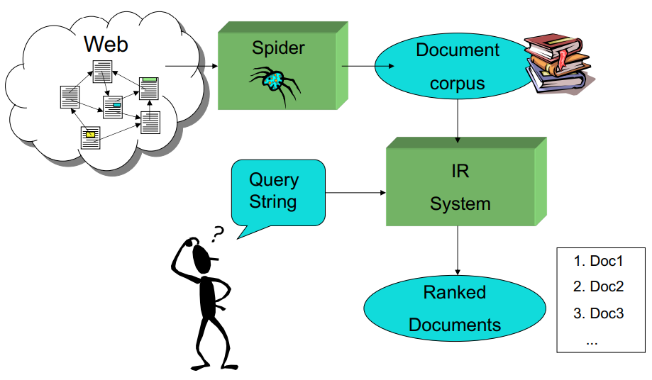
Information Retrieval

**Information retrieval** is finding material of an unstructured nature that satisfies an information need from within large collections. Uses:

* Web search
* E-mail search
* Searching your laptop
* Corporate knowledge bases
* Legal information retrieval

**Document**

* Significant texto contente
* Some structure

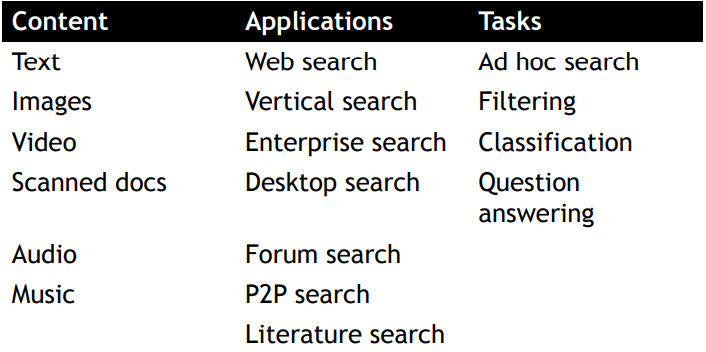


**Web search** – application of IR to HTML documents

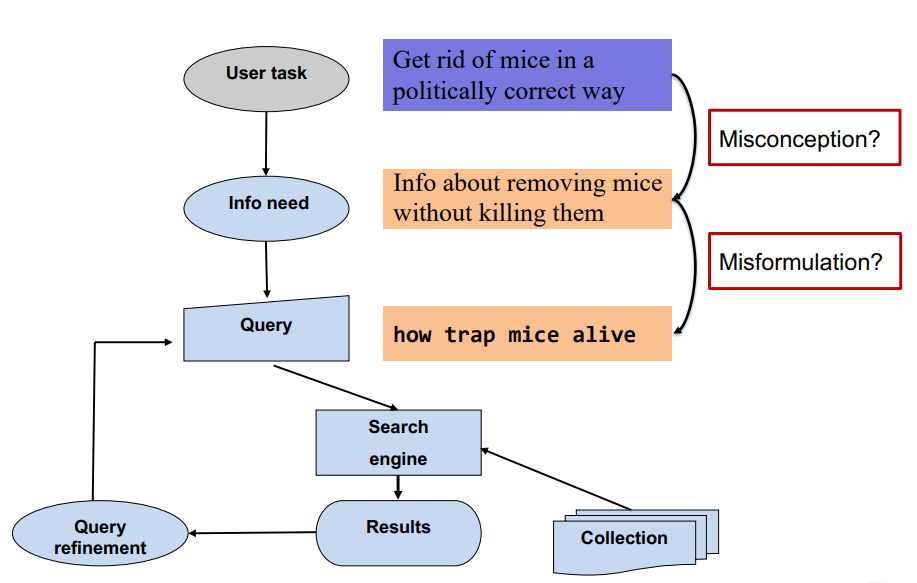
Differences:

* Must assembler document corpus by spidering the web
* Can exploit the structural layout information in HTML
* Can exploit the link structure of the web
* Documents change uncontrollably

**Dimensions of IR**



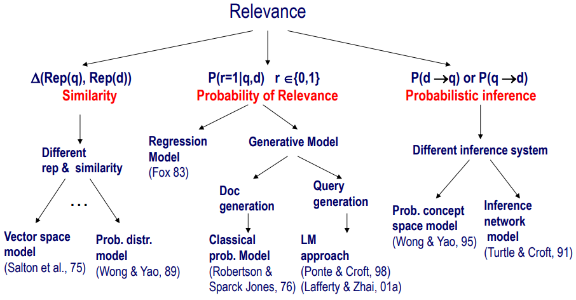
**The classic search model**



**How good are the retrieved documents?**

* Preicsion – fraction of retrieved documents that are relevant to the user’s information need
* Recall – fraction of relevant docs in collection that are retrieved

**Big Issues in IR**

* Relevance – a relevant document contains the information that a person was looking or when they submitted a query to the search engine; ranking is based on retrieval models, each defines a view of relevance and most descrive statistical properties of text
* Evaluation – experimental procedures and measures for comparing system output with user expectations; recall and precision are two exemples of measures used for evaluation
* Information needs
  + Search evaluation is user-centerd
  + Keyword queries are often poor descriptions of actial information needs
  + Intereaction and contexto are importante for understanding user intent
  + Query refinement techniques such as query expansion, query sugesstion, relevance feedback improve ranking

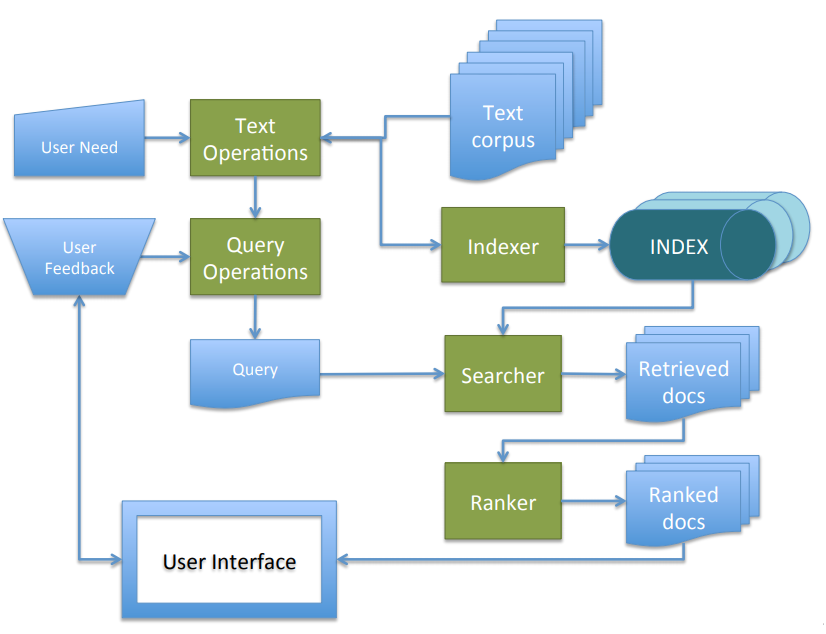
**Search Engine** is the pratical application of information retrieval techniques to large scale texto collections. Web search engines are best-known exemples

**Big issues in Search Engines**

Include not only the main issues of IR but also some others:

* Performance – efficient search and indexing
* Incorporating new data – coverage and freshness
* Scalability – growing with data and users
* Adaptability – tuning for applications
* Specific probles – e. g. spam

**IR System Architecture**



**IR Directions: NLP**

* Methods for determining the sense of an ambiguous word based on context
* Methods for identifying specific pieces of information in a document
* Methods for answering specific NL questions from document corpora

**IR Directions: Machine Learning**

* Text categorization
  + Automatic hierarchical classification
  + Adaptive filtering/routing/recommending
  + Automated spam filtering
* Text Clustering
  + Clustering of IR query results
  + Automatic formation of hierarchies
* Learning for information extraction
* Text mining

**Key Terms in IR**

* Query – a representation of what a user is looking for
* Document – na information entity that the user wants to retrieve
* Collection – a set of documents
* Index – a representation of information that makes querying easier
* Term – word or concept that appears in a document or query

**Incidence vectors**

Create 0/1 vectors for each term, then take the necessary vectors and bitwise AND them.

**Inverted index**

For each term *t*, store a listo f all documents that contain *t*. Identify each document by a document serial number. A variable-size posting list is a good data structure for this task.

**Initial stages of texto processing**

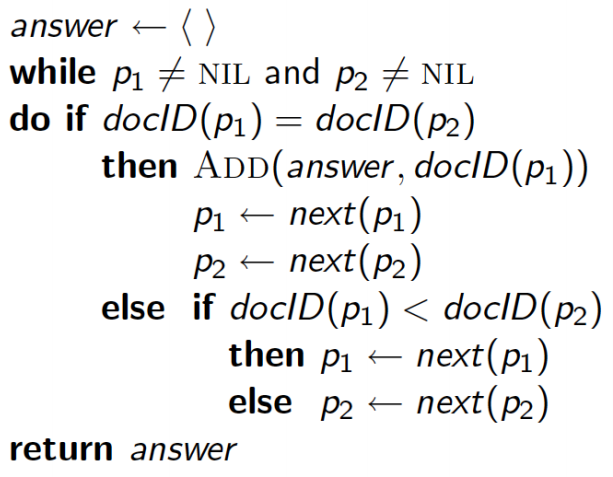
* Tokenization – cut character sequence into word tokens
* Normalization – map texto and query term to same form
* Stemming – we may wish different forms of root to match
* Stop words – we may omit very common words (or not)

**Indexer steps:**

* Token sequence – sequence of pairs (term, docID)
* Sort – sort by terms (and then docID)
* Dictionary and postings
  + Multiple term entries in a single document are merged
  + Split into dictionary and postings
  + Document frequency information is added

**Query processing: AND**

Merge the postings of the ANDed terms.



**Boolean queries: Exact match**

The Boolean retrieval model is being able to ask a query that is a Boolean expression.

For a query that is an AND of n terms, start the process in order of increasing frequency.

Estimate the size of each OR by the sum of its documents’ frequencies and then process in increasing order of OR sizes.

**Parsing Complications: Format/Language**

Documents being indexed can include documents from many different languages. Sometimes a documento r its componentes can contain multiple languages/formats.

**Tokenization**

A token is an instance of a sequence of characters.

Each such token is now a candidate for an index extry, after further processing:

* Alphanumeric sequence with 3 or more characters (Too simple for search applications or even large-scale experiments)

Issues:

* Whether a collection of words is one token only, or multiple tokens (e. g. “San Francisco” or “Hewlett-Packard”)
* Numbers
* Language dependency
  + German noun compounds are not segmented
  + Chinese and Japanese have no spaces between words
  + Japanese has multiple alphabets intermingled
  + Arabic is basically written right to left, but with certain items like numbers written left to right and words are separated, but letter forms within word form complex ligatures

**Stop words**

With a stop list, you can exclude from the dictionary entirely the commonest words.

**Normalization to terms**

We may need to “normalize” words in indexed texto as well as query words into the same form. A ter mis a (normalized) word type which is an entry in our IR system dictionary. This is often done by:

* Deleting periods to form a term
* Deleting hyphens to form a term

In other languages things like accents and umlauts should be handled (generally words with those things are normalized to a format where those things don’t appear)

Data forms must also be normalized. Normalization, just like Tokenization, is language dependent. An alternative to equivalence classing is to do asymmetric expansion.

**Thesauri and soundex**

Handling of synonyms and homonyms, as well as spelling mistakes.

**Lemmatization**

Reduce inflectional/variant forms to base form. Lemmatization implies doing proper reduction to dictionary headword form

**Stemming**

Many morphological variations of words. In most cases, these have the same or very similar meanings. Stemmers attempt to reduce morphological variations of words to a common stem. Can be done at indexing time or as part of query processing (like stopwords). Two types:

* Dictionary-based
* Algorthimic stemmers

**Porter’s algorthim**

Commonet algorithm for stemming English. Conventions plus 5 phases of reductions.

Other stemmers:

* Lovins stemmer
* Paice/Husk stemmer
* Snowball
* Full morphological analysis (lemmatization)

**Typical rules in porter**

* ational -> ate
* tional -> tion
* sses -> ss
* ies -> i
* weight of word sensitive rules (m > 1) ement -> EMPTY\_STRING

**Phrase queries** – this requires more than <term : docs> entries

**Phrases**

Approaches to recognize phrases:

* Use word n-grams (n-words)
* Store word positions in indexes and use proximity operators in queries

**A first attempt: Biword indexes**

Index every consecutive pair of terms in the text as a phrase. Each of these biwords is now a dictionary term. Two-word phrase query-processing is now immediate.

**Longer phrase queries**

Longer phrases are processed by combining shorter phrases. Without the docs we cannot verify that the docs matching the boolean queries do contain the phrase. (See 3rd pdf for a better understanding (page 37)).

**Issues for biword indexes**

* False positives
* Index blowup due to bigger dictionary
* Biword indexes are not the standard solution but can be part of a compound strategy

**Solution 2: Positional indexes**

In the postings, store for each term the position(s) in it appear(s)

**Positional index size**

You can compress position values/offsets. Nevertheless, a positional index expands postings storage substantially, yet, it is now standardly used because of the power and usefulness of phrase and proximity queries. Rules of thumb:

* A positional index is 2-4 as large as a non-positional index
* Positional index size 35-50% of volume of original texto
* All of this holds for english-like languages

**Processing a phrase query**

Extract inverted index entries for each distinct term. Merge their doc:position lists to enumerate all positions with

**Skip pointers**

When processing two posting lists in the merge process, we ignore all subconsequent numbers of the lowest next number between the two lists up to the larger next number of those two lists. (See example in the 3rd pdf, page 50).

But there is a tradeoff, the more skips have more successful skips, but they also have lots of comparsions. Fewer skips have less comparsions, but lower chances of skipping.

Placing them:

* Simple heuristic: for postings of length L, use √L evenly-spaced skip pointers.