Karlsruher Institut für Technologie

Algorithmen II
Vorlesung am 13.12.2012

INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER

KIT – Universität des Landes Baden-Württemberg und
nationales Forschungszentrum in der Helmholtz-Gemeinschaft

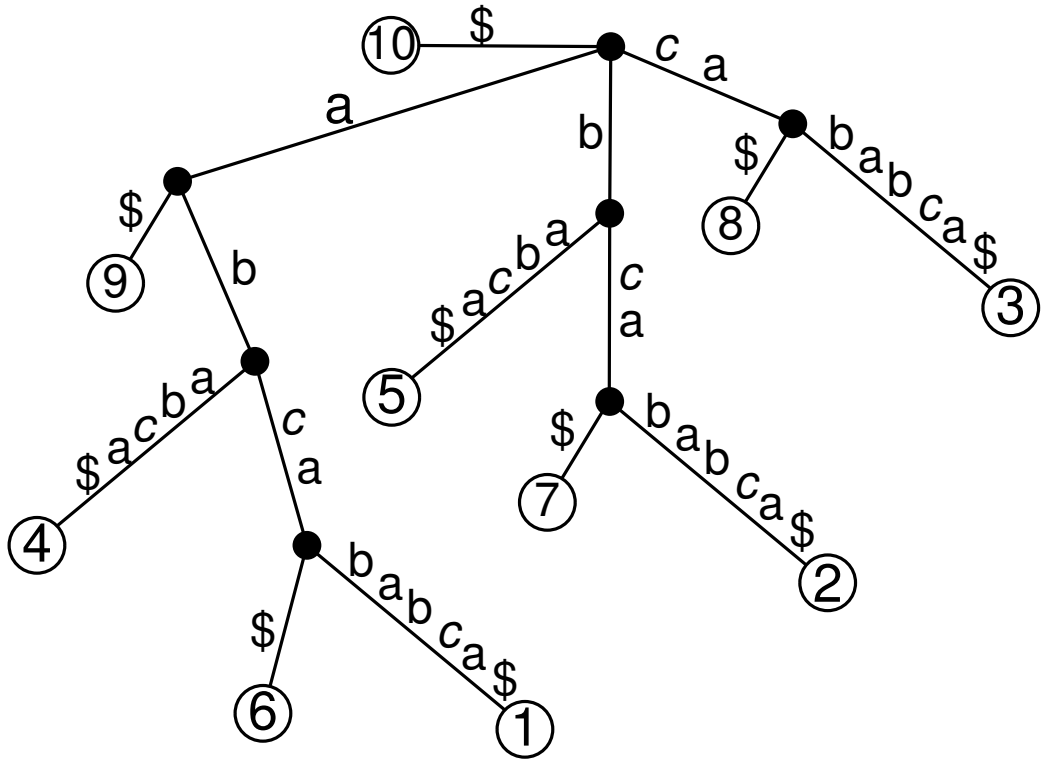
www.kit.edu

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.



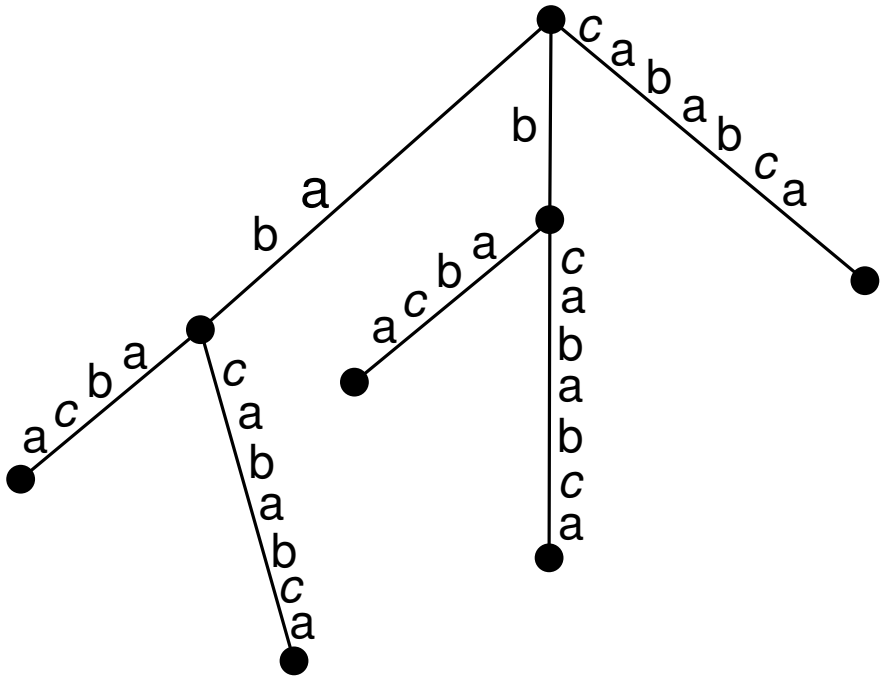
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

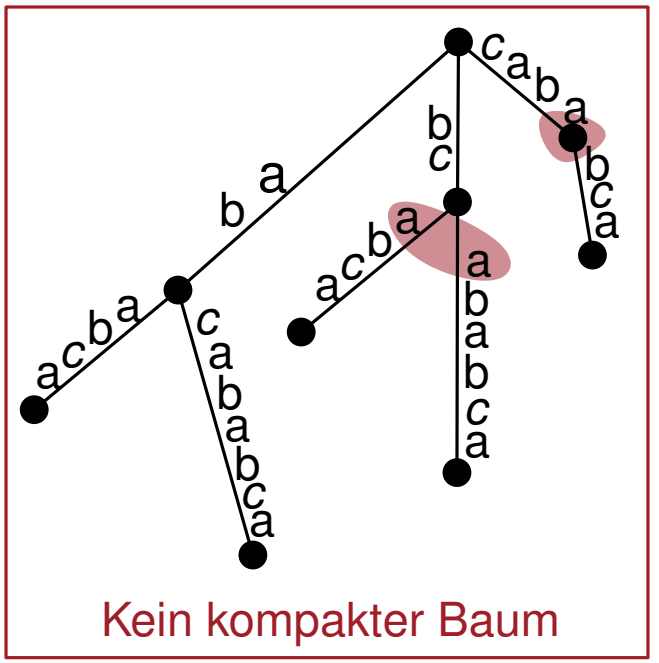
Kompakte Bäume

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





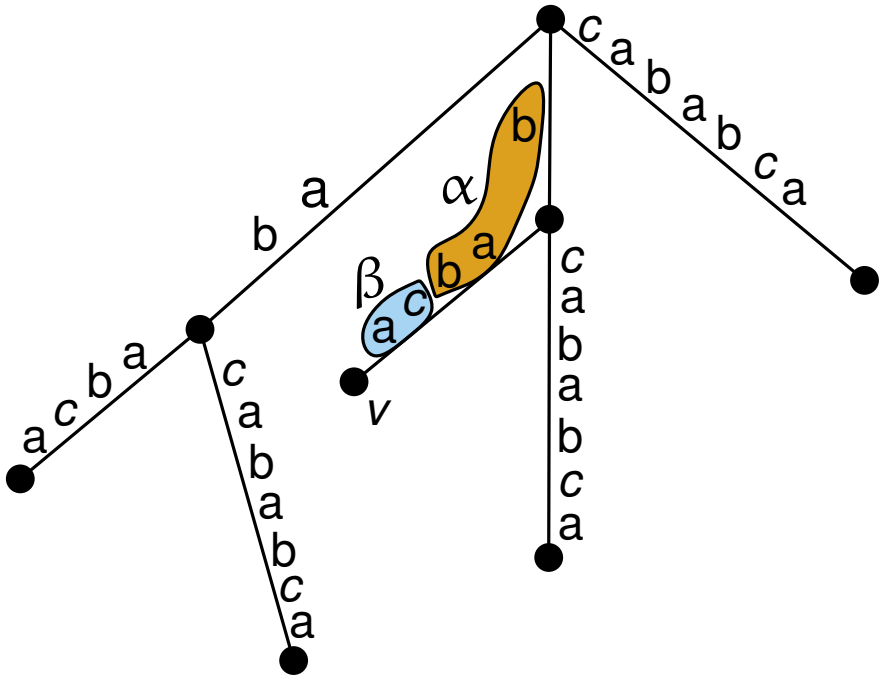
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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

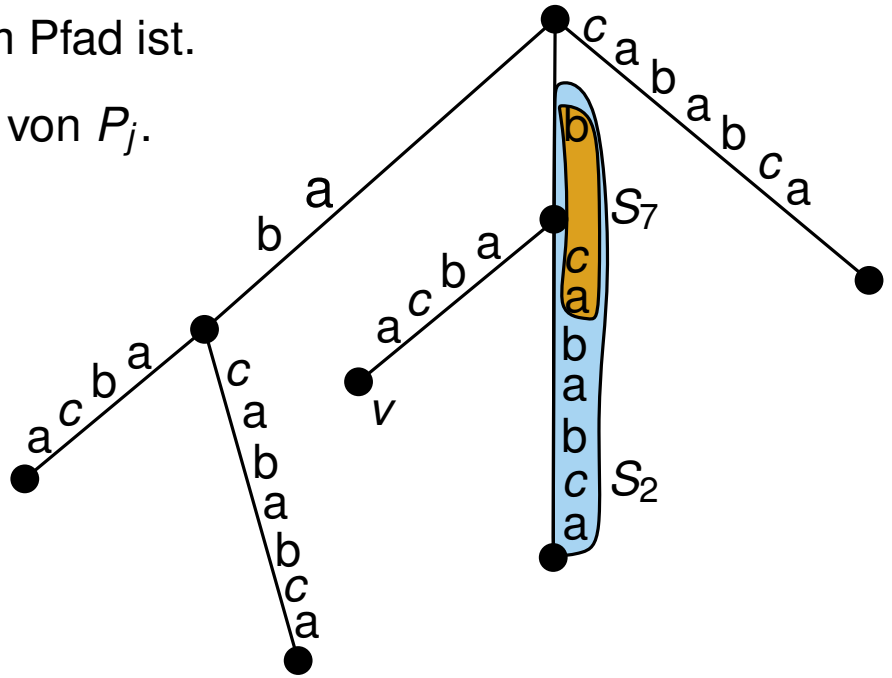
Definition: Ein *Suffixbaum* ist ein kompakter $\Sigma^\star$-Baum S, sodass S genau die Infixe von T enthält:

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

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



Später: Es gibt für jeden Text T einen Suffixbaum.

Notation:

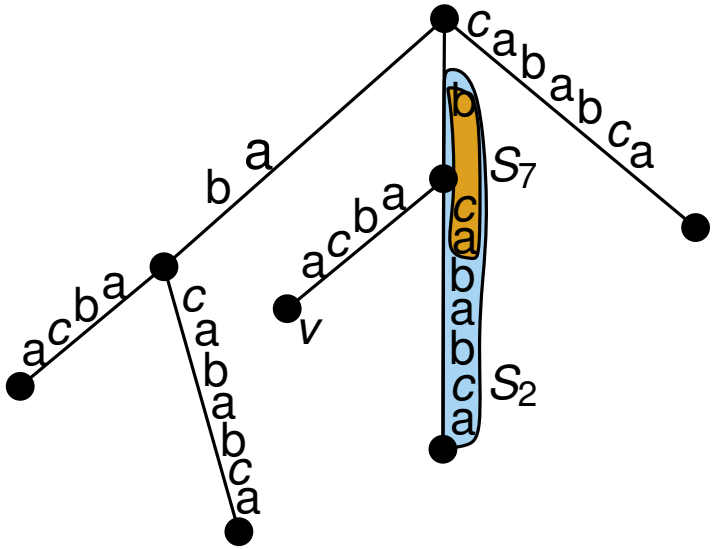
1. S_i bezeichnet das Suffix von T beginnend ab der i-ten Stelle.
2. $T[i,j]$ bezeichnet den Teilstring von T von der i-ten bis zur j-ten Stelle.

Suffixbaum

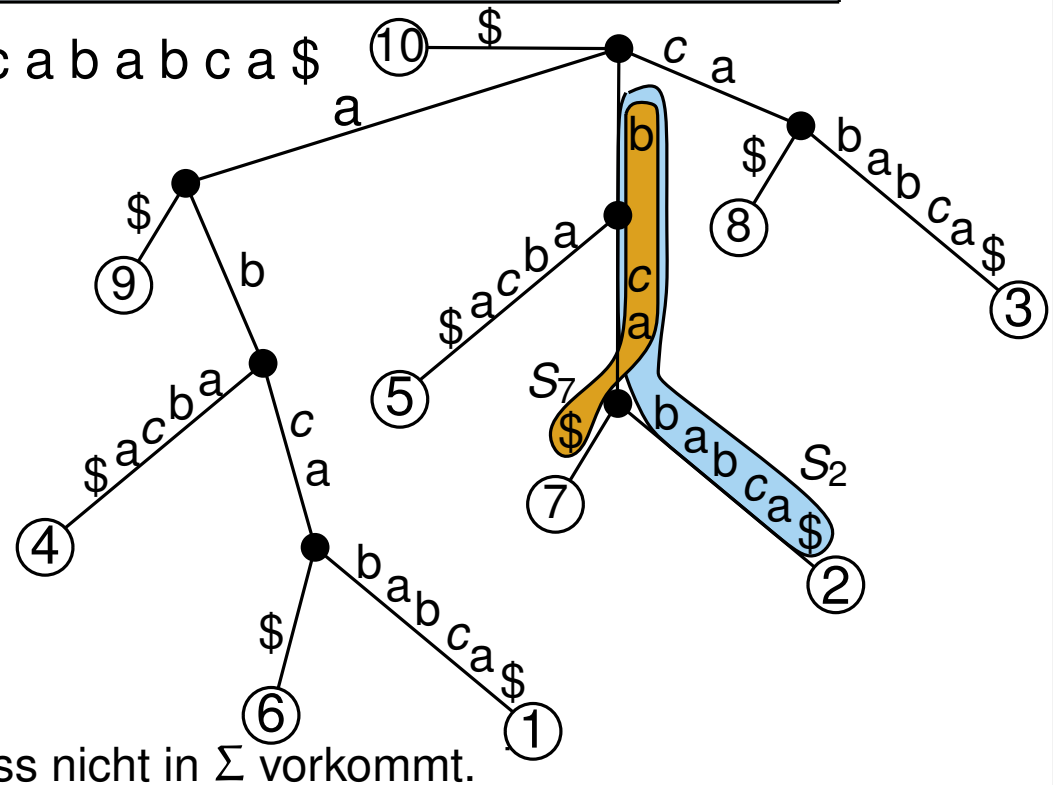
Definition: Ein *Suffixbaum* ist ein kompakter $\Sigma^\star$-Baum S, sodass S genau die Infixe von T enthält:

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

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 \$, dass 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.

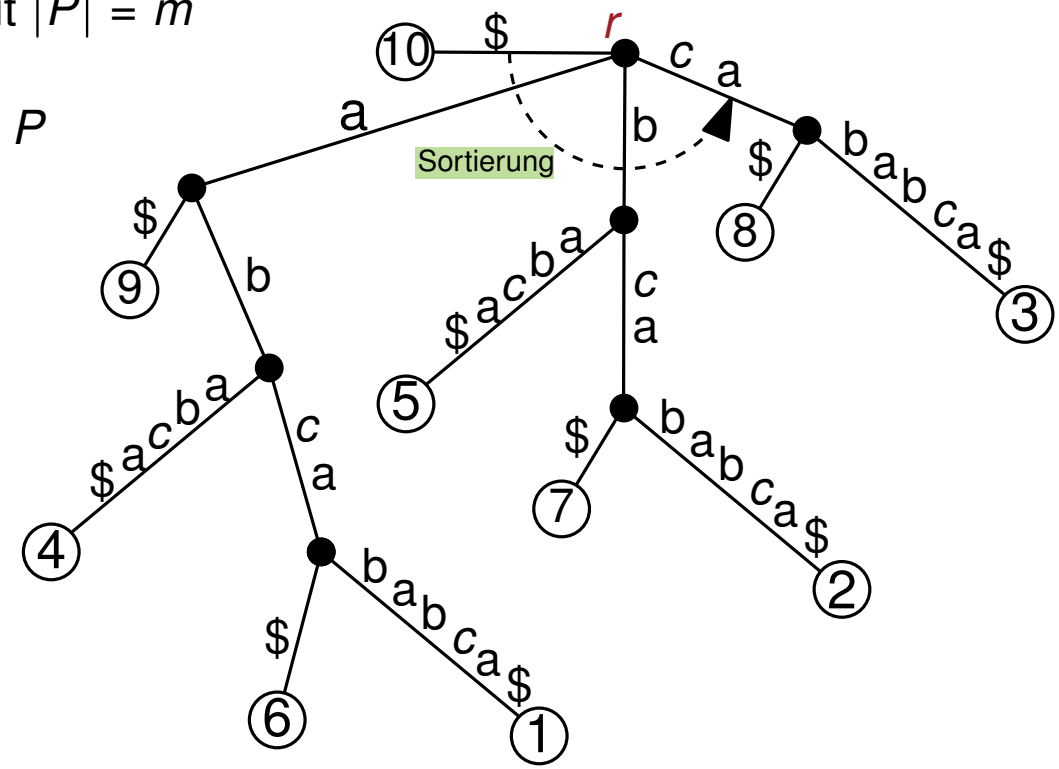
Suchen im Suffixbaum

T=a b c a b a b c a

Gegeben: Suffixbaum S und Muster P mit $|P| = m$
Gesucht: Anzahl Vorkommen von P
Idee: Suche in S alle Pfade S_i, so dass P Präfix von S_i ist.

Ausgehend von Wurzel r:

1. Wähle nächste ausgehende Kante mithilfe von binärer Suche.
2. Vergleiche P und Kantenbeschriftungen schrittweise.



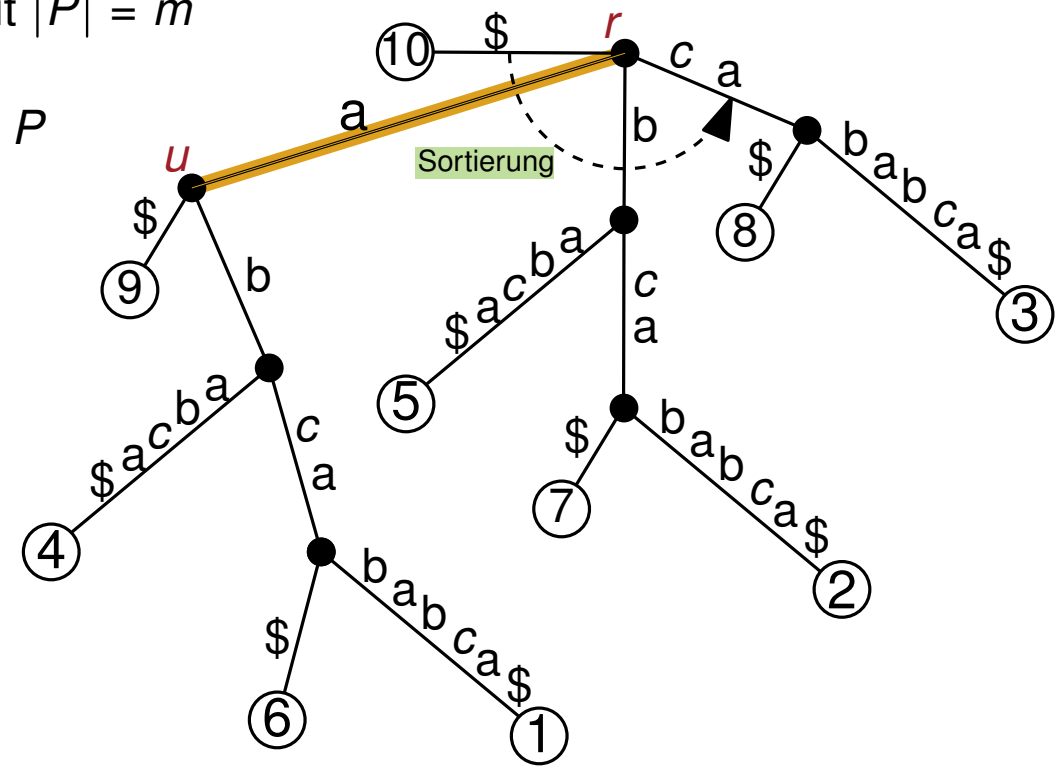
Beispiel: P= a b c a

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suchen im Suffixbaum

T=a b c a b a b c a

Gegeben: Suffixbaum S und Muster P mit $|P| = m$
Gesucht: Anzahl Vorkommen von P
Idee: Suche in S alle Pfade S_i, so dass P Präfix von S_i ist.



Ausgehend von Wurzel r:

1. Wähle nächste ausgehende Kante mithilfe von binärer Suche.
2. Vergleiche P und Kantenbeschriftungen schrittweise.

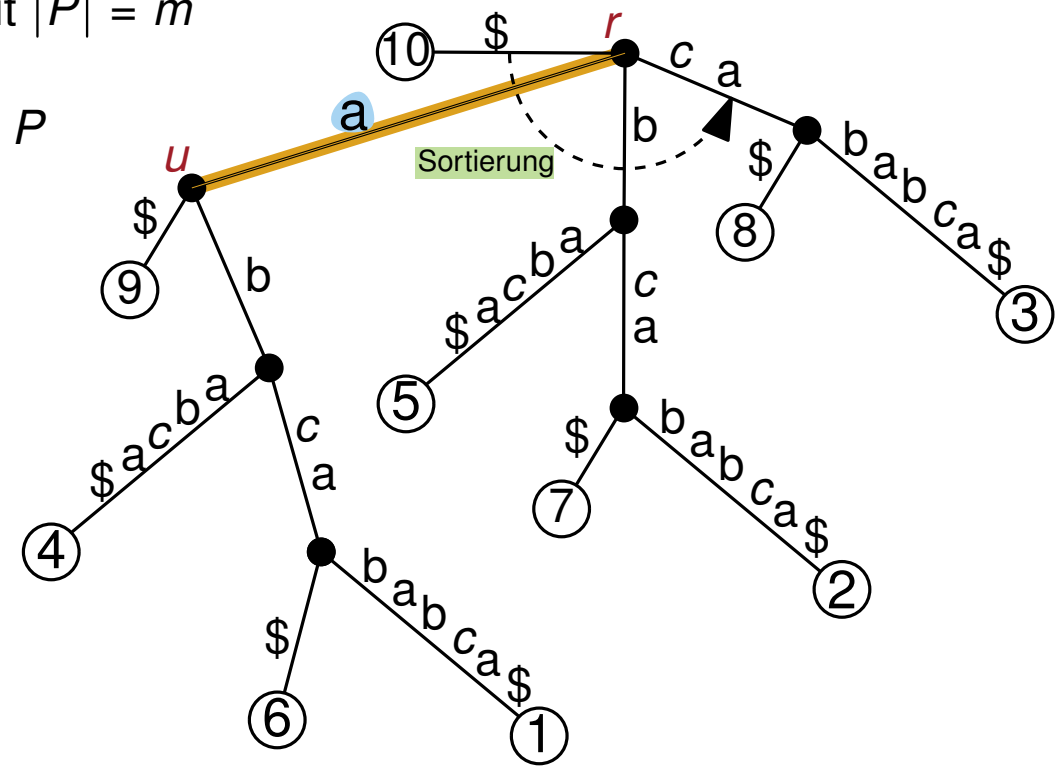
Beispiel: P= a b c a

1. Schritt: Wähle Kante (r, u) ausgehend von r, deren Beschriftung B mit $P[1] = a$ beginnt.

Suchen im Suffixbaum

T=a b c a b a b c a

Gegeben: Suffixbaum S und Muster P mit $|P| = m$
Gesucht: Anzahl Vorkommen von P
Idee: Suche in S alle Pfade S_i, so dass P Präfix von S_i ist.



Ausgehend von Wurzel r:

1. Wähle nächste ausgehende Kante mithilfe von binärer Suche.
2. Vergleiche P und Kantenbeschriftungen schrittweise.

Beispiel: P= a b c a

1. Schritt: Wähle Kante (r, u) ausgehend von r, deren Beschriftung B mit $P[1] = a$ beginnt.

2. Schritt: Vergleiche $P[1, 4]$ und B schrittweise.

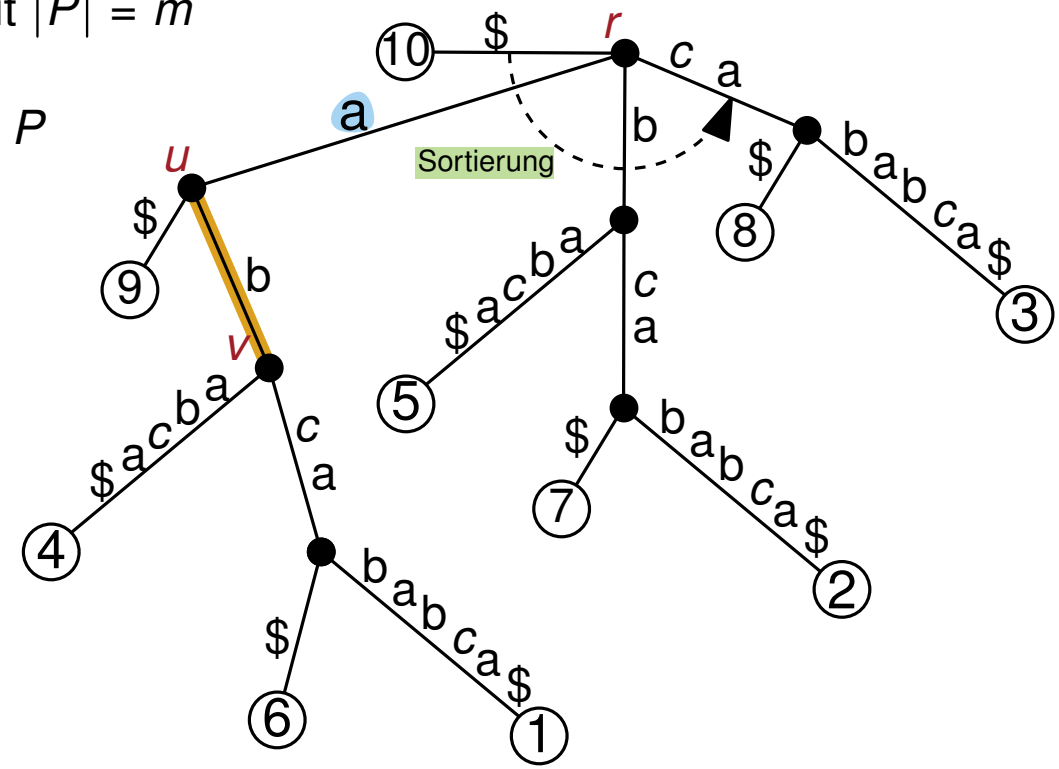
- Wenn Vergleich negativ, dann gebe 0 zurück.
- Wenn $P[1, 4]$ Präfix von B, gebe Anzahl Blätter im Unterbaum S_u aus.
- Wenn $B = P[1, j]$ für ein j mit $1 \leq j \leq m$, betrachte Knoten u und $P[j + 1, m]$.
  - Im Beispiel: Gilt für $j = 1$.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suchen im Suffixbaum

T=a b c a b a b c a

Gegeben: Suffixbaum S und Muster P mit $|P| = m$
Gesucht: Anzahl Vorkommen von P
Idee: Suche in S alle Pfade S_i, so dass P Präfix von S_i ist.



Ausgehend von Wurzel r:

1. Wähle nächste ausgehende Kante mithilfe von binärer Suche.
2. Vergleiche P und Kantenbeschriftungen schrittweise.

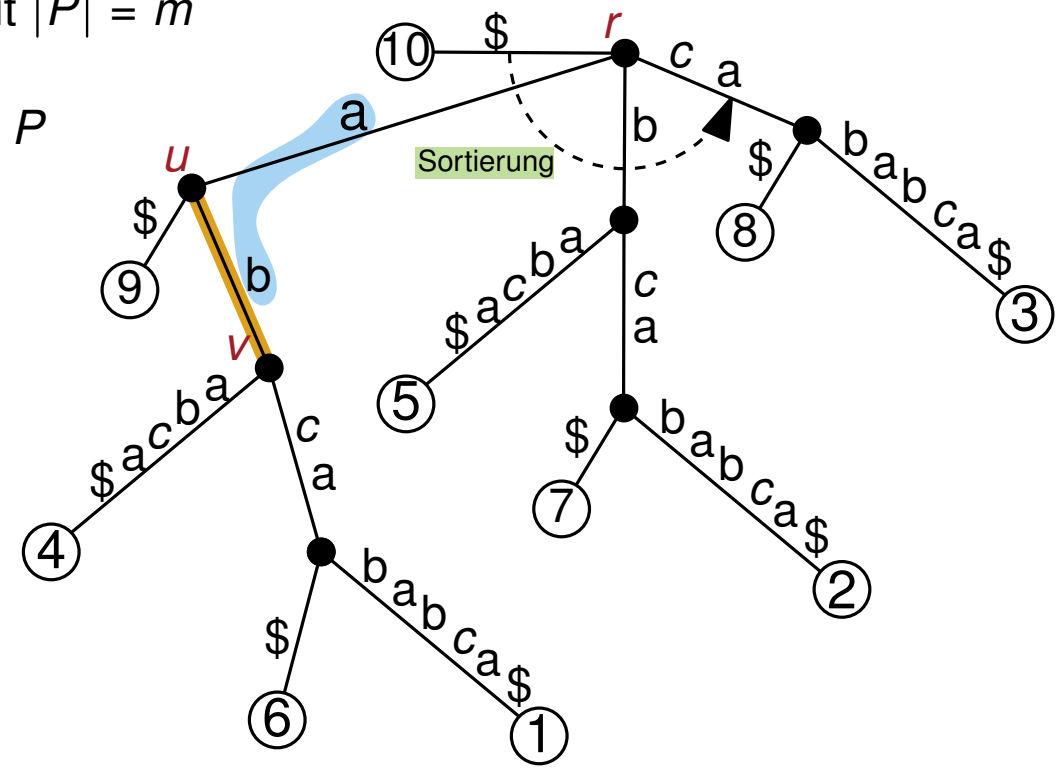
Beispiel: P= a b c a

3. Schritt: Wähle Kante (u, v) ausgehend von u, deren Beschriftung B mit $P[2] = b$ beginnt.

Suchen im Suffixbaum

T=a b c a b a b c a

Gegeben: Suffixbaum S und Muster P mit $|P| = m$
Gesucht: Anzahl Vorkommen von P
Idee: Suche in S alle Pfade S_i, so dass P Präfix von S_i ist.



Ausgehend von Wurzel r:

1. Wähle nächste ausgehende Kante mithilfe von binärer Suche.
2. Vergleiche P und Kantenbeschriftungen schrittweise.

Beispiel: P= a b c a

3. Schritt: Wähle Kante (u, v) ausgehend von u, deren Beschriftung B mit $P[2] = b$ beginnt.

4. Schritt: Vergleiche $P[2, 4]$ und B schrittweise.

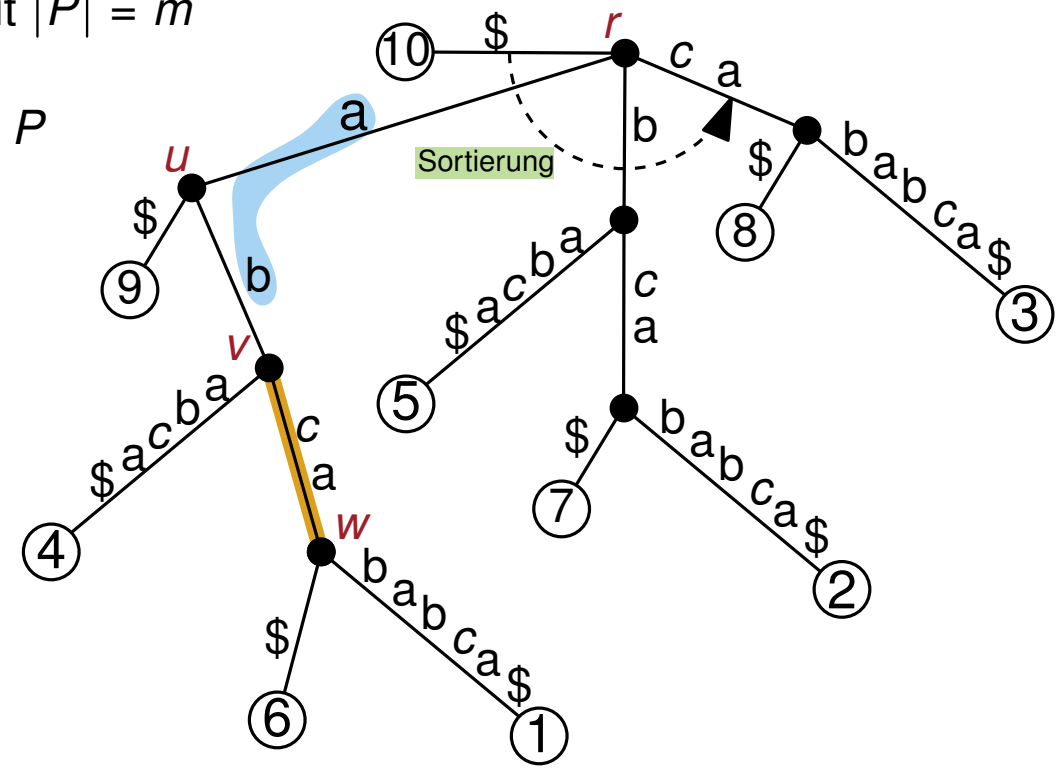
- Wenn Vergleich negativ, dann gebe 0 zurück.
- Wenn $P[2, 4]$ Präfix von B, gebe Anzahl Blätter im Unterbaum S_v aus.
- Wenn $B = P[2, j]$ für ein j mit $1 \leq j \leq m$, betrachte Knoten u und $P[j + 1, m]$.
  - Im Beispiel: Gilt für $j = 2$.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suchen im Suffixbaum

T=a b c a b a b c a

Gegeben: Suffixbaum S und Muster P mit $|P| = m$
Gesucht: Anzahl Vorkommen von P
Idee: Suche in S alle Pfade S_i, so dass P Präfix von S_i ist.



Ausgehend von Wurzel r:

1. Wähle nächste ausgehende Kante mithilfe von binärer Suche.
2. Vergleiche P und Kantenbeschriftungen schrittweise.

Beispiel: P= a b c a

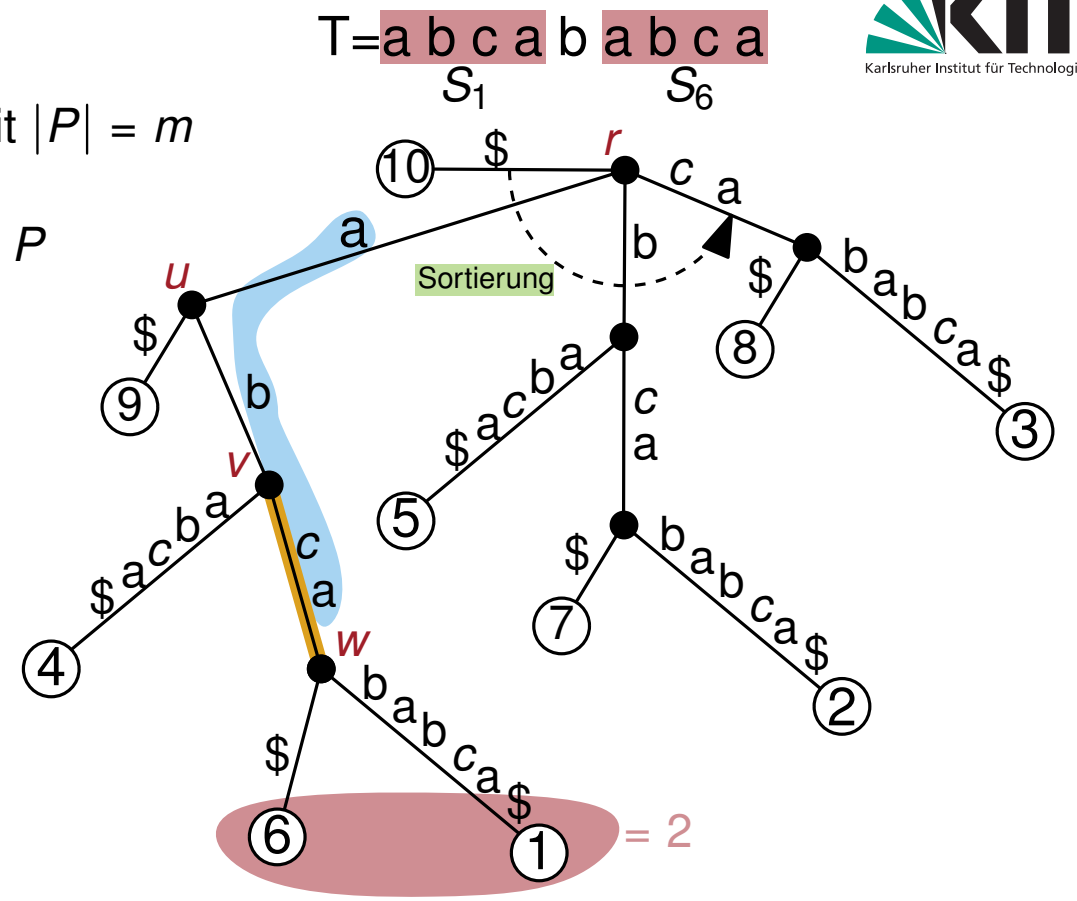
5. Schritt: Wähle Kante (v, w) ausgehend von v, deren Beschriftung B mit $P[3] = c$ beginnt.

Suchen im Suffixbaum

Gegeben: Suffixbaum S und Muster P mit $|P| = m$
Gesucht: Anzahl Vorkommen von P
Idee: Suche in S alle Pfade S_i, so dass P Präfix von S_i ist.

Ausgehend von Wurzel r:

1. Wähle nächste ausgehende Kante mithilfe von binärer Suche.
2. Vergleiche P und Kantenbeschriftungen schrittweise.



Beispiel: P= a b c a

5. Schritt: Wähle Kante (v, w) ausgehend von v, deren Beschriftung B mit $P[3] = c$ beginnt.

6. Schritt: Vergleiche $P[3, 4]$ und B schrittweise.

- Wenn Vergleich negativ, dann gebe 0 zurück.
- Wenn $P[3, 4]$ Präfix von B, gebe Anzahl Blätter im Unterbaum S_w aus.
- Wenn $B = P[3, j]$ für ein j mit $1 \leq j \leq m$, betrachte Knoten u und $P[j + 1, m]$.

Suchen im Suffixbaum

COUNT(P,S)
Ausgabe: Anzahl der Vorkommen von P im Text repräsentiert von Suffixbaum S.

$u \leftarrow$ Wurzel von S

$i \leftarrow 1$

solange *u nicht Blatt ist* **tue**
  Suche ausgehende Kante $e = (u, v)$, deren Beschriftung B mit $P[i]$ beginnt.
  wenn *e nicht existiert* **dann return** 0
  Vergleiche *B schrittweise mit* $P[i, m]$ **und analysiere Ergebnis**
    wenn $P[i, m]$ *ist Präfix von B* **dann**
      return Anzahl Blätter im Teilbaum S_v
    sonst wenn $P[i, j] = B$ *für ein* $j < m$ **dann**
      $i \leftarrow j + 1$
      $u \leftarrow v$
    sonst
      return 0

return 0

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suchen im Suffixbaum

COUNT(P,S)
Ausgabe: Anzahl der Vorkommen von P im Text repräsentiert von Suffixbaum S.

$u \leftarrow$ Wurzel von S

$i \leftarrow 1$

solange *u nicht Blatt ist* **tue**

  Suche ausgehende Kante $e = (u, v)$, deren Beschriftung B mit $P[i]$ beginnt. $O(\log |\Sigma|)$

  wenn *e nicht existiert* **dann return** 0

  Vergleiche *B schrittweise mit* $P[i, m]$ **und analysiere Ergebnis**

    wenn $P[i, m]$ *ist Präfix von B* **dann**

      return Anzahl Blätter im Teilbaum S_v

    sonst wenn $P[i, j] = B$ *für ein* $j < m$ **dann**

      $i \leftarrow j + 1$

      $u \leftarrow v$

    sonst

      return 0

return 0

Beschriftung jeder ausgehenden Kante eines Knoten beginnt unterschiedlich.
$\Rightarrow$ Sortierung eindeutig möglich.
$\Rightarrow$ Binäre Suche auf ausgehenden Kanten eines Knoten möglich.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suchen im Suffixbaum

COUNT(P,S)
Ausgabe: Anzahl der Vorkommen von P im Text repräsentiert von Suffixbaum S.

$u \leftarrow$ Wurzel von S

$i \leftarrow 1$

solange *u nicht Blatt ist* **tue**

  Suche ausgehende Kante $e = (u, v)$, deren Beschriftung B mit $P[i]$ beginnt. $O(\log |\Sigma|)$

  wenn *e nicht existiert* **dann return** 0

  Vergleiche *B schrittweise mit* $P[i, m]$ **und analysiere Ergebnis**

    wenn $P[i, m]$ *ist Präfix von B* **dann**

      return Anzahl Blätter im Teilbaum S_v $O(1)$

    sonst wenn $P[i, j] = B$ *für ein* $j < m$ **dann**

      $i \leftarrow j + 1$

      $u \leftarrow v$

    sonst

      return 0

return 0

Anzahl Blätter kann in $O(1)$ Zeit bestimmt werden, wenn an jedem Knoten Anzahl Blätter seines Teilbaums gespeichert ist.

Suchen im Suffixbaum

COUNT(P,S)
Ausgabe: Anzahl der Vorkommen von P im Text repräsentiert von Suffixbaum S.

$u \leftarrow$ Wurzel von S

$i \leftarrow 1$

solange *u nicht Blatt ist* **tue** $O(m \cdot \log |\Sigma|)$

  Suche ausgehende Kante $e = (u, v)$, deren Beschriftung B mit $P[i]$ beginnt. $O(\log |\Sigma|)$

  wenn *e nicht existiert* **dann return** 0

  Vergleiche *B schrittweise mit* $P[i, m]$ **und analysiere Ergebnis** max. $m - i$ Schritte

    wenn $P[i, m]$ *ist Präfix von B* **dann**

      return Anzahl Blätter im Teilbaum S_v $O(1)$

    sonst wenn $P[i, j] = B$ *für ein* $j < m$ **dann**

      $i \leftarrow j + 1$

      $u \leftarrow v$

    sonst

      return 0

return 0

Beim Abstieg im Baum wird jedes Zeichen von P maximal einmal betrachtet.

Suchen im Suffixbaum

Theorem 23: Gegeben ein Suffixbaum S und ein Muster P über dem Zeichenvorat Σ. Die Anzahl der Vorkommen von P kann in $O(m \cdot \log |\Sigma|)$ Zeit bestimmt werden, wenn $|P| = m$.

Verallgemeinerung:

Wenn anstatt der Anzahl die Vorkommen direkt gefragt sind, dann gebe entsprechend die Blätter aus.

Theorem 23: Gegeben ein Suffixbaum S und ein Muster P über dem Zeichenvorat Σ. Die Vorkommen von P können in $O(m \cdot \log |\Sigma| + k)$ Zeit bestimmt werden, wenn $|P| = m$ und k die Anzahl der Vorkommen von P in S bezeichnet .

Hinweis: In dem Fall ist Laufzeit *ausgabesensitiv* beschrieben.

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.

Konstruktion

Gegeben: Text T
Gesucht: Suffixbaum von T

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

Initialisierung: N_1 besteht aus zwei Knoten und einer Kante beschriftet mit S_1.

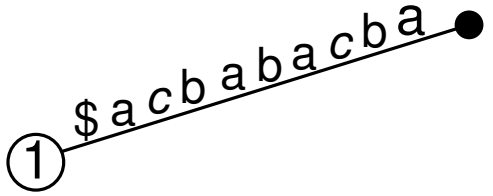
Konstruktion von $N_i + 1$ aus N_i:

Suche längsten Pfad in N_i, der Präfix von S_{i+1} matched.

1. Fall: Wenn Matching in der Mitte von e endet, dann spalte e vor erstem Mismatching in zwei Kanten auf, indem neuer Knoten w eingeführt wird.

2. Fall: Matching endet an Knoten w: Führe neue ausgehende Kante e für w ein.

T=a b c a b a b c a $
S_2



Nächster Schritt:

Füge S_2 = b c a b a b c a $ ein:

- Matching hört bereits bei Wurzel auf.
- Folglich: zweiter Fall.

Konstruktion

Gegeben: Text T
Gesucht: Suffixbaum von T

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

Initialisierung: N_1 besteht aus zwei Knoten und einer Kante beschriftet mit S_1.

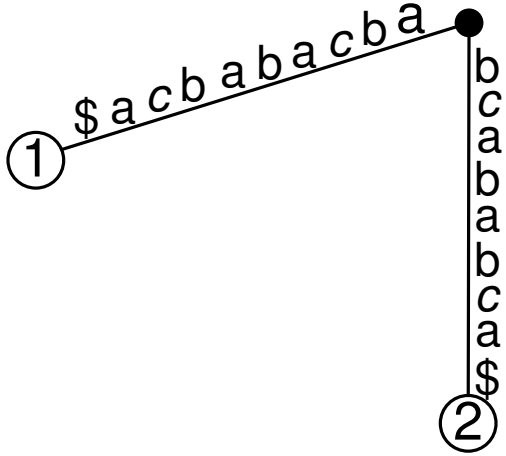
Konstruktion von $N_i + 1$ aus N_i:

Suche längsten Pfad in N_i, der Präfix von S_{i+1} matched.

1. Fall: Wenn Matching in der Mitte von e endet, dann spalte e vor erstem Mismatching in zwei Kanten auf, indem neuer Knoten w eingeführt wird.

2. Fall: Matching endet an Knoten w: Führe neue ausgehende Kante e für w ein.

T=a b c a b a b c a $
S_3



Nächster Schritt:

Füge S_3 = c a b a b c a $ ein.

- Matching hört bereits bei Wurzel auf.
- Folglich: zweiter Fall.

Konstruktion

Gegeben: Text T
Gesucht: Suffixbaum von T

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

Initialisierung: N_1 besteht aus zwei Knoten und einer Kante beschriftet mit S_1.

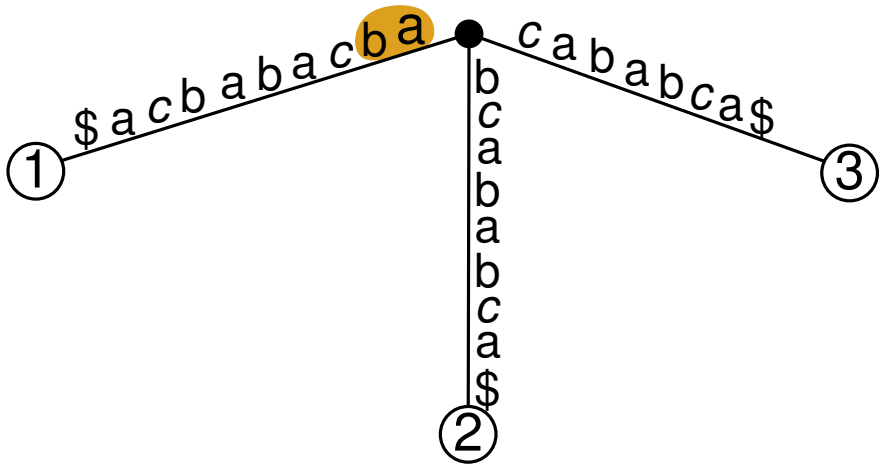
Konstruktion von $N_i + 1$ aus N_i:

Suche längsten Pfad in N_i, der Präfix von S_{i+1} matched.

1. Fall: Wenn Matching in der Mitte von e endet, dann spalte e vor erstem Mismatching in zwei Kanten auf, indem neuer Knoten w eingeführt wird.

2. Fall: Matching endet an Knoten w: Führe neue ausgehende Kante e für w ein.

T=a b c a b a b c a $
S_4



Nächster Schritt:

Füge S_4 = a b a b c a $ ein:

- Matching hört auf Pfad S_1 nach 2 Zeichen auf.
- Folglich: erster Fall.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Konstruktion

Gegeben: Text T
Gesucht: Suffixbaum von T

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

Initialisierung: N_1 besteht aus zwei Knoten und einer Kante beschriftet mit S_1.

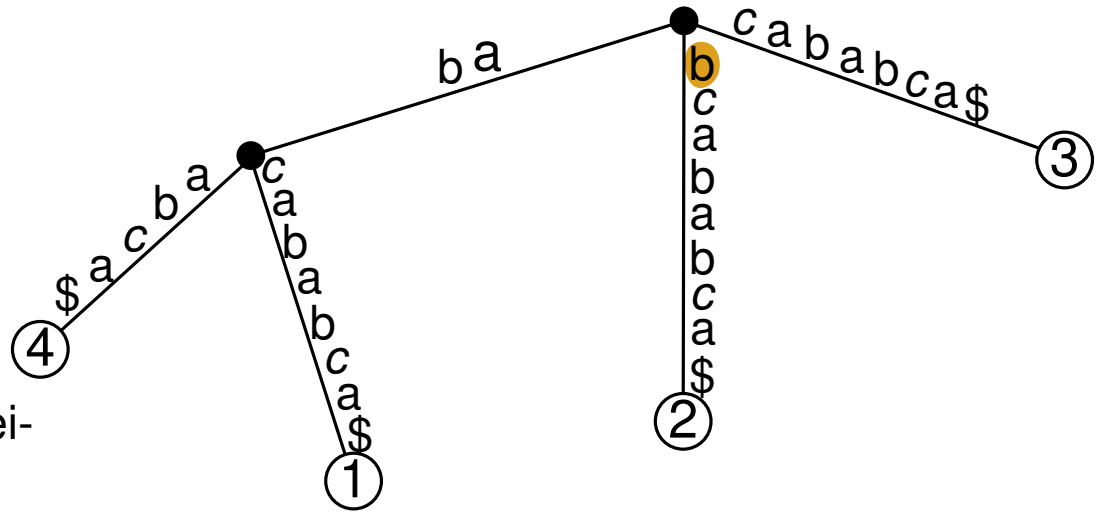
Konstruktion von $N_i + 1$ aus N_i:

Suche längsten Pfad in N_i, der Präfix von S_{i+1} matched.

1. Fall: Wenn Matching in der Mitte von e endet, dann spalte e vor erstem Mismatching in zwei Kanten auf, indem neuer Knoten w eingeführt wird.

2. Fall: Matching endet an Knoten w: Führe neue ausgehende Kante e für w ein.

T=a b c a b a b c a $
S_5



Nächster Schritt:

Füge S_5 = b a b c a $ ein:

- Matching hört auf Pfad S_2 nach einem Zeichen auf.
- Folglich: erster Fall.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Konstruktion

Gegeben: Text T
Gesucht: Suffixbaum von T

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

Initialisierung: N_1 besteht aus zwei Knoten und einer Kante beschriftet mit S_1.

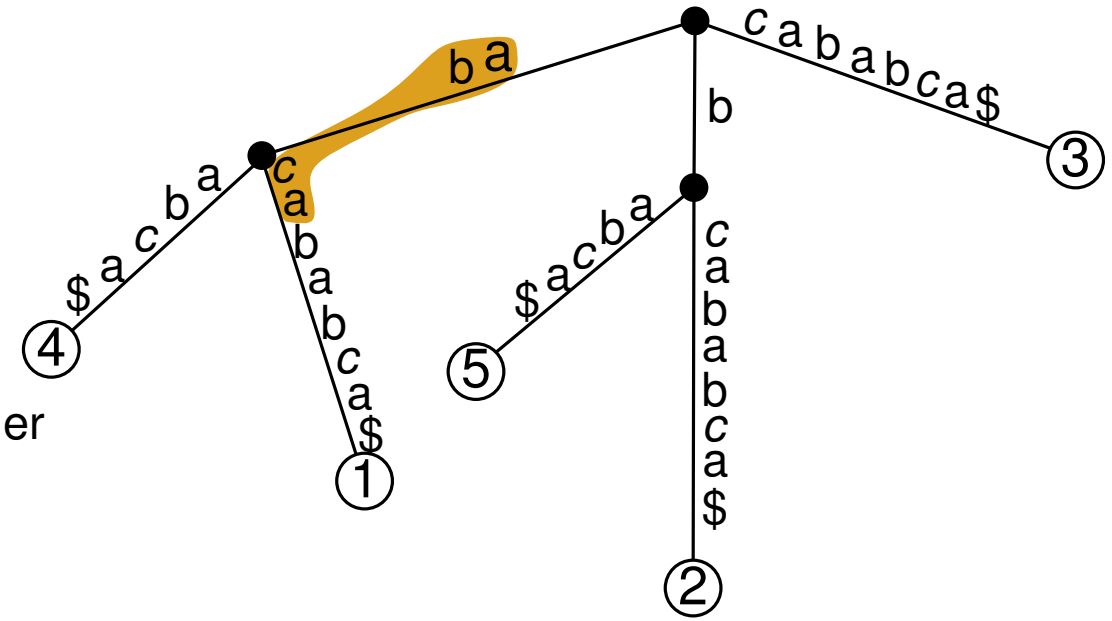
Konstruktion von $N_i + 1$ aus N_i:

Suche längsten Pfad in N_i, der Präfix von S_{i+1} matched.

1. Fall: Wenn Matching in der Mitte von e endet, dann spalte e vor erstem Mismatching in zwei Kanten auf, indem neuer Knoten w eingeführt wird.

2. Fall: Matching endet an Knoten w: Führe neue ausgehende Kante e für w ein.

T=a b c a b a b c a $
S_6



Nächster Schritt:

Füge S_6 = a b c a $ ein:

- Matching hört auf Pfad S_1 nach vier Zeichen auf.
- Folglich: erster Fall.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Konstruktion

Gegeben: Text T
Gesucht: Suffixbaum von T

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

Initialisierung: N_1 besteht aus zwei Knoten und einer Kante beschriftet mit S_1.

Konstruktion von $N_i + 1$ aus N_i:

Suche längsten Pfad in N_i, der Präfix von S_{i+1} matched.

1. Fall: Wenn Matching in der Mitte von e endet, dann spalte e vor erstem Mismatching in zwei Kanten auf, indem neuer Knoten w eingeführt wird.

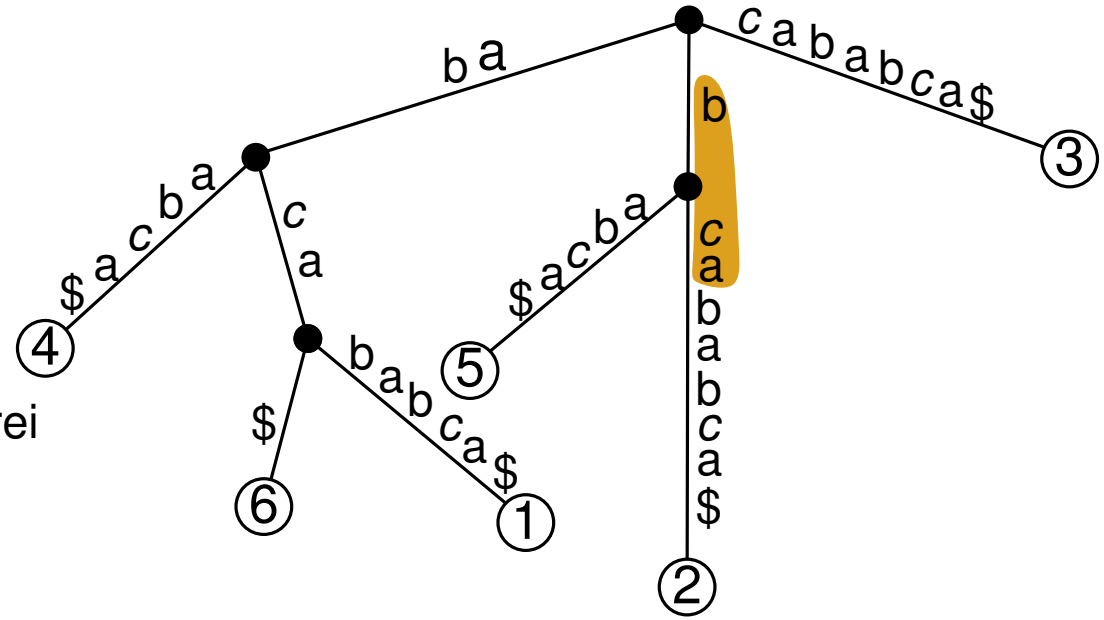
2. Fall: Matching endet an Knoten w: Führe neue ausgehende Kante e für w ein.

T=a b c a b a b c a $
S_7

Nächster Schritt:

Füge S_7 = b c a $ ein:

- Matching hört auf Pfad S_2 nach drei Zeichen auf.
- Folglich: erster Fall.



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Konstruktion

Gegeben: Text T
Gesucht: Suffixbaum von T

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

Initialisierung: N_1 besteht aus zwei Knoten und einer Kante beschriftet mit S_1.

Konstruktion von $N_i + 1$ aus N_i:

Suche längsten Pfad in N_i, der Präfix von S_{i+1} matched.

1. Fall: Wenn Matching in der Mitte von e endet, dann spalte e vor erstem Mismatching in zwei Kanten auf, indem neuer Knoten w eingeführt wird.

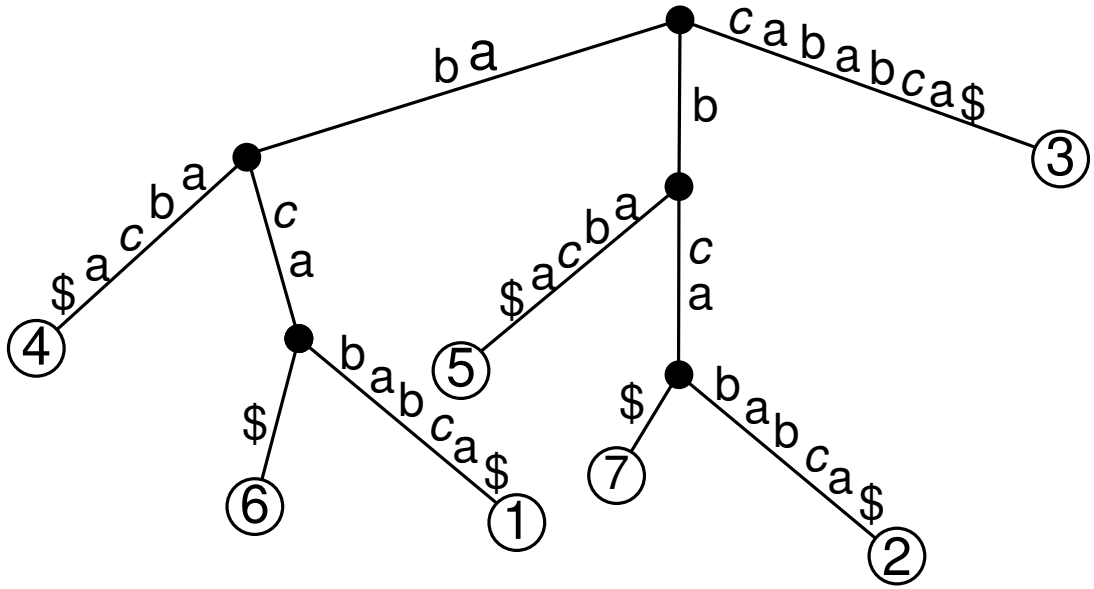
2. Fall: Matching endet an Knoten w: Führe neue ausgehende Kante e für w ein.

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

S_8, S_9 **und** S_{10} **analog.**

Kann in $O(n^2)$ implementiert werden.

Geht es auch besser?



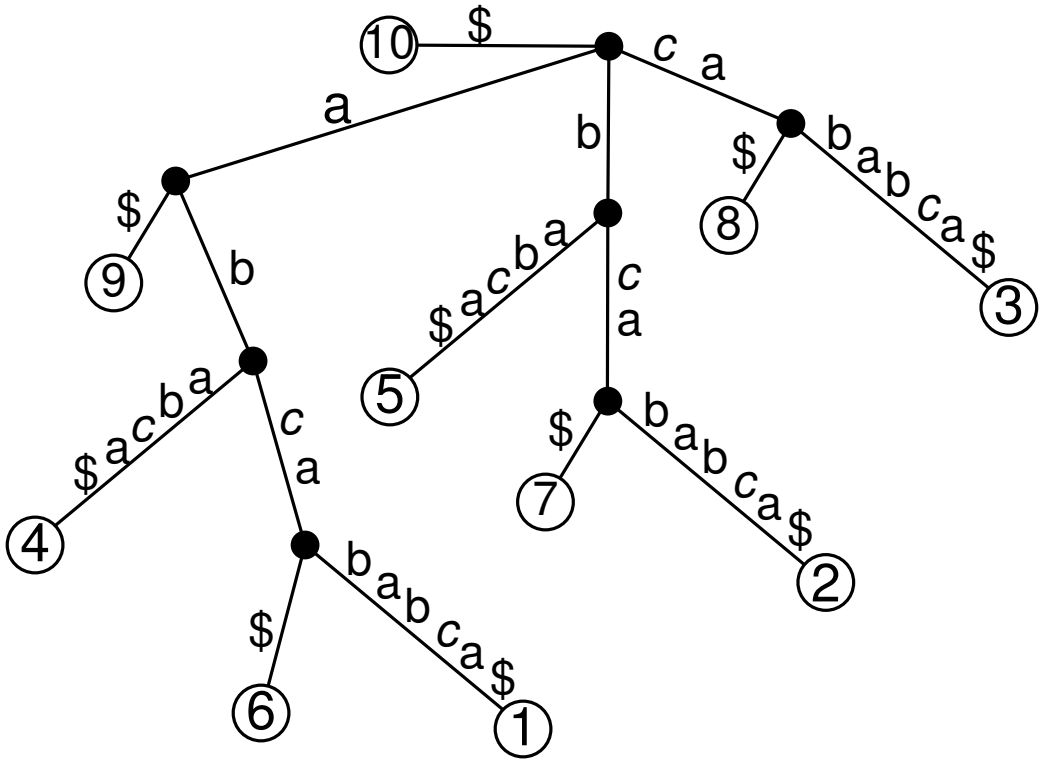
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suffix-Arrays

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 $1 < i \leq n$.

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

Beispiel: T= a b c a b a b c a $



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suffix-Arrays

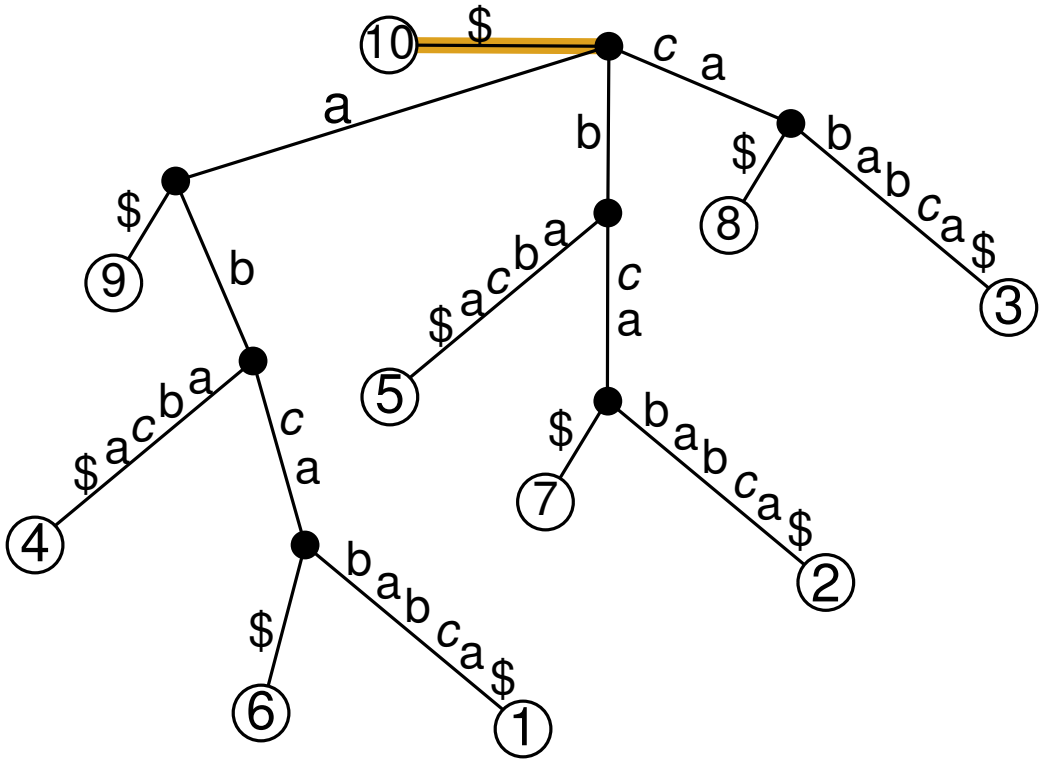
Definition: Ein *Suffix-Array* A eines Textes T ist eine Permutation von $\{1, \ldots, n\}$, sodass $S_{A[i]}$ das i-kleinste Suffix in lexikographischer Ordnung ist: $S_{A[i-1]} < S_{A[i]}$ für alle $1 < i \leq n$.

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

Beispiel: T= a b c a b a b c a $

$A =$

10									
$									



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suffix-Arrays

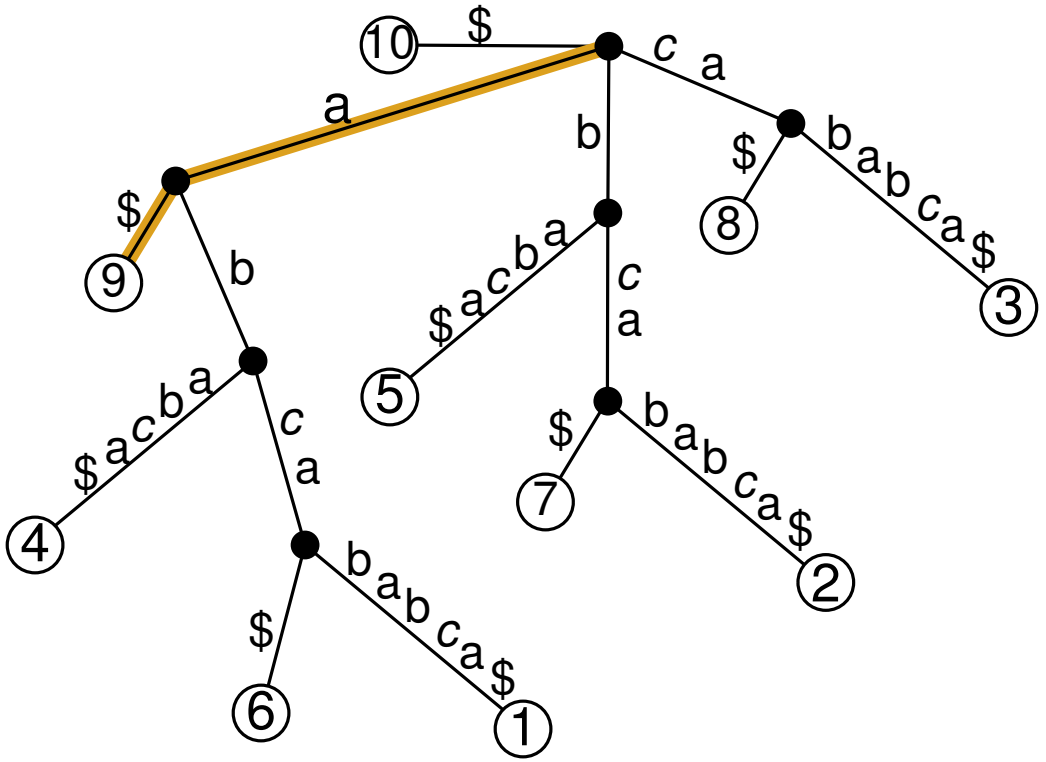
Definition: Ein *Suffix-Array* A eines Textes T ist eine Permutation von $\{1, \ldots, n\}$, sodass $S_{A[i]}$ das i-kleinste Suffix in lexikographischer Ordnung ist: $S_{A[i-1]} < S_{A[i]}$ für alle $1 < i \leq n$.

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

Beispiel: T= a b c a b a b c a $

$A =$

10	9								
$	a								
	$								



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suffix-Arrays

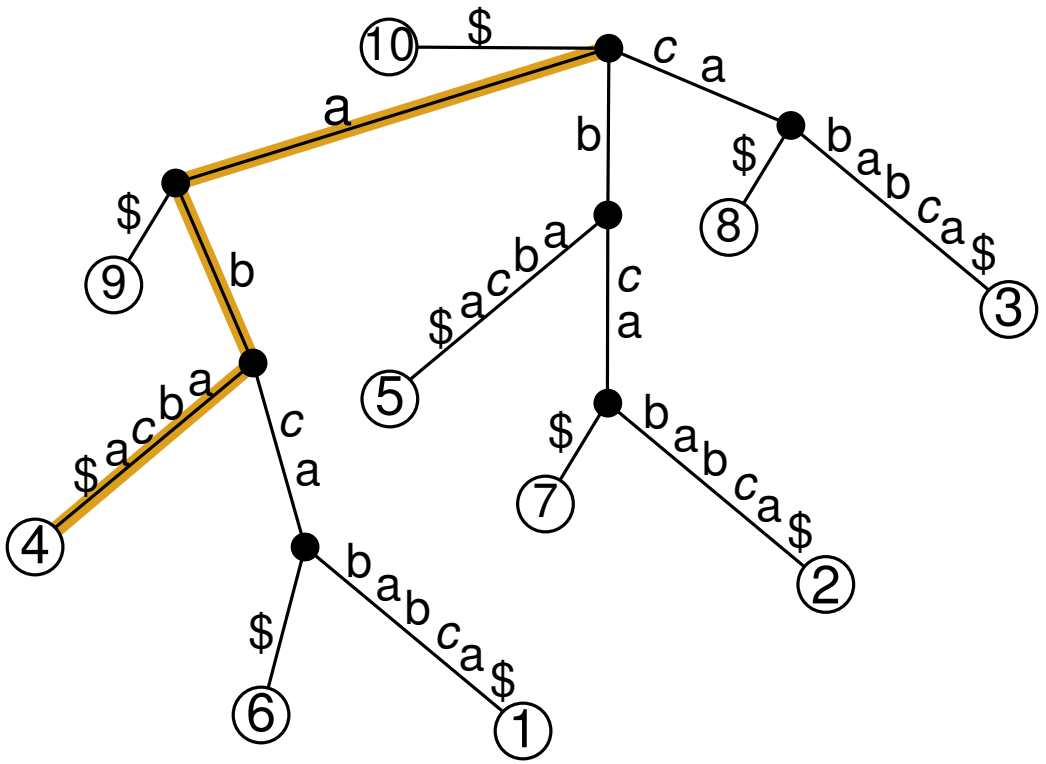
Definition: Ein *Suffix-Array* A eines Textes T ist eine Permutation von $\{1, \ldots, n\}$, sodass $S_{A[i]}$ das i-kleinste Suffix in lexikographischer Ordnung ist: $S_{A[i-1]} < S_{A[i]}$ für alle $1 < i \leq n$.

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

Beispiel: T= a b c a b a b c a $

$A =$

10	9	4							
$	a	a							
	$	*b*							
		a							
		b							
		c							
		a							
		$							



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suffix-Arrays

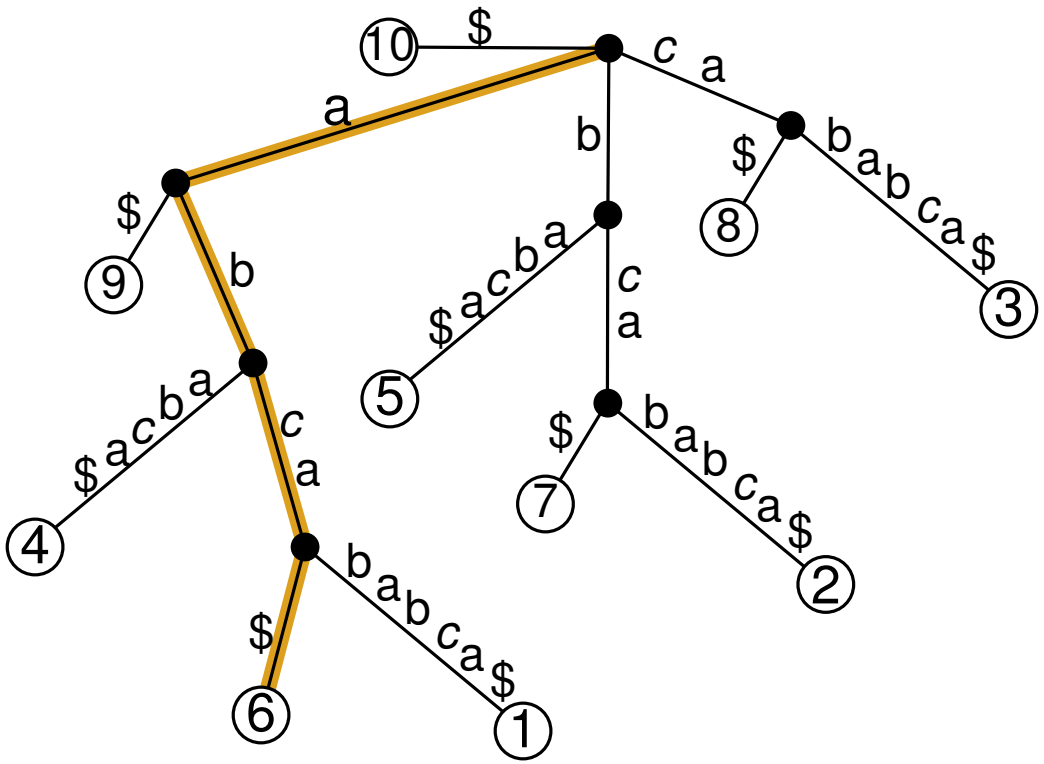
Definition: Ein *Suffix-Array* A eines Textes T ist eine Permutation von $\{1, \ldots, n\}$, sodass $S_{A[i]}$ das i-kleinste Suffix in lexikographischer Ordnung ist: $S_{A[i-1]} < S_{A[i]}$ für alle $1 < i \leq n$.

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

Beispiel: T= a b c a b a b c a $

$A =$

10	9	4	6						
$	a	a	a						
	$	*b*	*b*						
		a	*c*						
		b	a						
		c	$						
		a							
		$							



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suffix-Arrays

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 $1 < i \leq n$.

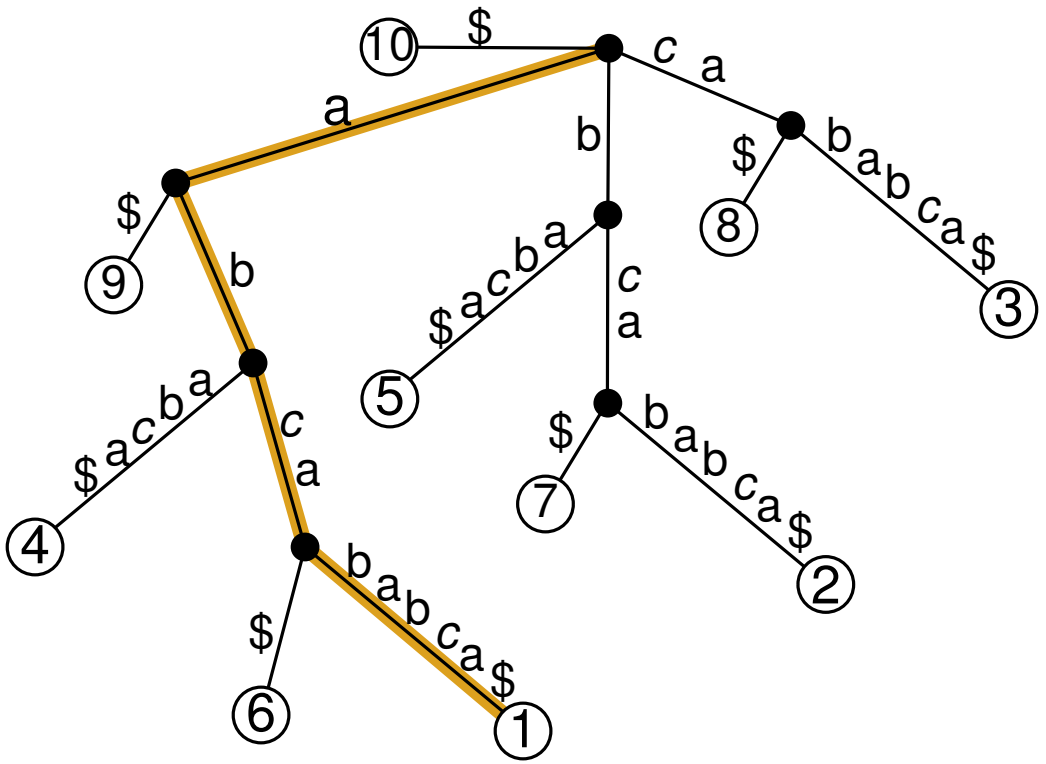
Sei S Suffixbaum von T. A entspricht der Reihenfolge der Blätter, die sich ergibt, wenn Tiefensuche auf S mit lexikographischer Ordnung angewendet wird.

Beispiel: T= a b c a b a b c a $

$A =$

10	9	4	6	1					

$	a	a	a	a
	$	*b*	*b*	*b*
		a	*c*	*c*
		b	a	a
		c	$	b
		a		a
		$		b
				c
				a
				$



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suffix-Arrays

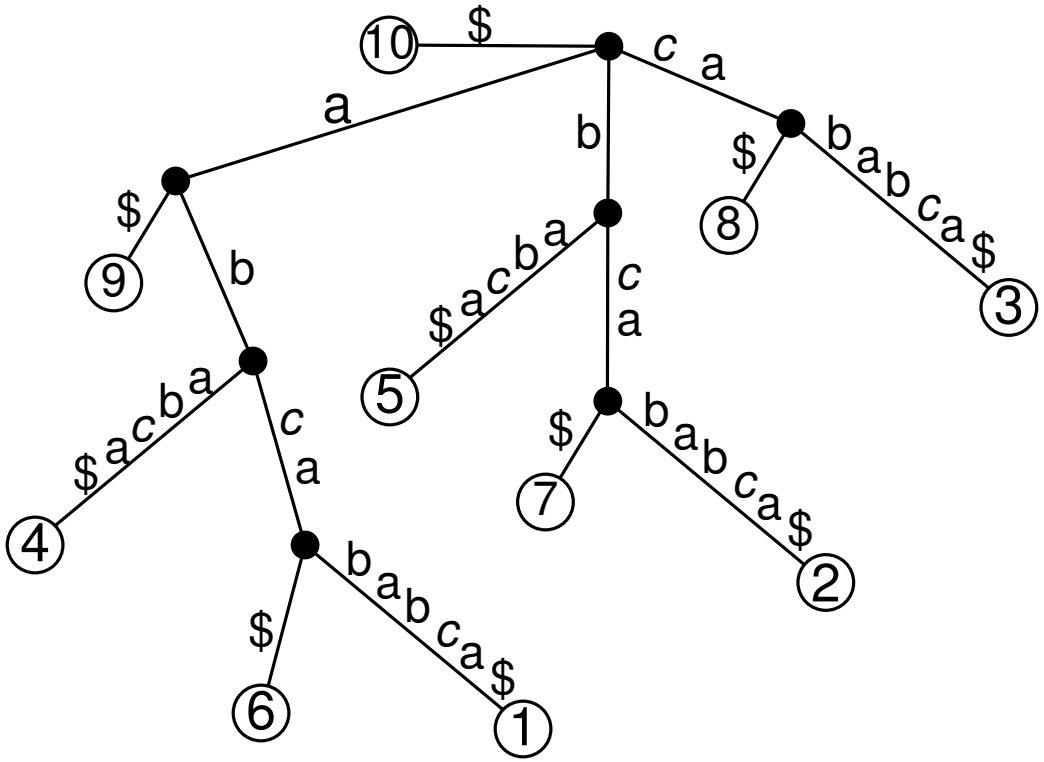
Definition: Ein *Suffix-Array* A eines Textes T ist eine Permutation von $\{1, \ldots, n\}$, sodass $S_{A[i]}$ das i-kleinste Suffix in lexikographischer Ordnung ist: $S_{A[i-1]} < S_{A[i]}$ für alle $1 < i \leq n$.

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

Beispiel: T= a b c a b a b c a $

$A =$

10	9	4	6	1	5	7	2	8	3
$	a $	a b a b c a $	a b c a $	a b c a b a b c a $	b a b c a $	b c a $	b c a b a b c a $	c a $	c a b a b c a $



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suchen in Suffix-Arrays

Gegeben: Suffix-Array A für Text T

Gesucht: Anzahl Vorkommen von Muster P in T

Beobachtung: Gesuchte Stellen bilden Intervall im Suffix-Array.

Beispiel: T= a b c a b a b c a $, P=a b

$A =$

10	9	4	6	1	5	7	2	8	3
$	a $	a b c a $	a b c a $	a b c a b a b c a $	b a b c a $	b c a $	b c a b a b c a $	c a $	c a b a b c a $

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suchen in Suffix-Arrays

Gegeben: Suffix-Array A für Text T

Gesucht: Anzahl Vorkommen von Muster P in T

Beobachtung: Gesuchte Stellen bilden Intervall im Suffix-Array.

Beispiel: T= a b c a b a b c a $, P=a b

$A =$

10	9	4	6	1	5	7	2	8	3
$	a $	a b a b c a $	a b c a $	a b c a b a b c a $	b a b c a $	b c a $	b c a b a b c a $	c a $	c a b a b c a $

Benutze binäre Suche um linke Intervalgrenze zu finden:

FINDELINKEGRENZE(Suffix-Array A)
$A' \leftarrow A$
solange $size(A') > 1$ **tue**
  Bestimme unteren Median i von A'.
  wenn *P lexikographisch größer als* $S_{A[i]}[1, m]$ **dann**
    $A' \leftarrow A'[i+1, size(A')]$ (rechtes Teilarray)
  sonst
    $A' \leftarrow A'[1, i]$ (linkes Teilarray)
$\ell \leftarrow$ Index von $A'[1]$ in A.
return ℓ

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suchen in Suffix-Arrays

Gegeben: Suffix-Array A für Text T

Gesucht: Anzahl Vorkommen von Muster P in T

Beobachtung: Gesuchte Stellen bilden Intervall im Suffix-Array.

Beispiel: T= a b c a b a b c a $, P=a b

$A =$

10	9	4	6	1	5	7	2	8	3
$	a $	a b a b c a $	a b c a $	a b c a b a b c a $	b a b c a $	b c a $	b c a b a b c a $	c a $	c a b a b c a $

Benutze binäre Suche um linke Intervalgrenze zu finden:

FINDELINKEGRENZE(Suffix-Array A)
$A' \leftarrow A$
solange $size(A') > 1$ **tue**
  Bestimme unteren Median i von A'.
  wenn *P lexikographisch größer als* $S_{A[i]}[1, m]$ **dann**
    $A' \leftarrow A'[i+1, size(A')]$ (rechtes Teilarray)
  sonst
    $A' \leftarrow A'[1, i]$ (linkes Teilarray)
$\ell \leftarrow$ Index von $A'[1]$ in A.
return ℓ

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suchen in Suffix-Arrays

Gegeben: Suffix-Array A für Text T

Gesucht: Anzahl Vorkommen von Muster P in T

Beobachtung: Gesuchte Stellen bilden Intervall im Suffix-Array.

Beispiel: T= a b c a b a b c a $, P=a b

$A =$

10	9	4	6	1	5	7	2	8	3
$	a $	a b a b c a $	a b c a $	a b c a b a b c a $	b a b c a $	b c a $	b c a b a b c a $	c a $	c a b a b c a $

Benutze binäre Suche um linke Intervalgrenze zu finden:

FINDELINKEGRENZE(Suffix-Array A)
$A' \leftarrow A$
solange $size(A') > 1$ **tue**
  Bestimme unteren Median i von A'.
  wenn *P lexikographisch größer als* $S_{A[i]}[1, m]$ **dann**
    $A' \leftarrow A'[i+1, size(A')]$ (rechtes Teilarray)
  sonst
    $A' \leftarrow A'[1, i]$ (linkes Teilarray)
$\ell \leftarrow$ Index von $A'[1]$ in A.
return ℓ

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suchen in Suffix-Arrays

Gegeben: Suffix-Array A für Text T

Gesucht: Anzahl Vorkommen von Muster P in T

Beobachtung: Gesuchte Stellen bilden Intervall im Suffix-Array.

Beispiel: T= a b c a b a b c a $, P=a b

$A =$

10	9	4	6	1	5	7	2	8	3
$	a $	a b a b c a $	a b c a $	a b c a b a b c a $	b a b c a $	b c a $	b c a b a b c a $	c a $	c a b a b c a $

Benutze binäre Suche um linke Intervalgrenze zu finden:

FINDELINKEGRENZE(Suffix-Array A)
$A' \leftarrow A$
solange $size(A') > 1$ **tue**
  Bestimme unteren Median i von A'.
  wenn *P lexikographisch größer als* $S_{A[i]}[1, m]$ **dann**
    $A' \leftarrow A'[i + 1, size(A')]$ (rechtes Teilarray)
  sonst
    $A' \leftarrow A'[1, i]$ (linkes Teilarray)
$\ell \leftarrow$ Index von $A'[1]$ in A.
return ℓ

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suchen in Suffix-Arrays

Gegeben: Suffix-Array A für Text T

Gesucht: Anzahl Vorkommen von Muster P in T

Beobachtung: Gesuchte Stellen bilden Intervall im Suffix-Array.

Beispiel: T= a b c a b a b c a $, P=a b

$A =$

10	9	4	6	1	5	7	2	8	3
$	a $	a b a b c a $	a b c a $	a b c a b a b c a $	b a b c a $	b c a $	b c a b a b c a $	c a $	c a b a b c a $

Benutze binäre Suche um linke Intervalgrenze zu finden:

FINDELINKEGRENZE(Suffix-Array A)
$A' \leftarrow A$
solange $size(A') > 1$ **tue**
  Bestimme unteren Median i von A'.
  wenn *P lexikographisch größer als* $S_{A[i]}[1, m]$ **dann**
    $A' \leftarrow A'[i+1, size(A')]$ (rechtes Teilarray)
  sonst
    $A' \leftarrow A'[1, i]$ (linkes Teilarray)
$\ell \leftarrow$ Index von $A'[1]$ in A.
return ℓ

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suchen in Suffix-Arrays

Gegeben: Suffix-Array A für Text T

Gesucht: Anzahl Vorkommen von Muster P in T

Beobachtung: Gesuchte Stellen bilden Intervall im Suffix-Array.

Beispiel: T= a b c a b a b c a $, P=a b

$A =$

10	9	4	6	1	5	7	2	8	3
$	a $	a b a b c a $	a b c a $	a b c a b a b c a $	b a b c a $	b c a $	b c a b a b c a $	c a $	c a b a b c a $

Benutze binäre Suche um linke Intervalgrenze zu finden:

FINDELINKEGRENZE(Suffix-Array A)
$A' \leftarrow A$
solange $size(A') > 1$ **tue**
  Bestimme unteren Median i von A'.
  wenn *P lexikographisch größer als* $S_{A[i]}[1, m]$ **dann**
    $A' \leftarrow A'[i+1, size(A')]$ (rechtes Teilarray)
  sonst
    $A' \leftarrow A'[1, i]$ (linkes Teilarray)
$\ell \leftarrow$ Index von $A'[1]$ in A.
return ℓ

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suchen in Suffix-Arrays

Gegeben: Suffix-Array A für Text T

Gesucht: Anzahl Vorkommen von Muster P in T

Beobachtung: Gesuchte Stellen bilden Intervall im Suffix-Array.

Beispiel: T= a b c a b a b c a $, P=a b

$A =$

10	9	4	6	1	5	7	2	8	3
$	a $	a b a b c a $	a b c a $	a b c a b a b c a $	b a b c a $	b c a $	b c a b a b c a $	c a $	c a b a b c a $

Benutze binäre Suche um linke Intervalgrenze zu finden:

FINDELINKEGRENZE(Suffix-Array A)
$A' \leftarrow A$
solange $size(A') > 1$ **tue**
  Bestimme unteren Median i von A'.
  wenn *P lexikographisch größer als* $S_{A[i]}[1, m]$ **dann**
    $A' \leftarrow A'[i+1, size(A')]$ (rechtes Teilarray)
  sonst
    $A' \leftarrow A'[1, i]$ (linkes Teilarray)
$\ell \leftarrow$ Index von $A'[1]$ in A.
return ℓ

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suchen in Suffix-Arrays

Gegeben: Suffix-Array A für Text T

Gesucht: Anzahl Vorkommen von Muster P in T

Beobachtung: Gesuchte Stellen bilden Intervall im Suffix-Array.

Beispiel: T= a b c a b a b c a $, P=a b

$A =$

10	9	4	6	1	5	7	2	8	3
$	a $	a b a b c a $	a b c a $	a b c a b a b c a $	b a b c a $	b c a $	b c a b a b c a $	c a $	c a b a b c a $

Benutze binäre Suche um linke Intervalgrenze zu finden:

FINDELINKEGRENZE(Suffix-Array A)
$A' \leftarrow A$
solange $size(A') > 1$ **tue**
  Bestimme unteren Median i von A'.
  wenn *P lexikographisch größer als* $S_{A[i]}[1, m]$ **dann**
    $A' \leftarrow A'[i + 1, size(A')]$ (rechtes Teilarray)
  sonst
    $A' \leftarrow A'[1, i]$ (linkes Teilarray)
$\ell \leftarrow$ Index von $A'[1]$ in A.
return ℓ

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Suchen in Suffix-Arrays

Gegeben: Suffix-Array A für Text T

Gesucht: Anzahl Vorkommen von Muster P in T

Beobachtung: Gesuchte Stellen bilden Intervall im Suffix-Array.

Beispiel: T= a b c a b a b c a $, P=a b

$A =$

10	9	4	6	1	5	7	2	8	3
$	a $	a b a b c a $	a b c a $	a b c a b a b c a $	b a b c a $	b c a $	b c a b a b c a $	c a $	c a b a b c a $

Analog binäre Suche für rechte Intervalgrenze:

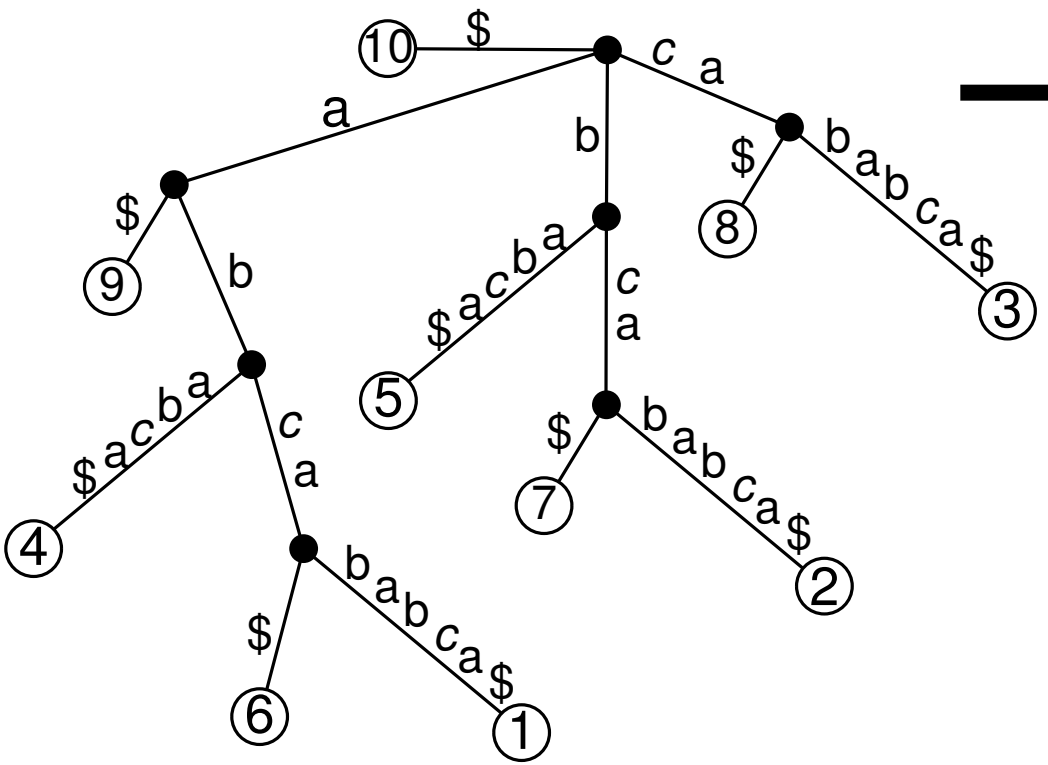
FINDERECHTEGRENZE(Suffix-Array A)
$A' \leftarrow A$
solange $size(A') > 1$ **tue**
  Bestimme oberen Median i von A'.
  wenn P *lexikographisch kleiner als* $S_{A[i]}[1, m]$ **dann**
    $A' \leftarrow A'[1, i-1]$ (linkes Teilarray)
  sonst
    $A' \leftarrow A'[i, size(A')]$ (rechtes Teilarray)
$r \leftarrow$ Index von $A'[1]$ in A.
return r

Lexikographischer Vergleich kann in $O(m)$ Zeit durchgeführt werden $\Rightarrow O(m \cdot \log n)$ Zeit.
Vergleiche: Suche im Suffixbaum in $O(m \log |\Sigma|)$ möglich. Normalerweise gilt $|\Sigma| \leq m \leq n$.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Konstruktion von Suffix-Arrays

1. Ansatz: Konstruktion aus Suffixbaum.



$A =$

10	9	4	6	1	5	7	2	8	3
$	a $	a b a b c a $	a b c a $	a b c a b a b c a $	b a b c a $	b c a $	b c a b a b c a $	c a $	c a b a b c a $

1. Konstruktion des Suffixbaums: $O(n^2)$ Zeit
2. Tiefensuche: $O(n)$ Zeit

$\Rightarrow$ $O(n^2)$ Zeit.

Im folgenden Konstruktion in $O(n)$ Zeit.

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 $i \not\equiv 0(3)$.

Beispiel:

T=	0	1	2	3	4	5	6	7	8	9	10	11	12
	y	a	b	b	a	d	a	b	b	a	d	o	$

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} =	1	4	8	2	7	5	10	11

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

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

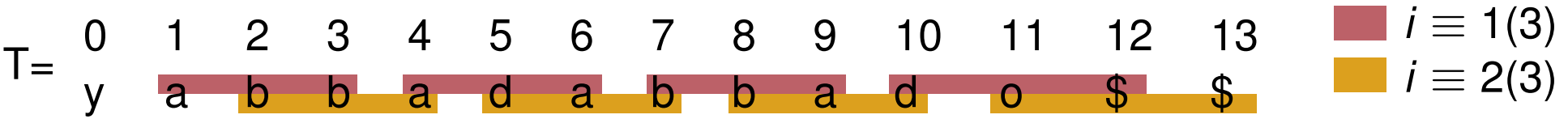
die sich zu $O(n)$ auflöst.

1. Schritt: Konstruktion von Suffixarray A_{12}

$T = t_0 t_1 \ldots t_{n-1}$ und A_{12} := Suffix-Array aller Suffixe mit $i \not\equiv 0(3)$.

Betrachte Tripel $t_i t_{i+1} t_{i+2}$ mit $i \not\equiv 0(3)$ als eigenes Zeichen (z.B. als Konkatenation der Binärrepräsentation). Fülle wenn nötig mit \$ auf.

Beispiel:

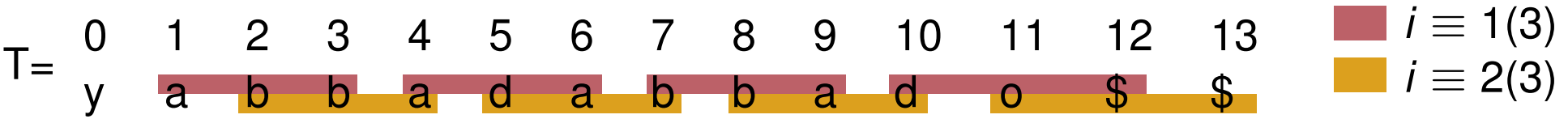


1. Schritt: Konstruktion von Suffixarray A_{12}

$T = t_0 t_1 \ldots t_{n-1}$ und A_{12} := Suffix-Array aller Suffixe mit $i \not\equiv 0(3)$.

Betrachte Tripel $t_i t_{i+1} t_{i+2}$ mit $i \not\equiv 0(3)$ als eigenes Zeichen (z.B. als Konkatenation der Binärrepräsentation). Fülle wenn nötig mit \$ auf.

Beispiel:



Sortiere die Menge $S = \{t_i t_{i+1} t_{i+2} \mid i \not\equiv 0(3)\}$ mithilfe von Radix-Sort in $O(n)$ Zeit (siehe Übung):

1	2	3	4	5	6	7
[*abb*]	[*ada*]	[*bad*]	[*bba*]	[*dab*]	[*do*\$]	[*o*\$\$]

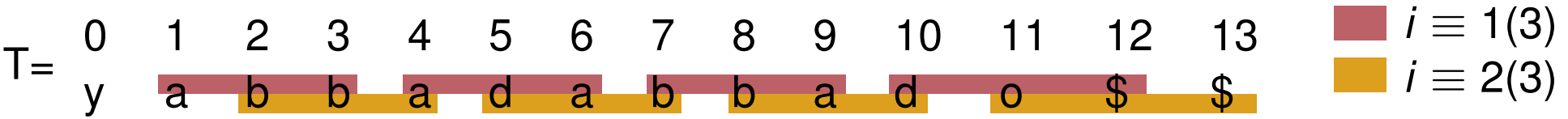
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

1. Schritt: Konstruktion von Suffixarray A_{12}

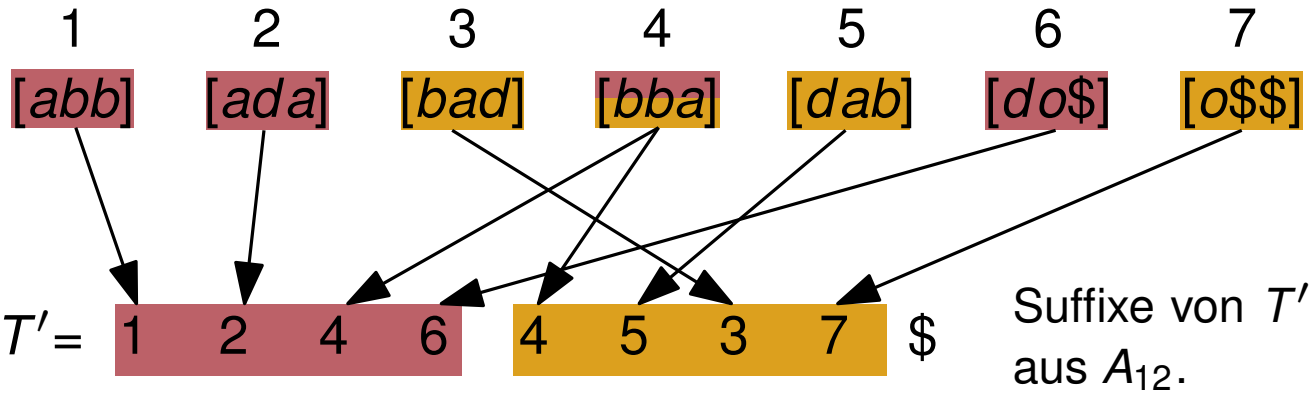
$T = t_0 t_1 \ldots t_{n-1}$ und A_{12} := Suffix-Array aller Suffixe mit $i \not\equiv 0(3)$.

Betrachte Tripel $t_i t_{i+1} t_{i+2}$ mit $i \not\equiv 0(3)$ als eigenes Zeichen (z.B. als Konkatenation der Binärrepräsentation). Fülle wenn nötig mit \$ auf.

Beispiel:



Sortiere die Menge $S = \{t_i t_{i+1} t_{i+2} \mid i \not\equiv 0(3)\}$ mithilfe von Radix-Sort in $O(n)$ Zeit (siehe Übung):



Suffixe von T' entsprechen gerade den Suffixen aus A_{12}.

Erste Hälfte von T' enthält Eimer-Nummern der Tripel $t_i t_{i+1} t_{i+2}$ für steigendes $i \equiv 1(3)$.

Zweite Hälfte von T' enthält Eimer-Nummern der Tripel $t_i t_{i+1} t_{i+2}$ für steigendes $i \equiv 2(3)$.

1. Schritt: Konstruktion von Suffix-Array A_{12}

gegeben: $T' =$ 1 2 4 6 4 5 3 7 $

Wenn $|T'| = O(1)$, dann konstruiere Suffix-Array A' von T' in $O(1)$ Zeit, ansonsten rufe SUFFIXARRAY(T') auf, um Suffix-Array A' zu erhalten.

A'

$A' =$ 8 0 1 6 4 2 5 3 7

daraus folgt:

$S' =$ \$ < 1 < 2 < 3 < 4 < 4 < 5 < 6 < 7

[abb] [ada] [bad] [bba] [bba] [dab] [do\$] [o\$\$]

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

1. A' beschreibt Sortierung S' der Suffixe aus T'.
2. Jedes Suffix aus T' entspricht einem Suffix S_i mit $i \not\equiv (0)3$.

Es ergibt sich also:

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

und somit $S_1 < S_4 < S_8 < S_2 < S_7 < S_5 < S_{10} < S_{11}$

2. Schritt: Konstruktion 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:

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

und somit $S_1 < S_4 < S_8 < S_2 < S_7 < S_5 < S_{10} < S_{11}$

Es gilt somit für Suffixe S_i mit $i \equiv 0(3)$:

$S_{12} < S_6$ weil \$ $< a$
$S_6 < S_9$ weil S_7 vor S_{10} in A_{12} steht.
$S_9 < S_3$ weil $a < b$
$S_3 < S_0$ weil $b < y$

$\Rightarrow S_{12} < S_6 < S_9 < S_3 < S_0$

T=

0	1	2	3	4	5	6	7	8	9	10	11	12	13
y	a	b	b	a	d	a	b	b	a	d	o	\$	\$

Vermengung von A_{12} und A_0

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:

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

und $S_1 < S_4 < S_8 < S_2 < S_7 < S_5 < S_{10} < S_{11}$

Aus Schritt 2 folgt:

$S_{12} < S_6 < S_9 < S_3 < S_0$

Vermengen wie bei Merge-Sort:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T=	y	a	b	b	a	d	a	b	b	a	d	o	$	$

$A=$ | |

Vermengung von A_{12} und A_0

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:

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

und $S_1 < S_4 < S_8 < S_2 < S_7 < S_5 < S_{10} < S_{11}$

Aus Schritt 2 folgt:

$S_{12} < S_6 < S_9 < S_3 < S_0$

Vermengen wie bei Merge-Sort:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T=	y	a	b	b	a	d	a	b	b	a	d	o	$	$

$A=$ | 12 |

$S_{12} < S_1$ weil $\$ < a$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Vermengung von A_{12} und A_0

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:

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

und $S_1 < S_4 < S_8 < S_2 < S_7 < S_5 < S_{10} < S_{11}$

Aus Schritt 2 folgt:

$S_{12} < S_6 < S_9 < S_3 < S_0$

Vermengen wie bei Merge-Sort:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T=	y	a	b	b	a	d	a	b	b	a	d	o	$	$

$A=$ | 12 | 1 |

$S_1 < S_6$ weil S_2 vor S_7 in A_{12} vorkommt.

Vermengung von A_{12} und A_0

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:

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

und $S_1 < S_4 < S_8 < S_2 < S_7 < S_5 < S_{10} < S_{11}$

Aus Schritt 2 folgt:

$S_{12} < S_6 < S_9 < S_3 < S_0$

Vermengen wie bei Merge-Sort:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T=	y	a	b	b	a	d	a	b	b	a	d	o	$	$

$A=$ | 12 | 1 | 6 |

$S_6 < S_4$ weil S_7 vor S_5 in A_{12} vorkommt.

Vermengung von A_{12} und A_0

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:

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

und $S_1 < S_4 < S_8 < S_2 < S_7 < S_5 < S_{10} < S_{11}$

Aus Schritt 2 folgt:

$S_{12} < S_6 < S_9 < S_3 < S_0$

Vermengen wie bei Merge-Sort:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T=	y	a	b	b	a	d	a	b	b	a	d	o	$	$

$A=$ | 12 | 1 | 6 | 4 |

$S_4 < S_9$ weil S_5 vor S_{10} in A_{12} vorkommt.

Vermengung von A_{12} und A_0

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:

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

und $S_1 < S_4 < S_8 < S_2 < S_7 < S_5 < S_{10} < S_{11}$

Aus Schritt 2 folgt:

$S_{12} < S_6 < S_9 < S_3 < S_0$

Vermengen wie bei Merge-Sort:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T=	y	a	b	b	a	d	a	b	b	a	d	o	$	$

$A=$ | 12 | 1 | 6 | 4 | 9 |

$S_9 < S_8$ weil $a < b$.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Vermengung von A_{12} und A_0

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:

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

und $S_1 < S_4 < S_8 < S_2 < S_7 < S_5 < S_{10} < S_{11}$

Aus Schritt 2 folgt:

$S_{12} < S_6 < S_9 < S_3 < S_0$

Vermengen wie bei Merge-Sort:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
T=	y	a	b	b	a	d	a	b	b	a	d	o	$	$

$A=$

12	1	6	4	9	3	8	2	7	5	10	11	0	–

Zusammenfassung: Suffix-Arrays

Theorem 31: Ein Suffix-Array kann für einen Text T der Länge n in $O(n)$ Zeit konstruiert werden.

Konstruktion von Suffixbäumen

Bisher sind Suffix-Arrays im Aufbau schneller, aber in der Abfrage langsamer als Suffix-Bäume.

Idee: Konstruiere Suffixbäume aus Suffix-Arrays in $O(n)$-Zeit.

Definition 11: Gegeben ein Text T der Länge n und dessen Suffix-Array A. Ein *LCP-Array* L ist ein Array der Größe n, sodass

- $L[1] = 0$
- $L[i]$ = Länge des längsten gemeinsamen Präfixes von $A[i]$ und $A[i-1]$ für $i > 1$

LCP = Longest Common Prefix

Beispiel LCP-Array

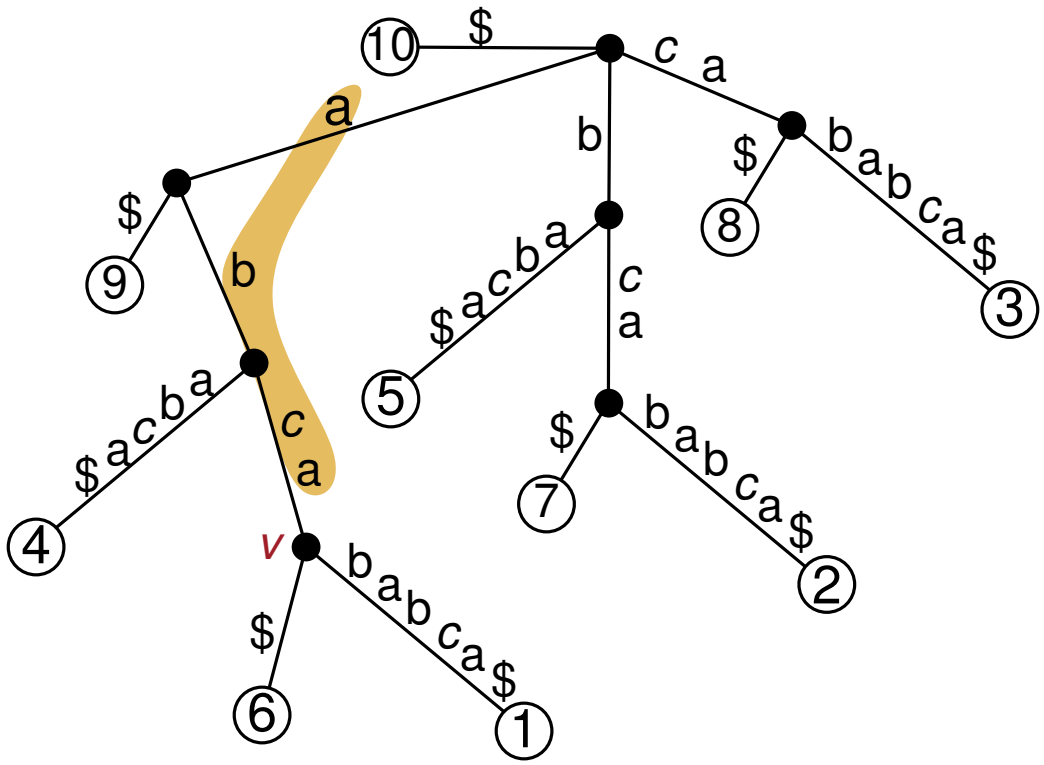
Definition 11: Gegeben ein Text T der Länge n und dessen Suffix-Array A. Ein *LCP-Array* L ist ein Array der Größe n, sodass

- $L[1] = 0$
- $L[i]$ = Länge des längsten gemeinsamen Präfixes von $A[i]$ und $A[i-1]$ für $i > 1$

Beispiel: T= a b c a b a b c a $

$A =$	10	9	4	6	1	5	7	2	8	3
$L =$	0	0	1	2	4	0	1	3	0	2

$
a $
a b a b c a $
a b c a $
a b c a b a b c a $
a b c a $
b c a $
b c a b a b c a $
c a $
c a b a b c a $



Bsp.: v ist niedrigster gemeinsamer Vorgänger von Knoten 6 und 1. Die Stringtiefe d(v) von v entspricht gerade der Länge des kleinsten gemeinsamen Präfixes von S_1 und S_6.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Konstruktion von Suffixbäumen

Gegeben: Text T, Suffix-Array A und LCP-Array L von T

Gesucht: Suffixbaum von T

Idee: Konstruiere $N_0, N_1, \ldots, N_n$ Suffixbäume, sodass N_i Suffixbaum von $S_{A[1]}, \ldots, S_{A[i]}$ ist, d.h. Suffix-Array gibt Reihenfolge an, in der Suffixe eingefügt werden.

Initialisierung: N_0 enthält nur die Wurzel und repräsentiert damit das leere Wort.

Konstruktion von N_{i+1} aus N_i:

1. Betrachte den Pfad P des Suffixes $S_{A[i]}$ in N_i, also den rechtesten Pfad in N_i.
2. Wähle den tiefsten Knoten v auf P mit $d(v) \leq L[i+1]$.

1. Fall: $d(v) = L[i+1]$.
Hänge an v ein neues Blatt x und beschrifte (v, x) mit $T[A[i+1] + L[i+1], n]$. Blatt x wird mit $A[i+1]$ beschriftet.

Konstruktion von Suffixbäumen

Gegeben: Text T, Suffix-Array A und LCP-Array L von T

Gesucht: Suffixbaum von T

Idee: Konstruiere $N_0, N_1, \ldots, N_n$ Suffixbäume, sodass N_i Suffixbaum von $S_{A[1]}, \ldots, S_{A[i]}$ ist, d.h. Suffix-Array gibt Reihenfolge an, in der Suffixe eingefügt werden.

Initialisierung: N_0 enthält nur die Wurzel und repräsentiert damit das leere Wort.

Konstruktion von N_{i+1} aus N_i:

1. Betrachte den Pfad P des Suffixes $S_{A[i]}$ in N_i, also den rechtesten Pfad in N_i.
2. Wähle den tiefsten Knoten v auf P mit $d(v) \leq L[i+1]$.

2. Fall $d(v) < L[i+1]$. Sei w das Kind v auf dem rechtesten Pfad von N_i.

1. Entferne (v, w).
2. Füge neuen Knoten y mit Kante (v, y) ein. Beschriftung von (v, y):
   $T[A[i] + d(v), A[i] + L[i+1] - 1]$
3. Füge (y, w) ein mit Beschriftung
   $T[A[i] + L[i+1], A[i] + d(w) - 1]$.
4. Füge Blatt x mit Beschriftung $A[i+1]$ ein und Kante (y, x) mit Beschriftung
   $T[A[i+1] + L[i+1], n]$.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

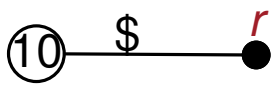
Beispiel

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

$A =$	10	9	4	6	1	5	7	2	8	3
$L =$	0	0	1	2	4	0	1	3	0	2

$
a $
a b a b c a $
a b c a $
a b c a b a b c a $
b a b c a $
b c a $
b c a b a b c a $
c a $
c a b a b c a $

Beispiel



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

$A =$	10	9	4	6	1	5	7	2	8	3
$L =$	0	0	1	2	4	0	1	3	0	2

$
a $
a b a b c a $
a b c a $
a b c a b a b c a $
b a b c a $
b c a $
b c a b a b c a $
c a $
c a b a b c a $

1. Betrachte den Pfad P des Suffixes $S_{A[i]}$ in N_i, also den rechtesten Pfad in N_i.
2. Wähle den tiefsten Knoten v auf P mit $d(v) \leq L[i+1]$.

1. Fall: $d(v) = L[i+1]$.
Hänge an v ein neues Blatt x und beschrifte (v, x) mit $T[A[i+1] + L[i+1], n]$. Blatt x wird mit $A[i+1]$ beschriftet.

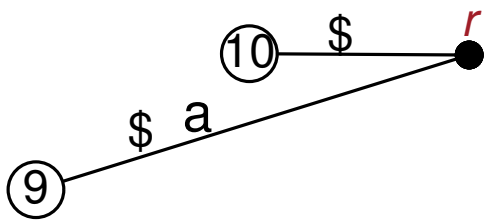
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Beispiel

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

$A =$	10	9	4	6	1	5	7	2	8	3
$L =$	0	0	1	2	4	0	1	3	0	2

$
a $
a b a b c a $
a b c a $
a b c a b a b c a $
b a b c a $
b c a $
b c a b a b c a $
c a $
c a b a b c a $



1. Betrachte den Pfad P des Suffixes $S_{A[i]}$ in N_i, also den rechtesten Pfad in N_i.
2. Wähle den tiefsten Knoten v auf P mit $d(v) \leq L[i+1]$.

1. Fall: $d(v) = L[i+1]$.
Hänge an v ein neues Blatt x und beschrifte (v, x) mit $T[A[i+1] + L[i+1], n]$. Blatt x wird mit $A[i+1]$ beschriftet.

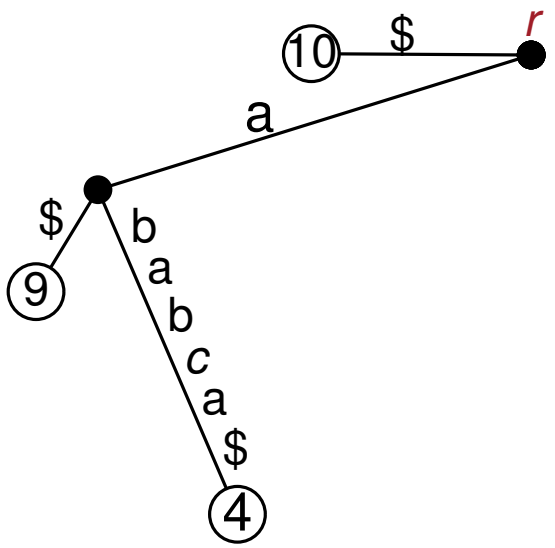
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Beispiel

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

$A =$	10	9	4	6	1	5	7	2	8	3
$L =$	0	0	1	2	4	0	1	3	0	2

$
a $
a b a b c a $
a b c a $
a b c a b a b c a $
b a b c a $
b c a $
b c a b a b c a $
c a $
c a b a b c a $



1. Betrachte den Pfad P des Suffixes $S_{A[i]}$ in N_i, also den rechtesten Pfad in N_i.
2. Wähle den tiefsten Knoten v auf P mit $d(v) \leq L[i+1]$.

2. Fall $d(v) < L[i+1]$. Sei w das Kind v auf dem rechtesten Pfad von N_i.

1. Entferne (v, w).
2. Füge neuen Knoten y mit Kante (v, y) ein. Beschriftung von (v, y): $T[A[i] + d(v), A[i] + L[i+1] - 1]$
3. Füge (y, w) ein mit Beschriftung $T[A[i] + L[i+1], A[i] + d(w) - 1]$.
4. Füge Blatt x mit Beschriftung $A[i+1]$ ein und Kante (y, x) mit Beschriftung $T[A[i+1] + L[i+1], n]$.

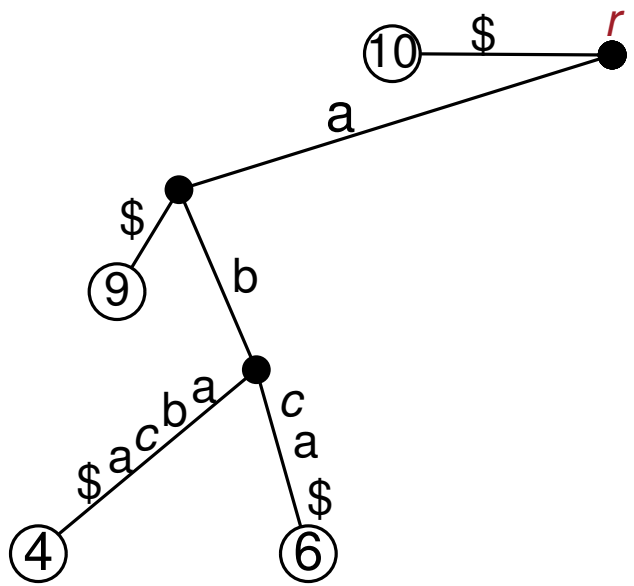
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Beispiel

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

$A =$	10	9	4	6	1	5	7	2	8	3
$L =$	0	0	1	2	4	0	1	3	0	2

$ a $ a b a b c a $ a b c a $ a b c a b a b c a $ b a b c a $ b c a $ b c a b a b c a $ c a $ c a b a b c a $



1. Betrachte den Pfad P des Suffixes $S_{A[i]}$ in N_i, also den rechtesten Pfad in N_i.
2. Wähle den tiefsten Knoten v auf P mit $d(v) \leq L[i+1]$.

2. Fall $d(v) < L[i+1]$. Sei w das Kind v auf dem rechtesten Pfad von N_i.

1. Entferne (v, w).
2. Füge neuen Knoten y mit Kante (v, y) ein. Beschriftung von (v, y): $T[A[i] + d(v), A[i] + L[i+1] - 1]$
3. Füge (y, w) ein mit Beschriftung $T[A[i] + L[i+1], A[i] + d(w) - 1]$.
4. Füge Blatt x mit Beschriftung $A[i+1]$ ein und Kante (y, x) mit Beschriftung $T[A[i+1] + L[i+1], n]$.

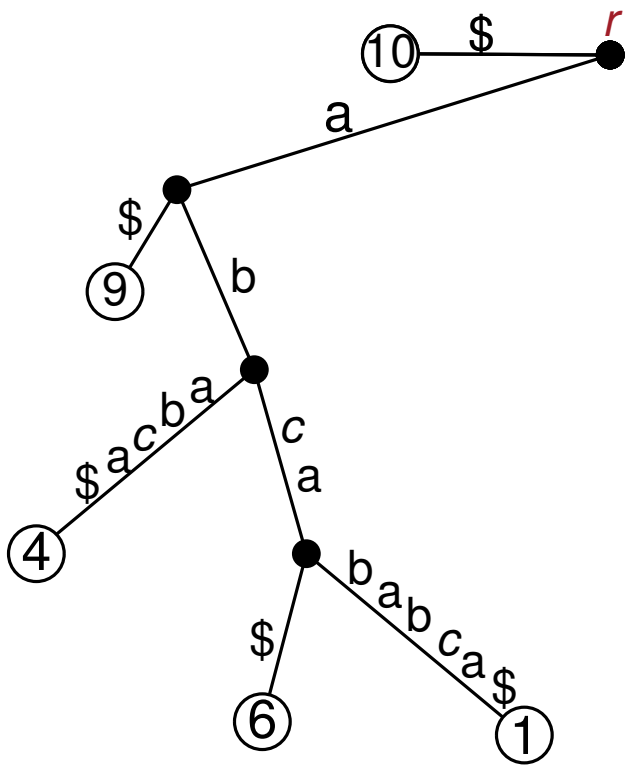
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Beispiel

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

$A =$	10	9	4	6	1	5	7	2	8	3
$L =$	0	0	1	2	4	0	1	3	0	2

$ a $ a b a b c a $ a b c a $ a b c a b a b c a $ b a b c a $ b c a $ b c a b a b c a $ c a $ c a b a b c a $



1. Betrachte den Pfad P des Suffixes $S_{A[i]}$ in N_i, also den rechtesten Pfad in N_i.
2. Wähle den tiefsten Knoten v auf P mit $d(v) \leq L[i+1]$.

2. Fall $d(v) < L[i+1]$. Sei w das Kind v auf dem rechtesten Pfad von N_i.

1. Entferne (v, w).
2. Füge neuen Knoten y mit Kante (v, y) ein. Beschriftung von (v, y): $T[A[i] + d(v), A[i] + L[i+1] - 1]$
3. Füge (y, w) ein mit Beschriftung $T[A[i] + L[i+1], A[i] + d(w) - 1]$.
4. Füge Blatt x mit Beschriftung $A[i+1]$ ein und Kante (y, x) mit Beschriftung $T[A[i+1] + L[i+1], n]$.

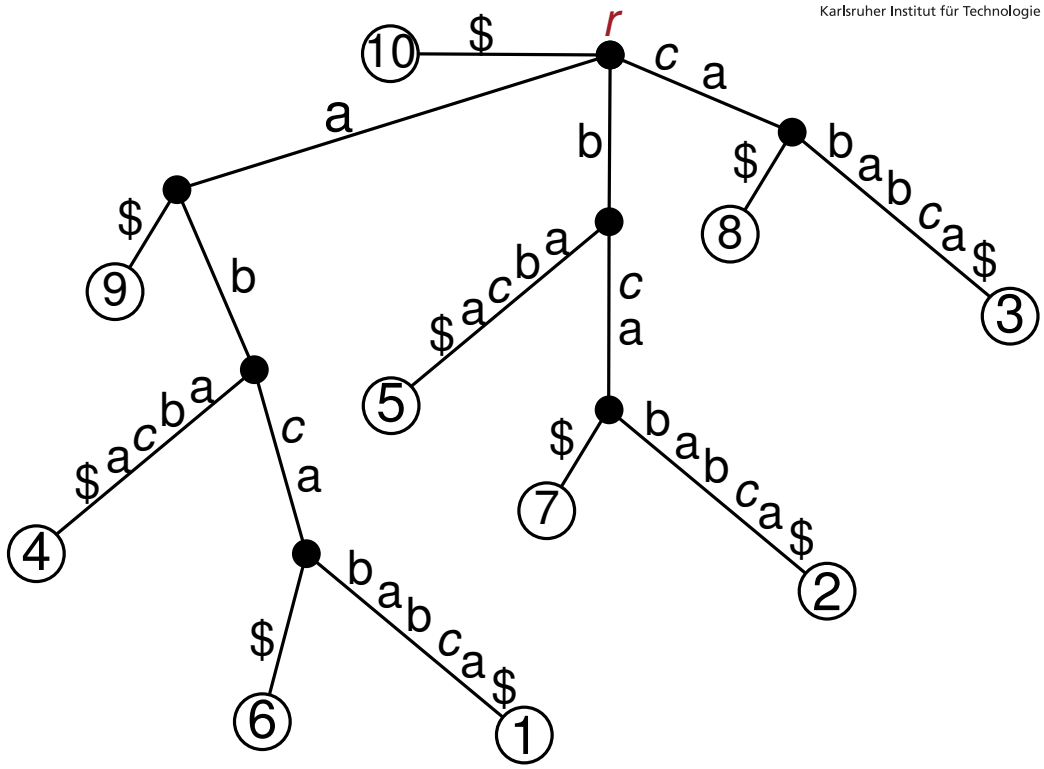
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Beispiel

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

$A =$	10	9	4	6	1	5	7	2	8	3
$L =$	0	0	1	2	4	0	1	3	0	2

$	a $	a b a b c a $	a b c a $	a b c a b a b c a $	b a b c a $	b c a $	b c a b a b c a $	c a $	c a b a b c a $



Theorem 28: Gegeben ein Suffix-Array A und ein LCP-Array L eines Texts T, dann kann der Suffixbaum von T in $O(n)$ Zeit konstruiert werden.

Wenn L in $O(n)$ konstruierbar, dann ist der Suffixbaum S in $O(n)$ Zeit konstruierbar.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Konstruktion LCP-Array

Gegeben: Text T, Suffix-Array A von T

$A =$

10	9	4	6	1	5	7	2	8	3

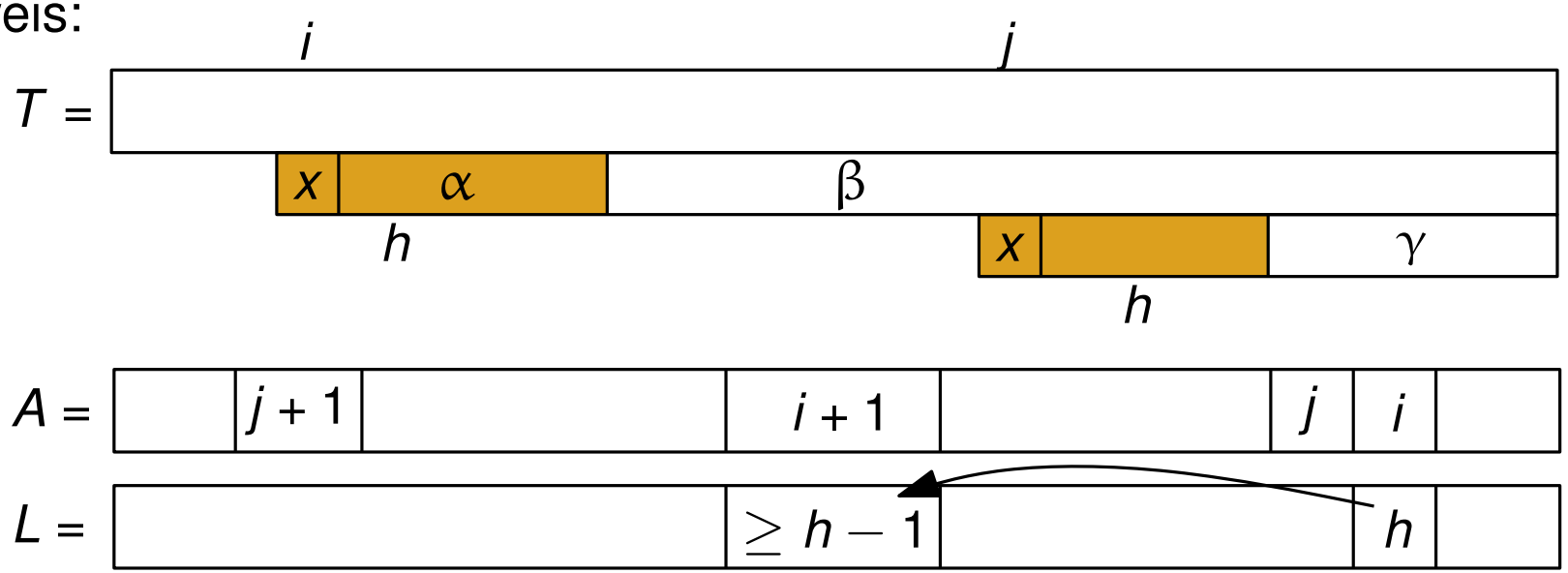
Sei A^{-1} das Inverse Array von A: $A^{-1}[A[i]] = i$.

$A^{-1} =$

5	8	10	3	6	4	7	9	2	1

Lemma 32: Für alle $1 \leq i < n$ gilt $L[A^{-1}[i+1]] \geq L[A^{-1}[i]] - 1$.

Skizze für Beweis:



Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Konstruktion LCP-Array

```
ConstructLCPArray(T=t_1 t_2 ... t_n, Suffix-Array A)
    h ← 0
    L[1] ← 0
    für i = 1 ..., n tue
        wenn A^{-1}[i] ≠ 1 dann
            solange t_{i+h} = t_{A[A^{-1}[i]-1]+h} tue
                h ← h + 1
            L[A^{-1}[i]] ← h
            h ← max(0, h − 1)
```

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

A =	10	9	4	6	1	5	7	2	8	3
A^{-1} =	5	8	10	3	6	4	7	9	2	1
L =										
	1	2	3	4	5	6	7	8	9	10

Konstruktion LCP-Array

```
ConstructLCPArray(T=t_1 t_2 ... t_n, Suffix-Array A)
    h ← 0
    L[1] ← 0
    für i = 1 ..., n tue
        wenn A^{-1}[i] ≠ 1 dann
            solange t_{i+h} = t_{A[A^{-1}[i]-1]+h} tue
                h ← h + 1
            L[A^{-1}[i]] ← h
            h ← max(0, h − 1)
```

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

A =	10	9	4	6	1	5	7	2	8	3
A^{-1} =	5	8	10	3	6	4	7	9	2	1
L =					4					
	1	2	3	4	5	6	7	8	9	10

Für i=1	$t_i + h$	$t_{A[A^{-1}[i]-1]+h}$
$h = 0$	t_1=a	$t_6 = a$
$h = 1$	t_2=b	$t_7 = b$
$h = 2$	t_3=c	$t_8 = c$
$h = 3$	t_4=a	$t_9 = a$
$h = 4$	t_5=b	t_{10} = $

Konstruktion LCP-Array

```
ConstructLCPArray(T=t_1 t_2 ... t_n, Suffix-Array A)
    h ← 0
    L[1] ← 0
    für i = 1 ..., n tue
        wenn A^{-1}[i] ≠ 1 dann
            solange t_{i+h} = t_{A[A^{-1}[i]-1]+h} tue
                h ← h + 1
            L[A^{-1}[i]] ← h
            h ← max(0, h − 1)
```

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

A =	10	9	4	6	1	5	7	2	8	3
A^{-1} =	5	8	10	3	6	4	7	9	2	1
L =	0	0	1	2	4	0	1	3	0	2
	1	2	3	4	5	6	7	8	9	10

Für i=1	$t_i + h$	$t_{A[A^{-1}[i]-1]+h}$
$h = 0$	t_1=a	$t_6 = a$
$h = 1$	t_2=b	$t_7 = b$
$h = 2$	t_3=c	$t_8 = c$
$h = 3$	t_4=a	$t_9 = a$
$h = 4$	t_5=b	t_{10} = $

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Zusammenfassung

Theorem 33: Gegeben ein Text T und dessen Suffix-Array A, dann kann das entsprechende LCP-Array L in $O(n)$ Zeit konstruiert werden.

Theorem : Geben ein Text T. Der Suffixbaum von T kann in $O(n)$ Zeit konstruiert werden.

Erinnerung: Suffixbaum bei Suche besser als Suffix-Array. Deshalb lohnt sich die Transformation.