Crawling the Web

Web Crawling

❖Retrieve (for indexing, storage, …) Web pages by using the links found on a page to locate more pages.

Must have some starting point

Type of crawl

- Web crawl versus

  crawl of more limited network – web
  - cs.princeton.edu
  - internal co. network
- complete crawl versus

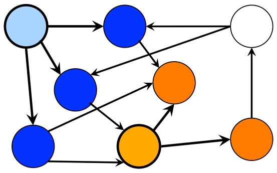
  focused crawl by some criteria
  - pages on one topic
- Type of crawl will affect necessity/usability of various techniques

Main Issues I

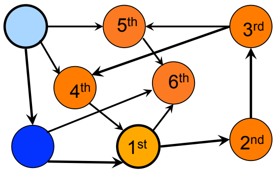
- starting set of pages?
  - a.k.a "seed" URLs
- can visit whole of Web (or web)?
- how determine order to visit links?
  - graph model:

    breadth first vs depth first
    - what are pros and cons of each?
    - "black holes"
  - other aspects /considerations
    - how deep want to go?
    - associate priority with links

- Breadth-first:



- Depth-first:



"Black holes" and other "baddies"

- "Black hole": Infinite chain of pages
  - dynamically generated
  - not always malicious
    - link to "next month", which uses perpetual calendar generator
- Other bad pages
  - other behavior damaging to crawler?
    - servers
  - spam content
    - use URLs from?

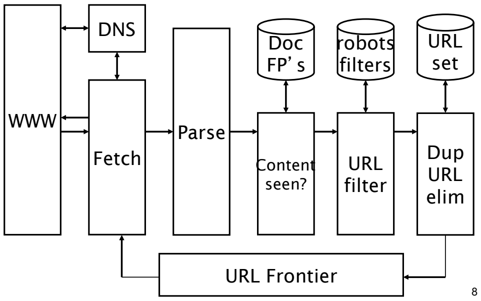
Robust crawlers must deal with black holes and other damaging behavior

Main Issues II

- Web is dynamic
  - continuous crawl
    - time to crawl “once” meaningful?
  - how mix crawl and re-crawl
    - priority of pages
- Social behavior
  - crawl only pages allowed by owner
    - robot exclusion protocol: *robots.txt*
  - not flood servers
    - expect many pages to visit on one server

from slides for Intro to IR, Sec. 20.2.1

Basic crawl architecture



Technical issues

- maintain one or **more** queues of URLs to be visited: URL frontier
  - order of URLs in queues?
    - FIFO = breadth first
    - LIFO = depth first
    - priority queues
- resolve hostname in URLs to get actual IP addresses – Domain Name Service servers (DNS lookup)
  - bottleneck:
    - servers distributed
    - can have high lookup latency

Technical issues continued

- To do large crawls must have multiple crawlers with multiple network connections (sockets) open and probably multiple queues
- large crawls generate large amount data
  - need fast access => main memory
  - cache: hold items most likely to use in main memory instead of
    - on disk
    - request from server

DNS lookup

- cache DNS map
  - large, local, in memory
  - hold most recently used mappings
- don’t want temporal locality of reference
  - be nice to servers (or else)
- prefetch DNS resolution for URLs on page when it parsed?
  - batch requests
  - put in cache
  - use when URL gets to head of queue
  - resolution stale?
- How “large” cache?
  - Problems?

(Near?) Duplicate pages

Has page been indexed already?

- mirror sites – different URLs, same page
  - bad: duplicate page in search results
  - worse?: add links from duplicate pages to queues
    - also mirrors?
  - mirrored pages may have slight differences
    - e.g. indicate which mirror they on
- other sources duplicates & near duplicates
  - eg …/spr14/cos435/ps1.html
    …/spr15/cos435/ps1.html

Removing (near) duplicates

- When apply?
  - while crawling versus for search results
  - crawling larger problem
  - search results demand faster results
- Duplicates versus near duplicates
  - same policy?
- How remove?
  - table of fingerprints or sketches of pages
  - fit in main memory?
  - if not, costs disk access per page crawler retrieves

Duplicate URL removal

IS URL in URL frontier?
Has URL already been visited? if not recrawling
⇒ Has URL ever been in URL frontier?

- Use:
  - canonical, fully specified URLs
  - canonical hostname provided by DNS
- *Visited?* hash table
  - hash canonical URL to entry
- *Visited?* table may be too large for MM

Caching *Visited?* table

- not temporal but "popularity" locality:
  - most popular URLs
  - most popular sites
    - *some* temporal locality within
- to exploit site-level locality need hash that brings pages on same site together:
  - two-level hash:
    - i. hash hostname and port
    - ii. hash path
- can use B+ tree, sorted on i then ii
  - if no entry for URL in tree, not visited

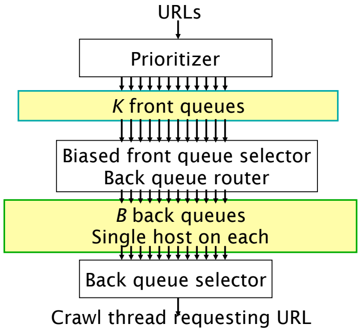
How integrate re-crawl?

- separate cycle for crawl of high priority pages?

- continuous crawl of all?
  - reinsert seed URLs in queue when fetch
  - also reinsert high-priority URLs when fetch
  - reinsert all URLs with varying priority when fetch

from slides for Intro to IR, Sec. 20.2.3

Mercator scheme



Mercator prioritizing

- Assigning priority
  - properties of page from previous visits
    - e.g. how often page change
  - class of pages
    - news, blogs, … high priority for recrawl
  - focused crawling
- Front queue for each priority: FIFO
- "Biased front queue selector"
  - implements policy
    - chooses which queue next

Mercator politeness enforcement: Back queues

- at any point each queue contains only URLs from one host
- additional information
  - table mapping host to queue
  - priority queue with entry for each queue/host: earliest time can next request from host
- priority queue min gives next queue to use for URL to fetch
  - wait until earliest allowed time to fetch

Maintaining back queues

- When a back queue emptied, remove URLs from front queues - putting in appropriate back queues until remove URL from new host
- put URL from new host in empty back queue
  - update host- back queue table
  - determine “earliest request time”
  - insert in priority queue

Crawling: Summary

- simple at high-level view
- “Devil in the details”
  - avoid duplication
  - minimize delays
    - avoid disk access when possible
  - be well-behaved
  - manage re-crawl versus discovery