Announcements

- Homework 7 due today
- Lab 9 reports due today
Today’s Plan

- Final project design tips
- Server programming
Famous?

- **Lycos**, 1994
- **Alta Vista**, 1995
- **Yahoo**, 1995
- **Backrub**, 1998
Famous?

Lycos, 1994
Alta Vista, 1995
Yahoo, 1995
Backrub, 1998

Lycos has recently launched a suite of wearable devices

Find Out More
Famous?

Lycos, 1994
Alta Vista, 1995
Yahoo, 1995
Backrub, 1998
Famous?

- Lycos, 1994
- Alta Vista, 1995
- Yahoo, 1995
- Backrub, 1998
Famous?

A Eulogy For AltaVista, The Google Of Its Time

Danny Sullivan on June 28, 2013 at 6:53 pm

Goodbye AltaVista. You deserved better than this. Better than the one-sentence send-off Yahoo gave you today, when announcing your July 8 closure date. But then again, you always were the bright child neglected by your parents.

The Amazing AltaVista

You appeared on the search engine scene in December 1995. You made us go “woah” when you arrived. You did that by indexing around 20 million web pages, at a time when indexing 2 million web pages was considered to be big.

Today, of course, pages get indexed in the billions, the tens of billions or more. But in 1995, 20 million was huge. Existing search engines like Lycos, Excite & InfoSeek (to name only a few) didn’t quite know what hit them. With so many pages, you seemed to find stuff they and others didn’t.
Famous?

- Lycos, 1994
- Alta Vista, 1995
- Yahoo, 1995
- Backrub, 1998
BackRub is a "web crawler" which is designed to traverse the web.

Currently we are developing techniques to improve web search engines. We will make various services available as soon as possible.

Sorry, many services are unavailable due to a local network failure beyond our control. We are working to fix the problem.
The Anatomy of a Large-Scale Hypertextual Web Search Engine

Sergey Brin and Lawrence Page
{sergey, page}@cs.stanford.edu
Computer Science Department, Stanford University, Stanford, CA 94305

Abstract

In this paper, we present Google, a prototype of a large-scale search engine which makes heavy use of the structure present in hypertext. Google is designed to crawl and index the Web efficiently and produce much more satisfying search results than existing systems. The prototype with a full text and hyperlink database of at least 24 million pages is available at http://google.stanford.edu/

To engineer a search engine is a challenging task. Search engines index tens to hundreds of millions of web pages involving a comparable number of distinct terms. They answer tens of millions of queries every day. Despite the importance of large-scale search engines on the web, very little academic research has been done on them. Furthermore, due to rapid advance in the algorithmic techniques, most of the current search engines are only now beginning to be able to produce useful results.
HTML Document Structure

<HTML>
  <HEAD>
    <TITLE> something short but sweet </TITLE>
  </HEAD>
  <BODY>
    something not so short but sweet
  </BODY>
</HTML>
A BRIEF EXAMPLE

<html>
<head>
<TITLE>Tom Murtagh's Home</TITLE>
</head>
<body>
<H1>Tom Murtagh</H1>
<p>OK!!!  I've given in and made a home page.  
   Well, it's a start...
</p>
</body>
</html>
HTTP

GET /~tom/index.html

HTTP/1.0 200 Document ...
Clients vs. Servers
Clients vs. Servers

Role of clients – retrieve data from servers
1. Contact server
2. Send request(s)
3. Retrieve response(s)
4. Disconnect

Role of servers – “serve” data to clients
1. Patiently wait for client connections
2. Accept valid connection
3. Receive request(s)
4. Send data to clients
5. Close client connection
Clients vs. Servers

Role of clients – retrieve data from servers
1. Contact server (... = new NetConnection("cortland", 110); )
2. Send request(s) ( toServer.out.println("RETR ..."); )
3. Retrieve response(s) ( response = toServer.in.nextLine(); )
4. Disconnect ( toServer.close(); )

Role of servers – “serve” data to clients
1. Patiently wait for client connections
2. Accept valid connection
3. Receive request(s)
4. Send data to clients
5. Close client connection;
Clients vs. Servers

Role of clients - retrieve data from servers
1. Contact server (... = new NetConnection("cortland", 110); )
2. Send request(s) ( toServer.out.println("RETR ..."); )
3. Retrieve response(s) ( response = toServer.in.nextLine(); )
4. Disconnect ( toServer.close(); )

Role of servers - “serve” data to clients
1. Patiently wait for client connections
2. Accept valid connection
3. Receive request(s) ( request = fromClient.in.nextLine(); )
4. Send data to clients ( fromClient.out.println( response )
5. Close client connection ( fromClient.close() );
Clients vs. Servers

Role of clients – retrieve data from servers

1. Contact server (... = new NetConnection(“cortland”, 110); )
2. Send request(s) ( toServer.out.println(“RETR ...”); )
3. Retrieve response(s) ( response = toServer.in.nextLine(); )
4. Disconnect ( toServer.close(); )

Role of servers – “serve” data to clients

1. Patiently wait for client connections ( ??? )
2. Accept valid connection ( fromClient = ??? )
3. Receive request(s) ( request = fromClient.in.nextLine(); )
4. Send data to clients ( fromClient.out.println( response )
5. Close client connection ( fromClient.close() );
Port Numbers

- Servers “listen” for client connection requests on specific ports
  - Recall that POP email uses port 110
  - Ports are like phone number extensions
- In Squint, we indicate that a server wants to activate a port by saying:

  `TCPPort connectPort = new TCPPort( 110 );`
Waiting for a Call

A server can

- wait for a client to create a NetConnection, and
- get access to the NetConnection by saying:

```java
NetConnection fromClient = connectPort.acceptNetConnection();
```
Network Client Events

In our client code, we added a MessageListener to be notified when the server sent us a new message:

toServer.addMessageListener(this);

...  

public void dataAvailable() {
    response = toServer.in.nextLine();
    ...
}

We can ask Java to execute a method when our server receives a connection request:

```java
connectPort.addConnectionListener( this );

public void connectionEstablished( TCPPort p ) {
    NetConnection fromClient = connectPort.acceptNetConnection();
    ...
}
```
Accessing Files

Programs can access data in files using println and nextLine (just as programs access data sent through NetConnections):

```java
Scanner inFile = new Scanner( new File ( "file's name") );

while ( inFile.hasNextLine() ) {
    String currentLine = inFile.nextLine();
    ... code to process current line ...
}
```
We can ask Java to execute code that may fail while providing a backup plan:

```java
try {
    . . . risky code . . .
} catch (SomeTypeOfException ex) {
    . . . code to address problem . . .
}
```