Algorithmische Bioinformatik

Suffixbäume

Ulf Leser

Wissensmanagement in der Bioinformatik

Ziele

- Perspektivenwechsel:
  Von Online zu Offline-Stringmatching
- Verständnis von Suffix-Bäumen als Datenstruktur
- Problem bei der Konstruktion erkennen

Inhalt dieser Vorlesung

- Suffixbäume
  - Motivation
  - Aufbau
  - Naive Konstruktion
- Verwendung von Suffixbäumen

Problemstellung

- Bisherige Algorithmen
  - Ein Template T (m) und ein oder mehrere Pattern P (n)
  - Prinzip: Preprocessing von P in O(n), dann Suche in O(m)
- Jetzt: Gegeben eine lange Zeichenkette T
  - Z.B. Komplettes Genom des Menschen
- Benutzer schicken ständig neue Pattern P
  - Z.B. Gene anderer Spezies
- T sollte vorverarbeitet werden
  - Kosten amortisieren sich über viele Suchen
  - Zählen nicht für die Suche eines Pattern
- Eine Lösung: Suffixbäume

Motivation: Datenbanksuchen

- Solche Suchen sind ein Hauptthema der Bioinformatik
  - I.d.R. approximativ
  - Schnelle Algorithmen für approx. Suchen benutzen (fast) immer ein exaktes Suchverfahren zum Finden aussichtsreicher Regionen
- Suffixbäume sind sehr schnell für exakte Suchen
  - Aber: Speicherplatz, Sekundärspeicherverhalten
  - Alternative Baumstrukturen: Suffix-Arrays, Enhanced Suffix-Arrays, Burrows-Wheeler-Transformation, …
  - Andere Alternative: Hashing
- Suffixbäume haben viele weitere Anwendungen
  - Suche nach längsten identischen Subsequenzen
  - Suche nach längsten Repeats
  - …

Suffixbäume

- Definition
  Der *Suffixbaum T für einen String S mit |S|=m ist ein Baum mit*
  - *T hat eine Wurzel und m Blätter, markiert mit 1,...,m*
  - *Jede Kante E ist mit einem Substring label(E)≠∅ von S beschriftet*
  - *Jeder innere Knoten k hat mindestens 2 Kinder*
  - *Alle Label der Kanten von einem Knoten k aus beginnen mit unterschiedlichen Zeichen*
  - *Sei $(k_1,k_2,...,k_n)$ ein Pfad von der Wurzel zu einem Blatt mit Markierung i. Dann ist die Konkatenation der Label der Kanten auf dem Pfad gleich S[i..m]*

Beispiel 1

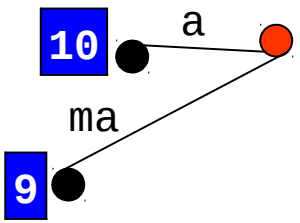
1234567890

- S= **BANANARAMA**

Beispiel 1

1234567890

- S= **BANANARAMA**

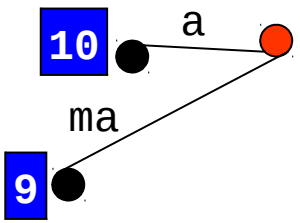


- Problem: Wohin kommt „AMA“?
  - Verlängerung von „a“ verboten – 10 sonst kein Blatt
  - Neue Kante „ama“ verboten – zwei Pfade aus der Wurzel würden sonst mit gleichem Zeichen beginnen
  - ➢ Es gibt keinen Suffixbaum für BANANARAMA
  - ➢ Problem tritt auf, sobald ein Suffix Präfix eines anderen Suffix ist
    - Also dauernd

Beispiel 1

1234567890

- S= **BANANARAMA**

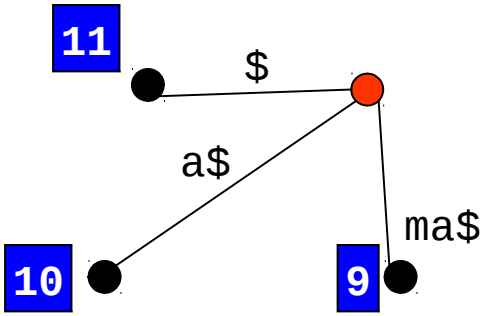


- Problem: Wohin kommt „AMA“?
  - Verlängerung von „a“ verboten – 10 sonst kein Blatt
  - Neue Kante „ama“ verboten – zwei Pfade aus der Wurzel würden sonst mit gleichem Zeichen beginnen
  - ➢ Es gibt keinen Suffixbaum für BANANARAMA
  - ➢ Problem tritt auf, sobald ein Suffix Präfix eines anderen Suffix ist
    - Also dauernd
- Trick: Wir betrachten „BANANARAMA$“
  - „$“ nicht Teil des Alphabets von S

Beispiel 2

12345678901

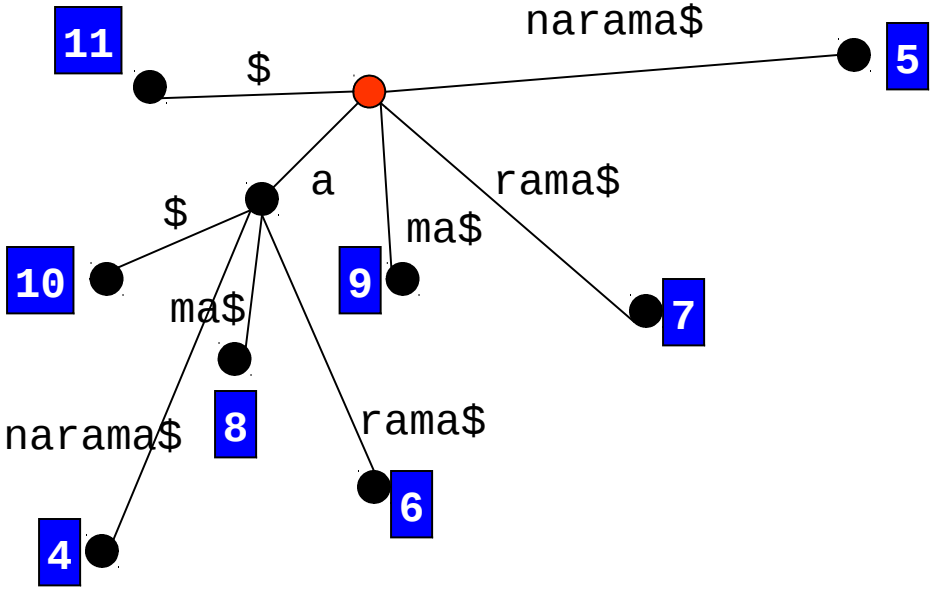
- S= **BANANARAMA$**



Beispiel 2

12345678901

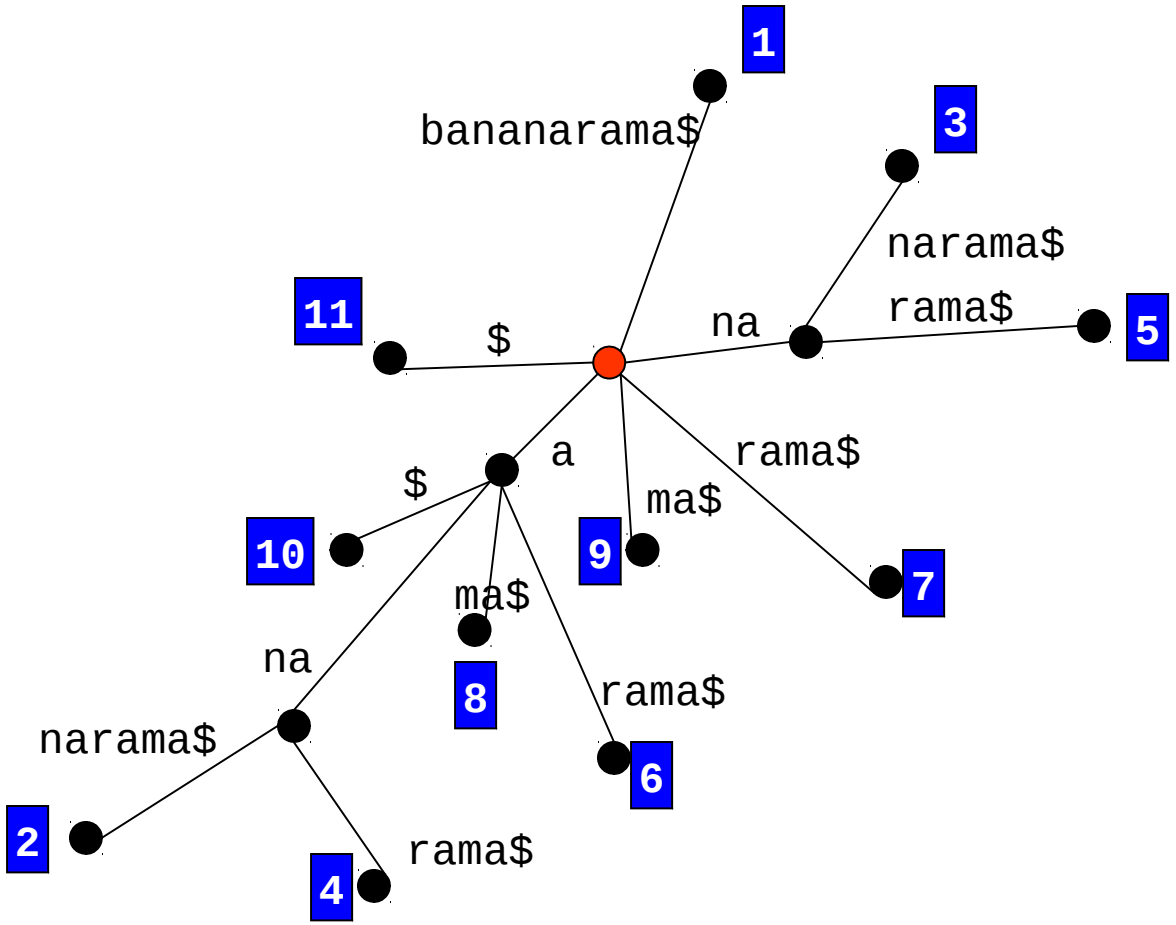
- S= **BANANARAMA$**



Beispiel 3

12345678901

- S= **BANANARAMA$**



Eigenschaften von Suffixbäumen

- Zu jedem String (plus $) gibt es genau einen Suffixbaum
- Jeder Pfad von Wurzel zu einem Blatt ist eindeutig
- Jede Verzweigung an einem inneren Knoten ist eindeutig
- Gleiche Substrings können an mehreren Kanten stehen
- Suffixbäume und Keyword-Trees
  - Betrachte alle Suffixe von S als Pattern
  - Konstruiere den Keyword-Tree
  - Verschmelze alle Knoten auf einem Pfad ohne Abzweigungen zu einer Kante
  - Dann haben wir einen Suffixbaum für S
  - Komplexität?

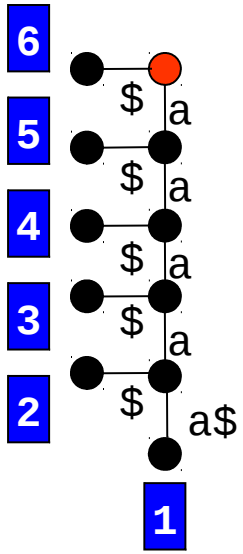
Weitere Beispiele

- S= aaaaa$

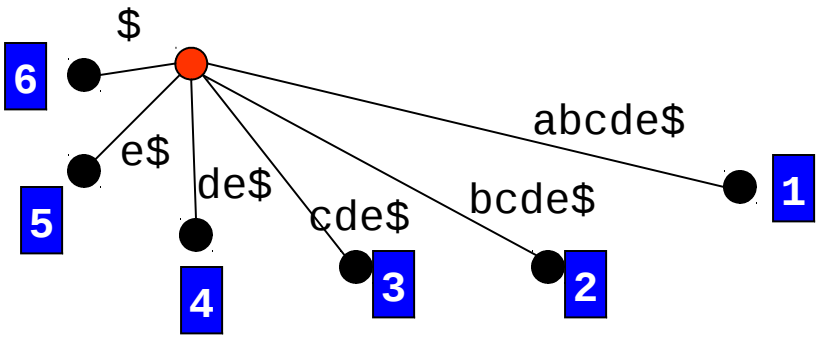
- S= abcde$

Weitere Beispiele

- S= aaaaa$



- S= abcde$



Inhalt dieser Vorlesung

- Suffixbäume
  - Motivation
  - Aufbau
  - Naive Konstruktion
- Verwendung von Suffixbäumen

Definitionen

- Definition
  Sei T der Suffixbaum für String S+“$“
  - *Sei p ein Pfad in T von root(T) zu einem Knoten k. Dann ist label(p) die Konkatenation der Label der Kanten auf dem Pfad p*
  - *Sei k ein Knoten von T und p der Pfad zu k. Dann ist label(k) = label(p)*
  - *Sei k ein Knoten von T. Dann ist depth(k)=|label(k)|*

Naive Konstruktion von Suffixbäumen

- Gesucht: Suffixbaum T für String S
- Start: Bilde Baum T_0 mit Wurzelknoten und einer Kante mit Label „S$“ zu einem Blatt mit Markierung 1
- Konstruiere T_{i+1} aus T_i wie folgt
  - Betrachte das Suffix S_{i+1} = S[i+1..]$
  - Matche S_{i+1} in T_i so weit wie möglich
  - Schließlich muss es einen Mismatch geben
    - Alle bisher eingefügten Suffixe sind länger als S_{i+1}, also wird $ nie mit $ matchen
    - $ kommt sonst nicht in S vor
  - Folgendes kann passieren ...

Naive Konstruktion von Suffixbäumen 2

- Konstruiere T_{i+1} aus T_i wie folgt ...
  1. S_{i+1} matched bis auf \$; Mismatch auf einer Kante n an Position j
     - Füge in n an Position j einen neuen Knoten k ein
     - Erzeuge Kante von k zu neuen Blatt k‘; beschrifte die Kante mit „\$“
     - Markiere k‘ mit i+1
  2. S_{i+1} matched bis auf \$; Mismatch am Ende einer Kante n
     - Sei k der Zielknoten von n
     - Erzeuge Kante von k zu neuen Blatt k‘; beschrifte die Kante mit „\$“
     - Markiere k‘ mit i+1
  3. Mismatch vor \$ auf einer Kante n an Position j des Labels; der Mismatch in S_{i+1} sei an Position j‘<$|S_{i+1}|$
     - Füge in n an Position j einen neuen Knoten k ein
     - Erzeuge Kante von k zu neuen Blatt k‘; beschrifte Kante mit „S[j‘..]\$“
     - Markiere k‘ mit i+1

Beispiel

- “barbapapa”
- …

Komplexität – Naive Konstruktion

- Jeder Schritt von Ti zu Ti+1 ist O(m)
- Es gibt m-1 solcher Schritte
- Zusammen: O(m2)

→ Thema nächste Vorlesung:
O(m) Algorithmus von Ukkonen

Inhalt dieser Vorlesung

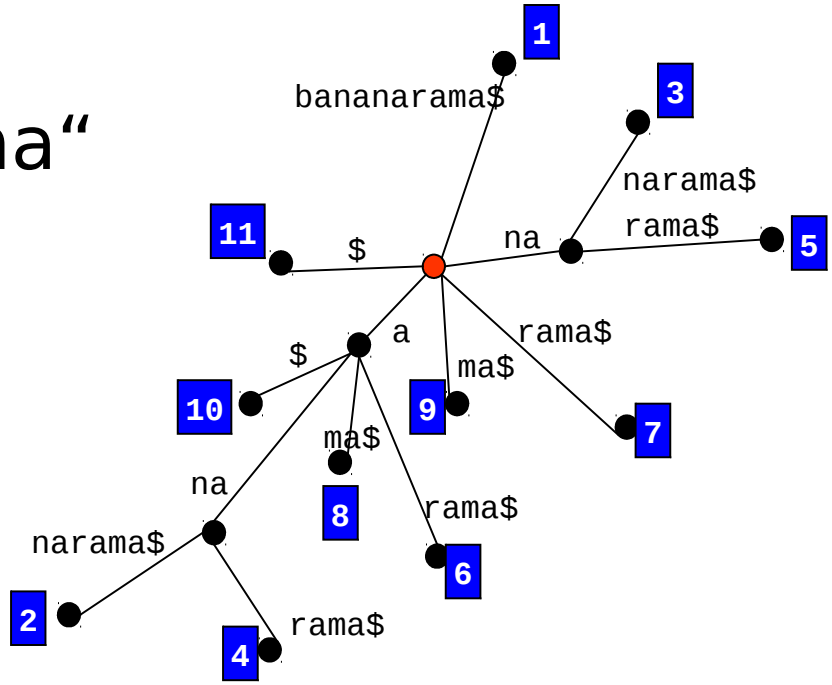
- Suffixbäume
  - Motivation
  - Aufbau
  - Naive Konstruktion
- Verwendung von Suffixbäumen

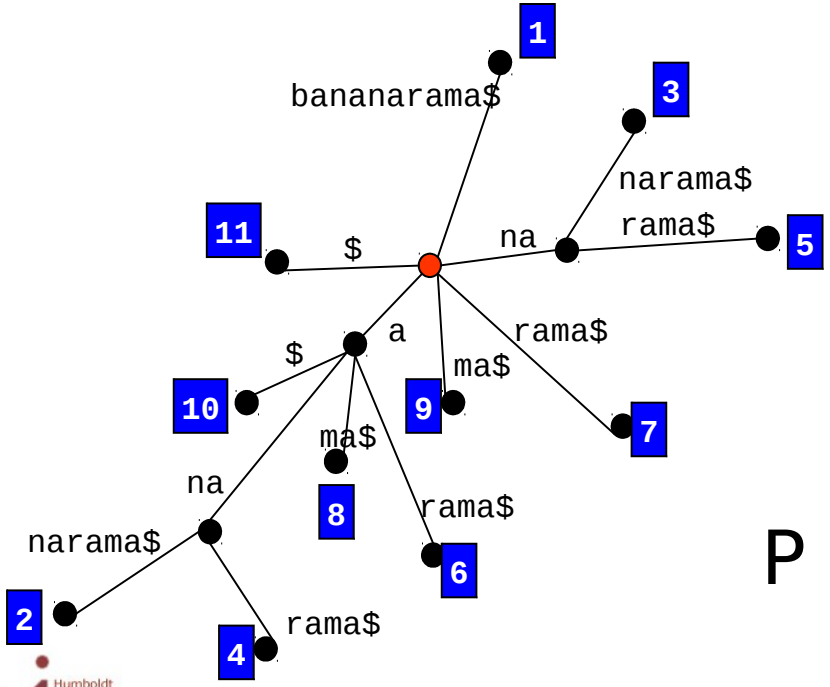
Suche mit Suffixbäumen

- Intuition
  - Jedes Vorkommen eines Pattern P muss Präfix eines Suffix sein
  - Und die haben wir alle auf Pfaden von der Wurzel aus
- Gegeben S und P. Finde alle Vorkommen von P in S
  - Konstruiere den Suffixbaum T zu S+“$“
    - Das geht in O(|S|), wie wir sehen werden
  - Matche P auf einen Pfad in T ab der Wurzel
    - Wenn das nicht geht, kommt P in S nicht vor
    - P kann in einem Knoten k enden; merke k
    - Oder P endet in einem Kantenlabel; sei k der Endknoten dieser Kante
  - Die Markierungen aller unterhalb von k gelegenen Blätter sind Startpunkte von Vorkommen von P in S

Beispiel: bananarama$

P = „na“

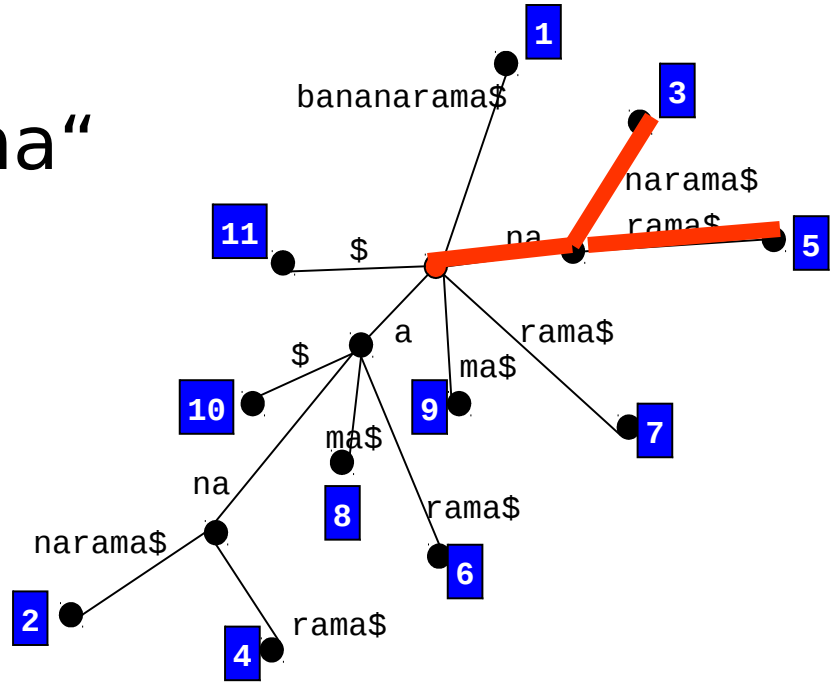


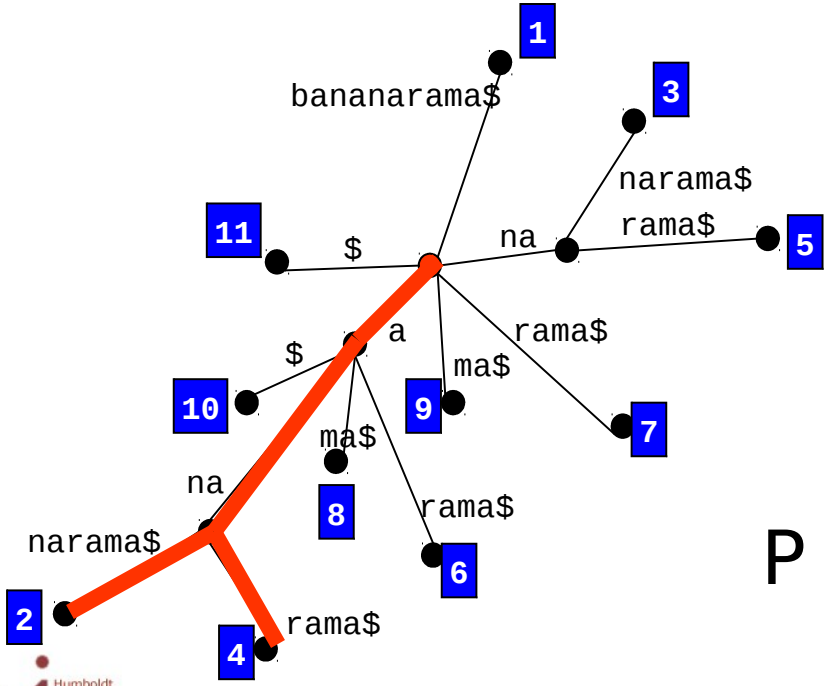


P = „an“

Beispiel: bananarama$

P = „na“





P = „an“

Komplexität

- Theorem
  Sei T der Suffixbaum für S+"$". Die Suche nach allen Vorkommen eines Pattern P, |P|=n, in S hat die Komplexität O(n+k), wenn k die Anzahl Vorkommen von P in S ist.
- Beweisidee
  - P in T matchen kostet O(n)
    - Pfade sind eindeutig – Entscheidung an jedem Knoten ist klar
  - Blätter aufsammeln ist O(k)
    - Baum unterhalb Knoten K hat k Blätter
    - Wie findet man die O(k)?

Längster gemeinsamer Substring

- Gegeben zwei Strings S_1 und S_2
- Gesucht: Längster gemeinsamer Substring s
- Vorschläge ?

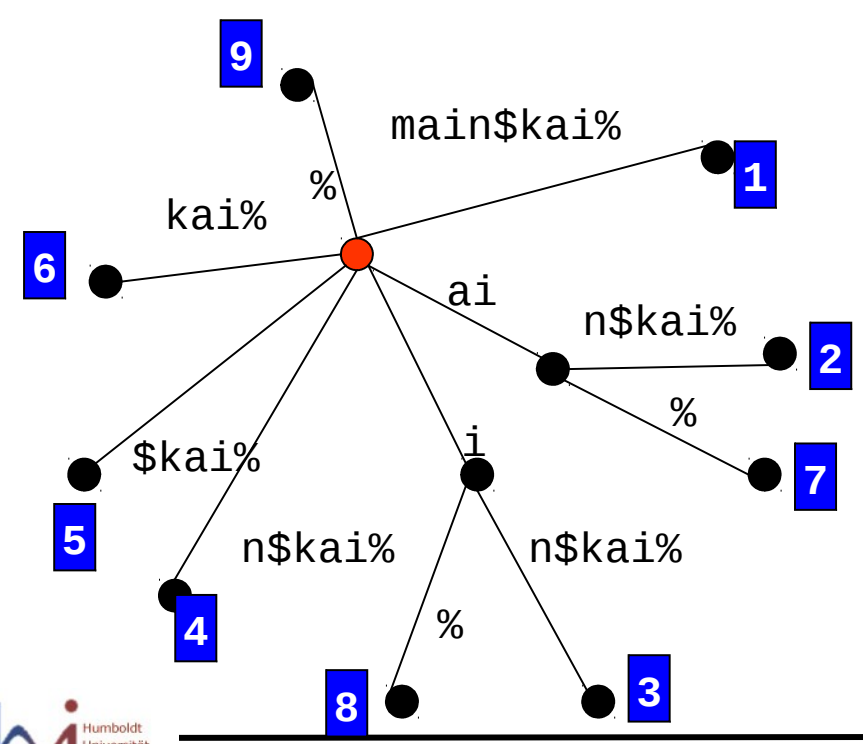
Längster gemeinsamer Substring

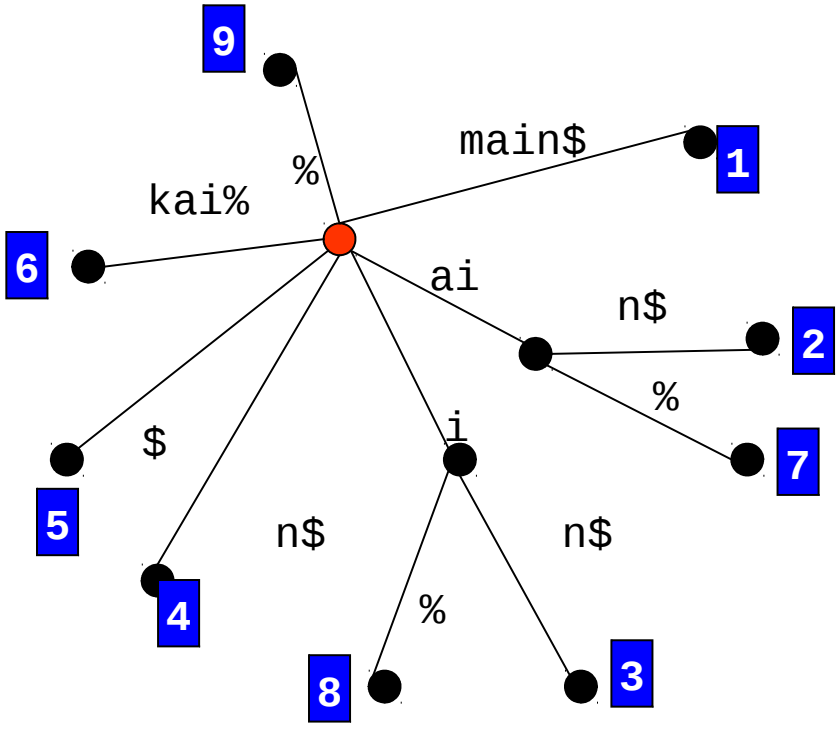
- Gegeben zwei Strings S_1 und S_2
- Gesucht: Längster gemeinsamer Substring s
- Vorschläge ?

- Lösung
  - Konstruiere Suffixbaum T für $S_1\$S_2\%$
  - Streiche aus diesem Baum alle Pfade unterhalb eines „$“
  - Durchlaufe den Baum
    - markiere alle internen Knoten mit 1, wenn im Baum darunter ein Blatt aus S_1 kommt
    - markiere Knoten mit 2, wenn ... Blatt aus S_2 vorkommt
  - Suche den tiefsten Knoten mit Beschriftung 1 und 2

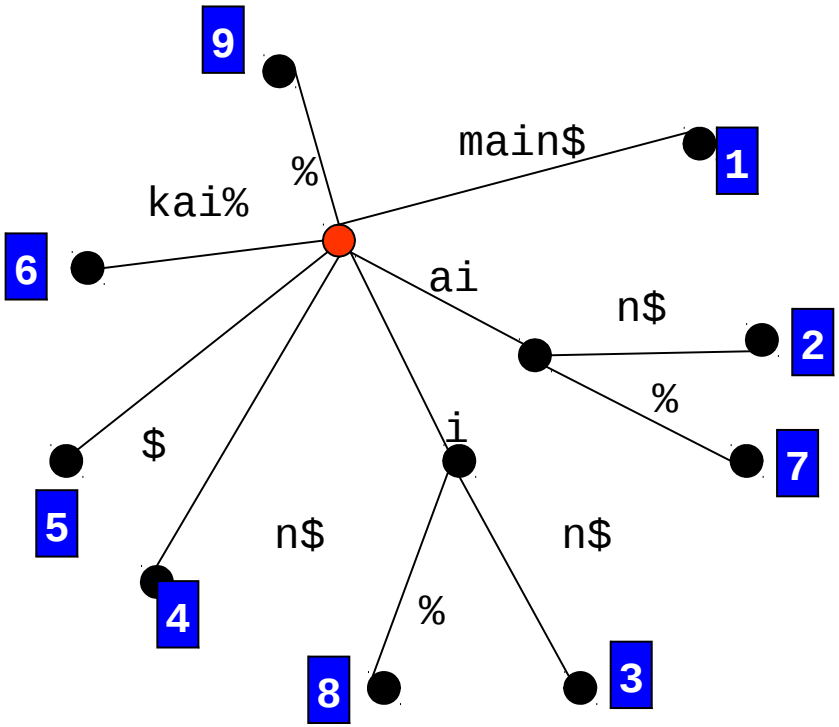
Beispiel

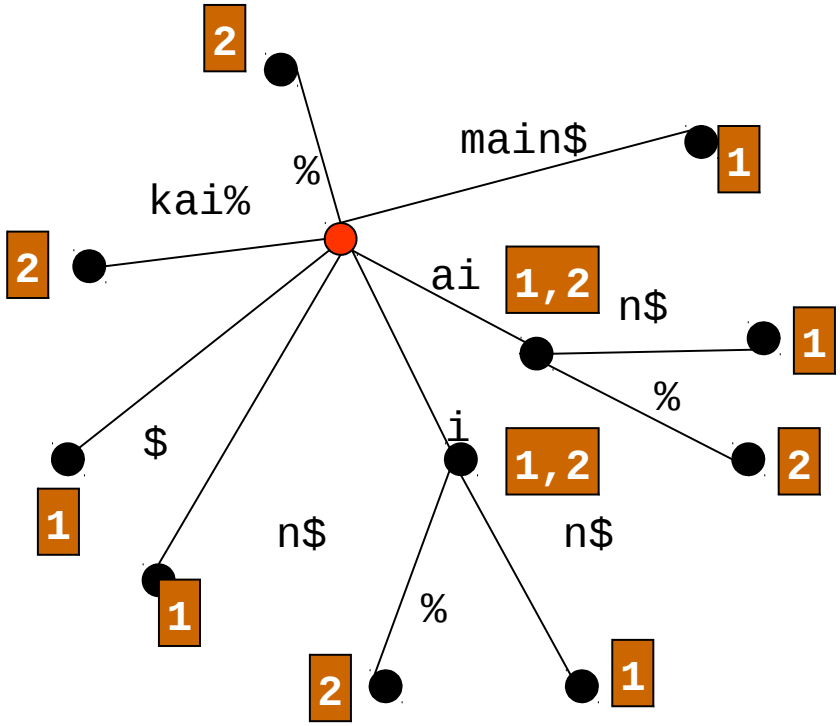
- S_1=main$, S_2=kai%
- ...





Beispiel





- Verallgemeinerbar zu n Strings $S_1,\ldots,S_n$

Komplexität

- Annahme: Wir können T für S in O(|S|) berechnen
- Die Schritte
  - Sei $m=|S_1|+|S_2|$
  - Konstruiere Suffixbaum T für $S_1\$S_2\%$
    - Ist O(m) nach Annahme
  - Streiche aus diesem Baum alle Pfade unterhalb eines „$“
    - Depth-First Traversal – O(m)
  - Durchlaufe den Baum und markiere innere Knoten mit 1,2
    - Depth-First Traversal – O(m)
  - Suche den tiefsten Knoten mit Beschriftung 1 und 2
    - Breadth-First Traversal – O(m)
- Zusammen: O(m)

Längstes „Palindrom“

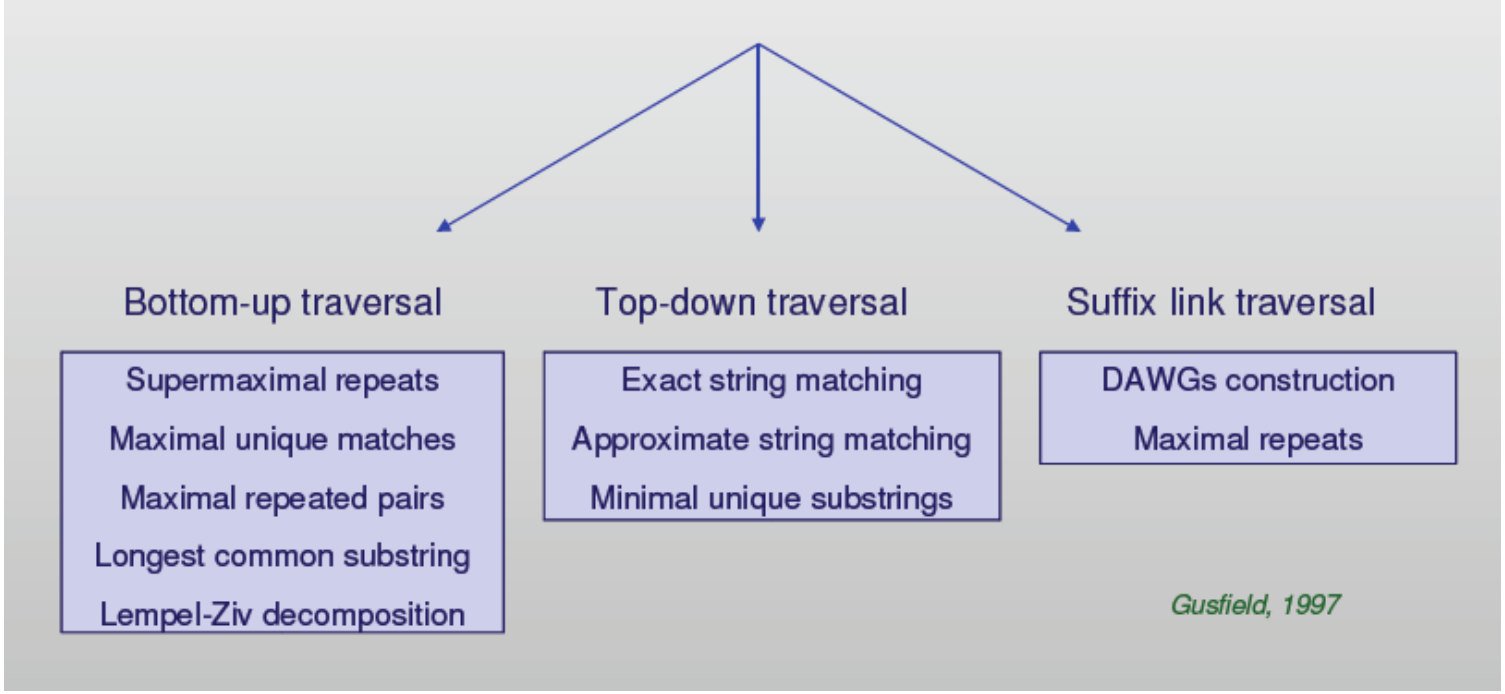
- Gegeben String S.
- Finde den längsten Substring s, der sowohl vorwärts als auch rückwärts in S vorkommt
- Ideen ?

Längstes „Palindrom“

- Gegeben String S.
- Finde den längsten Substring s, der sowohl vorwärts als auch rückwärts in S vorkommt
- Ideen ?

- Lösung
  - Suche längsten gemeinsamen Substring für S und reverse(S)

Weitere Anwendungen



- A. Apostolico: „The myriad virtues of subword trees, in: Combinatorial Algorithms on Words“, 1985

Selbsttest

- Wir kann man die k Blätter unterhalb eines Baumknoten p in einem Suffixbaum in O(k) erreichen?
- Naiver Konstruktionsalgorithmus für Suffixbäume und dessen Komplexität?
- Wie kann man in O(|P|) zeigen, ob P in einem String S enthalten ist (mit Präprozessierung)
- Wie kann man das in quasi O(k) erreichen, wenn n=|P| fest ist?