How to build Google in an afternoon
(or any other large web search engine)

Djoerd Hiemstra
SIKS course on Information Retrieval, 19 June 2015
http://www.cs.utwente.nl/~hiemstra
Ingredients of this talk:

1. A bit of high school mathematics
2. Zipf's law
3. Indexing, query processing

Shake well…
Course objectives

• Understand the scale of “things”
• Estimate index size and query time
• Index compression
• Top-$k$ optimization
Dear bank,

- How much money do we need for our startup?
A web-scale search engine

• We budget one desktop PC
• We put the entire web index on a desktop PC and search it in reasonable time:
  a) probably
  b) maybe
  c) no
  d) no, are you crazy?
“Google” Circa 1997 (google.stanford.edu)
“Corkboards” (1999)
Google '98: Forward & Inverted Index

Hit: 2 bytes

<table>
<thead>
<tr>
<th>plain</th>
<th>cap: 1</th>
<th>imp: 3</th>
<th>position: 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>fancy</td>
<td>cap: 1</td>
<td>imp = 7</td>
<td>type: 4</td>
</tr>
<tr>
<td>anchor</td>
<td>cap: 1</td>
<td>imp = 7</td>
<td>type: 4</td>
</tr>
</tbody>
</table>

Forward Barrels: total 43 GB

```
docid | wordid: 24 | nhits: 8 | hit hit hit hit
      | wordid: 24 | nhits: 8 | hit hit hit hit
      | null wordid
      | docid | wordid: 24 | nhits: 8 | hit hit hit hit
      | wordid: 24 | nhits: 8 | hit hit hit hit
      | wordid: 24 | nhits: 8 | hit hit hit hit
      | null wordid
```

Lexicon: 293MB

```
wordid | ndocs
-------|------
wordid | ndocs
wordid | ndocs
```

Inverted Barrels: 41 GB

```
docid | nhits: 5 | hit hit hit hit
      | nhits: 5 | hit hit hit hit
      | nhits: 5 | hit hit hit hit
      | nhits: 5 | hit hit
```

UNIVERSITY OF TWENTE.
## Google'98: Storage numbers

<table>
<thead>
<tr>
<th>Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Size of Fetched Pages</td>
<td>147.8 GB</td>
</tr>
<tr>
<td>Compressed Repository</td>
<td>53.5 GB</td>
</tr>
<tr>
<td>Short Inverted Index</td>
<td>4.1 GB</td>
</tr>
<tr>
<td>Full Inverted Index</td>
<td>37.2 GB</td>
</tr>
<tr>
<td>Lexicon</td>
<td>293 MB</td>
</tr>
<tr>
<td>Temporary Anchor Data (not in total)</td>
<td>6.6 GB</td>
</tr>
<tr>
<td>Document Index Incl. Variable Width Data</td>
<td>9.7 GB</td>
</tr>
<tr>
<td>Links Database</td>
<td>3.9 GB</td>
</tr>
<tr>
<td><strong>Total Without Repository</strong></td>
<td><strong>55.2 GB</strong></td>
</tr>
<tr>
<td><strong>Total With Repository</strong></td>
<td><strong>108.7 GB</strong></td>
</tr>
</tbody>
</table>

UNIVERSITY OF TWENTE.
# Google'98: Page search

<table>
<thead>
<tr>
<th>Web Page Statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Web Pages Fetched</td>
<td>24 million</td>
</tr>
<tr>
<td>Number of URLs Seen</td>
<td>76.5 million</td>
</tr>
<tr>
<td>Number of Email Addresses</td>
<td>1.7 million</td>
</tr>
<tr>
<td>Number of 404's</td>
<td>1.6 million</td>
</tr>
</tbody>
</table>

UNIVERSITY OF TWENTE.
## Google'98: Search speed

<table>
<thead>
<tr>
<th>Query</th>
<th>Initial Query</th>
<th>Same Query Repeated (IO mostly cached)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU Time(s)</td>
<td>Total Time(s)</td>
</tr>
<tr>
<td>al gore</td>
<td>0.09</td>
<td>2.13</td>
</tr>
<tr>
<td>vice president</td>
<td>1.77</td>
<td>3.84</td>
</tr>
<tr>
<td>hard disks</td>
<td>0.25</td>
<td>4.86</td>
</tr>
<tr>
<td>search engines</td>
<td>1.31</td>
<td>9.63</td>
</tr>
</tbody>
</table>
Google’s 16th birthday
Larger-Scale Computing
Google’s 16th birthday

- World's largest cluster of commodity hardware (way over 100,000 servers)
- These are partitioned between index servers and page servers (and more)
  - Index servers resolve the queries (massively parallel processing)
  - Page servers deliver the results of the queries: urls, title, snippets
- Over XX(?) billion web pages are indexed
  - 1 trillion pages reportedly found
1. The web server sends the query to the index servers. The content inside the index servers is similar to the index in the back of a book - it tells which pages contain the words that match the query.

2. The query travels to the doc servers, which actually retrieve the stored documents. Snippets are generated to describe each search result.

3. The search results are returned to the user in a fraction of a second.
More info:

Jeff Dean's WSDM 2009 keynote:

Challenges in Building Large-Scale Information Retrieval Systems

http://research.google.com/people/jeff/WSDM09-keynote.pdf
http://videolectures.net/wsdm09_dean_cblirs/
## Queries per day? (December 2007)

<table>
<thead>
<tr>
<th>Service</th>
<th>Searches per day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>180 million</td>
</tr>
<tr>
<td>Yahoo</td>
<td>70 million</td>
</tr>
<tr>
<td>Microsoft</td>
<td>30 million</td>
</tr>
<tr>
<td>Ask</td>
<td>13 million</td>
</tr>
</tbody>
</table>

[http://searchenginewatch.com](http://searchenginewatch.com)
Search Engine Popularity

Marketshare The Netherlands, 2011
- Google: 94.0%
- Vinden.nl, Bing, Yahoo: 6.0%

Source: Comscore 2011

Marketshare US, 2011
- Google: 65.4%
- Yahoo: 15.9%
- Bing: 14.1%
- Ask: 1.5%
- AOL: 0.1%

UNIVERSITY OF TWENTE.
Q1: How many bytes is 10 billion pages?

• Only the text
How many bytes?

• About 10 billion pages
• Assume a page contains 500 terms on average (ClueWeb09: about 900)
• Each term consists of 5 characters on average (Witten, Moffat & Bell, 1999)
• To store the web you need:
  \[ 10^{10} \times 500 \times 6 \sim 30 \text{ TB} \]
Table 4.2  Collection statistics for Reuters-RCV1. Values are rounded for the computations in this book. The unrounded values are: 806,791 documents, 222 tokens per document, 391,523 (distinct) terms, 6.04 bytes per token with spaces and punctuation, 4.5 bytes per token without spaces and punctuation, 7.5 bytes per term, and 96,969,056 tokens. The numbers in this table correspond to the third line ("case folding") in Table 5.1 (page 87).

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>documents</td>
<td>800,000</td>
</tr>
<tr>
<td>$L_{ave}$</td>
<td>avg. # tokens per document</td>
<td>200</td>
</tr>
<tr>
<td>$M$</td>
<td>terms</td>
<td>400,000</td>
</tr>
<tr>
<td></td>
<td>avg. # bytes per token (incl. spaces/punct.)</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>avg. # bytes per token (without spaces/punct.)</td>
<td>4.5</td>
</tr>
<tr>
<td></td>
<td>avg. # bytes per term</td>
<td>7.5</td>
</tr>
<tr>
<td>$T$</td>
<td>tokens</td>
<td>100,000,000</td>
</tr>
</tbody>
</table>
What did we ignore?

• Text statistics:
  – Term frequency
  – Collection frequency
  – Inverse document frequency …

• Hypertext statistics:
  – Ingoing and outgoing links
  – Anchor text
  – Term positions, proximities, sizes, and characteristics …
Q2: How fast can we scan 30 TB?

• How would you estimate this?
How fast can we search 30 TB?

• We need to find a very large hard disk
  – Size: 30 TB ??
  – Hard disk transfer time 100 MB/s

• Time needed to sequentially scan the data:
  – 300,000 seconds …
  – … so, we have to wait for 3.5 days to get the answer to one (1) query

• We can definitely do better than that!
**Table 4.1** Typical system parameters in 2007. The seek time is the time needed to position the disk head in a new position. The transfer time per byte is the rate of transfer from disk to memory when the head is in the right position.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td>average seek time</td>
<td>$5 \text{ ms} = 5 \times 10^{-3} \text{ s}$</td>
</tr>
<tr>
<td>$b$</td>
<td>transfer time per byte</td>
<td>$0.02 \mu s = 2 \times 10^{-8} \text{ s}$</td>
</tr>
<tr>
<td></td>
<td>processor’s clock rate</td>
<td>$10^9 \text{ s}^{-1}$</td>
</tr>
<tr>
<td>$p$</td>
<td>lowlevel operation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(e.g., compare &amp; swap a word)</td>
<td>$0.01 \mu s = 10^{-8} \text{ s}$</td>
</tr>
<tr>
<td></td>
<td>size of main memory</td>
<td>several GB</td>
</tr>
<tr>
<td></td>
<td>size of disk space</td>
<td>$1 \text{ TB or more}$</td>
</tr>
</tbody>
</table>
Issues that we do not address

• Web crawling
  – politeness, freshness, duplicates, missing links, loops, server problems, virtual hosts, etc.

• Maintain large cluster of servers
  – Page servers: store and deliver the results of the queries
  – Index servers: resolve the queries

• Answer 100 million of user queries per day
  – Caching, replicating, parallel processing, etc.
  – Indexing, compression, coding, fast access, etc.
Implementation issues

• Analyze the collection
  – Avoid non-informative data for indexing
  – Decision on relevant statistics and info
• Index the collection
  – How to organize the index?
• Compress the data
  – Data compression
  – Index compression
Ingredients of this talk:

1. A bit of high school mathematics
2. Zipf's law
3. Indexing, query processing

Shake well…
Zipf's law

• Count how many times a term occurs in the collection
  – call this $f$

• Order them in descending order
  – call the rank $r$

• Zipf's claim:
  – For each word, the product of frequency and rank is approximately constant: $f \times r = c$
Zipf distribution

Term count

Linear scale

Terms by rank order
Zipf distribution

Logarithmic scale

Term count

Terms by rank order

Logarithmic scale
Consequences

Few terms occur very frequently: a, an, the, ...
=> non-informative (stop) words

• Many terms occur very infrequently:
  spelling mistakes, foreign names, ...

• Medium number of terms occur with medium frequency
Figure 2.1. A plot of the hyperbolic curve relating $f$, the frequency of occurrence, and $i$, the rank of the word. (Adapted from Schultz, page 120)
Heap’s law for dictionary size

Number of unique terms vs. collection size.
Ingredients of this talk:

1. A bit of high school mathematics
2. Zipf's law
3. Indexing

Shake well…
## Example

<table>
<thead>
<tr>
<th>Document number</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pease porridge hot, pease porridge cold</td>
</tr>
<tr>
<td>2</td>
<td>Pease porridge in the pot</td>
</tr>
<tr>
<td>3</td>
<td>Nine days old</td>
</tr>
<tr>
<td>4</td>
<td>Some like it hot, some like it cold</td>
</tr>
<tr>
<td>5</td>
<td>Some like it in the pot</td>
</tr>
<tr>
<td>6</td>
<td>Nine days old</td>
</tr>
</tbody>
</table>

Stop words: in, the, it.

(Witten, Moffat & Bell, 1999)
Inverted index

<table>
<thead>
<tr>
<th>term</th>
<th>offset</th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>cold</td>
<td>2</td>
<td>1, 4</td>
</tr>
<tr>
<td>days</td>
<td>4</td>
<td>3, 6</td>
</tr>
<tr>
<td>hot</td>
<td>6</td>
<td>1, 4</td>
</tr>
<tr>
<td>like</td>
<td>8</td>
<td>4, 5</td>
</tr>
<tr>
<td>nine</td>
<td>10</td>
<td>3, 6</td>
</tr>
<tr>
<td>old</td>
<td>12</td>
<td>3, 6</td>
</tr>
<tr>
<td>pease</td>
<td>14</td>
<td>1, 2</td>
</tr>
<tr>
<td>porridge</td>
<td>16</td>
<td>1, 2</td>
</tr>
<tr>
<td>pot</td>
<td>18</td>
<td>2, 5</td>
</tr>
<tr>
<td>some</td>
<td>20</td>
<td>4, 5</td>
</tr>
</tbody>
</table>

dictionary
postings
Q3: Estimate the size of the inverted index
Size of the inverted index

- Number of postings (term-document pairs):
  - Number of documents: $\sim 10^{10}$,
  - Average number of unique terms per document (document size $\sim 500$): $\sim 250$
  - 5 bytes for each posting (why?)
  - So, $10^{10} \times 250 \times 5 = 12.5$ TB
  - postings take about half the size of the data
Size of the inverted index

• Number of unique terms is, say, $10^8$
  – 6 bytes on average
  – plus off-set in postings, another 8 bytes
  – So, $10^8 \times 14 = 1.4$ GB
  – So, dictionary is tiny compared to postings (0.01 %)

• Another optimization (Galago):
  – sort dictionary alphabetically
  – at maximum one vocabulary entry for each 32 KB block
Inverted index encoding

• The inverted file entries are usually stored in order of increasing document number

  – [<i>retrieval</i>; 7; [2, 23, 81, 98, 121, 126, 180]>

  (the term “retrieval” occurs in 7 documents with document identifiers 2, 23, 81, 98, etc.)
Query processing (1)

• Each inverted file entry is an ascending ordered sequence of integers
  – allows merging (joining) of two lists in a time linear in the size of the lists
Query processing (2)

• Usually queries are assumed to be conjunctive queries
  – query: information retrieval
  – is processed as information AND retrieval

[<retrieval; 7; [2, 23, 81, 98, 121, 126, 139]>  
[<information; 9; [1, 14, 23, 45, 46, 84, 98, 111, 120]>  
  – intersection of posting lists gives:  
  [23, 98]
Query processing (3)

- Remember the Boolean model?
  - intersection, union and complement is done on posting lists
  - so, information OR retrieval

\[
\text{[<retrieval; 7; [2, 23, 81, 98, 121, 126, 139]>}
\]
\[
\text{[<information; 9; [1, 14, 23, 45, 46, 84, 98, 111, 120]>}
\]
  - union of posting lists gives:
\[
[1, 2, 14, 23, 45, 46, 81, 84, 98, 111, 120, 121, 126, 139]
\]
Q4: Estimate the time needed for the query “information retrieval” using the inverted file

• Assume the selectivity of terms:
  – Suppose *information* occurs on 1 billion pages
  – Suppose *retrieval* occurs on 10 million pages

*Is this a reasonable estimate?*
Query processing (4)

• Estimate of selectivity of terms:
  – Suppose *information* occurs on 1 billion pages
  – Suppose *retrieval* occurs on 10 million pages

• Size of postings (5 bytes per docid):
  – 1 billion * 5B = 5 GB for *information*
  – 10 million * 5B = 50 MB for *retrieval*

• Hard disk transfer time:
  – 50 sec. for *information* + 0.5 sec. for *retrieval*
  – (ignore CPU time and disk latency)
Query processing (5)

• We just brought query processing down from 3 days to just 50.5 seconds (!)

  :-)  

• Still… way too slow…

  :-(
Inverted file compression (1)

- **Trick 1**, store sequence of doc-ids:
  - $[<\text{retrieval}; 7; [2, 23, 81, 98, 121, 126, 180]>$

  as a sequence of gaps
  - $[<\text{retrieval}; 7; [2, 21, 58, 17, 23, 5, 54]>$

- No information is lost.
- Always process posting lists from the beginning, so easily decoded into the original sequence.
Inverted file compression (2)

• Does it help?
  – maximum gap determined by the number of indexed web pages...
  – infrequent terms coded as a few large gaps
  – frequent terms coded by many small gaps

• Trick 2: use variable byte length encoding.
Variable byte encoding (1)

<table>
<thead>
<tr>
<th>Gap $x$</th>
<th>Unary</th>
<th>$\gamma$</th>
<th>$\delta$</th>
<th>Golomb $b = 3$</th>
<th>Golomb $b = 6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>000</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>100</td>
<td>1000</td>
<td>010</td>
<td>001</td>
</tr>
<tr>
<td>3</td>
<td>110</td>
<td>101</td>
<td>1001</td>
<td>011</td>
<td>0100</td>
</tr>
<tr>
<td>4</td>
<td>1110</td>
<td>11000</td>
<td>10100</td>
<td>100</td>
<td>0101</td>
</tr>
<tr>
<td>5</td>
<td>11110</td>
<td>11001</td>
<td>10101</td>
<td>1010</td>
<td>0110</td>
</tr>
<tr>
<td>6</td>
<td>111110</td>
<td>11010</td>
<td>10110</td>
<td>1011</td>
<td>0111</td>
</tr>
<tr>
<td>7</td>
<td>1111110</td>
<td>11011</td>
<td>10111</td>
<td>1100</td>
<td>1000</td>
</tr>
<tr>
<td>8</td>
<td>11111110</td>
<td>1110000</td>
<td>11000000</td>
<td>11010</td>
<td>1001</td>
</tr>
<tr>
<td>9</td>
<td>111111110</td>
<td>1110001</td>
<td>11000001</td>
<td>11011</td>
<td>10100</td>
</tr>
<tr>
<td>10</td>
<td>1111111110</td>
<td>1110010</td>
<td>1100010</td>
<td>11100</td>
<td>10101</td>
</tr>
</tbody>
</table>

Table 3.5  Example codes for integers.

(Witten, Moffat & Bell, 1999)
Q5: Give $\gamma$ code for $x=5$

- $\gamma$ code: represent number $x$ as:
  - first bits as the unary code for $1 + \left\lfloor \log_2 x \right\rfloor$
  - remainder bits as binary code for $x - 2^{\left\lfloor \log_2 x \right\rfloor}$
  - unary part (minus 1) specifies how many bits are required to code the remainder part
Variable byte encoding (2)

- γ code: represent number $x$ as:
  - first bits as the unary code for $1 + \left\lfloor \frac{2 \log x}{\log x - 2} \right\rfloor$
  - remainder bits as binary code for $x - 2^{\left\lfloor \frac{2 \log x}{\log x - 2} \right\rfloor}$
  - unary part (minus 1) specifies how many bits are required to code the remainder part

- For example $x = 5$:
  - first bits: 110
  - remainder: 01

\[
\left( 1 + \left\lfloor \frac{2 \log 5}{\log 5 - 2} \right\rfloor = 1 + \left\lfloor 2.32 \right\rfloor = 3 \right)
\]
\[
\left( 5 - 2^{\left\lfloor \frac{2 \log 5}{\log 5 - 2} \right\rfloor} = 5 - 2^2 = 1 \right)
\]
# Index sizes

<table>
<thead>
<tr>
<th>Method</th>
<th>Bits per pointer</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bible</td>
<td>GNUbib</td>
<td>Comact</td>
<td>TREC</td>
</tr>
<tr>
<td><strong>Global methods</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unary</td>
<td>264</td>
<td>920</td>
<td>490</td>
<td>1719</td>
</tr>
<tr>
<td>Binary</td>
<td>15.00</td>
<td>16.00</td>
<td>18.00</td>
<td>20.00</td>
</tr>
<tr>
<td>Bernoulli</td>
<td>9.67</td>
<td>11.65</td>
<td>10.58</td>
<td>12.61</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>6.55</td>
<td>5.69</td>
<td>4.48</td>
<td>6.43</td>
</tr>
<tr>
<td>( \delta )</td>
<td>6.26</td>
<td>5.08</td>
<td>4.36</td>
<td>6.19</td>
</tr>
<tr>
<td>Observed frequency</td>
<td>5.92</td>
<td>4.83</td>
<td>4.21</td>
<td>5.83</td>
</tr>
<tr>
<td><strong>Local methods</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bernoulli</td>
<td>6.13</td>
<td>6.17</td>
<td>5.40</td>
<td>5.73</td>
</tr>
<tr>
<td>Hyperbolic</td>
<td>5.77</td>
<td>5.17</td>
<td>4.65</td>
<td>5.74</td>
</tr>
<tr>
<td>Skewed Bernoulli</td>
<td>5.68</td>
<td>4.71</td>
<td>4.24</td>
<td>5.28</td>
</tr>
<tr>
<td>Batched frequency</td>
<td>5.61</td>
<td>4.65</td>
<td>4.03</td>
<td>5.27</td>
</tr>
</tbody>
</table>

Table 3.7  Compression of inverted files, in bits per pointer.

(Witten, Moffat & Bell, 1999)
Q6: Estimate the compressed index size
Index size of our search engine

• Number of postings (term-document pairs):
  – 10 billion documents
  – 250 unique terms on average
  – Assume on average 6 bits per doc-id
  – $10^{10} \times 250 \times 6$ bits $\sim= 1.9$ TB
  – about 15% of the uncompressed inverted file.

• It exactly fits one big hard drive :-}
Q7: Estimate the time needed for the query “information retrieval” using the compressed inverted file

• Assume the selectivity of terms:
  – Suppose *information* occurs on 1 billion pages
  – Suppose *retrieval* occurs on 10 million pages
Query processing on compressed index

• size of postings (6 bits per docid):
  – 1 billion * 6 bits = 750 Mb for "information"
  – 10 million * 6 bits = 7.5 Mb for "retrieval"

• Hard disk transfer time:
  – 7.5 sec. for information + 0.08 sec. for retrieval
  – (ignore CPU time and disk latency)
Query processing – Continued (1)

• We already brought down query processing from more than 1 day to 50.5 seconds...
• and brought that down to 7.58 seconds :-)

• but that is still too slow... :-(

UNIVERSITY OF TWENTE.
Google PageRank

• Given a document $D$, the documents page rank at step $n$ is:

$$P_n(D) = (1 - \lambda) P_0(D) + \lambda \left( \sum_{I \text{ linking to } D} P_{n-1}(I) P(D|I) \right)$$

• where

$P(D | I)$: probability that the monkey reaches page $D$ through page $I$ ($= 1 / \#\text{outlinks of } I$)

$\lambda$: probability that the follows a link

$1-\lambda$: probability that the monkey types a url

TO BE DONE IN NEXT LECTURE
Early termination (1)

- Suppose we re-sort the document ids for each posting such that the best documents come first
  - e.g., sort document identifiers for "retrieval" by their tf.idf values.
  - \[\langle\text{retrieval}; 7; [98, 23, 180, 81, 98, 121, 2, 126,]\rangle\]
  - then: top 10 documents for the query "retrieval" can be retrieved very quickly: stop after processing the first 10 document ids from the posting list!
  - but compression and merging (multi-word queries) of postings no longer possible...
Early termination (2)

• **Trick 3**: define a static (or global) ranking of all documents
  – such as Google PageRank (!)
  – re-assign document identifiers by ascending PageRank
  – For every term, documents with a high PageRank are in the initial part of the posting list
  – Estimate the selectivity of the query and only process part of the posting files.

(see e.g. Croft, Metzler & Strohman 2009)
Q8: Estimate the time when early termination is implemented?
Early termination (3)

- Probability that a document contains a term:
  - $1 \text{ billion} / 10 \text{ billion} = 0.1$ for \textit{information}
  - $10 \text{ million} / 10 \text{ billion} = 0.001$ for \textit{retrieval}

- Assume independence between terms:
  - $0.1 \times 0.001 = 0.0001$ of the documents contains both terms
  - so, every $1 / 0.0001 = 10,000$ documents on average contains \textit{information AND retrieval}.
  - for top 30, process $3,000,000$ documents.
  - $3,000,000 / 10 \text{ billion} = 0.0003$ of the posting files
Query processing on compressed index with early termination

• process about 0.0003 of postings:
  – 0.0003 * 750 Mb = 225 kb for information
  – 0.0003 * 7.5 Mb = 2.25 kb for retrieval

• Hard disk transfer time:
  – 2 msec. for information + 0.02 msec. for retrieval
  – (NB now, ignoring CPU time, disk latency and decompressing time is no longer reasonable, so it is likely that it takes some more time)
Query processing on compressed index with early termination

• process about 0.0003 of postings:
  – 0.0003 * 750 Mb = 225 kb for information
  – 0.0003 * 7.5 Mb = 2.25 kb for retrieval

• Hard disk transfer time:
  – 2 msec. for information + 0.02 msec. for retrieval

   – (NB now, ignoring CPU time, disk latency and decompressing time is no longer reasonable, so it is likely that it takes some more time)
Query processing – Continued (2)

• We just brought query processing down from more than 3.5 days to about 2 ms. ! :-) 

“This engine is incredibly, amazingly, ridiculously fast!”

(from “Top Gear”)
Indexing - Recap

• Inverted files
  – dictionary & postings
  – merging of posting lists
  – delta encoding + variable byte encoding
  – static ranking + early termination

• Put the entire web index on a desktop PC and search it in reasonable time:
  a) probably
Ingredients of this talk:

1. A bit of high school mathematics
2. Zipf's law
3. Indexing

Shake well...
Summary

• Search engines
  – Google: first steps and now

• Term distribution and statistics
  – What is useful and what is not

• Indexing techniques (inverted files)
  – How to index the web

• Compression, coding, and querying
  – How to squeeze the index for efficient search
Acknowledgements

• Thanks to the following people for contributing slides:
  – Vojkan Mihajlovic (Philips Research)
  – Jeffrey Dean (Google)