Karlsruher Institut für Technologie

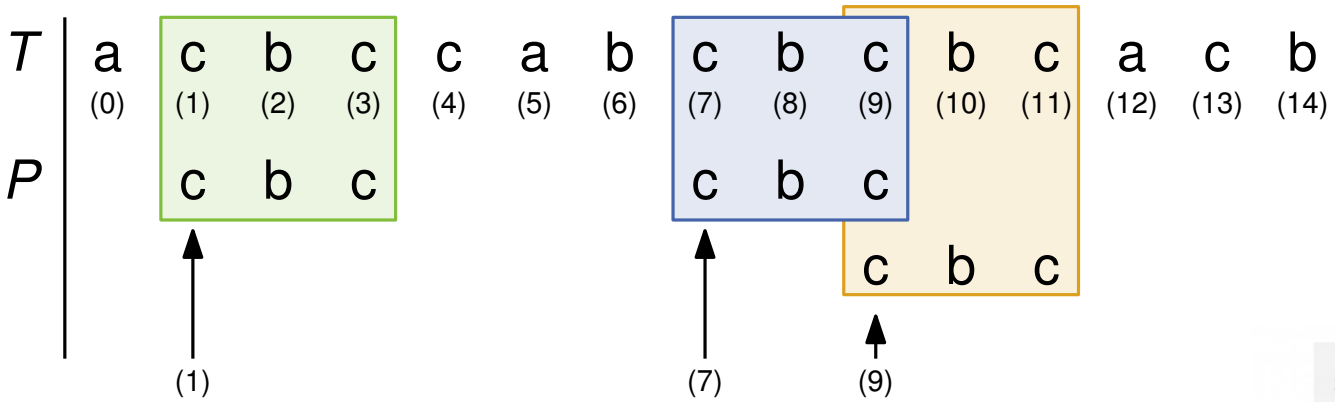
Algorithmen II
Übung am 09.01.2014

INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER

Definition & Motivation

Problem: String-Matching
Seien P und T Zeichenfolgen mit Zeichen aus dem Alphabet Σ, wobei $|P| < |T|$.
Finde alle Vorkommen von P (Muster, engl. Pattern) in T (Text).

Beispiel:
Das Muster P = cbc taucht im Text T = acbccabcbcbcacb an den Stellen 1, 7 und 9 auf.



Anwendungsbeispiele:

- Suche nach Textstellen in einem Textdokument.
- Suche nach einer bestimmten Sequenz von Basenpaaren in einer DNA.

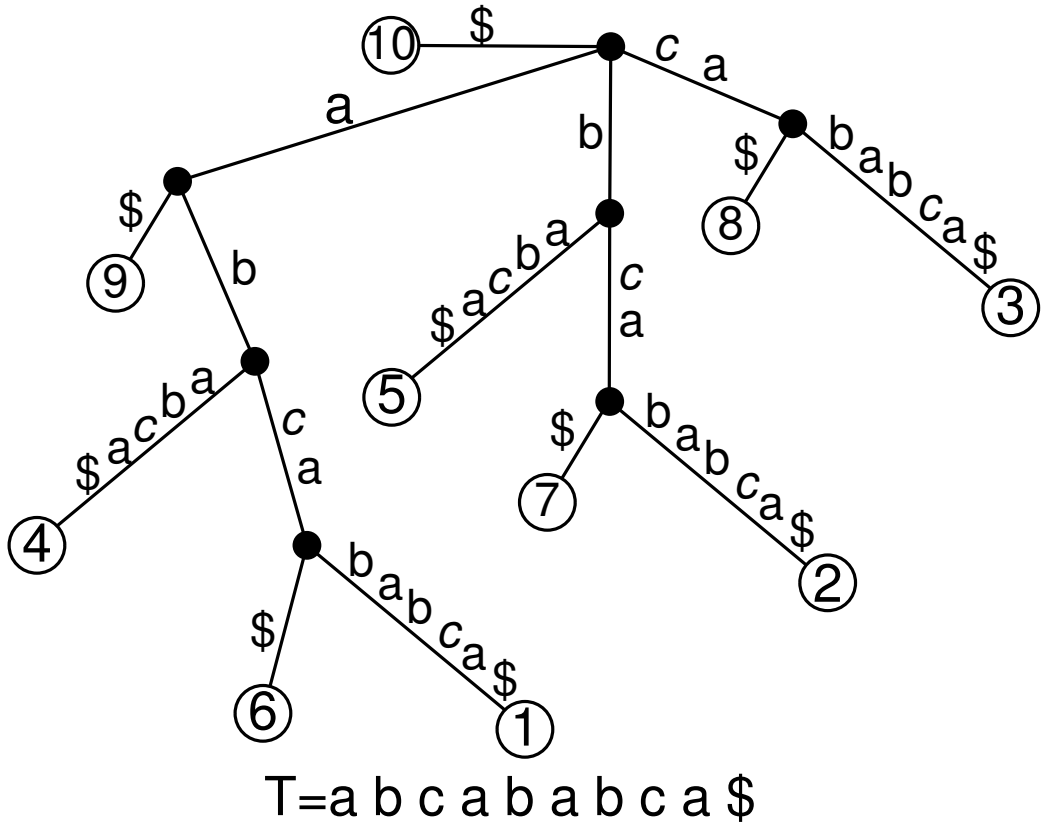
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problemstellung

Gegeben: Text T der Länge n.

Fragestellung: Wie kann T vorverarbeitet werden, so dass Suchanfragen auf T schnell möglich sind?

Idee: Repräsentiere T als Suchbaum.



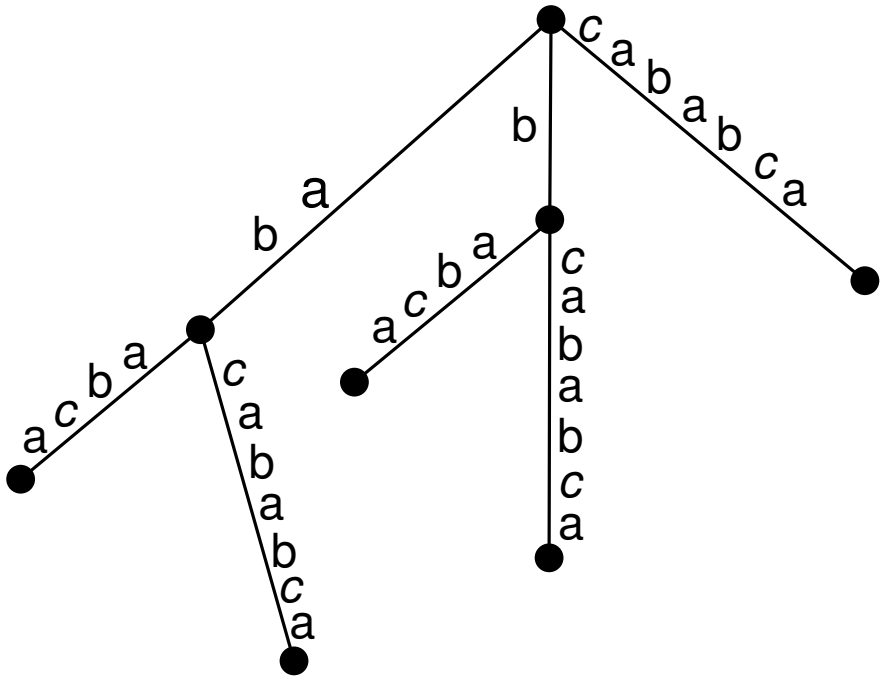
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

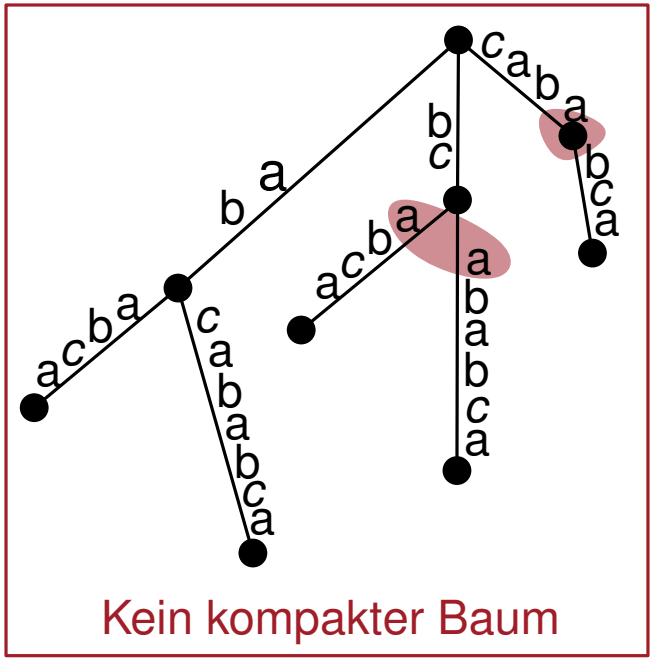
Kompakte Bäume

Definition 7: Ein *kompakter $\Sigma^\star$-Baum* ist ein Baum $\mathcal{T} = (V, E)$ mit Wurzel r und Kantenbeschriftungen aus $\Sigma^\star$, so dass für alle inneren Knoten $v \in V$ gilt

- die Kantenbeschriftungen der ausgehenden Kanten von v beginnen mit unterschiedlichem Zeichen $a \in \Sigma$, und
- wenn v nicht Wurzel ist, dann ist der Ausgangsgrad ungleich 1.

T=a b c a b a b c a





Kein kompakter Baum

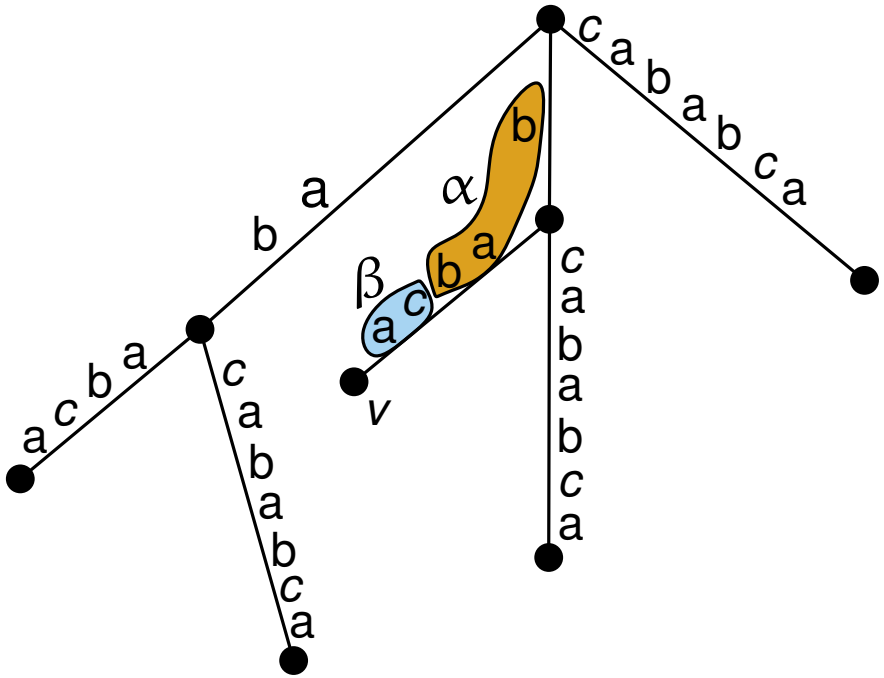
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Kompakte Bäume

Definition 8:

- $\overline{v}$:= Konkatenation der Kantenbeschriftungen auf dem Pfad von r zu v
- $d(v) := |\overline{v}|$ heißt String-Tiefe von v.
- $\mathcal{T}$ *enthält* $\alpha \in \Sigma^\star$, falls es Knoten $v \in V$ und Wort $\beta \in \Sigma^\star$ gibt, so dass $\overline{v} = \alpha\beta$.
- words($\mathcal{T}$) := $\{\alpha \in \Sigma^\star \mid \mathcal{T}$ enthält $\alpha\}$

T=a b c a b a b c a



$\overline{v}$ = *babca*

$d(v) = |\overline{v}| = 5$

$\mathcal{T}$ enthält *bab*, da für α = *bab* und β = *ca* es Knoten v gibt mit $\overline{v} = \alpha\beta$.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suffixbaum

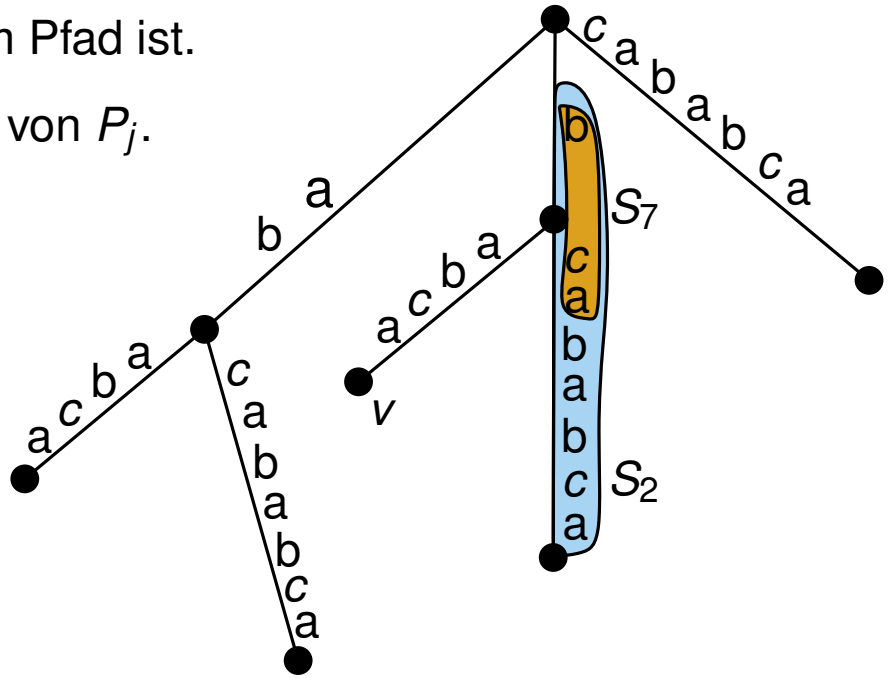
Definition: Ein *Suffixbaum* ist ein kompakter $\Sigma^\star$-Baum S, sodass S genau die Infixe von T enthält:

$$\text{words}(S) = \{T[i,j] \mid 1 \leq i \leq j \leq n\}$$

Beobachtung:

- Für jedes Suffix S_i gibt es einen Pfad P_i, der von der Wurzel zu einem Blatt führt, sodass S_i Präfix von diesem Pfad ist.
- Wenn S_i Präfix von S_j ist, dann ist P_i Teilpfad von P_j.

T=a b c a b a b c a



Notation:

1. S_i bezeichnet das Suffix von T beginnend ab der i-ten Stelle.
2. $T[i,j]$ bezeichnet den Teilstring von T von der i-ten bis zur j-ten Stelle.

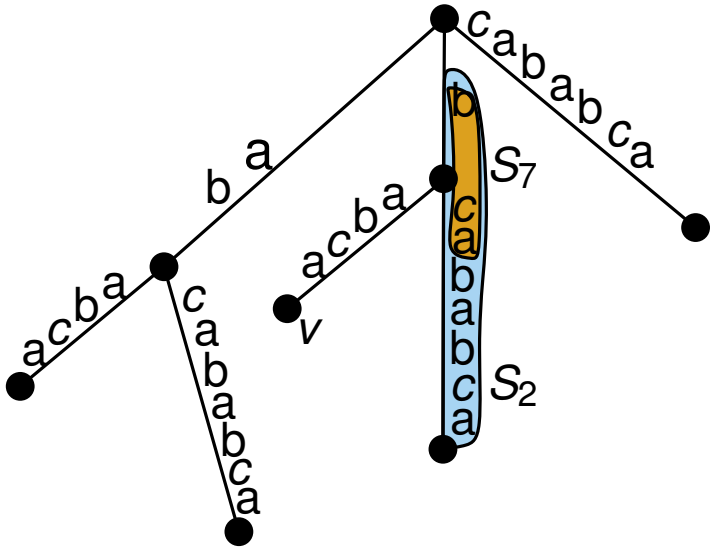
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suffixbaum

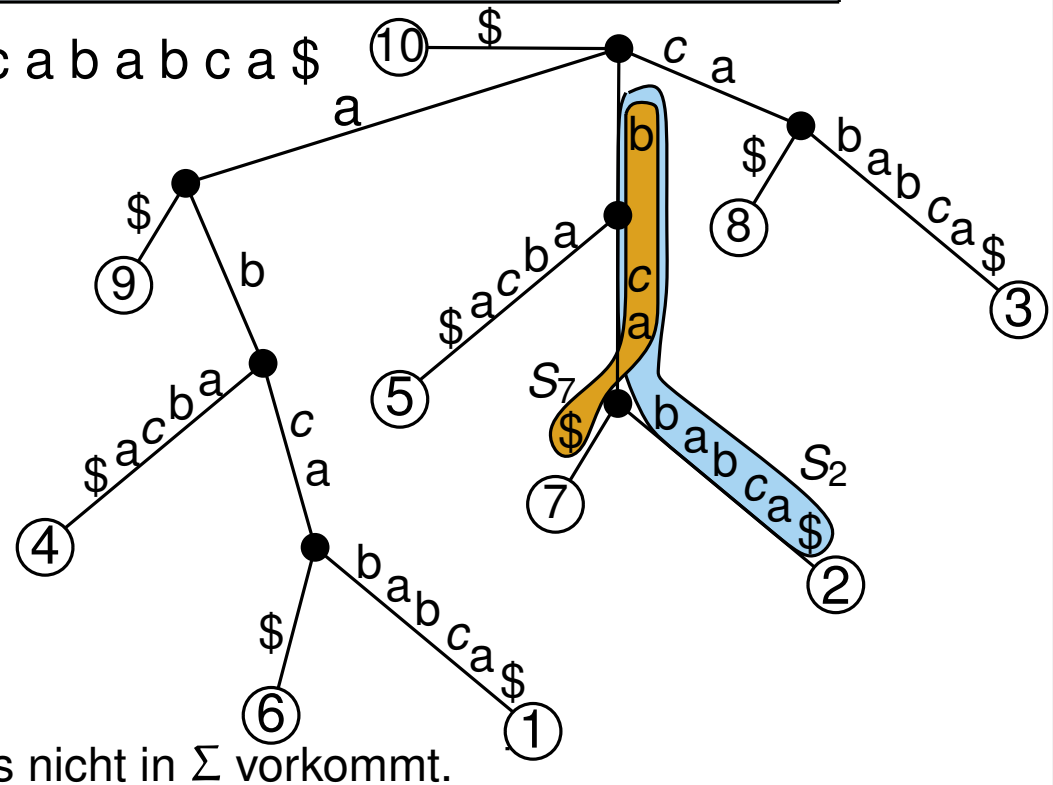
Definition: Ein *Suffixbaum* ist ein kompakter $\Sigma^\star$-Baum S, sodass S genau die Infixe von T enthält:

$$\text{words}(S) = \{T[i,j] \mid 1 \leq i \leq j \leq n\}$$

T=a b c a b a b c a



T=a b c a b a b c a $



Nützlicher Trick: Hänge an T ein Zeichen \$, das nicht in Σ vorkommt.

1. Suffix S_i kann nicht Präfix eines anderen Suffixes S_j sein.
2. Jedes Suffix endet an einem Blatt.
3. Es gibt so viele Blätter wie Suffixe.

$\Rightarrow$ nummeriere Blätter so, dass i-te Blatt dem Suffix S_i entspricht.

Ab jetzt:
S_i := i-te Suffix
:= Pfad von Wurzel zu Blatt i.

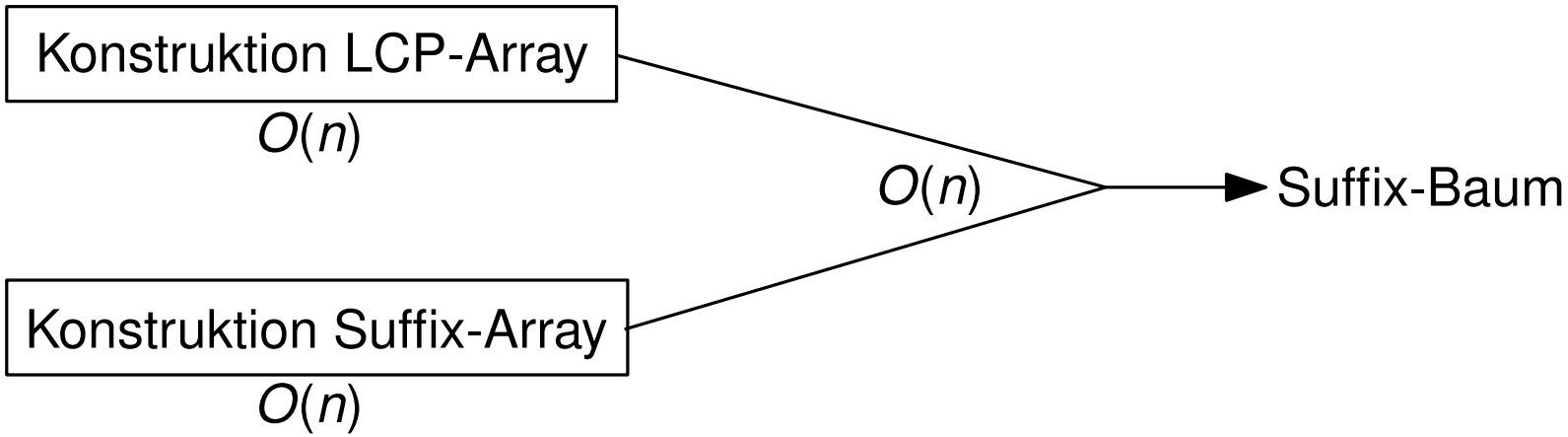
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

In der Vorlesung wurden zwei Möglichkeiten vorgestellt:

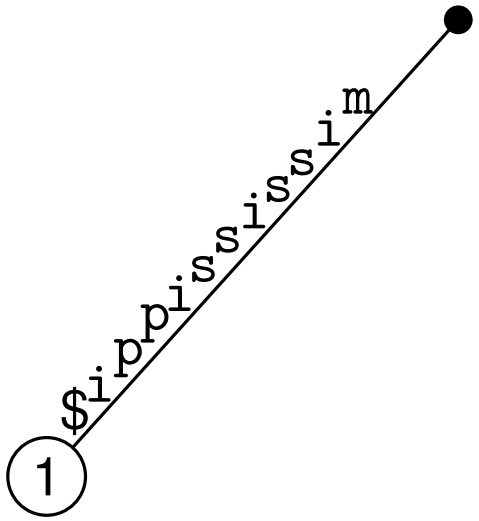
1. Direktes Aufbauen in $O(n^2)$ Zeit.
2. Umweg über LCP-Array und Suffix-Array:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

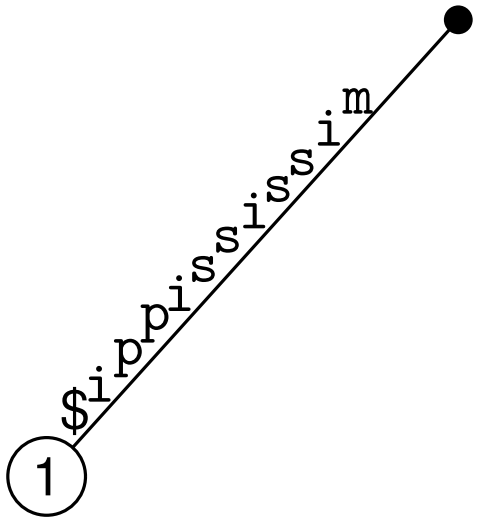


Beginne mit S_1=`mississippi$`.

Idee: Konstruiere $N_1, \ldots, N_n$ Bäume, sodass N_i die Suffixe $S_1, \ldots, S_i$ enthält.

Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

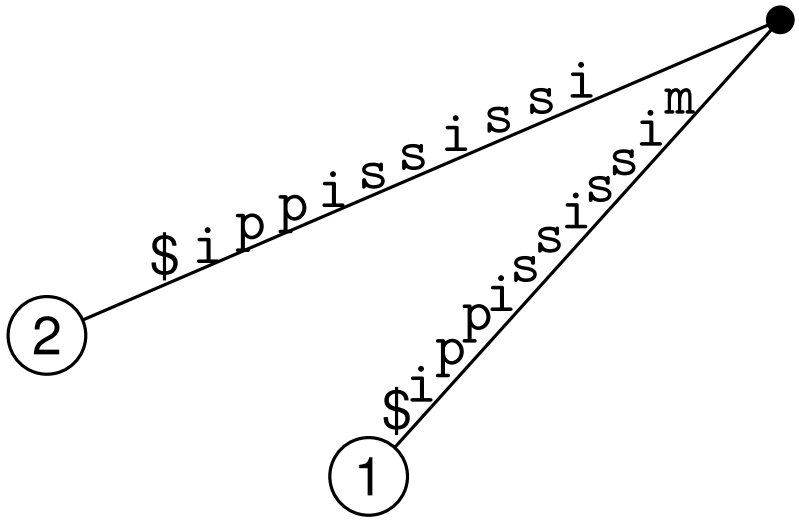


Füge S_2=`ississippi$` ein: Spalte an Wurzel neue Kante ab.

Idee: Konstruiere $N_1, \ldots, N_n$ Bäume, sodass N_i die Suffixe $S_1, \ldots, S_i$ enthält.

Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

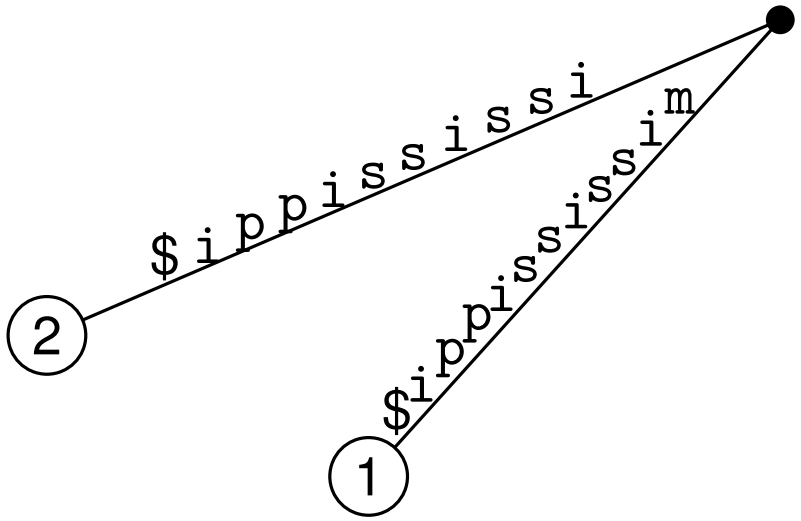


Füge S_2=`ississippi$` ein: Spalte an Wurzel neue Kante ab.

Idee: Konstruiere $N_1, \ldots, N_n$ Bäume, sodass N_i die Suffixe $S_1, \ldots, S_i$ enthält.

Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

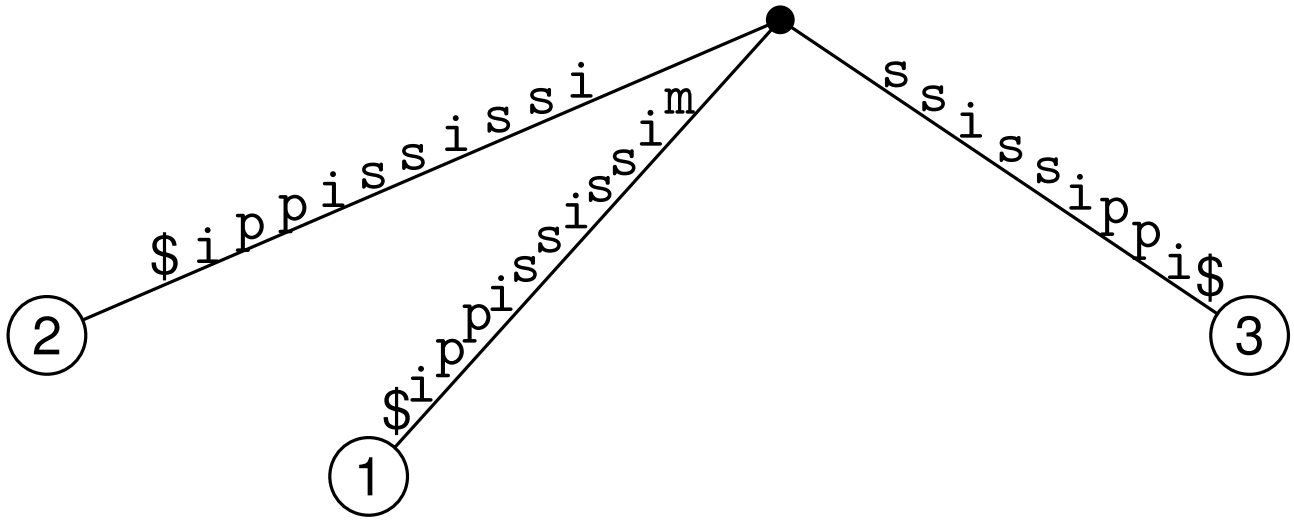


Füge S_3=`ssissippi$` ein: Spalte an Wurzel neue Kante ab.

Idee: Konstruiere $N_1, \ldots, N_n$ Bäume, sodass N_i die Suffixe $S_1, \ldots, S_i$ enthält.

Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

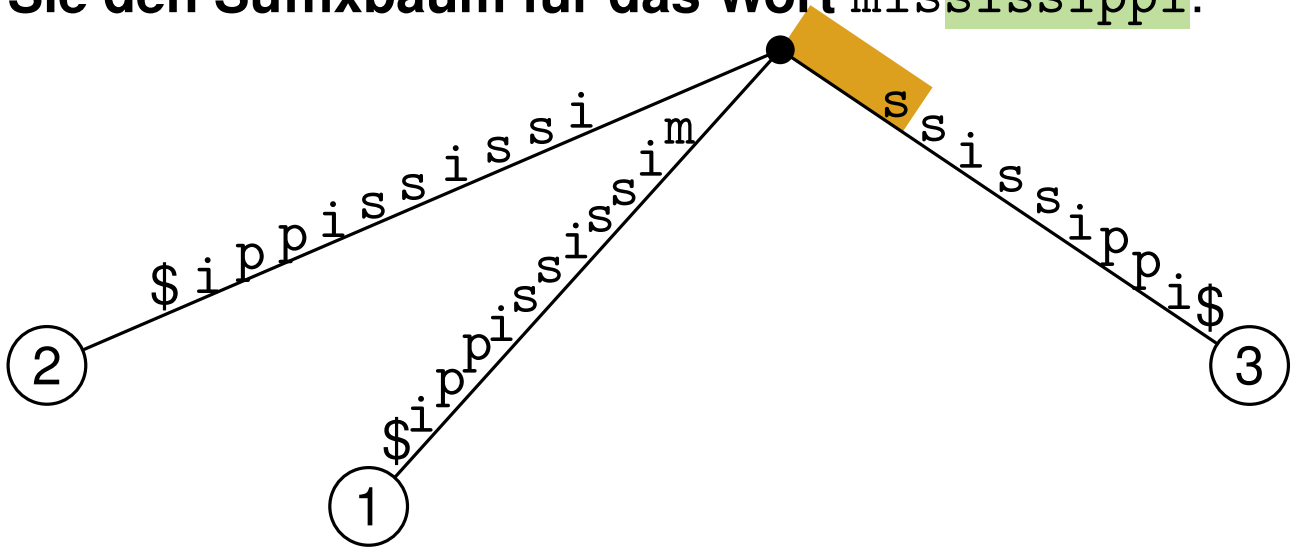


Füge S_3=`ssissippi$` ein: Spalte an Wurzel neue Kante ab.

Idee: Konstruiere $N_1, \ldots, N_n$ Bäume, sodass N_i die Suffixe $S_1, \ldots, S_i$ enthält.

Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

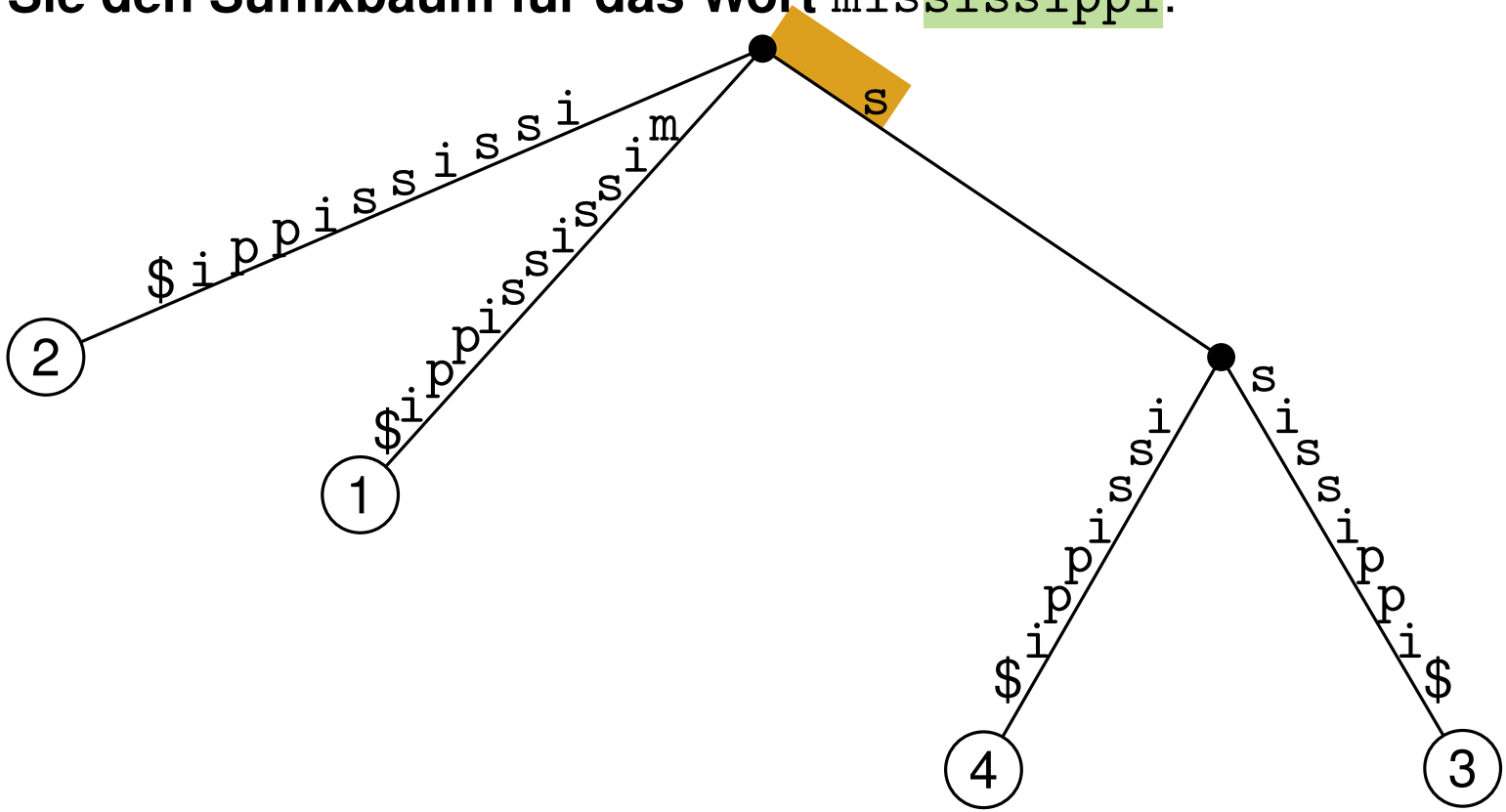


Füge S_4=`sissippi$` ein: Spalte Pfad von S_3 nach gemeinsamen Anteil auf.

Idee: Konstruiere $N_1, \ldots, N_n$ Bäume, sodass N_i die Suffixe $S_1, \ldots, S_i$ enthält.

Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

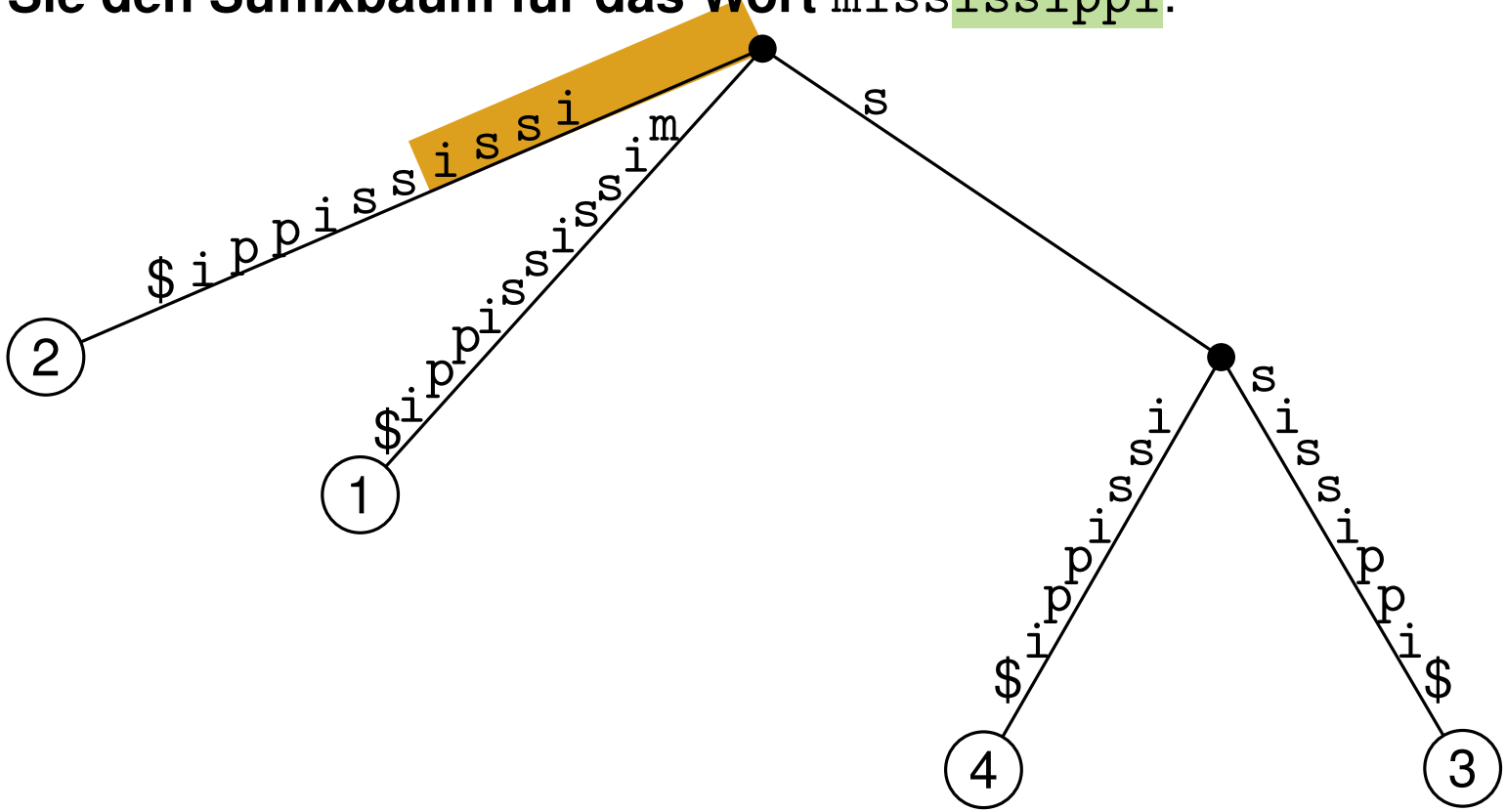


Füge S_4=`sissippi$` ein: Spalte Pfad von S_3 nach gemeinsamen Anteil auf.

Idee: Konstruiere $N_1, \ldots, N_n$ Bäume, sodass N_i die Suffixe $S_1, \ldots, S_i$ enthält.

Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

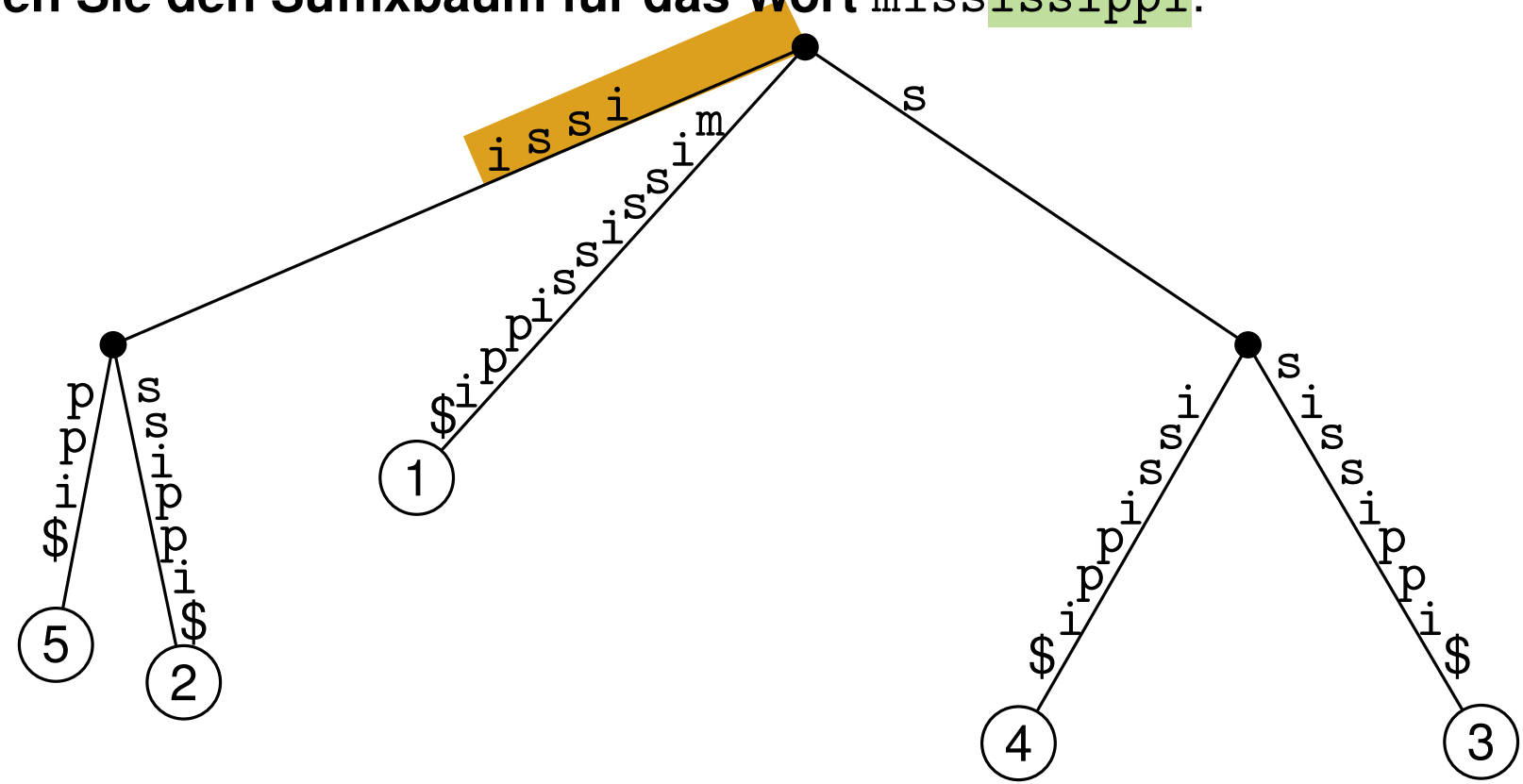


Füge S_5=`issippi$` ein: Spalte Pfad von S_2 nach gemeinsamen Anteil auf.

Idee: Konstruiere $N_1, \ldots, N_n$ Bäume, sodass N_i die Suffixe $S_1, \ldots, S_i$ enthält.

Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

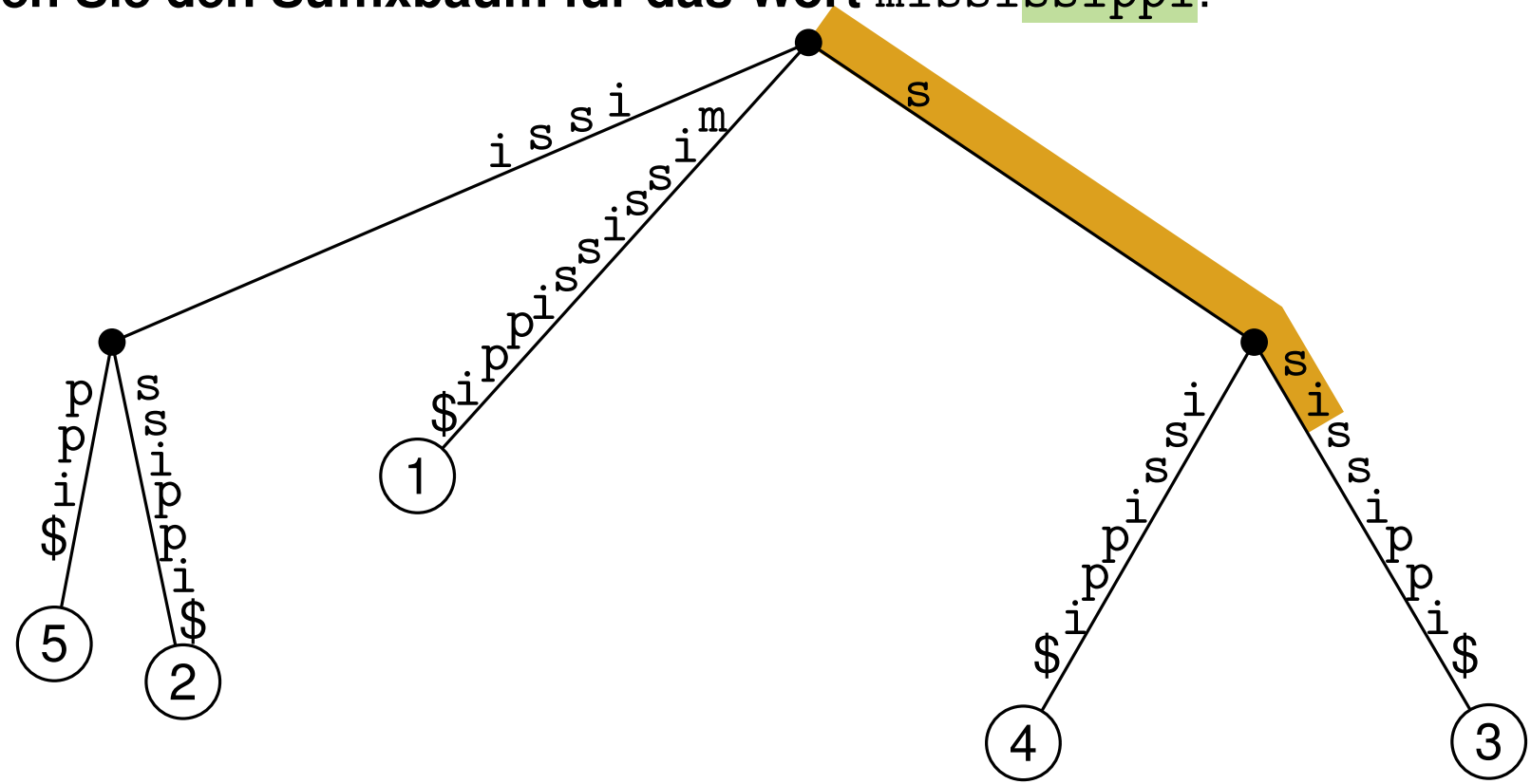


Füge S_5=`issippi$` ein: Spalte Pfad von S_2 nach gemeinsamen Anteil auf.

Idee: Konstruiere $N_1, \ldots, N_n$ Bäume, sodass N_i die Suffixe $S_1, \ldots, S_i$ enthält.

Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

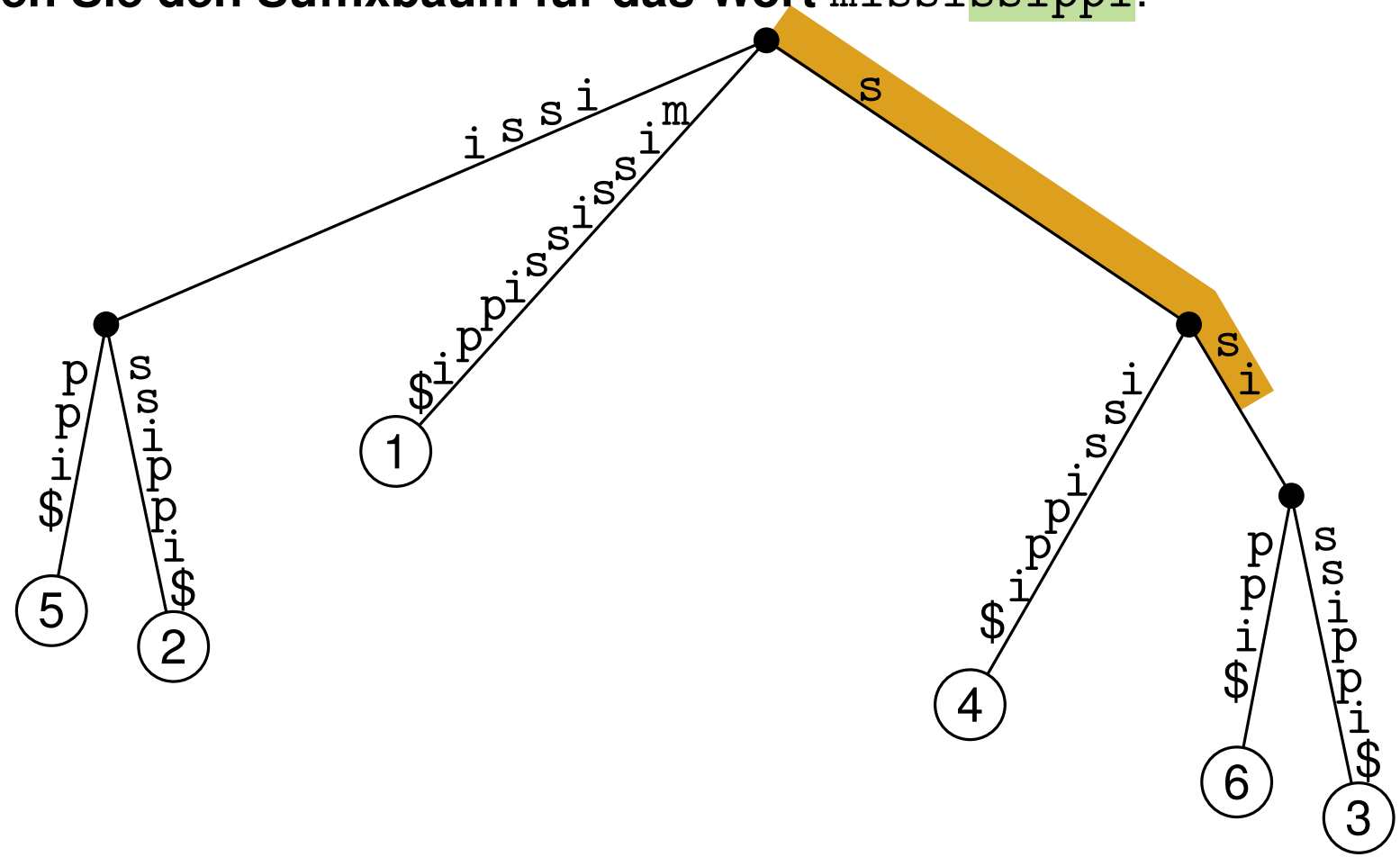


Füge S_6=`ssippi$` ein: Spalte Pfad von S_3 nach gemeinsamen Anteil auf.

Idee: Konstruiere $N_1, \ldots, N_n$ Bäume, sodass N_i die Suffixe $S_1, \ldots, S_i$ enthält.

Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

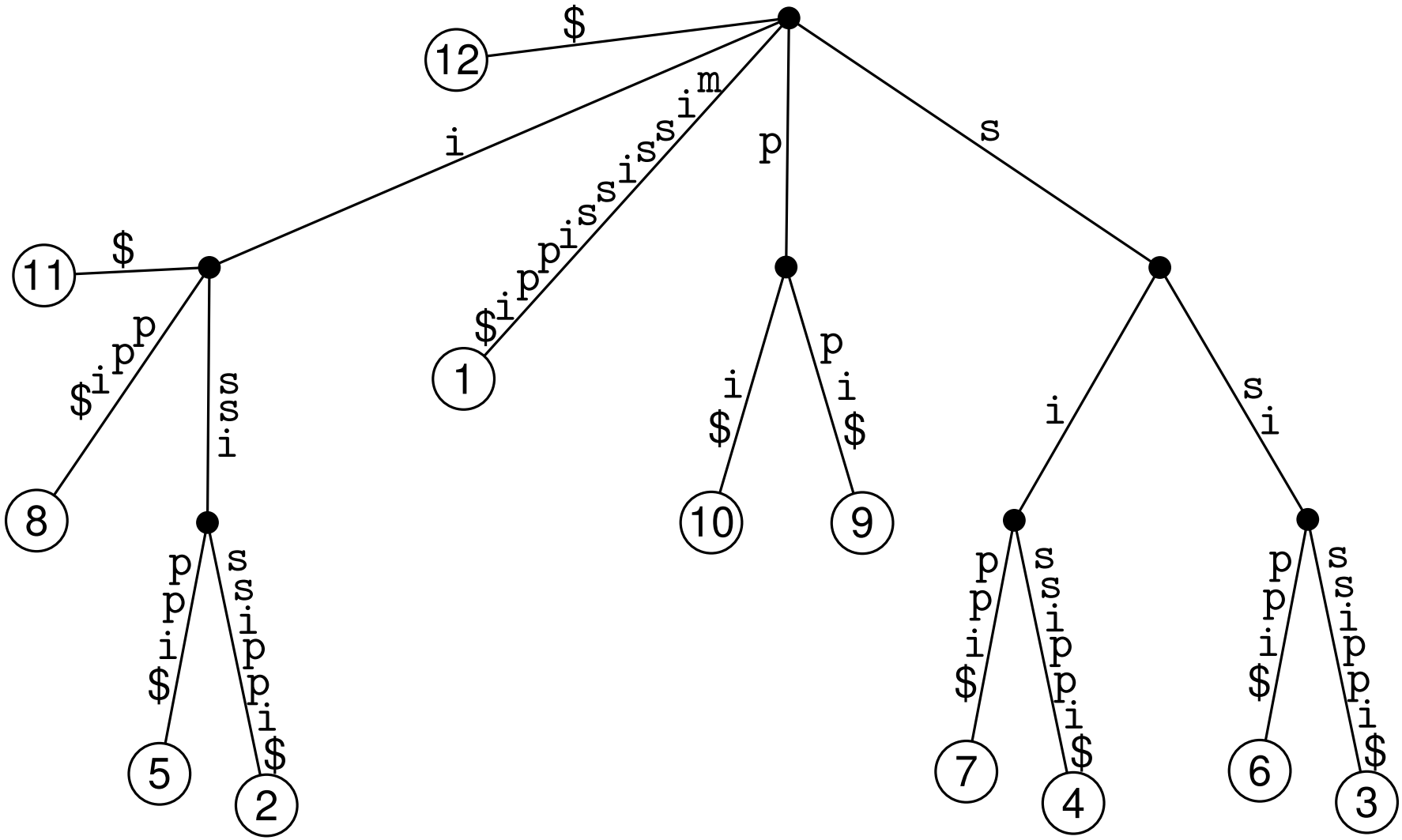


Füge S_6=`ssippi$` ein: Spalte Pfad von S_3 nach gemeinsamen Anteil auf.

Idee: Konstruiere $N_1, \ldots, N_n$ Bäume, sodass N_i die Suffixe $S_1, \ldots, S_i$ enthält.

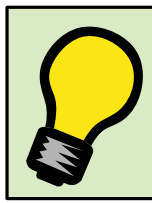
Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

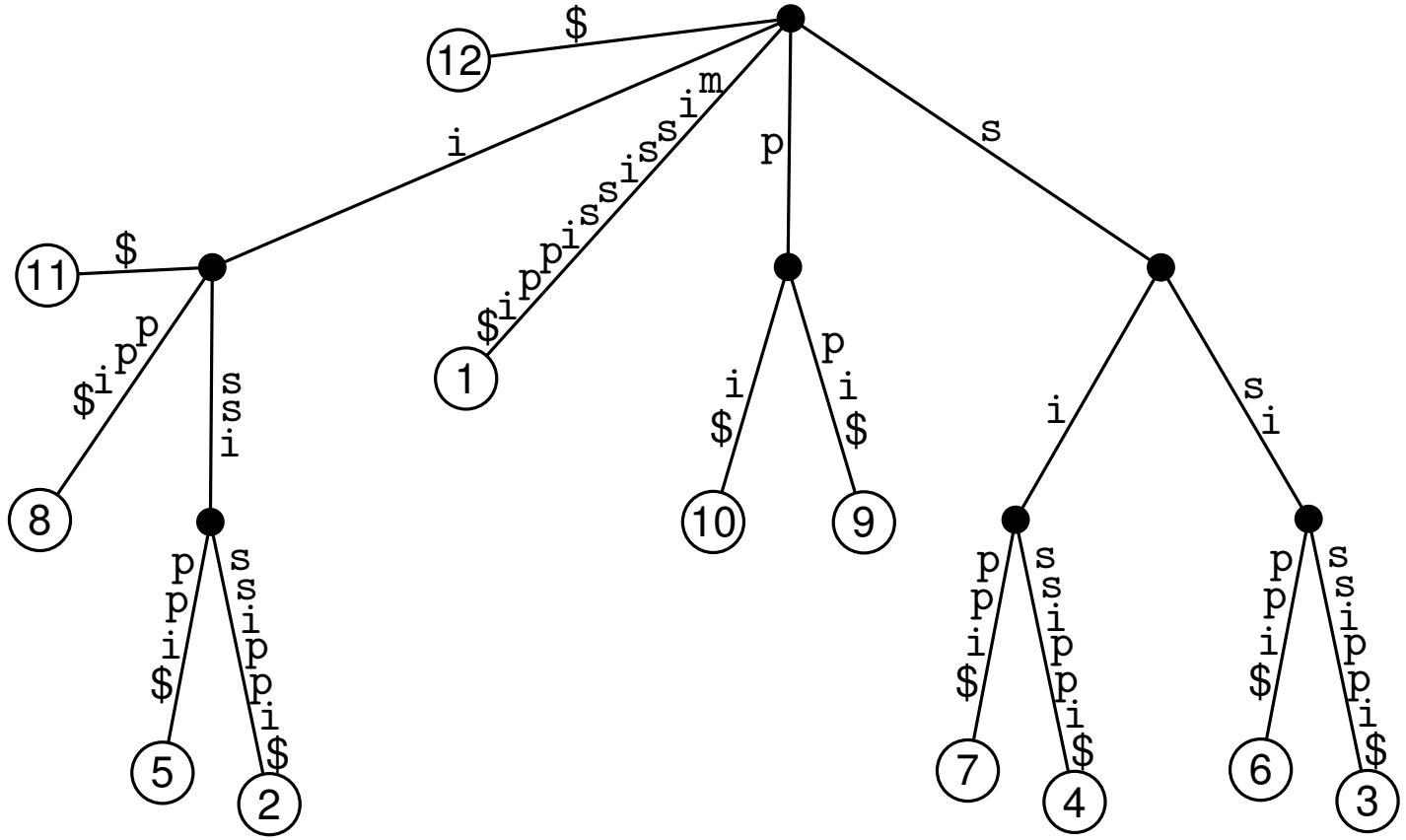


Idee: Konstruiere $N_1, \ldots, N_n$ Bäume, sodass N_i die Suffixe $S_1, \ldots, S_i$ enthält.

Problem 2



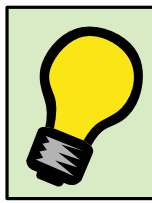
Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.



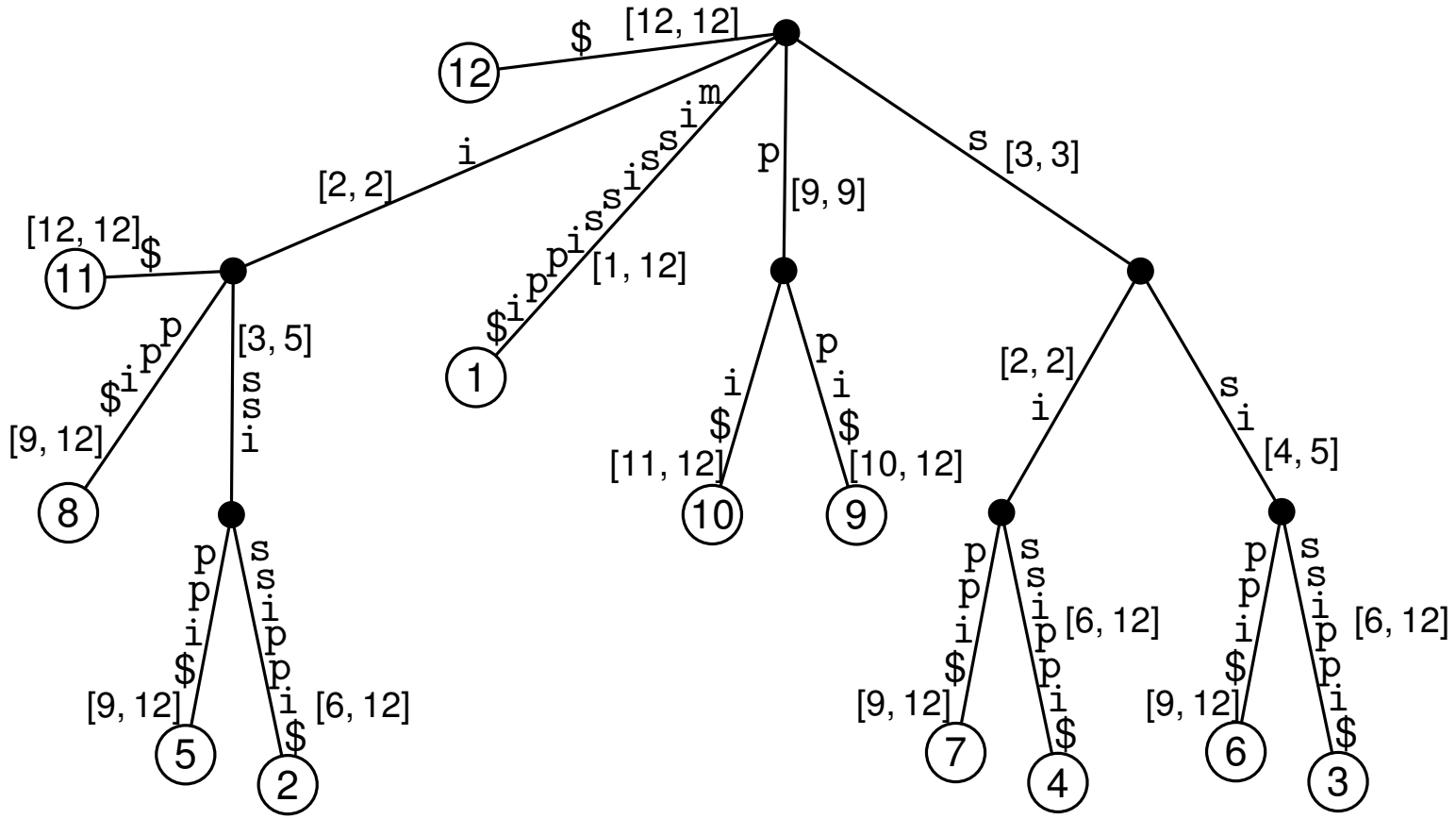
1	2	3	4	5	6	7	8	9	10	11	12
m	i	s	s	i	s	s	i	p	p	i	$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 2



Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.



1	2	3	4	5	6	7	8	9	10	11	12
m	i	s	s	i	s	s	i	p	p	i	$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 2

Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.

Kann es wirklich passieren, dass man mehr als $O(n)$ Speicher benötigt, wenn man die Beschriftungen direkt an den Kanten speichert?

Was wenn Alphabet Σ nicht fest ist?

Problem 2

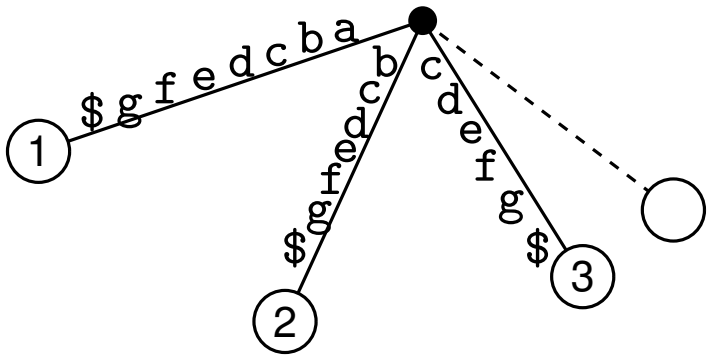
Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.

Kann es wirklich passieren, dass man mehr als $O(n)$ Speicher benötigt, wenn man die Beschriftungen direkt an den Kanten speichert?

Was wenn Alphabet Σ nicht fest ist?

Idee: Gebe Text T und Alphabet Σ an, sodass gilt Speicherverbrauch liegt in $\Omega(n^2)$.

T=`abcdefg...`\$, $\Sigma = \{a, b, c, d, e, f, g, \ldots\}$ T besteht aus unterschiedlichen Zeichen aus Σ.



$$\text{Speicherverbrauch} \geq \sum_{i=1}^{n} i = \frac{n^2 + n}{2} \geq \frac{n^2}{2} = \Omega(n^2)$$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 2

Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.

Kann es wirklich passieren, dass man mehr als $O(n)$ Speicher benötigt, wenn man die Beschriftungen direkt an den Kanten speichert?

Was wenn Alphabet Σ fest ist? $\Sigma = \{0, 1, 2\}$

Problem 2

Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.

Kann es wirklich passieren, dass man mehr als $O(n)$ Speicher benötigt, wenn man die Beschriftungen direkt an den Kanten speichert?

Was wenn Alphabet Σ fest ist? $\Sigma = \{0, 1, 2\}$

Betrachte Wort: $w = w_1 w_2$

$w_1 = 123 \ldots k$, binär kodiert: $\lceil \log k \rceil$ Zeichen pro Zahl.

$w_2 = 2 \ldots 2$, bestehend aus $k \lceil \log k \rceil$ Zweien.

Beispiel:

$w_1 = 0123 =_2 00011011$ $\quad w_2 = 222222$

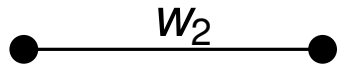
w= 00 01 10 11 222222

Problem 2

Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.

Kann es wirklich passieren, dass man mehr als $O(n)$ Speicher benötigt, wenn man die Beschriftungen direkt an den Kanten speichert?

Was wenn Alphabet Σ fest ist? $\Sigma = \{0, 1, 2\}$



Betrachte Wort: $w = w_1 w_2$

$w_1 = 123 \ldots k$, binär kodiert: $\lceil \log k \rceil$ Zeichen pro Zahl.

$w_2 = 2 \ldots 2$, bestehend aus $k \lceil \log k \rceil$ Zweien.

Beispiel:

$w_1 = 0123 =_2 00011011$ $\quad w_2 = 222222$

w= 00 01 10 11 222222

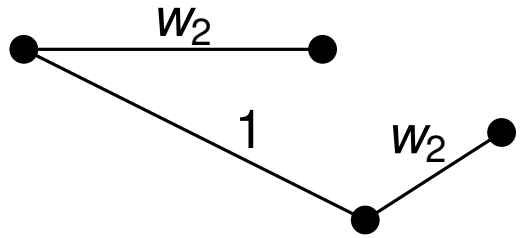
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 2

Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.

Kann es wirklich passieren, dass man mehr als $O(n)$ Speicher benötigt, wenn man die Beschriftungen direkt an den Kanten speichert?

Was wenn Alphabet Σ fest ist? $\Sigma = \{0, 1, 2\}$



Betrachte Wort: $w = w_1 w_2$

$w_1 = 123 \ldots k$, binär kodiert: $\lceil \log k \rceil$ Zeichen pro Zahl.

$w_2 = 2 \ldots 2$, bestehend aus $k \lceil \log k \rceil$ Zweien.

Beispiel:

$w_1 = 0123 =_2 00011011$ $\quad w_2 = 222222$

w= 00 01 10 11 222222

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 2

Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.

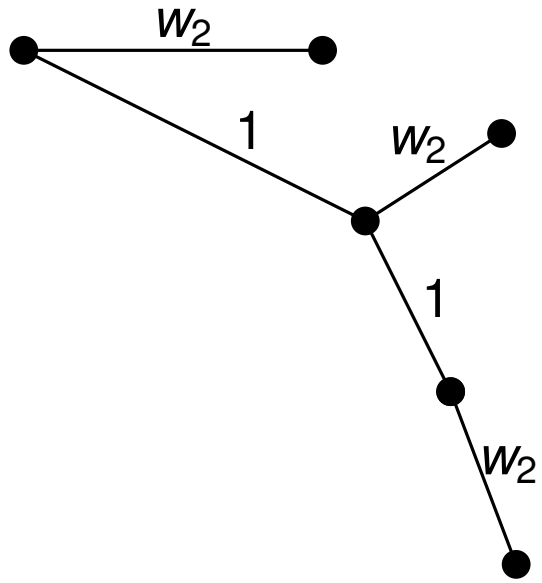
Kann es wirklich passieren, dass man mehr als $O(n)$ Speicher benötigt, wenn man die Beschriftungen direkt an den Kanten speichert?

Was wenn Alphabet Σ fest ist? $\Sigma = \{0, 1, 2\}$

Betrachte Wort: $w = w_1 w_2$

$w_1 = 123 \ldots k$, binär kodiert: $\lceil \log k \rceil$ Zeichen pro Zahl.

$w_2 = 2 \ldots 2$, bestehend aus $k \lceil \log k \rceil$ Zweien.



Beispiel:

$w_1 = 0123 =_2 00011011$ $\quad w_2 = 222222$

w= 00 01 10 11 222222

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 2

Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.

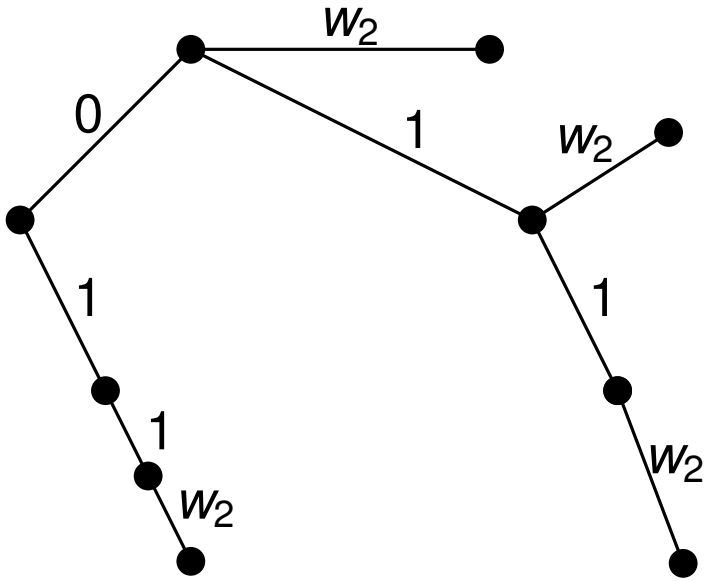
Kann es wirklich passieren, dass man mehr als $O(n)$ Speicher benötigt, wenn man die Beschriftungen direkt an den Kanten speichert?

Was wenn Alphabet Σ fest ist? $\Sigma = \{0, 1, 2\}$

Betrachte Wort: $w = w_1 w_2$

$w_1 = 123 \ldots k$, binär kodiert: $\lceil \log k \rceil$ Zeichen pro Zahl.

$w_2 = 2 \ldots 2$, bestehend aus $k \lceil \log k \rceil$ Zweien.



Beispiel:

$w_1 = 0123 =_2 00011011$ $w_2 = 222222$

w= 00 01 10 11 222222

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 2

Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.

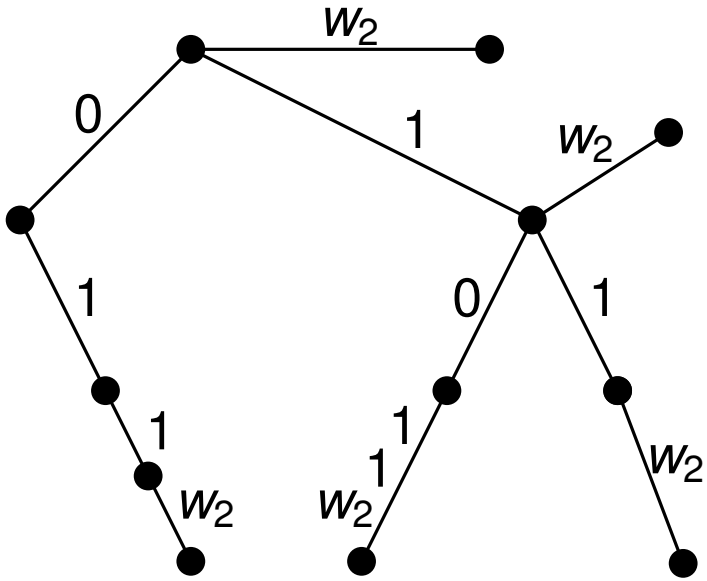
Kann es wirklich passieren, dass man mehr als $O(n)$ Speicher benötigt, wenn man die Beschriftungen direkt an den Kanten speichert?

Was wenn Alphabet Σ fest ist? $\Sigma = \{0, 1, 2\}$

Betrachte Wort: $w = w_1 w_2$

$w_1 = 123 \ldots k$, binär kodiert: $\lceil \log k \rceil$ Zeichen pro Zahl.

$w_2 = 2 \ldots 2$, bestehend aus $k \lceil \log k \rceil$ Zweien.



Beispiel:

$w_1 = 0123 =_2 00011011$ $\quad w_2 = 222222$

w= 00 01 10 11 222222

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 2

Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.

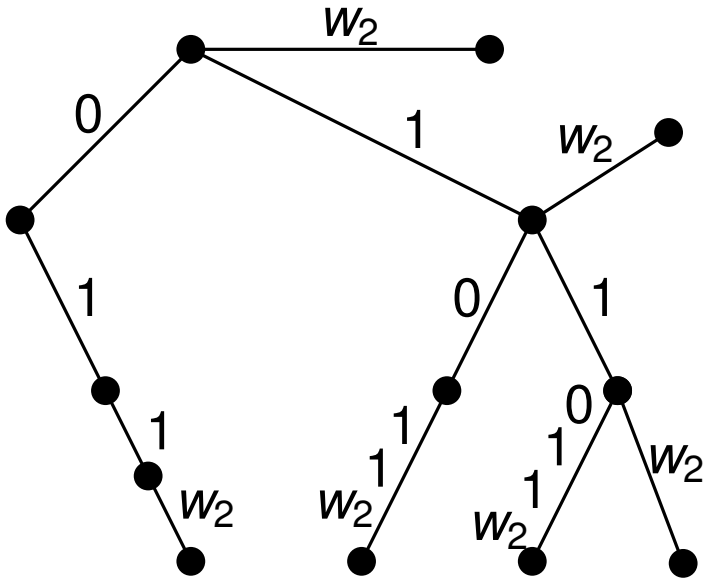
Kann es wirklich passieren, dass man mehr als $O(n)$ Speicher benötigt, wenn man die Beschriftungen direkt an den Kanten speichert?

Was wenn Alphabet Σ fest ist? $\Sigma = \{0, 1, 2\}$

Betrachte Wort: $w = w_1 w_2$

$w_1 = 123 \ldots k$, binär kodiert: $\lceil \log k \rceil$ Zeichen pro Zahl.

$w_2 = 2 \ldots 2$, bestehend aus $k \lceil \log k \rceil$ Zweien.



Beispiel:

$w_1 = 0123 =_2 00011011$ $\quad w_2 = 222222$

w= 00 01 10 11 222222

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 2

Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.

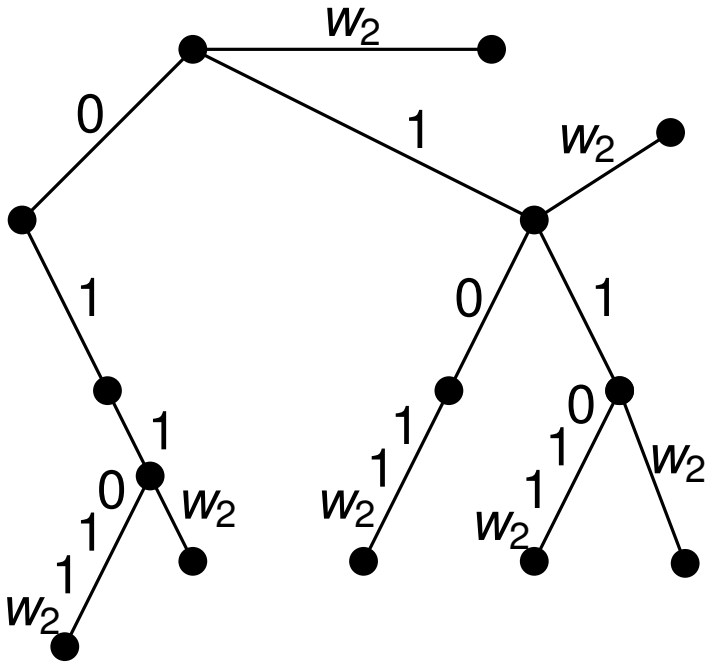
Kann es wirklich passieren, dass man mehr als $O(n)$ Speicher benötigt, wenn man die Beschriftungen direkt an den Kanten speichert?

Was wenn Alphabet Σ fest ist? $\Sigma = \{0, 1, 2\}$

Betrachte Wort: $w = w_1 w_2$

$w_1 = 123 \ldots k$, binär kodiert: $\lceil \log k \rceil$ Zeichen pro Zahl.

$w_2 = 2 \ldots 2$, bestehend aus $k \lceil \log k \rceil$ Zweien.



Beispiel:

$w_1 = 0123 =_2 00011011$ $\quad w_2 = 222222$

w= 00 01 10 11 222222

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 2

Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.

Kann es wirklich passieren, dass man mehr als $O(n)$ Speicher benötigt, wenn man die Beschriftungen direkt an den Kanten speichert?

Was wenn Alphabet Σ fest ist? $\Sigma = \{0, 1, 2\}$

Betrachte Wort: $w = w_1 w_2$

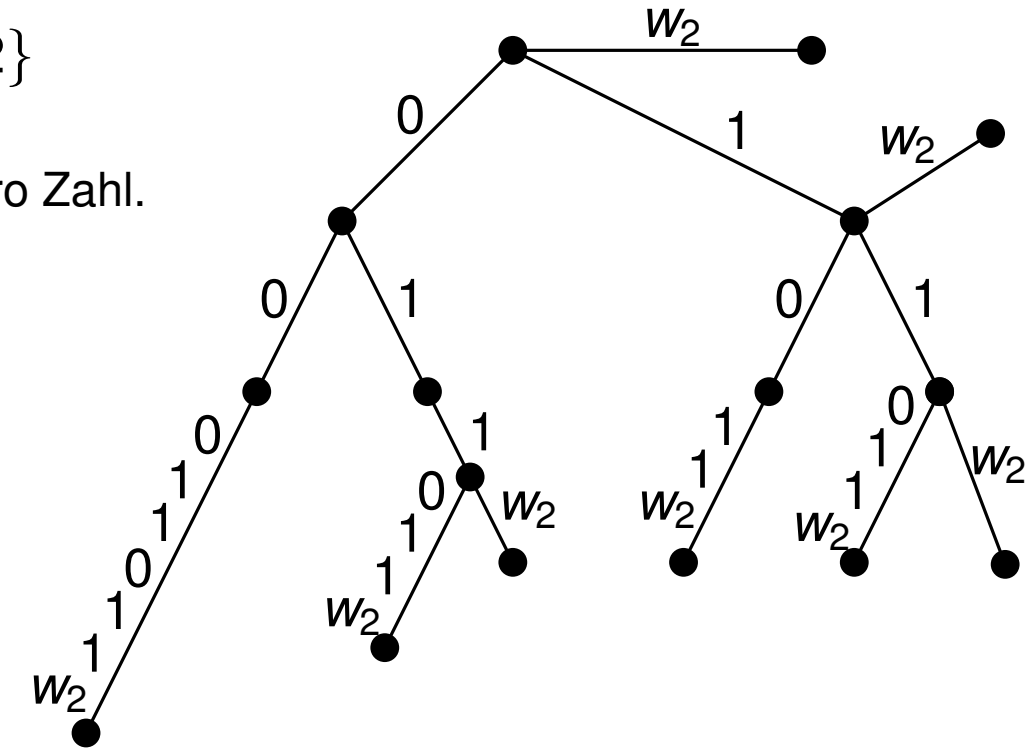
$w_1 = 123 \ldots k$, binär kodiert: $\lceil \log k \rceil$ Zeichen pro Zahl.

$w_2 = 2 \ldots 2$, bestehend aus $k \lceil \log k \rceil$ Zweien.

Beispiel:

$w_1 = 0123 =_2 00011011$ $w_2 = 222222$

w= 00 01 10 11 222222



Problem 2

Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.

Kann es wirklich passieren, dass man mehr als $O(n)$ Speicher benötigt, wenn man die Beschriftungen direkt an den Kanten speichert?

Was wenn Alphabet Σ fest ist? $\Sigma = \{0, 1, 2\}$

Betrachte Wort: $w = w_1 w_2$

$w_1 = 123 \ldots k$, binär kodiert: $\lceil \log k \rceil$ Zeichen pro Zahl.

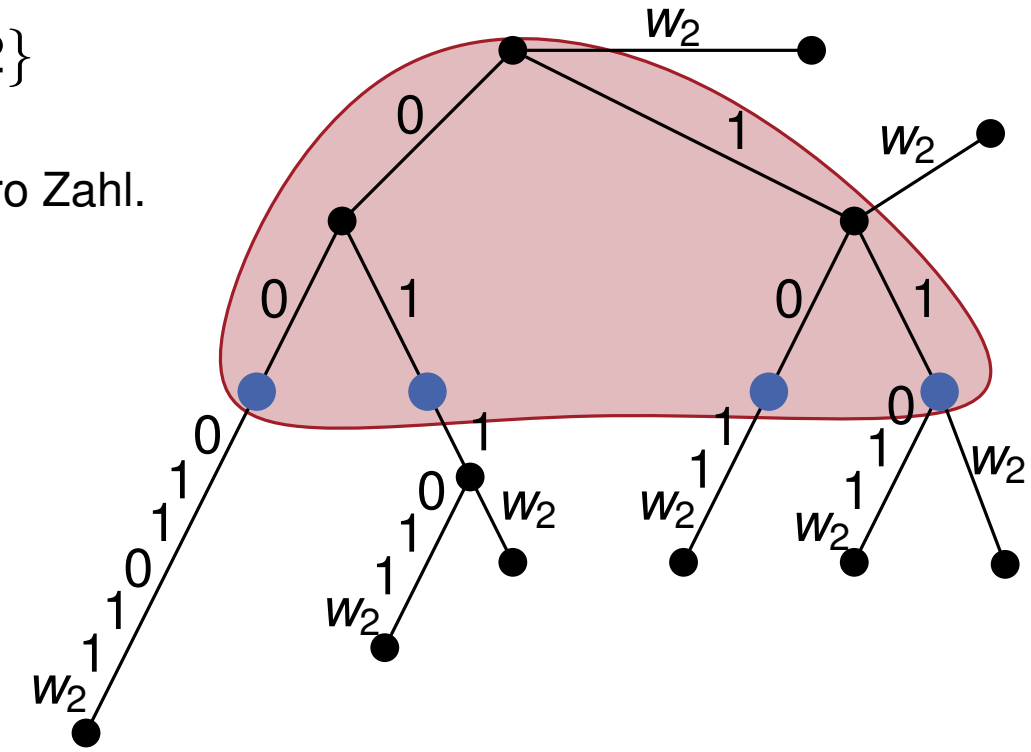
$w_2 = 2 \ldots 2$, bestehend aus $k \lceil \log k \rceil$ Zweien.

Beispiel:

$w_1 = 0123 =_2 00011011$ $\quad w_2 = 222222$

w= 00 01 10 11 222222

Binärbaum:
Jedes Blatt entspricht einer Ziffer von w_1



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 2

Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.

Kann es wirklich passieren, dass man mehr als $O(n)$ Speicher benötigt, wenn man die Beschriftungen direkt an den Kanten speichert?

Was wenn Alphabet Σ fest ist? $\Sigma = \{0, 1, 2\}$

Betrachte Wort: $w = w_1 w_2$

$w_1 = 123 \ldots k$, binär kodiert: $\lceil \log k \rceil$ Zeichen pro Zahl.

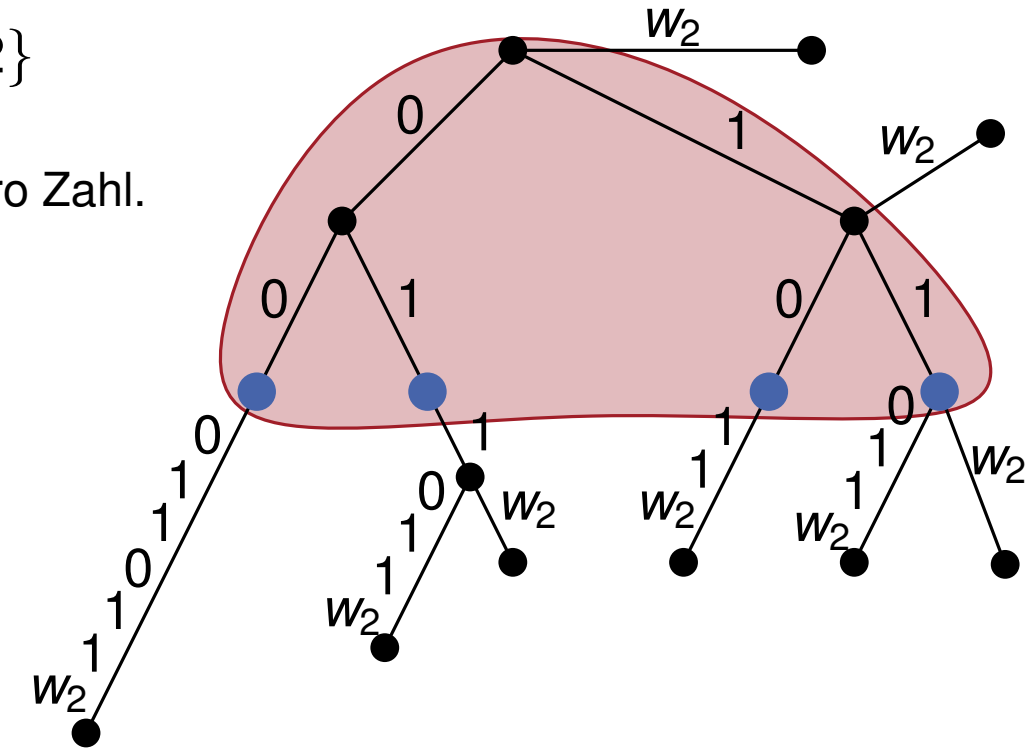
$w_2 = 2 \ldots 2$, bestehend aus $k \lceil \log k \rceil$ Zweien.

Beispiel:

$w_1 = 0123 =_2 00011011$ $\quad w_2 = 222222$

w= 00 01 10 11 222222

Binärbaum:
Jedes Blatt entspricht einer Ziffer von w_1



Für jedes Blatt gibt es ein Suffix mit mind. Länge $|w_2|$

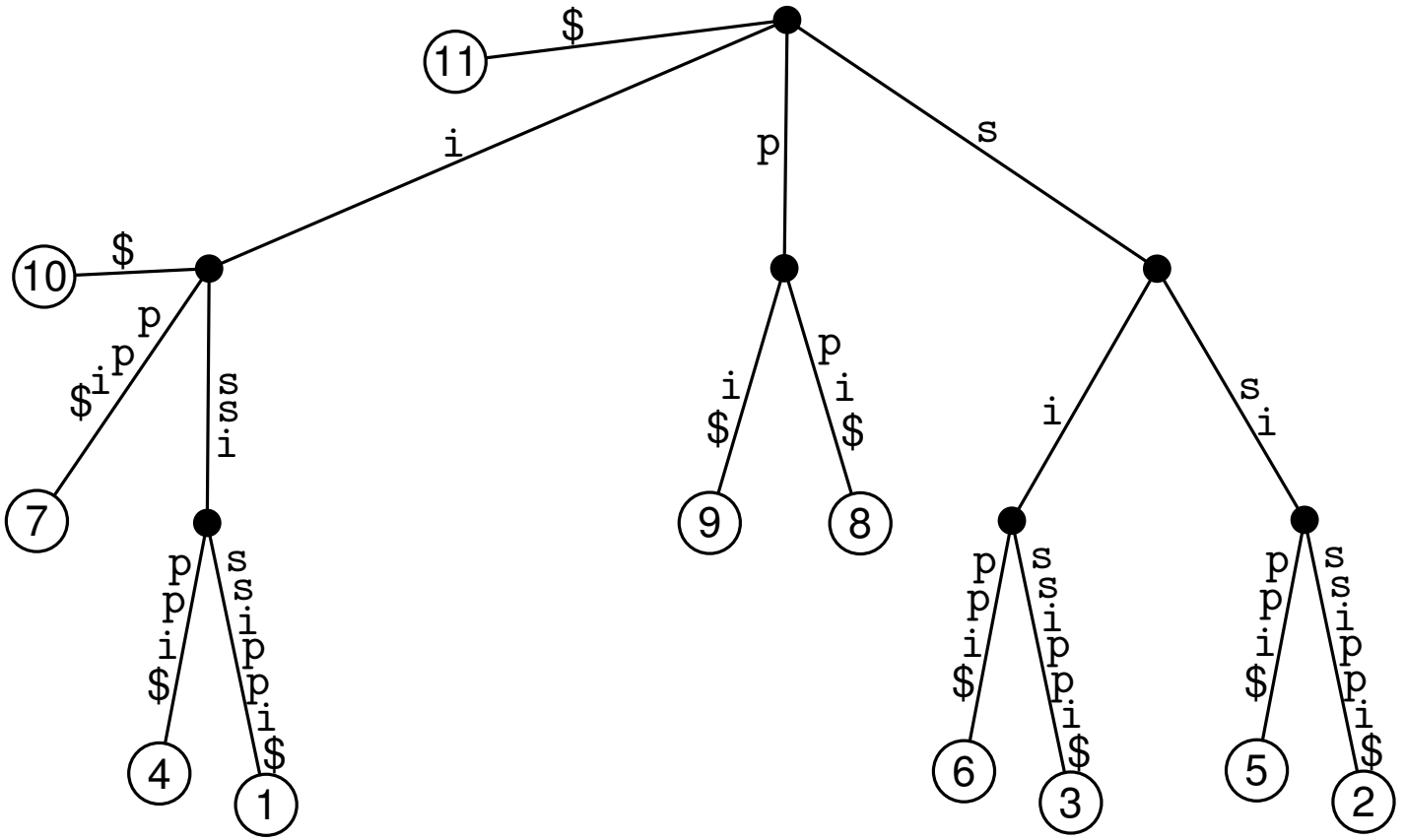
Binärbaum besitzt k Blätter. $\longrightarrow$ Speicherverbrauch $\Omega(k^2 \log k)$

Problem 3

Berechnen Sie alle kürzesten eindeutigen Teilstrings in einem Text T.

Teilstring S ist eindeutig, falls er genau einmal in T auftaucht.

T=issisippi$

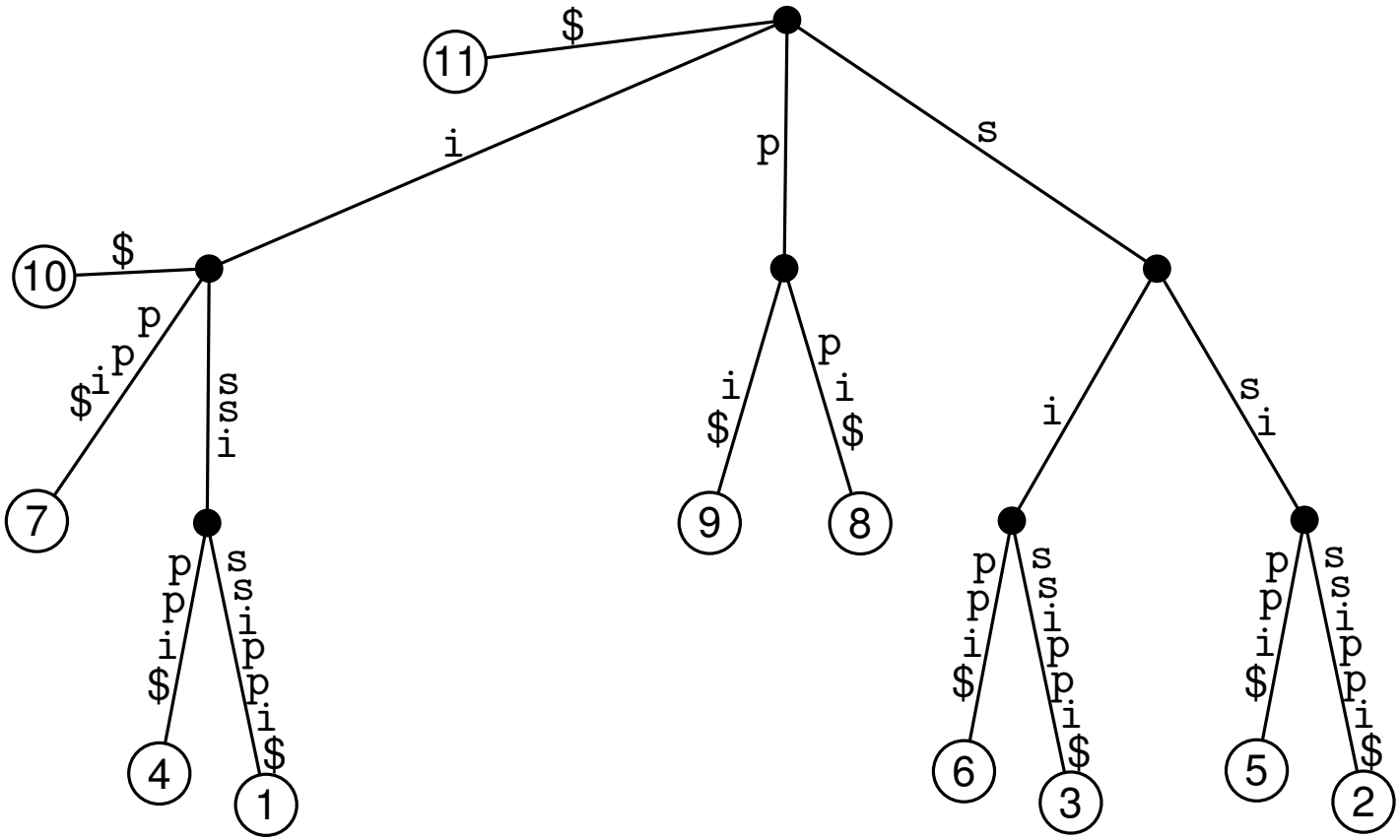


Problem 3

Berechnen Sie alle kürzesten eindeutigen Teilstrings in einem Text T.

Teilstring S ist eindeutig, falls er genau einmal in T auftaucht.

T=ississippi$



Beobachtung: Eindeutige Teilstrings enden auf einer Kante, die in ein Blatt mündet und nicht nur $ enthält.

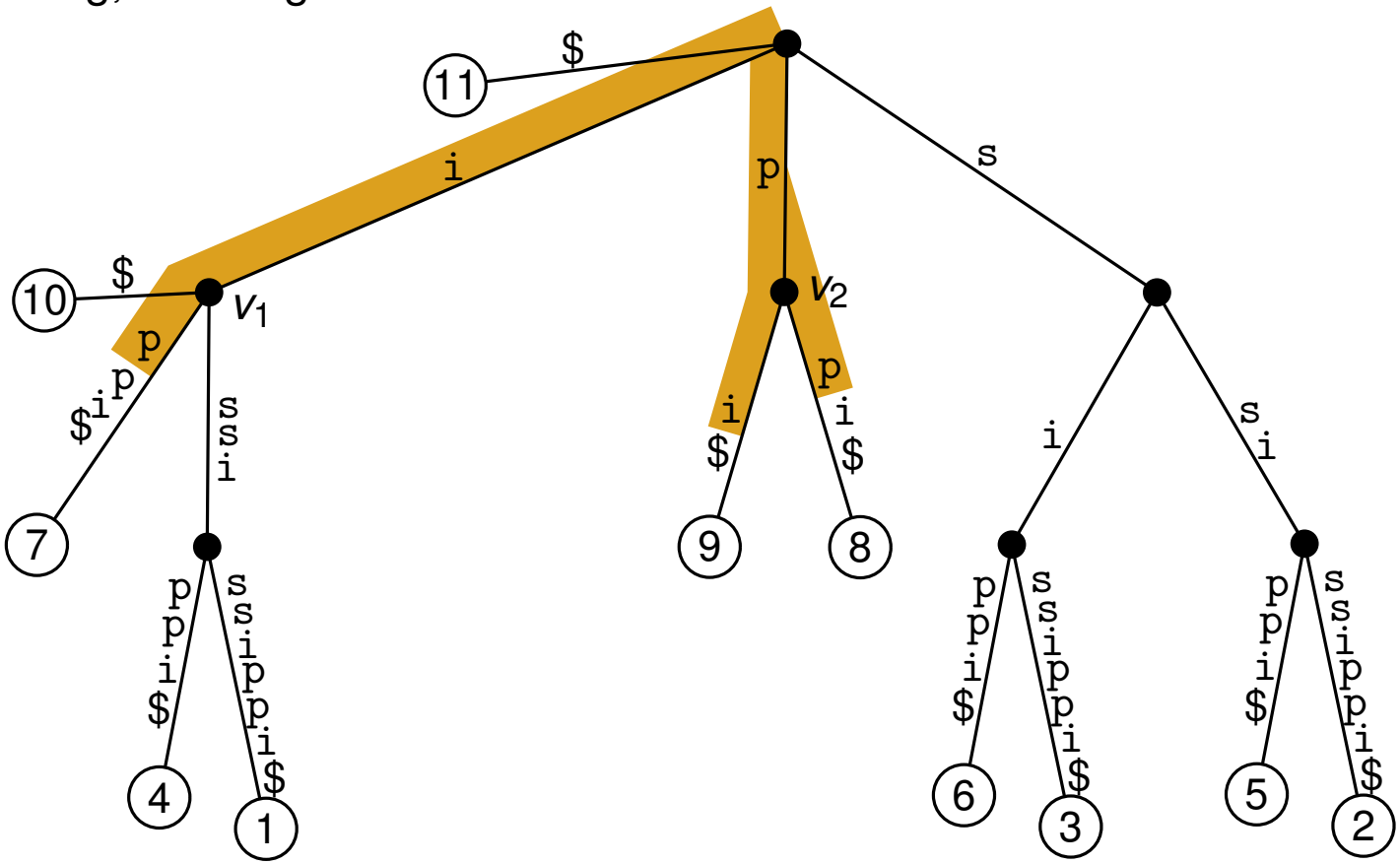
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 3

Berechnen Sie alle kürzesten eindeutigen Teilstrings in einem Text T.

Teilstring S ist eindeutig, falls er genau einmal in T auftaucht.

T=ississippi$

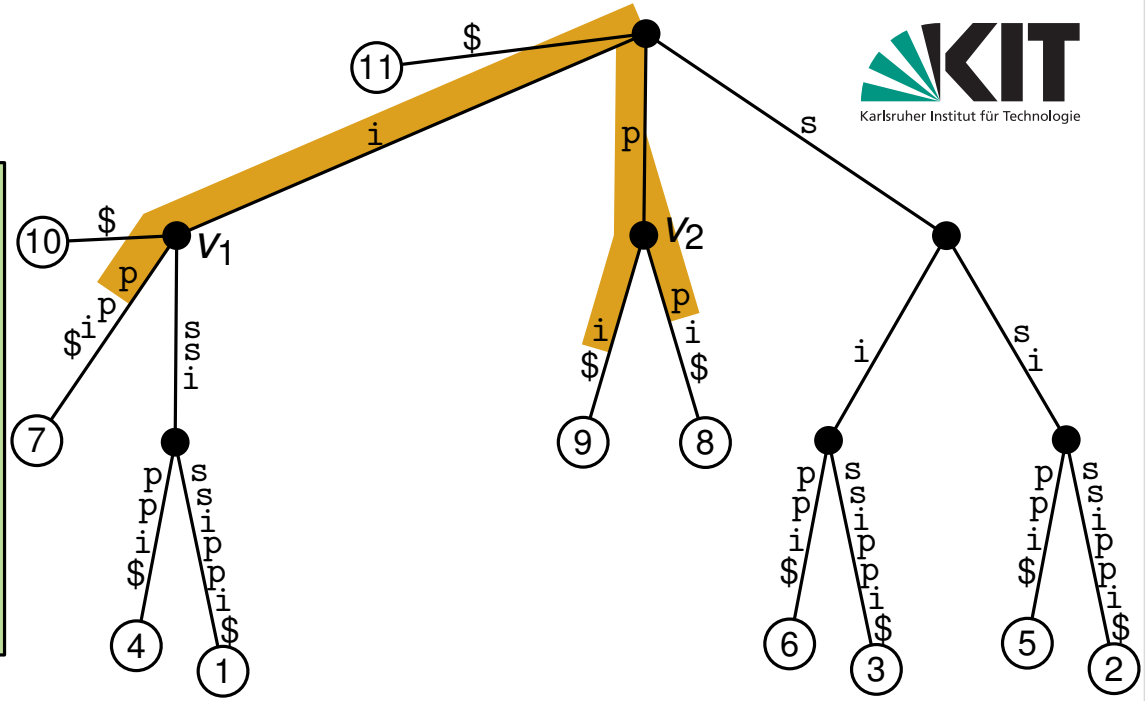


Beobachtung: Eindeutige Teilstrings enden auf einer Kante, die in ein Blatt mündet und nicht nur $ enthält.

Idee: Finde alle Knoten v mit geringster String-Tiefe, so dass v nicht Blatt ist und ein Kante zu einem Blatt besitzt, die nicht nur mit $ beschriftet ist.

Problem 3

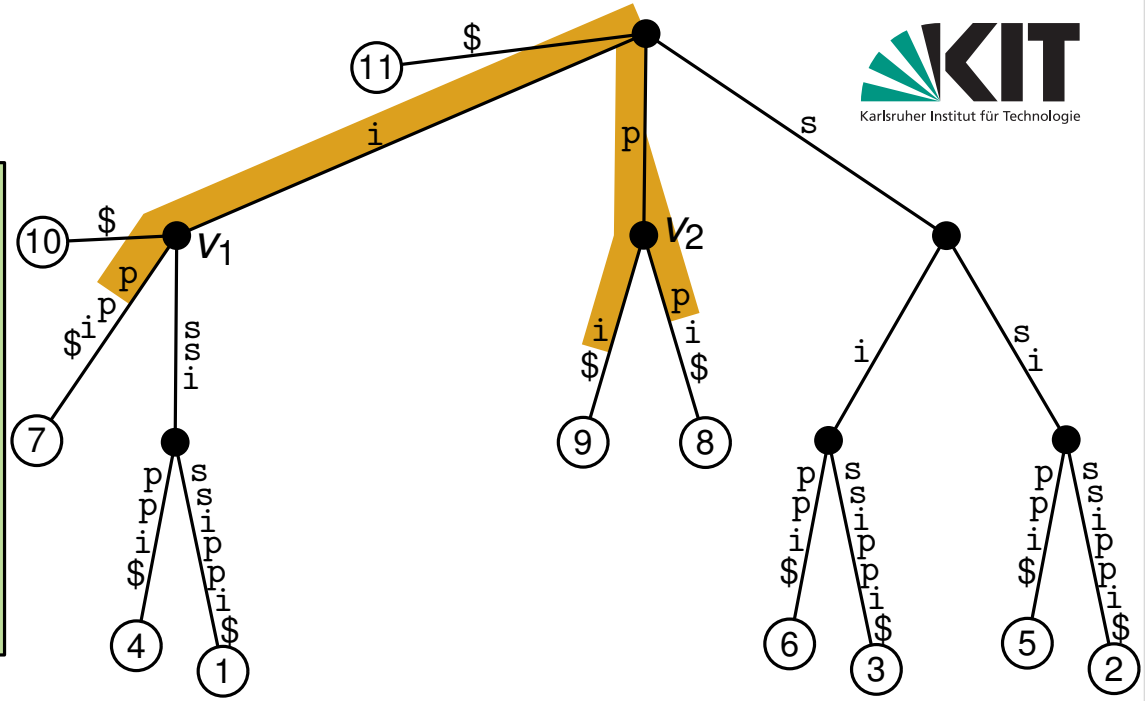
Eingabe: Suffixbaum für Text T.
Ausgabe Kürzeste eindeutige Teilstrings.
1. Schritt:
Bestimme Länge ℓ_{min} der kürzesten eindeutigen Teilstrings.
2. Schritt:
Gebe alle eindeutigen Teilstrings mit Länge ℓ_{min} aus.



Im Folgenden:
$d(v)$ = String-Tiefe von v (also wie viele Zeichen auf Pfad von Wurzel zu v liegen).
$B(u, v)$ = Beschriftung der Kante (u, v).

Problem 3

Eingabe: Suffixbaum für Text T.
Ausgabe Kürzeste eindeutige Teilstrings.
1. Schritt:
Bestimme Länge ℓ_{min} der kürzesten eindeutigen Teilstrings.
2. Schritt:
Gebe alle eindeutigen Teilstrings mit Länge ℓ_{min} aus.



1. Schritt:

Q = leere Warteschlange

$\ell_{min} \leftarrow n$

Füge Wurzel von S in Q ein.

solange *Q nicht leer* **tue**

  $u \leftarrow$ entferne erstes Element aus Q.

  für *ausgehende Kanten* (u, v) *von* u **tue**

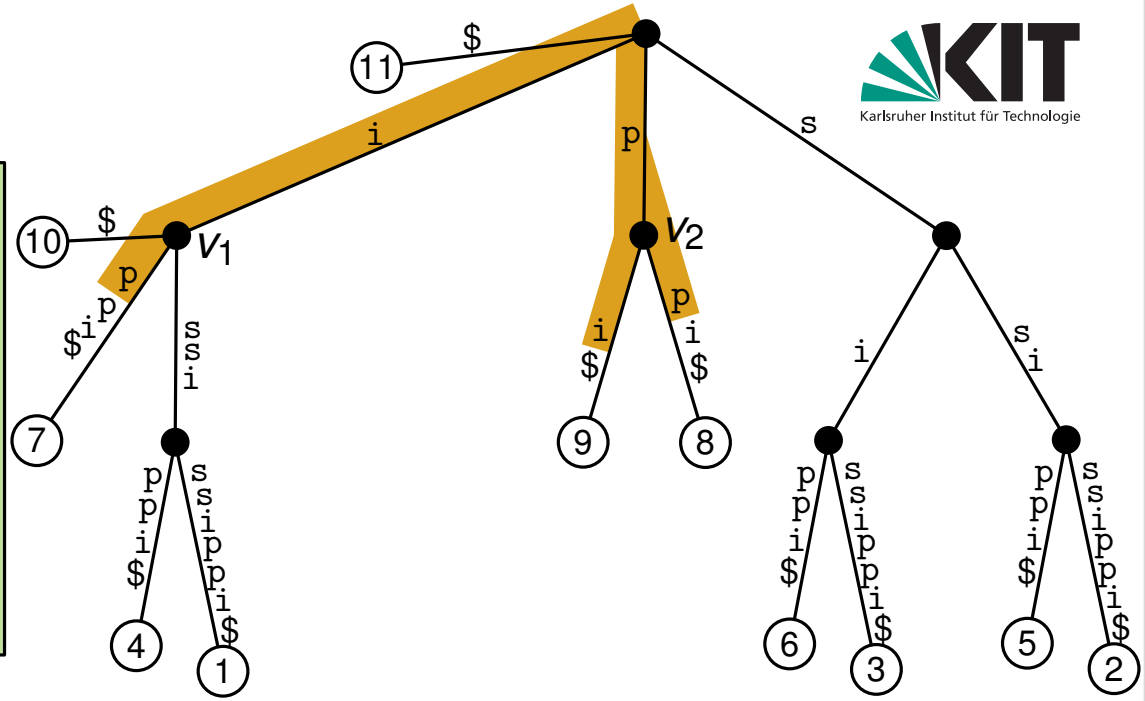
    wenn *v Blatt und* $|B(u, v)| > 1$ **dann** $\ell_{min} \leftarrow \min\{d(u) + 1, \ell_{min}\}$

    Füge v in Q ein

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 3

Eingabe: Suffixbaum für Text T.
Ausgabe Kürzeste eindeutige Teilstrings.
1. Schritt:
Bestimme Länge ℓ_{min} der kürzesten eindeutigen Teilstrings.
2. Schritt:
Gebe alle eindeutigen Teilstrings mit Länge ℓ_{min} aus.



1. Schritt:

Laufzeit: $O(n)$

Q = leere Warteschlange

$\ell_{min} \leftarrow n$

Füge Wurzel von S in Q ein.

solange *Q nicht leer* **tue**

  $u \leftarrow$ entferne erstes Element aus Q.

  für *ausgehende Kanten* (u, v) *von* u **tue**

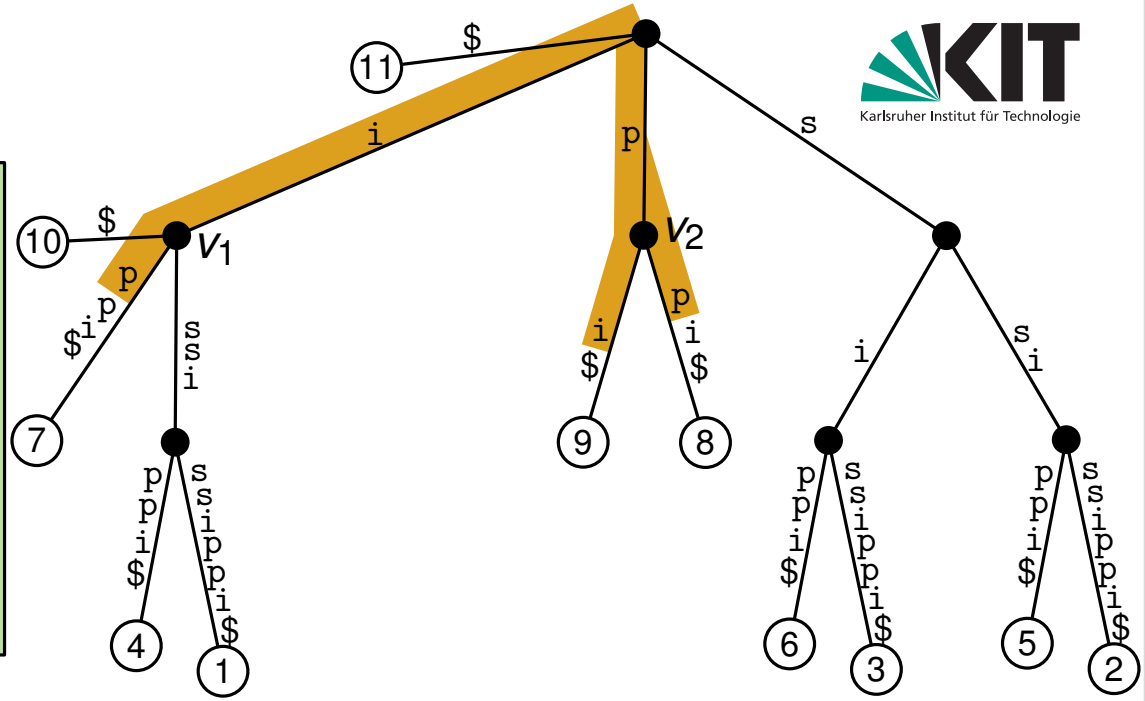
    wenn *v Blatt und* $|B(u, v)| > 1$ **dann** $\ell_{min} \leftarrow \min\{d(u) + 1, \ell_{min}\}$

    Füge v in Q ein

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 3

Eingabe: Suffixbaum für Text T.
Ausgabe Kürzeste eindeutige Teilstrings.
1. Schritt:
Bestimme Länge ℓ_{min} der kürzesten eindeutigen Teilstrings.
2. Schritt:
Gebe alle eindeutigen Teilstrings mit Länge ℓ_{min} aus.



2. Schritt:

Q = leere Warteschlange

Füge Wurzel von S in Q ein.

solange *Q nicht leer* **tue**

  $u \leftarrow$ entferne erstes Element aus Q.

  für *ausgehende Kanten* (u, v) *von* u **tue**

    wenn *v Blatt und* $|B(u, v)| > 1$ *und* $d(u) = \ell_{min} - 1$ **dann**

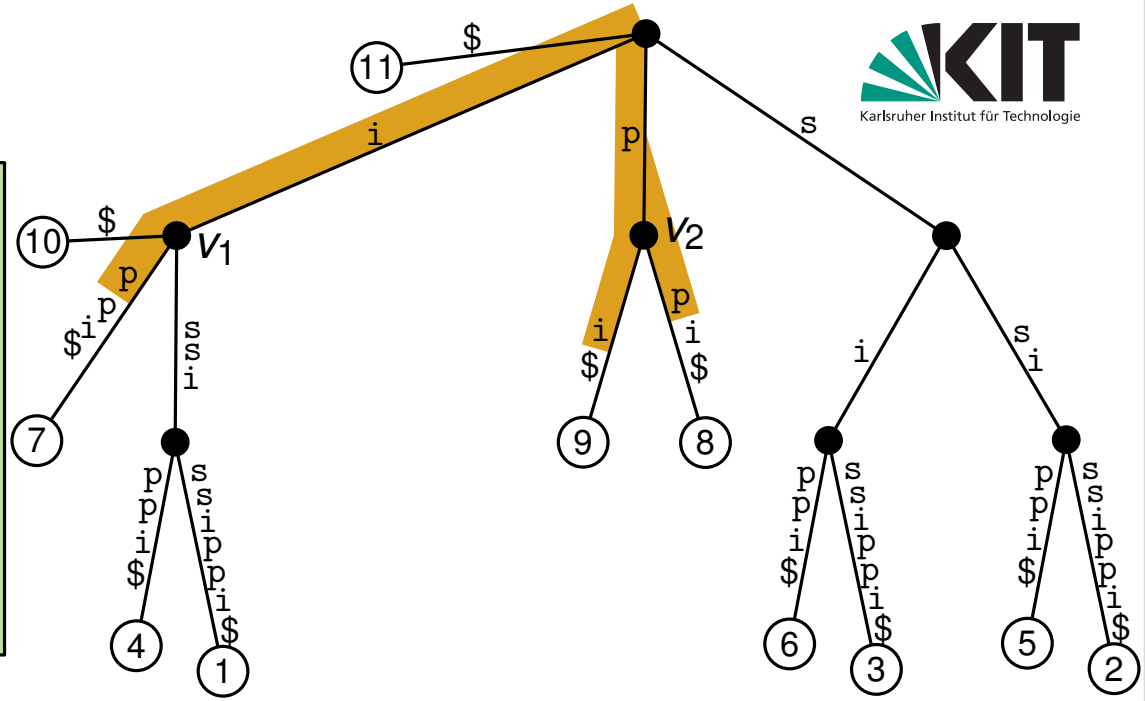
      $s \leftarrow$ String entlang Pfad von Wurzel zu u + erstes Zeichen auf (u, v).

      Gebe s aus.

    Füge v in Q ein

Problem 3

Eingabe: Suffixbaum für Text T.
Ausgabe Kürzeste eindeutige Teilstrings.
1. Schritt:
Bestimme Länge ℓ_{min} der kürzesten eindeutigen Teilstrings.
2. Schritt:
Gebe alle eindeutigen Teilstrings mit Länge ℓ_{min} aus.



2. Schritt:

Laufzeit: $O(n)$

Q = leere Warteschlange

Füge Wurzel von S in Q ein.

solange *Q nicht leer* **tue**

  $u \leftarrow$ entferne erstes Element aus Q.

  für *ausgehende Kanten* (u, v) *von* u **tue**

    wenn *v Blatt und* $|B(u, v)| > 1$ *und* $d(u) = \ell_{min} - 1$ **dann**

      $s \leftarrow$ String entlang Pfad von Wurzel zu u + erstes Zeichen auf (u, v).

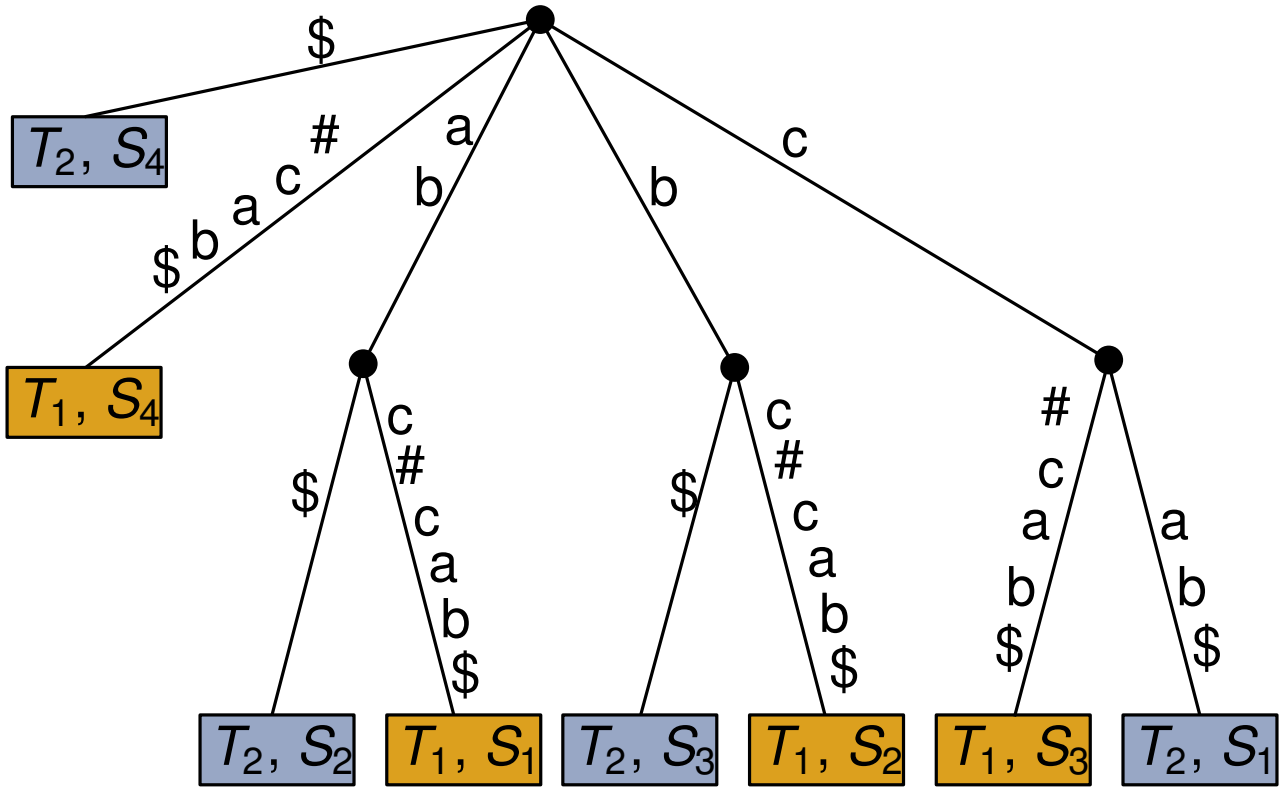
      Gebe s aus.

    Füge v in Q ein

Problem 4

Gegeben: Text T_1 und T_2.
Gesucht: Längster gemeinsamer Teilstring von T_1 und T_2.
Hinweis: Verwende generalisierten Suffixbaum für $T_1 \# T_2 \$$

T_1 =abc
T_2 =cab



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 4

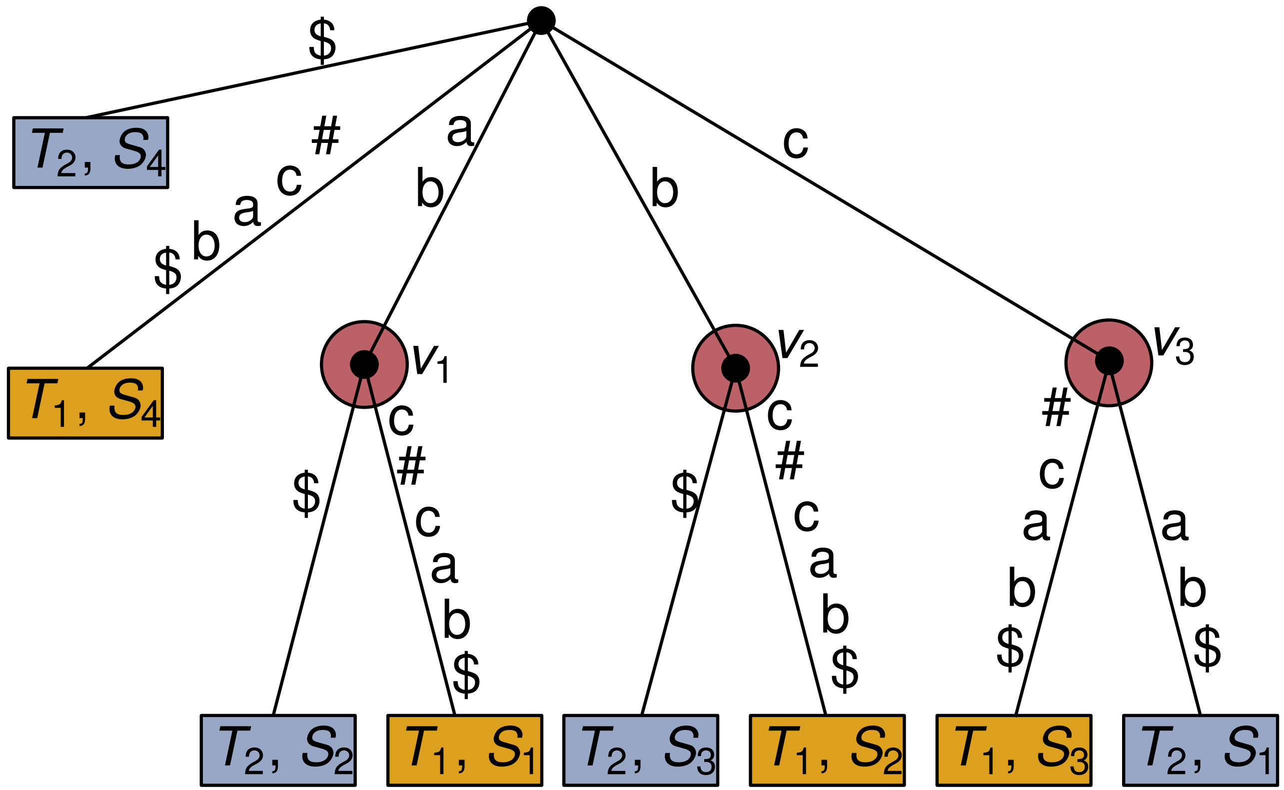
Gegeben: Text T_1 und T_2.

Gesucht: Längster gemeinsamer Teilstring von T_1 und T_2.

Hinweis: Verwende generalisierten Suffixbaum für $T_1 \# T_2 \$$

T_1 =abc

T_2 =cab

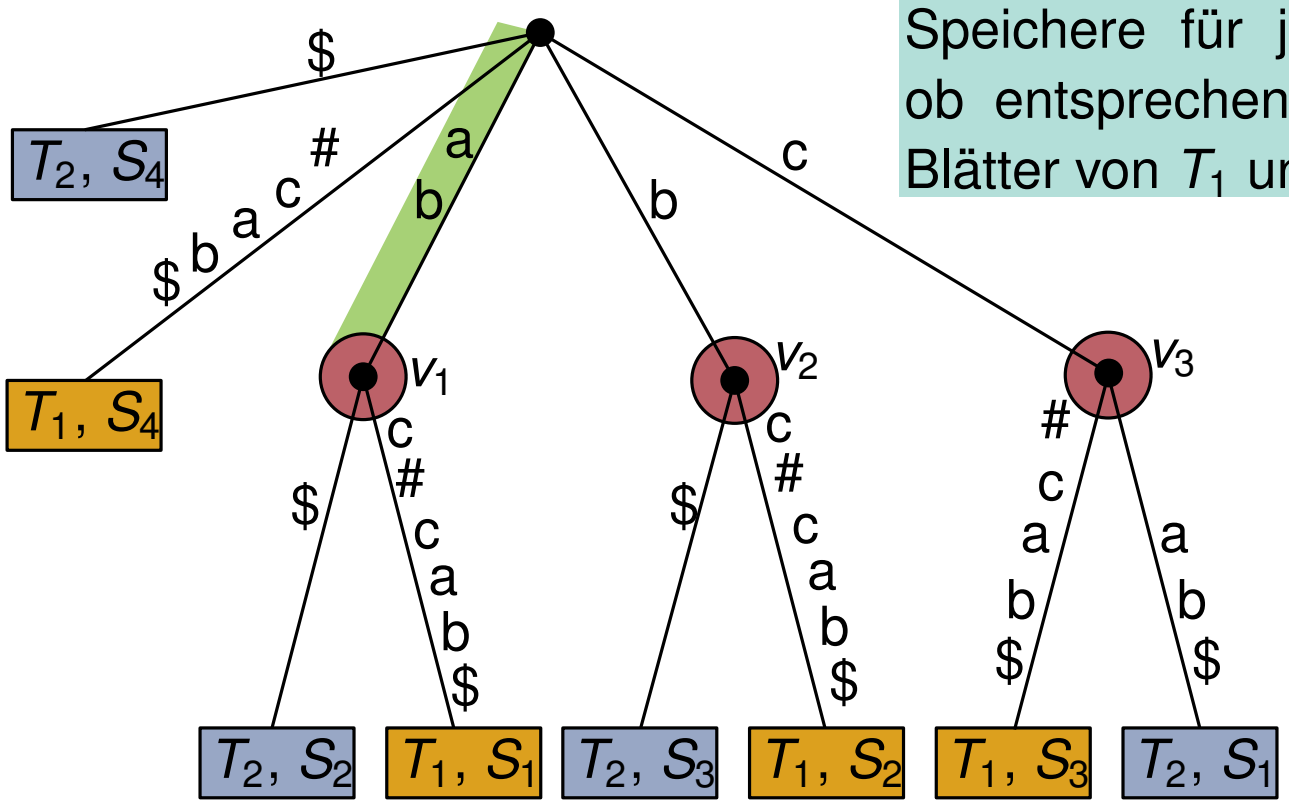


Teilstring von T_1 und T_2 endet in einem Knoten v, so dass angehängter Teilbaum von v sowohl Blätter von T_1 als auch von T_2 enthält.

Problem 4

Gegeben: Text T_1 und T_2.
Gesucht: Längster gemeinsamer Teilstring von T_1 und T_2.
Hinweis: Verwende generalisierten Suffixbaum für $T_1 \# T_2 \$$

T_1 =abc
T_2 =cab



Speichere für jeden Knoten ob entsprechender Teilbaum Blätter von T_1 und T_2 enthält.

Teilstring von T_1 und T_2 endet in einem Knoten v, so dass angehängter Teilbaum von v sowohl Blätter von T_1 als auch von T_2 enthält.

Idee: Wähle von diesen Knoten denjenigen, der maximale String-Tiefe besitzt.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Karlsruher Institut für Technologie

Evaluation

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Einschub: Radix-Sort

Eingabe: Menge $a_1, \ldots, a_n$ an Zahlen.

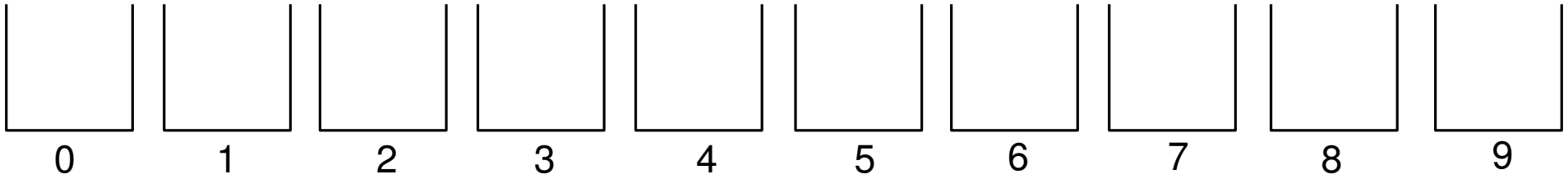
Annahme: Jede Zahl hat m Ziffern.

Ausgabe: Sortierung von $a_1, \ldots, a_n$ in $O(m \cdot n)$ Zeit.

Sortierung der einzelnen Stellen mithilfe von Buckets. Beginne mit niedrigster Stelle.

Eingabe 329 457 657 839 436 720 355

1. Schritt: Sortierung nach i-ter Stelle.



2. Schritt: Werte einsammeln (dabei Reihenfolge erhalten).

Einschub: Radix-Sort

Eingabe: Menge $a_1, \ldots, a_n$ an Zahlen.

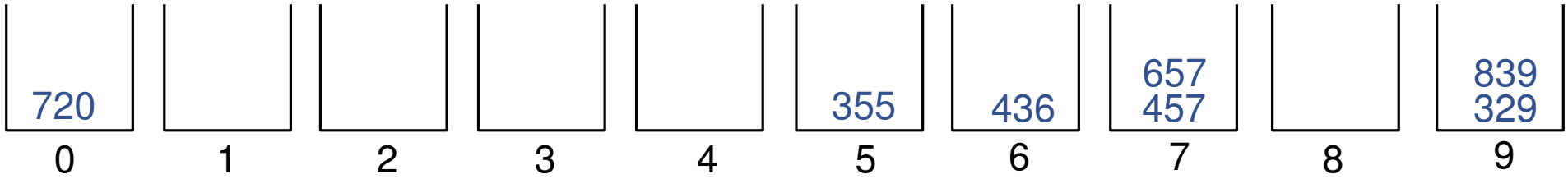
Annahme: Jede Zahl hat m Ziffern.

Ausgabe: Sortierung von $a_1, \ldots, a_n$ in $O(m \cdot n)$ Zeit.

Sortierung der einzelnen Stellen mithilfe von Buckets. Beginne mit niedrigster Stelle.

Eingabe 329 457 657 839 436 720 355

1. Schritt: Sortierung nach i-ter Stelle.



2. Schritt: Werte einsammeln (dabei Reihenfolge erhalten).

720 355 436 457 657 329 839

Einschub: Radix-Sort

Eingabe: Menge $a_1, \ldots, a_n$ an Zahlen.

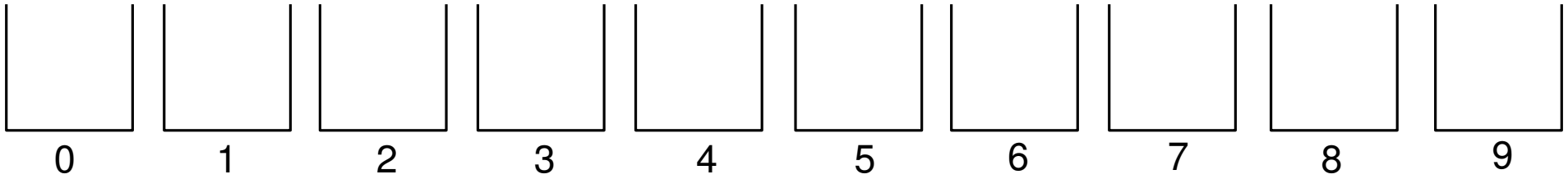
Annahme: Jede Zahl hat m Ziffern.

Ausgabe: Sortierung von $a_1, \ldots, a_n$ in $O(m \cdot n)$ Zeit.

Sortierung der einzelnen Stellen mithilfe von Buckets. Beginne mit niedrigster Stelle.

Eingabe	329	457	657	839	436	720	355
1. Zwischenergebnis:	720	355	436	457	657	329	839

1. Schritt: Sortierung nach i-ter Stelle.



2. Schritt: Werte einsammeln (dabei Reihenfolge erhalten).

Einschub: Radix-Sort

Eingabe: Menge $a_1, \ldots, a_n$ an Zahlen.

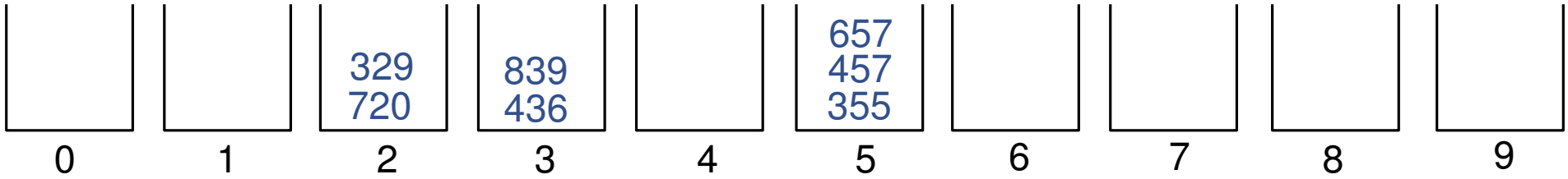
Annahme: Jede Zahl hat m Ziffern.

Ausgabe: Sortierung von $a_1, \ldots, a_n$ in $O(m \cdot n)$ Zeit.

Sortierung der einzelnen Stellen mithilfe von Buckets. Beginne mit niedrigster Stelle.

Eingabe	329	457	657	839	436	720	355
1. Zwischenergebnis:	720	355	436	457	657	329	839

1. Schritt: Sortierung nach i-ter Stelle.



2. Schritt: Werte einsammeln (dabei Reihenfolge erhalten).

720 329 436 839 355 457 657

Einschub: Radix-Sort

Eingabe: Menge $a_1, \ldots, a_n$ an Zahlen.

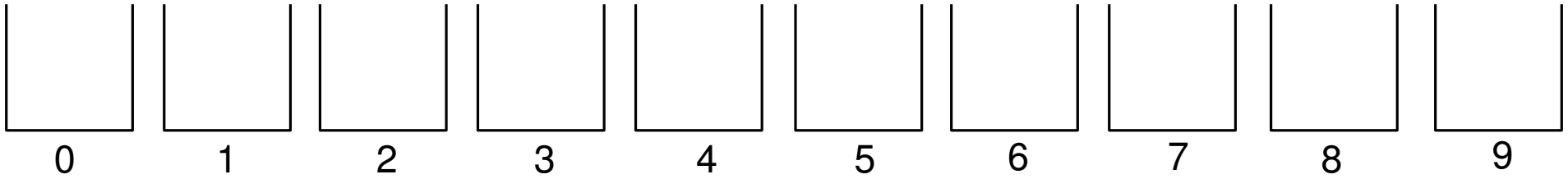
Annahme: Jede Zahl hat m Ziffern.

Ausgabe: Sortierung von $a_1, \ldots, a_n$ in $O(m \cdot n)$ Zeit.

Sortierung der einzelnen Stellen mithilfe von Buckets. Beginne mit niedrigster Stelle.

Eingabe	329	457	657	839	436	720	355
1. Zwischenergebnis:	720	355	436	457	657	329	839
2. Zwischenergebnis:	720	329	436	839	355	457	657

1. Schritt: Sortierung nach i-ter Stelle.



2. Schritt: Werte einsammeln (dabei Reihenfolge erhalten).

Einschub: Radix-Sort

Eingabe: Menge $a_1, \ldots, a_n$ an Zahlen.

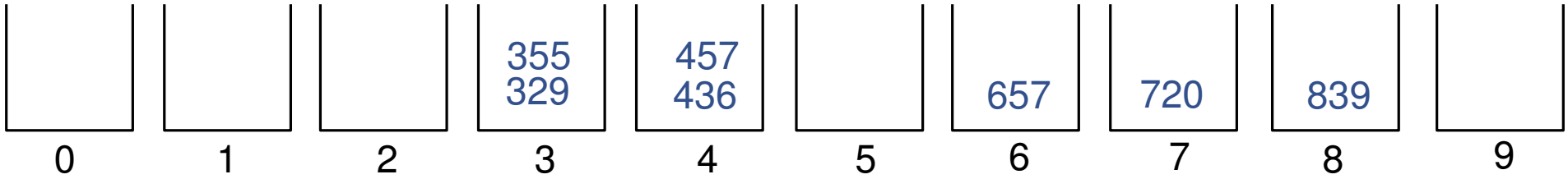
Annahme: Jede Zahl hat m Ziffern.

Ausgabe: Sortierung von $a_1, \ldots, a_n$ in $O(m \cdot n)$ Zeit.

Sortierung der einzelnen Stellen mithilfe von Buckets. Beginne mit niedrigster Stelle.

Eingabe	329	457	657	839	436	720	355
1. Zwischenergebnis:	720	355	436	457	657	329	839
2. Zwischenergebnis:	720	329	436	839	355	457	657

1. Schritt: Sortierung nach i-ter Stelle.



2. Schritt: Werte einsammeln (dabei Reihenfolge erhalten).

329 355 436 457 657 720 839

Einschub: Radix-Sort

Eingabe: Menge $a_1, \ldots, a_n$ an Zahlen.

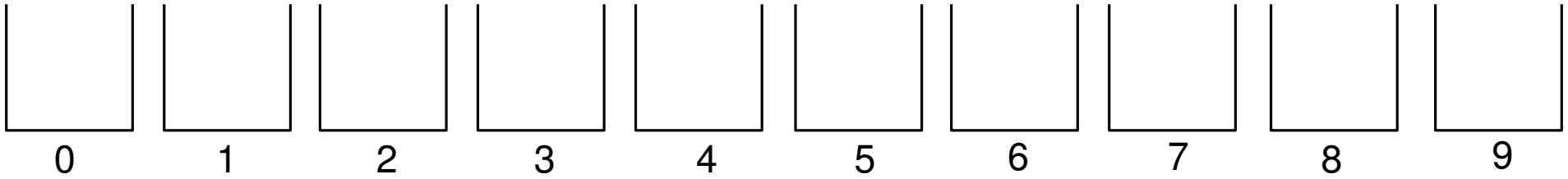
Annahme: Jede Zahl hat m Ziffern.

Ausgabe: Sortierung von $a_1, \ldots, a_n$ in $O(m \cdot n)$ Zeit.

Sortierung der einzelnen Stellen mithilfe von Buckets. Beginne mit niedrigster Stelle.

Eingabe	329	457	657	839	436	720	355
1. Zwischenergebnis:	720	355	436	457	657	329	839
2. Zwischenergebnis:	720	329	436	839	355	457	657
Ergebnis:	329	355	436	457	657	720	839

1. Schritt: Sortierung nach i-ter Stelle.



2. Schritt: Werte einsammeln (dabei Reihenfolge erhalten).

Problem 1

Berechnen Sie das Suffix-Array für das Wort `mississippi`:

Definition: Ein *Suffix-Array* A eines Textes T ist eine Permutation von $\{1, \ldots, n\}$, sodass $S_{A[i]}$ das i-kleinste Suffix in lexikographischer Ordnung ist: $S_{A[i-1]} < S_{A[i]}$ für alle $1 < i \leq n$.

1	S_{10}	i
2	S_7	i p p i
3	S_4	i s s i p p i
4	S_1	i s s i s s i p p i
5	S_0	m i s s i s s i p p i
6	S_9	p i
7	S_8	p p i
8	S_6	s i p p i
9	S_3	s i s s i p p i
10	S_5	s s i p p i
11	S_2	s s i s s i p p i

Suffix-Array gibt Sortierung der Suffixe an.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

Berechnen Sie das Suffix-Array für das Wort `mississippi`:

Definition: Ein *Suffix-Array* A eines Textes T ist eine Permutation von $\{1, \ldots, n\}$, sodass $S_{A[i]}$ das i-kleinste Suffix in lexikographischer Ordnung ist: $S_{A[i-1]} < S_{A[i]}$ für alle $1 < i \leq n$.

1	S_{10}	i
2	S_7	i p p i
3	S_4	i s s i p p i
4	S_1	i s s i s s i p p i
5	S_0	m i s s i s s i p p i
6	S_9	p i
7	S_8	p p i
8	S_6	s i p p i
9	S_3	s i s s i p p i
10	S_5	s s i p p i
11	S_2	s s i s s i p p i

Suffix-Array gibt Sortierung der Suffixe an.

Zwei Möglichkeiten:

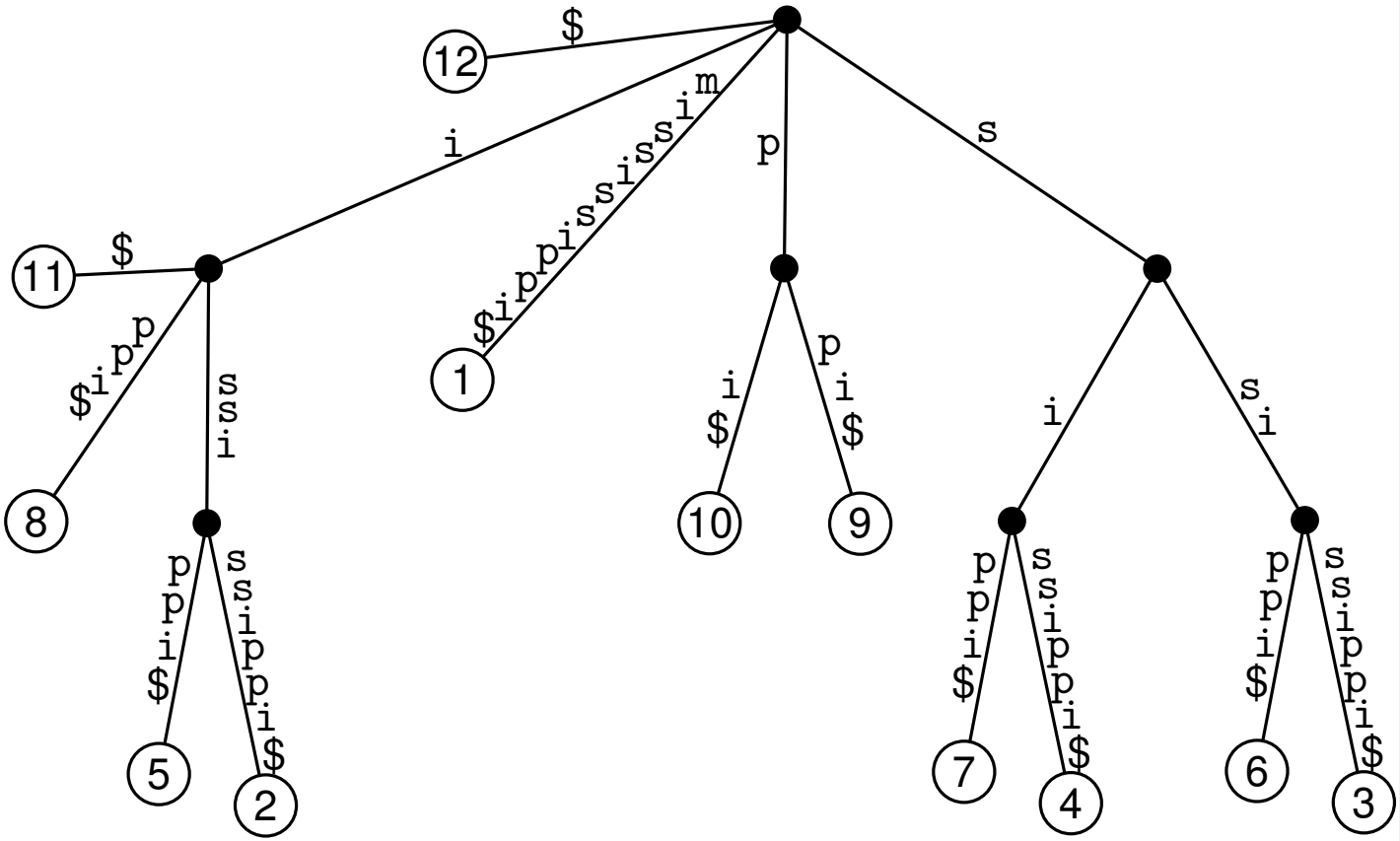
1. Mithilfe des entsprechenden Suffixbaum ($O(n^2)$ Zeit).
2. Direkt in $O(n)$.

Problem 1

Berechnen Sie das Suffix-Array für das Wort `mississippi`:

Wie Suffix-Array aus einem Suffix-Baum erstellen?

12	$
11	i$
8	ippi$
5	issippi$
2	ississippi$
1	mississippi$
10	pi$
9	ppi $
7	sippi$
4	sissippi$
6	ssippi$
3	ssissippi$

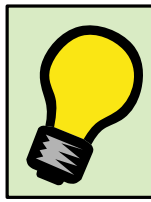


Suffix-Array gibt lexikographische Ordnung der Suffixe an!

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

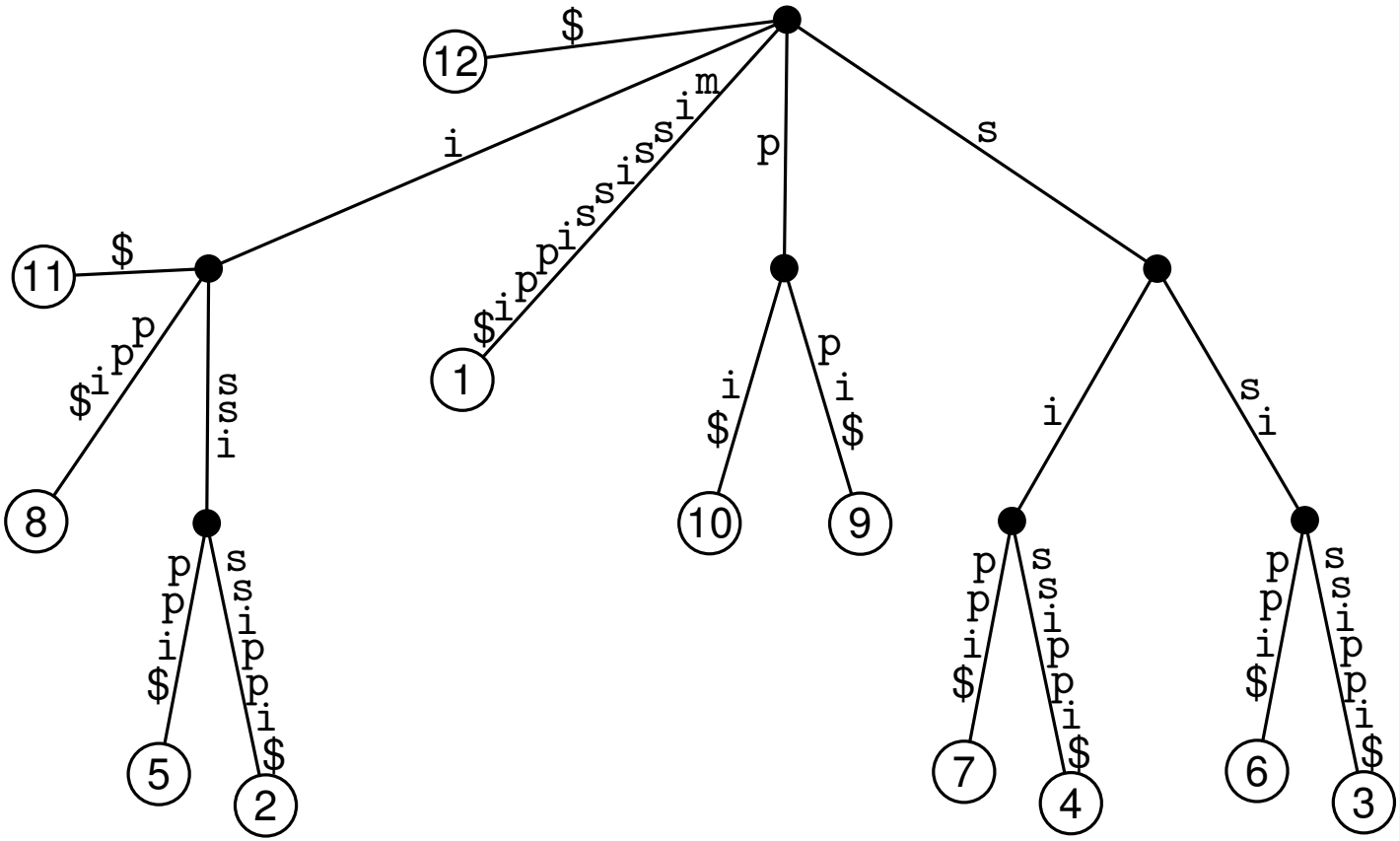
Problem 1

Berechnen Sie das Suffix-Array für das Wort `mississippi`:



Sei S Suffixbaum von T. A entspricht der Reihenfolge der Blätter, die sich ergibt, wenn Tiefensuche auf S mit lexikographischer Ordnung angewendet wird.

12	$
11	i$
8	ippi$
5	issippi$
2	ississippi$
1	mississippi$
10	pi$
9	ppi $
7	sippi$
4	sissippi$
6	ssippi$
3	ssissippi$

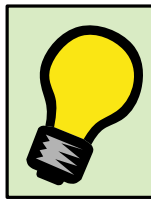


Suffix-Array gibt lexikographische Ordnung der Suffixe an!

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

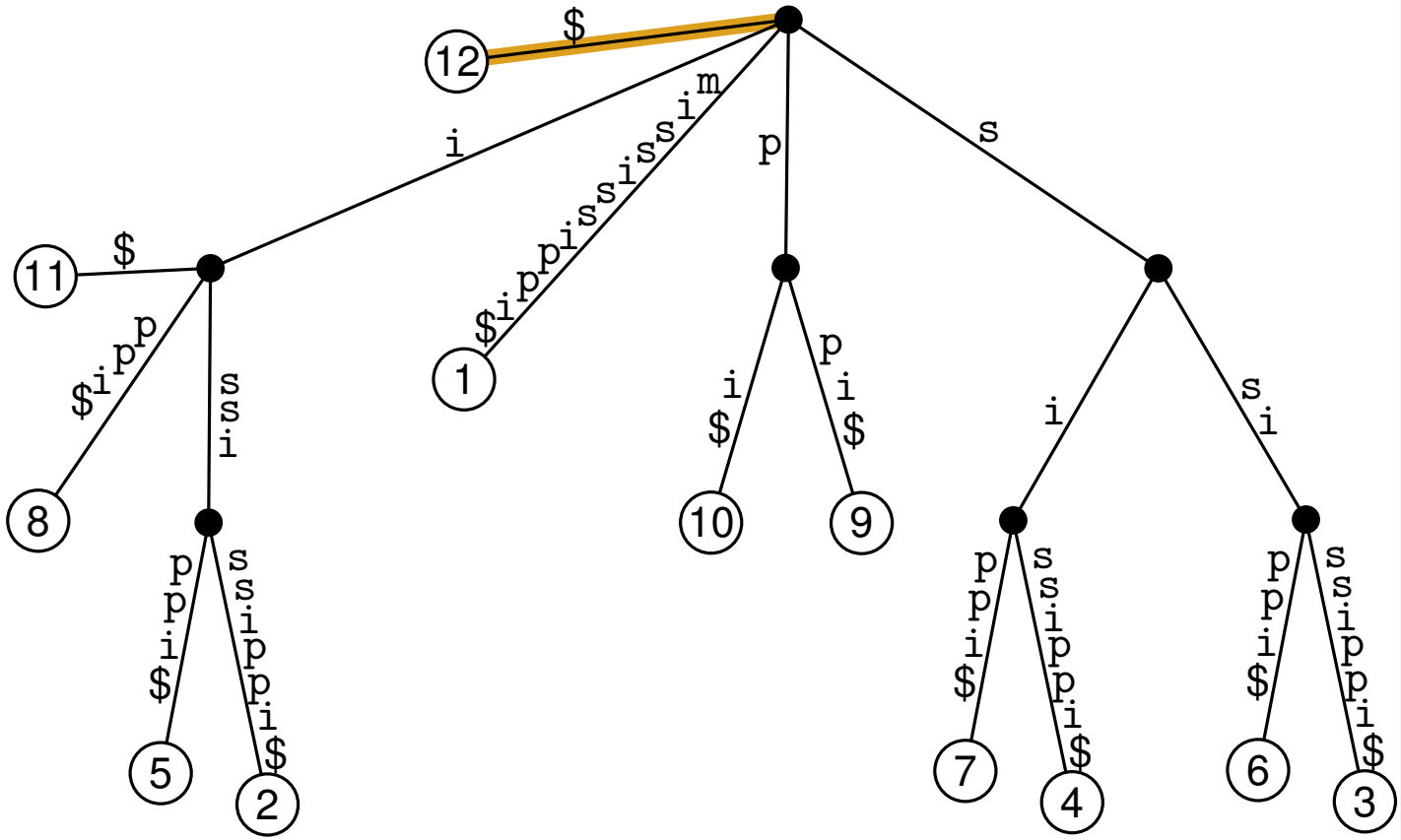
Problem 1

Berechnen Sie das Suffix-Array für das Wort `mississippi`:



Sei S Suffixbaum von T. A entspricht der Reihenfolge der Blätter, die sich ergibt, wenn Tiefensuche auf S mit lexikographischer Ordnung angewendet wird.

12	$
11	i$
8	ippi$
5	issippi$
2	ississippi$
1	mississippi$
10	pi$
9	ppi $
7	sippi$
4	sissippi$
6	ssippi$
3	ssissippi$



Suffix-Array gibt lexikographische Ordnung der Suffixe an!

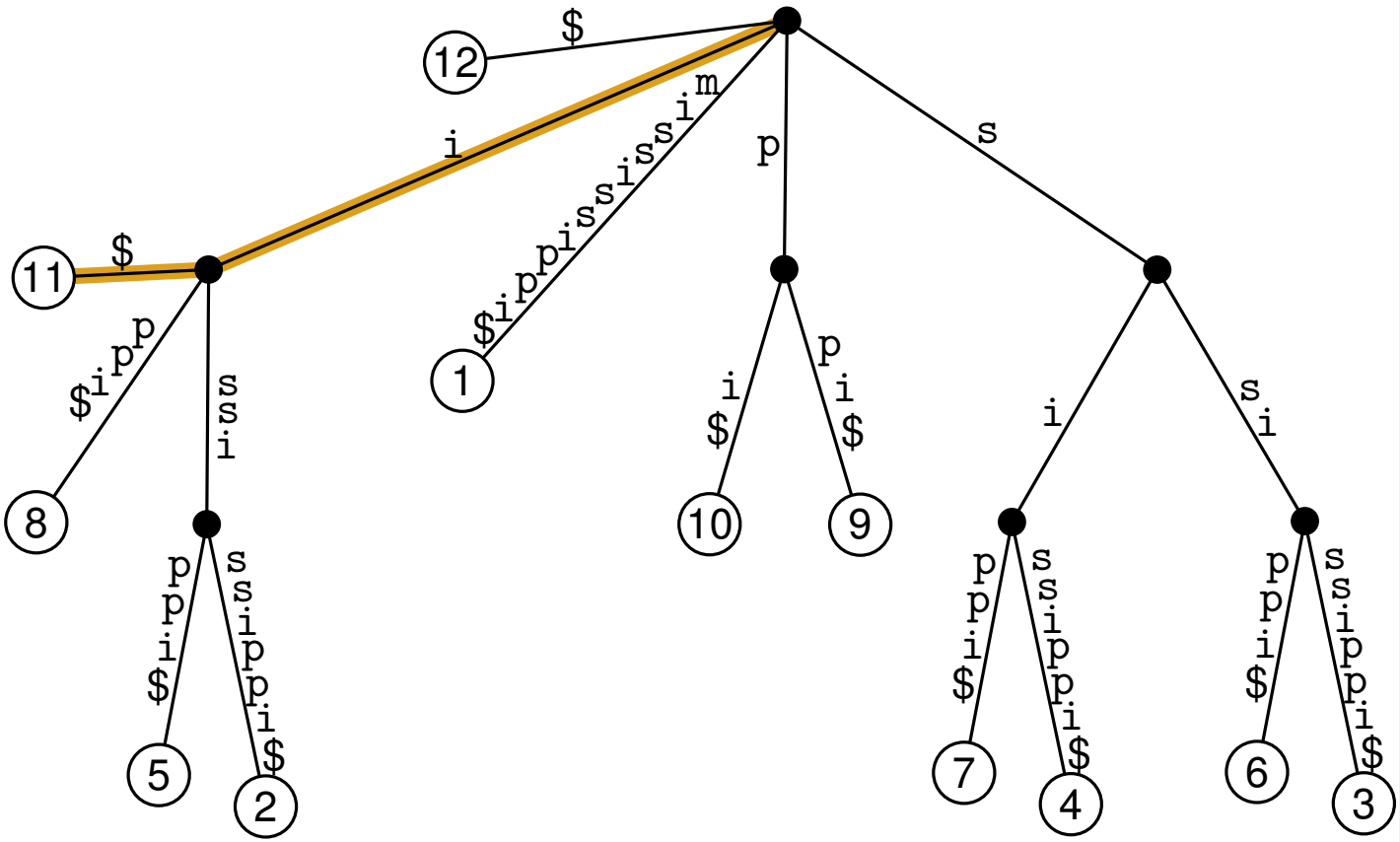
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

Berechnen Sie das Suffix-Array für das Wort `mississippi`:

Sei S Suffixbaum von T. A entspricht der Reihenfolge der Blätter, die sich ergibt, wenn Tiefensuche auf S mit lexikographischer Ordnung angewendet wird.

12	$
11	i$
8	ippi$
5	issippi$
2	ississippi$
1	mississippi$
10	pi$
9	ppi $
7	sippi$
4	sissippi$
6	ssippi$
3	ssissippi$



Suffix-Array gibt lexikographische Ordnung der Suffixe an!

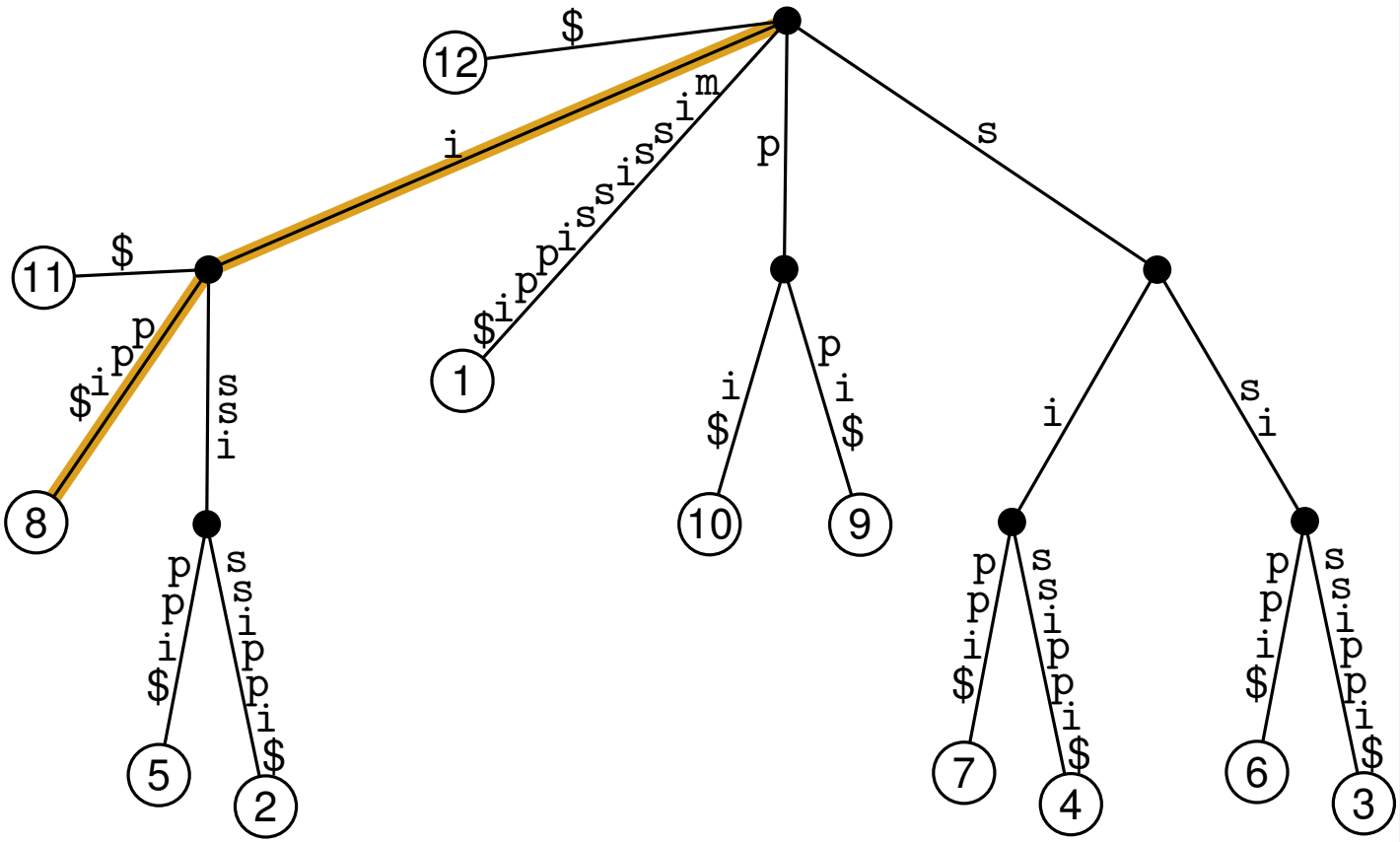
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

Berechnen Sie das Suffix-Array für das Wort `mississippi`:

Sei S Suffixbaum von T. A entspricht der Reihenfolge der Blätter, die sich ergibt, wenn Tiefensuche auf S mit lexikographischer Ordnung angewendet wird.

12	$
11	i$
8	ippi$
5	issippi$
2	ississippi$
1	mississippi$
10	pi$
9	ppi $
7	sippi$
4	sissippi$
6	ssippi$
3	ssissippi$

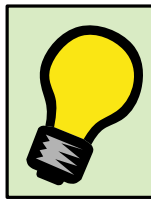


Suffix-Array gibt lexikographische Ordnung der Suffixe an!

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

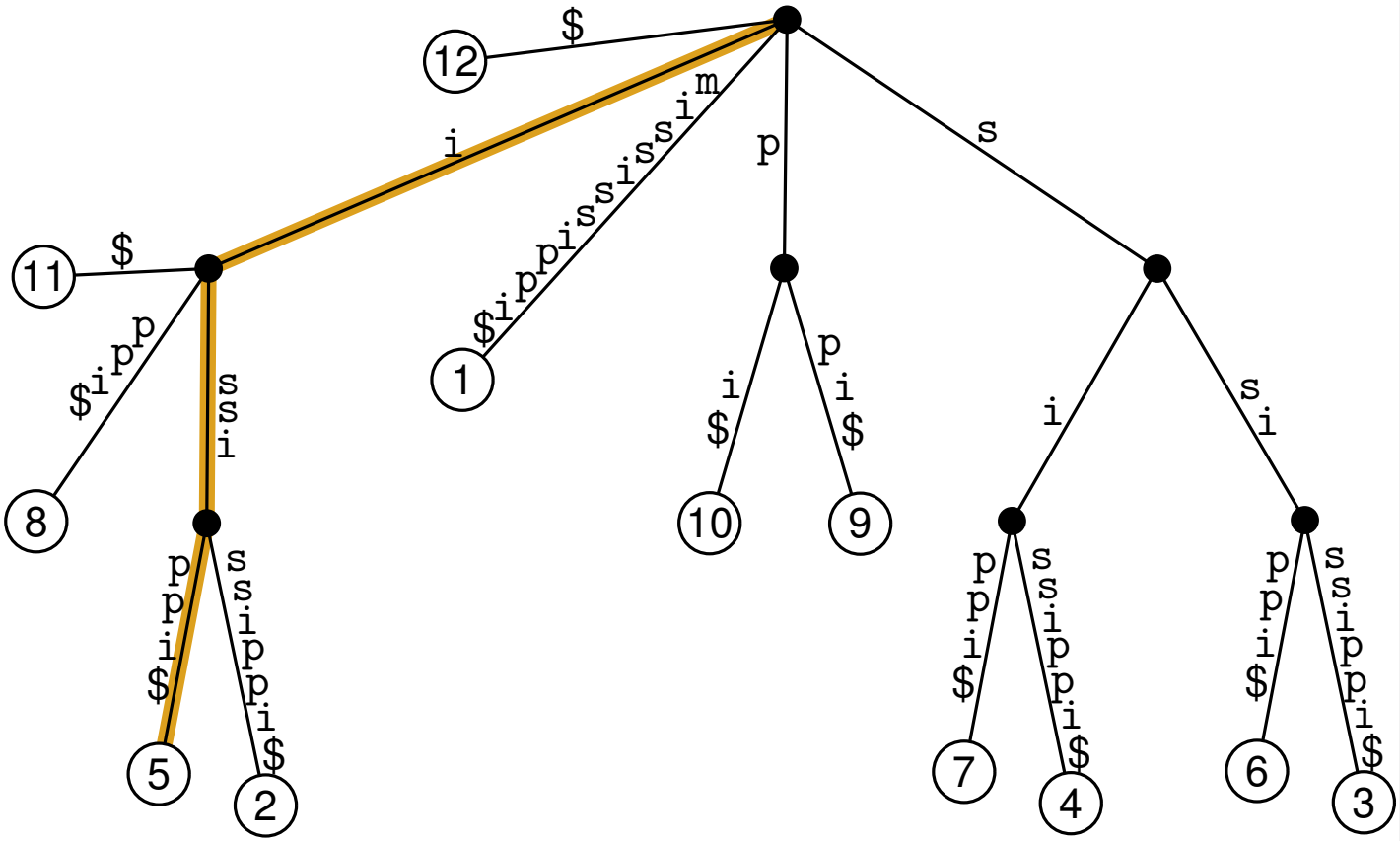
Problem 1

Berechnen Sie das Suffix-Array für das Wort `mississippi`:



Sei S Suffixbaum von T. A entspricht der Reihenfolge der Blätter, die sich ergibt, wenn Tiefensuche auf S mit lexikographischer Ordnung angewendet wird.

12	$
11	i$
8	ippi$
5	issippi$
2	ississippi$
1	mississippi$
10	pi$
9	ppi $
7	sippi$
4	sissippi$
6	ssippi$
3	ssissippi$

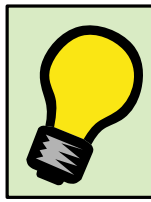


Suffix-Array gibt lexikographische Ordnung der Suffixe an!

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

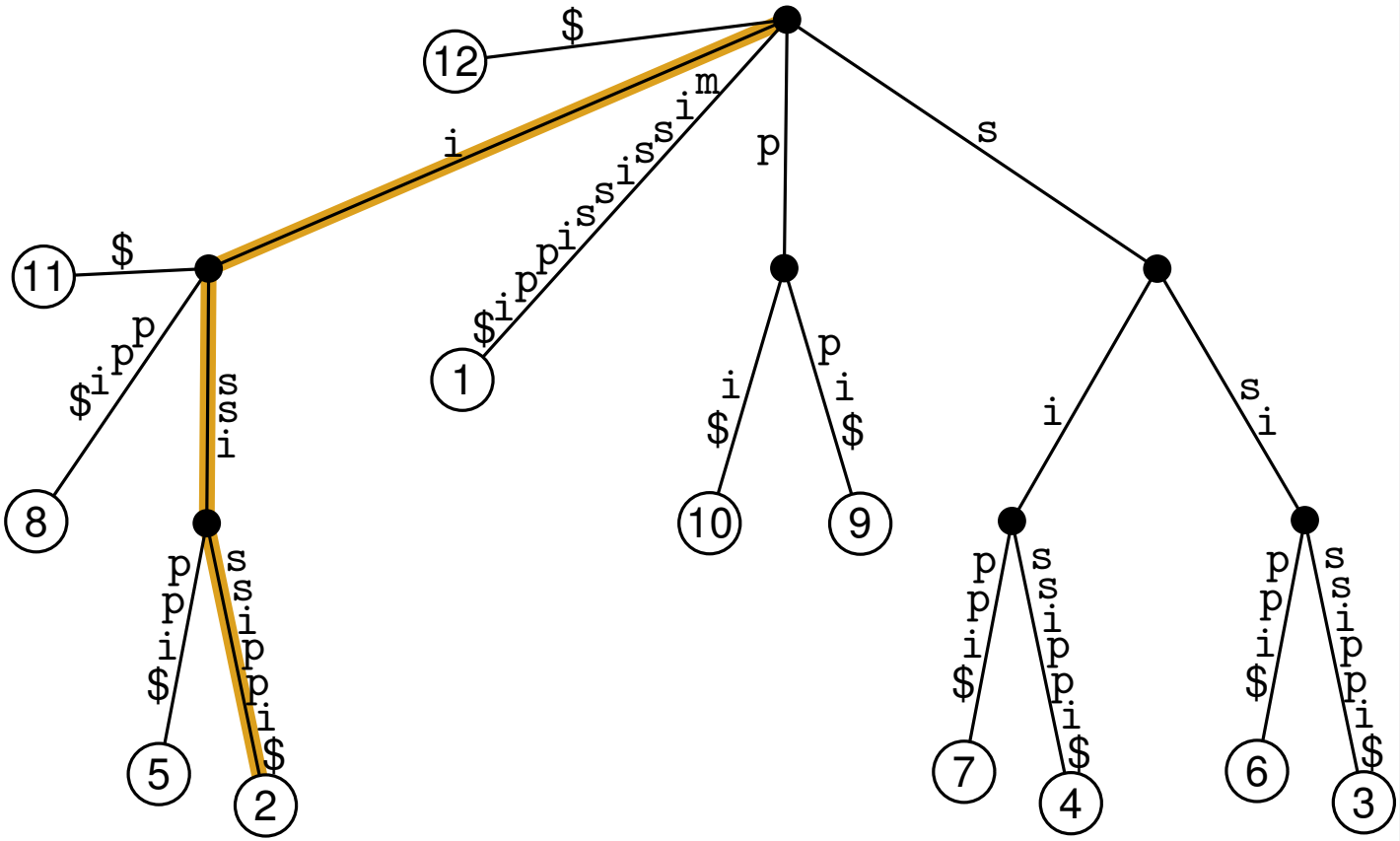
Problem 1

Berechnen Sie das Suffix-Array für das Wort `mississippi`:



Sei S Suffixbaum von T. A entspricht der Reihenfolge der Blätter, die sich ergibt, wenn Tiefensuche auf S mit lexikographischer Ordnung angewendet wird.

12	$
11	i$
8	ippi$
5	issippi$
2	ississippi$
1	mississippi$
10	pi$
9	ppi $
7	sippi$
4	sissippi$
6	ssippi$
3	ssissippi$



Suffix-Array gibt lexikographische Ordnung der Suffixe an!

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Konstruktion von Suffix-Arrays

Eingabe: Text $T := t_0 t_1 \ldots t_{n-1}$

	0	1	2	3	4	5	6	7	8	9	10
T=	m	i	s	s	i	s	s	i	p	p	i

Gesucht: Suffix-Array A von T, d.h. lexikographische Sortierung von Suffixen von T.

Konstruktion von Suffix-Arrays

Eingabe: Text $T := t_0 t_1 \ldots t_{n-1}$

	0	1	2	3	4	5	6	7	8	9	10
T=	m	i	s	s	i	s	s	i	p	p	i

Gesucht: Suffix-Array A von T, d.h. lexikographische Sortierung von Suffixen von T.

Kürze $x \mod y = z$ mit $x \equiv z(y)$ ab.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

$\mathcal{S}$ = Suffixe von T

$\mathcal{S}_0$ = Suffixe mit Index $i \equiv 0(3)$

$\mathcal{S}_1$ = Suffixe mit Index $i \equiv 1(3)$

$\mathcal{S}_2$ = Suffixe mit Index $i \equiv 2(3)$

Konstruktion von Suffix-Arrays

Eingabe: Text $T := t_0 t_1 \ldots t_{n-1}$

	0	1	2	3	4	5	6	7	8	9	10
T=	m	i	s	s	i	s	s	i	p	p	i

Gesucht: Suffix-Array A von T, d.h. lexikographische Sortierung von Suffixen von T.

Kürze $x \mod y = z$ mit $x \equiv z(y)$ ab.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

$\mathcal{S}$ = Suffixe von T

$\mathcal{S}_0$ = Suffixe mit Index $i \equiv 0(3)$

$\mathcal{S}_1$ = Suffixe mit Index $i \equiv 1(3)$

$\mathcal{S}_2$ = Suffixe mit Index $i \equiv 2(3)$

SUFFIXARRAY(Text $T = t_0 t_1 \ldots t_{n-1}$)
wenn $n = O(1)$ **dann**
  Konstruiere A in $O(1)$ Zeit.
sonst
  Berechne Suffix-Array A_{12} für $\mathcal{S}_1 \cup \mathcal{S}_2$.
  Berechne Suffix-Array A_0 für $\mathcal{S}_0$ basierend auf A_{12}.
  Vermenge A_{12} mit A_0.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

1. Schritt: Erstelle Suffix-Array A_{12}

Eingabe: Text $T := t_0 t_1 \ldots t_{n-1}$

	0	1	2	3	4	5	6	7	8	9	10
T=	m	i	s	s	i	s	s	i	p	p	i

Fasse Tripel $[t_i t_{i+1} t_{i+2}]$ mit $i \not\equiv 0(3)$ als eigenes Zeichen auf. Fülle gegebenfalls mit \$ auf.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Konstruiere für $k \in \{1, 2\}$ den String

$$R_k = [t_k t_{k+1} t_{k+2}][t_{k+3} t_{k+4} t_{k+5}][t_{k+6} t_{k+7} t_{k+8}] \ldots$$

R_1=[iss][iss][ipp][i\$\$]

R_2=[ssi][ssi][ppi]

$\mathcal{S}_0$ = Suffixe mit Index $i \equiv 0(3)$

$\mathcal{S}_1$ = Suffixe mit Index $i \equiv 1(3)$

$\mathcal{S}_2$ = Suffixe mit Index $i \equiv 2(3)$

1. Schritt: Erstelle Suffix-Array A_{12}

Eingabe: Text $T := t_0 t_1 \ldots t_{n-1}$

	0	1	2	3	4	5	6	7	8	9	10
T=	m	i	s	s	i	s	s	i	p	p	i

Fasse Tripel $[t_i t_{i+1} t_{i+2}]$ mit $i \not\equiv 0(3)$ als eigenes Zeichen auf. Fülle gegebenfalls mit \$ auf.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Konstruiere für $k \in \{1, 2\}$ den String

$$R_k = [t_k t_{k+1} t_{k+2}][t_{k+3} t_{k+4} t_{k+5}][t_{k+6} t_{k+7} t_{k+8}] \ldots$$

R_1=[iss][iss][ipp][i\$\$]

R_2=[ssi][ssi][ppi]

Konkatenation von R_1 und R_2.

$$R = R_1 \cdot R_2$$

R=[iss][iss][ipp][i\$\$][ssi][ssi][ppi]

$\mathcal{S}_0$ = Suffixe mit Index $i \equiv 0(3)$

$\mathcal{S}_1$ = Suffixe mit Index $i \equiv 1(3)$

$\mathcal{S}_2$ = Suffixe mit Index $i \equiv 2(3)$

1. Schritt: Erstelle Suffix-Array A_{12}

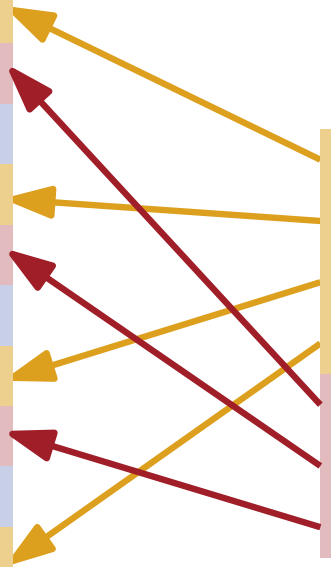
Beobachtung Jedes Suffix von R hat eindeutige Entsprechung in $\mathcal{S}_1 \cup \mathcal{S}_2$.

S_i entspricht Suffix $S'_j = [t_i\, t_{i+1}\, t_{i+2}][t_{i+3}\, t_{i+4}\, t_{i+5} \ldots]$ von R'

Sortierung von Suffixen $\mathcal{S}(R)$ induziert gesuchte Sortierung von $S_1 \cup S_2$.

Wie $\mathcal{S}(R)$ effizient sortieren?

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i



R=[iss][iss][ipp][i$$][ssi][ssi][ppi]

$\mathcal{S}(R)$=

$S_1(R)$	[iss][iss][ipp][i$$][ssi][ssi][ppi]
$S_2(R)$	[iss][ipp][i$$][ssi][ssi][ppi]
$S_3(R)$	[ipp][i$$][ssi][ssi][ppi]
$S_4(R)$	[i$$][ssi][ssi][ppi]
$S_5(R)$	[ssi][ssi][ppi]
$S_6(R)$	[ssi][ppi]
$S_7(R)$	[ppi]

Erinnerung: t_i erstes Zeichen von S_i.

1. Schritt: Erstelle Suffix-Array A_{12}

Wie $\mathcal{S}(R)$ effizient sortieren?

R=[iss][iss][ipp][i$$][ssi][ssi][ppi]

$\mathcal{S}(R)$=

$S_1(R)$	[iss][iss][ipp][i$$][ssi][ssi][ppi]
$S_2(R)$	[iss][ipp][i$$][ssi][ssi][ppi]
$S_3(R)$	[ipp][i$$][ssi][ssi][ppi]
$S_4(R)$	[i$$][ssi][ssi][ppi]
$S_5(R)$	[ssi][ssi][ppi]
$S_6(R)$	[ssi][ppi]
$S_7(R)$	[ppi]

1. Schritt: Erstelle Suffix-Array A_{12}

Wie $\mathcal{S}(R)$ effizient sortieren?

R=[iss][iss][ipp][i$$][ssi][ssi][ppi]

Wende Radixsort auf Zeichen von R an.

$\mathcal{S}(R)$=

$S_1(R)$	[iss][iss][ipp][i$$][ssi][ssi][ppi]
$S_2(R)$	[iss][ipp][i$$][ssi][ssi][ppi]
$S_3(R)$	[ipp][i$$][ssi][ssi][ppi]
$S_4(R)$	[i$$][ssi][ssi][ppi]
$S_5(R)$	[ssi][ssi][ppi]
$S_6(R)$	[ssi][ppi]
$S_7(R)$	[ppi]

Rang	Zeichen
1	[i$$]
2	[ipp]
3	[iss]
4	[ppi]
5	[ssi]

1. Schritt: Erstelle Suffix-Array A_{12}

Wie $\mathcal{S}(R)$ effizient sortieren?

R=[iss][iss][ipp][i$$][ssi][ssi][ppi]

Wende Radixsort auf Zeichen von R an.

Transformation: Ersetze Zeichen in R durch ihren Rang $\rightarrow R'$

R' = 3 3 2 1 5 5 4

Rang	Zeichen
1	[i$$]
2	[ipp]
3	[iss]
4	[ppi]
5	[ssi]

$\mathcal{S}(R)$=

$S_1(R)$	[iss][iss][ipp][i$$][ssi][ssi][ppi]
$S_2(R)$	[iss][ipp][i$$][ssi][ssi][ppi]
$S_3(R)$	[ipp][i$$][ssi][ssi][ppi]
$S_4(R)$	[i$$][ssi][ssi][ppi]
$S_5(R)$	[ssi][ssi][ppi]
$S_6(R)$	[ssi][ppi]
$S_7(R)$	[ppi]

1. Schritt: Erstelle Suffix-Array A_{12}

Wie $\mathcal{S}(R)$ effizient sortieren?

R=[iss][iss][ipp][i$$][ssi][ssi][ppi]

Wende Radixsort auf Zeichen von R an.

$\mathcal{S}(R)$=

$S_1(R)$	[iss][iss][ipp][i$$][ssi][ssi][ppi]
$S_2(R)$	[iss][ipp][i$$][ssi][ssi][ppi]
$S_3(R)$	[ipp][i$$][ssi][ssi][ppi]
$S_4(R)$	[i$$][ssi][ssi][ppi]
$S_5(R)$	[ssi][ssi][ppi]
$S_6(R)$	[ssi][ppi]
$S_7(R)$	[ppi]

Transformation: Ersetze Zeichen in R durch ihren Rang $\rightarrow R'$

R' = 3 3 2 1 5 5 4

Rang	Zeichen
1	[i$$]
2	[ipp]
3	[iss]
4	[ppi]
5	[ssi]

Suffix $S_i(R)$ entspricht $S_i(R')$

→ Sortierung von $\mathcal{S}(R')$ liefert gewünschte Sortierung von $\mathcal{S}(R)$.

Suffixe von R'

$S_1(R')$	3 3 2 1 5 5 4
$S_2(R')$	3 2 1 5 5 4
$S_3(R')$	2 1 5 5 4
$S_4(R')$	1 5 5 4
$S_5(R')$	5 5 4
$S_6(R')$	5 4
$S_7(R')$	4

1. Schritt: Erstelle Suffix-Array A_{12}

Wie $\mathcal{S}(R)$ effizient sortieren?

R=[iss][iss][ipp][i$$][ssi][ssi][ppi]

Wende Radixsort auf Zeichen von R an.

$\mathcal{S}(R)$=

$S_1(R)$	[iss][iss][ipp][i$$][ssi][ssi][ppi]
$S_2(R)$	[iss][ipp][i$$][ssi][ssi][ppi]
$S_3(R)$	[ipp][i$$][ssi][ssi][ppi]
$S_4(R)$	[i$$][ssi][ssi][ppi]
$S_5(R)$	[ssi][ssi][ppi]
$S_6(R)$	[ssi][ppi]
$S_7(R)$	[ppi]

Transformation: Ersetze Zeichen in R durch ihren Rang $\rightarrow R'$

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

Suffix $S_i(R)$ entspricht $S_i(R')$

$\rightarrow$ Sortierung von $\mathcal{S}(R')$ liefert gewünschte Sortierung von $\mathcal{S}(R)$.

Rang	Zeichen
1	[i$$]
2	[ipp]
3	[iss]
4	[ppi]
5	[ssi]

Suffixe von R'

$S_1(R')$	3 3 2 1 5 5 4
$S_2(R')$	3 2 1 5 5 4
$S_3(R')$	2 1 5 5 4
$S_4(R')$	1 5 5 4
$S_5(R')$	5 5 4
$S_6(R')$	5 4
$S_7(R')$	4

Erstelle Suffix-Array A' von R':

Rekursiver Aufruf $A' \leftarrow$ SUFFIXARRAY(R')

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

1. Schritt: Erstelle Suffix-Array A_{12}

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

$\mathcal{S}(R)=$

$S_1(R)$	[iss][iss][ipp][i$$][ssi][ssi][ppi]
$S_2(R)$	[iss][ipp][i$$][ssi][ssi][ppi]
$S_3(R)$	[ipp][i$$][ssi][ssi][ppi]
$S_4(R)$	[i$$][ssi][ssi][ppi]
$S_5(R)$	[ssi][ssi][ppi]
$S_6(R)$	[ssi][ppi]
$S_7(R)$	[ppi]

Suffixe von R'

$S_1(R')$	3 3 2 1 5 5 4
$S_2(R')$	3 2 1 5 5 4
$S_3(R')$	2 1 5 5 4
$S_4(R')$	1 5 5 4
$S_5(R')$	5 5 4
$S_6(R')$	5 4
$S_7(R')$	4

Sortierung von $\mathcal{S}(R')$ → liefert → Sortierung von $\mathcal{S}(R)$ → liefert → Sortierung von $\mathcal{S}_1 \cup \mathcal{S}_2$ → Suffix-Array A_{12}

$A_{12} =$

1	S_{10}	i	$S_4(R')$
2	S_7	i p p i	$S_3(R')$
3	S_4	i s s i p p i	$S_2(R')$
4	S_1	i s s i s s i p p i	$S_1(R')$
5	S_8	p p i	$S_7(R')$
6	S_5	s s i p p i	$S_6(R')$
7	S_2	s s i s s i p p i	$S_5(R')$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

1. Schritt: Erstelle Suffix-Array A_{12}

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

$\mathcal{S}(R)=$

$S_1(R)$	[iss][iss][ipp][i$$][ssi][ssi][ppi]
$S_2(R)$	[iss][ipp][i$$][ssi][ssi][ppi]
$S_3(R)$	[ipp][i$$][ssi][ssi][ppi]
$S_4(R)$	[i$$][ssi][ssi][ppi]
$S_5(R)$	[ssi][ssi][ppi]
$S_6(R)$	[ssi][ppi]
$S_7(R)$	[ppi]

Suffixe von R'

$S_1(R')$	3 3 2 1 5 5 4
$S_2(R')$	3 2 1 5 5 4
$S_3(R')$	2 1 5 5 4
$S_4(R')$	1 5 5 4
$S_5(R')$	5 5 4
$S_6(R')$	5 4
$S_7(R')$	4

Sortierung von $\mathcal{S}(R')$ → liefert → Sortierung von $\mathcal{S}(R)$ → liefert → Sortierung von $\mathcal{S}_1 \cup \mathcal{S}_2$ → Suffix-Array A_{12}

A_{12} =

1	S_{10}	i	$S_4(R')$
2	S_7	i p p i	$S_3(R')$
3	S_4	i s s i p p i	$S_2(R')$
4	S_1	i s s i s s i p p i	$S_1(R')$
5	S_8	p p i	$S_7(R')$
6	S_5	s s i p p i	$S_6(R')$
7	S_2	s s i s s i p p i	$S_5(R')$

Weshalb wird R' aus R erstellt und nicht direkt die Suffixe von R sortiert?

Konstruktion von Suffix-Arrays

Eingabe: Text $T := t_0 t_1 \ldots t_{n-1}$

	0	1	2	3	4	5	6	7	8	9	10
T=	m	i	s	s	i	s	s	i	p	p	i

Gesucht: Suffix-Array A von T, d.h. lexikographische Sortierung von Suffixen von T.

Kürze $x \mod y = z$ mit $x \equiv z(y)$ ab.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

$\mathcal{S}$ = Suffixe von T

$\mathcal{S}_0$ = Suffixe mit Index $i \equiv 0(3)$

$\mathcal{S}_1$ = Suffixe mit Index $i \equiv 1(3)$

$\mathcal{S}_2$ = Suffixe mit Index $i \equiv 2(3)$

SUFFIXARRAY(Text $T = t_0 t_1 \ldots t_{n-1}$)
wenn $n = O(1)$ **dann**
  Konstruiere A in $O(1)$ Zeit.
sonst
  Berechne Suffix-Array A_{12} für $\mathcal{S}_1 \cup \mathcal{S}_2$.
  Berechne Suffix-Array A_0 für $\mathcal{S}_0$ basierend auf A_{12}.
  Vermenge A_{12} mit A_0.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

2. Schritt: Konstruktion von A_0

Beobachtung: Für zwei $S_i, S_j \in \mathcal{S}_0$ gilt

$S_i < S_j$ genau dann wenn $t_i < t_j$, oder $t_i = t_j$ und $S_{i+1} < S_{j+1}$.

Beobachtung gibt Definition der lexikographischen Ordnung wider.

↳ Beschreibt wie Suffixe $\mathcal{S}_0$ sortiert werden müssen. Verwende Radixsort.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Erinnerung:

t_i = erstes Zeichen von S_i

2. Schritt: Konstruktion von A_0

Beobachtung: Für zwei $S_i, S_j \in \mathcal{S}_0$ gilt

$S_i < S_j$ genau dann wenn $t_i < t_j$, oder $t_i = t_j$ und $S_{i+1} < S_{j+1}$.

Beobachtung gibt Definition der lexikographischen Ordnung wider.

↳ Beschreibt wie Suffixe $\mathcal{S}_0$ sortiert werden müssen. Verwende Radixsort.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Aus Schritt 1 folgt A_{12}:

1	S_{10}	i
2	S_7	i p p i
3	S_4	i s s i p p i
4	S_1	i s s i s s i p p i
5	S_8	p p i
6	S_5	s s i p p i
7	S_2	s s i s s i p p

Erinnerung:

t_i = erstes Zeichen von S_i

2. Schritt: Konstruktion von A_0

Beobachtung: Für zwei $S_i, S_j \in \mathcal{S}_0$ gilt

$S_i < S_j$ genau dann wenn $t_i < t_j$, oder $t_i = t_j$ und $S_{i+1} < S_{j+1}$.

Beobachtung gibt Definition der lexikographischen Ordnung wider.

↳ Beschreibt wie Suffixe $\mathcal{S}_0$ sortiert werden müssen. Verwende Radixsort.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Aus Schritt 1 folgt A_{12}:

1	S_{10}	i
2	S_7	i p p i
3	S_4	i s s i p p i
4	S_1	i s s i s s i p p i
5	S_8	p p i
6	S_5	s s i p p i
7	S_2	s s i s s i p p

Erinnerung:

t_i = erstes Zeichen von S_i

$S_0 < S_9$ weil $m < p$
$S_9 < S_6$ weil $p < s$
$S_6 < S_3$ weil $S_7 < S_4$

$\Rightarrow \underbrace{S_0 < S_9 < S_6 < S_3}_{\text{beschreibt } A_0}$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Konstruktion von Suffix-Arrays

Eingabe: Text $T := t_0 t_1 \ldots t_{n-1}$

	0	1	2	3	4	5	6	7	8	9	10
T=	m	i	s	s	i	s	s	i	p	p	i

Gesucht: Suffix-Array A von T, d.h. lexikographische Sortierung von Suffixen von T.

Kürze $x \mod y = z$ mit $x \equiv z(y)$ ab.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

$\mathcal{S}$ = Suffixe von T

$\mathcal{S}_0$ = Suffixe mit Index $i \equiv 0(3)$

$\mathcal{S}_1$ = Suffixe mit Index $i \equiv 1(3)$

$\mathcal{S}_2$ = Suffixe mit Index $i \equiv 2(3)$

SUFFIXARRAY(Text $T = t_0 t_1 \ldots t_{n-1}$)
wenn $n = O(1)$ **dann**
  Konstruiere A in $O(1)$ Zeit.
sonst
  Berechne Suffix-Array A_{12} für $\mathcal{S}_1 \cup \mathcal{S}_2$.
  Berechne Suffix-Array A_0 für $\mathcal{S}_0$ basierend auf A_{12}.
  Vermenge A_{12} mit A_0.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

3. Schritt: Vermengen von A_0 und A_{12}

Beobachtung: Sei $S_i \in \mathcal{S}_0$

1. Für $S_j \in \mathcal{S}_1$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und $S_{i+1} < S_{j+1}$.
2. Für $S_j \in S_2$ gilt: $S_i < S_j$ genau dann, wenn
   $t_i < t_j$, oder $t_i = t_j$ und $t_{i+1} < t_{j+1}$, oder $t_i t_{i+1} = t_i t_{j+1}$ und $S_{i+2} < S_{j+2}$.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Erinnerung:

t_i = erstes Zeichen von S_i

3. Schritt: Vermengen von A_0 und A_{12}

Beobachtung: Sei $S_i \in \mathcal{S}_0$

1. Für $S_j \in \mathcal{S}_1$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und $S_{i+1} < S_{j+1}$.
2. Für $S_j \in S_2$ gilt: $S_i < S_j$ genau dann, wenn
   $t_i < t_j$, oder $t_i = t_j$ und $t_{i+1} < t_{j+1}$, oder $t_i t_{i+1} = t_i t_{j+1}$ und $S_{i+2} < S_{j+2}$.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Aus Schritt 2 folgt:

$S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt:

$S_0 < S_9 < S_6 < S_3$

Vermengen wie bei Merge-Sort:

Erinnerung:

t_i = erstes Zeichen von S_i

3. Schritt: Vermengen von A_0 und A_{12}

Beobachtung: Sei $S_i \in \mathcal{S}_0$

1. Für $S_j \in \mathcal{S}_1$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und $S_{i+1} < S_{j+1}$.
2. Für $S_j \in S_2$ gilt: $S_i < S_j$ genau dann, wenn
   $t_i < t_j$, oder $t_i = t_j$ und $t_{i+1} < t_{j+1}$, oder $t_i t_{i+1} = t_i t_{j+1}$ und $S_{i+2} < S_{j+2}$.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Aus Schritt 2 folgt:

$S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt:

$S_0 < S_9 < S_6 < S_3$

Vermengen wie bei Merge-Sort:

S_{10}

Erinnerung:

t_i = erstes Zeichen von S_i

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

3. Schritt: Vermengen von A_0 und A_{12}

Beobachtung: Sei $S_i \in \mathcal{S}_0$

1. Für $S_j \in \mathcal{S}_1$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und $S_{i+1} < S_{j+1}$.
2. Für $S_j \in S_2$ gilt: $S_i < S_j$ genau dann, wenn
   $t_i < t_j$, oder $t_i = t_j$ und $t_{i+1} < t_{j+1}$, oder $t_i t_{i+1} = t_i t_{j+1}$ und $S_{i+2} < S_{j+2}$.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Aus Schritt 2 folgt:

$S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt:

$S_0 < S_9 < S_6 < S_3$

Vermengen wie bei Merge-Sort:

$S_{10} < S_7$

Erinnerung:

t_i = erstes Zeichen von S_i

3. Schritt: Vermengen von A_0 und A_{12}

Beobachtung: Sei $S_i \in \mathcal{S}_0$

1. Für $S_j \in \mathcal{S}_1$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und $S_{i+1} < S_{j+1}$.
2. Für $S_j \in S_2$ gilt: $S_i < S_j$ genau dann, wenn
   $t_i < t_j$, oder $t_i = t_j$ und $t_{i+1} < t_{j+1}$, oder $t_i t_{i+1} = t_i t_{j+1}$ und $S_{i+2} < S_{j+2}$.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Aus Schritt 2 folgt:

$S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt:

$S_0 < S_9 < S_6 < S_3$

Vermengen wie bei Merge-Sort:

$S_{10} < S_7 < S_4$

Erinnerung:

t_i = erstes Zeichen von S_i

3. Schritt: Vermengen von A_0 und A_{12}

Beobachtung: Sei $S_i \in \mathcal{S}_0$

1. Für $S_j \in \mathcal{S}_1$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und $S_{i+1} < S_{j+1}$.
2. Für $S_j \in S_2$ gilt: $S_i < S_j$ genau dann, wenn
   $t_i < t_j$, oder $t_i = t_j$ und $t_{i+1} < t_{j+1}$, oder $t_i t_{i+1} = t_i t_{j+1}$ und $S_{i+2} < S_{j+2}$.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Aus Schritt 2 folgt:

$S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt:

$S_0 < S_9 < S_6 < S_3$

Vermengen wie bei Merge-Sort:

$S_{10} < S_7 < S_4 < S_1$

Erinnerung:

t_i = erstes Zeichen von S_i

3. Schritt: Vermengen von A_0 und A_{12}

Beobachtung: Sei $S_i \in \mathcal{S}_0$

1. Für $S_j \in \mathcal{S}_1$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und $S_{i+1} < S_{j+1}$.
2. Für $S_j \in S_2$ gilt: $S_i < S_j$ genau dann, wenn
   $t_i < t_j$, oder $t_i = t_j$ und $t_{i+1} < t_{j+1}$, oder $t_i t_{i+1} = t_i t_{j+1}$ und $S_{i+2} < S_{j+2}$.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Aus Schritt 2 folgt:

$S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt:

$S_0 < S_9 < S_6 < S_3$

Vermengen wie bei Merge-Sort:

$S_{10} < S_7 < S_4 < S_1 < S_0$

Erinnerung:

t_i = erstes Zeichen von S_i

3. Schritt: Vermengen von A_0 und A_{12}

Beobachtung: Sei $S_i \in \mathcal{S}_0$

1. Für $S_j \in \mathcal{S}_1$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und $S_{i+1} < S_{j+1}$.
2. Für $S_j \in S_2$ gilt: $S_i < S_j$ genau dann, wenn
   $t_i < t_j$, oder $t_i = t_j$ und $t_{i+1} < t_{j+1}$, oder $t_i t_{i+1} = t_i t_{j+1}$ und $S_{i+2} < S_{j+2}$.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Aus Schritt 2 folgt:

$S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt:

$S_0 < S_9 < S_6 < S_3$

Vermengen wie bei Merge-Sort:

$S_{10} < S_7 < S_4 < S_1 < S_0 < S_9$

Erinnerung:

t_i = erstes Zeichen von S_i

3. Schritt: Vermengen von A_0 und A_{12}

Beobachtung: Sei $S_i \in \mathcal{S}_0$

1. Für $S_j \in \mathcal{S}_1$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und $S_{i+1} < S_{j+1}$.
2. Für $S_j \in S_2$ gilt: $S_i < S_j$ genau dann, wenn
   $t_i < t_j$, oder $t_i = t_j$ und $t_{i+1} < t_{j+1}$, oder $t_i t_{i+1} = t_i t_{j+1}$ und $S_{i+2} < S_{j+2}$.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Aus Schritt 2 folgt:

$S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt:

$S_0 < S_9 < S_6 < S_3$

Vermengen wie bei Merge-Sort:

$S_{10} < S_7 < S_4 < S_1 < S_0 < S_9 < S_6$

Erinnerung:

t_i = erstes Zeichen von S_i

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

3. Schritt: Vermengen von A_0 und A_{12}

Beobachtung: Sei $S_i \in \mathcal{S}_0$

1. Für $S_j \in \mathcal{S}_1$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und $S_{i+1} < S_{j+1}$.
2. Für $S_j \in S_2$ gilt: $S_i < S_j$ genau dann, wenn
   $t_i < t_j$, oder $t_i = t_j$ und $t_{i+1} < t_{j+1}$, oder $t_i t_{i+1} = t_i t_{j+1}$ und $S_{i+2} < S_{j+2}$.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Aus Schritt 2 folgt:

$S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt:

$S_0 < S_9 < S_6 < S_3$

Vermengen wie bei Merge-Sort:

$S_{10} < S_7 < S_4 < S_1 < S_0 < S_9 < S_6 < S_8$

Erinnerung:

t_i = erstes Zeichen von S_i

3. Schritt: Vermengen von A_0 und A_{12}

Beobachtung: Sei $S_i \in \mathcal{S}_0$

1. Für $S_j \in \mathcal{S}_1$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und $S_{i+1} < S_{j+1}$.
2. Für $S_j \in S_2$ gilt: $S_i < S_j$ genau dann, wenn
   $t_i < t_j$, oder $t_i = t_j$ und $t_{i+1} < t_{j+1}$, oder $t_i t_{i+1} = t_i t_{j+1}$ und $S_{i+2} < S_{j+2}$.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Aus Schritt 2 folgt:

$S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt:

$S_0 < S_9 < S_6 < S_3$

Vermengen wie bei Merge-Sort:

$$S_{10} < S_7 < S_4 < S_1 < S_0 < S_9 < S_6 < S_8 < S_3$$

Erinnerung:

t_i = erstes Zeichen von S_i

3. Schritt: Vermengen von A_0 und A_{12}

Beobachtung: Sei $S_i \in \mathcal{S}_0$

1. Für $S_j \in \mathcal{S}_1$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und $S_{i+1} < S_{j+1}$.
2. Für $S_j \in S_2$ gilt: $S_i < S_j$ genau dann, wenn
   $t_i < t_j$, oder $t_i = t_j$ und $t_{i+1} < t_{j+1}$, oder $t_i t_{i+1} = t_i t_{j+1}$ und $S_{i+2} < S_{j+2}$.

S_0	m i s s i s s i p p i
S_1	i s s i s s i p p i
S_2	s s i s s i p p i
S_3	s i s s i p p i
S_4	i s s i p p i
S_5	s s i p p i
S_6	s i p p i
S_7	i p p i
S_8	p p i
S_9	p i
S_{10}	i

Aus Schritt 2 folgt:

$S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt:

$S_0 < S_9 < S_6 < S_3$

Vermengen wie bei Merge-Sort:

$S_{10} < S_7 < S_4 < S_1 < S_0 < S_9 < S_6 < S_8 < S_3 < S_5 < S_2$

Erinnerung:

t_i = erstes Zeichen von S_i

String-Matching – Rabin & Karp (1981)

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Rabin & Karp – Idee

Annahme:
Für das Alphabet gilt $\Sigma = \{0, 1, \ldots, 9\}$.

Bemerkung: Das ist keine echte Einschränkung, da im allgemeinen Fall jeder String als Zahl in d-ärer Darstellung mit $d = |\Sigma|$ aufgefasst werden kann.

Interpretation als Zahl:

- Bezeichne den durch P repräsentierten Zahlenwert mit p.
- Sei t_s die durch den Teilstring $T[s]\,T[s+1]\ldots T[s+m]$ repräsentierte Zahl.

Es gilt: $p = t_s$ genau dann, wenn $P[j] = T[s+j]$ für alle $0 \leq j < m$.

Rabin & Karp – Idee

Annahme:
Für das Alphabet gilt $\Sigma = \{0, 1, \ldots, 9\}$.

Bemerkung: Das ist keine echte Einschränkung, da im allgemeinen Fall jeder String als Zahl in d-ärer Darstellung mit $d = |\Sigma|$ aufgefasst werden kann.

Interpretation als Zahl:

- Bezeichne den durch P repräsentierten Zahlenwert mit p.
- Sei t_s die durch den Teilstring $T[s]\,T[s+1]\ldots T[s+m]$ repräsentierte Zahl.

Es gilt: $p = t_s$ genau dann, wenn $P[j] = T[s+j]$ für alle $0 \leq j < m$.

Der Algorithmus von Rabin & Karp:

- **Idee:** Der Vergleich von zwei Zahlen ($p = t_s$) kann in konstanter Zeit ausgeführt werden (Vergleich zweier Integers), wenn die Zahlen nicht zu groß sind.
- **Problem:** p und t_s haben $O(m)$ Bits $\rightarrow$ Vergleich braucht $O(m)$ Zeit wie beim naiven Algorithmus.
- **Trick:** Berechne p und t_s modulo einer geeigneten Zahl q und vergleiche p und t_s nur dann, wenn ihrer Reste bezüglich q gleich sind.

Problem 1

Wenden Sie den Rabin-Karp-Algorithmus auf $T = 3141592653589793$ und $P = 26$ an. Nehmen Sie hierzu $q = 11$ an.

$p = 26 = 2 \cdot 11 + 4$

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3

Problem 1

Wenden Sie den Rabin-Karp-Algorithmus auf $T = 3141592653589793$ und $P = 26$ an. Nehmen Sie hierzu $q = 11$ an.

$p = 26 = 2 \cdot 11 + 4$

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
$31 = 2 \cdot 11 +$	9														

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

Wenden Sie den Rabin-Karp-Algorithmus auf $T = 3141592653589793$ und $P = 26$ an. Nehmen Sie hierzu $q = 11$ an.

$p = 26 = 2 \cdot 11 + 4$

3 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3

$14 = 1 \cdot 11 + 3$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

Wenden Sie den Rabin-Karp-Algorithmus auf $T = 3141592653589793$ und $P = 26$ an. Nehmen Sie hierzu $q = 11$ an.

$p = 26 = 2 \cdot 11 + 4$

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
		41	=	3	·	11	+	8							

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

Wenden Sie den Rabin-Karp-Algorithmus auf $T = 3141592653589793$ und $P = 26$ an. Nehmen Sie hierzu $q = 11$ an.

$p = 26 = 2 \cdot 11 + 4$

3 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3

$15 = 1 \cdot 11 + 4$

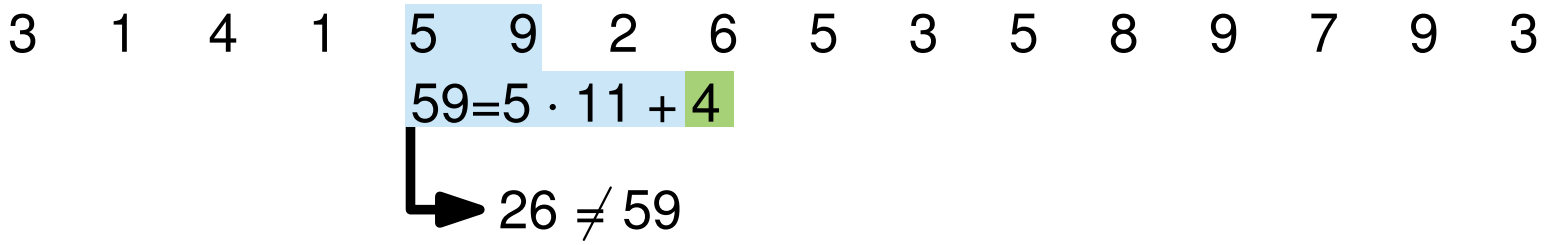
$\rightarrow 26 \neq 15$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

Wenden Sie den Rabin-Karp-Algorithmus auf $T = 3141592653589793$ und $P = 26$ an. Nehmen Sie hierzu $q = 11$ an.

$p = 26 = 2 \cdot 11 + 4$

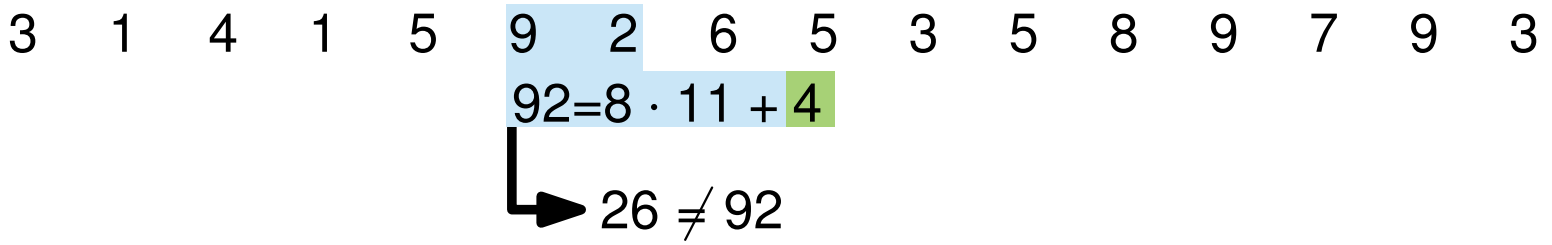


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

Wenden Sie den Rabin-Karp-Algorithmus auf $T = 3141592653589793$ und $P = 26$ an. Nehmen Sie hierzu $q = 11$ an.

$p = 26 = 2 \cdot 11 + 4$

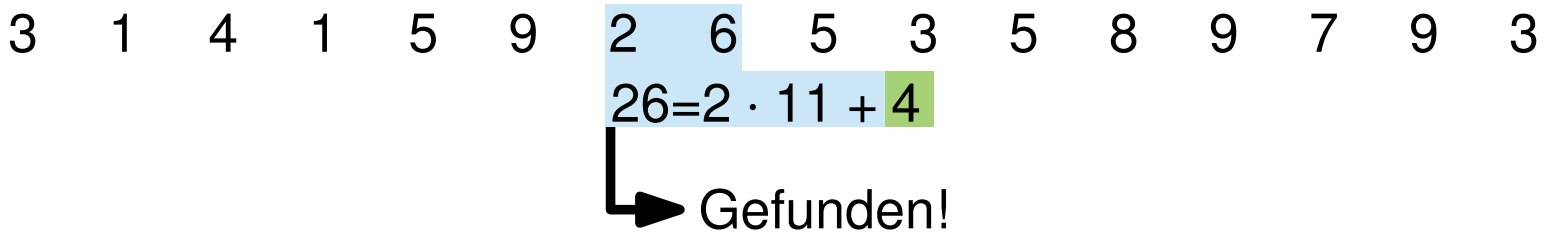


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

Wenden Sie den Rabin-Karp-Algorithmus auf $T = 3141592653589793$ und $P = 26$ an. Nehmen Sie hierzu $q = 11$ an.

$p = 26 = 2 \cdot 11 + 4$



Karlsruher Institut für Technologie

String-Matching mit endlichen Automaten

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Idee

1. Baue einen endlichen Automation $\mathcal{A}_P$ (bezüglich des Musters P), sodass:
   - $\mathcal{A}_P$ akzeptiert genau die Wörter, die P als Suffix haben (also auf P enden).
2. Führe $\mathcal{A}_P$ mit dem Text T als Eingabe aus.
   - Nach jedem Vorkommen von P in T ist $\mathcal{A}_P$ in einem akzeptierenden Zustand.

Definition: Endlicher Automat **(Definition 7.1)**

Ein *endlicher Automat* $\mathcal{A}$ ist ein Tupel $(Q, q_0, A, \Sigma, \delta)$, mit

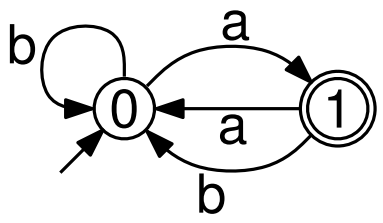
Q	–	endliche Menge von *Zuständen*
$q_0 \in Q$	–	*Startzustand*
$A \subseteq Q$	–	Menge von *akzeptierenden Zuständen*
Σ	–	endliches *Eingabealphabet*
$\delta : Q \times \Sigma \longrightarrow Q$	–	*Übertragungsfunktion*

Beispiel:

$Q = \{0, 1\}$, $q_0 = 0$, $A = \{1\}$, $\Sigma = \{a, b\}$ und

$\delta(0, a) = 1$, $\delta(0, b) = 0$,

$\delta(1, a) = 0$, $\delta(1, b) = 0$



Akzeptiert Wörter, die auf ungerade Anzahl a's enden.

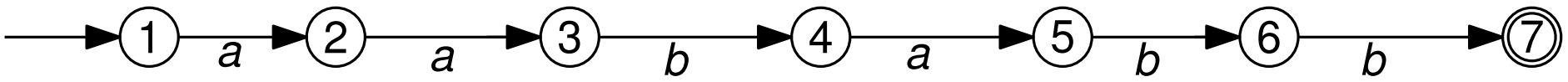
Problem 1

Konstruieren Sie einen String-Matching-Automat für $P = aababb$.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

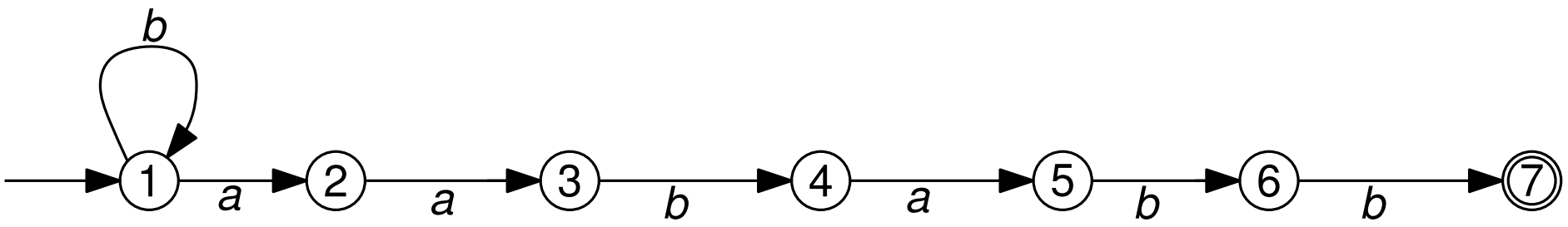
Problem 1

Konstruieren Sie einen String-Matching-Automat für $P = aababb$.



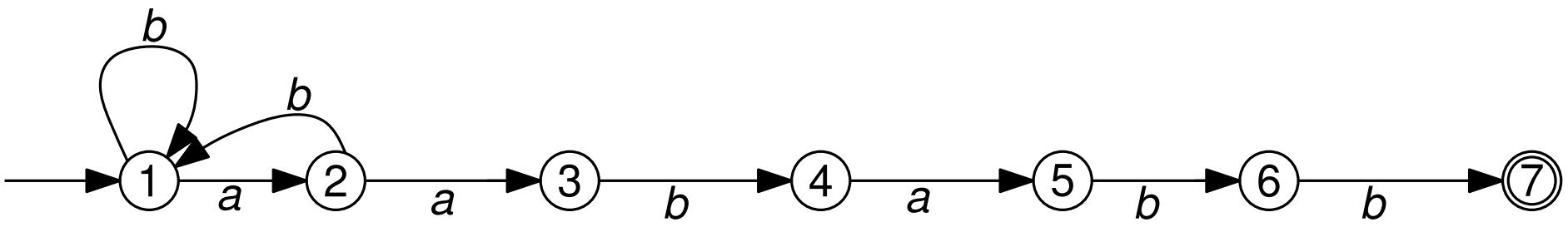
Problem 1

Konstruieren Sie einen String-Matching-Automat für $P = aababb$.



Problem 1

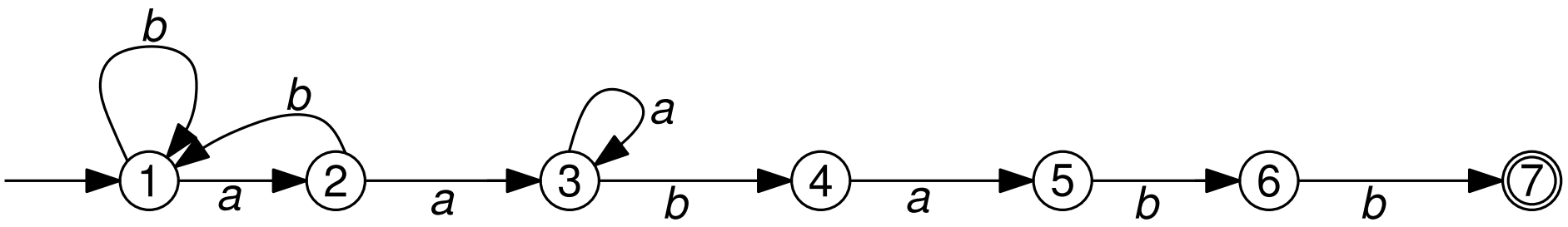
Konstruieren Sie einen String-Matching-Automat für $P = aababb$.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

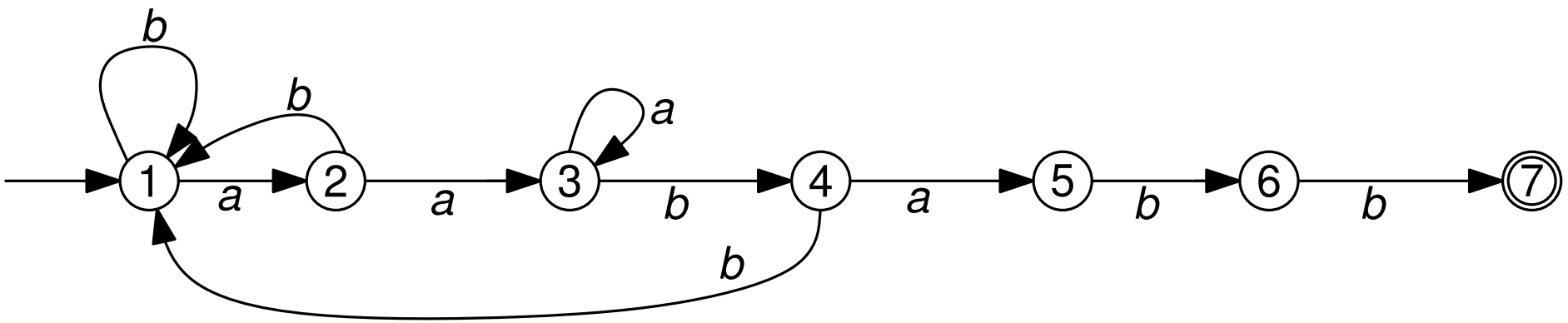
Konstruieren Sie einen String-Matching-Automat für $P = aababb$.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

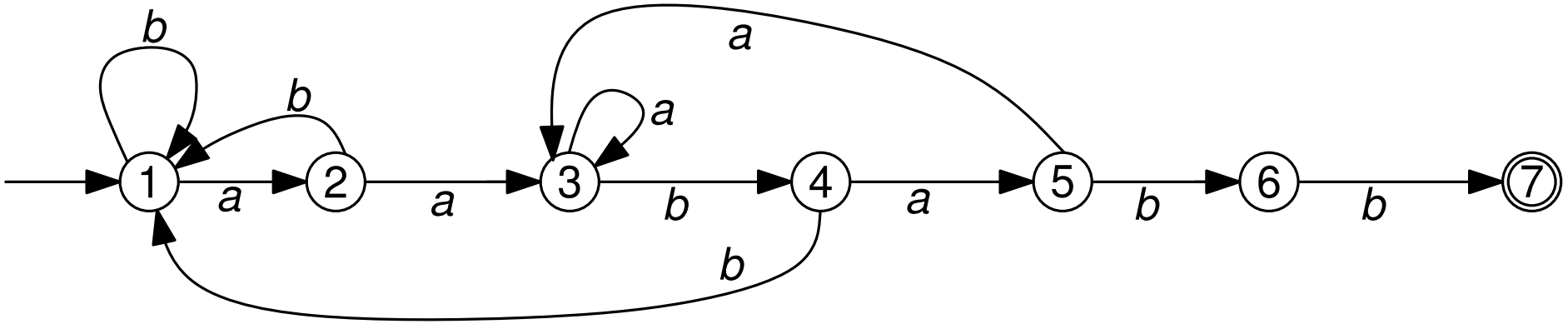
Konstruieren Sie einen String-Matching-Automat für $P = aababb$.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

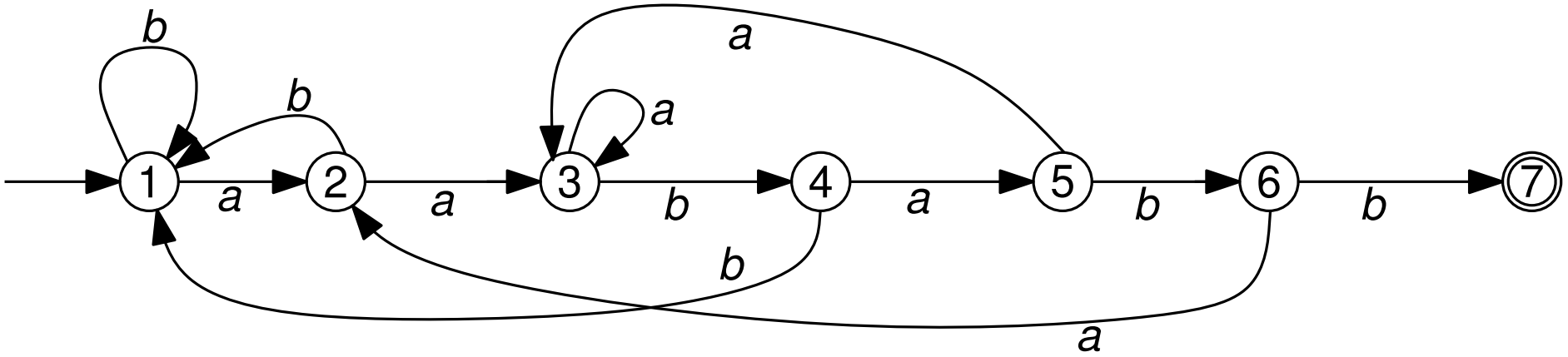
Konstruieren Sie einen String-Matching-Automat für $P = aababb$.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

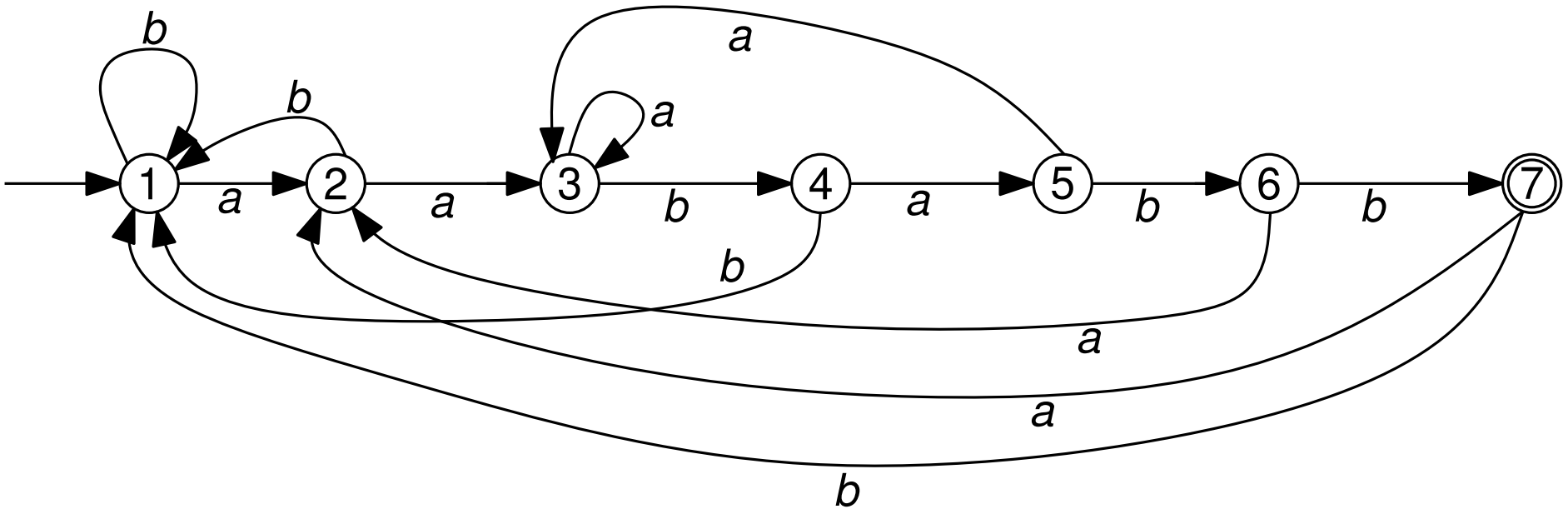
Konstruieren Sie einen String-Matching-Automat für $P = aababb$.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

Konstruieren Sie einen String-Matching-Automat für $P = aababb$.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner