CSCI 5417
Information Retrieval Systems

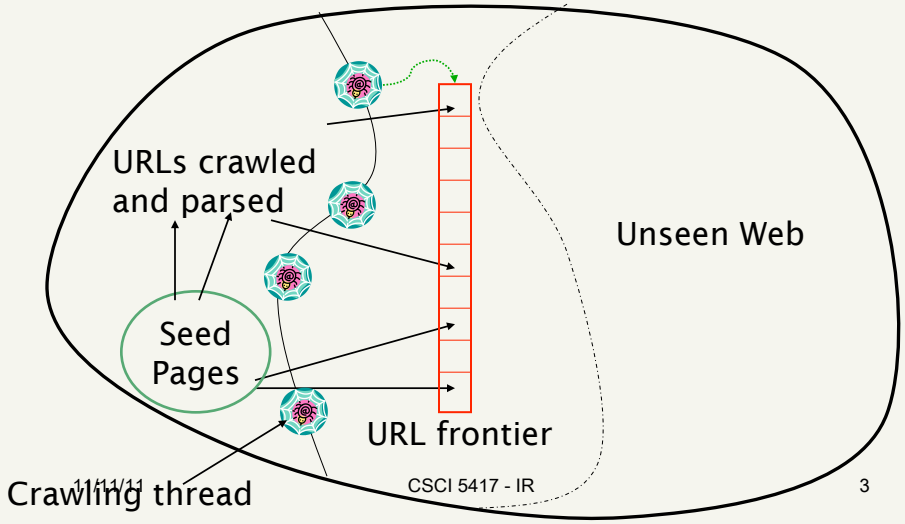
Jim Martin

Lecture 19
11/1/2011

Today

- Crawling
- Start on link-based ranking

Updated crawling picture



URL frontier

- URL *frontier contains URLs that have been* discovered but have not yet been explored (retrieved and analyzed for content and more URLs)
  - Can include multiple URLs from the same host
  - Must avoid trying to fetch them all at the same time
    - Even from different crawling threads
  - Must try to keep all crawling threads busy

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:
```

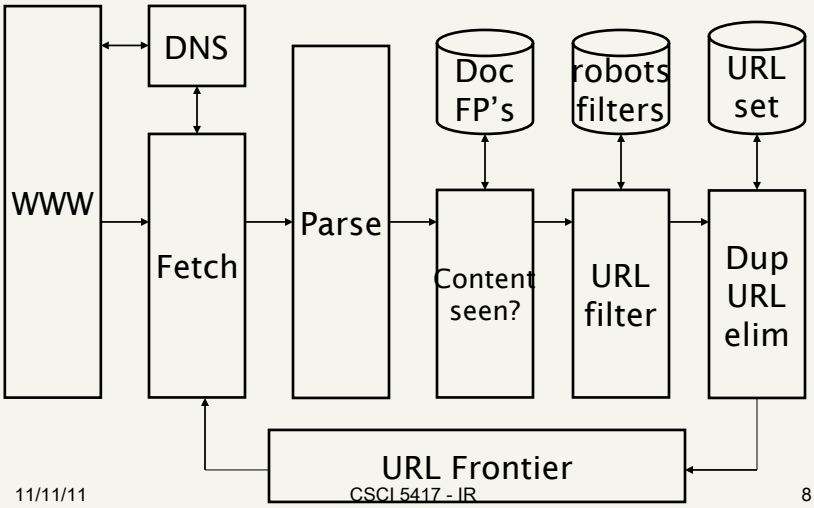
Processing steps in crawling

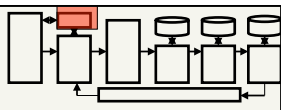
- Pick a URL from the frontier
- Fetch the document at the URL
- Parse the document
  - Extract links from it to other docs (URLs)
- Check if document has content already seen
  - If not, add to indexes
- For each extracted URL
  - Ensure it passes certain URL filter tests
  - Check if it is already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

Which one?

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

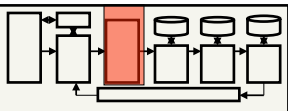
Basic crawl architecture





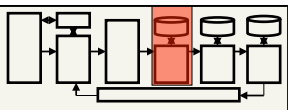
DNS (Domain Name Server)

- 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



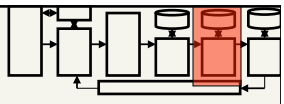
Parsing: URL normalization

- 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



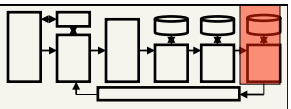
Content seen?

- 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 put into to the URL frontier
  - This may or may not be needed based on the crawling architecture

Distributing the crawler

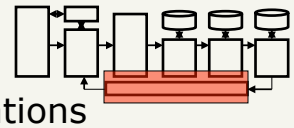
- Run multiple crawl threads, under different processes – potentially at different nodes
  - Geographically distributed nodes
- Partition hosts being crawled into nodes

Overview: Frontier

Crawlers provide discovered URLs (subject to filtering)

URL Frontier (discovered, but not yet crawled sites)

Crawler threads request URLs to crawl



URL frontier: two main considerations

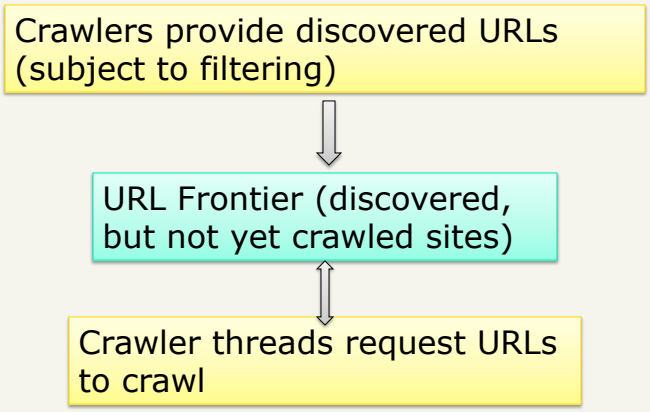
- Politeness: do not hit a web server too frequently
- Freshness: crawl some sites/pages more often than others
  - E.g., pages (such as News sites) whose content changes often

These goals may conflict each other.

Politeness – challenges

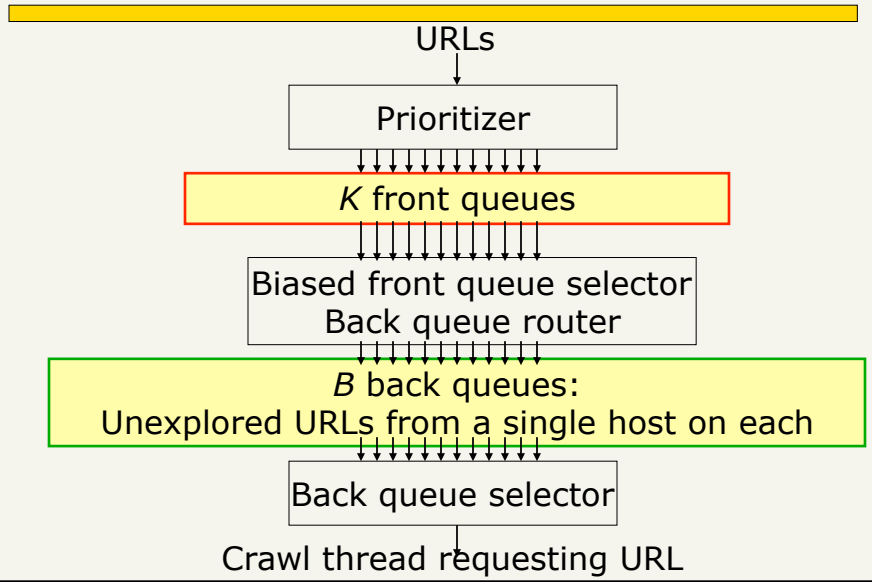
- Even if we restrict only one thread to fetch from a host, it can hit it repeatedly
- Common heuristic: insert time gap between successive requests to a host that is >> time for most recent fetch from that host

Overview: Frontier



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

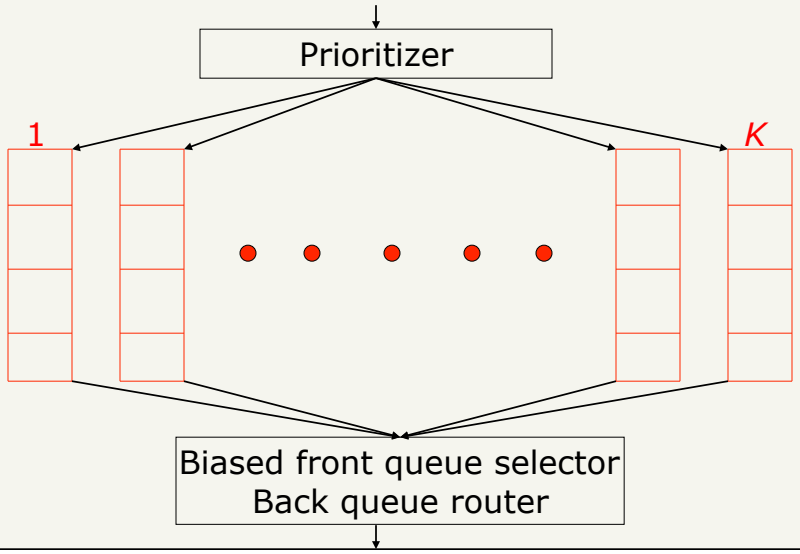
Explicit and implicit politeness

- 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

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")

Front queues

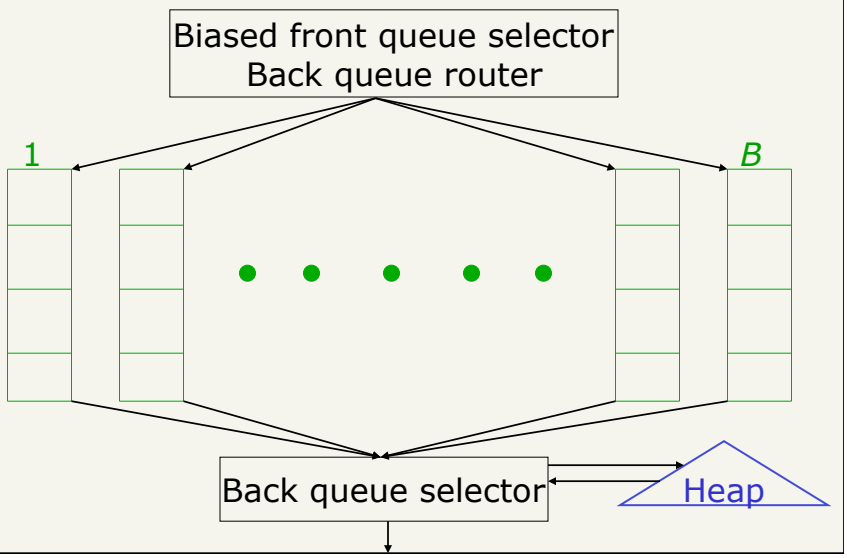


Biased front queue selector

- 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

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

Back Queue Management

- Each back queue corresponds to a given host
- When a crawler thread wants a URL to process we want to get it from the host that has been bothered least recently
- So we need a way to quickly get at the least recently visited host and a way to update that structure as hosts are added, deleted, and accessed
  - Hence the heap

Back Queue Management

- When a thread retrieves the last URL from a back queue it has to do additional work
- It visits the front queues based on the priority scheme and retrieves a URL
  - If it is a URL from a host that is already assigned to a back queue then it adds that URL to the right queue AND returns to the front queue for another
  - If it corresponds to a host not currently in the back queues then it adds it to the queue that it had emptied

Missing From the Pictures

- We better be indexing the documents. Not just storing the URLs
  - If we're caching ala google then we need to store the docs as well.
- And we had better be constructing a connectivity graph as we go along
  - To facilitate what we're going to do next

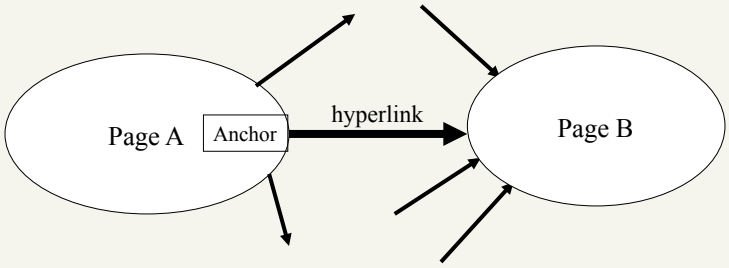
Break

- Nearly done with readings from the book. Current material is from Chapter 20 and 21.
- Then we're going back to Ch. 15; Section 15.4
- To support the material in Ch 15 I'm going to add some videos from the Stanford ML class. Basically 2 or 3 days worth
- And I'll post the reading for the Topic Models stuff.

Break

- For those of you who want to get started
- Go to ml-class.org
  - Video Lectures
    - Watch the videos in Sections II, IV and VI
      - Basically equiv to 3 of our classes

The Web as a Directed Graph

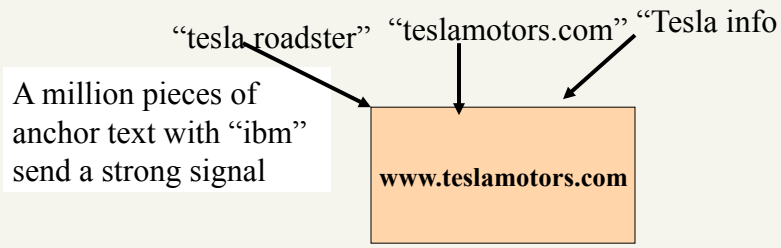


Assumption 1: A hyperlink between pages denotes author perceived relevance (quality signal)

Assumption 2: The anchor text of a hyperlink describes the target page (textual context)

Anchor Text

- For a query like ***tesla*** we would like it to return the Tesla home page first. But...
  - Tesla's home page is mostly graphical (I.e., low term count)
  - So use anchor text to augment



Indexing Anchor Text

- Can sometimes have unexpected side effects - *e.g.,* ***french military*** spoof
- Can index anchor text with less (or more) weight
  - Globally across all incoming links
  - Or as a function of the quality of the page the link is coming from

Traditional Citation Analysis

- Citation frequency
  - Citation impact ratings (of journals and authors)
    - Used in tenure decisions
- Bibliographic coupling frequency
  - Articles that co-cite the same articles are related
- Citation indexing
  - Who is author cited by? (Garfield [Garf72])

Web Link-Based Versions

- Two methods based on citation analysis literature
  - PageRank
    - Page and Brin -> Google
  - HITs
    - Kleinberg

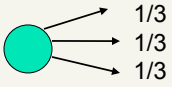
PageRank Sketch

- The pagerank of a page is based on the pagerank of the pages that point at it.
  - Roughly

$$\Pr(P) = \sum_{in \in P} \frac{\Pr(in)}{V(in)}$$

PageRank scoring

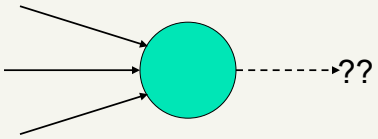
- Imagine a browser doing a random walk on web pages:
  - Start at a random page
  - At each step, go out of the current page along one of the links on that page, equiprobably



- "In the steady state" each page has a long-term visit rate - use this as the page's score
  - Pages with low rank are pages rarely visited during a random walk

Not quite enough

- The web is full of dead-ends. Pages that are pointed to but have no outgoing links
  - Random walk can get stuck in such dead-ends
  - Makes no sense to talk about long-term visit rates in the presence of dead-ends.



Teleporting

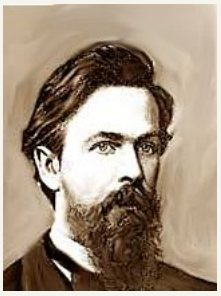
- At a dead end, jump to a random web page
- At any non-dead end, with probability 10%, jump to a random web page
  - With remaining probability (90%), go out on a random link.
  - 10% - a parameter (call it alpha)

Result of teleporting

- Now you can't get stuck locally.
- There is a long-term rate at which any page is visited
- How do we compute this visit rate?
  - Can't directly use the random walk metaphor

State Transition Probabilities

We're going to use the notion of a transition probability. If we're in some particular state, what is the probability of going to some other particular state from there.



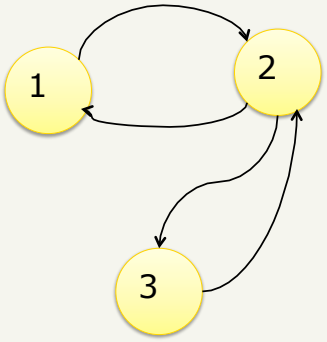
If there are n states (pages) then we need an n x n table of probabilities.

Markov Chains

- So if I'm in a particular state (say the start of a random walk)
- And I know the whole n x n table
- Then I can compute the probability distribution over all the next states I might be in in the next step of the walk...
- And in the step after that
  - And the step after that

Example

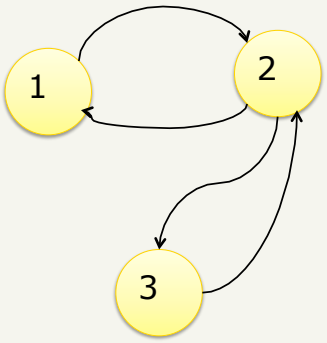
- Say alpha = .5



Example

- Say alpha = .5

P(3→2)

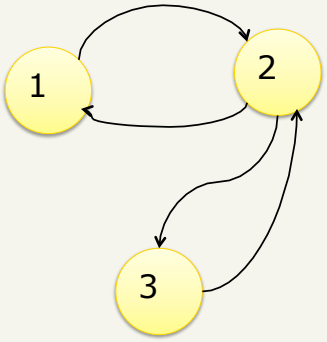


	?	

Example

- Say alpha = .5

P(3→2)

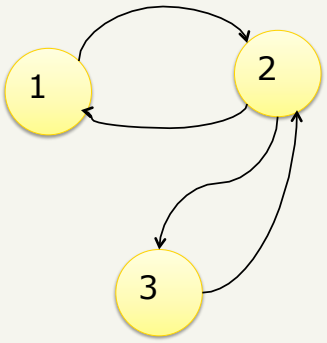


	2/3	

Example

- Say alpha = .5

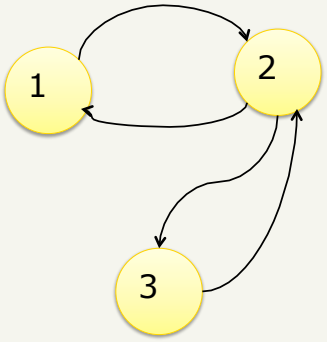
P(3→*)



1/6	2/3	1/6

Example

- Say alpha = .5

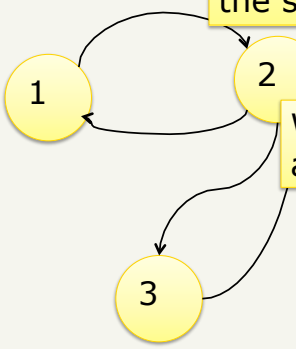


1/6	2/3	1/6
5/12	1/6	5/12
1/6	2/3	1/6

Example

- Say alpha = .5

Assume we start a walk in 1 at time T0. Then what should we believe about the state of affairs in T1?



What should we believe about things at T2?

1/6	2/3	1/6
5/12	1/6	5/12
1/6	2/3	1/6

Example

$\vec{x}_0$	1	0	0
$\vec{x}_1$	1/6	2/3	1/6
$\vec{x}_2$	1/3	1/3	1/3
$\vec{x}_3$	1/4	1/2	1/4
$\vec{x}_4$	7/24	5/12	7/24
...	...	...	...
$\vec{x}$	5/18	4/9	5/18

PageRank values

Pagerank summary

- Preprocessing:
  - Given graph of links, build matrix **P**.
  - From it compute **the page rank for each page**.
- Query processing:
  - Retrieve pages meeting query using the usual methods we've discussed
  - Rank them by their pagerank
  - Order is query-*independent*