If we didn’t have search …

- Contains all books with
  - 25 unique characters
  - 80 characters per line
  - 40 lines per page
  - 410 pages
  - $410 \times 40 \times 80 = 1,312,000$ characters
  - $2^{1312000}$ books
- Would contain any book
  imaginable and every book that
  ever existed (within those page
  and character bounds)

All information = Zero information
**SEARCH, QUERY, RETRIEVAL**

---

**Search, Query & Retrieval**

- **Search**: the goal/aim of the user
- **Query**: the expression of a search
- **Retrieval**: the machine method to “solve” a query

... roughly speaking

---

**Retrieval**

1. Machine has a bunch of information resources of some sort (let’s call it a set \( I \))
   - e.g., documents, movie pages, actor descriptions
2. A user search wants to find some subset of \( I \)
   - e.g., Irish actors, documents about Hadoop
3. User expresses search criteria as a query \( Q \)
   - e.g., “Irish actors”, “hadoop”, “SELECT ?movie ...”
4. Retrieval engine returns results: \( R \) is a minimal subset of \( I \) relevant to \( Q \)
5. Results \( R \) may be ordered by a ranking
   - e.g., by most famous Irish actors

---

**Data Retrieval**

- Retrieval over “structured data”
- Typical of databases
  - \( I \) is a dataset, e.g., a set of relations
  - \( Q \) is a structured query, e.g., SQL
  - \( R \) is a list of tuples, possibly ordered

```
SELECT * FROM actor WHERE country = "Ireland" ORDER BY earnings;
```

---

**Information Retrieval**

- Retrieval over “unstructured data” or textual data
- Typical of web search
  - \( I \) is a set of text documents, e.g., web pages
  - \( Q \) is a keyword query
  - \( R \) is a list of documents, e.g., relevant pages

---

**WEB SEARCH/RETRIEVAL**

*most famous Irish actors*
Google Architecture (ca. 1998)

**Information Retrieval**

- Crawling
- Inverted indexing
- PageRank

---

**INFORMATION RETRIEVAL: CRAWLING**

**How does Google know about the Web?**

---

**Crawling**

- Download the Web.

```java
 crawl(list seedUrls)
   frontier_i = seedUrls
   new list frontier_i+1
   for url : frontier_i
     page = downloadPage(url)
     frontier_i+1.addAll(extractUrls(page))
   store(page)
   i++
```

---

**Crawling: Avoid Cycles**

- Download the Web.

```java
 crawl(list seedUrls)
   frontier_i = seedUrls
   new set urlsSeen
   while(frontier_i is not empty)
     new list frontier_i+1
     for url : frontier_i
       page = downloadPage(url)
       urlsSeen.add(url)
       frontier_i+1.addAll(extractUrls(page) .removeAll(urlsSeen))
       store(page)
     i++
```

---

What’s missing from this code?

What about the performance of this code?
Crawling: Catering for Slow Network

- Majority of the time spent will be spent waiting for connection
- Disk and CPU of crawling machine barely occupied
- Bandwidth will not be maximised (stop / start)

Crawling: Multi-threading Important

```
crawl(list seedUrls)
frontier_i = seedUrls
new set urlsSeen
while(!frontier_i.isEmpty())
    new list frontier_i+1
    new list threads
    for url : frontier_i
        thread = new DownloadPageThread.run(url,urlsSeen,fronter_i+1)
        threads.add(thread)
        threads.poll()
i++
```

DownloadPageThread: run(url,urlsSeen,frontier_i+1)
```
    page = downloadPage(url)
synchronized: urlsSeen.add(url)
synchronized: frontier_i+1.addAll(extractUrls(page).removeAll(urlsSeen))
synchronized: store(page)
```

Crawling: Important to be Polite!

- (Distributed) Denial of Server Attack: (D)DoS

Crawling: Avoid (D)DoSing

- But more likely your IP range will be banned by the web-site (DoS attack)

Crawling: Web-site Scheduler

```
crawl(list seedUrls)
frontier_i = seedUrls
new set urlsSeen
while(!frontier_i.isEmpty())
    new list frontier_i+1
    new list threads
    for url : schedule(frontier_i)
        thread = new DownloadPageThread.run(url,urlsSeen,fronter_i+1)
        threads.add(thread)
        threads.poll()
i++
```

DownloadPageThread: run(url,urlsSeen,frontier_i+1)
```
    page = downloadPage(url)
synchronized: urlsSeen.add(url)
synchronized: frontier_i+1.addAll(extractUrls(page).removeAll(urlsSeen))
synchronized: store(page)
```
Robots Exclusion Protocol

http://website.com/robots.txt

User-agent: *
Disallow: / No bots allowed on the website.

User-agent: *
Disallow: /user/
Disallow: /main/login.html
No bots allowed in /user/ sub-folder or login page.

User-agent: googlebot
Disallow: /
Tell the googlebot to only crawl a page from this host no more than once every 10 seconds.

User-agent: *
Disallow: /user/
Allow: /public/
Ban everything but the /public/ folder for all agents

User-agent: *
Sitemap: http://example.com/main/sitemap.xml
Tell user-agents about your site-map

Robots Exclusion Protocol (non-standard)

User-agent: googlebot
Crawl-delay: 10
Tell the googlebot to only crawl a page from this host no more than once every 10 seconds.

User-agent: *
Disallow: /
Tell everything about the robots.txt file

User-agent: *
Sitemap: http://example.com/main/sitemap.xml
Tell user-agents about your site-map

Site-Map

<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://example.com/1/</loc>
    <lastmod>2009-09-24T19:49:50Z</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.9</priority>
  </url>
  <url>
    <loc>http://example.com/2/</loc>
    <lastmod>2009-09-24T19:49:50Z</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.9</priority>
  </url>
  <url>
    <loc>http://example.com/3/</loc>
    <lastmod>2009-09-24T19:49:50Z</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.9</priority>
  </url>
  <url>
    <loc>http://example.com/4/</loc>
    <lastmod>2009-09-24T19:49:50Z</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.9</priority>
  </url>
  <url>
    <loc>http://example.com/5/</loc>
    <lastmod>2009-09-24T19:49:50Z</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.9</priority>
  </url>
</urlset>

Crawling: Important Points

- **Seed-list:** Entry point for crawling
- **Frontier:** Extract links from current pages for next round
- **Threading:** Keep machines busy; mitigate waits for connection
- **Seen-list:** Avoid cycles
- **Politeness:** Don’t annoy web-sites
  - Set a politeness delay between crawling pages on the same web-site
  - Stick to what’s stated in the robots.txt file
  - Check for a site-map

Crawling: Distribution

- Similar benefits to multi-threading

  How might we implement a distributed crawler?

  for url : frontier_i-1
  map(url,count)

- Local frontier and seen-URL list!

  What will be the bottleneck as machines increase?

Crawling: Other Options

- **Breadth-first:** As per the pseudo-code, crawl in rounds
  - Extract one-hop from seed URLs ...
  - Extract n-hop from seed URLs
- **Depth-first:** Follow first link in first page, first link in second page, etc.
- **Best/topic-first:** Rank the URLs according to topic, number of in-links, etc.
- **Hybrid:** A mix of strategies
Crawling: Inaccessible (Bow-Tie)  


Crawling: Inaccessible (Deep-Web)  

- **Deep-web:**
  - Dynamically-generated content
  - Password protected / firewalled
  - Dark Web

Apache Nutch

- Open-source crawling framework!
- Compatible with Hadoop!

[Image: https://nutch.apache.org/]

INFORMATION RETRIEVAL: INVERTED-INDEXING

Inverted Index

- **Inverted Index:** A map from words to documents
  - “Inverted” because usually documents map to words
  - At the core of all keyword search applications

Inverted Index: Example

Fruitvale Station is a 2013 American drama film written and directed by Ryan Coogler.

<table>
<thead>
<tr>
<th>Inverted index:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fruitvale Station</td>
</tr>
</tbody>
</table>
Inverted Index: Example Search

**american drama**

- **AND**: Posting lists intersected (optimised!)
- **OR**: Posting lists unioned
- **PHRASE**: AND + check locations

<table>
<thead>
<tr>
<th>Word</th>
<th>Posting Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>[1, 21, 96, 103] ...</td>
</tr>
<tr>
<td>american</td>
<td>[2, 9, 12] ...</td>
</tr>
<tr>
<td>and</td>
<td>[2, 87, 119] ...</td>
</tr>
<tr>
<td>by</td>
<td>[1, 30, 157] ...</td>
</tr>
<tr>
<td>directed</td>
<td>[2, 61, 212] ...</td>
</tr>
<tr>
<td>drama</td>
<td>[1, 38, 87, 146] ...</td>
</tr>
</tbody>
</table>

Inverted Index Flavours

- **Record-level inverted index**: Maps words to documents without positional information
- **Word-level inverted index**: Additionally maps words with positional information

Inverted Index: Word Normalisation

**drama america**

- **Word normalisation**: grammar removal, case, lemmatisation, accents, etc.
- **Query side and/or index side**

<table>
<thead>
<tr>
<th>Term List</th>
<th>Posting Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>[1, 21, 96, 103] ...</td>
</tr>
<tr>
<td>american</td>
<td>[2, 9, 12] ...</td>
</tr>
<tr>
<td>and</td>
<td>[2, 87, 119] ...</td>
</tr>
<tr>
<td>by</td>
<td>[1, 30, 157] ...</td>
</tr>
<tr>
<td>directed</td>
<td>[2, 61, 212] ...</td>
</tr>
<tr>
<td>drama</td>
<td>[1, 38, 87, 146] ...</td>
</tr>
</tbody>
</table>

Inverted Index: Space

- **Not so many unique words** ...
  - but lots of new proper nouns
  - Heap’s law:
    - \( \text{UW}(n) \approx Kn^\beta \)
    - English text
      - \( K \approx 10 \)
      - \( \beta \approx 0.6 \)

Inverted Index: Common Words

- **Many occurrences of few words**
  - Few occurrences of many words
  - Zipf’s law
    - In English text:
      - “the” 7%
      - “of” 3.5%
      - “and” 2.7%
      - 135 words cover half of all occurrences

Zipf’s law: the most popular word will occur twice as often as the second most popular word, thrice as often as the third most popular word, \( n \) times as often as the \( n \)-most popular word.
Inverted Index: Common Words

- Expect long posting lists for common words
- Also expect more queries for common words

Inverted Index: Common Words

- Perhaps implement stop-words?
  - Most common words contain least information
- Perhaps implement block-addressing?

Block 1

<table>
<thead>
<tr>
<th>Term List</th>
<th>Posting Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1,2,3,4,5,6,7</td>
</tr>
<tr>
<td>american</td>
<td>1,2,3,4,5,6,7</td>
</tr>
<tr>
<td>and</td>
<td>1,2,3,4,5,6,7</td>
</tr>
<tr>
<td>by</td>
<td>1,2,3,4,5,6,7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

What is the effect on phrase search?

Term List Posting Lists

country (1), (2), (3), (4), (6), (7), ...

Term List Posting Lists

country (1–4), (6–7), ...

Search Implementation

- Vocabulary keys:
  - Hashing: O(1) lookups (assuming good hashing)
    - no range queries
    - relatively easy to update (though rehashing expensive!)
  - Sorting/B-Tree: O(log(u)) lookups, u unique words
    - range queries
    - tricky to update (standard methods for B-trees)
  - Tries: O(l) lookups, l length of the word
    - range queries, compressed, auto-completion!
    - referencing becomes tricky (esp. on disk)

Memory Sizes

- Vocabulary keys:
  - Often will fit in memory!
  - Posting lists may be kept on disk
    - (hot regions cached)
- The long-tail of search:

Compression techniques: High Level

- Interval indexing
  - Example for record-level indexing
  - Could be applied for block-level indexing

<table>
<thead>
<tr>
<th>Term List</th>
<th>Posting Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>country</td>
<td>1,2,3,4,6,7,8</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term List</th>
<th>Posting Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>country</td>
<td>1-4,6-7,8,9,</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Compression techniques: Low Level

- Variable length coding: bit-level techniques
- For example, Elias encoding
  - Assumes many small numbers

<table>
<thead>
<tr>
<th>Integer to encode</th>
<th>Encoding values in bytes</th>
<th>Bytes needed</th>
<th>Most significant numbers</th>
<th>Least significant code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>101</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>0</td>
<td>00</td>
<td>1000</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>0</td>
<td>001</td>
<td>11000</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>0</td>
<td>010</td>
<td>110010</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>0</td>
<td>110</td>
<td>110011</td>
</tr>
<tr>
<td>8</td>
<td>111</td>
<td>0</td>
<td>000</td>
<td>110000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

For example, "0100001100110000011001" = <1,2,3,4,8,5>
Other Optimisations

- **Top-Doc**: Order posting lists to give likely “top documents” first: good for top-k results

- **Selectivity**: Load the posting lists for the most rare keywords first; apply thresholds

- **Sharding**: Distribute an index over multiple machines

How might an inverted index be split over multiple machines?

Extremely Scalable/Efficient

- When engineered correctly 😊

Apache Lucene

- **Inverted Index**
  - They built one so you don’t have to!
- **Open Source in Java**

(Apache Solr)

- Built on top of Apache Lucene
- Lucene is the inverted index
- Solr is a distributed search platform, with distribution, fault tolerance, etc.
- (We will work with Lucene)

Apache Lucene: Indexing Documents

... continued ...
Information overload in Big Data

- **Search**: user intent
- **Query**: user expression of search
- **Retrieval**: machine methods to execute search
CLASS PROJECTS

Course Marking

• 45% for Weekly Labs (~3% a lab!)
• 35% for Final Exam
• 20% for Small Class Project

Class Project

• Done in pairs (typically)
• Goal: Use what you’ve learned to do something cool (basically)
• Expected difficulty: A bit more than a lab’s worth
  – But without guidance (can extend lab code)
• Marked on: Difficulty, appropriateness, scale, good use of techniques, presentation, coolness
  – Ambition is appreciated, even if you don’t succeed: feel free to bite off more than you can chew!
• Process:
  – Pair up (default random) by next Monday
  – Start thinking up topics
  – If you need data or get stuck, I will (try to) help out
• Deliverables: 5 minute presentation & 3-page report

Questions