Web Search

Advances, Crawling &
Link Analysis

Overview

A simple crawler

A real crawler

How hard can crawling be?

- Web search engines must crawl their documents.
- Getting the content of the documents is easier for many other IR systems.
  - E.g., indexing all files on your hard disk: just do a recursive descent on your file system
- Ok: for web IR, getting the content of the documents takes longer . . .
- . . . because of latency.
- But is that really a design/systems challenge?

Basic crawler operation

- Initialize queue with URLs of known seed pages
- Repeat
  - Take URL from queue
  - Fetch and parse page
  - Extract URLs from page
  - Add URLs to queue
- Fundamental assumption: The web is well linked.

Exercise: What’s wrong with this crawler?

```
urlqueue := (some carefully selected set of seed urls)
while urlqueue is not empty:
myurl := urlqueue.getlastanddelete()
mypage := myurl.fetch()
fetchedurls.add(myurl)
newurls := mypage.extracturls()
for myurl in newurls:
if myurl not in fetchedurls and not in urlqueue:
urlqueue.add(myurl)
addtoinvertedindex(mypage)
```

What’s wrong with the simple crawler

- Scale: we need to distribute.
- We can’t index everything: we need to subselect. How?
- Duplicates: need to integrate duplicate detection
- Spam and spider traps: need to integrate spam detection
- Politeness: we need to be “nice” and space out all requests for a site over a longer period (hours, days)
- Freshness: we need to recrawl periodically.
    - Because of the size of the web, we can do frequent recrawls only for a small subset.
    - Again, subselection problem or prioritization

Magnitude of the crawling problem

- To fetch 20,000,000,000 pages in one month . . .
- . . . we need to fetch almost 8000 pages per second!
- Actually: many more since many of the pages we attempt to crawl will be duplicates, unfetchable, spam etc.

What a crawler must do

Be polite

- Don’t hit a site too often
- Only crawl pages you are allowed to crawl: robots.txt

Be robust

- Be immune to spider traps, duplicates, very large pages, very large websites, dynamic pages etc

Robots.txt

- Protocol for giving crawlers ("robots") limited access to a website, originally from 1994
- Examples:
  - User-agent: *

    Disallow: /yoursite/temp/
  - User-agent: searchengine

    Disallow: /
- Important: cache the robots.txt file of each site we are crawling

Example of a robots.txt (nih.gov)

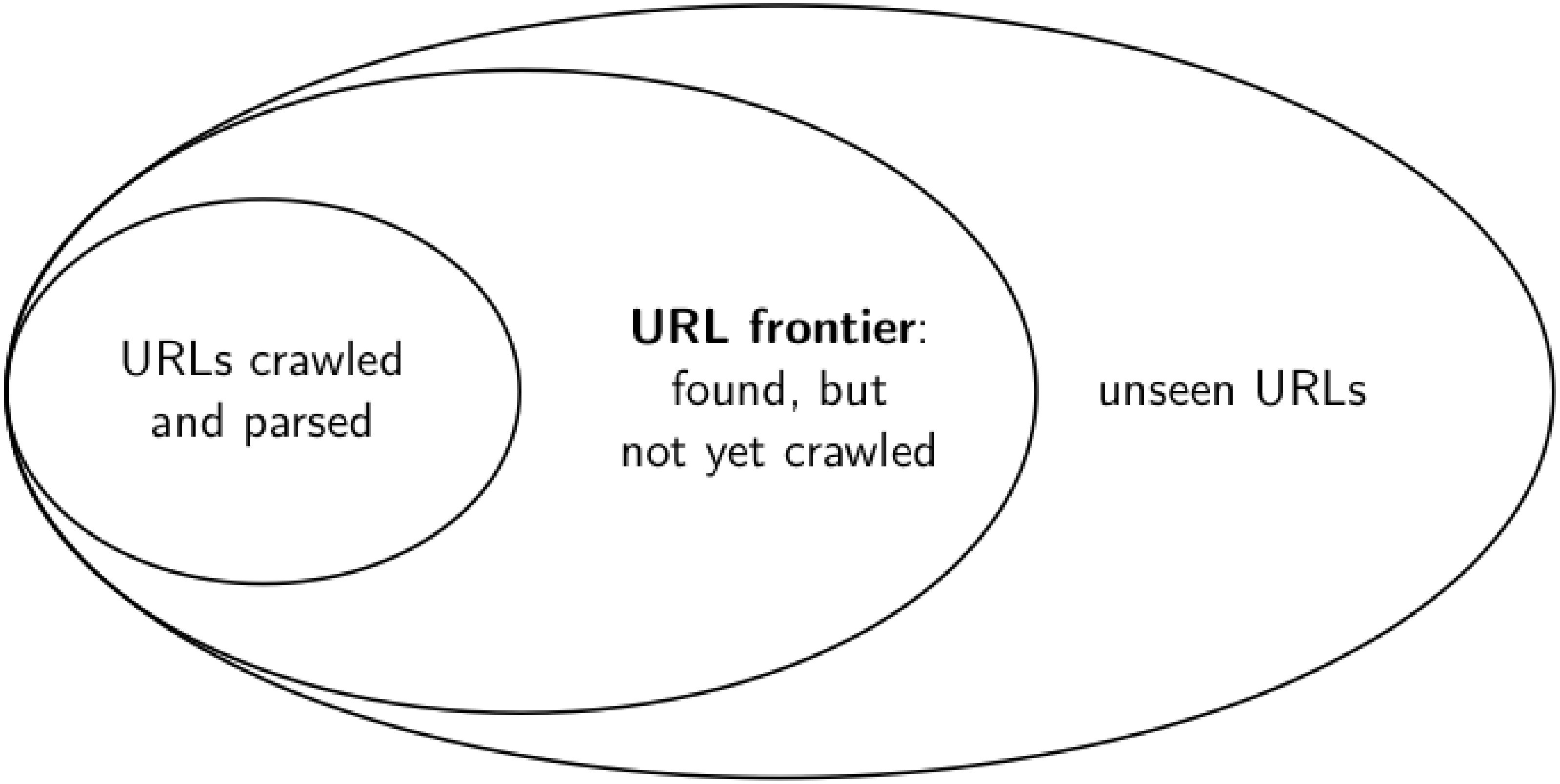
```
User-agent: PicoSearch/1.0
Disallow: /news/information/knight/
Disallow: /nidcd/
...
Disallow: /news/research_matters/secure/
Disallow: /od/ocpl/wag/
User-agent: *
Disallow: /news/information/knight/
Disallow: /nidcd/
...
Disallow: /news/research_matters/secure/
Disallow: /od/ocpl/wag/
Disallow: /ddir/
Disallow: /sdminutes/
```


10

What any crawler should do

- Be capable of distributed operation
- Be scalable: need to be able to increase crawl rate by adding more machines
- Fetch pages of higher quality first
- Continuous operation: get fresh version of already crawled pages

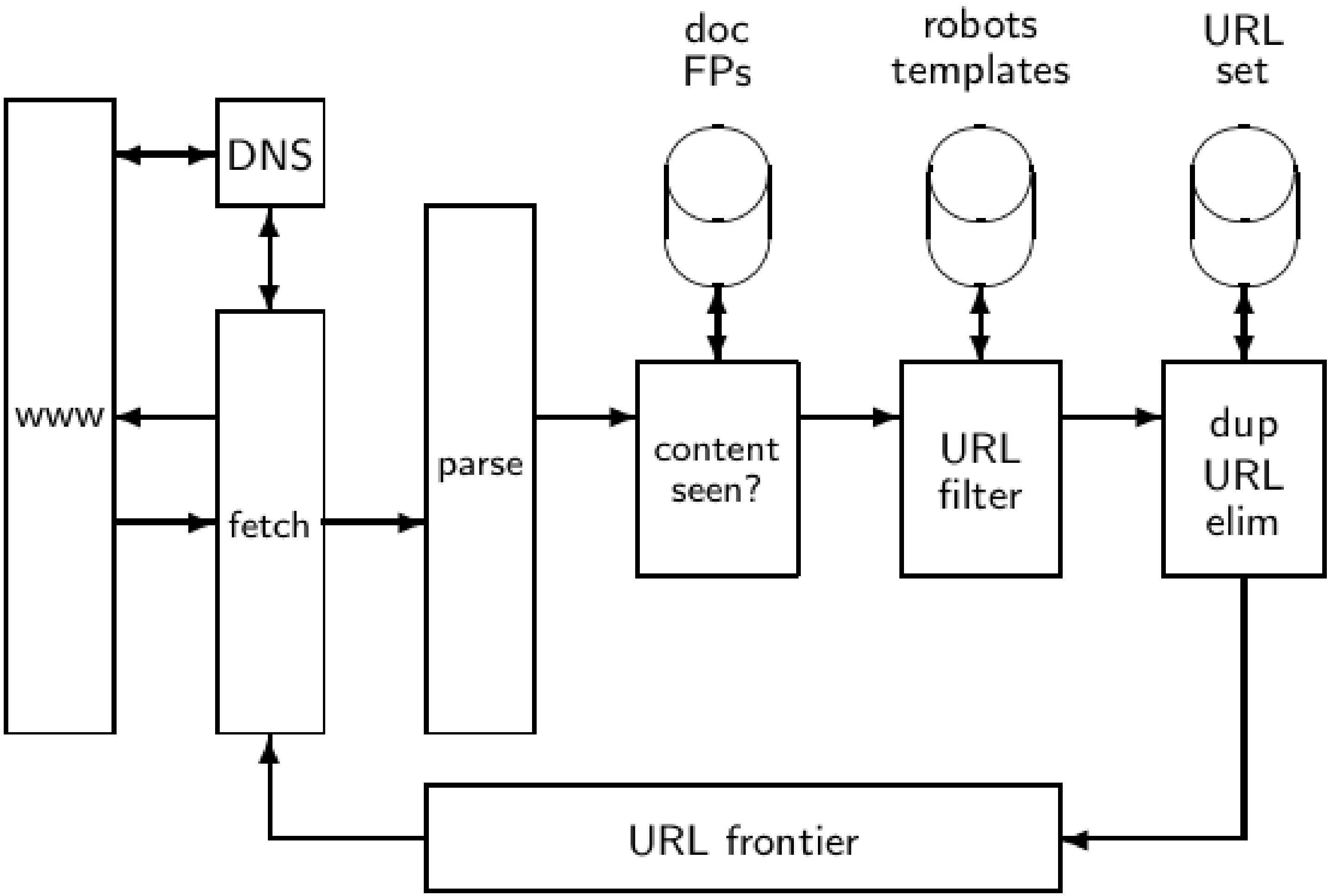
URL frontier



URL frontier

- The URL frontier is the data structure that holds and manages URLs we've seen, but that have not been crawled yet.
- Can include multiple pages from the same host
- Must avoid trying to fetch them all at the same time
- Must keep all crawling threads busy

Basic crawl architecture



URL normalization

- Some URLs extracted from a document are relative URLs.
  - E.g., at http://mit.edu, we may have aboutsite.html
    - This is the same as: http://mit.edu/aboutsite.html
- During parsing, we must normalize (expand) all relative URLs.

Content seen

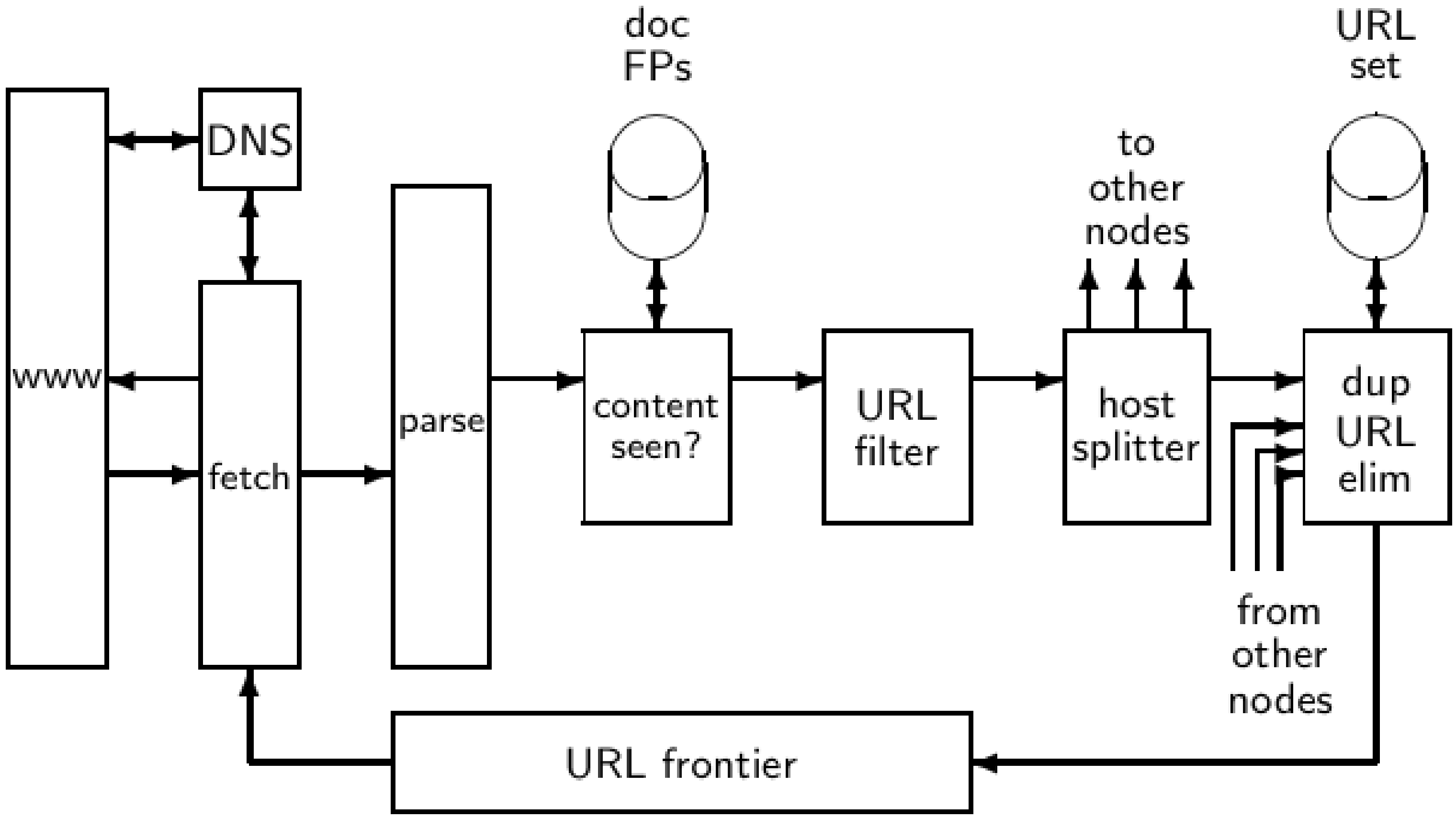
- For each page fetched: check if the content is already in the index
  - Check this using document fingerprints or shingles
  - Skip documents whose content has already been indexed

Distributing the crawler

- Run multiple crawl threads, potentially at different nodes
  - Usually geographically distributed nodes
  - Partition hosts being crawled into nodes

Google data centers (wazfaring. com)

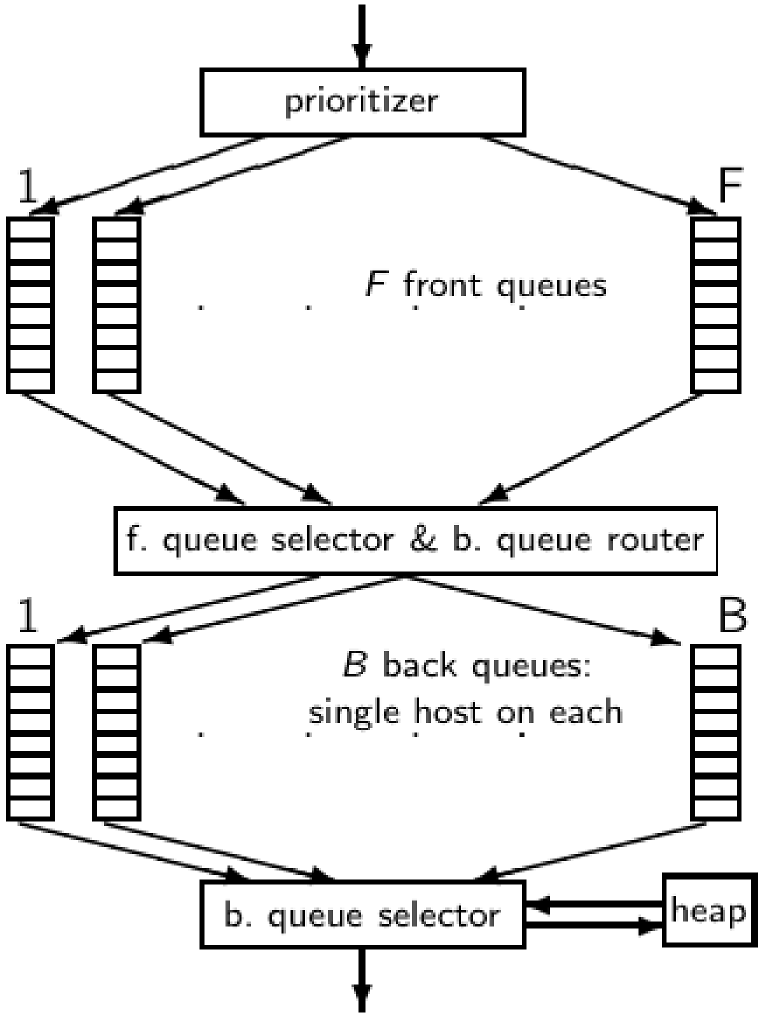
Distributed crawler



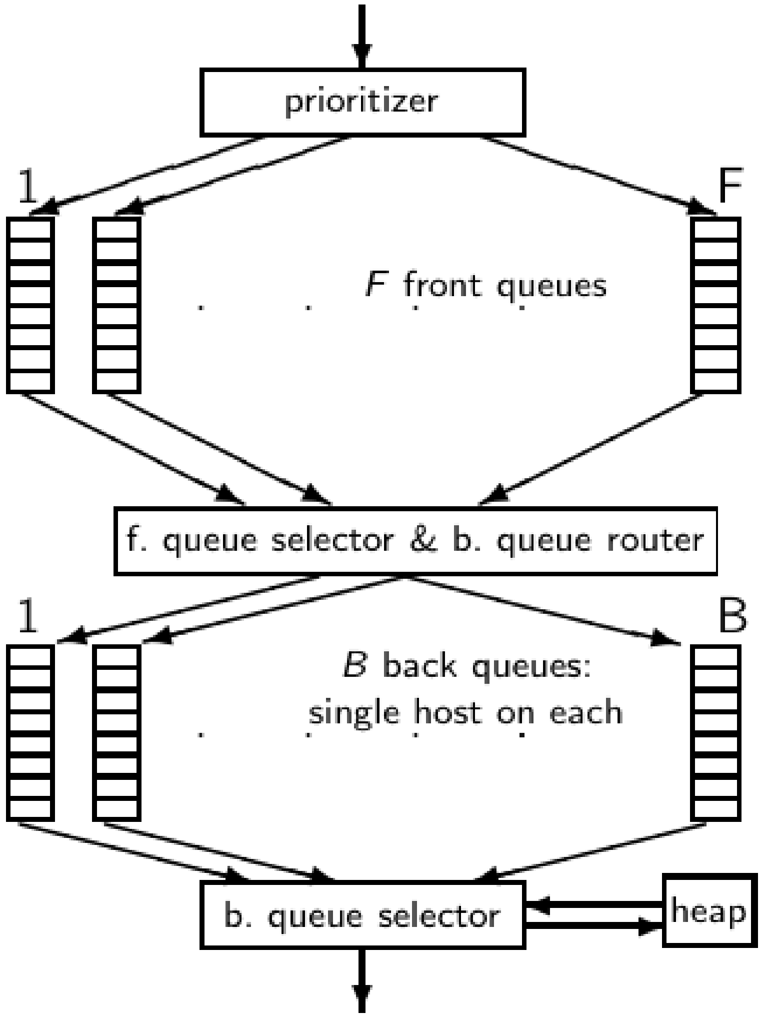
URL frontier: Two main considerations

- Politeness: Don’t hit a web server too frequently
  - E.g., insert a time gap between successive requests to the same server
- Freshness: Crawl some pages (e.g., news sites) more often than others
  - Not an easy problem: simple priority queue fails.

Mercator URL frontier



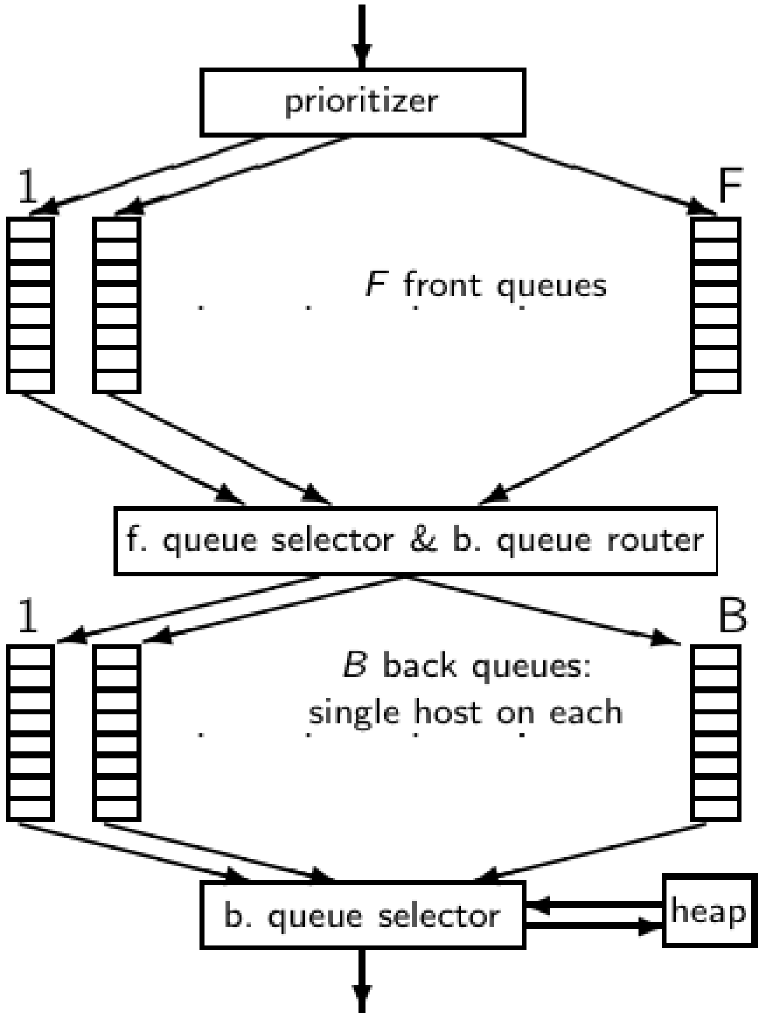
Mercator URL frontier



- URLs flow in from the top into the frontier.

22

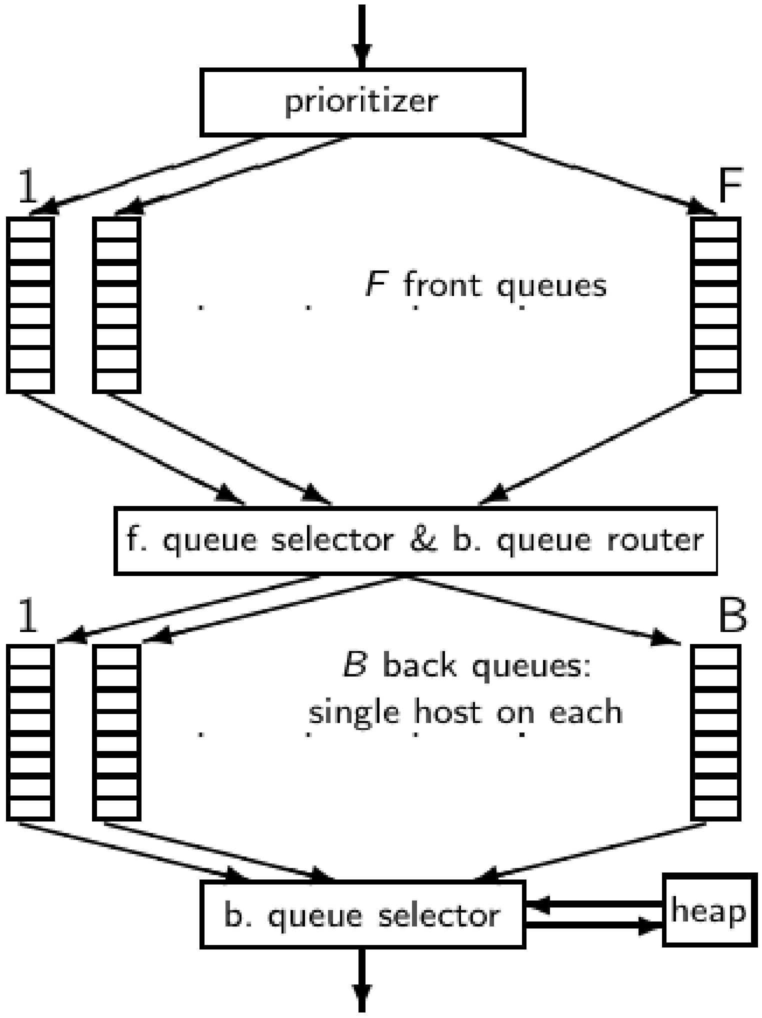
Mercator URL frontier



- URLs flow in from the top into the frontier.
- Front queues manage prioritization.

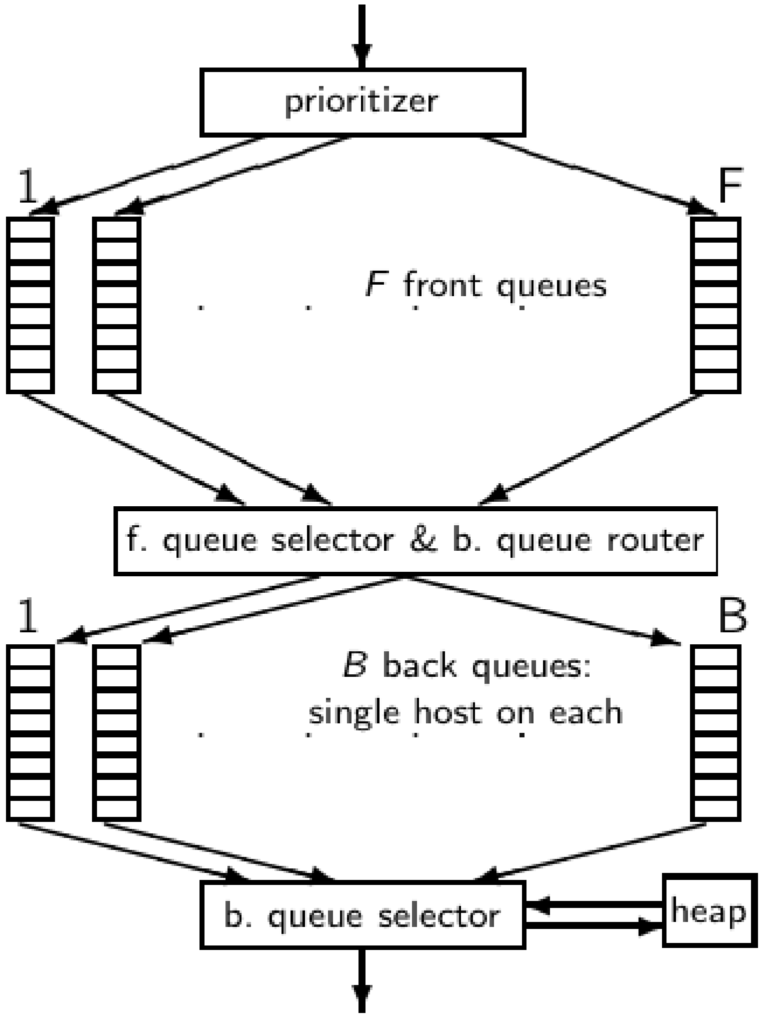
23

Mercator URL frontier



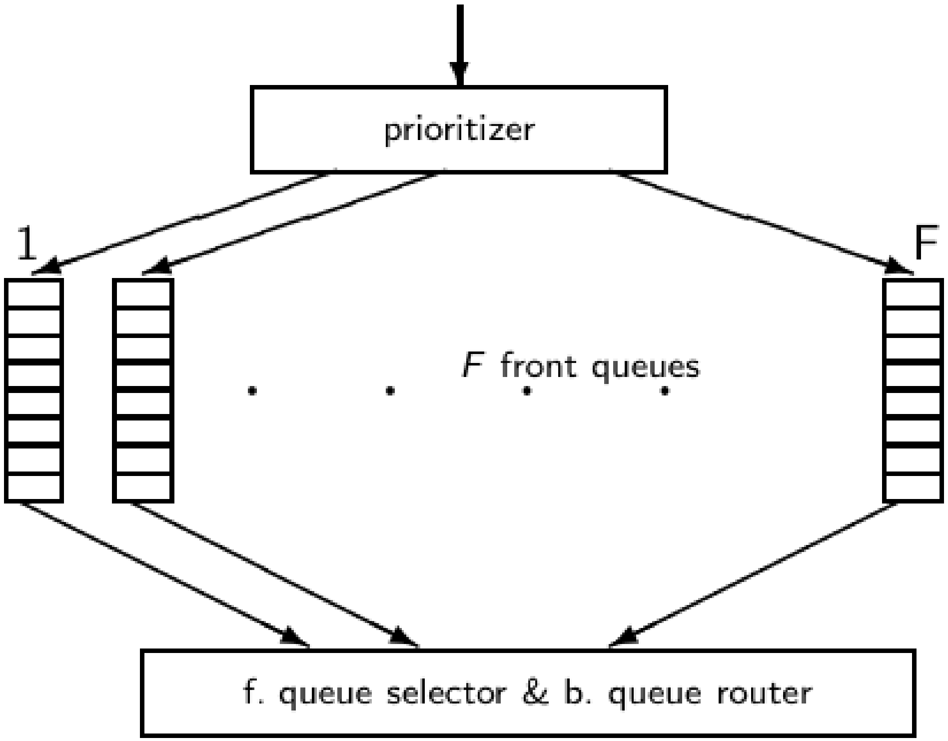
- URLs flow in from the top into the frontier.
- Front queues manage prioritization.
- Back queues enforce politeness.

Mercator URL frontier

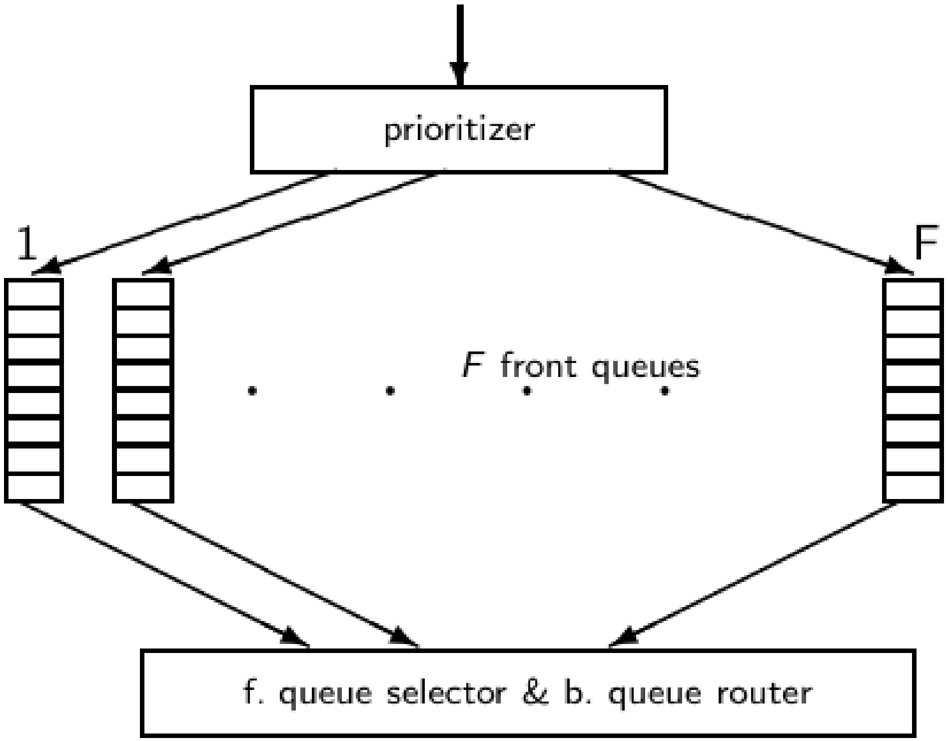


- URLs flow in from the top into the frontier.
- Front queues manage prioritization.
- Back queues enforce politeness.
- Each queue is FIFO.

Mercator URL frontier: Front queues



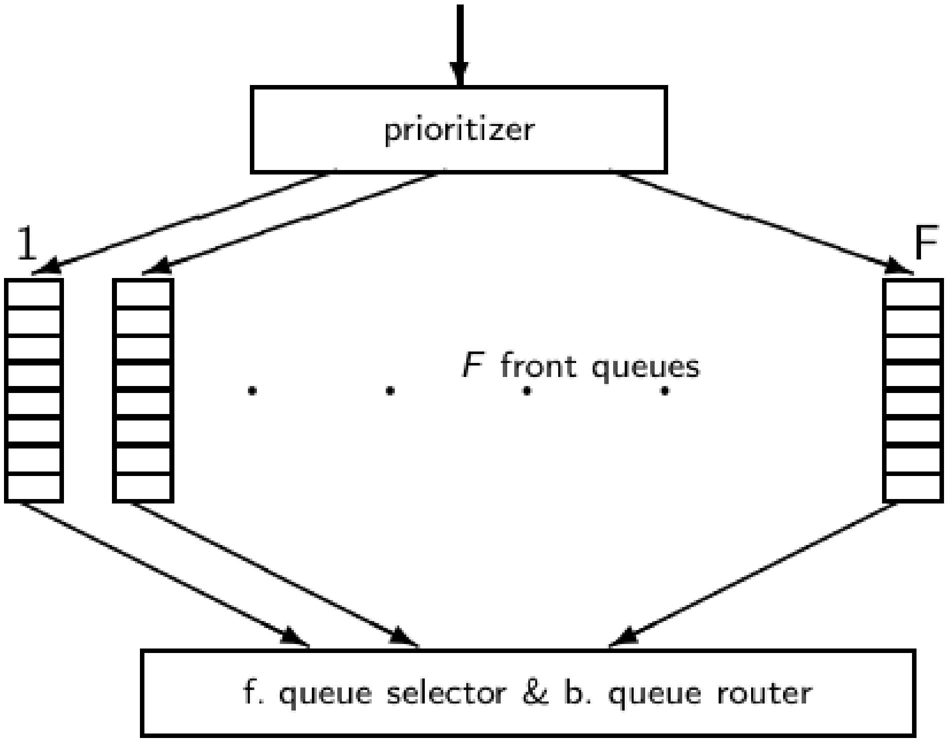
Mercator URL frontier: Front queues



- Prioritizer assigns to URL an integer priority between 1 and *F*.

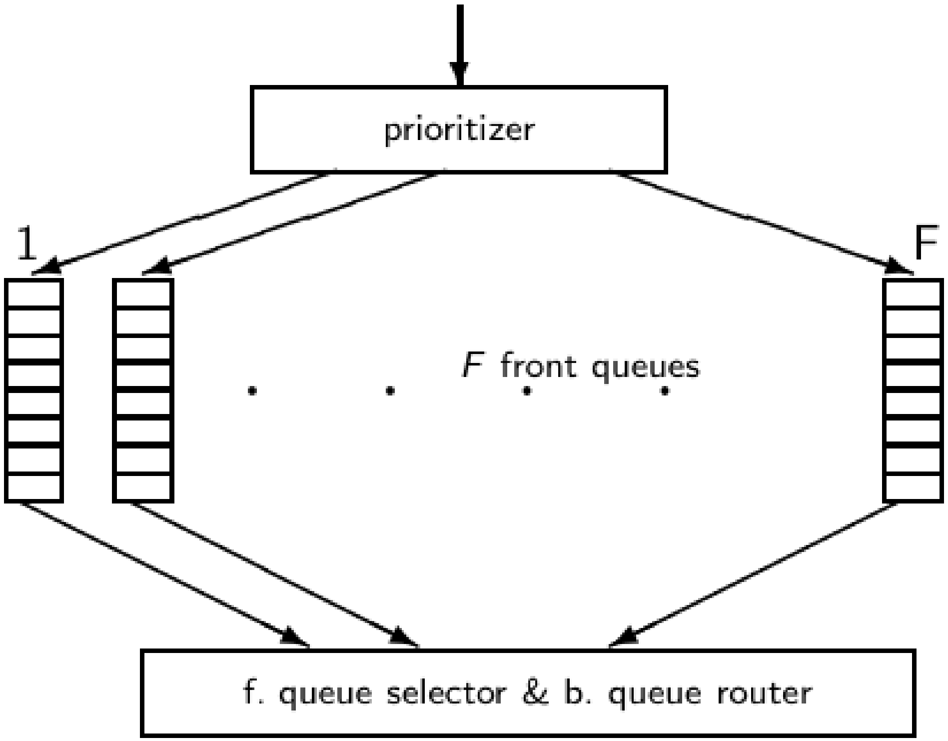
27

Mercator URL frontier: Front queues



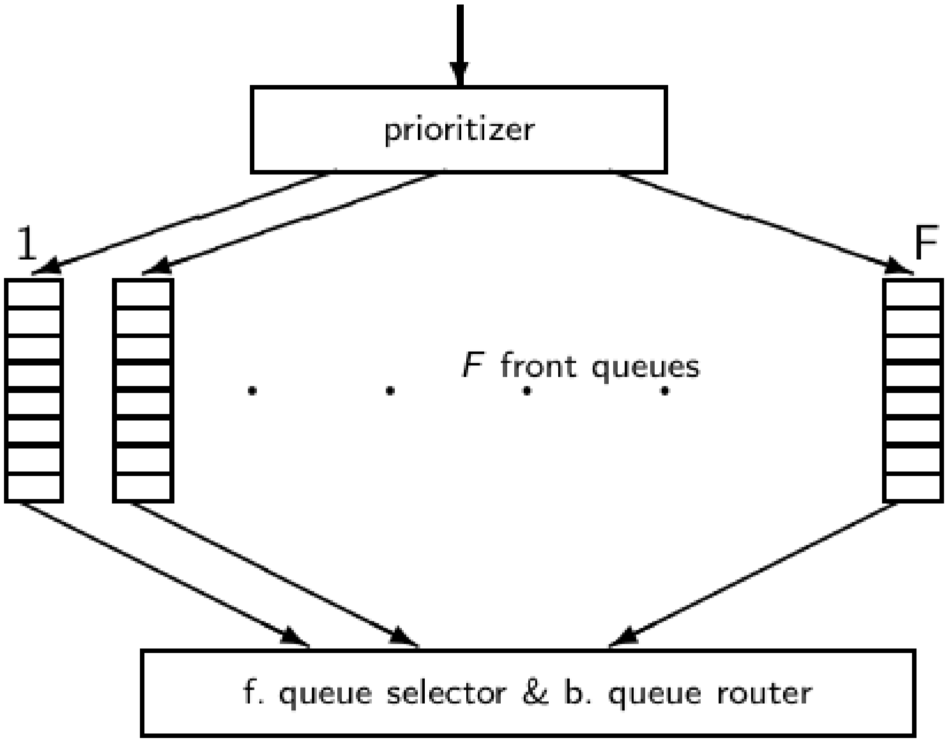
- Prioritizer assigns to URL an integer priority between 1 and *F*.
- Then appends URL to corresponding queue

Mercator URL frontier: Front queues



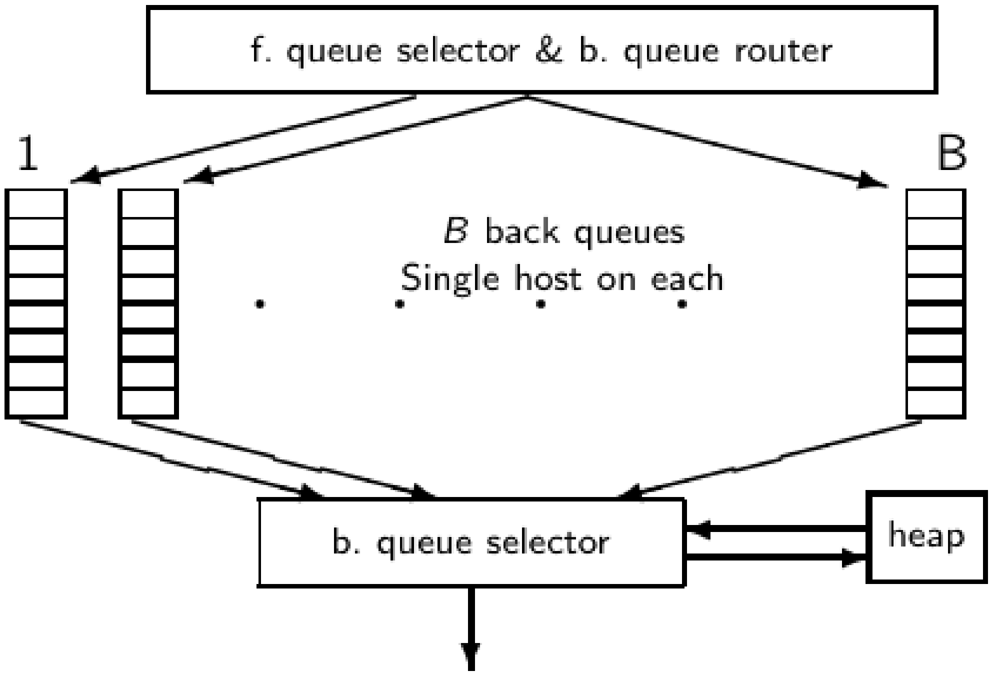
- Prioritizer assigns to URL an integer priority between 1 and *F*.
- Then appends URL to corresponding queue
- Heuristics for assigning priority: refresh rate, PageRank etc

Mercator URL frontier: Front queues

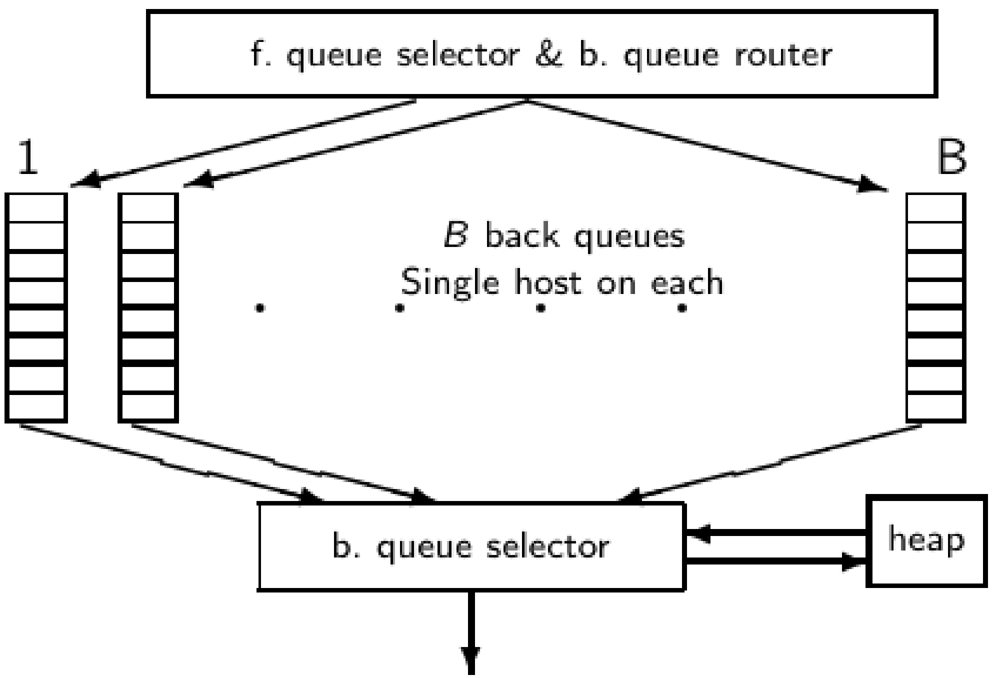


- Selection from front queues is initiated by back queues
- Pick a front queue from which to select next URL: Round robin, randomly, or more sophisticated variant
  - But with a bias in favor of high-priority front queues

Mercator URL frontier: Back queues

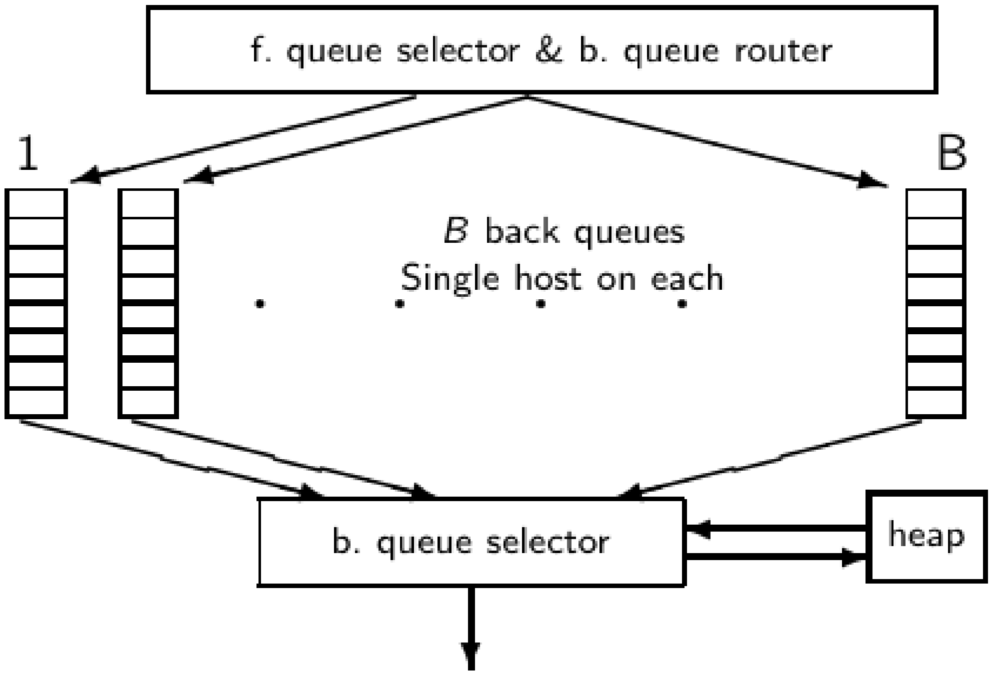


Mercator URL frontier: Back queues



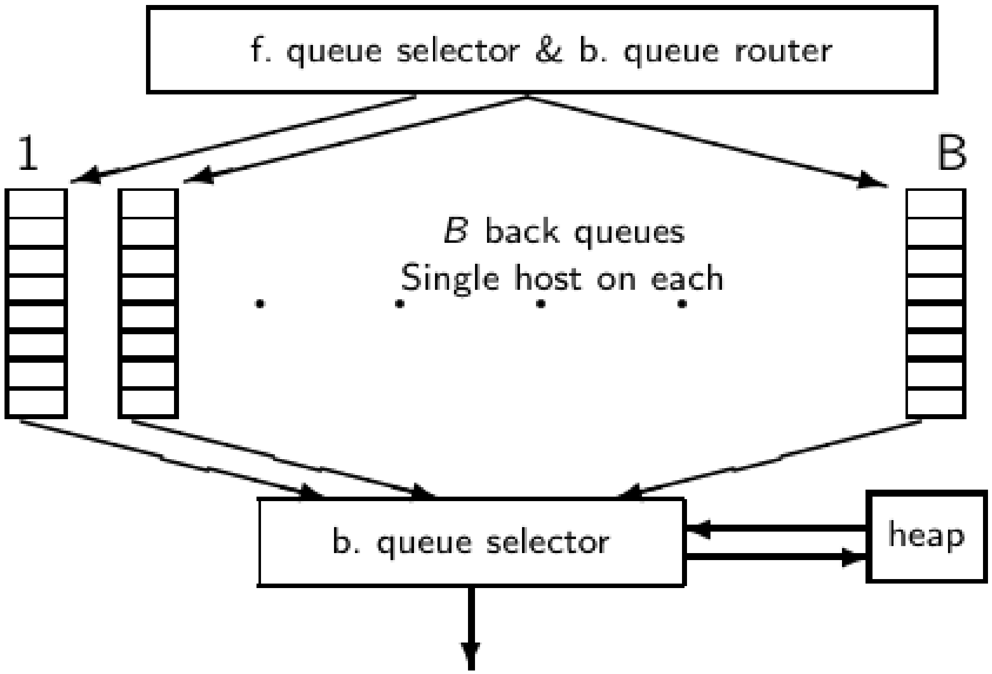
- Invariant 1. Each back queue is kept non-empty while the crawl is in progress.
- Invariant 2. Each back queue only contains URLs from a single host.
- Maintain a table from hosts to back queues.

Mercator URL frontier: Back queues



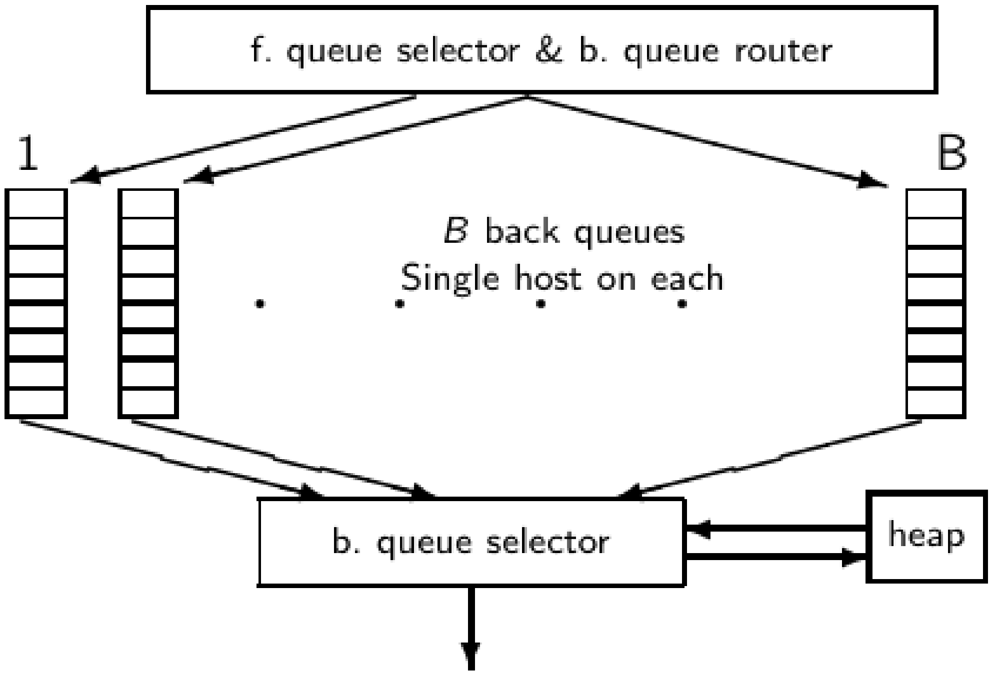
- In the heap:
- One entry for each back queue
- The entry is the earliest time t_e at which the host corresponding to the back queue can be hit again.
- The earliest time t_e is determined by (i) last access to that host (ii) time gap heuristic

Mercator URL frontier: Back queues



- How fetcher interacts with back queue:
- Repeat (i) extract current root q of the heap (q is a back queue)
- and (ii) fetch URL u at head of q . . .
- . . . until we empty the q we get.
- (i.e.: u was the last URL in q)

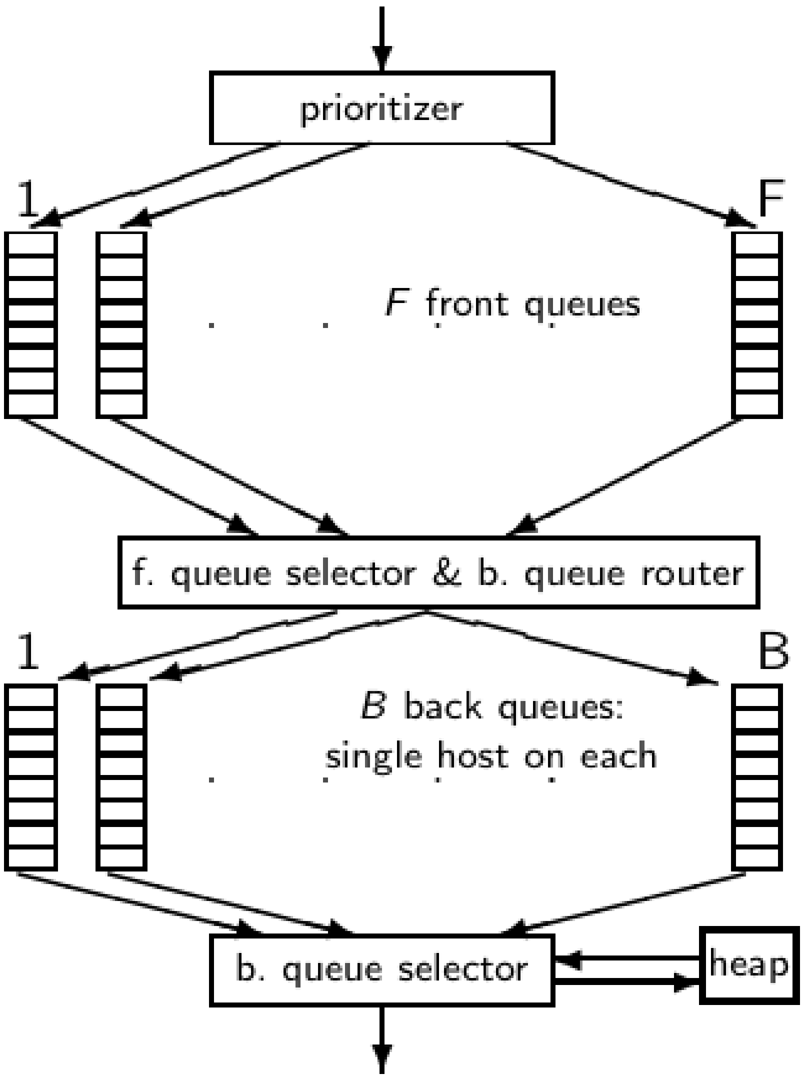
Mercator URL frontier: Back queues



- When we have emptied a back queue q:
- Repeat (i) pull URLs u from front queues and (ii) add u to its corresponding back queue . . .
- . . . until we get a u whose host does not have a back queue.
- Then put u in q and create heap entry for it.

35

Mercator URL frontier



- URLs flow in from the top into the frontier.
- Front queues manage prioritization.
- Back queues enforce politeness.

Spider trap

- Malicious server that generates an infinite sequence of linked pages
- Sophisticated spider traps generate pages that are not easily identified as dynamic.

Resources

- Chapter 20 of IIR
- Resources at http://ifnlp.org/ir
- Paper on Mercator by Heydon et al.
- Robot exclusion standard

Meta-Search Engines

- Search engine that passes query to several other search engines and integrate results.
  - Submit queries to host sites.
  - Parse resulting HTML pages to extract search results.
  - Integrate multiple rankings into a “consensus” ranking.
  - Present integrated results to user.
- Examples:
  - Metacrawler
  - SavvySearch
  - Dogpile

HTML Structure & Feature Weighting

- Weight tokens under particular HTML tags more heavily (Semi-structured Data):
  – <TITLE> tokens (Google seems to like title matches)
  – <H1>,<H2>… tokens
  – <META> keyword tokens
- Parse page into sections and weight tokens differently based on section: Multitier Indexing

  Title, Abstract, Body,……
- Links can also be a major factor (*Citations?*)

Bibliometrics: Citation Analysis

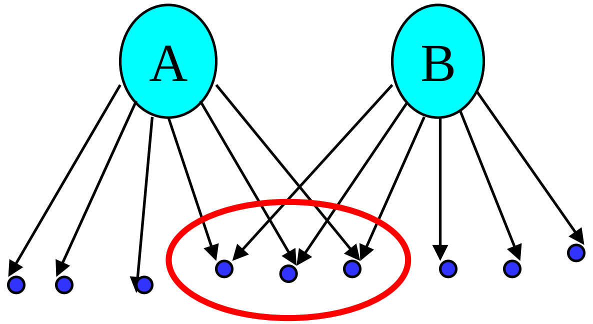
- Many standard documents include ***own*** *bibliographies* (or *references*), explicit *citations* to ***other*** previously published documents.
- Using citations as links, standard corpora can be viewed as a *graph*.
- The structure of this graph, independent of content, can provide interesting information about the similarity of documents and the structure of information.
- In Science/Academia this is the norm! Promotions

Impact Factor

- Developed by Garfield in 1972 to measure the importance (quality, influence) of scientific journals.
- Measure of how often papers in the journal are cited by other scientists.
- Computed and published annually by, e.g. the Institute for Scientific Information (ISI).
- The *Impact Factor* (IF) of a journal J in year Y is the average number of citations (from indexed documents published in year Y) to a paper published in J in year Y–1 or Y–2.
- **Does not account for the quality of the citing article**.

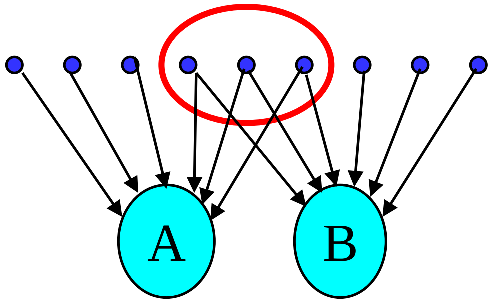
Bibliographic Coupling

- BC: A Measure of similarity of documents introduced by Kessler in 1963.
- The bibliographic coupling of two documents *A* and *B* is the number of documents cited by *both A* and *B* *(documents based on same data are similar!)-overlap-*.
- Size of the intersection of their bibliographies.
- Maybe want to normalize by size of bibliographies?



Co-Citation

- An alternate citation-based measure of similarity introduced by Small in 1973.
- Number of documents that cite both *A* and *B*. (*Similar documents are cited in same articles!*)
- Maybe want to normalize by total number of documents citing either *A* or *B* ?

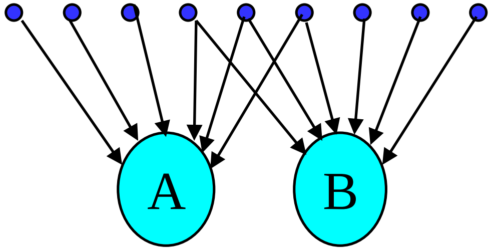


Citations vs. Links

- Web links are a bit different than citations:
  - Many links are navigational.
  - Many pages with high in-degree are portals not content providers (not documents).
  - Not all links are endorsements.
  - Company websites don’t point to their competitors.
  - Citations to relevant literature is ***enforced*** by peer-review. Not the case for web pages

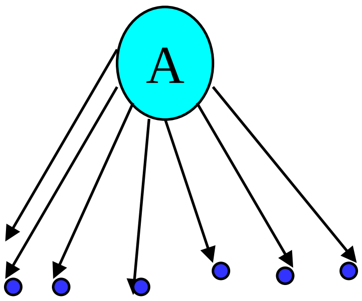
Authorities

- ***Authorities*** are pages that are recognized as providing significant, trustworthy, and useful information on a topic (مرجع).
- *In-degree* (number of pointers to a page) is one simple measure of authority (6 here).
- However in-degree treats all links as equal.
- Should links from pages that are themselves authoritative count more?



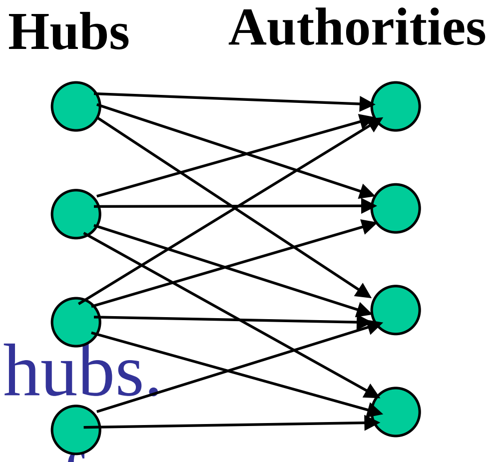
Hubs

- ***Hubs*** are index pages that provide lots of useful links to relevant content pages (topic authorities). Large Out-Degree.
- Hub pages for IR are included in the course home page:



HITS (Hypertext Induced Topic Search)

- Developed by Kleinberg in 1998.
- Attempts to determine hubs and authorities on a particular topic through analysis of a ***relevant*** subgraph of the web.
- Based on mutually recursive facts:
  - Hubs **point to** lots of authorities.
  - Authorities are **pointed to** by lots of hubs.
  - Hubs and Authorities together tend to form a bipartite graph:

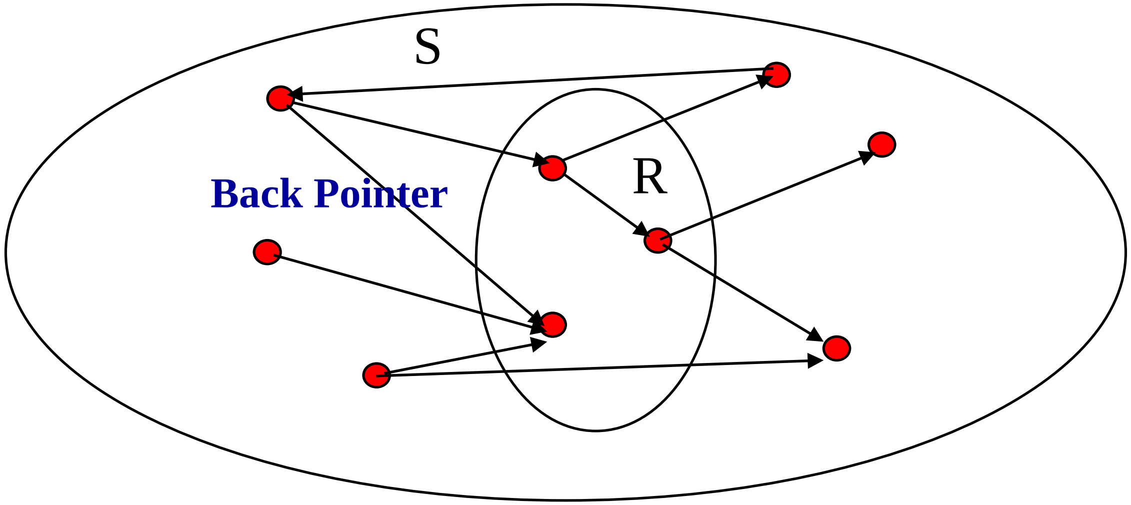


HITS Algorithm

- Computes hubs and authorities for a particular topic specified by a normal query Q.
- First determines a set of relevant pages for the query called the ***base*** set *S*.
- Analyze the link structure of the web subgraph defined by *S* (pages linked with –to, from- S) to find **authority** and **hub** pages in this set.

Constructing a Base Subgraph

- For a specific query *Q*, let the set of documents returned by a standard search engine be called the *root* set *R*.
- Initialize *S* to *R*. Then expand as follows:
- Add to *S* all pages pointed to by any page in *R*.
- Add to *S* all pages that point to any page in *R*.



Base Limitations

- To limit computational expense:
  - Limit number of root pages to the top 200 pages retrieved for the query.
  - Limit number of "back-pointer" pages to a random set of at most 50 pages returned by a "reverse link" query.
- To eliminate purely navigational links:
  - Eliminate links between two pages on the same host.
- To eliminate "non-authority-conveying" links:
  - Allow only m ($m \simeq 4$–8) pages from a given host as pointers to any individual page.

Authorities and In-Degree

- Even within the base set *S* for a given query, the nodes with highest in-degree are not necessarily authorities (may just be generally popular pages like Yahoo or Amazon).
- True authority pages are pointed to by a number of hubs (i.e. pages that point to lots of authorities).

Iterative Algorithm

- Use an iterative algorithm to slowly converge on a mutually reinforcing set of hubs and authorities.
- Maintain for each page $p \in S$:
  - Authority score: a_p (vector $\boldsymbol{a}$)
  - Hub score: h_p (vector $\boldsymbol{h}$)
- Initialize all $a_p = h_p = 1$
- Maintain normalized scores:

$$\sum_{p \in S} (a_p)^2 = 1 \qquad \sum_{p \in S} (h_p)^2 = 1$$

HITS Update Rules

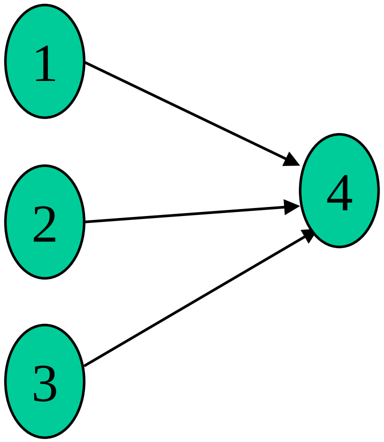
- Authorities are pointed to by lots of good hubs:

$$a_p = \sum_{q:q \to p} h_q$$

- Hubs point to lots of good authorities:

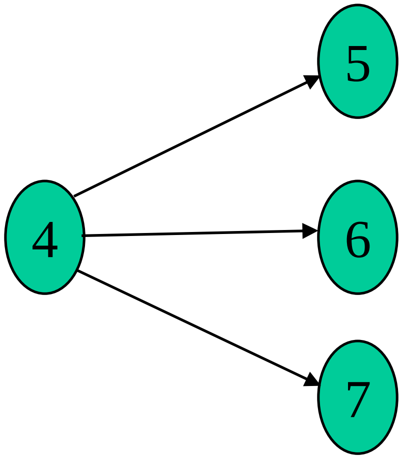
$$h_p = \sum_{q:p \to q} a_q$$

Illustrated Update Rules



$a_4 = h_1 + h_2 + h_3$

$h_4 = a_5 + a_6 + a_7$



HITS Iterative Algorithm

Initialize for all $p \in S$: $a_p = h_p = 1$

For i = 1 to k:

For all $p \in S$: $a_p = \sum_{q:q \to p} h_q$ (*update auth. scores*)

For all $p \in S$: $h_p = \sum_{q:p \to q} a_q$ (*update hub scores*)

For all $p \in S$: $a_p = a_p/c$ c: $\sum_{p \in S} (a_p / c)^2 = 1$ (*normalize* ***a***)

For all $p \in S$: $h_p = h_p/c$ c: $\sum_{p \in S} (h_p / c)^2 = 1$ (*normalize* **h**)

Convergence

- Algorithm converges to a *fix-point* if iterated indefinitely.
- Define A to be the adjacency matrix for the subgraph defined by S.
  - $A_{ij} = 1$ for $i \in$ S, $j \in$ S iff $i \rightarrow j$
- Authority vector, $\boldsymbol{a}$, converges to the principal eigenvector of $A^T A$
- Hub vector, $\boldsymbol{h}$, converges to the principal eigenvector of AA^T
- In practice, 20 iterations produces fairly stable results.

Results

- Authorities for query: "Java"
  - java.sun.com
  - comp.lang.java FAQ
- Authorities for query "search engine"
  - Yahoo.com
  - Excite.com
  - Lycos.com
  - Altavista.com
- Authorities for query "Gates"
  - Microsoft.com
  - roadahead.com

Result Comments

- In most cases, the final authorities were not in the initial root set generated using Altavista.
- Authorities were brought in from linked and reverse-linked pages and then HITS computed their high authority score.

Finding Similar Pages Using Link Structure

- Given a page, *P*, let *R* (the root set) be *t* (e.g. 200) pages that point to *P*.
- Grow a base set *S* from *R*.
- Run HITS on *S*.
- Return the best authorities in *S* as the best similar-pages for *P*.
- Finds authorities in the "link neighbor-hood" of *P*.

Similar Page Results

- Given “honda.com”
  - toyota.com
  - ford.com
  - bmwusa.com
  - saturncars.com
  - nissanmotors.com
  - audi.com
  - volvocars.com

HITS for Clustering

- An ambiguous query can result in the principal eigenvector only covering one of the possible meanings.
- Non-principal eigenvectors may contain hubs & authorities for other meanings.
- Example: “jaguar”:
  - Atari video game (principal eigenvector)
  - NFL Football team (2^{nd} non-princ. eigenvector)
  - Automobile (3^{rd} non-princ. eigenvector)

PageRank

- Alternative link-analysis method used by Google (Brin & Page, 1998).
- Does not attempt to capture the distinction between hubs and authorities.
- Ranks pages just by authority.
- Applied to the entire web rather than a local neighborhood of pages surrounding the results of a query.

Initial PageRank Idea

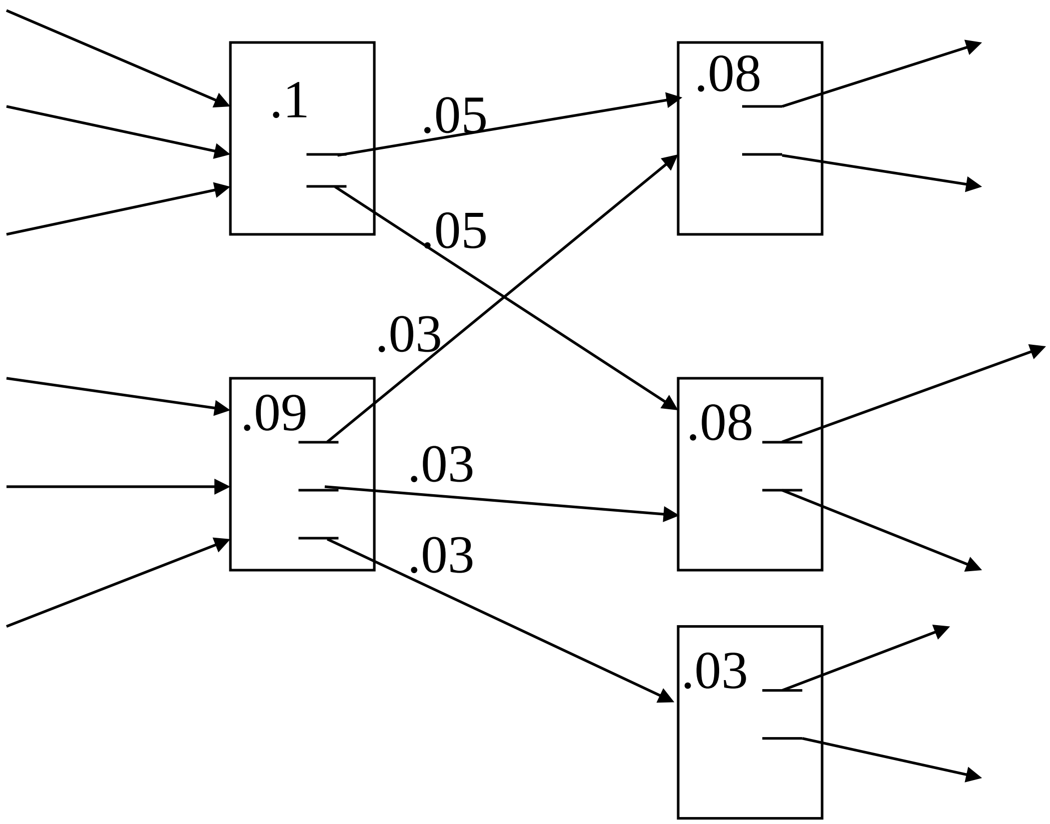
- Just measuring in-degree (citation count) doesn't account for the authority of the source of a link.
- Initial page rank equation for page *p*:

$$R(p) = c \sum_{q:q \to p} \frac{R(q)}{N_q}$$

  - N_q is the total number of out-links from page *q*.
  - A page, *q*, "gives" an equal fraction of its authority to all the pages it points to (e.g. *p*).
  - c is a normalizing constant set so that the rank of all pages always sums to 1.

Initial PageRank Idea (cont.)

- Can view it as a process of PageRank "flowing" from pages to the pages they cite.



Initial Algorithm

- Iterate rank-flowing process until convergence:

  Let S be the total set of pages.

  Initialize $\forall p \in S$: $R(p) = 1/|S|$

  Until ranks do not change (much) (*convergence*)

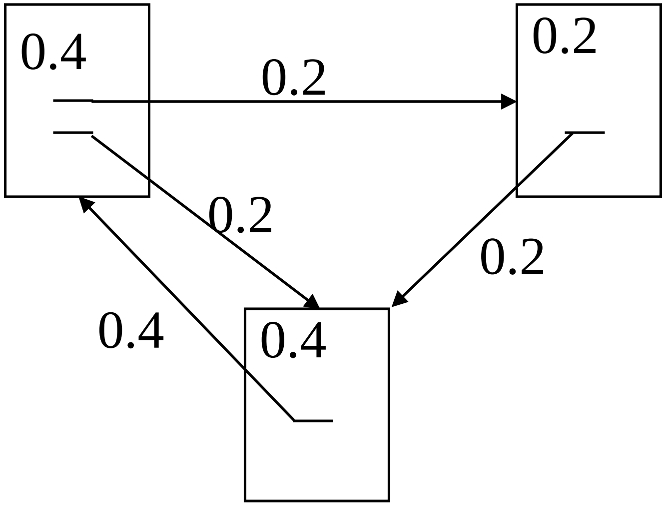
  For each $p \in S$:

  $$R'(p) = \sum_{q:q \to p} \frac{R(q)}{N_q}$$

  $$c = 1/\sum_{p \in S} R'(p)$$

  For each $p \in S$: $R(p) = cR'(p)$ (*normalize*)

Sample Stable Fixpoint

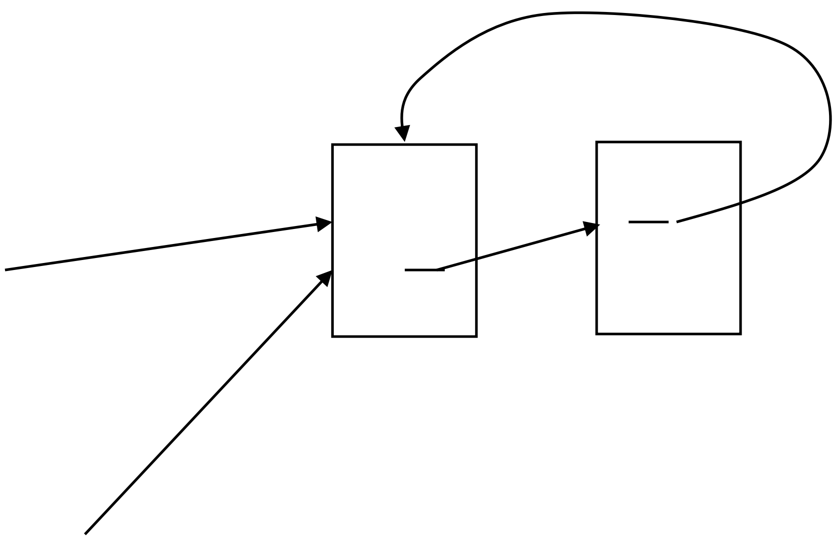


Linear Algebra Version

- Treat **R** as a vector over web pages.
- Let **A** be a 2-d matrix over pages where
  - $\mathbf{A}_{vu} = 1/N_u$ if $u \rightarrow v$ else $\mathbf{A}_{vu} = 0$
- Then $\mathbf{R} = c\mathbf{AR}$
- **R** converges to the principal eigenvector of **A**.

Problem with Initial Idea

- A group of pages that only point to themselves but are pointed to by other pages act as a “rank sink” and absorb all the rank in the system.



Rank flows into
cycle and can’t get out

Rank Source

- Introduce a "rank source" E that continually replenishes the rank of each page, p, by a fixed amount $E(p)$.

$$R(p) = c\left(\sum_{q:q \to p} \frac{R(q)}{N_q} + E(p)\right)$$

PageRank Algorithm

Let S be the total set of pages.

Let $\forall p \in S$: $E(p) = \alpha/|S|$ (for some $0<\alpha<1$, e.g. 0.15)

Initialize $\forall p \in S$: $R(p) = 1/|S|$

Until ranks do not change (much) (*convergence*)

For each $p \in S$:

$$R'(p) = \left[(1-\alpha) \sum_{q:q \to p} \frac{R(q)}{N_q} \right] + E(p)$$

$$c = 1 / \sum_{p \in S} R'(p)$$

For each $p \in S$: $R(p) = cR'(p)$ (*normalize*)

Linear Algebra Version

- $\mathbf{R} = c(\mathbf{AR} + \mathbf{E})$
- Since $\|\mathbf{R}\|_1 = 1$: $\mathbf{R} = c(\mathbf{A} + \mathbf{E}\times\mathbf{1})\mathbf{R}$
  - Where $\mathbf{1}$ is the vector consisting of all 1's.
- So $\mathbf{R}$ is an eigenvector of $(\mathbf{A} + \mathbf{E}\times\mathbf{1})$

Random Surfer Model

- PageRank can be seen as modeling a "random surfer" that starts on a random page and then at each point:
  - With probability $E(p)$ randomly jumps to page p.
  - Otherwise, randomly follows a link on the current page.
- $R(p)$ models the probability that this random surfer will be on page p at any given time.
- "E jumps" are needed to prevent the random surfer from getting "trapped" in web sinks with no outgoing links.

Speed of Convergence

- Early experiments on Google used 322 million links.
- PageRank algorithm converged (within small tolerance) in about 52 iterations.
- Number of iterations required for convergence is empirically O(log n) (where n is the number of links).
- Therefore calculation is quite efficient.

Simple Title Search with PageRank

- Use simple Boolean search to search web-page titles and rank the retrieved pages by their PageRank.
- Sample search for “university”:
  - Altavista returned a random set of pages with “university” in the title (seemed to prefer short URLs).
  - Primitive Google returned the home pages of top universities.

Google Ranking

- Complete Google ranking includes (based on university publications prior to commercialization).
  - Vector-space similarity component.
  - Keyword proximity component.
  - HTML-tag weight component (e.g. title preference).
  - PageRank component.
- Details of current commercial ranking functions are trade secrets.

Personalized PageRank

- PageRank can be biased (personalized) by changing **E** to a non-uniform distribution.
- Restrict "random jumps" to a set of specified relevant pages.
- For example, let $E(p) = 0$ except for one's own home page, for which $E(p) = \alpha$
- This results in a bias towards pages that are closer in the web graph to your own homepage.

Google PageRank-Biased Spidering

- Use PageRank to direct (focus) a spider on "important" pages.
- Compute page-rank using the current set of crawled pages.
- Order the spider's search queue based on current estimated PageRank.

Link Analysis Conclusions

- Link analysis uses information about the structure of the web graph to aid search.
- It is one of the major innovations in web search.
- It was one of the primary reasons for Google's initial success.