Searching

- A breathtaking number of information resources are available: books, databases, the web, newspapers, ...
- To locate relevant information, we need to be able to search these resources, which often are written texts:
  - Searching in a library catalogue (e.g., using IUCAT)
  - Searching the web (e.g., using Google)
  - Advanced searching in text corpora (e.g., using regular expressions in Opus)

Types of data

1. **Structured data**: organized & searchable by categories: author, title, subject, and so forth.
   - Useful when the searcher knows the general topic that they are searching through
   - e.g., for duck-billed platypuses, look through zoology and animal topics
   - Problem: someone has to structure it
2. **Unstructured data**: much more available (e.g., the internet)
   - Keyword search can be highly effective
3. **Semi-structured data**: contains some categorization, but lacks much in the way of structure
   - Structure and format are often inconsistent, even for the same type of document (e.g., blogs, web reviews)

Basic searching with IUCAT

- Literal strings are composed of characters which naturally must be in the same character encoding system (e.g. ASCII, ISO8859-1, UTF-8) as the strings encoded in the database.
- For literal strings, the search engine does not distinguish between upper and lower-case letters (i.e. they aren't so literal after all ;-)!
- Adjacent words are searched as a phrase.
  - art therapy
  - vitamin c
- **Stop words** are ignored in searches, unless enclosed in double quotes *(a, an, as, at, be, but, by, do, for, if, in, is, it, of, on, the, to)*

Structured data

- To find articles, books, and other library holdings, a library generally provides:
  - a **database** containing information on its holdings, and
  - a **database frontends** for users to interact with the database.
    - e.g., IUCAT, WorldCat
- Users search for the occurrence of literal strings occurring in the author, title, keywords, call number, etc. associated with an item held by the library.
Information need

Searching involves information need: the information a searcher is seeking

- Information need gets translated into a query, hoping to capture that information need
- This is an imperfect process

1. a. Information need: one or more Russian translations of the English word table
   b. Possible query: Russian translation table

Information need is unambiguous; query is ambiguous
- Could be looking for a table/chart of Russian translations (which may not include the word table)

Evaluation search results

Use of information need can be seen in the evaluations for the Text REtrieval Conference (TREC, http://trec.nist.gov/)

<b>Number: 363</b>
<b>Title: Hubble Telescope Achievements</b>

Description: Identify positive accomplishments of the Hubble telescope since it was launched in 1991.

Narrative: Documents are relevant that show the Hubble telescope has produced new data, better quality data than previously available, data that has increased human knowledge of the universe, or data that has led to disproving previously existing theories or hypotheses. Documents limited to the shortcomings of the telescope would be irrelevant. Details of repairs or modifications to the telescope without reference to positive achievements would not be relevant.

Evaluation measurements

Numerical question: how many pages is the search engine getting right?

- precision: How many of the pages returned are the ones we want?
  - e.g., Google gives me 400 hits for a query, 200 of which are related to the topic I want; precision = 50%.
- recall: How many pages on the topic we wanted were actually given? (hard to calculate for web searching)
  - e.g., Google gave me 200 pages I wanted, but there were actually 1000 pages on that topic out there somewhere on the internet; recall = 20%.

Precision & recall are often competing priorities: e.g., 100% precision at the cost of lack of coverage
Searching the web

A computer user

- wants to find something on “the web”, i.e., in files accessible via the hypertext transfer protocol (http) protocol on the internet
- goes to a search engine = program that matches documents to a user’s search requests
- enters a query = request for information
- gets a list of websites that might be relevant to the query
- evaluates the results: either picks a website with the information looked for or reformulates the query

Search engines

- Search engines (e.g., Google)
  - store a copy of all web pages
  - create an index to provide efficient access to this large number of pages (e.g., Google currently searches over 8 billion pages)
  - compute a rank for each web page to be able to rank the query results
- Some ways in which search engines can differ:
  - Treatment of word tokens:
    - stemming: treat bird and birds as the same or not
    - capitalization: treat trip and Trip the same or not
  - Options for searching: use of operators or special interface for advanced searching
  - how search results are ranked and potentially clustered (group similar results)

Google: Operators

- +: Require a word to occur in the result
e.g., To find a restaurant that serves both tofu and BBQ one could try
  +tofu +BBQ
- -: Disallow a word from occurring in the result
e.g., As a potatoes purist :-), I search for
  potatoes -potatoes
- *: Include synonyms of the word
e.g., looking for sites on What Cheer, iowa with
  "What Cheer"
- * match anything (wildcard), e.g., “who’s that”

See: http://www.google.com/help/refinesearch.html

Search engine indexing

Manning et al (2008)

As a search engine crawls the web, it builds a term-by-document matrix

- shows which terms (i.e., words) appear in which documents
- e.g., for some mystery novels:

<table>
<thead>
<tr>
<th>Affair at Styles</th>
<th>Secret Adversary</th>
<th>Sherlock Holmes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poitot</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Sherlock</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>adventure</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>exceedingly</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>strychnine</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>subsided</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- 1 denotes that the word appears in that document, and
- a 0 denotes that it does not

Inverted indexing

Matrix-building is done offline, i.e., before a search engine is queried

- We derive a representation which is faster for page look-up, namely an inverted index
- e.g., assuming every document has a unique ID:

| Poitot → 1, 4, 13, 15, 45, ... |
| Sherlock → 3, 111, ... |
| adventure → 1, 2, 3, 4, 5, 9, 15, ... |
| exceedingly → 1, 3, 11, 25, ... |
| strychnine → 1, 15, 60, ... |
| subsided → 1, 3, 12, 13, 25, ... |

- Each term now points to a list of documents that it appears in
- To search for, e.g., strychnine, we have an immediate list of documents that it appears in
Ideally, the webpages matching a query are returned as an ordered list based on a page’s relevance.

How can a search engine, which does not understand language, determine the relevance of a particular page?

Information used to rank results

- Counting the number of links to and from a page, to determine how popular a page is.
  - As a result, unpopular or new pages require a more specific query to be found.
  - Keeping track of the nature of links to a page; linked pages might be thematically related.
  - e.g., Even if I never mention Sinclair Lewis on a page describing his book Babbit, it can be identified if many Sinclair Lewis sites link to my page.
- bonuses/penalties for known sites of high/low quality
- looking for keywords in metadata
- counting how often a web result was clicked on by a user (click-through measurement)
- various secret ingredients

Weblinking

Example of how pages link

In this example, pages X, Y, and Z all link to page A.

Are these links better or worse than the links to page B?

That depends on how popular, or authoritative, the links are.

Weblinking

Figuring popularity

In order to compare how popular website A is as compared to how popular website B is, we can add up how popular each incoming site is.

It’s like each site that links to A gets to vote for A, but they get so many votes based on how popular each one of them is.

e.g., X casts 15 votes for A, Y casts 10, and Z casts 20:

But now A has 45 votes. That’s too many!

Weblinking

Factoring out outgoing links

The solution Google uses is to spread out each page’s votes through all the pages it links to.

So, after spreading votes out among their different webpages, let’s say A’s final score is: 12.

On its own, the number means nothing.
But if we compare the number with website B, which we’ll say has a score of 10, A is more “authoritative”

To sum, there are two main things to consider when calculating a ranking for a website based on its weblinks:

- Links coming in
- Links going out

The formula (for Google) is as follows, where $R(A)$ means rank of page $A$; $C(X)$ means number of pages going out of $X$

$$R(A) = \frac{R(X)}{C(X)} + \frac{R(Y)}{C(Y)} + \frac{R(Z)}{C(Z)}$$
**Weblinking**

Explanation of weblinks ranking formula

1. We add up all the pages coming into page A because to know how popular A is, we need to know how popular everyone else thinks it is.
2. We divide by the pages going out of X, Y, and Z because we're spreading out its weight among all the pages they link to.
   - If we didn't divide, page A would have a huge ranking.
This tells us how “popular” a site is, which is one factor used in ranking results

---

**Semi-structured data**

**Semi-structured data** contains some categorization, but is not fully structured

- e.g., Wikipedia entries, Internet Movie Database (http://www.imdb.com)
  - Since users add much of the content, the way it is structured and categorized varies from user to user

Compare pages of two actors on IMDB (as of July 15, 2009):

- Bruce Campbell (I), but William H. Macy (no (I))
- Bruce Campbell’s page lists Mini Biography, but not on William H. Macy’s page

---

**Semi-structured example**

**IMDB**

Some snippets of trivia about Bruce Campbell:

- **Spouse**
  - Ida Gearon  (1991 - present)
  - Christine Devea  (13 March 1983 - 1989) (divorced) 2 children

- **Trade Mark**
  - His role of Ash from the Evil Dead films and video games. His large jaw bone, giving him the nickname “The Chin”

  - To search for dates, they come in different formats with different information: 1991, 13 March 1983
  - Likely also dates listed on IMDB in the format March 13, 1983 or March 13, 1983
  - No field for “nickname”, yet information is there

How would we do a search for actor nicknames?

---

**Motivating regular expressions**

If one wants to be able to describe more complex patterns of words and text, sometimes boolean expressions aren’t enough:

- In a large document I want to find addresses with a zip code starting with 911 (around Pasadena, CA); but clearly we would not want to report back all occurrences of emergency phone numbers in the document.
- I want to find all Indiana email addresses which occur in a long text.

Anything where you have to match a complex pattern so-called regular expressions are useful.

---

**Regular expressions: What they are**

- A regular expression is a compact description of a set of strings, i.e., a language (in formal language theory).
- They can be used to search for occurrences of these strings
- Regular expressions can only describe so-called regular languages.
- This means that some patterns cannot be specified using regular expressions, e.g., finding a string containing matching left and right parentheses.
- Note that just like any other formalism, regular expressions as such have no linguistic contents, but they can be used to refer to strings encoding a natural language text.

---

**Regular expressions: Tools that use them**

- A variety of unix tools (grep, sed, . . .), editors (emacs, jEdit, . . .), and programming languages (perl, python, Java, . . .) incorporate regular expressions.
- Implementations are very efficient so that large text files can be searched quickly; but still becoming efficient enough for web searching
  - One exception: Google Code Search
- The various tools and languages differ w.r.t. the exact syntax of the regular expressions they allow.
The syntax of regular expressions (I)

Regular expressions consist of
- strings of literal characters: c, A100, natural language, 30 years!
- disjunction:
  - ordinary disjunction: devoured|ate, family|ties
  - character classes: [Tt]he, be[c|o]me
  - ranges: [A-Z] (any capital letter)
- negation:
  \[^a\] (any symbol but a)
  [\[A\-Z\-9\] (not an uppercase letter or number)

The syntax of regular expressions (II)

- counters
  - optionality: ?
  - any number of occurrences: * (Kleene star)
  - at least one occurrence: +
- wildcard for any character: .
- beg.\.n for any character in between beg and n

The syntax of regular expressions (III)

- Escaped characters: to specify a character with a special meaning (\*, +, ?, \.), \[, \] it is preceded by a backslash (\)
e.g., a period is expressed as \.
- Operator precedence, from highest to lowest:
  - parentheses ()
  - counters * + ?
  - character sequences disjunction |

Grep

- grep is a powerful and efficient program for searching in text files using regular expressions.
- It is standard on Unix, Linux, and Mac OSX, and there also are various ports to Windows (e.g.,
  http://gnuwin32.sourceforge.net/packages/grep.htm,
- The version of grep that supports the full set of operators mentioned above is generally called egrep
  (for extended grep).

Grep: Examples for using regular expressions (I)

In the following, we assume a text file f.txt containing,
among others, the strings that we mention as matching.

- Strings of literal characters:
  egrep 'and' f.txt matches and, Ayn Rand, Candy and so on
- Character classes:
  egrep 'the year [0-9][0-9][0-9][0-9]' f.txt matches the year 1776, the year 1812, the year 2001, and so on
- Escaped characters:
  egrep 'why\?' f.txt matches why?, whereas egrep 'why?' f.txt matches why and wh

Grep: Examples for using regular expressions (II)

- disjunction (|): egrep 'couch|sofa' f.txt matches couch or sofa
- grouping with parentheses:
  egrep 'un(interest|excit)ing' f.txt matches uninteresting or unexciting.
- Any character (\):
  egrep 'o.e' f.txt matches ore, one, ole
Grep: Examples for using regular expressions (III)

- Kleene star (*):
  - `egrep 'a*rgh' f.txt` matches `argh`, `aargh`, `aaargh`
- One or more (+):
  - `egrep 'john+y' f.txt` matches `johny`, `johnny`, ..., but not `johy`
- Optionality (?):
  - `egrep 'joh?n' f.txt` matches `jon` and `john`

How corpora can be searched

- Both the BNC and the European Parliament corpus can be searched using on-line web-forms.
- Both of the web forms allow regular expressions for advanced searching.
- To provide efficient searching in large corpora, in these search engines regular expressions over characters are limited to single tokens (i.e. generally words).
- BNC:
  - web form: http://sara.natcorp.ox.ac.uk/lookup.html
  - regular expressions are enclosed in `{ }
- European Parliament Corpus:
  - web form: http://logos.uio.no/cgi-bin/opuscp.pl?corpus=EUROPARL;lang=en
  - in the simplest case, regular expressions are encosed in " "

Corpora

- A corpus is a collection of text.
- Corpora with the works of various writers, newspaper texts, etc. have been collected and electronically encoded.
- Corpora can be quite large
- The British National Corpus is a 100 million word collection representing a wide cross-section of current written and spoken British English.