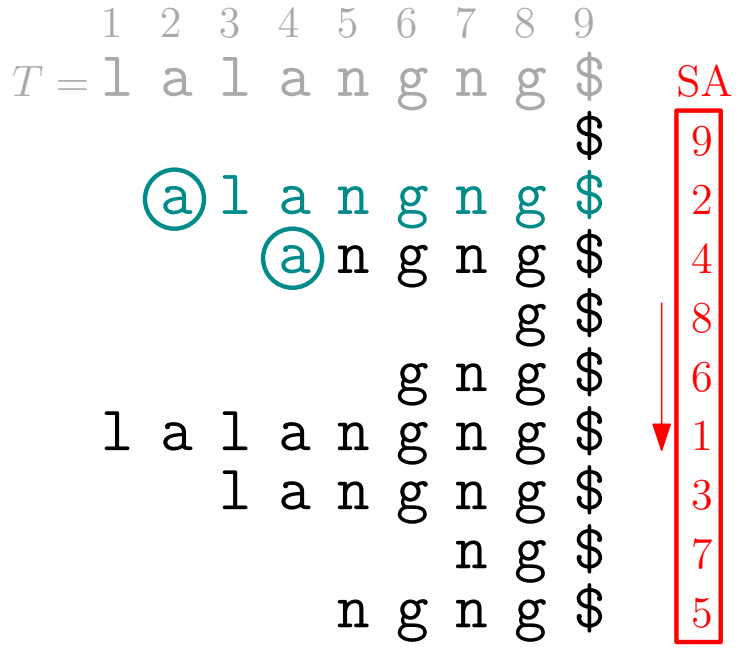
- $\mathrm{SA}[i] \triangleq$ Index des i-ten sortierten Suffix
  (einer Zeichenkette T)



Burrows-Wheeler-Transformation

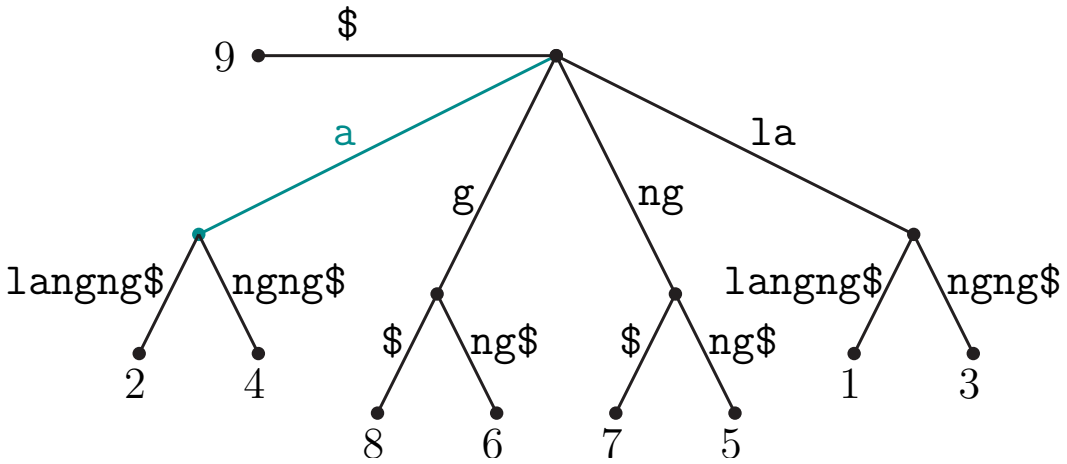
Wiederholung: Suffix-Arrays (und Suffixbäume)

- Textindizierung
  $\rightarrow$ schnelle Suche in Texten
- kann in $O(n)$ erzeugt werden
  (DC3-Algorithmus)



Definition

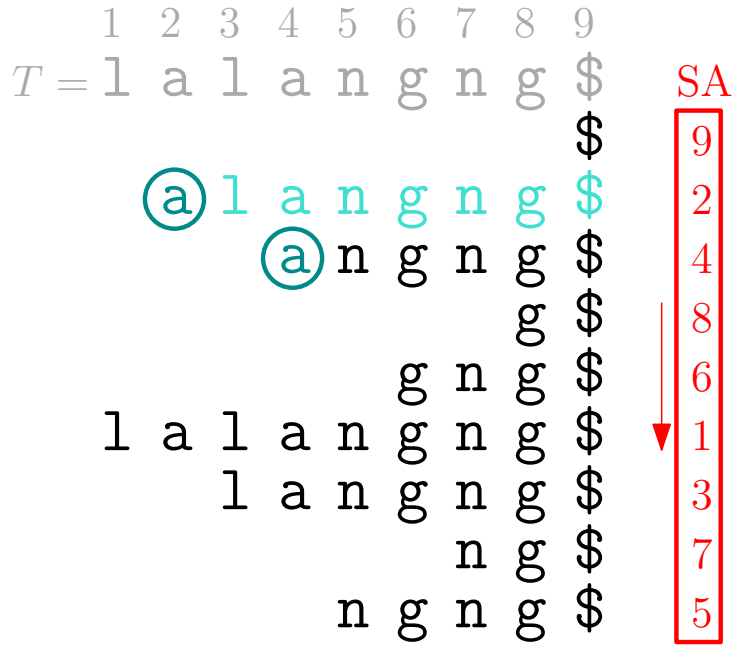
- SA$[i] \triangleq$ Index des i-ten sortierten Suffix
  (einer Zeichenkette T)



Burrows-Wheeler-Transformation

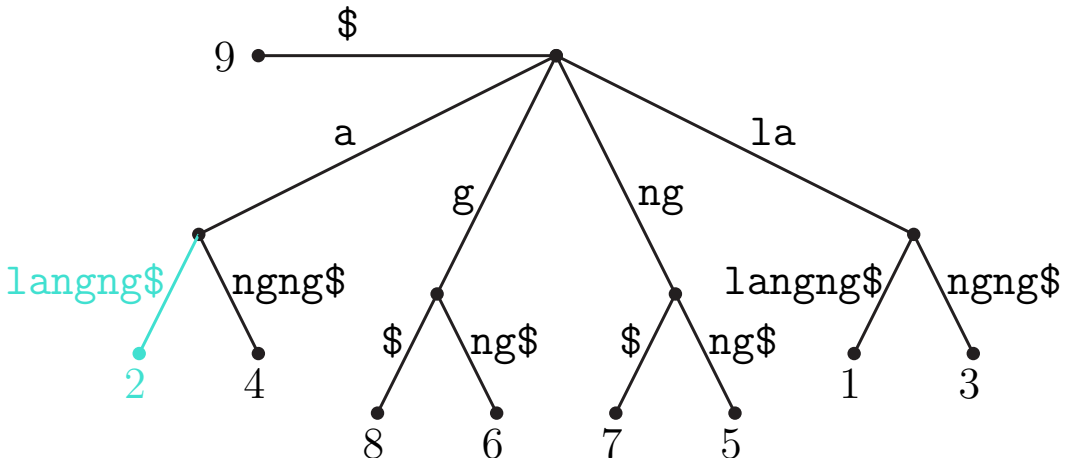
Wiederholung: Suffix-Arrays (und Suffixbäume)

- Textindizierung
  $\rightarrow$ schnelle Suche in Texten
- kann in $O(n)$ erzeugt werden
  (DC3-Algorithmus)



Definition

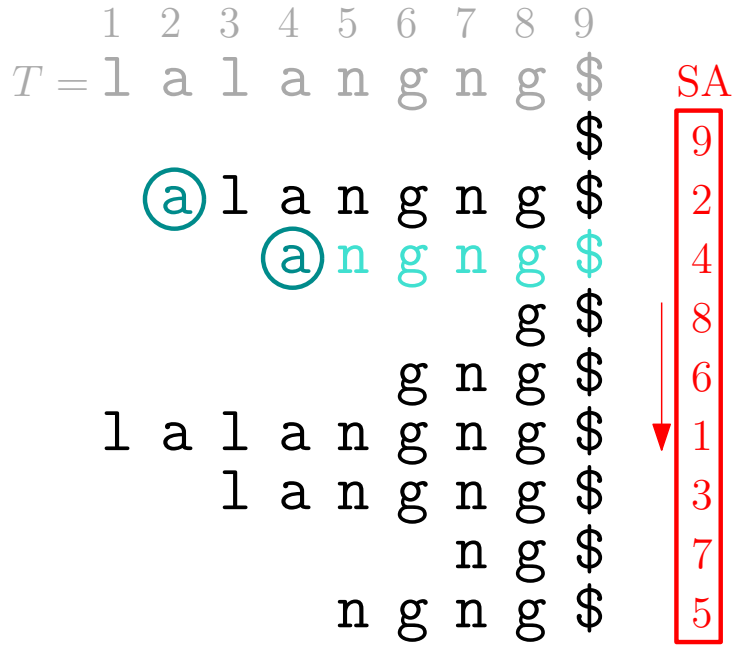
- SA$[i] \triangleq$ Index des i-ten sortierten Suffix
  (einer Zeichenkette T)



Burrows-Wheeler-Transformation

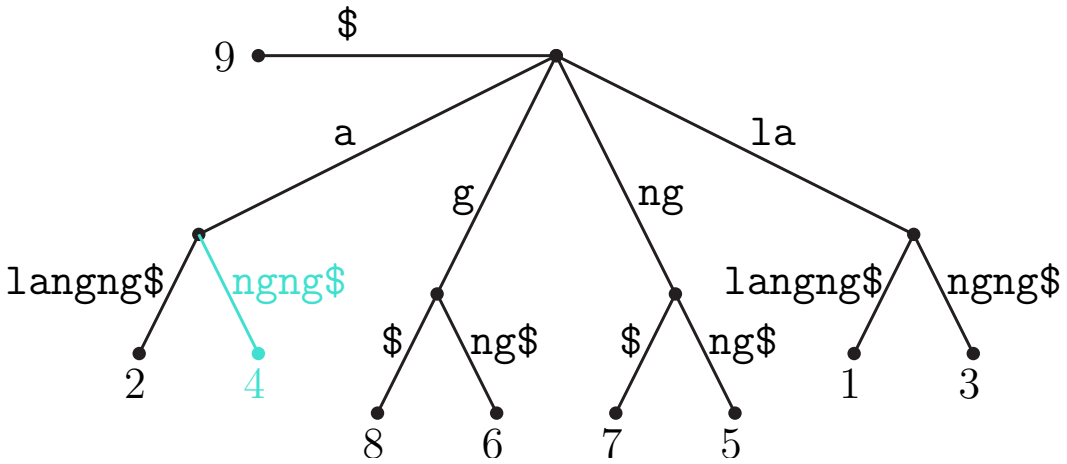
Wiederholung: Suffix-Arrays (und Suffixbäume)

- Textindizierung
  $\rightarrow$ schnelle Suche in Texten
- kann in $O(n)$ erzeugt werden
  (DC3-Algorithmus)



Definition

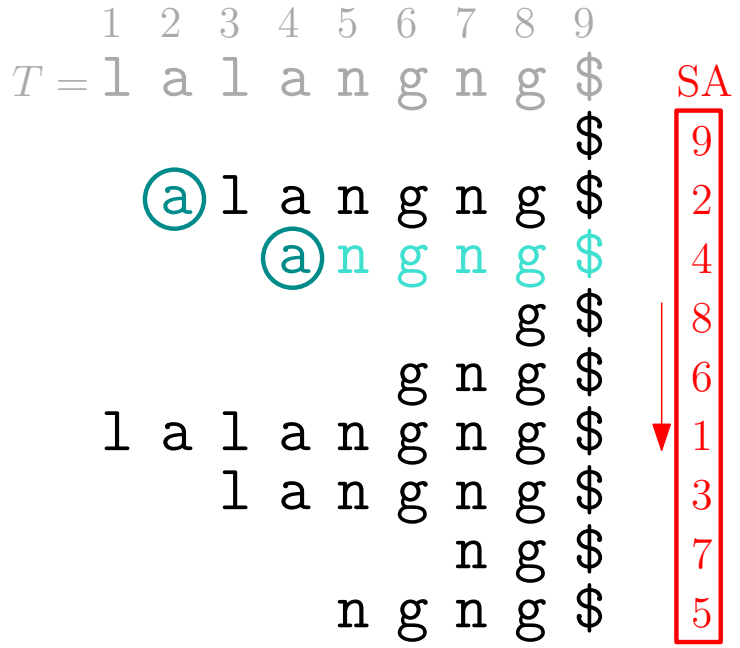
- SA$[i] \triangleq$ Index des i-ten sortierten Suffix
  (einer Zeichenkette T)



Burrows-Wheeler-Transformation

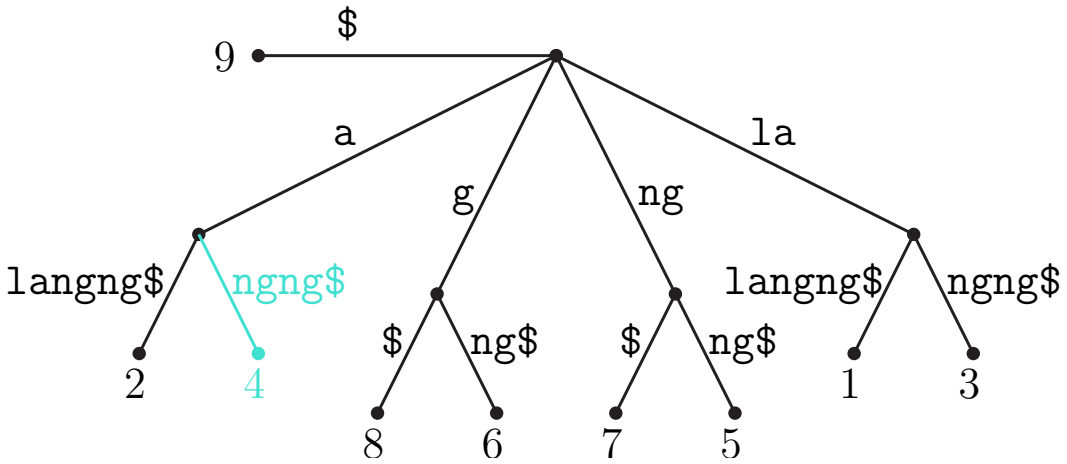
Wiederholung: Suffix-Arrays (und Suffixbäume)

- Textindizierung
  $\rightarrow$ schnelle Suche in Texten
- kann in $O(n)$ erzeugt werden
  (DC3-Algorithmus)



Definition

- SA$[i] \triangleq$ Index des i-ten sortierten Suffix
  (einer Zeichenkette T)



USW.

Burrows-Wheeler-Transformation

Transformation

	1	2	3	4	5	6	7	8	9
$T =$	l	a	l	a	n	g	n	g	$

Burrows-Wheeler-Transformation

Transformation

	1	2	3	4	5	6	7	8	9
$T =$	l	a	l	a	n	g	n	g	$
	a	l	a	n	g	n	g	$	l

Burrows-Wheeler-Transformation

Transformation

	1	2	3	4	5	6	7	8	9
$T =$	l	a	l	a	n	g	n	g	$
	a	l	a	n	g	n	g	$	l
	l	a	n	g	n	g	$	l	a

Burrows-Wheeler-Transformation

Transformation

	1	2	3	4	5	6	7	8	9
$T =$	l	a	l	a	n	g	n	g	$
	a	l	a	n	g	n	g	$	l
	l	a	n	g	n	g	$	l	a
	a	n	g	n	g	$	l	a	l

Burrows-Wheeler-Transformation

Transformation

$$
T = \begin{array}{ccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\mathtt{l} & \mathtt{a} & \mathtt{l} & \mathtt{a} & \mathtt{n} & \mathtt{g} & \mathtt{n} & \mathtt{g} & \mathtt{\$} \\
\mathtt{a} & \mathtt{l} & \mathtt{a} & \mathtt{n} & \mathtt{g} & \mathtt{n} & \mathtt{g} & \mathtt{\$} & \mathtt{l} \\
\mathtt{l} & \mathtt{a} & \mathtt{n} & \mathtt{g} & \mathtt{n} & \mathtt{g} & \mathtt{\$} & \mathtt{l} & \mathtt{a} \\
\mathtt{a} & \mathtt{n} & \mathtt{g} & \mathtt{n} & \mathtt{g} & \mathtt{\$} & \mathtt{l} & \mathtt{a} & \mathtt{l} \\
 & & & & \vdots & & & &
\end{array}
$$

Burrows-Wheeler-Transformation

Transformation

	1	2	3	4	5	6	7	8	9
$T =$	l	a	l	a	n	g	n	g	$
	a	l	a	n	g	n	g	$	l
	l	a	n	g	n	g	$	l	a
	a	n	g	n	g	$	l	a	l
	n	g	n	g	$	l	a	l	a
	g	n	g	$	l	a	l	a	n
	n	g	$	l	a	l	a	n	g
	g	$	l	a	l	a	n	g	n
	$	l	a	l	a	n	g	n	g

Burrows-Wheeler-Transformation

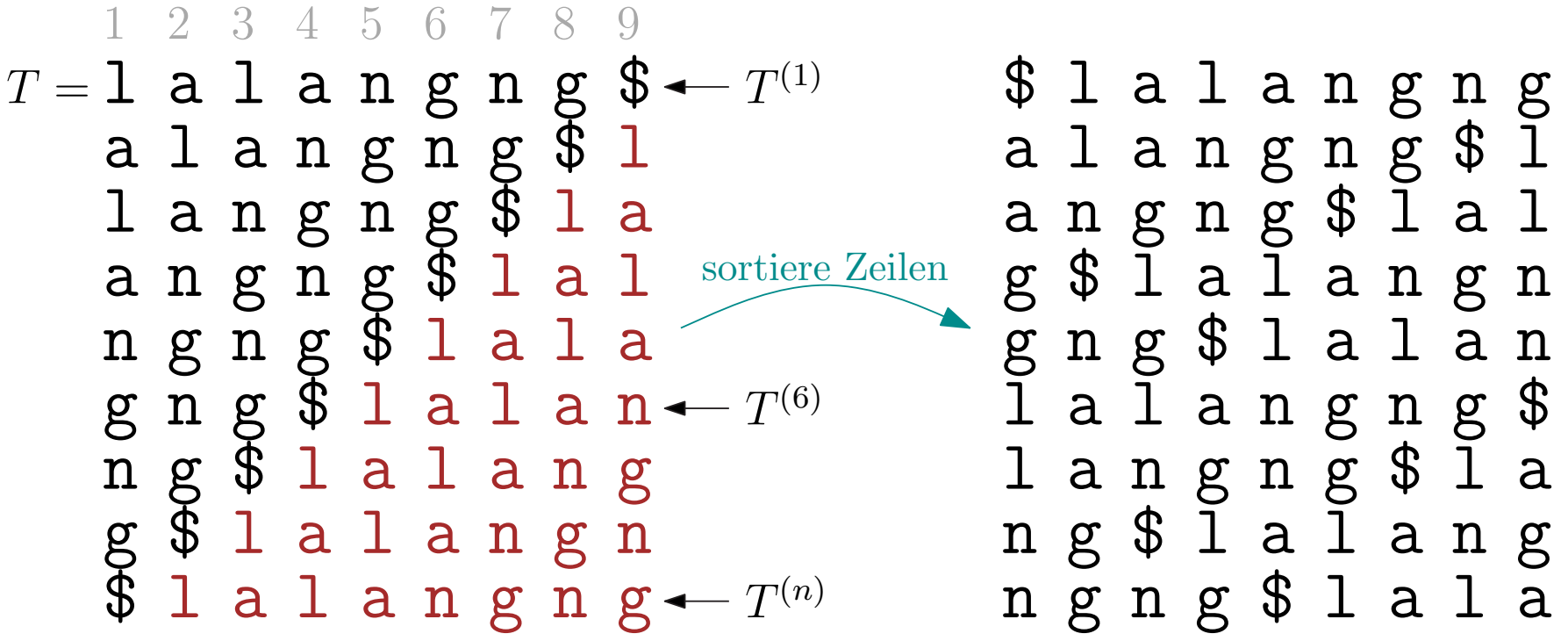
Transformation

	1	2	3	4	5	6	7	8	9	
$T =$	l	a	l	a	n	g	n	g	$	$\leftarrow T^{(1)}$
	a	l	a	n	g	n	g	$	l	
	l	a	n	g	n	g	$	l	a	
	a	n	g	n	g	$	l	a	l	
	n	g	n	g	$	l	a	l	a	
	g	n	g	$	l	a	l	a	n	$\leftarrow T^{(6)}$
	n	g	$	l	a	l	a	n	g	
	g	$	l	a	l	a	n	g	n	
	$	l	a	l	a	n	g	n	g	$\leftarrow T^{(n)}$

- $T^{(i)} \triangleq T$ zyklisch ab Position i, Länge $n = |T|$

Burrows-Wheeler-Transformation

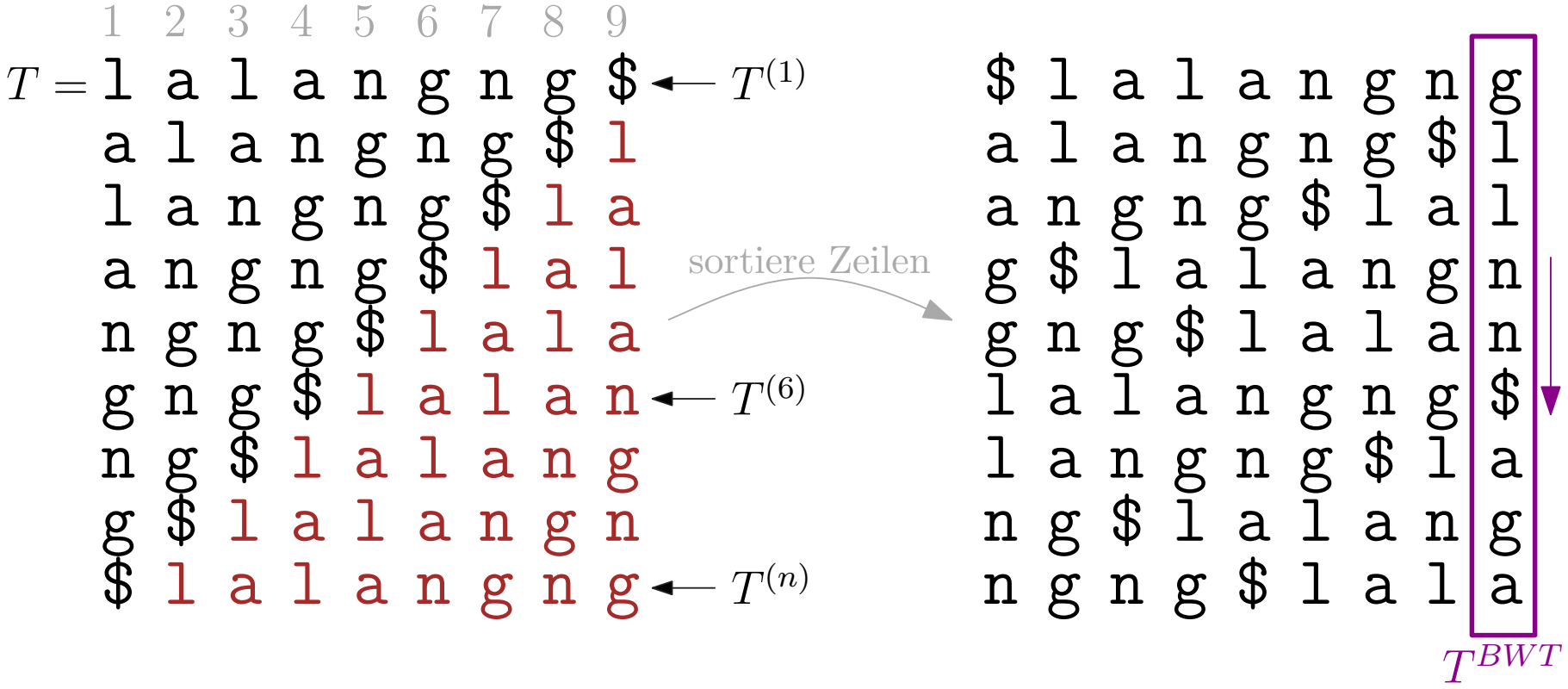
Transformation



- $T^{(i)} \triangleq T$ zyklisch ab Position i, Länge $n = |T|$

Burrows-Wheeler-Transformation

Transformation

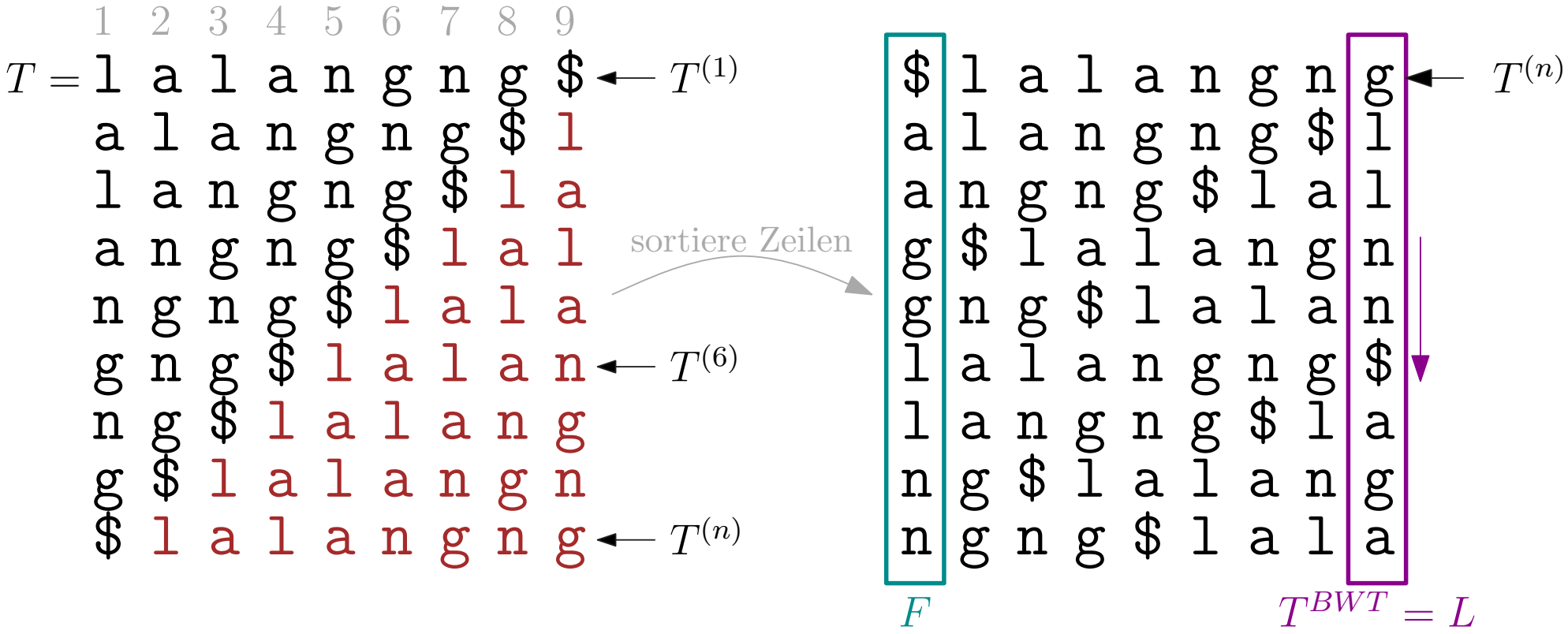


- $T^{(i)} \triangleq T$ zyklisch ab Position i, Länge $n = |T|$
- $T = \texttt{lalangng\$} \quad \rightarrow \quad T^{BWT} = \texttt{gllnn\$aga}$

$\mathcal{O}(n^2 + n \log n)$

Burrows-Wheeler-Transformation

Transformation



- $T^{(i)} \triangleq T$ zyklisch ab Position i, Länge $n = |T|$
- $T = \texttt{lalangng\$} \quad \rightarrow \quad T^{BWT} = \texttt{gllnn\$aga}$

$\mathcal{O}(n^2 + n \log n)$

Burrows-Wheeler-Transformation

Eigenschaften

	1	2	3	4	5	6	7	8	9
$T =$	l	a	l	a	n	g	n	g	$

- BWT in $\mathcal{O}(n)$ berechenbar
- Zeilen enthalten sortierte Suffixe
- Zeichen in letzer Spalte entspricht Zeichen vor Suffix in T

$	l	a	l	a	n	g	n	g	1
a	l	a	n	g	n	g	$	l	2
a	n	g	n	g	$	l	a	l	3
g	$	l	a	l	a	n	g	n	4
g	n	g	$	l	a	l	a	n	5
l	a	l	a	n	g	n	g	$	6
l	a	n	g	n	g	$	l	a	7
n	g	$	l	a	l	a	n	g	8
n	g	n	g	$	l	a	l	a	9

Burrows-Wheeler-Transformation

Eigenschaften

	1	2	3	4	5	6	7	8	9
$T =$	l	a	l	a	n	g	n	g	$

- BWT in $\mathcal{O}(n)$ berechenbar
- Zeilen enthalten sortierte Suffixe
- Zeichen in letzer Spalte entspricht Zeichen vor Suffix in T

$	l	a	l	a	n	g	n	g	1
a	l	a	n	g	n	g	$	l	2
a	n	g	n	g	$	l	a	l	3
g	$	l	a	l	a	n	g	n	4
g	n	g	$	l	a	l	a	n	5
l	a	l	a	n	g	n	g	$	6
l	a	n	g	n	g	$	l	a	7
n	g	$	l	a	l	a	n	g	8
n	g	n	g	$	l	a	l	a	9

Burrows-Wheeler-Transformation

Eigenschaften

	1	2	3	4	5	6	7	8	9
$T =$	l	a	l	a	n	g	n	g	$

- BWT in $\mathcal{O}(n)$ berechenbar
- Zeilen enthalten sortierte Suffixe
- Zeichen in letzer Spalte entspricht Zeichen vor Suffix in T

$	l	a	l	a	n	g	n	g	1
a	l	a	n	g	n	g	$	l	2
a	n	g	n	g	$	l	a	l	3
g	$	l	a	l	a	n	g	n	4
g	n	g	$	l	a	l	a	n	5
l	a	l	a	n	g	n	g	$	6
l	a	n	g	n	g	$	l	a	7
n	g	$	l	a	l	a	n	g	8
n	g	n	g	$	l	a	l	a	9

Burrows-Wheeler-Transformation

Eigenschaften

	1	2	3	4	5	6	7	8	9
$T =$	l	a	l	a	n	g	n	g	$

- BWT in $\mathcal{O}(n)$ berechenbar
- Zeilen enthalten sortierte Suffixe
- Zeichen in letzer Spalte entspricht Zeichen vor Suffix in T

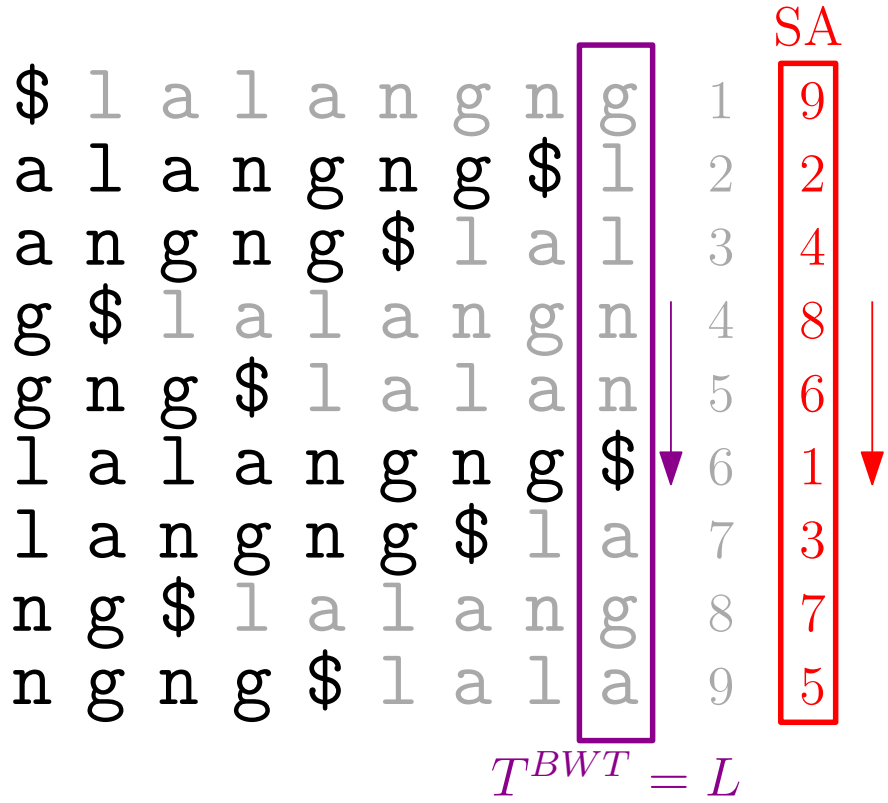
$	l	a	l	a	n	g	n	g	1
a	l	a	n	g	n	g	$	l	2
a	n	g	n	g	$	l	a	l	3
g	$	l	a	l	a	n	g	n	4
g	n	g	$	l	a	l	a	n	5
l	a	l	a	n	g	n	g	$	6
l	a	n	g	n	g	$	l	a	7
n	g	$	l	a	l	a	n	g	8
n	g	n	g	$	l	a	l	a	9

$T^{BWT} = L$

Burrows-Wheeler-Transformation

Eigenschaften

	1	2	3	4	5	6	7	8	9
$T =$	l	a	l	a	n	g	n	g	$



- BWT in $\mathcal{O}(n)$ berechenbar
- Zeilen enthalten sortierte Suffixe
- Zeichen in letzer Spalte entspricht Zeichen vor Suffix in T
- $T^{BWT}[i] = L[i] = T[SA[i] - 1] = T^{(SA[i])}[n]$
  ($T^{BWT}[i]$ ist das Zeichen vor dem i-ten Suffix in T)

Burrows-Wheeler-Transformation

Rücktransformation – Vorüberlegungen

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

- betrachte Matrix aus Transformation
  - erste Zeile enthält Lösung $T^{(n)}$
  - $T^{BWT} = L$ gegeben
  - F leicht zu bestimmen (sortiere L)
  - L ist *immer* Spalte vor F (zyklisch)

\$	l	a	l	a	n	g	n	g
a	l	a	n	g	n	g	\$	l
a	n	g	n	g	\$	l	a	l
g	\$	l	a	l	a	n	g	n
g	n	g	\$	l	a	l	a	n
l	a	l	a	n	g	n	g	\$
l	a	n	g	n	g	\$	l	a
n	g	\$	l	a	l	a	n	g
n	g	n	g	\$	l	a	l	a

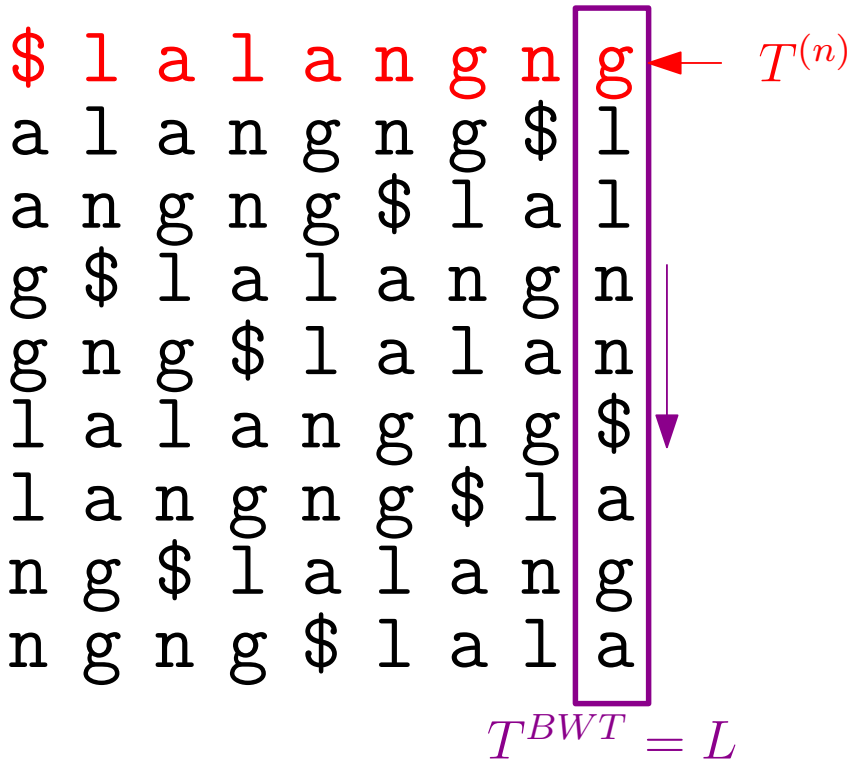
$T^{BWT} = L$

Burrows-Wheeler-Transformation

Rücktransformation – Vorüberlegungen

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

- betrachte Matrix aus Transformation
  - erste Zeile enthält Lösung $T^{(n)}$
  - $T^{BWT} = L$ gegeben
  - F leicht zu bestimmen (sortiere L)
  - L ist *immer* Spalte vor F (zyklisch)



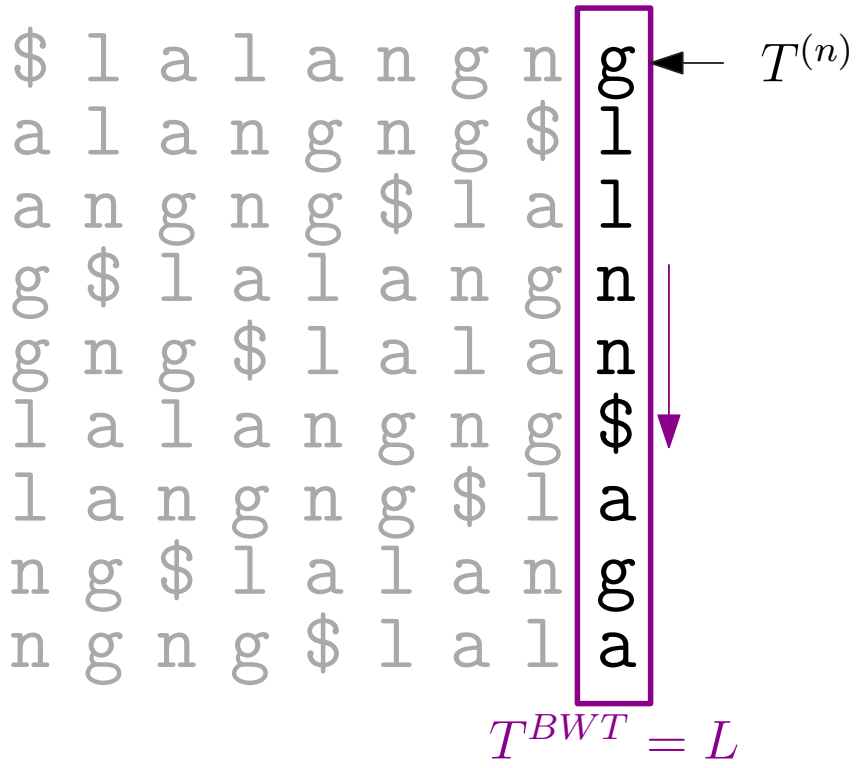
Burrows-Wheeler-Transformation

Rücktransformation – Vorüberlegungen

Karlsruher Institut für Technologie

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

- betrachte Matrix aus Transformation
  - erste Zeile enthält Lösung $T^{(n)}$
  - $T^{BWT} = L$ gegeben
  - F leicht zu bestimmen (sortiere L)
  - L ist *immer* Spalte vor F (zyklisch)

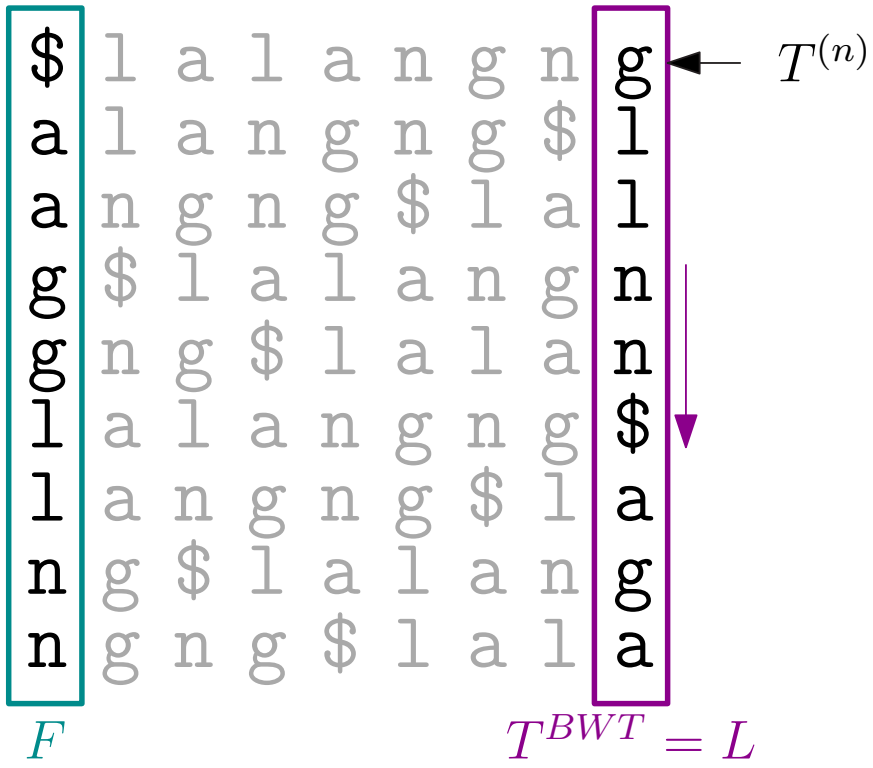


Burrows-Wheeler-Transformation

Rücktransformation – Vorüberlegungen

$$T^{BWT} = \underset{1}{\texttt{g}}\ \underset{2}{\texttt{l}}\ \underset{3}{\texttt{l}}\ \underset{4}{\texttt{n}}\ \underset{5}{\texttt{n}}\ \underset{6}{\texttt{\$}}\ \underset{7}{\texttt{a}}\ \underset{8}{\texttt{g}}\ \underset{9}{\texttt{a}}$$

- betrachte Matrix aus Transformation
  - erste Zeile enthält Lösung $T^{(n)}$
  - $T^{BWT} = L$ gegeben
  - F leicht zu bestimmen (sortiere L)
  - L ist *immer* Spalte vor F (zyklisch)

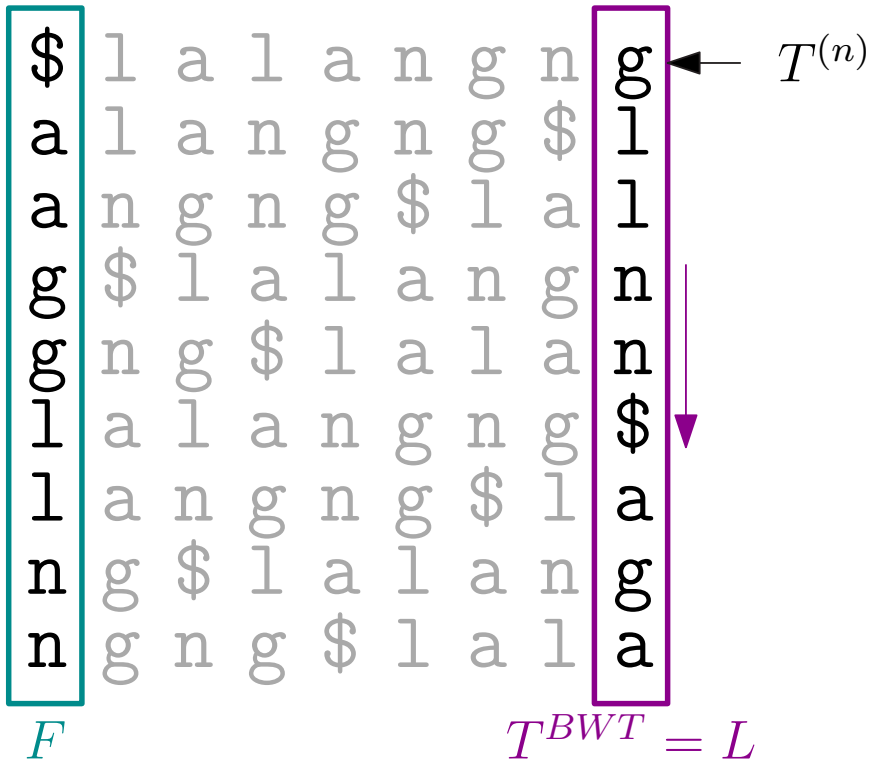


Burrows-Wheeler-Transformation

Rücktransformation – Vorüberlegungen

$$T^{BWT} = \underset{1}{\texttt{g}}\ \underset{2}{\texttt{l}}\ \underset{3}{\texttt{l}}\ \underset{4}{\texttt{n}}\ \underset{5}{\texttt{n}}\ \underset{6}{\texttt{\$}}\ \underset{7}{\texttt{a}}\ \underset{8}{\texttt{g}}\ \underset{9}{\texttt{a}}$$

- betrachte Matrix aus Transformation
  - erste Zeile enthält Lösung $T^{(n)}$
  - $T^{BWT} = L$ gegeben
  - F leicht zu bestimmen (sortiere L)
  - L ist *immer* Spalte vor F (zyklisch)



Burrows-Wheeler-Transformation

Rücktransformation – Vorüberlegungen

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

- betrachte Matrix aus Transformation
  - erste Zeile enthält Lösung $T^{(n)}$
  - $T^{BWT} = L$ gegeben
  - F leicht zu bestimmen (sortiere L)
  - L ist *immer* Spalte vor F (zyklisch)

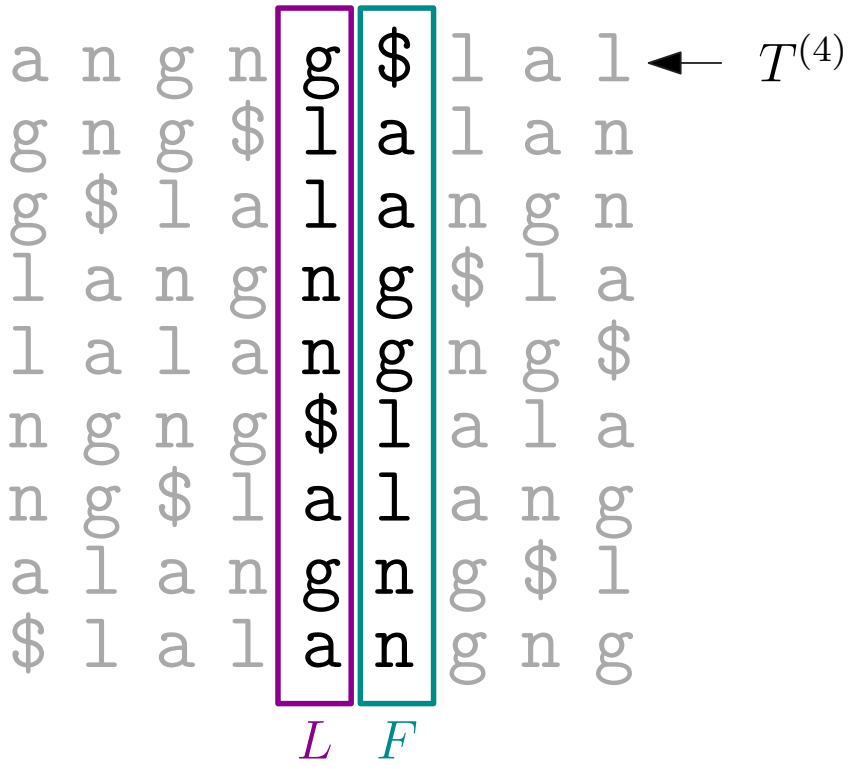
							L	F	
l	a	l	a	n	g	n	g	\$	← $T^{(1)}$
l	a	n	g	n	g	\$	l	a	
n	g	n	g	\$	l	a	l	a	
\$	l	a	l	a	n	g	n	g	
n	g	\$	l	a	l	a	n	g	
a	l	a	n	g	n	g	\$	l	
a	n	g	n	g	\$	l	a	l	
g	\$	l	a	l	a	n	g	n	
g	n	g	\$	l	a	l	a	n	

Burrows-Wheeler-Transformation

Rücktransformation – Vorüberlegungen

Karlsruher Institut für Technologie

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

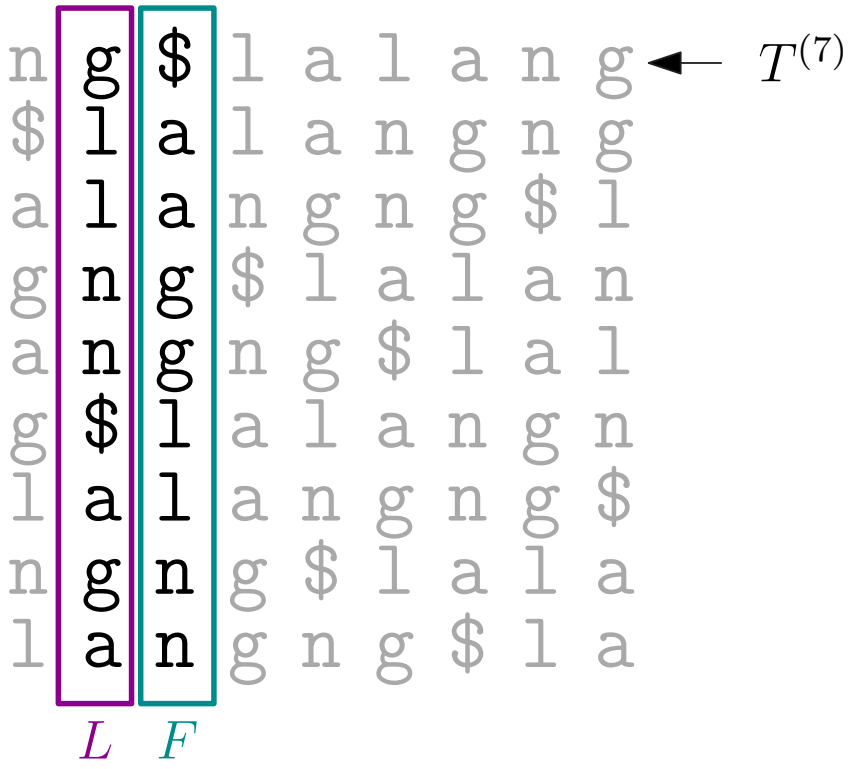


- betrachte Matrix aus Transformation
  - erste Zeile enthält Lösung $T^{(n)}$
  - $T^{BWT} = L$ gegeben
  - F leicht zu bestimmen (sortiere L)
  - L ist *immer* Spalte vor F (zyklisch)

Burrows-Wheeler-Transformation

Rücktransformation – Vorüberlegungen

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$



- betrachte Matrix aus Transformation
  - erste Zeile enthält Lösung $T^{(n)}$
  - $T^{BWT} = L$ gegeben
  - F leicht zu bestimmen (sortiere L)
  - L ist *immer* Spalte vor F (zyklisch)

Burrows-Wheeler-Transformation

Rücktransformation

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

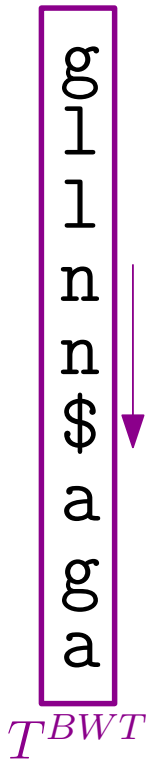
- schreibe T^{BWT} in Spaltenform
- sortiere zeilenweise
- schreibe T^{BWT} in Spaltenform davor
- wiederhole bis $|T^{BWT}|$ mal sortiert

Burrows-Wheeler-Transformation

Rücktransformation

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

- schreibe T^{BWT} in Spaltenform
- sortiere zeilenweise
- schreibe T^{BWT} in Spaltenform davor
- wiederhole bis $|T^{BWT}|$ mal sortiert



T^{BWT}

Burrows-Wheeler-Transformation

Rücktransformation

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

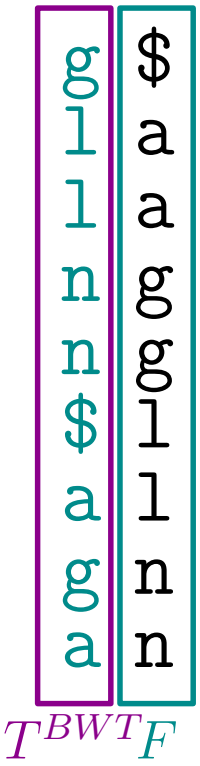
- schreibe T^{BWT} in Spaltenform
- sortiere zeilenweise
- schreibe T^{BWT} in Spaltenform davor
- wiederhole bis $|T^{BWT}|$ mal sortiert

Burrows-Wheeler-Transformation

Rücktransformation

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

- schreibe T^{BWT} in Spaltenform
- sortiere zeilenweise
- schreibe T^{BWT} in Spaltenform davor
- wiederhole bis $|T^{BWT}|$ mal sortiert

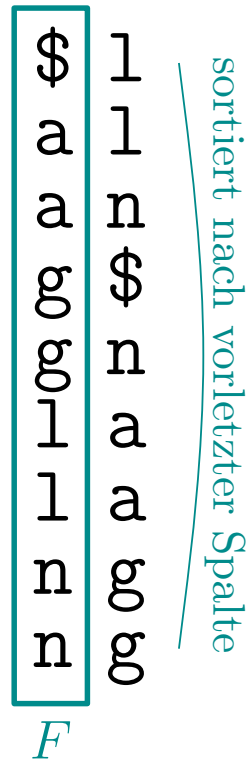


Burrows-Wheeler-Transformation

Rücktransformation

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

- schreibe T^{BWT} in Spaltenform
- sortiere zeilenweise
- schreibe T^{BWT} in Spaltenform davor
- wiederhole bis $|T^{BWT}|$ mal sortiert

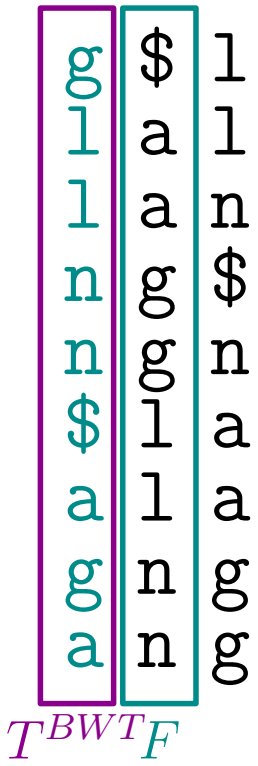


Burrows-Wheeler-Transformation

Rücktransformation

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

- schreibe T^{BWT} in Spaltenform
- sortiere zeilenweise
- schreibe T^{BWT} in Spaltenform davor
- wiederhole bis $|T^{BWT}|$ mal sortiert



Burrows-Wheeler-Transformation

Rücktransformation

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \mathtt{g} & \mathtt{l} & \mathtt{l} & \mathtt{n} & \mathtt{n} & \mathtt{\$} & \mathtt{a} & \mathtt{g} & \mathtt{a} \end{array}$$

- schreibe T^{BWT} in Spaltenform
- sortiere zeilenweise
- schreibe T^{BWT} in Spaltenform davor
- wiederhole bis $|T^{BWT}|$ mal sortiert

$$\begin{array}{|c|cc} \hline \mathtt{\$} & \mathtt{l} & \mathtt{a} \\ \mathtt{a} & \mathtt{l} & \mathtt{a} \\ \mathtt{a} & \mathtt{n} & \mathtt{g} \\ \mathtt{g} & \mathtt{\$} & \mathtt{l} \\ \mathtt{l} & \mathtt{n} & \mathtt{g} \\ \mathtt{l} & \mathtt{a} & \mathtt{l} \\ \mathtt{n} & \mathtt{a} & \mathtt{n} \\ \mathtt{n} & \mathtt{g} & \mathtt{\$} \\ \mathtt{n} & \mathtt{g} & \mathtt{n} \\ \hline \end{array}$$

F

sortiert nach drittletzter Spalte

Burrows-Wheeler-Transformation

Rücktransformation

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

- schreibe T^{BWT} in Spaltenform
- sortiere zeilenweise
- schreibe T^{BWT} in Spaltenform davor
- wiederhole bis $|T^{BWT}|$ mal sortiert

T^{BWT}	F		
g	$	l	a
l	a	l	a
l	a	n	g
n	g	$	l
n	g	n	g
$	l	a	l
a	l	a	n
g	n	g	$
a	n	g	n

Burrows-Wheeler-Transformation

Rücktransformation

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

- schreibe T^{BWT} in Spaltenform
- sortiere zeilenweise
- schreibe T^{BWT} in Spaltenform davor
- wiederhole bis $|T^{BWT}|$ mal sortiert

$$\ldots \quad \begin{array}{|c|c|}\hline \texttt{g} & \texttt{\$} \\ \texttt{l} & \texttt{a} \\ \texttt{l} & \texttt{a} \\ \texttt{n} & \texttt{g} \\ \texttt{n} & \texttt{g} \\ \texttt{\$} & \texttt{l} \\ \texttt{a} & \texttt{l} \\ \texttt{g} & \texttt{n} \\ \texttt{a} & \texttt{n} \\ \hline \end{array} \begin{array}{l} \texttt{l a} \\ \texttt{l a} \\ \texttt{n g} \\ \texttt{\$ l} \\ \texttt{n g} \\ \texttt{a l} \\ \texttt{a n} \\ \texttt{g \$} \\ \texttt{g n} \end{array}$$

T^{BWT} F

Burrows-Wheeler-Transformation

Rücktransformation

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a

- schreibe T^{BWT} in Spaltenform
- sortiere zeilenweise
- schreibe T^{BWT} in Spaltenform davor
- wiederhole bis $|T^{BWT}|$ mal sortiert

```
$ l a l a n g n g
a l a n g n g $ l
a n g n g $ l a l
g $ l a l a n g n
g n g $ l a l a n
l a l a n g n g $
l a n g n g $ l a
n g $ l a l a n g
n g n g $ l a l a
```

Burrows-Wheeler-Transformation

Rücktransformation

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a

$	l	a	l	a	n	g	n	g	← $T^{(n)}$
a	l	a	n	g	n	g	$	l	
a	n	g	n	g	$	l	a	l	
g	$	l	a	l	a	n	g	n	
g	n	g	$	l	a	l	a	n	
l	a	l	a	n	g	n	g	$	
l	a	n	g	n	g	$	l	a	
n	g	$	l	a	l	a	n	g	
n	g	n	g	$	l	a	l	a	

- schreibe T^{BWT} in Spaltenform
- sortiere zeilenweise
- schreibe T^{BWT} in Spaltenform davor
- wiederhole bis $|T^{BWT}|$ mal sortiert

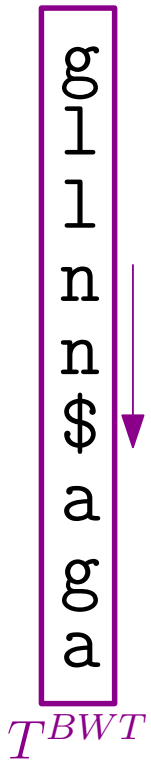
- $T^{BWT} = \texttt{gllnn\$aga} \quad \rightarrow \quad T = \texttt{lalangng\$}$ $\mathcal{O}(n^2 \log n)$

Burrows-Wheeler-Transformation

Rücktransformation – weitere Überlegungen

$$\begin{array}{rccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ T^{BWT} = & \mathtt{g} & \mathtt{l} & \mathtt{l} & \mathtt{n} & \mathtt{n} & \mathtt{\$} & \mathtt{a} & \mathtt{g} & \mathtt{a} \\ T = & \multicolumn{4}{c}{\text{———————}} & ? & \multicolumn{4}{c}{\text{———————}} \end{array}$$

- Wie lautet das Vorgängerzeichen?
  $\rightarrow$ *last-to-front mapping* $LF[\cdot]$
  (Position in $L[\cdot]$ an der Vorgänger steht)

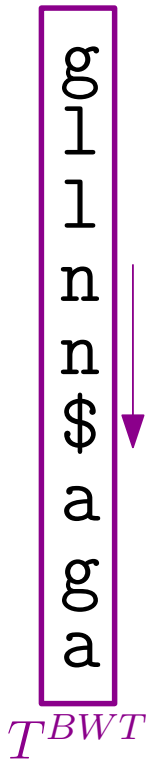


Burrows-Wheeler-Transformation

Rücktransformation – weitere Überlegungen

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$T =$									$

- Wie lautet das Vorgängerzeichen?
  $\rightarrow$ *last-to-front mapping* $LF[\cdot]$
  (Position in $L[\cdot]$ an der Vorgänger steht)

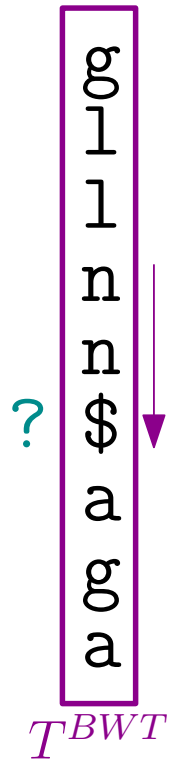


Burrows-Wheeler-Transformation

Rücktransformation – weitere Überlegungen

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$T =$								?	$

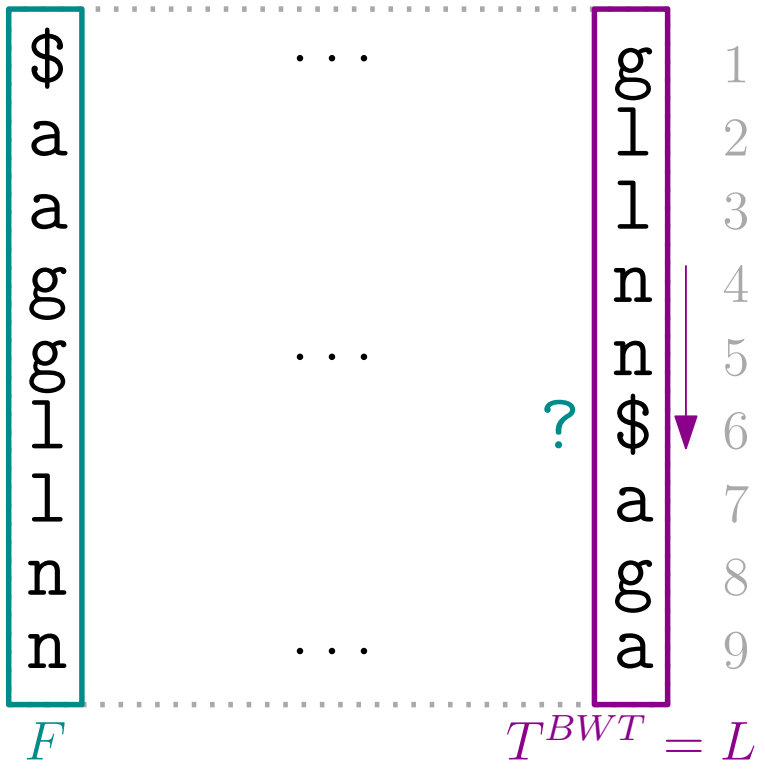
- Wie lautet das Vorgängerzeichen?
  $\rightarrow$ *last-to-front mapping* $LF[\cdot]$
  (Position in $L[\cdot]$ an der Vorgänger steht)



Burrows-Wheeler-Transformation

Rücktransformation – weitere Überlegungen

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$T =$								?	$



- Wie lautet das Vorgängerzeichen?
  → *last-to-front mapping* $LF[\cdot]$
  (Position in $L[\cdot]$ an der Vorgänger steht)

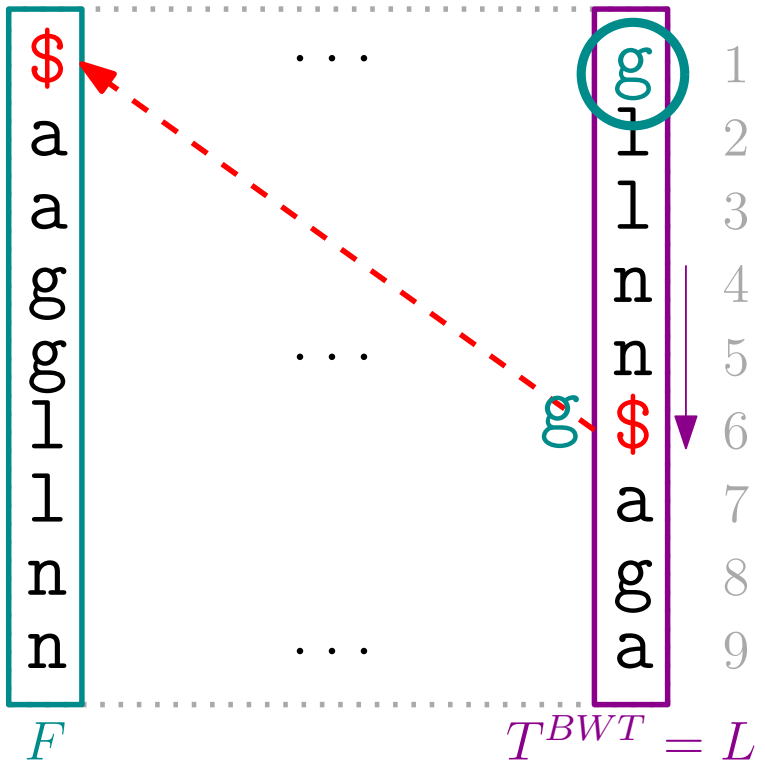
- $LF[6] = ?$

Burrows-Wheeler-Transformation

Rücktransformation – weitere Überlegungen

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$T =$								?	$

- Wie lautet das Vorgängerzeichen?
  $\rightarrow$ *last-to-front mapping* $LF[\cdot]$
  (Position in $L[\cdot]$ an der Vorgänger steht)

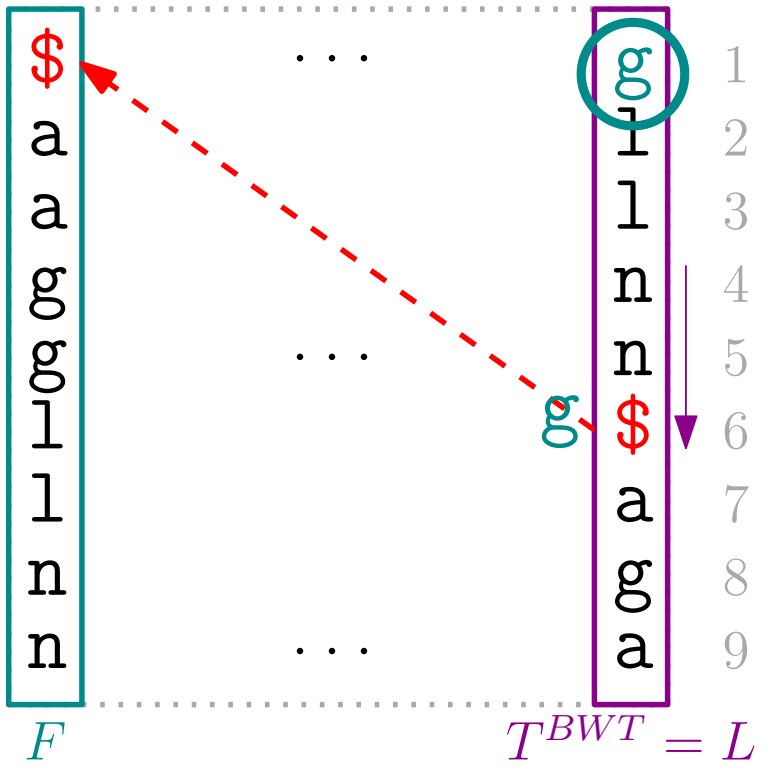


- $LF[6] = 1$
  ($LF[6]$ ist die Position in $F[\cdot]$ an der $L[6]$ steht)

Burrows-Wheeler-Transformation

Rücktransformation – weitere Überlegungen

$$
\begin{array}{rccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
T^{BWT} = & \mathtt{g} & \mathtt{l} & \mathtt{l} & \mathtt{n} & \mathtt{n} & \mathtt{\$} & \mathtt{a} & \mathtt{g} & \mathtt{a} \\
T = & & & & & & & & ? & \mathtt{\$}
\end{array}
$$



- Wie lautet das Vorgängerzeichen?
  $\rightarrow$ *last-to-front mapping* $LF[\cdot]$
  (Position in $L[\cdot]$ an der Vorgänger steht)

- $LF[i] = j \Leftrightarrow T^{(SA[j])} = (T^{(SA[i])})^{(n)} \Leftrightarrow SA[i] = SA[j] - 1 \pmod n$
  ($LF[i]$ ist die Position in $F[\cdot]$ an der $L[i]$ steht)

Burrows-Wheeler-Transformation

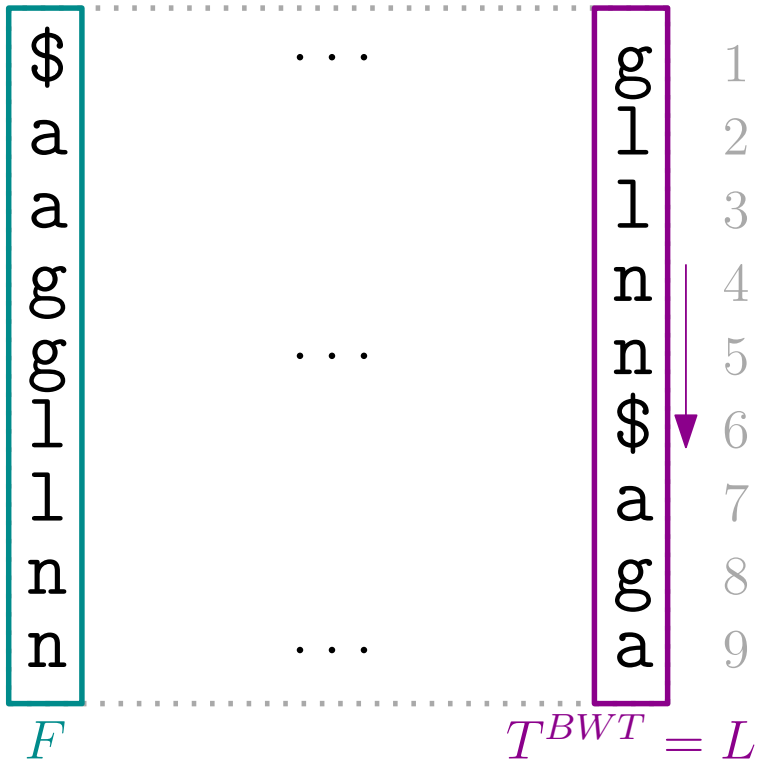
Rücktransformation – weitere Überlegungen

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

- T^{BWT} hat Struktur, so dass gilt

  → gleiche Zeichen besitzen gleiche Reihenfolge in $F[\cdot]$ und $L[\cdot]$

  → falls $L[i] = L[j]$ mit $i < j$, dann $LF[i] < LF[j]$



- Grund: $\alpha < \beta$ lexikographisch (nach Konstruktion)

Burrows-Wheeler-Transformation

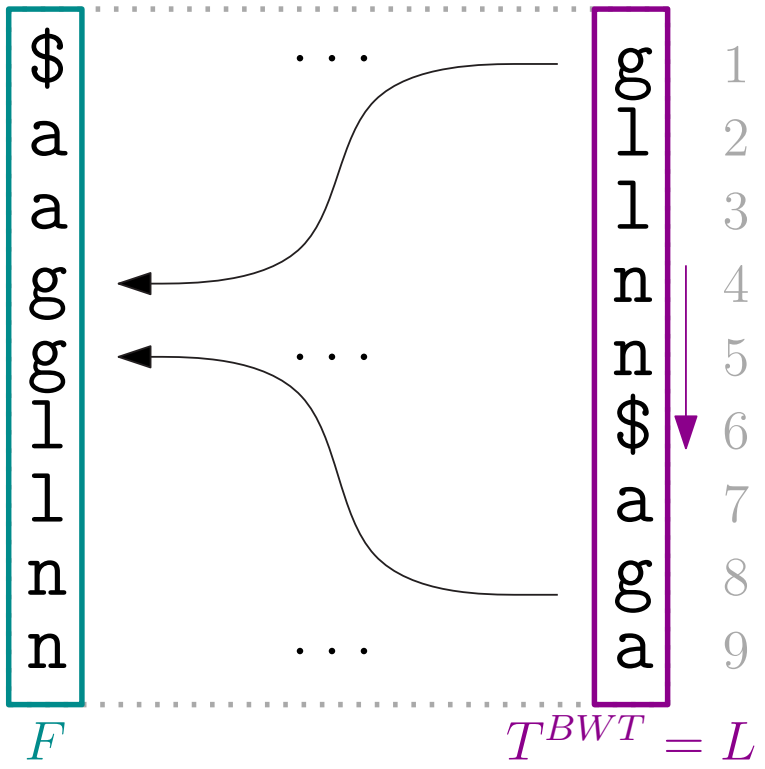
Rücktransformation – weitere Überlegungen

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

- T^{BWT} hat Struktur, so dass gilt

  $\rightarrow$ gleiche Zeichen besitzen gleiche Reihenfolge in $F[\cdot]$ und $L[\cdot]$

  $\rightarrow$ falls $L[i] = L[j]$ mit $i < j$, dann $LF[i] < LF[j]$

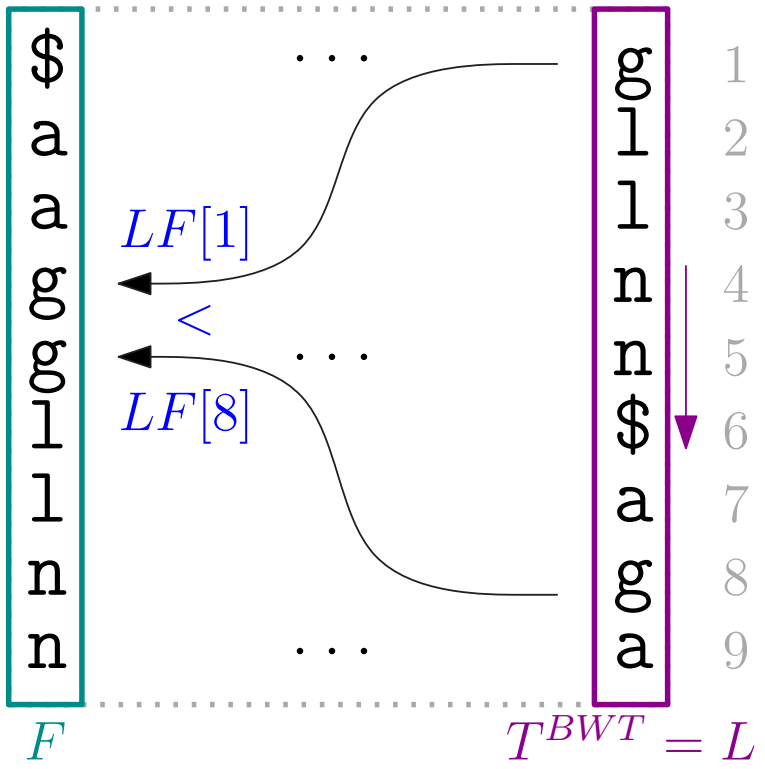


- Grund: $\alpha < \beta$ lexikographisch (nach Konstruktion)

Burrows-Wheeler-Transformation

Rücktransformation – weitere Überlegungen

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$



- T^{BWT} hat Struktur, so dass gilt

  → gleiche Zeichen besitzen gleiche Reihenfolge in $F[\cdot]$ und $L[\cdot]$

  → falls $L[i] = L[j]$ mit $i < j$, dann $LF[i] < LF[j]$

- Grund: $\alpha < \beta$ lexikographisch (nach Konstruktion)

Burrows-Wheeler-Transformation

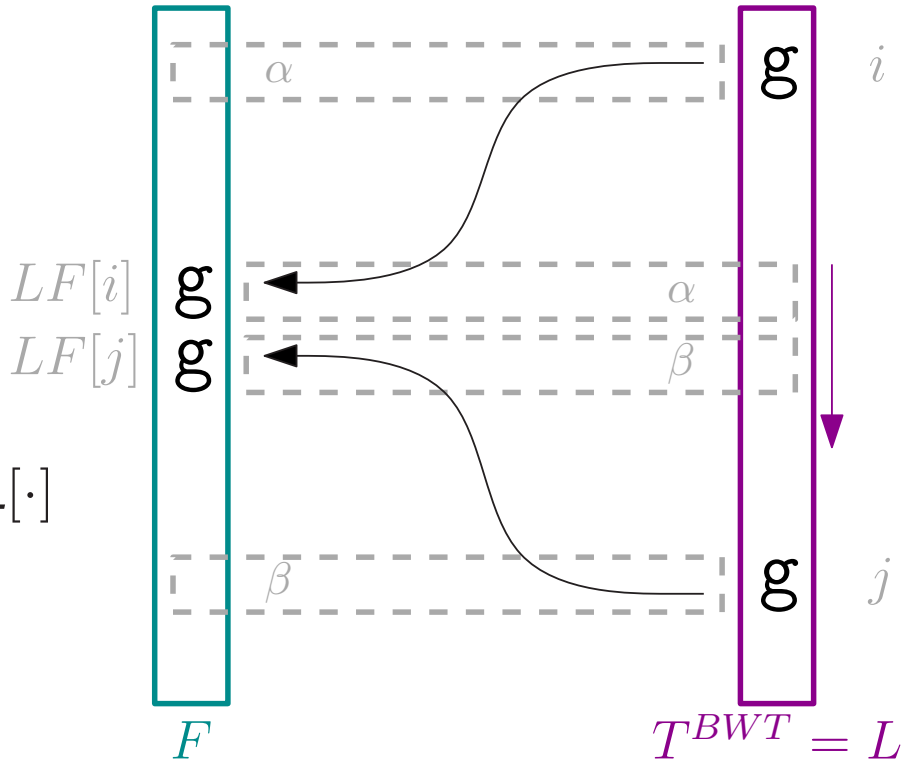
Rücktransformation – weitere Überlegungen

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \mathtt{g} & \mathtt{l} & \mathtt{l} & \mathtt{n} & \mathtt{n} & \mathtt{\$} & \mathtt{a} & \mathtt{g} & \mathtt{a} \end{array}$$

- T^{BWT} hat Struktur, so dass gilt

  $\rightarrow$ gleiche Zeichen besitzen gleiche Reihenfolge in $F[\cdot]$ und $L[\cdot]$

  $\rightarrow$ falls $L[i] = L[j]$ mit $i < j$, dann $LF[i] < LF[j]$



- Grund: $\alpha < \beta$ lexikographisch (nach Konstruktion)

Burrows-Wheeler-Transformation

Berechnung von $LF[\cdot]$

$$T^{BWT} = \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \texttt{g} & \texttt{l} & \texttt{l} & \texttt{n} & \texttt{n} & \texttt{\$} & \texttt{a} & \texttt{g} & \texttt{a} \end{array}$$

- $LF[\cdot]$ nur aus $T^{BWT}[\cdot]$ berechenbar
- $LF[i] = C(L[i]) + occ[i]$
  - $C(a)$ $= \#_{\text{Zeichen}}$ kleiner als a
  - $occ[i] = \#_{\text{Zeichen}}$ gleich $L[i]$ in $L[1..i]$

  ($LF[i]$ ist Position in $F[\cdot]$, an der $L[i]$ steht)

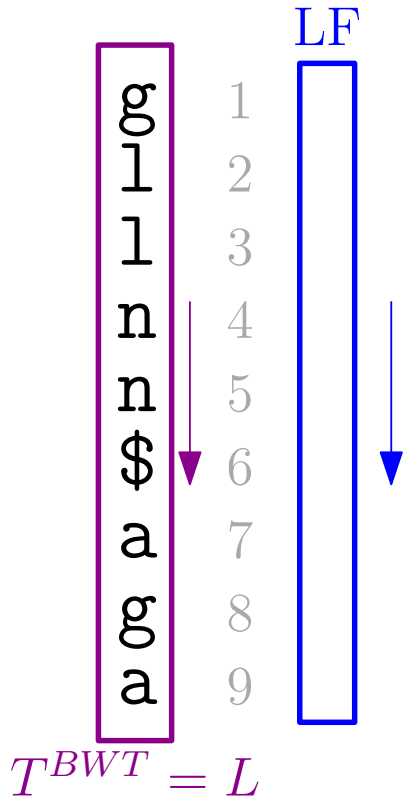
Burrows-Wheeler-Transformation

Berechnung von $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a

- $LF[\cdot]$ nur aus $T^{BWT}[\cdot]$ berechenbar
- $LF[i] = C(L[i]) + occ[i]$
  - $C(a)$ $= \#_{\text{Zeichen}}$ kleiner als a
  - $occ[i] = \#_{\text{Zeichen}}$ gleich $L[i]$ in $L[1..i]$

($LF[i]$ ist Position in $F[\cdot]$, an der $L[i]$ steht)



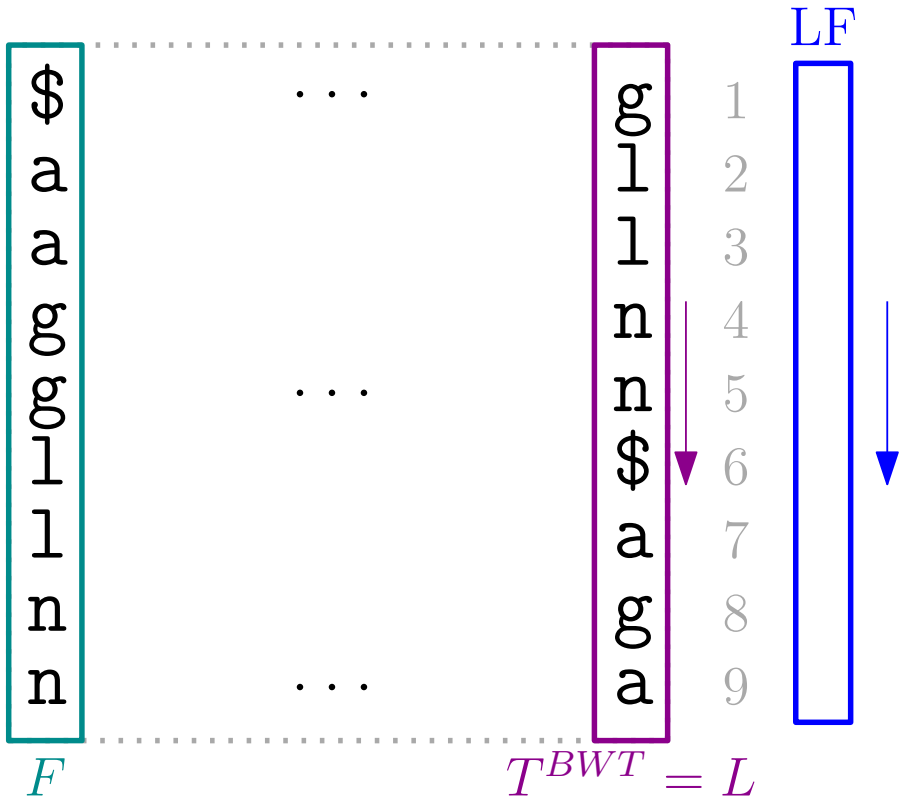
Burrows-Wheeler-Transformation

Berechnung von $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a

- $LF[\cdot]$ nur aus $T^{BWT}[\cdot]$ berechenbar
- $LF[i] = C(L[i]) + occ[i]$
  - $C(a)$ $= \#_{\text{Zeichen}}$ kleiner als a
  - $occ[i] = \#_{\text{Zeichen}}$ gleich $L[i]$ in $L[1..i]$

  ($LF[i]$ ist Position in $F[\cdot]$, an der $L[i]$ steht)



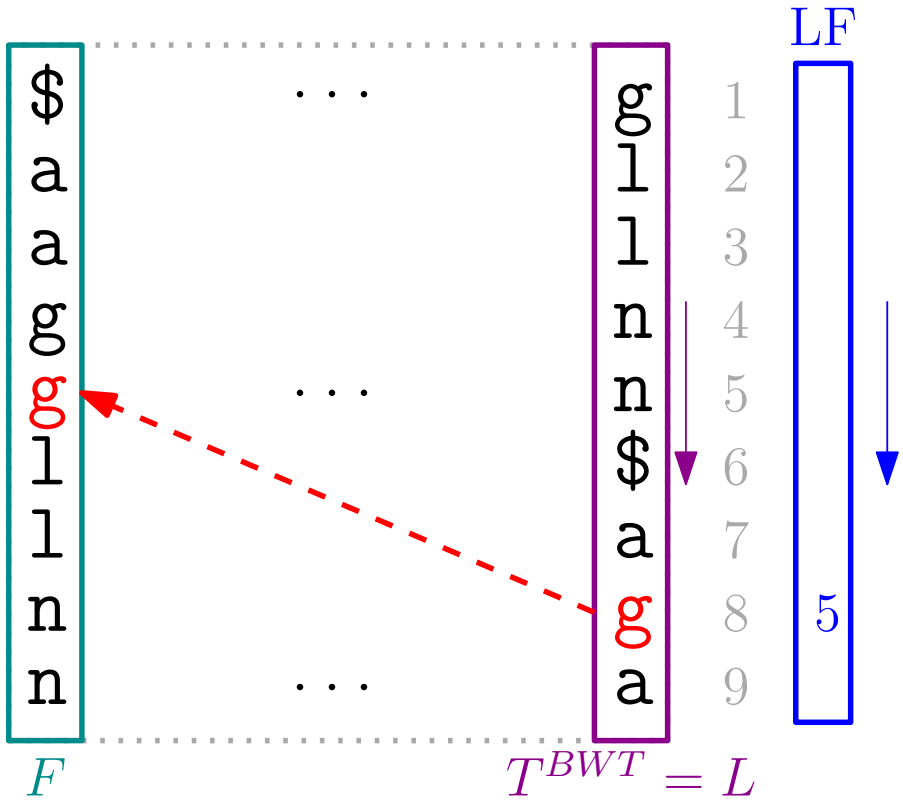
Burrows-Wheeler-Transformation

Berechnung von $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a

- $LF[\cdot]$ nur aus $T^{BWT}[\cdot]$ berechenbar
- $LF[i] = C(L[i]) + occ[i]$
  - $C(a)$ $= \#_{\text{Zeichen}}$ kleiner als a
  - $occ[i] = \#_{\text{Zeichen}}$ gleich $L[i]$ in $L[1..i]$

  ($LF[i]$ ist Position in $F[\cdot]$, an der $L[i]$ steht)



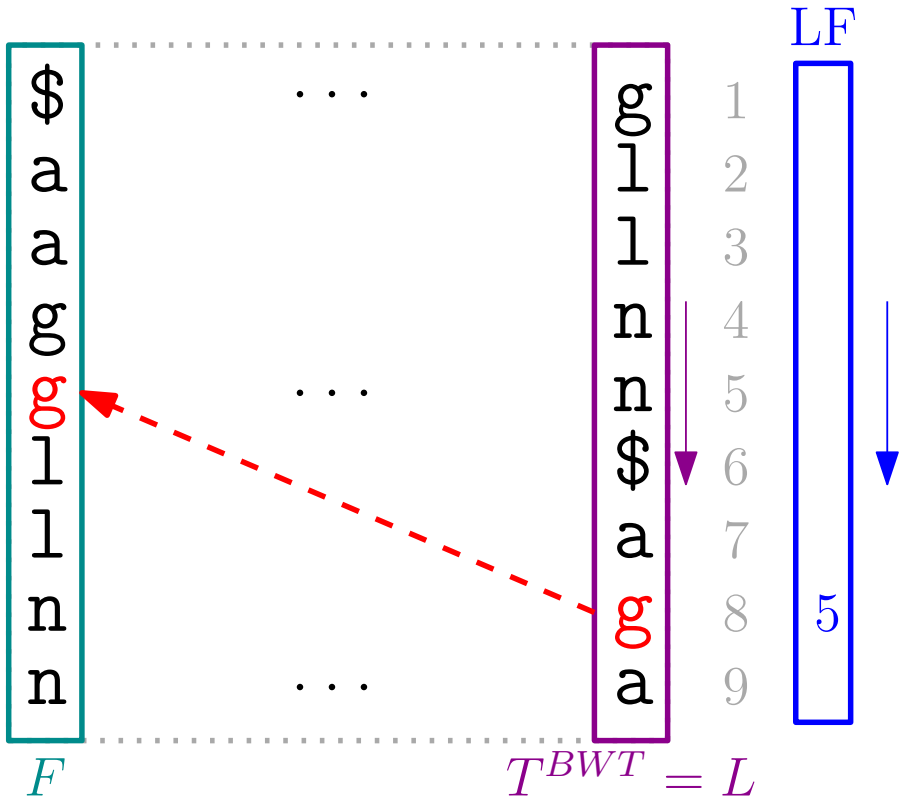
Burrows-Wheeler-Transformation

Berechnung von $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a

- $LF[\cdot]$ nur aus $T^{BWT}[\cdot]$ berechenbar
- $LF[i] = C(L[i]) + occ[i]$
  - $C(a)$ $= \#_{\text{Zeichen}}$ kleiner als a
  - $occ[i] = \#_{\text{Zeichen}}$ gleich $L[i]$ in $L[1..i]$

  ($LF[i]$ ist Position in $F[\cdot]$, an der $L[i]$ steht)



- $C(\cdot)$, $occ[\cdot]$ in $\mathcal{O}(n)$ berechenbar $\rightarrow$ $LF[\cdot]$ auch in $\mathcal{O}(n)$ berechenbar

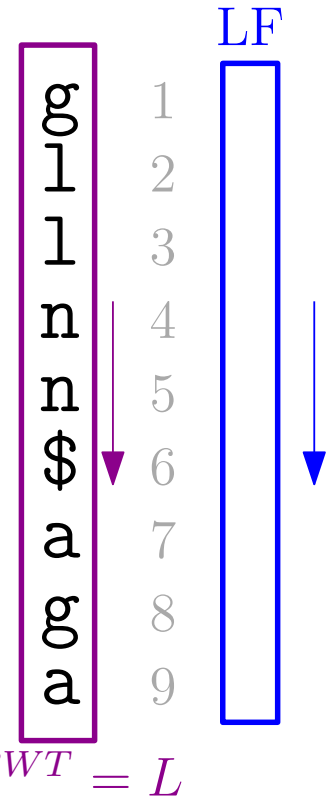
Burrows-Wheeler-Transformation

Ablauf der Berechnung von $LF[\cdot]$

$$\begin{array}{rccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ T^{BWT} = & \mathtt{g} & \mathtt{l} & \mathtt{l} & \mathtt{n} & \mathtt{n} & \mathtt{\$} & \mathtt{a} & \mathtt{g} & \mathtt{a} \\ occ = & & & & & & & & & \end{array}$$

$$\begin{array}{rccccc} & \mathtt{\$} & \mathtt{a} & \mathtt{g} & \mathtt{l} & \mathtt{n} \\ h = & 0 & 0 & 0 & 0 & 0 \end{array}$$

(zählt Zeichen)



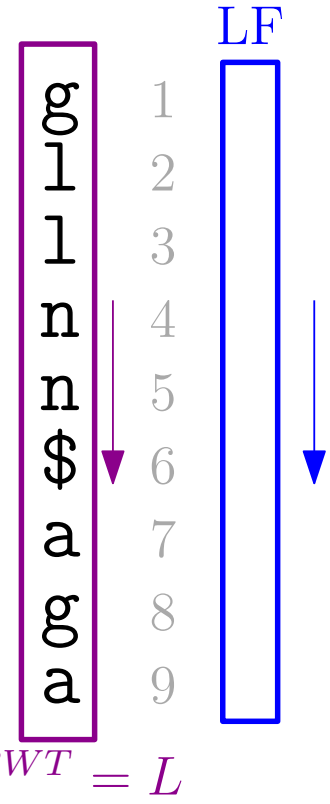
- initialisiere *occ* und *h*
- laufe durch $T^{BWT} = L$ $(i = 1..n)$
  - $h(\ L[i]\ ) = h(\ L[i]\ ) + 1$
  - $occ(\ L[i]\ ) = h(\ L[i]\ )$
- bilde exkl. Präfixsumme von $h \rightarrow C$
- $LF[i] = C(L[i]) + occ[i]$

Burrows-Wheeler-Transformation

Ablauf der Berechnung von $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$occ =$									

	$	a	g	l	n
$h =$ (zählt Zeichen)	0	0	0	0	0



- initialisiere *occ* und *h*
- laufe durch $T^{BWT} = L$ $(i = 1..n)$
  - $h(\ L[i]\ ) = h(\ L[i]\ ) + 1$
  - $occ(\ L[i]\ ) = h(\ L[i]\ )$
- bilde exkl. Präfixsumme von $h \rightarrow C$
- $LF[i] = C(L[i]) + occ[i]$

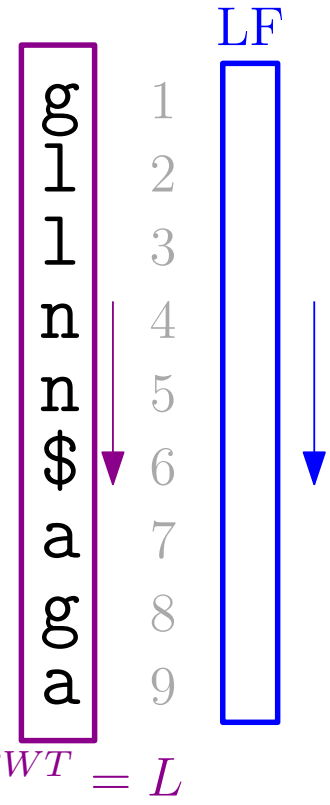
Burrows-Wheeler-Transformation

Ablauf der Berechnung von $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$occ =$									

	$	a	g	l	n
$h =$	0	0	1	0	0

(zählt Zeichen)



- initialisiere *occ* und *h*
- laufe durch $T^{BWT} = L$ $(i = 1..n)$
  - $h(\ L[i]\ ) = h(\ L[i]\ ) + 1$
  - $occ(\ L[i]\ ) = h(\ L[i]\ )$
- bilde exkl. Präfixsumme von $h \rightarrow C$
- $LF[i] = C(L[i]) + occ[i]$

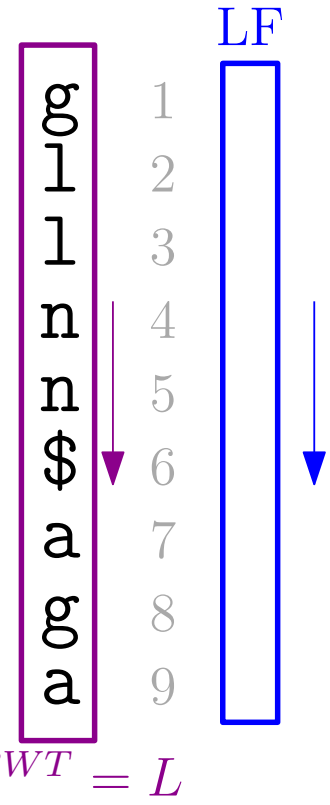
Burrows-Wheeler-Transformation

Ablauf der Berechnung von $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$occ =$	1								

	$	a	g	l	n
$h =$	0	0	1	0	0

(zählt Zeichen)



- initialisiere *occ* und *h*
- laufe durch $T^{BWT} = L$ $(i = 1..n)$
  - $h(\ L[i]\ ) = h(\ L[i]\ ) + 1$
  - $occ(\ L[i]\ ) = h(\ L[i]\ )$
- bilde exkl. Präfixsumme von $h \rightarrow C$
- $LF[i] = C(L[i]) + occ[i]$

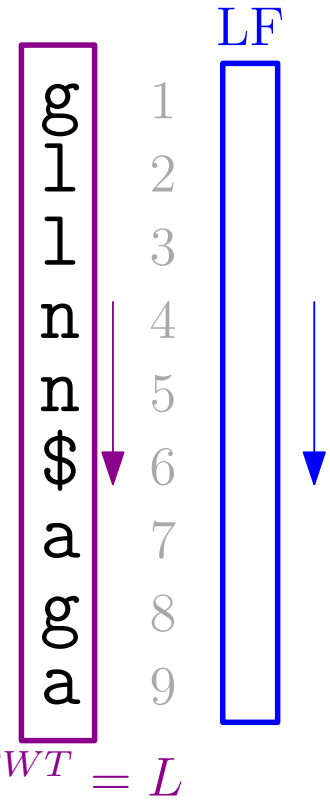
Burrows-Wheeler-Transformation

Ablauf der Berechnung von $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$occ =$	1	1							

	$	a	g	l	n
$h =$	0	0	1	1	0

(zählt Zeichen)



- initialisiere *occ* und *h*
- laufe durch $T^{BWT} = L$ $(i = 1..n)$
  - $h(\ L[i]\ ) = h(\ L[i]\ ) + 1$
  - $occ(\ L[i]\ ) = h(\ L[i]\ )$
- bilde exkl. Präfixsumme von $h \rightarrow C$
- $LF[i] = C(L[i]) + occ[i]$

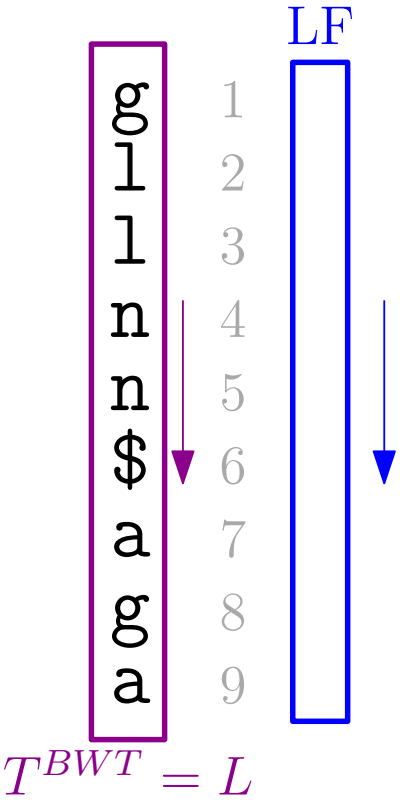
Burrows-Wheeler-Transformation

Ablauf der Berechnung von $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$occ =$	1	1	2						

	$	a	g	l	n
$h =$	0	0	1	2	0

(zählt Zeichen)



- initialisiere *occ* und *h*
- laufe durch $T^{BWT} = L$ $(i = 1..n)$
  - $h(\ L[i]\ ) = h(\ L[i]\ ) + 1$
  - $occ(\ L[i]\ ) = h(\ L[i]\ )$
- bilde exkl. Präfixsumme von $h \rightarrow C$
- $LF[i] = C(L[i]) + occ[i]$

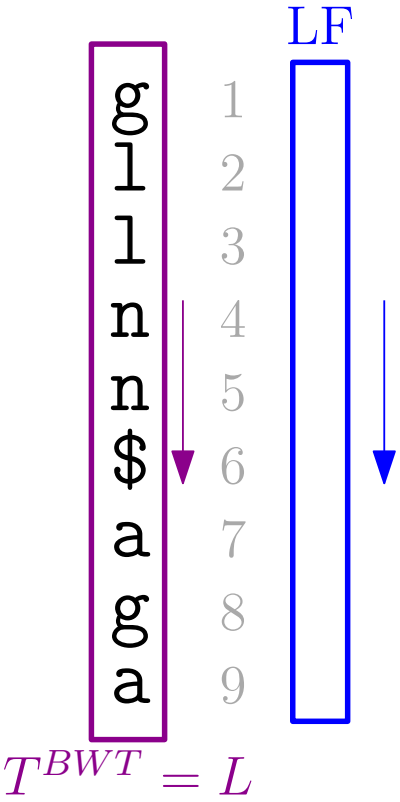
Burrows-Wheeler-Transformation

Ablauf der Berechnung von $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$occ =$	1	1	2	...					

	$	a	g	l	n
$h =$	0	0	1	2	0

(zählt Zeichen)



- initialisiere *occ* und *h*
- laufe durch $T^{BWT} = L$ $(i = 1..n)$
  - $h(\ L[i]\ ) = h(\ L[i]\ ) + 1$
  - $occ(\ L[i]\ ) = h(\ L[i]\ )$
- bilde exkl. Präfixsumme von $h \rightarrow C$
- $LF[i] = C(L[i]) + occ[i]$

Burrows-Wheeler-Transformation

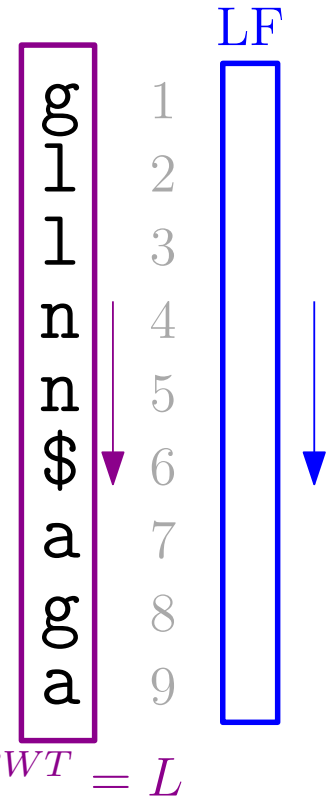
Ablauf der Berechnung von $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$occ =$	1	1	2	1	2	1	1	2	2

	$	a	g	l	n
$h =$	1	2	2	2	2

(zählt Zeichen)

- initialisiere *occ* und *h*
- laufe durch $T^{BWT} = L$ $(i = 1..n)$
  - $h(\ L[i]\ ) = h(\ L[i]\ ) + 1$
  - $occ(\ L[i]\ ) = h(\ L[i]\ )$
- bilde exkl. Präfixsumme von $h \rightarrow C$
- $LF[i] = C(L[i]) + occ[i]$



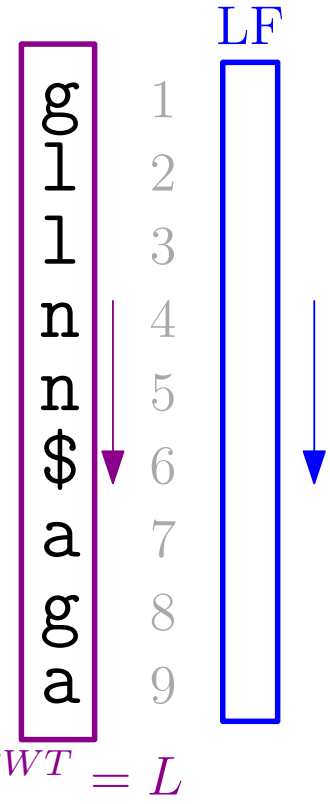
Burrows-Wheeler-Transformation

Ablauf der Berechnung von $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$occ =$	1	1	2	1	2	1	1	2	2

	$	a	g	l	n
$h =$	1	2	2	2	2
$C =$	0	1	3	5	7

(Präfixsumme von h)



- initialisiere *occ* und *h*
- laufe durch $T^{BWT} = L$ $(i = 1..n)$
  - $h(\ L[i]\ ) = h(\ L[i]\ ) + 1$
  - $occ(\ L[i]\ ) = h(\ L[i]\ )$
- bilde exkl. Präfixsumme von $h \rightarrow C$
- $LF[i] = C(L[i]) + occ[i]$

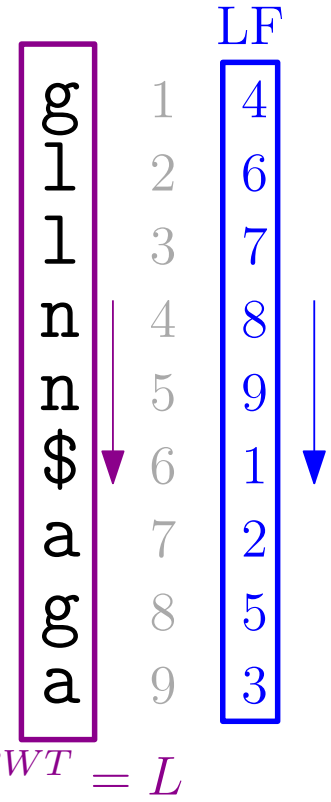
Burrows-Wheeler-Transformation

Ablauf der Berechnung von $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$occ =$	1	1	2	1	2	1	1	2	2

	$	a	g	l	n
$h =$	1	2	2	2	2
$C =$	0	1	3	5	7

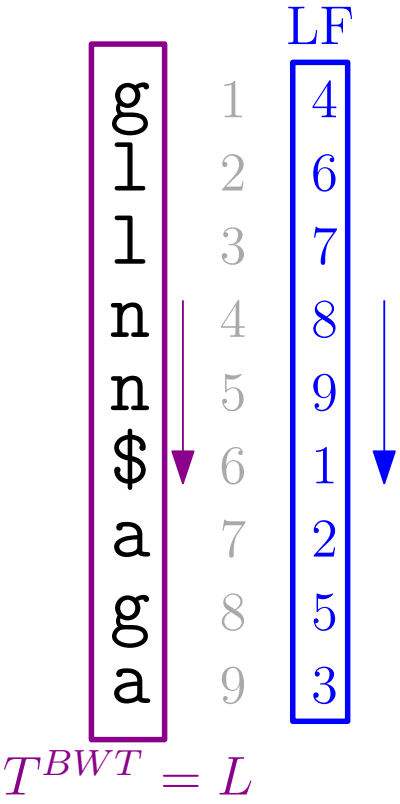
- initialisiere *occ* und *h*
- laufe durch $T^{BWT} = L$ $(i = 1..n)$
  - $h(\ L[i]\ ) = h(\ L[i]\ ) + 1$
  - $occ(\ L[i]\ ) = h(\ L[i]\ )$
- bilde exkl. Präfixsumme von $h \rightarrow C$
- $LF[i] = C(L[i]) + occ[i]$



Burrows-Wheeler-Transformation

Rücktransformation mit $LF[\cdot]$

$$\begin{array}{rccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ T^{BWT} = & \mathtt{g} & \mathtt{l} & \mathtt{l} & \mathtt{n} & \mathtt{n} & \mathtt{\$} & \mathtt{a} & \mathtt{g} & \mathtt{a} \\ T = & & & & & & & & & \end{array}$$

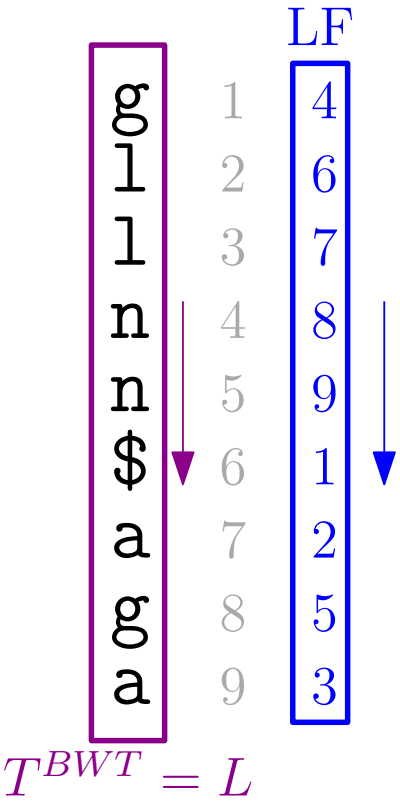


- Berechnung von T von rechts nach links
  - Initialisierung: $T[n] = \$ \Rightarrow LF(?) = 1$ (immer!)
  - $L[1] = \mathtt{g} \Rightarrow T[n-1] = g \Rightarrow LF(1) = 4$
  - $L[4] = \mathtt{n} \Rightarrow T[n-2] = n \Rightarrow LF(4) = 8$
  - ...

Burrows-Wheeler-Transformation

Rücktransformation mit $LF[\cdot]$

$$
\begin{array}{rccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
T^{BWT} = & \mathtt{g} & \mathtt{l} & \mathtt{l} & \mathtt{n} & \mathtt{n} & \mathtt{\$} & \mathtt{a} & \mathtt{g} & \mathtt{a} \\
T = & & & & & & & & & \mathtt{\$}
\end{array}
$$



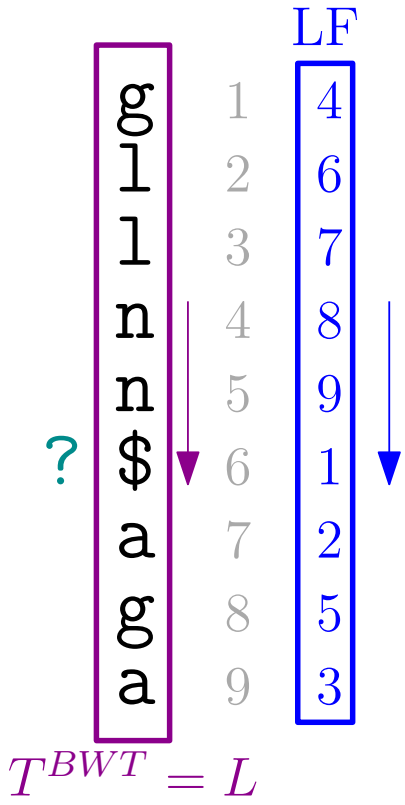
- Berechnung von T von rechts nach links
  - Initialisierung: $T[n] = \$ \Rightarrow LF(?) = 1$ (immer!)
  - $L[1] = \mathtt{g} \Rightarrow T[n-1] = g \Rightarrow LF(1) = 4$
  - $L[4] = \mathtt{n} \Rightarrow T[n-2] = n \Rightarrow LF(4) = 8$
  - ...

Burrows-Wheeler-Transformation

Rücktransformation mit $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$T =$								?	$

- Berechnung von T von rechts nach links
  - Initialisierung: $T[n] = \$ \Rightarrow LF(?) = 1$ (immer!)
  - $L[1] = \mathtt{g} \Rightarrow T[n-1] = g \Rightarrow LF(1) = 4$
  - $L[4] = \mathtt{n} \Rightarrow T[n-2] = n \Rightarrow LF(4) = 8$
  - ...

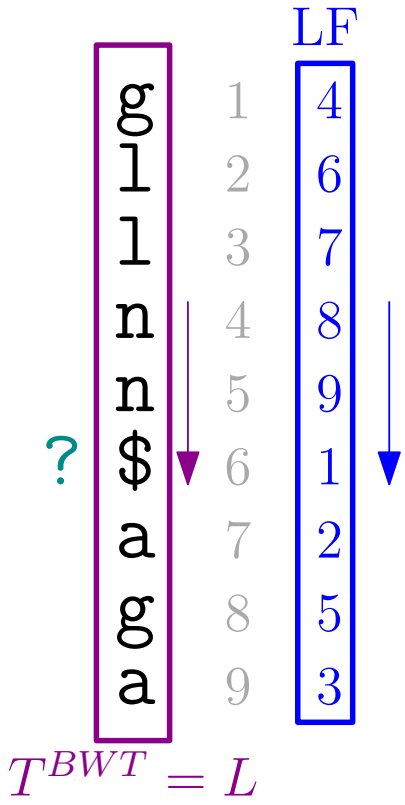


Burrows-Wheeler-Transformation

Rücktransformation mit $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$T =$								?	$

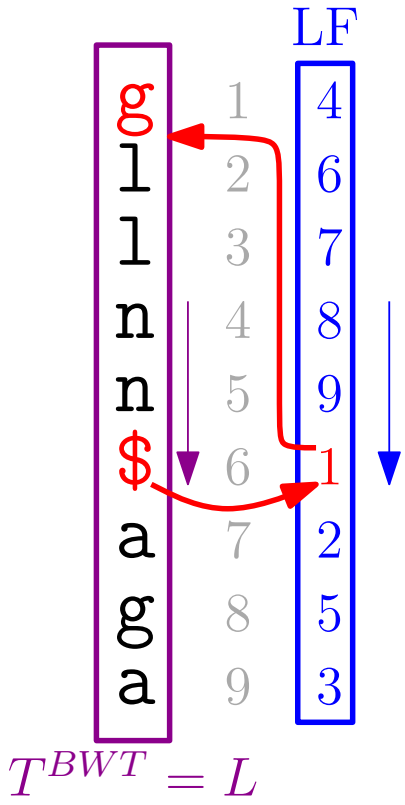
- Berechnung von T von rechts nach links
  - Initialisierung: $T[n] = \$ \Rightarrow LF(?) = 1$ (immer!)
  - $L[1] = \mathtt{g} \Rightarrow T[n-1] = g \Rightarrow LF(1) = 4$
  - $L[4] = \mathtt{n} \Rightarrow T[n-2] = n \Rightarrow LF(4) = 8$
  - ...



Burrows-Wheeler-Transformation

Rücktransformation mit $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$T =$								g	$

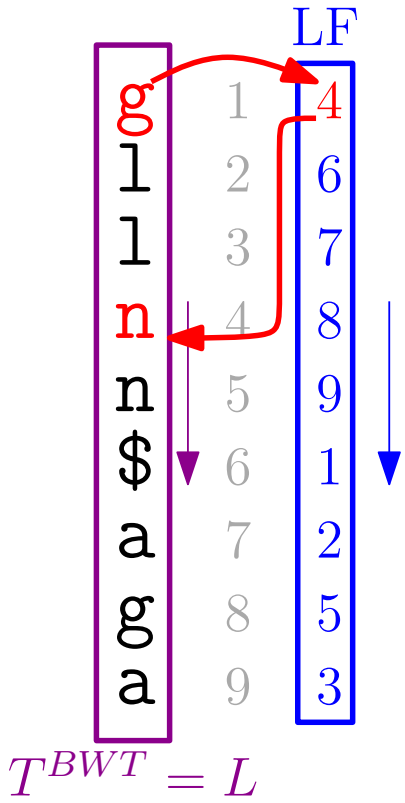


- Berechnung von T von rechts nach links
  - Initialisierung: $T[n] = \$ \Rightarrow LF(?) = 1$ (immer!)
  - $L[1] = \mathtt{g} \Rightarrow T[n-1] = g \Rightarrow LF(1) = 4$
  - $L[4] = \mathtt{n} \Rightarrow T[n-2] = n \Rightarrow LF(4) = 8$
  - ...

Burrows-Wheeler-Transformation

Rücktransformation mit $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$T =$							n	g	$



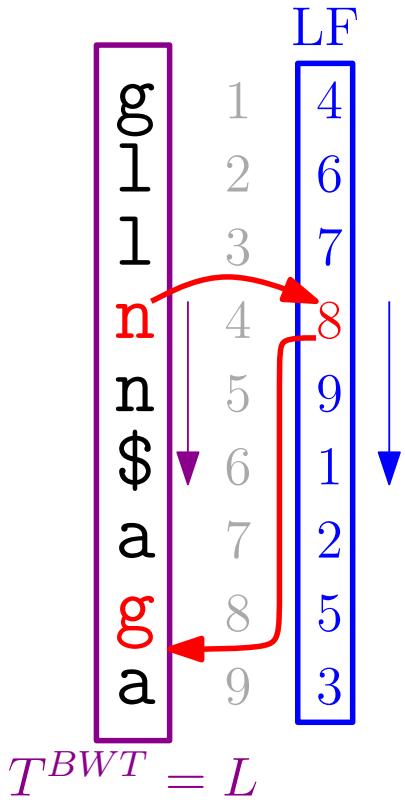
- Berechnung von T von rechts nach links
  - Initialisierung: $T[n] = \$ \Rightarrow LF(?) = 1$ (immer!)
  - $L[1] = \mathtt{g} \Rightarrow T[n-1] = g \Rightarrow LF(1) = 4$
  - $L[4] = \mathtt{n} \Rightarrow T[n-2] = n \Rightarrow LF(4) = 8$
  - ...

Burrows-Wheeler-Transformation

Rücktransformation mit $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$T =$					...	g	n	g	$

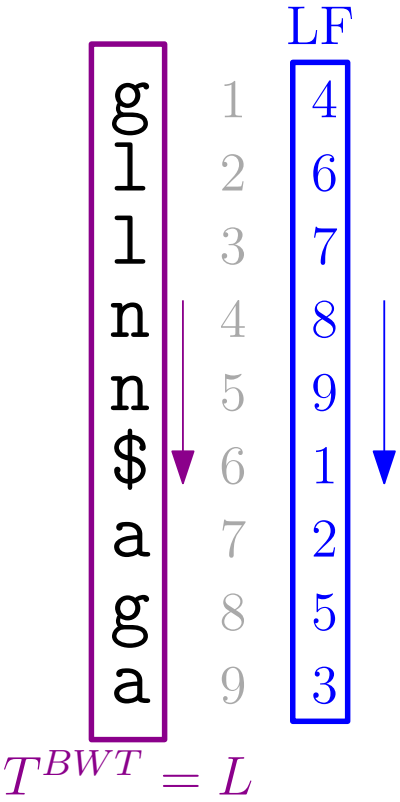
- Berechnung von T von rechts nach links
  - Initialisierung: $T[n] = \$ \Rightarrow LF(?) = 1$ (immer!)
  - $L[1] = \mathtt{g} \Rightarrow T[n-1] = g \Rightarrow LF(1) = 4$
  - $L[4] = \mathtt{n} \Rightarrow T[n-2] = n \Rightarrow LF(4) = 8$
  - ...



Burrows-Wheeler-Transformation

Rücktransformation mit $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$T =$	l	a	l	a	n	g	n	g	$

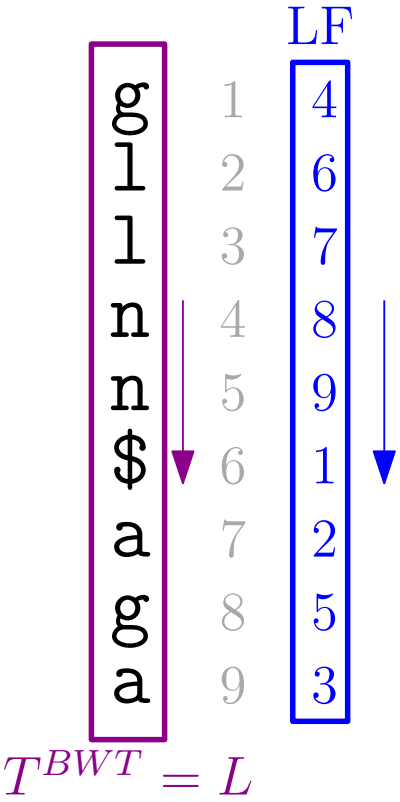


- Berechnung von T von rechts nach links
  - Initialisierung: $T[n] = \$ \Rightarrow LF(?) = 1$ (immer!)
  - $L[1] = \mathtt{g} \Rightarrow T[n-1] = g \Rightarrow LF(1) = 4$
  - $L[4] = \mathtt{n} \Rightarrow T[n-2] = n \Rightarrow LF(4) = 8$
  - ...
- Allgemein: $T[n-i] = L[LF(LF(\ldots(LF(1))\ldots))]$ $(i-1)$ LF Anwendungen

Burrows-Wheeler-Transformation

Rücktransformation mit $LF[\cdot]$

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	$	a	g	a
$T =$	l	a	l	a	n	g	n	g	$



- Berechnung von T von rechts nach links
  - Initialisierung: $T[n] = \$ \Rightarrow LF(?) = 1$ (immer!)
  - $L[1] = \mathtt{g} \Rightarrow T[n-1] = g \Rightarrow LF(1) = 4$
  - $L[4] = \mathtt{n} \Rightarrow T[n-2] = n \Rightarrow LF(4) = 8$
  - ...
- Allgemein: $T[n-i] = L[LF(LF(\ldots(LF(1))\ldots))]$ $(i-1)$ LF Anwendungen
- Rücktransformation in $\mathcal{O}(n)$ (ohne Zusatzinformationen)

Burrows-Wheeler-Transformation

Was bringt die BWT?

- benötigt gleichen Platz, verwendet gleiche Zeichen wie T
  $\rightarrow$ scheinbar keine Vorteile ?!?

- Permutation einfach umkehrbar
  (benötigt keine Zusatzinformationen, $\mathcal{O}(n)$)
- Zeichen mit ähnlichem Kontext gruppiert
  $\rightarrow$ vereinfacht Komprimierung

- besonders gut auf Texten mit
  vielen gleichen Substrings
  $\rightarrow$ Beispiel: englischer Text
  (u.a. viele “the”, . . .)

(Außerdem: Vorgänger von Suffixen einfach bestimmbar $\rightarrow$ Indizierung / Suche)

Burrows-Wheeler-Transformation

Was bringt die BWT?

- benötigt gleichen Platz, verwendet gleiche Zeichen wie T
  $\rightarrow$ scheinbar keine Vorteile ?!?

- Permutation einfach umkehrbar
  (benötigt keine Zusatzinformationen, $\mathcal{O}(n)$)
- Zeichen mit ähnlichem Kontext gruppiert
  $\rightarrow$ vereinfacht Komprimierung

- besonders gut auf Texten mit
  vielen gleichen Substrings
  $\rightarrow$ Beispiel: englischer Text
  (u.a. viele “the”, . . .)

(Außerdem: Vorgänger von Suffixen einfach bestimmbar $\rightarrow$ Indizierung / Suche)

Burrows-Wheeler-Transformation

Was bringt die BWT?

- benötigt gleichen Platz, verwendet gleiche Zeichen wie T
  $\rightarrow$ scheinbar keine Vorteile ?!?

- Permutation einfach umkehrbar
  (benötigt keine Zusatzinformationen, $\mathcal{O}(n)$)
- Zeichen mit ähnlichem Kontext gruppiert
  $\rightarrow$ vereinfacht Komprimierung

- besonders gut auf Texten mit
  vielen gleichen Substrings
  $\rightarrow$ Beispiel: englischer Text
  (u.a. viele “the”, . . .)

(Außerdem: Vorgänger von Suffixen einfach bestimmbar $\rightarrow$ Indizierung / Suche)

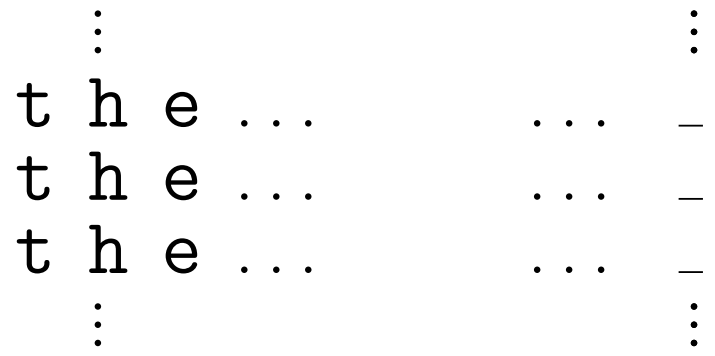
Burrows-Wheeler-Transformation

Was bringt die BWT?

- benötigt gleichen Platz, verwendet gleiche Zeichen wie T
  $\rightarrow$ scheinbar keine Vorteile ?!?

- Permutation einfach umkehrbar
  (benötigt keine Zusatzinformationen, $\mathcal{O}(n)$)
- Zeichen mit ähnlichem Kontext gruppiert
  $\rightarrow$ vereinfacht Komprimierung

- besonders gut auf Texten mit
  vielen gleichen Substrings
  $\rightarrow$ Beispiel: englischer Text
  (u.a. viele “the”, . . .)



(Außerdem: Vorgänger von Suffixen einfach bestimmbar $\rightarrow$ Indizierung / Suche)

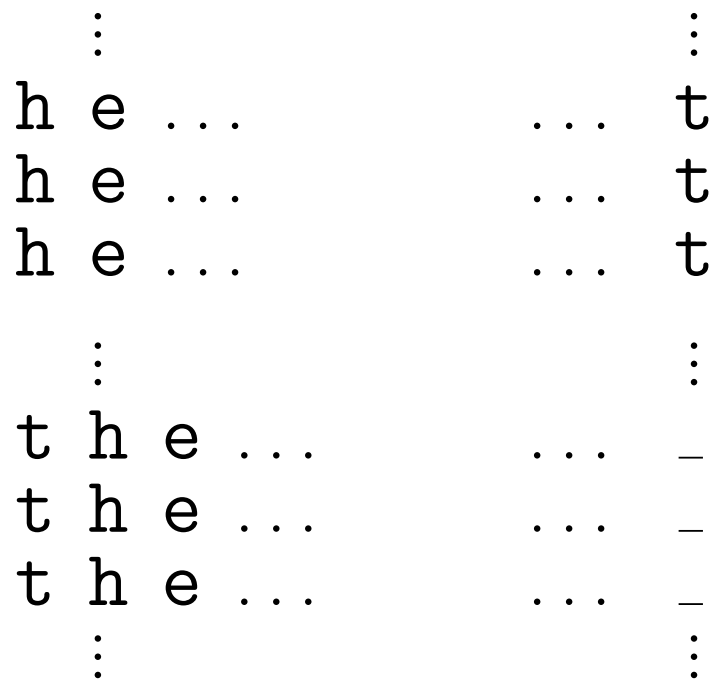
Burrows-Wheeler-Transformation

Was bringt die BWT?

- benötigt gleichen Platz, verwendet gleiche Zeichen wie T
  $\rightarrow$ scheinbar keine Vorteile ?!?

- Permutation einfach umkehrbar
  (benötigt keine Zusatzinformationen, $\mathcal{O}(n)$)
- Zeichen mit ähnlichem Kontext gruppiert
  $\rightarrow$ vereinfacht Komprimierung

- besonders gut auf Texten mit vielen gleichen Substrings
  $\rightarrow$ Beispiel: englischer Text
  (u.a. viele “the”, . . .)



(Außerdem: Vorgänger von Suffixen einfach bestimmbar $\rightarrow$ Indizierung / Suche)

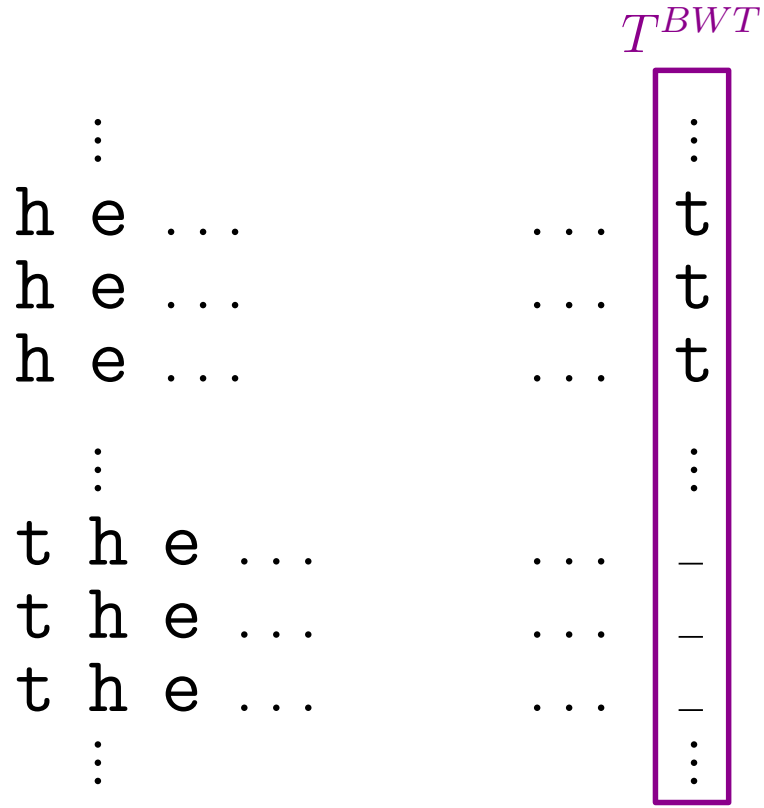
Burrows-Wheeler-Transformation

Was bringt die BWT?

- benötigt gleichen Platz, verwendet gleiche Zeichen wie T
  $\rightarrow$ scheinbar keine Vorteile ?!?

- Permutation einfach umkehrbar
  (benötigt keine Zusatzinformationen, $\mathcal{O}(n)$)
- Zeichen mit ähnlichem Kontext gruppiert
  $\rightarrow$ vereinfacht Komprimierung

- besonders gut auf Texten mit vielen gleichen Substrings
  $\rightarrow$ Beispiel: englischer Text
  (u.a. viele “the”, . . .)



(Außerdem: Vorgänger von Suffixen einfach bestimmbar $\rightarrow$ Indizierung / Suche)

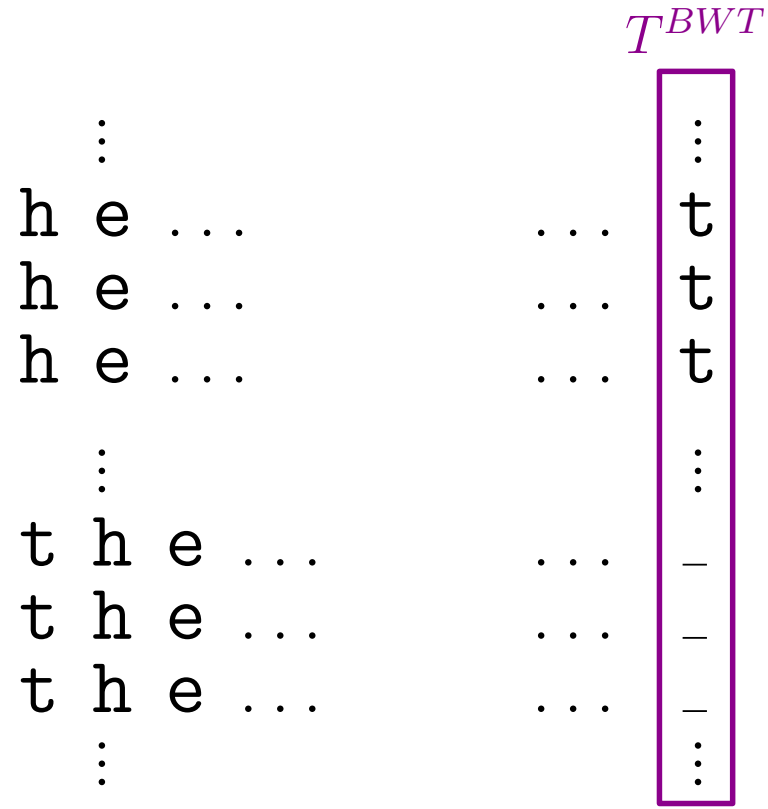
Burrows-Wheeler-Transformation

Was bringt die BWT?

- benötigt gleichen Platz, verwendet gleiche Zeichen wie T
  $\rightarrow$ scheinbar keine Vorteile ?!?

- Permutation einfach umkehrbar
  (benötigt keine Zusatzinformationen, $\mathcal{O}(n)$)
- Zeichen mit ähnlichem Kontext gruppiert
  $\rightarrow$ vereinfacht Komprimierung

- besonders gut auf Texten mit vielen gleichen Substrings
  $\rightarrow$ Beispiel: englischer Text
  (u.a. viele “the”, . . .)



(Außerdem: Vorgänger von Suffixen einfach bestimmbar $\rightarrow$ Indizierung / Suche)

Burrows-Wheeler-Transformation

Kompression

gegeben: Text T
gesucht : komprimierter Text C

bzip2 (1996)

- (Huffmann Kodierung)
- erzeuge *Burrows-Wheeler-Transformation*
- *Move-To-Front* (MTF) Kodierung
- Huffmann Kodierung
  (eigentliche Kompression)

bzip2

Burrows-Wheeler-Transformation

Kompression: *Move-To-Front* (MTF) Kodierung

- nutzt lokale Redundanz
- erzeugt kleine Zahlen für gleiche Zeichen, die nahe beieinander sind

Ablauf

- initialisiere Y mit Alphabet von T^{BWT}
- durchlaufe T^{BWT} $(i = 1..n)$, generiere $R[1..n]$
  - $R[i] =$ Position von $T^{BWT}[i]$ in Y
  - Schiebe $T^{BWT}[i]$ an den Anfang von Y

Burrows-Wheeler-Transformation

Kompression: *Move-To-Front* (MTF) Kodierung

- nutzt lokale Redundanz
- erzeugt kleine Zahlen für gleiche Zeichen, die nahe beieinander sind

Ablauf

- initialisiere Y mit Alphabet von T^{BWT}
- durchlaufe T^{BWT} $(i = 1..n)$, generiere $R[1..n]$
  - $R[i] =$ Position von $T^{BWT}[i]$ in Y
  - Schiebe $T^{BWT}[i]$ an den Anfang von Y

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	a	g	a	$
$R =$									

	1	2	3	4	5
$Y =$	$	a	g	l	n

Burrows-Wheeler-Transformation

Kompression: *Move-To-Front* (MTF) Kodierung

- nutzt lokale Redundanz
- erzeugt kleine Zahlen für gleiche Zeichen, die nahe beieinander sind

Ablauf

- initialisiere Y mit Alphabet von T^{BWT}
- durchlaufe T^{BWT} $(i = 1..n)$, generiere $R[1..n]$
  - $R[i] =$ Position von $T^{BWT}[i]$ in Y
  - Schiebe $T^{BWT}[i]$ an den Anfang von Y

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	a	g	a	$

$R = 3$

	1	2	3	4	5
$Y =$	$	a	g	l	n

Burrows-Wheeler-Transformation

Kompression: *Move-To-Front* (MTF) Kodierung

- nutzt lokale Redundanz
- erzeugt kleine Zahlen für gleiche Zeichen, die nahe beieinander sind

Ablauf

- initialisiere Y mit Alphabet von T^{BWT}
- durchlaufe T^{BWT} $(i = 1..n)$, generiere $R[1..n]$
  - $R[i] =$ Position von $T^{BWT}[i]$ in Y
  - Schiebe $T^{BWT}[i]$ an den Anfang von Y

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	a	g	a	$
$R =$	3								

	1	2	3	4	5
	$	a	g	l	n
$Y =$	g	$	a	l	n

Burrows-Wheeler-Transformation

Kompression: *Move-To-Front* (MTF) Kodierung

- nutzt lokale Redundanz
- erzeugt kleine Zahlen für gleiche Zeichen, die nahe beieinander sind

Ablauf

- initialisiere Y mit Alphabet von T^{BWT}
- durchlaufe T^{BWT} $(i = 1..n)$, generiere $R[1..n]$
  - $R[i] =$ Position von $T^{BWT}[i]$ in Y
  - Schiebe $T^{BWT}[i]$ an den Anfang von Y

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	a	g	a	$
$R =$	3	4							

	1	2	3	4	5
	$	a	g	l	n
$Y =$	g	$	a	l	n

Burrows-Wheeler-Transformation

Kompression: *Move-To-Front* (MTF) Kodierung

- nutzt lokale Redundanz
- erzeugt kleine Zahlen für gleiche Zeichen, die nahe beieinander sind

Ablauf

- initialisiere Y mit Alphabet von T^{BWT}
- durchlaufe T^{BWT} $(i = 1..n)$, generiere $R[1..n]$
  - $R[i]$ = Position von $T^{BWT}[i]$ in Y
  - Schiebe $T^{BWT}[i]$ an den Anfang von Y

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	a	g	a	$
$R =$	3	4							

	1	2	3	4	5
	$	a	g	l	n
	g	$	a	l	n
$Y =$	l	g	$	a	n

Burrows-Wheeler-Transformation

Kompression: *Move-To-Front* (MTF) Kodierung

- nutzt lokale Redundanz
- erzeugt kleine Zahlen für gleiche Zeichen, die nahe beieinander sind

Ablauf

- initialisiere Y mit Alphabet von T^{BWT}
- durchlaufe T^{BWT} $(i = 1..n)$, generiere $R[1..n]$
  - $R[i]$ = Position von $T^{BWT}[i]$ in Y
  - Schiebe $T^{BWT}[i]$ an den Anfang von Y

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	a	g	a	$
$R =$	3	4	1						

	1	2	3	4	5
	$	a	g	l	n
	g	$	a	l	n
$Y =$	l	g	$	a	n

Burrows-Wheeler-Transformation

Kompression: *Move-To-Front* (MTF) Kodierung

- nutzt lokale Redundanz
- erzeugt kleine Zahlen für gleiche Zeichen, die nahe beieinander sind

Ablauf

- initialisiere Y mit Alphabet von T^{BWT}
- durchlaufe T^{BWT} $(i = 1..n)$, generiere $R[1..n]$
  - $R[i] =$ Position von $T^{BWT}[i]$ in Y
  - Schiebe $T^{BWT}[i]$ an den Anfang von Y

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	a	g	a	$
$R =$	3	4	1						

	1	2	3	4	5
	$	a	g	l	n
	g	$	a	l	n
	l	g	$	a	n
$Y =$	l	g	$	a	n

Burrows-Wheeler-Transformation

Kompression: *Move-To-Front* (MTF) Kodierung

- nutzt lokale Redundanz
- erzeugt kleine Zahlen für gleiche Zeichen, die nahe beieinander sind

Ablauf

- initialisiere Y mit Alphabet von T^{BWT}
- durchlaufe T^{BWT} $(i = 1..n)$, generiere $R[1..n]$
  - $R[i]$ = Position von $T^{BWT}[i]$ in Y
  - Schiebe $T^{BWT}[i]$ an den Anfang von Y

$$
\begin{array}{rccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
T^{BWT} = & \mathtt{g} & \mathtt{l} & \mathtt{l} & \mathtt{n} & \mathtt{n} & \mathtt{a} & \mathtt{g} & \mathtt{a} & \mathtt{\$} \\
R = & 3 & 4 & 1 & \ldots & & & & &
\end{array}
$$

$$
\begin{array}{rccccc}
 & 1 & 2 & 3 & 4 & 5 \\
 & \mathtt{\$} & \mathtt{a} & \mathtt{g} & \mathtt{l} & \mathtt{n} \\
 & \mathtt{g} & \mathtt{\$} & \mathtt{a} & \mathtt{l} & \mathtt{n} \\
 & \mathtt{l} & \mathtt{g} & \mathtt{\$} & \mathtt{a} & \mathtt{n} \\
Y = & \mathtt{l} & \mathtt{g} & \mathtt{\$} & \mathtt{a} & \mathtt{n}
\end{array}
$$

Burrows-Wheeler-Transformation

Kompression: *Move-To-Front* (MTF) Kodierung

- nutzt lokale Redundanz
- erzeugt kleine Zahlen für gleiche Zeichen, die nahe beieinander sind

Ablauf

- initialisiere Y mit Alphabet von T^{BWT}
- durchlaufe T^{BWT} $(i = 1..n)$, generiere $R[1..n]$
  - $R[i] =$ Position von $T^{BWT}[i]$ in Y
  - Schiebe $T^{BWT}[i]$ an den Anfang von Y

	1	2	3	4	5	6	7	8	9
$T^{BWT} =$	g	l	l	n	n	a	g	a	$
$R =$	3	4	1	5	1	5	4	2	5

	1	2	3	4	5
	$	a	g	l	n
	g	$	a	l	n
	l	g	$	a	n
	l	g	$	a	n
	n	l	g	$	a
	n	l	g	$	a
	a	n	l	g	$
	g	a	n	l	$
	a	g	n	l	$
$Y =$	$	a	g	n	l

Burrows-Wheeler-Transformation

Kompression: *Move-To-Front* (MTF) Kodierung

- nutzt lokale Redundanz
- erzeugt kleine Zahlen für gleiche Zeichen, die nahe beieinander sind

Ablauf

- initialisiere Y mit Alphabet von T^{BWT}
- durchlaufe T^{BWT} $(i = 1..n)$, generiere $R[1..n]$
  - $R[i] =$ Position von $T^{BWT}[i]$ in Y
  - Schiebe $T^{BWT}[i]$ an den Anfang von Y

	1	2	3	4	5	6	7	8	9	10	11	12	13
$T^{BWT} =$	a	a	a	a	b	b	b	b	c	c	c	c	d
$R =$													

	1	2	3	4
$Y =$	a	b	c	d

Burrows-Wheeler-Transformation

Kompression: *Move-To-Front* (MTF) Kodierung

- nutzt lokale Redundanz
- erzeugt kleine Zahlen für gleiche Zeichen, die nahe beieinander sind

Ablauf

- initialisiere Y mit Alphabet von T^{BWT}
- durchlaufe T^{BWT} $(i = 1..n)$, generiere $R[1..n]$
  - $R[i] =$ Position von $T^{BWT}[i]$ in Y
  - Schiebe $T^{BWT}[i]$ an den Anfang von Y

	1	2	3	4	5	6	7	8	9	10	11	12	13
$T^{BWT} =$	a	a	a	a	b	b	b	b	c	c	c	c	d
$R =$	1	1	1	1	2	1	1	1	3	1	1	1	4

	1	2	3	4
	a	b	c	d
	⋮			
	b	a	c	d
	⋮			
	c	b	a	d
	⋮			
$Y =$	d	c	b	a

Burrows-Wheeler-Transformation

Kompression: Huffmann Kodierung

- präfixfreie Codes variabler Länge
- können greedy konstruiert werden

Ablauf

- erzeuge binären Baum *bottom-up*
  - nimm seltenste 2 Zeichen(-gruppen)
  - erzeuge neuen Knoten, der beide Zeichen(-gruppen) repräsentiert, neue Häufigkeit $\hat{=}$ Summe beider Häufigkeiten
- Beschriftungen der Baumkanten (links:0, rechts:1) ergeben Zeichenkodierungen

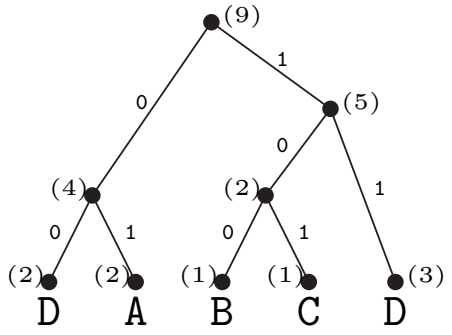
Burrows-Wheeler-Transformation

Kompression: Huffmann Kodierung

- präfixfreie Codes variabler Länge
- können greedy konstruiert werden

Ablauf

- erzeuge binären Baum *bottom-up*
  - nimm seltenste 2 Zeichen(-gruppen)
  - erzeuge neuen Knoten, der beide Zeichen(-gruppen) repräsentiert, neue Häufigkeit $\widehat{=}$ Summe beider Häufigkeiten
- Beschriftungen der Baumkanten (links:0, rechts:1) ergeben Zeichenkodierungen



Burrows-Wheeler-Transformation

Kompression: Huffmann Kodierung

$T =$ l a l a n g n g $

$R =$ 3 4 1 5 1 5 4 2 5

Burrows-Wheeler-Transformation

Kompression: Huffmann Kodierung

$T =$ l a l a n g n g $

$R =$ 3 4 1 5 1 5 4 2 5

Symbol	Häufigkeit
1	2
2	1
3	1
4	2
5	3

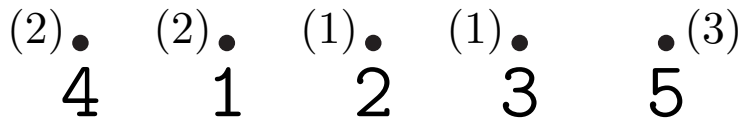
Burrows-Wheeler-Transformation

Kompression: Huffmann Kodierung

$T =$ l a l a n g n g $

$R =$ 3 4 1 5 1 5 4 2 5

Symbol	Häufigkeit
1	2
2	1
3	1
4	2
5	3



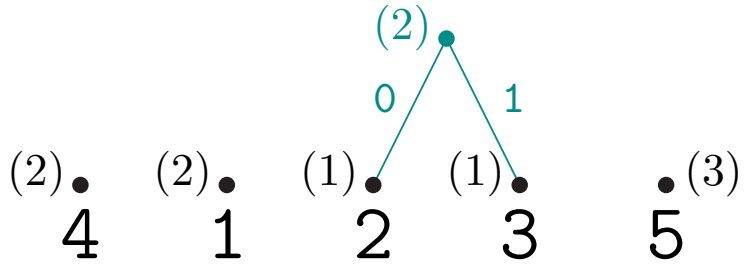
Burrows-Wheeler-Transformation

Kompression: Huffmann Kodierung

$T =$ l a l a n g n g $

$R =$ 3 4 1 5 1 5 4 2 5

Symbol	Häufigkeit
1	2
2	1
3	1
4	2
5	3



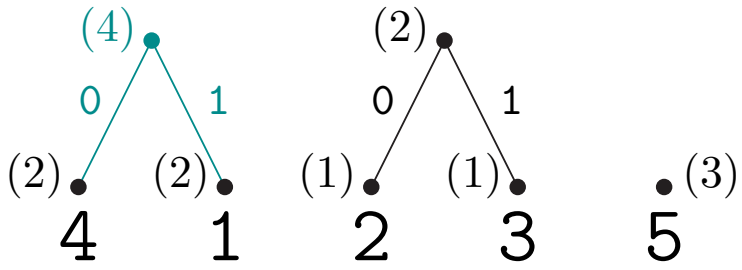
Burrows-Wheeler-Transformation

Kompression: Huffmann Kodierung

$T =$ l a l a n g n g $

$R =$ 3 4 1 5 1 5 4 2 5

Symbol	Häufigkeit
1	2
2	1
3	1
4	2
5	3

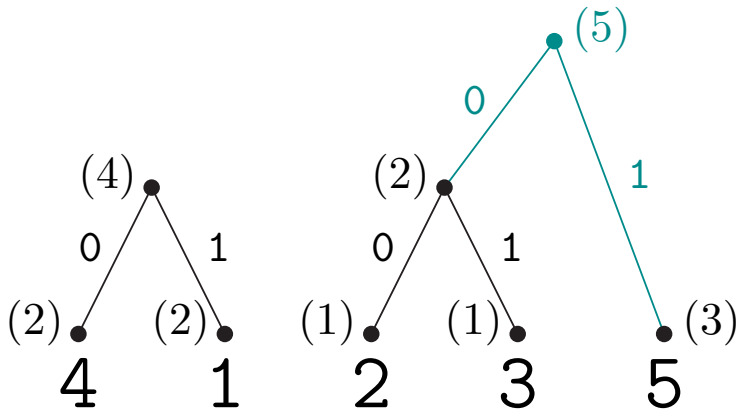


Burrows-Wheeler-Transformation

Kompression: Huffmann Kodierung

$T =$ l a l a n g n g $
$R =$ 3 4 1 5 1 5 4 2 5

Symbol	Häufigkeit
1	2
2	1
3	1
4	2
5	3



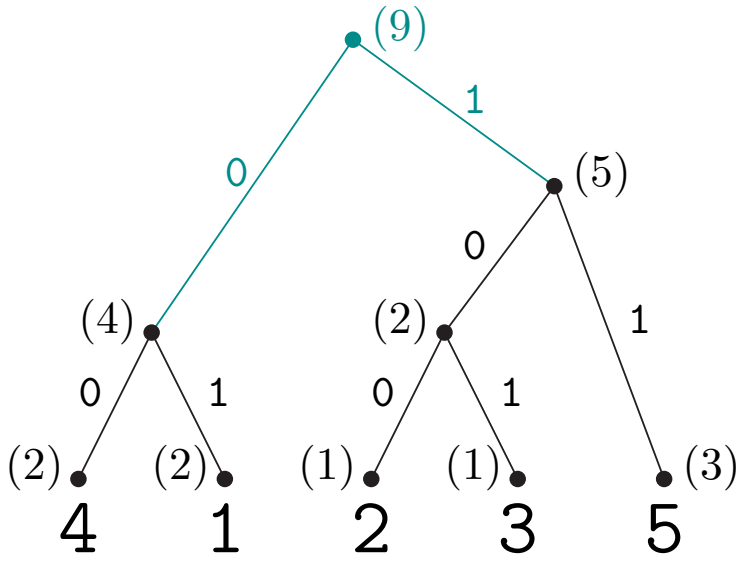
Burrows-Wheeler-Transformation

Kompression: Huffmann Kodierung

$T =$ l a l a n g n g $

$R =$ 3 4 1 5 1 5 4 2 5

Symbol	Häufigkeit
1	2
2	1
3	1
4	2
5	3



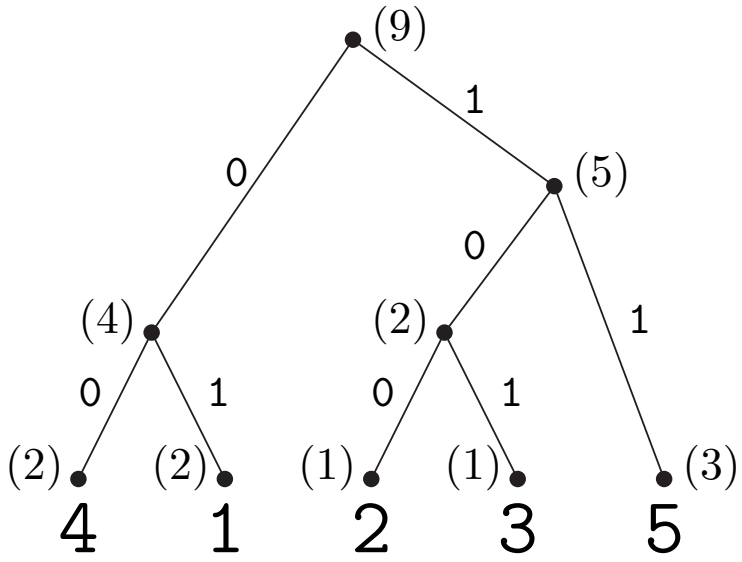
Burrows-Wheeler-Transformation

Kompression: Huffmann Kodierung

$T =$ l a l a n g n g $

$R =$ 3 4 1 5 1 5 4 2 5

Symbol	Häufigkeit	Code
1	2	01
2	1	100
3	1	101
4	2	00
5	3	11



Burrows-Wheeler-Transformation

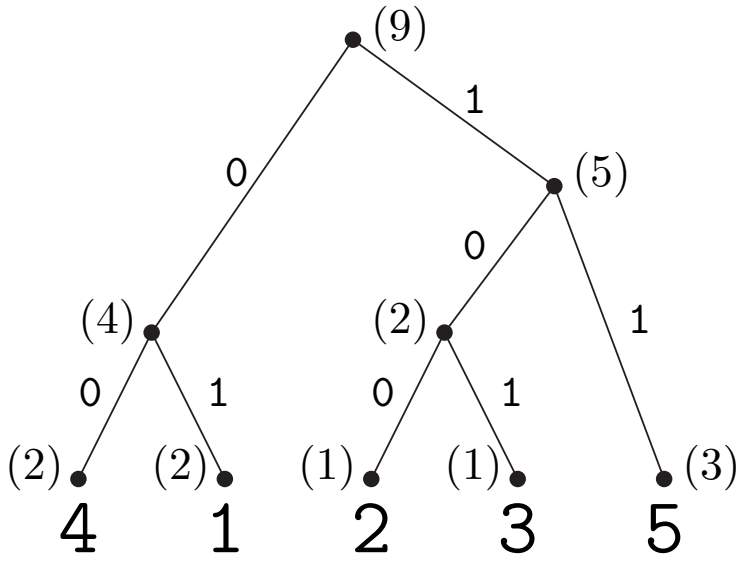
Kompression: Huffmann Kodierung

$T =$ l a l a n g n g $

$R =$ 3 4 1 5 1 5 4 2 5

101 00 01 11 01 11 00 100 11 20 Bits

Symbol	Häufigkeit	Code
1	2	01
2	1	100
3	1	101
4	2	00
5	3	11



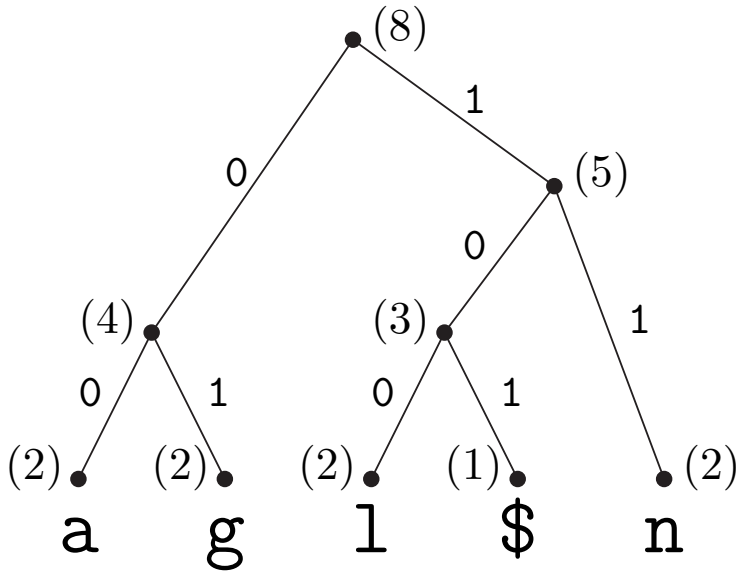
Burrows-Wheeler-Transformation

Kompression: Huffmann Kodierung

$T =$ l a l a n g n g $

100 00 100 00 11 01 11 01 101 21 Bits

Symbol	Häufigkeit	Code
$	1	101
a	2	00
g	2	01
l	2	100
n	2	11



Burrows-Wheeler-Transformation

Zusammenfassung

- erzeugt (sinnvolle) Permutation der Eingabe
  (gruppiert Zeichen mit ähnlichem Kontext nahe beieinander)
- keine Zusatzinformation für Rücktransformation nötig
  (alle Informationen in Struktur der Permutation)
- Hin- und Rücktransformation in $\mathcal{O}(n)$
  (einfache Papier-und-Bleistift-Methode existiert auch)
- Vorverarbeitung (statischer) Texte
  (Komprimierung, Indizierung, Suche)

Suche in der Burrwos-Wheeler-Transformation

Karlsruher Institut für Technologie

Ferragina &Manzini (2000)

- Index basierend auf der Burrows-Wheeler-Transformation (BWT)
- Vergleich des Musters von rechts nach links
- Zeitkomplexität: $\mathcal{O}(m \log \sigma)$

BWT

- $BWT[i] = \mathcal{T}[SA[i] - 1 \bmod n]$
- Unkomprimierte Größe: $n \log \sigma$ Bits
- Komprimierte Größe: $nH_k(\mathcal{T})$ Bits (+Kontextinformation)

Backward Search

i	$SA[i]$	BWT	$\mathcal{T}[SA[i]..n-1]$
0	18	a	\$
1	17	r	a\$
2	10	r	abarbara\$
3	7	d	abrabarbara\$
4	0	\$	abracadabrabarbara\$
5	3	r	acadabrabarbara\$
6	5	c	adabrabarbara\$
7	15	b	ara\$
8	12	b	arbara\$
9	14	r	bara\$
10	11	a	barbara\$
11	8	a	brabarbara\$
12	1	a	bracadabrabarbara\$
13	4	a	cadabrabarbara\$
14	6	a	dabrabarbara\$
15	16	a	ra\$
16	9	b	rabarbara\$
17	2	b	racadabrabarbara\$
18	13	a	rbara\$

- $BWT[i] = \mathcal{T}[SA[i]-1]$, for $SA[i] > 0$
- $BWT[i] = \mathcal{T}[n-1]$, for $SA[i] = 0$
- I.e. $BWT[i]$ is the character preceding suffix $SA[i]$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

Array C contains for each $c \in \Sigma$ the position of the first suffix in SA which starts with c:

$	a	b	c	d	r	r+1
0	1	9	13	14	15	19

- Operation $rank(i, X, BWT)$ returns how often character $X \in \Sigma$ occurs in the prefix $BWT[0..i-1]$.
- Example: search for $\mathcal{P} = bar$.

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Initial interval: $[sp_0, ep_0] = [0..n-1]$
- Determine interval for r:
  $sp_1 = C[r] + rank(sp_0, r, BWT)$
  $ep_1 = C[r] + rank(ep_0 + 1, r, BWT) - 1$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Initial interval: $[sp_0, ep_0] = [0..n-1]$
- Determine interval for r:
  $sp_1 = 15 + rank(0, r, BWT)$
  $ep_1 = 15 + rank(19, r, BWT)$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Initial interval: $[sp_0, ep_0] = [0..n-1]$
- Determine interval for r:
  $sp_1 = 15 + 0$
  $ep_1 = 15 + rank(19, r, BWT) - 1$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C

$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Initial interval: $[sp_0, ep_0] = [0..n-1]$
- Determine interval for r:
  $sp_1 = 15+0 = 15$
  $ep_1 = 15+4-1 = 18$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C

$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_1, ep_1] = [15..18]$
- Determine interval for *ar*:
  $sp_2 = C[a] + rank(sp_1 - 1, a, BWT)$
  $ep_2 = C[a] + rank(ep_1, a, BWT) - 1$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C

$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_1, ep_1] = [15..18]$
- Determine interval for *ar*:
  $sp_2 = 1 + rank(15, a, BWT)$
  $ep_2 = 1 + rank(ep_1, a, BWT)$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_1, ep_1] = [15..18]$
- Determine interval for *ar*:
  $sp_2 = 1 + rank(15, a, BWT)$
  $ep_2 = 1 + rank(ep_1, a, BWT)$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_1, ep_1] = [15..18]$
- Determine interval for *ar*:
  $sp_2 = 1 + 6$
  $ep_2 = 1 + rank(19, a, BWT) - 1$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C

$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_1, ep_1] = [15..18]$
- Determine interval for *ar*:
  $sp_2 = 1+6 = 7$
  $ep_2 = 1+8-1 = 8$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C

$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_2, ep_2] = [7..8]$
- Determine interval for *bar*:
  $sp_3 = C[b] + rank(sp_2, b, BWT)$
  $ep_3 = C[b] + rank(ep_2 + 1, b, BWT) - 1$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_2, ep_2] = [7..8]$
- Determine interval for *bar*:
  $sp_3 = 9 + rank(7, b, BWT)$
  $ep_3 = 9 + rank(ep_1, b, BWT)$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C

$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_2, ep_2] = [7..8]$
- Determine interval for *bar*:
  $sp_3 = 9 + rank(7, b, BWT)$
  $ep_3 = 9 + rank(ep_1, b, BWT)$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_2, ep_2] = [7..8]$
- Determine interval for *bar*:
  $sp_3 = 9 + 0$
  $ep_3 = 9 + rank(9, b, BWT) - 1$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_2, ep_2] = [7..8]$
- Determine interval for *bar*:
  $sp_3 = 9+0 = 9$
  $ep_3 = 9+2-1 = 10$

Backward Search

Summary

- Only C and a data structure R supporting the *rank* operation on *BWT* are required for existence and count queries.
- Space: $\sigma \log n$ bits for C + space for R
- Time: $\mathcal{O}(m \cdot t_{rank})$, where t_{rank} is time for one rank operation. Independent from n?
- Next: How to implement *rank*?

Rank operation

- Constant time and $o(n)$ extra space solution on bitvectors (Jacobson 1989)
- Solution on general sequences: Wavelet Tree (Grossi & Vitter 2003)

Backward Search

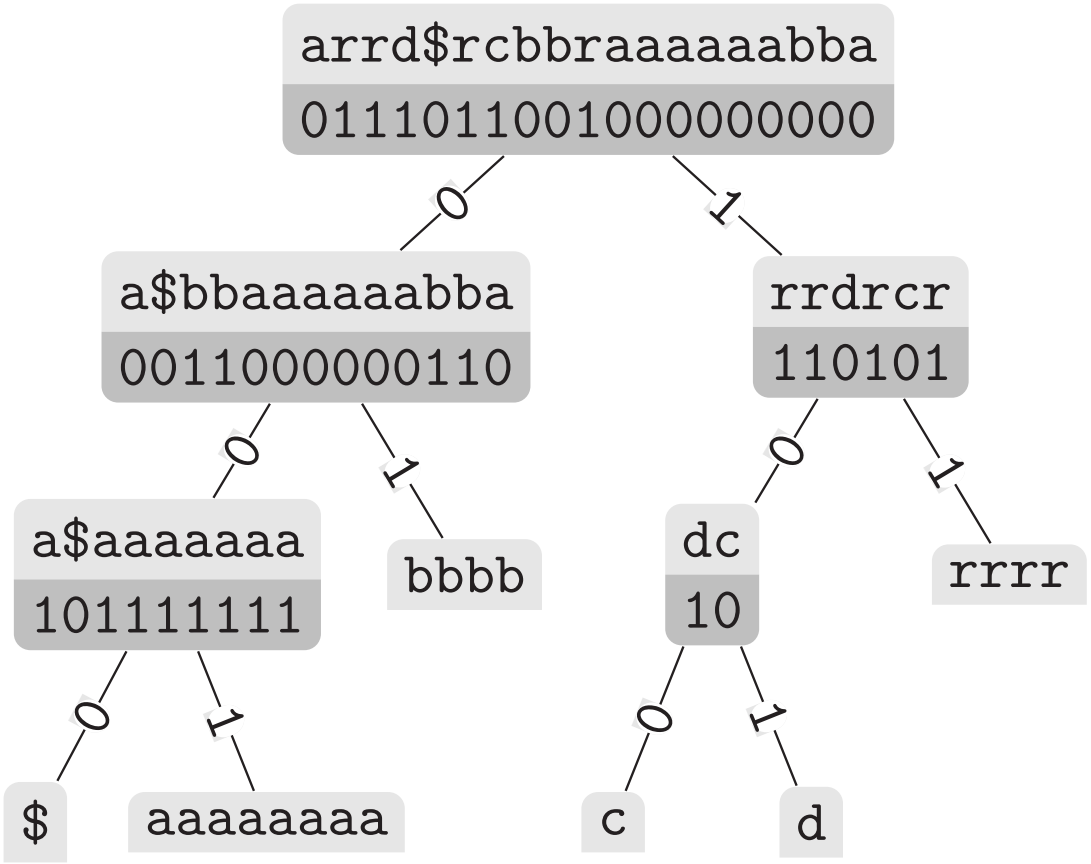
Summary

- Only C and a data structure R supporting the *rank* operation on *BWT* are required for existence and count queries.
- Space: $\sigma \log n$ bits for C + space for R
- Time: $\mathcal{O}(m \cdot t_{rank})$, where t_{rank} is time for one rank operation. Independent from n? If t_{rank} is independent from n
- Next: How to implement *rank*?

Rank operation

- Constant time and $o(n)$ extra space solution on bitvectors (Jacobson 1989)
- Solution on general sequences: Wavelet Tree (Grossi & Vitter 2003)

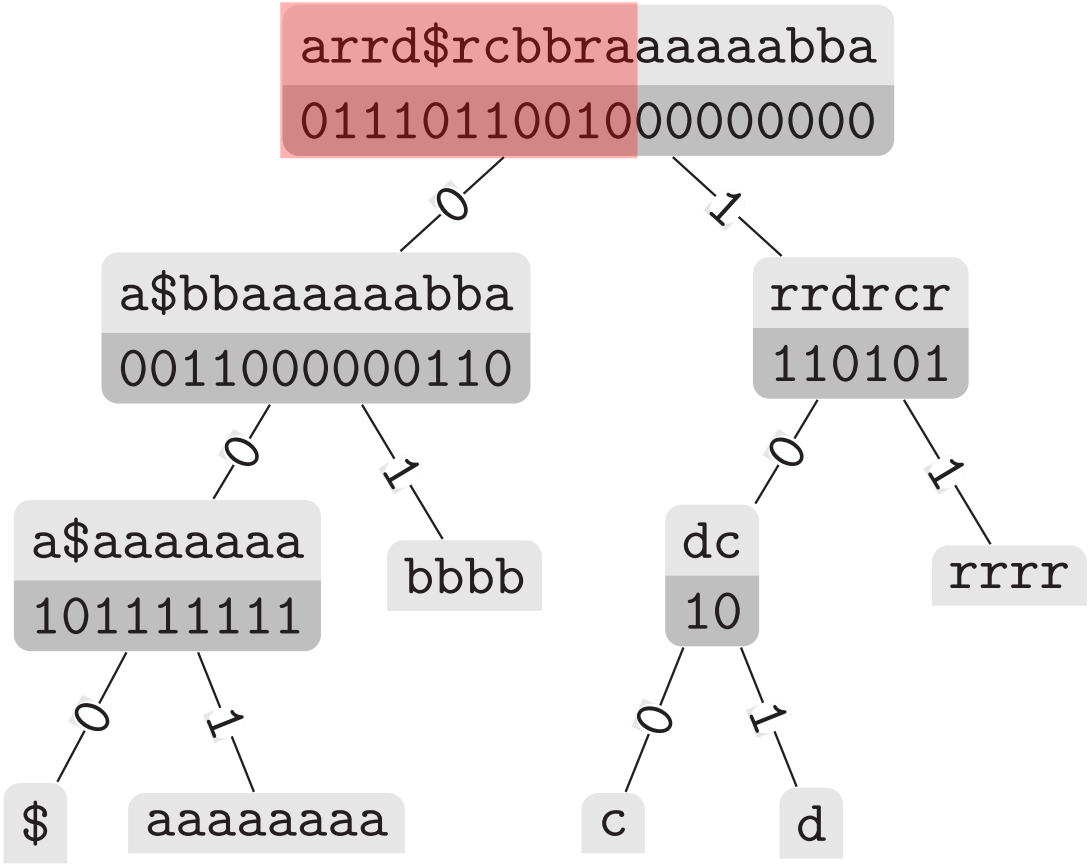
Wavelet Tree Example: Calculate Rank



a = 001

$$rank(11, a, WT) = rank(rank(rank(11, 0, b_\epsilon) = 5, 0, b_0) = 3, 1, b_{00}) = 2$$

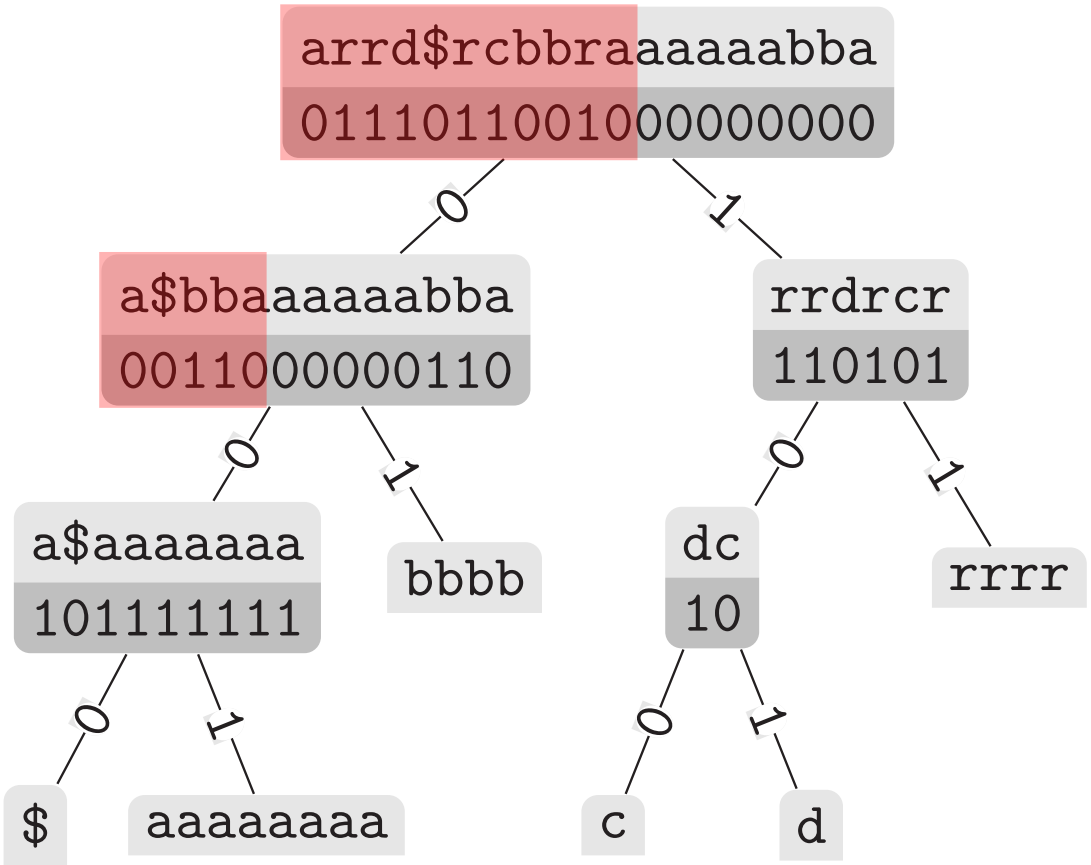
Wavelet Tree Example: Calculate Rank



a = 001

$$rank(11, a, WT) = rank(rank(rank(11, 0, b_\epsilon) = 5, 0, b_0) = 3, 1, b_{00}) = 2$$

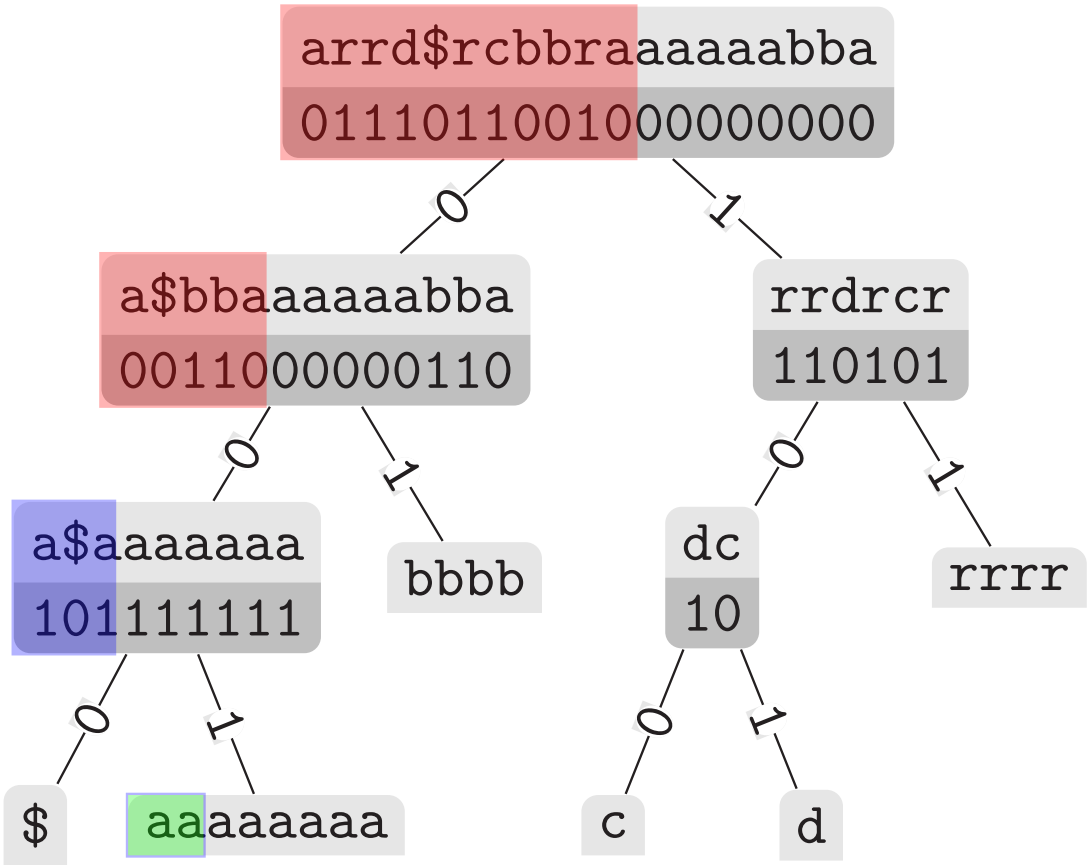
Wavelet Tree Example: Calculate Rank



a = 001

$$rank(11, a, WT) = rank(rank(rank(11, 0, b_\epsilon) = 5, 0, b_0) = 3, 1, b_{00}) = 2$$

Wavelet Tree Example: Calculate Rank



a = 001

$$rank(11, a, WT) = rank(rank(rank(11, 0, b_\epsilon) = 5, 0, b_0) = 3, 1, b_{00}) = 2$$

$o(n)$ space / constant query time

- Given bitvector B of length n bits
- Divide B into superblocks of size L
- For each superblock SB_j store $\Sigma_{i=0}^{(j-1)L-1} B[i]$ in $\log n$ bits
- Divide each superblock into blocks of size S
- For each block B_k of superblock j store $\Sigma_{i=(j-1)L}^{(j-1)L+kS-1} B[i]$ in $\log L$ bits

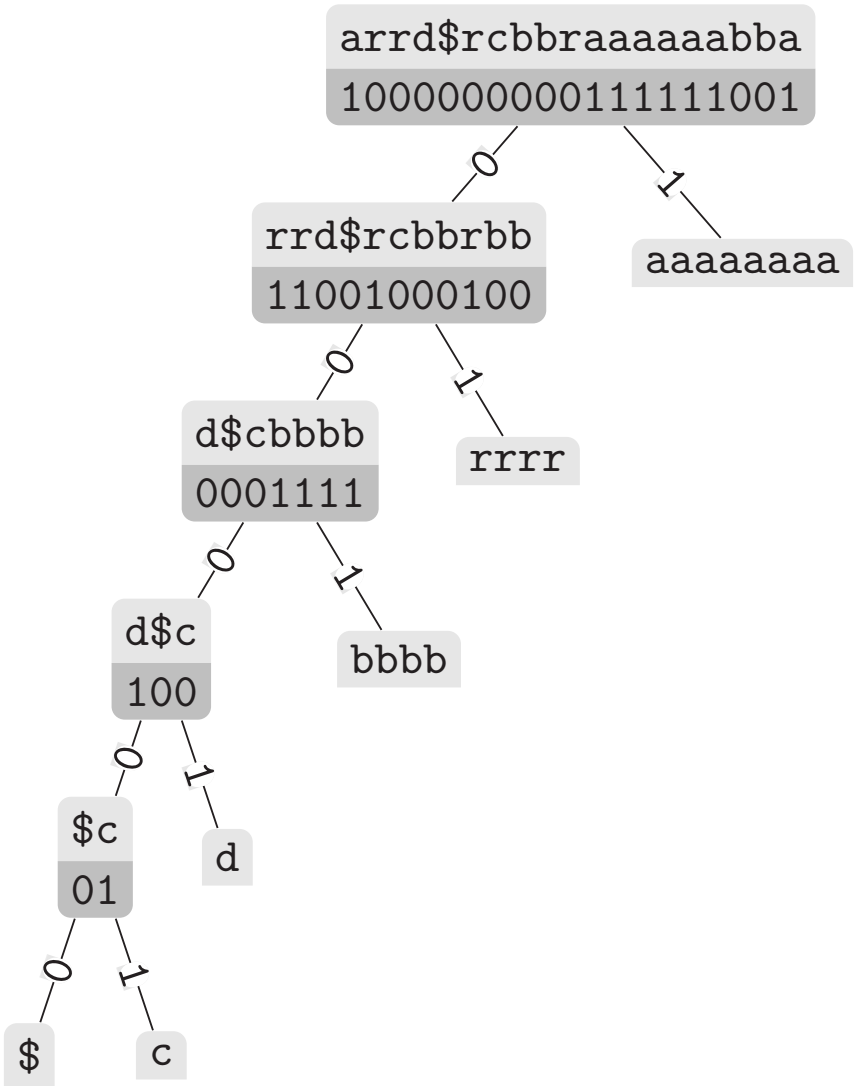
If blocks are small enough, we can pre-compute a table which stores all answers for every block and position (four Russian-Tick).

Solution

- $L = \log^2 n$
- $S = \frac{1}{2} \log n$

Final space: $\mathcal{O}(\frac{n}{\log n} + \frac{n \log\log n}{\log n} + \sqrt{n} \log n \log\log n)$ bits

Huffman-shaped Wavelet Tree



Char c	$codeword(c)$
$	00000
a	1
b	001
c	00001
d	0001
r	01

Avg. depth: $H_0(BWT)$. Total space: $\approx nH_0 + 2\sigma \log n$ bits

Practical Performance of FM-Index

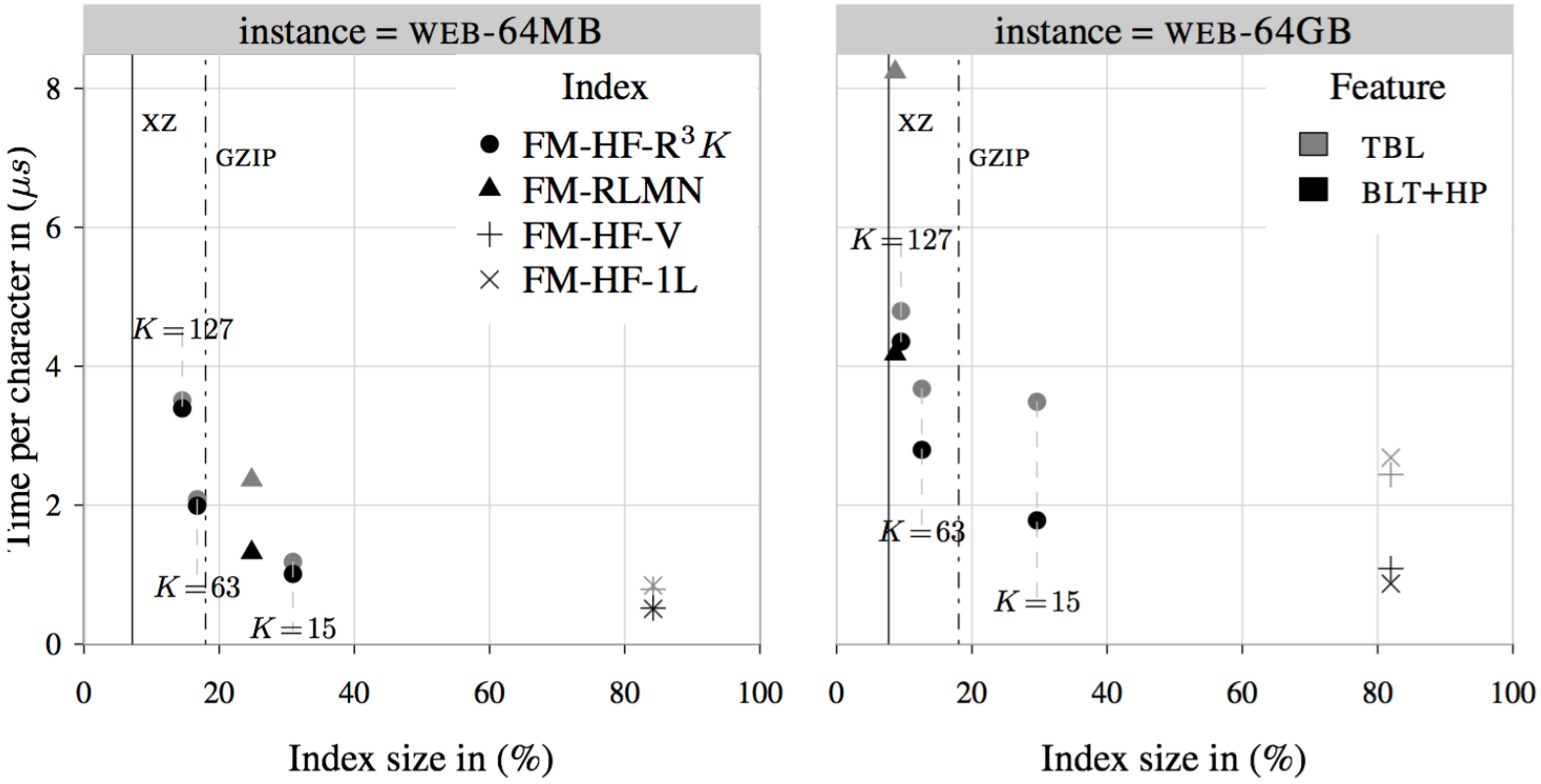


Figure 10. Count time and space of our index implementations on input instances of different size with compression effectiveness baselines using standard compression utilities XZ and GZIP with option `--best`.

Succinct Data Structures

Karlsruher Institut für Technologie

Definition

A *succinct data structure* uses space „close” to the information-theoretical lower bound, but still supports operations time-efficiently.

Let L be the information-theoretical lower bound to represent a class of objects. Then a data structure which still supports time-efficient operations is called

- *implicit*, if it takes $L + O(1)$ bits of space
- *succinct*, if it takes $L + o(L)$ bits of space
- *compact*, if it takes $O(L)$ bits of space

Succinct representation of trees

- First consider binary trees
- Number of n-node binary trees: $C_n = \frac{1}{n+1}\binom{2n}{n}$
- We need $\log C_n = 2n + o(n)$ bits (using Sterling's Approximation)
- Operations: $\mathit{parent}(v)$, $\mathit{leftchild}(v)$, $\mathit{rightchild}(v)$

Succinct representation of trees

- First consider binary trees
- Number of n-node binary trees: $C_n = \frac{1}{n+1}\binom{2n}{n}$
- We need $\log C_n = 2n + o(n)$ bits (using Sterling's Approximation)
- Operations: $\mathit{parent}(v)$, $\mathit{leftchild}(v)$, $\mathit{rightchild}(v)$

Succinct representation of trees

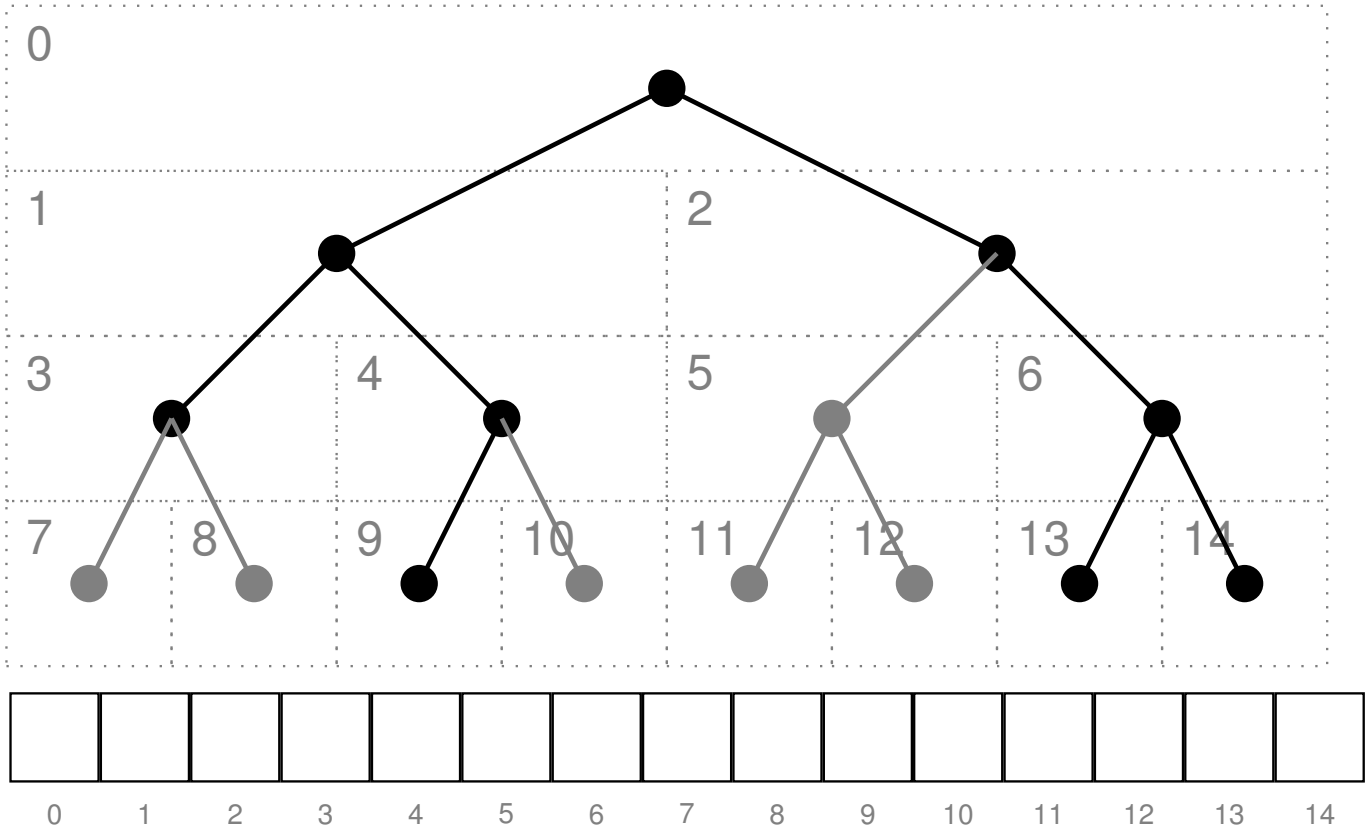
- First consider binary trees
- Number of n-node binary trees: $C_n = \frac{1}{n+1}\binom{2n}{n}$
- We need $\log C_n = 2n + o(n)$ bits (using Sterling's Approximation)
- Operations: $\mathit{parent}(v)$, $\mathit{leftchild}(v)$, $\mathit{rightchild}(v)$

Succinct representation of trees

- First consider binary trees
- Number of n-node binary trees: $C_n = \frac{1}{n+1}\binom{2n}{n}$
- We need $\log C_n = 2n + o(n)$ bits (using Sterling's Approximation)
- Operations: $\mathit{parent}(v)$, $\mathit{leftchild}(v)$, $\mathit{rightchild}(v)$

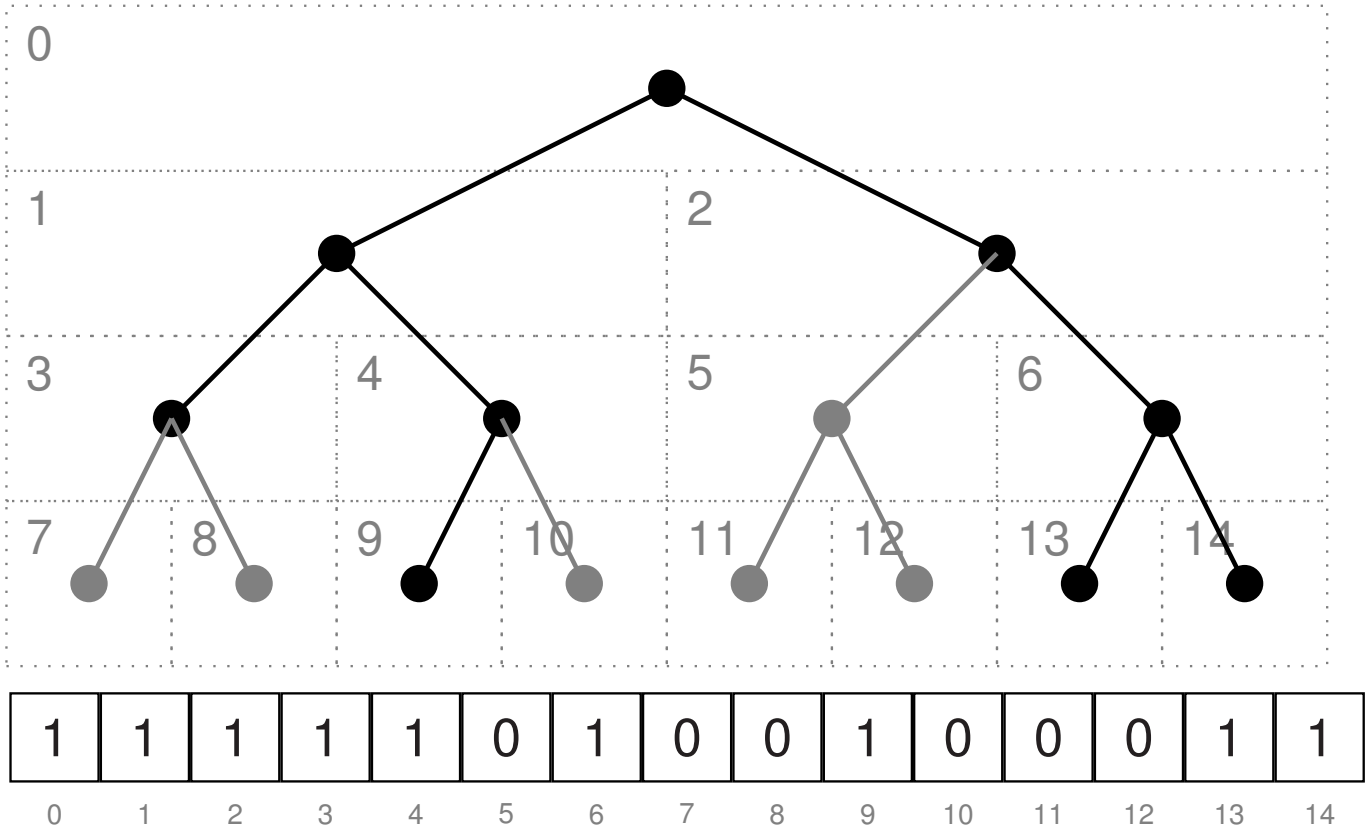
Succinct representation of trees

- First consider binary trees
- Number of n-node binary trees: $C_n = \frac{1}{n+1}\binom{2n}{n}$
- We need $\log C_n = 2n + o(n)$ bits (using Sterling's Approximation)
- Operations: $parent(v)$, $leftchild(v)$, $rightchild(v)$



Succinct representation of trees

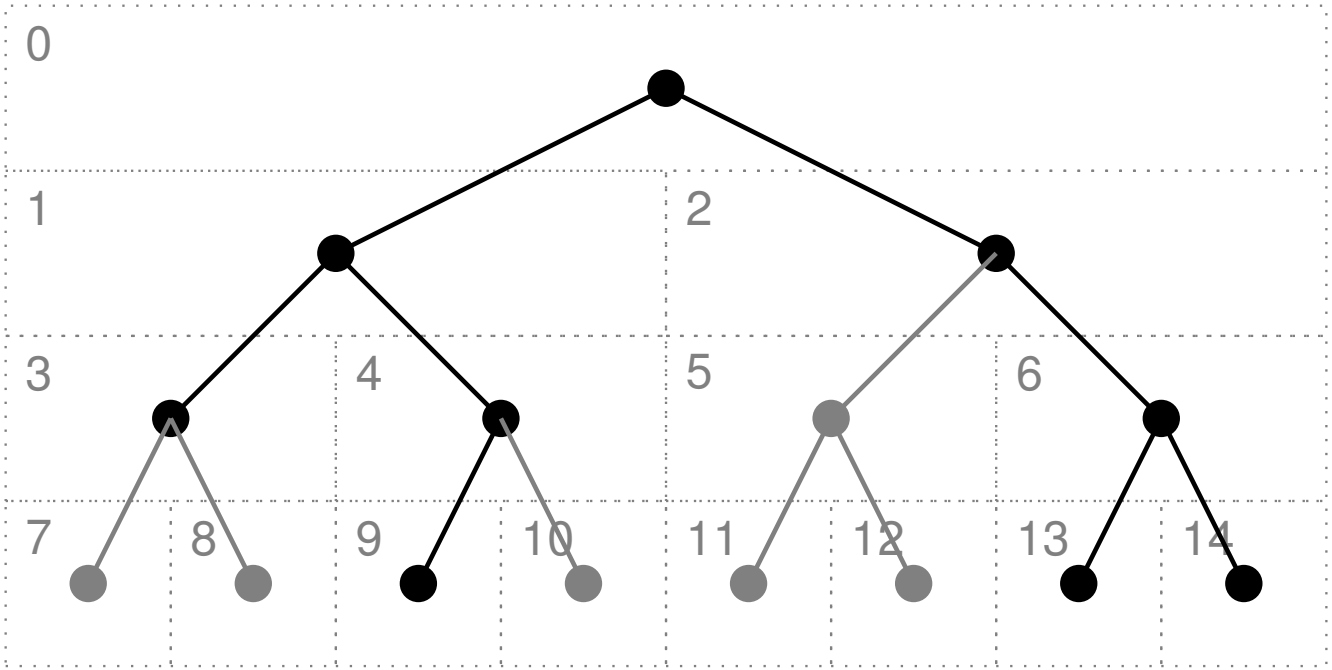
- First consider binary trees
- Number of n-node binary trees: $C_n = \frac{1}{n+1}\binom{2n}{n}$
- We need $\log C_n = 2n + o(n)$ bits (using Sterling's Approximation)
- Operations: $parent(v)$, $leftchild(v)$, $rightchild(v)$



Succinct representation of trees

- In a very balanced binary tree (like in a heap) operations are easy
- Let 0 be the root identifier
  - $\mathit{parent}(v) = \lfloor \frac{v-1}{2} \rfloor$ for $v > 0$
  - $\mathit{leftchild}(v) = 2v + 1$
  - $\mathit{rightchild}(v) = 2v + 2$

Child operation in detail

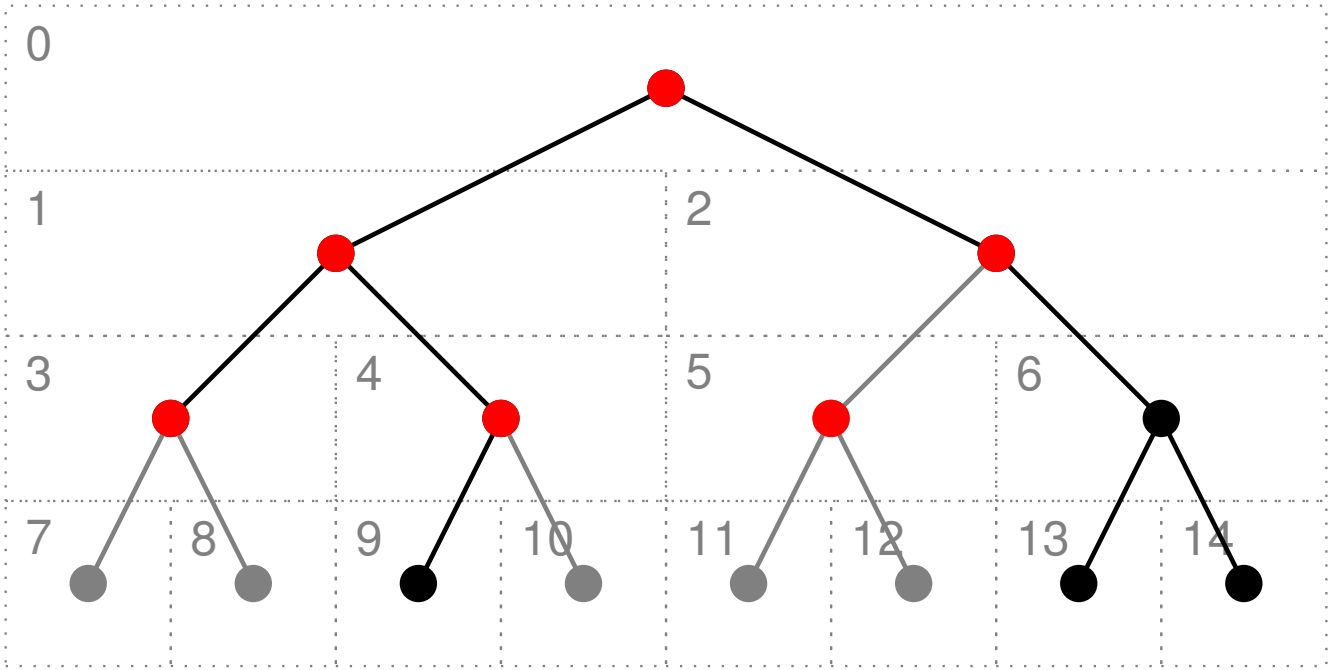


w=leftchild(v)

- Mark $P(v)$
- $P(w) = \{$children of $P(v)\} \cup$
- w is at position $|P(w)| = 2P(v) + 1 = 2v + 1$

- Let $\delta(v)$ be the distance of a node v to the root node
- $P(v) := \{w \mid \delta(w) \leq \delta(v) \wedge w < v\}$
- Note: $|P(v)| = v$

Child operation in detail

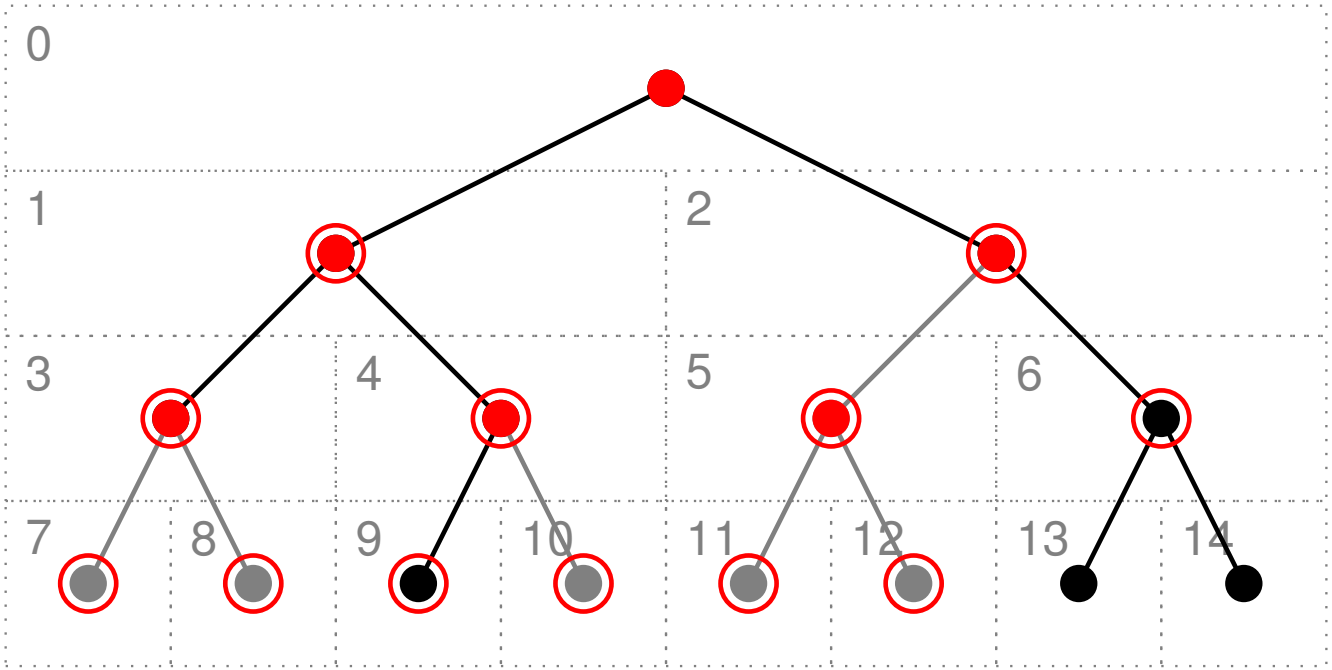


w=leftchild(v)

- Mark $P(v)$
- $P(w) = \{$children of $P(v)\} \cup$
- w is at position $|P(w)| = 2P(v) + 1 = 2v + 1$

- Let $\delta(v)$ be the distance of a node v to the root node
- $P(v) := \{w \mid \delta(w) \leq \delta(v) \wedge w < v\}$
- Note: $|P(v)| = v$

Child operation in detail

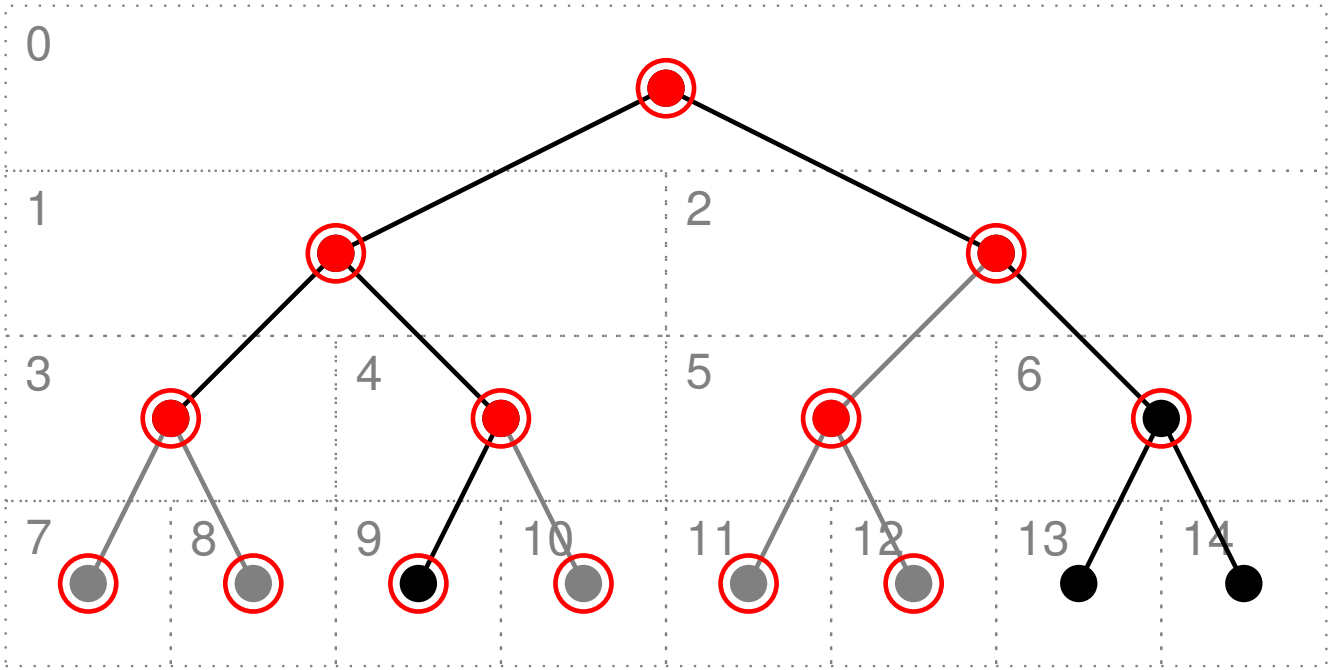


w=leftchild(v)

- Mark $P(v)$
- $P(w) = \{$children of $P(v)\} \cup$
- w is at position $|P(w)| = 2P(v) + 1 = 2v + 1$

- Let $\delta(v)$ be the distance of a node v to the root node
- $P(v) := \{w \mid \delta(w) \leq \delta(v) \wedge w < v\}$
- Note: $|P(v)| = v$

Child operation in detail

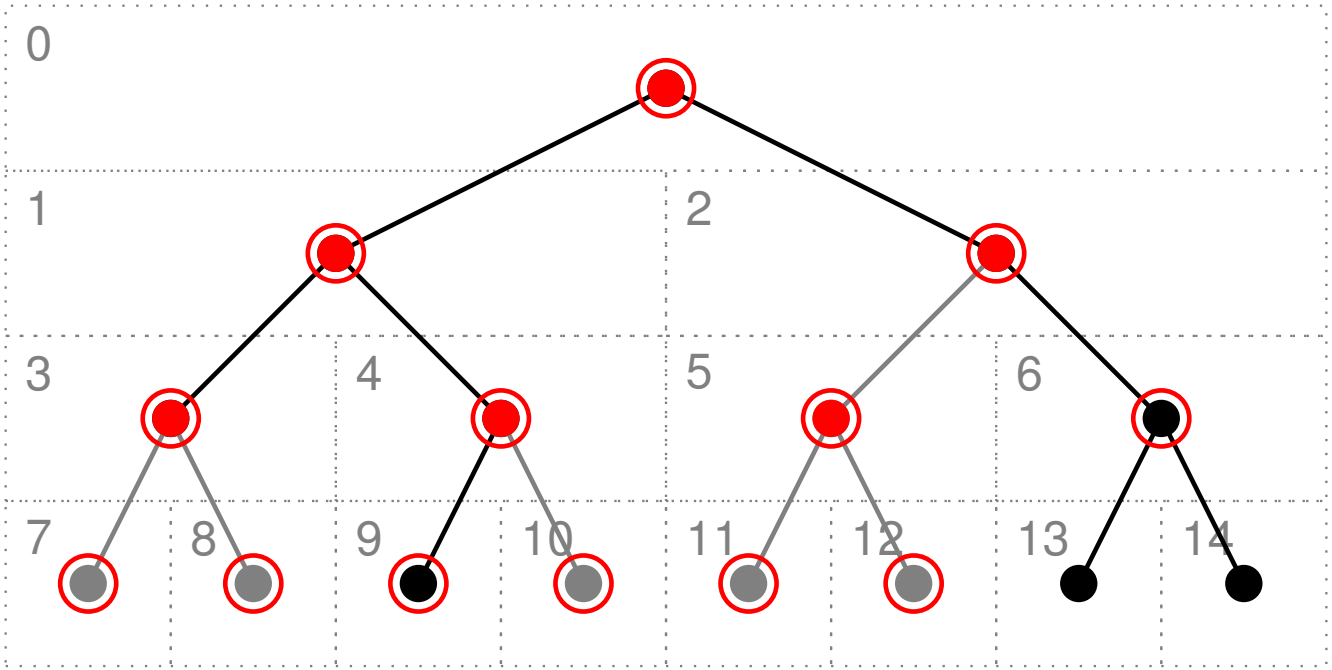


w=leftchild(v)

- Mark $P(v)$
- $P(w) = \{$children of $P(v)\} \cup \{root\}$
- w is at position $|P(w)| = 2P(v) + 1 = 2v + 1$

- Let $\delta(v)$ be the distance of a node v to the root node
- $P(v) := \{w \mid \delta(w) \leq \delta(v) \wedge w < v\}$
- Note: $|P(v)| = v$

Child operation in detail



w=leftchild(v)

- Mark $P(v)$
- $P(w) = \{$children of $P(v)\} \cup \{root\}$
- w is at position $|P(w)| = 2P(v) + 1 = 2v + 1$

- Let $\delta(v)$ be the distance of a node v to the root node
- $P(v) := \{w \mid \delta(w) \leq \delta(v) \wedge w < v\}$
- Note: $|P(v)| = v$

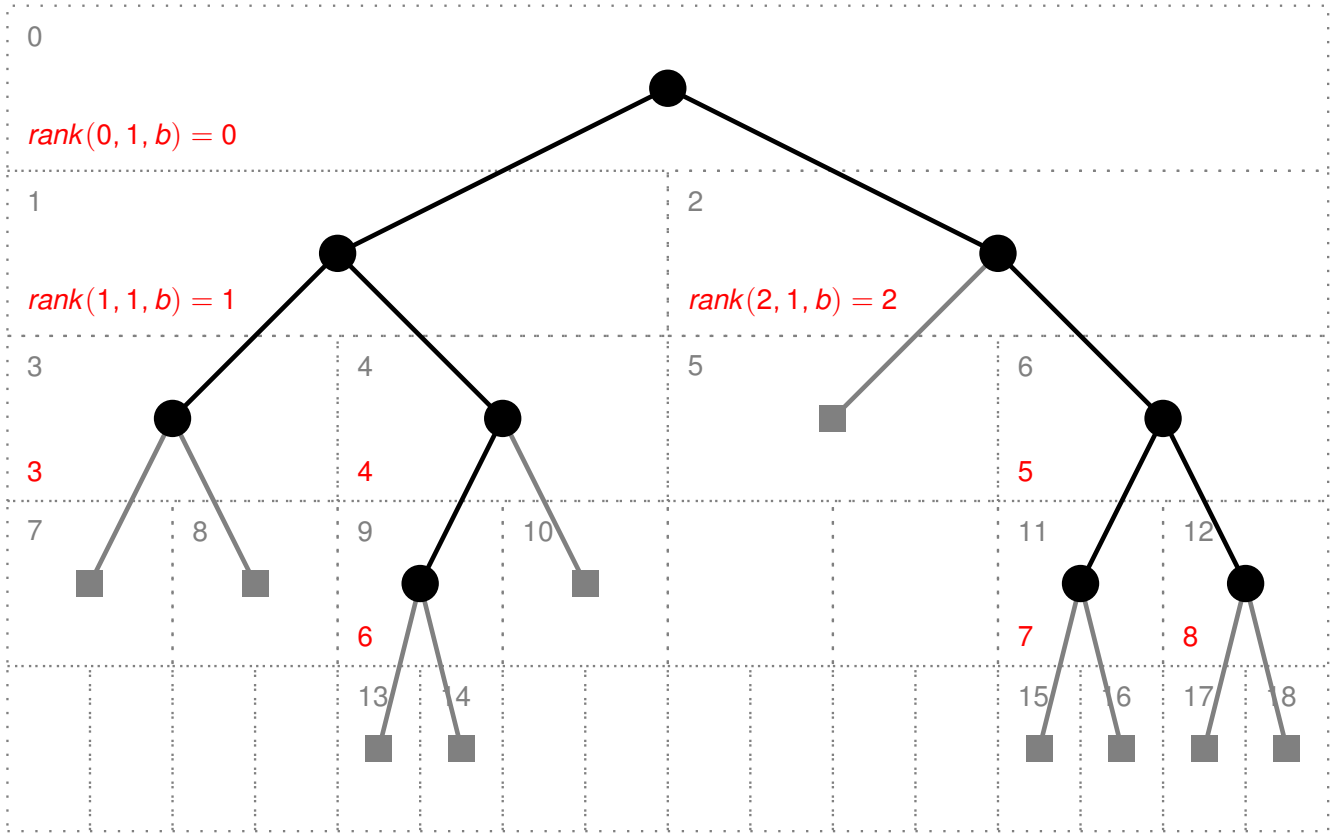
Succinct representation of trees

Probelm with previous approach:
For unbalanced trees the space would be 2^d bits, where d is the maximum depth of a node.

Proposal of Jacobson (FOCS 1989):

1. Mark all the nodes of the tree with a 1.
2. Add external nodes to the tree, and mark them all with 0-bits.
3. Read off the bits marking the nodes of the tree in (left-to-right) level-order.

Succinct representation of trees



$b =$

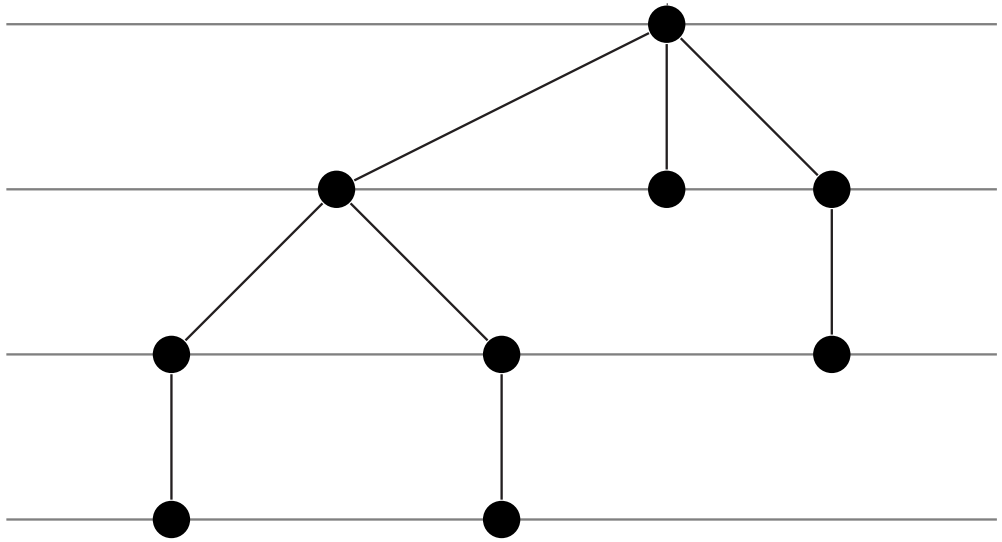
1	1	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Succinct representation of trees

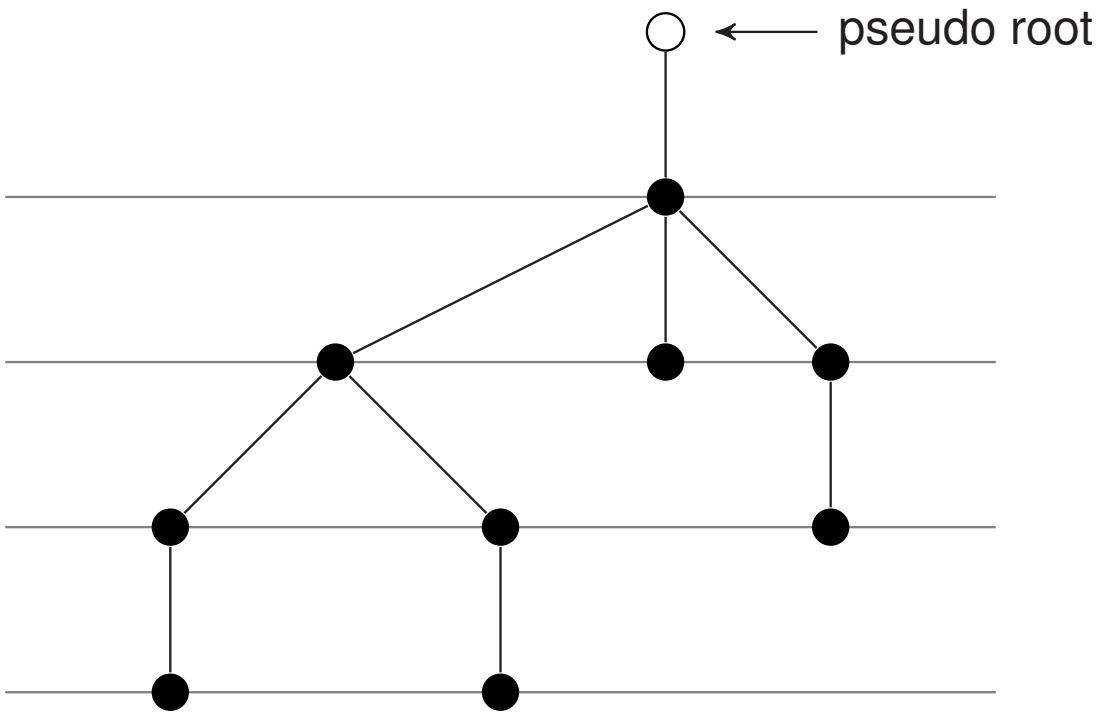
- b contains n set bits and is of length $2n + 1$.
- A node is represented by the positon of its corresponing 1-bit in b.
  - $leftchild(v) = 2 \cdot rank(v) + 1$
  - $rightchild(v) = 2 \cdot rank(v) + 2$
  - $parent(v)$ also possible in constant time (homework)
- Total space (including rank and select): $2n + o(n)$ bits

Jacobson also considered rooted, ordered tree with degree higher than 2.

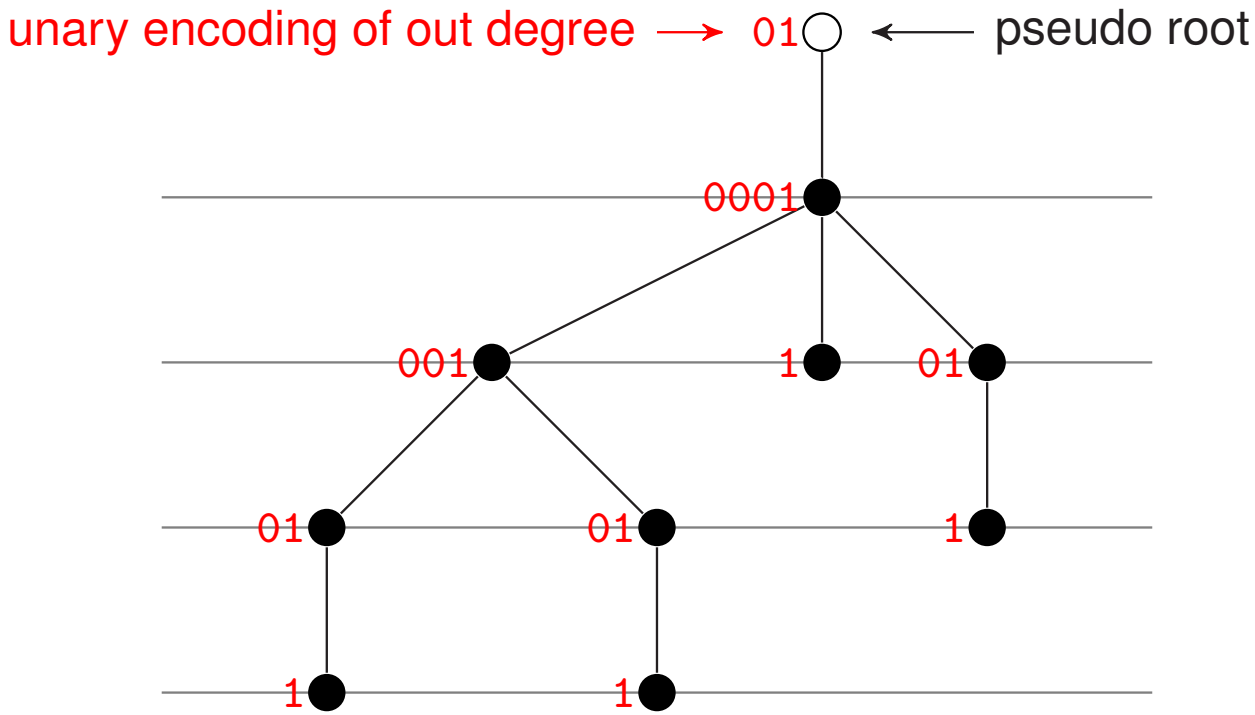
LOUDS – level order unary degree sequence



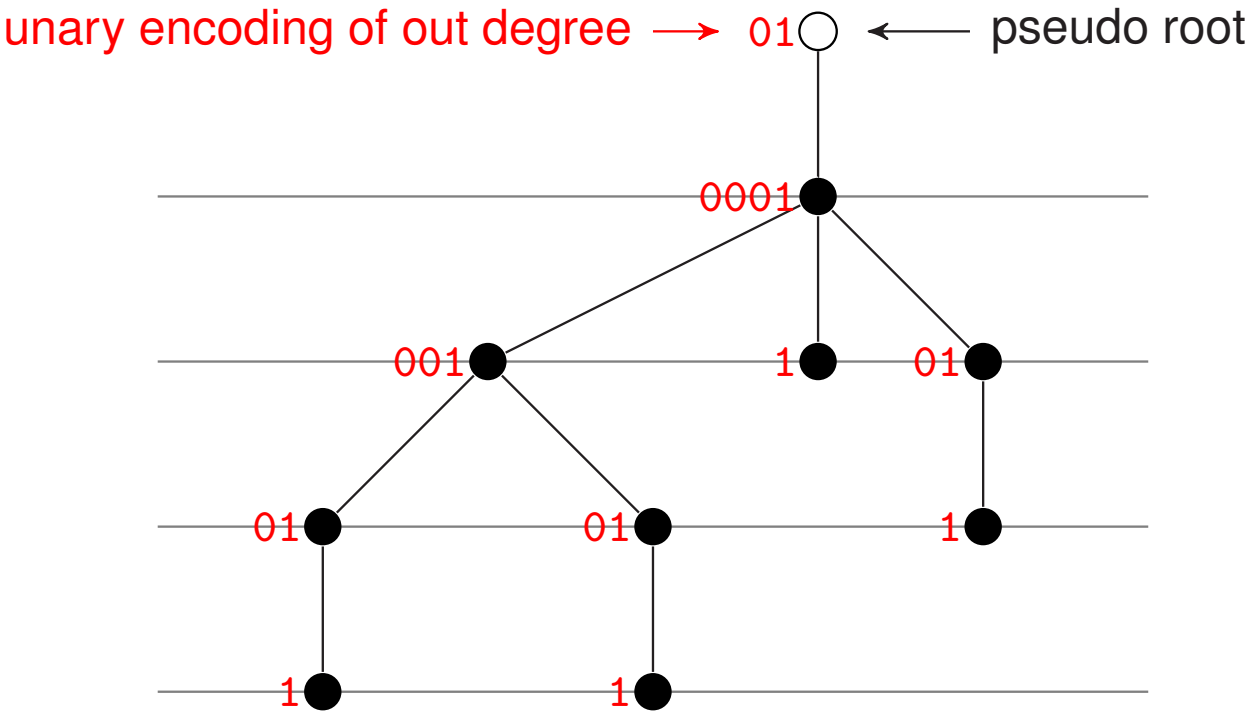
LOUDS – level order unary degree sequence



LOUDS – level order unary degree sequence



LOUDS – level order unary degree sequence



LOUDS sequence = 01000100110101011111
(concatenation of unary codes in level order)

LOUDS – level order unary degree sequence

- Each node (excpet the pseudo root) is represented twice
  - Once as „0” in the child list of its parent
  - Once as the terminal („1”) in its child list
- Represent node *v* by the index of its corresponding „0”
- I.e. *root* corresponds to „0”

LOUDS – level order unary degree sequence

Karlsruher Institut für Technologie

```
00  is_leaf(v)
01      id ← rank(v, 0, LOUDS)
02      p ← select(id + 2, 1, LOUDS)
03      if LOUDS[p − 1] = 1 then
04          return true
05      return false

00  out_degree(v)
01      if is_leaf(v) then
02          return 0
03      id ← rank(v, 0, LOUDS)
04      return select(id + 2, 1, LOUDS) − select(id + 1, 1, LOUDS) − 1
```

LOUDS – level order unary degree sequence

Get i-th child ($i \in [1, out\_degree(v)]$) of v and parent:

```
00  child(v,i)
01      if i > out_degree(v) then
02          return ⊥
03      id ← rank(v, 0, LOUDS)
04      return select(id + 1, 1, LOUDS) + i

00  parent(v)
01      if is_root(v) then
02          return ⊥
01      pid ← rank(v, 1, LOUDS)
04      return select(pid, 0, LOUDS)
```

LOUDS – level order unary degree sequence

Karlsruher Institut für Technologie

Conclusion

- Total space for LOUDS representation is $2n + 1 + o(n)$ bits
- All operations take constant time