CS 50:
Software Design and Implementation

Tiny Search Engine

Agenda

1. Review web concepts
2. Web search overview
3. Crawler demo
4. Activity

CS10 Review: To transfer data between computers we use pre-defined protocols

Network protocols

- Network protocols define how data will be exchanged so everyone knows the "rules"
- There are dozens of protocols used for different purposes:
  - TCP/IP, FTP
  - Wi-Fi, Bluetooth
- HyperText Transfer Protocol (HTTP) is the protocol commonly used by the World Wide Web to get HyperText Markup Language (HTML) documents that describe how to render a web page
- We use a Uniform Resource Locator (URL) to specify what page we want to get:

  http://www.cs.dartmouth.edu/~tjp/cs10/index.html

Protocol: how we will talk (http)

Computer that has data

Directory where data located

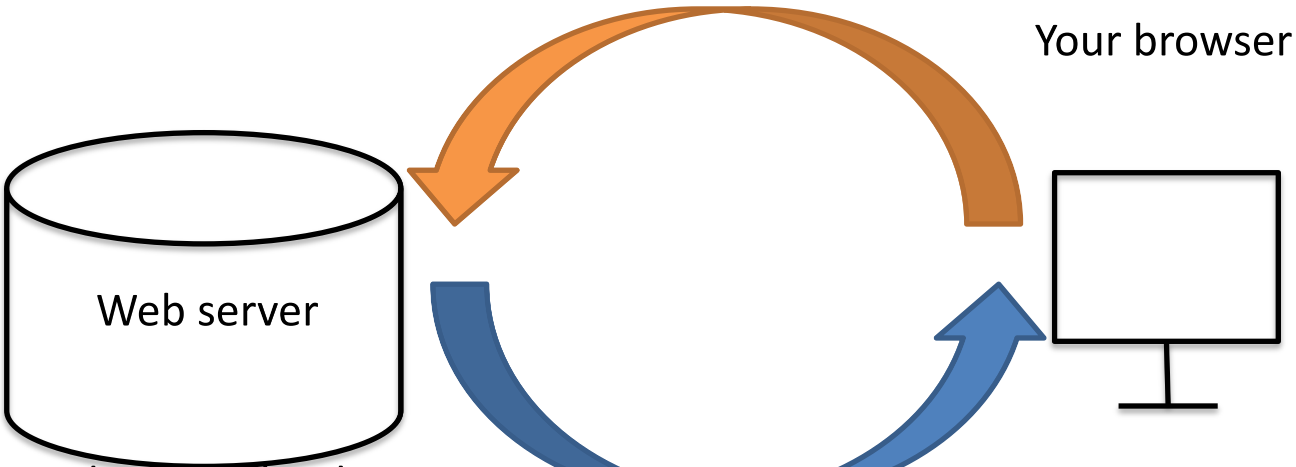
File (assume index.html or index.php if not provided)

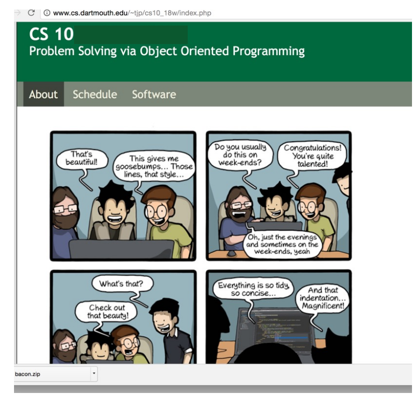
Client makes a request to a Server for a web page; Server responds to request

Process

Request:
http://www.cs.dartmouth.edu/~tjp/cs10/index.html

Your browser





Browser interprets HTML text and renders page

cs.dartmouth.edu

Response:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="content-type" content="text/html;charset=utf-8" />
<title>CS 10 | Problem solving              </title>
</head>

<body>
<div id="page">
<div id="header">
  <div id="title">CS 10</div>
  <div id="subtitle">Problem Solving via Object Oriented Programming</div>
</div> …
```

Big idea:
- **Client makes request to server for web page**
- **Server responds to client's request**

A web page is simply a text document with a description of what to display on the screen (and maybe some Javascript for user interaction) in a format called HTML

Web pages are just a text document written in HTML, which uses tags

```
<!DOCTYPE html>
<html>
   <body>
      <h1>My First Heading</h1>

      My first paragraph

      <a href="page2.html">Link to page2</a>
   </body>
</html>
```

Tag declares this is an HTML file

HTML and nested body section

<h1> mean Heading size 1 (big)
</h1> means end tag <h1>

Our search engine will look for words outside of tags for queries

You will not write any HTML for CS50

Tags are mainly for formatting

We care about <a> tags (anchor tags)

- These are links to other pages
- href give URL to another page
- We will use a "test Internet" on plank

We will provide you with a C function (getNextURL) to parse the HTML

Feel free to write your own if you prefer!

Agenda

1. Review web concepts
2. Web search overview
3. Crawler demo
4. Activity

How a search engine works

https://www.youtube.com/watch?v=BNHR6IQJGZs

Our search engine will proceed in three stages: crawler->indexer->querier

Crawler

- Start from a specified "seed" URL
- Fetch page pointed to by seed URL
- Make a log of this page, saving URL, page depth (how far from seed, where seed has depth=0), and HTML of the page
- Scan page for links to other pages
- Follow links and repeat
- Do not repeat page

Our search engine will proceed in three stages: crawler->indexer->querier

Crawler

- Start from a specified “seed” URL
- Fetch page pointed to by seed URL
- Make a log of this page, saving URL, page depth (how far from seed, where seed has depth=0), and HTML of the page
- Scan page for links to other pages
- Follow links and repeat
- Do not repeat page

Indexer

- Start with results of crawler’s logs
- Process logs to create index where given a word, can find all pages that contain that word (we will use Lab 3 hash tables and counters)
- Store index

Our search engine will proceed in three stages: crawler->indexer->querier

Crawler

- Start from a specified "seed" URL
- Fetch page pointed to by seed URL
- Make a log of this page, saving URL, page depth (how far from seed, where seed has depth=0), and HTML of the page
- Scan page for links to other pages
- Follow links and repeat
- Do not repeat page

Indexer

- Start with results of crawler's logs
- Process logs to create index where given a word, can find all pages that contain that word (we will use Lab 3 hash tables and counters)
- Store index

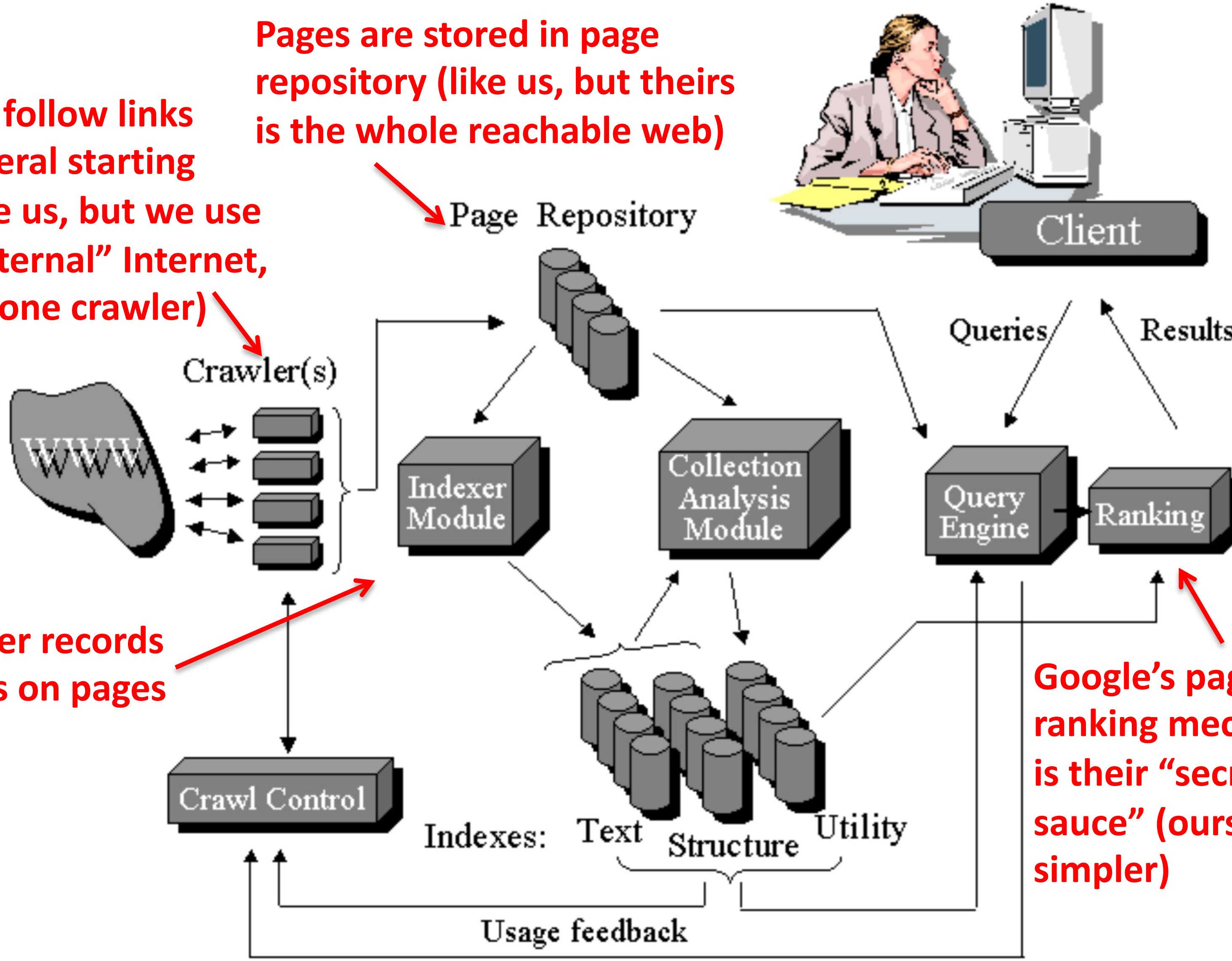
Querier

- Start with indexer's stored index
- Get user's query which may contain AND as well as OR queries
- Search index for pages with highest matching score
- Return results in sorted order

Google's implementation is more complicated than ours

Crawlers follow links from several starting URLs (like us, but we use small "internal" Internet, we have one crawler)

Pages are stored in page repository (like us, but theirs is the whole reachable web)



Indexer records words on pages

Google's page ranking mechanism is their "secret sauce" (ours is simpler)

Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas Paepcke, and Sriram Raghavan. *Searching the Web.* ACM Transactions on Internet Technology 1, 1 (Aug. 2001), 2–43.

Agenda

1. Review web concepts
2. Web search overview
3. Crawler demo
4. Activity

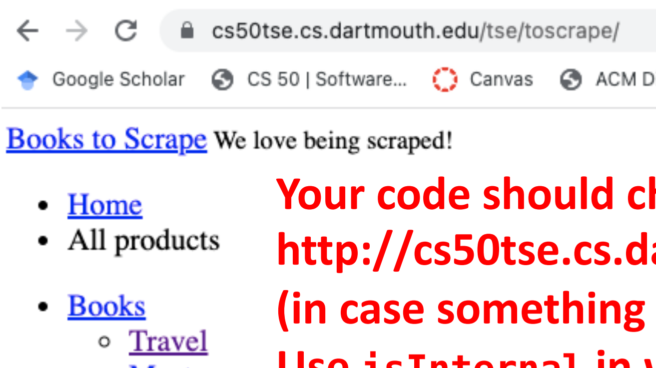
We have established a “test Internet” to use for the TSE project

http://cs50tse.cs.dartmouth.edu/tse/(letters|toscrape|wikipedia)

- letters (small)
- toscrape (medium)
- wikipedia (large)

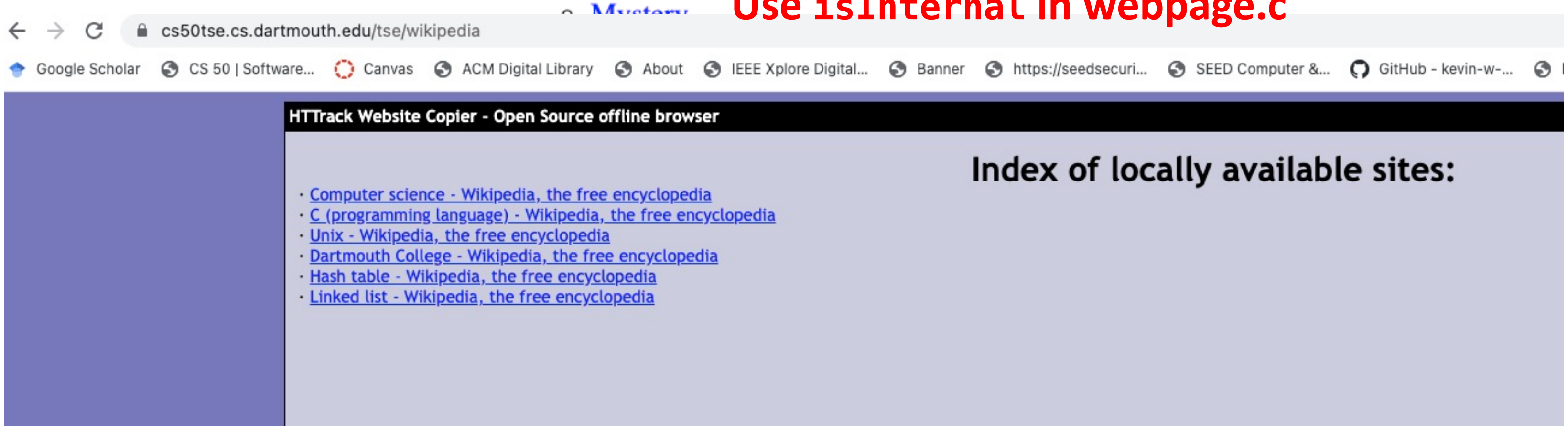
letters

toscrape



Your code should check that all URLs start with http://cs50tse.cs.dartmouth.edu/tse
(in case something goes wrong, we only crash plank!)
Use `isInternal` in webpage.c

wikipedia



Crawler starts at a seed URL and indexes reachable pages to a given depth

Your executable — **Seed URL** — **Directory to store results**

```
$ ./crawler http://cs50tse.cs.dartmouth.edu/tse/letters/index.html ../data 2
 0   Fetched: http://cs50tse.cs.dartmouth.edu/tse/letters/index.html
 0  Scanning: http://cs50tse.cs.dartmouth.edu/tse/letters/index.html
 0     Found: http://cs50tse.cs.dartmouth.edu/tse/letters/A.html
 0     Added: http://cs50tse.cs.dartmouth.edu/tse/letters/A.html
 1    Fetched: http://cs50tse.cs.dartmouth.edu/tse/letters/A.html
 1   Scanning: http://cs50tse.cs.dartmouth.edu/tse/letters/A.html
 1      Found: https://en.wikipedia.org/wiki/Algorithm
 1   IgnExtrn: https://en.wikipedia.org/wiki/Algorithm
 1      Found: http://cs50tse.cs.dartmouth.edu/tse/letters/B.html
 1      Added: http://cs50tse.cs.dartmouth.edu/tse/letters/B.html
 1      Found: http://cs50tse.cs.dartmouth.edu/tse/letters/index.html
 1    IgnDupl: http://cs50tse.cs.dartmouth.edu/tse/letters/index.html
 2     Fetched: http://cs50tse.cs.dartmouth.edu/tse/letters/B.html
```

Depth to crawl

Started at index.html Then found A.html

Ignored Wikpedia page (not in our test Internet!)

Found B.html

Ignored index.html because we've already crawled it

The crawler stores data in a directory, each reachable page has a file in that directory

```
$ cd ../data
data$ ls
1  2  3
data$ cat 1
http://cs50tse.cs.dartmouth.edu/tse/letters/index.html
0
<html>
<title>home</title>
This is the home page for a CS50 TSE playground.
<a href=A.html>A</a>
</html>


data$ cat 2
http://cs50tse.cs.dartmouth.edu/tse/letters/A.html
1
<html>
<title>A</title>
A is for <a href=https://en.wikipedia.org/wiki/Algorithm>Algorithm</a>.
<a href=B.html>B</a>
<a href=index.html>home</a>
</html>
```

List files in data directory
Here three pages were indexed, named 1…3 in order of discovery

For each web page discovered:
Line 1: URL
Line 2: depth
Line 3: HTML for page

Do not make a file for the same URL multiple times!

Agenda

1. Review web concepts
2. Web search overview
3. Crawler demo
4. Activity

Activity

In your group, start thinking about a design for the *Crawler* portion of the TSE. Consider the following informal description of the Crawler:

- It takes three parameters: the URL for a web page to use as a starting point (*seed*) for the crawl, the *maximum depth* it should crawl from that seed, and the name of a *directory* where it can cache copies of the web pages it crawls
- It should start from a given URL called the *seed*. The web page at that URL is said to be at depth 0.
- It should *explore* that URL; that is, it should download the web page at that URL, and scan that page's HTML for embedded links to URLs. (Assume you are given a function that can pick URLs out of HTML). When exploring a page at depth *d*, its embedded URLs refer to pages that are said to be at depth *d+1*
- Ignore URLs that don't point at HTML
- Ignore URLs at depth greater than *maxDepth*
- Explore each non-ignored URL by downloading its HTML and scanning that HTML for URLs, as above
- For each page it explores, it should create one file that contains the URL of that page, its depth in the crawl, and the HTML for that page.

Discuss how you could structure a crawler to accomplish the above goals - probably two nested loops - and leverage your Lab 3 data structures.

Two big questions for data structures to use

What should you use to:

1. Keep track of web sites to explore?
2. Make sure you don’t visit a site more than one time?