VK Multimedia Information Systems

Mathias Lux, mlux@itec.uni-klu.ac.at

Dienstags, 16.oo Uhr c.t., E.2.69

Content

- **Introduction**
- Common Architecture
- Ranking
  - PageRank
  - HITS
- Application: Nutch

Retrieval in the WWW

- General Retrieval is based on content
  – Represented e.g. by terms, keywords ...
- What is different with the WWW?
  – Structured text (markup)
  – Hypermedia (links)
  – Heterogeneous formats (gif, pdf, flv, ...)
  – Distributed content (access over network)

Web based Retrieval: Challenges

Working with an enormous amount of data

- 10 billion pages a 500kB estimated in 01-2004
  - 2 pages / person on the globe
- Surface web 20 times larger than the LoC print collection
  - estimated in 2003
- Furthermore there is a **Deep Web**
  - 550 billion pages estimated in 2004

Web based Retrieval: Challenges

- Example for the amount of web pages:
  - Searching for 'Star Trek' yielded on Google ~ 84 millions of results
  - Users investigate up to 20 result list entries.
- What web page is the **most interesting**?
  - Cp. Concept of relevance (IR)
- How to **index** this amount of pages?
  - Eg. In an inverted list

Web based Retrieval: Challenges

The Web is highly dynamic

- Study by Cho & Garcia-Molina (2002):
  - 40% of the web pages changed their dataset within a week
  - 23% of the .com pages changed on daily basis
- Study by Fetterly et al. (2003):
  - 35 % of the pages changed while the investigations
  - Larger web pages change more often

Web based Retrieval: Challenges

The Web is self-organized

- No central authority / main index
  - For the WWW
- Everyone can add (or edit) pages
  - Cp. Personal homepages, blogs, wikis, …
- Pages disappear on regular basis
  - US study claimed that in 2 investigated tech. journals 50% of the cited links were inaccessible after four years.
- Lots of errors and falsehood, no quality control

Web based Retrieval: Challenges

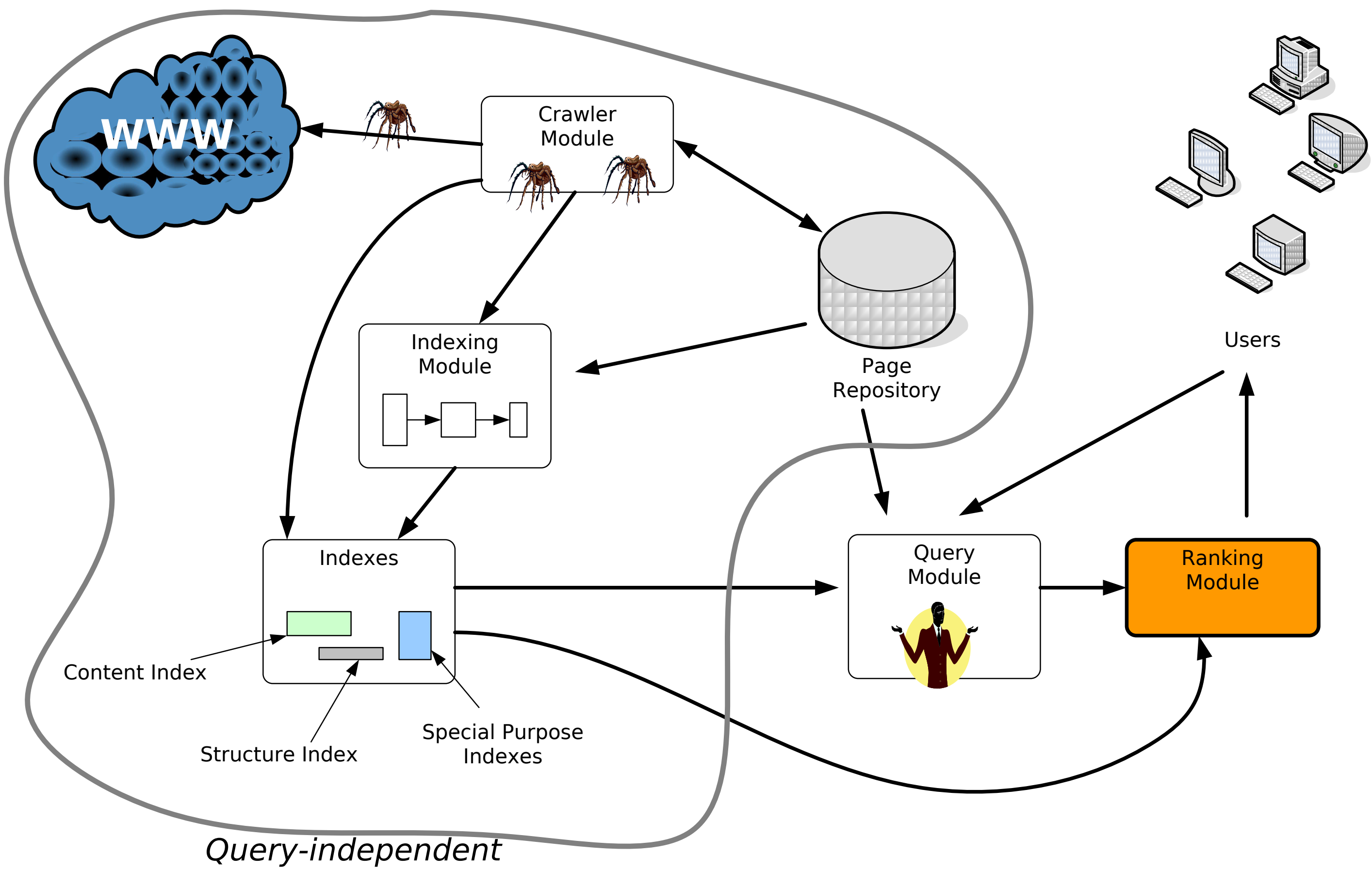
The Web is hyperlinked

- Based on HTML **Markup** tags and **URIs**
- Pages are interconnected
  - Unidirectional links
  - in-link, out-link and self-link
- Network structures emerge from the links
  - Interdepence / graph analysis possible

Content

- Introduction
- **Common Architecture**
- Ranking
  - PageRank
  - HITS
- Application: Nutch

Common Architecture



Crawler

Crawlers, robots & spiders harvest sites

- Starting with a **root set** of URLs
- **Following links**, that are found on the pages
- Applying **filters** to the links
  - e.g. only .at domains -> Austrian web pages
  - e.g. based on link title & position (cp. focused crawling)

Crawlers: Index Update

- Update index: What & When?
- A page content might have changed since last visit
  - last modified dates are eventually inaccurate
- Different strategies are possible:
  - Refresh only portions ...
  - Prefer most popular sites ...

Crawling: Ethical Questions ...

- Consumption of bandwidth?
- Impact on statistics: hit counts, visits, ...
- Server load of crawled web server?
- Let loose several spiders at once
  – Decrease of crawling time
  – Increase load

Crawling: Robots.txt

- Robots.txt is option for webmasters to
  - restrict crawler access
  - point crawlers to interesting URLs
  - identify crawlers (with hit on the robots.txt)
  - see http://www.robotstxt.org/wc/robots.html

- Example

```
User-agent: *
Disallow: /wp-admin/
Disallow: /netadmin/
```

Crawler: Google sitemaps

- XML schema to identify interesting portions & updates of a web page
- Integration into CMS
- Example:

```
<url>
   <loc>http://www.semanticmetadata.net/</loc>
   <lastmod>2007-02-06T11:26:06+00:00</lastmod>
   <changefreq>daily</changefreq>
   <priority>1</priority>
</url>
```

Google Sitemaps

<urlset>

– Collection of URIs, main tag

<url>

– Parent tag of a web page definition

<loc>

– URI of the web page,

– < 2048 chars

<lastmod>

– last modification in W3C date format

– time portion optional

Google Sitemaps

<changefreq>

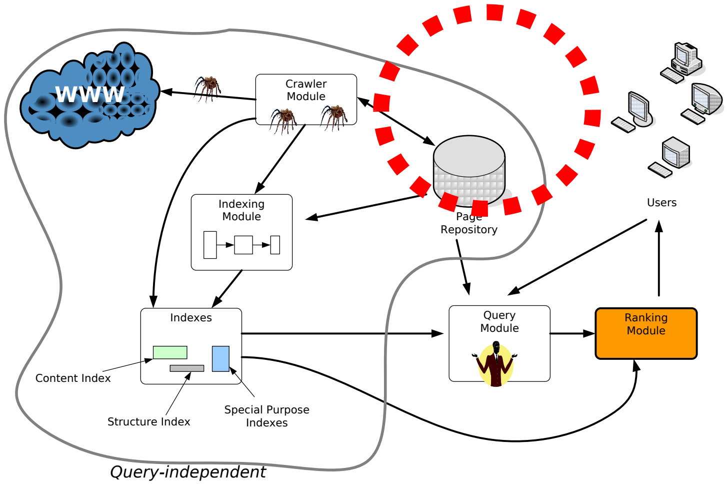
- How frequently the page is likely to change
- always, hourly, daily, weekly, monthly, yearly, never

<priority>

- out of [0,1]
- default is 0.5

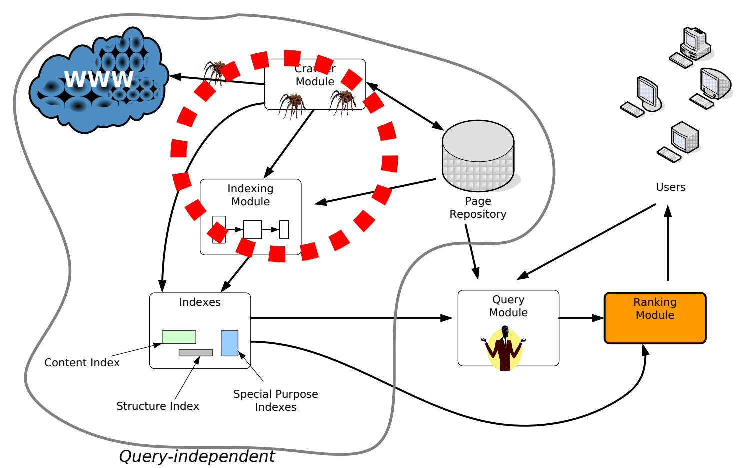
Page Repository

- Twofold benefits
  - Temporary storage for indexing process
  - Cache for pages
    - summarization of search results
    - snapshot
- After indexing
  - Information is compressed
  - e.g. Stripping tags



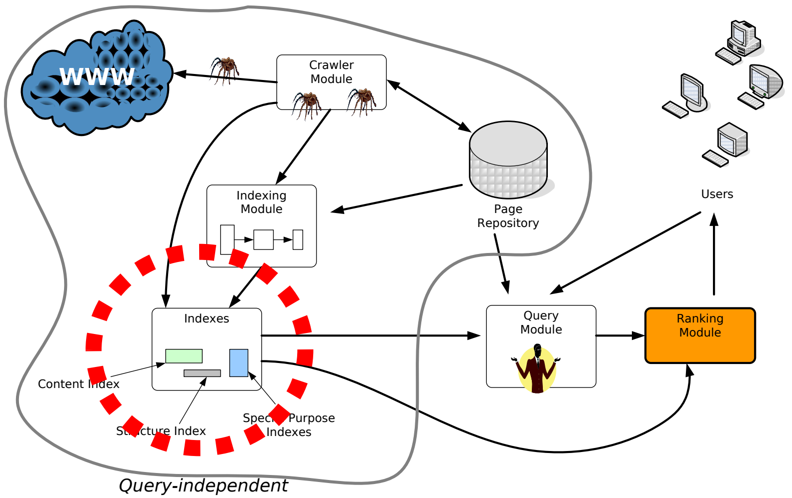
Indexing Module

- Takes each new uncompressed page
- Extracts vital descriptors
  - terms, positions, links
- Creates compressed version of page
- Stores
  - Page in cache
  - Descriptors in index



Indexes

- Content Index
- Structure Index
- Special Purpose Index
  - Document Formats (PDF, Doc, …)
  - Media (Images, Video, …)
  - Usage (Hits, bookmarks, …)



Content Index

- Inverted Index
  - term x -> <d11>, <d28>, <d31>, ...
  - term y -> <d10>, <d35>, <d36>, ...
- Index is a
  - quick lookup table
  - smaller than documents

Content Index

Characteristics of Web Retrieval

- Huge amount of different terms
  - Multiple languages
  - No stemming
- Huge amount of pages / term
  - e.g. for broad terms (*weather*, *sports*)
- Needs to be compressed & distributed

Structure Index

- Hyperlink Information
  - In-links, out-links & self-links
- Stored for …
  - Later analysis
  - Later queries (who links to whom)

Query Module

- Creates query
  - From user input (natural language)
- Distributes query to indexes
  - Multiple indexes on multiple machines
- Create result set
  - Set of relevant pages

Ranking Module

- Orders set of relevant pages
  - Input from query module
- Employs **ranking algorithm**
  - Based on several aspects (terms, links, etc.)
  - Overall score is combination of
    - Content score (TF*IDF)
    - Popularity score (PageRank, HITS, etc.)

Content

- Introduction
- Common Architecture
- **Ranking**
  - PageRank
  - HITS
- Application: Nutch

Ranking by Popularity

- Problem with amount of data
  – Queries on popular terms yield many results
- Idea for selecting the most relevant …
  – Combine content with **popularity** of page
  – More popular pages are “authorities”
- How to define popularity?
  – Only hypertext documents are given ...

Popularity Ranking

- 2 Algorithms developed independently
  - PageRank, Brin & Page
  - Hypertext Induced Topic Search (HITS), Kleinberg
- Basic idea of popularity
  - Someone likes a page
  - Gives a recommendation (on another page)
  - Using a hyperlink

Popularity Ranking: Basic Idea

- There are different types of people:
  - Regarding their idea of recommendation
    - People giving a lot of recommendations (links)
    - People giving few recommendations (links)
  - Regarding their state of recommendation
    - Recommended by a lot of people
    - Recommended by few people
- Combinations are possible:
  - Having no recommendation, but recommending a lot, ...

Popularity Ranking: Basic Idea

Think of ....

- people as pages
- recommendations as links

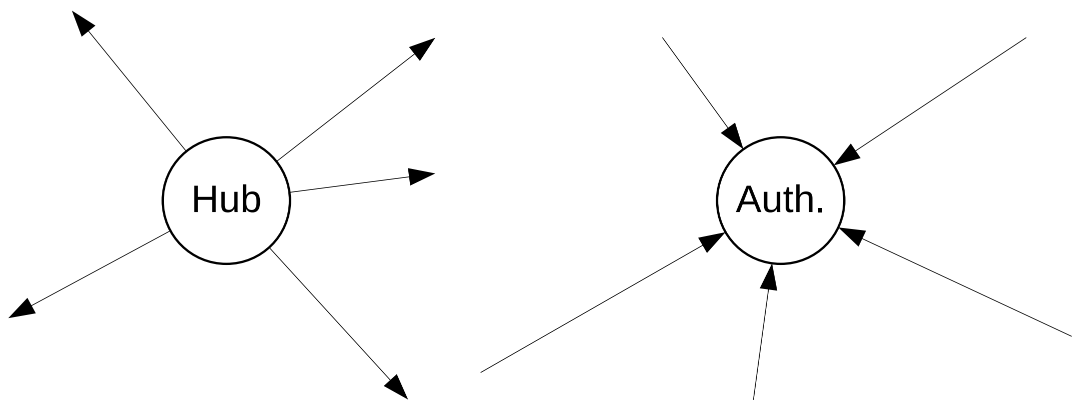
Therefore:

PageRank (Google)

"Pages are popular, if popular pages link them"

Popularity Ranking: Basic Idea

- Additional assumptions:
  - **Hubs** are pages that refer a lot
  - **Authorities** are pages, which are referred a lot



HITS

PageRank: Original Summation Formula

- Original summation formula
  - PageRank of page P_i is given by the summation of all pages that link to P_i given by Set B_{Pi}

$$r(P_i) = \sum_{P_j \in B_{P_i}} \frac{r(P_j)}{|P_j|},$$

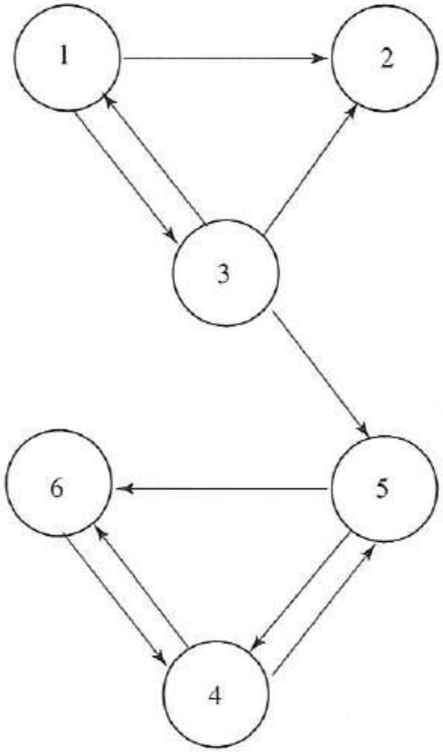
- Iterative formula, starting with rank 1/n for all n pages:

$$r_{k+1}(P_i) = \sum_{P_j \in B_{P_i}} \frac{r_k(P_j)}{|P_j|}$$

PageRank: Original Summation Formula

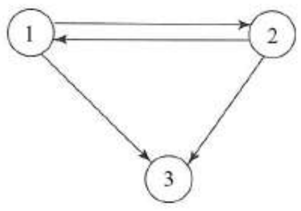
$$r_{k+1}(P_i) = \sum_{P_j \in B_{P_i}} \frac{r_k(P_j)}{|P_j|}$$

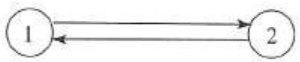
Iteration 0	Iteration 1	Iteration 2	Rank at Iter. 2
$r_0(P_1) = 1/6$	$r_1(P_1) = 1/18$	$r_2(P_1) = 1/36$	5
$r_0(P_2) = 1/6$	$r_1(P_2) = 5/36$	$r_2(P_2) = 1/18$	4
$r_0(P_3) = 1/6$	$r_1(P_3) = 1/12$	$r_2(P_3) = 1/36$	5
$r_0(P_4) = 1/6$	$r_1(P_4) = 1/4$	$r_2(P_4) = 17/72$	1
$r_0(P_5) = 1/6$	$r_1(P_5) = 5/36$	$r_2(P_5) = 11/72$	3
$r_0(P_6) = 1/6$	$r_1(P_6) = 1/6$	$r_2(P_6) = 14/72$	2



Initial Problems

- Rank sinks & cycles:
  - Some pages get all of the score, other pages none
  - Cycles just flip the rank
- How many iterations?
  - Will the process converge?
  - Will it converge to one single vector?

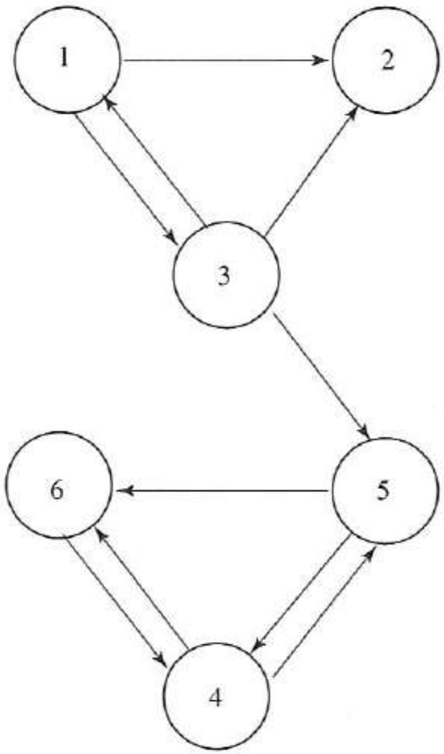




Approach of Brin & Page

- Notion of the random surfer
  - Someone navigates through the web using hyperlinks.
  - If there are 6 links, there is a probability of 1/6 that s/he takes a specific link
  - On dangling nodes (without out links) s/he can jump everywhere with equal chance
  - Furthermore s/he can leave the link path with a given probability every time

Approach of Brin & Page: Dangling nodes



$$
\mathbf{H} = \begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \end{array}
\begin{array}{c} \begin{array}{cccccc} P_1 & P_2 & P_3 & P_4 & P_5 & P_6 \end{array} \\
\begin{pmatrix}
0 & 1/2 & 1/2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1/3 & 1/3 & 0 & 0 & 1/3 & 0 \\
0 & 0 & 0 & 0 & 1/2 & 1/2 \\
0 & 0 & 0 & 1/2 & 0 & 1/2 \\
0 & 0 & 0 & 1 & 0 & 0
\end{pmatrix} \end{array}
$$

↓ Dangling nodes

$$
\mathbf{S} = \begin{pmatrix}
0 & 1/2 & 1/2 & 0 & 0 & 0 \\
1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\
1/3 & 1/3 & 0 & 0 & 1/3 & 0 \\
0 & 0 & 0 & 0 & 1/2 & 1/2 \\
0 & 0 & 0 & 1/2 & 0 & 1/2 \\
0 & 0 & 0 & 1 & 0 & 0
\end{pmatrix}
$$

Approach of Brin & Page:

Teleportation of the *Random Surfer*

- Introduction of the Google Matrix:

$$\mathbf{G} = \alpha\mathbf{S} + (1-\alpha)1/n\,\mathbf{e}\mathbf{e}^T$$

Umformung: $G = \alpha \cdot H + (\alpha \cdot a + (1-\alpha)e)\frac{1}{n}e^T$ a … dangling nodes vector

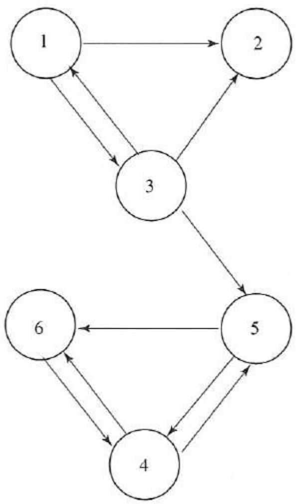
$$\mathbf{G} = .9\mathbf{H} + (.9\begin{pmatrix}0\\1\\0\\0\\0\\0\end{pmatrix} + .1\begin{pmatrix}1\\1\\1\\1\\1\\1\end{pmatrix})\,1/6\,(1\quad 1\quad 1\quad 1\quad 1\quad 1)$$

$$= \begin{pmatrix} 1/60 & 7/15 & 7/15 & 1/60 & 1/60 & 1/60 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 19/60 & 19/60 & 1/60 & 1/60 & 19/60 & 1/60 \\ 1/60 & 1/60 & 1/60 & 1/60 & 7/15 & 7/15 \\ 1/60 & 1/60 & 1/60 & 7/15 & 1/60 & 7/15 \\ 1/60 & 1/60 & 1/60 & 11/12 & 1/60 & 1/60 \end{pmatrix}.$$

Approach of Brin & Page: Result of the adaptations

- Iterative Formula $\pi^{(k+1)T} = \pi^{(k)T}\mathbf{G},$
  - Converges to a single PageRank vector
- In our example:

$$\pi^T = \begin{pmatrix} \overset{1}{.03721} & \overset{2}{.05396} & \overset{3}{.04151} & \overset{4}{.3751} & \overset{5}{.206} & \overset{6}{.2862} \end{pmatrix}$$



taken from “Google’s PageRank & Beyond”, Langville & Meyer

Features of PageRank

- Mathematical model
  - Created later on, based on Markov chains
- Can be handled in a distributed way
  - "Worlds biggest matrix multiplication"

HITS

- Every page *i* has a authority score x_i and a hub score y_i
- Successive refinement of scores:

$$x_i^{(k)} = \sum_{j:e_{ji} \in E} y_j^{(k-1)} \text{ and } y_i^{(k)} = \sum_{j:e_{ji} \in E} x_j^{(k)} \text{ for } k = 1, 2, 3, \ldots$$

Search Engine "Optimization"

- Business for "optimizing" rank in search listings (SEO)
- There are two ways:
  - Ethical: Good content and communication leads to extensive linking and a high content score as well as popularity
  - Unethical: Try to get a lot of links to the site of the customer or lay a *Google Bomb*.

Content

- Introduction
- Common Architecture
- Ranking
  - PageRank
  - HITS
- **Application: Nutch**

Nutch

- Open Source Web Search Engine
- Currently in version 1.2
- Includes
  - Crawler (simple to expert)
  - Document parsers (doc, pdf, ...)
  - Lucene & OPIC Ranking
  - Distribution framework (Hadoop)
  - ...

What can Nutch do?

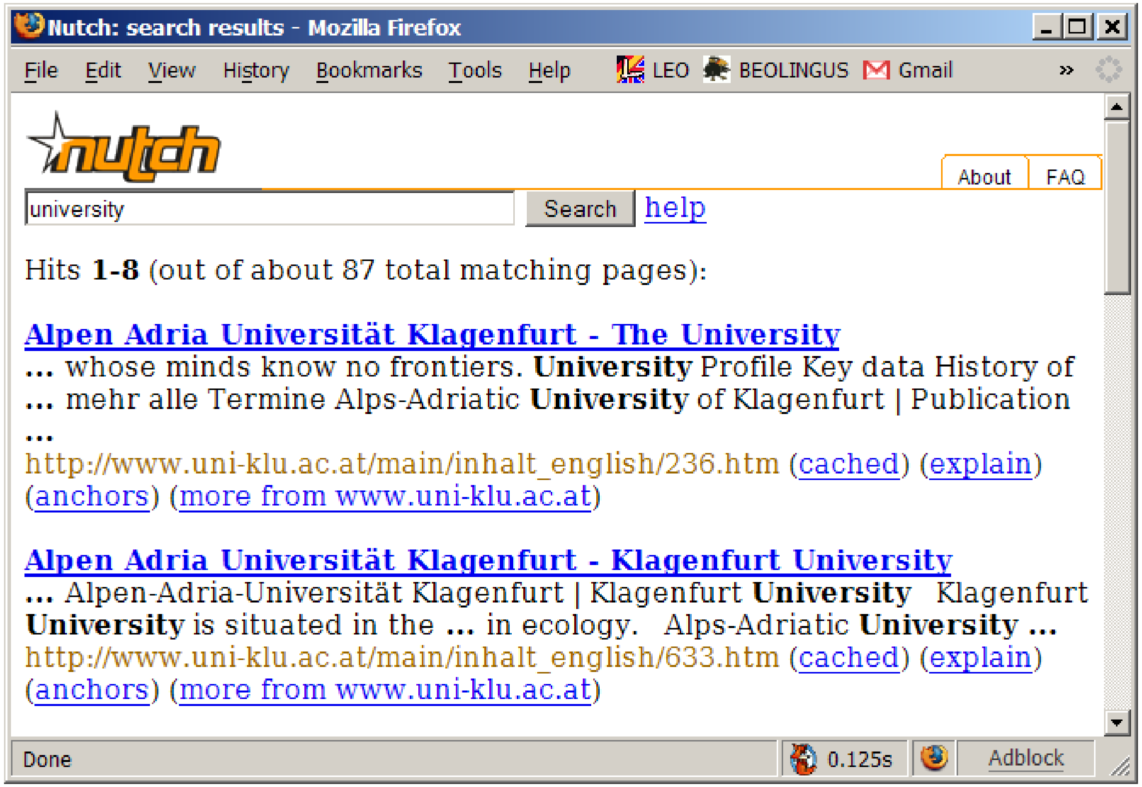
- Crawling
  - Configurable threads
  - Distributed approach
  - Control on various levels (inject, fetch, index)
- Searching
  - Web based as well as Java Bean
  - Distributed indexes
  - Ranking similar to PageRank

Simple Use Example

- Download and unzip Nutch
  - ~ 83 MB
- Intranet Crawl:
  - Set up root URLs
    - e.g. `http://www.uni-klu.ac.at/main/index.htm`
  - Set up URL filters
    - e.g. `+^http://([a-z0-9]*\.)*uni-klu.ac.at/`
  - Set up configuration
    - Add user agent string (mandatory)
  - Run crawler:
    - `bin/nutch crawl urls -dir crawl.test -depth 3 >& crawl.log`

Simple Use Example

- Search using Apache Tomcat



Costs for Web Crawling

- How much does it cost to run a search engine?
  - Monthly amount of pages to crawl: 4 billion
  - 4.000.000.000 pages @ 200K = 80.000 GB per month.
- One connection:
  - 100mbs connection
    / 8 megabits per MB
    * 60 seconds in a minute
    * 60 minutes in an hour
    * 24 hours in a day
    * 30 days in a month=32.400 GB / month

Costs for Web Crawling

- Therefore at least 3 100 MBit connections are needed
  - Running at full capacity 24/7
  - Only with a simple calculation (w/o overhead)
- Also at least 3 servers are needed
- And a lot of storage
  - ~ 80.000 GB with caching

taken from http://www.mail-archive.com/nutch-user@lucene.apache.org/msg05577.html