Präfix Trie zur Stringverarbeitung

Cheng Ying
Sabine Laubichler
Vasker Pokhrel

Übersicht:

- Einführung
- Eigenschaften von Tries
- Verwendung von Tries
- Allgemeine Definition von Patricia Tries
- Eigenschaften von Patricia Tries
- Präfix- bzw. Radix Baum

Einführung:

- Ein Präfixbaum oder Trie ist eine Datenstruktur, um Datenketten zu suchen.
- Trie stammt ursprünglich aus dem Englischen (retrieval) ab, und wurde von Brandais (1959) und Fredkin (1960) vorgestellt.

Einführung (Forts.):

- Zerlegung des Schlüssels – bestehend aus Zeichen eines Alphabets.
- Aufbau des Baumes nach Schlüsselteilen.
- Suche im Baum und vergleiche die Schlüsselteile miteinander.
- Jede unterschiedliche Folge vom Teilschlüssel ergibt einen eigenen Suchweg im Baum.

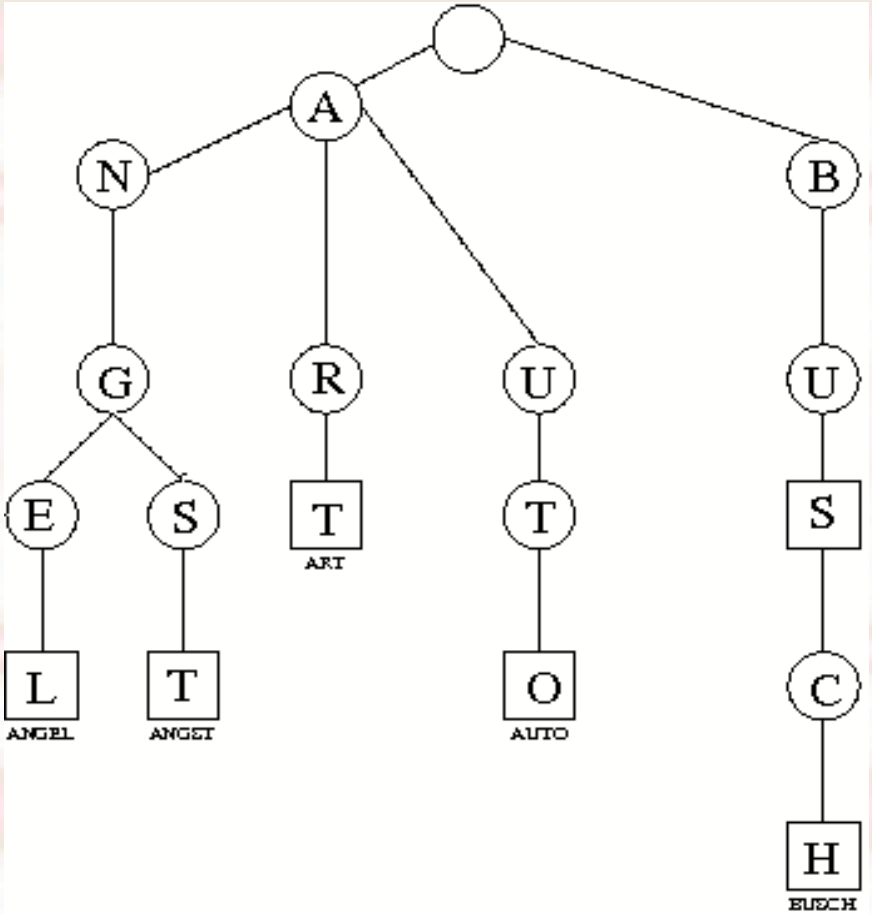
Einführung (Forts.):

- Alle Schlüssel mit dem gleichen Präfix haben in der Länge des Präfixes den gleichen Suchweg.
- vorteilhaft u.a. bei variabel langen Schlüsseln z.B. Strings

Einführung (Forts.):

- Tries erlauben große Dokumentenbestände nach Zeichenketten zu durchsuchen.
- Der Trie bildet einen Index (key) zu diesen String.

Eigenschaften von Tries:



- Die zu Schlüsseln führenden Wurzelpfade sind geordnet.

Eigenschaften von Tries:

- Ein Trie ist unabhängig von der Einfügereihenfolge der Schlüssel.
- oder: Ein Trie ist eindeutig durch die Menge der einzufügenden Schlüssel bestimmt.
- Wurzelpfade haben jeweils gesamte Schlüssellänge.

Arten von Tries:

- Präfix (Standard) Trie
- Compressed (Praticia) Trie und
- Suffix Tries

Verwendung:

- Datenbanksuche
- P2P Netzwerk
- IP Routing Tabelle
- Wörterbuch
- Suchmaschinen
- Anwendungen in der Biologie z. B. DNA
- XML

Defintion von Tries:

- Positionsbaum über Alphabet ∑
- wenn Trie T k Schlüssel besitzt, besitzt T n Blätter
- Jedes Blatt besitzt n Schlüssel.
- Sei S eine Menge von n Strings im Alphabet ∑ S[1,n] = S [1], S[2].....S[n]

Definition von Tries (Forts.):

- S[i,j] = S[i]....S[j] ist ein Substring von S, dann ist S[i,j] Präfix von S und S[i,n] ist Suffix von S.
- Strings werden durch lexikografische Ordnung sortiert. $\sum = \{a,b,...,z\}$ wo $a < b < ... < z$

  ist.

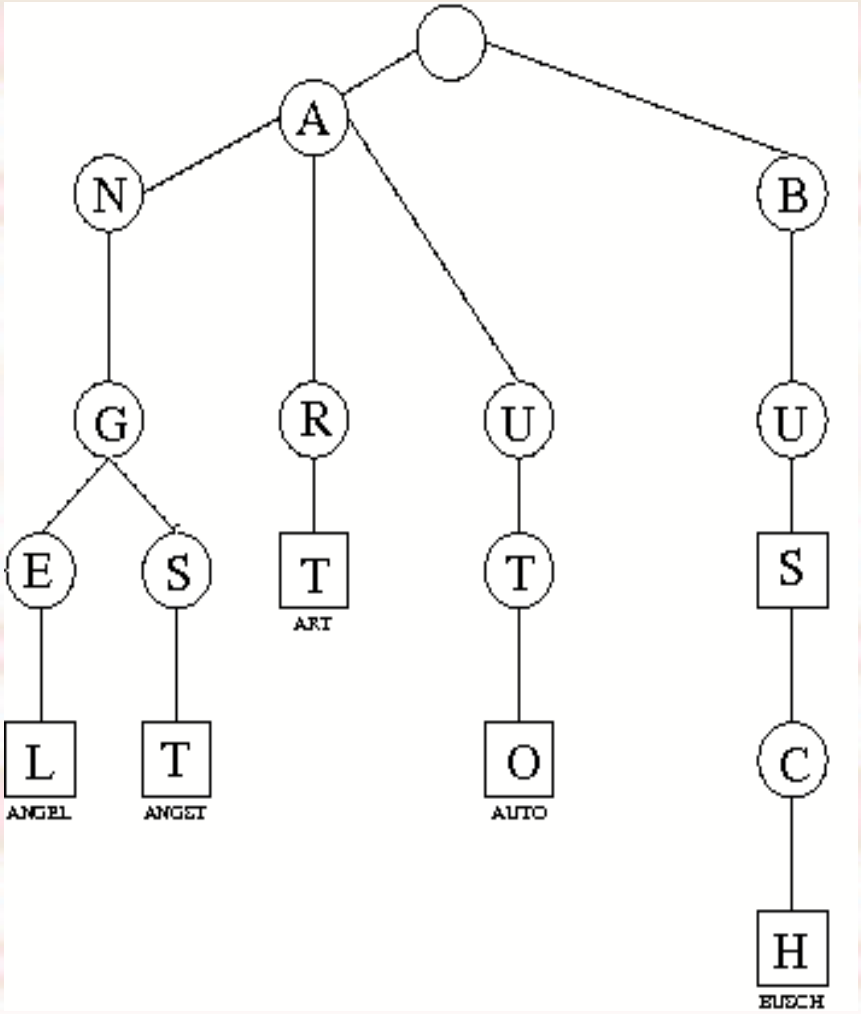
Definition von Tries (Forts.):

- Sei Trie T mit folgenden Eigenschaften:
  - Jeder Knoten, von T mit Ausnahme der Wurzel, ist mit einem Zeichen von ∑ versehen.
  - Kinder, eines internen Knotens von T, sind kanonisch angeordnet.
  - Kein String in S ist Präfix eines anderen Strings.

Definition von Tries (Forts.):

- T besitzt k externe(n) Knoten, die jeweils einen String von S repräsentieren.
- Die Aneinanderreihung der Knotenbezeichnungen auf dem Weg von der Wurzel zu einem externen Knoten v von T ergibt den String von S, den v repräsentiert.

Definition von Tries (Forts.):

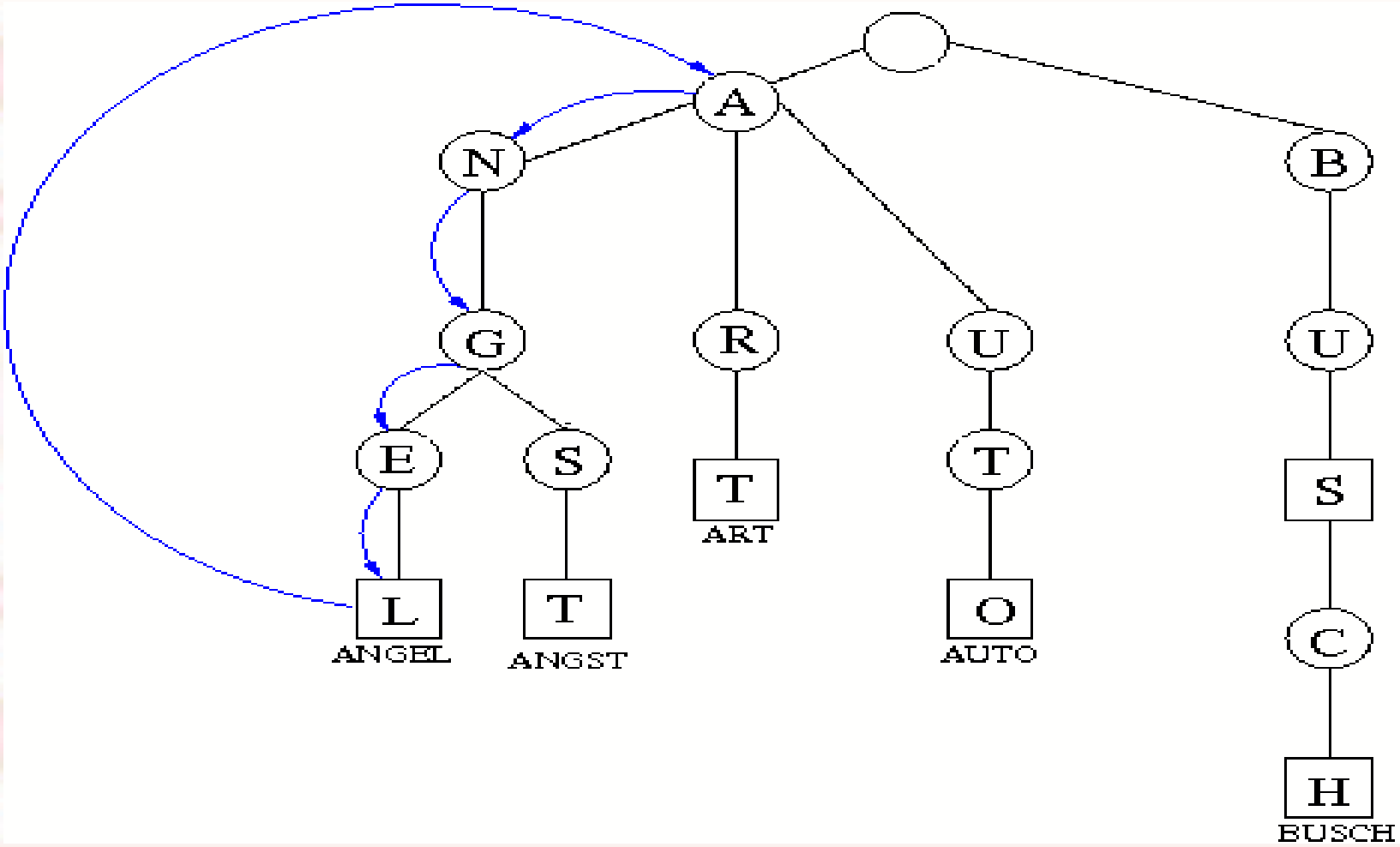


- Trie T für die Strings {ANGEL, ANGST, ART, AUTO, BUS, BUSCH}
- Alphabet $\sum$ = Buchstaben {A,..,Z}

Suche (search):

- Die Funktion Suchen überprüft einen Trie auf das Vorhandensein eines Strings s[1...n].
- In der Wurzel wird nach dem ersten Zeichen des Suchschlüssels verglichen. Bei Gleichheit wird der zugehörige Zeiger verfolgt.
- Im gefundenen Knoten wird nach dem zweiten Zeichen verglichen usw.

Suche (search)



Java-Implementierung (Search)

```
public boolean search (Trie t, String s)
      // sei der aktuelle Knoten die Wurzel von t
{
      for (int n=0; n < s.length(); n++)
      // überprüfe für alle s[i]
      {
            int index = s.charAt(k) – 'a';
            if (t.next[index] == null)
            // ob der aktuelle Knoten einen Zeiger auf
            {
                  // den Knoten besitzt, der s[i] repräsentiert
                  return false;
                  // falls nicht, gib false zurück
            }
```

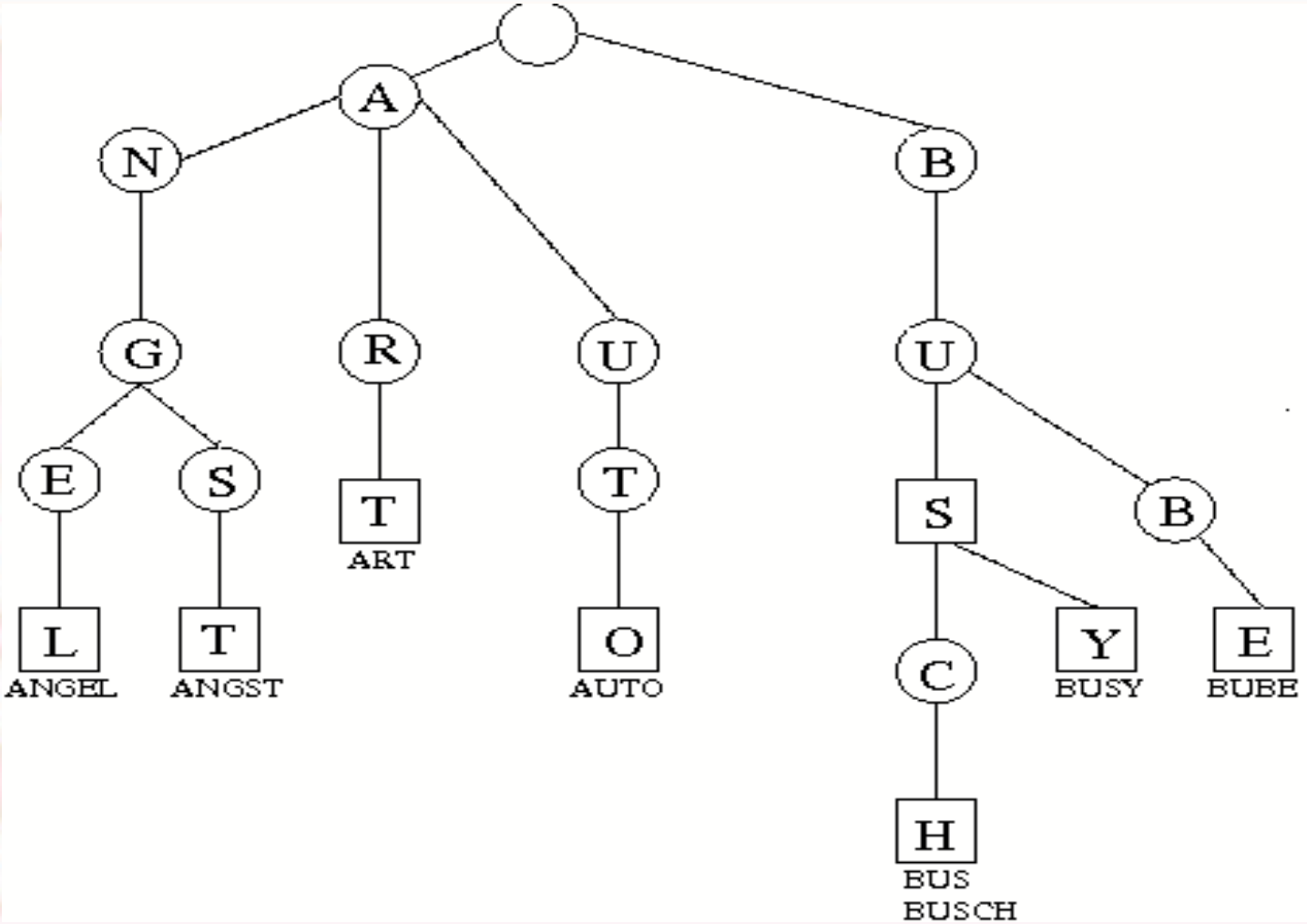
Java-Implementierung (Search) (Forts.):

```
            t = t.next[index];
            // setze den aktuellen Knoten auf
            }
            // eben dieses Kind des aktuellen Knotens
            return t.isWord;
            // gib beim Knoten, der s[n] repräsentiert,
}
      // den Wert des Flags isWord zurück
```

Einfügen (insert):

- Die Funktion insert fügt einen String s [1...n] in einen Trie T ein (t,s).
  Wenn Suchpfad schon vorhanden, wird NULL Zeiger in ein * (mit einem Zeichen belegt) Zeiger umgewandelt und es wird ein neuer Knoten eingefügt.

Einfügen (insert) in Tries:



Java-Implementierung (Insert)

```
public void insert (Trie t, String s)
        // sei der aktuelle Knoten die Wurzel von t
{
        for (int n=0; n < s.length(); n++)
        // überprüfe für alle s[i]
        {
        int index = s.charAt(n) – 'a';
        if (t.next[index] == null)
        // ob der aktuelle Knoten einen Zeiger auf
                {
                // den Knoten besitzt, der s[i] repräsentiert
                t.next[index] = new Trie();
                // falls nicht, füge diesen Knoten ein
                }
}
```

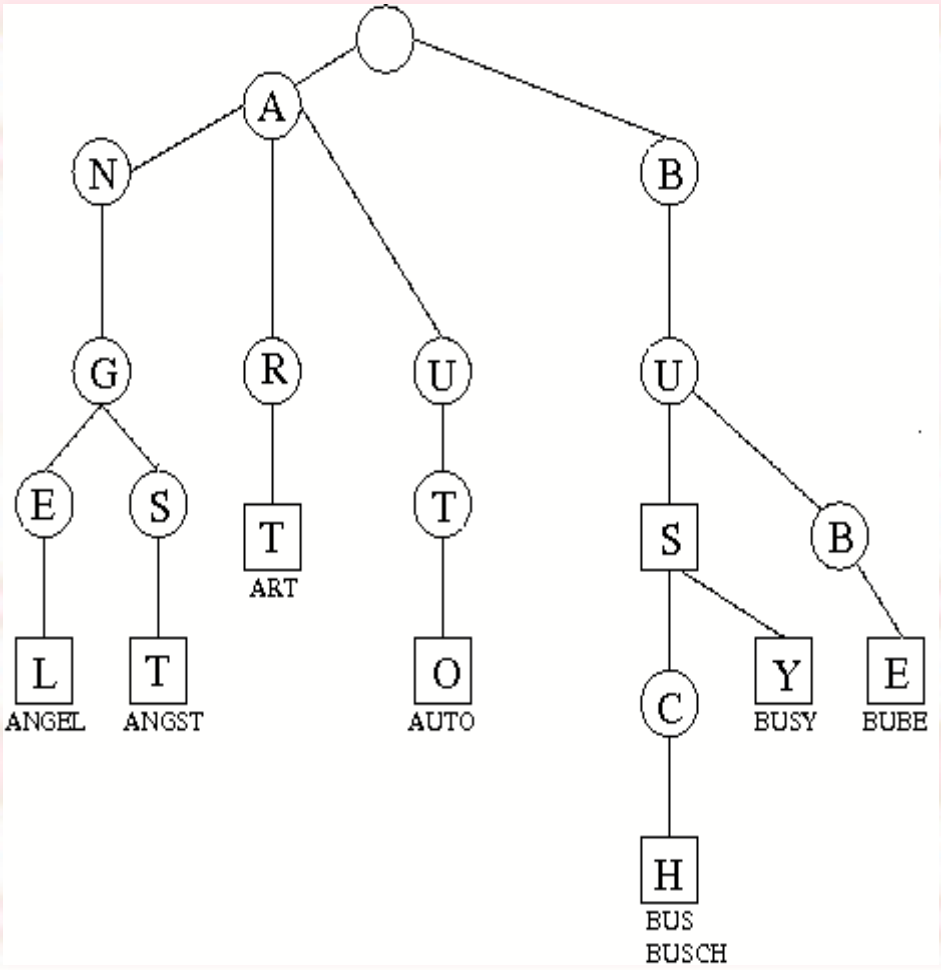
Löschen (delete):

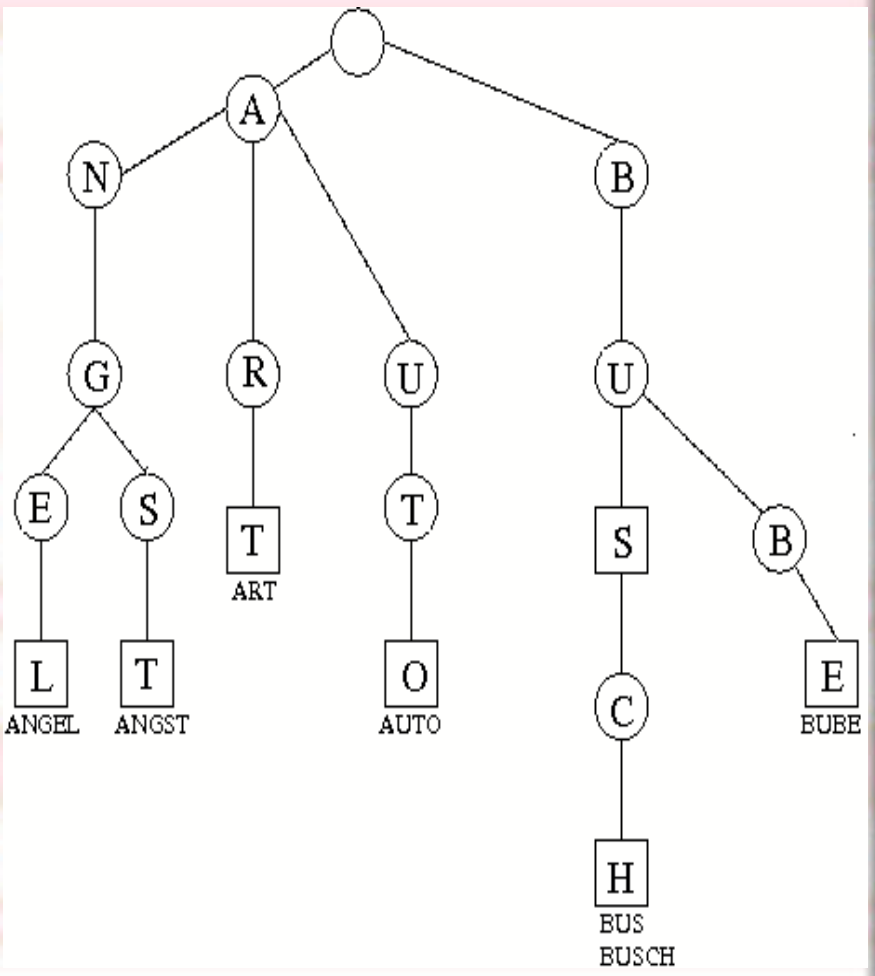
- Die Funktion delete löscht einen String s [1...n] aus einem Standard Trie T.
- Vor jeder Löschung geht eine erfolgreiche Suche voraus.
- Die Kante, die auf das Blatt zeigt, das den zu löschenden Schlüssel enthält, wird entfernt.

Löschen (delete) (Forts.):

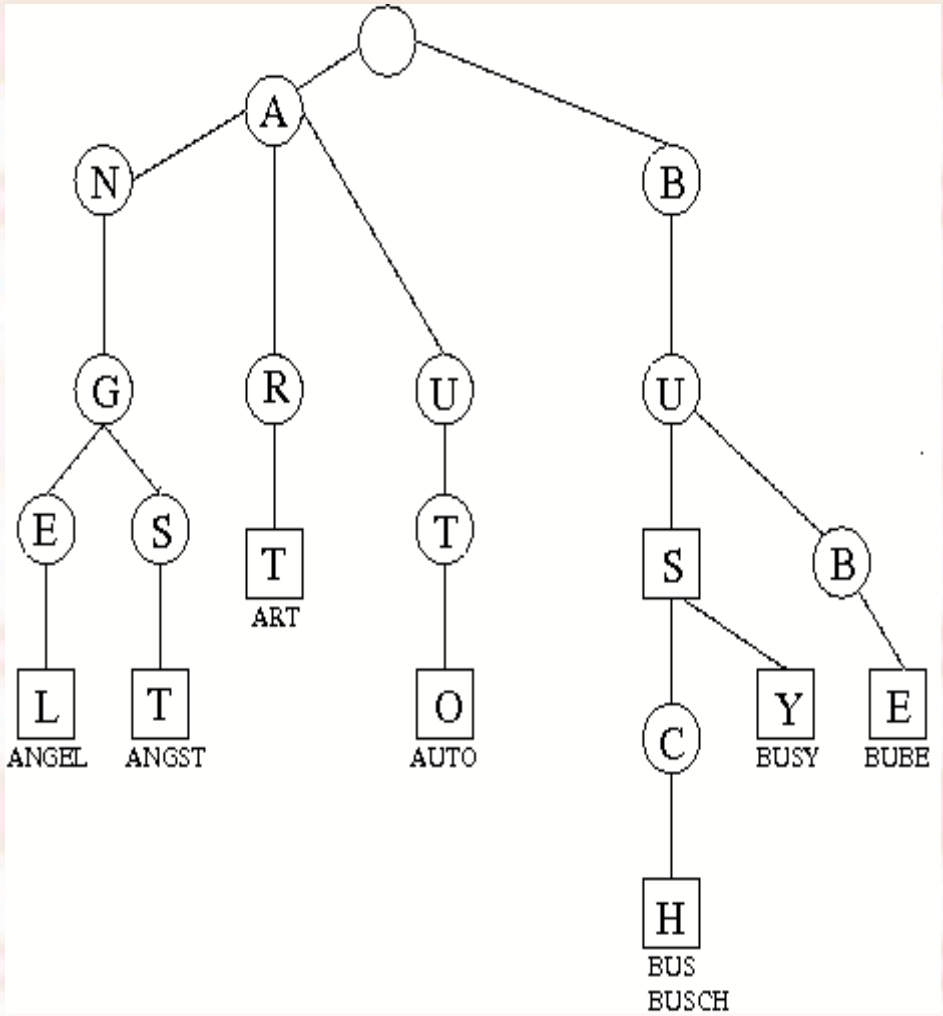
- Fall 1:
  - Falls der Knoten kein Nachfolger hat, lösche ihn.
- Fall 2:
  - Falls der Knoten Nachfolger hat, wird nur der Vermerk auf dem Knoten als Blattknoten gelöscht, aber nicht der Knoten selber.

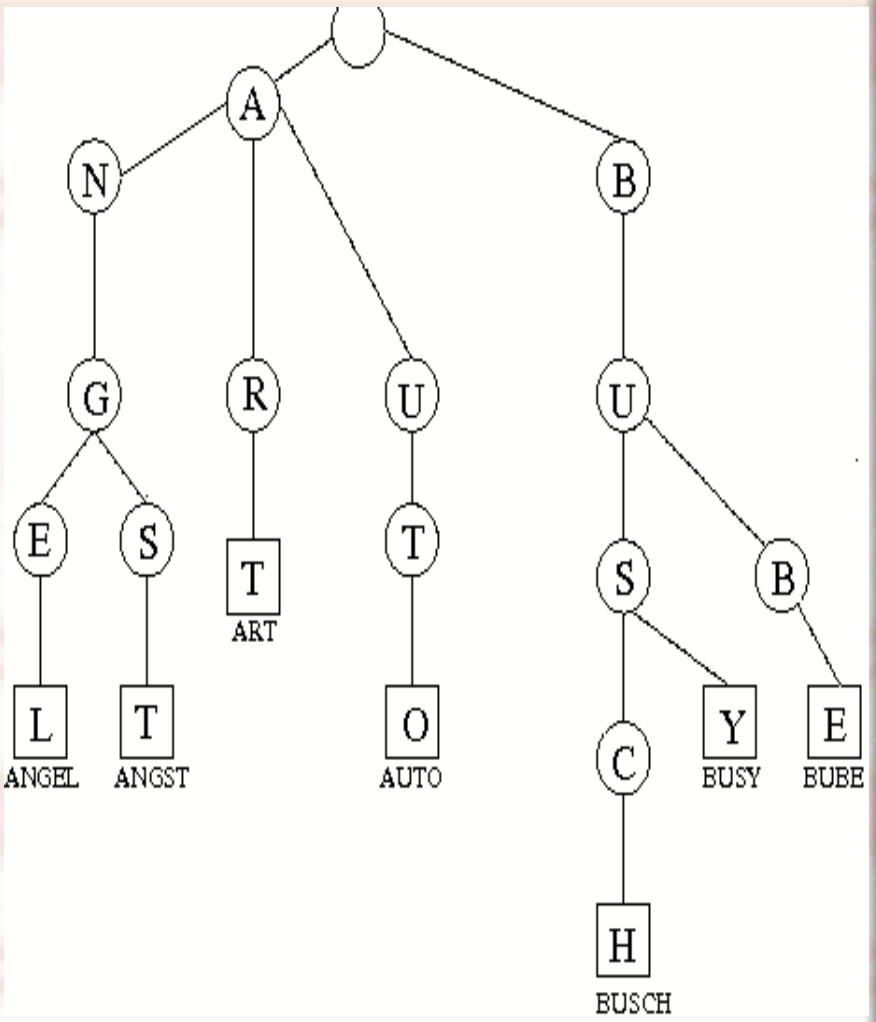
Fall 1:


A
N
B
G
R
U
U
E
S
T
S
B
L
T
O
C
Y
E
H
ANGEL
ANGST
ART
AUTO
BUSY
BUBE
BUS
BUSCH


A
N
B
G
R
U
U
E
S
T
T
S
B
L
T
O
C
E
H
ANGEL
ANGST
ART
AUTO
BUBE
BUS
BUSCH

Fall 2:


A
N
B
G
R
U
U
E
S
T
S
B
L
T
O
Y
E
C
ANGEL
ANGST
ART
AUTO
BUSY
BUBE
H
BUS
BUSCH


A
N
B
G
R
U
U
E
S
T
S
B
L
T
O
Y
E
C
ANGEL
ANGST
ART
AUTO
BUSY
BUBE
H
BUSCH

Java-Implementierung (Delete)

```
public boolean search (Trie t, String s)
        {
        if (!search(s))
        // überprüfe zunächst, ob der zu löschende String im Trie vorhanden
        // ist
                {
                return;
                // wenn nicht:fertig!
                }
        doDelete(s, 0, t, t.next[s.charAt(o) – 'a']);
        // ansonsten: rufe doDelete auf
        }
```

Java-Implementierung (Delete) (Forts.)

```
void doDelete(String s, int n, Trie prev, Trie current)
{
	if (n < (s,length() - 1)
	// gehe zuerst rekursiv zum Knoten, der s[n] repräsentiert
		doDelete (s, (n+1), current, current.next[s.charAt(n+1) – 'a']);
	if (k == (s.lentgh() - 1))
	// lösche an diesem Knoten das Flag
		current.isWord = false;
	if (current.isEmty() && (current.isWord == false))
	// ist dieser Knoten extern und .
		next[s.charAt(k) – 'a'] = null;
		//repräsentiert er kleinen String aus S, so lösche ihn.
		// verfahre dann analog mit allen “darüberliegenden” Knoten
}
```

Nachteile:

- schlechte Speicherplatzausnutzung
- dünn besetzt (pro Knoten ein Zeichen)
- viele Einweg Verzweigungen

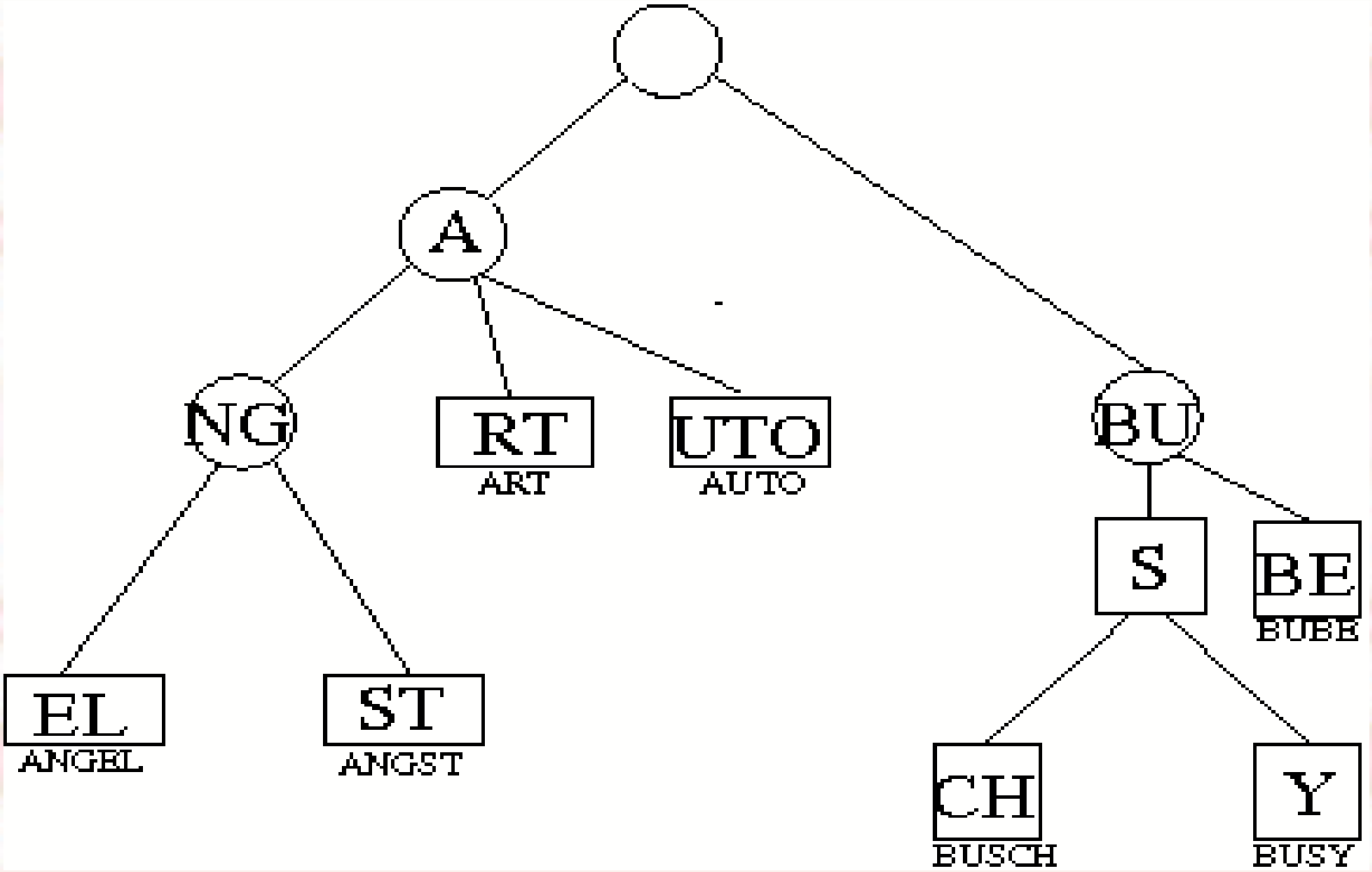
Kompression der Tries:

- Pfade mit je einem Nachfolger sind redundant und lassen sich verkürzen, falls die Schlüssel in den Blättern vorhanden sind.
- Der (Rest-) Schlüssel nimmt ein spezielles Knotenformat an und der Unterbaum wird dadurch eingespart.

Vorteile der Kompression:

- vermeidet Einwegverzweigungen
- nur besetzte Verweise werden gespeichert

Patricia Trie



Patricia Trie:

Practical Algorithm to Retrieve Information Coded in Alphanumeric

- Merkmale:
  - Binärdarstellung für Schlüsselwert => ähnlich wie digitaler Binärbaum
  - Speicherung der Schlüssel in den Blättern
  - in innere Knoten wird gespeichert, wie viele Zeichen (Bits) zur Wegwahl zu überspringen sind.

Patricia Trie (Forts.):

- Vermeidung von Einwegverzweigungen, in dem bei nur einem verbleibenden Schlüssel direkt auf entsprechendes Blatt verwiesen wird.

Eigenschaften:

- speichereffizient
- sehr gut geeignet für (variabel) lange Schlüssel und (sehr lange) Binärdarstellungen von Schlüsselwerten.

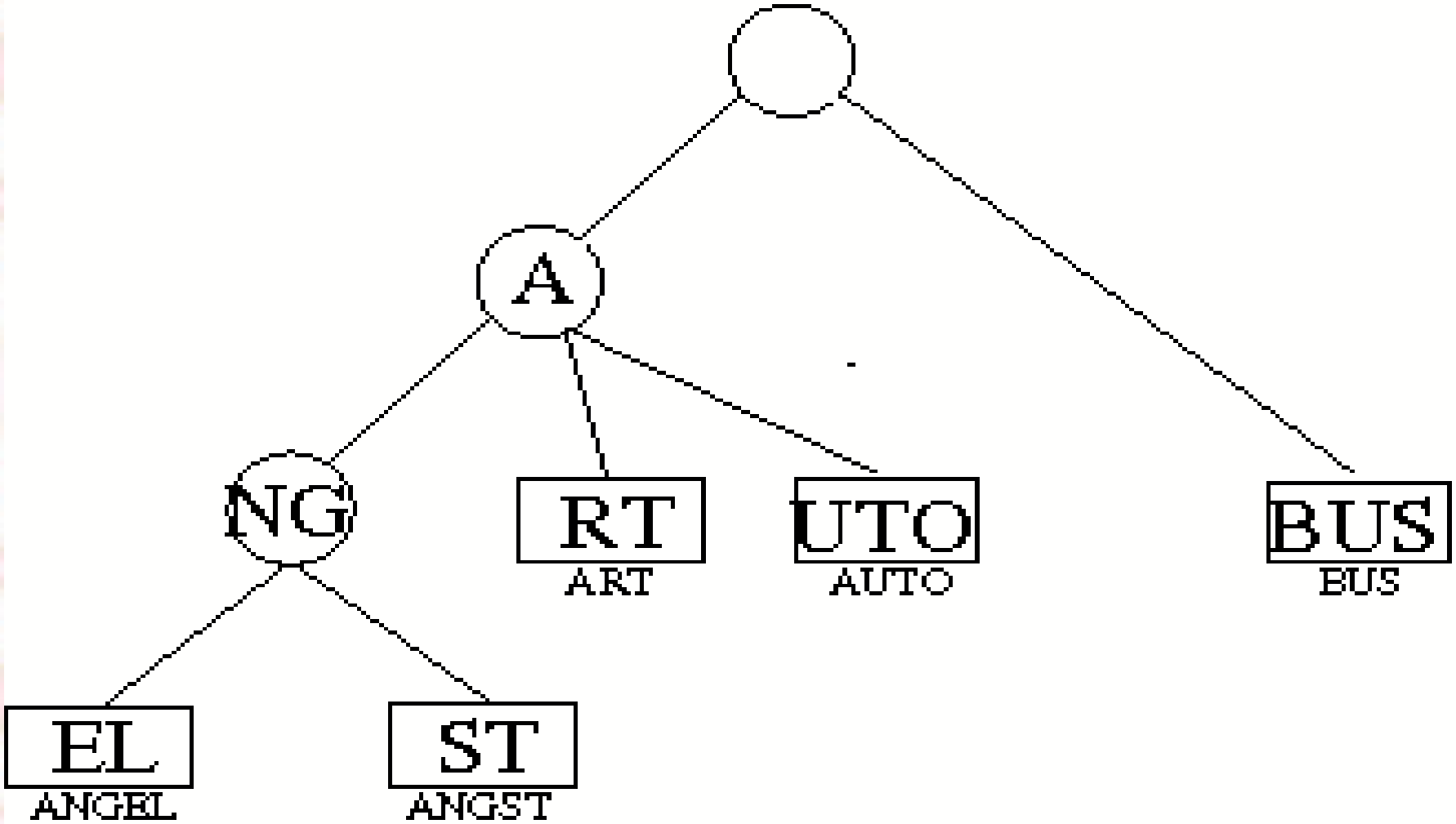
Eigenschaften (Forts.):

- Bei jedem Suchschlüssel muss die Testfolge von der Wurzel beginnend ganz ausgeführt werden, bevor über Erfolg und Misserfolg der Suche entschieden werden kann.
- Beide Suchergebnisse enden in einem Blattknoten.

Funktionen für Compressed Tries

- Die Funktionen insert, search, und delete für Compressed Tries entsprechen denen der Standard Tries und besitzen die gleiche Laufzeit. Dabei muss beachtet werden, dass beim Einfügen bzw. Löschen Knoten aufgespaltet bzw. zusammengefasst werden, um sicherzustellen, dass jeder interne Knoten mehr als ein Kind besitzt.

Patricia Trie



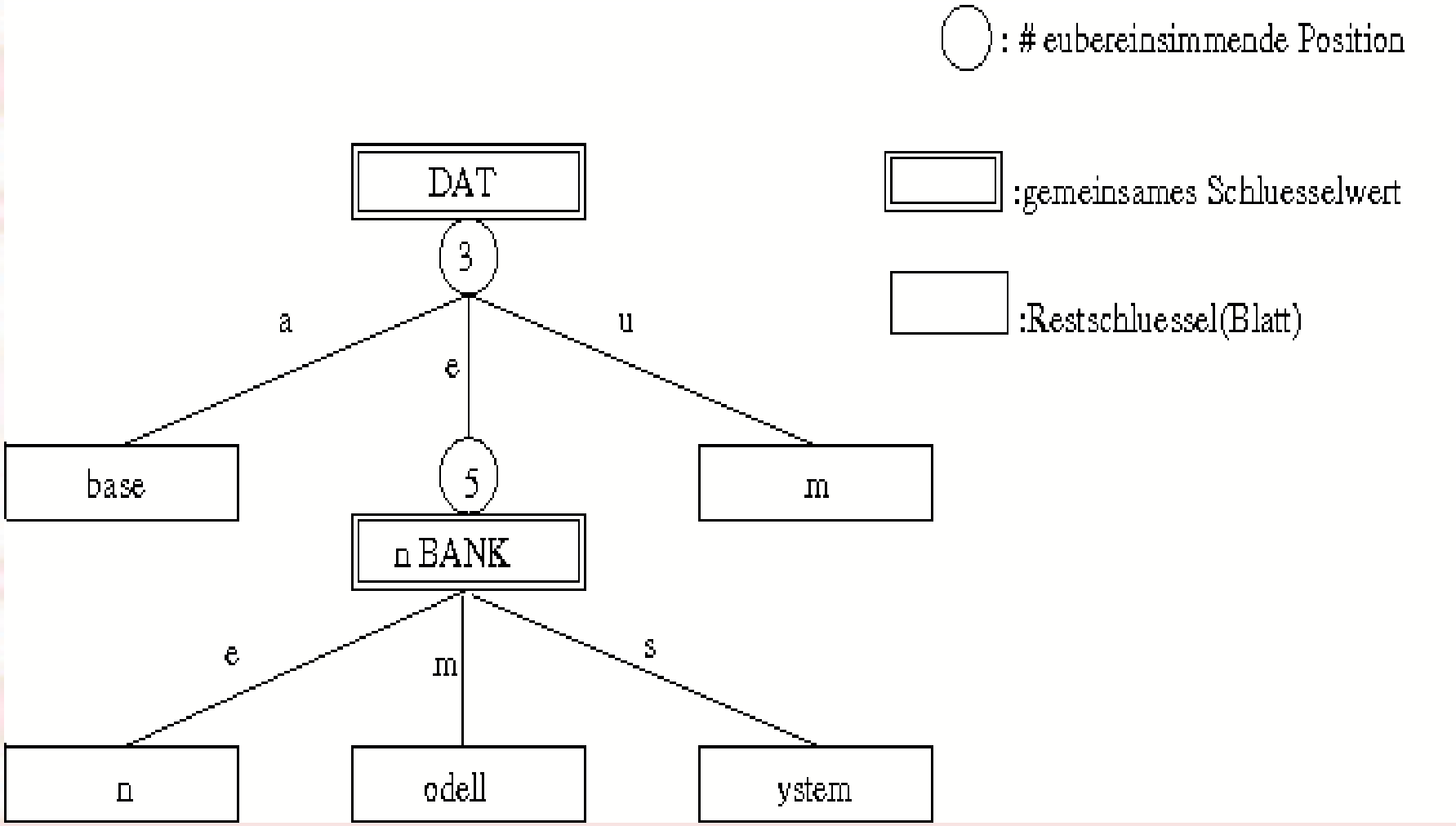
Präfix- bzw. Radix Baum:

- Digitalbaum als Variante des Praticia Trie
- Speichere variabel lange Schlüsselteile im inneren Knoten, sobald sie sich als Präfixe für die Schlüssel des zugehörigem Unterbaums abspalten lassen.

Präfix- bzw. Radix Baum (Forts.):

- komplexe Knotenformate und aufwendige Such- und Aktualisierungsoperationen
- erfolglose Suche lässt sich oft schon in einem inneren Knoten abbrechen.

Präfix- bzw. Radix Baum



Danke für Ihre Aufmerksamkeit

Quellen:

- Donald Knuth
  The Art of Computer Programming, Volume 3: Sorting and Searching, Second Edition. Addison-Wesley, 1997

- Fredkin, E.
  Trie Memory. Communication of the ACM. Vol. 3.9 (Sep 1960)

- Aho, A. V., Sethi, R., Ullman, J. D.
  Compilers : Principles, Techniques, and Tools. Addison-Wesley. 1985

- Ranjan Sinha and Justin Zobel, RMIT University and David Ring Palo Alto, CA
  Cache-Efficient String Sorting Using Copying

- Steffen Heinz, Justin Zobel
  School of Computer Science and Information Technology, RMIT University
  Performance of Data Structures for Small Sets of Strings