Πανεπιστήμιο Κρήτης, Τμήμα Επιστήμης Υπολογιστών
Άνοιξη 2007

ΗΥ463 - Συστήματα Ανάκτησης Πληροφοριών
Information Retrieval (IR) Systems

Web Searching

I: History and Basic Notions, Crawling
II: Link Analysis Techniques
III: Web Spam Page Identification

Γιάννης Τζίτζικας
Διάλεξη : 8
Ημερομηνία : 20/ 4 / 2007

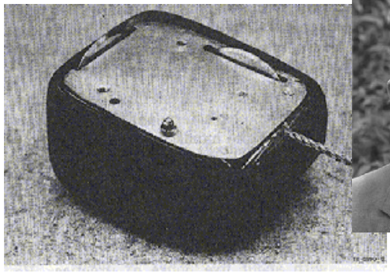
Διάρθρωση Διάλεξης I

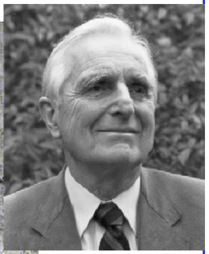
- Ιστορική Αναδρομή
- Ανάκτηση Πληροφοριών από τον Ιστό: Προκλήσεις και Απαιτήσεις
- Ο νόμος του Zipf και ο Ιστός
- Η δομή του γράφου του Ιστού
- Κατάλογοι (Yahoo/ODP) έναντι Μηχανών Αναζήτησης
  - Automatic Document Classification
  - Automatic Document Hierarchies
- Έρπειν (Crawling/Spidering)
  - Διάσχιση (spidering/crawling)
  - Depth/Breadth and Technical Details
  - Directed (Topic/Link/…) Spidering
  - Multi-Threaded Spidering
- Αποθήκευση και Ευρετηρίαση Σελίδων

Προϊστορία

- 1965:
  - Ο Ted Nelson συνέλαβε και ανέπτυξε την ιδέα του **υπερκειμένου** (hypertext)

- Τέλη δεκαετίας 60:
  - Ο Doug Engelbart επινόησε το **ποντίκι** και πρώτος <u>υλοποίησε</u> το υπερκείμενο.





- 1970's:
  - Ανάπτυξη του ARPANET

Άρα οι ιδέες και η βασική τεχνολογία υπήρχε από το 70.
Έπρεπε να έρθει η εποχή των PC και της ευρείας αλληλοσύνδεσης για να εμπνευστούμε και να φτιάξουμε τον Παγκόσμιο Ιστό.

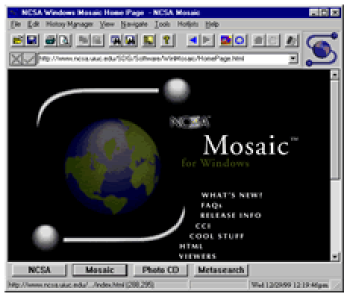
Το **Παγκόσμιος Ιστος** (the World Wide Web)

1990: Αναπτύχθηκε από τον **Tim Berners-Lee** (στο CERN) για την οργάνωση των ερευνητικών εγγράφων που ήταν διαθέσιμα στο Διαδίκτυο

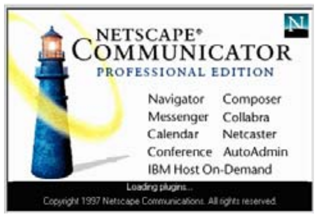
- Ανέπτυξε το πρωτόκολο HTTP, όρισε τα URLs και τη γλώσσα HTML, και υλοποίησε τον πρώτο "web server."
- Συνδυασμός 2 ιδεών:
- Έγγράφων διαθέσιμων με FTP
- Διασύνδεση εγγράφων (υπερκείμενο)

Η Ιστορία των **Πλοηγητών** (Web Browsers)

- Early browsers were developed in **1992**
  - Erwise, ViolaWWW
- In **1993**, Marc Andreessen and Eric Bina at UIUC NCSA developed the **Mosaic browser** and distributed it widely.



- **1994:** Andreessen joined with James Clark (Stanford Prof. and Silicon Graphics founder) to form Mosaic Communications Inc. (which became **Netscape** to avoid conflict with UIUC).

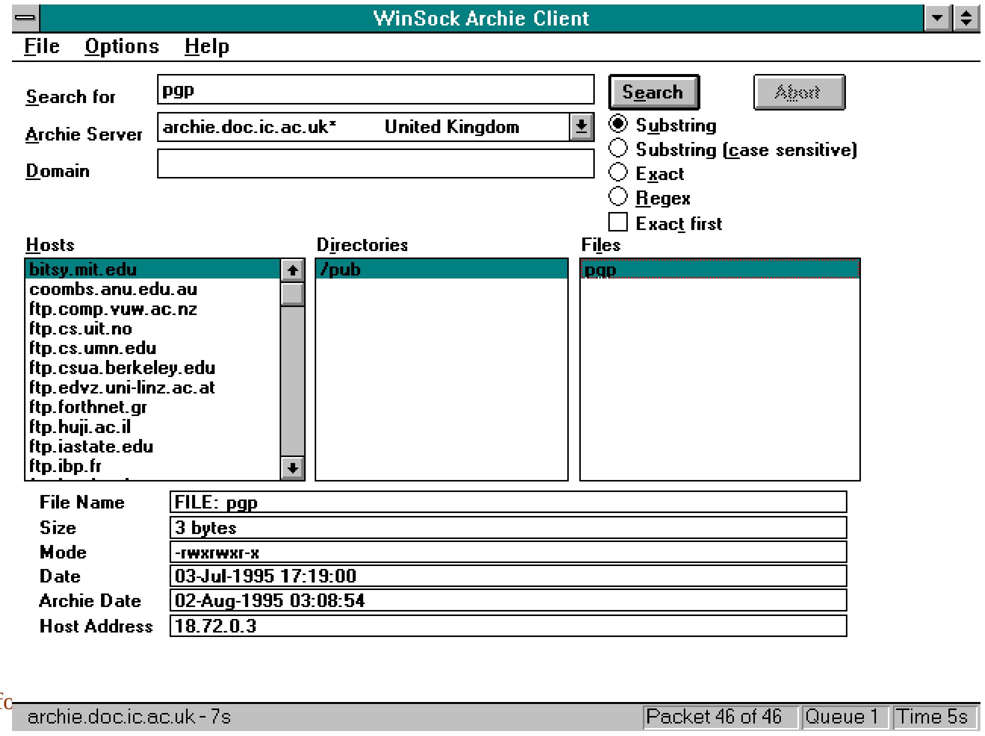


- **1995:** Microsoft licensed the original Mosaic from UIUC and used it to build **Internet Explorer**.

Η Ιστορία των **Μηχανών Αναζήτησης** Search Engine Early History: **FTP, Archie**

- By late 1980's many files were available by **anonymous FTP**.
- In **1990**, Alan Emtage of McGill Univ. developed **Archie** (short for "archives")
  - Assembled lists of files available on many FTP servers.
  - Allowed regex search of these filenames.

Archie Client



Search for: pgp

Archie Server: archie.doc.ic.ac.uk* United Kingdom

Domain:

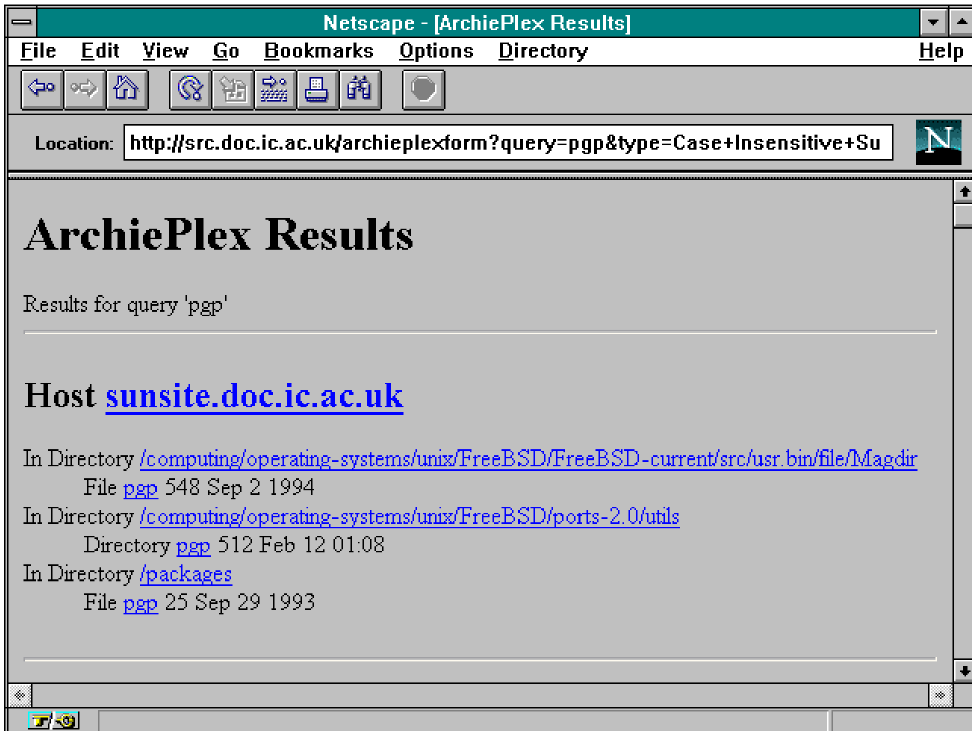
[✓ Substring] Substring (case sensitive) Exact Regex
Exact first

Hosts	Directories	Files
bitsy.mit.edu	/pub	pgp
coombs.anu.edu.au		
ftp.comp.vuw.ac.nz		
ftp.cs.uit.no		
ftp.cs.umn.edu		
ftp.csua.berkeley.edu		
ftp.edvz.uni-linz.ac.at		
ftp.forthnet.gr		
ftp.huji.ac.il		
ftp.iastate.edu		
ftp.ibp.fr		

File Name	FILE: pgp
Size	3 bytes
Mode	-rwxrwxr-x
Date	03-Jul-1995 17:19:00
Archie Date	02-Aug-1995 03:08:54
Host Address	18.72.0.3

archie.doc.ic.ac.uk - 7s Packet 46 of 46 Queue 1 Time 5s

Archie via WWW gateway


Netscape - [ArchiePlex Results]
File Edit View Go Bookmarks Options Directory Help

Location: http://src.doc.ic.ac.uk/archieplexform?query=pgp&type=Case+Insensitive+Su

ArchiePlex Results

Results for query 'pgp'

Host sunsite.doc.ic.ac.uk

In Directory /computing/operating-systems/unix/FreeBSD/FreeBSD-current/src/usr.bin/file/Magdir
File pgp 548 Sep 2 1994
In Directory /computing/operating-systems/unix/FreeBSD/ports-2.0/utils
Directory pgp 512 Feb 12 01:08
In Directory /packages
File pgp 25 Sep 29 1993

Η Ιστορία των **Μηχανών Αναζήτησης** Search Engine Early History: **Gopher**

- In **1993**, Veronica and Jughead were developed to search names of text files available through **Gopher** servers.

- *Gopher* is a menu-driven Internet browser
- Presents users with a hierarchy of items and directories **much like a file system.**
  - The Gopher interface resembles a file system since a file system is a good model for organizing documents and services;
  - the user sees what amounts to one big networked information system containing primarily **document items**, **directory items**, and **search items** (the latter allowing searches for documents across subsets of the information base).
- Servers return either directory lists or documents.

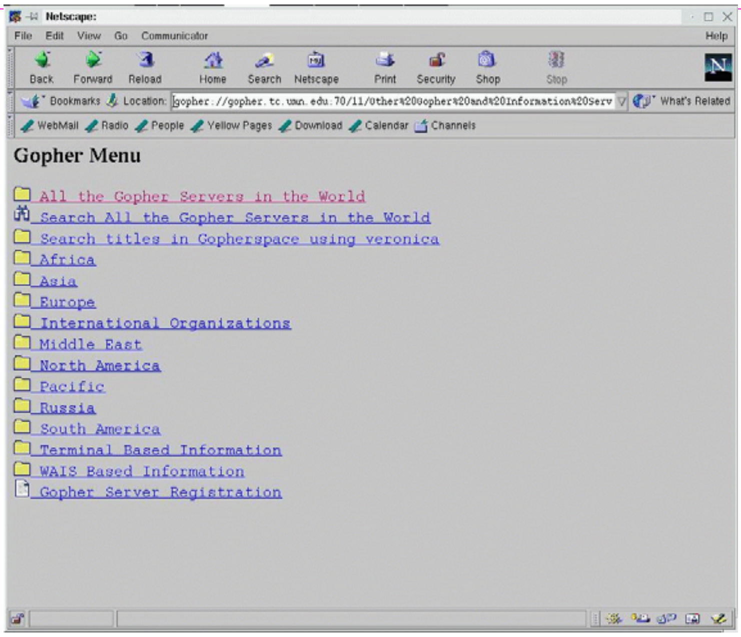
Η Ιστορία των **Μηχανών Αναζήτησης** Search Engine Early History: **Gopher** (II)

- Each item in a directory is identified by
  - a **type** (the kind of object the item is),
  - **user-visible name** (used to browse and select from listings),
  - an opaque **selector string** (typically containing a pathname used by the destination host to locate the desired object),
  - a **host name** (which host to contact to obtain this item), and
  - an **IP port number** (the port at which the server process listens for connections).
- The user only sees the user-visible name. The client software can locate and retrieve any item by the trio of selector, hostname, and port.
- To use a search item, the client submits a query to a special kind of Gopher server: a **search server.** In this case, the client sends the selector string (if any) and the list of words to be matched. The response yields "virtual directory listings" that contain items matching the search criteria.

Gopher (III)

- **Veronica** has three phases that are performed periodically:
  - Harvesting. **Collects** all the menu entries and file names for all items in Gopherspace and **extracts** all the keywords.
    Indexing. Creates a **searchable index** with the keywords "harvested".
    Searching. When you enter a specific keyword, veronica searches its harvested index and prints out (displays) all the matching entries and their addresses, thereby providing a list that you can then search.

Gopher via WWW gateway


Netscape:
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: gopher://gopher.tc.umn.edu:70/11/Other%20Gopher%20and%20Information%20Serv
What's Related
WebMail Radio People Yellow Pages Download Calendar Channels

Gopher Menu

All the Gopher Servers in the World
Search All the Gopher Servers in the World
Search titles in Gopherspace using veronica
Africa
Asia
Europe
International Organizations
Middle East
North America
Pacific
Russia
South America
Terminal Based Information
WAIS Based Information
Gopher Server Registration

Try: http://gopher.quux.org:70/Software/Gopher

Η Ιστορία των **Μηχανών Αναζήτησης** **<u>του Ιστού</u>**

- **1993**: early **web robots (**else called **spiders** or **robots)** were built to collect URL's:
  - Wanderer, ALIWEB (Archie-Like Index of the WEB), WWW Worm (indexed URL's and titles for regex search)

- **1994a**: Stanford grad students David Filo and Jerry Yang started manually collecting popular web sites into a topical hierarchy called **Yahoo**.
- **1994b**: Brian Pinkerton developed WebCrawler as a class project at U Wash. (eventually became part of **Excite** and AOL).
- **1994c**: Fuzzy Maudlin, a grad student at CMU developed **Lycos**. First to use a standard IR system as developed for the DARPA Tipster project. First to index a large set of pages.
- **1995**: DEC developed **Altavista**. Used a large farm of Alpha machines to quickly process large numbers of queries. Supported boolean operators, phrases, and "reverse pointer" queries.

Η Ιστορία των **Μηχανών Αναζήτησης** **<u>του Ιστού</u>**

- In 1998, Larry Page and Sergey Brin, Ph.D. students at Stanford, started **Google**. Main advance is use of *<u>link analysis</u>* to rank results partially based on authority.

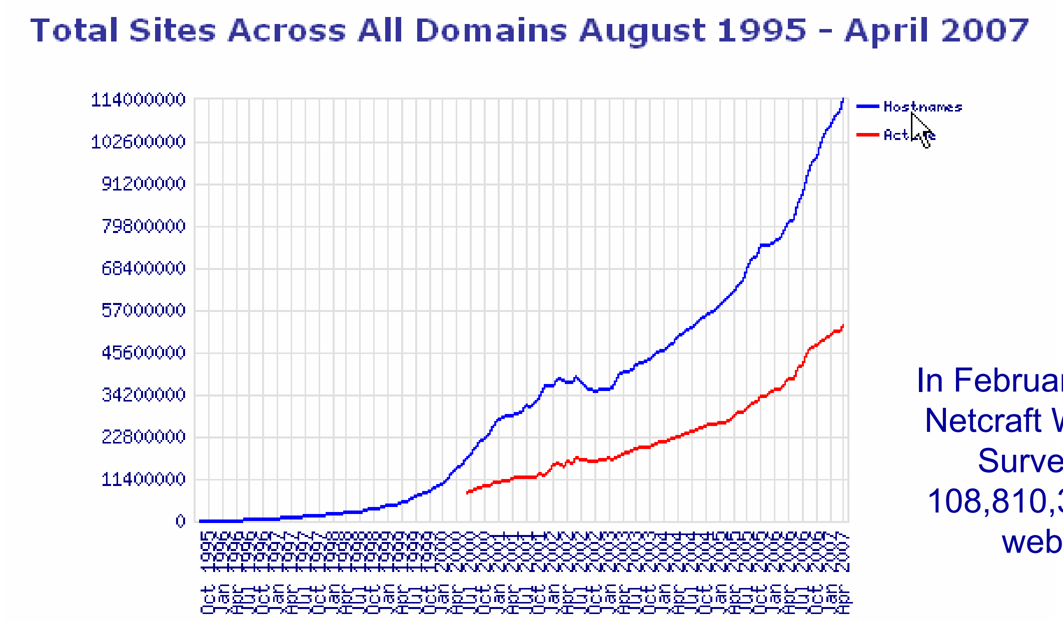
Ανάκτηση Πληροφοριών από τον Ιστό: Προκλήσεις

- Distributed Data: Documents spread over millions of different web servers.
- Volatile Data: Many documents change or disappear rapidly (e.g. dead links).
  - 23% of pages change daily
  - .com pages: 40% change daily, half-life=10 days (in 10 days half of the pages are gone)
- Large Volume: Billions of separate documents.
- Unstructured and Redundant Data: No uniform structure, HTML errors, up to 30% (near) duplicate documents.
- Quality of Data: No editorial control, false information, poor quality writing, typos, spam, etc.
- Heterogeneous Data: Multiple media types (images, video, VRML), languages, character sets, etc.

Ανάκτηση Πληροφοριών από τον Ιστό: Προκλήσεις και Απαιτήσεις

- Gathering techniques
- Scalable Index Structures efficiently updatable
- Improve the discrimination ability

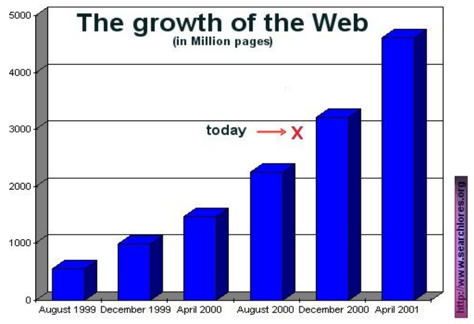
Number of Web sites



In February 2007, the Netcraft Web Server Survey found 108,810,358 distinct web sites.

Source: http://news.netcraft.com/archives/web_server_survey.html

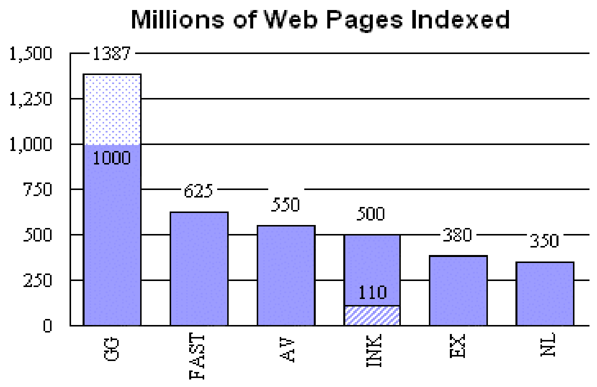
Number of Web Pages



Web pages in the world, February 2007:

- multiplying our estimate of the number of web pages per web site by Netcraft's February 2007 count of web sites, we arrive at **29.7 billion pages** on the World Wide Web as of February 2007.

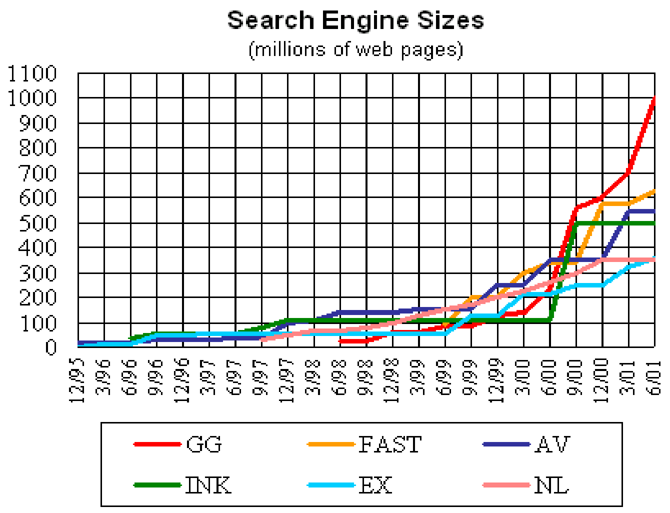
Number of Web Pages Indexed



SearchEngineWatch, Aug. 15, 2001

Assuming about 20KB per page,
1 billion pages is about 20 terabytes of data.

Growth of Web Pages Indexed



SearchEngineWatch, Aug. 15, 2001

Google lists current number of pages searched.
May 2005: **8 billion pages**

Ο νόμος του Ziph στον Παγκόσμιο Ιστό

Ο Νόμος του Ziph για τα κείμενα:

- *Η συχνότητα της i-th πιο συχνά εμφανιζόμενης λέξης είναι 1/i φορές η συχνότητα της πιο συχνής.*
  - *Πιο ακριβές: $1/i^{\theta}$ όπου θ μεταξύ 1.5 και 2*

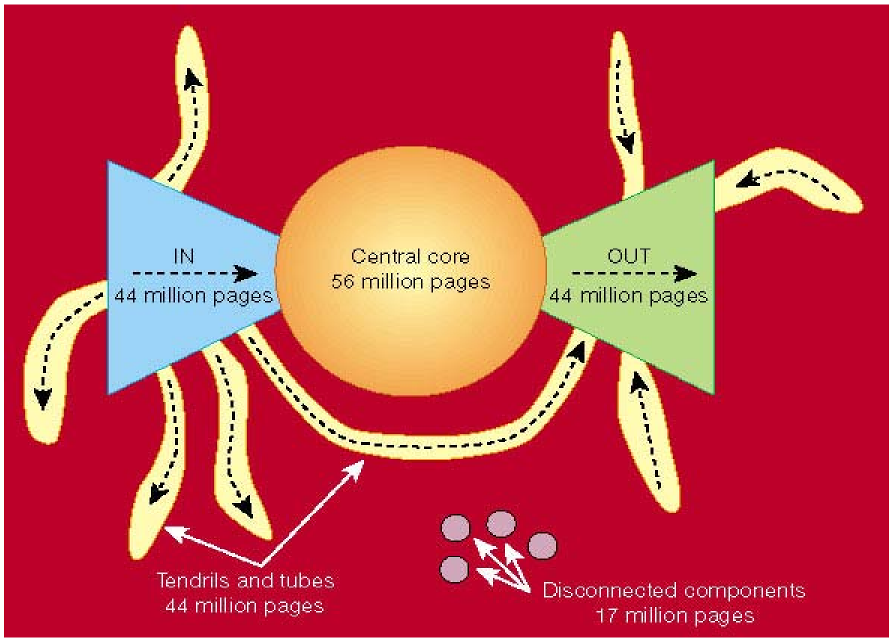
Ο Νόμος του Ziph στον Παγκόσμιο Ιστό:

- Number of in-links/out-links to/from a page has a Zipfian distribution.
  - the probability that a node has in-degree *i* is proportional to *1/i^x*, for some *x > 1*.
- Length of web pages has a Zipfian distribution.
- Number of hits to a web page has a Zipfian distribution.

Graph structure in the Web (I)

- Most (over 90%) of nodes form a single connected component if links are treated as *undirected* edges.
- This connected web breaks naturally into four pieces.
  - (1) The first piece is a central core, all of whose pages can reach one another along directed links -- this "giant strongly connected component" (*SCC*) is at the heart of the web.
  - *(2) IN* consists of pages that can reach the *SCC*, but cannot be reached from it
    - - possibly new sites that people have not yet discovered and linked to.
  - *(3) OUT* consists of pages that are accessible from the *SCC*, but do not link back to it,
    - such as corporate websites that contain only internal links.
  - (4) *TENDRILS* contain pages that cannot reach the *SCC,* and cannot be reached from the *SCC*.
- Each of the other three sets contain about 44 million pages -- thus, all four sets have roughly the same size.

Graph Structure of the Web (II)



http://www9.org/w9cdrom/160/160.html

Graph Structure in the Web (III)

- The diameter of the central core (*SCC*) is at least 28
- The diameter of the graph as a whole is over 500
- For randomly chosen source and destination pages, the probability that any path exists from the source to the destination is only 24%.
- If a directed path exists, its average length will be about 16.
- If an undirected path exists (i.e., links can be followed forwards or backwards), its average length will be about 6.
- In a sense the web is much like a complicated organism, in which the local structure at a microscopic scale looks very regular like a biological cell, but the global structure exhibits interesting morphological structure (body and limbs) that are not obviously evident in the local structure. Therefore, while it might be tempting to draw conclusions about the structure of the web graph from a local picture of it, such conclusions may be misleading.

Εύρεση πληροφορίας στον Παγκόσμιο Ιστό

Χειροποίητες Ταξινομίες Ιστού (Manual Hierarchical Web Taxonomies)

- Yahoo approach of using human editors to assemble a large hierarchically structured directory of web pages.
  - http://www.yahoo.com/
- Open Directory Project is a similar approach based on the distributed labor of volunteer editors (“net-citizens provide the collective brain”). Used by most other search engines. Started by Netscape.
  - http://www.dmoz.org/

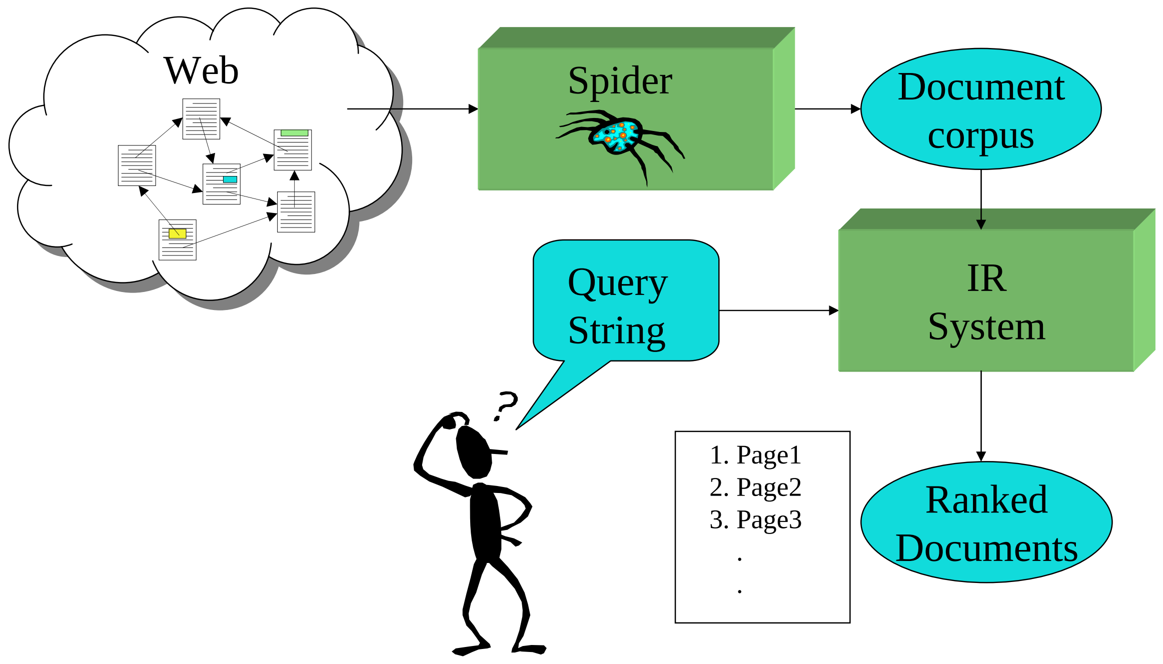
Αυτόματη Ταξινόμηση Σελίδων (Automatic Document Classification)

- Manual classification into a given hierarchy is labor intensive, subjective, and error-prone.
- **Text categorization** methods provide a way to automatically classify documents.
- Best methods based on training a *machine learning* (*pattern recognition*) system on a labeled set of examples (*supervised learning*).

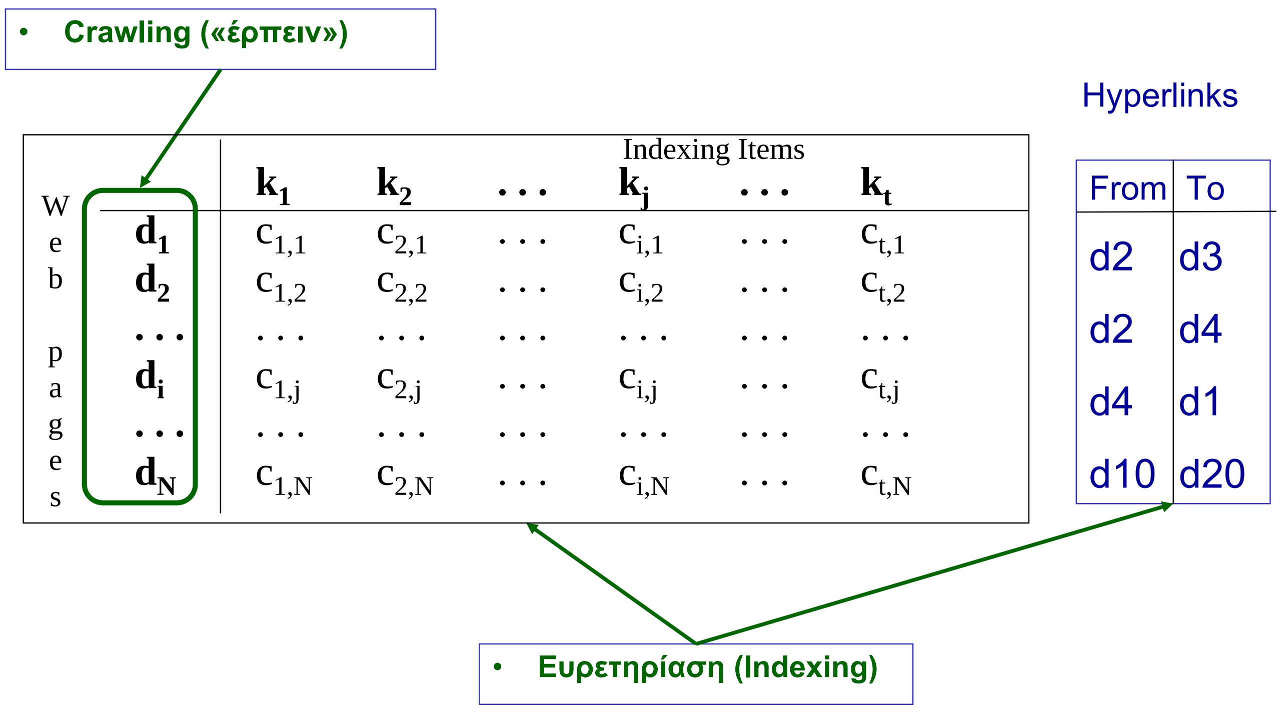
Αυτόματες Ιεραρχίες Εγγράφων (Automatic Document Hierarchies)

- Manual hierarchy development is labor intensive, subjective, and error-prone.
- It would nice to automatically construct a meaningful hierarchical taxonomy from a corpus of documents.
- This is possible with hierarchical text clustering (unsupervised learning).
  - Hierarchical Agglomerative Clustering (HAC)
  - ***Θα μιλήσουμε για Ομαδοποίηση (Clustering) σε επόμενο μάθημα***

Ανάκτηση Πληροφοριών στον Ιστό



Γενική μορφή ενός Ευρετηρίου για ανάκτηση πληροφοριών από τον Ιστό



Indexing Items

Web pages	k_1	k_2	...	k_j	...	k_t
d_1	$c_{1,1}$	$c_{2,1}$	...	$c_{i,1}$	...	$c_{t,1}$
d_2	$c_{1,2}$	$c_{2,2}$	...	$c_{i,2}$	...	$c_{t,2}$
...	...	...	...	...	...	...
d_i	$c_{1,j}$	$c_{2,j}$	...	$c_{i,j}$	...	$c_{t,j}$
...	...	...	...	...	...	...
d_N	$c_{1,N}$	$c_{2,N}$	...	$c_{i,N}$	...	$c_{t,N}$

From	To
d2	d3
d2	d4
d4	d1
d10	d20

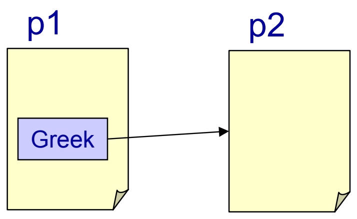
Τύποι Ευρετηρίων

Τύποι Ευρετηρίων

- Ευρετήριο **Κειμένων**
  - Π.χ. **Ανεστραμμένα Αρχεία (Inverted files)**, Αρχεία Καταλήξεων, Αρχεία Υπογραφών (γενικά ό,τι είδαμε στα προηγούμενα μαθήματα)
- Ευρετήριο **Συνδέσμων** (Link Index)
  - connectivity graph of the Web (nodes: pages, edges: links)
  - common tasks (find pages that point to page p, find pages pointed by p)
  - billion nodes => efficient data structure
- Ευρετήριο **Χρησιμότητας** (Utility Index)
  - e.g. Site -> page index, PageRank index, etc
  - *.(θα μιλήσουμε για αυτό σε επόμενο μάθημα)*

Ευρετηρίαση Ιστοσελίδων> **Anchor Text Indexing**

Μια διαφορά σε σχέση με αυτά που έχουμε δει μέχρι τώρα



vector(p2) = <Greek, … , >

- Extract **anchor text** (between <a> and </a>) of each link followed.
- Anchor text is usually **descriptive** of the document **to which it points**.
- Add anchor text to the content of the destination page to provide additional relevant keyword indices.

Ευρετηρίαση Ιστοσελίδων> **Anchor Text Indexing**

- Good Examples
  - <A HREF="www.csd.uoc.gr/~hy463"> IR Course in Greek </A>
  - <a href="http://www.ibm.com">IBM</a>
- Many times anchor text is not useful:
  - E.g. "click here"
  - <a href="http://www.microsoft.com">Evil Empire</a>
- Increases content more for popular pages with many in-coming links, increasing recall of these pages.
- May even give higher weights to tokens from anchor text.

Λίγα λόγια για το HTTP

HTTP: HyperText Transfer Protocol

HTTP: Hypertext Transfer Protocol

- It is an **application-level protocol** with the lightness and speed necessary for distributed, collaborative, hypermedia information systems (is has been in use by the World-Wide Web global information initiative since 1990).
- HTTP allows an open-ended set of methods to be used to indicate the purpose of a request. It builds on the discipline of reference provided by the Uniform Resource Identifier (URI), as a location (URL) or name (URN), for indicating the resource on which a method is to be applied. Messages are passed in a format similar to that used by Internet Mail and the Multipurpose Internet Mail Extensions (MIME).
- HTTP is also used as a generic protocol for communication between user agents and proxies/gateways to other Internet protocols, such as SMTP, NNTP, FTP, Gopher, and WAIS, allowing basic hypermedia access to resources available from diverse applications and simplifying the implementation of user agents.

HTTP: A request/response protocol

- The HTTP protocol is a r**equest/response** protocol. A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a connection with a server. The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity meta information, and possible entity-body content.
- HTTP/0.9: it was a simple protocol for raw data transfer across the Internet (this was the first version of HTTP)
- HTTP/1.0: (as defined by RFC 1945), improved the protocol by allowing messages to be in the format of MIME-like messages, containing meta information about the data transferred and modifiers on the request/response semantics. However, HTTP/1.0 does not sufficiently take into consideration the effects of hierarchical proxies, caching, the need for persistent connections, or virtual hosts.
- HTTP/1.1: includes more stringent requirements than HTTP/1.0 in order to ensure reliable implementation of its features.
- There is also a Secure HTTP (S-HTTP) specification.

HTTP protocol structure

The request message has the following format:

<Request Line, General header, Request header, Entity header, Message Body>

- The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The elements are separated by SP characters. No CR or LF is allowed except in the final CRLF sequence. The details of the general header, request header and entity header could be found in the reference documents.

The response message has the following format:

<Status Line, General header, Response header, Entity header, Message Body>

- The Status-Code element is a 3-digit integer result code of the attempt to understand and satisfy the request. We will see some codes later on. The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata and the Reason-Phrase is intended for the human user. The client is not required to examine or display the Reason-Phrase. The details of the general header, response header and entity header could be found in the reference documents.

Examples (HTTP 1.0)

- The simplest HTTP message is "GET *url*", to which the server replies by sending the named document. If the document doesn't exist, the server will probably send an HTML-encoded message stating this. I say *probably*, because this simple method offers poor error handling and has been deprecated in favor of the more elaborate scheme outlined below.

- A complete HTTP 1.0 message begins "GET *url* HTTP/1.0". The addition of the third field indicates that full headers are being used. The client may then send additional header fields, one per line, terminating the message with a blank link. The server replies in a similar vein, first with a series of header lines, then a blank line, then the document proper.

Examples (HTTP 1.0) cont

A sample HTTP 1.0 exchange:

```
GET / HTTP/1.0>
              >
            < HTTP/1.0 200 OK
            < Date: Wed, 18 Sep 1996 20:18:59 GMT
            < Server: Apache/1.0.0
            < Content-type: text/html
            < Content-length: 1579
            < Last-modified: Mon, 22 Jul 1996 22:23:34 GMT
            <
            < HTML document
```

Examples (HTTP 1.0) cont

The use of full headers is preferred for several reasons:

- The first line of a server header includes a response code indicating the success or failure of the operation.
- One of the server header fields will be Content-type:, which specifies a MIME type to describe how the document should be interpreted.
- If the document has moved, the server can specify its new location with a Location: field, allowing the client to transparently retry the request using the new URL.
- The Authorization: and WWW-Authenticate: fields allow access controls to be placed on Web documents.
- The Referer: field allows the client to tell the server the URL of the document that triggered this request, permitting savvy servers to trace clients through a series of requests.

In addition to GET requests, clients can also send HEAD and POST requests, of which POSTs are the most important. POSTs are used for HTML forms and other operations that require the client to transmit a block of data to the server. After sending the header and the blank line, the client transmits the data. The header must have included a Content-Length: field, which permits the server to determine when all the data has been received.

HTTP Status Codes

The status code is a three-digit integer, and the first digit identifies the general category of response:

- **1xx** indicates an informational message only
- **2xx** indicates success of some kind
- **3xx** redirects the client to another URL
- **4xx** indicates an error on the client's part
- **5xx** indicates an error on the server's part

The most common status codes are:

- **200 OK**
  - The request succeeded, and the resulting resource (e.g. file or script output) is returned in the message body.
- **404 Not Found**
  - The requested resource doesn't exist.
- **301 Moved Permanently**
  302 Moved Temporarily
  303 See Other *(HTTP 1.1 only)*
  - The resource has moved to another URL (given by the **Location:** response header), and should be automatically retrieved by the client. This is often used by a CGI script to redirect the browser to an existing file.
- **500 Server Error**
  - An unexpected server error. The most common cause is a server-side script that has bad syntax, fails, or otherwise can't run correctly.

HTTP Error Codes

- **400 -Bad Request -** The request could not be understood by the server due to incorrect syntax.
- **401 -Unauthorized User -** authentication is required.
- **403 - Forbidden -** The server understood the request, but is refusing to fulfill it.
- **404 - Page Not Found -** The server has not found anything matching the Request-URI.
- **405 - Method Not Allowed -** The method specified in the Request-Line is not allowed for the resource identified by the Request-URI.
- **406 - Not Acceptable -** The server cannot generate a response that the requestor is willing to accept.
- **407 - Proxy Authentication Required -** This code is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy.
- **408 - Request Timed Out -** The server stopped waiting for a client request.
- **409 - Conflict -** The request could not be completed due to a conflict with the current state of the resource.
- **410 - Gone -** The requested resource is no longer available at the server and no forwarding address is known. This condition is similar to 404, except that the 410 error condition is expected to be permanent.

HTTP Error Codes

- **411 - Length Required -** The server requires a content-length in the request.
- **412 - Precondition Failed -** The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server.
- **413 - Request Entity Too Large -** The server is refusing to process a request because the request entity is larger than the server is willing or able to process.
- **414 - Request URL Too Long** - The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret.
- **415 - Unsupported Media Type -** The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.
- **500 - Server Error -** Internal Web server error
- **501 - No Server -** Function not implemented in Web server software
- **502 - Server Overload -** Bad Gateway; a server being used by this Web server has sent an invalid response.
- **503 - Service Unavailable -** Service unavailable because of temporary overload or maintenance.
- **504 - Gateway Timeout -** A server being used by this server has not responded in time.
  505 - HTTP Version Not Supported - The server does not support the HTTP protocol version that was used in the request message.

Manually Experimenting with HTTP

Using telnet, you can open an interactive socket to an HTTP server. This lets you manually enter a request, and see the response written to your screen. It's a great help when learning HTTP, to see exactly how a server responds to a particular request. It also helps when troubleshooting. From a Unix prompt, open a connection to an HTTP server with something like

```
telnet www.somehost.com 80
```

Then enter your request line by line, like

```
GET /path/file.html HTTP/1.0
[headers here, if any]
[blank line here]
```

```
>telnet www.csd.uoc.gr 80
>GET / HTTP/1.0
>
```

- After you finish your request with the blank line, you'll see the raw response from the server, including the status line, headers, and message body.

From a Unix prompt write

```
>telnet www.ics.forth.gr 80
>GET http://www.ics.forth.gr/about.html  HTTP/1.0
>
```

Έρπειν
Crawling/Spidering

(Robots/Bots/Crawlers)

Crawling

Η γενική ιδέα:

- Start with a comprehensive **set of root URL's** from which to start the search.
- **Follow all links** on these pages recursively to find additional pages.
- Index all **novel** found pages in an inverted index as they are encountered.

May allow users to directly submit pages to be indexed (and crawled from).

Κρίσιμα ερωτήματα:

- *What pages should the crawler download ?*
  - *In most cases it cannot download all, thus we need to prioritize the URLs*
- *How should the crawler refresh pages?*
- *How should the load on the visited Web sites by minimized ?*
- *How should the crawling process be parallelized ?*

Μορφή Αλγορίθμου Διάσχισης (Spidering Algorithm)

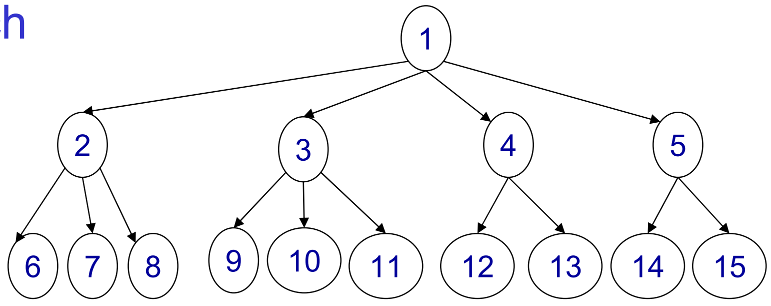
```
Initialize queue (Q) with initial set of known URL's.
Until Q empty or page or time limit exhausted:
    Pop URL, L, from front of Q.
    If L is not an HTML page (.gif, .jpeg, .ps, .pdf, .ppt…)
        continue loop.
    If already visited L, continue loop.
    Download page, P, for L.
    If cannot download P (e.g. 404 error, robot excluded)
        continue loop.
    Index P (e.g. add to inverted index or store cached copy).
    Parse P to obtain list of new links N.
    Append N to the end of Q.
```

Στρατηγικές Διάσχισης (Crawling Strategies)

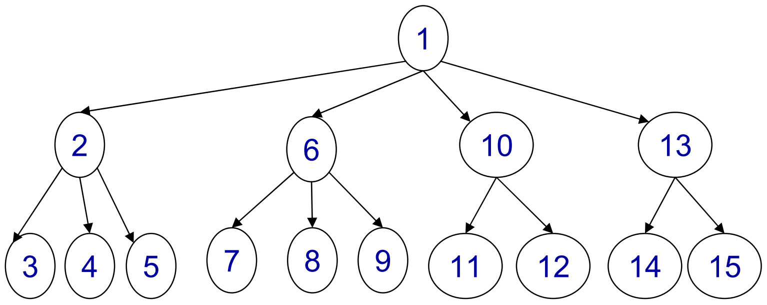
- (I) Breadth-first Search
- (II) Depth-first Search
- (III) Importance-first Search
  - Topic-directed search
  - Link-directed search
  - Location-directed search
  - Weighted combination of the above

Στρατηγικές Διάσχισης (Crawling Strategies)

(I) Breadth-first Search


$$\begin{tikzcd} & & & 1 \arrow[dlll] \arrow[dl] \arrow[d] \arrow[dr] & \\ 2 \arrow[d, "6,7,8"] & & 3 \arrow[d, "9,10,11"] & 4 \arrow[d, "12,13"] & 5 \arrow[d, "14,15"] \\ {} & & {} & {} & {} \end{tikzcd}$$

(II) Depth-first Search


$$\begin{tikzcd} & & & 1 \arrow[dlll] \arrow[dl] \arrow[d] \arrow[dr] & \\ 2 \arrow[d, "3,4,5"] & & 6 \arrow[d, "7,8,9"] & 10 \arrow[d, "11,12"] & 13 \arrow[d, "14,15"] \\ {} & & {} & {} & {} \end{tikzcd}$$

Στρατηγικές Διάσχισης: Σύγκριση

- (I) Breadth-first
  - explores uniformly outward from the root page but requires **memory for all nodes on the previous level** (exponential in depth). Standard spidering method.
- (II) Depth-first
  - requires **memory of only depth times branching-factor** (linear in depth) but gets "lost" pursuing a single thread.

Both strategies implementable using a queue of links (URL's).

- How new links added to the queue determines search strategy.
- FIFO (append to end of Q) gives **breadth-first** search.
- LIFO (add to front of Q) gives **depth-firs**t search.
- Heuristically ordering the Q gives a "**focused crawler**" that directs its search towards "interesting" pages.

Μορφή Αλγορίθμου Διάσχισης (Spidering Algorithm)

```
Initialize queue (Q) with initial set of known URL's.
Until Q empty or page or time limit exhausted:
    Pop URL, L, from front of Q.
    If L is not an HTML page (.gif, .jpeg, .ps, .pdf, .ppt…)
        continue loop.
    If already visited L, continue loop.
    Download page, P, for L.
    If cannot download P (e.g. 404 error, robot excluded)
        continue loop.
    Index P (e.g. add to inverted index or store cached copy).
    Parse P to obtain list of new links N.
    Append N to the end of Q.        // =>Breadth First Search)
    Insert N at the beginning of Q  // =>Depth First Search)
```

(III) Importance-first Search

Objective: Download and index **important** pages first

Related questions:

(A) What is the meaning of importance ?

(B) How a crawler operates ?

How a crawler guesses good pages to visit ?

(C) How we should refresh pages ?

(A) What is the meaning of importance ? (Importance Metrics)

Value(p) = k1 **ContentVal**(p) + k2 **PopularityVal**(p) + k3 **LocVal**(url(p))

- **ContentValue**(p)
  - can be based on a **driving query q**
  - e.g. ContentValue(p)= Cosine(vec(text(p)), **q**)
  - Είναι ο βαθμός ομοιότητας της p με το q βάσει του διανυσματικού μοντέλου.
    - Notice that **idf** (inverse document frequency) can be only estimated
      - We would know its value if we had downloaded all pages of the Web
  - We could call this *topic-directed search*
- **Popularity**(p)
  - backlink count: e.g. PopularityVal(p)=| { p' | p' -> p }|
  - Related to *citation count* (coming from bibliometrics)
  - Again, this can be only estimated (because we have not completed crawling)
  - We could call this *link-directed search*
- **LocationValue**(url(p))
  - e.g. high if ".com", low if it contains "home", low if it contains many slashes «/»
  - We could call this *location-directed search*

You could think many other metrics

(B) How a crawler operates ? (crawler models of operation)

Design a crawler that if possible visits high importance pages before lower ranked ones (given an imporance metric).

Crawler models:

- ***Crawl & Stop***
  - Starts at its initial page p0 and stops after visiting K pages
  - Ideally these should be the K pages with the highest value
    - But this it is again impossible to know (unless downloading the entire Web)
- ***Crawl & Stop with Threshold***
  - Starts at its initial page p0 and stops after visiting K pages whose **value** is greater than a threshold
  - …

(C) How we should refresh pages ?
Keeping Spidered Pages Up to Date

- Web is very dynamic: many <u>new</u> pages, <u>updated</u> pages, <u>deleted</u> pages, etc.
- **Periodically** check spidered pages for updates and deletions:
  - Just look at **header info** (e.g. META tags on last update) to determine if page has changed, only reload entire page if needed.
    - Κατά την επικοινωνία με τον web server μπορούμε να χρησιμοποιήσουμε το την επικεφαλίδα (header) **If-Modified-Since** έτσι ώστε εάν η σελίδα δεν έχει αλλάξει, να μην την «κατεβάσει» ο crawler.
- **Track how often** each page is updated and preferentially return to pages which are historically more dynamic.
- Preferentially update pages that are **accessed more often** to optimize <u>freshness</u> of <u>more popular pages</u>.

Avoiding Page Duplication

- Must detect when revisiting a page that has already been spidered (web is a graph not a tree).
- Must efficiently index visited pages to allow rapid recognition test.
  - Tree indexing (e.g. trie)
  - Hashtable
- Index page using <u>URL</u> as a key.
  - Must canonicalize URL's (e.g. delete ending "/")
  - Not detect duplicated or mirrored pages.
- Index page using <u>textual content</u> as a key.
  - Requires first downloading page.

URL Syntax and Link Extraction

- URL Syntax
  - <scheme>://<authority><path>?<query>#<fragment>
  - An *authority* has the syntax:
    - <host>:<port-number>
  - A *query* passes variable values from an HTML form and has the syntax:
    - <variable>=<value>&<variable>=<value>…
  - A *fragment* is also called a *reference* or a *ref* and is a pointer within the document to a point specified by an anchor tag of the form:
    - <A NAME="<fragment>">
- Link Extraction
  - Must find all links in a page and extract URLs.
  - Must complete relative URL's using current page URL:
    - <a href="project"> to http://www.csd.uoc.gr/~hy463/project
    - <a href="../index.html"> to http://www.csd.uoc.gr/~hy463/index.html
  - BASE tag in the header section of an HTML file changes the base URL for all relative pointers:
    - <BASE HREF="<base-URL>">

Κανονικοποίηση Συνδέσμων (Link Normalization)

- Equivalent variations of ending directory normalized by removing ending slash.
  - http://www.csd.uoc.gr/~hy463/
  - http://www.csd.uoc.gr/~hy463
- Internal page fragments (ref's) removed:
  - http://www.csd.uoc.gr/~hy463/index.html
  - http://www.csd.uoc.gr/~hy463/index.html#grades
- http://WWW.cSd.uoc.gR:80/ equiv with http://www.csd.uoc.gr:

Link Extraction in Java

- Java Swing contains an HTML parser.
- In HandleStartTag, if it is an “A” tag, take the HREF attribute value as an initial URL.
- Complete the URL using the base URL:
  - new URL(URL baseURL, String relativeURL)
  - Fails if baseURL ends in a directory name but this is not indicated by a final “/”
  - Append a “/” to baseURL if it does not end in a file name with an extension (and therefore presumably is a directory).

Also:
http://htmlparser.sourceforge.net/javadoc_1_3/org/htmlparser/
parserapplications/LinkExtractor.html

Restricting Crawling

- Restrict spider to a particular site.
  - Remove links to other sites from the queue Q.
- Restrict spider to a particular directory.
  - Remove links not in the specified directory.
- Obey page-owner restrictions (robot exclusion).

  - Web sites and pages can specify that robots should not crawl/index certain areas.
  - Two ways:
    - Robots Exclusion Protocol: Site wide specification of excluded directories.
    - Robots META Tag: Individual document tag to exclude indexing or following links.

Robots Exclusion Protocol

- Site administrator puts a "robots.txt" file at the root of the host's web directory.
  - www.enet.gr/robots.txt
  - http://www.cnn.com/robots.txt
- File is a list of excluded directories for a given robot (user-agent).
  - Exclude all robots from the entire site:

```
User-agent: *
Disallow: /
```

Robot Exclusion Protocol Examples

- Exclude specific directories:

```
User-agent: *
Disallow: /tmp/
Disallow: /cgi-bin/
Disallow: /users/paranoid/
```

- Exclude a specific robot:

```
User-agent: GoogleBot
Disallow: /
```

- Allow a specific robot:

```
User-agent: GoogleBot
Disallow:
```

- Some Details
  - Only use blank lines to separate different User-agent disallowed directories.
  - One directory per "Disallow" line.
  - No regex patterns in directories.

Δείτε:
www.enet.gr/robots.txt:
User-agent: *
Disallow: /past/
Disallow: /online/

http://www.cnn.com/robots.txt

```
User-agent: *
Disallow:   /cgi-bin
Disallow:   /TRANSCRIPTS
Disallow:   /development
Disallow:   /third
Disallow:   /beta
Disallow:   /java
Disallow:   /shockwave
Disallow:   /JOBS
Disallow:   /pr
Disallow:   /Interactive
Disallow:   /alt_index.html
Disallow:   /webmaster_logs
Disallow:   /newscenter
Disallow:       /virtual
Disallow:       /DIGEST
Disallow:       /QUICKNEWS
Disallow:       /SEARCH
```

```
User-agent:Mozilla/3.01 (hotwired-
      test/0.1)
Disallow:   /cgi-bin
Disallow:   /TRANSCRIPTS
Disallow:   /development
Disallow:   /third
Disallow:   /beta
Disallow:   /java
Disallow:   /shockwave
Disallow:   /JOBS
Disallow:   /pr
Disallow:   /Interactive
Disallow:   /alt_index.html
Disallow:   /webmaster_logs
Disallow:   /newscenter
Disallow:       /virtual
Disallow:       /DIGEST
Disallow:       /QUICKNEWS
Disallow:       /SEARCH
```

```
User-agent: GoogleBot
Disallow: /cgi-bin
Disallow: /java
Disallow: /images
Disallow: /development
Disallow:   /third
Disallow: /beta
Disallow: /webmaster_logs
Disallow: /virtual
Disallow: /shockwave
Disallow: /TRANSCRIPTS
Disallow:   /newscenter
Disallow:  /virtual
Disallow:  /DIGEST
Disallow:  /QUICKNEWS
Disallow:  /SEARCH
Disallow:   /alt_index.html
```

Robots META Tag

- Include META tag in HEAD section of a specific HTML document.
  - <meta name="robots" content="none">
- Content value is a pair of values for two aspects:
  - index | noindex: Allow/disallow indexing of this page.
  - follow | nofollow: Allow/disallow following links on this page.
- Special values:
  - all = index,follow
  - none = noindex,nofollow
- Examples:

```
<meta name="robots" content="noindex,follow">
<meta name="robots" content="index,nofollow">
<meta name="robots" content="none">
```

Robot Exclusion Issues

- META tag is newer and less well-adopted than “robots.txt”.
- Standards are conventions to be followed by “good robots.”
- Companies have been prosecuted for “disobeying” these conventions and “trespassing” on private cyberspace.
- “Good robots” also try not to “hammer” individual sites with lots of rapid requests.
  - “Denial of service” attack.

Πολυνηματικός Ερπυστής (Multi-Threaded Crawling)

- Bottleneck is network delay in downloading individual pages.
- Best to have multiple threads running in parallel each requesting a page from a different host.
- Distribute URL’s to threads to guarantee equitable distribution of requests across different hosts to maximize throughput and avoid overloading any single server.
- Early Google spider had multiple co-ordinated crawlers with about 300 threads each, together able to download over 100 pages per second.

Αποθήκη Σελίδων (Page Repository)

Objective

- for storing pages downloaded by crawler
- provide a fast API for indexer

Challenges

- scalability
  - seamlessly distribute the repository across a **cluster of computers and disks**
    - one computer is not enough
- dual access mode
  - **random access**: retrieve quickly a specific page (given its id)
  - **streaming access**: retrieve quickly the entire set or a subset (e.g. all .edu) pages
- large bulk updates
  - high rate of modifications (additions/deletions of pages, read vs. write accesses)
- obsolete pages
  - detect and remove obsolete pages

Σχεδιάζοντας μια **Κατανεμημένη Αποθήκη Σελίδων (**Distributed Page Repository)

(A) Page distribution across nodes

- Uniform (all nodes are treated identically)
- Hash distribution policy
  - allocation of pages to nodes depends on the page identifier (URL)

(B) Physical page organization within a single node

- Operations to support (page addition/insertion, high speed streaming,random page access)
- Hash-based organization
  - treats a disk as a set of hash buckets each of which is small to fit in memory
  - pages are allocated to buckets depending on their URL
- Log-based organization
  - incoming pages are appended. Random access is supported using a separate B-tree index
- Hybrid approaches (e.g. Hash-Log organization)

Σχεδιάζοντας μια **Κατανεμημένη Αποθήκη Σελίδων** (II)

(C) Update strategies: Execution Mode

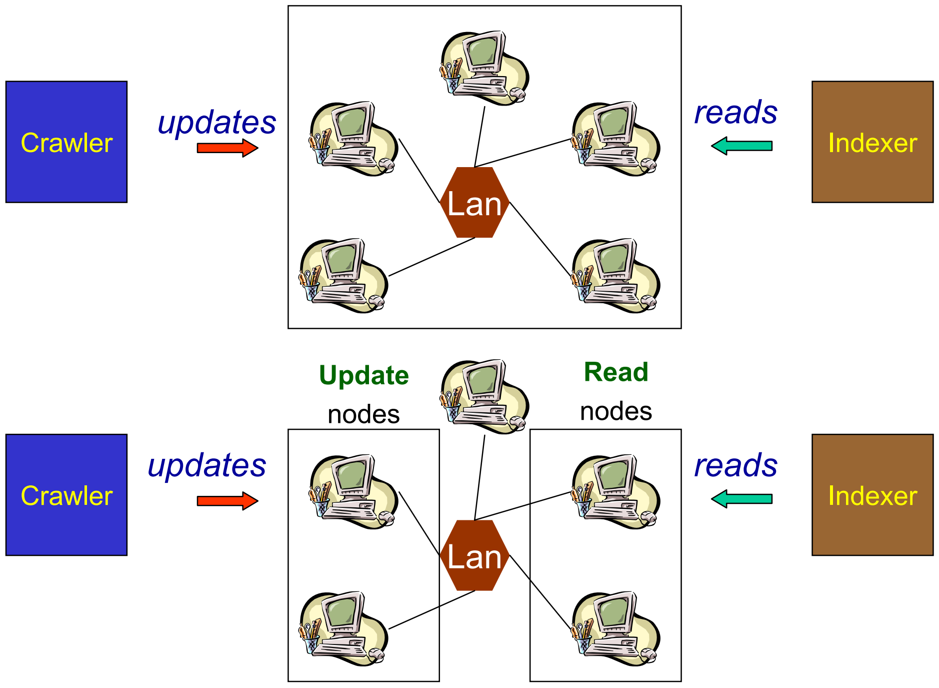
- **Steady mode**
  - crawler runs without any stop
- **Batch mode**
  - crawler is executed periodically (e.g. twice every month)
  - stops by time-limit or by amount of pages found
  - **Batch mode Partial**
    - recrawls only a specific set of pages or sites at each execution
  - **Batch mode Complete**
    - recrawls the entire collection at each execution

Σχεδιάζοντας μια **Κατανεμημένη Αποθήκη Σελίδων (III)**

(C) Update strategies: InPlace vs. Shadowing updates

- **In-Place updates**
  - pages received from the crawler are directly integrated into the repository's existing collections (possibly replacing older versions)
- **Shadowing**
  - pages from crawler are stored separately and updates are applied in a separate step
  - allows separating between **update** and **read** accesses
    - a single storage node does not have to concurrently handle page addition and page retrieval
    - may degrade the freshness of the repository

Updates: In-Place vs. Shadowing (with read and update nodes)



Ευρετηρίαση Ιστοσελίδων: Κατασκευή Ανεστραμμένου Αρχείου

- For each term with a have a sorted list of locations.
- Extra: we may store if the words appears in bold face, in heading, …
  - Commonly we multiply their weight with a constant
- Building the index
  - due to frequent update rate, usually we **rebuild** the index (incremental updates perform poorly)
- Partitioning the index
  - by pages
    - each node is responsible for a disjoint set of pages
  - by terms
    - each node is responsible for a disjoint set of terms
  - *(θα μιλήσουμε για αυτά σε επόμενο μάθημα (Distributed IR))*
- Pipelined index build (loading (LAN) preprocessing (CPU) and flushing (DISK) concurrently

Διάρθρωση Διάλεξης

- Ιστορική Αναδρομή
- Web Challenges and Requirements for IR
- Zipf Law in the Web
- Graph Structure of the Web
- Yahoo/ODP vs Search Engines
  - Automatic Document Classification
  - Automatic Document Hierarchies
- Crawling/Spidering
  - Διάσχιση (spidering/crawling)
  - Depth/Breadth and Technical Details
  - Directed (Topic/Link/…) Spidering
  - Multi-Threaded Spidering
- Αποθήκευση και Ευρετηρίαση Σελίδων

Διαβάστε:

http://groogle.csd.uoc.gr/apache2-default/index.php/Crawler

Searching the Web, A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke and S. Raghavan,TOIT,2001