CIS 455/555: Internet and Web Systems

Google

November 10, 2021
Plan for today

- **PageRank**
  - Iterative computation
  - Random-surfer model
  - Refinements

- Google
Recap: PageRank

- Estimates absolute 'quality' or 'importance' of a given page based on inbound links
  - Query-independent
  - Can be computed via fixpoint iteration
  - Can be interpreted as the fraction of time a 'random surfer' would spend on the page
  - Several refinements, e.g., to deal with sinks

- Considered relatively stable
  - But vulnerable to black-hat SEO

- An important factor, but not the only one
  - Overall ranking is based on many factors (Google: >200)
  - Need to perform rank merging, e.g., with TF/IDF scores
    - e.g., TF/IDF can ensure high precision, and PageRank high quality
Beyond PageRank

- PageRank assumes a “random surfer” who starts at any node and estimates likelihood that the surfer will end up at a particular page.

- A more general notion: label propagation
  - Take a set of start nodes each with a different label
  - Estimate, for every node, the distribution of arrivals from each label
  - In essence, captures the relatedness or influence of nodes
  - Used in YouTube video matching, schema matching, ...
Plan for today

- **PageRank**
  - Random-surfer model
  - Refinements

- **Google**
  - Modern Search
Let’s Look at a Model for Your Project

- Google, circa 1998, provides a good model for the project

- As well as some lessons learned
Focus was on scalability to the size of the Web

First to really exploit Link Analysis

Started as an academic project @ Stanford; became a startup

Our discussion will be on early Google – today they keep things secret!
The Heart of Google Storage

- “BigFile” system for storing indices, tables
  - Support for $2^{64}$ bytes across multiple drives, filesystems
  - Manages its own file descriptors, resources
  - This was the predecessor to GFS

- First use: Repository
  - Basically, a warehouse of every HTML page (this is the 'cached page' entry), compressed in zlib (faster than bzip)
  - Useful for doing additional processing, any necessary rebuilds
  - Repository entry format:
    - [DocID][ECode][UrlLen][PageLen][Url][Page]
  - The repository is indexed (not inverted here)
Repository Index

- One index for looking up documents by DocID
  - Done in ISAM (think of this as a B+ Tree without smart re-balancing)
  - Index points to repository entries (or to URL entry if not crawled)

- One index for mapping URL to DocID
  - Sorted by checksum of URL
  - Compute checksum of URL, then perform binary search by checksum
  - Allows update by merge with another similar file
    - Why is this done?
Lexicon

- The list of searchable words
  - (Presumably, today it’s used to suggest alternative words as well)
  - The “root” of the inverted index

- As of 1998, 14 million “words”
  - Kept in memory (was 256MB)
  - Two parts:
    - Hash table of pointers to words and the “barrels” (partitions) they fall into
    - List of words (null-separated)
Indices – Inverted and “Forward”

- Inverted index divided into “barrels” (partitions by range)
  - Indexed by the lexicon; for each DocID, consists of a Hit List of entries in the document
  - Two barrels: short (anchor and title); full (all text)

- Forward index uses the same barrels
  - Indexed by DocID, then a list of WordIDs in this barrel and this document, then Hit Lists corresponding to the WordIDs

Lexicon: 293 MB

<table>
<thead>
<tr>
<th>WordID</th>
<th>ndocs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

“barrels” are now called “shards”

Inverted Barrels: 41 GB

<table>
<thead>
<tr>
<th>DocID</th>
<th>nhits: 8</th>
<th>hi t</th>
<th>hi t</th>
<th>hi t</th>
<th>hi t</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td></td>
<td>hi t</td>
<td>hi t</td>
<td>hi t</td>
<td>hi t</td>
</tr>
</tbody>
</table>

forward barrels: total 43 GB

<table>
<thead>
<tr>
<th>DocID</th>
<th>WordID: 24</th>
<th>nhits: 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>WordID: 24</td>
<td>nhits: 8</td>
</tr>
<tr>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>WordID: 24</td>
<td>nhits: 8</td>
</tr>
</tbody>
</table>

original tables from
Hit Lists

- Used in inverted and forward indices
- Goal was to minimize the size – the bulk of data is in hit entries
  - For 1998 version, made it down to 2 bytes per hit (though that’s likely climbed since then):

<table>
<thead>
<tr>
<th></th>
<th>cap</th>
<th>font</th>
<th>type</th>
<th>hash</th>
<th>pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>Fancy</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Anchor</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

special-cased to:
Google’s Distributed Crawler

- **Single URL Server** – the coordinator
  - A queue that farms out URLs to crawler nodes
  - Implemented in Python!

- **Crawlers had 300 open connections apiece**
  - Each needs own DNS cache – DNS lookup is major bottleneck, as we have seen
  - Based on asynchronous I/O

- **Many caveats in building a “friendly” crawler**
  (remember robot exclusion protocol?)
Google’s Search Algorithm

1. Parse the query
2. Convert words into wordIDs
3. Seek to start of doclist in the short barrel for every word
4. Scan through the doclists until there is a document that matches all of the search terms
5. Compute the rank of that document
   - IR score: Uses dot product of count weights and type weights
   - Final rank: IR score combined with PageRank
6. If we’re at the end of the short barrels, start at the doclists of the full barrel, unless we have enough
7. If not at the end of any doclist, goto step 4
8. Sort the documents by rank; return the top K
Features in Google’s Early Ranking Function

- Considers many types of information:
  - Position, font size, capitalization
  - Anchor text
  - PageRank
  - Count of occurrences (basically, TF) in a way that tapers off
    - (Not clear if they did IDF at the time?)

- Multi-word queries consider proximity as well
  - How?
Google’s Resources

- In 1998:
  - 24M web pages
  - About 55GB data w/o repository
  - About 110GB with repository
  - Lexicon 293MB
  - Worked quite well with low-end PC

- In 2007: > 27 billion pages, >1.2B queries/day:
  - Don’t attempt to include all barrels on every machine!
    - e.g., 5+TB repository on special servers separate from index servers
  - Many special-purpose indexing services (e.g., images)
  - Much greater distribution of data (~500K PCs?), huge net BW
  - Advertising needs to be tied in (>1M advertisers in 2007)

- 2019: 3.5 billion searches/day (internetlivestats)
Google over the years

- August 2001: Search algorithm revamped
  - Incorporate additional ranking criteria more easily
- February 2003: Local connectivity analysis
  - More weight to links from experts' sites. Google's first patent.
- Summer 2003: Fritz
  - Index updated incrementally, rather than in big batches
- June 2005: Personalized results
  - User click-throughs and searches affect ranking!
- December 2005: Engine update
  - Allows for more comprehensive web crawling

Source: http://www.wired.com/magazine/2010/02/ff_google_algorithm/all/1
Google over the years

- **May 2007: Universal search**
  - Users can get links to any medium (images, news, books, maps, etc) on the same results page
- **December 2009: Real-time search**
  - Display results from Twitter & blogs as they are posted
- **August 2010: Caffeine**
  - New indexing system; "50 percent fresher results"
- **February 2011: Major change to algorithm**
  - The "Panda update" (revised since; Panda 3.3 in Feb 2012)
  - "Designed to reduce the rankings of low-quality sites"
- **Algorithm is still updated frequently**
- **Today: annotated concepts from Knowledge Graph, experiments with consistency with what’s known:**


Source: [http://www.wired.com/magazine/2010/02/ff_google_algorithm/all/1](http://www.wired.com/magazine/2010/02/ff_google_algorithm/all/1)
Plan for today

- PageRank
  - Random-surfer model
  - Refinements
- Google
  - Modern Search
What Can You Do beyond PageRank?

- Google does much more today than TF*IDF and PageRank

- Some natural language parsing of the query and the text in a document, to better understand the context and question

- Matching against known entities in the knowledge graph

- Using frequencies of co-occurrences to understand the task

- Task-specific search
Consider “Michael Jordan Statistics”

<table>
<thead>
<tr>
<th>Players</th>
<th>Teams</th>
<th>Seasons</th>
<th>Leaders</th>
<th>Score</th>
<th>Add</th>
<th>PFP</th>
<th>Add</th>
<th>Add</th>
<th>Add</th>
</tr>
</thead>
</table>

Michael Jordan
- **Michael Jeffrey Jordan**
  - **Born**: February 17, 1963 (Age: 58-0564) in Brooklyn, New York
  - **College**: UNC

**Summary**
- **Career**:
  - **G**: 1072
  - **PTS**: 50.1
  - **TRB**: 6.2
  - **AST**: 5.3
  - **FG%**: 49.7
  - **FG3%**: 32.7
  - **FT%**: 83.5
  - **PCT**: 50.0
  - **PER**: 21.9
  - **WS**: 214.0

Michael I. Jordan
- **Pehong Chen Distinguished Professor**
- **Department of EECS**
- **Department of Statistics**
- **AMP Lab**
- **Berkeley AI Research Lab**
- **University of California, Berkeley**

**Emails**:
- jordan@cs.berkeley.edu
- jordan@stat.berkeley.edu

**EECS Address**:
- University of California, Berkeley
- EECS Department
- 387 Soda Hall #1776
- Berkeley, CA 94720-1776

**Statistics Address**:
- University of California, Berkeley
- Statistics Department
- 427 Evans Hall #3860
- Berkeley, CA 94720-3860

Michael B. Jordan
- **American actor**

Michael Bakari Jordan is an American actor and producer. He is known for his film roles as shooting victim Oscar Grant in the drama Fruitvale Station, boxer Donnie Creed in Creed, and Erik Killmonger in Black Panther, all three of which were directed by Ryan Coogler. [Wikipedia](https://en.wikipedia.org/wiki/Michael_B._Jordan)

**Upcoming movies**: Black Panther II, Babylon, Space Jam: A New Legacy

**Trending**

**Born**: February 9, 1987 (age 34 years), Santa Ana, CA
What Do We Need to Know?

- Who are the different Michael Jordans?

- Which Michael Jordan is referred to in each document?

- Which Michael Jordan is referred to in each document *clicked on* when “Michael Jordan statistics” is asked?
Knowledge Graphs

- **people/person**
  - subclassOf **people/athlete**
  - subclassOf **people/professor**
  - subclassOf **people/actor**

- **instanceOf**
  - Michael J Jordan
  - Michael I Jordan
  - Michael B Jordan
Exploiting Knowledge Graphs

https://www.wikidata.org/wiki/Q15241312

Freebase: the basis of the **Google Knowledge Graph**

No longer updated externally, data available at [http://freebase-easy.cs.uni-freiburg.de/browse/](http://freebase-easy.cs.uni-freiburg.de/browse/)

DBpedia: a crawl and extraction of Wikipedia, [wiki.dbpedia.org](http://wiki.dbpedia.org)

YAGO, [https://github.com/yago-naga/yago3](https://github.com/yago-naga/yago3)
Co-occurrence Frequency

- If we can connect “Michael Jordan” in each page
- Then we can see which pages the user clicks on for each query, and connect which “sense” was meant
- We can also aggregate – is “people/actor” or “people/athlete” more likely to be used with the word “statistics”

- Google can figure out the most likely interpretation of our query!
Task-Specific Search

Google, Bing, etc also use knowledge graphs to plot common queries, and build custom task-specific searches:

“country gdp” →

“city to city”
“Adwords” – Google “auctions” search terms

- Buy the most frequent slot, 2\textsuperscript{nd} most, ...
- “Generalized second price auction” – highest bidder pays 2\textsuperscript{nd} highest price, 2\textsuperscript{nd} highest bidder pays 3\textsuperscript{rd} highest, etc.
Summary

- TF*IDF and PageRank were state of the art in 2000

- Today we need to know about entities and relationships to do well
  - Which also helps us do a better job of building “vertical” search functionality

- And ads are based on keywords too!
Plan for today

- **PageRank**
  - Random-surfer model
  - Refinements

- **Google**
  - Modern Search

- **Transactions**
  - ACID properties
  - Serializability
  - Concurrency control: 2PL
  - Durable storage
  - Distributed transactions: 2PC
We’ve Not Discussed Failures in Detail...

- eBay wants to sell an item to:
  - The highest bidder, once the auction is over, or
  - The person who’s first to click “Buy It Now!”

- But: What if the bidder doesn’t have the cash?

- A solution:
  - Tentatively record the item as sold
  - Validate the PayPal or credit card info with a 3rd party
  - If not valid, discard this bidder and resume in prior state
"No Payment" isn’t the only source of failure

Suppose we start to transfer the money, but a server goes down...

Purchase:

\[
\begin{align*}
\text{sb} &= \text{Seller.bal} \\
\text{bb} &= \text{Buyer.bal} \\
\text{Write Buyer.bal} &= \text{bb} - 100 \\
\text{Write Item.sellTo} &= \text{Buyer} \\
\text{Write Seller.bal} &= \text{sb} + 100
\end{align*}
\]

CRASH!
Transactions

- There are many (especially, financial) applications where we want to create atomic operations that either commit or roll back
  - ... despite hardware/software/application/other failures
  - ... whether or not other operations are executed concurrently

- This is one of the most basic services provided by database management systems, but we want to do it in a broader sense

- Part of “ACID” semantics...
ACID Semantics

- **Atomicity**: operations are atomic, either committing or aborting as a single entity
- **Consistency**: the state of the data is internally consistent
  - **Note**: NOT like the consistency of “sequential consistency”
- **Isolation**: all operations act as if they were run by themselves
- **Durability**: all writes stay persistent!

- As opposed to “BASE” – Basically Available, Soft state, Eventual consistency
- What would a violation of each property look like?