Search Engine Architecture

CISC489/689-010, Lecture #2

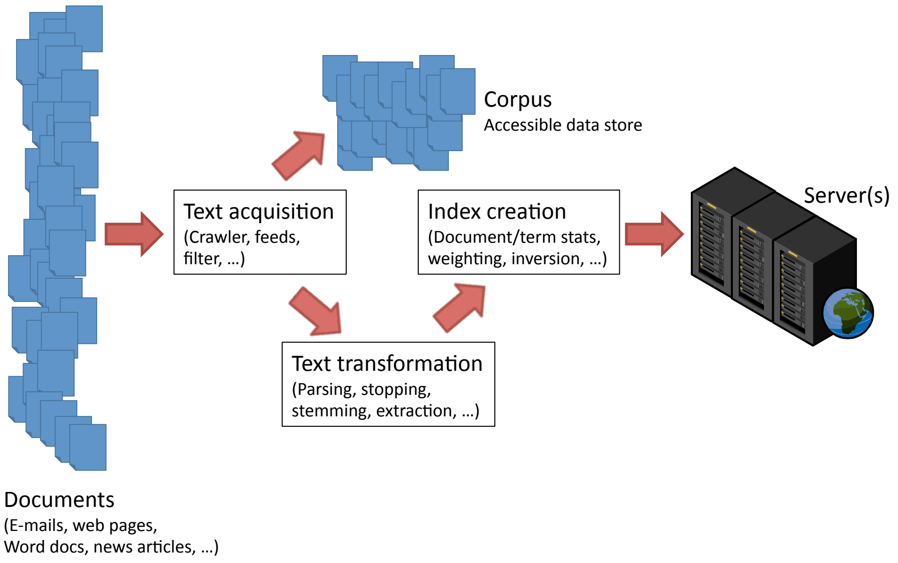
Wednesday, Feb. 11

Ben Carterette

Search Engine Architecture

- A software architecture consists of software components, the interfaces provided by those components, and the relationships between them
  - describes a system at a particular level of abstraction
- Architecture of a search engine determined by 2 requirements
  - effectiveness (quality of results) and efficiency (response time and throughput)

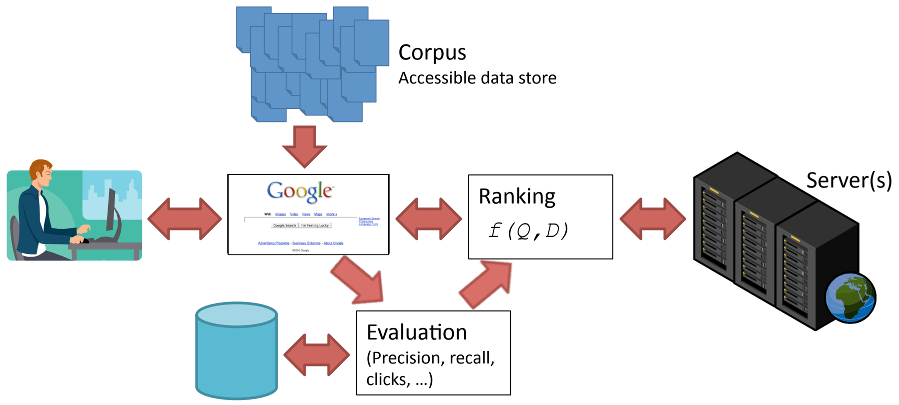
Indexing Process


Corpus
Accessible data store
Text acquisition
(Crawler, feeds, filter, …)
Index creation
(Document/term stats, weighting, inversion, …)
Server(s)
Text transformation
(Parsing, stopping, stemming, extraction, …)
Documents
(E-mails, web pages, Word docs, news articles, …)

Indexing Process

- Text acquisition
  - identifies and stores documents for indexing
- Text transformation
  - transforms documents into *index terms* or *features*
- Index creation
  - takes index terms and creates data structures (*indexes*) to support fast searching

Query Process



Query Process

- User interaction
  - supports creation and refinement of query, display of results
- Ranking
  - uses query and indexes to generate ranked list of documents
- Evaluation
  - monitors and measures effectiveness and efficiency (primarily offline)

Details: Text Acquisition

- Crawler
  - Identifies and acquires documents for search engine
  - Many types – web, enterprise, desktop
  - Web crawlers follow *links* to find documents
    - Must efficiently find huge numbers of web pages (*coverage*) and keep them up-to-date (*freshness*)
    - Single site crawlers for *site search*
    - *Topical* or *focused* crawlers for vertical search
  - *Document* crawlers for enterprise and desktop search
    - Follow links and scan directories

Text Acquisition

- Feeds
  - Real-time streams of documents
    - e.g., web feeds for news, blogs, video, radio, tv
  - RSS is common standard
    - RSS "reader" can provide new XML documents to search engine
- Conversion
  - Convert variety of documents into a consistent text plus metadata format
    - e.g. HTML, XML, Word, PDF, etc. → XML
  - Convert text encoding for different languages
    - Using a Unicode standard like UTF-8

Text Acquisition

- Document data store
  - – Stores text, metadata, and other related content for documents
    - Metadata is information about document such as type and creation date
    - Other content includes links, anchor text
  - – Provides fast access to document contents for search engine components
    - e.g. result list generation
  - – Could use relational database system
    - More typically, a simpler, more efficient storage system is used due to huge numbers of documents

Text Transformation

- Parser
  - – Processing the sequence of text *tokens* in the document to recognize structural elements
    - e.g., titles, links, headings, etc.
  - – *Tokenizer* recognizes “words” in the text
    - must consider issues like capitalization, hyphens, apostrophes, non-alpha characters, separators
  - – *Markup languages* such as HTML, XML often used to specify structure
    - *Tags* used to specify document *elements*
      - – E.g., <h2> Overview </h2>
    - Document parser uses *syntax* of markup language (or other formatting) to identify structure

Text Transformation

- Stopping
  - Remove common words
    - e.g., “and”, “or”, “the”, “in”
  - Some impact on efficiency and effectiveness
  - Can be a problem for some queries
- Stemming
  - Group words derived from a common *stem*
    - e.g., “computer”, “computers”, “computing”, “compute”
  - Usually effective, but not for all queries
  - Benefits vary for different languages

Text Transformation

- Link Analysis
  - Makes use of *links* and *anchor text* in web pages
  - Link analysis identifies *popularity* and *community* information
    - e.g., PageRank
  - Anchor text can significantly enhance the representation of pages pointed to by links
  - Significant impact on web search
    - Less importance in other applications

Text Transformation

- Information Extraction
  - – Identify classes of index terms that are important for some applications
  - – e.g., *named entity recognizers* identify classes such as *people*, *locations*, *companies*, *dates,* etc.
- Classifier
  - – Identifies class-related metadata for documents
    - i.e., assigns labels to documents
    - e.g., topics, reading levels, sentiment, genre
  - – Use depends on application

Index Creation

- Document Statistics
  - – Gathers counts and positions of words and other features
  - – Used in ranking algorithm
- Weighting
  - – Computes weights for index terms
  - – Used in ranking algorithm
  - – e.g., *tf.idf* weight
    - Combination of *term frequency* in document and *inverse document frequency* in the collection

Index Creation

- Inversion
  - Core of indexing process
  - Converts document-term information to term-document for indexing
    - Difficult for very large numbers of documents
  - Format of inverted file is designed for fast query processing
    - Must also handle updates
    - Compression used for efficiency

Index Creation

- Index Distribution
  - Distributes indexes across multiple computers and/or multiple sites
  - Essential for fast query processing with large numbers of documents
  - Many variations
    - Document distribution, term distribution, replication
  - *P2P* and *distributed IR* involve search across multiple sites

User Interaction

- Query input
  - Provides interface and parser for *query language*
  - Most web queries are very simple, other applications may use forms
  - Query language used to describe more complex queries and results of query transformation
    - e.g., Boolean queries, Indri and Galago query languages
    - similar to SQL language used in database applications
    - IR query languages also allow content and structure specifications, but focus on content

User Interaction

- Query transformation
  - Improves initial query, both before and after initial search
  - Includes text transformation techniques used for documents
  - *Spell checking* and *query suggestion* provide alternatives to original query
  - *Query expansion* and *relevance feedback* modify the original query with additional terms

User Interaction

- Results output
  - Constructs the display of ranked documents for a query
  - Generates *snippets* to show how queries match documents
  - *Highlights* important words and passages
  - Retrieves appropriate *advertising* in many applications
  - May provide *clustering* and other visualization tools

Ranking

- Scoring
  - Calculates scores for documents using a ranking algorithm
  - Core component of search engine
  - Basic form of score is $f(Q,D) = \sum_{t \in Q} q_t d_t$
    - q_t and d_t are query and document term weights for term t
  - Many variations of ranking algorithms and retrieval models

Ranking

- Performance optimization
  - Designing ranking algorithms for efficient processing
    - *Term-at-a time* vs. *document-at-a-time* processing
    - *Safe* vs. *unsafe* optimizations
- Distribution
  - Processing queries in a distributed environment
  - *Query broker* distributes queries and assembles results
  - *Caching* is a form of distributed searching

Evaluation

- Logging
  - Logging user queries and interaction is crucial for improving search effectiveness and efficiency
  - *Query logs* and *clickthrough data* used for query suggestion, spell checking, query caching, ranking, advertising search, and other components
- Ranking analysis
  - Measuring and tuning ranking effectiveness
- Performance analysis
  - Measuring and tuning system efficiency

How Does It *Really* Work?

- This course explains these components of a search engine in more detail
- Often many possible approaches and techniques for a given component
  - Focus is on the most important alternatives
  - i.e., explain a small number of approaches in detail rather than many approaches
  - "Importance" based on research results and use in actual search engines
  - Alternatives described in references

Text Acquisition

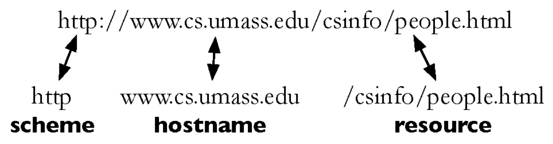
Web Crawling, Feeds, and Storage

Web Crawler

- Finds and downloads web pages automatically
  - provides the collection for searching
- Web is huge and constantly growing
- Web is not under the control of search engine providers
- Web pages are constantly changing
- Crawlers also used for other types of data

Retrieving Web Pages

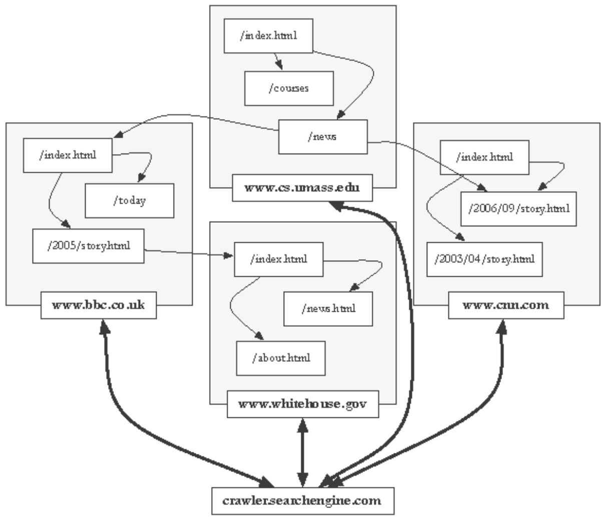
- Every page has a unique *uniform resource locator* (URL)
- Web pages are stored on web servers that use HTTP to exchange information with client software
- e.g.,



Retrieving Web Pages

- Web crawler client program connects to a *domain name system* (DNS) server
- DNS server translates the hostname into an *internet protocol* (IP) address
- Crawler then attempts to connect to server host using specific *port*
- After connection, crawler sends an HTTP request to the web server to request a page
  - usually a GET request

Crawling the Web

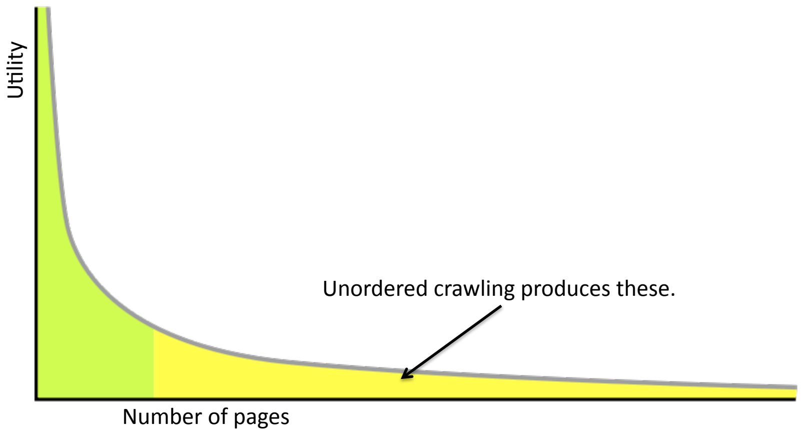


Web Crawler

- Starts with a set of *seeds*, which are a set of URLs given to it as parameters
- Seeds are added to a URL request queue
- Crawler starts fetching pages from the request queue
- Downloaded pages are parsed to find link tags that might contain other useful URLs to fetch
- New URLs added to the crawler's request queue, or *frontier*
- Continue until no more new URLs or disk full

Web Crawling

- The "long tail" of web pages.



Web Crawling

- Ordering URLs
  - Crawl URLs in some order of “importance”
  - “Random surfer” model:
    - A user starts on a page and randomly clicks links.
    - Occasionally switches to a different page with no click.
    - What is the probability the user will land on any given page?
  - Higher probability → greater importance.
  - PageRank

Web Crawling

- Web crawlers spend a lot of time waiting for responses to requests
- To reduce this inefficiency, web crawlers use threads and fetch hundreds of pages at once
- Crawlers could potentially flood sites with requests for pages
- To avoid this problem, web crawlers use *politeness policies*
  - e.g., delay between requests to same web server

Controlling Crawling

- Even crawling a site slowly will anger some web server administrators, who object to any copying of their data
- Robots.txt file can be used to control crawlers

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

User-agent: FavoredCrawler
Disallow:

Sitemap: http://mysite.com/sitemap.xml.gz
```

Simple Crawler Thread

```
procedure CrawlerThread(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
```

Freshness

- Web pages are constantly being added, deleted, and modified
- Web crawler must continually revisit pages it has already crawled to see if they have changed in order to maintain the *freshness* of the document collection
  - *stale* copies no longer reflect the real contents of the web pages

Freshness

- HTTP protocol has a special request type called HEAD that makes it easy to check for page changes
  - returns information about page, not page itself

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
```

Freshness

- Not possible to constantly check all pages
  - must check important pages and pages that change frequently
- Freshness is the proportion of pages that are fresh
- Optimizing for this metric can lead to bad decisions, such as not crawling popular sites
- *Age* is a better metric

Age

- Expected age of a page *t* days after it was last crawled:

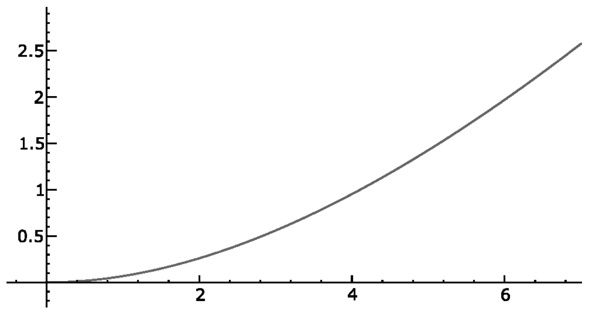
$$\text{Age}(\lambda, t) = \int_0^t P(\text{page changed at time } x)(t - x)dx$$

- Web page updates follow the Poisson distribution on average
  - time until the next update is governed by an exponential distribution

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

Age

- Older a page gets, the more it costs not to crawl it
  - e.g., expected age with mean change frequency λ = 1/7 (one change per week)



Focused Crawling

- Attempts to download only those pages that are about a particular topic
  - used by *vertical search* applications
- Rely on the fact that pages about a topic tend to have links to other pages on the same topic
  - popular pages for a topic are typically used as seeds
- Crawler uses *text classifier* to decide whether a page is on topic

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

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 ...

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

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

Large Files

- Store many documents in large files, rather than each document in a file
  - avoids overhead in opening and closing files
  - reduces seek time relative to read time
- Compound documents formats
  - used to store multiple documents in a file
  - e.g., TREC Web, Wikipedia XML