Information Retrieval and Web Search

Cornelia Caragea

Computer Science
University of Illinois at Chicago

Credits for slides: Manning

Web Crawling

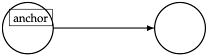
Required Reading

- "Information Retrieval" textbook
    - Chapter 19: Web Search Basics
    - Chapter 20: Web Crawling

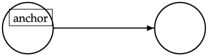
Web Challenges for IR

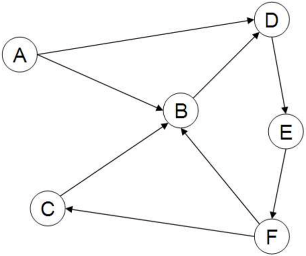
- **Distributed Data:** Documents spread over millions of different web servers.
- **Volatile Data:** Many documents change or disappear rapidly (e.g. dead links).
- **Large Volume:** Billions of separate documents.
- **Unstructured and Redundant Data:** No uniform structure, HTML errors, up to 40% (near) duplicate documents.
- **Quality of Data:** No editorial control, false information, poor quality writing, typos, etc.
- **Heterogeneous Data:** Multiple media types (images, video), languages, character sets, etc.

The Graph Structure of the Web

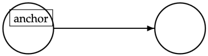


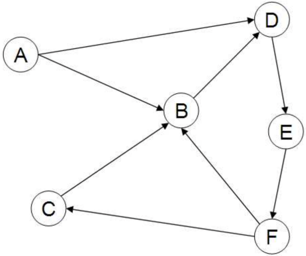
The Graph Structure of the Web



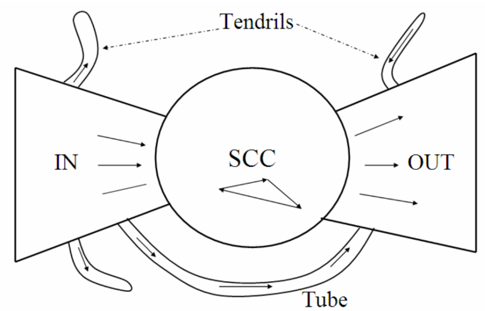


The Graph Structure of the Web





The Web Graph

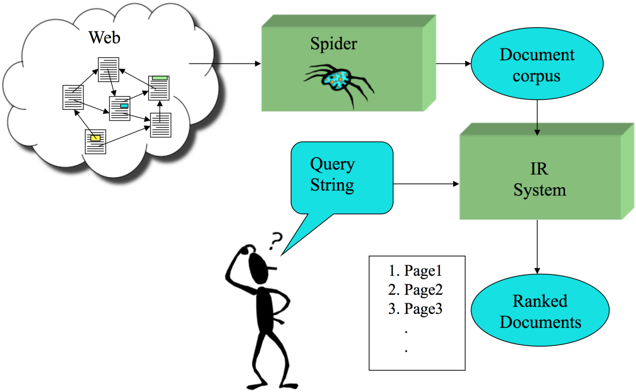


Zipf's Law on the Web

- Number of in-links/out-links to/from a page has a Zipfian distribution.
- Length of web pages has a Zipfian distribution.
- Number of hits to a web page has a Zipfian distribution.

Web Search Using IR

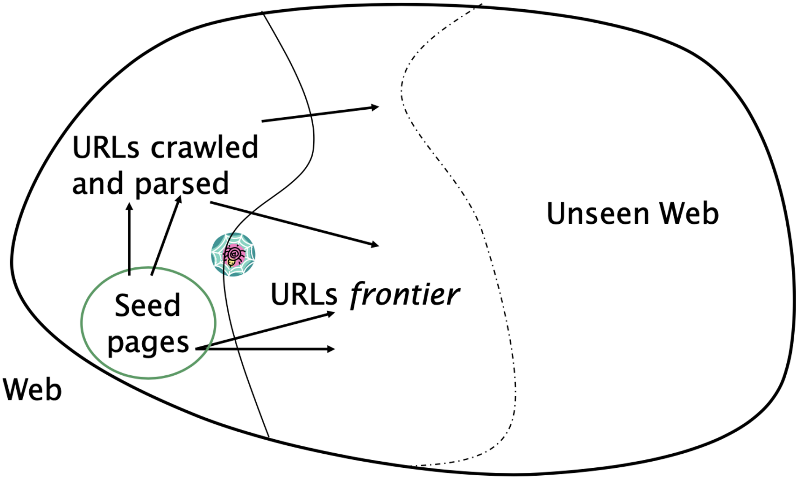
- The crawler (or spider) represents the main difference compared to traditional IR.



Crawlers (Robots/Bots/Spiders)

- Begin with known “seed” URLs
- Fetch and parse them
    - Extract URLs they point to
    - Place the extracted URLs on a queue
- Fetch each URL on the queue and repeat

Crawling Picture



Simple Picture - Complications

- Web crawling is not feasible with one machine
  - All of the above steps have to be distributed
- Malicious pages
  - Spam pages
  - Spider traps
- Even non-malicious pages pose challenges
  - Latency/bandwidth to remote servers vary
  - Site mirrors and duplicate pages

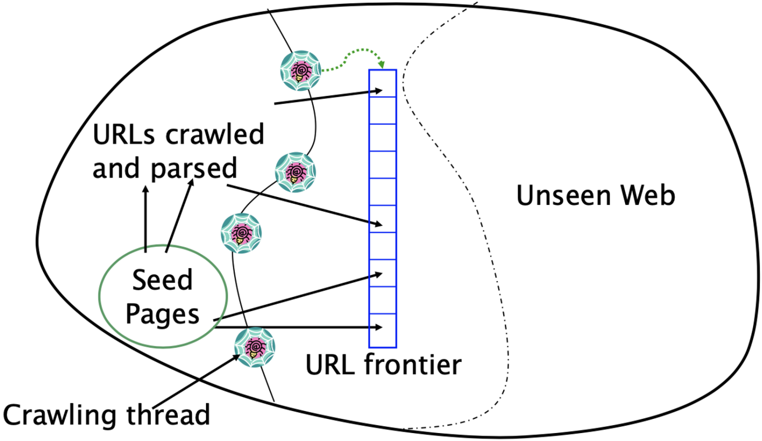
What any Crawler *Must* Do

- **Be Robust:** Be immune to spider traps and other malicious behavior from web servers
- **Be Polite:** Respect implicit and explicit politeness considerations
  - Only crawl allowed pages
  - Respect robot exclusion protocols

What any Crawler *Should* Do

- Be capable of distributed operation: designed to run on multiple distributed machines
- Be scalable: designed to increase the crawl rate by adding more machines
- Performance/efficiency: permit full use of available processing and network resources
- Fetch pages of "higher quality" first
- Continuous operation: Continue fetching fresh copies of a previously fetched page
- Extensible: Adapt to new data formats, protocols

Updated Crawling Picture



URL Frontier

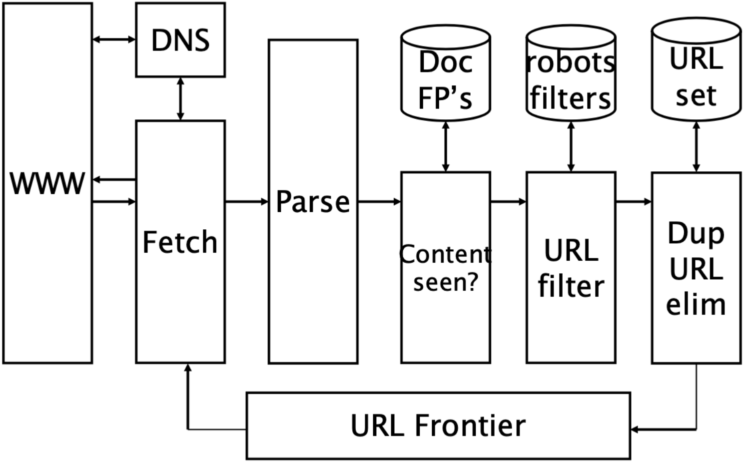
- Can include multiple pages from the same host
- **Must avoid trying to fetch them all at the same time**
- Must try to keep all crawling threads busy

- Implementation:
  - only one connection is open at a time to any host;
  - a waiting time of a few seconds occurs between successive requests to a host
  - high-priority pages are crawled preferentially.

Processing Steps in Crawling

- Pick a URL from the frontier
- Fetch the document at the URL
- Parse the URL
  - Extract links from it to other documents (URLs)
- Check if URL has content already seen
  - If not, add to indexes
- For each extracted URL
  - Ensure it passes certain URL filter tests
  - Check if it is already in the frontier (duplicate URL elimination)

Basic Crawl Architecture

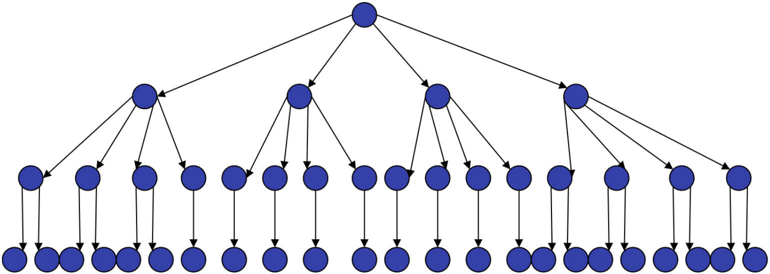


DNS (Domain Name Server)

- A lookup service on the internet
  - Given a URL, retrieve its IP address
  - Service provided by a distributed set of servers - thus, lookup latencies can be high (even seconds)
- Common OS implementations of DNS lookup are blocking: only one outstanding request at a time
  - Biggest bottleneck in Web crawling
- Solutions
  - DNS caching
  - Batch DNS resolver - collects requests and sends them out together

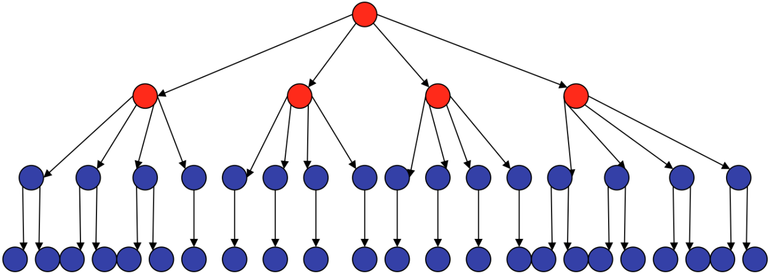
Parsing: Traversal Strategies

- The Web is a graph
  - Graph traversal strategies
- Breadth-first Search



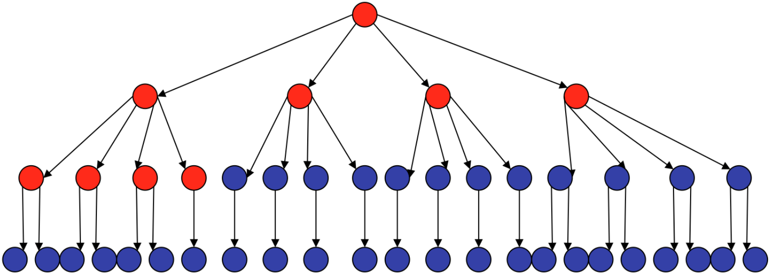
Parsing: Traversal Strategies

- The Web is a graph
  - Graph traversal strategies
- Breadth-first Search



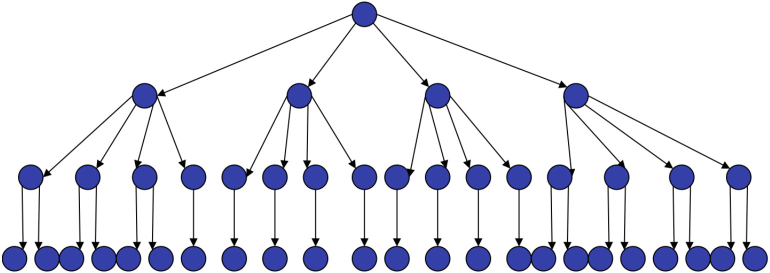
Parsing: Traversal Strategies

- The Web is a graph
  - Graph traversal strategies
- Breadth-first Search



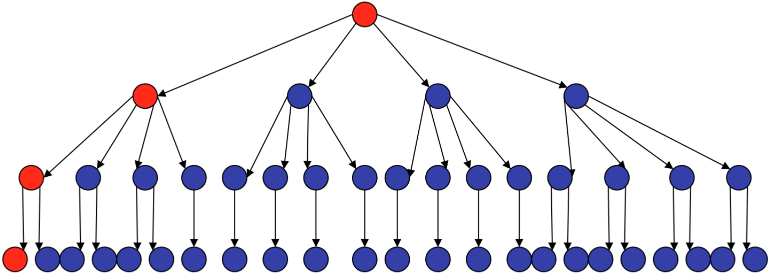
Parsing: Traversal Strategies

- The Web is a graph
  - Graph traversal strategies
- Depth-first Search



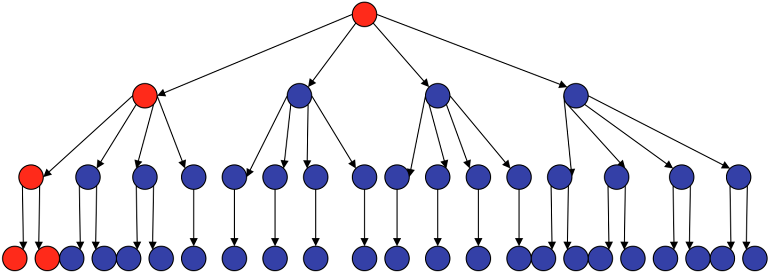
Parsing: Traversal Strategies

- The Web is a graph
  - Graph traversal strategies
- Depth-first Search



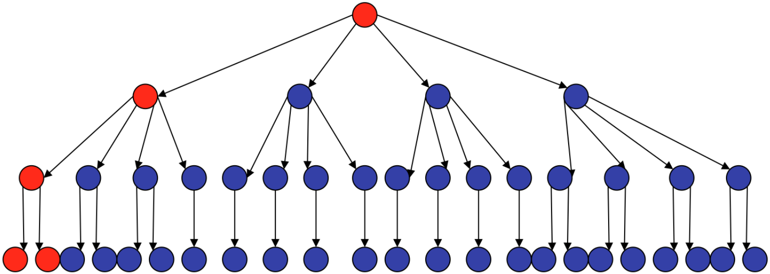
Parsing: Traversal Strategies

- The Web is a graph
  - Graph traversal strategies
- Depth-first Search



Parsing: Traversal Strategies

- The Web is a graph
  - Graph traversal strategies
- Depth-first Search



- Both strategies can be easily implemented using a queue of links (URLs).

Traversal Algorithm

- Initialize queue (Q) with initial set of known URLs.
- Until Q empty or page or time limit exhausted:
    - Pop URL, L, from front of Q.
    - If L is not to an HTML page (.gif, .jpeg, .ps, .pdf, .ppt, etc.)
        - 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 if the content not seen (e.g., add to inverted index or store cached copy).
    - Parse P to obtain a list of new links N.
    - Append N to the end of Q.

Queueing Strategy

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

Directed/Focused Crawling

- Sort the queue to explore more "interesting" pages first.
- Two styles of focus:
  - Topic-Directed
  - Link-Directed

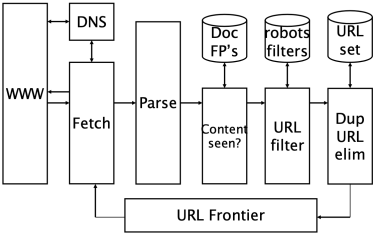
Topic-Directed Crawling

- Assume that a desired topic description or sample pages of interest are given.
- Sort the queue of links by the similarity (e.g. cosine metric) of their source pages and/or anchor text to this topic description.

Link-Directed Crawling

- Monitor links and keep track of in-degree and out-degree of each page encountered.
- Sort queue to prefer popular pages with many in-coming links (authorities).
- Sort queue to prefer summary pages with many out-going links (hubs).

URL Normalization/Duplicate Elimination



- When a fetched document is parsed, some of the extracted links are relative URLs.
  - E.g., http://en.wikipedia.org/wiki/Main_Page has a relative link to /wiki/Wikipedia:General_disclaimer which is the same as the absolute URL http://en.wikipedia.org/wiki/Wikipedia:General_disclaimer.
- During parsing, must normalize (expand) such relative URLs.
- Other normalization: remove ending:
  - https://www.cs.uic.edu/∼cornelia/
    https://www.cs.uic.edu/∼cornelia

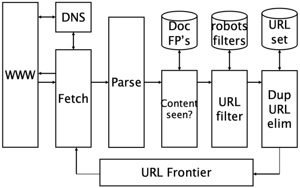
Anchor Text Indexing

- Extract anchor text (between <a> and) 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.
- Used by Google:
  - IBM

Anchor Text Indexing

- Helps when descriptive text in destination page is embedded in image logos rather than in accessible text.
- Many times anchor text is not useful:
  - "click here"
- Increases content more for popular pages with many in-coming links, increasing recall of these pages.
- A system may even give higher weights to tokens from anchor text.

URL Filters: Restricting Crawling



- You can restrict crawler to a particular site.
  - Remove links to other sites from Q.
- You can restrict crawler to a particular directory.
  - Remove links not in the specified directory.
- Obey page-owner restrictions (robot exclusion): Crawler politeness
  - Explicit politeness: specifications from webmasters on what portions of site can be crawled (e.g., robots.txt).
  - Implicit politeness: even with no specification, avoid hitting any site too often.

Robot Exclusion

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

Robot Exclusion Protocol

- Site administrator puts a “robots.txt” file at the root of the host’s web directory.
  - 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:`

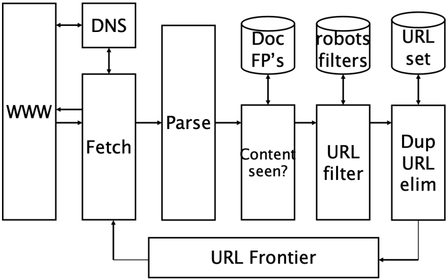
http://www.cnn.com/robots.txt

```
Sitemap: https://www.cnn.com/sitemaps/cnn/index.xml
Sitemap: https://www.cnn.com/sitemaps/cnn/news.xml
Sitemap: https://www.cnn.com/sitemaps/sitemap-section.xml
Sitemap: https://www.cnn.com/sitemaps/sitemap-interactive.xml
Sitemap: https://www.cnn.com/ampstories/sitemap.xml
Sitemap: https://edition.cnn.com/sitemaps/news.xml
Sitemap: https://www.cnn.com/sitemap/article/cnn-underscored.xml
User-agent: *
Allow: /partners/ipad/live-video.json
Disallow: /*.jsx$
Disallow: *.jsx$
Disallow: /*.jsx/
Disallow: *.jsx?
Disallow: /ads/
Disallow: /aol/
Disallow: /beta/
Disallow: /browsers/
Disallow: /cl/
Disallow: /cnews/
Disallow: /cnn_adspaces
Disallow: /cnnbeta/
Disallow: /cnnintl_adspaces
Disallow: /development
Disallow: /editionssi
Disallow: /help/cnnx.html
Disallow: /NewsPass
Disallow: /NOKIA
Disallow: /partners/
Disallow: /pipeline/
Disallow: /pointroll/
Disallow: /POLLSERVER/
Disallow: /pr/
Disallow: /privacy
Disallow: /PV/
Disallow: /Quickcast/
Disallow: /quickcast/
Disallow: /QUICKNEWS/
Disallow: /search/
Disallow: /terms
Disallow: /test/
Disallow: /virtual/
Disallow: /WEB-INF/
Disallow: /web.projects/
Disallow: /webview/
```

Robots META Tag

- Include META tag in the 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=“none”>
- META tag is newer and less well-adopted than “robots.txt.”

Avoiding Page Duplication



- The web is full of duplicated content
- Strict duplicate detection = exact match
  - Not as common
- But many, many cases of near duplicates
  - E.g., Last modified date the only difference between two copies of a page

Duplicate/Near-Duplicate Detection

- *Duplication:* Exact match can be detected with fingerprints
- *Near-Duplication:* Approximate match
  - Compute syntactic similarity with an edit-distance measure
  - Use similarity threshold to detect near-duplicates
    - E.g., Similarity $>$ 80% => Documents are "near duplicates"

Computing Document Similarity

- Features:
  - Segments of a document
  - Shingles (Word N-Grams)

a rose is a rose is a rose → 4-grams are

a_rose_is_a

rose_is_a_rose

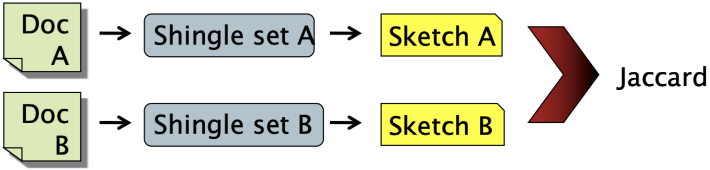
is_a_rose_is

a_rose_is_a

- Similarity Measure between two docs (= two sets of shingles)
  - Compare using set operations: (Size_of_Intersection / Size_of_Union) (Jaccard)

Shingles + Set Intersection

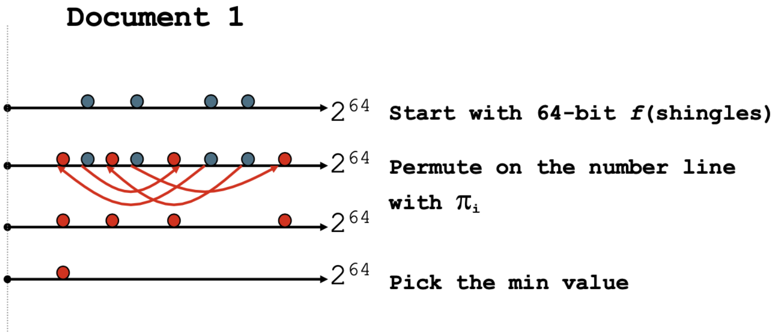
- Computing exact set intersection of shingles between all pairs of documents is expensive
- Approximate using a cleverly chosen subset of shingles from each (a sketch)
- Estimate (size_of_intersection / size_of_union) based on a short sketch



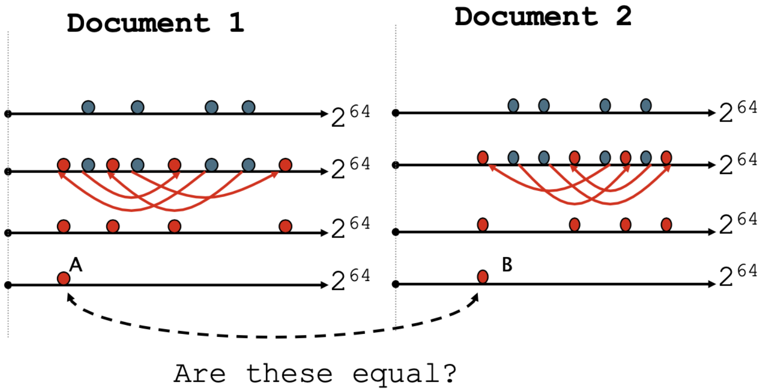
Sketch of a Document

- Create a "sketch vector" (of size 200) for each document
  - Documents that share $\geq$ t (say 80%) corresponding vector elements are deemed near duplicates
  - For document D, sketch$_D[i]$ is as follows:
    - Let f map all shingles in the universe to $1...2^m$ (e.g., f = fingerprinting)
    - Let π_i be a random permutation on $1...2^m$
    - Pick $MIN\{\pi(f(s))\}$ over all shingles s in D

Computing Sketch[i] for Doc1

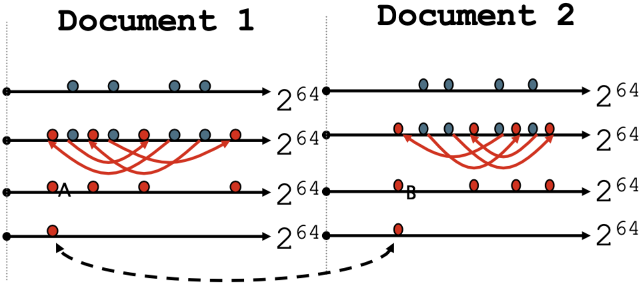


Test if Doc1.Sketch[i] = Doc2.Sketch[i]



Test for 200 random permutations: $\pi_1, \pi_2, \ldots \pi_{200}$

However...



- A = B iff the shingle with the MIN value in the union of Doc1 and Doc2 is common to both (i.e., lies in the intersection)
- Claim: This happens with probability Size_of_intersection / Size_of_union

Example

π	Documents			
2	1	0	1	1
3	0	0	1	1
5	0	1	1	0
4	0	0	1	1
1	0	0	1	0
7	1	1	1	0
6	1	1	1	0

Shingles

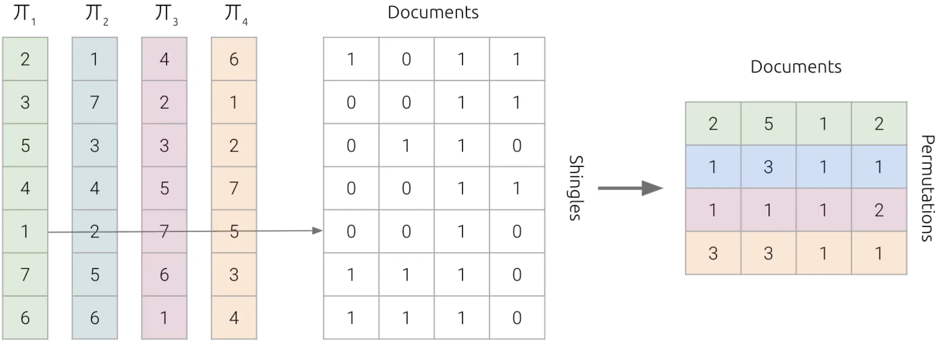
π	Documents			
2	1	0	1	1
3	0	0	1	1
5	0	1	1	0
4	0	0	1	1
1	0	0	1	0
7	1	1	1	0
6	1	1	1	0

Shingles

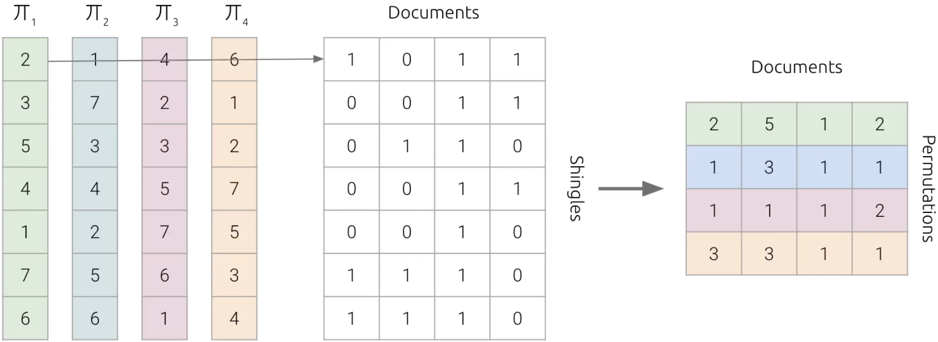
π	Documents			
2	1	0	1	1
3	0	0	1	1
5	0	1	1	0
4	0	0	1	1
1	0	0	1	0
7	1	1	1	0
6	1	1	1	0

Shingles

Example



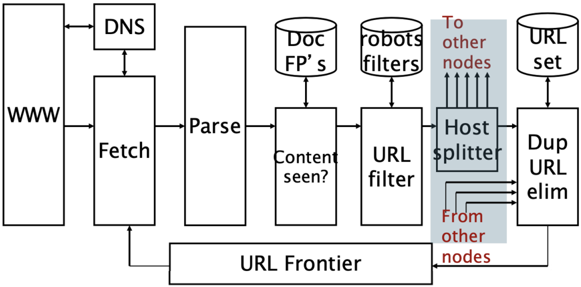
Example



MinHash Property: the similarity of the derived signatures is the fraction of the rows that are the same amongst two columns. E.g., similarity between D1 and D3 is 2/4 because half the items match between both columns.

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 URLs to threads to guarantee equitable distribution of requests across different hosts to maximize throughput and avoid overloading any single server.



Keeping Crawled Pages Up to Date

- Web is very dynamic: many new pages, updated pages, deleted pages, etc.
- Periodically check crawled pages for updates and deletions:
  - Just look at header info (e.g. META tags on last update) to determine if a page has changed
  - Only reload the entire page if needed.
- 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 freshness of more popular pages.