Karlsruher Institut für Technologie

Algorithmen II
Vorlesung am 19.12.2013

INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER

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

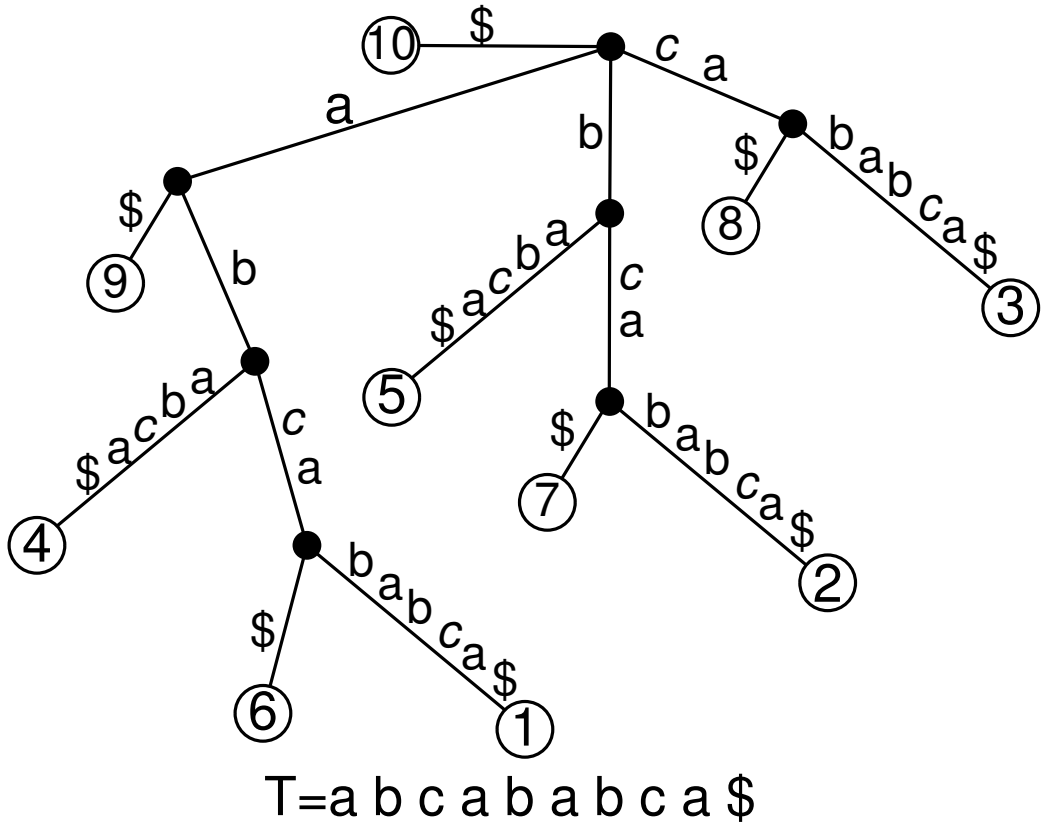
www.kit.edu

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.



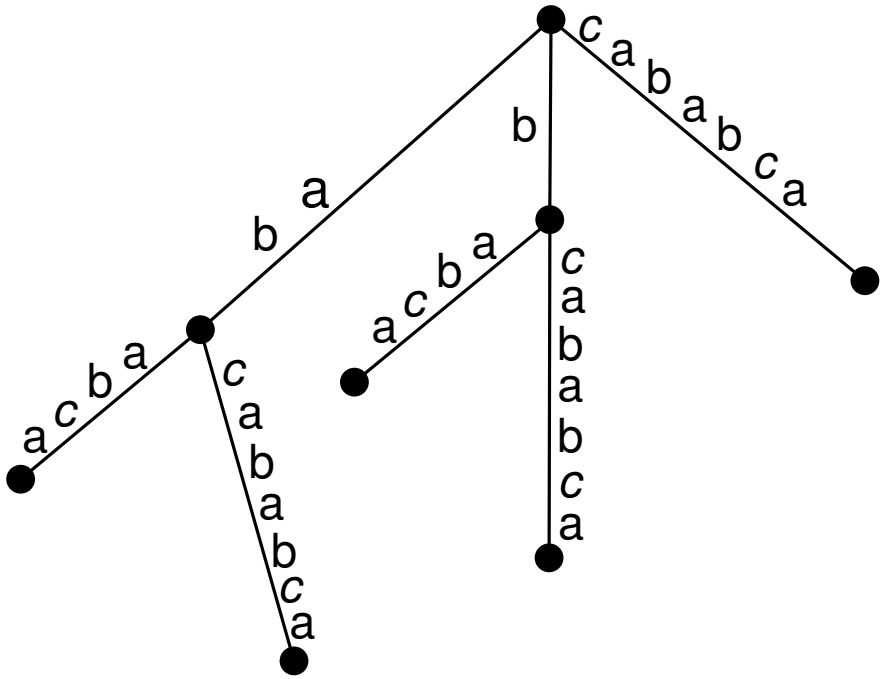
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

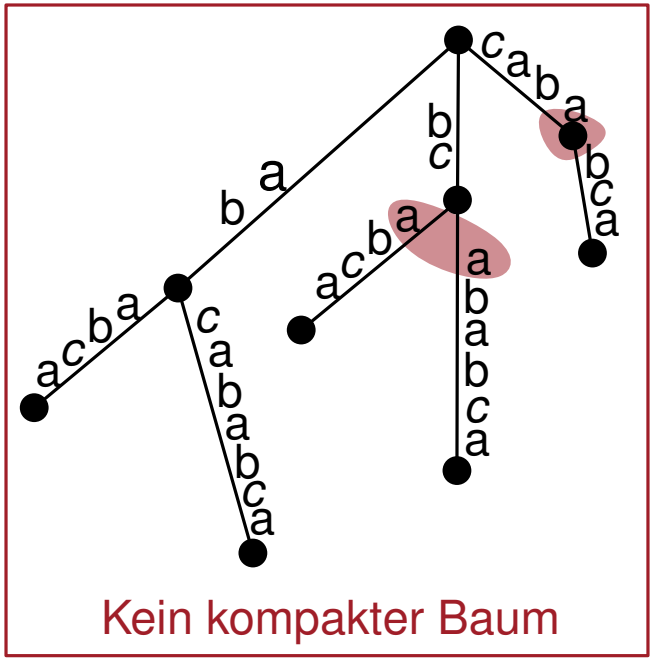
Kompakte Bäume

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





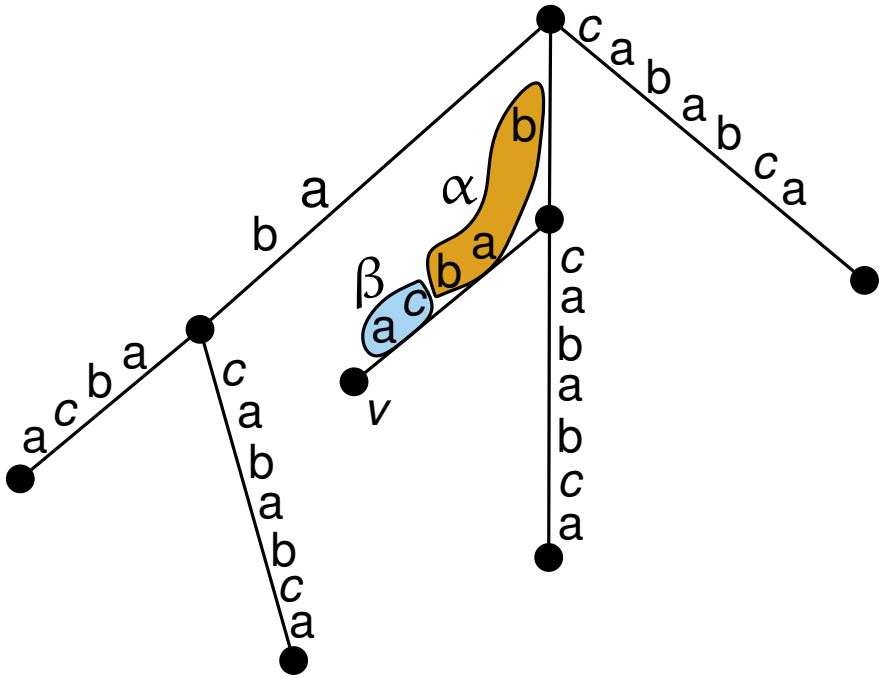
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

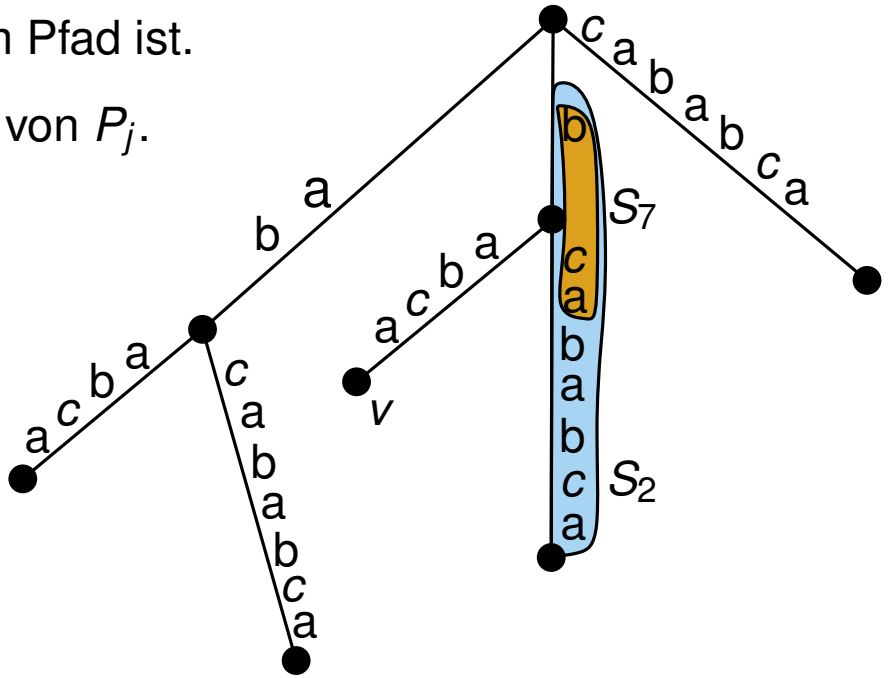
Suffixbaum

Definition: Ein *Suffixbaum* ist ein kompakter $\Sigma^\star$-Baum S, sodass S 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:

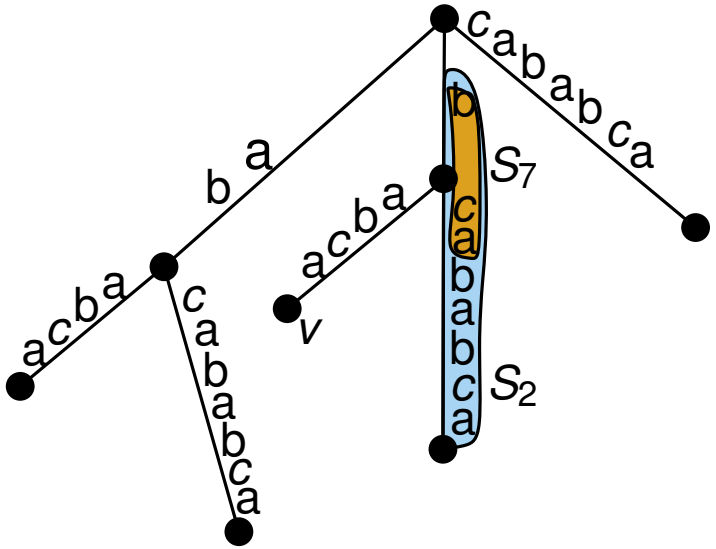
1. S_i bezeichnet das Suffix von T beginnend ab der i-ten Stelle.
2. $T[i,j]$ bezeichnet den Teilstring von T von der i-ten bis zur j-ten Stelle.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

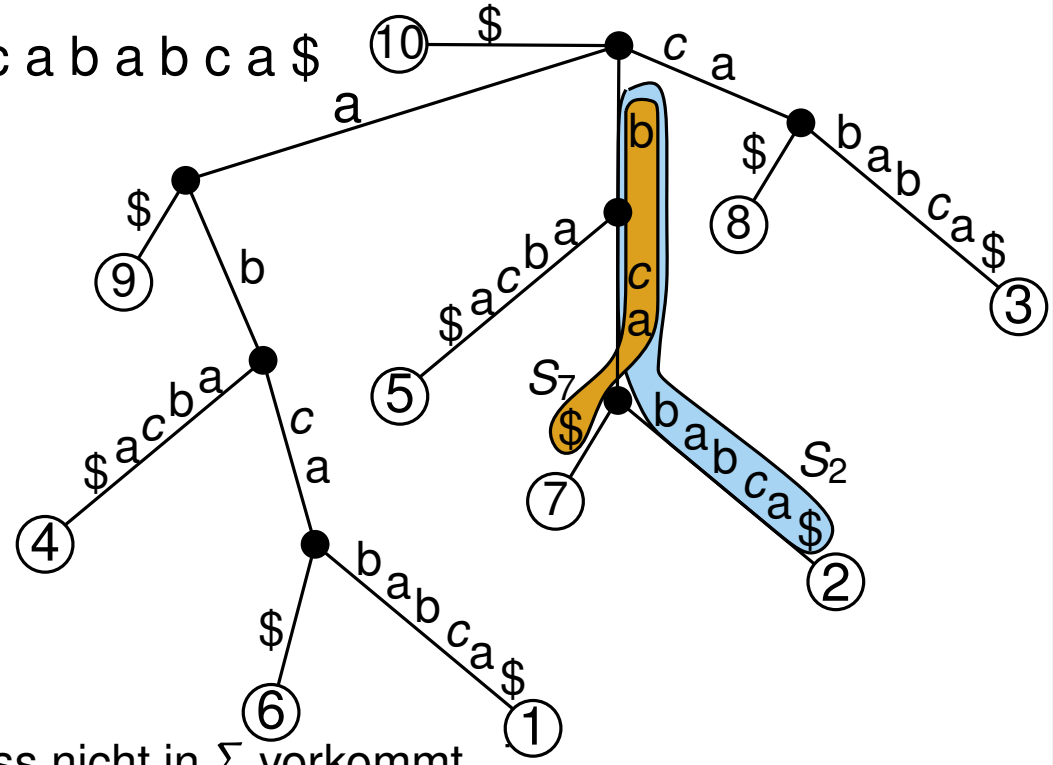
Suffixbaum

Definition: Ein *Suffixbaum* ist ein kompakter $\Sigma^\star$-Baum S, sodass S 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.

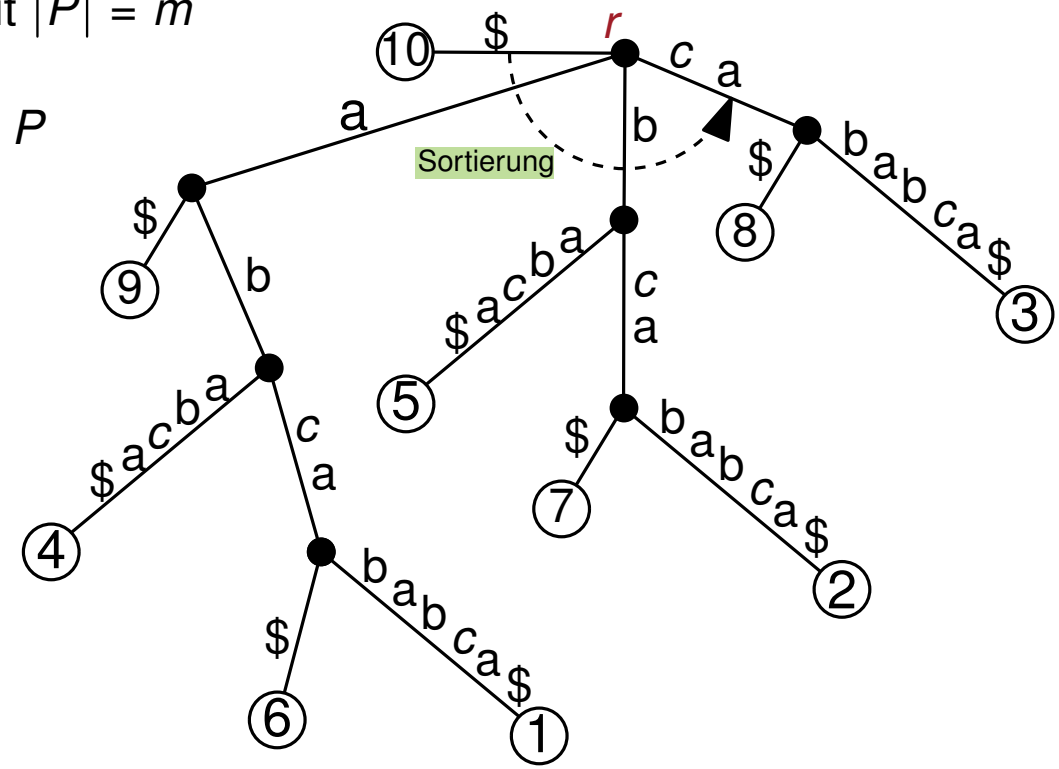
Suchen im Suffixbaum

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.



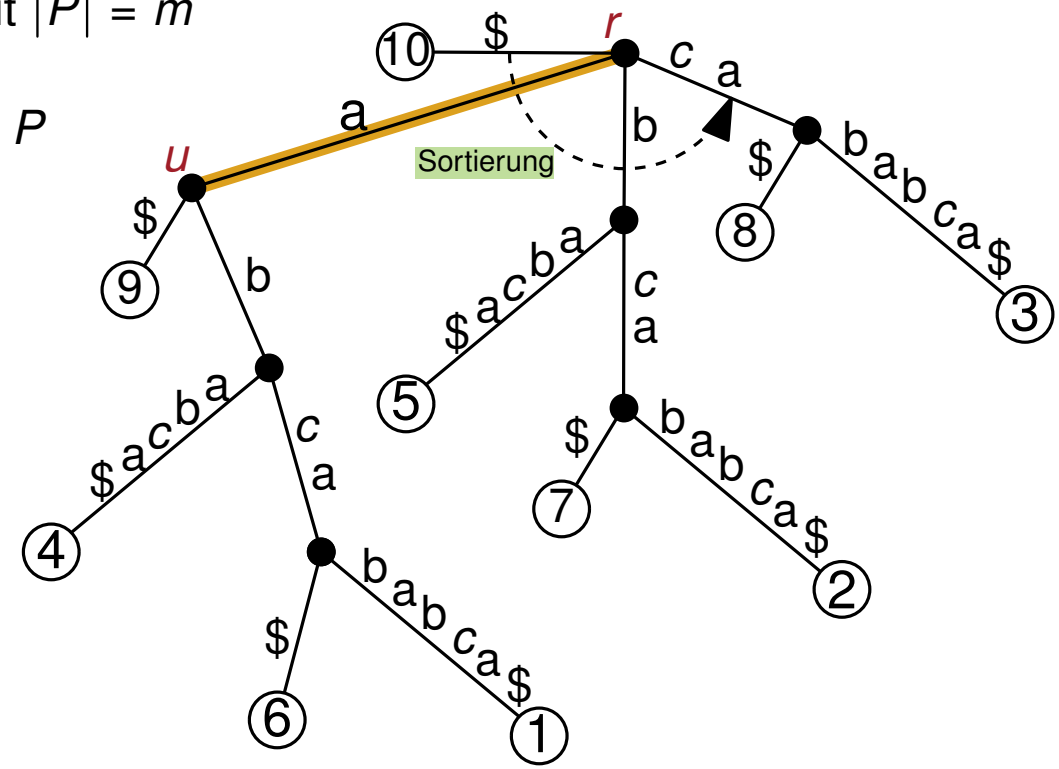
Beispiel: P= a b c a

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.

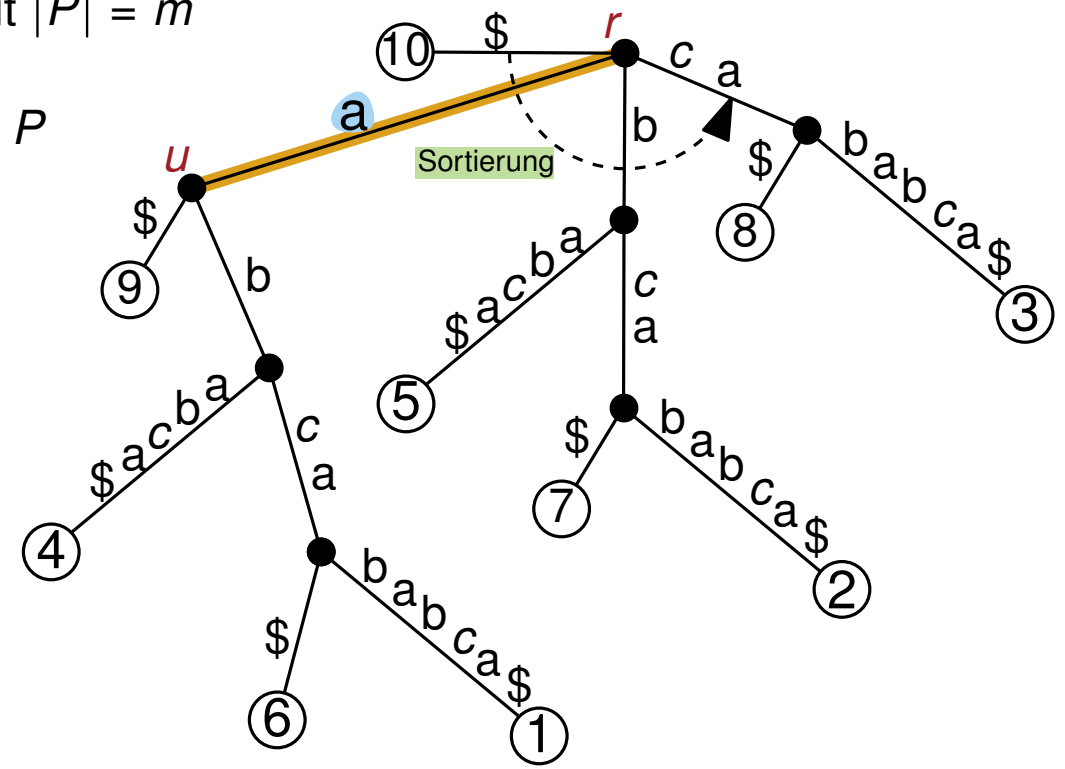
Beispiel: P= a b c a

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.

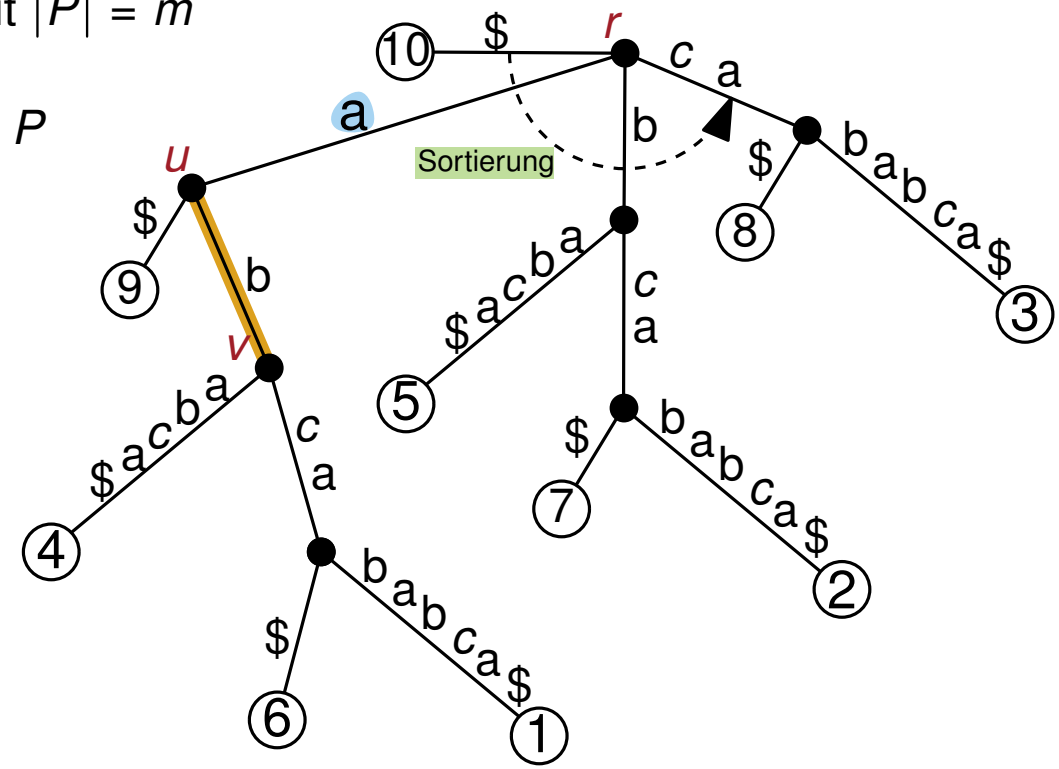
- Wenn Vergleich negativ, dann gebe 0 zurück.
- Wenn $P[1, 4]$ Präfix von B, gebe Anzahl Blätter im Unterbaum S_u aus.
- Wenn $B = P[1, j]$ für ein j mit $1 \leq j \leq m$, betrachte Knoten u und $P[j + 1, m]$.
  - Im Beispiel: Gilt für $j = 1$.

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.

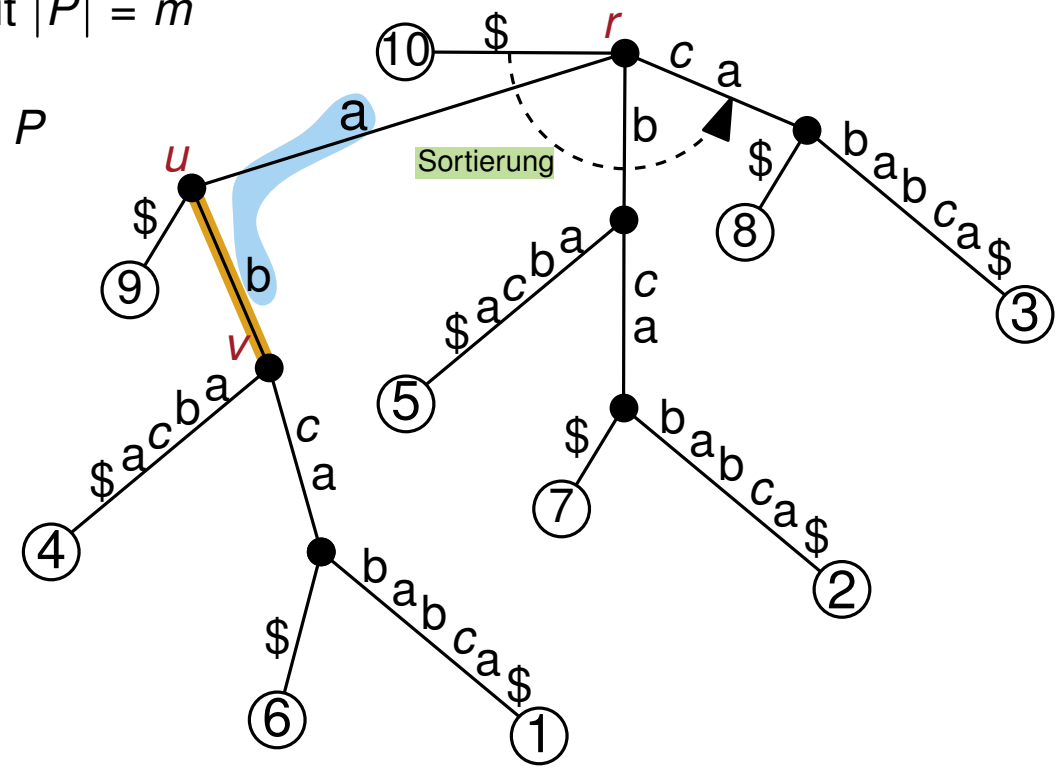
Beispiel: P= a b c a

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

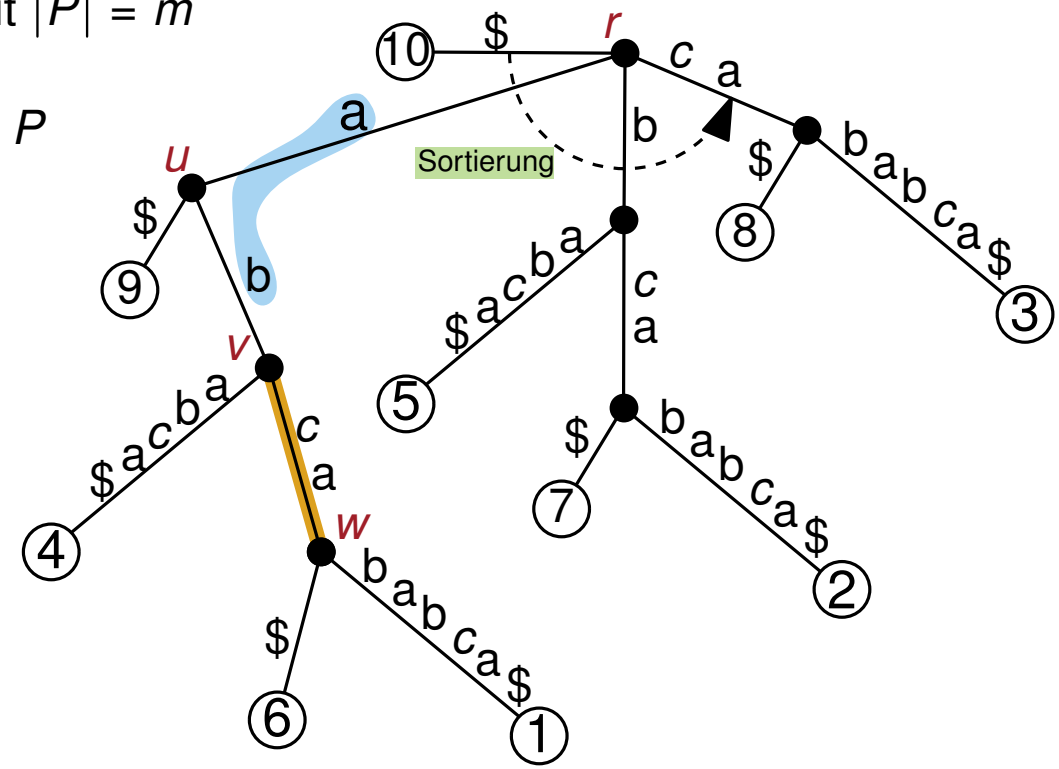
Suchen im Suffixbaum

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.

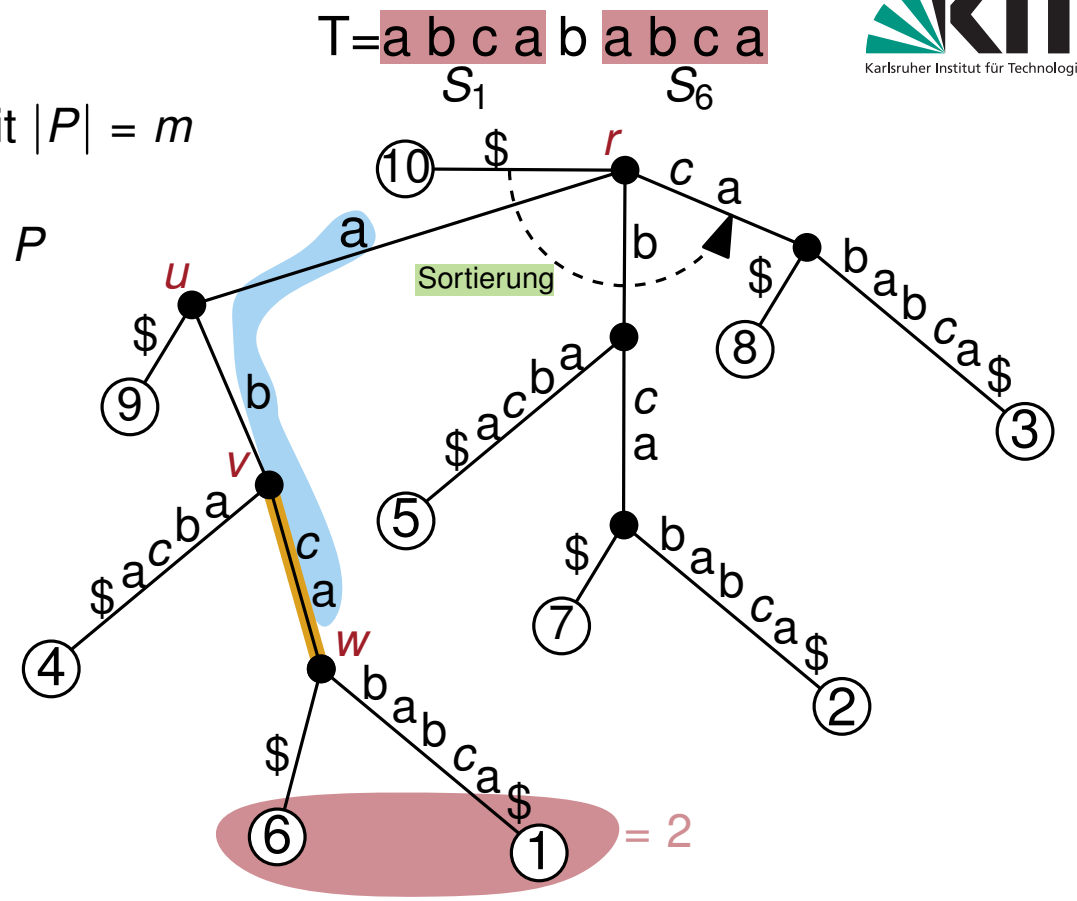
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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.

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** $O(m \cdot \log |\Sigma|)$

- 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.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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 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.

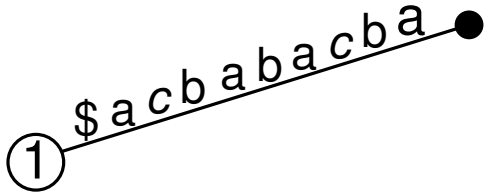
Konstruktion von $N_i + 1$ aus N_i:

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

1. Fall: Wenn Übereinstimmung in der Mitte von e endet, dann spalte e danach in zwei Kanten auf, indem neuer Knoten w eingeführt wird.

2. Fall: Übereinstimmung 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.

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.

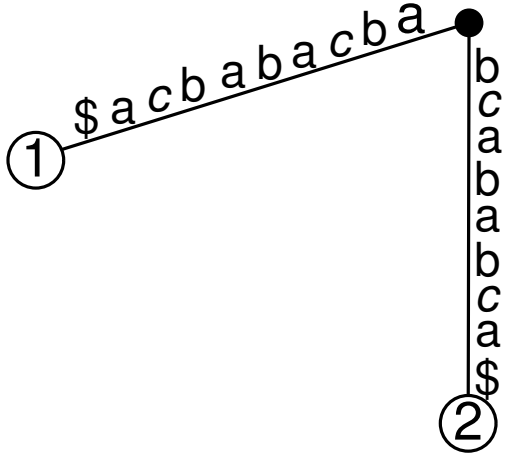
Konstruktion von $N_i + 1$ aus N_i:

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

1. Fall: Wenn Übereinstimmung in der Mitte von e endet, dann spalte e danach in zwei Kanten auf, indem neuer Knoten w eingeführt wird.

2. Fall: Übereinstimmung 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.

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.

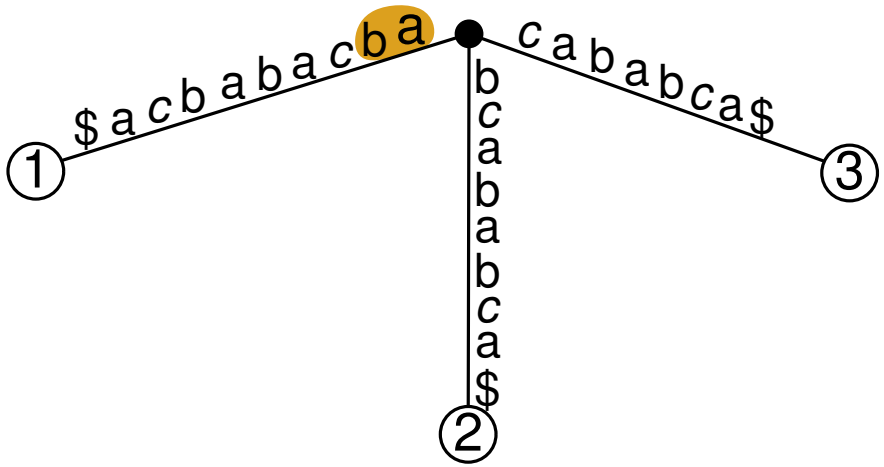
Konstruktion von $N_i + 1$ aus N_i:

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

1. Fall: Wenn Übereinstimmung in der Mitte von e endet, dann spalte e danach in zwei Kanten auf, indem neuer Knoten w eingeführt wird.

2. Fall: Übereinstimmung 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.

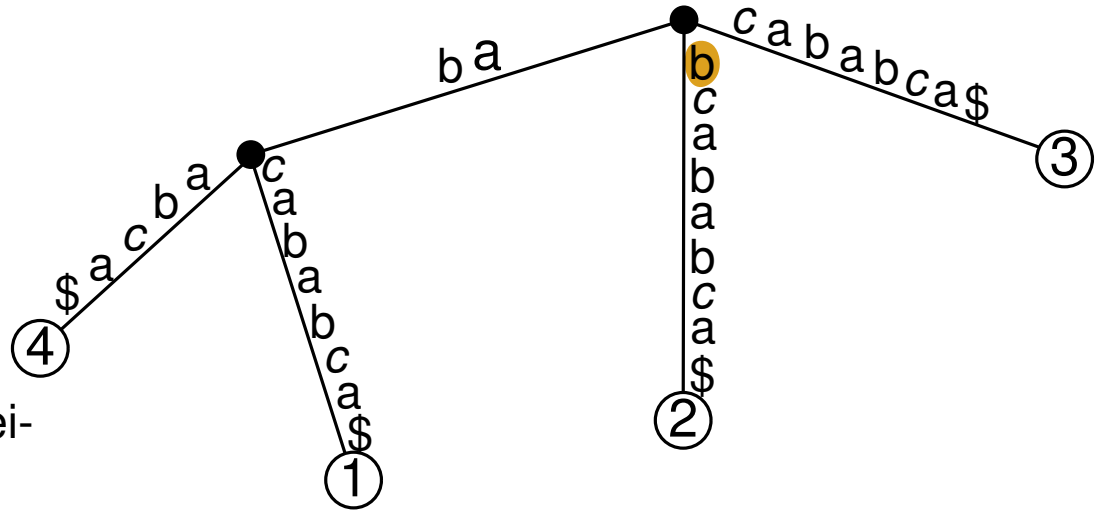
Konstruktion von $N_i + 1$ aus N_i:

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

1. Fall: Wenn Übereinstimmung in der Mitte von e endet, dann spalte e danach in zwei Kanten auf, indem neuer Knoten w eingeführt wird.

2. Fall: Übereinstimmung 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.

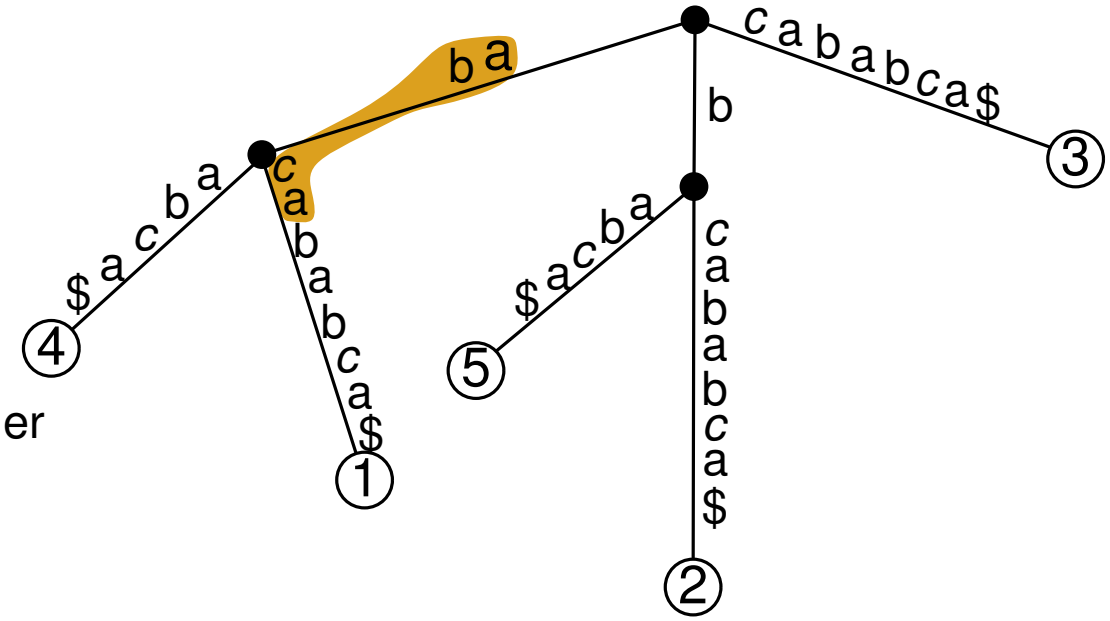
Konstruktion von $N_i + 1$ aus N_i:

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

1. Fall: Wenn Übereinstimmung in der Mitte von e endet, dann spalte e danach in zwei Kanten auf, indem neuer Knoten w eingeführt wird.

2. Fall: Übereinstimmung 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 mit Präfix von S_{i+1} übereinstimmt.

1. Fall: Wenn Übereinstimmung in der Mitte von e endet, dann spalte e danach in zwei Kanten auf, indem neuer Knoten w eingeführt wird.

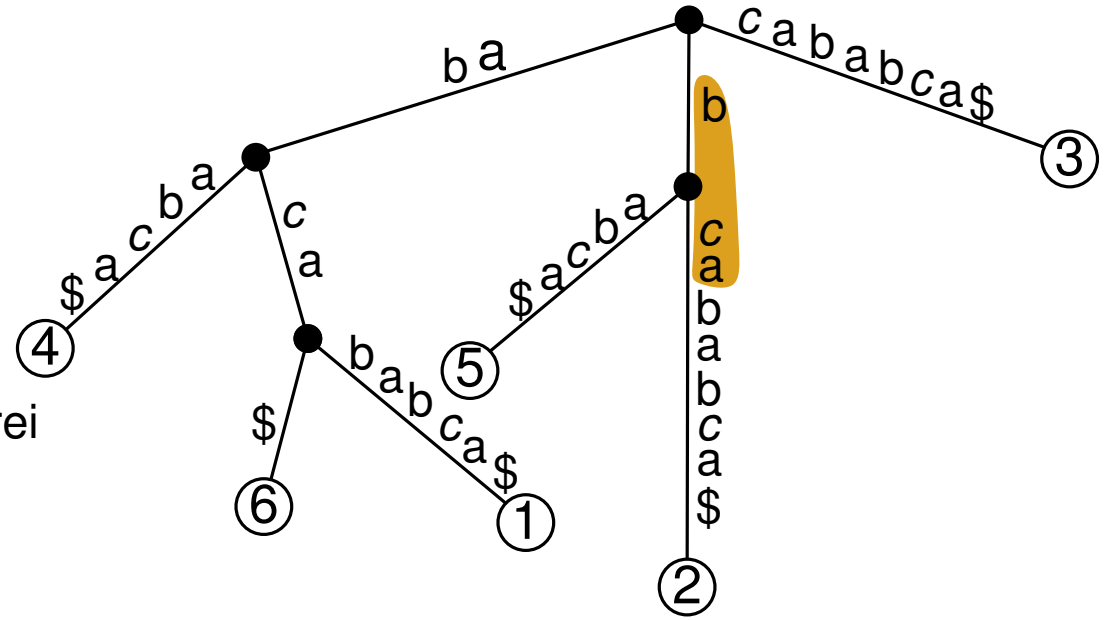
2. Fall: Übereinstimmung endet an Knoten w: Führe neue ausgehende Kante e für w ein.

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:

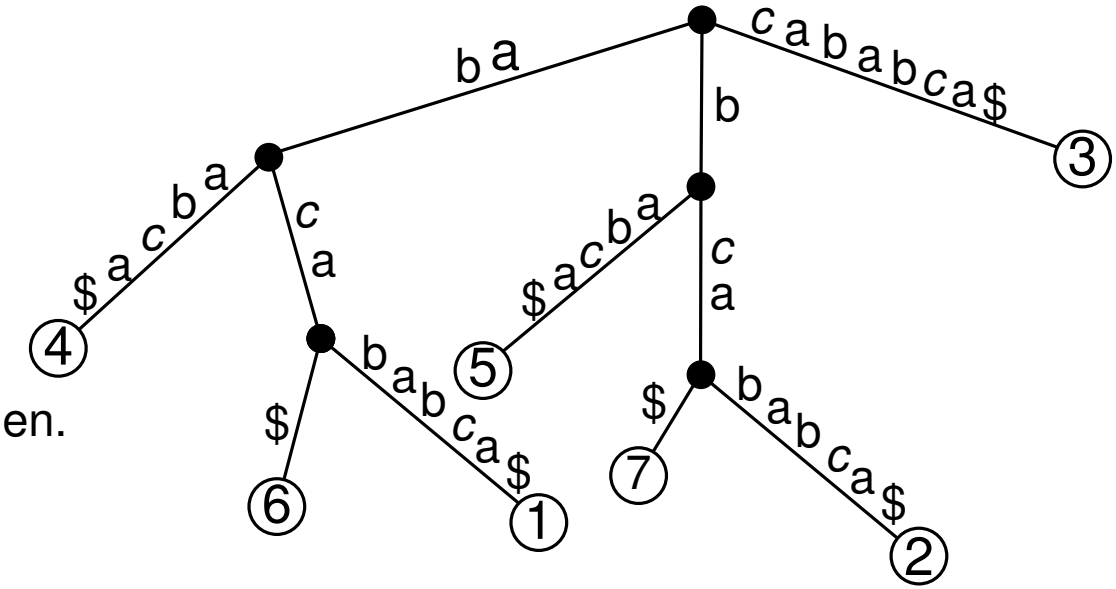
Suche längsten Pfad in N_i, der mit Präfix von S_{i+1} übereinstimmt.

1. Fall: Wenn Übereinstimmung in der Mitte von e endet, dann spalte e danach in zwei Kanten auf, indem neuer Knoten w eingeführt wird.

2. Fall: Übereinstimmung 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 $O(n^2 \log |\Sigma|)$ implementiert werden.

Geht es auch besser?

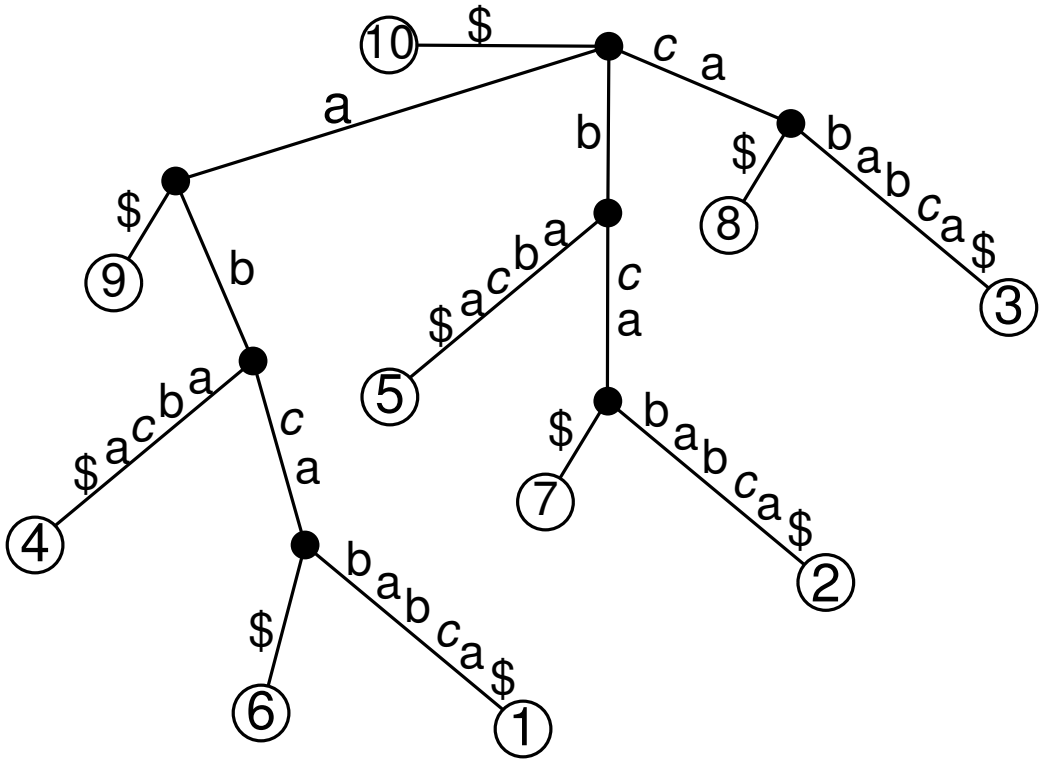
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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 $1 < i \leq n$.

1. Suffix-Array gibt lexikographische Sortierung der Suffixe an.
2. 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

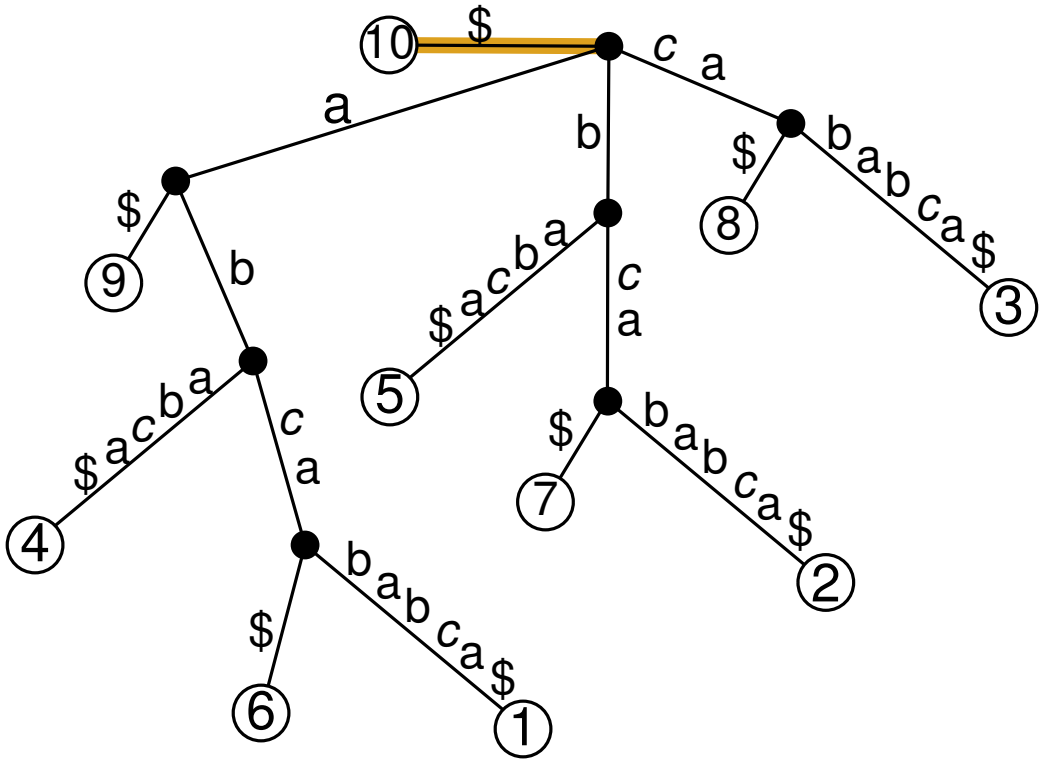
Definition: Ein *Suffix-Array* A eines Textes T ist eine Permutation von $\{1, \ldots, n\}$, sodass $S_{A[i]}$ das i-kleinste Suffix in lexikographischer Ordnung ist: $S_{A[i-1]} < S_{A[i]}$ für alle $1 < i \leq n$.

1. Suffix-Array gibt lexikographische Sortierung der Suffixe an.
2. 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

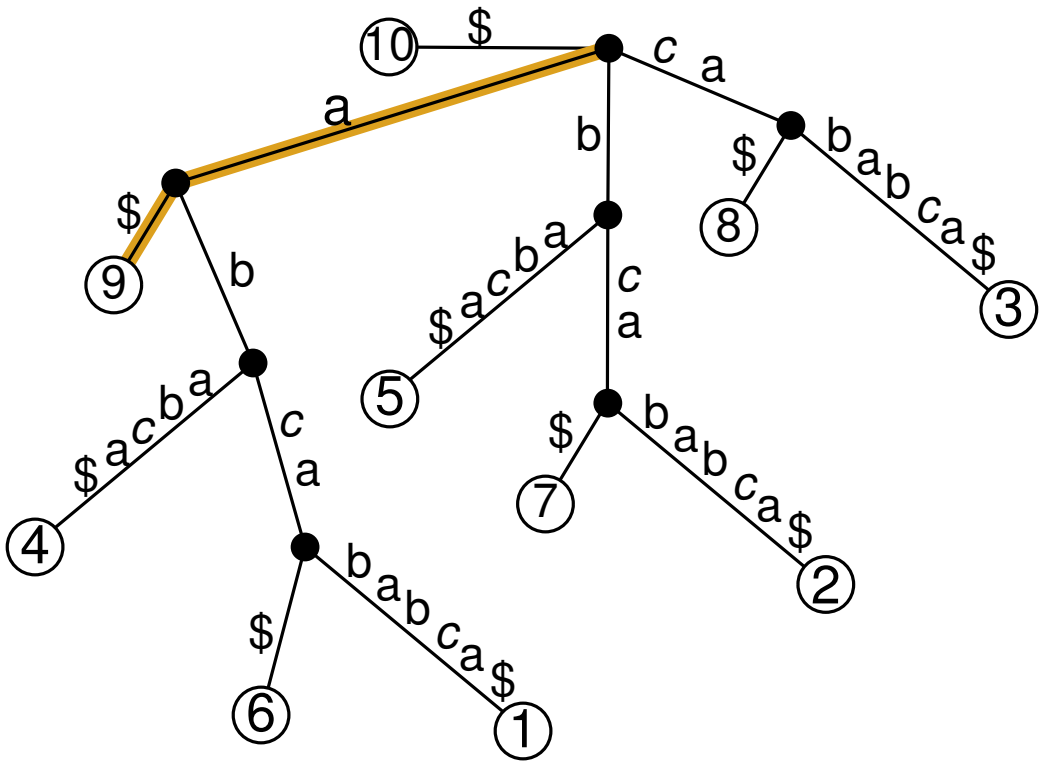
Definition: Ein *Suffix-Array* A eines Textes T ist eine Permutation von $\{1, \ldots, n\}$, sodass $S_{A[i]}$ das i-kleinste Suffix in lexikographischer Ordnung ist: $S_{A[i-1]} < S_{A[i]}$ für alle $1 < i \leq n$.

1. Suffix-Array gibt lexikographische Sortierung der Suffixe an.
2. 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

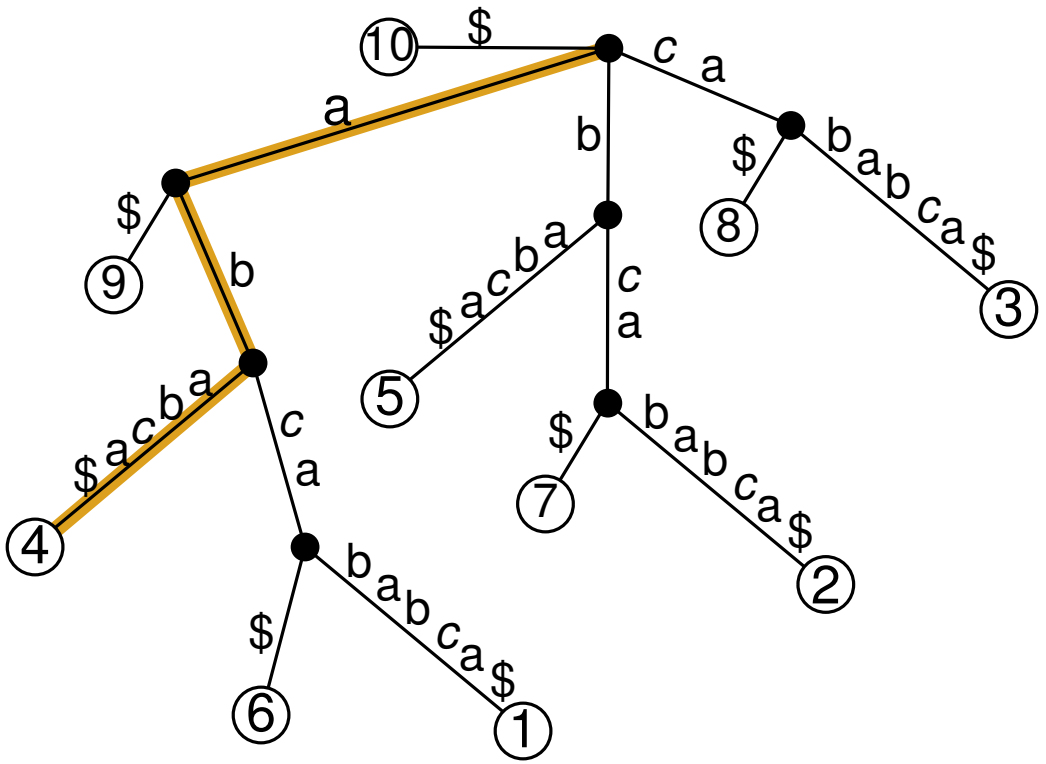
Definition: Ein *Suffix-Array* A eines Textes T ist eine Permutation von $\{1, \ldots, n\}$, sodass $S_{A[i]}$ das i-kleinste Suffix in lexikographischer Ordnung ist: $S_{A[i-1]} < S_{A[i]}$ für alle $1 < i \leq n$.

1. Suffix-Array gibt lexikographische Sortierung der Suffixe an.
2. 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

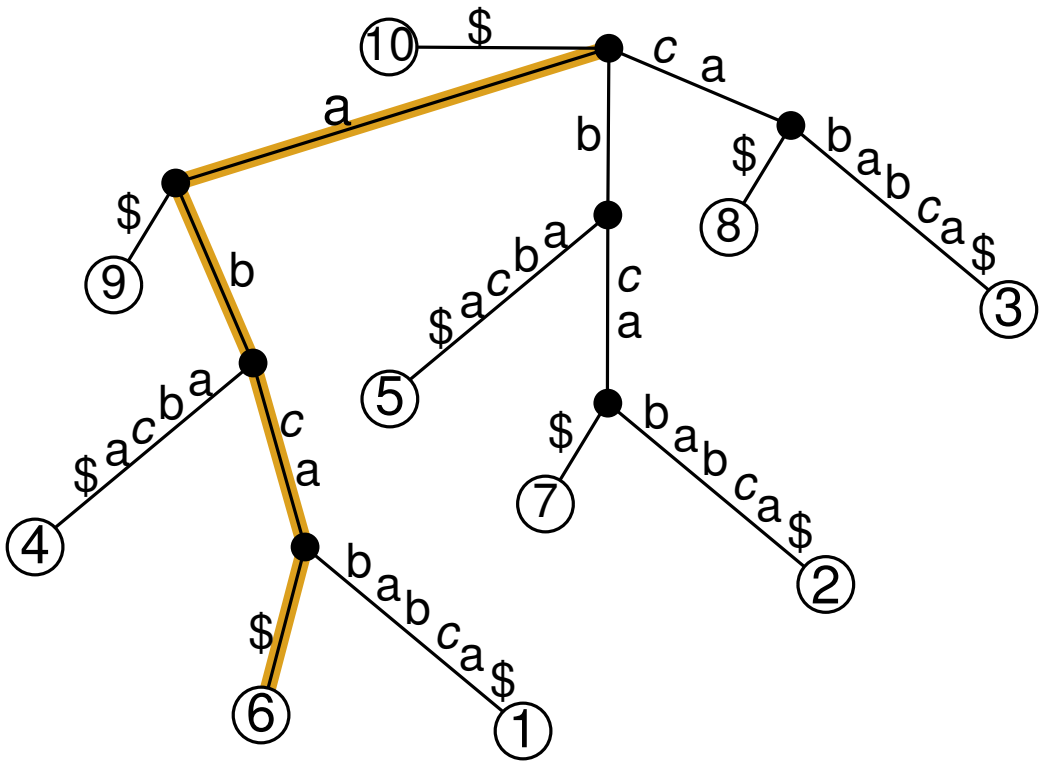
Definition: Ein *Suffix-Array* A eines Textes T ist eine Permutation von $\{1, \ldots, n\}$, sodass $S_{A[i]}$ das i-kleinste Suffix in lexikographischer Ordnung ist: $S_{A[i-1]} < S_{A[i]}$ für alle $1 < i \leq n$.

1. Suffix-Array gibt lexikographische Sortierung der Suffixe an.
2. 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 $1 < i \leq n$.

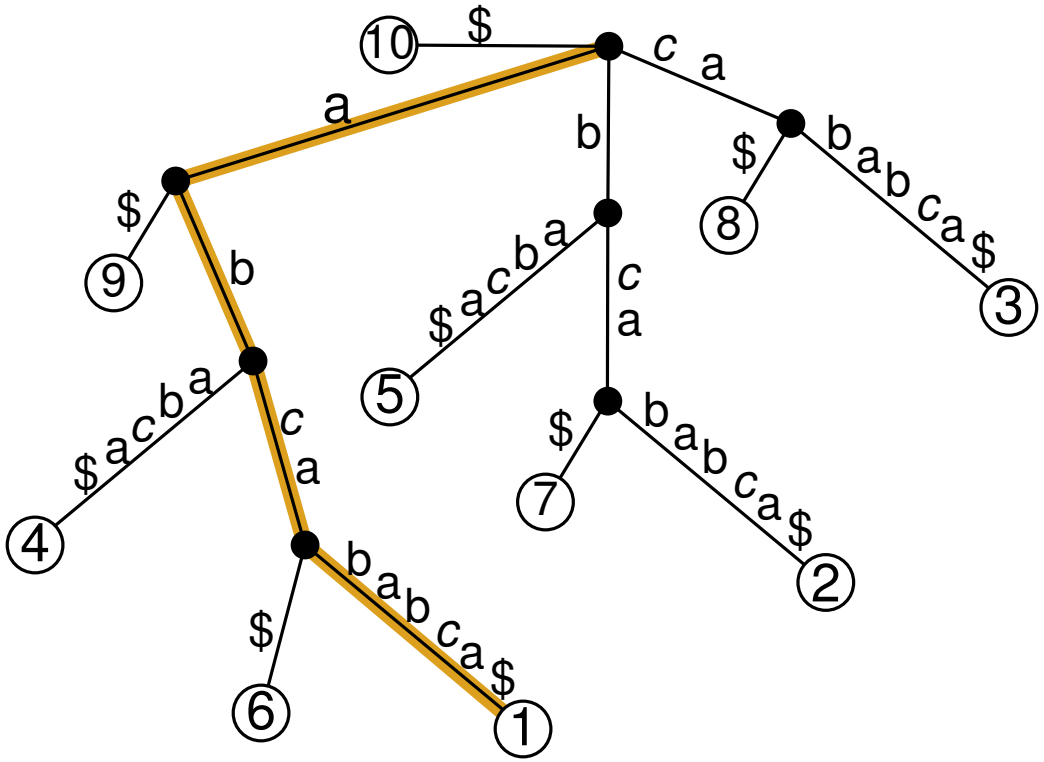
1. Suffix-Array gibt lexikographische Sortierung der Suffixe an.
2. Sei S Suffixbaum von T. A entspricht der Reihenfolge der Blätter, die sich ergibt, wenn Tiefensuche auf S mit lexikographischer Ordnung angewendet wird.

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

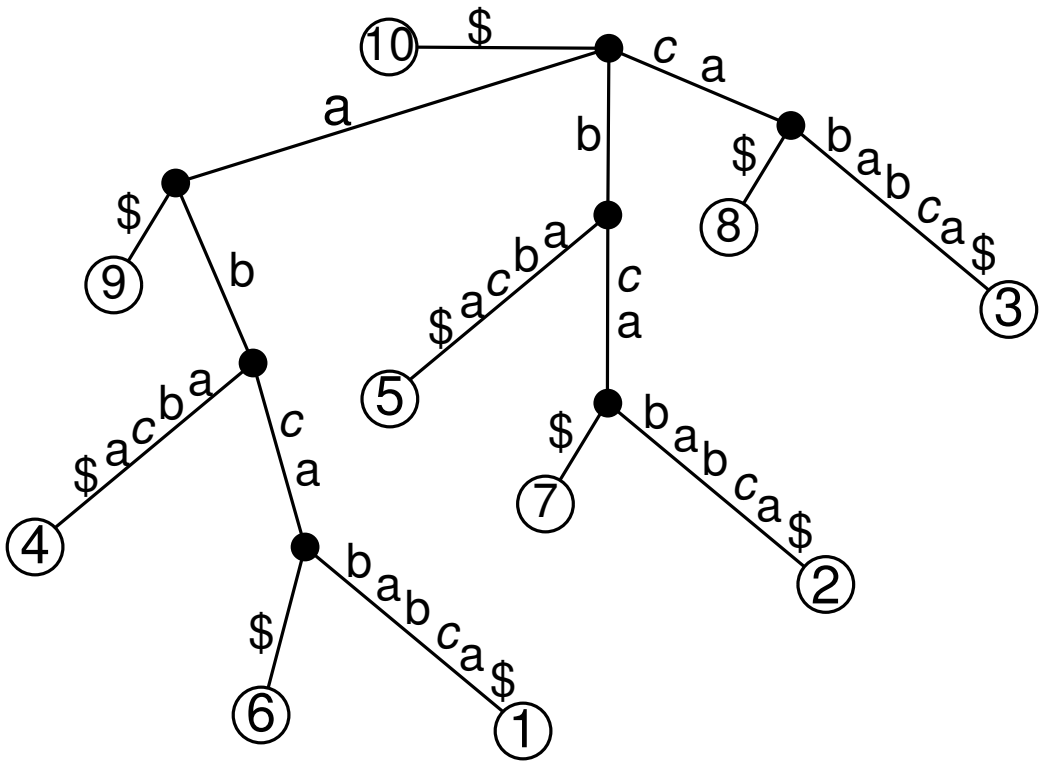
Definition: Ein *Suffix-Array* A eines Textes T ist eine Permutation von $\{1, \ldots, n\}$, sodass $S_{A[i]}$ das i-kleinste Suffix in lexikographischer Ordnung ist: $S_{A[i-1]} < S_{A[i]}$ für alle $1 < i \leq n$.

1. Suffix-Array gibt lexikographische Sortierung der Suffixe an.
2. 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' ← A
solange size(A') > 1 tue
    Bestimme unteren Median i von A'.
    wenn P lexikographisch größer als S_A[i][1, m] dann
        A' ← A'[i + 1, size(A')] (rechtes Teilarray)
    sonst
        A' ← A'[1, i] (linkes Teilarray)
ℓ ← Index von A'[1] in A.
return ℓ
```

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' ← A
solange size(A') > 1 tue
    Bestimme unteren Median i von A'.
    wenn P lexikographisch größer als S_A[i][1, m] dann
        A' ← A'[i + 1, size(A')] (rechtes Teilarray)
    sonst
        A' ← A'[1, i] (linkes Teilarray)
ℓ ← Index von A'[1] in A.
return ℓ
```

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 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' ← A
solange size(A') > 1 tue
    Bestimme unteren Median i von A'.
    wenn P lexikographisch größer als S_A[i][1, m] dann
        A' ← A'[i + 1, size(A')] (rechtes Teilarray)
    sonst
        A' ← A'[1, i] (linkes Teilarray)
ℓ ← Index von A'[1] in A.
return ℓ
```

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 $

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 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' ← A
solange size(A') > 1 tue
    Bestimme unteren Median i von A'.
    wenn P lexikographisch größer als S_A[i][1, m] dann
        A' ← A'[i + 1, size(A')] (rechtes Teilarray)
    sonst
        A' ← A'[1, i] (linkes Teilarray)
ℓ ← Index von A'[1] in A.
return ℓ
```

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 $

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

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 $

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

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

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

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

Konstruktion von Suffix-Arrays

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

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

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

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

$\mathcal{S}$ = Suffixe von T

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

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

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

Konstruktion von Suffix-Arrays

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

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

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

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

$\mathcal{S}$ = Suffixe von T

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

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

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

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

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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

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

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

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

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

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

R_1=[abb][ada][bba][do\$]

R_2=[bba][dab][bad][o\$\$]

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

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

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

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

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

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

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

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

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

R_1=[abb][ada][bba][do\$] R_2=[bba][dab][bad][o\$\$]

Konkatenation von R_1 und R_2.

$$R = R_1 \cdot R_2$$

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

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

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

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

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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

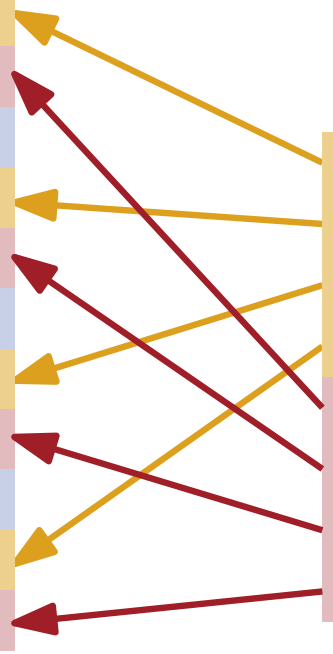
Beobachtung Jedes Suffix von R hat eindeutige Entsprechung in $\mathcal{S}_1 \cup \mathcal{S}_2$.

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

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

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o



R=[abb][ada][bba][do$][bba][dab][bad][o$$]

$\mathcal{S}(R)$=

$S_1(R)$	[abb][ada][bba][do$][bba][dab][bad][o$$]
$S_2(R)$	[ada][bba][do$][bba][dab][bad][o$$]
$S_3(R)$	[bba][do$][bba][dab][bad][o$$]
$S_4(R)$	[do$][bba][dab][bad][o$$]
$S_5(R)$	[bba][dab][bad][o$$]
$S_6(R)$	[dab][bad][o$$]
$S_7(R)$	[bad][o$$]
$S_8(R)$	[o$$]

Erinnerung: t_i erstes Zeichen von S_i.

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

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

R=[abb][ada][bba][do$][bba][dab][bad][o$$]

$\mathcal{S}(R)$=

$S_1(R)$	[abb][ada][bba][do$][bba][dab][bad][o$$]
$S_2(R)$	[ada][bba][do$][bba][dab][bad][o$$]
$S_3(R)$	[bba][do$][bba][dab][bad][o$$]
$S_4(R)$	[do$][bba][dab][bad][o$$]
$S_5(R)$	[bba][dab][bad][o$$]
$S_6(R)$	[dab][bad][o$$]
$S_7(R)$	[bad][o$$]
$S_8(R)$	[o$$]

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

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

R=[abb][ada][bba][do$][bba][dab][bad][o$$]

Wende Radixsort auf Zeichen von R an.
(siehe Übung)

$\mathcal{S}(R)$=

$S_1(R)$	[abb][ada][bba][do$][bba][dab][bad][o$$]
$S_2(R)$	[ada][bba][do$][bba][dab][bad][o$$]
$S_3(R)$	[bba][do$][bba][dab][bad][o$$]
$S_4(R)$	[do$][bba][dab][bad][o$$]
$S_5(R)$	[bba][dab][bad][o$$]
$S_6(R)$	[dab][bad][o$$]
$S_7(R)$	[bad][o$$]
$S_8(R)$	[o$$]

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

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

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

R=[abb][ada][bba][do$][bba][dab][bad][o$$]

Wende Radixsort auf Zeichen von R an.
(siehe Übung)

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

R' = 1 2 4 6 4 5 3 7

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

$\mathcal{S}(R)$=

$S_1(R)$	[abb][ada][bba][do$][bba][dab][bad][o$$]
$S_2(R)$	[ada][bba][do$][bba][dab][bad][o$$]
$S_3(R)$	[bba][do$][bba][dab][bad][o$$]
$S_4(R)$	[do$][bba][dab][bad][o$$]
$S_5(R)$	[bba][dab][bad][o$$]
$S_6(R)$	[dab][bad][o$$]
$S_7(R)$	[bad][o$$]
$S_8(R)$	[o$$]

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

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

R=[abb][ada][bba][do$][bba][dab][bad][o$$]

Wende Radixsort auf Zeichen von R an.
(siehe Übung)

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

$\mathcal{S}(R)$=

$S_1(R)$	[abb][ada][bba][do$][bba][dab][bad][o$$]
$S_2(R)$	[ada][bba][do$][bba][dab][bad][o$$]
$S_3(R)$	[bba][do$][bba][dab][bad][o$$]
$S_4(R)$	[do$][bba][dab][bad][o$$]
$S_5(R)$	[bba][dab][bad][o$$]
$S_6(R)$	[dab][bad][o$$]
$S_7(R)$	[bad][o$$]
$S_8(R)$	[o$$]

R' = 1 2 4 6 4 5 3 7

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

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

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

Suffixe von R'

$S_1(R')$	1 2 4 6 4 5 3 7
$S_2(R')$	2 4 6 4 5 3 7
$S_3(R')$	4 6 4 5 3 7
$S_4(R')$	6 4 5 3 7
$S_5(R')$	4 5 3 7
$S_6(R')$	5 3 7
$S_7(R')$	3 7
$S_8(R')$	7

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

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

R=[abb][ada][bba][do$][bba][dab][bad][o$$]

Wende Radixsort auf Zeichen von R an.
(siehe Übung)

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

$\mathcal{S}(R)$=

$S_1(R)$	[abb][ada][bba][do$][bba][dab][bad][o$$]
$S_2(R)$	[ada][bba][do$][bba][dab][bad][o$$]
$S_3(R)$	[bba][do$][bba][dab][bad][o$$]
$S_4(R)$	[do$][bba][dab][bad][o$$]
$S_5(R)$	[bba][dab][bad][o$$]
$S_6(R)$	[dab][bad][o$$]
$S_7(R)$	[bad][o$$]
$S_8(R)$	[o$$]

R' = 1 2 4 6 4 5 3 7

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

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

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

Suffixe von R'

$S_1(R')$	1 2 4 6 4 5 3 7
$S_2(R')$	2 4 6 4 5 3 7
$S_3(R')$	4 6 4 5 3 7
$S_4(R')$	6 4 5 3 7
$S_5(R')$	4 5 3 7
$S_6(R')$	5 3 7
$S_7(R')$	3 7
$S_8(R')$	7

Erstelle Suffix-Array A' von R':

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

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

$\mathcal{S}(R)=$

$S_1(R)$	[abb][ada][bba][do$][bba][dab][bad][o$$]
$S_2(R)$	[ada][bba][do$][bba][dab][bad][o$$]
$S_3(R)$	[bba][do$][bba][dab][bad][o$$]
$S_4(R)$	[do$][bba][dab][bad][o$$]
$S_5(R)$	[bba][dab][bad][o$$]
$S_6(R)$	[dab][bad][o$$]
$S_7(R)$	[bad][o$$]
$S_8(R)$	[o$$]

Suffixe von R'

$S_1(R')$	1 2 4 6 4 5 3 7
$S_2(R')$	2 4 6 4 5 3 7
$S_3(R')$	4 6 4 5 3 7
$S_4(R')$	6 4 5 3 7
$S_5(R')$	4 5 3 7
$S_6(R')$	5 3 7
$S_7(R')$	3 7
$S_8(R')$	7

Sortierung von $\mathcal{S}(R')$ liefert Sortierung von $\mathcal{S}(R)$ liefert Sortierung von $\mathcal{S}_1 \cup \mathcal{S}_2$

$\downarrow$

Suffix-Array A_{12}

A_{12} =

1	S_1	a b b a d a b b a d o	$S_1(R')$
2	S_4	a d a b b a d o	$S_2(R')$
3	S_8	b a d o	$S_7(R')$
4	S_2	b b a d a b b a d o	$S_5(R')$
5	S_7	b b a d o	$S_3(R')$
6	S_5	d a b b a d o	$S_6(R')$
7	S_{10}	d o	$S_4(R')$
8	S_{11}	o	$S_8(R')$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

$\mathcal{S}(R)=$

$S_1(R)$	[abb][ada][bba][do$][bba][dab][bad][o$$]
$S_2(R)$	[ada][bba][do$][bba][dab][bad][o$$]
$S_3(R)$	[bba][do$][bba][dab][bad][o$$]
$S_4(R)$	[do$][bba][dab][bad][o$$]
$S_5(R)$	[bba][dab][bad][o$$]
$S_6(R)$	[dab][bad][o$$]
$S_7(R)$	[bad][o$$]
$S_8(R)$	[o$$]

Suffixe von R'

$S_1(R')$	1 2 4 6 4 5 3 7
$S_2(R')$	2 4 6 4 5 3 7
$S_3(R')$	4 6 4 5 3 7
$S_4(R')$	6 4 5 3 7
$S_5(R')$	4 5 3 7
$S_6(R')$	5 3 7
$S_7(R')$	3 7
$S_8(R')$	7

Sortierung von $\mathcal{S}(R')$ liefert Sortierung von $\mathcal{S}(R)$ liefert Sortierung von $\mathcal{S}_1 \cup \mathcal{S}_2$

$\downarrow$

Suffix-Array A_{12}

$A_{12} =$

1	S_1	a b b a d a b b a d o	$S_1(R')$
2	S_4	a d a b b a d o	$S_2(R')$
3	S_8	b a d o	$S_7(R')$
4	S_2	b b a d a b b a d o	$S_5(R')$
5	S_7	b b a d o	$S_3(R')$
6	S_5	d a b b a d o	$S_6(R')$
7	S_{10}	d o	$S_4(R')$
8	S_{11}	o	$S_8(R')$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

$\mathcal{S}(R)=$

$S_1(R)$	[abb][ada][bba][do$][bba][dab][bad][o$$]
$S_2(R)$	[ada][bba][do$][bba][dab][bad][o$$]
$S_3(R)$	[bba][do$][bba][dab][bad][o$$]
$S_4(R)$	[do$][bba][dab][bad][o$$]
$S_5(R)$	[bba][dab][bad][o$$]
$S_6(R)$	[dab][bad][o$$]
$S_7(R)$	[bad][o$$]
$S_8(R)$	[o$$]

Suffixe von R'

$S_1(R')$	1 2 4 6 4 5 3 7
$S_2(R')$	2 4 6 4 5 3 7
$S_3(R')$	4 6 4 5 3 7
$S_4(R')$	6 4 5 3 7
$S_5(R')$	4 5 3 7
$S_6(R')$	5 3 7
$S_7(R')$	3 7
$S_8(R')$	7

Sortierung von $\mathcal{S}(R')$ liefert Sortierung von $\mathcal{S}(R)$ liefert Sortierung von $\mathcal{S}_1 \cup \mathcal{S}_2$

$\downarrow$

Suffix-Array A_{12}

$A_{12} =$

1	S_1	a b b a d a b b a d o	$S_1(R')$
2	S_4	a d a b b a d o	$S_2(R')$
3	S_8	b a d o	$S_7(R')$
4	S_2	b b a d a b b a d o	$S_5(R')$
5	S_7	b b a d o	$S_3(R')$
6	S_5	d a b b a d o	$S_6(R')$
7	S_{10}	d o	$S_4(R')$
8	S_{11}	o	$S_8(R')$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

$\mathcal{S}(R)=$

$S_1(R)$	[abb][ada][bba][do$][bba][dab][bad][o$$]
$S_2(R)$	[ada][bba][do$][bba][dab][bad][o$$]
$S_3(R)$	[bba][do$][bba][dab][bad][o$$]
$S_4(R)$	[do$][bba][dab][bad][o$$]
$S_5(R)$	[bba][dab][bad][o$$]
$S_6(R)$	[dab][bad][o$$]
$S_7(R)$	[bad][o$$]
$S_8(R)$	[o$$]

Suffixe von R'

$S_1(R')$	1 2 4 6 4 5 3 7
$S_2(R')$	2 4 6 4 5 3 7
$S_3(R')$	4 6 4 5 3 7
$S_4(R')$	6 4 5 3 7
$S_5(R')$	4 5 3 7
$S_6(R')$	5 3 7
$S_7(R')$	3 7
$S_8(R')$	7

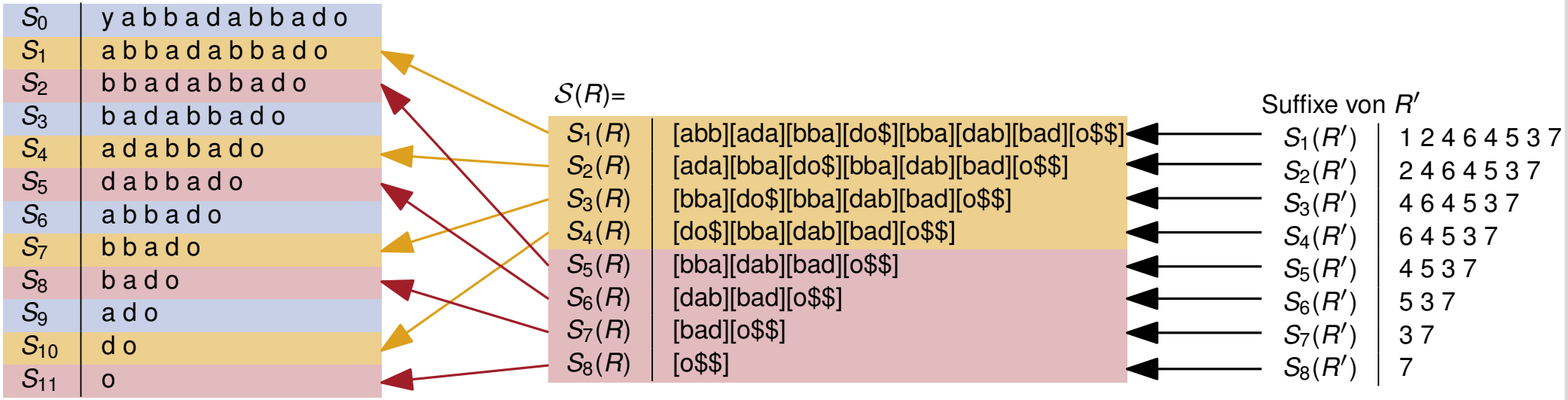
Sortierung von $\mathcal{S}(R')$ → liefert → Sortierung von $\mathcal{S}(R)$ → liefert → Sortierung von $\mathcal{S}_1 \cup \mathcal{S}_2$ → Suffix-Array A_{12}

$A_{12} =$

1	S_1	a b b a d a b b a d o	$S_1(R')$
2	S_4	a d a b b a d o	$S_2(R')$
3	S_8	b a d o	$S_7(R')$
4	S_2	b b a d a b b a d o	$S_5(R')$
5	S_7	b b a d o	$S_3(R')$
6	S_5	d a b b a d o	$S_6(R')$
7	S_{10}	d o	$S_4(R')$
8	S_{11}	o	$S_8(R')$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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



Sortierung von $\mathcal{S}(R')$ liefert Sortierung von $\mathcal{S}(R)$ liefert Sortierung von $\mathcal{S}_1 \cup \mathcal{S}_2$

↓

Suffix-Array A_{12}

Laufzeit:

$O(n)$ Zeit für Konstruktion von R.

$O(n)$ Zeit für Konstruktion von R'.

Laufzeit $T_1(n)$ für 1. Schritt $T_1(n) = O(n) + T(\frac{2}{3}n)$

wobei $T(n)$ = Laufzeit für Konstruktion eines Suffix-Arrays

Konstruktion von Suffix-Arrays

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

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

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

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

$\mathcal{S}$ = Suffixe von T

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

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

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

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

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

2. Schritt: Konstruktion von A_0

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

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

Beobachtung gibt Definition der lexikographischen Ordnung wider.

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

Erinnerung:

t_i = erstes Zeichen von S_i

2. Schritt: Konstruktion von A_0

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

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

Beobachtung gibt Definition der lexikographischen Ordnung wider.

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

Aus Schritt 1 folgt A_{12}:

1	S_1	a b b a d a b b a d o
2	S_4	a d a b b a d o
3	S_8	b a d o
4	S_2	b b a d a b b a d o
5	S_7	b b a d o
6	S_5	d a b b a d o
7	S_{10}	d o
8	S_{11}	o

Erinnerung:

t_i = erstes Zeichen von S_i

2. Schritt: Konstruktion von A_0

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

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

Beobachtung gibt Definition der lexikographischen Ordnung wider.

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

Erinnerung:

t_i = erstes Zeichen von S_i

Aus Schritt 1 folgt A_{12}:

1	S_1	a b b a d a b b a d o
2	S_4	a d a b b a d o
3	S_8	b a d o
4	S_2	b b a d a b b a d o
5	S_7	b b a d o
6	S_5	d a b b a d o
7	S_{10}	d o
8	S_{11}	o

$S_6 < S_9$ weil $S_7 < S_{10}$
$S_9 < S_3$ weil $a < b$
$S_3 < S_0$ weil $b < y$

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

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Konstruktion von Suffix-Arrays

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

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

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

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

$\mathcal{S}$ = Suffixe von T

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

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

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

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

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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

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

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

Erinnerung:

t_i = erstes Zeichen von S_i

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

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

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

Aus Schritt 2 folgt:

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

Aus Schritt 2 folgt:

$S_6 < S_9 < S_3 < S_0$

Vermengen wie bei Merge-Sort:

Erinnerung:

t_i = erstes Zeichen von S_i

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

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

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

Aus Schritt 2 folgt:

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

Aus Schritt 2 folgt:

$S_6 < S_9 < S_3 < S_0$

Vermengen wie bei Merge-Sort:

S_1

Erinnerung:

t_i = erstes Zeichen von S_i

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

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

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

Aus Schritt 2 folgt:

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

Aus Schritt 2 folgt:

$S_6 < S_9 < S_3 < S_0$

Vermengen wie bei Merge-Sort:

$S_1 < S_6$

Erinnerung:

t_i = erstes Zeichen von S_i

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

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

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

Aus Schritt 2 folgt:

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

Aus Schritt 2 folgt:

$S_6 < S_9 < S_3 < S_0$

Vermengen wie bei Merge-Sort:

$S_1 < S_6 < S_4$

Erinnerung:

t_i = erstes Zeichen von S_i

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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

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

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

Aus Schritt 2 folgt:

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

Aus Schritt 2 folgt:

$S_6 < S_9 < S_3 < S_0$

Vermengen wie bei Merge-Sort:

$S_1 < S_6 < S_4 < S_9$

Erinnerung:

t_i = erstes Zeichen von S_i

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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

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

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

Aus Schritt 2 folgt:

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

Aus Schritt 2 folgt:

$S_6 < S_9 < S_3 < S_0$

Vermengen wie bei Merge-Sort:

$S_1 < S_6 < S_4 < S_9 < S_3$

Erinnerung:

t_i = erstes Zeichen von S_i

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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

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

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

S_0	y a b b a d a b b a d o
S_1	a b b a d a b b a d o
S_2	b b a d a b b a d o
S_3	b a d a b b a d o
S_4	a d a b b a d o
S_5	d a b b a d o
S_6	a b b a d o
S_7	b b a d o
S_8	b a d o
S_9	a d o
S_{10}	d o
S_{11}	o

Aus Schritt 2 folgt:

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

Aus Schritt 2 folgt:

$S_6 < S_9 < S_3 < S_0$

Vermengen wie bei Merge-Sort:

$S_1 < S_6 < S_4 < S_9 < S_3 <$

$S_8 < S_2 < S_7 < S_5 < S_{10} < S_{11} < S_0$

Erinnerung:

t_i = erstes Zeichen von S_i

Zusammenfassung: Suffix-Arrays

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

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

Laufzeit:

$$T(n) = \begin{cases} O(1), & \text{falls n=O(1)} \\ O(n) + T(\frac{2}{3}n), & \text{sonst} \end{cases}$$

Kann mithilfe des Master-Theorems aufgelöst werden.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Konstruktion von Suffixbäumen

Beobachtung: 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

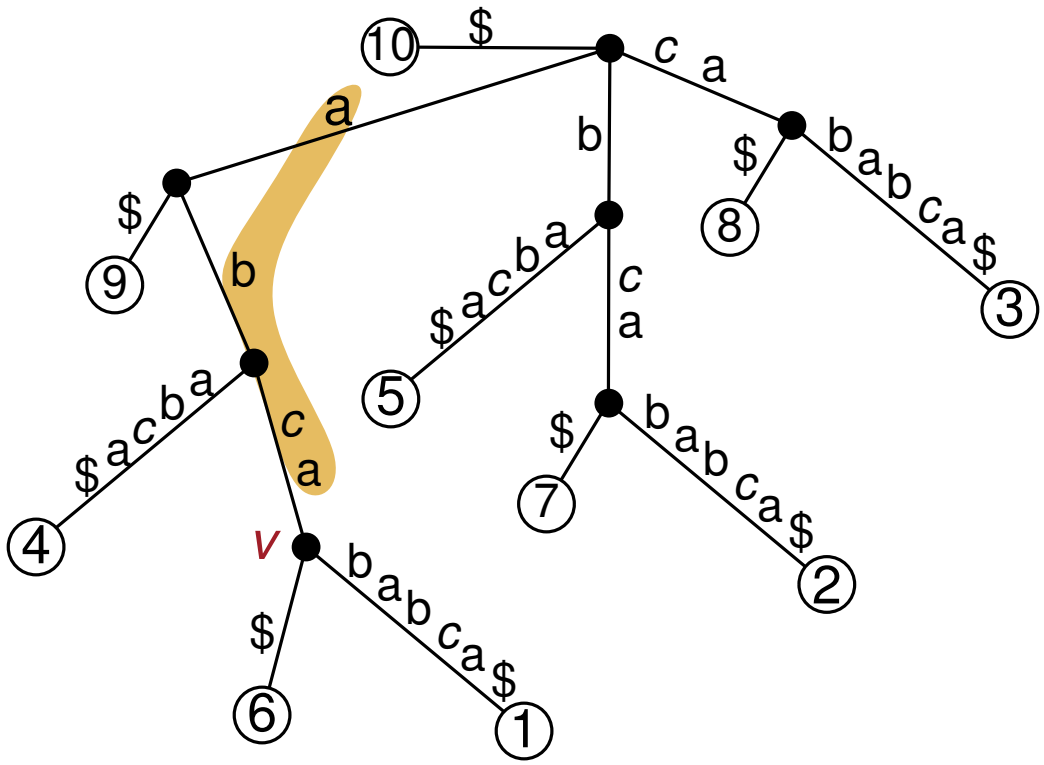
Definition 11: Gegeben ein Text T der Länge n und dessen Suffix-Array A. Ein *LCP-Array* L ist ein Array der Größe n, sodass

- $L[1] = 0$
- $L[i]$ = 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 $
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 $



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

→ L gibt für zwei Suffixe an, an welcher Stelle sich Baum aufgespaltet.

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.

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 $d(v) \leq L[i+1]$.

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

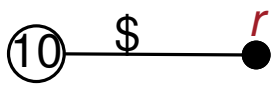
T= a b c a b a b c a $

$A =$	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 $

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.

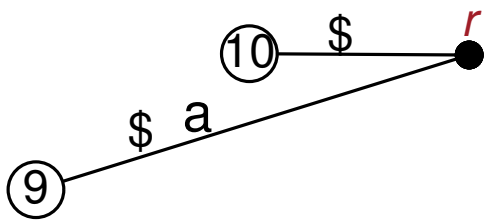
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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 $d(v) \leq L[i+1]$.

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.

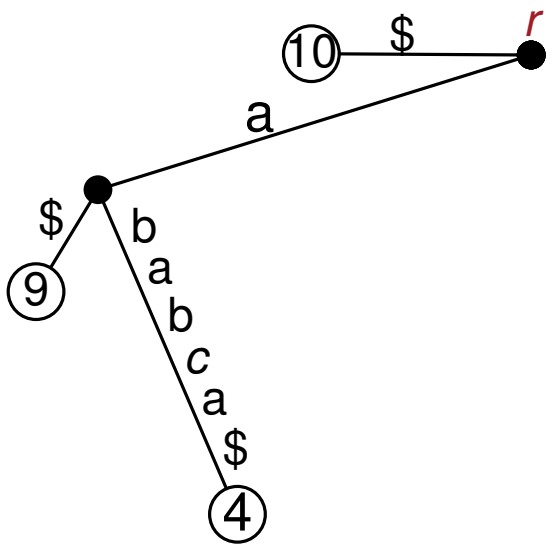
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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.

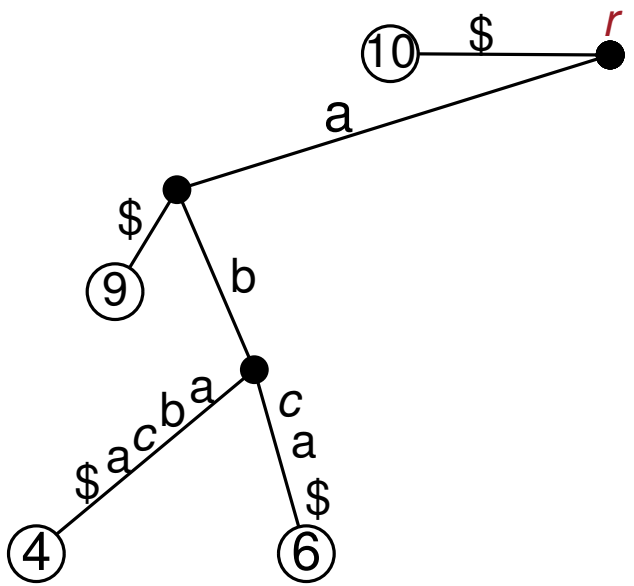
1. Entferne (v, w).
2. Füge neuen Knoten y mit Kante (v, y) ein. Beschriftung von (v, y): $T[A[i] + d(v), A[i] + L[i+1] - 1]$
3. Füge (y, w) ein mit Beschriftung $T[A[i] + L[i+1], A[i] + d(w) - 1]$.
4. Füge Blatt x mit Beschriftung $A[i+1]$ ein und Kante (y, x) mit Beschriftung $T[A[i+1] + L[i+1], n]$.

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 $T[A[i+1] + L[i+1], n]$.

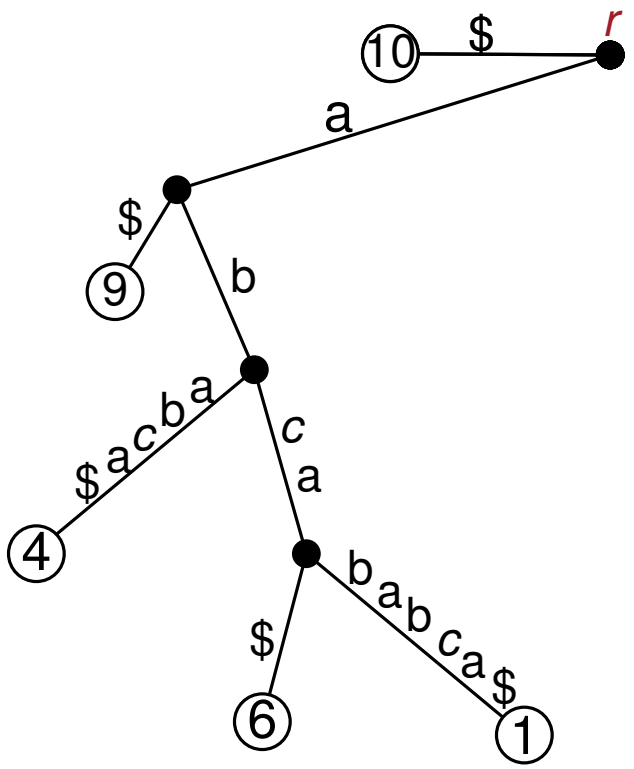
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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 $T[A[i+1] + L[i+1], n]$.

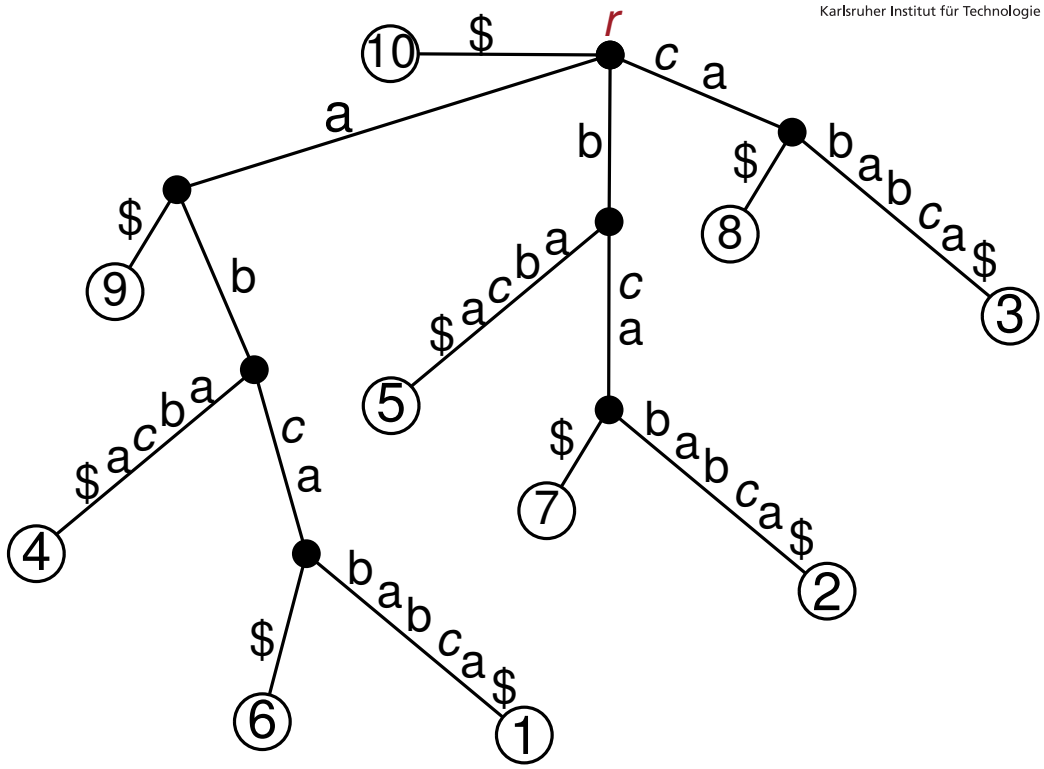
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

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.

Ohne Beweis: LCP-Array kann in $O(n)$ Zeit konstruiert werden.