Web Search Basics and Web Crawling

CS 7263 Information Retrieval
Lecture 01

Jiho Noh

Department of Computer Science
Kennesaw State University

Fall 2025

KENNESAW STATE UNIVERSITY
COLLEGE OF COMPUTING AND SOFTWARE ENGINEERING

The World Wide Web

- The Web is a system of interlinked hypertext documents accessed via the Internet.
- Developed by Tim Berners-Lee in 1989 at CERN
    - To organize research documents for scientists.
    - Combined the idea of *hypertext* to link documents with the documents available via FTP.
    - Developed HTTP network protocol, URLs, HTML, and the first web browser.

Early Search Engines

1990 Alan E. Hall's **Archie** was the first search engine

- Indexes FTP sites, not the web
- Searchable database of file names, allowing regular expressions (regex) search

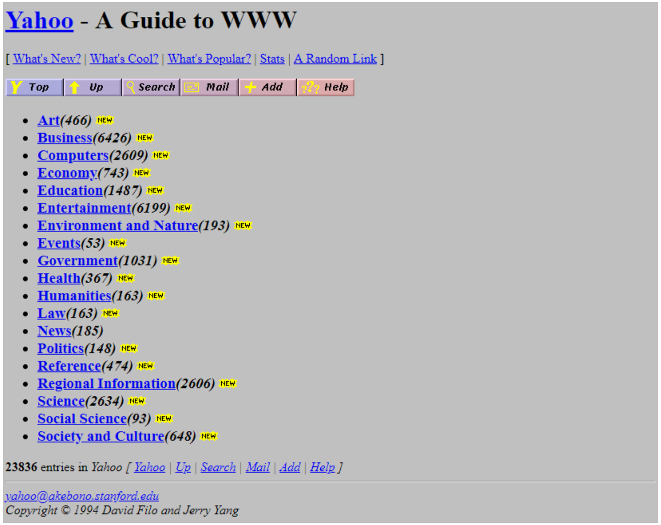
1993 **Gopher** was the first search engine to index the web; searching names of text files

1993 Spider robots were developed to crawl the web and collect URL's

1994 Stanford grad students David Filo and Jerry Yang created **Yahoo!**

- A directory (hierarchy) of web pages, not a search engine
- Manually collecting popular web pages
- Users could submit their own pages to be indexed

Directory of Web Pages



Yahoo - A Guide to WWW

[What's New? | What's Cool? | What's Popular? | Stats | A Random Link]

- Art*(466)* NEW
- Business*(6426)* NEW
- Computers*(2609)* NEW
- Economy*(743)* NEW
- Education*(1487)* NEW
- Entertainment*(6199)* NEW
- Environment and Nature*(193)* NEW
- Events*(53)* NEW
- Government*(1031)* NEW
- Health*(367)* NEW
- Humanities*(163)* NEW
- Law*(163)* NEW
- News*(185)*
- Politics*(148)* NEW
- Reference*(474)* NEW
- Regional Information*(2606)* NEW
- Science*(2634)* NEW
- Social Science*(93)* NEW
- Society and Culture*(648)* NEW

23836 entries in *Yahoo [Yahoo | Up | Search | Mail | Add | Help]*

yahoo@akebono.stanford.edu
Copyright © 1994 David Filo and Jerry Yang

Early Search Engines (Cont.)

1994 Brian Pinkerton developed WebCrawler at the University of Washington

- Eventually became part of Excite and AOL

1998 Larry Page and Sergey Brin developed **BackRub** (Google's original name) at Stanford

- A search engine that used *PageRank* algorithm to rank web pages

1998 Microsoft launched **MSN Search** (now Bing)

- A web search engine that used a combination of algorithms and human editors to rank web pages
- Bing now evolved focused on features like image and video search, and providing contextual search suggestions

Core Components of Traditional Search Engines

- **Crawling**
  - Discovering and fetching web pages from the internet
- **Indexing**
  - Organizing and storing the crawled data in a way that allows for efficient retrieval
- **Query Processing**
  - Analyzing and interpreting user queries to retrieve relevant results
- **Ranking**
  - Determining the order in which search results are presented to the user

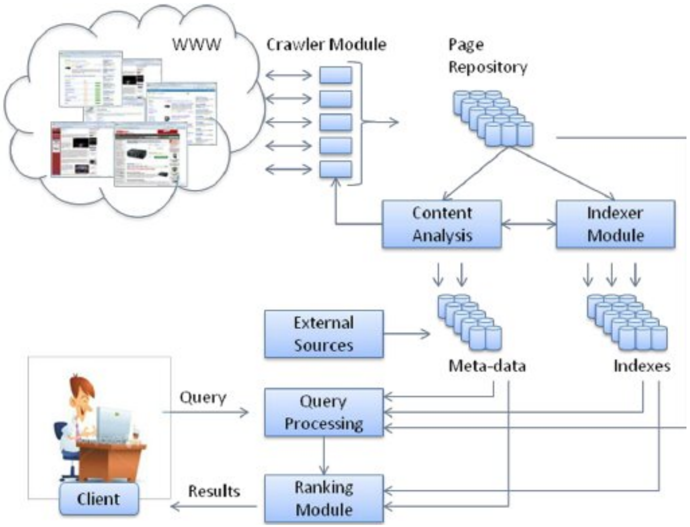


Figure: The Architecture of a Web Search Engine; image by Michalis Vazigiannis

Challenges of Web Search

- **Scale of the Web**
  - Large volume of data
  - Distributed nature: Documents spread across millions of web servers
- **Language understanding**
  - Language Ambiguity: Words can have multiple meanings, and the same meaning can be expressed in different ways
  - Unstructured data: Most web content is unstructured, making it difficult to extract relevant information
- **Heterogeneous content**
  - Different formats and media types: Web pages can be in HTML, PDF, images, videos, etc.
  - Different languages: The web is multilingual, and search engines need to handle multiple languages

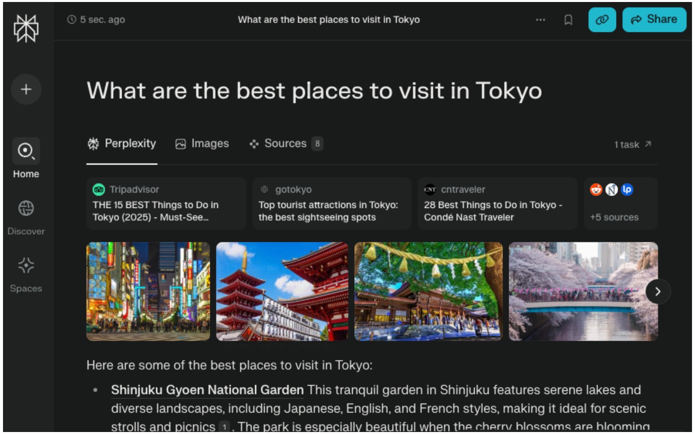
Limitations of keyword search

- **Keyword Search**
  - Users enter keywords, and the search engine retrieves documents containing those keywords
  - Limited understanding of user intent and context
  - Often returns irrelevant or low-quality results

Rise of Conversational AI and LLMs

- **Conversational AI**
  - AI systems that can engage in natural language conversations with users
  - Examples: Chatbots, virtual assistants (e.g., Siri, Alexa)
- **Large Language Models (LLMs)**
  - Deep learning models trained on vast amounts of text data
  - Capable of understanding and generating human-like text
  - Examples: GPT-3, BERT, T5

Case Study: Perplexity AI



What are the best places to visit in Tokyo

Tripadvisor
THE 15 BEST Things to Do in Tokyo (2025) - Must-See...

gotokyo
Top tourist attractions in Tokyo: the best sightseeing spots

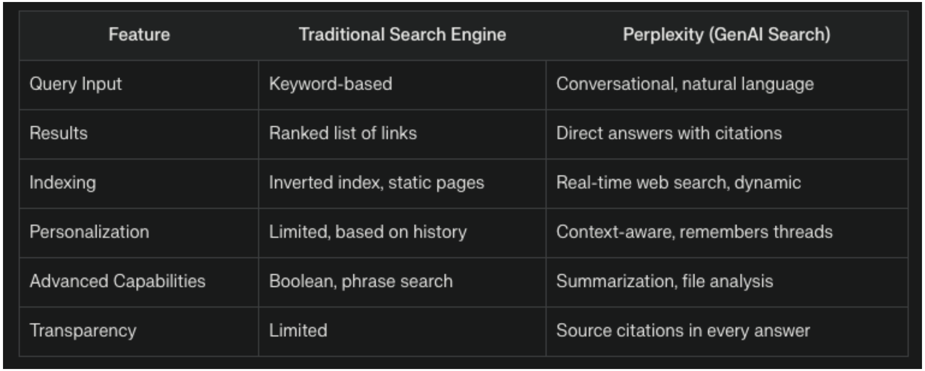
cntraveler
28 Best Things to Do in Tokyo - Condé Nast Traveler

+5 sources

Here are some of the best places to visit in Tokyo:

- **Shinjuku Gyoen National Garden** This tranquil garden in Shinjuku features serene lakes and diverse landscapes, including Japanese, English, and French styles, making it ideal for scenic strolls and picnics 1 . The park is especially beautiful when the cherry blossoms are blooming

How Perplexity AI Works



Feature	Traditional Search Engine	Perplexity (GenAI Search)
Query Input	Keyword-based	Conversational, natural language
Results	Ranked list of links	Direct answers with citations
Indexing	Inverted index, static pages	Real-time web search, dynamic
Personalization	Limited, based on history	Context-aware, remembers threads
Advanced Capabilities	Boolean, phrase search	Summarization, file analysis
Transparency	Limited	Source citations in every answer

Figure: Perplexity generated answer to "GenAI-based search engines"

Web Crawling

- **Web crawling** is the process of locating, fetching, and storing the pages available in the Web
- A program that automatically downloads web pages is called a *web crawler*
- Follow all links on these visited pages recursively to find additional pages.
- Other names of web crawlers are:
    - spiders
    - scrapers
    - robots
    - harvesters

Retrieving Web Pages

- Each web page has its own unique *Uniform Resource Locator (URL)*
- Most web pages can be retrieved via *Hypertext Transfer Protocol (HTTP)*

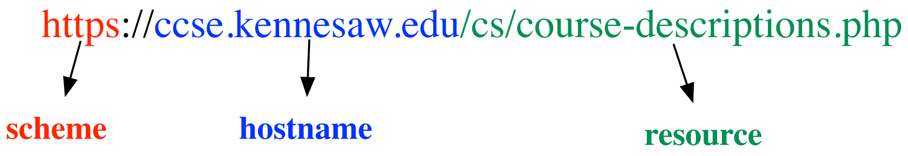
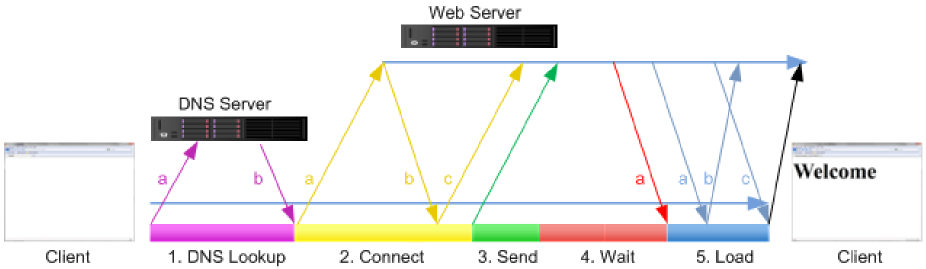


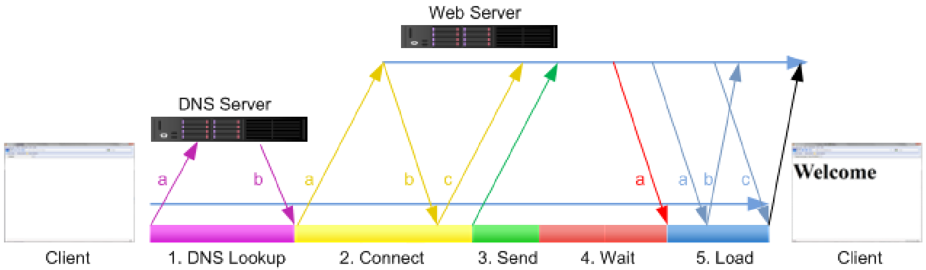
Figure: A uniform resource locator (URL)

Simple HTTP Transaction



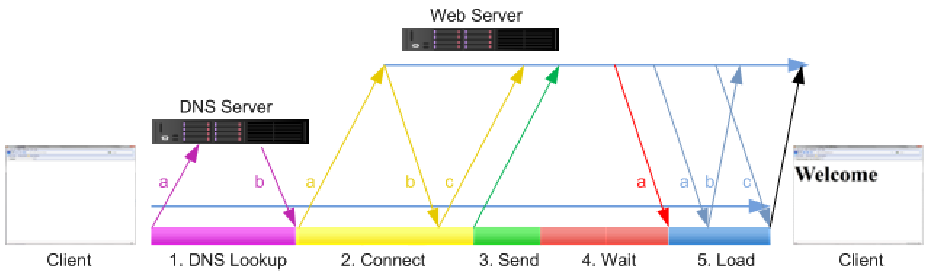
- **(1). DNS Lookup**: A client (e.g., browser or crawler) request IP for a domain name
- DNS server responses with the IP address for a requested domain name

Simple HTTP Transaction (cont.)



- **(2). Connect**: Client establishes TCP connection with the IP address
- Client sends SYN packet
- Web server sends SYN-ACK packet
- Client sends ACK packet, concluding the *three-way handshake* for TCP connection

Simple HTTP Transaction (cont.)



- **(3). Send**: Client sends the HTTP request.
- **(4). Wait**: Client waits for the server to respond to the request.
- **(5). Load**: Client keeps communicating with the server to load all the TCP segments.
- Client sends a a FIN packet to close the TCP connection.

HTTP Requests

An *HTTP Request* is made by a client to a web server for accessing a resource on the server.

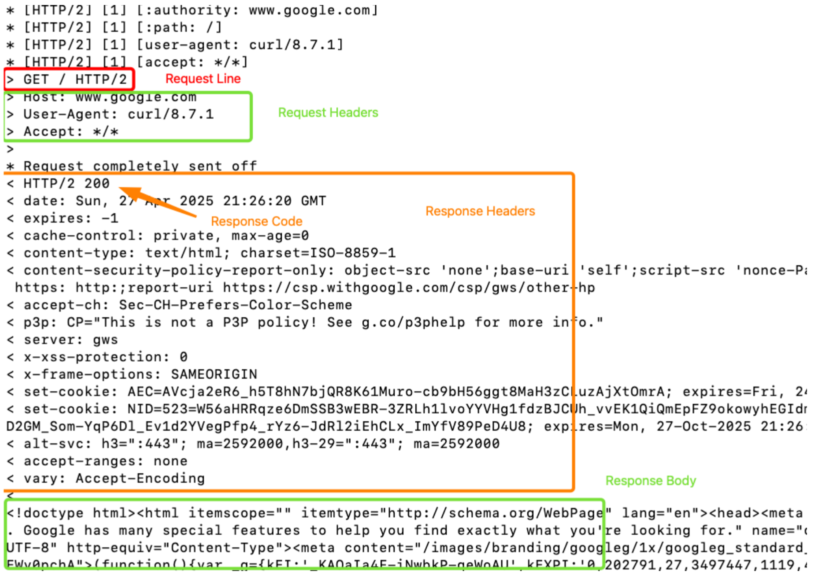
HTTP request components

- A request line (`<method><url><HTTP version>`)
- A series of HTTP headers
- A message body, if needed

What's going on "under the hood" using cURL

- cURL is a command-line tool for transferring data with URLs
- With `-verbose` option, cURL shows the details of the HTTP request and response
- The printout contains four components:
  - Additional info. such as http transaction info, provided by curl (prefixed *)
  - "header data" sent by curl (prefixed >)
  - "header data" received from the server (prefixed <)

```
» curl -v -s -stderr https://www.google.com
```



```
* [HTTP/2] [1] [:authority: www.google.com]
* [HTTP/2] [1] [:path: /]
* [HTTP/2] [1] [user-agent: curl/8.7.1]
* [HTTP/2] [1] [accept: */*]
> GET / HTTP/2
> Host: www.google.com
> User-Agent: curl/8.7.1
> Accept: */*
>
* Request completely sent off
< HTTP/2 200
< date: Sun, 27 Apr 2025 21:26:20 GMT
< expires: -1
< cache-control: private, max-age=0
< content-type: text/html; charset=ISO-8859-1
< content-security-policy-report-only: object-src 'none';base-uri 'self';script-src 'nonce-P
 https: http:;report-uri https://csp.withgoogle.com/csp/gws/other-hp
< accept-ch: Sec-CH-Prefers-Color-Scheme
< p3p: CP="This is not a P3P policy! See g.co/p3phelp for more info."
< server: gws
< x-xss-protection: 0
< x-frame-options: SAMEORIGIN
< set-cookie: AEC=AVcja2eR6_h5T8hN7bjQR8K61Muro-cb9bH56ggt8MaH3zCLuzAjXtOmrA; expires=Fri, 2
< set-cookie: NID=523=W56aHRRqze6DmSSB3wEBR-3ZRLh1lvoYYVHg1fdzBJCUh_vvEK1QiQmEpFZ9okowyhEGId
D2GM_Som-YqP6Dl_Ev1d2YVegPfp4_rYz6-JdRl2iEhCLx_ImYfV89PeD4U8; expires=Mon, 27-Oct-2025 21:26
< alt-svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
< accept-ranges: none
< vary: Accept-Encoding
<
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head><meta
. Google has many special features to help you find exactly what you're looking for." name="
UTF-8" http-equiv="Content-Type"><meta content="/images/branding/googleg/1x/googleg_standard
EWv0pchA">(function(){var _g={kEI:'_KAOaIa4F-iNwbkP-qeWoAU',kEXPI:'0,202791,27,3497447,1119,
```

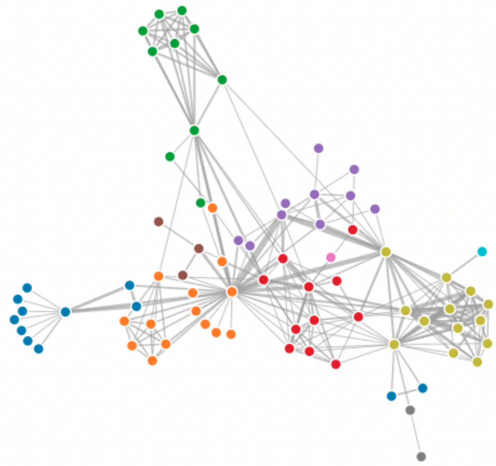
HTTP Request Methods

Method	Description
GET	Requests a representation of the specified resource. Requests using GET should only retrieve data.
HEAD	Asks for a response identical to a GET request, but without the response body.
POST	Submits an entity to the specified resource, often causing a change in state or side effects on the server. (e.g., submitting form inputs)
PUT	Replaces all current representations of the target resource with the request payload. (e.g., saving an object)
DELETE	Deletes the specified resource

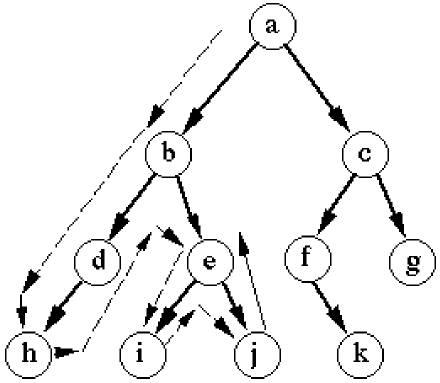
- Full list of HTTP Methods
- Full list of HTTP Response Stat codes

Web Graph

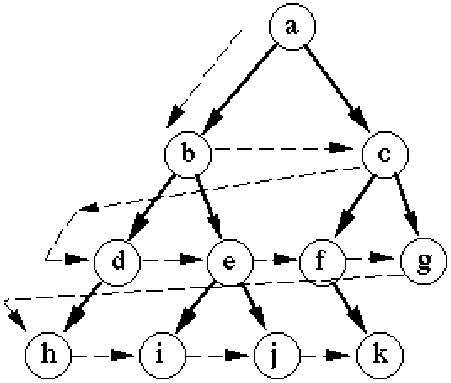
Web crawlers exploit the hyperlink structure of the Web



DFS vs. BFS



Depth-first search



Breadth-first search

DFS vs. BFS

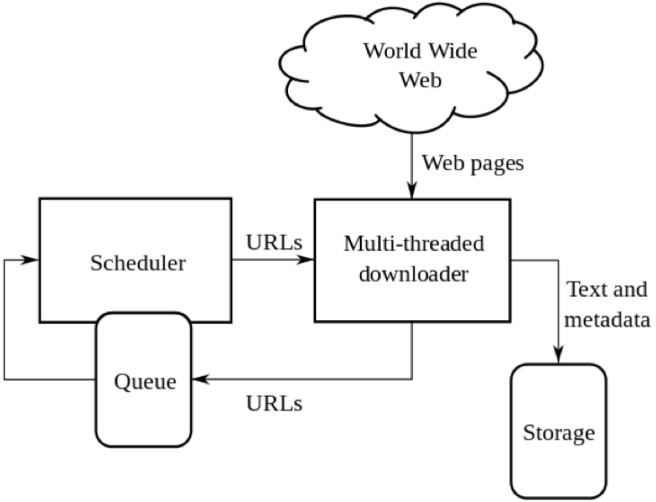
- **Depth-First Search (DFS)**
  - Requires less memory
  - A seed page may not get crawled in the end
- **Breadth-First Search (BFS)**
  - Explores the Web uniformly outward
  - Requires memory of all nodes on the previous level (exponential in depth)
  - Can avoid too many requests to a single host
- **Both,**
  - Implemented using a queue of links (FIFO, LIFO)
  - Heuristically ordering the Queue give a "focused crawler" that directs its search towards "interesting" pages.

Web Crawling Process

A crawler maintains a **frontier** with a collection of pages to be crawled and iteratively fetches and stores pages from it.

1. Initialize the frontier with a list of seed pages
2. Selects the next page carefully, with the politeness and performance in considerations
3. Downloads the selected pages, and extracts the hyperlinks within
4. Locates new pages by parsing the downloaded pages
5. Processes the new URLs and filters before being added to the frontier
6. Continues downloading until the fetch queue gets empty or a satisfactory number of pages are downloaded.

Web Crawling Process

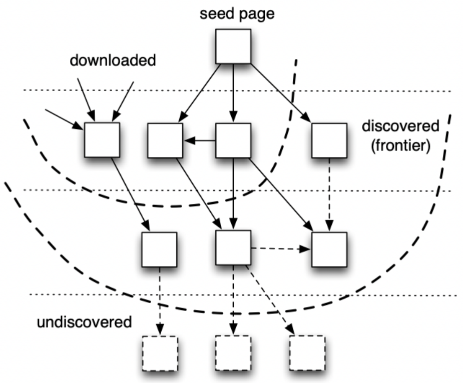


Web Crawling — Frontier

Crawler frontier is a data structure used to store eligible URLs for crawling. It can be seen as a priority queue.

Crawler divides the Web into three sets

- downloaded
- discovered
- undiscovered



URL Prioritization

A state-of-the-art web crawler maintains two separate queues for prioritizing the download of URLs:

Discovery queue

- Downloads pages pointed by already discovered links
- Tries to increase the **coverage** of the crawler

Refreshing queue

- re-downloads already downloaded pages
- tries to increase the **freshness** of the repository

Goals of Web Crawlers

- **Coverage**
  - No crawler can index all the Internet content,
  - Hence it should prioritize high-quality content
  - Discovering new pages and web sites as they appear online
- **Freshness**
  - The recency of the content in your index
  - A crawler should measure the rate at which each page changes
  - A crawler should balance between the coverage and freshness goals
- **Politeness**
  - Be respectful of other's web service resources by obeying the requested policies
  - Put a delay between a consecutive downloads from the same server

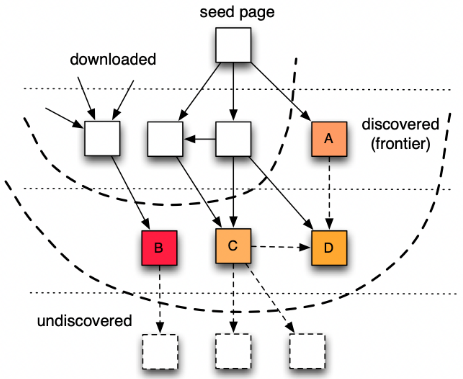
URL Selection Strategies

Methods to select the next page to crawl

- **Breadth-First Search (BFS)**: This distributes requests across domains relatively well and tends to download high-PageRank pages early
- **Backlink count**: Prioritize pages with more in-links from already-crawled pages.
- **Larger sites first**: Prioritize pages on domains with many pages in the frontier.
- **Partial PageRank**: Approximate PageRank scores are calculated based on already-crawled pages.

* **PageRank** measures the importantce of a webpage by link analysis

URL Selection Strategies (Discovery)



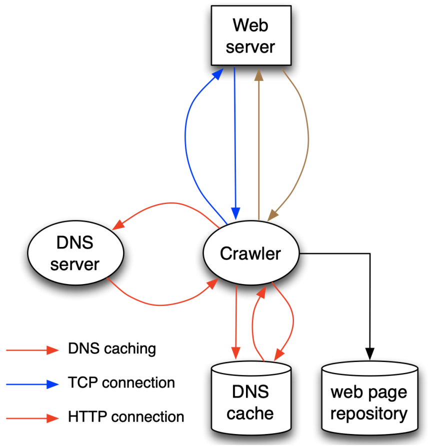
* More intense red color indicates higher PageRank score

Which webpage will be selected for next download?

- Random: Any of (A, B, C, D)
- Breadth-first: A
- In-degree: C
- PageRank: B

DNS Caching

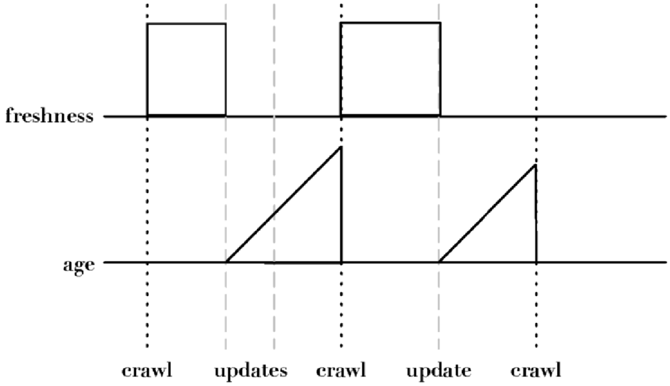
- Before a web page is crawled, the host name needs to be resolved to an IP address
- Since the same host name appears many times, DNS entries are locally cached by the crawler



Freshness vs. Age

- Webpage modification rate has a huge variations.
- Crawlers need to measure the rate at which each page changes.
- Two metrics:
    - **Freshness**: a page is *fresh* if the crawl has the most recent copy of a web page
    - **Age**: a page has age 0 until it is changed and then its age grows

Freshness vs Age



Freshness is binary, age is continuous.

- It's better to re-crawl pages when the age of the last crawled version exceeds some limit.
- The age of a page is the elapsed time since the first update after the most

Expected Page Age

Suppose we have a page with change frequency λ, (i.e., a page changes λ times in a one-day period)
The expected age of a page t days after it was crawled depends on its update probability:

$$age(\lambda, t) = \int_0^t P(\text{page changed at time x})(t - x)dx$$

On average, page updates follow a Poisson distribution – the time until the next update is governed by an exponential distribution. This makes the expected age:

$$age(\lambda, t) = \int_0^t \lambda e^{-\lambda x}(t - x)dx$$

Freshness vs Age

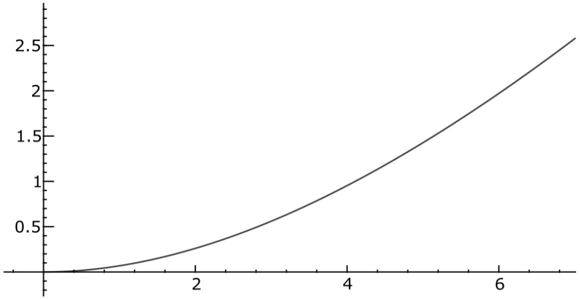


Fig. 3.7. Expected age of a page with mean change frequency $\lambda = 1/7$ (one week)

Politeness

- In general, we don’t want to make heavy demands on other’s web service resources.
- Some web content is considered private or under copyright.
- Politeness policies can mediate the conflicts between search providers and the web sites.

Our Web

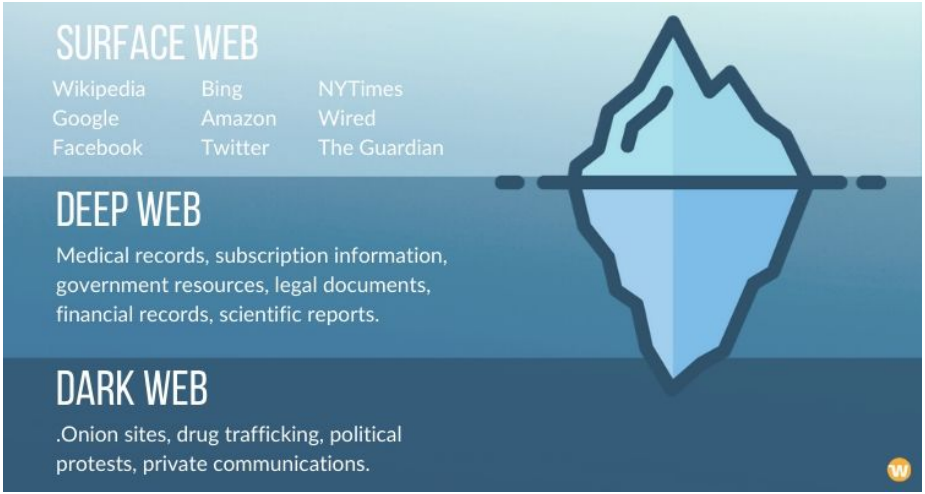

SURFACE WEB
Wikipedia
Google
Facebook
Bing
Amazon
Twitter
NYTimes
Wired
The Guardian
DEEP WEB
Medical records, subscription information, government resources, legal documents, financial records, scientific reports.
DARK WEB
.Onion sites, drug trafficking, political protests, private communications.

Figure: Web by indexability and accessibility.

Politeness Policy — Obey Robot Exclusion

- A standard from the early days of the Web
- The owner of a website can specify that robots should not crawl/index certain areas.
- **visible $\neq$ accessible $\neq$ storable**
- Always check before crawling the website

Robot Exclusion

Robot Exclusion Protocol

- Two components:
  - **Robots Exclusion Protocol**: Site wide specification of excluded directories. Specified in a file called ‘robots.txt’
  - **Robots META Tag**: Individual document tag to exclude indexing or following links

Robots.txt

'Robots.txt' is a file contains a list of excluded directories for given robot (user-agent)

Exclude all robots from the entire site:

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

Exclude specific directories:

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

Exclude a specific robot:

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

Politeness Policy — Robots.txt

Robots.txt

```
user-agent: googlebot        # all services
disallow: /private/          # disallow this directory

user-agent: googlebot-news   # only the news service
disallow: /                  # on everything

user-agent: *                # all robots
disallow: /something/        # on this directory

user-agent: *                # all robots
crawl-delay: 10              # wait at least 10 seconds

disallow: /directory1/       # disallow this directory
allow: /directory1/myfile.html # allow a subdirectory

host: www.example.com        # use this mirror
```

google robots.txt

Robots META Tag

- Include META tag in HEAD section of a specific HTML document.
  `<meta name="robots" content="noindex, nofollow">`
- 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

Robots META Tag

- 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">
```

Sitemaps

- In addition to robots.txt, which asks crawlers not to index certain content, a site can request that certain content be indexed by hosting a file at `/sitemap.xml`.
- Sitemaps can direct your crawler toward the most important content on the site, indicate when it has changed, etc.
- ex. Google's sitemaps

Consolidating Duplicate URLs

URL Canonicalization Process

- Many possible URLs ca refer to the same resource.
- Many URL normalization rules are available:
  - some can make mistakes.
  - the order of rules can also affect the results

http://example.com/some/../folder?id=1#anchor ⇒
http://example.com/some/../folder
⇒ http://example.com/folder

URL Syntax

- A URL has the following 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
  <ANAME="<fragment>">

URL Canonicalization Example

W3lib example (used in Scrapy package)

Example process: w3lib (used by the Python *Scrapy* package)

1. Sort query arguments, first by key, then by value
   https://example.com/item?id=1231&cat=service&cat=product
2. Percent encode paths; non-ASCII characters are percent-encoded using UTF-8 (RFC-3986)
   http://www.example.com/r\u00e9sum\u00e9
3. Percent encode query arguments; non-ASCII characters are percent-encoded using passed encoding (UTF-8 by default)
4. Normalize all spaces (in query arguments) '+' (plus symbol)
   https://example.com/item?id=1231&cat=service&cat=europeancafe

URL Canonicalization Example

Example process: w3lib (used by the Python *Scrapy* package)

5. Normalize percent encodings case (%2f → %2F)
   http://www.example.com/r%c3%a9sum%c3%a9
6. Remove query arguments with blank values
   https://example.com/item?id=1231&cat=service&cat=europeancafe&loc=
7. Remove fragments
   https://example.com/book.php#second_title

Keeping Similar or Duplicate Pages

- Multiple device types
  - https://example.com/news/koala-rampage
  - https://m.example.com/news/koala-rampage
- Dynamic URLs (e.g., search parameters, session IDs)
  - https://www.example.com/products?category=dresses&color=green
- Different naming schemes for blog posts
  - https://blog.example.com/dresses/green-dresses-are-awesome/
  - https://github.com/alexeygumirov/pandoc-beamer-how-to
- Different network protocols
  - http://example.com/green-dresses
  - https://example.com/green-dresses
  - http://www.example.com/green-dresses

Duplicate Detection

Why?

- More than 30% of pages are (near-) duplicate documents on the Web
- Copies, mirror sites, versioning, plagiarism, spam, etc.
- Detecting duplicate documents is important for conserving limited crawling and indexing resources
- Also, an essential tool in natural language processing

How?

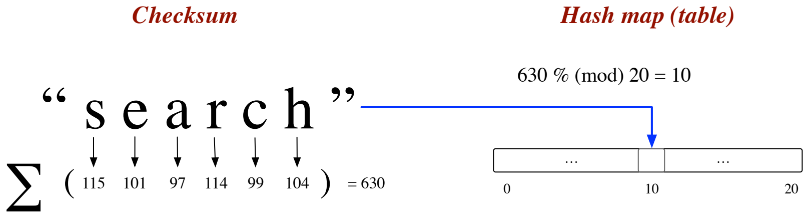
- For exact duplicates, use **checksumming** techniques
- For near-duplicates, use a threshold value for some similarity measure between pairs of documents (e.g., **fingerprints**)

Duplicate Detection

Checksums vs. Hash functions

- *Hash (Checksum) functions* are mathematical functions that maps variable lenth data into a fixed value.
- Same concept with different purposes
  - **Checksum** is to detect errors (or accidental changes) in data, often in a simple and fast manner.
  - **Hash** is to speed up comparison of data using fingerprints or create a hash table.
- ASCII parity bit is a 1-bit hash function (i.e., even/odd parity)

Hash functions (proof of concept)



Fingerprints

Process of generating fingerprints using n-grams

1. A document is converted into a sequence of words; all punctuation and formatting is removed.
2. Words are grouped into (possibly overlapping) n-grams, for some n.
3. A subset of the n-grams are selected to represent the document.
4. The selected n-grams are hashed, and the resulting hashes stored in an inverted index.
5. Documents are compared based on the number of overlapping n-grams.

Fingerprints (example)

(a) Original text

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

(b) 3-grams

tropical fish include, fish include fish, include fish found, fish found in, found in tropical, in tropical environments, tropical environments around, environments around the, around the world, the world including, world including both, including both freshwater, both freshwater and, freshwater and salt, and salt water, salt water species

(c) Hash values

938 664 463 822 492 798 78 969 143 236 913 908 694 553 870 779

(d) Selected hash values using $0 mod 4$

664 492 236 908

Summary

- Questions?
- Discussion?