Karlsruher Institut für Technologie

Algorithmen II
Übung am 20.12.2012

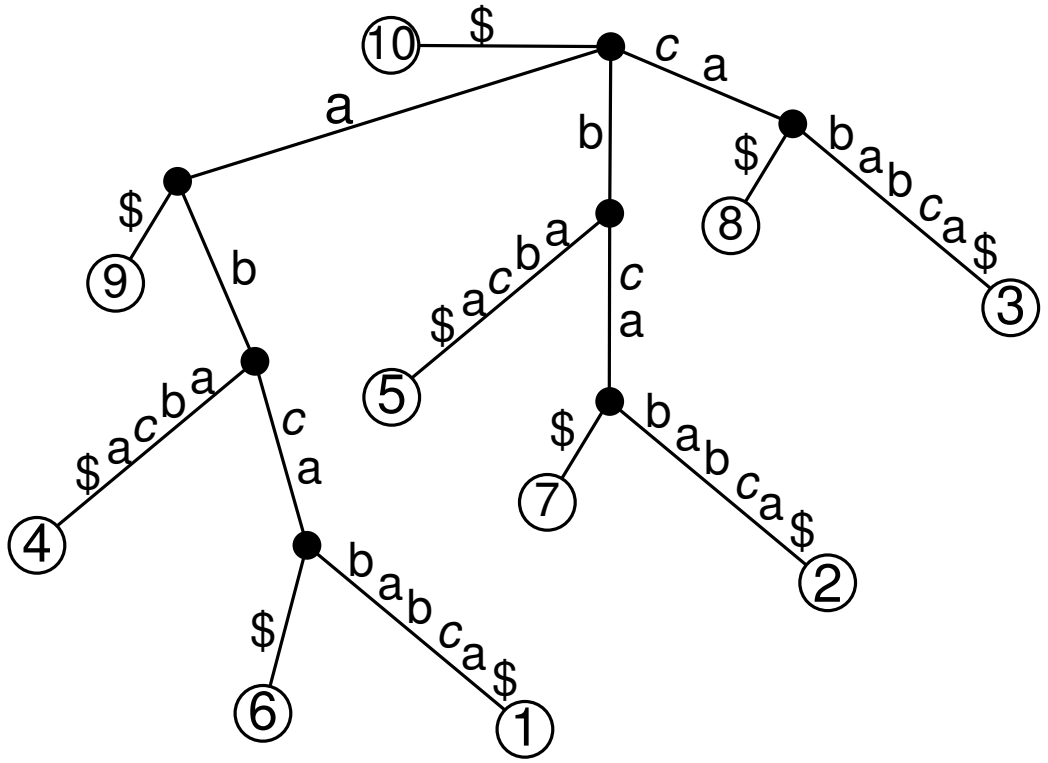
INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER

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.



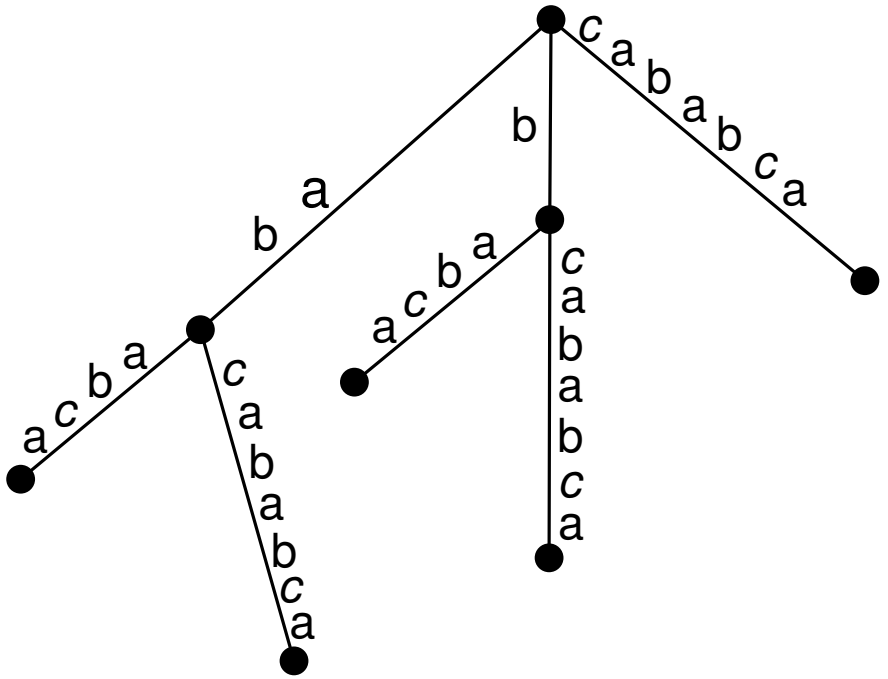
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

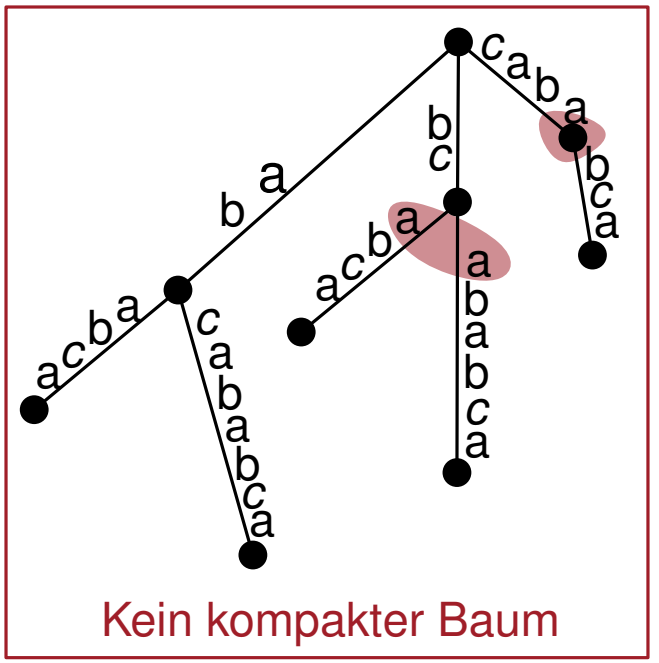
Kompakte Bäume

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 Knoten $v \in V$ gilt

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

T=a b c a b a b c a





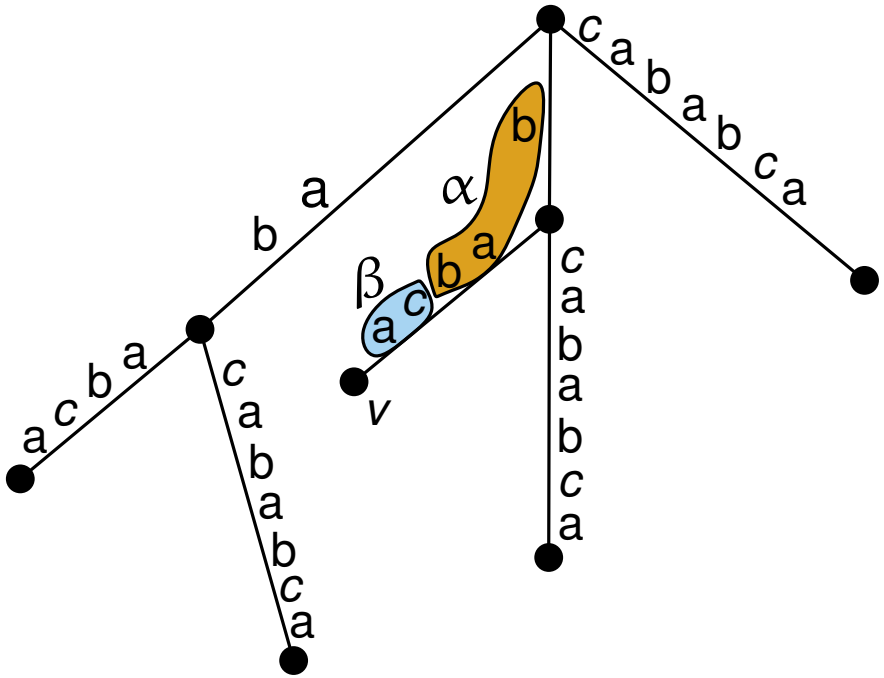
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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 $\alpha = bab$ und $\beta = ca$ es Knoten v gibt mit $\overline{v} = \alpha\beta$.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suffixbaum

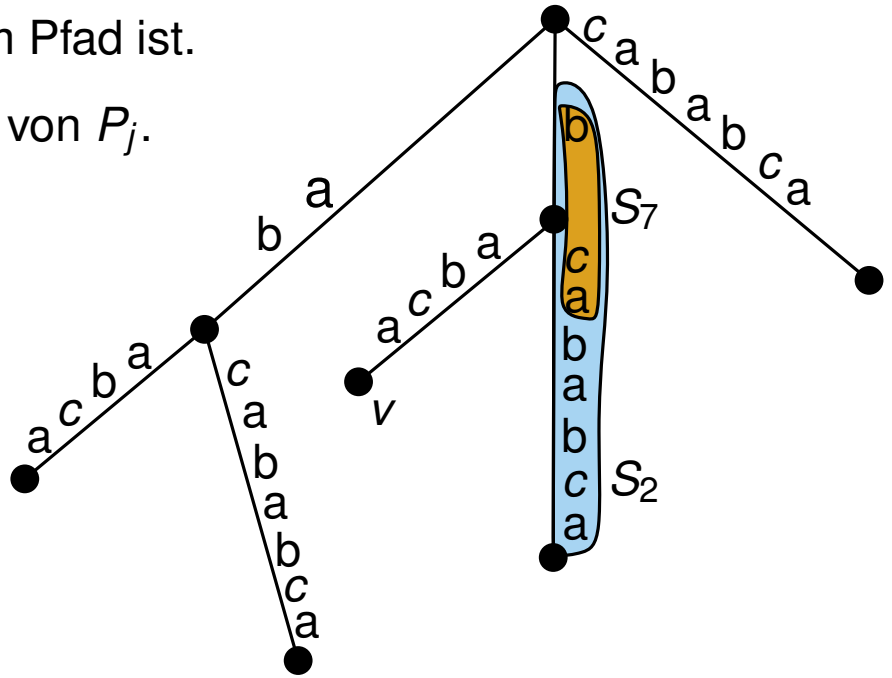
Definition: Ein *Suffixbaum* ist ein kompakter $\Sigma^\star$-Baum S, sodass S genau die Infixe von T enthält:

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

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:

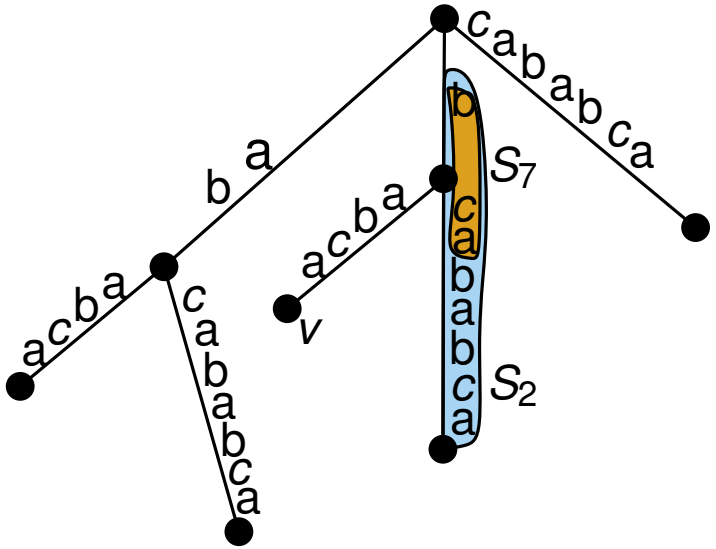
1. S_i bezeichnet das Suffix von T beginnend ab der i-ten Stelle.
2. $T[i,j]$ bezeichnet den Teilstring von T von der i-ten bis zur j-ten Stelle.

Suffixbaum

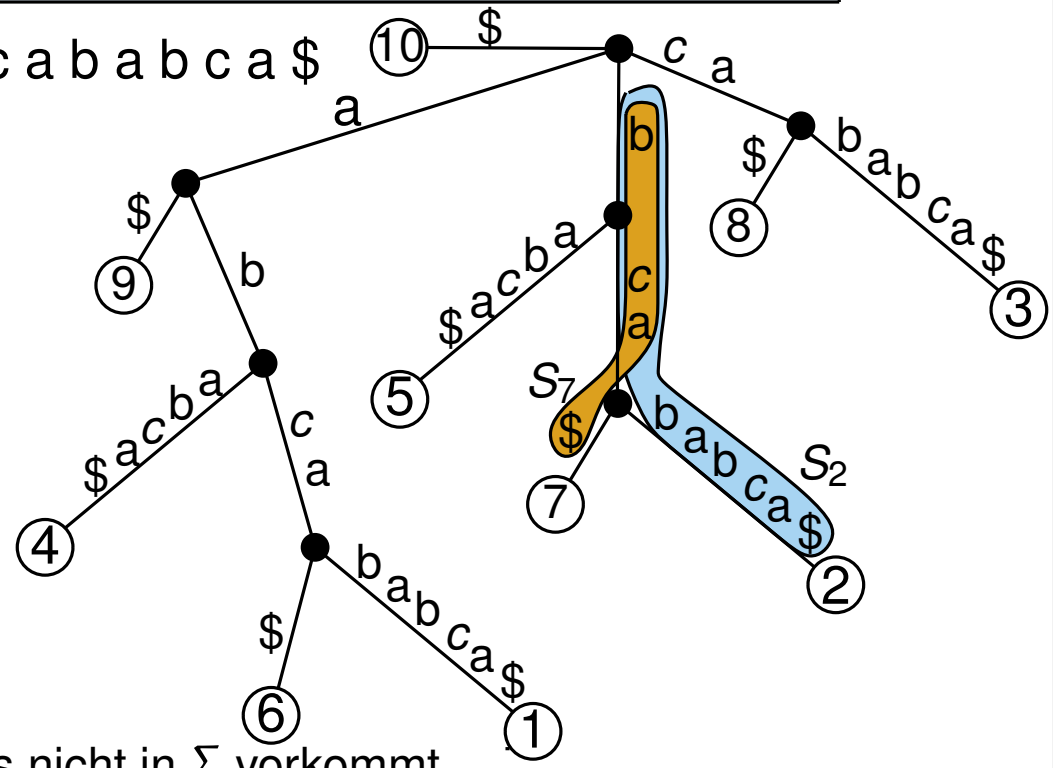
Definition: Ein *Suffixbaum* ist ein kompakter $\Sigma^\star$-Baum S, sodass S genau die Infixe von T enthält:

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

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.

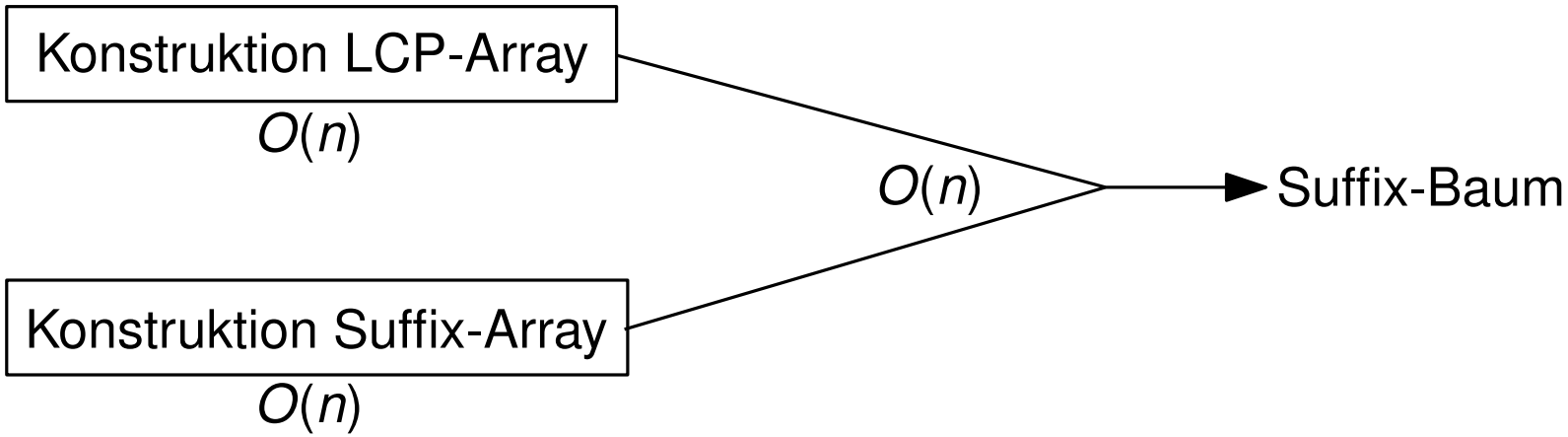
Ab jetzt:
S_i := i-te Suffix
:= Pfad von Wurzel zu Blatt i.

Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

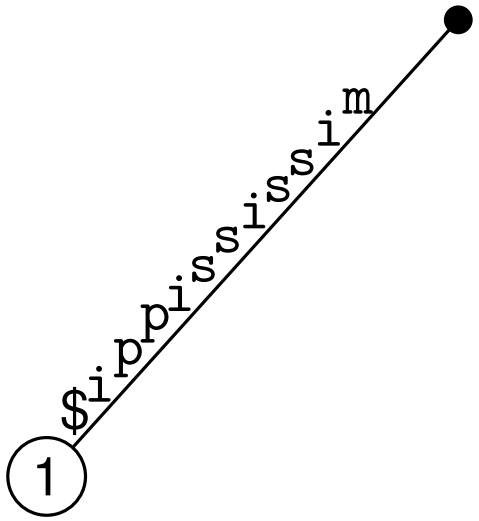
In der Vorlesung wurden zwei Möglichkeiten vorgestellt:

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



Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

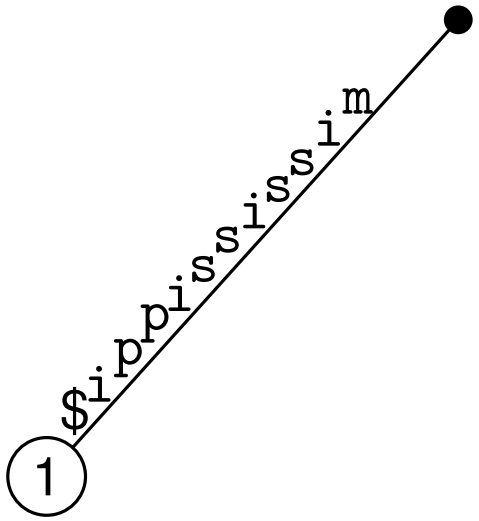


Beginne mit S_1=`mississippi$`.

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`:

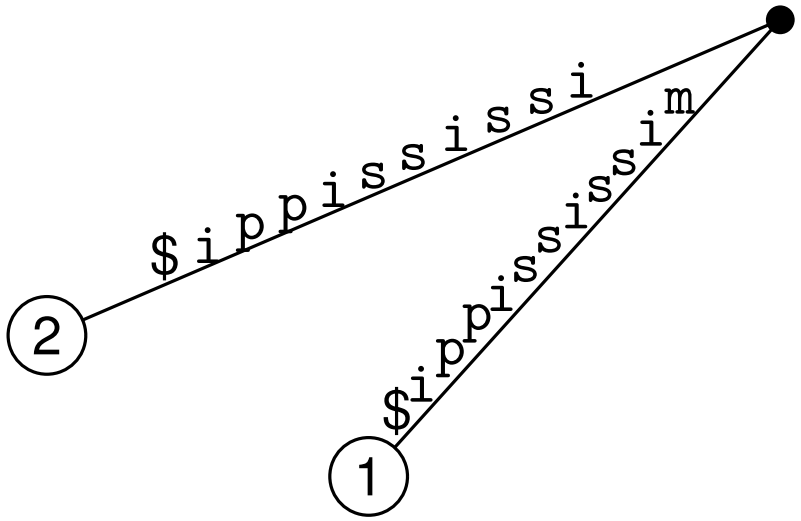


Füge S_2=`ississippi$` ein: Spalte an Wurzel neue Kante ab.

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`:

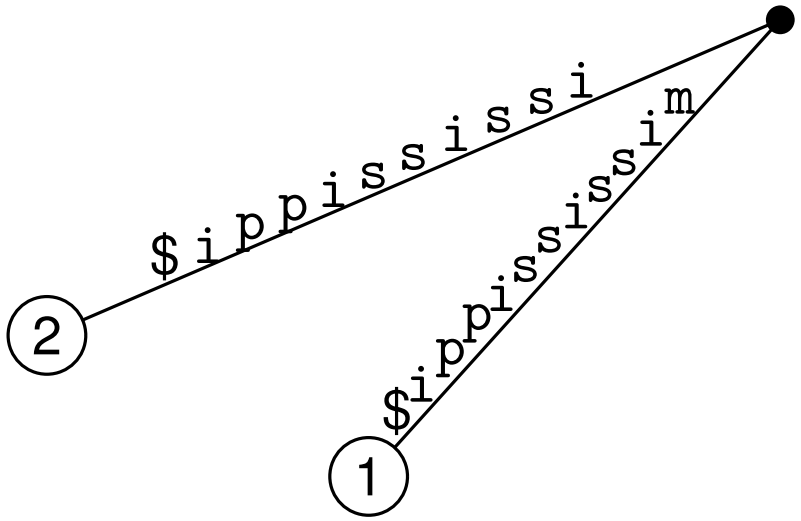


Füge S_2=`ississippi$` ein: Spalte an Wurzel neue Kante ab.

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`:

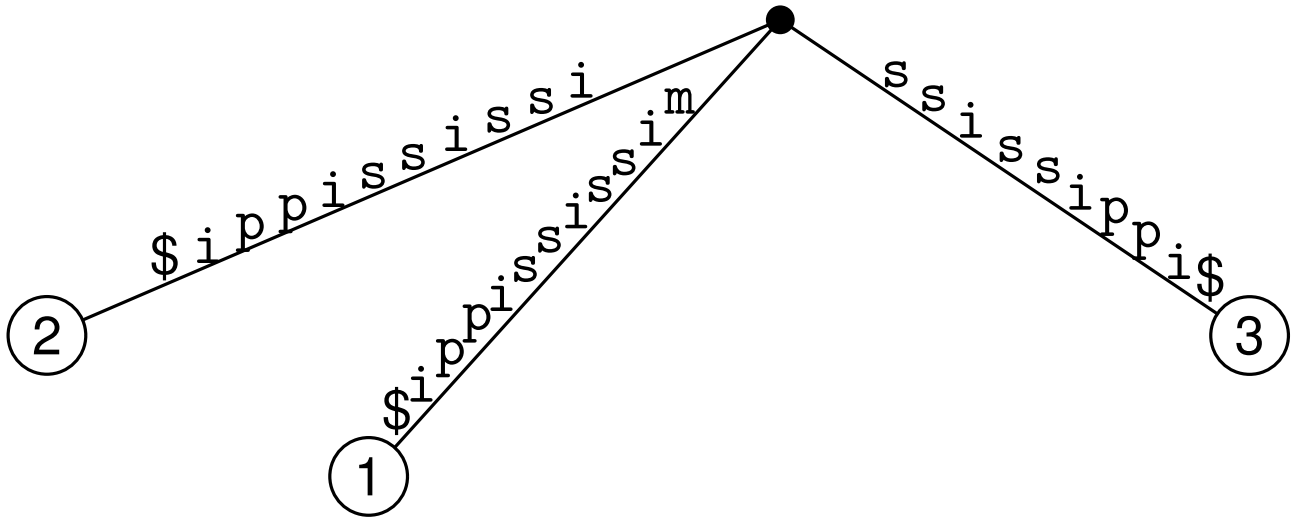


Füge S_3=`ssissippi$` ein: Spalte an Wurzel neue Kante ab.

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`:

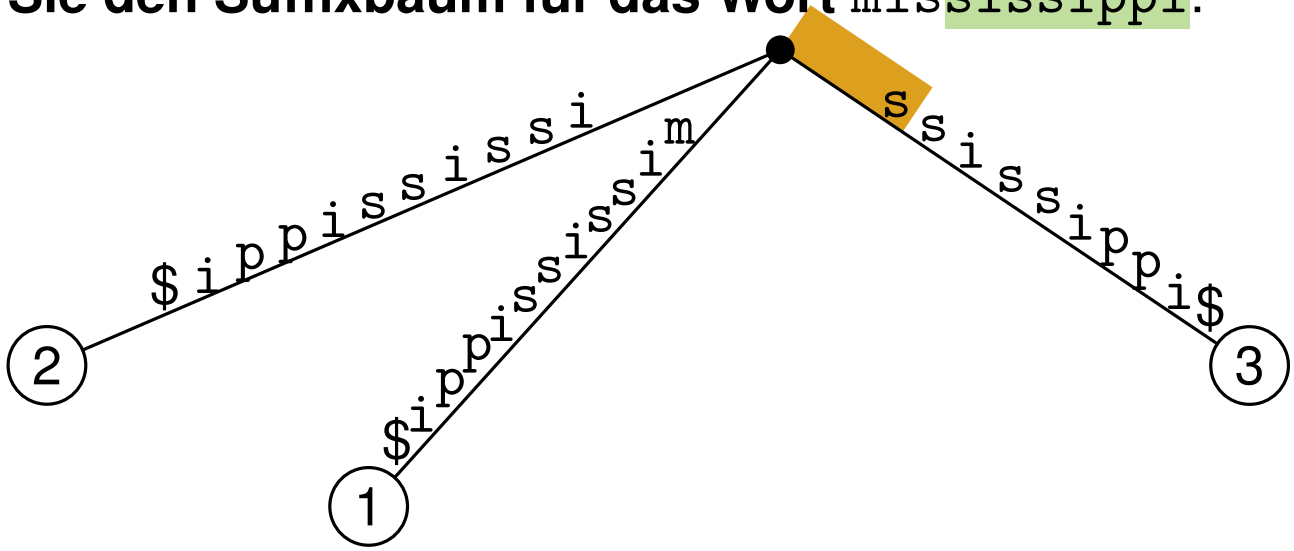


Füge S_3=`ssissippi$` ein: Spalte an Wurzel neue Kante ab.

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`:

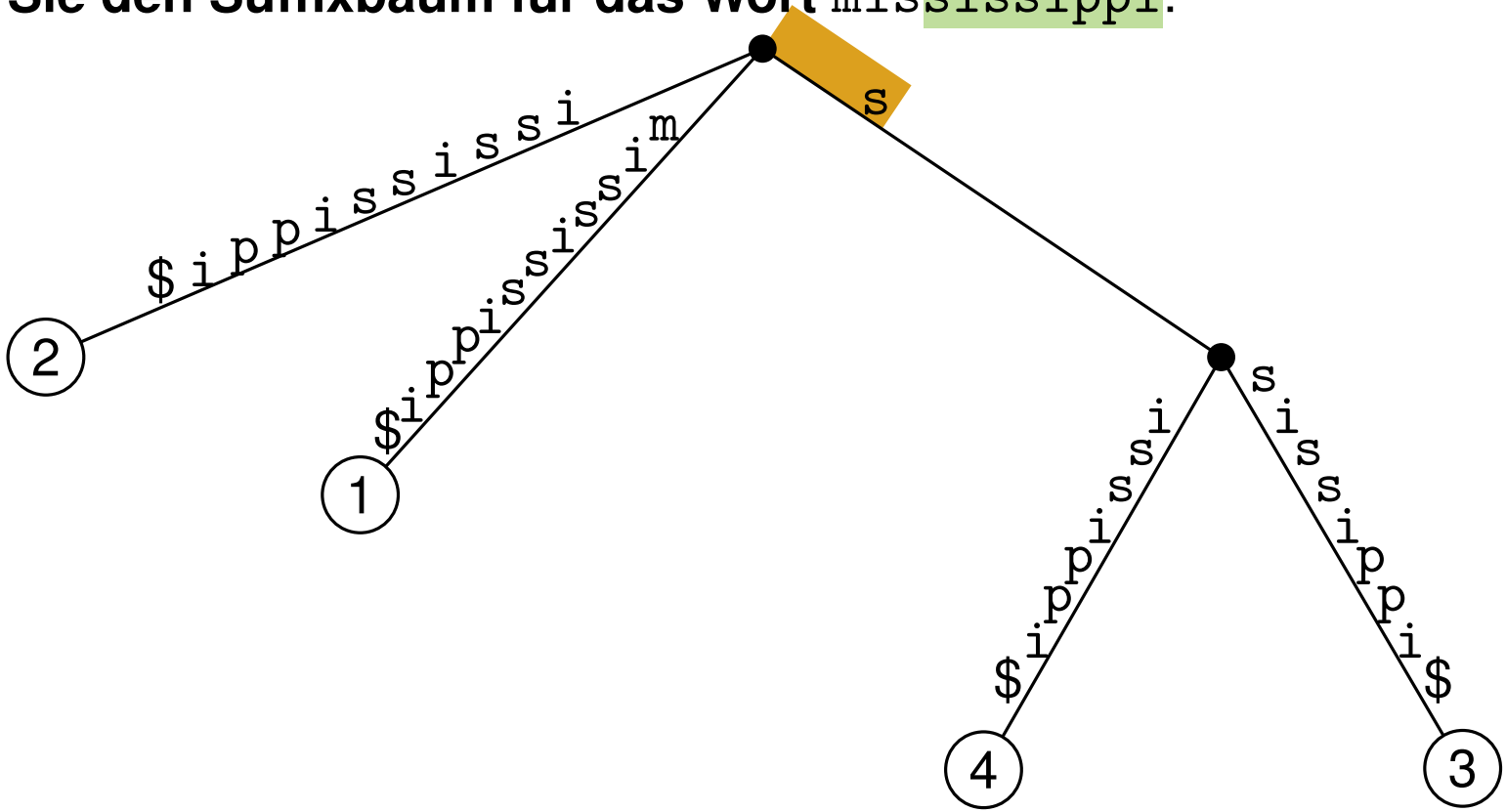


Füge S_4=`sissippi$` ein: Spalte Pfad von S_3 nach gemeinsamen Anteil auf.

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`:

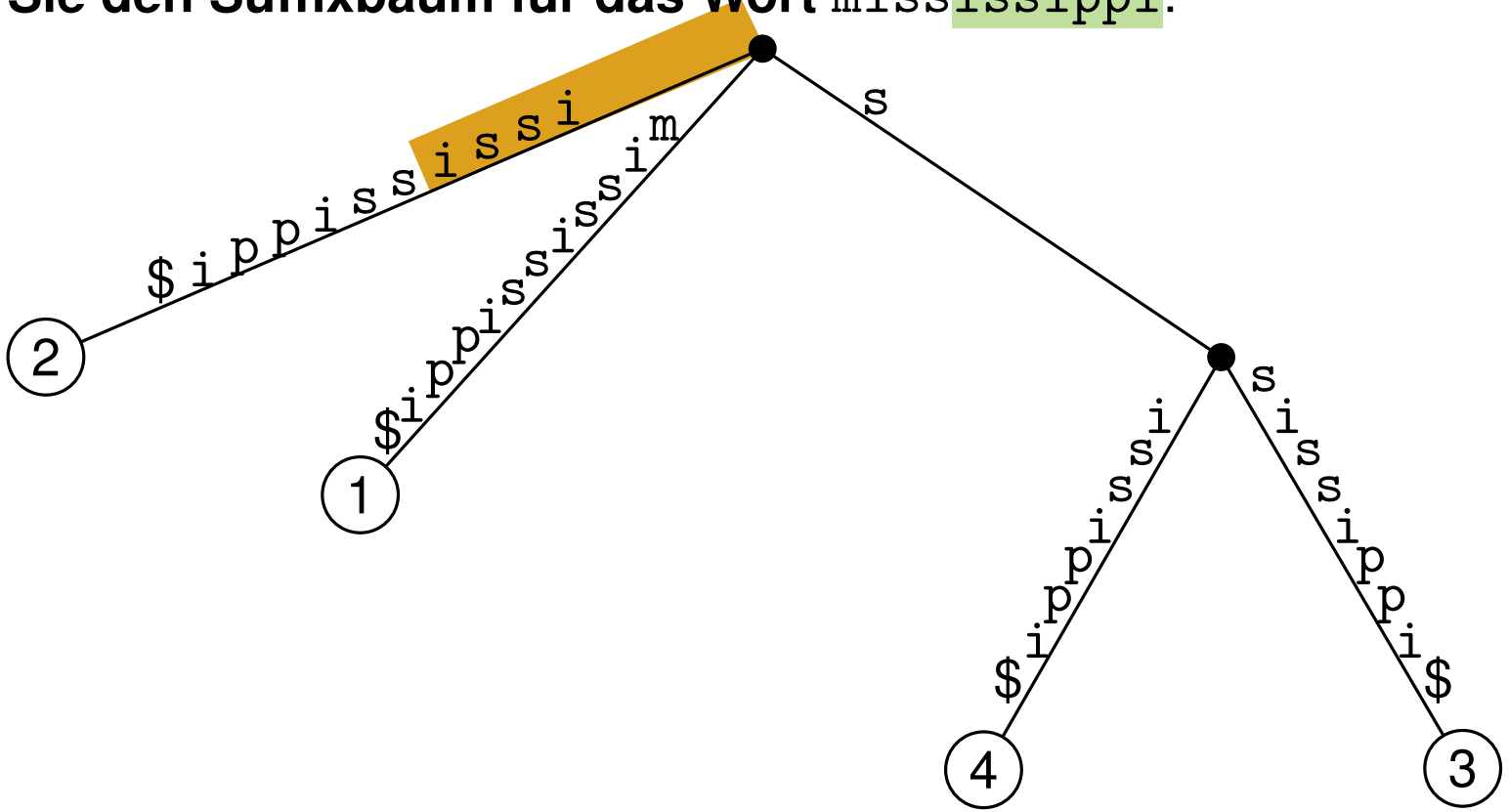


Füge S_4=`sissippi$` ein: Spalte Pfad von S_3 nach gemeinsamen Anteil auf.

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`:

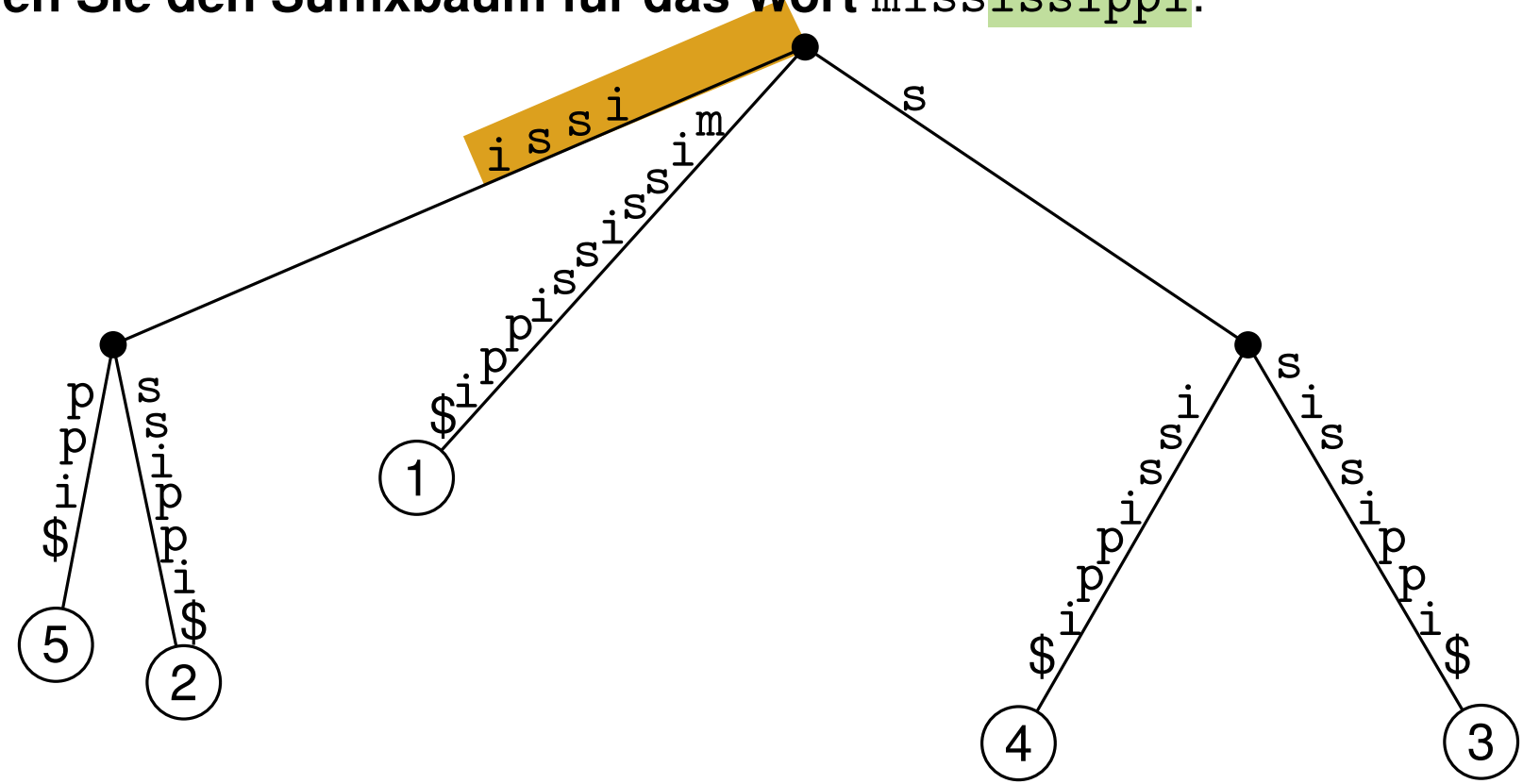


Füge S_5=`issippi$` ein: Spalte Pfad von S_2 nach gemeinsamen Anteil auf.

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`:

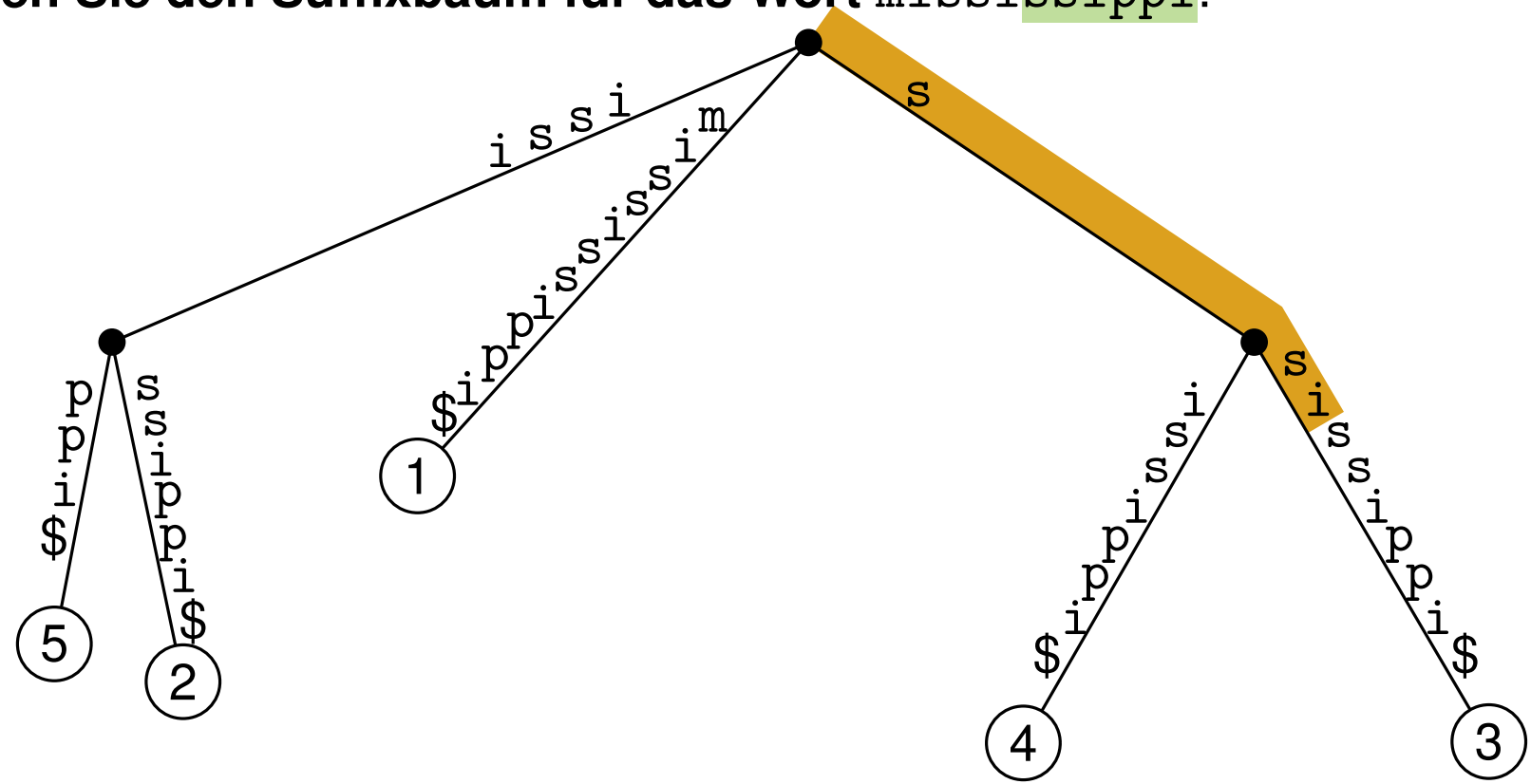


Füge S_5=`issippi$` ein: Spalte Pfad von S_2 nach gemeinsamen Anteil auf.

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`:

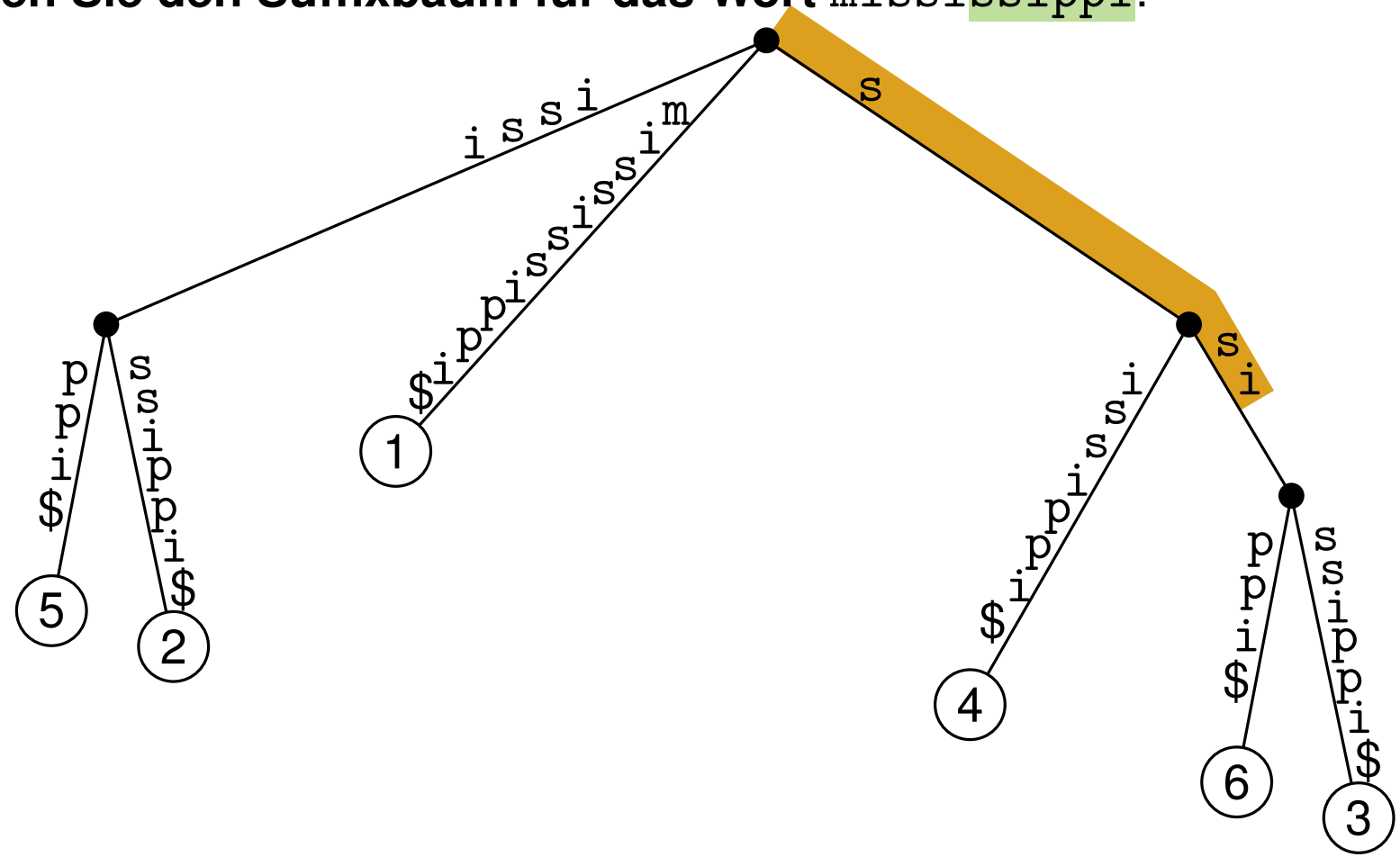


Füge S_6=`ssippi$` ein: Spalte Pfad von S_3 nach gemeinsamen Anteil auf.

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`:

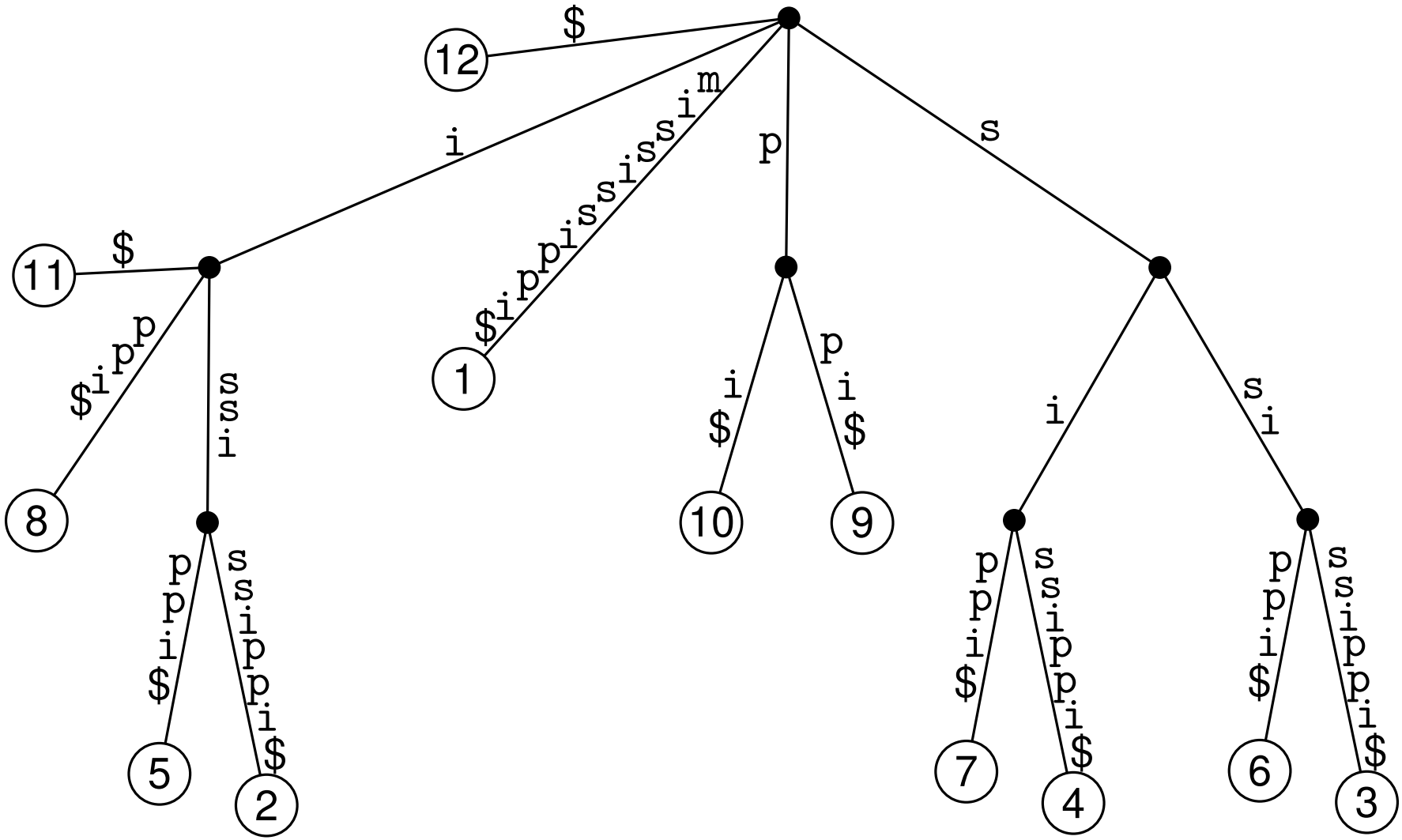


Füge S_6=`ssippi$` ein: Spalte Pfad von S_3 nach gemeinsamen Anteil auf.

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

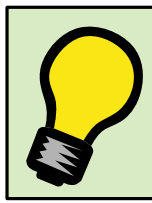
Problem 1

Berechnen Sie den Suffixbaum für das Wort `mississippi`:

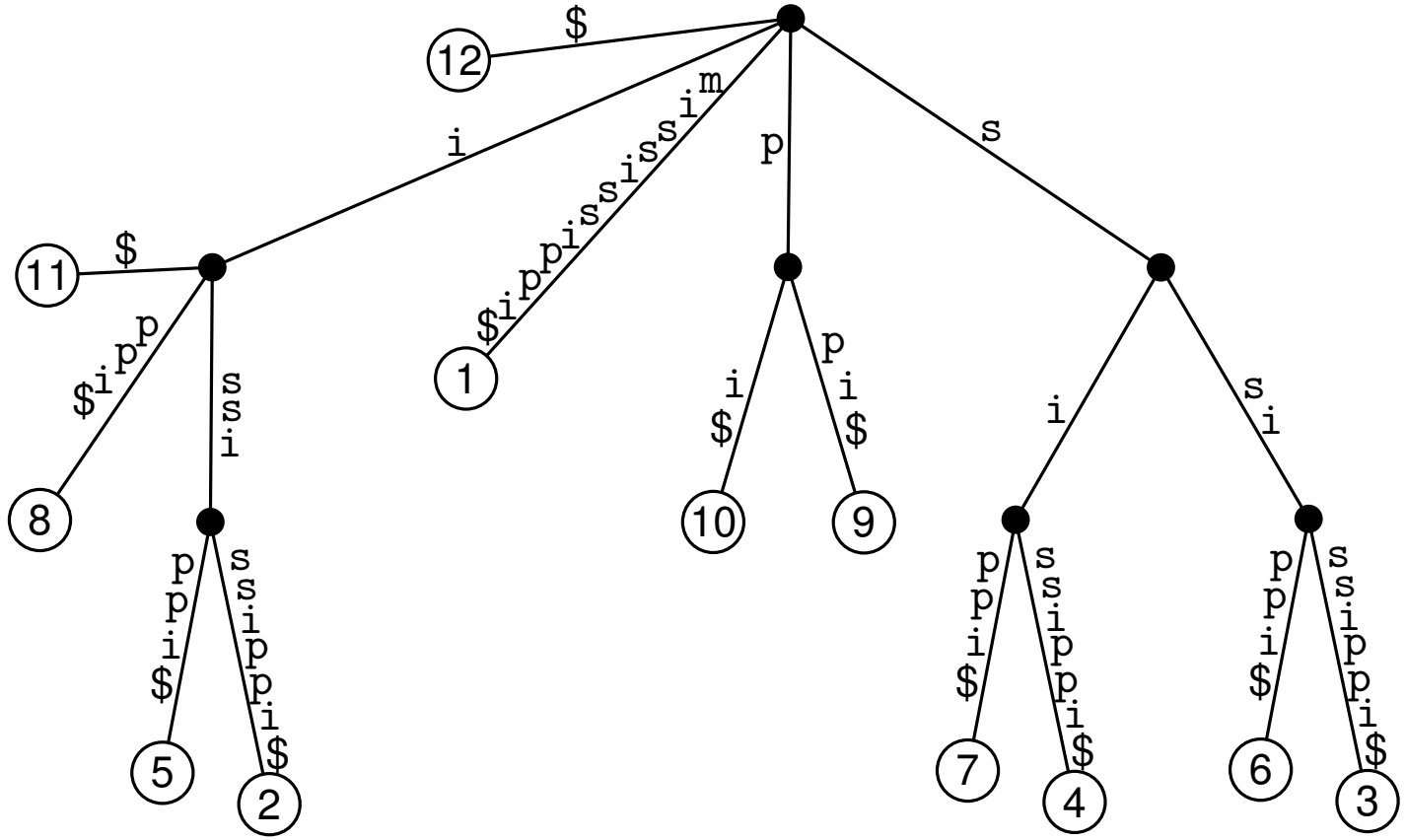


Idee: Konstruiere $N_1, \ldots, N_n$ Bäume, sodass N_i die Suffixe $S_1, \ldots, S_i$ enthält.

Problem 2



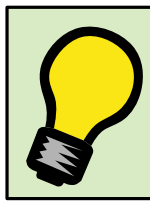
Implementierungsdetail: Jede Kantenbeschriftung B ist durch ein Paar (i, j) mit $1 \leq i \leq j \leq n$ repräsentiert, sodass $B = T[i, j]$.
$\Rightarrow$ Da S genau n Blätter hat, ergibt sich damit $O(n)$ Speicherverbrauch.



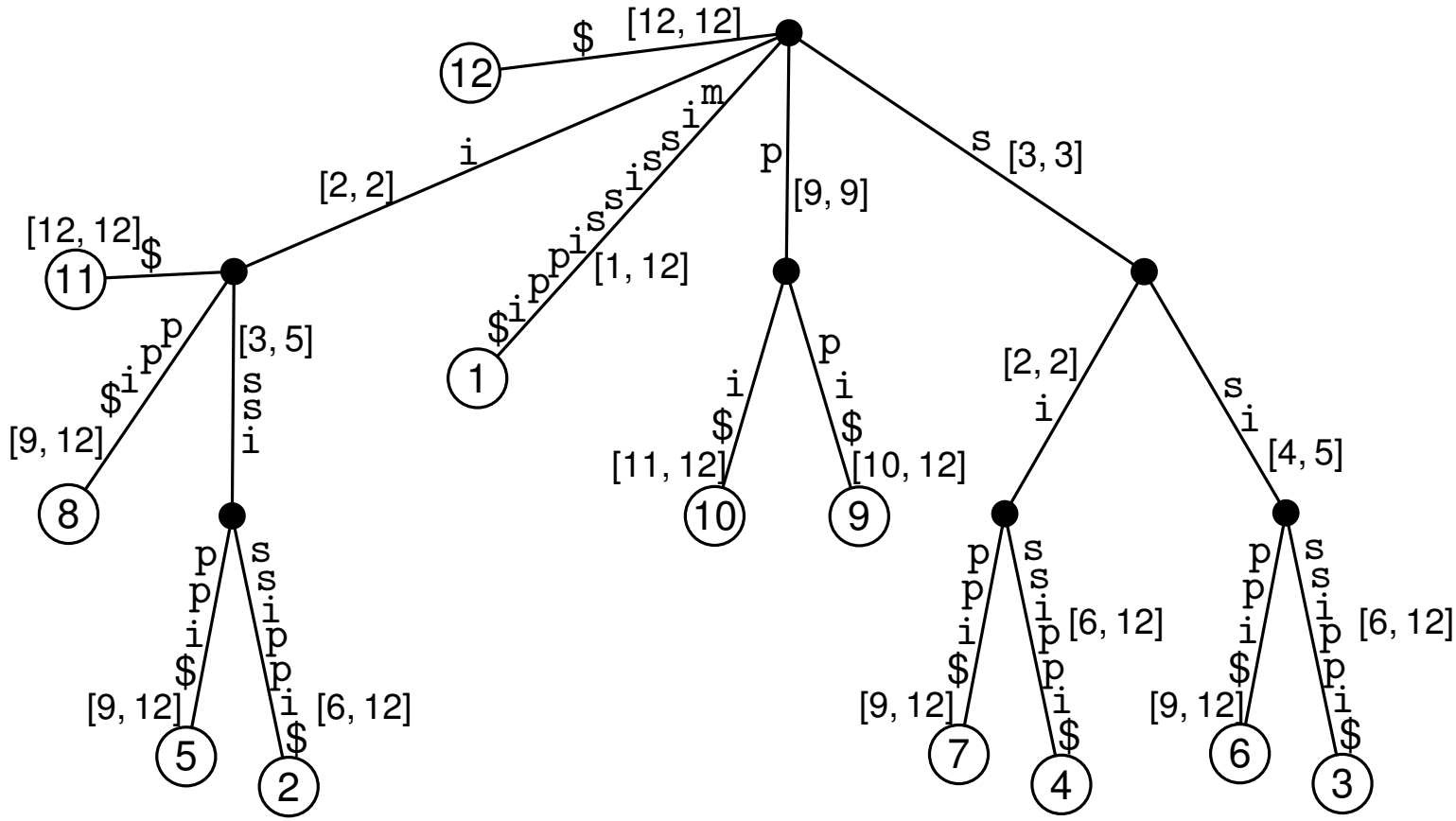
1	2	3	4	5	6	7	8	9	10	11	12
m	i	s	s	i	s	s	i	p	p	i	$

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?

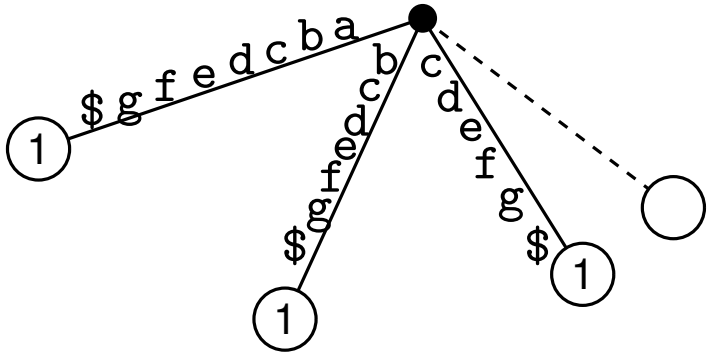
Problem 2

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?

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

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



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

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?

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?

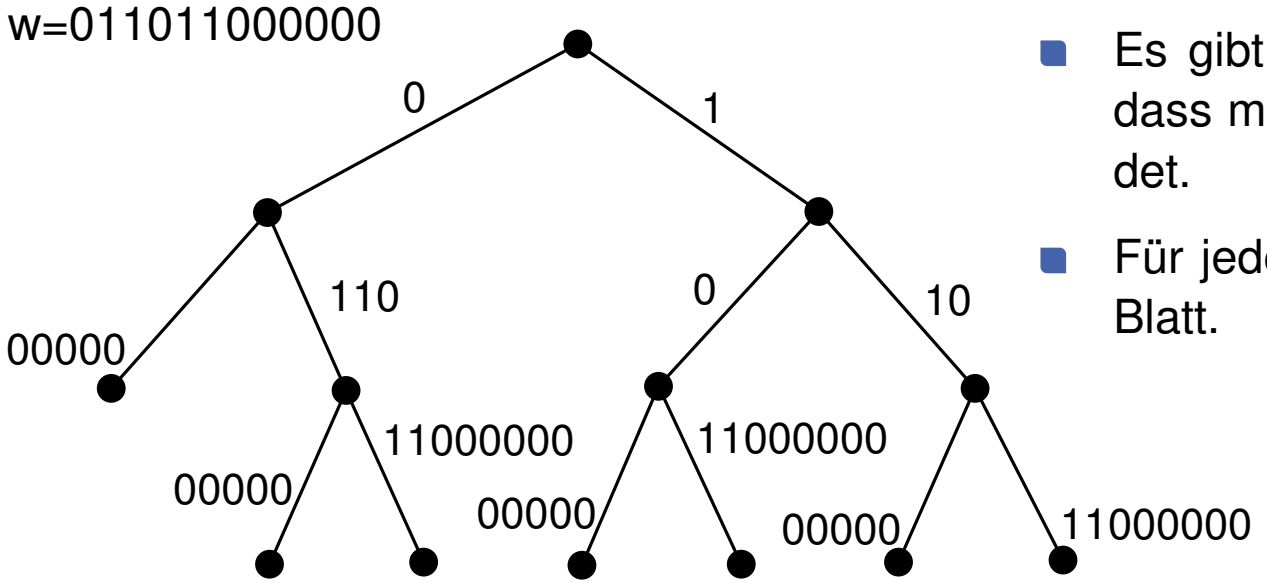
Was wenn Alphabet Σ fest ist? $\Sigma = \{0, 1\}$

Betrachte Wort: $w = w_1 w_2$

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

$w_2 = 0 \ldots 0$, bestehend aus $k \log k$ Nullen.

$|w| = O(k \log k)$



- Es gibt für jede Stelle in w_1 ein Suffix, dass mit mit mindestens $|w_2|$ Nullen endet.
- Für jedes Suffix von w_1 gibt es eigenes Blatt.

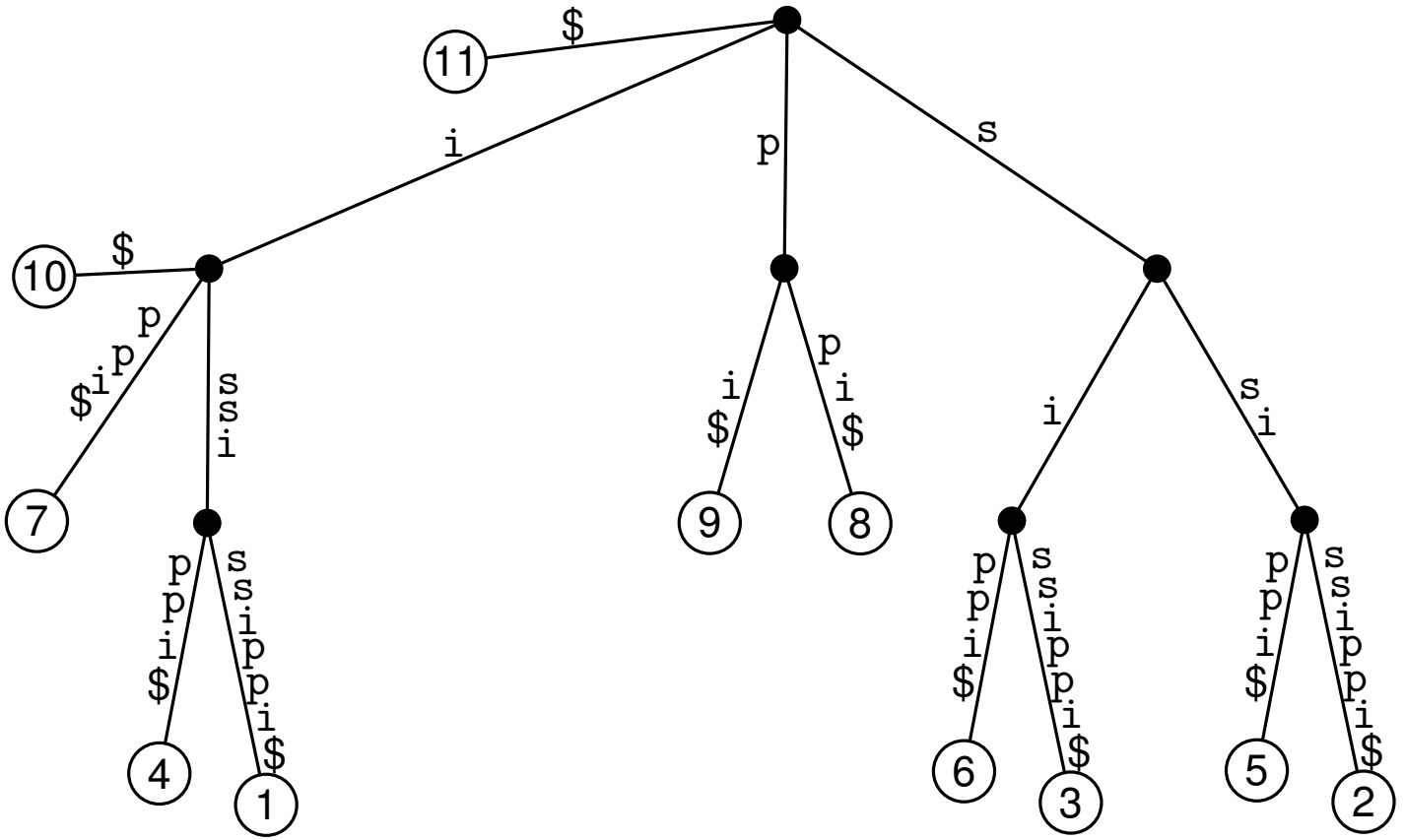
Speicherverbrauch: $O(k^2 \log k)$

Problem 3

Berechnen Sie alle kürzesten eindeutigen Teilstrings in einem Text T.

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

T=issississippi$

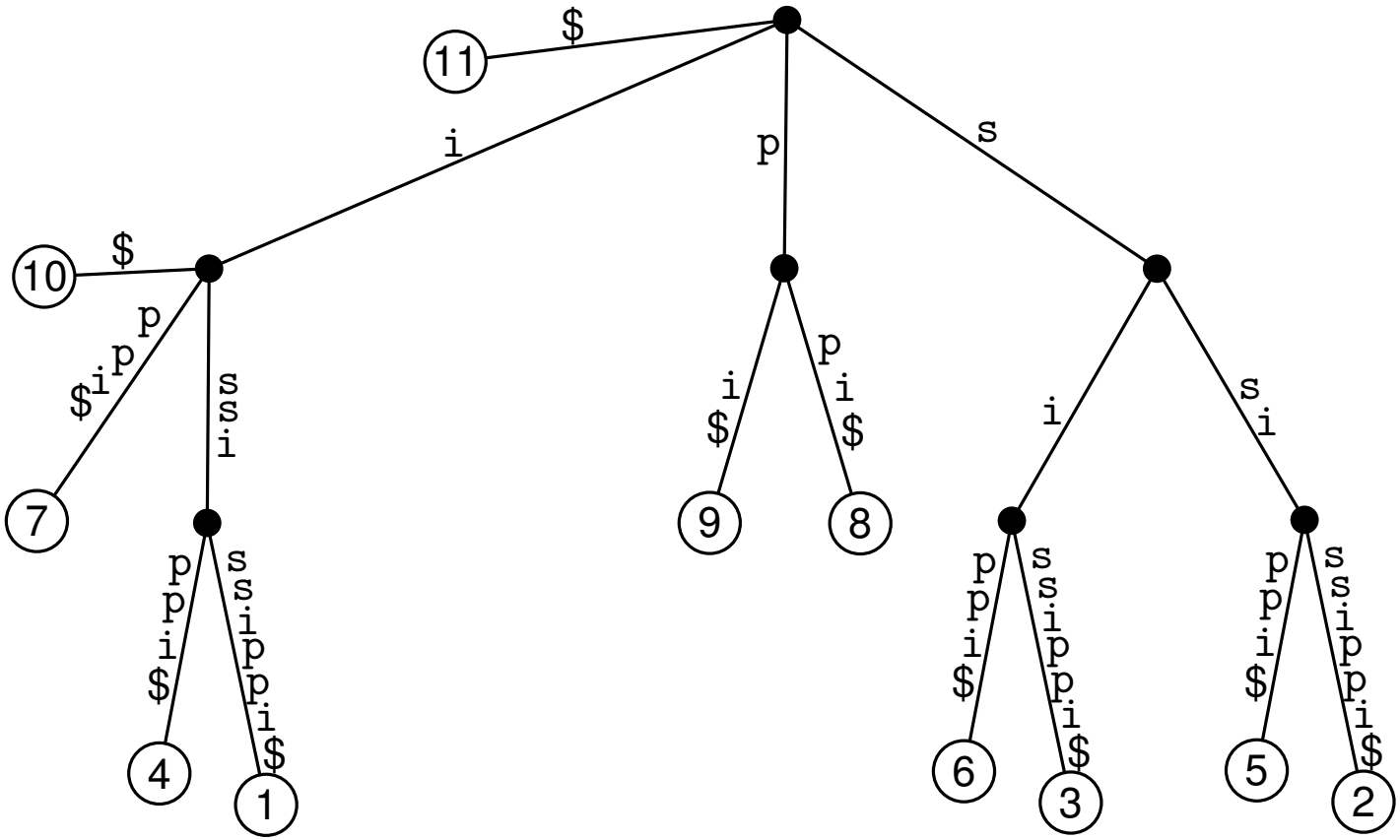


Problem 3

Berechnen Sie alle kürzesten eindeutigen Teilstrings in einem Text T.

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

T=ississippi$



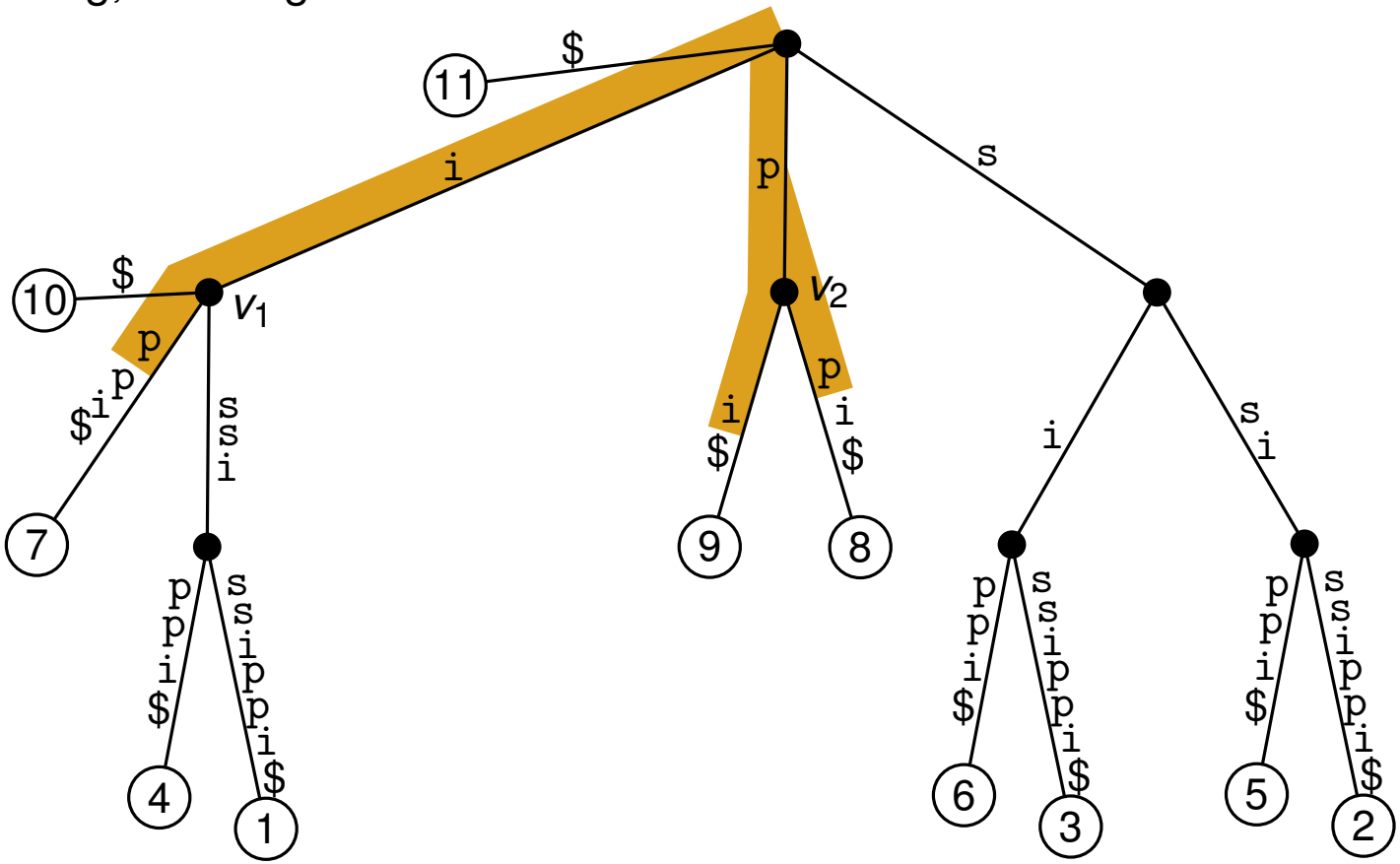
Beobachtung: Eindeutige Teilstrings enden auf einer Kante, die in ein Blatt mündet und nicht nur $ enthält.

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$



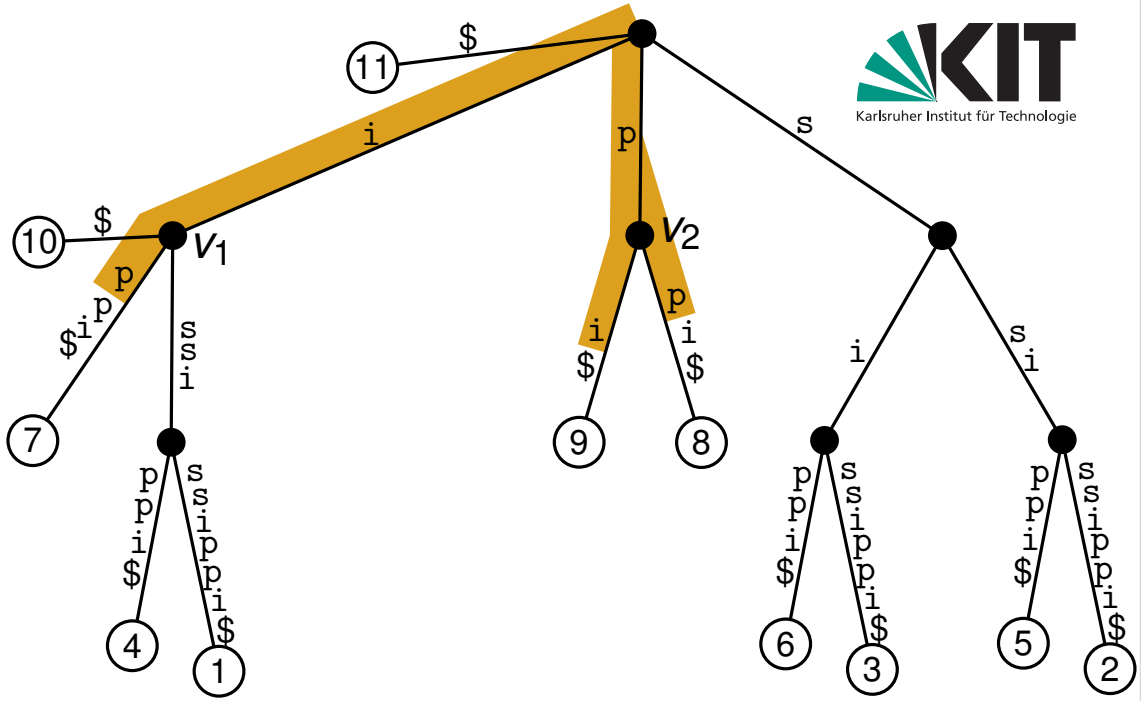
Beobachtung: Eindeutige Teilstrings enden auf einer Kante, die in ein Blatt mündet und nicht nur $ enthält.

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.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 3

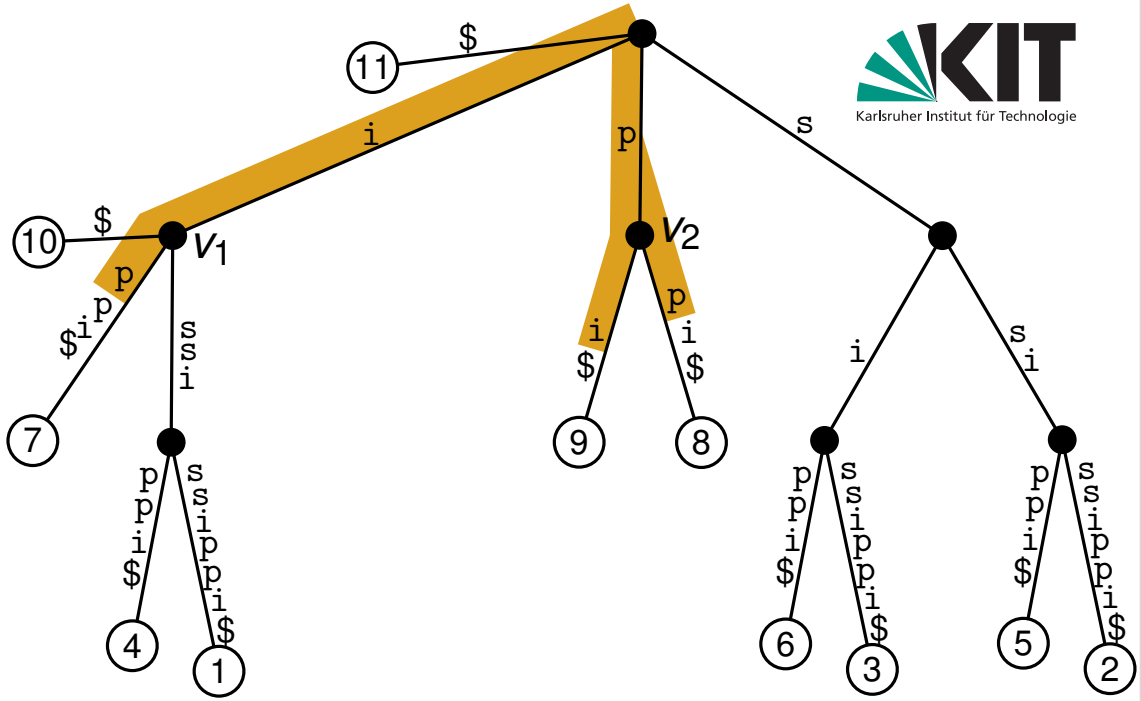
Eingabe: Suffixbaum für Text T.
Ausgabe Kürzeste eindeutige Teilstrings von T.
1. Schritt:
Bestimme Länge ℓ_{min} der kürzesten eindeutigen Teilstrings.
2. Schritt:
Gebe alle eindeutigen Teilstrings mit Länge ℓ_{min} aus.



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 von T.
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

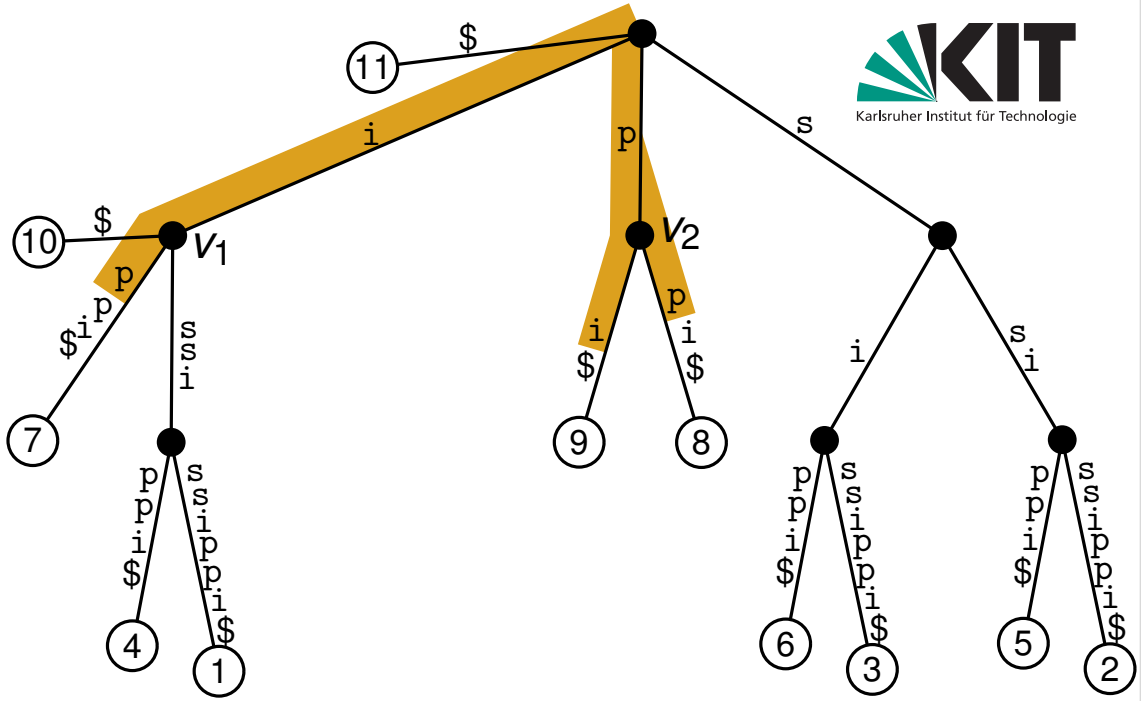
$\ell_{min} \leftarrow n$

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

Problem 3

Eingabe: Suffixbaum für Text T.
Ausgabe Kürzeste eindeutige Teilstrings von T.
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

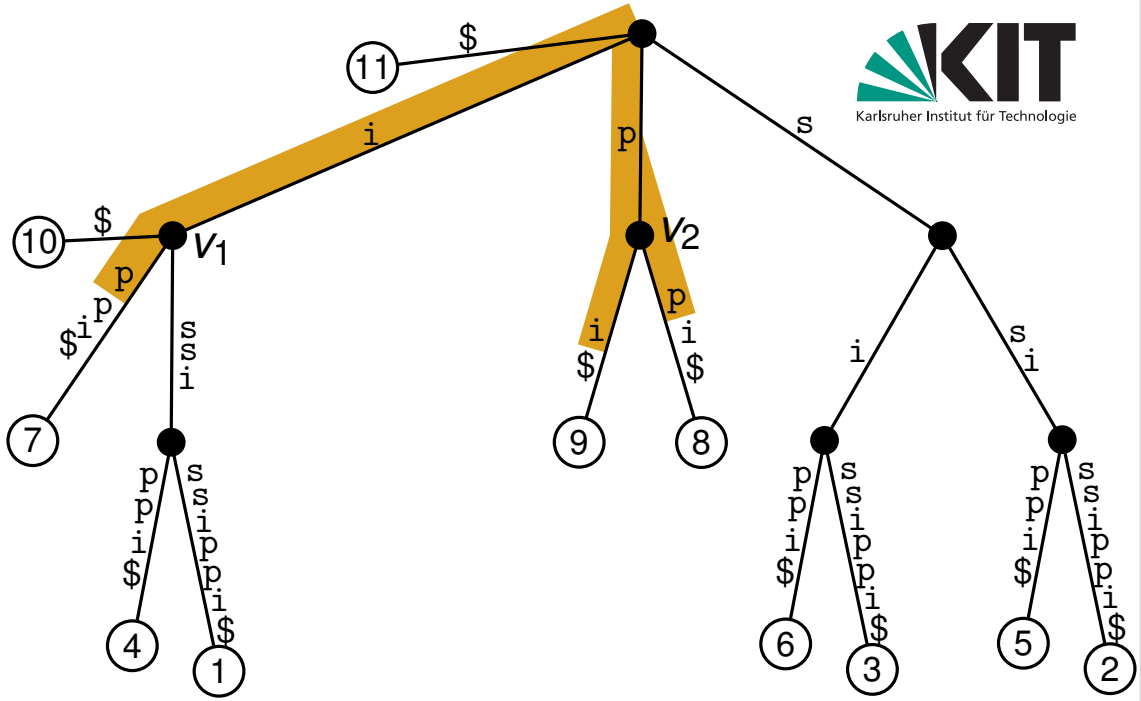
$\ell_{min} \leftarrow n$

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

Problem 3

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



2. Schritt:

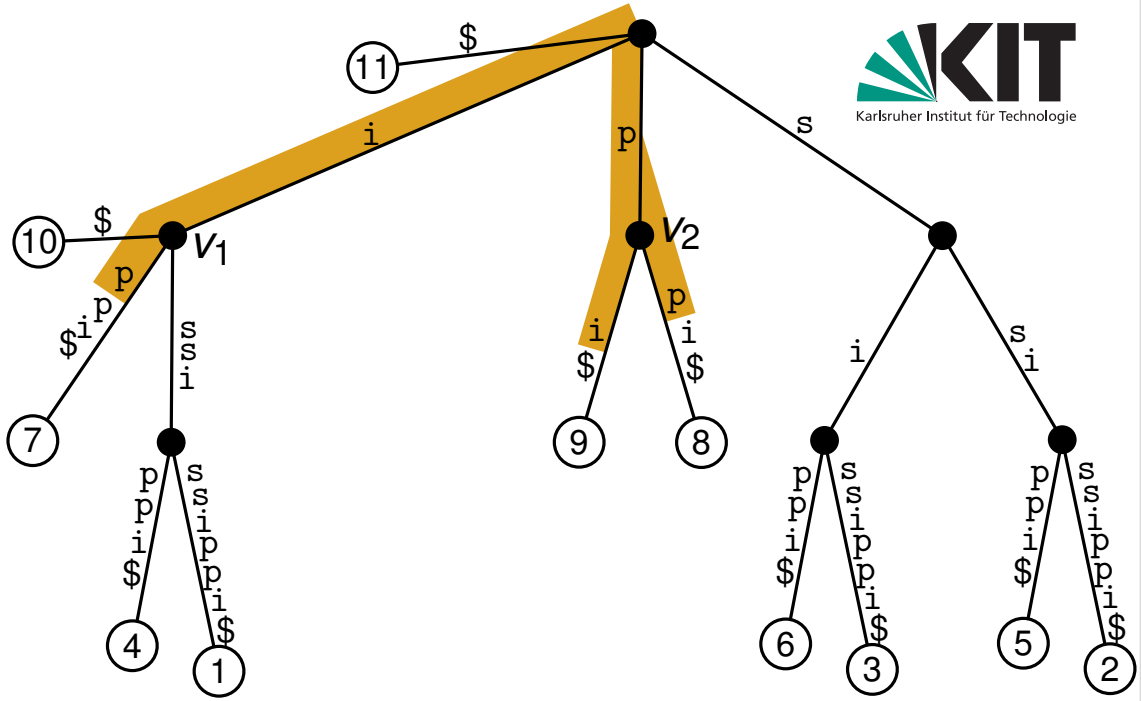
Q = leere Warteschlange

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 von T.
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.

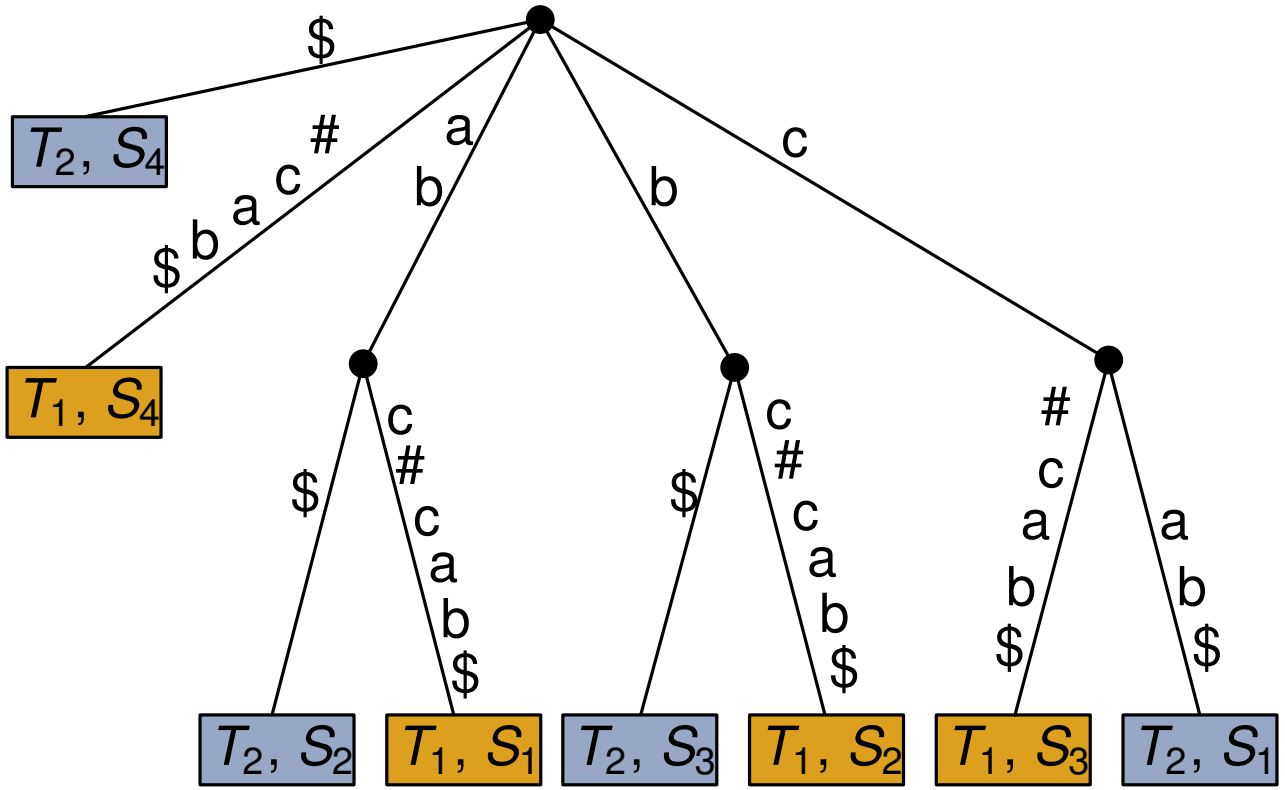
solange *Q nicht leer* **tue**
  $u \leftarrow$ entferne erstes Element aus Q.
  für *ausgehende Kanten* (u, v) *von* u **tue**
    wenn *v Blatt und* $|B(u, v)| > 1$ *und* $d(u) = \ell_{min} - 1$ **dann**
      $s \leftarrow$ String entlang Pfad von Wurzel zu u + erstes Zeichen auf (u, v).
      Gebe s aus.
    Füge v in Q ein

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



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 4

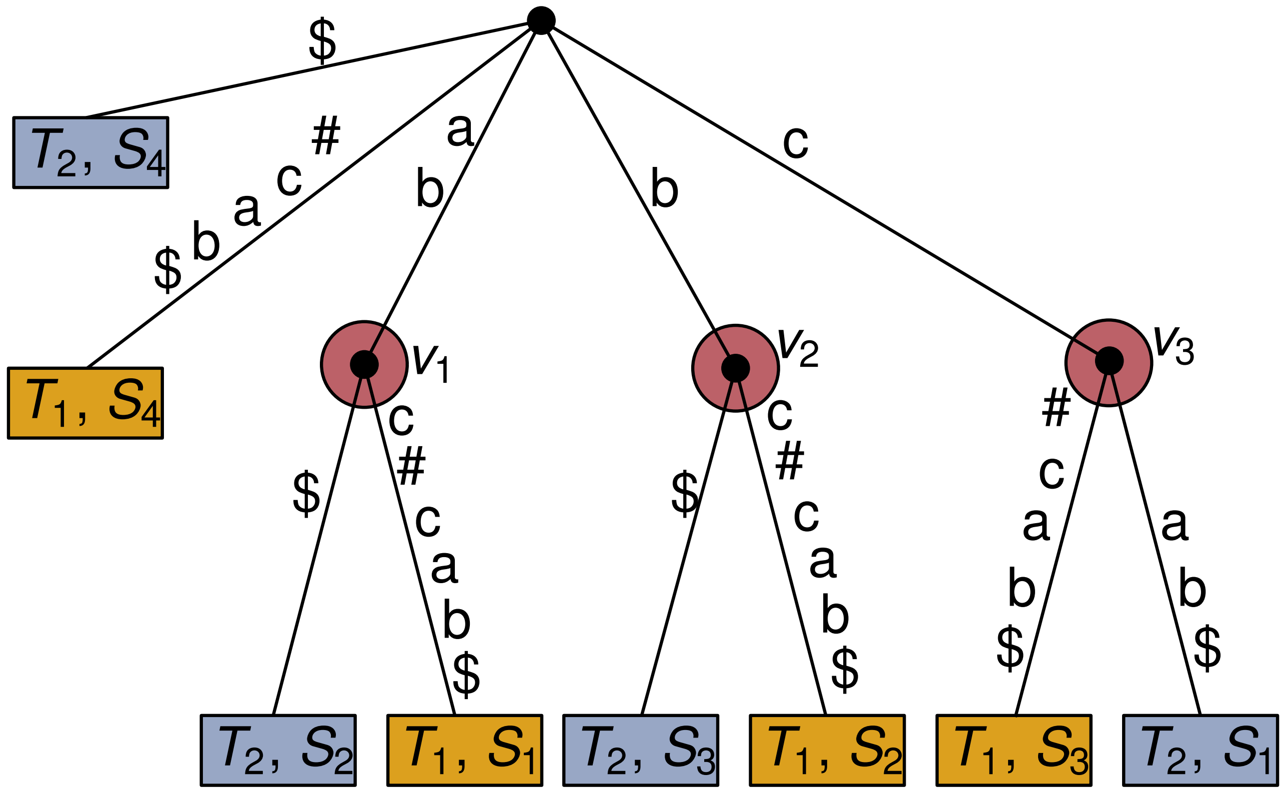
Gegeben: Text T_1 und T_2.

Gesucht: Längster gemeinsamer Teilstring von T_1 und T_2.

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

T_1 =abc

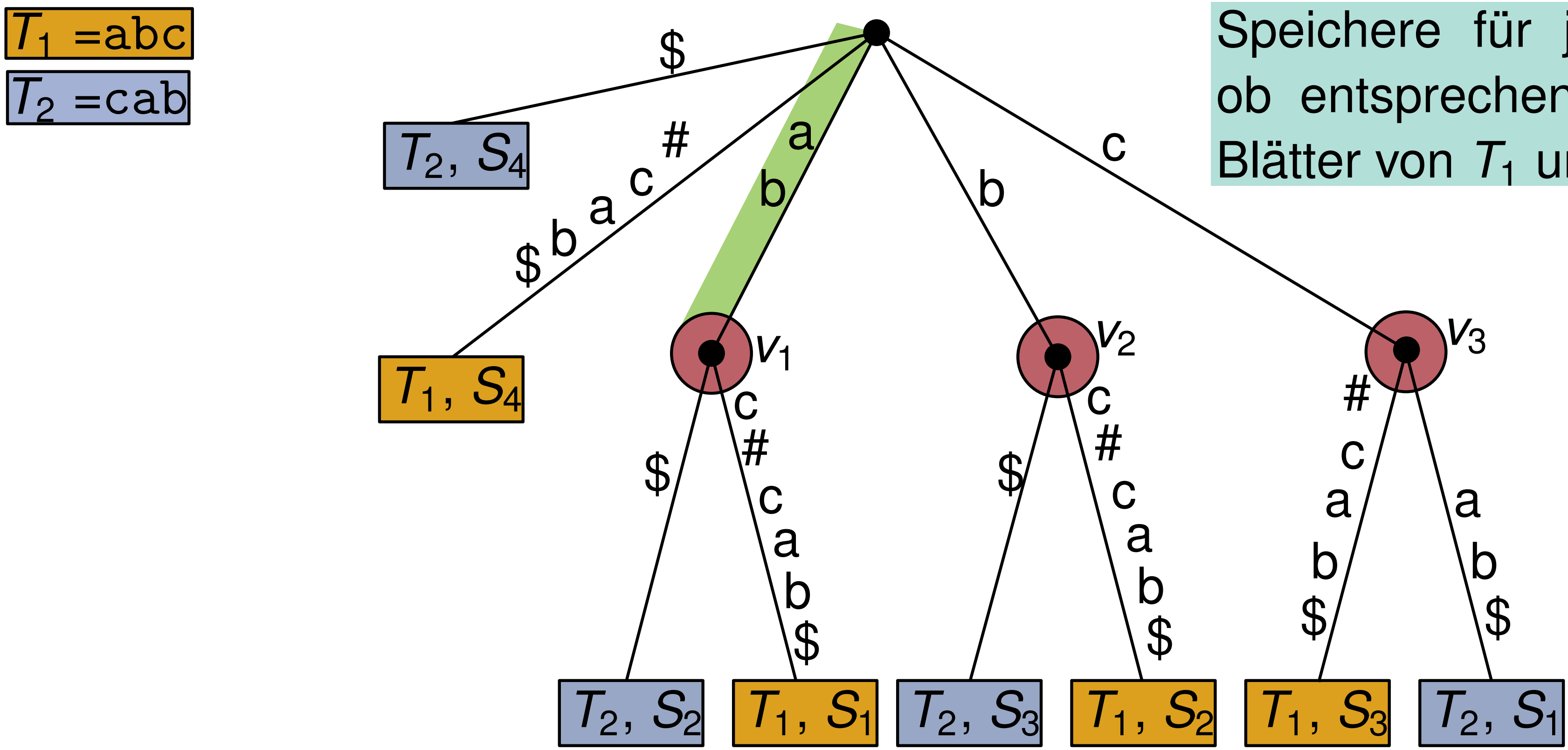
T_2 =cab



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 $T_1\#T_2\$$



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.

Einschub: Radix-Sort

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

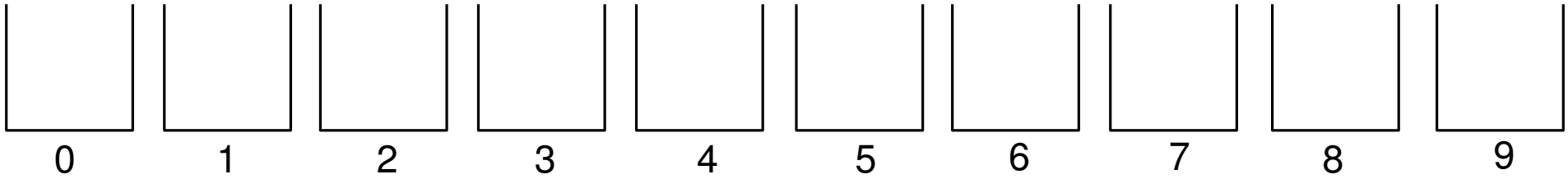
Annahme: Jede Zahl hat m Ziffern.

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.

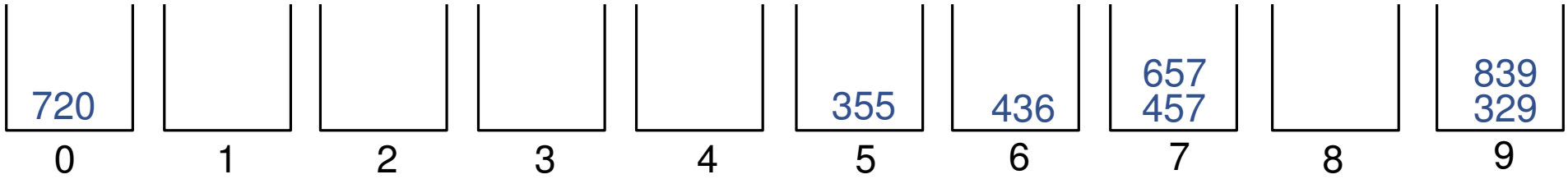
Annahme: Jede Zahl hat m Ziffern.

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.

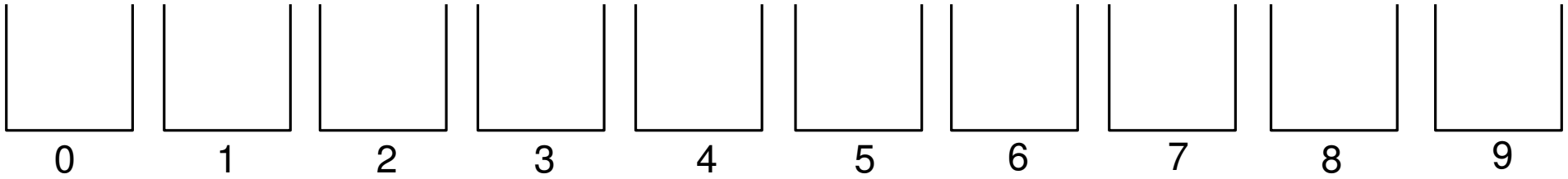
Annahme: Jede Zahl hat m Ziffern.

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.

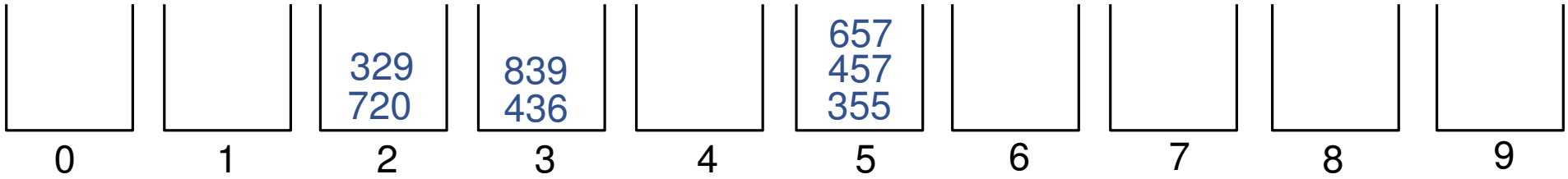
Annahme: Jede Zahl hat m Ziffern.

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.

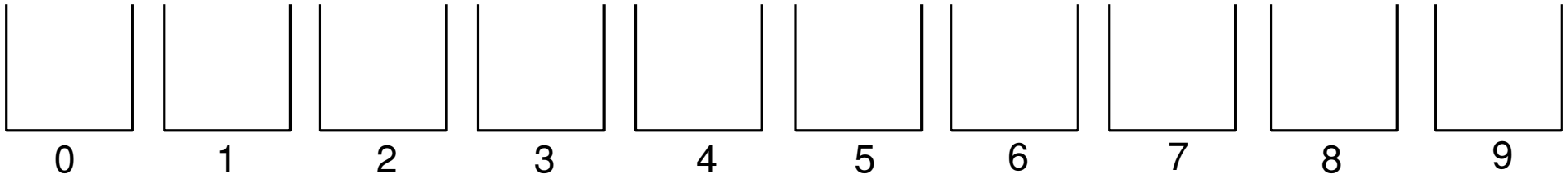
Annahme: Jede Zahl hat m Ziffern.

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.

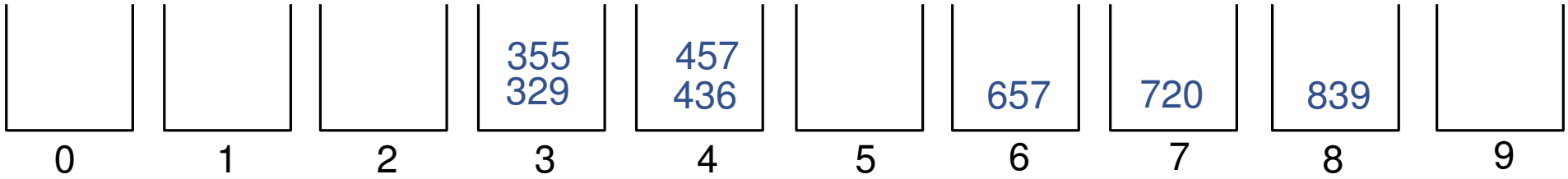
Annahme: Jede Zahl hat m Ziffern.

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.

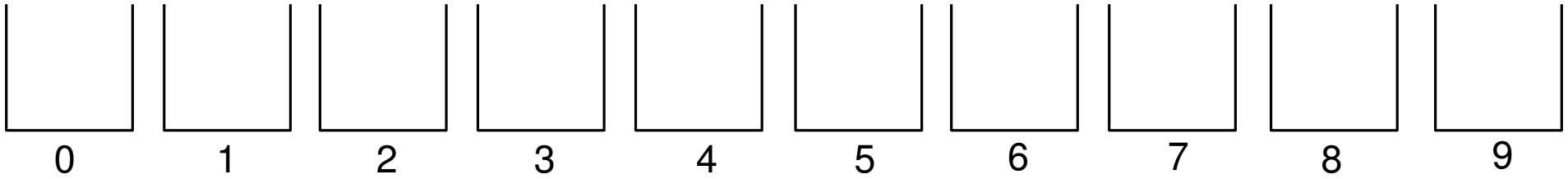
Annahme: Jede Zahl hat m Ziffern.

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`:

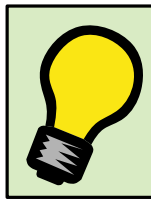
Definition: Ein *Suffix-Array* A eines Textes T ist eine Permutation von $\{1, \ldots, n\}$, sodass $S_{A[i]}$ das i-kleinste Suffix in lexikographischer Ordnung ist: $S_{A[i-1]} < S_{A[i]}$ für alle $1 < i \leq n$.

In der Vorlesung zwei Möglichkeiten vorgestellt:

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

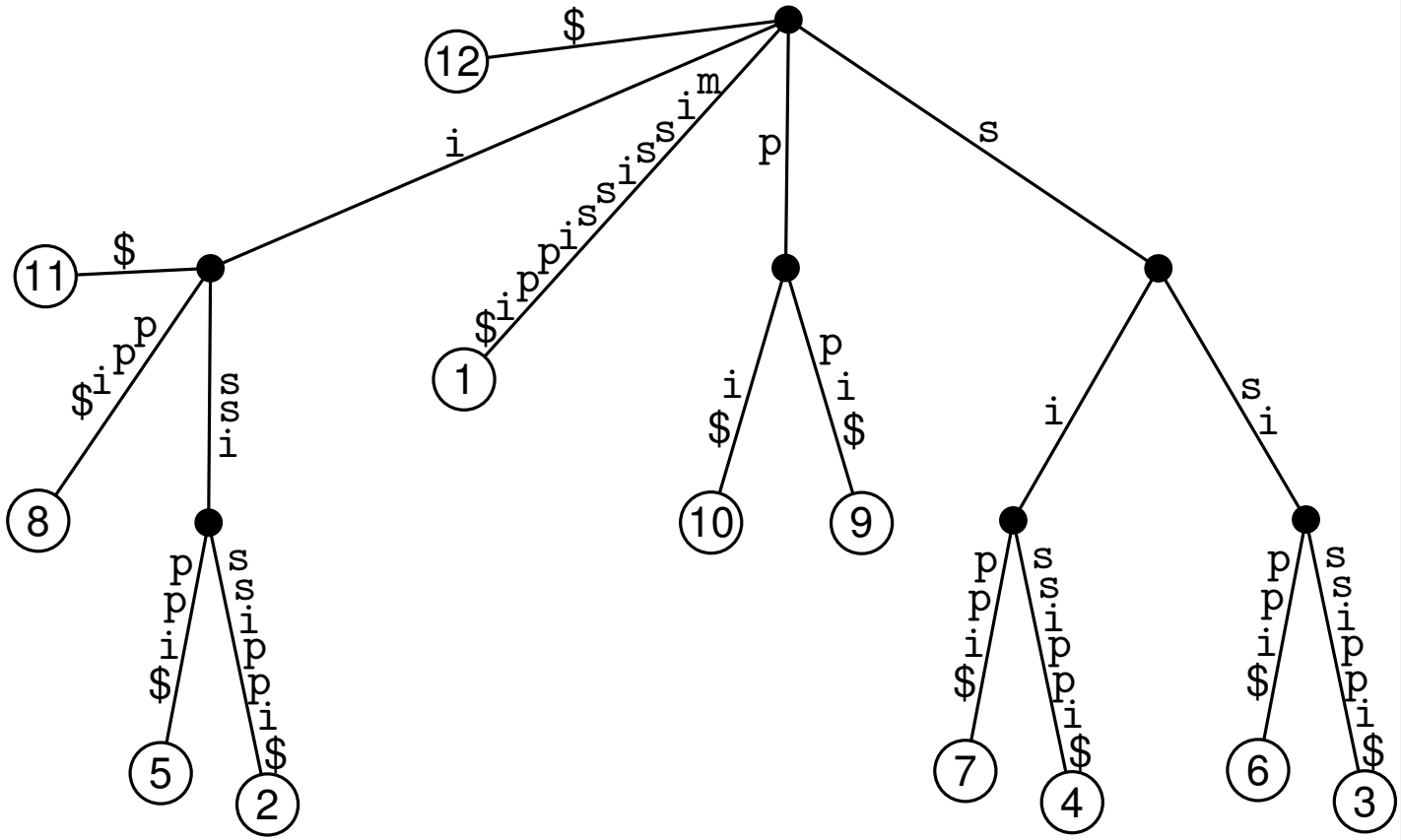
Problem 1

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



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$

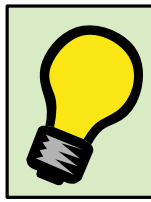


Suffix-Array gibt lexikographische Ordnung der Suffixe an!

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

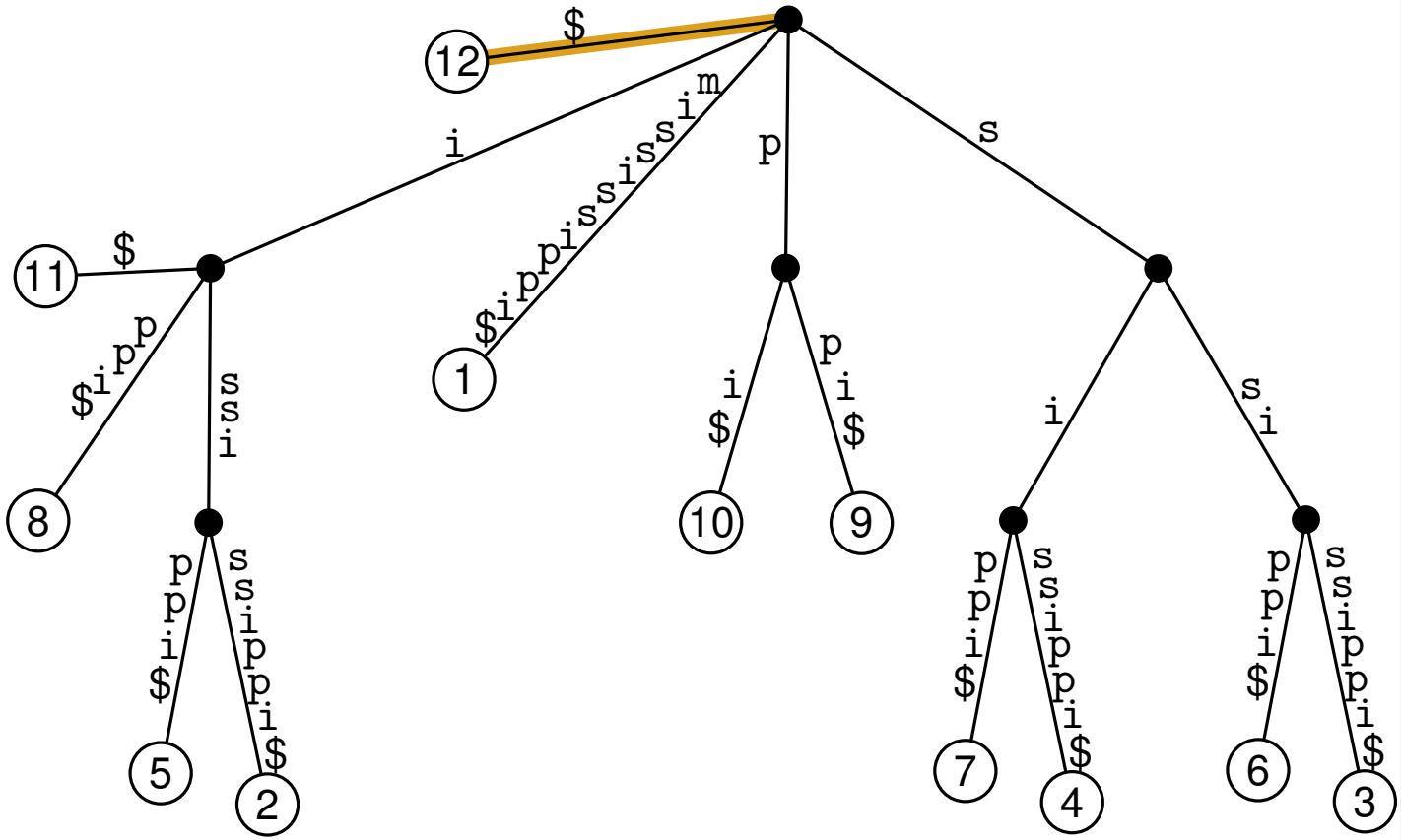
Problem 1

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$

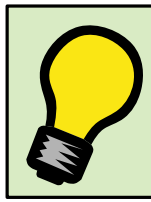


Suffix-Array gibt lexikographische Ordnung der Suffixe an!

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

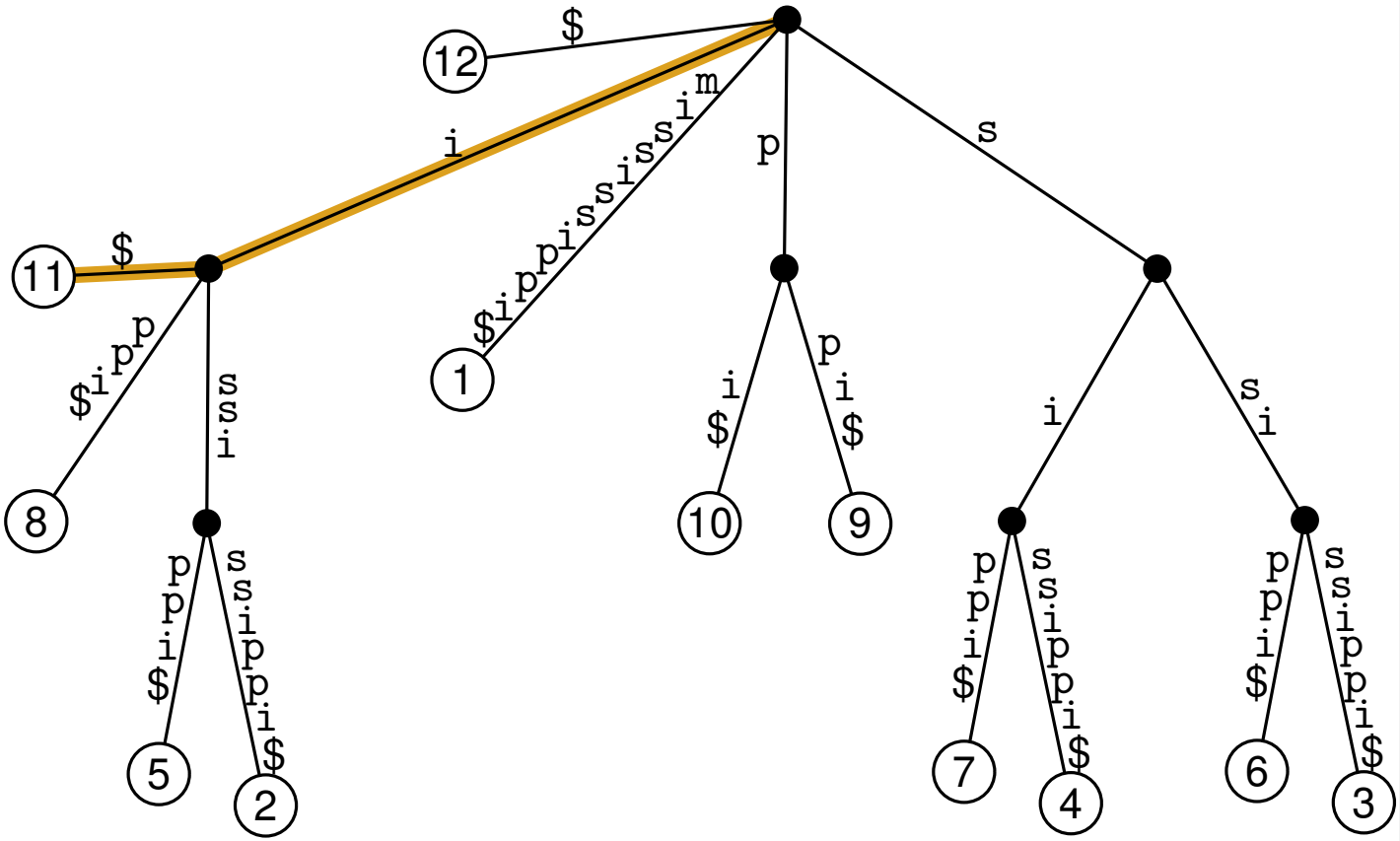
Problem 1

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$

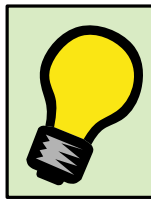


Suffix-Array gibt lexikographische Ordnung der Suffixe an!

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

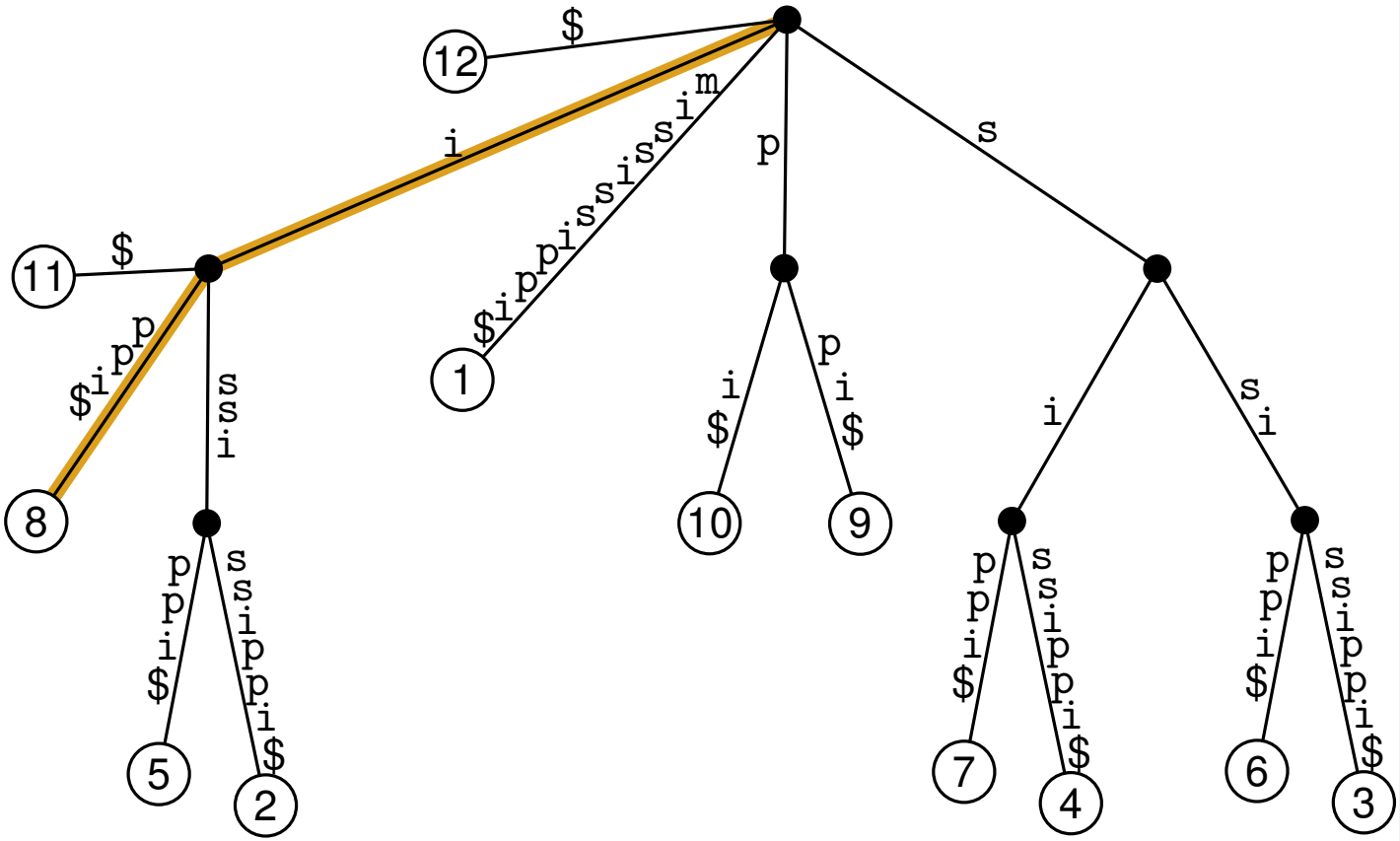
Problem 1

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.

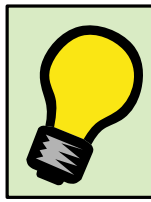
12	$
11	i$
8	ippi$
5	issippi$
2	ississippi$
1	mississippi$
10	pi$
9	ppi $
7	sippi$
4	sissippi$
6	ssippi$
3	ssissippi$



Suffix-Array gibt lexikographische Ordnung der Suffixe an!

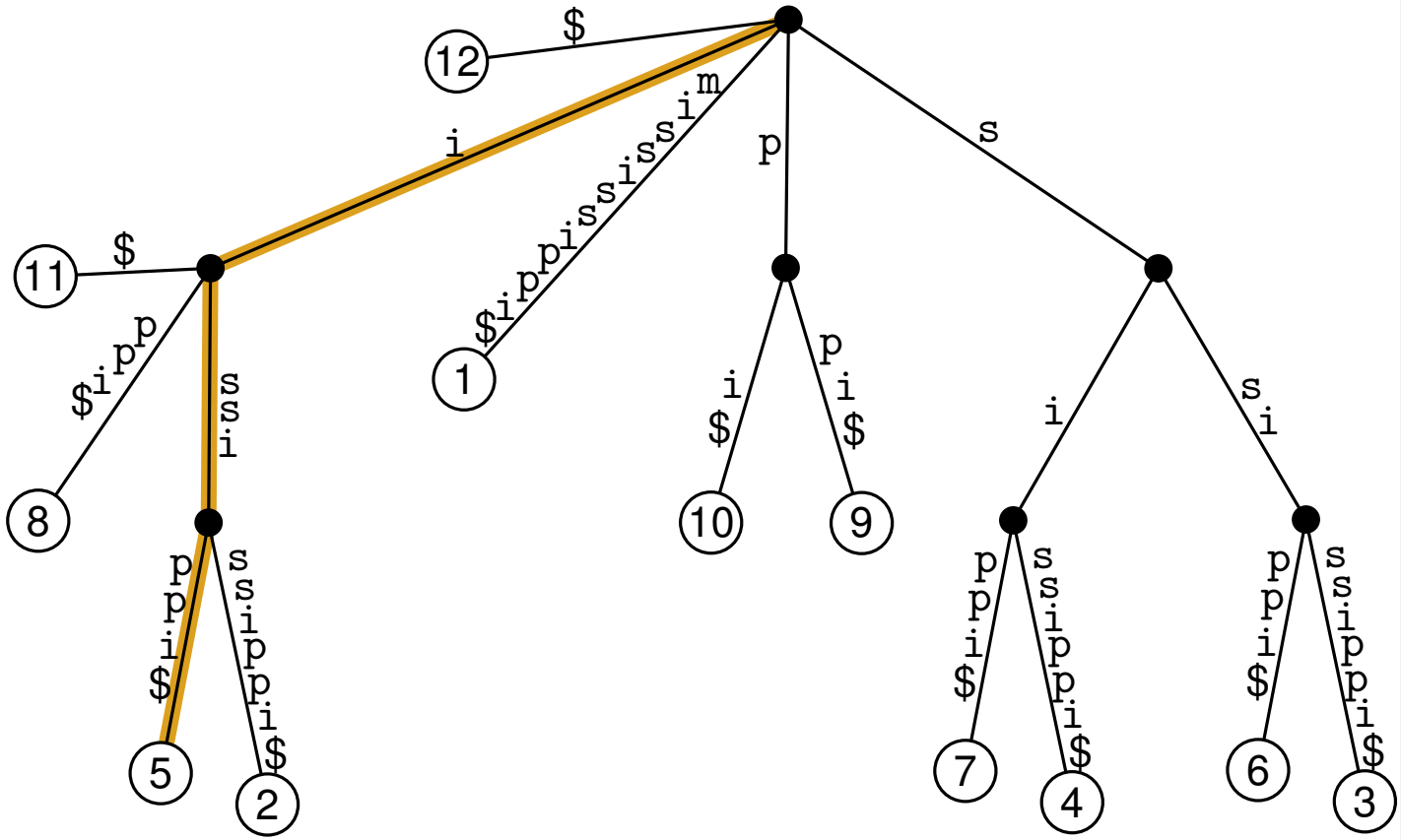
Problem 1

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$

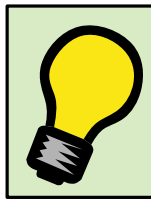


Suffix-Array gibt lexikographische Ordnung der Suffixe an!

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

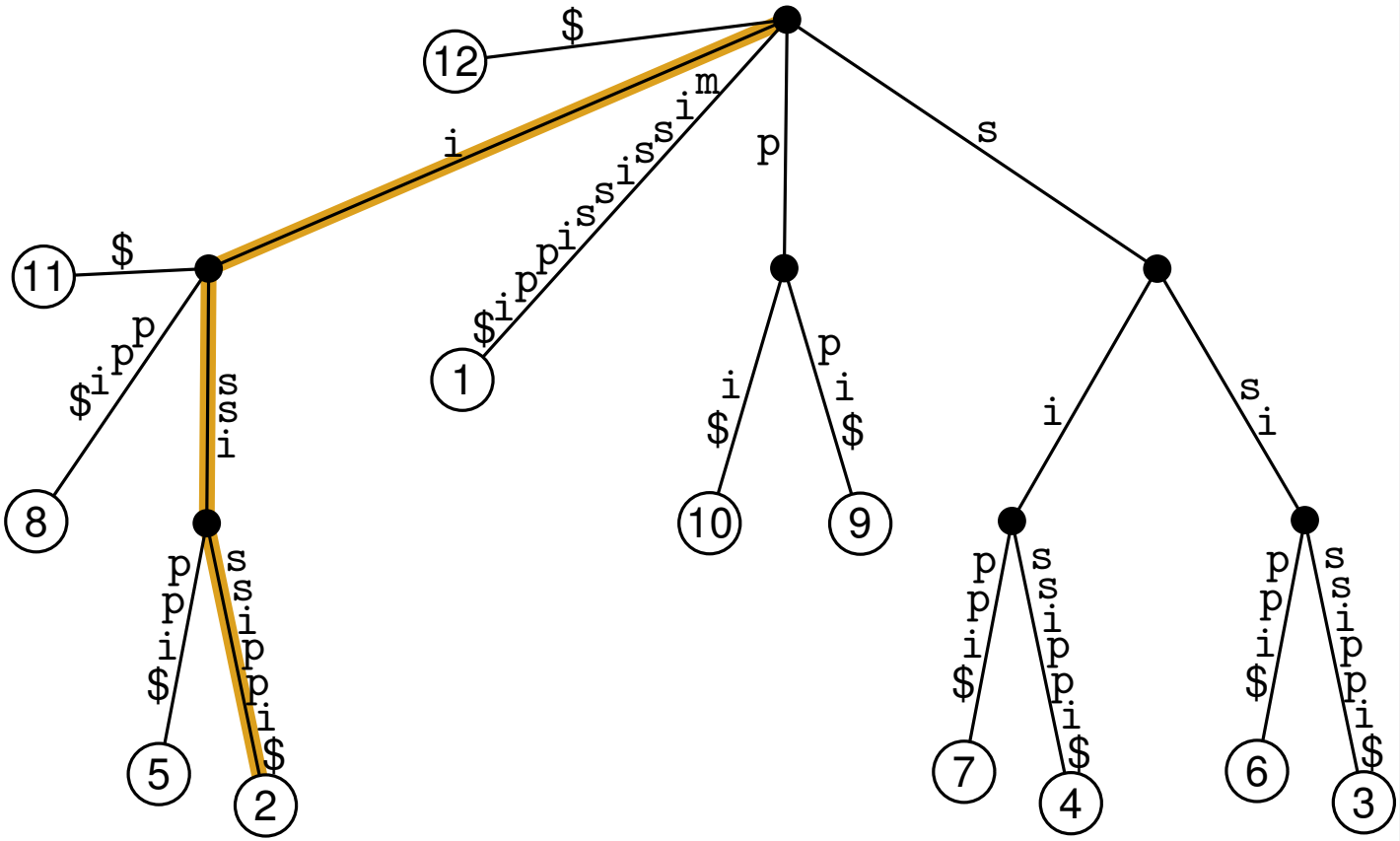
Problem 1

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

Notation:

- Text $T := t_0 t_1 \ldots t_{n-1}$
- $x \mod y = z$ wird abgekürzt als $x \equiv z(y)$ (z ist Rest bei ganzzahliger Division von x durch y).
- A_0 := Suffix-Array aller Suffixe S_i in T mit $i \equiv 0(3)$.
- A_{12} := Suffix-Array aller Suffixe S_i in T mit $i \not\equiv 0(3)$.

S_{11}	\$
S_{10}	i\$
S_7	ippi\$
S_4	issippi\$
S_1	ississippi\$
S_0	mississippi\$
S_9	pi\$
S_8	ppi \$
S_6	sippi\$
S_3	sissippi\$
S_5	ssippi\$
S_2	ssissippi\$

Beispiel:

T=	0	1	2	3	4	5	6	7	8	9	10	11
	m	i	s	s	i	s	s	i	p	p	i	\$

A_{12} für orange Suffixe definiert.

A_0 für blaue Suffixe definiert.

Bemerkung: Suffix-Arrays repräsentieren eine Ordnung der Suffixe. A_{12} definiert also eine Ordnung auf allen Suffixen S_i von T mit $i \not\equiv 0(3)$:

$A_{12} =$ | 11 | 10 | 7 | 4 | 1 | 8 | 5 | 2 |

Konstruktion von Suffix-Arrays

Skizze:

SUFFIXARRAY(Text $T = t_0 t_1 \ldots t_{n-1}$)

1. Berechne A_{12} von T.
2. Berechne A_0 von T.
3. Vermenge A_{12} mit A_0.

- Schritt 1 berechnet in $O(n)$ Zeit das Suffix-Array A_{12}. Hierzu wird ein Text T' aufgebaut, der $\frac{2}{3}$ so groß ist wie T und die Ordnung der Suffixe von T erhält. Rekursiver Aufruf von SUFFIXARRAY auf T' berechnet Suffix-Array von T', das dann in A_{12} transformiert wird.
- Schritt 2 berechnet in $O(n)$ aus A_{12} und T das Suffix-Array A_0.
- Schritt 3 vermengt A_{12} und A_0 zu Suffix-Array A.
- Insgesamt ergibt sich dann die Rekurenzgleichung

$$T(n) = T\left(\frac{2}{3}n\right) + O(n),$$

die sich zu $O(n)$ auflöst.

Erstellung von A_{12}.

Wie können S_i mit $i \not\equiv 0(3)$ schnell sortiert werden?

S_1	ississippi$
S_2	ssissippi$
S_4	issippi$
S_5	ssippi$
S_7	ippi$
S_8	ppi$
S_{10}	i$$
S_{11}	$$$

Erstellung von A_{12}.

Wie können S_i mit $i \not\equiv 0(3)$ schnell sortiert werden?

S_1	ississippi$
S_2	ssissippi$
S_4	issippi$
S_5	ssippi$
S_7	ippi$
S_8	ppi$
S_{10}	i$$
S_{11}	$$$

Sortiere Tripel →

0	$$$
1	i$$
2	ipp
3	iss
4	ppi
5	ssi

Kann mithilfe eines dreistufigen Radix-Sort in $O(n)$ Zeit sortiert werden.

Erstellung von A_{12}.

Wie können S_i mit $i \not\equiv 0(3)$ schnell sortiert werden?

S_1	ississippi$
S_2	ssissippi$
S_4	issippi$
S_5	ssippi$
S_7	ippi$
S_8	ppi$
S_{10}	i$$
S_{11}	$$$

Sortiere Tripel →

0	$$$
1	i$$
2	ipp
3	iss
4	ppi
5	ssi

Erstelle neuen Text T' in zwei Schritten:

Erstellung von A_{12}.

Wie können S_i mit $i \not\equiv 0(3)$ schnell sortiert werden?

S_1	ississippi$
S_2	ssissippi$
S_4	issippi$
S_5	ssippi$
S_7	ippi$
S_8	ppi$
S_{10}	i$$
S_{11}	$$$

Sortiere Tripel →

0	$$$
1	i$$
2	ipp
3	iss
4	ppi
5	ssi

Erstelle neuen Text T' in zwei Schritten:

1. Schritt: Konkatenation der Tripel

$T' =$ [iss][iss][ipp][i$$] [ssi][ssi][ppi][$$$]

S_1 S_4 S_7 S_{10} S_2 S_5 S_8 S_{11}

Erstellung von A_{12}.

Wie können S_i mit $i \not\equiv 0(3)$ schnell sortiert werden?

S_1	ississippi$
S_2	ssissippi$
S_4	issippi$
S_5	ssippi$
S_7	ippi$
S_8	ppi$
S_{10}	i$$
S_{11}	$$$

Sortiere Tripel →

0	$$$
1	i$$
2	ipp
3	iss
4	ppi
5	ssi

Erstelle neuen Text T' in zwei Schritten:

1. Schritt: Konkatenation der Tripel

$T' =$ [iss][iss][ipp][i$$] [ssi][ssi][ppi][$$$]

S_1 S_4 S_7 S_{10} S_2 S_5 S_8 S_{11}

2. Schritt: Ersetze Tripel durch Eimer-Nummern.

$T' =$ 3 3 2 1 5 5 4 0 $

S_1 S_4 S_7 S_{10} S_2 S_5 S_8 S_{11}

Erstellung von A_{12}.

Wie können S_i mit $i \not\equiv 0(3)$ schnell sortiert werden?

S_1	ississippi$
S_2	ssissippi$
S_4	issippi$
S_5	ssippi$
S_7	ippi$
S_8	ppi$
S_{10}	i$$
S_{11}	$$$

Sortiere Tripel →

0	$$$
1	i$$
2	ipp
3	iss
4	ppi
5	ssi

Erstelle neuen Text T' in zwei Schritten:

1. Schritt: Konkatenation der Tripel

$T' =$ [iss][iss][ipp][i$$] [ssi][ssi][ppi][$$$]

S_1 S_4 S_7 S_{10} S_2 S_5 S_8 S_{11}

2. Schritt: Ersetze Tripel durch Eimer-Nummern.

$T' =$ 3 3 2 1 5 5 4 0 $

S_1 S_4 S_7 S_{10} S_2 S_5 S_8 S_{11}

Berechne Suffix-Array A' von T' (rekursiver Aufruf von SUFFIXARRAY):

S'_8	$	–	
S'_7	0$	S_{11}	$$$
S'_3	15540$	S_{10}	i$$
S'_2	215540$	S_7	ippi$
S'_1	3215540$	S_4	issippi$
S'_0	33215540$	S_1	ississippi$
S'_6	40$	S_8	ppi$
S'_5	540$	S_5	ssippi$
S'_4	5540$	S_2	ssissippi$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Erstellung von A_{12}.

Wie können S_i mit $i \not\equiv 0(3)$ schnell sortiert werden?

S_1	ississippi$
S_2	ssissippi$
S_4	issippi$
S_5	ssippi$
S_7	ippi$
S_8	ppi$
S_{10}	i$$
S_{11}	$$$

Sortiere Tripel →

0	$$$
1	i$$
2	ipp
3	iss
4	ppi
5	ssi

Erstelle neuen Text T' in zwei Schritten:

1. Schritt: Konkatenation der Tripel

$T' =$ [iss][iss][ipp][i$$] [ssi][ssi][ppi][$$$]

S_1 S_4 S_7 S_{10} S_2 S_5 S_8 S_{11}

2. Schritt: Ersetze Tripel durch Eimer-Nummern.

$T' =$ 3 3 2 1 5 5 4 0 $

S_1 S_4 S_7 S_{10} S_2 S_5 S_8 S_{11}

Berechne Suffix-Array A' von T' (rekursiver Aufruf von SUFFIXARRAY):

S'_8	$	–	
S'_7	0$	S_{11}	$$$
S'_3	15540$	S_{10}	i$$
S'_2	215540$	S_7	ippi$
S'_1	3215540$	S_4	issippi$
S'_0	33215540$	S_1	ississippi$
S'_6	40$	S_8	ppi$
S'_5	540$	S_5	ssippi$
S'_4	5540$	S_2	ssissippi$

Suffix-Array A' gibt Sortierung von S_i mit $i \not\equiv 0(3)$ an.

Konstruktion von Suffix-Arrays

Skizze:

SUFFIXARRAY(Text $T = t_0 t_1 \ldots t_{n-1}$)

1. Berechne A_{12} von T.
2. Berechne A_0 von T.
3. Vermenge A_{12} mit A_0.

- Schritt 1 berechnet in $O(n)$ Zeit das Suffix-Array A_{12}. Hierzu wird ein Text T' aufgebaut, der $\frac{2}{3}$ so groß ist wie T und die Ordnung der Suffixe von T erhält. Rekursiver Aufruf von SUFFIXARRAY auf T' berechnet Suffix-Array von T', das dann in A_{12} transformiert wird.
- Schritt 2 berechnet in $O(n)$ aus A_{12} und T das Suffix-Array A_0.
- Schritt 3 vermengt A_{12} und A_0 zu Suffix-Array A.
- Insgesamt ergibt sich dann die Rekurenzgleichung

$$T(n) = T\left(\frac{2}{3}n\right) + O(n),$$

die sich zu $O(n)$ auflöst.

Erstellung von A_0

A_0 kann mithilfe von A_{12} konstruiert werden:

Lemma: Seien i und j so gewählt dass $i, j \equiv 0(3)$. Es gilt

$S_i < S_j$ genau dann wenn $t_i < t_j$, oder $t_i = t_j$ und S_{i+1} steht vor S_{j+1} in A_{12}.

Lemma beschreibt wie restliche Suffixe sortiert werden müssen. Kann mithilfe von Bucket-Sort gemacht werden.

Aus Schritt 1 folgt: $S_{11} < S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

S_0	mississippi$
S_1	ississippi$
S_2	ssissippi$
S_3	sissippi$
S_4	issippi$
S_5	ssippi$
S_6	sippi$
S_7	ippi$
S_8	ppi$
S_9	pi$
S_{10}	i$$
S_{11}	$$$

es müssen noch S_0, S_3, S_6 und S_9 sortiert werden, um A_0 zu erhalten.

$S_0 < S_9$, da $m < p$

$S_9 < S_6$, da $p < s$

$S_6 < S_3$, da $S_7 < S_4$

Es ergibt sich: $S_0 < S_9 < S_6 < S_3$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Konstruktion von Suffix-Arrays

Skizze:

SUFFIXARRAY(Text $T = t_0 t_1 \ldots t_{n-1}$)

1. Berechne A_{12} von T.
2. Berechne A_0 von T.
3. Vermenge A_{12} mit A_0.

- Schritt 1 berechnet in $O(n)$ Zeit das Suffix-Array A_{12}. Hierzu wird ein Text T' aufgebaut, der $\frac{2}{3}$ so groß ist wie T und die Ordnung der Suffixe von T erhält. Rekursiver Aufruf von SUFFIXARRAY auf T' berechnet Suffix-Array von T', das dann in A_{12} transformiert wird.
- Schritt 2 berechnet in $O(n)$ aus A_{12} und T das Suffix-Array A_0.
- Schritt 3 vermengt A_{12} und A_0 zu Suffix-Array A.
- Insgesamt ergibt sich dann die Rekurenzgleichung

$$T(n) = T\left(\frac{2}{3}n\right) + O(n),$$

die sich zu $O(n)$ auflöst.

Vermengen von A_0 und A_{12}

Lemma 30: Sei i so gewählt, dass $i \equiv 0(3)$.

1. Für $j \equiv 1(3)$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und S_{i+1} steht vor S_{j+1} in A_{12}.
2. Für $j \equiv 2(3)$ 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} steht vor S_{j+2} in A_{12}.

Aus Schritt 1 folgt: $S_{11} < S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt: $S_0 < S_9 < S_6 < S_3$

S_0	mississippi$
S_1	ississippi$
S_2	ssissippi$
S_3	sissippi$
S_4	issippi$
S_5	ssippi$
S_6	sippi$
S_7	ippi$
S_8	ppi$
S_9	pi$
S_{10}	i$
S_{11}	$

Vermengen von A_0 und A_{12}

Lemma 30: Sei i so gewählt, dass $i \equiv 0(3)$.

1. Für $j \equiv 1(3)$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und S_{i+1} steht vor S_{j+1} in A_{12}.
2. Für $j \equiv 2(3)$ 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} steht vor S_{j+2} in A_{12}.

Aus Schritt 1 folgt: $S_{11} < S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt: $S_0 < S_9 < S_6 < S_3$

S_0	mississippi$
S_1	ississippi$
S_2	ssissippi$
S_3	sissippi$
S_4	issippi$
S_5	ssippi$
S_6	sippi$
S_7	ippi$
S_8	ppi$
S_9	pi$
S_{10}	i$
S_{11}	$

$S_{11} < S_0$ weil $\$ < m$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Vermengen von A_0 und A_{12}

Lemma 30: Sei i so gewählt, dass $i \equiv 0(3)$.

1. Für $j \equiv 1(3)$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und S_{i+1} steht vor S_{j+1} in A_{12}.
2. Für $j \equiv 2(3)$ 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} steht vor S_{j+2} in A_{12}.

Aus Schritt 1 folgt: $S_{11} < S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt: $S_0 < S_9 < S_6 < S_3$

S_0	mississippi$
S_1	ississippi$
S_2	ssissippi$
S_3	sissippi$
S_4	issippi$
S_5	ssippi$
S_6	sippi$
S_7	ippi$
S_8	ppi$
S_9	pi$
S_{10}	i$
S_{11}	$

$S_{11} < S_0$ weil $\$ < m$

$S_{10} < S_0$ weil $i < m$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Vermengen von A_0 und A_{12}

Lemma 30: Sei i so gewählt, dass $i \equiv 0(3)$.

1. Für $j \equiv 1(3)$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und S_{i+1} steht vor S_{j+1} in A_{12}.
2. Für $j \equiv 2(3)$ 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} steht vor S_{j+2} in A_{12}.

Aus Schritt 1 folgt: $S_{11} < S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt: $S_0 < S_9 < S_6 < S_3$

S_0	mississippi$
S_1	ississippi$
S_2	ssissippi$
S_3	sissippi$
S_4	issippi$
S_5	ssippi$
S_6	sippi$
S_7	ippi$
S_8	ppi$
S_9	pi$
S_{10}	i$
S_{11}	$

$S_{11} < S_0$ weil \$ $< m$

$S_{10} < S_0$ weil $i < m$

$S_7 < S_0$ weil $i < m$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Vermengen von A_0 und A_{12}

Lemma 30: Sei i so gewählt, dass $i \equiv 0(3)$.

1. Für $j \equiv 1(3)$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und S_{i+1} steht vor S_{j+1} in A_{12}.
2. Für $j \equiv 2(3)$ 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} steht vor S_{j+2} in A_{12}.

Aus Schritt 1 folgt: $S_{11} < S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt: $S_0 < S_9 < S_6 < S_3$

S_0	mississippi$
S_1	ississippi$
S_2	ssissippi$
S_3	sissippi$
S_4	issippi$
S_5	ssippi$
S_6	sippi$
S_7	ippi$
S_8	ppi$
S_9	pi$
S_{10}	i$
S_{11}	$

$S_{11} < S_0$	weil \$ $< m$
$S_{10} < S_0$	weil $i < m$
$S_7 < S_0$	weil $i < m$
$S_4 < S_0$	weil $i < m$

Vermengen von A_0 und A_{12}

Lemma 30: Sei i so gewählt, dass $i \equiv 0(3)$.

1. Für $j \equiv 1(3)$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und S_{i+1} steht vor S_{j+1} in A_{12}.
2. Für $j \equiv 2(3)$ 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} steht vor S_{j+2} in A_{12}.

Aus Schritt 1 folgt: $S_{11} < S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt: $S_0 < S_9 < S_6 < S_3$

S_0	mississippi$
S_1	ississippi$
S_2	ssissippi$
S_3	sissippi$
S_4	issippi$
S_5	ssippi$
S_6	sippi$
S_7	ippi$
S_8	ppi$
S_9	pi$
S_{10}	i$
S_{11}	$

$S_{11} < S_0$	weil \$ $< m$
$S_{10} < S_0$	weil $i < m$
$S_7 < S_0$	weil $i < m$
$S_4 < S_0$	weil $i < m$
$S_1 < S_0$	weil $i < m$

Vermengen von A_0 und A_{12}

Lemma 30: Sei i so gewählt, dass $i \equiv 0(3)$.

1. Für $j \equiv 1(3)$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und S_{i+1} steht vor S_{j+1} in A_{12}.
2. Für $j \equiv 2(3)$ 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} steht vor S_{j+2} in A_{12}.

Aus Schritt 1 folgt: $S_{11} < S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt: $S_0 < S_9 < S_6 < S_3$

S_0	mississippi$
S_1	ississippi$
S_2	ssissippi$
S_3	sissippi$
S_4	issippi$
S_5	ssippi$
S_6	sippi$
S_7	ippi$
S_8	ppi$
S_9	pi$
S_{10}	i$
S_{11}	$

$S_{11} < S_0$	weil $\$ < m$
$S_{10} < S_0$	weil $i < m$
$S_7 < S_0$	weil $i < m$
$S_4 < S_0$	weil $i < m$
$S_1 < S_0$	weil $i < m$
$S_0 < S_8$	weil $m < p$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Vermengen von A_0 und A_{12}

Lemma 30: Sei i so gewählt, dass $i \equiv 0(3)$.

1. Für $j \equiv 1(3)$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und S_{i+1} steht vor S_{j+1} in A_{12}.
2. Für $j \equiv 2(3)$ 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} steht vor S_{j+2} in A_{12}.

Aus Schritt 1 folgt: $S_{11} < S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt: $S_0 < S_9 < S_6 < S_3$

S_0	mississippi$
S_1	ississippi$
S_2	ssissippi$
S_3	sissippi$
S_4	issippi$
S_5	ssippi$
S_6	sippi$
S_7	ippi$
S_8	ppi$
S_9	pi$
S_{10}	i$
S_{11}	$

$S_{11} < S_0$	weil $\$ < m$
$S_{10} < S_0$	weil $i < m$
$S_7 < S_0$	weil $i < m$
$S_4 < S_0$	weil $i < m$
$S_1 < S_0$	weil $i < m$
$S_0 < S_8$	weil $m < p$
$S_9 < S_8$	weil $i < p$

Vermengen von A_0 und A_{12}

Lemma 30: Sei i so gewählt, dass $i \equiv 0(3)$.

1. Für $j \equiv 1(3)$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und S_{i+1} steht vor S_{j+1} in A_{12}.
2. Für $j \equiv 2(3)$ 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} steht vor S_{j+2} in A_{12}.

Aus Schritt 1 folgt: $S_{11} < S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt: $S_0 < S_9 < S_6 < S_3$

S_0	mississippi$
S_1	ississippi$
S_2	ssissippi$
S_3	sissippi$
S_4	issippi$
S_5	ssippi$
S_6	sippi$
S_7	ippi$
S_8	ppi$
S_9	pi$
S_{10}	i$
S_{11}	$

$S_{11} < S_0$	weil $\$ < m$
$S_{10} < S_0$	weil $i < m$
$S_7 < S_0$	weil $i < m$
$S_4 < S_0$	weil $i < m$
$S_1 < S_0$	weil $i < m$
$S_0 < S_8$	weil $m < p$
$S_9 < S_8$	weil $i < p$
$S_8 < S_6$	weil $p < s$

Vermengen von A_0 und A_{12}

Lemma 30: Sei i so gewählt, dass $i \equiv 0(3)$.

1. Für $j \equiv 1(3)$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und S_{i+1} steht vor S_{j+1} in A_{12}.
2. Für $j \equiv 2(3)$ 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} steht vor S_{j+2} in A_{12}.

Aus Schritt 1 folgt: $S_{11} < S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt: $S_0 < S_9 < S_6 < S_3$

S_0	mississippi$
S_1	ississippi$
S_2	ssissippi$
S_3	sissippi$
S_4	issippi$
S_5	ssippi$
S_6	sippi$
S_7	ippi$
S_8	ppi$
S_9	pi$
S_{10}	i$
S_{11}	$

$S_{11} < S_0$	weil $\$ < m$
$S_{10} < S_0$	weil $i < m$
$S_7 < S_0$	weil $i < m$
$S_4 < S_0$	weil $i < m$
$S_1 < S_0$	weil $i < m$
$S_0 < S_8$	weil $m < p$
$S_9 < S_8$	weil $i < p$
$S_8 < S_6$	weil $p < s$
$S_6 < S_5$	weil $i < s$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Vermengen von A_0 und A_{12}

Lemma 30: Sei i so gewählt, dass $i \equiv 0(3)$.

1. Für $j \equiv 1(3)$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und S_{i+1} steht vor S_{j+1} in A_{12}.
2. Für $j \equiv 2(3)$ 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} steht vor S_{j+2} in A_{12}.

Aus Schritt 1 folgt: $S_{11} < S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt: $S_0 < S_9 < S_6 < S_3$

S_0	mississippi$
S_1	ississippi$
S_2	ssissippi$
S_3	sissippi$
S_4	issippi$
S_5	ssippi$
S_6	sippi$
S_7	ippi$
S_8	ppi$
S_9	pi$
S_{10}	i$
S_{11}	$

$S_{11} < S_0$	weil $\$ < m$
$S_{10} < S_0$	weil $i < m$
$S_7 < S_0$	weil $i < m$
$S_4 < S_0$	weil $i < m$
$S_1 < S_0$	weil $i < m$
$S_0 < S_8$	weil $m < p$
$S_9 < S_8$	weil $i < p$
$S_8 < S_6$	weil $p < s$
$S_6 < S_5$	weil $i < s$
$S_3 < S_5$	weil $i < s$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Vermengen von A_0 und A_{12}

Lemma 30: Sei i so gewählt, dass $i \equiv 0(3)$.

1. Für $j \equiv 1(3)$ gilt: $S_i < S_j$ genau dann, wenn $t_i < t_j$, oder $t_i = t_j$ und S_{i+1} steht vor S_{j+1} in A_{12}.
2. Für $j \equiv 2(3)$ 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} steht vor S_{j+2} in A_{12}.

Aus Schritt 1 folgt: $S_{11} < S_{10} < S_7 < S_4 < S_1 < S_8 < S_5 < S_2$

Aus Schritt 2 folgt: $S_0 < S_9 < S_6 < S_3$

S_0	mississippi$
S_1	ississippi$
S_2	ssissippi$
S_3	sissippi$
S_4	issippi$
S_5	ssippi$
S_6	sippi$
S_7	ippi$
S_8	ppi$
S_9	pi$
S_{10}	i$
S_{11}	$

$S_{11} < S_0$	weil $\$ < m$
$S_{10} < S_0$	weil $i < m$
$S_7 < S_0$	weil $i < m$
$S_4 < S_0$	weil $i < m$
$S_1 < S_0$	weil $i < m$
$S_0 < S_8$	weil $m < p$
$S_9 < S_8$	weil $i < p$
$S_8 < S_6$	weil $p < s$
$S_6 < S_5$	weil $i < s$
$S_3 < S_5$	weil $i < s$
$S_5 < S_2$	
S_2	

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

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
$31 = 2 \cdot 11 +$	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

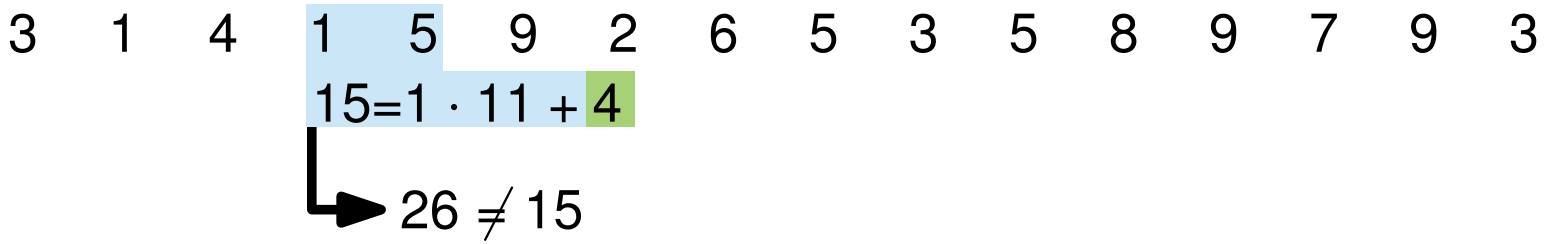
$41 = 3 \cdot 11 + 8$

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.

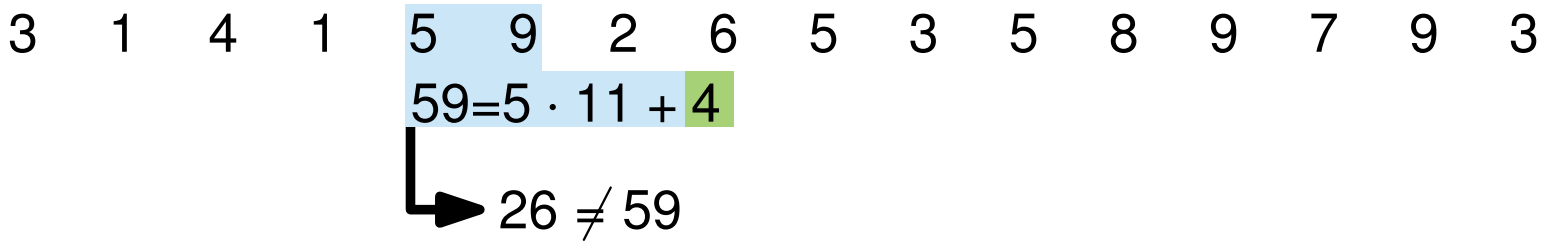
$p = 26 = 2 \cdot 11 + 4$



Problem 1

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

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

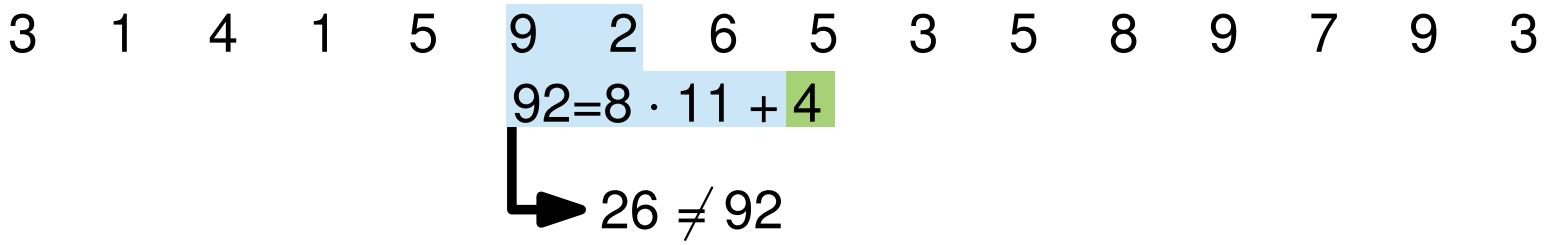


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

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

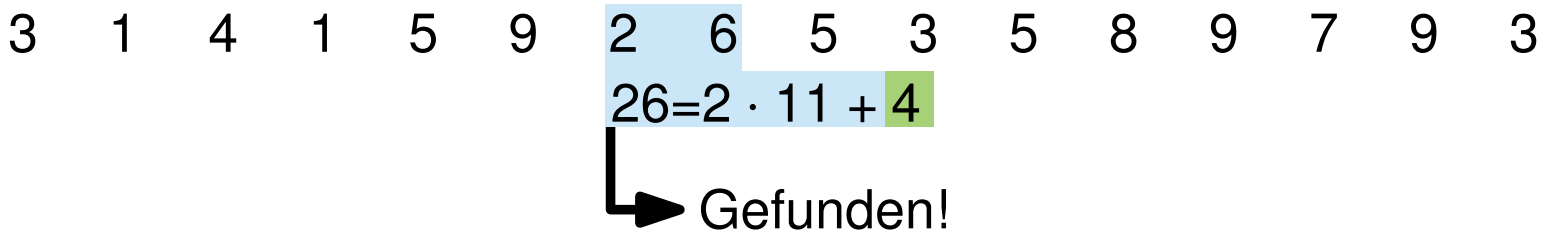
$p = 26 = 2 \cdot 11 + 4$



Problem 1

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

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



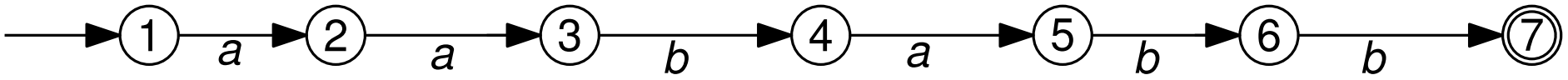
Problem 1

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

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

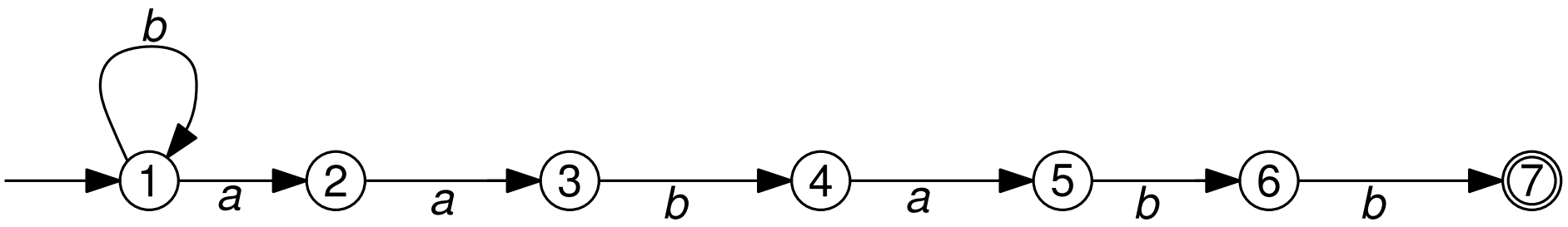
Problem 1

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



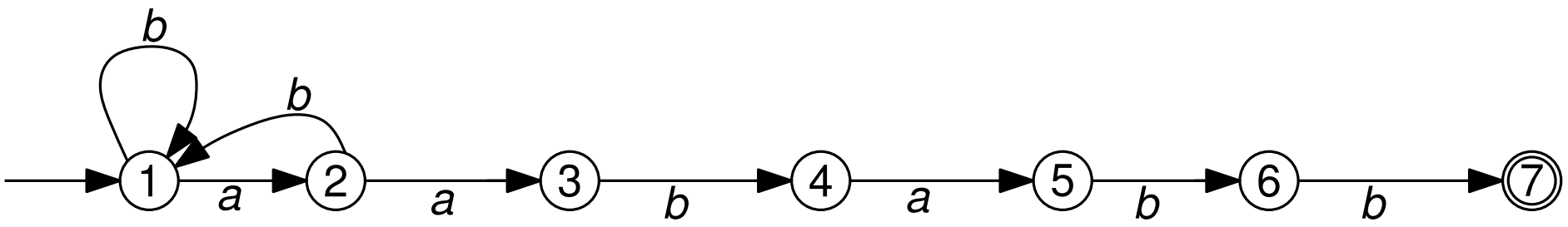
Problem 1

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



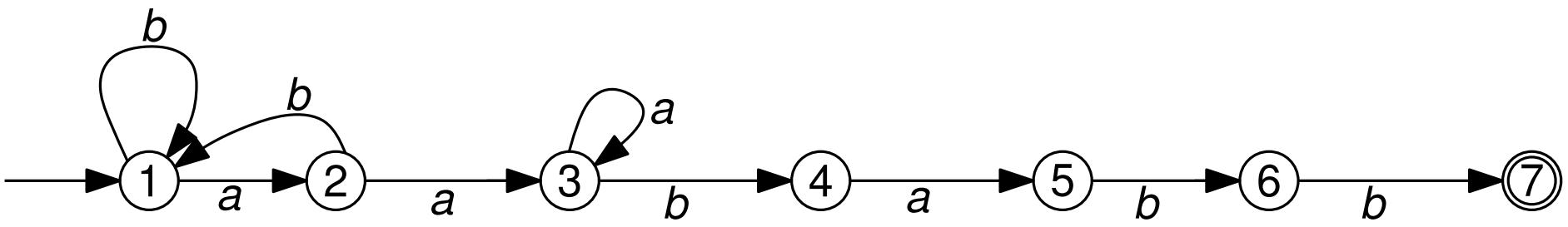
Problem 1

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



Problem 1

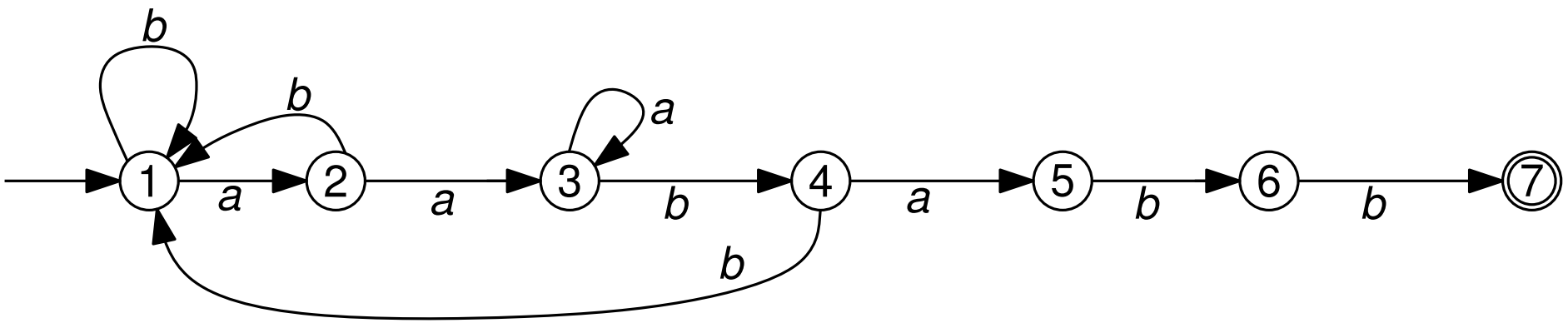
Konstruieren Sie einen String-Matching-Automat für $P = aababb$.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

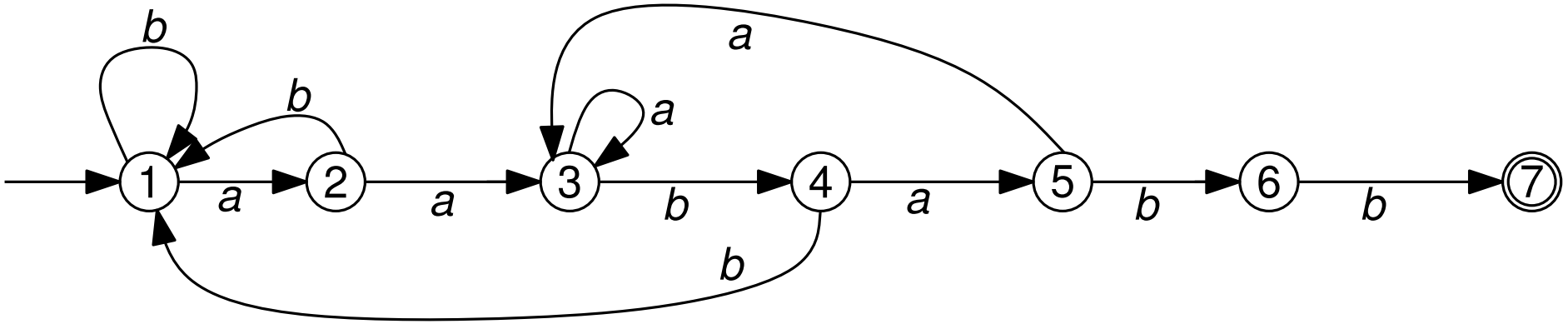
Problem 1

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



Problem 1

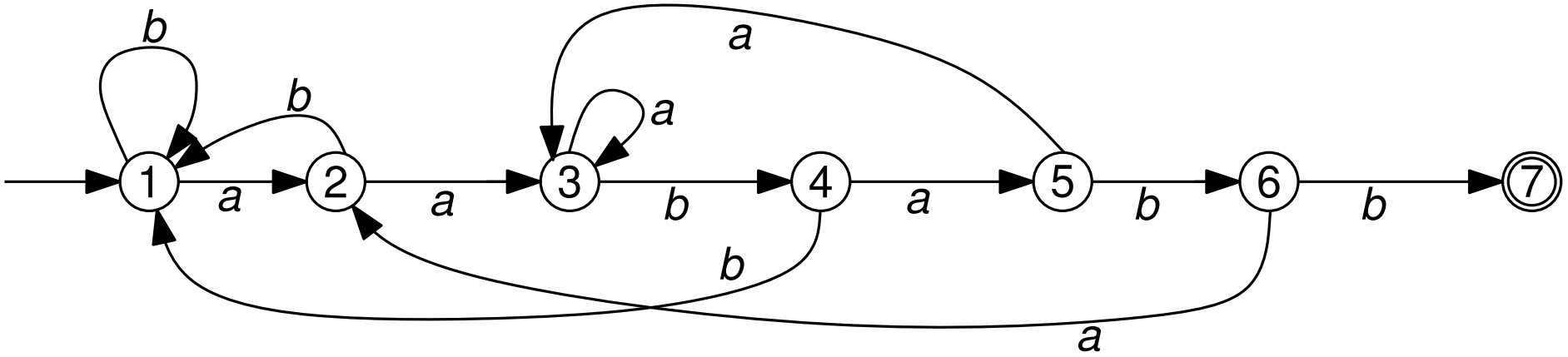
Konstruieren Sie einen String-Matching-Automat für $P = aababb$.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Problem 1

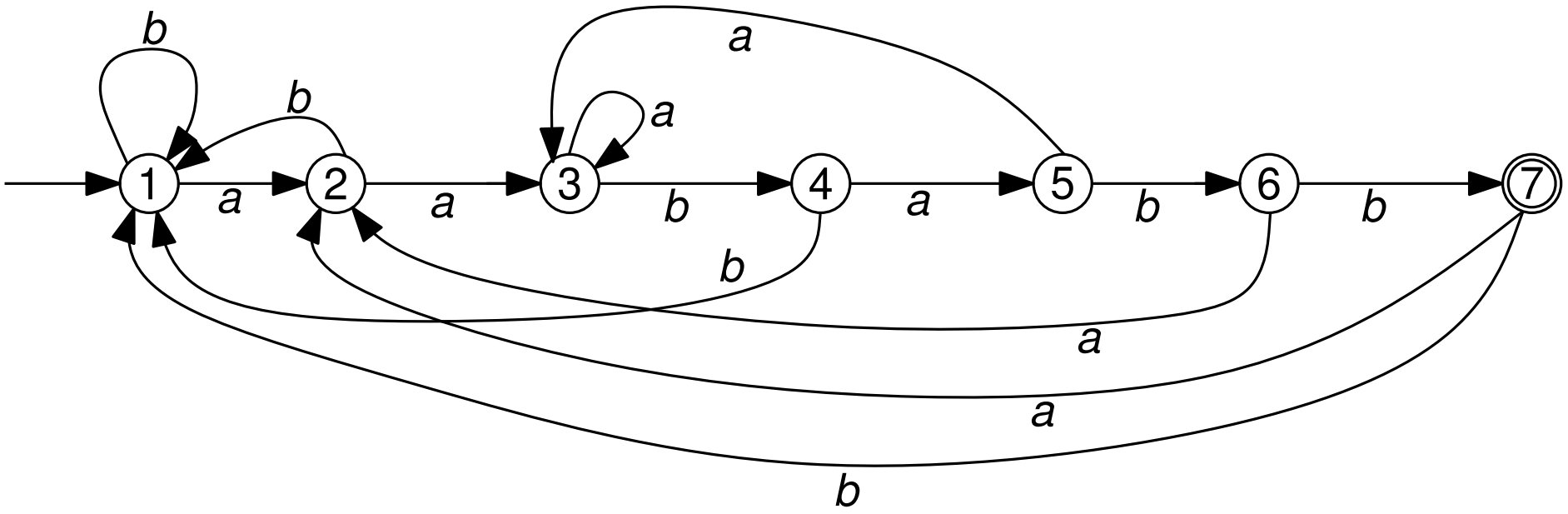
Konstruieren Sie einen String-Matching-Automat für $P = aababb$.



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