Web Services and Web Data

XJCO 3011

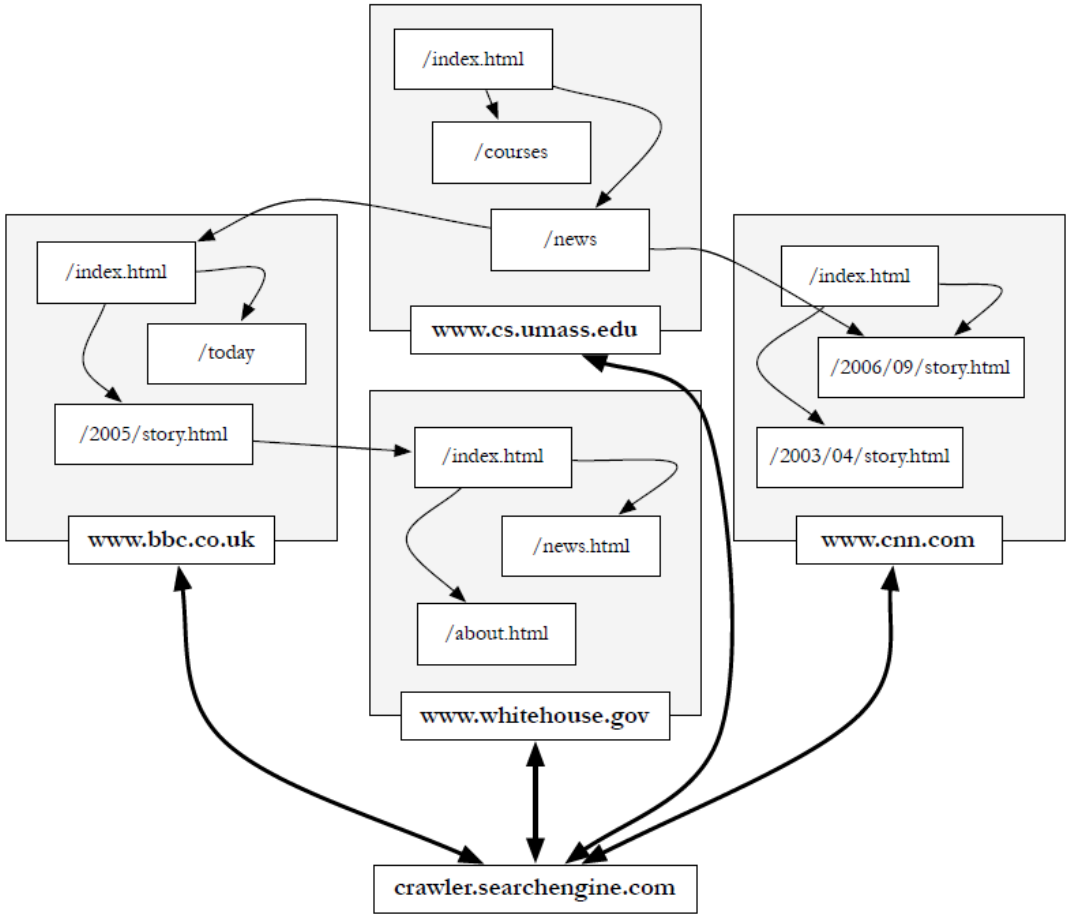
Session 9. Web Crawling

Crawlers for Search Engines

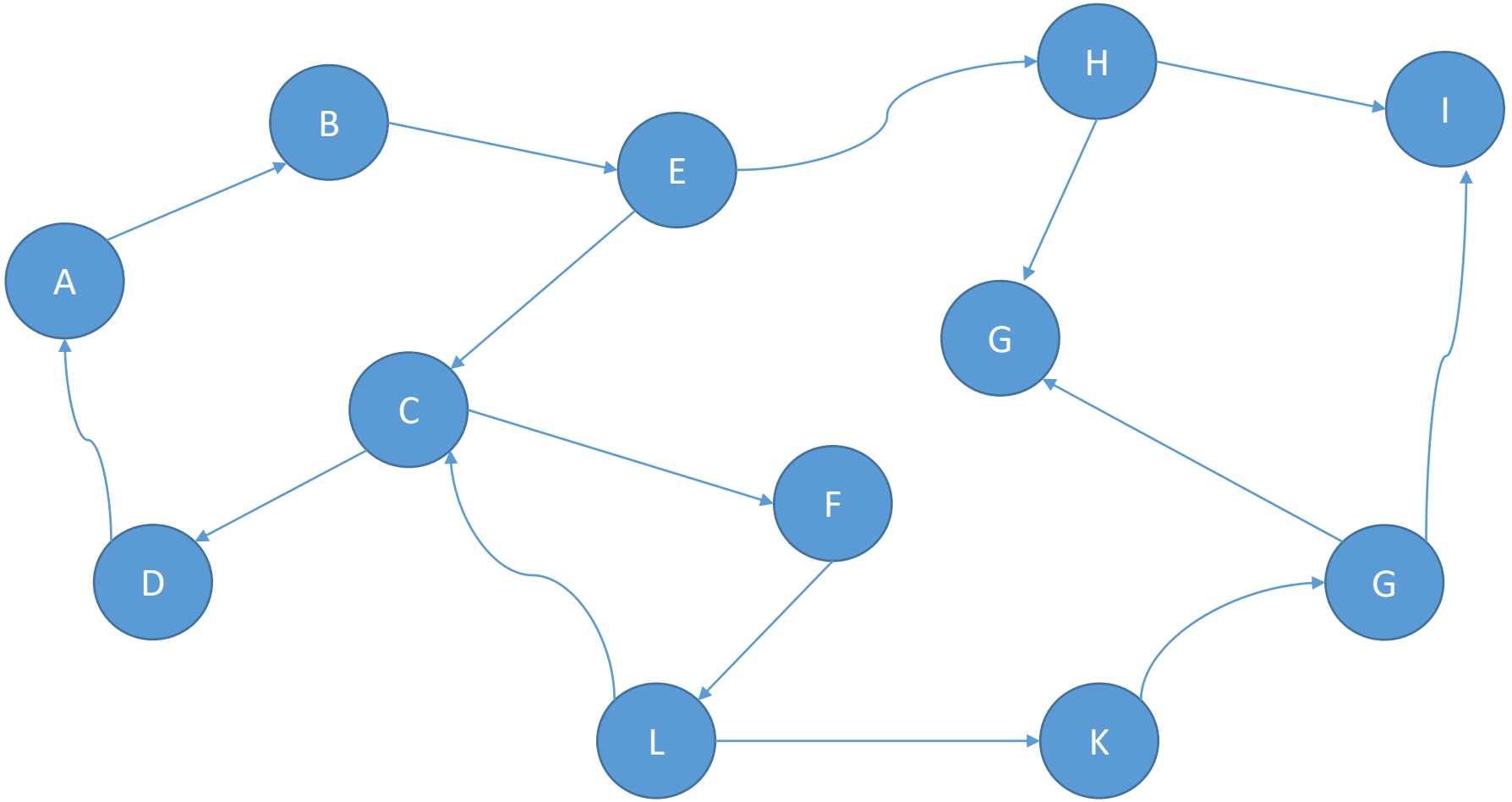
- To build a search engine, you first need a copy of the pages that you want to search. Unlike some of the other sources of text we will consider later, web pages are particularly easy to copy, since they are meant to be retrieved over the Internet by browsers. This instantly solves one of the major problems of getting information to search, which is how to get the data from the place it is stored to the search engine.
- Finding and downloading web pages automatically is called crawling or spidering, and a program that downloads pages is called a web crawler.
- There are some unique challenges to crawling web pages:
  1. The sheer scale of the Web (billions of pages on the Internet)
  2. Pages are constantly being created and removed.
  3. Web pages are usually not under the control of the people building the search engine.
  4. Website owners do not like crawlers.
  5. Some of the data you want to copy may be available only by typing a request into a form, which is a difficult process to automate.

How Web Crawlers Work

- The web crawler has two jobs: downloading pages and finding URLs.
- The crawler starts with a set of seeds, which are a set of URLs given to it (the crawler) as parameters.
- These seeds are added to a URL request queue.
- The crawler starts fetching pages from the request queue. Once a page is downloaded, it is parsed to find link tags that might contain other useful URLs to fetch.
- If the crawler finds a new URL that it has not seen before, it is added to the crawler's request queue, or frontier.
- The frontier may be a standard queue, or it may be ordered so that important pages move to the front of the list.
- This process continues until the crawler either runs out of disk space to store pages or runs out of useful links to add to the request queue.



Web Crawling is a Kind of Graph Traversal



- A web crawler starts by visiting a specific website or URL. The crawler then follows the links on that page to other pages, and continues to follow links until it has visited all the pages it needs to.

Web Crawlers Are Not Always Welcome

- Fetching hundreds of pages at once is good for the web crawler, but not good for the web server on the other end.
- If the web server is not very powerful, it might spend all of its time handling requests from crawlers instead of handling requests from 'real' users
- This kind of behaviour from web crawlers can make web server administrators very angry
- They can consequently block the IP address from which the crawler is working.
- To avoid this problem, web crawlers use politeness policies.
  1. Reasonable web crawlers do not fetch more than one page at a time from a particular web server.
  2. In addition, web crawlers wait at least a few seconds, and sometimes minutes, between requests to the same web server.
  3. To support this, the request queue is logically split into a single queue per web server.
  4. The crawler is free to read page requests only from queues that haven't been accessed within the specified politeness window.

The request queue can be very large

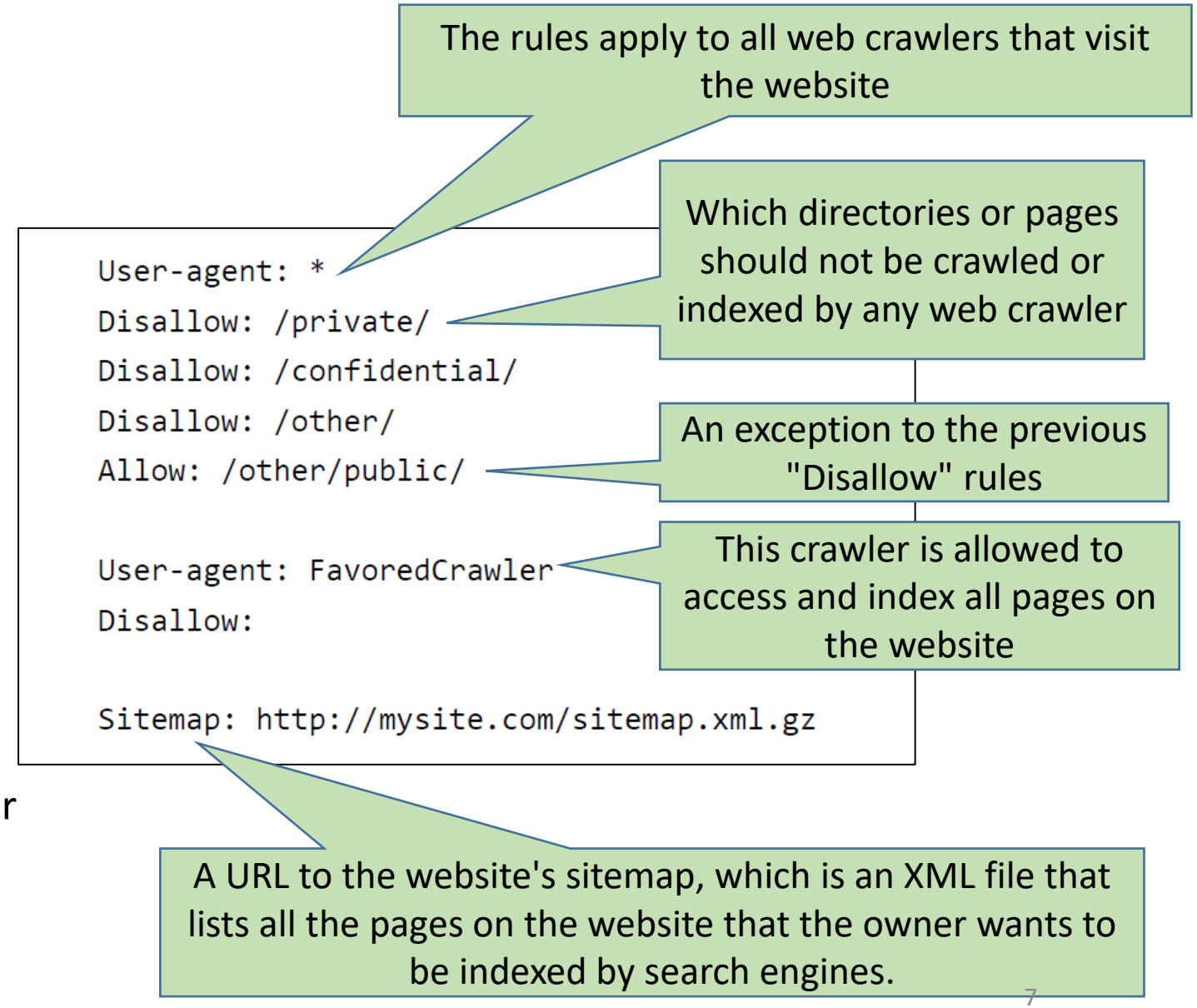
- Suppose a web crawler can fetch 100 pages each second, and that its politeness policy dictates that it cannot fetch more than one page each 30 seconds from a particular web server.

- The web crawler needs to have URLs from at least 3,000 different web servers in its request queue in order to achieve the required 100 pages/second throughput.

- Since many URLs will come from the same servers, the request queue needs to have tens of thousands of URLs in it before a crawler can reach its peak throughput.

The robots.txt file

- Even crawling a site slowly will anger some web server administrators who object to any copying of their data. Web server administrators who feel this way can store a file called /robots.txt on their web servers.
- The file is split into blocks of commands that start with a User-agent: specification.
- The User-agent: line identifies a crawler, or group of crawlers, affected by the following rules.
- Following this line are Allow and Disallow rules that dictate which resources the crawler is allowed to access.
- An optional Crawl-delay (non standard extension) can be included.



Detecting Robots

Websites can 'catch' unwanted crawlers in many ways:

- Very quick page requests to the same website is the first and most obvious feature of a badly designed robot. Hence, a robot should always respect the fair use policy of websites and servers by allowing sufficient and reasonable time between requests.
- A hidden trap link in some pages (for example /trap_unwanted_robots) that human users cannot see (and hence will not click on). However, When a naive robot 'steps' into this 'booby trap', it is detected and further action can be taken against the robot. Robots can avoid this kind of trap by reading the robots.txt file, which usually points to these robot trap links.
- A dynamically created link to a sequence fictitious resources that never terminates, for example /followme/1/1, /followme/1/2, /followme/1/3, …. This will cause naive robots to go into infinite recursion, and destabilizes the robot. Crawlers, can avoid this by having a limit on the maximum number of levels it can go deep down the same page hierarchy.

- The User-agent HTTP header field in a request usually contains information about the user agent (web browser). Unwanted bots which identify themselves through this field can be black listed. Crawlers can avoid this by changing their User-agent name (spoofing).
- Crawlers that make requests to the same website at a regular and monotonous rate can also be easily detected, hence sophisticated bots should randomize the timing of page requests.

```
# section 1
User-agent: BadCrawler
Disallow: /

# section 2
User-agent: *
Crawl-delay: 5
Disallow: /trap

# section 3
Sitemap: http://example.webscraping.com/sitemap.xml
```

How do you know if your robot is being blocked

- CAPTCHA pages.
- Unusual content delivery delay.
- Frequent appearance of these status codes is also indication of blocking:
  - 301 Moved Temporarily
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
  - 408 Request Timeout
  - 429 Too Many Requests
  - 503 Service Unavailable

A 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
```

- The crawler first retrieves a website from the frontier.
- The crawler then identifies the next URL in the website's queue.
- The crawler checks to see if the URL is okay to crawl according to the website's robots.txt file.
- If it can be crawled, the crawler fetches the document contents. This is the most expensive part of the loop, and the crawler thread may block here for many seconds.
- Once the text has been retrieved the document is stored in the database, the document text is then parsed so that other URLs can be found.
- These URLs are added to the frontier, which adds them to the appropriate website queues.
- When all this is finished, the website object is returned to the frontier, which takes care to enforce its politeness policy by not giving the website to another crawler thread until an appropriate amount of time has passed.

Freshness

- Web pages are constantly being added, deleted, and modified.
- To keep an accurate view of the Web, a 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.
- The opposite of a fresh copy is a stale copy
- The HTTP HEAD request, which only returns header information, is useful for checking for page changes.
- The Last-Modified value indicates the last time the page content was changed. This compared with the date it received from a previous GET request for the same page.

- A HEAD request reduces the cost of checking on a page, but does not eliminate it. It simply is not possible to check every page every minute.
- It does little good to continuously check sites that are rarely updated.
- Hence, a crawler can be made to learn about the frequency of change of a web page, and guess when it is time to check the freshness of a page.

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

Sitemaps

- A robots.txt file can contain a reference to a sitemap. A sitemap contains a list of URLs and data about those URLs, such as modification time and modification frequency.
- There are three URL entries shown in the example sitemap.
- Each one contains a URL in a loc tag.
- The changefreq tag indicates how often this resource is likely to change.
- The lastmod tag indicates the last time it was changed.
- The first entry also includes a priority tag with a value of 0.7, which is higher than the default of 0.5. This tells crawlers that this page is more important than other pages on this site.
- A sitemap tells search engines about pages it might not otherwise find (e.g. because they require form entry).

```
<?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>
```

Sitemap XML tags

Attribute		Description
`<urlset>`	required	Encapsulates the file and references the current protocol standard.
`<url>`	required	Parent tag for each URL entry. The remaining tags are children of this tag.
`<loc>`	required	URL of the page. This URL must begin with the protocol (such as http) and end with a trailing slash, if your web server requires it. This value must be less than 2,048 characters.
`<lastmod>`	optional	The date of last modification of the file. This date should be in W3C Datetime format. This format allows you to omit the time portion, if desired, and use YYYY-MM-DD. Note that this tag is separate from the If-Modified-Since (304) header the server can return, and search engines may use the information from both sources differently.
`<changefreq>`	optional	How frequently the page is likely to change. This value provides general information to search engines and may not correlate exactly to how often they crawl the page. Valid values are: • always • hourly • daily • weekly • monthly • yearly • never The value "always" should be used to describe documents that change each time they are accessed. The value "never" should be used to describe archived URLs. Please note that the value of this tag is considered a *hint* and not a command. Even though search engine crawlers may consider this information when making decisions, they may crawl pages marked "hourly" less frequently than that, and they may crawl pages marked "yearly" more frequently than that. Crawlers may periodically crawl pages marked "never" so that they can handle unexpected changes to those pages.
`<priority>`	optional	The priority of this URL relative to other URLs on your site. Valid values range from 0.0 to 1.0. This value does not affect how your pages are compared to pages on other sites—it only lets the search engines know which pages you deem most important for the crawlers. The default priority of a page is 0.5. Please note that the priority you assign to a page is not likely to influence the position of your URLs in a search engine's result pages. Search engines may use this information when selecting between URLs on the same site, so you can use this tag to increase the likelihood that your most important pages are present in a search index. Also, please note that assigning a high priority to all of the URLs on your site is not likely to help you. Since the priority is relative, it is only used to select between URLs on your site.

Distributed Crawling

- For crawling individual websites, a single computer is sufficient.
- However, crawling the entire Web requires many computers devoted to crawling to:
  1. put the crawler closer to the sites it crawls,
  2. reduce the number of sites the crawler has to remember (the queue and the visited list), and
  3. reduce individual computing resources, including CPU resources for parsing and network bandwidth for crawling pages. Crawling a large portion of the Web is too much work for a single computer to handle.
- In a distributed crawler there are many queues. The distributed crawler uses a hash function, on the URL's host part, to assign URLs to crawling computers.
- When a crawler sees a new URL, it computes a hash function on that URL to decide which crawling computer is responsible for it.
- These URLs are gathered in batches, then sent periodically to reduce the network overhead of sending a single URL at a time.

The Conversion Problem

- Search engines are built to search through text.
- Unfortunately, text is stored on computers in hundreds of incompatible file formats, such as plain text, RTF, HTML, XML, Microsoft Word, ODF (Open Document Format) and PDF (Portable Document Format), to name just a few.
- It is not uncommon for a commercial search engine to support more than a hundred file types.
- The most common way to handle a new file format is to use a conversion tool that converts the document content into a tagged text format such as HTML or XML.
- Documents could be converted to plain however this would strip the file of information about headings and font sizes that could be useful to the indexer.
- Headings and bold text tend to contain words that describe the document content well, so these words must be given preferential treatment during scoring.
  - Scoring: used to determine the order in which search results are presented. The score is usually based on the importance of the keywords found in the document

That's it Folks

Further Reading

Chapter 3: Search Engines Information Retrieval in Practice by W. Bruce Croft, Donald Metzler, and Trevor Strohman