Introduction to Information Retrieval and Text Mining Phrase Queries, Dictionaries

Roman Klinger

Institute for Natural Language Processing, University of Stuttgart

2021-10-28

Overview

Outline

Documents

- Definition of the unit in the IR system
  - Paragraph? Sentence? XML elements?
- Problems with parsing
  - RTF, XML, Word, PDF, ...
  - Encodings, Languages

Type/token distinction

- Token
  an instance of a word or term occurring in a document
- Type
  an equivalence class of tokens
- *In June, the dog likes to chase the cat in the barn.*

Normalization

- Need to "normalize" words in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- Two different approaches:
  - Implicitly define equivalence classes of terms.
    (what does implicit mean here?)
  - Asymmetric expansion
    - window $\rightarrow$ window, windows
    - windows $\rightarrow$ Windows, windows
    - Windows (no expansion)
    - More powerful, but less efficient
- Why don't you want to put *window*, *Window*, *windows*, and *Windows* in the same equivalence class?

Tokenization problems: One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
- co-education
- the hold-him-back-and-drag-him-away maneuver
- data base
- San Francisco
- Los Angeles-based company
- cheap San Francisco-Los Angeles fares
- York University vs. New York University

Chinese: No whitespace

莎拉波娃现在居住在美国东南部的佛罗里达。今年４月９日，莎拉波娃在美国第一大城市纽约度过了１８岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

Other cases of “no whitespace”

- Compounds in Dutch, German, Swedish
- Computerlinguistik $\rightarrow$ Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter
  - Which results would you like to get if this was your query??
- $\rightarrow$ leben + versicherung + gesellschaft + angestellter
- Inuit: tusaatsiarunnanngittualuujunga (I can’t hear very well.)
- Many other languages with segmentation difficulties: Finnish, Urdu, . . .

Case folding

- Reduce all letters to lower case
- Even though case can be semantically meaningful
  - capitalized words in mid-sentence
  - MIT vs. mit
  - Fed vs. fed vs. FeD vs. FED
  - ...
- It's often best to lowercase everything since users will use lowercase regardless of correct capitalization.
- Counter example:
  Human Gene name: CES4A, rat gene name: Ces4a

Stop words

- stop words = extremely common words which would appear to be of little value in helping select documents matching a user need
- Examples: *a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with*
- Stop word elimination used to be standard in older IR systems.
- But you need stop words for phrase queries, e.g. *"King of Denmark"*
- Most web search engines index stop words.

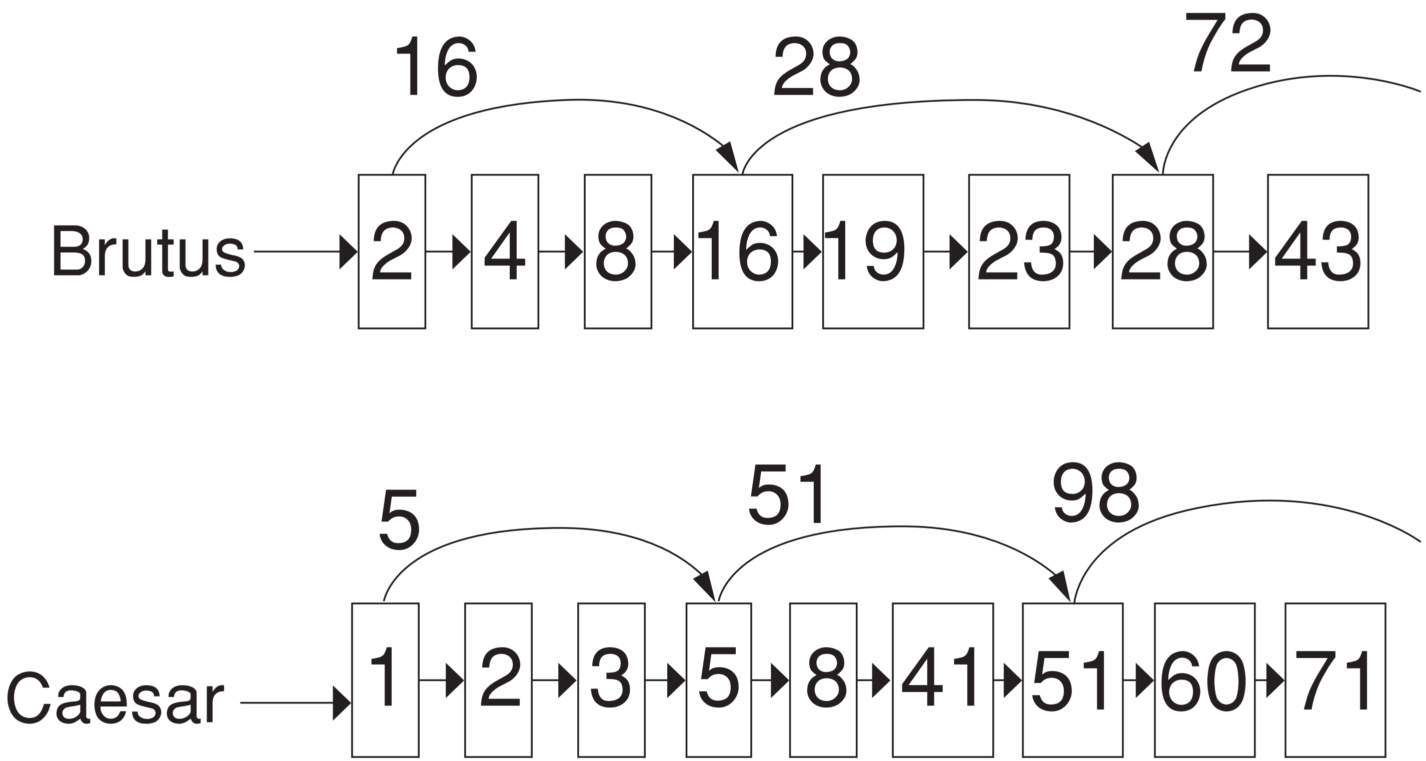
Lemmatization

- Reduce variant forms to base form
- Example: *am, are, is* $\rightarrow$ *be*
- Example: *car, cars, car's, cars'* $\rightarrow$ *car*
- Example: *the boy's cars are different colors* $\rightarrow$ *the boy car be different color*
- Lemmatization implies doing "proper" reduction to dictionary headword form (the lemma).
- Inflectional morphology (*cutting* $\rightarrow$ *cut*) vs. derivational morphology (*destruction* $\rightarrow$ *destroy*)

Stemming

- Definition of stemming: Crude heuristic process that chops off the ends of words in the hope of achieving what "principled" lemmatization attempts to do with a lot of linguistic knowledge.
- Language dependent
- Often inflectional and derivational
- Example for derivational:
  automate, automatic, automation all reduce to *automat*

Skip lists: Example



Take-away

- Soundex: Another term normalization approach
- Positional Index: Answer phrase and proximity queries
- Dictionaries: How to find terms and postings lists (preparation for tolerant retrieval)

Organizational Notes

- Assignment 1 will be published next Tuesday.
- I will combine groups next Tuesday evening, probably by collapsing 1/2, such that online groups for even/odd weeks remain.
- I'll send another mail about that on this Friday (tomorrow).

Outline

Soundex

- Soundex is the basis for finding phonetic (as opposed to orthographic) alternatives.
- Example: *chebyshev* / *tchebyscheff*
- Algorithm:
  - Turn every token to be indexed into a 4-character reduced form
  - Do the same with query terms
  - Build and search an index on the reduced forms

Soundex algorithm

1. Retain the first letter of the term.
2. Change all occurrences of the following letters to '0' (zero): A, E, I, O, U, H, W, Y
3. Change letters to digits as follows:
   - B, F, P, V to 1
   - C, G, J, K, Q, S, X, Z to 2
   - D,T to 3
   - L to 4
   - M, N to 5
   - R to 6
4. Repeatedly remove one out of each pair of consecutive identical digits
5. Remove all zeros from the resulting string; pad the resulting string with trailing zeros and return the first four positions, which will consist of a letter followed by three digits

Klinger

Kl0ng0r

k405206

k4526

k452 (kLng)

Example: Soundex of *HERMAN*

- Retain H
- *ERMAN* → *0RM0N*
- *0RM0N* → *06505*
- *06505* → *06505*
- *06505* → *655*
- Return *H655*
- Note: *HERMANN* will generate the same code

How useful is Soundex?

- Not very – for general information retrieval
- Ok for "high recall" tasks in other applications (e.g., Interpol)
- More sophisticated and better alternatives exist, general idea is the same. (e.g., Zobel and Dart (1996))

Exercise

- Compute Soundex code of your last name

Soundex algorithm

1. Retain the first letter of the term.
2. Change all occurrences of the following letters to '0' (zero): A, E, I, O, U, H, W, Y
3. Change letters to digits as follows:
   - B, F, P, V to 1
   - C, G, J, K, Q, S, X, Z to 2
   - D,T to 3
   - L to 4
   - M, N to 5
   - R to 6
4. Repeatedly remove one out of each pair of consecutive identical digits
5. Remove all zeros from the resulting string; pad the resulting string with trailing zeros and return the first four positions, which will consist of a letter followed by three digits

Outline

Phrase queries

- We want to answer a query such as
  [stanford university]
  – as a phrase.
- "*The inventor Stanford Ovshinsky never went to university*" should not be a match.
- Easily understood concept for end users
- ≈10% of web queries are phrase queries.
  Even more implicit phrase queries.
- Consequence for inverted index:
  it no longer suffices to store docIDs in postings lists. Ideas?
- Two ways of extending the inverted index:
  - biword index (phrase index)
  - positional index

Biword indexes

- Index every consecutive pair of terms in the text as a phrase.
- For example, *Friends, Romans, Countrymen* would generate two biwords: *"friends romans"* and *"romans countrymen"*
- Each of these biwords is now a vocabulary term.
- Two-word phrases can now easily be answered.
- What about phrases with more than two words?

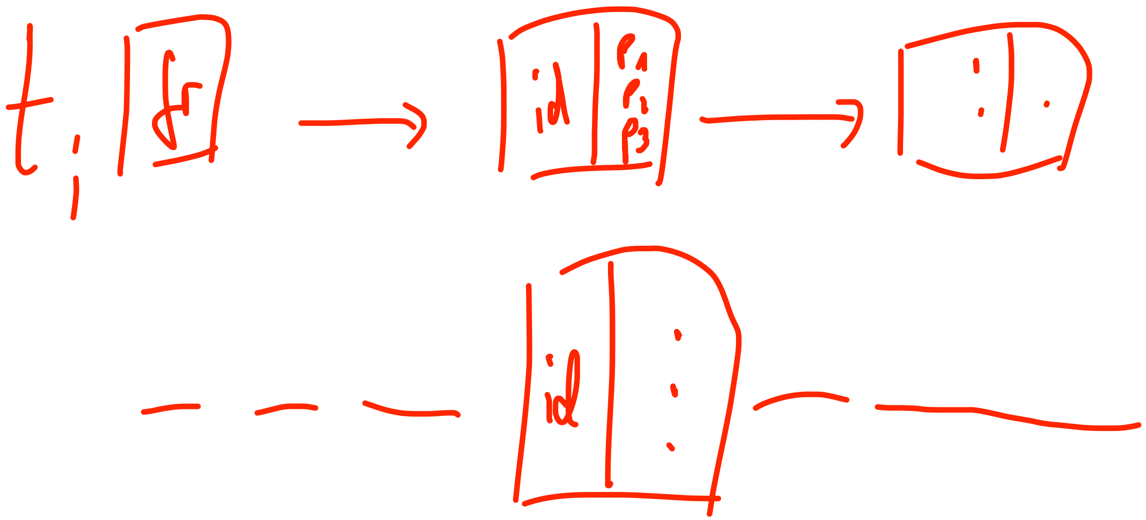
Longer phrase queries

- A long phrase like *"stanford university palo alto"* can be represented as the Boolean query "STANFORD UNIVERSITY" AND "UNIVERSITY PALO" AND "PALO ALTO"
- We need to do post-filtering of hits to identify subset that actually contains the 4-word phrase.

Issues with biword indexes

- Why are biword indexes rarely used?
- False positives, as noted above
- Index blowup due to very large term vocabulary

Positional indexes



Positional indexes

- Positional indexes: more efficient
- Postings lists in a nonpositional index:
  each posting is just a docID
- Postings lists in a positional index:
  each posting is a docID and a list of positions

Positional indexes: Example

Query: “to_1 be_2”

TO, 993427:

⟨ 1: ⟨7, 18, 33, 72, 86, 231⟩;
2: ⟨1, 17, 74, 222, 255⟩;
4: ⟨8, 16, 190, 429, 433⟩;
5: ⟨363, 367⟩;
7: ⟨13, 23, 191⟩; . . . ⟩

BE, 178239:

⟨ 1: ⟨17, 25⟩;
4: ⟨17, 191, 291, 430, 434⟩;
5: ⟨14, 19, 101⟩; . . . ⟩

Document 4 is a match!

Proximity search

- We just saw how to use a positional index for phrase searches.
- We can also use it for proximity search.
- For example: employment /4 place
- Find all documents that contain EMPLOYMENT and PLACE within 4 words of each other.
- *Employment agencies that place healthcare workers are seeing growth* is a hit.
- *Employment agencies that have learned to adapt now place healthcare workers* is not a hit.

Proximity search

- Use the positional index
- Simplest algorithm: look at cross-product of positions of (i) EMPLOYMENT in document and (ii) PLACE in document
- Very inefficient for frequent words, especially stop words
- Note that we want to return the actual matching positions, not just a list of documents.
- This is important for dynamic summaries

Combination scheme

- Biword indexes and positional indexes can be combined.
  - Many biwords are extremely frequent:
    Michael Jackson, Britney Spears etc
  - For these biwords, increased speed compared to positional postings intersection is substantial.
- Combination scheme:
  Include frequent biwords as vocabulary terms in the index. Do all other phrases by positional intersection.

"Positional" queries on Google

- For web search engines, positional queries are much more expensive than regular Boolean queries.
- Let's look at the example of phrase queries.
- Why are they more expensive than regular Boolean queries?
- Can you demonstrate on Google that phrase queries are more expensive than Boolean queries?

Outline

Inverted index

For each term t, we store a list of all documents that contain t.

BRUTUS	⟶	1	2	4	11	31	45	173	174	
CAESAR	⟶	1	2	4	5	6	16	57	132	...
CALPURNIA	⟶	2	31	54	101					
⋮										

dictionary **postings**

Dictionaries

- Dictionary: the data structure for storing the term vocabulary
- Term vocabulary: the data

Dictionary as array of fixed-width entries

- For each term, we need to store a couple of items:
  - document frequency
  - pointer to postings list
  - ...
- For now: Assume that we can store this information in a fixed-length entry.
- Assume that we store these entries in an array.

Dictionary as array of fixed-width entries

	term	document frequency	pointer to postings list
	a	656,265	$\longrightarrow$
	aachen	65	$\longrightarrow$
	...	...	...
	zulu	221	$\longrightarrow$
space needed:	20 bytes	4 bytes	4 bytes

How do we look up a query term q_i in this array at query time?
That is: which data structure do we use to locate the entry (row) in the array where q_i is stored?

Data structures for looking up term

- Two main classes of data structures: hashes and trees
- Some IR systems use hashes, some use trees.
- Criteria for when to use hashes vs. trees:
  - Is there a fixed number of terms or will it keep growing?
  - What are the relative frequencies with which various keys will be accessed?
  - How many terms are we likely to have?

Hashes

- Each vocabulary term is hashed into an integer: its row number in the array
- At query time: hash query term, locate entry in fixed-width array
- Pros: Lookup in a hash is faster than lookup in a tree.
  - Lookup time is constant. (assuming no collisions, worst case linear lookup time!)
- Cons
  - no way to find minor variants (*resume* vs. *résumé*)
  - Lookup time could be higher depending on hash functions
  - no prefix search (all terms starting with *automat*)

Example hash function

- How to calculate a hash function for a string?
- Requirements:
  - hash(s) and hash(s') are not likely to yield the same value ($s \neq s'$)
  - hash is efficiently calculated
- Example: polynomial rolling hash function

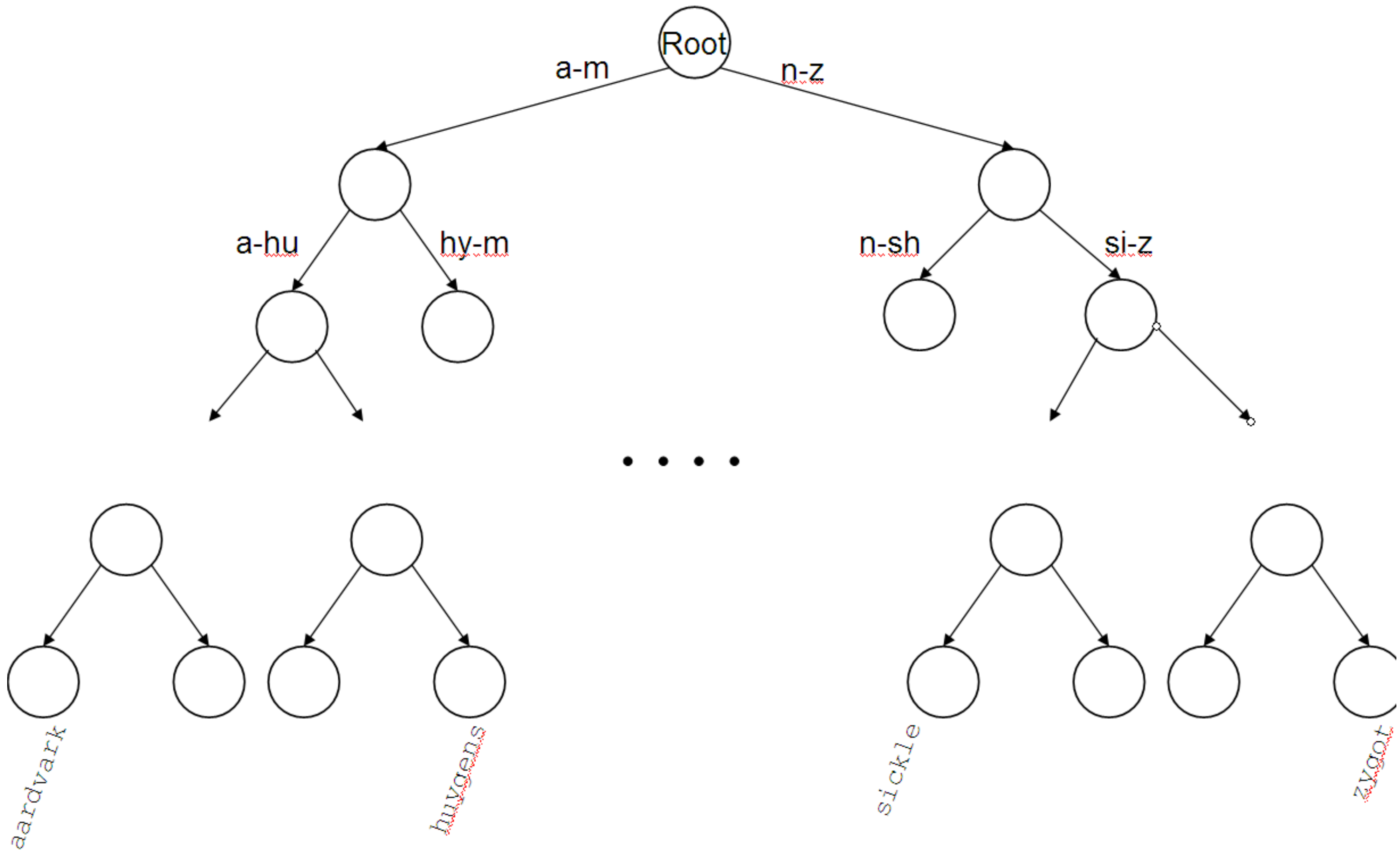
$$\text{hash}(s) = \sum_{i=0}^{n-1} s[i] \cdot p^i \mod m$$

  - p is prime number close to the number of characters in the alphabet (e.g., 31)
  - m is large, because collisions happen with probability $\frac{1}{m}$, but not too large to limit memory needs

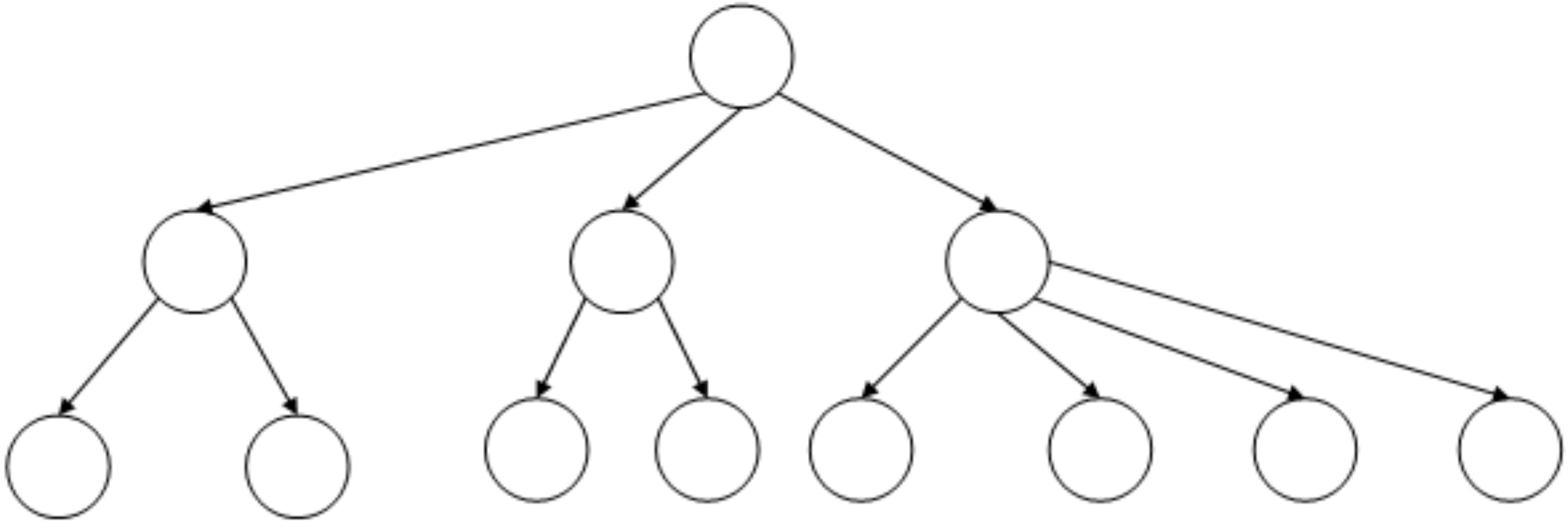
Trees

- Trees solve the prefix problem
  (find all terms starting with *automat*).
- Simplest tree: binary tree
- Search is slightly slower than in hashes: $O(\log M)$,
  where M is the size of the vocabulary.
  (but independent of a hash function and collisions)
- $O(\log M)$ only holds for balanced trees.
- Rebalancing binary trees is expensive.
- B-trees address the rebalancing problem.

Binary tree



B-tree



Take-away

- Soundex: Another term normalization approach
- Positional Index: Answer phrase and proximity queries
- Dictionaries: How to find terms and postings lists (preparation for tolerant retrieval)