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 the IUCAT part of OneSearch)
  - 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
   - Not every document has the same structure or format as every other document (e.g., internet movie reviews)

Structured data

Searching in a library catalogue

- To find articles, books, and other library holdings, a library generally provides a **database** containing information on its holdings.
- **IUCAT** is one of the **database frontends** providing access to the library databases at Indiana.
- This makes it possible to search for the occurrence of **literal strings** occurring in the author, title, keywords, call number, etc. associated with an item held by the library.

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**
- In addition to querying literal strings, the query language also supports the use of
  - **special characters** to abbreviate multiple options
  - **special operators** for combining two query strings (boolean operators) or modifying the meaning of a single string (unary operators)
Special characters and operators

- Use $ to truncate the characters at the end of a word
- ? can be used to replace a single character

The use of and, or, and the parentheses provides you with basic **boolean expression** operations

- Use and or (or xor) to specify multiple words in any field, any order.
  - art and therapy
  - art or therapy
  - art xor therapy (excludes “art therapy”)
- Use not to exclude words.
  - art not therapy
- Use parentheses to group words together when using more than one operator.
  - art therapy not ((music or dance) therapy)


Unstructured data

No explicit categorization of the documents to be retrieved

- Related to doing a keyword search in structured data
- Scale of the data is different: e.g., billions of webpages to search through
  - Types of search operators & ways to improve searches can differ from structured data

Some “unstructured” data contains hidden structure

- e.g., webpages with Chinese-English translations

By **unstructured**, we mean:

- the structure is not predetermined
- it is not uniformly applied or standardized
- queries cannot be formulated on that particular type of structure

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*
2. 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*)

Evaluating search results

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

**<top>**

**<num> Number: 363**

**<title> Hubble Telescope Achievements**

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

**<narr> 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.

**<top>**

Information need & evaluation

To evaluate search technology, TREC expresses information needs in natural language

- Evaluation: judge particular documents as to whether they meet information need in such descriptions

More specifically, TREC defines “right answers” as:

> If you were writing a report on the subject of the topic and would use the information contained in the document in the report, then the document is relevant

([http://trec.nist.gov/data/rejudge_eng.html](http://trec.nist.gov/data/rejudge_eng.html))
### 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.

### The nature of the web

- Web pages are generally less structured than a record in a library database (with title, author, subject, and other fields).
- One generally searches for words found anywhere in the document.
- It is, however, possible to include meta data in a web page.
- Meta data is additional, structured information that is not shown in the web page itself: e.g., the language a web page is in, its character encoding, author, keywords, etc.
- Example for a meta tag: `<META name=“keywords” lang=”en-us” content=“Vacation, Greece”>`

### 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 (I)

- +: 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
- Quotation Marks (phrases)
  - e.g., looking for sites on *What Cheer, Iowa* with
  ```
  "What Cheer"
  ```
- *: match anything (wildcard), e.g., “who’s that”

### Google: Operators (II)

- **intitle:** Find words used in a title
  - e.g., intitle:Hoosier finds only web pages which has this word in the title
- **inurl:** Find words used in the url
  - e.g., inurl:ling returns more linguistics webpages than ling does
- **link:** Find pages that link to a certain page
  - e.g., link:www.indiana.edu to show pages linking to the main Indiana web page
- **site:** Find pages that are part of a single domain
  - e.g., I want to find strange attractions involving fish. Knowing one site which has such stuff, one can try fish site:www.roadsideamerica.com.
Looking for counting how often a web result was clicked on by a user (click-through measurement)

▶ Various secret ingredients

Bonuses/penalties for known sites of high/low quality

As a result, unpopular or new pages require a more

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 Babbitt, it can be identified if many Sinclair Lewis sites link to my 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 Babbitt, it can be identified if many Sinclair Lewis sites link to my 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 Babbitt, 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

Improving searching (I)

How can I make my searches better?

▶ Be on the watch for ambiguity = one word has multiple meanings

▶ E.g., bed: flower bed, sleeping bed, truck bed

▶ Use synonyms and other related words

▶ E.g., plant: building, complex, works, power (distinguish from flora)

▶ Be aware of stop words = words that some search engines ignore because they are “uninformative,” such as the, of, and so on

Improving searches (II)

Exclude problematic words

▶ E.g., “jefferson airplane -starship” (if you don’t want info on the Starship years)

Be aware of parts of speech and what other guises they come in.

▶ E.g., plant: planting, planter, planted (distinguish from power plant)

Continually narrow your focus (using the feedback)

▶ E.g., Want to find information on the game Hearts

Hearts is too vague, but looking over results indicates cards or tricks could help

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:

**Name**

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

- 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 not efficient enough for web searching → no web search engine offers them (yet).
- 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|ies
  - character classes: [Tt]he, bec[a]me
  - ranges: [A−Z] (any capital letter)
- negation:
  - `[^a]` (any symbol but a)
  - `[^A-Z0−9]` (not an uppercase letter or number)

The syntax of regular expressions (II)

- counters
  - optionality: ?
  - `colou?r`
- any number of occurrences: `*` (Kleene star)
  - `[0−9]*` years
- at least one occurrence: `+`
  - `[6−9]+` dollars
- 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 OS X, 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`, `aarrgh`
  - `egrep 'shala*la' f.txt` matches `shala`, `shalala`, or if you're Van Morrison `shalalalalalalala`
- **One or more (`+`):**
  - `egrep 'john+y' f.txt` matches `johny`, `johnny`, . . . , but not `johy`
- **Optionality (`?`):**
  - `egrep 'joh?n' f.txt` matches `jon` and `john`

### 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.
- **Another example is the European Parliament Proceedings Parallel Corpus 1996–2003.**
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](http://sara.natcorp.ox.ac.uk/lookup.html)
  - regular expressions are enclosed in `{ }`
- European Parliament Corpus:
  - web form: [http://logos.uio.no/cgi-bin/opus/opuscqp.pl?corpus=EUROPARL.lang=en](http://logos.uio.no/cgi-bin/opus/opuscqp.pl?corpus=EUROPARL.lang=en)
  - in the simplest case, regular expressions are enclosed in `" "`