Detecting Duplicates

- Duplicate and near-duplicate documents occur in many situations
  - Copies, versions, plagiarism, spam, mirror sites
  - 30% of the web pages in a large crawl are exact or near duplicates of pages in the other 70%
- Duplicates consume significant resources during crawling, indexing, and search
  - Little value to most users
Duplicate Detection

• **Exact** duplicate detection is relatively easy

• **Checksum** techniques
  – A checksum is a value that is computed based on the content of the document
    • e.g., sum of the bytes in the document file

    \[
    \text{Tropical fish Sum} \quad 54 \quad 72 \quad 6F \quad 70 \quad 69 \quad 63 \quad 61 \quad 6C \quad 20 \quad 66 \quad 69 \quad 73 \quad 68 \quad 508
    \]
  – Possible for files with different text to have same checksum

• Functions such as a **cyclic redundancy check** (CRC), have been developed that consider the positions of the bytes

Near-Duplicate Detection

• More challenging task
  – Are web pages with same text context but different advertising or format near-duplicates?

• A near-duplicate document is defined using a threshold value for some similarity measure between pairs of documents
  – e.g., document $D1$ is a near-duplicate of document $D2$ if more than 90% of the words in the documents are the same
Computing Similarity

- Features:
  - Segments of a document (natural or artificial breakpoints)
  - Shingles (Word N-Grams)
    - *a rose is a rose is a rose* → 4-grams are
      - a_rose_is_a
      - rose_is_a_rose
      - is_a_rose_is
      - a_rose_is_a
  - Similarity Measure between two docs (= sets of shingles)
    - Jaccard coefficient: \( \frac{\text{Size of Intersection}}{\text{Size of Union}} \)

Shingles + Set Intersection

- Computing exact set intersection of shingles between all pairs of documents is expensive
- Approximate using a cleverly chosen subset of shingles from each (a sketch)
- Estimate \( \frac{\text{size of intersection}}{\text{size of union}} \) based on a short sketch

![Diagram](https://via.placeholder.com/150)

© 2017 Christopher W. Clifton
Sketch of a document

- Create a “sketch vector” (of size ~200) for each document
  - Documents that share $\geq t$ (say 80%) corresponding vector elements are deemed near duplicates
  - For doc $D$, sketch$D[i]$ is as follows:
    - Let $f$ map all shingles in the universe to $1..2^m$ (e.g., $f =$ fingerprinting)
    - Let $\pi_i$ be a random permutation on $1..2^m$
    - Pick $\text{MIN}\{\pi_i(f(s))\}$ over all shingles $s$ in $D$

Computing Sketch[i] for Doc1

Document 1

- Start with 64-bit $f$ (shingles)
- Permute on the number line with $\pi_i$
- Pick the min value
Test if Doc1.Sketch[i] = Doc2.Sketch[i]

Are these equal?

Test for 200 random permutations: \( \pi_1, \pi_2, \ldots, \pi_{200} \)

Final notes

- Shingling is a *randomized algorithm*
  - Our analysis did not presume any probability model on the inputs
  - It will give us the right (wrong) answer with some probability on *any input*
- We've described how to detect near duplication in a pair of documents
- In "real life" we'll have to concurrently look at many pairs
  - See text book for details
Removing Noise

- Many web pages contain text, links, and pictures that are not directly related to the main content of the page.
- This additional material is mostly noise that could negatively affect the ranking of the page.
- Techniques have been developed to detect the content blocks in a web page.
  - Non-content material is either ignored or reduced in importance in the indexing process.

Noise Example

Content block
Finding Content Blocks

- Cumulative distribution of tags in the example web page

- Main text content of the page corresponds to the “plateau” in the middle of the distribution

Finding Content Blocks

- Represent a web page as a sequence of bits, where $b_n = 1$ indicates that the $n$th token is a tag

- Optimization problem where we find values of $i$ and $j$ to maximize both the number of tags below $i$ and above $j$ and the number of non-tag tokens between $i$ and $j$

- i.e., maximize

$$\sum_{n=0}^{i-1} b_n + \sum_{n=i}^{j} (1 - b_n) + \sum_{n=j+1}^{N-1} b_n$$
Finding Content Blocks

- Other approaches use DOM structure and visual (layout) features.

Deep Web

- Sites that are difficult for a crawler to find are collectively referred to as the deep (or hidden) Web
  - much larger than conventional Web
- Three broad categories:
  - private sites
    - no incoming links, or may require log in with a valid account
  - form results
    - sites that can be reached only after entering some data into a form
  - scripted pages
    - pages that use JavaScript, Flash, or another client-side language to generate links
Sitemaps

- Sitemaps contain lists of URLs and data about those URLs, such as modification time and modification frequency
- Generated by web server administrators
- Tells crawler about pages it might not otherwise find
- Gives crawler a hint about when to check a page for changes

Sitemap 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>
```
Distributed Crawling

- Three reasons to use multiple computers for crawling:
  - Helps to put the crawler closer to the sites it crawls
  - Reduces the number of sites the crawler has to remember
  - Reduces computing resources required
- Distributed crawler uses a hash function to assign URLs to crawling computers
  - hash function should be computed on the host part of each URL

Desktop Crawls

- Used for desktop search and enterprise search
- Differences to web crawling:
  - Much easier to find the data
  - Responding quickly to updates is more important
  - Must be conservative in terms of disk and CPU usage
  - Many different document formats
  - Data privacy very important
Document Feeds

- Many documents are published
  - created at a fixed time and rarely updated again
  - e.g., news articles, blog posts, press releases, email
- Published documents from a single source can be ordered in a sequence called a document feed
  - new documents found by examining the end of the feed

Document Feeds

- Two types:
  - A push feed alerts the subscriber to new documents
  - A pull feed requires the subscriber to check periodically for new documents
- Most common format for pull feeds is called RSS
  - Really Simple Syndication, RDF Site Summary, Rich Site Summary, or ...
RSS Example

```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>
```
RSS

- **ttl** tag (time to live)
  - amount of time (in minutes) contents should be cached
- RSS feeds are accessed like web pages
  - using HTTP GET requests to web servers that host them
- Easy for crawlers to parse
- Easy to find new information

Conversion

- Text is stored in hundreds of incompatible file formats
  - e.g., raw text, RTF, HTML, XML, Microsoft Word, ODF, PDF
- Other types of files also important
  - e.g., PowerPoint, Excel
- Typically use a conversion tool
  - converts the document content into a tagged text format such as HTML or XML
  - retains some of the important formatting information
Character Encoding

• A character encoding is a mapping between bits and glyphs
  – i.e., getting from bits in a file to characters on a screen
  – Can be a major source of incompatibility
• ASCII is basic character encoding scheme for English
  – encodes 128 letters, numbers, special characters, and control characters in 7 bits, extended with an extra bit for storage in bytes

Character Encoding

• Other languages can have many more glyphs
  – e.g., Chinese has more than 40,000 characters, with over 3,000 in common use
• Many languages have multiple encoding schemes
  – e.g., CJK (Chinese-Japanese-Korean) family of East Asian languages, Hindi, Arabic
  – must specify encoding
  – can’t have multiple languages in one file
• Unicode developed to address encoding problems
Unicode

- Single mapping from numbers to glyphs that attempts to include all glyphs in common use in all known languages
- Unicode is a mapping between numbers and glyphs
  - does not uniquely specify bits to glyph mapping!
  - e.g., UTF-8, UTF-16, UTF-32

Unicode

- Proliferation of encodings comes from a need for compatibility and to save space
  - UTF-8 uses one byte for English (ASCII), as many as 4 bytes for some traditional Chinese characters
  - variable length encoding, more difficult to do string operations
  - UTF-32 uses 4 bytes for every character
- Many applications use UTF-32 for internal text encoding (fast random lookup) and UTF-8 for disk storage (less space)
Unicode

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

- e.g., Greek letter pi (π) is Unicode symbol number 960
- In binary, 00000011 11000000 (3C0 in hexadecimal)
- Final encoding is 11001111 10000000 (CF80 in hexadecimal)

Storing the Documents

- Many reasons to store converted document text
  - saves crawling time when page is not updated
  - provides efficient access to text for snippet generation, information extraction, etc.
- Database systems can provide document storage for some applications
  - web search engines use customized document storage systems
Storing the Documents

- Requirements for document storage system:
  - Random access
    - request the content of a document based on its URL
    - hash function based on URL is typical
  - Compression and large files
    - reducing storage requirements and efficient access
  - Update
    - handling large volumes of new and modified documents
    - adding new anchor text

Compression

- Text is highly redundant (or predictable)
- Compression techniques exploit this redundancy to make files smaller without losing any of the content
- Compression of indexes covered later
- Popular algorithms can compress HTML and XML text by 80%
  - e.g., DEFLATE (zip, gzip) and LZW (UNIX compress, PDF)
  - may compress large files in blocks to make access faster
BigTable

- Google’s document storage system
  - Customized for storing, finding, and updating web pages
  - Handles large collection sizes using inexpensive computers

![Diagram of BigTable structure]

BigTable

- No query language, no complex queries to optimize
- Only row-level transactions
- Tablets are stored in a replicated file system that is accessible by all BigTable servers
- Any changes to a BigTable tablet are recorded to a transaction log, which is also stored in a shared file system
- If any tablet server crashes, another server can immediately read the tablet data and transaction log from the file system and take over
BigTable

- Logically organized into rows
- A row stores data for a single web page

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

- Combination of a row key, a column key, and a timestamp point to a single *cell* in the row

---

BigTable

- BigTable can have a huge number of columns per row
  - all rows have the same column groups
  - not all rows have the same columns
  - important for reducing disk reads to access document data
- Rows are partitioned into tablets based on their row keys
  - simplifies determining which server is appropriate