Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

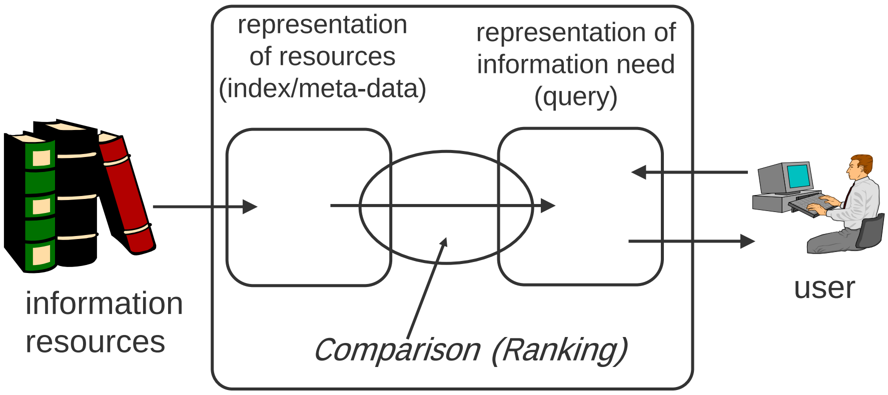
2 Retrieval of Text Information

Fachhochschule Nordwestschweiz
Hochschule für Wirtschaft

Motivation

- Information Retrieval is a field of activity for many years
- IR was long seen as an area of narrow interest. Advent of the Web changed this perception
  - universal repository of knowledge
  - free (low cost) universal access
  - no central editorial board
- Objective: representation, storage, organization of, and access to information items
- Emphasis on the retrieval of information (not data)
- Focus is on the user information need
- The information need is expressed as a query

Generic Schema of an Information System



Information Retrieval systems do not search through the documents but through the representation (also called index, meta-data or description).

source: (Ferber 2004)

Example

D1	D2	D3
Heavy accident Because of a heavy car accident 4 people died yesterday morning in Vienna.	**More vehicles** In this quarter more cars became registered in Vienna.	**Truck causes accident** In Vienna a trucker drove into a crowd of people. Four people were injured.

Information need: documents containing information about car accidents in Vienna where heavy vehicles were involved

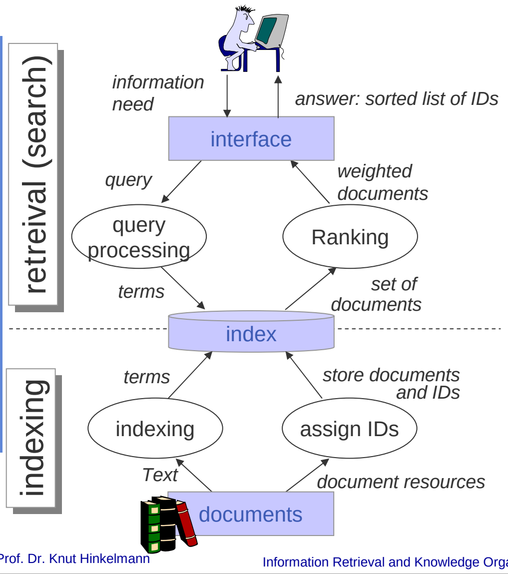
Query: accident heavy vehicles vienna [Search]

Expected result: document D3

but:

- not all terms of the query occur in the document
- the occurring terms „accident“ and „heavy“ also occur in D1

Retrieval System



- Each document represented by a set of representative keywords or index terms
- An index term is a document word useful for remembering the document main themes
- the index is stored in an efficient system or data structured
- Queries are answered using the index
- with the ID die document can be retrieved

Indexing

- manual indexing – key words
  - user specifies key words, he/she assumes useful
  - Usually, key words are nouns because nouns have meaning by themselves
  - there are two possibilities
    1. user can assign any terms
    2. user can select from a predefined set of terms (→ controlled vocabulary)
- automatic indexing – full text search
  - search engines assume that all words are index terms (full text representation)
  - system generates index terms from the words occurring in the text

Automatic Indexing:
1. Decompose a Document into Terms

```
D1: heavy accident because
    of a heavy car accident
    4 people died yesterday
    morning in vienna
D2: more vehicles in this
    quarter more cars
    became registered in
    vienna
D3: Truck causes accident
    in vienna a trucker
    drove into a crowd of
    people four people were
    injured
```

- rules determine how texts are decomposed into terms by defining separators like
  - punctuation marks, blanks or hyphens
- Additional preprocessing, e.g..
  - exclude specific strings (stop words, numbers)
  - generate normal form
    - stemming
    - substitute characters (e.g. upper case – lower case, Umlaut)

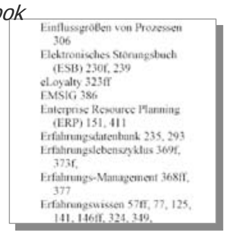
Automatic Indexing
2. Index represented as an inverted list

Term	*Dokument-IDs*
a	D1,D3
accident	D1,D3
became	D2
because	D1
car	D1
cars	D2
died	D1
heavy	D1
in	D1,D2,D3
more	D2
of	D1
people	D1,D3
quarter	D2
registered	D2
truck	D3
vehicles	D2
…	

For each term:

- list of documents in which the term occurs
- additional information can be stored with each document like
  - frequency of occurrence
  - positions of occurrence

In inverted list is similar to an index in a book


Einflussgrößen von Prozessen 306
Elektronisches Störungsbuch (ESB) 230f, 239
eLoyalty 323ff
EMSIG 386
Enterprise Resource Planning (ERP) 151, 411
Erfahrungsdatenbank 235, 293
Erfahrungslebenszyklus 369f, 373f,
Erfahrungs-Management 368ff, 377
Erfahrungswissen 57ff, 77, 125, 141, 146ff, 324, 349,

Index as Inverted List with Frequency

In this example the inverted list contains the document identifier an the frequency of the term in the document.

term	**(*document; frequency)***
a	(D1,1) (D3,2)
accident	(D1,2) (D3,1)
became	(D2,1)
because	(D1,1)
car	(D1,1)
cars	(D2,1)
died	(D1,1)
heavy	(D1,2)
in	(D1,1) (D2,1) (D3,1)
more	(D2,1)
of	(D1,1)
people	(D1,1) (D3,2)
quarter	(D2,1)
registered	(D2,1)
truck	(D3,1)
vehicles	(D2,1)
...	

Problems of Information Retrieval

- Word form
  - A word can occur in different forms, e.g. singular or plural.
  - Example: A query for „car" should find also documents containing the word „cars"
- Meaning
  - A singular term can have different meanings; on the other hand the same meaning can be expressed using different terms.
  - Example: when searching for „car" also documents containing „vehicle" should be found.
- Wording, phrases
  - The same issue can be expressed in various ways
  - Example: searching for „motorcar" should also find documents containing „motorized car"

Word Forms

- **Flexion:** Conjugation and declension of a word
  `car - cars`
  `run – ran - running`
- **Derivations:** words having the same stem
  `form – format – formation`
- **compositions: statements**
  `information management - management of information`

  In German, compositions are written as single words, sometimes with hyphen
  `Informationsmanagement`
  `Informations-Management`

Word Meaning and Phrases

Dealing with words having the same or similar meaning

- Synonyms
  `record – file - dossier`
  `seldom – not often`
- Variants in spelling (e.g BE vs. AE)
  `organisation – organization`
  `night - nite`
- Abbrevations
  `UN – United Nations`
- Polyseme: words with multiple meanings
  `Bank`

2.1 Dealing with Word Forms and Phrases

In principle, we distinguish two ways to deal with word forms and phrases

- Indexing **without preprocessing**
  - All occuring word forms are included in the index
  - Different word forms are unified at search time
    - ➢ **string operations**
- Indexing **with preprocessing**
  - Unification of word forms during indexing
  - Terms normal forms of occuring word forms
  - index is largely independent of the concrete formulation of the text
    - ➢ **computerlinguistic approach**

2.1.1 Indexing Without Preprocessing – String Operations

- Index: contains all the word forms occuring in the document
- Query:
  - Searching for specific word forms is possible (e.g. searching for „cars“ but not for „car“)
  - To search for different word forms string operations can be applied
    - Operators for truncation and masking, e.g.
      - ? covers exactly one character
      - * covers arbitrary number of characters
    - Context operators, e.g.
      - [n] exact distance between terms
      - <n> maximal distance between terms

Index Without Preprocessing and Query

Term	*Dokument-IDs*
a	D1,D3
accident	D1,D3
became	D2
because	D1
car	D1
cars	D2
died	D1
heavy	D1
in	D1,D2,D3
more	D2
of	D1
people	D1,D3
quarter	D2
registered	D2
truck	D3
vehicles	D2
…	

vehicle? car? people [Search]

Truncation and Masking: Searching for Different Word Forms

- **Truncation**: Wildcards cover characters at the beginning and end of words – prefix or suffix
  - `schreib*` finds `schreiben, schreibt, schreibst, schreibe,` …
  - `??schreiben` finds `anschreiben, beschreiben`, but not `verschreiben`
- **Masking** deals with characters in words – in particular in German, declensions and conjugation affect not only suffix and prefix
  - `schr??b*` can find `schreiben, schrieb`
  - `h??s*` can find `Haus, Häuser`
- Disadvantage: With truncation and masking not only the intended words are found
  - `schr??b*` also finds `schrauben`
  - `h??s*` also finds `Hans, Hanse, hausen, hassen` and also words in other languages like `horse`

Context Operators

Context operators allow searching for variations of text phrases

- exact word distance

```
Bezug [3] Telefonat
      Bezug nehmend auf unser Telefonat
```

- maximal word distance

```
text <2> retrieval
      text retrieval
      text and fact retrieval
```

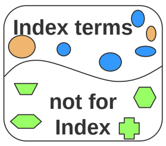
For context operators to be applicable, the positions of the words must be stored in the index

Indexing Without Preprocessing

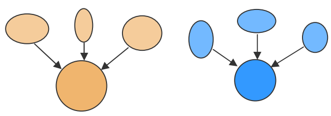
- Efficiency
  - Efficient Indexing
  - Overhead at retrieval time to apply string operators
- Wort forms
  - user has to codify all possible word forms and phrases in the query using truncation and masking operators
  - no support given by search engine
  - retrieval engine is language independent
- Phrases
  - Variants in text phrases can be coded using context operators

2.1.2 Preprocessing of the Index – Computerlinguistic Approach

- Each document is represented by a set of representative keywords or index terms
- An index term is a document word useful for remembering the document's main themes
- Index contains standard forms of useful terms
  1. Restrict allowed terms



  2. Normalisation: Map terms to a standard form



Restricting allowed Index Terms

- Objective:
  - increase efficiency effectivity by neglecting terms that do not contribute to the assessment of a document‘s relevance
- There are two possibilities to restrict allowed index terms
  1. Explicitly specify **allowed** index terms
     - **controlled vocabulary**
  2. Specify terms that are **not allowed** as index terms
     - **stopwords**

Stop Words

- Stop words are terms that are not stored in the index
- Candidates for stop words are
  - words that occur very frequently
    - A term occurring in every document ist useless as an index term, because it does not tell anything about which document the user might be interested in
    - a word which occurs only in 0.001% of the documents is quite useful because it narrows down the space of documents which might be of interest for the user
  - words with no/little meanings
  - terms that are not words (e.g. numbers)
- Examples:
  - General: articles, conjunctions, prepositions, auxiliary verbs (to be, to have)
    - occur very often and in general have no meaning as a search criteria
  - application-specific stop words are also possible

Normalisation of Terms

- There are various possibilities to compute standard forms
  - N-Grams
  - stemming: removing suffixes or prefixes

N-Grams

- Index: sequence of charcters of length N
  - Example: „persons“
  - 3-Grams (N=3): per, ers, rso, son, ons
  - 4-Grams (N=4): pers, erso, rson, sons
- N-Grams can also cross word boundaries
  - Example: „persons from switzerland“
  - 3-Grams (N=3):
    er, ers, rso, son, ons, ns_, s_f, _fr, rom, om_, m_s, _sw,
    swi, wit, itz, tze, zer, erl, rla, lan, and

Stemming

- Stemming: remove suffixes and prefixes to find a comming stem, e.g.
  - remove `–ing` and `–ed` for verbs
  - remove plural `-s` for nouns
- There are a number of exceptions, e.g.
  - `–ing` and `–ed` may belong to a stem as in `red` or `ring`
  - irregular verbs like `go - went - gone, run - ran - run`
- Approaches for stemming:
  - rule-based approach
  - lexicon-based approach

Rules for Stemming in English

Kuhlen (1977) derived a rule set for stemming of most English words:

	Ending	Replacement	Condition
1	ies	y	
2	*XY*es	*XY*	*XY* = *C*o, ch, sh, ss, zz oder *Xx*
3	*XY*s	*XY*	*XY* = *XC*, Xe, *Vy*, *Vo*, oa oder ea
4	ies'	y	
5	*X*es'	*X*	
6	*X*s'	*X*	
7	*X* 's	*X*	
8	*X*'	*X*	
9	*XY*ing	*XY*	*XY*= *CC*, *XV*, *Xx*
10	*XY*ing	*XY*e	*XY*= *VC*
11	ied	y	
12	*XY*ed	*XY*	*XY* = *CC*, *XV*, *Xx*
13	*XY*ed	*XY*e	*XY*= *VC*

X and *Y* are any letters
C stands for a consonant
V stands for any vowel

Source: (Ferber 2003)

Problems for Stemming

- In English a small number of rules cover most of the words
- In German it is more difficult because also stem changes for many words
  - insertion of Umlauts, e.g. `Haus – Häuser`
  - new prefixes, e.g `laufen – gelaufen`
  - separation/retaining of prefix, e.g.
    - `mitbringen – er brachte den Brief mit`
    - `überbringen – er überbrachte den Brief`
  - Irregular insertion of „Fugen“ when building composita
    - `Schwein-kram, Schwein-s-haxe, Schwein-e-braten`
- These problems that cannot be easily dealt with by general rules operating only on strings

Lexicon-based Approaches for Stemming

Principle idea: A lexicon contains stems for word forms

- complete lexicon: for each possible form the stem is stored

```
persons – person              went – go
running – run                 going – go
ran – run                     gone - go
```

- word stem lexicon: to each stem all the necessary data are stored to derive all word forms
  - Distinction of different flexion classes
  - specification of anomalies
  - Example: To compute the stem of `Flüssen`, the last characters are removed successively and the Umlaut is exchanged until a valid stem is found (Lezius 1995)

Fall/Endung	-	n	en	sen	...
normal	Flüssen-	Flüsse-n	Flüss-en	Flüs-sen	...
Umlaut	Flussen-	Flusse-n	**Fluss**-en	Flus-sen	...

Source: (Ferber 2003)

Index with Stemming and Stop Word Elimination

D1: heavy accident because of a heavy car accident 4 people died yesterday morning in vienna

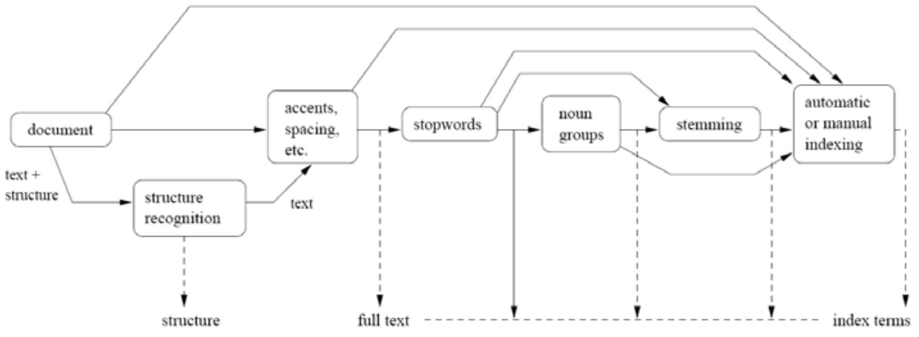
D2: more vehicles in this quarter more cars became registered in vienna

D3: Truck causes accident in vienna a trucker drove into a crowd of people four people were injured

Index:

Terms	***Document IDs***
accident	D1,D3
car	D1, D2
cause	D3
crowd	D3
die	D1
drive	D3
four	D3
heavy	D1
injur	D3
more	D2
morning	D1
people	D1, D3
quarter	D2
register	D2
truck	D3
trucker	D3
vehicle	D2
vienna	D1, D2, D3
yesterday	D1

Indexing Variants



Index as Matrix

	d1	d2	d3	…	dn
t1	$w_{1,1}$	$w_{1,2}$	$w_{1,3}$		$w_{1,n}$
t2	$w_{2,1}$	$w_{2,2}$	$w_{2,3}$		$w_{2,n}$
t3	$w_{3,1}$	$w_{3,2}$	$w_{3,3}$		$w_{3,n}$
…					
tk	$w_{k,1}$	$w_{k,2}$	$w_{k,3}$		$w_{k,n}$

- The index can be represented as a matrix
- The values of the matrix represent, whether a term occurs in a document
- Each column represents a **document vector**
  - $vec(d_j) = (w_{1j}, w_{2j}, \ldots, w_{kj})$
  - the document d_j contains a term t_i if $w_{ij} > 0$
- Each row represents a **term vector**
  - $tvec(t_i) = (w_{i1}, w_{i2}, \ldots, w_{in})$
  - the term t_i is in document d_j if $w_{ij} > 0$

Boolean Document Vectors

d1: heavy accident because of a heavy car accident 4 people died yesterday morning in vienna

d2: more vehicles in this quarter more cars became registered in vienna

d3: Truck causes accident in vienna a trucker drove into a crowd of people four people were injured

	d1	d2	d3
accident	1	0	1
car	1	1	0
cause	0	0	1
crowd	0	0	1
die	1	0	0
drive	0	0	1
four	0	0	1
heavy	1	0	0
injur	0	0	1
more	0	1	0
morning	1	0	0
people	1	0	1
quarter	0	1	0
register	0	1	0
truck	0	0	1
trucker	0	0	1
vehicle	0	1	0
vienna	1	1	1
yesterday	1	0	0

Term weights

- Not all terms are equally useful for representing the document contents
- The *importance* of the index terms is represented by weights associated to them
- Let t_i be an index term
  d_j be a document
  w_{ij} is a weight associated with (t_i,d_j)
- The weight w_{ij} quantifies the importance of the index term for describing the content of the document
  - A simple weight could be the frequency of the term, i.e. how often the term occurs in the document
- (Stop words can be regarded as terms where $w_{ij} = 0$ for every document)

(Baeza-Yates & Ribeiro-Neto 1999)

Classic IR Models - Basic Concepts

- t_i is an index term
- d_j is a document
- N is the total number of docs
- $T = (t_1, t_2, \ldots, t_k)$ is the set of all index terms
- $w_{ij} >= 0$ is a weight associated with (t_i, d_j)
- $w_{ij} = 0$ indicates that term t_i does not belong to d_j
- $vec(d_j) = (w_{1j}, w_{2j}, \ldots, w_{kj})$ is a weighted vector associated with the document d_j
- $g_i(vec(d_j)) = w_{ij}$ is a function which returns the weight associated with pair (t_i, d_j)
- f_i is the number of documents containing term t_i

source: teaching material of Ribeiro-Neto

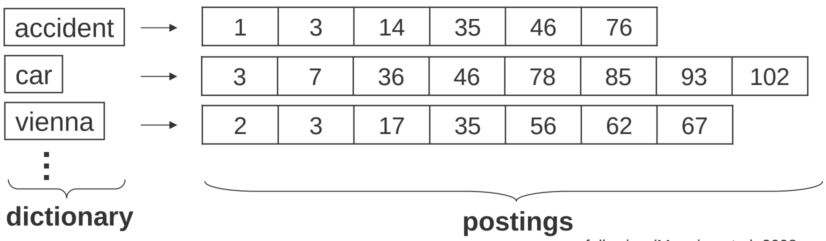
Representing the Index

- Some thoughts about the size of the Index:
- Each document typically contains only a smal fraction of all the terms
  - Assume the document collection (often also called corpus) consists of 1 million documents
  - It is not unrealistic to assume that there are 500'000 different terms
  - Assume each document contains 5000 words (about 10 pages) than even if each would occur only once in a document, 10% of the cells would be zero
  - Realistically, about 99% of the cells or more are zero.
  - A better representation of the matrix would to record onle the things that do occur, that is the 1 position

following (Manning et al. 2008, p. 4)

Representing the Index as an Inverted Index

- The vector space model is usually implemented with an inverted index
- The inverted index maps from the terms to the documents where they occur
- The basic idea of the inverted index is shown in the figure:
  - The dictionary is a sorted set of terms (sometimes also called lexicon or vocabulary)
  - For each term there is a sorted list that records which documents the term occurs in
  - Each item in the list is called "posting", the list is called "posting list"

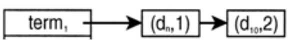


following (Manning et al. 2008, p. 6)

Representing the Index as an Inverted List (cont.)

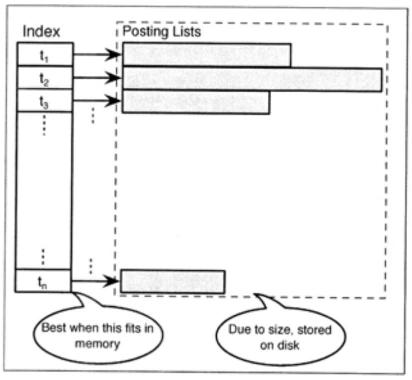
term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
I	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

- The dictionary can have additional information for each term, e.g. the length of the posting (i.e. the number of documents the term occurs in, usually called document frequency)
- The posting can simply be the ID of the document. However, it can also contain additional information, e.g. the frequency of the term in the document or its positions.



following (Manning et al. 2008, p. 8)

Data Structures for the Inverted Index

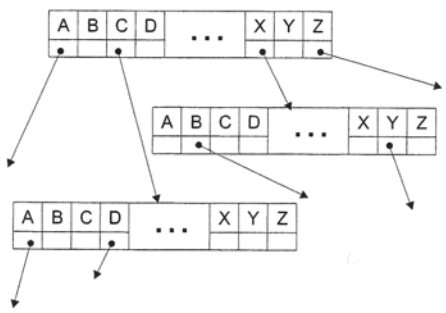


- The dictionary is usually kept in memory
  - For each index term a pointer references to a posting list
  - The posting lists can be stored on disk
- The posting lists can be implemented as
  - linked lists or
  - more efficient data structures that reduce the storage requirements (index pruning)
- To answer a query, efficient retrieval of posting lists is essential
  - Sorting of dictionary and posting lists can be useful for efficient query processing

Source: D. Grossman, O. Frieder (2004) Information Retrieval, Springer-Verlag

Implementing the Term Structure as a Trie

Example: Structure of a node in a trie*)

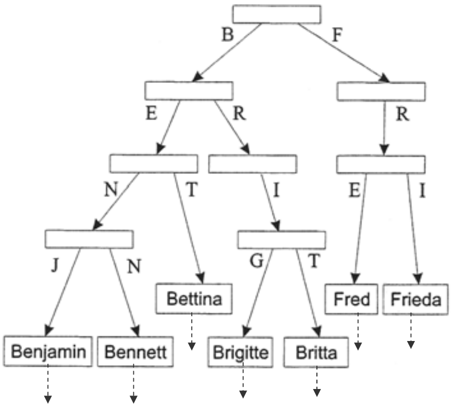


*) the characters and their order are identical for each node. Therefore they do not need to be stored explicitly.

- Sequentially scanning the index for query terms/posting lists is inefficient
- a trie is a tree structure
  - each node is an array, one element for each character
  - each element contains a link to another node

Source: G. Saake, K.-U. Sattler: Algorithmen und Datenstrukturen – Eine Einführung mit Java. dpunkt Verlag 2004

The Index as a Trie

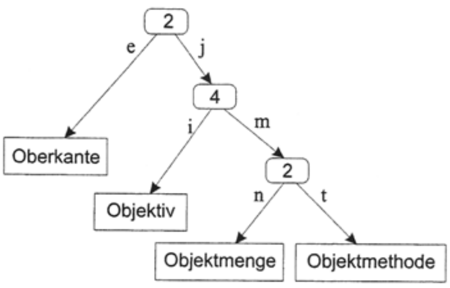


- The leaves of the trie are the index terms, pointing to the corresponding position lists
- Searching a term in in a trie:
  - search starts at the root
  - subsequently for each character of the term the reference to the corresponding subtree is followed until
    - a leave with the term is found
    - search stops without success

(Saake, Sattler 2004)

Patricia Trees

- Idea: Skip irrelevant parts of terms
- This is achieved by storing in each node the number of characters to be skipped.
- Example:



(Saake, Sattler 2004)

Patricia = Practical Algorithm To Retrieve Information Coded in Alphanumeric