Web Crawling
Crawling and Crawler

• Crawling (spidering): finding and downloading web pages automatically
  – Web crawler (spider): a program that downloads pages

• Challenges in crawling
  – Scale: tens of billions of pages on the web
  – Many new pages emerge or change at any time
  – No central control of all web pages – no easy way to find out how many pages in a website
Features of Crawlers

• Must-have features of a crawler
  – Robustness: should not fall into spider traps which generate an infinite number of pages
  – Politeness: respect the upper limit of visit frequencies of websites

• Should-have features of a crawler
  – Distributed, scalability, performance and efficiency, quality (biased on useful pages), freshness, extensibility
Crawling the Web

crawler.searchengine.com

www.bbc.co.uk

/www.bbc.co.uk/index.html

/www.bbc.co.uk/2005/story.html

www.cs.umass.edu

/index.html
/courses
/news

www.whitehouse.gov

/index.html
/about.html
/news.html

www.cnn.com

/index.html
/2006/09/story.html
/2003/04/story.html
A Simple Crawler

- Two major jobs
  - Downloading pages
  - Finding URLs

- Frontier – a request queue of URLs that the crawler needs to download
  - May be organized as a standard queue
  - Alternatively, more important pages can be moved to the front of the list so that they can be downloaded earlier

- Starting point: seeds – a set of URLs
  - Putting all those seed URLs into the frontier
  - Once a page is downloaded, it is parsed to find link tags
  - If the crawler finds a new URL that has not been seen before or the crawled version is outdated, add the URL into the frontier
procedure CRAWLER_THREAD(frontier)
while not frontier.done() do
    website ← frontier.nextSite()
    url ← website.nextURL()
    if website.permitsCrawl(url) then
        text ← retrieveURL(url)
        storeDocument(url, text)
        for each url in parse(text) do
            frontier.addURL(url)
        end for
    end if
    frontier.releaseSite(website)
end while
end procedure
Politeness

• Within website www.cs.sfu.ca, many pages contain many links to other pages in the same site
• Downloading many pages simultaneously from one website may overwhelm the server
Politeness Policies

- A reasonable web crawler should use only a tiny portion of the bandwidth of a website server – not fetching more than one page at a time
- Implementation: logically split the request queue into a single queue per web server – a server queue is open only if it has not been accessed within the specified politeness window
- Suppose a web crawler can fetch 100 pages per second, and the politeness policy dictates that it cannot fetch more than 1 page every 30 seconds from a server – we need URLs from at least 3,000 different servers to make the crawler reach its peak throughput
URL Frontier

Prioritizer

Biased front queue selector
Back queue router

B back queues
Single host on each

Back queue selector

Heap

F front queues
Robots.txt file

- A web server administrator can use a file /robots.txt to notify a crawler what is allowed and what is not

```
User-agent: *
Disallow: /private/
Disallow: /confidential/
Disallow: /other/
Allow: /other/public/

User-agent: FavoredCrawler
Disallow:
```
Detecting Updates

- If a web page is updated, the page should be crawled again.
- In HTTP, request HEAD returns only header information about the page but not the page itself.
  - A crawler can compare the date it received from the last GET request with the Last-Modified value from a HEAD request.

Client request:
```
HEAD /csinfo/people.html HTTP/1.1
Host: www.cs.umass.edu
```

Server response:
```
HTTP/1.1 200 OK
Date: Thu, 03 Apr 2008 05:17:54 GMT
Server: Apache/2.0.52 (CentOS)
Last-Modified: Fri, 04 Jan 2008 15:28:39 GMT
ETag: "239c33-2576-2a2837c0"
Accept-Ranges: bytes
Content-Length: 9590
Connection: close
Content-Type: text/html; charset=ISO-8859-1
```
The Freshness Measure

- **Freshness**: the fraction of the crawled pages that are currently fresh
  - A page is fresh if the crawl has the most recent copy
- **Discussion**: Consider a website X that changes its front page slightly every minute
  - How to optimize for freshness?
- Freshness may not be a good measure to use
Age

- A page has age 0 until it is changed, then its age grows until the page is crawled again.
- Comparison between age and freshness.

![Diagram showing the comparison between age and freshness](image)
Age Calculation

- Consider a page with change frequency $\lambda$ (i.e., the page is expected to change $\lambda$ times per day), the expected age of a page $t$ days after it was last crawled

$$ Age(\lambda, t) = \int_0^t P(\text{page changed at time } x)(t - x)dx $$

- Pages updates follow the Poison distribution

$$ Age(\lambda, t) = \int_0^t \lambda e^{-\lambda x} (t - x)dx $$

If a crawler crawls each page once a week, and each page in the collection has a mean update time of once a week, on average the crawl will be 2.6 days old

Expected age of a page with mean change frequency $\lambda = 1 / 7$
Why Optimizing Age?

• The second derivative of the age function is always positive
• The age is not only monotonically increasing, but its increase rate is always increasing
• The older a page gets, the more it costs you to NOT crawl it in terms of crawl staleness
• Optimizing age never results in the problem that freshness has (i.e., sometimes it is economical to not crawl a page at all)
Focus Crawling

- Vertical search: only search web pages about a specific topic (e.g., movies)
  - More accurate information
  - Less costly in computation since the database will be much smaller
- Crawl a full copy of the web, and throw out all unrelated pages: a most accurate but very expensive way
- Focused/topical crawling: a less expensive approach
  - Download only pages about a specific topic
  - Using a topic classifier (will be discussed later in this course)
Deep / Hidden Web

- Sites that are difficult for a crawler to find
  - Probably over 100 times larger than the traditionally indexed web
- Three major categories of sites in deep web
  - Private sites intentionally private – no incoming links or may require login
  - Form results – only accessible by entering data into a form, e.g., airline ticket queries
    - Hard to detect changes behind a form
  - Scripted pages – using JavaScript, Flash, or another client-side language in the web page
    - A crawler needs to execute the script – can slow down crawling significantly
- Deep web is different from dynamic pages
  - Wikis dynamically generates web pages but are easy to crawl
  - Private sites are static but cannot be crawled
Sitemaps

• If a website owner wants to tell a crawler some information to facilitate crawling, is there a way?

• Optional reference to a sitemap file in robots.txt file
  – A list of URLs
  – Data about those URLs like modification time and modification frequency
Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.company.com/</loc>
    <lastmod>2008-01-15</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.7</priority>
  </url>
  <url>
    <loc>http://www.company.com/items?item=truck</loc>
    <changefreq>weekly</changefreq>
  </url>
  <url>
    <loc>http://www.company.com/items?item=bicycle</loc>
    <changefreq>daily</changefreq>
  </url>
</urlset>
```

A more important directory
Crawling Documents and Emails

• In some other kinds of search engines, we need to crawl emails, word processing documents, presentations, or spreadsheets

• Challenges
  – Update speed is expected extremely fast – a user always wants to see the CURRENT version
  – Using operating system services to obtain update notification
  – A desktop crawler should not create another copy of the content – they are already stored in the computer
  – Rich file format in a computer system
  – Data privacy concern – user A should not be able to search user B’s data
Document Feeds

• A crawler can use feed information to find updates

• Pull: the subscriber checks periodically for new documents, like checking your mailbox
  – Examples: news feeds
  – RSS: Really Simple Syndication / RDF Site Summary / Rich Site Summary

• Push: alert the subscribers to the new document, like a telephone rings
Example

Time to live: the contents should only be cached for 60 minutes – should be considered stale after 60 minutes

```xml
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
    <title>Search Engine News</title>
    <link>http://www.search-engine-news.org/</link>
    <description>News about search engines.</description>
    <language>en-us</language>
    <pubDate>Tue, 19 Jun 2008 05:17:00 GMT</pubDate>
    <ttl>60</ttl>

    <item>
      <title>Upcoming SIGIR Conference</title>
      <link>http://www.sigir.org/conference</link>
      <description>The annual SIGIR conference is coming! Mark your calendars and check for cheap flights.</description>
      <pubDate>Tue, 05 Jun 2008 09:50:11 GMT</pubDate>
      <guid>http://search-engine-news.org#500</guid>
    </item>

  </channel>
</rss>
```
Putting Crawler Closer to Sites

- Crawling by Multiple Computers
- Putting the crawler closer to the sites it crawls
- Shortening network connections to achieve higher throughput
  - Suppose a crawler on average can copy 50 pages per second, i.e., transferring a page takes 20ms.
  - If it takes 80ms for a site to start sending data, in total 5 seconds waiting time in transferring 50 pages
  - If the latency time is 500ms, in total it needs 600ms x 50 = 30 seconds to transfer 50 pages
Crawling by Multiple Computers

- Reducing the number of sites the crawler has to remember
  - To remove duplicates, a crawler needs to maintain the frontier in main memory
  - The larger the throughput, the less sites need to keep
- Using more computational sources
  - CPU time for parsing and network bandwidth for crawling
Distributed Crawlers

- Using many URL queues
Assigning URLs to Computers

- [Discussion] If there are many computers, when a new URL is met which has not been crawled in the local computer, how can we know whether it is crawled in other computers?

- Assigning URLs from the same domain to the same computer – only need to check the queue in one computer
  - Implementation: computing the hash value of a URL using only the hostname
DNS Resolution

• DNS (Domain Name Service) resolution/lookup: translating a URL to an IP address (e.g., www.wikipedia.org is translated to 207.142.131.248)
  – Often done by communication among multiple distributed domain name servers
  – Service time: multiple seconds or even longer
  – Typically, a machine can only request one DNS lookup at a time

• DNS solution for web search: caching frequently used URLs on DNS servers
  – It does not work well for crawling due to the politeness constraint

• Most web crawlers implement their own DNS resolver
  – Each thread can request one DNS lookup at a time, but a computer can have multiple threads
The Conversion Problem

- Text is stored in many different formats
- A search engine needs to convert all text into a tagged text format like HTML or XML

An example of text in the TREC Web compound document format:

```
<DOC>
<DOCTYPE WTX001-B01-10></DOCTYPE>
<DOCTYPE>
http://www.example.com/test.html 204 244.59.33 19970101013145 text/html 440
HTTP/1.0 200 OK
Date: Wed, 01 Jan 1997 01:21:13 GMT
Server: Apache/1.0.3
Content-type: text/html
Content-length: 270
Last-modified: Mon, 25 Nov 1996 05:31:24 GMT
</DOCTYPE>
<DOC>
<TITLE>Tropical Fish Store</TITLE>
Coming soon!
</DOC>
</DOC>

<DOC>
<DOCTYPE WTX001-B01-109></DOCTYPE>
<DOCTYPE>
http://www.example.com/fish.html 204 244.59.33 19970101013149 text/html 440
HTTP/1.0 200 OK
Date: Wed, 01 Jan 1997 01:21:19 GMT
Server: Apache/1.0.3
Content-type: text/html
Content-length: 270
Last-modified: Mon, 25 Nov 1996 05:31:24 GMT
</DOCTYPE>
<DOC>
<TITLE>Fish Information</TITLE>
This page will soon contain interesting information about tropical fish.
</DOC>
```
Character Encodings – Unicode

- Unicode: a single mapping from numbers to glyphs
- Implementations of Unicode – mapping bits to numbers in Unicode
  - UTF-8 encodes as short as possible, while UTF-32 uses 4 bytes per glyph
- Often use UTF-32 as internal encoding but UTF-8 to store text on disk

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 127</td>
<td>0 - 7F 0xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>128 - 2047</td>
<td>80 - 7FF 110xxxxx 10xxxxx</td>
<td></td>
</tr>
<tr>
<td>2048 - 55295</td>
<td>800 - D7FF 1110xxxx 10xxxxx 10xxxxx</td>
<td>Undefined</td>
</tr>
<tr>
<td>55296 - 57343</td>
<td>D800 - DFFF</td>
<td></td>
</tr>
<tr>
<td>57344 - 65535</td>
<td>E000 - FFFF</td>
<td></td>
</tr>
<tr>
<td>65536 - 1114111</td>
<td>10000 - 10FFFF</td>
<td>11110xxxx 10xxxxx 10xxxxx 10xxxxx</td>
</tr>
</tbody>
</table>
Storing Documents

- In many cases, documents need to be stored in a search engine
  - Generating snippets, information extraction, managing updates, …
- Major search engines do not use conventional relational databases to store documents
  - The volume of documents data overwhelms traditional database systems
- Major operation: looking up a document using a URL
  - Solution: using hashing
Compression and Large Files

• Should we store each web page using an individual file?

• Many web pages are small in size – a page can be transferred in a millisecond
  – The hard disk seek time might be ten milliseconds
  – Reading many small files needs a substantial overhead to open them

• Solution: storing many documents in a single file so that transferring the file contents takes much more time than seeking to the beginning
  – A good size choice might be in the hundreds of megabytes
BigTable at Google

• A working distributed database system used internally at Google
• A database contains only one table split into small pieces called tablets, which are served by thousands of machines
Queries in BigTable

- No query language – no complex queries
- Only row-level transactions – can scale up to very large database sizes
  - Any changes to a BigTable tablet are recorded to a transaction log stored in a shared file system – to be used for recovery in the case of crashes
- BigTable stores data in immutable (i.e., unchangeable) files, and periodically merges files to reduce the total number of disk files

A BigTable row

www.example.com →

<table>
<thead>
<tr>
<th>anchor:other.com</th>
<th>anchor:null.com</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>title</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>document text</th>
<th>example</th>
<th>click here</th>
<th>example site</th>
</tr>
</thead>
</table>
Columns and Column Groups

- Columns are divided into groups
- BigTable can have a huge number of columns per row
  - Not all rows have the same columns
  - All rows have the same column groups
- Rows are partitioned into tablets based on their row keys
  - To look up a particular row, use a hashing to identify the tablet holding the row
Duplicate Detection

• About 30% of the web pages in a large crawl are exact or near-duplicates
  – Different versions of a page
  – Plagiarism and spam
  – Using URL alias and mirror sites

• Exact duplicate detection using checksums
  – Cyclic redundancy check (CRC) considers the positions of the bytes

Checksum:

<table>
<thead>
<tr>
<th>Tropical</th>
<th>fish</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>54 72 6F 70 69 63 61 6C 20 66 69 73 68 508</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Near-Duplicate Detection

- If the similarity between two documents is over a threshold, they are considered near-duplicate
  - $D_1$ could be regarded as a near-duplicate of $D_2$ if more than 90% of the words in the documents are the same
- Search scenario: given a document $D$, find near-duplicates of $D$
  - Complexity $O(N)$ if there are $N$ documents
- Discovery scenario: find all pairs of near-duplicates documents in the database
  - Complexity $O(N^2)$
Central Ideas

• Extract some features for each document
• Compare features of documents instead of the documents themselves
  – Hopefully, the feature comparison cost is much lower
• Organize features of documents in a way that can be retrieved efficiently
  – Index, such as a sorted list
  – Hashing
Fingerprints Generation

- A document is parsed into words
  - Non-word content, such as punctuation, HTML tags, and additional whitespace, is removed
- The words are grouped into contiguous n-grams for some n
- Some of the n-grams are selected to represent the document
- The selected n-grams are hashed to improve retrieval efficiency and further reduce the size of the representation
  - 0 mod p: select all n-grams whose value of modulo p is 0, where p is a parameter
Example – N-gram Hashing

Original text
Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

3-grams
tropical fish include, fish include fish, include fish found, fish found in, found in tropical, in tropical environments, tropical environments around, environments around the, around the world, the world including, world including both, including both freshwater, both freshwater and, freshwater and salt, and salt water, salt water species

Hash values
938 664 463 822 492 798 78 969 143 236 913 908 694 553 870 779

Selected hash values using 0 mod 4
664 492 236 908
Detection Using Fingerprints

• Find all documents whose fingerprints match that of D

• How to make the detection efficient?
  – The hash values are stored in an inverted index
  – Treat each hash value as a virtual keyword
Shingling

- Use every n-gram as a shingle
- Let $S(d_1)$ and $S(d_2)$ be the sets of shingles in documents $d_1$ and $d_2$, respectively
- Jaccard similarity
  \[ J(S(d_1), S(d_2)) = \frac{|S(d_1) \cap S(d_2)|}{|S(d_1) \cup S(d_2)|} \]
  - However, computing Jaccard similarity is costly
- Compute the hashing value of each shingle
- Let $\pi$ be a random permutation, and $x^{\pi_1}$ and $x^{\pi_2}$ be the minimum values of $S(d_1)$ and $S(d_2)$ in the permutation
- \[ J(S(d_1), S(d_2)) = P(x^{\pi_1} = x^{\pi_2}) \]
  - $P(x^{\pi_1} = x^{\pi_2})$ can be approximated by tries
Shingling – Example

• Use the frequencies of n-grams as the fingerprint
• Check the probability that two documents have the smallest shingle

\[
\begin{align*}
H(d_1) & : 0, 1, 2, 3, 4, 2^{64} - 1 \\
H(d_2) & : 0, 1, 2, 3, 4, 2^{64} - 1 \\
H(d_1) \text{ and } \Pi(d_1) & : 0, 3, 1, 4, 2, 2^{64} - 1 \\
H(d_2) \text{ and } \Pi(d_2) & : 0, 3, 1, 4, 2, 2^{64} - 1 \\
\Pi(d_1) & : 0, 3, 1, 4, 2, 2^{64} - 1 \\
\Pi(d_2) & : 0, 3, 1, 4, 2, 2^{64} - 1 \\
x_1^\pi & : 0, 3, 2^{64} - 1 \\
x_2^\pi & : 0, 3, 2^{64} - 1 \\
\end{align*}
\]
Simhash Fingerprint – Generation

- Idea: frequencies of words in a document may capture the features of the document better
- Simhash fingerprint generation
  - Process the document into a set of features with associated weights, e.g., the frequencies
  - Generate a unique hash value with b bits for each word
  - In b-dimensional vector V, update the components of the vector by adding the weight for a word to every component for which the corresponding bit in the word’s hash value is 1, and subtracting the weight if the value is 0
  - After all words are processed, generate a b-bit fingerprint by setting the i-th bit to 1 if the i-th component of V is positive, or 0 otherwise
Simhash Fingerprint – Detection

• Idea: frequencies of words in a document may capture the features of the document better

• Detection
  – Example: if the fingerprints have 384 bits, and a page P agrees on more than 372 bits in the fingerprint to the target page, P is considered a near-duplicate
Example – Simhash

Original text
Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

Words with weights

<table>
<thead>
<tr>
<th>Word</th>
<th>Weight</th>
<th>Hash Value</th>
<th>Word</th>
<th>Weight</th>
<th>Hash Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tropical</td>
<td>2</td>
<td>01100001</td>
<td>include</td>
<td>1</td>
<td>10101011</td>
</tr>
<tr>
<td>fish</td>
<td>2</td>
<td>10101011</td>
<td>around</td>
<td>1</td>
<td>11100110</td>
</tr>
<tr>
<td>found</td>
<td>1</td>
<td>00111110</td>
<td>environments</td>
<td>1</td>
<td>00101101</td>
</tr>
<tr>
<td>environments</td>
<td>1</td>
<td>00101101</td>
<td>world</td>
<td>1</td>
<td>10001011</td>
</tr>
<tr>
<td>including</td>
<td>1</td>
<td>00101010</td>
<td>both</td>
<td>1</td>
<td>10101110</td>
</tr>
<tr>
<td>freshwater</td>
<td>1</td>
<td>00111111</td>
<td>salt</td>
<td>1</td>
<td>10110101</td>
</tr>
<tr>
<td>species</td>
<td>1</td>
<td>11101110</td>
<td>water</td>
<td>1</td>
<td>00100101</td>
</tr>
</tbody>
</table>

8 bit hash values

Vector V formed by summing weights

```
 1 -5 9 -9 3 1 3 3
```

8-bit fingerprint formed from V

```
1 0 1 0 1 1 1 1
```
Noise

- The main content of a page may take only a small display area of the page
- Noise: banners, advertisements, images, general navigation links, services, copyright, and miscellaneous information
- How can we remove noise from web pages automatically?
A Heuristic

- There are less HTML tags in the text of the main content of typical web pages than there is in the additional material
- Using cumulative distribution of tags
- Find a plateau in the middle of the distribution
Algorithm

- Represent a web page as a sequence of bits
  - \( b_n = 1 \) if the \( n \)-th token is a tag
  - \( b_n = 0 \) otherwise
- Find values of \( i \) and \( j \) to maximize the number of tags below \( i \) and above \( j \) and the number of non-tag tokens between \( i \) and \( j \)
- Let \( N \) be the number of tokens in the page

\[
\max \sum_{n=0}^{i-1} b_n + \sum_{n=i}^{j} (1 - b_n) + \sum_{n=j+1}^{N-1} b_n
\]
Using DOM Representation

- A browser creates a Document Object Model (DOM) representation when displaying a web page in HTML.
- Navigate the DOM tree recursively to remove and modify nodes in the tree, leave only content.
Extensions

- Use visual features (e.g., the position of the block, the size of the font used, the background and font colors, the presence of separators) to define blocks of information that would be apparent to users in the displayed web page.
- Using DOM tree, multiple content blocks can be identified and relative importance of each block can be used to produce a more effective representation.
  - Train a classifier that will assign an importance category based on visual and content features.
Summary

- Crawling and basic crawler architecture
- Politeness and frontier
- Handling updates
- Deep web
- Sitemap
- Document feeds
- Distributed crawlers
- Conversion
- Storing documents and BigTable
- Near-duplicate detection
- Removing noise
To-Do List

• Read Chapter 3
• Between the shingling method and the simhash method, which one is more accurate? Why?
• Web pages often contain ads. How can we detect web pages containing duplicate content but different ads?