CSCI 5417
Information Retrieval Systems

Jim Martin

Lecture 18
10/27/2011

Today

- Start on web search

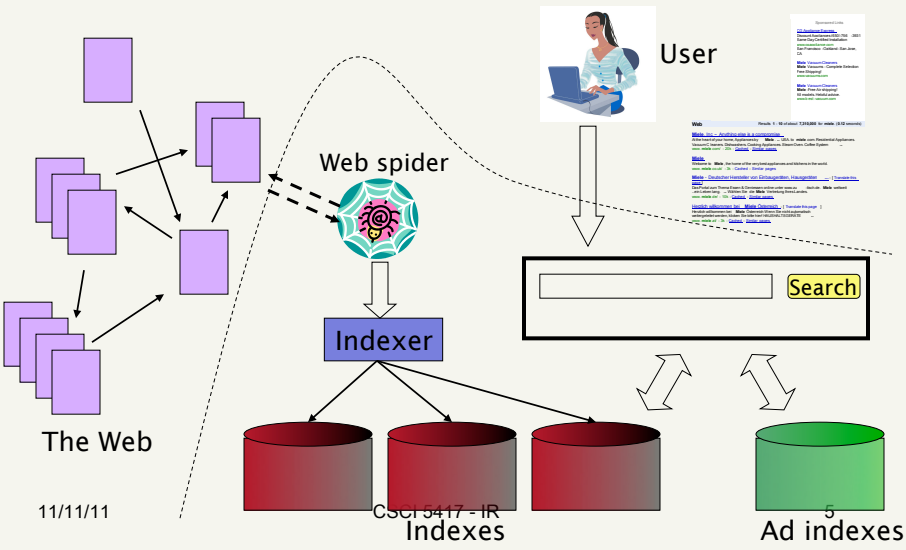
Brief History of Web Search

- Early keyword-based engines
  - Altavista, Excite, Infoseek, Inktomi, Lycos ca. 1995-1997
- Sponsored search ranking:
  - WWWW (1994) (Colorado/McBryan) -> Goto.com (morphed into Overture.com → Yahoo! → ???)
  - Your search ranking depended on how much you paid
  - Auction for keywords: ***casino*** was an expensive keyword!

Brief history

- 1998+: Link-based ranking introduced by Google
  - Perception was that it represented a fundamental improvement over existing systems
  - Great user experience in search of a business model
  - Meanwhile Goto/Overture's annual revenues were nearing $1 billion
- Google adds paid-placement "ads" to the side, distinct from search results
  - 2003: Yahoo follows suit
    - acquires Overture (for paid placement)
    - and Inktomi (for search)

Web search basics



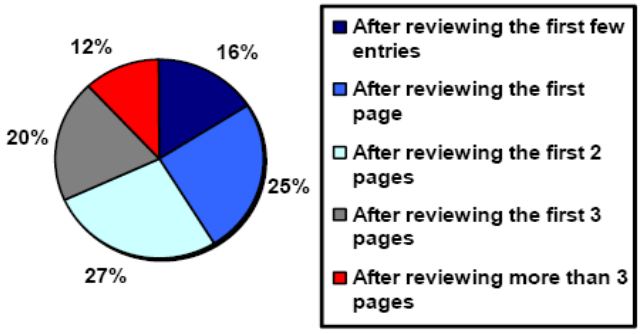
Sec. 19.4.1

User Needs

- Need [Brod02, RL04]
  - **Informational** – want to learn about something (~40% / 65%)
    `Low hemoglobin`
  - **Navigational** – want to go to that page (~25% / 15%)
    `United Airlines`
  - **Transactional** – want to do something (web-mediated) (~35% / 20%)
    - Access a service
      `Seattle weather`
    - Downloads
      `Mars surface images`
    - Shop
      `Canon S410`
  - **Gray areas**
    - Find a good hub
      `Car rental Brazil`
    - Exploratory search "see what's there"

How far do people look for results?

"When you perform a search on a search engine and don't find what you are looking for, at what point do you typically either revise your search, or move on to another search engine? (Select one)"



(Source: iprospect.com WhitePaper_2006_SearchEngineUserBehavior.pdf)

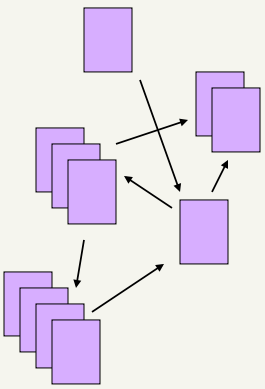
Users' empirical evaluation of results

- Quality of pages varies widely
  - Relevance is not enough
  - Other desirable qualities
    - Content: Trustworthy, diverse, non-duplicated, well maintained
    - Web readability: display correctly & fast
    - No annoyances: pop-ups, etc
- Precision vs. recall
  - On the web, recall seldom matters
- What matters
  - Precision at 1? Precision at k?
  - Comprehensiveness – must be able to deal with obscure queries
    - Recall matters when the number of matches is very small

Users' empirical evaluation of engines

- Relevance and validity of results
- UI – Simple, no clutter, error tolerant
- Trust – Results are objective
- Coverage of topics for polysemic queries
- Pre/Post process tools provided
  - Mitigate user errors (auto spell check, search assist,…)
  - Explicit: Search within results, more like this, refine ...
  - Anticipative: related searches, suggest, instant search
- Deal with idiosyncrasies
  - Web specific vocabulary
    - Impact on stemming, spell-check, etc
  - Web addresses typed in the search box

The Web as a Document Collection



The Web

- No design/co-ordination
- Distributed content creation, linking, democratization of publishing
- Content includes truth, lies, obsolete information, contradictions …
- Unstructured (text, html, …), semi-structured (XML, annotated photos), structured (Databases)…
- Scale much larger than previous text collections … but corporate records are catching up
- Growth – slowed down from initial "volume doubling every few months" but still expanding
- Content can be *dynamically generated*

Web search engine pieces

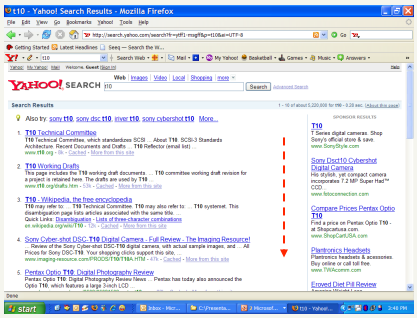
- Spider (a.k.a. crawler/robot) – builds corpus
  - Collects web pages recursively
    - For each known URL, fetch the page, parse it, and extract new URLs
    - Repeat
  - Additional pages from direct submissions & other sources
- The indexer – creates inverted indexes
  - Usual issues wrt which words are indexed, capitalization, support for Unicode, stemming, support for phrases, language issues, etc.
- Query processor – serves query results
  - Front end – query reformulation, word stemming, capitalization, optimization of Booleans, phrases, wildcards, spelling, etc.
  - Back end – finds matching documents and ranks them

Search Engine: Three sub-problems

1. Match ads to query/context
2. Generate and Order the ads
3. Pricing on a click-through

IR

Econ



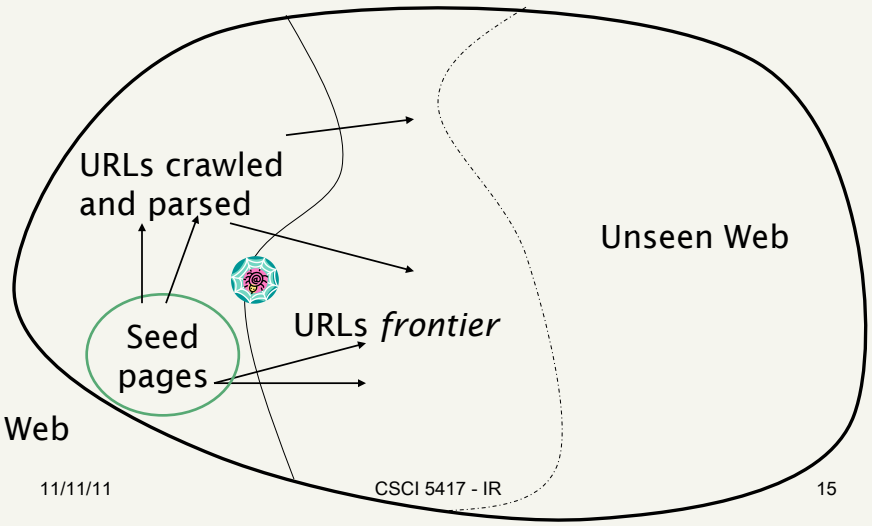
The trouble with search ads…

- They cost real money.
- *Search Engine Optimization:*
  - "Tuning" your web page to rank highly in the search results for select keywords
  - Alternative to paying for placement
  - Thus, intrinsically a marketing function
- Performed by companies, webmasters and consultants ("Search engine optimizers") for their clients
- Some perfectly legitimate, some very shady

Basic crawler operation

- Begin with known "seed" pages
- 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

- Effective Web crawling isn't feasible with one machine
  - All of the above steps need to be distributed
- Even non-malicious pages pose challenges
  - Latency/bandwidth to remote servers vary
  - Webmasters' stipulations
    - How "deep" should you crawl a site's URL hierarchy?
  - Site mirrors and duplicate pages
- Malicious pages
  - Spam pages
  - Spider traps – incl dynamically generated
- Politeness – don't hit a server too often

What any crawler *must* do

- Be Polite: Respect implicit and explicit politeness considerations for a website
  - Only crawl pages you're allowed to
  - Respect *robots.txt*
- Be Robust: Be immune to spider traps and other malicious behavior from web servers

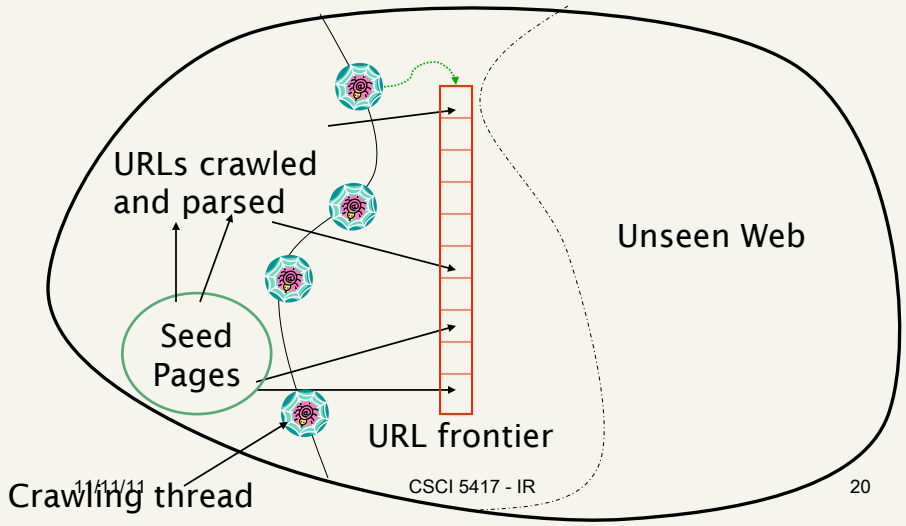
What any crawler *should* do

- 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

What any crawler *should* do

- Fetch important stuff first
  - Pages with "higher quality"
- Continuous operation: Continue to fetch fresh copies of a previously fetched page
- Extensible: Adapt to new data formats, protocols, etc.

Updated crawling picture



Break

- HW 3
  - Currently the best F1 scores are
    - .3988, .3955, .3918
    - Lots of folks bunched between .2 and .25
    - Some lower
  - Scoring issue
    - Some folks didn't assign tags to all the docs
    - Makes computing an average F1 score problematic
      - What should the denominator be?

Break

- Come to today's colloquium

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

Explicit and implicit politeness

- Explicit politeness: specifications from webmasters on what portions of site can be crawled
  - robots.txt
- Implicit politeness: even with no specification, avoid hitting any site too often

Robots.txt

- Protocol for giving spiders ("robots") limited access to a website, originally from 1994
- Website announces its request on what can(not) be crawled
  - For a URL, create a file `URL/robots.txt`
  - This file specifies access restrictions

Robots.txt example

- No robot should visit any URL starting with "/yoursite/temp/", except the robot called "searchengine":

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

User-agent: searchengine
Disallow:
```

A Web Document: Three views

How it relates to other web texts
- Links to it
  - And the anchor texts
- What it links to

The document itself
- Content
- Language
- Structure

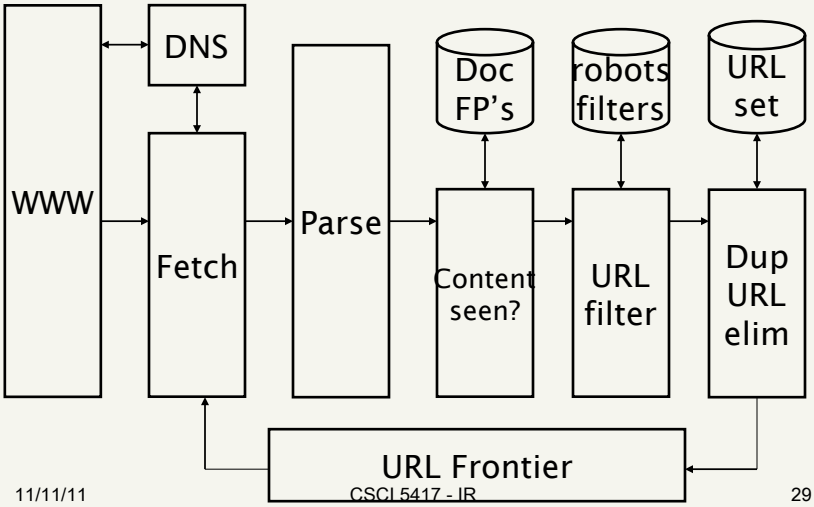
How it relates to your current index
- Is it already there
- Is the content already there
- Is it the kind of stuff you care about

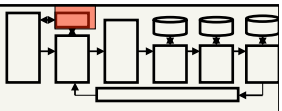
Processing steps in crawling

- Pick a URL from the frontier ← Which one?
- Fetch the document at the URL
- Parse the document
  - Extract links from it to other docs (URLs)
- Check if document 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)

E.g., only crawl .edu, obey robots.txt, etc.

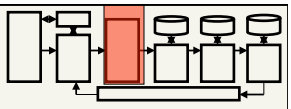
Basic crawl architecture





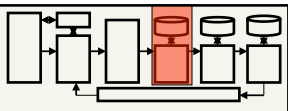
DNS (Domain Name Server)

- 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 implementations of DNS lookup are *blocking*: only one outstanding request at a time
- Solutions
  - DNS caching
  - Batch DNS resolver – collects requests and sends them out together



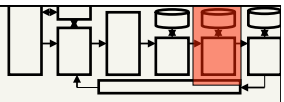
Parsing: URL normalization

- When a fetched document is parsed, some of the extracted links are *relative* URLs
  - en.wikipedia.org/wiki/Main_Page has a relative link to /wiki/Wikipedia:General_disclaimer which is the same as the absolute URL en.wikipedia.org/wiki Wikipedia:General_disclaimer
  - Must expand such relative URLs
- URL shorteners (bit.ly, etc) are a new problem



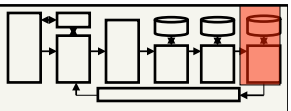
Content seen?

- Duplication is widespread on the web
- If the page just fetched is already in the index, do not further process it
- This is verified using document fingerprints or shingles



Filters and robots.txt

- Filters – regular expressions for URL's to be crawled/not
- Once a robots.txt file is fetched from a site, need not fetch it repeatedly
  - Doing so burns bandwidth, hits web server
- Cache robots.txt files



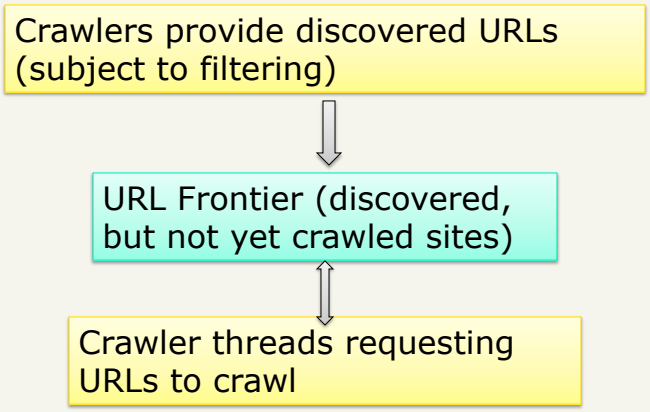
Duplicate URL elimination

- Check to see if an extracted+filtered URL has already been passed to the frontier

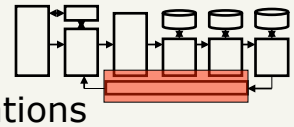
Distributing the crawler

- Run multiple crawl threads, under different processes – potentially at different nodes
  - Geographically distributed nodes
- Partition hosts being crawled into nodes

Overview: Frontier



URL frontier: two main considerations



- Politeness: do not hit a web server too frequently
- Freshness: crawl some pages more often than others
  - E.g., pages (such as News sites) whose content changes often

These goals may conflict each other.

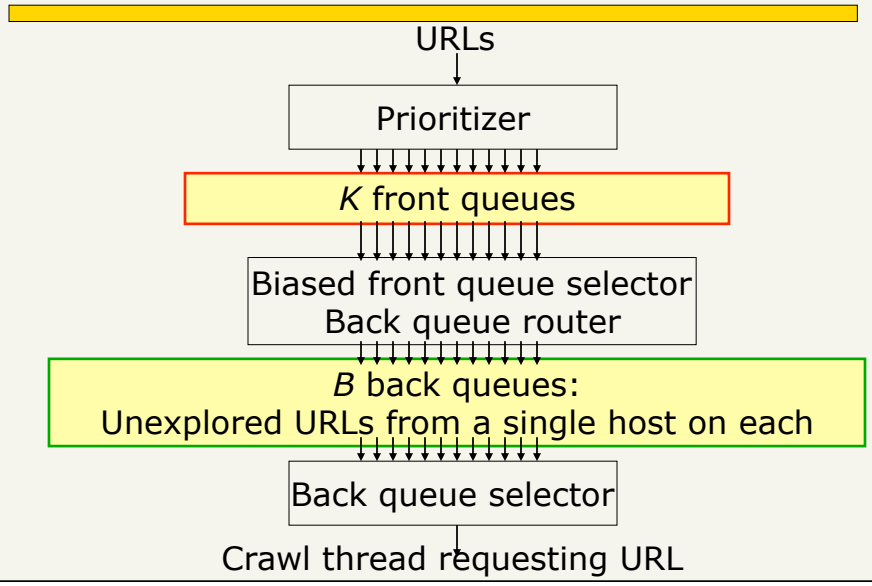
(E.g., simple priority queue fails – many links out of a page go to its own site, creating a burst of accesses to that site.)

Politeness – challenges

- Even if we restrict only one thread to fetch from a host, can hit it repeatedly
- Common heuristic: insert time gap between successive requests to a host that is >> time for most recent fetch from that host

Sec. 20.2.3

URL Frontier: Mercator scheme



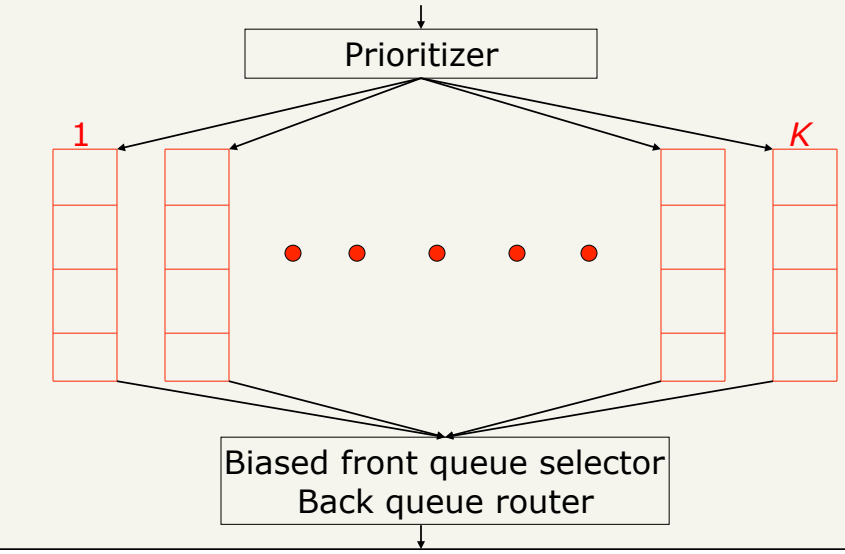
Sec. 20.2.3

Mercator URL frontier

- URLs flow in from the top into the frontier
- Front queues manage prioritization
- Back queues enforce politeness
- Each queue is FIFO

Sec. 20.2.3

Front queues



Sec. 20.2.3

Front queues

- Prioritizer assigns each URL an integer priority between 1 and *K*
  - Appends URL to corresponding queue
- Heuristics for assigning priority
  - Refresh rate sampled from previous crawls
  - Application-specific (e.g., "crawl news sites more often")

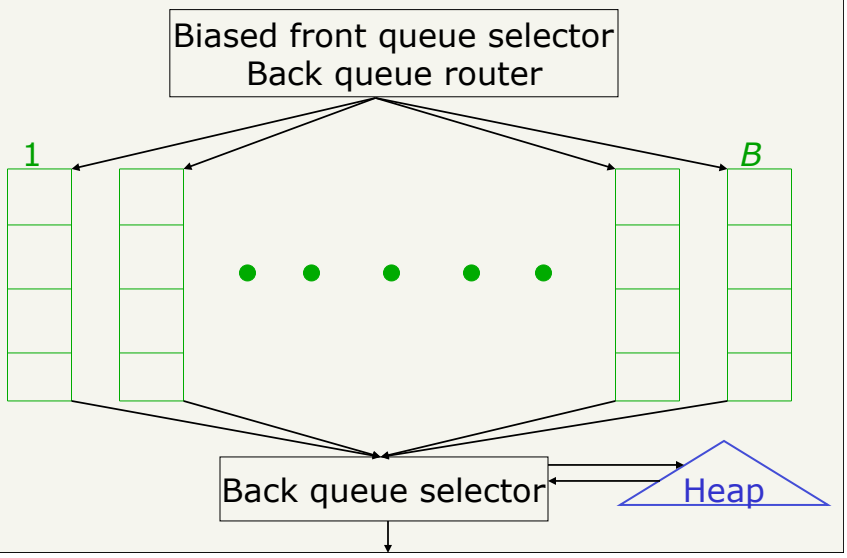
Sec. 20.2.3

Biased front queue selector

- When a back queue requests URLs (in a sequence to be described): picks a front queue from which to pull a URL
- This choice can be round robin biased to queues of higher priority, or some more sophisticated variant

Sec. 20.2.3

Back queues



Sec. 20.2.3

Back queue invariants

- Each back queue is kept non-empty while the crawl is in progress
- Each back queue only contains URLs from a single host
  - Maintain a table from hosts to back queues

Sec. 20.2.3

Back queue heap

- One entry for each back queue
- The entry is the earliest time t_e at which the host corresponding to the back queue can be hit again
- This earliest time is determined from
  - Last access to that host
  - Any time heuristic we choose

Back queue processing

- A crawler thread seeking a URL to crawl:
- Extracts the root of the heap
- Fetches URL at head of corresponding back queue *q* (look up from table)
- Checks if queue *q* is now empty – if so, pulls a URL *v* from front queues
  - If there's already a back queue for *v*'s host, append *v* to *q* and pull another URL from front queues, repeat
  - Else add *v* to *q*
- When *q* is non-empty, create heap entry for it

Number of back queues *B*

- Keep all threads busy while respecting politeness
- Mercator recommendation: three times as many back queues as crawler threads