

# Programmierpraktikum

Sommersemester 2007 - 15.06.2007

Björn Wilmsmann, B.A.  
Sprachwissenschaftliches Institut  
Ruhr-Universität Bochum  
[wilmsmann@linguistics.rub.de](mailto:wilmsmann@linguistics.rub.de)

# Heute

- Datenstrukturen zur Speicherung textueller Daten: Tries, Suffix Trees und Suffix Arrays
- Motivation
  - Strings lassen sich in speziellen Datenstrukturen effizienter speichern
  - spezielle Datenstrukturen ermöglichen bestimmte Algorithmen überhaupt erst

# Tries

- spezielle geordnete Baumstruktur
- Knoten enthalten keine Schlüssel, sondern lediglich Schlüsselkomponenten
- Schlüssel werden durch vollständige Pfade repräsentiert
- Kanten werden mit Symbolen markiert

# Tries

- üblicherweise werden Tries Characterweise aufgebaut
- allerdings sind auch andere Schlüssel möglich (z.B. Worte)
- Tries sind deterministische endliche Automaten

# Automaten

- deterministisch: Nur eine mögliche Zustandsänderung für jeden spezifischen Input
- endlich: Endliche Anzahl von Zuständen
- Mealy-Automat
- Moore-Automat
- Harel-Automat

# Mealy-Automat

- $M = (S, \Sigma, \Delta, \delta, \lambda, s)$
- $S$  = Menge von Zuständen,  $s$  = Startzustand
- $\Sigma$  = Eingabealphabet
- $\Delta$  = Ausgabealphabet
- $\delta$  = Abbildung von  $S \times \Sigma$  auf  $S$
- $\lambda$  = Abbildung von  $S \times \Sigma$  auf  $\Delta$

# Moore-Automat

- $M = (S, \Sigma, \Delta, \delta, \lambda, s)$
- $S$  = Menge von Zuständen,  $s$  = Startzustand
- $\Sigma$  = Eingabealphabet
- $\Delta$  = Ausgabealphabet
- $\delta$  = Abbildung von  $S \times \Sigma$  auf  $S$
- $\lambda$  = Abbildung von  $S$  auf  $\Delta$

# Harel-Automat

- Hybrider Zustandsautomat, der sowohl  $\lambda$  des Mealy-Automaten als auch  $\lambda$  des Moore-Automaten beinhaltet

# Tries

- Unterschied zu BST (binary search trees):
  - bei BSTs hängt die Höhe des Baums von Größe der Schlüsselmenge ab
  - bei Tries hängt die Höhe des Baums von der maximalen Schlüssellänge ab

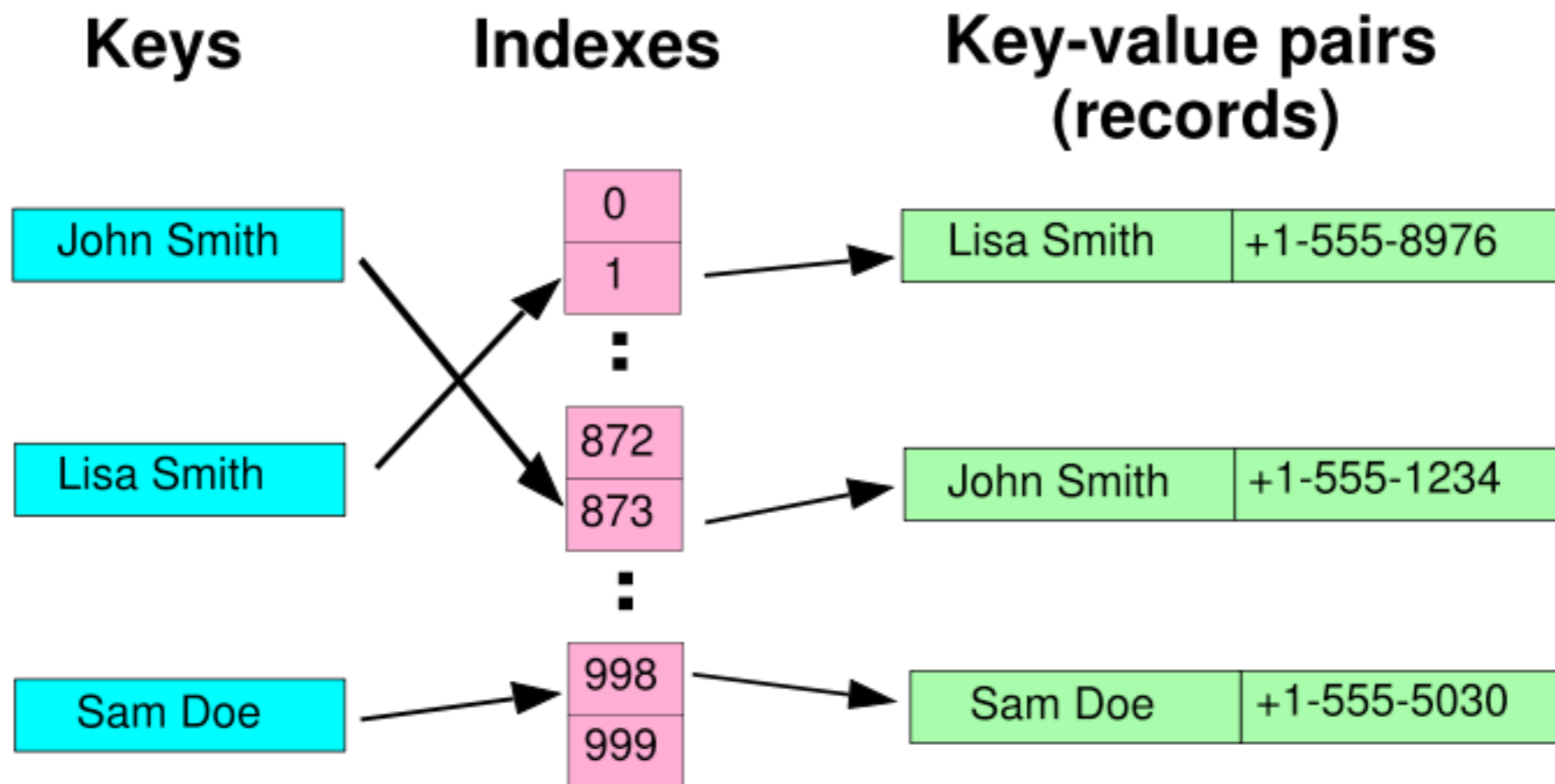
# Tries

- Vorteile gegenüber BSTs
  - schnellere Suche ( $O(m)$  vs.  $O(\log n)$ )
  - weniger Platzbedarf (Schlüssel sind nicht explizit gespeichert, sondern ergeben sich aus den Pfaden)
  - ermöglichen das Auffinden längster Präfixe

# Hashes

- weisen Werten anhand einer Hashfunktion einem Schlüssel zu
- je nach Hashfunktion kann ein Schlüssel dabei mehrere Werte referenzieren
- Beispiel für Hashfunktion:  $H(x) = x \bmod 3$

# Hashes



# Tries

- Vorteile gegenüber Hashes
  - schnellere Suche im Worst Case Fall ( $O(1)$  vs.  $O(n)$  bei kollidierenden Werten im Hash)
  - keine Kollisionen
  - keine Hashfunktion
  - alphabetische Sortierung

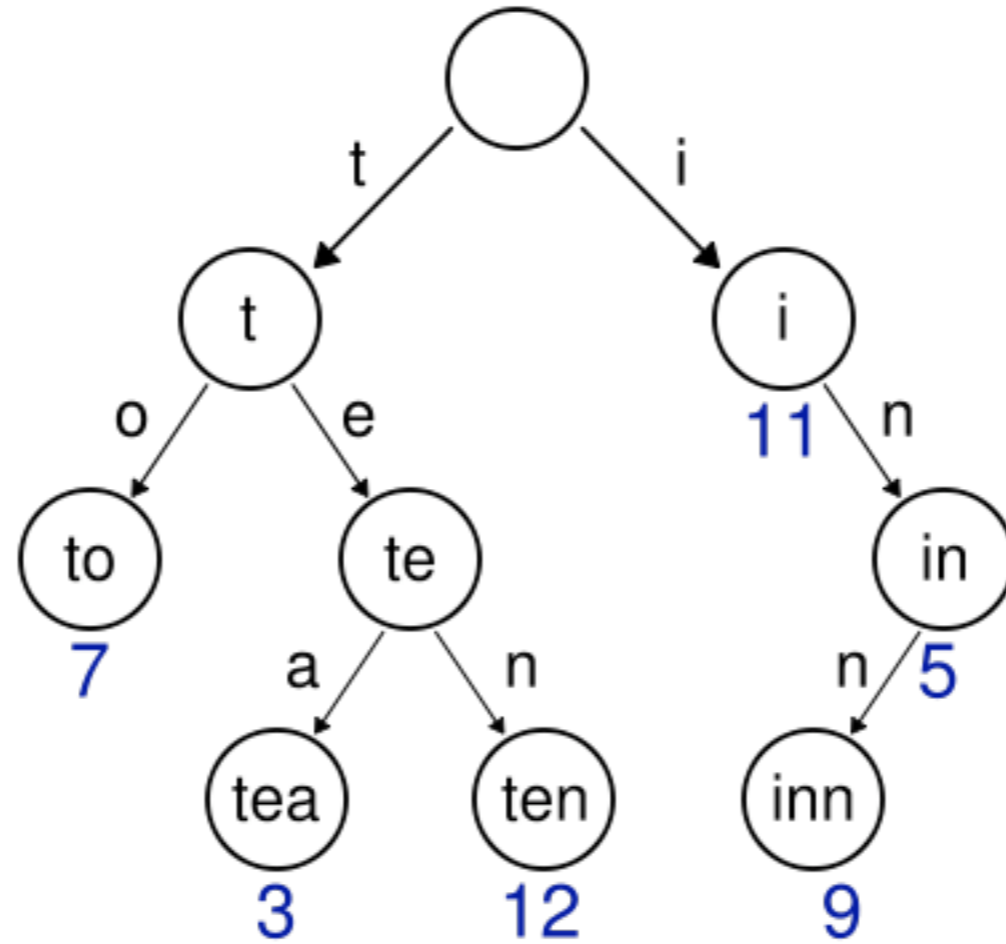
# Tries

- Nachteile gegenüber Hashes
  - können bei vielen gleichverteilten Zugriffen langsamer sein
  - nicht alle Schlüssel lassen sich als Strings darstellen
  - benötigen mehr Speicher als Hashes
  - sind meist nicht direkt verfügbar

# Tries

- Anwendungen
  - Präfixsuche
  - Lexika
  - T9
  - unscharfe Suche
  - Sortieren von Strings: BurstSort

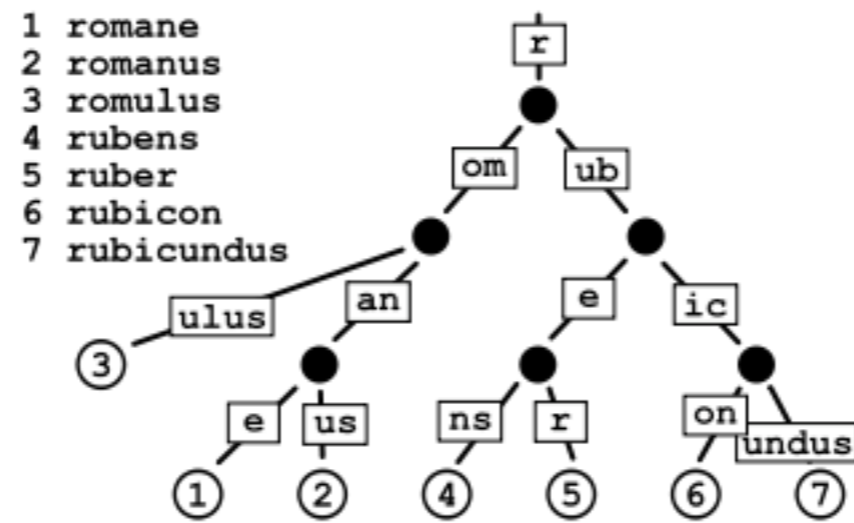
# Tries



# Patricia Trees

- PATRICIA: Practical Algorithm to Retrieve Information Coded in Alphanumeric
- auch: Radix Tree
- Sonderfall von Tries
- im Unterschied zu normalen Tries werden bei PAT Trees unäre Pfade reduziert
- Kanten sind mit Sequenzen bezeichnet

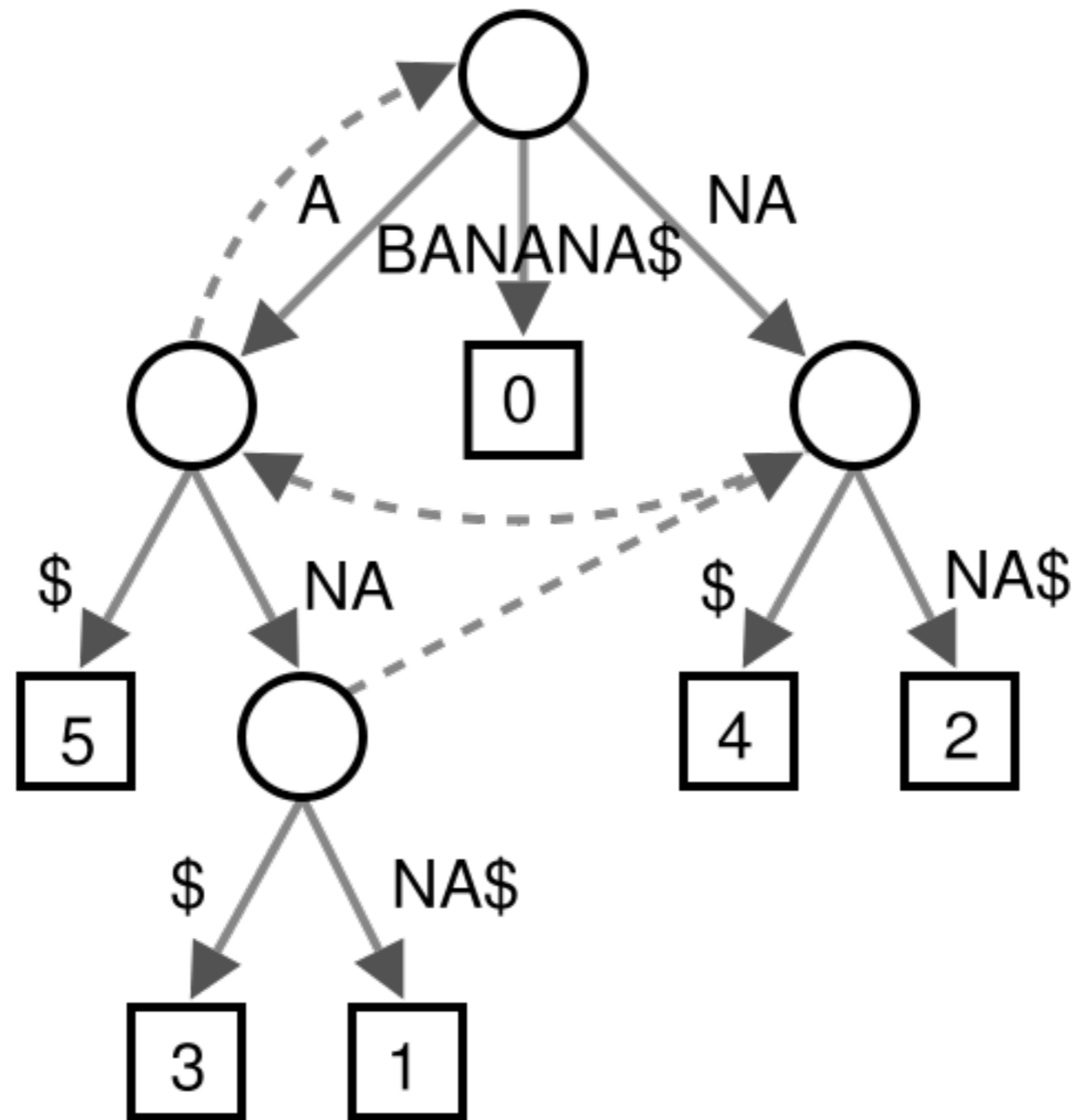
# Patricia Trees



# Suffix Trees

- Sonderfall eines Patricia Trees
- speichert Suffixe anstelle von Präfixen
- jedes Suffix entspricht einem Pfad
- daher haben alle Tochterknoten jeweils garantiert dasselbe Suffix

# Suffix Trees



# Suffix Arrays

- bieten gleiche Funktionalität wie Suffix Trees
- aber: Reduzierter Speicherverbrauch im Vergleich zu Suffix Trees
- speichern Suffixe für einen String in lexikographischer Reihenfolge (Konstruktion mittels Comparison Sort, z.B. Quicksort oder Mergesort)

# Beispiel

- abracadabra
  - a
  - abra
  - abracadabra
  - acadabra
  - adabra

# Beispiel

- bra
- bracadabra
- cadabra
- dabra
- ra
- racadabra

# Beispiel

- Suffix Array für diesen String:  
{11,8,1,4,6,9,2,5,7,10,3}
- Anwendung: Longest Common Substring Problem

# Beispiel

- weitere Anwendung: Auffinden aller Vorkommen eines Strings
- zweimalige binäre Suche
- Ergebnis: Anfangs- und Endindex
- $\text{Endindex} - \text{Anfangsindex} = \text{Anzahl}$

# Binary Search (Anfangsindex)

```
BinarySearchStart(A[0..N-1], value, low, high) {  
    if (high < low)  
        return high  
    mid = (low + high) / 2  
    if (A[mid] starts with value)  
        return BinarySearchEnd(A, value, low, mid-1)  
    else  
        return BinarySearchEnd(A, value, mid+1, high)  
}
```

# Binary Search (Endindex)

```
BinarySearchEnd(A[0..N-1], value, low, high) {  
    if (high < low)  
        return high  
    mid = (low + high) / 2  
    if (A[mid] starts with value)  
        return BinarySearchEnd(A, value, mid+1, high)  
    else  
        return BinarySearchEnd(A, value, low, mid-1)  
}
```

# Aufgabe

- erstellen Sie ein Modul, das
  - ein Corpus einliest
  - daraus ein lexikographisch sortiertes Suffix Array erstellt
  - einen String als Argument nimmt
  - die Häufigkeit des Vorkommens dieses Strings ausgibt

# Aufgabe

- benutzen Sie zum Konstruieren des Suffix Arrays einen beliebigen Comparison Sort (Mergesort, Quicksort) oder  $\langle = \rangle$
- implementieren Sie dazu als Hilfsklasse eine binäre Suche zum Auffinden der Start- und Endindices für Substrings, die mit dem gesuchten String beginnen

# Aufgabe

- Abgabe der Aufgabe in Teams per E-Mail
- Abgabe bis zum 28.06.2007, 0 Uhr

# Nächste Sitzungen

- letzte Sitzungen bis zum Semesterende:  
Entwicklung einer vollständigen Applikation  
in Perl
- Vorschläge für die zu entwickelnde  
Applikation
- bisherige Idee: Akronym Generator

# Referenzen

- [1] Baeza-Yates, Ricardo / Ribeiro-Neto, Berthier (1999): Modern Information Retrieval. Boston, MA: Addison-Wesley.
- [2] Güting, R. H. u. Dieker, S. (2004): Datenstrukturen und Algorithmen, Teubner Verlag, Wiesbaden.

# Referenzen

- [3] Mikio Yamamoto and Kenneth Church: Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus. Proceedings of the 6th Workshop on Very Large Corpora, 1998

# Referenzen

- [4] Ukkonen, Esko: On-line construction of suffix trees: <http://www.cs.helsinki.fi/u/ukkonen/SuffixT1withFigs.pdf>

# Referenzen

- [5] [http://en.wikipedia.org/wiki/Hash\\_table](http://en.wikipedia.org/wiki/Hash_table)
- [6] [http://en.wikipedia.org/wiki/Patricia tree](http://en.wikipedia.org/wiki/Patricia_tree)
- [7] [http://en.wikipedia.org/wiki/Suffix\\_array](http://en.wikipedia.org/wiki/Suffix_array)
- [8] [http://en.wikipedia.org/wiki/Suffix tree](http://en.wikipedia.org/wiki/Suffix_tree)
- [9] <http://en.wikipedia.org/wiki/Trie>

**Vielen Dank für Ihre  
Aufmerksamkeit!**