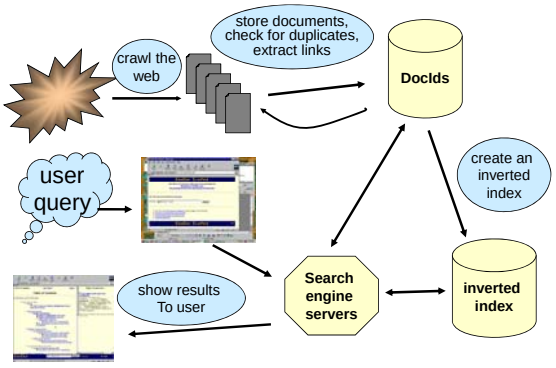
Crawling HTML

Standard Web Search Engine Architecture



Search Engine Architecture

- Crawler (Spider)
  - Searches the web to find pages. Follows hyperlinks. Never stops
- Indexer
  - Produces data structures for fast searching of all words in the pages
- Retriever
  - Query interface
  - Database lookup to find hits
    - 300 million documents
    - 300 GB RAM, terabytes of disk
  - Ranking, summaries
- Front End

CRAWLERS
(aka Spiders, Bots)…

Crawlers

- 1000s of spiders out there…
- Various purposes:
  - Search engines
  - Digital rights management
  - Advertising
  - Spam harvesters
  - Link checking – site validation

Open-Source Crawlers

- GNU Wget
  - Utility for downloading files from the Web.
  - Fine if you just need to fetch files from 2-3 sites.
- Heritix
  - Open-source, extensible, Web-scale crawler
  - Easy to get running.
  - Web-based UI
- Nutch
  - Featureful, industrial strength, Web search package.
  - Includes Lucene information retrieval part
    - TF/IDF and other document ranking
    - Optimized, inverted-index data store
  - You get complete control thru easy programming.

Danger Will Robinson!!

- Consequences of a bug

Max 6 hits/server/minute
plus….
http://www.cs.washington.edu/lab/policies/crawlers.html

Robot Exclusion

- Person may not want certain pages indexed.
- Crawlers should obey Robot Exclusion Protocol.
  - But some don't
- Look for file robots.txt at highest directory level
  - If domain is www.ecom.cmu.edu, robots.txt goes in www.ecom.cmu.edu/robots.txt
- Specific document can be shielded from a crawler by adding the line:

<META NAME="ROBOTS" CONTENT="NOINDEX">

Robots Exclusion Protocol

- Format of robots.txt
  - Two fields. User-agent to specify a robot
  - Disallow to tell the agent what to ignore
- To exclude all robots from a server:

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

- To exclude one robot from two directories:

```
User-agent: WebCrawler
Disallow: /news/
Disallow: /tmp/
```

- View the robots.txt specification at
  http://info.webcrawler.com/mak/projects/robots/norobots.html

Danger, Danger

- **Ensure that your crawler obeys robots.txt**
- **Be sure to:**
  - **Notify the CS Lab Staff**
  - **Provide contact info in user-agent field.**
  - **Monitor the email address**
  - **Honor all Do Not Scan requests**
  - **Post all "stop-scanning" requests to classmates**
  - **"The scanee is *always* right."**
  - **Max 6 hits/server/minute**

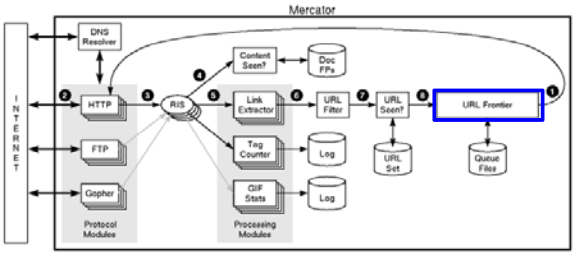
Crawling

- Queue := initial page URL_0
- Do forever
  - Dequeue URL
  - Fetch P
  - Parse P for more URLs; add them to queue
  - Pass P to (specialized?) indexing program
- Issues…
  - Which page to look at next?
    - keywords, recency, focus, ???
  - Politeness: Avoiding overload, scripts
  - Parsing links
  - How deep within a site to go?
  - How frequently to visit pages?
  - Traps!

Web Crawling Strategy

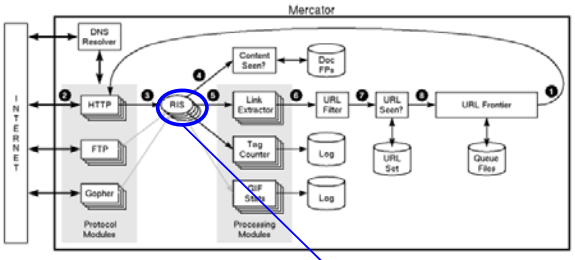
- Starting location(s)
- Traversal order
  - Depth first (LIFO)
  - Breadth first (FIFO)
  - Or ???
- Politeness
- Cycles?
- Coverage?

Structure of Mercator Spider



1. Remove URL from queue
2. Simulate network protocols & REP

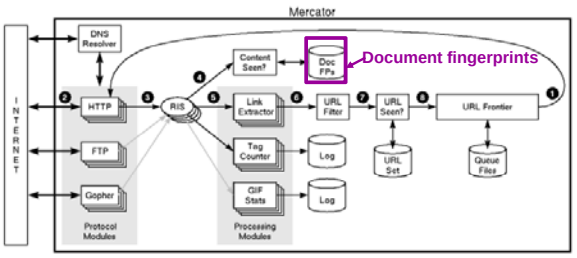
Structure of Mercator Spider



1. Remove URL from queue
2. Simulate network protocols & REP
3. **Read w/ RewindInputStream (RIS)**

I/O abstraction
Allows multiple reads
Own thread

Structure of Mercator Spider

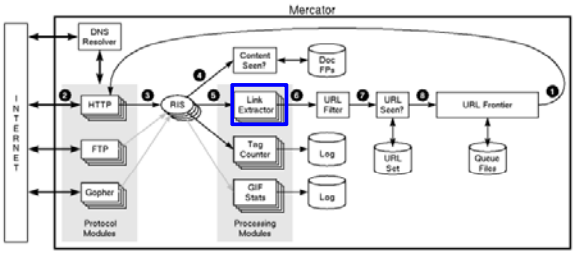


1. Remove URL from queue
2. Simulate network protocols & REP
3. Read w/ RewindInputStream (RIS)
4. **Has document been seen before? (checksums and fingerprints)**

Duplicate Detection

- URL-seen test: has URL been seen before?
  - To save space, store a hash
- Content-seen test: different URL, same doc.
  - Supress link extraction from mirrored pages.
- What to save for each doc?
  - 64 bit “document fingerprint”
  - Minimize number of disk reads upon retrieval.

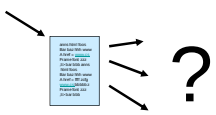
Structure of Mercator Spider



1. Remove URL from queue
2. Simulate network protocols & REP
3. Read w/ RewindInputStream (RIS)
4. Has document been seen before? (checksums and fingerprints)
5. **Extract links**

Outgoing Links?

- Parse HTML…
- Looking for…what?



Which tags / attributes hold URLs?

Anchor tag: …

Option tag: <option value="URL"…> … </option>

Map: <area href="URL" …>

Frame: <frame src="URL" …>

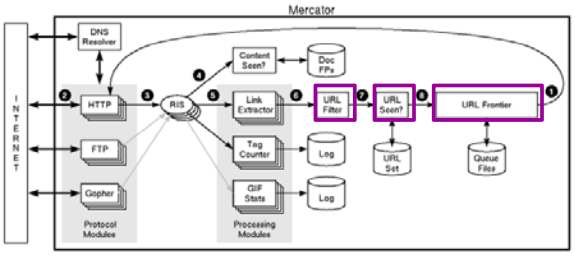
Link to an image: <img src="URL" …>

Relative path *vs.* absolute path: <base href= …>

Bonus problem: Javascript

In our favor: Search Engine Optimization

Structure of Mercator Spider



1. Remove URL from queue
2. Simulate network protocols & REP
3. Read w/ RewindInputStream (RIS)
4. Has document been seen before? (checksums and fingerprints)
5. Extract links
6. **Download new URL?**
7. **Has URL been seen before?**
8. **Add URL to frontier**

URL Frontier (priority queue)

- Most crawlers do breadth-first search from seeds.
- Politeness constraint: don't hammer servers!
  - Obvious implementation: "live host table"
  - Will it fit in memory?
  - Is this efficient?
- Mercator's politeness:
  - One FIFO subqueue per thread.
  - Choose subqueue by hashing host's name.
  - Dequeue first URL whose host has NO outstanding requests.

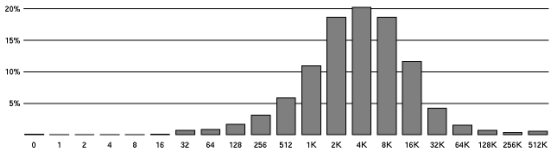
Fetching Pages

- Need to support http, ftp, gopher, ...
  - Extensible!
- Need to fetch multiple pages at once.
- Need to cache as much as possible
  - DNS
  - robots.txt
  - Documents themselves (for later processing)
- Need to be defensive!
  - Need to time out http connections.
  - Watch for "crawler traps" (e.g., infinite URL names.)
  - See section 5 of Mercator paper.
  - Use URL filter module
  - Checkpointing!

Nutch: A simple architecture

- Seed set
- Crawl
- Remove duplicates
- Extract URLs (minus those we've been to)
  - new frontier
- Crawl again
- Can do this with Map/Reduce architecture

Mercator Statistics



Exponentially increasing size

PAGE TYPE	PERCENT
text/html	69.2%
image/gif	17.9%
image/jpeg	8.1%
text/plain	1.5
pdf	0.9%
audio	0.4%
zip	0.4%
postscript	0.3%
other	1.4%

Outdated (1999)

Advanced Crawling Issues

- Limited resources
  - Fetch most ***important*** pages first
- Topic specific search engines
  - Only care about pages which are ***relevant*** to topic

"Focused crawling"

- Minimize stale pages
  - Efficient re-fetch to keep index timely
  - How track the rate of change for pages?

Focused Crawling

- Priority queue instead of FIFO.
  -
- How to determine priority?
  - Similarity of page to driving query
    - Use traditional IR measures
    - Exploration / exploitation problem
  - Backlink
    - How many links point to this page?
  - PageRank (Google)
    - Some links to this page count more than others
  - Forward link of a page
  - Location Heuristics
    - E.g., Is site in .edu?
    - E.g., Does URL contain 'home' in it?
  - Linear combination of above