AIW: Indicizzazione e Ricerca nei Testi

http://www.di.unipi.it/~grossi/AIW

Roberto Grossi

Dipartimento di Informatica
grossi@di.unipi.it

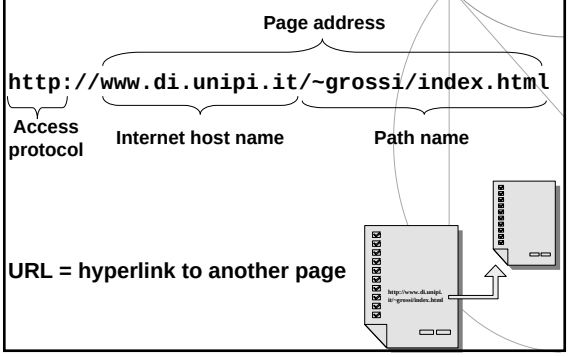
Outline of the Course (in English)

- **Part I: How to search the Web**
  - **Representation of the Web**
  - **Search engines**
- **Part II: How to index the Web (and textual data)**
  - **Inverted lists**
  - **Suffix arrays**
  - **Suffix trees**

Part I

WEB SEARCHING

Each page: unique address (URL)



Web as a graph

- **Web page u = node u in the graph**
- **Hyperlink from page u to page v: directed edge (u,v)**

- **Alternatively: some relation between u and v (e.g., after loading u, users typically load v)**

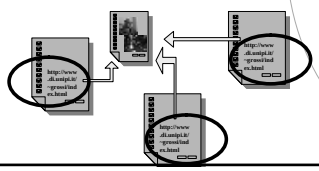
Succinct representation of the Web

- **Represent and encode hyperlinks (URLs ! short integer numbers)**
- **Use few memory storage**
- **Fast preprocessing**
- **Low query time:**
  - **Find URLs arriving in u**
  - **Find URLs departing from u**
- **Typical assumption: *Zipf's law***
  - **$1 / j^{2.1}$ pages have *in*degree j**
  - **$1 / j^{2.3}$ pages have *out*degree j**

Searching the Web

- **Web searching is text searching**
  - **Textual data for describing audio and video**
  - **How to get textual data:**

Put together the (con)text around URLs pointing to a page with audio or video



Which kind of queries?

- **Boolean: AND, OR, NOT:** `" web and search "`
- **Proximity searching:** `" web near search "`
- **Phrase searching:** `" web search "`
- **Links referring to a page: input is an URL**
- **Tags searching: e.g., " intitle: web search "**

www.google.com/help/operators.html

WORDS (or partial words)

Difference from classical IR

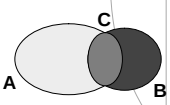
- **Retrieve *high quality* pages:**
  - **Static documents: text, audio**
  - **Dynamic documents: generated by queries on data bases via the Web (e.g., Amazon books)**
- **Huge collections of data (Google is one of the largest indexes, but it collects only 10% of the Web pages)**
- **No persistency (data is volatile)**
- **Heterogeneity (type of documents, languages, no typographical control, etc.)**
- **Duplication (nearly same documents, different URLs)**
- **Removing non relevant info (e.g., banners)**

Quality of search results

- **Human-driven judgment on relevance**
- **Precision = % of returned pages that are relevant**

  C / A
- **Recall = % of relevant pages that are returned**

  C / B



- **Web: need to rank the quality of pages**

Web and IR tools to help searching

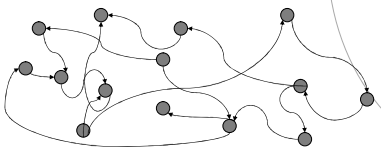
- **Hierarchical directories (e.g., Yahoo)**
- **Topic-specialized engines**
- **Search by example (from a set of URLs)**
- **Meta-information (e.g.,compare search engines)**
- **Clustering (from IR)**
- **Categorization**
- **Summarization**
- **Latent semanting indexing**

Search engine structure

1. **Spider/crawler: collect the documents**
2. **Indexer: Process and store data for quick access**
3. **Search interface: answer queries**

1. Spider or crawler

- Traverse the graph and collects documents
  - Add pages to the index
  - Remove duplication (hash q-grams)
  - Use some suitable traversal order of the Web graph
  - Load balancing
  - Filtering heterogeneous sources



2. Indexer

- The heart of search engines: full scan is impractical…
- …**massive** collections of texts
- Low space occupancy
- Fast preprocessing
- Fast query answering

… more details later in Part II

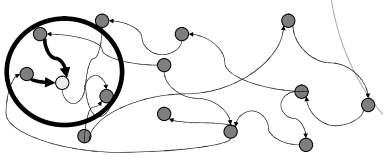
3. Search Interface

Too many reported documents!
Ranking using the Web graph

- Human annotation: costly and impractical
- *Local ranking:*
  - based on the only content of each document
  - good for filtering pages with offending content
  - not valid for raking the quality of pages
- *Global ranking*: exploit the Web graph
  - PageRank (and Salsa): assign rank before any query
  - Hits: assign rank after each query

Search Interface: Ranking via Web

- URLs connect related pages
- An URL between pages is recommendation
- Ideas from ranking scientific papers by citations



Search Interface: PageRank

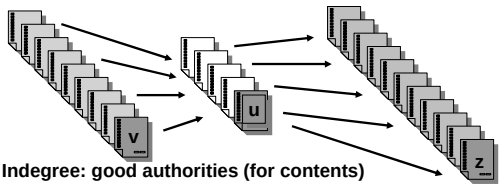
- Quality of u is related (after normalization) to
  - Indegree of u
  - Recursive quality of pages pointing to u
- Based on a random walk on the Web graph with n nodes
- Random surfer on u goes to:
  - Random page with probability p
  - One of the "next" pages v with probability 1-p

$$PR(u) = p / n + (1-p) \sum_{(v,u)} PR(v) / outdegree(v)$$

- The value of PR converges to the stationary distribution (Markov chain, see also SALSA)

Search Interface: HITS

- Graph on a subset of reported pages (with a IR criterion)



- Indegree: good authorities (for contents)
  Outdegree: good hubs (for links)
- Recursive interaction between
  - $HUB(u) = \sum_{(u,z)} AUTH(z)$
  - $AUTH(u) = \sum_{(v,u)} HUB(v)$ (and normalize)
- Solve with iterative methods; extensions of HITS

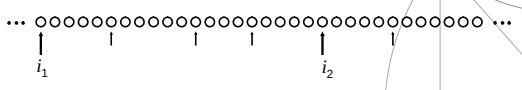
Part II

TEXT INDEXING

What is an index?

- Persistent and space-efficient data structure.
- Quickly answering lots of input queries.
- Accessing a small portion of the (huge) data collection.
- Basic building block of any IR system.

Word-level indexing



1. Split the text into words
2. Collect all distinct words in a dictionary
3. For each word *w*, store the (inverted or position) list of its locations in the text

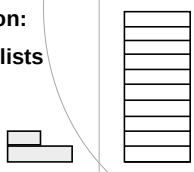
$i_1, i_2, \cdots$

Inverted lists (or files)

Simple implementation: one pointer per location

Avg. word size ¸ pointer size) index space ¼ text size

- ✓ Much better implementation:
  compress the inverted lists (e.g., γ and δ codes)
- ✓ Index space reduces to
  10%-15% of the text



1 ½ floor + 10 floors

Full-Text Indexing

- Word-level is special case of full-text indexing:
  Not only word beginnings, but each text position is a potential occurrence

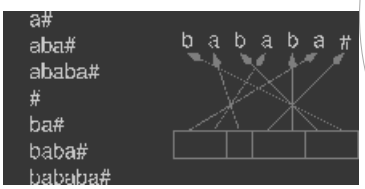
- Alphabet Σ, text *T* of size *n* (i.e., *n* d log $|\Sigma|$ e bits) :

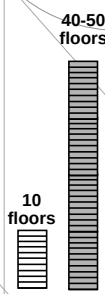
Naive approach: O(n^2) *space blow-up*

Better approach: O(*n*) *space (i.e.,* O(*n* log *n*) *bits) only the n suffixes are stored*

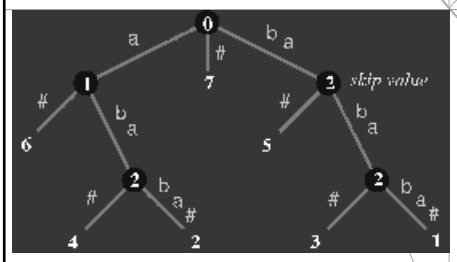
Suffix array

- Sorted list of suffixes (a < # < b)
- Better space occupancy: 4*n* bytes
- Additional *n* bytes for fast searching





Suffix tree / Patricia

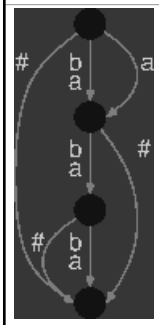


160 floors

10 floors

- Compact trie / Patricia storing the suffixes of bababa#
- Space is roughly 16*n* bytes

Automaton and DAWG



- Collapse isomorphic nodes to reduce their number
- Average size is about 1.3 nodes per character
- Nodes are of larger size
- Number of bytes does not reduce much