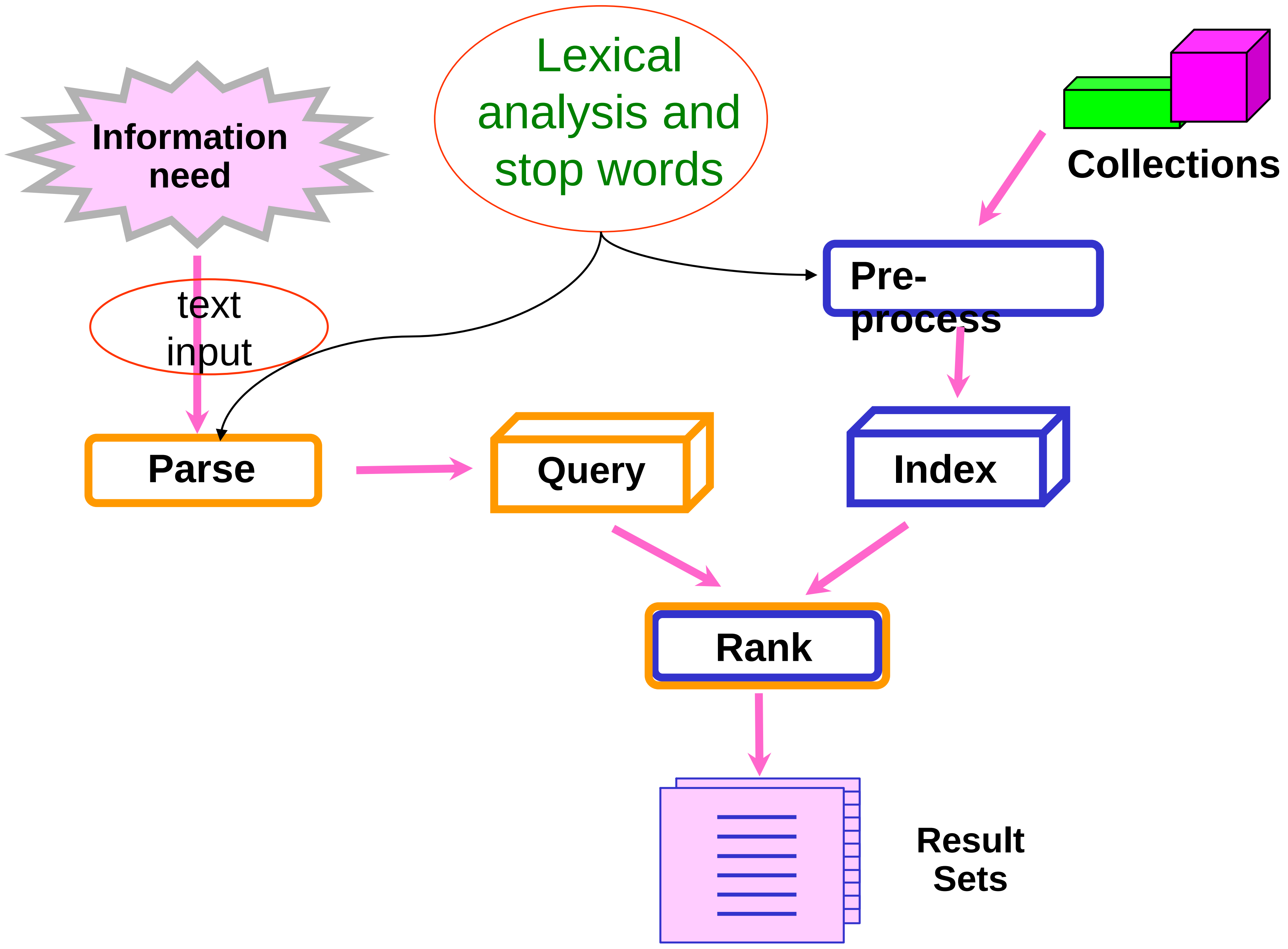
Basic Automatic Indexing Process

1. Parse documents to recognize structure
   – e.g. title, date, other meta data
2. Scan for word tokens (Tokenization)
   – lexical analysis
   – Normalization: deal with numbers, dates, special characters, hyphenation, capitalization, etc.

1. *Stopword* removal
   – based on short list of common words such as “the”, “and”, “or”
   – saves storage overhead of very long indexes
   – can be dangerous (e.g. “flights to London”, “As we may think” by Vannevar Bush’s article)

Basic Automatic Indexing Process

4. Stemming and lemmatization
   – reduce inflectional forms
   – derivationally related forms
5. Weight words
   – using frequency in documents and database
   – frequency data is independent of retrieval model
6. Optional
   – phrase indexing / positional indexing
   – thesaurus classes / concept indexing


Information need
Lexical analysis and stop words
Collections
text input
Pre-process
Parse
Query
Index
Rank
Result Sets

Tokenization: Lexical Analysis

- The stream of characters must be converted into a stream of tokens
  - *Tokens* are groups of characters with collective significance/meaning
  - This process must be applied to both the text stream (*lexical analysis*) and the query string (*query processing*).
  - Often it also involves other preprocessing tasks such as, removing extra white-space, conversion to lowercase, date conversion, normalization, etc.

Issues with Tokenization

- ***Finland's capital*** →
  Finland? Finlands? Finland's?
- ***Hewlett-Packard*** → ***Hewlett*** and ***Packard*** as two tokens?
  - ***State-of-the-art***: break up hyphenated sequence.
  - ***co-education***?
  - ***the hold-him-back-and-drag-him-away-maneuver*** ?
  - It's effective to get the user to put in possible hyphens
- ***San Francisco***: one token or two? How do you decide it is one token?

Tokenization: Numbers

- *3/12/91*   *Mar. 12, 1991*
- *55 B.C.*
- *B-52*

Tokenization: Normalization

- Need to "normalize" terms in indexed text as well as query terms into the same form
  - We want to match ***U.S.A.*** and ***USA***
- We most commonly implicitly define equivalence classes of terms
  - e.g., by deleting periods in a term
  - e.g., converting to lowercase
  - e.g., deleting hyphens to form a term
    - ***anti-discriminatory, antidiscriminatory***

Stop words

- Idea: exclude from the dictionary the list of words with little semantic content: a, and, or , how, where, to, ….
- You need them for:
    - Phrase queries: "King of Denmark"
    - Various titles, etc.: "Let it be", "To be or not to be"
    - "Relational" queries: "flights to London"
  - Google ignores common words and characters such as where, the, how, and other digits and letters which slow down your search without improving the results." (Though you can explicitly ask for them to remain)

Thesauri

- Handle synonyms and homonyms
  - Hand-constructed equivalence classes
    - ***car*** = ***automobile***
    - ***color*** = ***colour***
- Index such equivalences
  - When the document contains ***automobile***, index it under ***car*** as well (usually, also vice-versa)
- Or expand query?
  - When the query contains ***automobile***, look under ***car*** as well

Stemming and Morphological Analysis

- Goal: “normalize” similar words by reducing them to their roots before indexing
- Morphology (“form” of words)
  - Inflectional Morphology
    - E.g,. inflect verb endings
    - Never change grammatical class
      - *dog, dogs*
  - Derivational Morphology
    - Derive one word from another,
    - Often change grammatical class
      - *build, building; health, healthy*

Porter's Stemming Algorithm

- Most common algorithm for stemming English
  - Results suggest it's at least as good as other stemming options
- Conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands/rules
  - sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*

Porter's Stemming Algorithm

- Based on a measure of vowel-consonant sequences
  - measure **m** for a stem is **[C](VC)^{m}[V]** where **C** is a sequence of consonants and **V** is a sequence of vowels (including "y") (**[ ]** indicates optional)
  - **m=0** (tree, by), **m=1** (trouble, oats, trees), **m=2** (troubles, private)
- Some Notation:

*<X>	-->	stem ends with letter X (similarly for other letters)
v	-->	stem contains a vowel
*d	-->	stem ends in double consonant
*o	-->	stem ends with a **cvc** sequence where the final consonant is not w, x, y

- Algorithm is based on a set of condition action rules
  - old suffix --> new suffix
  - rules are divided into steps and are examined in sequence
- Good average recall and precision

Source: https://tartarus.org/martin/PorterStemmer/def.txt

Porter's Stemming Algorithm

- A selection of rules from Porter's algorithm:

STEP	CONDITION	SUFFIX	REPLACEMENT	EXAMPLE
1a	NULL	sses	ss	stresses -> stress
	NULL	ies	I	ponies -> poni
	NULL	ss	ss	caress -> caress
	NULL	s	NULL	cats -> cat
1b	*v*	ing	NULL	making -> mak
	. . .	. . .	. . .	. . .
1b1	NULL	at	ate	inflat(ed) -> inflate
	. . .	. . .	. . .	. . .
1c	*v*	y	I	happy -> happi
2	m > 0	aliti	al	formaliti > formal
	m > 0	izer	ize	digitizer -> digitize
	. . .	. . .	. . .	. . .
3	m > 0	icate	ic	duplicate -> duplic
	. . .	. . .	. . .	. . .
4	m > 1	able	NULL	adjustable -> adjust
	m > 1	icate	NULL	microscopic -> microscop
	. . .	. . .	. . .	. . .
5a	m > 1	e	NULL	inflate -> inflat
	. . .	. . .	. . .	. . .
5b	M > 1, *d, *<L>	NULL	single letter	controll -> control, roll -> roll

Stemming Example

- Original text:

  marketing strategies carried out by U.S. companies for their agricultural chemicals, report predictions for market share of such chemicals, or report market statistics for agrochemicals, pesticide, herbicide, fungicide, insecticide, fertilizer, predicted sales, market share, stimulate demand, price cut, volume of sales

- Porter stemmer results:

  market strategi carri out by U.S. compani for their agricultur chemic , report predict for market share of such chemic , or report market statist for agrochem , pesticid , herbicid , fungicid , insecticid , fertil , predict sale , market share , stimul demand , price cut , volum of sale

Problems with Stemming

- Lack of domain-specificity (“relate”, “relativity”) and context can lead to occasional serious retrieval failures
- Stemmers are often difficult to understand and modify
- Sometimes too aggressive in conflation
  - e.g. “policy”/“police”, “university”/“universe”, “organization”/“organ” are conflated by Porter
- Miss good conflations
  - e.g. “European”/“Europe”, “matrices”/“matrix”, “machine”/“machinery” are not conflated by Porter
- Produce stems that are not words or are difficult for a user to interpret
  - e.g. “iteration” produces “iter” and “general” produces “gener”
- Corpus analysis can be used to improve a stemmer or replace it

Other stemmers

- Other stemmers exist:
  - Lovins stemmer
    - bigger than Porter algorithm, because of its very extensive endings list (about 250 rules)
    - faster
  - Paice/Husk stemmer
  - Snowball
- Full morphological analysis (lemmatization)
  - At most modest benefits for retrieval

Lemmatization

- *Lemmatization, unlike Stemming, reduces inflected words properly ensuring root word belongs to language.*

‘Caring’->Lemmatization->‘Care’
‘Caring’ -> Stemming -> ‘Car’

- *In Lemmatization root word is called **Lemma**.*

Lemmatization

- For example, *runs*, *running*, *ran* are all forms of the word *run*, therefore *run* is lemma of all these words.
- Algorithms refer a dictionary to understand meaning of word before reducing it to its root word, or lemma.
- Refer a dictionary -> time consuming

N-grams

- N-gram: given a string, n-grams for that string are fixed length consecutive overlapping) substrings of length *n*

- Example: “`statistics`”
  - bigrams: `st, ta, at, ti, is, st, ti, ic, cs`
  - trigrams: `sta, tat, ati, tis, ist, sti, tic, ics`

- N-grams can also be used for indexing
  - powerful method for getting fast, “search as you type” functionality
  - index all possible n-grams of the text (e.g., using inverted lists)

N-grams and Stemming (Example)

“`statistics`”

bigrams: `st, ta, at, ti, is, st, ti, ic, cs`

7 unique bigrams: `at, cs, ic, is, st, ta, ti`

“`statistical`”

bigrams: `st, ta, at, ti, is, st, ti, ic, ca, al`

8 unique bigrams: `al, at, ca, ic, is, st, ta, ti`

Now use Dice’s coefficient to compute “similarity” for pairs of words”

$$S = \frac{2C}{A + B}$$

where A is no. of unique bigrams in first word, B is no. of unique bigrams in second word, and C is no. of unique shared bigrams. In this case,

(2*6)/(7+8) = .80.

Now we can form a *word-word similarity matrix* (with word similarities as entries). This matrix is used to cluster similar terms.

N-gram indexes

- Enumerate all *n*-grams occurring in any term
- *e.g.*, from text "***April is the cruelest month***" we get *bigrams:*

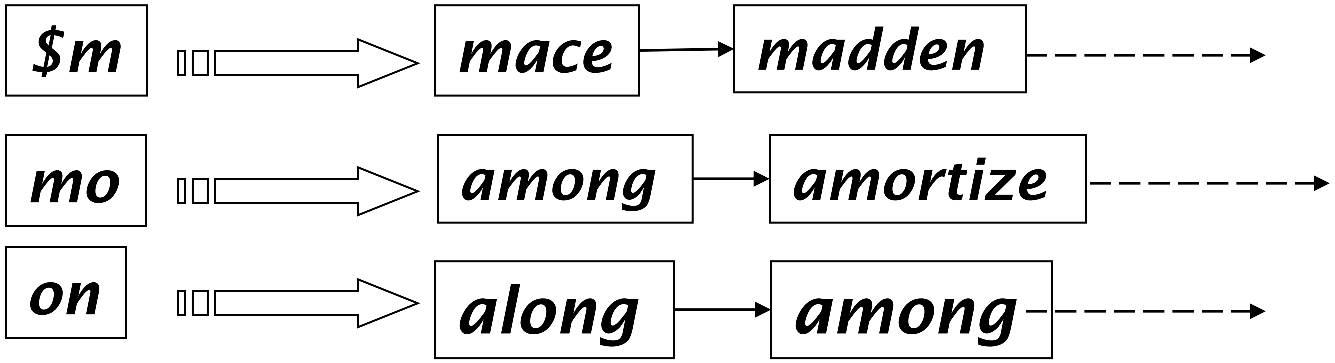
a, ap, pr, ri, il, l, i, is, s, t, th, he, e, $c, cr, ru, ue, el, le, es, st, t$, m, mo, on, nt, h

  – $ is a special word boundary symbol

- Maintain a *second* inverted index *from bigrams to dictionary terms* that match each bigram.

Bigram index example

- The *n*-gram index finds *terms* based on a query consisting of *n*-grams (here *n*=2).



Using N-gram Indexes

- Wild-Card Queries
  - Query ***mon**** can now be run as
    - ***$m** AND **mo** AND **on***
  - Gets terms that match AND version of wildcard query
  - Surviving enumerated terms are then looked up in the term-document inverted index.
- Spell Correction
  - Enumerate all the *n*-grams in the query
  - Use the *n*-gram index (wild-card search) to retrieve all lexicon terms matching any of the query *n*-grams
  - Threshold based on matching *n*-grams and present to user as alternatives
    - Can use Dice or Jaccard coefficients

Using N-gram Indexes

- no semantic meaning, so tokens not suitable for representing concepts
- can get false hits, e.g., searching for “retail” using trigrams, may get matches with “retain” since it includes all trigrams for “retail”

Positional indexes

- In the postings, store, for each ***term*** the position(s) in which tokens of it appear:

  <***term***, number of docs containing ***term***;

  doc1: position1, position2 … ;

  doc2: position1, position2 … ;

  etc.>

- Allows for "proximity" searches
  - e.g., find Term1 *within k words* of Term2