Algorithmen und Datenstrukturen 2

Sommersemester 2007
7. Vorlesung

Peter F. Stadler

Universität Leipzig
Institut für Informatik
studla@bioinf.uni-leipzig.de

Lempel-Ziv Algorithmen

LZ77 (Sliding Window)

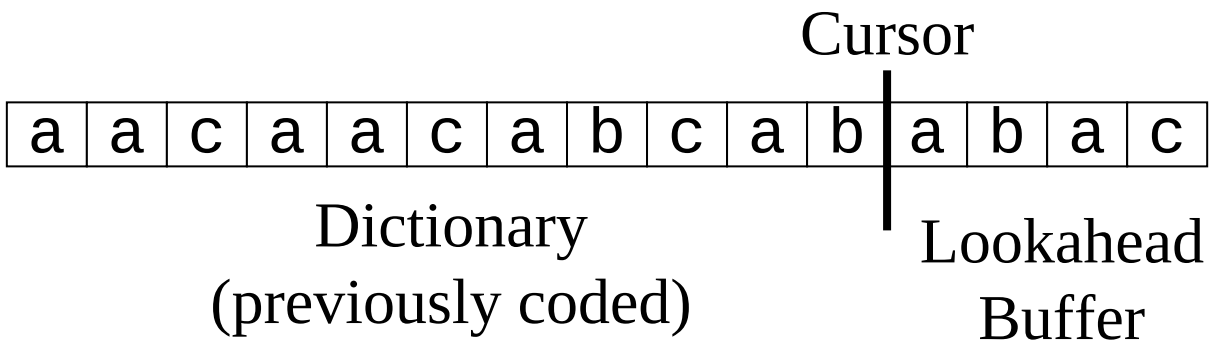
- Varianten: LZSS (Lempel-Ziv-Storer-Szymanski)
- Applications: `gzip`, Squeeze, LHA, PKZIP, ZOO

LZ78 (Dictionary Based)

- Variants: LZW (Lempel-Ziv-Welch), LZC (Lempel-Ziv-Compress)
- Applications: `compress`, GIF, CCITT (modems), ARC, PAK

Normalerweise wurde LZ77 als besser und langsamer als LZ78 betrachtet, aber auf leistungsfähigeren Rechnern ist LZ77 auch schnell.

LZ77: Sliding Window Lempel-Ziv

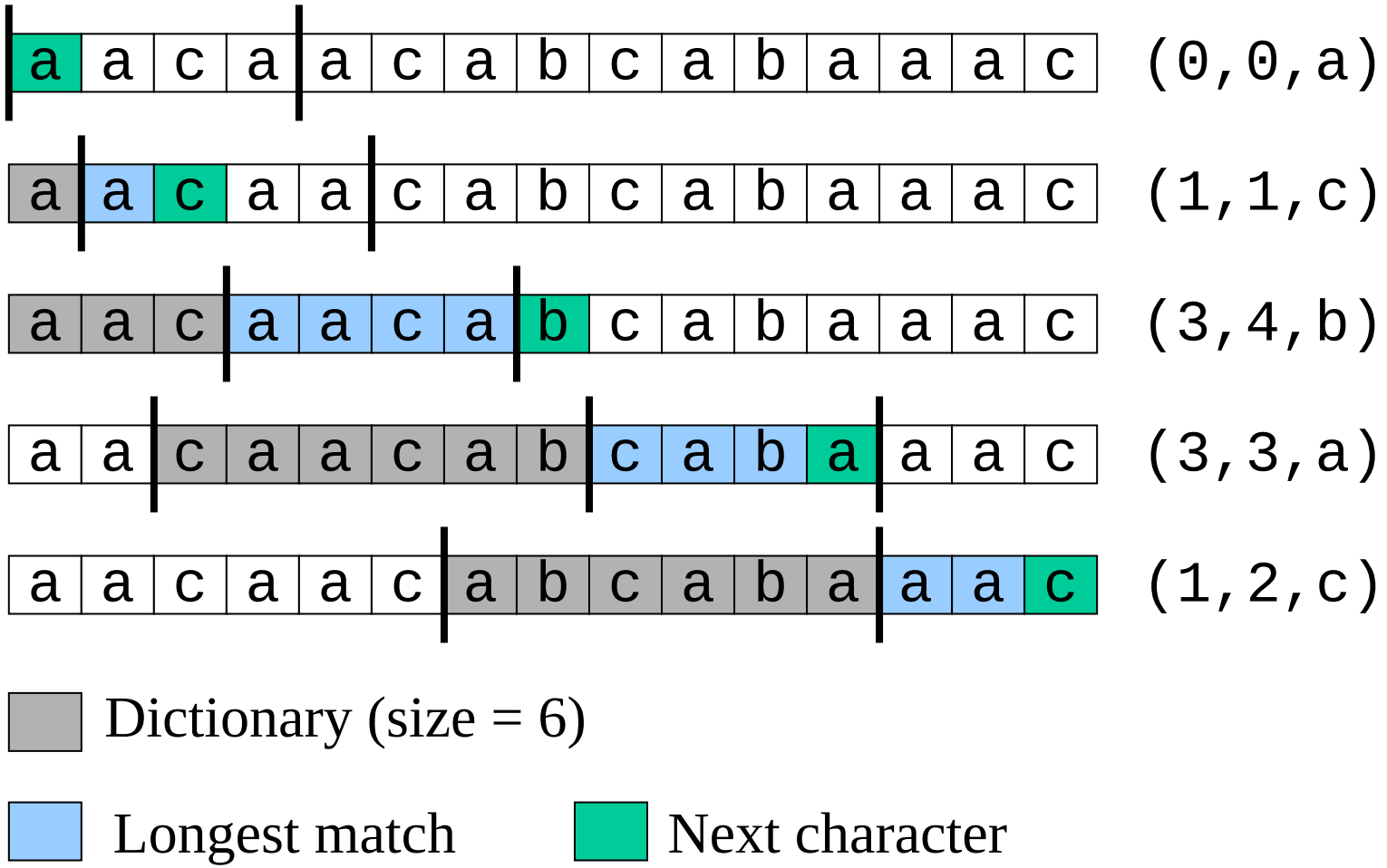


Dictionary- und Buffer-Windows haben feste Länge und verschieben sich zusammen mit dem Cursor.

An jeder Cursor-Position passiert folgendes:

- Ausgabe des Tripels (p,l,c)
  - p = relative Position des longest match im Dictionary
  - l = Länge des longest match
  - c = nächstes Zeichen rechts vom longest match
- Verschiebe das Window um l + 1

LZ77: Example



LZ77 Decoding

Der Decodierer arbeitet mit dem selben Dictionary-Window wie der Codierer

- Im Falle des Tripels (p,l,c) geht er p Schritte zurück, liest die nächsten l Zeichen und kopiert diese nach hinten. Dann wird noch c angefügt.

Was ist im Falle l > p? (d.h. nur ein Teil der zu copierenden Nachricht ist im Dictionary)

- Beispiel dict = `abcd`, codeword = `(2,9,e)`
- Lösung: Kopiere einfach zeichenweise:

```
for (i = 0; i < length; i++)
  out[cursor+i] = out[cursor-offset+i]
```

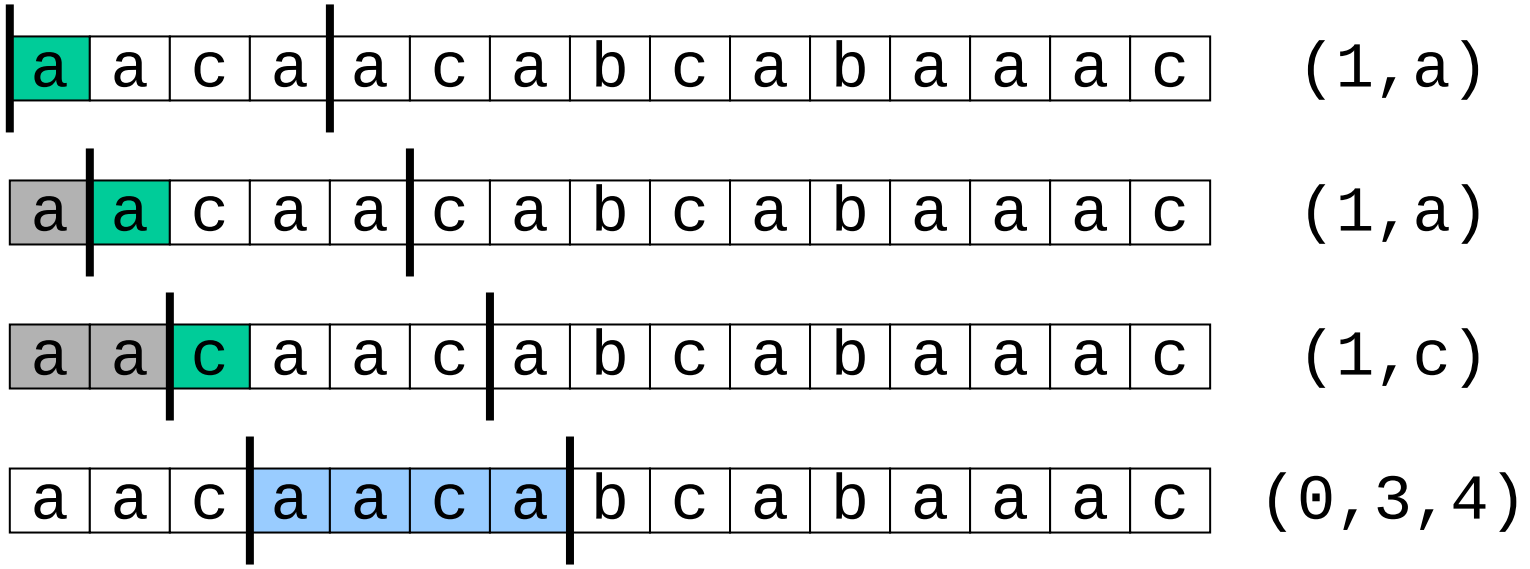
- Out = `abcdcdcdcdcdce`

LZ77 Optimierungen bei gzip I

LZSS: Der Output hat eins der zwei Formate

`(0, position, length)` oder `(1,char)`

Benutze das zweite Format, falls length < 3.



Optimierungen bei gzip II

- Nachträgliche Huffman-Codierung der Ausgabe
- Clevere Strategie bei der Codierung: Möglicherweise erlaubt ein kürzerer Match in diesem Schritt einen viel längeren Match im nächsten Schritt
- Benutze eine Hash-Tabelle für das Wörterbuch.
  - Hash-Funktion für Strings der Länge drei.
  - Suche für längere Strings im entsprechenden Überlaufbereich die längste Übereinstimmung.

Theorie zu LZ77

LZ77 ist asymptotisch optimal [Wyner-Ziv,94]

LZ77 komprimiert hinreichend lange Strings entsprechend seiner Entropie, falls die Fenstergröße gegen unendlich geht.

$$H_n = \sum_{X \in A^n} p(X) \log \frac{1}{p(X)}$$

$$H = \lim_{n \to \infty} H_n$$

Achtung, hier ist wirklich eine sehr große Fenstergröße nötig.

In der Praxis wird meist ein Puffer von 2^{16} Zeichen verwendet.

Die Burrows-Wheeler-Transformation

Michael Burrows und David Wheeler, Mai 1994

Verwendet in bzip

Die Kompression ist sehr gut geeignet für Text, da Kontext berücksichtigt wird.

Die eigentliche Burrows-Wheeler-Transformation ist eine vorgelagerte Umordnung von Textblöcken, die eine spätere Kompression mit einem Move-to-Front-Kodierer und einer weiteren Huffman-Codierung vorbereitet.

BWT: Erster Schritt

Ein Eingabeblock der Länge N wird als quadratische Matrix dargestellt, die alle Rotationen des Eingabeblocks enthält.

Die Zeilen der Matrix werden alphabetisch sortiert.

Die letzte Spalte und die Zeilennummer des Originalblocks werden ausgegeben.

In dieser Ausgabe sorgt ein ähnlicher Kontext von Buchstaben links davor zu langen Runs gleicher Buchstaben.

Daraus lässt sich der Originalblock wieder rekonstruieren (wird hier nicht bewiesen, sondern nur am Beispiel gezeigt).

Die Burrows-Wheeler-Transformation

Vorwärtstransformation von HelloCello

	0	1	2	3	4	5	6	7	8	9
0	H	e	l	l	o	C	e	l	l	o
1	e	l	l	o	C	e	l	l	o	H
2	l	l	o	C	e	l	l	o	H	e
3	l	o	C	e	l	l	o	H	e	l
4	o	C	e	l	l	o	H	e	l	l
5	C	e	l	l	o	H	e	l	l	o
6	e	l	l	o	H	e	l	l	o	C
7	l	l	o	H	e	l	l	o	C	e
8	l	o	H	e	l	l	o	C	e	l
9	o	H	e	l	l	o	C	e	l	l

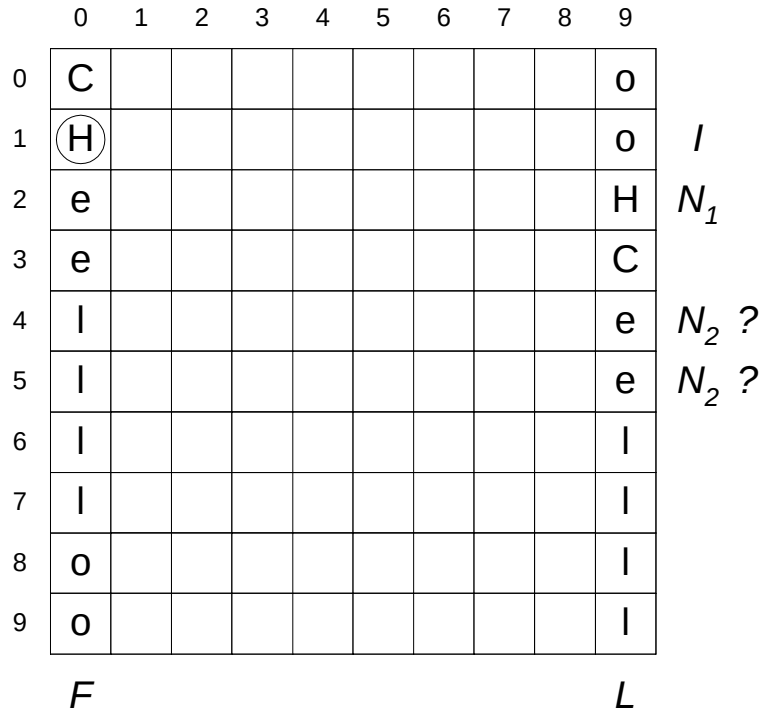
HelloCello

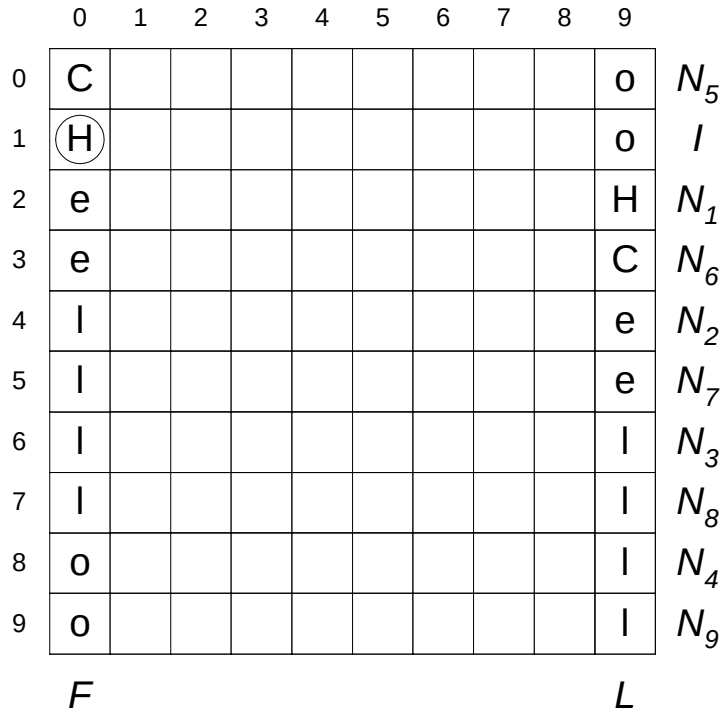
	0	1	2	3	4	5	6	7	8	9	
0	C	e	l	l	o	H	e	l	l	o	
1	H	e	l	l	o	C	e	l	l	o	*I*
2	e	l	l	o	C	e	l	l	o	H	
3	e	l	l	o	H	e	l	l	o	C	
4	l	l	o	C	e	l	l	o	H	e	
5	l	l	o	H	e	l	l	o	C	e	
6	l	o	C	e	l	l	o	H	e	l	
7	l	o	H	e	l	l	o	C	e	l	
8	o	C	e	l	l	o	H	e	l	l	
9	o	H	e	l	l	o	C	e	l	l	
										L	

ooHCeellll

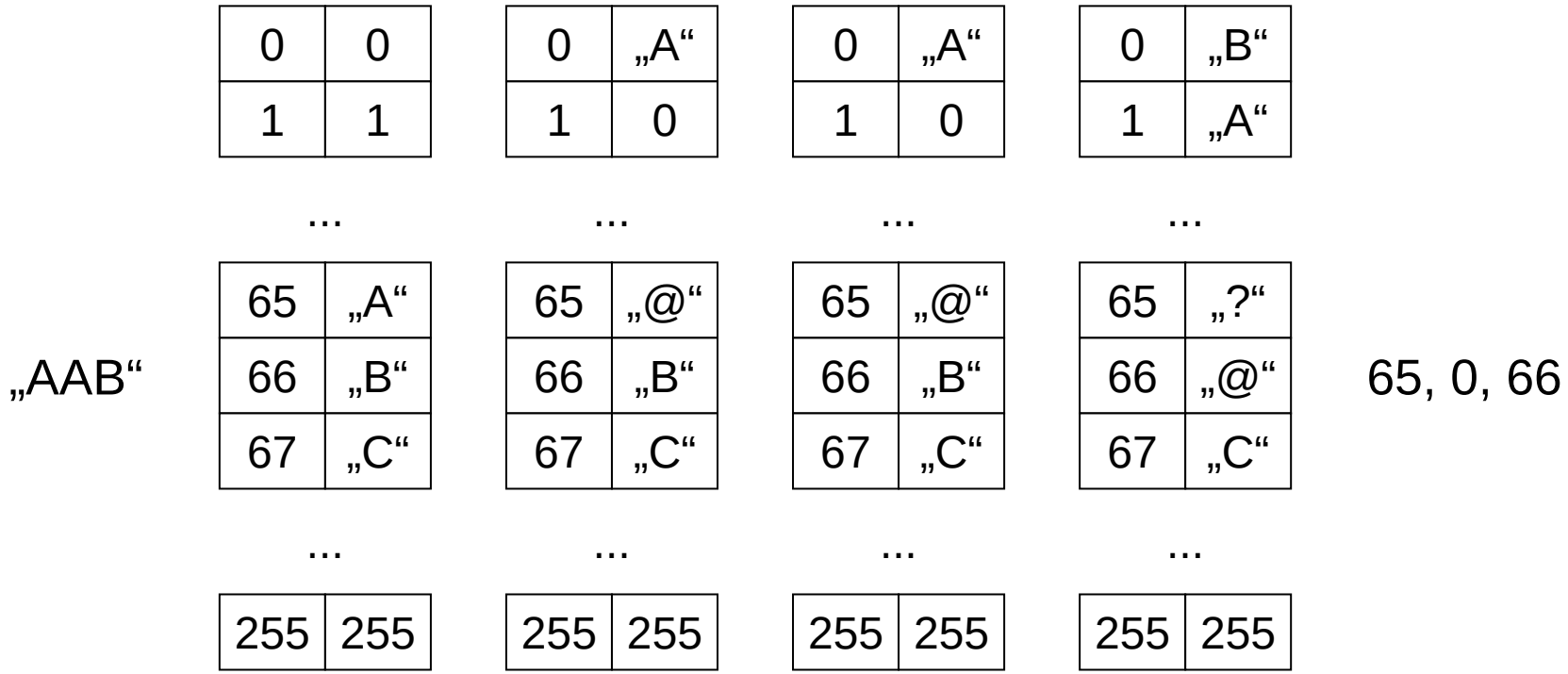
Die Burrows-Wheeler-Transformation

Rücktransformation





MTF: Move-To-Front-Coding



Eingabe: A A B B C C C A A B C C C

Ausgabe: 65 0 66 0 67 0 0 2 0 2 2 0 0

Effektive Datenstrukturen für Lexika

Viele Anwendungen besitzen Wörterbücher, in denen häufig und schnell nachgeschlagen werden muß:

- Textverarbeitung: Rechtschreibkontrolle
- Maschinelle Übersetzung
- Voice recognition

Die nötigen Wörterbücher sind meist sehr groß. Deshalb: Hohe Kompressionsrate ist erwünscht, damit die Wörterbücher im Hauptspeicher gehalten werden können.

Beobachtungen zu Wörterbüchern

Wörterbücher enthalten fast nur Listen von Wörtern oder Wortgruppen:

- Reine Wortlisten für Rechtschreibkontrolle
- Wort-Bigramme oder –Trigramme (evtl. mit Gewichtung) bei Voice recognition
- Wörter und Angabe dazu (z.B. Grammatikangabe)

Beobachtungen

- Wir können die Liste z.B. alphabetisch sortieren, dann gibt es vorn große Wiederholungen, die wir nutzen können.
- Es gibt auch hinten große Regelmäßigkeiten. Nutzung ist zunächst unklar.
- Je größer eine Wortliste ist, desto ausgeprägter ist die Regelmäßigkeit. Möglicherweise lassen sich also größere Listen noch besser komprimieren.
- Häufig müssen Wörterbücher nur einmal erstellt und nicht mehr verändert werden. Die Datenstruktur muß also kein Einfügen unterstützen. Falls nötig, kann ein traditionelles, kleines Nutzerwörterbuch angelegt werden.

Die Idee: Trie und DAWG

Lexikon mit den Wörtern *dance*, *darts*, *start* und *smart*

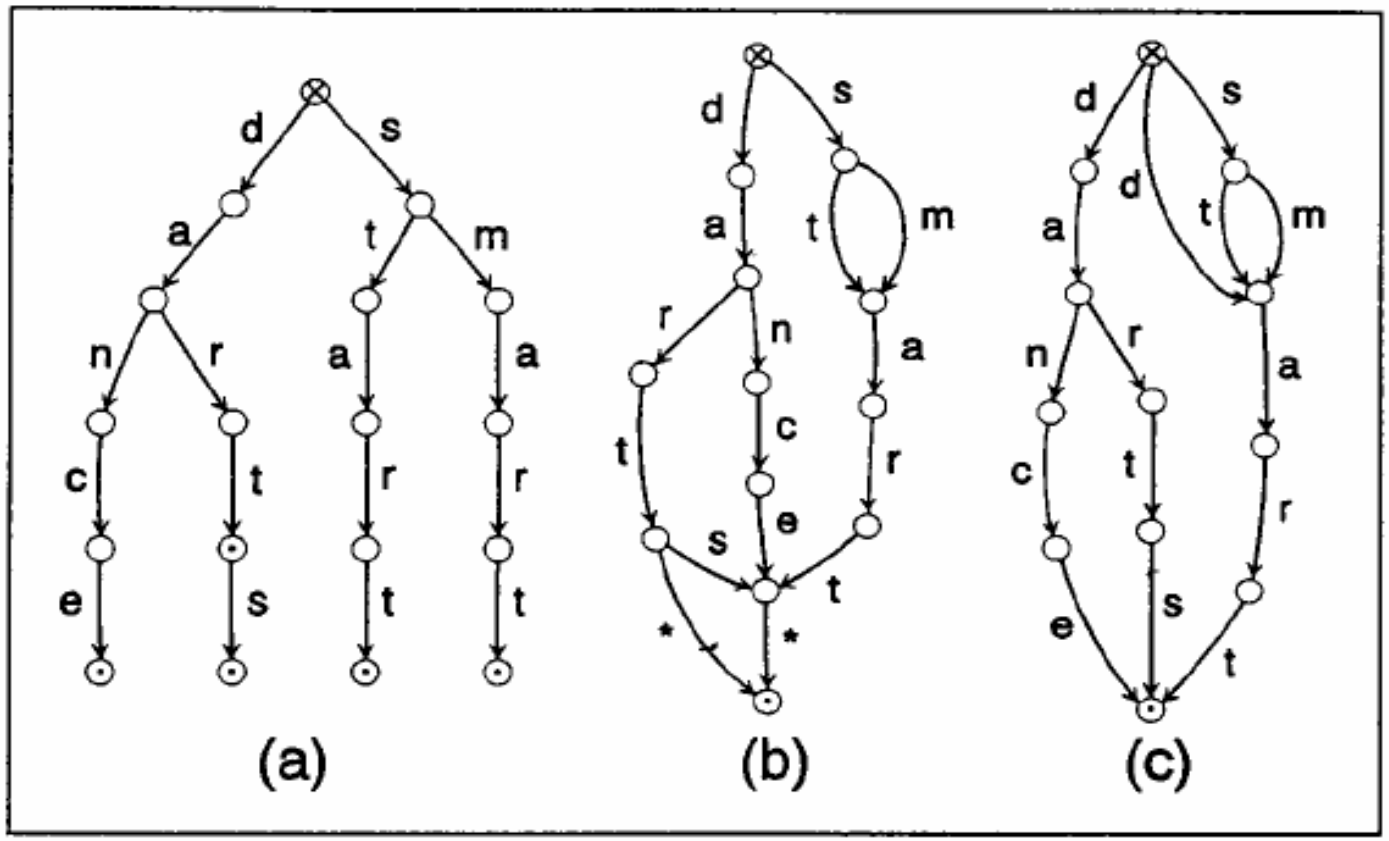


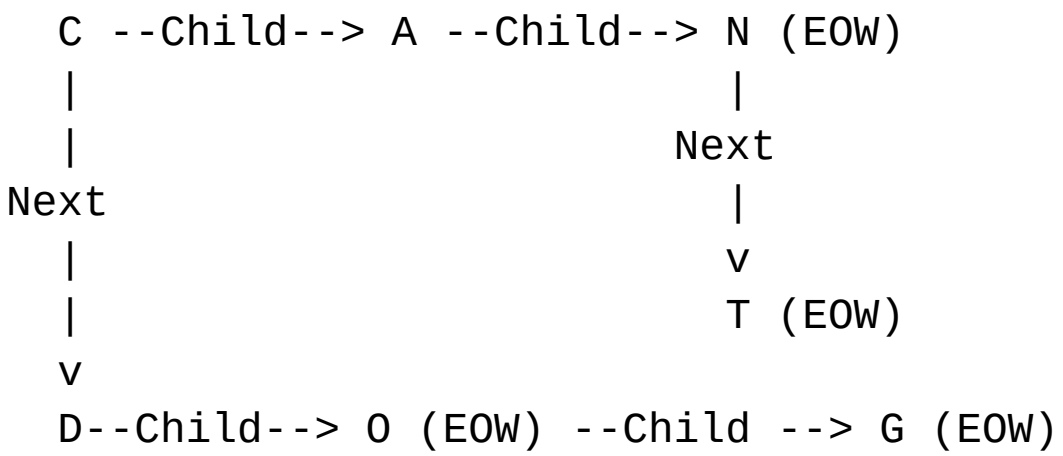
Fig. 1. The lexicon as a trie (a), deterministic DAWG (b) and non-deterministic DAWG (c).

DAWG: Directed Acyclic Word Graph

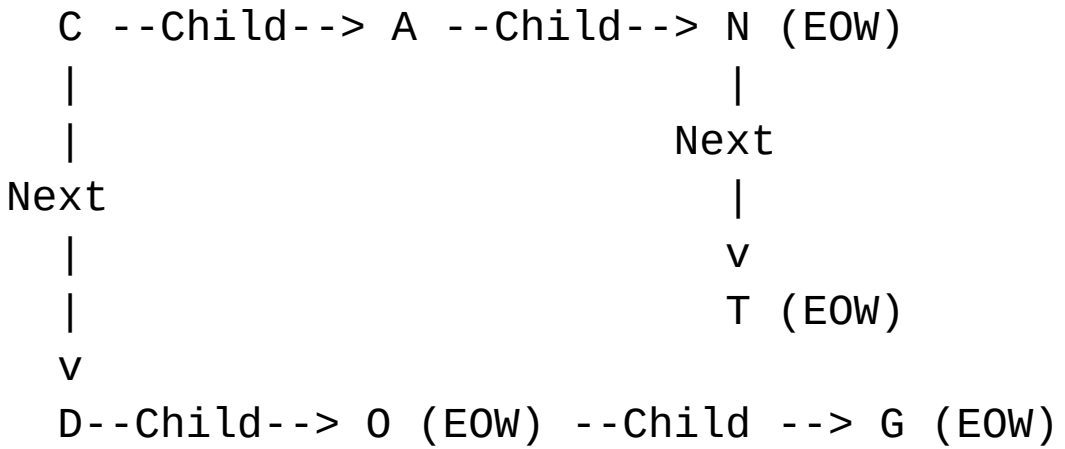
Ein DAWG erlaubt extrem schnelle Suche nach Wörtern. Die Suche erfolgt analog zum Trie: Die aufzusuchende Buchstabenfolge entspricht der Knotenfolge im Baum.

In den folgenden Darstellungen gibt es von jedem Knoten ausgehend Kindknoten (Child) und Nextknoten. Letztere zeigen auf den nächsten Knoten, der sich in der gleichen Hierarchieebene wie der gegenwärtige Knoten befindet. Zusätzlich gibt es Wortende-Markierungen (EOW)

Beispiel: DAWG mit den Wörtern CAT, CAN, DO und DOG:



Suchen von CAT im DAWG



Starten beim Startknoten C.

Dieser ist korrekt, also weiter zu Kind-Knoten. Wir finden A.

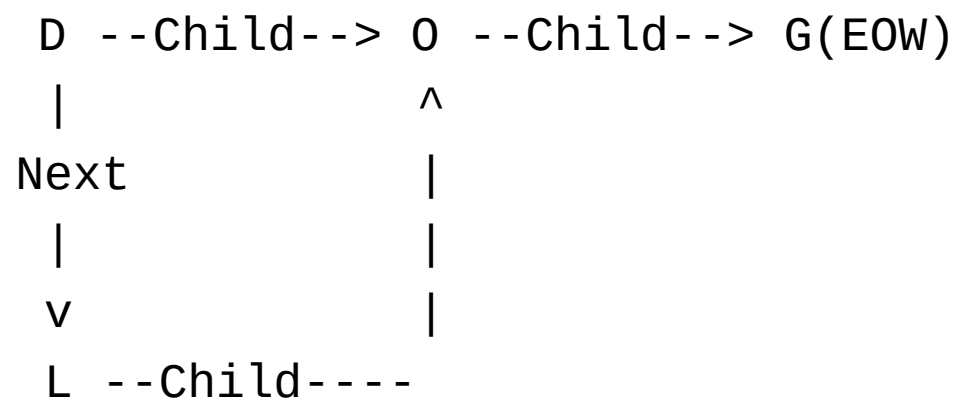
A ist korrekt, also weiter zu Kind-Knoten. Wir finden N.

Das ist falch, also weiter zu Next-Knoten. Wir finden T.

T ist korrekt, wir befinden uns an einem EOW-Knoten, also haben wir das gesuchte Wort gefunden.

Kompression an Wortenden

Zur Kompression an Wortenden werden Knoten weiter unten im Graphen „zusammengehängt“. Das Beispiel enthält DOG und LOG, aber nicht DO.



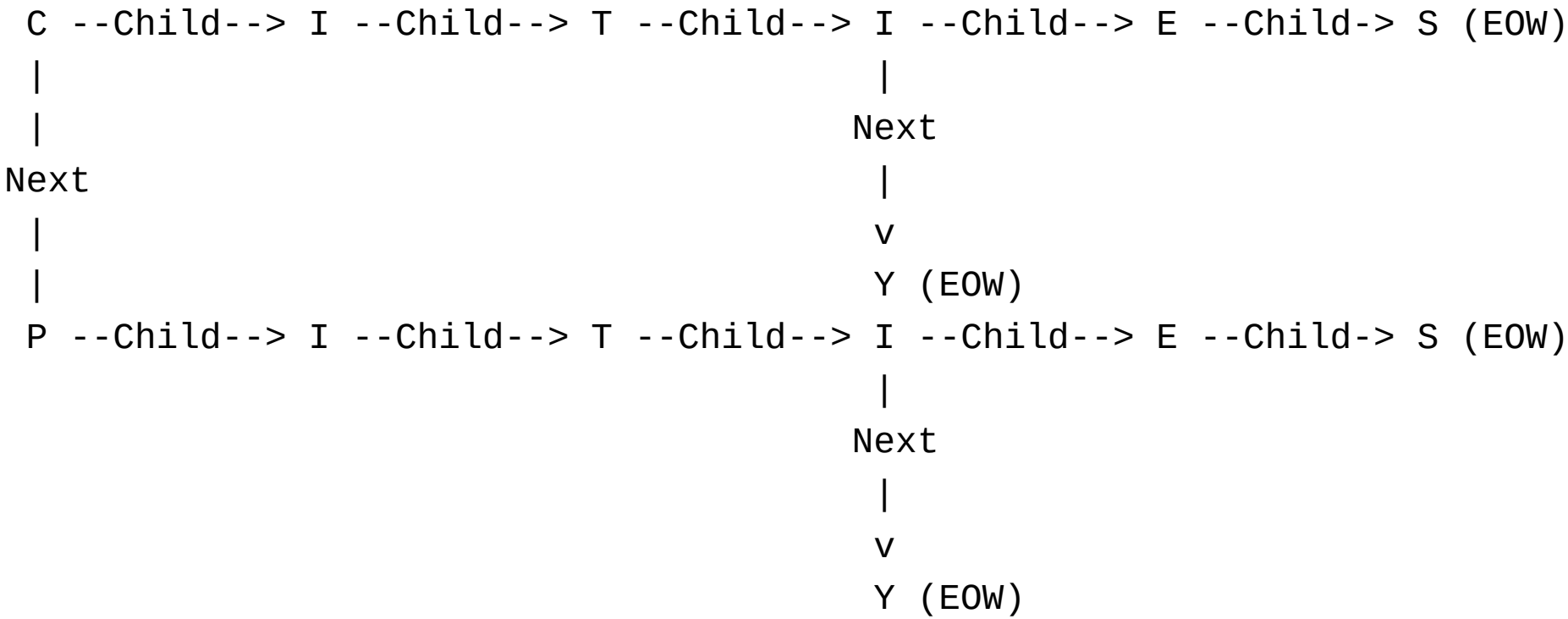
Aufbau eines DAWG

Der Aufbau erfolgt in zwei Schritten:

Schritt 1: Aufbau eines Tries in beliebiger Reihenfolge wie üblich. Einfügen ist an jeder Stelle möglich.

Schritt 2: Zusammenfassen gleicher Wortenden. Dazu dient der folgende Algorithmus, erklärt am Beispiel.

Beispiel: Wörterbuch mit CITIES, CITY, PITIES und PITY, zunächst als Trie.



Verschmelzen von Wortenden

Zu verschmelzende Wortenden müssen völlig übereinstimmen.

Von den Blättern ausgehend wird eine „inverse Höhe“ eingeführt: Blätter haben inverse Höhe 0. Die Inverse Höhe eines Knotens ist 1 + Maximum der inversen Höhen seiner Kinder.

Für alle Paare von Child-Knoten gleicher inverser Höhe wird von 0 beginnend der folgende Verschmelzungsschritt ausgeführt:

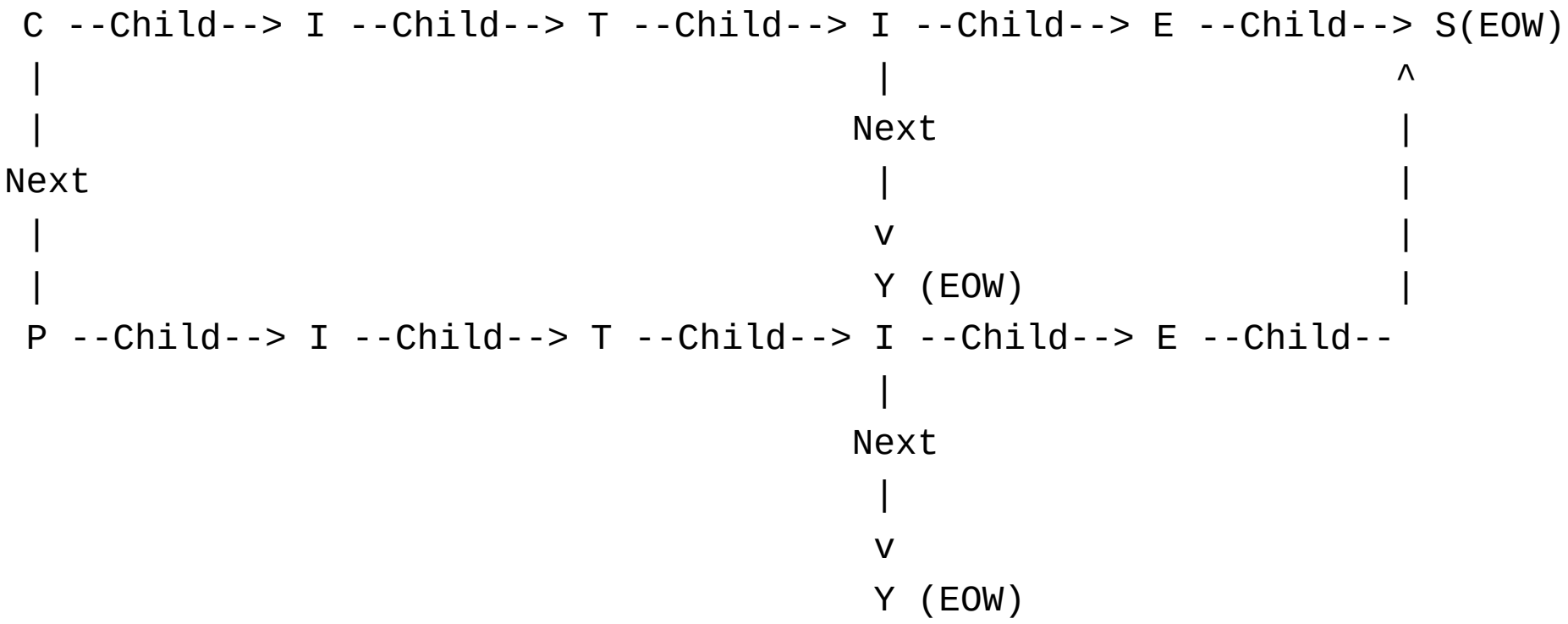
Zwei Knoten werden verschmolzen, falls sie vollkommen identisch sind, d.h.

- Sie tragen den gleichen Buchstaben als Bezeichner.
- Sie tragen beide ein oder beide kein EOW-Flag.
- Falls sie Next-Knoten besitzen, müssen auch diese vollkommen identisch sein.

Erster Verschmelzungsschritt

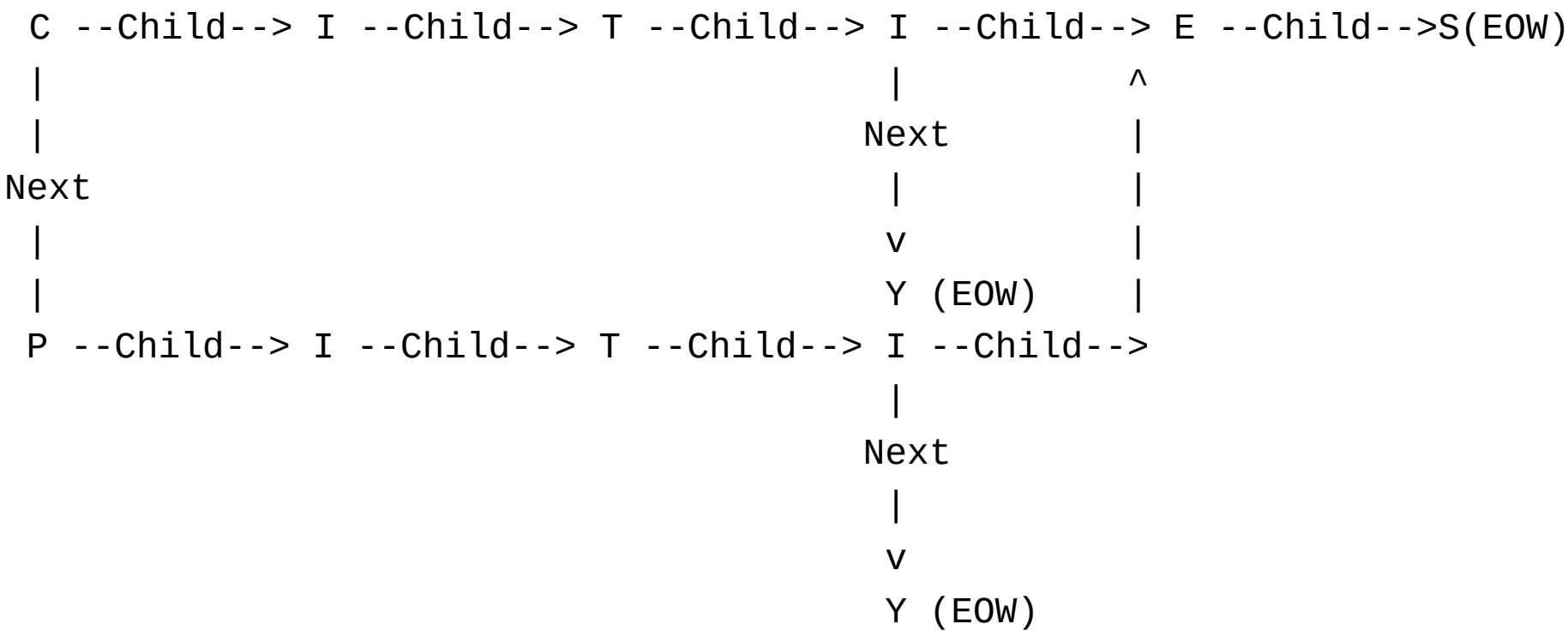
S und Y haben inverse Hohe 0. Verschmelzung der S-Blätter.

Achtung: Die Y-Knoten werden nicht verschmolzen, weil sie keine Child-Knoten sind.



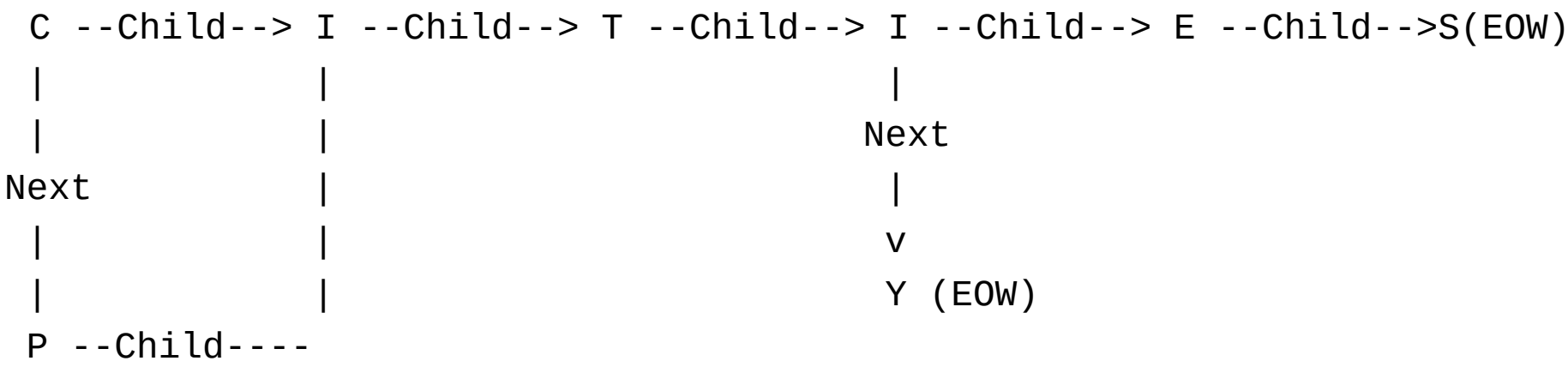
Zweiter Verschmelzungsschritt

Nur E hat inverse Höhe 2. Die E-Knoten werden verschmolzen.



Letzter Verschmelzungsschritt

Nach dem letzten Verschmelzungsschritt:



Dateiformat (byteweise)

Darstellung der Knoten als 4-Byte-Datensätze:

- (2 Byte) Verweis auf ersten Child-Knoten
- (1 Byte) Flags für EOW und letzten Knoten in Next-Kette (1 Byte dafür ist Verschwendung!)
- (1 Byte) Buchstabe des Knotens

Wir benötigen keine Pointer für die Next-Liste, weil diese physich unmittelbar hinter dem ersten Child-Knoten gespeichert werden kann.

Im CITY-PITY-Beispiel erhalten wir:

```
00 00 03 00  Dummy null value, allows 0 to indicate no child
00 03 00 43  C, child at 4-byte-word #3
00 03 02 50  P, child at 4-byte-word #3, end-of-list
00 04 00 49  I, child at 4-byte-word #4
00 05 00 54  T, child at 4-byte-word #5
00 06 00 49  I, child at 4-byte-word #7
00 00 03 59  Y, no child, end-of-word, end-of-list
00 08 00 45  E, child at 4-byte-word #8
00 00 03 53  S, no child, end-of-word, end of list
```

Dateiformat (bitweise)

Angenommen, wir wollen Wörter ohne Berücksichtigung der Groß- und Kleinschreibung speichern (z.B. für Rechtschreibkontrolle)

Wir betrachten die 4 Byte als 32 bit und verwenden sie folgendermaßen:

- 25 bit für die Knoten-Adresse
- 2 bit für die Flags
- 5 bit für die Buchstaben (32 Buchstaben, evtl. mehr ganz seltene Zeichen mit Escape-Sequenz)

Das ermöglicht 33 Millionen Knoten, ausreichend für eine normale Sprache.

Eine Sortierung der Knoten in einer Next-Kette entsprechend der Häufigkeit erlaubt zusätzlich eine Beschleunigung bei der Suche.

Wohin mit Angaben zu Wörtern?

Falls wir Angaben zu Wörtern einfach hinten anfügen und diese nicht zur Struktur passen, geht die Komprimierbarkeit an den Wortenden verloren.

Außer in dem Fall, dass gleiche Wortenden häufig gleiche Angaben implizieren.

Beispiele dafür sind:

- Grammatikangaben zu Wörtern, z.B. Umleitung|f. Das komprimiert wieder gut, da fast alle Wörter auf ...ung weiblich sind, also die Darstellung ...ung|f haben.
- Angaben zur Grundformreduktion: Das Format der Regeln sei |nstring, wobei zur Transformation auf Grundform zuerst hinten n Zeichen entfernt werden und dann (ggf.) string angefügt wird. Also z.B. Tassen|1, Hochhäuser|5haus.
- Wortpaare mit Übergangswahrscheinlichkeiten oder anderen Zahlenangaben. Zunächst werden die Zahlenangaben diskretisiert. Bester Platz ist zwischen den zwei Wörtern, damit die Kompression von hinten und vorn funktioniert..

Anwendung: Rechtschreibkontrolle

Aufgabenstellung: Für ein gegebenes Wort soll festgestellt werden, ob es in einer gegebenen Wortliste ist. Wenn nicht, sollen die ähnlichsten Wörter aus der Wortliste vorgeschlagen werden.

Lösung: Die Wortliste wird als DAWG gespeichert, für ein vorgegebenes Wort kann entschieden werden, ob es sich in der Liste befindet. Es werden zur Eingabe ähnliche Strings erzeugt und in der Liste gesucht. Falls sie vorhanden sind, handelt es sich (per Definition) um ähnliche Wörter.

Ähnliche Strings entstehen, indem man einen oder zwei (selten auch mehrere) der folgenden Editierschritte ausführt. In Klammern die Anzahl der so erzeugten Vorschläge für ein Wort aus n Buchstaben und 30 Buchstaben im Alphabet.

- Weglassen eines Buchstaben [Anzahl: n]
- Einfügen eines Buchstaben an einer beliebigen Stelle [Anzahl: 30(n+1)]
- Austauschen eines Buchstaben an einer beliebigen Stelle [Anzahl: 29n]
- Vertauschen zweier benachbarter Buchstaben [Anzahl: n-1]
- Sprachspezifische Ersetzungen wie f-ph, ss-ß, ö-oe, ...

An Ode to the Spelling Chequer

Von Janet E. Byford

Gefunden auf:
http://www.cooper.com/alan/homonym.html

Prays the Lord for the spelling chequer
That came with our pea sea!
Mecca mistake and it puts you rite
Its so easy to ewes, you sea.

I never used to no, was it e before eye?
(Four sometimes its eye before e.)
But now I've discovered the quay to success
It's as simple as won, too, free!

Sew watt if you lose a letter or two,
The whirled won't come two an end!
Can't you sea? It's as plane as the knows on yore face
S. Chequer's my very best friend

I've always had trubble with letters that double
"Is it one or to S's?" I'd wine
But now, as I've tolled you this chequer is grate
And its hi thyme you got won, like mine.