PART II
Finding Needles in the World's Biggest Haystack

- or -

The Page Match & Page Rank Problems

We will look at how a search engine carries out an impossible task:

- Read your search word or phrase;
- search all 40 billion pages on the Internet;
- list all the pages that contain some or all of your phrase;
- rank these matches so the best matches are first;
- and do this in less than two seconds.

Topics for Part II

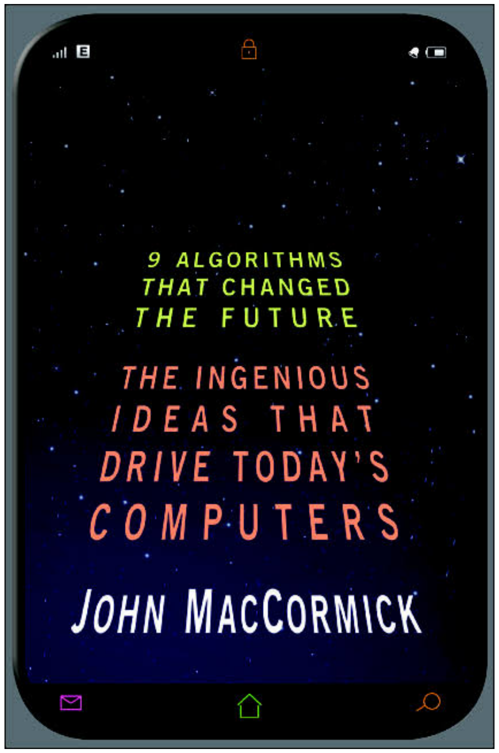
1. The PageMatch Problem investigates how your browser takes your internet search string, passes it to a search engine which returns an answer incredibly fast. We will see that this process involves a number of hidden tricks and indexes.
2. The PageRank Problem considers how it is possible, once many matching web pages have been found, to select the very few that correspond most closely to your search string.

Goals for this lecture

1. To realize that a search engine doesn't wait for your search string and then start checking one page after another on the web;
2. To realize that it's possible to create an index of the web, just like the index of a book;
3. To understand how Web crawling is done to create an index;
4. To recognize how information searches were done before the Internet.
5. To watch a video from a Google engineer explaining in general terms how Web searches work at Google.

Reading Assignment

Read Chapter 2, pages 10–23, “9 Algorithms that Changed the Future”.


9 ALGORITHMS THAT CHANGED THE FUTURE
THE INGENIOUS IDEAS THAT DRIVE TODAY'S COMPUTERS
JOHN MACCORMICK

What's the Internet?

Once, computers were simply stand-alone devices.

A computer could be used to create information and store it on its own hard drive.

I could use my computer to write a letter to you, but to send it, I'd have to print it out and put it in an envelope.

Or, if you were in the same office, I could copy my letter onto a removable disk, carry it to your computer, and load it onto your hard disk, so you could view it.

People very quickly invented ways of connecting all the computers in an office using cables, so that information could be moved from one hard disk to another by simple keyboard commands.

What's the Internet?

Gradually, it became clear that it was a good idea to connect all the computers in the whole world this way.

This required modifying computers to include an Internet port which could be connected by a thin cable to a local network, over which information could be sent and received.

It required building a network of Internet Servers, so that every message could be passed from its sender computer, through a sequence of servers, til it reached its recipient computer.

It required coming up with a universal system of numeric addresses, called the Internet Protocol or **IP**, so that each message included an exact address that the servers could read.

It required setting up Domain Name Servers (DNS) so that humans could specify an easy-to-remember address such as **facebook.com** or **fsu.edu**, but these would be automatically translated to numeric IP addresses.

What's the Internet?

Up to this point, the Internet simply connected computers.

This was good for scientists, who often wanted to access a big, far away supercomputer without having to get on an airplane, or package their data in an evelope or tape and mail it to the computer.

Instead, using the Internet, a scientist could start on a desktop computer, and log in to the supercomputer using programs called **ssh** or **putty**.

They could run a big scientific program on the supercomputer.

Then they could bring the data back to their desktop computer using a file transfer program called **ftp** or **sftp**.

In fact, scientists still use the Internet this way.

...But then some other stuff happened.

What’s the World Wide Web?

The Internet had been set up to make it easy for a person on one computer to send information to another computer, which was like sending a letter.

But sometimes someone wanted to make an announcement to many people they knew, (so a lot of individual letters), or perhaps even to any interested person (even people unknown to the author.) This was more like a newspaper or poster.

This idea eventually brought about the creation of the World Wide Web (WWW), by Tim Berners-Lee at the European Center for Nuclear Research.

WWW took advantage of the existence of the Internet, and its addresses and server system to create something that had not been imagined before.

What's the World Wide Web?

The web began with the idea that any computer user might want to share some information with people around the world.

To make this possible, the user simply had to set aside a special location or directory or folder on their computer, often called **public_html**.

Any documents placed inside this location would be presumed to be intended to be viewed by any interested person.

To access the documents, however, required knowledge of the address of the computer. The addresses had a specific format, starting with the prefix **http://www** and then listing information that specified the country, type of institution, department of the institution, person, directory and filename.

Such an address can be a lot to remember. A fairly short example is: **https://www.sc.fsu.edu/undergraduate/courses**

What's the World Wide Web?

These documents were usually fairly short, and so were called **web pages**.

Thus, to publicize a new class to any interested computer user, an instructor could create a document called newclass.txt and place it in the public_html of a computer:

```
Announcing a new class in Computational Thinking!

This fall, the Department of Scientific Computing
will be offering a class called "Computational Thinking".

For details, stop by room 445 Dirac Science Library.

Or check out the syllabus at
http://people.sc.fsu.edu/~jburkardt/classes/ct_2016/syllabus.txt
```

What's the World Wide Web?

As WWW became better known, people began to take advantage of it, posting interesting and useful information in their public areas.

But the crazy complicated address system was a huge drawback. No one could find the information without knowing the address, and the address was so long and complicated that it wasn't possible to try to guess where things were.

The first improvement came with the invention of links, which allowed a person writing a web page to make a bridge or simple connection to other web pages, saving readers from having to know more addresses.

What's the World Wide Web?

A web page using links would allow a user to jump to another web page simply by clicking on highlighted or underlined or colored text, rather than having to enter that address separately:

```
Announcing a new class in Computational Thinking!

This fall, the Department of Scientific Computing
will be offering a class called "Computational Thinking".

For details, stop by room 445 Dirac Science Library.

Or check out the SYLLABUS.
                 --------
```

What’s the World Wide Web?

Now that interesting web pages existed on the web, and it wasn’t so hard to find them, a new kind of program was developed for computers, called a browser.

One early browser was called Mosaic, which transformed into NetScape, and then into FireFox.

The browser’s job was to display a web page on the computer screen, allow the user to specify an address for a new web page, to effortlessly jump to a new web page if the user clicked on a link on the current web page, and to jump back if the user changed their mind.

Because it still wasn’t obvious how to find the most interesting information, browsers often included lists of recommended web pages for users to try.

Some browsers included a simple feature which would allow the user to ask a question, and get back a recommendation for a good web page to investigate.

What’s the World Wide Web?

Because people added, modified, and deleted web pages every day, the web was very dynamic, and the most interesting places to examine changed from day to day.

The simple programs inside a browser for guiding users relied on lists created by a staff who could not keep up with the rapid growth in the number of web pages and users.

Browser companies began desperately researching new methods of satisfying users, who wanted good information on what was new on the internet, and how to get there.

The problem was that the web changed too rapidly for human analysis, but how could a computer be of any use in finding and evaluating web pages?

Browsers and Search Engines

Originally, internet browsers tried to handle user searching themselves, or simply included pointers to a few places that listed resources.

Now users were demanding a much improved search facility.

It seemed sensible to split up the work, to separate the browser from the search engine. The browser would take care of moving to any particular web page, but the search engine would be a separate program that figured out which web page was the right place to go to next, in response to the user's question.

In some ways, the browser was like a car which could go anywhere, while the search engine was like a GPS system, which could direct the browser to a desired location.

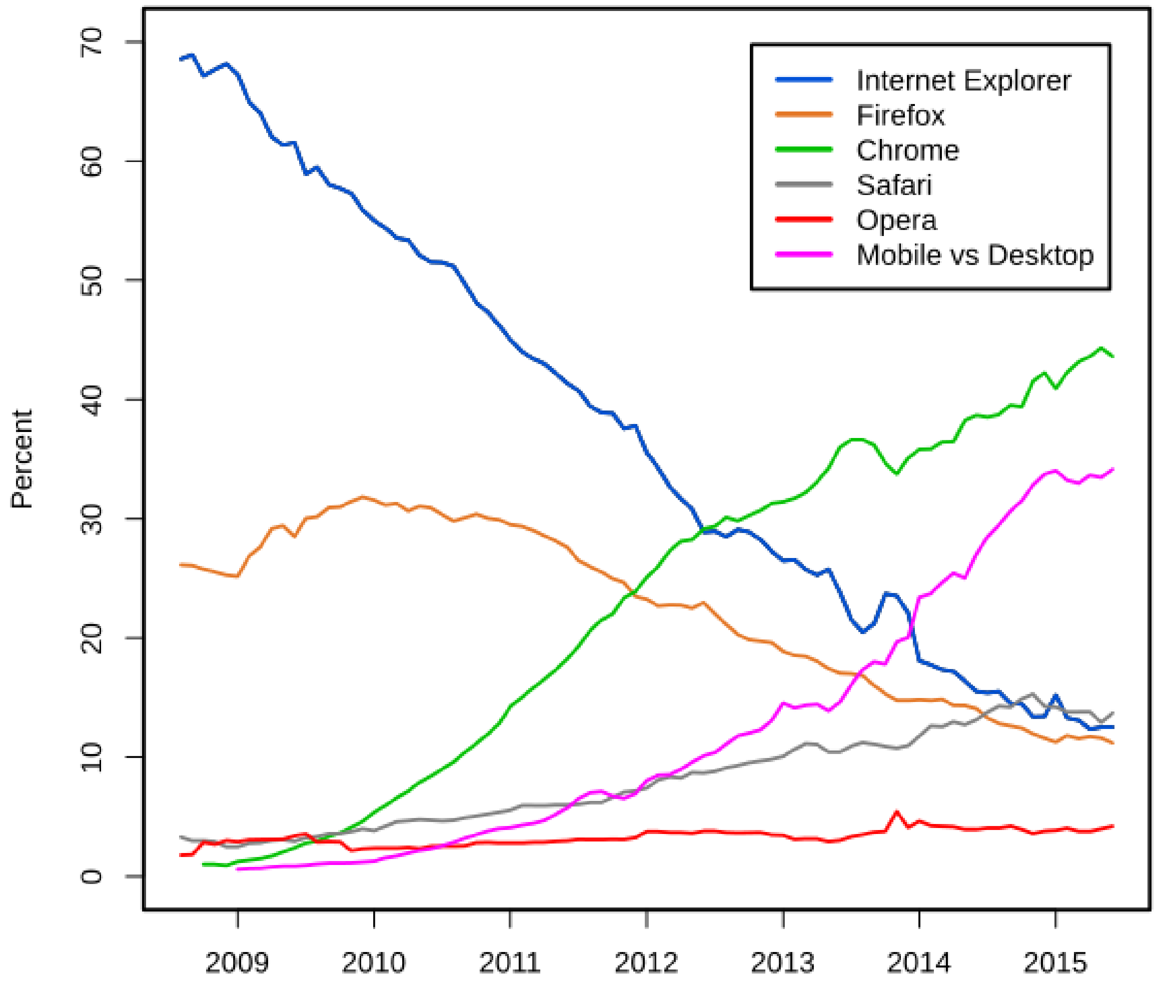
Browsers and Search Engines

Ever since those days, there have been two clearly distinct but cooperating programs for navigating on the web.

Browsers like Mosaic/NetScape/Firefox, Google Chrome, Microsoft Internet Explorer/Edge, Apple Safari, and Opera allow people to view and retrieve information on the internet.

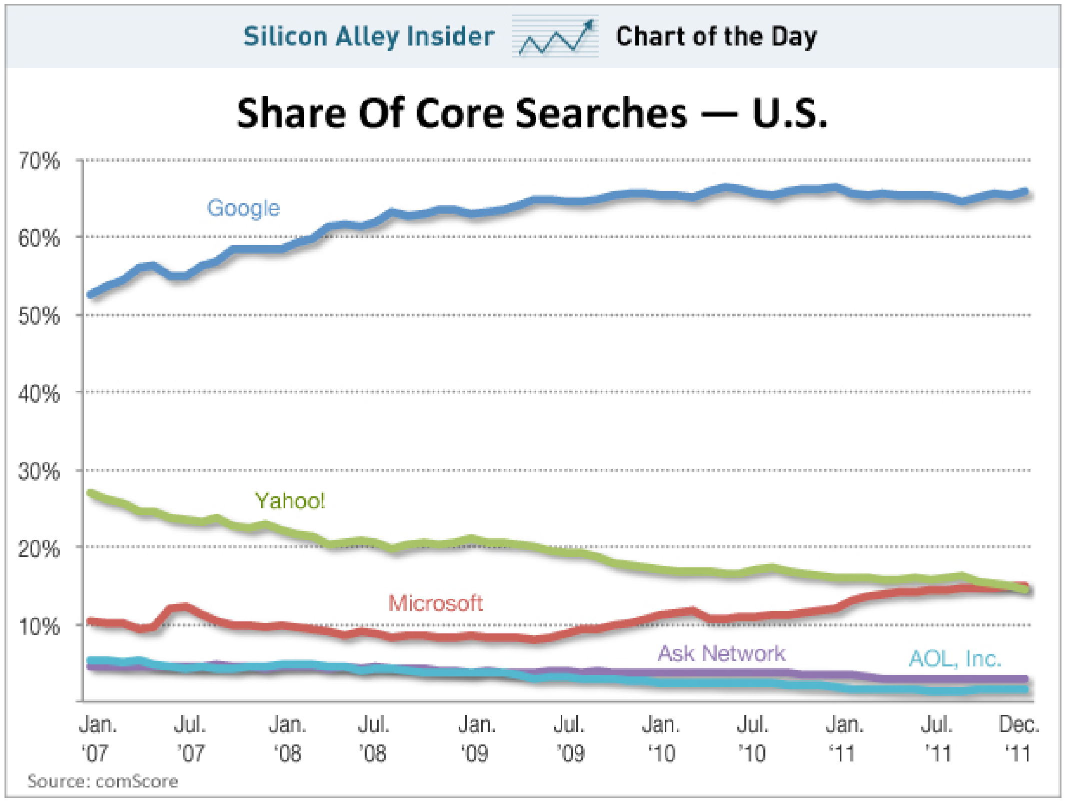
Search engines, like Google Search, Microsoft Bing, DuckDuckGo, Yahoo Search!, Ask.com, are used from within a browser, by typing a search string into a search box. The search engine then seeks to list web pages that pertain to that search string, after which the user can ask the browser to display information from a selected site.

Winners and losers in the browser wars

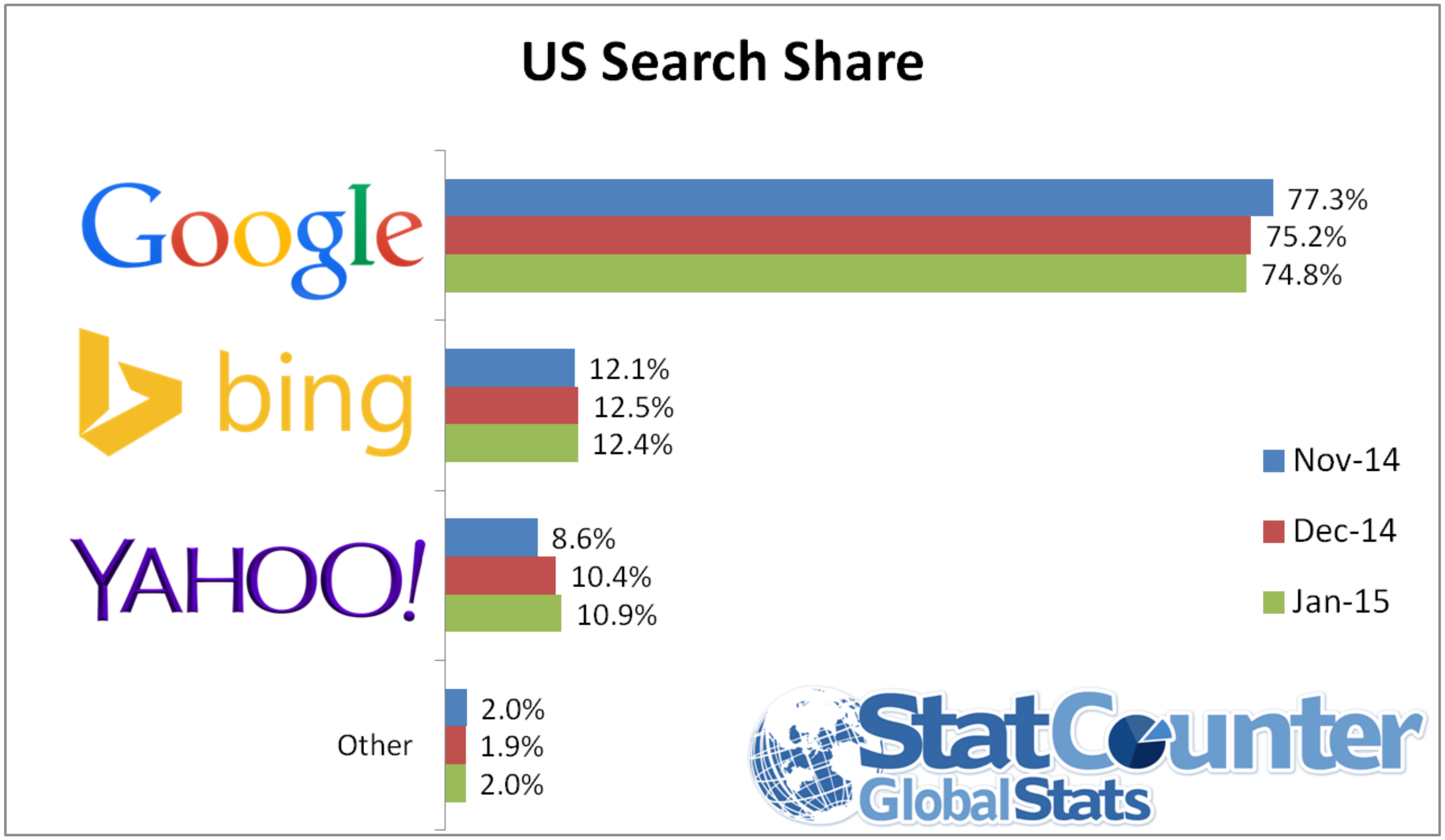


The search engine wars

In 2002, Google, Yahoo, and Microsoft had about 30% of the search engine market share, but Google made some dramatic improvements to its search engine, and drove Yahoo and Microsoft down to less than 20% shares each (and dropping).



Winners and losers in the search engine wars



It seems like everything is somewhere on the Web

To most of us, the web is simply an enormous pile of

- documents;
- photographs, images, pictures;
- email, messages, chats;
- songs;
- magazines;
- videos, movies;
- programs;
- advertisement, commercials and promotional videos;
- retail stores and ticket outlets;
- social media sites.

Can’t find it = can’t use it!

The Web may seem to be able to connect us to everything; but if we don’t know where something is, we might as well not have it.

We don’t even know what we’re looking for

So as the Internet has grown in size (number of things connected) and complexity (kinds of things connected), it has become increasingly important to develop and improve ways of:

- finding things you are looking for;
- recommending new things you might be interested in;
- learning enough about your interests to guess what other things you might want to see.

How hard is a search engine’s job?

Specialized search engines have a simple task

You are probably familiar with some specialized search programs that help you find information on the web, such as NetFlix and Yelp and Orbitz and Amazon.

These programs work well because they have a very limited vocabulary. You often select choices from a list, or enter simple information into boxes, like your credit card number and address.

For such a program, it's easy to imagine a simple procedure for getting the desired information from the user, and then checking a list of song titles, travel dates, or shoe sizes, until a desired match is found, and then finishing up the billing and shipping arrangements.

While such programs seem to carry out an elaborate sequence of steps, in general the choices are not so large, and it's easy to determine when you've found a match (the right size, the right price, the correct dates).

But a real search engine has a much more difficult task!

A search engine must be prepared for anything

Suppose you are looking for a list of the **rulers of Russia**, or advice on **how to whistle**, or instructions on how to get rid of a **wasp infestation**.

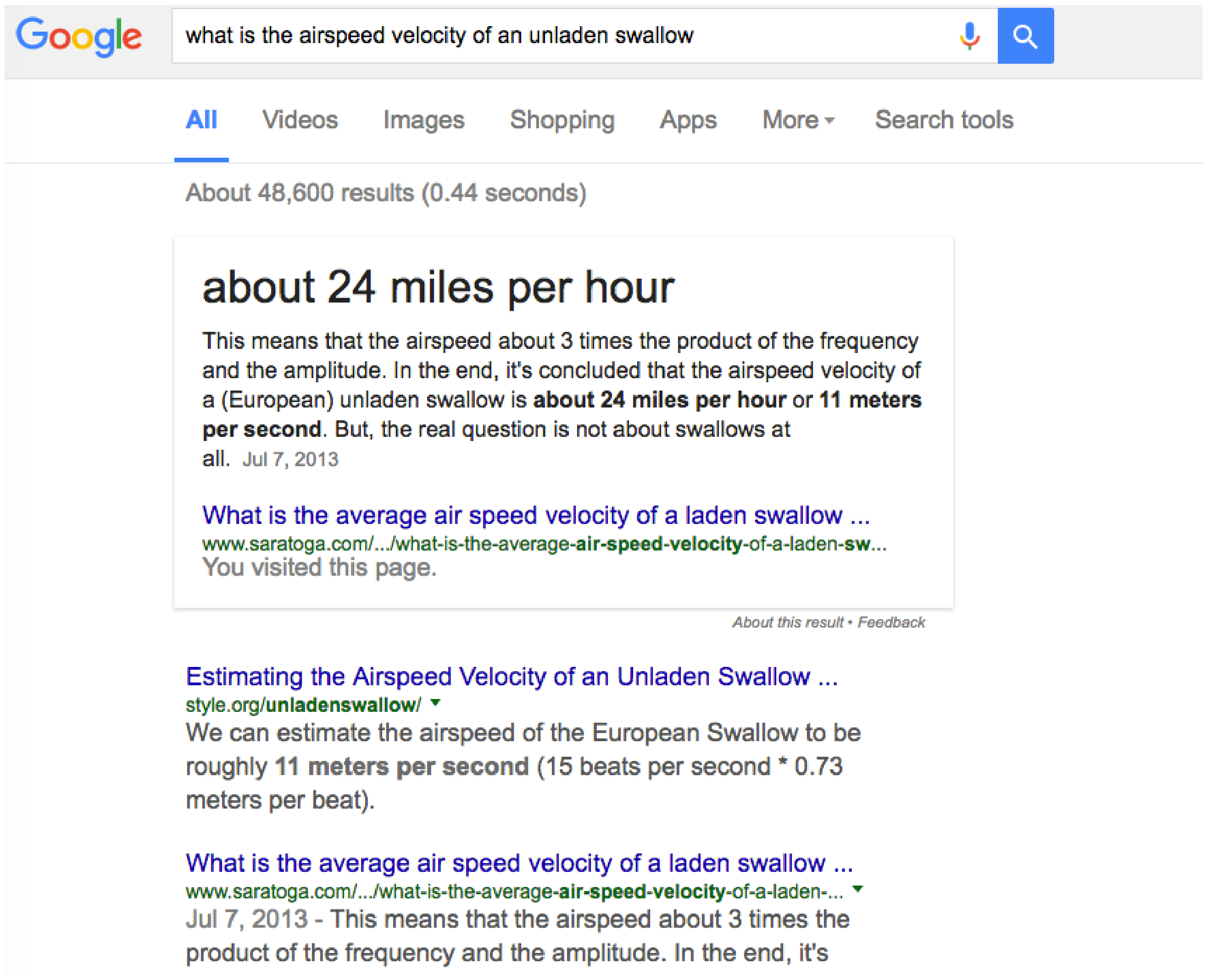
As soon as the browser realizes you're doing a search, it hands the information to the search engine, as a list of one or more words or search strings.

The search engine doesn't speak English or French, or Mandarin. It has no idea what your words actually mean.

If your search word is **apple**, should it concentrate on fruit, on the computer company, on the recording label for the Beatles, Fiona Apple, a movie named "The Apple"? What about Applebee's restaurant?

What makes matters worse is that if you search on "Apple", there seem to be more than a billion web pages on which that word appears!

How does our search phrase become an answer?



About 48,600 results (0.44 seconds)

about 24 miles per hour

This means that the airspeed about 3 times the product of the frequency and the amplitude. In the end, it's concluded that the airspeed velocity of a (European) unladen swallow is **about 24 miles per hour** or **11 meters per second**. But, the real question is not about swallows at all. Jul 7, 2013

What is the average air speed velocity of a laden swallow ...
www.saratoga.com/.../what-is-the-average-**air-speed-velocity**-of-a-laden-**sw**...
You visited this page.

About this result • Feedback

Estimating the Airspeed Velocity of an Unladen Swallow ...
style.org/**unladenswallow**/
We can estimate the airspeed of the European Swallow to be roughly **11 meters per second** (15 beats per second * 0.73 meters per beat).

What is the average air speed velocity of a laden swallow ...
www.saratoga.com/.../what-is-the-average-**air-speed-velocity**-of-a-laden-...
Jul 7, 2013 - This means that the airspeed about 3 times the product of the frequency and the amplitude. In the end, it's

What happens between your question and the answers?

Your browser is running on the computer right in front of you.

The information you need is somewhere else, perhaps far away.

We assume you've got a connection, wireless or wired, to your local network. Your request goes out from the local network to the Internet... and a few seconds later, your answer appears, a list of hundreds or thousands of “hits”, showing about 15 or 20 on the first page, including the location and an extract from the matching text.

It seems like magic; in fact, it's impossible for a search engine to receive a search string and check every word of every web page on the Internet and return the good matches to you unless you are willing to wait **days** for an answer.

But we got the answer in two seconds!

It seems impossible for this to work!

Why does this seem an impossible task?

For your browser to access just one web page:

- the browser converts the web address to a numeric IP (Internet Protocol) address;
- it has to set up a connection to that address;
- once the connection is made, it has to request a copy of the page from the remote computer;
- it has to wait for the remote computer to find the file locally;
- it has to wait for the information in the web page to be copied across the Internet into a local file.

Each single web page access can take on the order of 1 second.

There are more than 40 billion web pages on the Internet;

To copy every web page would take 40 billion seconds = 1240 years.

When 1 second is too long to wait: The Cache Trick

Browser users expect rapid response, and browser developers try every trick they can think of to keep their users happy.

Although a typical web page access might take a second, sometimes there is a longer delay because the network is busy, or the path to the web page is unusually long, or the server that controls the web page is very slow.

Since users often refer to a single web page several times in a single session, browsers added a cache, that is, they saved copies of the most recent web pages that have been visited. When you request a page, rather than going immediately to the web, the browser first checks to see if it already has a copy in its cache.

This **Cache Trick** usually works fine. But you can see it break down sometimes. If a web page has been updated since you first referred to it, your browser may continue to show you the out-of-date version. This is why most browsers include a **Refresh** button, which really means *throw away the cached page and get a fresh copy from the web!*

What really happens is a little more complicated

It's natural to assume that if you search for **polar bears**, then the search engine...really searches every web page on the Web, and comes back with a list of all the pages that contain that word.

That is impossible. So what does happen?

In order to find your matching pages, the search engine did a great deal of work long before your fingers touched the keyboard:

- **web crawling**: making a map of the Web;
- **examining links**: making an index of every word.

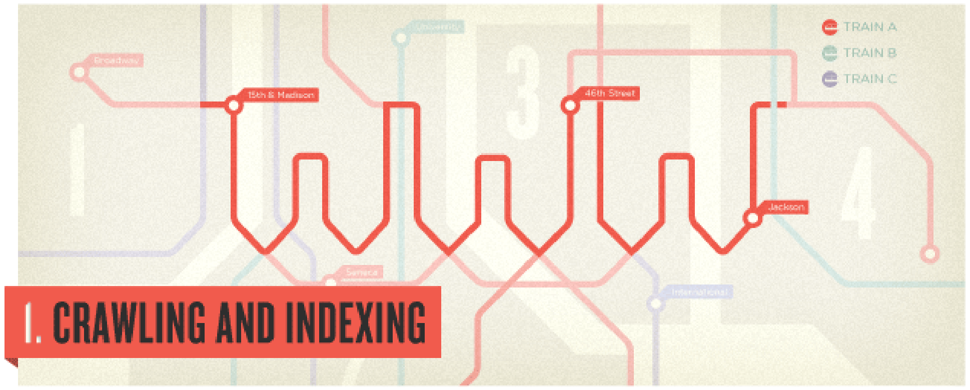
Let's see how careful advance work makes your search happen so fast!

Step 1: Making a map

A search engine prepares in advance for your question, whatever it is.

It starts this process by locating all the web pages it can find. This is called **web crawling**. It's hard to do right, because there's no map of the internet, and new sites appear and disappear every minute.

Imagine the Web as a network of stops in the New York subway system. The search engine needs to take random rides on this subway system, and notice every place it stops, and how it got there, and whether anything has changed since the last visit. All of this goes to updating a map of the Web.

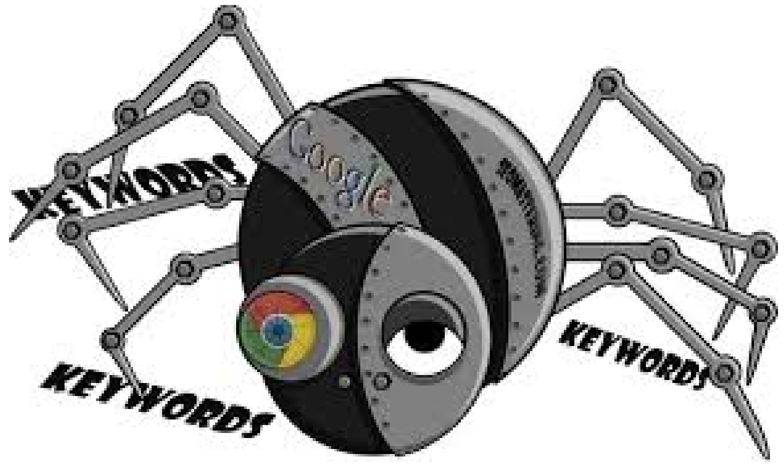


A crawler explores the web

A web crawler, often called a spider, is actually a program. A search engine lets many web crawlers explore the web. From time to time, a crawler visits the main FSU web page.

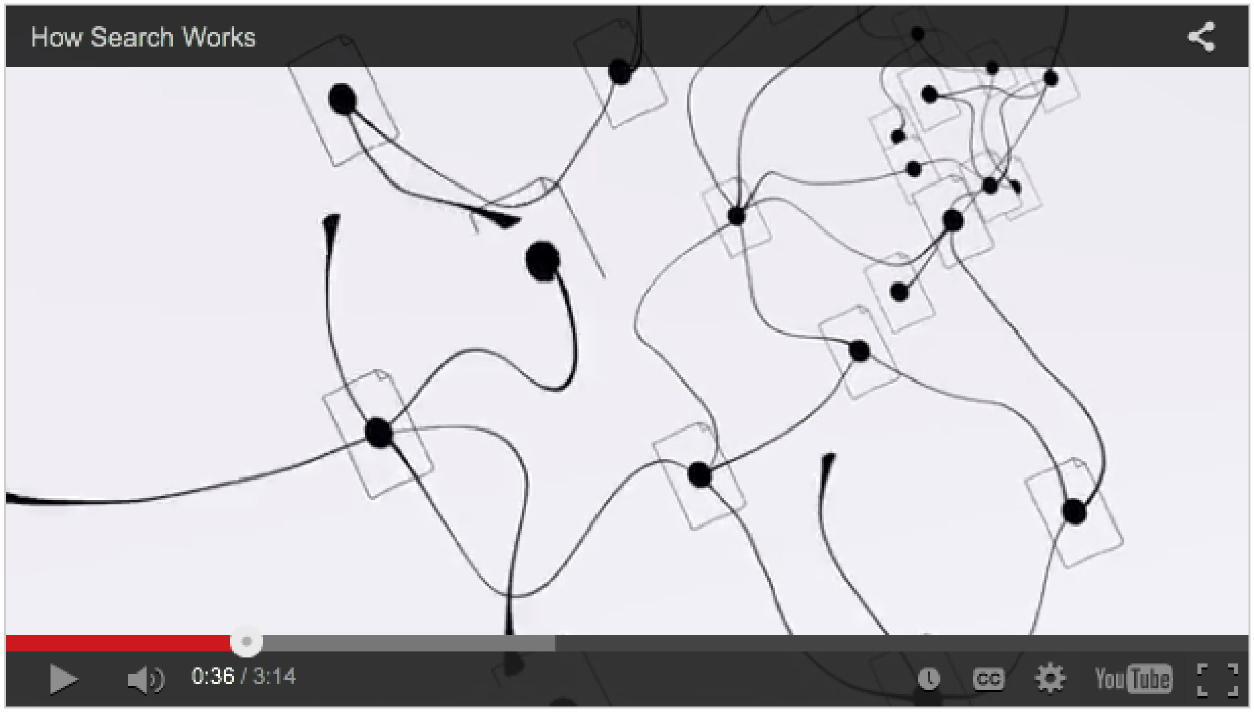
It notes all the links on the FSU web page, some of which changed since the last visit. It makes as many observations as possible in the local FSU web page directory, sending this information back to the search engine.

Once it has extracted all the information it can from the main FSU web page, it decides somewhat randomly where to explore next.



Spiders figure out the shape of the web

From their starting sites, the swarm of spiders follow links across the web, recording what they see and sending it back to the main search engine site, where all the information is combined into an updated snapshot of the web, its web sites, the web pages, and the links in those web pages.



How does a spider start its travels over the Web?

Some parts of the web are well known, and don't change their locations. These are servers, computers devoted to a special task. These are good places for an internet crawler to begin its exploration.

A **web server** does nothing but send and retrieve web pages to your browser while you're surfing the Internet.

An **e-mail server** is a computer that works as a virtual post office, receiving and storing mail messages, and interacting with you when you check your mailbox.

Facebook servers do nothing but handle all the activities of their users, and they are packed with information that a crawler wants to see.

Creepy crawlers

If you have a web site, you may be surprised or unhappy to realize that a robotic spider will look at your information from time to time and send a summary of it back to Microsoft or Google or Yahoo.

Usually, if you have a web site, anybody can look at your information, but still, it can be unsettling to realize that some viewers are actually vacuuming up your information for their own use.

Because of user complaints and privacy concerns, some major web sites and social media sites try to restrict access by these crawlers.

The Map Trick Can Break Down

Creating a map of the web is part of the search engine's plan to efficiently respond to your requests.

When you send a question to the search engine, it relies first on this map, rather than on the actual web. That means that, sometimes, the map and the web differ.

The map may include a web page that has since actually disappeared. For this reason, your search request will sometimes point to what seems like an interesting web page, but when you try to view it, it says "Can't find this page!" and you're left thinking, "What? You just told me to look at it!"

Also, if you put a web page up, it will not be visible to any search engine for some time, until a web crawler runs into it and adds it to the map.

These failures are two consequences of the otherwise very reliable **Map Trick**.

Step 2: Making an index

So now we have some idea of how it's possible to map the web using crawlers.

But just because we know where every web page is, we still have the impossible task of checking them all against the user's search string.

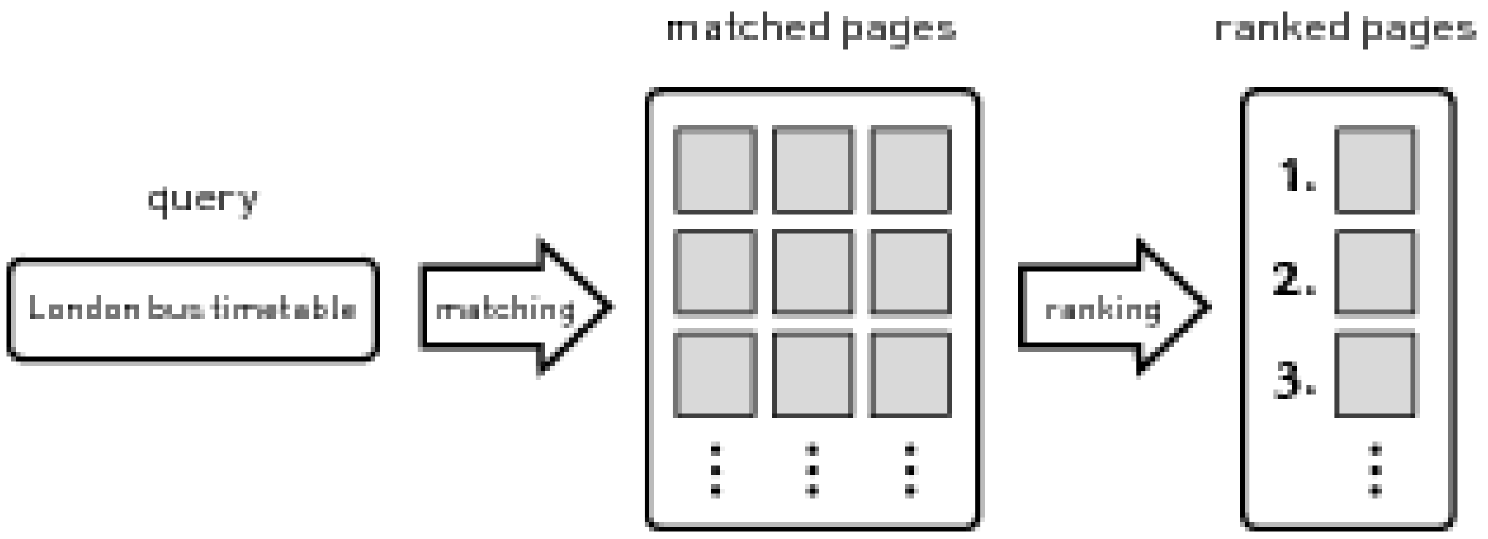
Luckily, Google solved this problem, using other information gathered by the web crawlers.

When you do a Google search, you are NOT actually searching the web, but rather Google's index of the web.

Google's index is well over 100,000,000 gigabytes of information, requiring one million computing hours to build, and is constantly updated.

Now we want to understand what it means to index the web.

Web search = match + rank



When you issue a web search query, it is processed in two stages:

- matching searches for all matching pages using the web map;
- ranking orders those matching pages so the best appear first.

The query "London bus timetable" seeks matches, and then ranks them.

Library search: Search engines have always existed

One way to realize how search engine indexing makes the page match task possible is to think about what used to happen, in the good old days, when you went to the library to work on a term paper.

You might start by wandering through the library, looking for the books you need, but after a while, you realize this is hopeless.

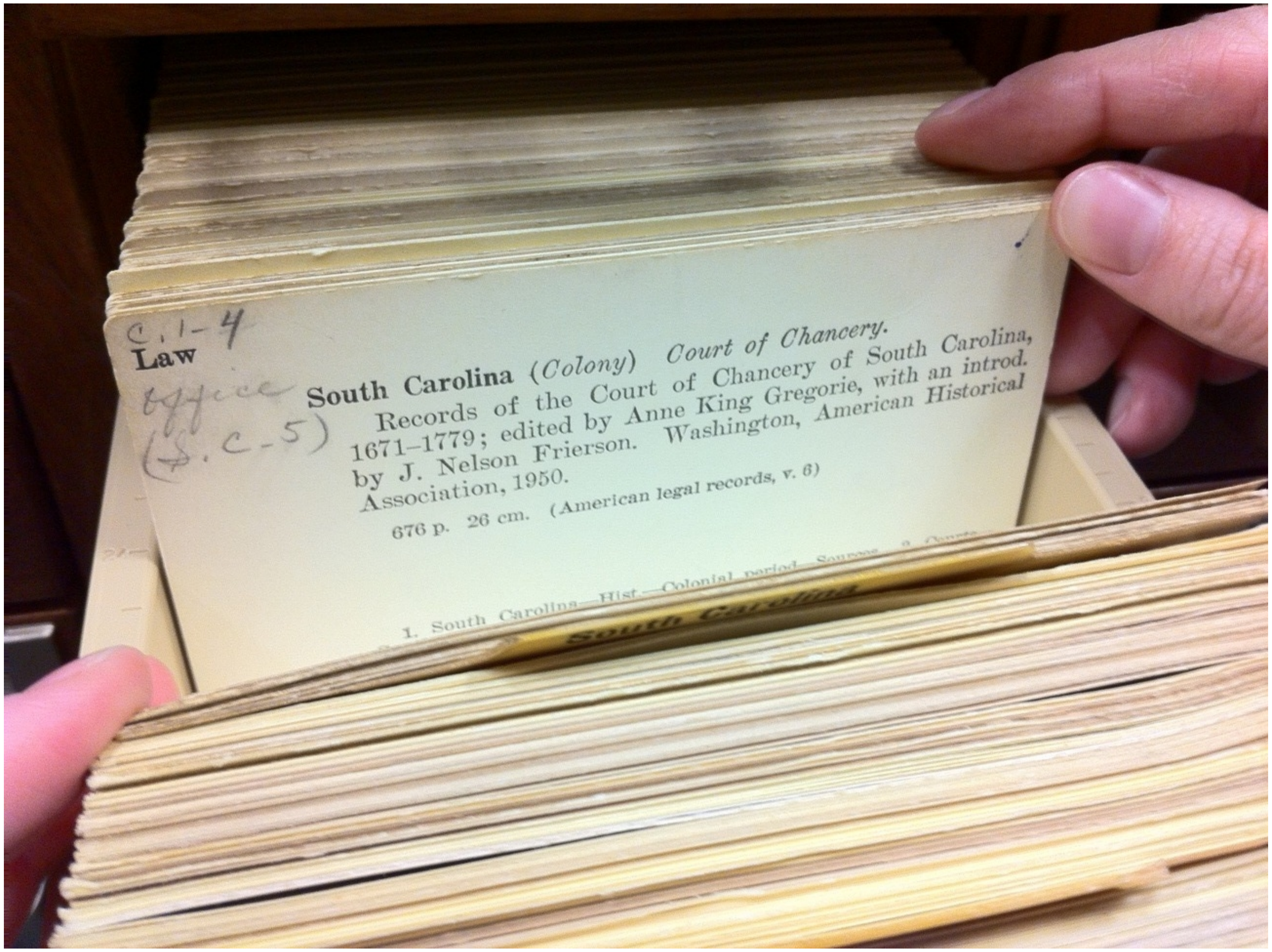
Then you find a librarian and say:

"I need a list of the rulers of Russia."

The librarian doesn't know where your information is either. But the librarian knows how you can find it.

The librarian says: *"Search the catalog card index!"*

Library search: A search engine from the good old days



C.1-4
Law
office
(S.C-5)

South Carolina (*Colony*) *Court of Chancery.*
Records of the Court of Chancery of South Carolina, 1671–1779; edited by Anne King Gregorie, with an introd. by J. Nelson Frierson. Washington, American Historical Association, 1950.

676 p. 26 cm. (American legal records, v. 6)

1. South Carolina—Hist.—Colonial period—Sources. 2. Courts

Library search: Creating an index takes time

The old card catalog index contained hundreds of thousands of search phrases: topics, author names, events, all in alphabetical order.

Every time a new book came into the library, the librarians prepared cards for every topic covered by the book, and added these to the index.

With great work, and over a long time, the card catalog was built up to be a labor saving device that allowed you to "instantly" (well, within a minute or two) discover the "addresses" (library call numbers) of every book in the library that might pertain to your topic.

Library search: Search phrase + index gets us the address

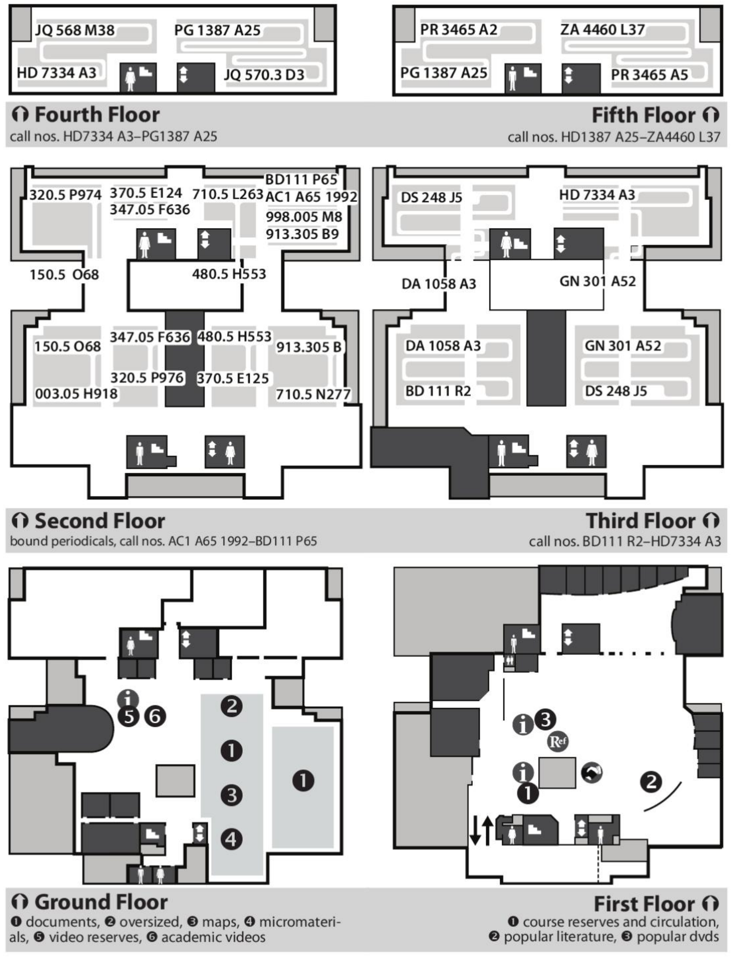
Searching the card catalog under **Russia, Rulers** we might see a sequence of cards like:

Lieven, Dominic	“Russia’s rulers under the old regime”	DK253.L54
Lev, Timofeev	“Russia’s secret rulers”	DK510.73.T56
Warnes, David	“Chronicle of the Russian Tsars”	DK37.6W37
Gooding, John	“Rulers and subjects”	DK189.G86

Perhaps the title by Warnes seems the best match for our interest. In that case, we need to make a note of the information **DK37.6W37** because this is the Library of Congress catalog number for the book.

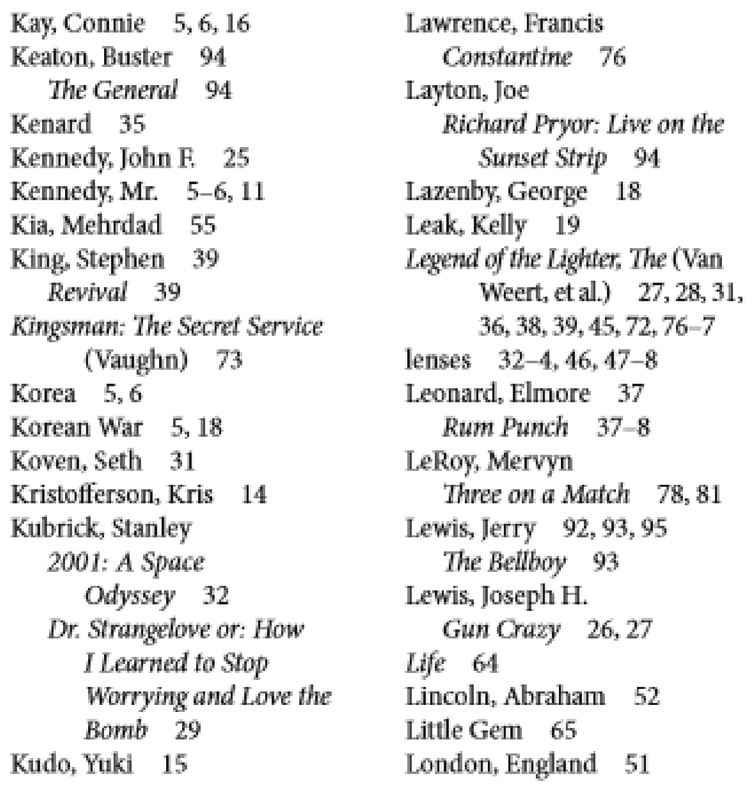
This number amounts to an address for the book, so once we have a map of the library, we now know where to search next.

Library search: Address + Map gets us book



Library search: Books have an index too!

Most nonfiction books include an index, which lists names and topics in alphabetical order, and the pages on which these are discussed.



Kay, Connie 5, 6, 16
Keaton, Buster 94
  The General 94
Kenard 35
Kennedy, John F. 25
Kennedy, Mr. 5–6, 11
Kia, Mehrdad 55
King, Stephen 39
  Revival 39
Kingsman: The Secret Service (Vaughn) 73
Korea 5, 6
Korean War 5, 18
Koven, Seth 31
Kristofferson, Kris 14
Kubrick, Stanley
  2001: A Space Odyssey 32
  Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb 29
Kudo, Yuki 15
Lawrence, Francis
  Constantine 76
Layton, Joe
  Richard Pryor: Live on the Sunset Strip 94
Lazenby, George 18
Leak, Kelly 19
Legend of the Lighter, The (Van Weert, et al.) 27, 28, 31, 36, 38, 39, 45, 72, 76–7
lenses 32–4, 46, 47–8
Leonard, Elmore 37
  Rum Punch 37–8
LeRoy, Mervyn
  Three on a Match 78, 81
Lewis, Jerry 92, 93, 95
  The Bellboy 93
Lewis, Joseph H.
  Gun Crazy 26, 27
Life 64
Lincoln, Abraham 52
Little Gem 65
London, England 51

Library search: Book + book index gives us our information

Once we get the book in our hands, we're holding 300 or 400 pages of dense text; we probably don't want to read or even scan the whole book to find a list of Russia's rulers!

Instead, we rely on the fact that books include an index at the back, making it possible to rapidly locate the pages on which key topics are presented.

So we turn to the back of the book, and luckily, find "Rulers, List: 217" and turn to page 217 and finally have what we want:

Rurik I	862-879
Oleg of Novgorod	879-912
Igor I	913-945
Olga of Kiev	945-962
...	...

Library search: Pretty efficient?

Now Strozier library claims to hold about 3.3 million books.

A typical book might have 300 pages.

So we can estimate there are a total of 3,300,000 x 300 $\approx$ 1 billion pages.

Our search lead us to one page on one book, so essentially we picked one page out of a billion in about 15 minutes.

This means that in every single second, we essentially skipped over a million pages of useless text in our search for the right one.

Let's remember this!

Library search rate $\approx$ one million pages per second.

Library search: The Index Trick

Twice, in our library search, we referred to an index as an efficient way to rapidly narrow our search.

The library index listed books in the library that might pertain to our topic.

The book index listed pages in the book that might pertain to our topic.

The indices saved us from having to search every book, or search every page.

Of course, the reason we had a fast search was that other people (librarians and publishers) had already done the hard work of preparing accurate indexes for us.

Web search is similar to library search

Searching the web is similar to searching the library:

- There are a huge number of items;
- Each item contains many pieces of information;
- Each item has an address.

We need:

- a way to specify a search topic;
- a list of items that might include this topic;
- an address for each item;
- a map that tells us how to reach any address;
- a way to reach the address;
- a way to view the item;
- a way to locate our search topic within the item.

Rapid search requires indexing

For instance, every time a politician in Washington D.C. publishes a book, everyone rushes to the bookstore, goes to the back of the book, and looks to see if their own name appears in the index!

An index takes a long time to make; most of the information will never actually be used, and it's important to select only useful occurrences of important items.

But once the index is prepared, then it's possible to rapidly locate any indexed item, no matter how big the book, or the library.

Thus, the next ingredient in our rapid page matching procedure is the The Index Trick.

Next time we will see how to create and use an index of all the search words on all the web pages on all the web sites on the World Wide Web.

Matt Cutts's Video on "How Search Works"

Matt Cutts is an engineer at Google and explains how Google's search engine works.

https://www.youtube.com/watch?v=Md7K90FfJhg

Socrative Quiz PartII_Quiz1

IUZGAZ34E

Answer T for true and F for false.

1. Safari is a search engine.
2. Google Chrome is a browser.
3. You have to use a browser to get to a search engine.
4. A search engine checks every word of every web page.
5. A spider or a crawler is an electronic robot which is inside your computer.
6. A spider starts its Web search, at a popular Web site or server.
7. Currently, Yahoo is the most popular search engine in the U.S.
8. You are not actually searching the Web but an index of the Web.
9. The Internet and the World Wide Web were developed at the same time.
10. Links on a web page provide a bridge to other web pages.

Goals for this lecture

1. By looking at simple examples we want to see how a Web index can be made;
2. To understand the importance of the location of keywords on a Web page;
3. To realize that Web pages are typically written in a language called HTML;
4. How HTML tags can indicate what a web page is about.

Map + Match + Rank

We have seen that when you enter a search phrase in your browser, the browser passes this request to a search engine, and the search engine doesn't actually go out and check the web, but rather looks at an index of the web created from information gathered by web crawlers as they constructed a map of the web.

The stages involved in this process include:

1. map the web, its web pages and topics;
2. match the search words to web pages;
3. rank the matching web pages by importance.

Let's assume that the web map has been built, and that it's time to think about how to rapidly match web pages to an arbitrary user search word.

The Index Trick

Let's recall from our discussion last time how an index can be used to speed up the search for occurrences of some word, phrase, or topic.

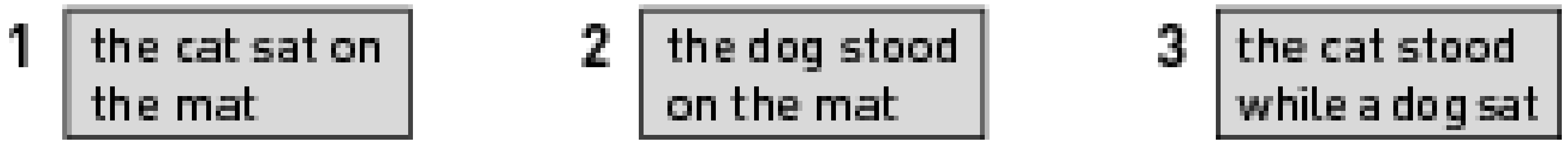
An index essentially says *"If you're looking for this topic, check out these places."*

It should be clear that this idea, which worked so well for libraries and books, can be extended to the web and web pages.

An index takes a lot of work to prepare; once it's created, it makes searching incredibly faster.

Where the Google Search index is calculated and stored

The tiny web



1 the cat sat on the mat

2 the dog stood on the mat

3 the cat stood while a dog sat

The web is too huge to think of indexing right away. Let's start with a simple model, in which there are just three objects, pages 1, 2 and 3.

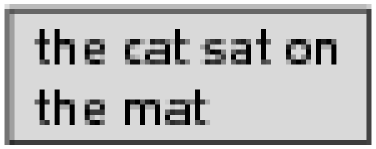
Indexing these web pages is very similar to indexing a book. We will consider every word important. So we start by listing alphabetically the unique words we find. Each word will be followed by a "1" if it occurs in page 1, a "2" if in page 2, and so on.

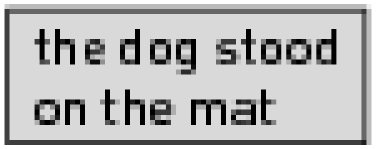
Let's go through this painful indexing process now.

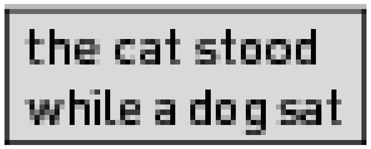
The tiny web: index #1

Our resulting index should look like this. It is a single file that contains all the words that appear, and which web pages use them:

Word	Pages containing word		
a	3		
cat	1	3	
dog	2	3	
mat	1	2	
on	1	2	
sat	1	3	
stood	2	3	
the	1	2	3
while	3		

1 
the cat sat on
the mat

2 
the dog stood
on the mat

3 
the cat stood
while a dog sat

The tiny web: One keyword on any page

A search engine could answer questions with just this single index file.

If we enter the search string **dog**, the search engine can quickly find the corresponding line in the index (this is quick, because a computer takes advantage of alphabetical ordering even more efficiently than we do). Then the search engine can report that “dog” appears in pages 2 and 3.

Of course, a real search engine would include a bit of the text surrounding the occurrence of “dog” but that’s an added feature that we don’t need to worry about right now.

If we enter the search string **cat**, our search engine will tell us that this word occurs in pages 1 and 3.

The tiny web: Several keywords on the same page

Often, our search string involves more than one word.

For example, if we are interested in finding a web page that discusses the problems of having both a cat and a dog, then we don't want pages that have cats and pages that have dogs, we want pages that contain *both keywords at the same time*.

So let's enter a search phrase **dog cat**.

The search engine can determine that "dog" occurs on pages 2 and 3; Then the search engine can find "cat" on pages 1 and 3.

Now it must put these two facts together, realizing that "dog" and "cat" both appear only on page 3, and this is the single result returned.

So finding two search words together on the same page takes two searches, three words would require three searches, and so on.

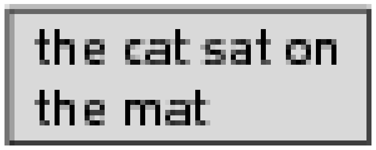
The tiny web: We didn't access the web to answer the question

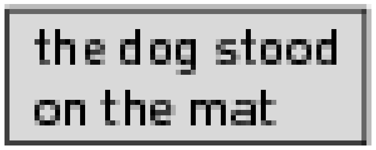
Notice an important fact: to answer these questions, we did not have to have access to the original web pages. We needed that when we made the index, but now a single index file allows us to answer questions about all three pages.

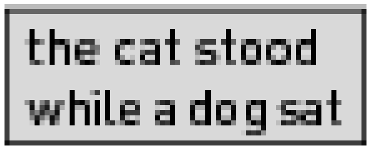
Now the World Wide Web has 40 billion web pages. Suppose we can create a similar (but huge!) index file for them. Then, to answer simple matching questions about all the web pages, we only have to search one place, the index file, not the web itself.

This is one clue to how a search task that should take a thousand years can be cut down to 2 seconds.

The tiny web: Can we handle connected keywords?

1 
the cat sat on the mat

2 
the dog stood on the mat

3 
the cat stood while a dog sat

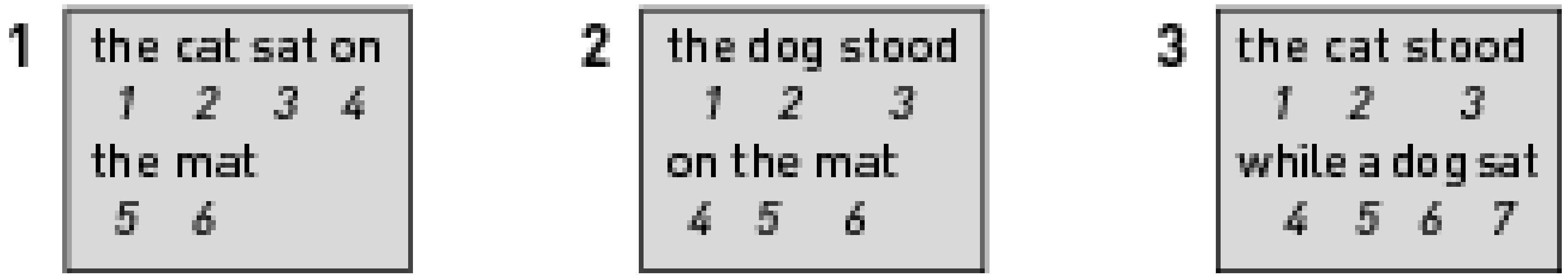
In most search engines, it is possible to enter a phrase, using quotation marks, such as “cat sat”. In that case, you are not just asking that both words appear somewhere on the same page, but that they appear immediately together, in that order. That is, we want the bigram **cat sat**.

The index file we created for the tiny web can tell us that both **cat** and **sat** occur on pages 1 and 3, but not whether they occur together.

It might seem that the solution is to look up **cat** and then go to those web pages and find whether **sat** occurs in the right position.

This is not acceptable! It would require downloading every web page that had **cat**, and reading the entire web page to see whether **sat** was the next word. There is no way to guarantee a fast answer.

The tiny web: Include keyword locations in the index



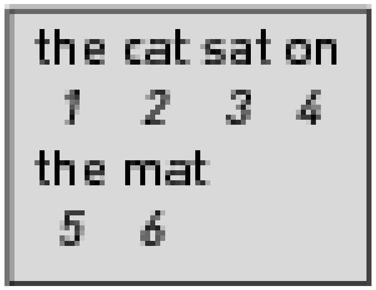
Maybe we can fix this problem, if we include more information in our index.

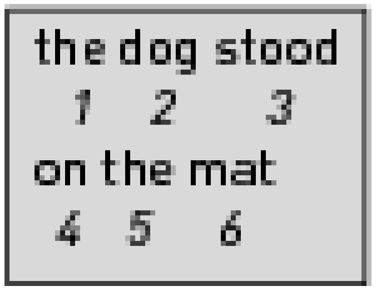
Suppose in version #2 of our index file:

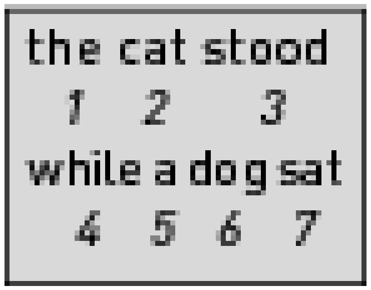
- we label every word in each page with its position;
- we record every occurrence of a word in the page along with its position.

So if **the** occurs twice in a web page, each occurrence is listed, along with its position.

The tiny web: Index #2

1

the cat sat on
1 2 3 4
the mat
5 6

2

the dog stood
1 2 3
on the mat
4 5 6

3

the cat stood
1 2 3
while a dog sat
4 5 6 7

Word	Page-Position
a	3-5
cat	1-2 3-2
dog	2-2 3-6
mat	1-6 2-6
on	1-4 2-4
sat	1-3 3-7
stood	2-3 3-3
the	1-1 1-5 2-1 2-5 3-1
while	3-4

The tiny web: Searching index #2

Now suppose we are given the search phrase **“cat sat”**.

We look up the word **cat**, and see that it occurs on page 1 as word 2, and on page 3 as word 2.

sat is on page 1 as word 3, right after **cat**, so we have a hit on page 1.

sat is on page 3 as word 7, not immediately following **cat** there, and so that counts as a miss.

By making our index more intelligent, we can now answer *any* phrase inquiry about our web pages, and we still don’t need to access the web to do this.

We call this The Word Location Trick.

Socrative Quiz PartII_Practice_Quiz1

IUZGAZ34E

Use the following table of page numbers and word locations:

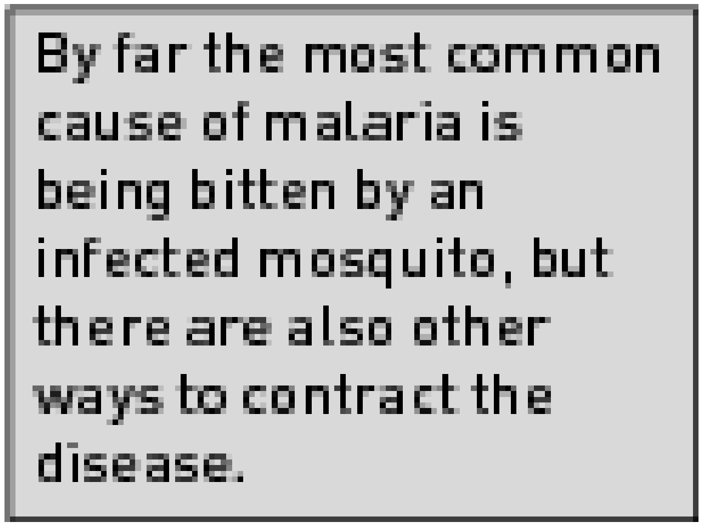
Word	Page, Position				
a	1-7				
animal	1-2	2-6	3-5		
be	2-4				
cheetah	1-8	2-1	3-1	2-5	3-1
earth	1-4				
fastest	1-1	2-5	3-3		
is	1-5	3-2			
land	3-4				
may	2-2				
not	1-6	2-3			
on	1-3				

1. On what page(s) does the word earth occur?
2. On what page(s), if any, does the bigram on earth occur? If it doesn't occur, enter “0”.

3. On what page(s), if any, does the trigram fastest land animal occur? If it doesn't occur, enter "0".

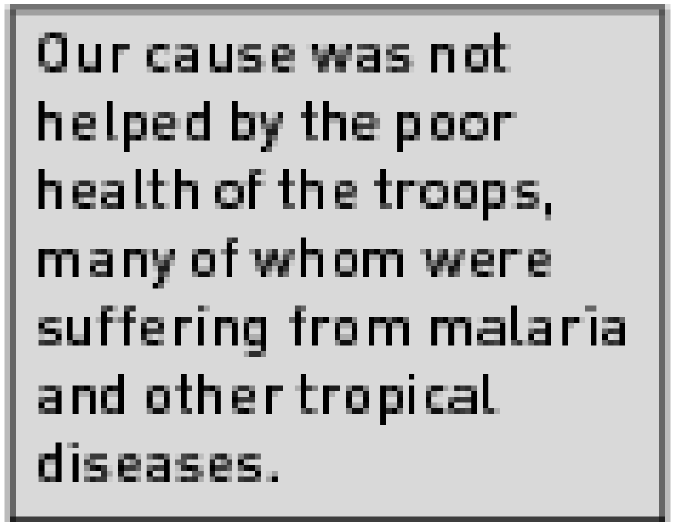
Nearness: When is one matching pair better than another?

1



By far the most common cause of malaria is being bitten by an infected mosquito, but there are also other ways to contract the disease.

2



Our cause was not helped by the poor health of the troops, many of whom were suffering from malaria and other tropical diseases.

Suppose we were interested in learning the cause of malaria. We might naturally search on **malaria cause** although we probably don't insist that those two words occur exactly together.

Suppose the search engine discovered two web pages with both match words. We can see the first web page is a better match. What clues could a search engine use?

Nearness: We know where things are

Here we see part of our index file, with keywords highlighted.

Word	Page, Position			
by	1-1			
cause	1-6	2-2		
common	1-5			
...				
malaria	1-8	2-19		
many	2-13			
of	2-10	2-14		
...				
the	1-3	1-24	2-7	2-11

Nearness: Close keywords are better

Using our word location index, the search engine can see that on page 1, **malaria** and **cause** are just 1 word apart, versus 16 words apart on page 2.

Although both pages match both keywords, page 1 may be the better match because the keywords occur much closer to each other.

Notice that the computer does not understand what it is reading! It could do the same kind of analysis for keywords and text written in Italian, or in ancient Mayan.

It may look like an intelligent action, but it's based on a very simple idea: physically close keywords suggest a better match.

The Nearness Trick is thus also useful for our upcoming page ranking task.

Nearness in Google Search

We already know two ways to specify multiple key words to a search engine:

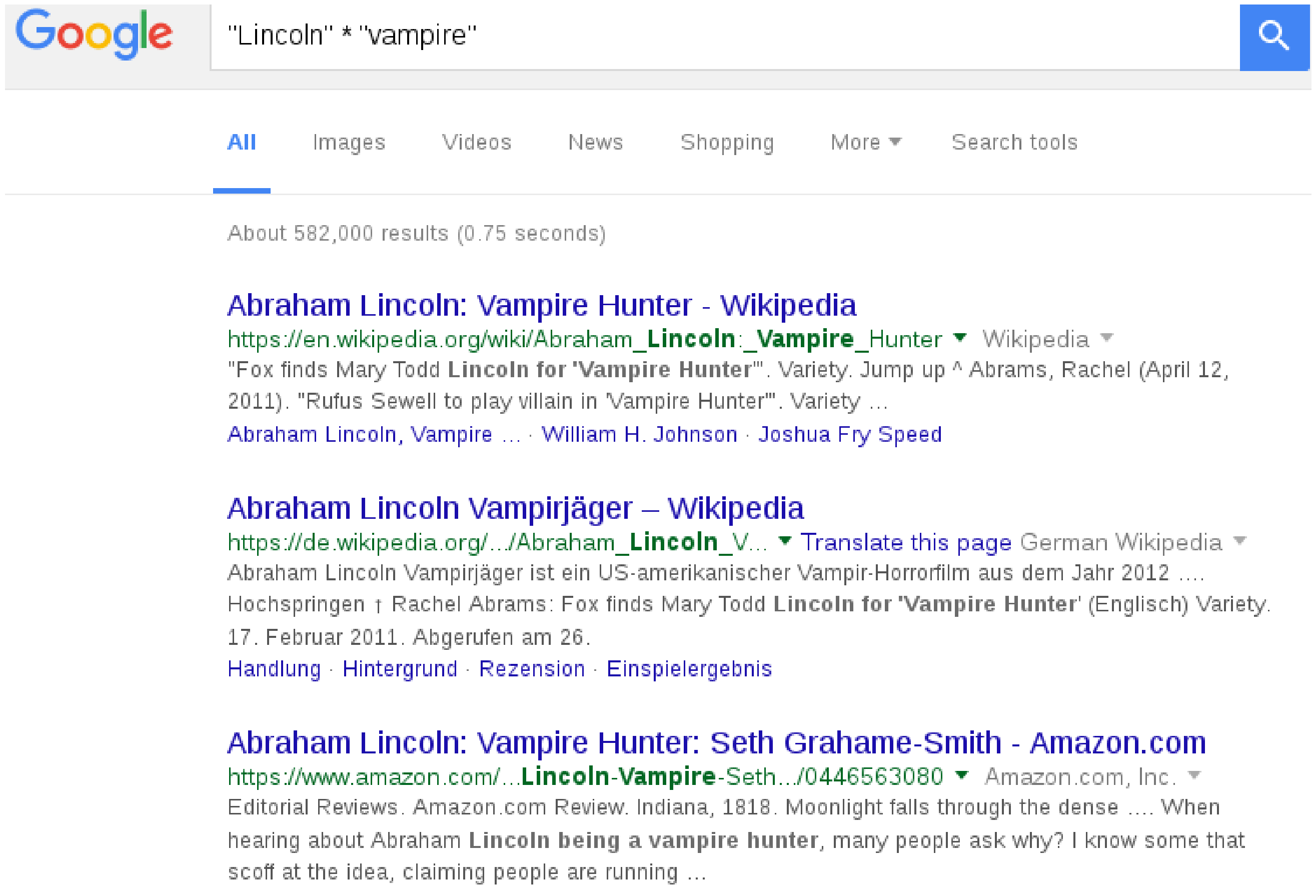
- quoted, we prefer that the words appear together, in that order;
- unquoted, the words can be in any order and far apart.

It turns out that most search engines automatically prefer situations in which the keywords are close. However, one interesting feature in Google Search allows us to specify how close we want the words to be.

If we use one asterisk between two quoted keywords, then we are asking for pages where the keywords appear in that order, separated by exactly one word

```
"string1" * "string2"
```

Nearness: Search results for “Lincoln” * “Vampire”



Google "Lincoln" * "vampire"

All Images Videos News Shopping More Search tools

About 582,000 results (0.75 seconds)

Abraham Lincoln: Vampire Hunter - Wikipedia
https://en.wikipedia.org/wiki/Abraham_**Lincoln**:_**Vampire**_Hunter ▾ Wikipedia ▾
"Fox finds Mary Todd **Lincoln for 'Vampire Hunter**'". Variety. Jump up ^ Abrams, Rachel (April 12, 2011). "Rufus Sewell to play villain in 'Vampire Hunter'". Variety ...
Abraham Lincoln, Vampire ... · William H. Johnson · Joshua Fry Speed

Abraham Lincoln Vampirjäger – Wikipedia
https://de.wikipedia.org/.../Abraham_**Lincoln**_V... ▾ Translate this page German Wikipedia ▾
Abraham Lincoln Vampirjäger ist ein US-amerikanischer Vampir-Horrorfilm aus dem Jahr 2012 .... Hochspringen ↑ Rachel Abrams: Fox finds Mary Todd **Lincoln for 'Vampire Hunter**' (Englisch) Variety. 17. Februar 2011. Abgerufen am 26.
Handlung · Hintergrund · Rezension · Einspielergebnis

Abraham Lincoln: Vampire Hunter: Seth Grahame-Smith - Amazon.com
https://www.amazon.com/...**Lincoln**-**Vampire**-Seth.../0446563080 ▾ Amazon.com, Inc. ▾
Editorial Reviews. Amazon.com Review. Indiana, 1818. Moonlight falls through the dense .... When hearing about Abraham **Lincoln being a vampire hunter**, many people ask why? I know some that scoff at the idea, claiming people are running ...

Nearness: Avoiding spam pages

One reason that search engines also prefer matches in which multiple keywords are close is to avoid being trapped by spamdexes. A spamdex is an artificial web page that simply contains a grab bag of keywords, without any information. You could make such a web page by posting a dictionary, minus the definitions, for instance. We will look at this more closely later in the course.

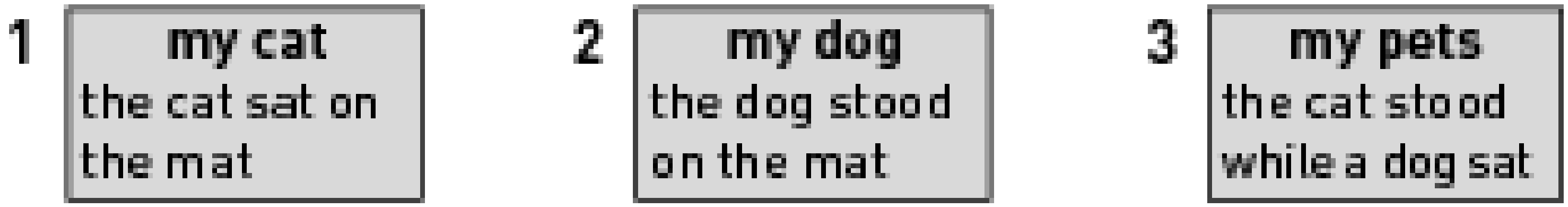
A search engine looking for **hair loss remedy** or **tap dance lessons** or **perpetual motion machines** will find matches (but no information!) on a spamdex page, and the spamdex operator will pick up some money by displaying ads to the annoyed user.

Thus, even if the user doesn't request that the keywords be close, search engines avoid matching pages that fail the proximity test.

Nearness: A spam page, your keywords are here somewhere!

a aa aaa aaas aah aahed aahs aal aalii aam aani aardvark aardvarks aardwolf aardwolves aargh aarhus aaron aaronic aaronical aaronite aaronitic aaru aau aaziz ab aba ababa ababdeh ababua abac abaca abacate abacay abaci abacinate abacination abaciscus abacist aback abacterial abactinal abactinally abaction abactor abaculus abacus abacuses abada abade abadite abaff abaft abail abaisance abaiser abaissed abait abaka abalienate abalienation abalone abalones abama abamp abampere aband abandon abandonable abandoned abandonedly abandonee abandoner abandoning abandonment abandonments abandons abanic abantes abaptiston abarambo abaris abarthrosis abarticular abarticulation abas abase abased abasedly abasedness abasement abasements abaser abases abasgi abash abashed abashedly abashedness abashes abashing abashless abashlessly abashment abashments abasia abasic abasing abask abassin abastardize abatable abate abated abatement abatements abater abaters abates abating abatis abatised abatises abaton abator abattoir abattoirs abatua abature abave abaxial abaxile abaya abaze abb abba

HTML: Indexing should notice a web page title



1 my cat
the cat sat on the mat

2 my dog
the dog stood on the mat

3 my pets
the cat stood while a dog sat

Web pages are actually a little more complicated than the simple text files we have used so far as examples.

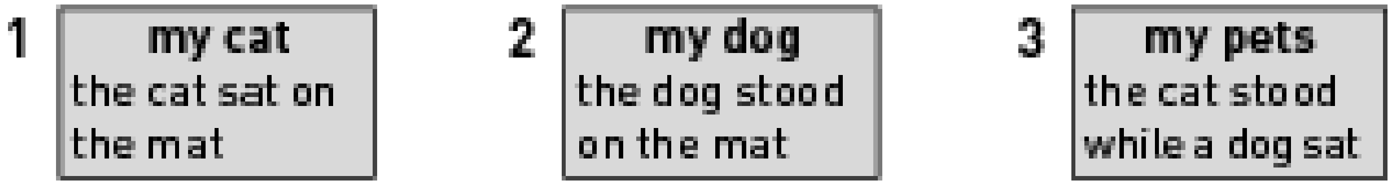
Web pages are written in HyperText Markup Language **HTML**. HTML allows the author to vary the font type and size, to include tables, lists and figures, and to indicate the structure of the document.

A web page author can specify a title for the web page using specific **HTML tags** to indicate where the title starts and stops.

If a search engine is looking for the key word **malaria**, doesn't it make a huge difference if the actual title of a page is "Malaria"?

HTML: An example of title searching

Here are three web pages which include title information.

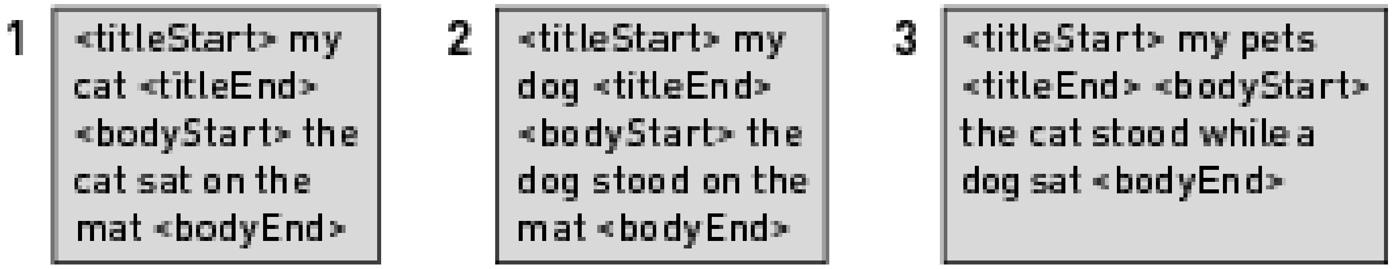


1 **my cat** the cat sat on the mat

2 **my dog** the dog stood on the mat

3 **my pets** the cat stood while a dog sat

The pages as we see them.



1 <titleStart> my cat <titleEnd> <bodyStart> the cat sat on the mat <bodyEnd>

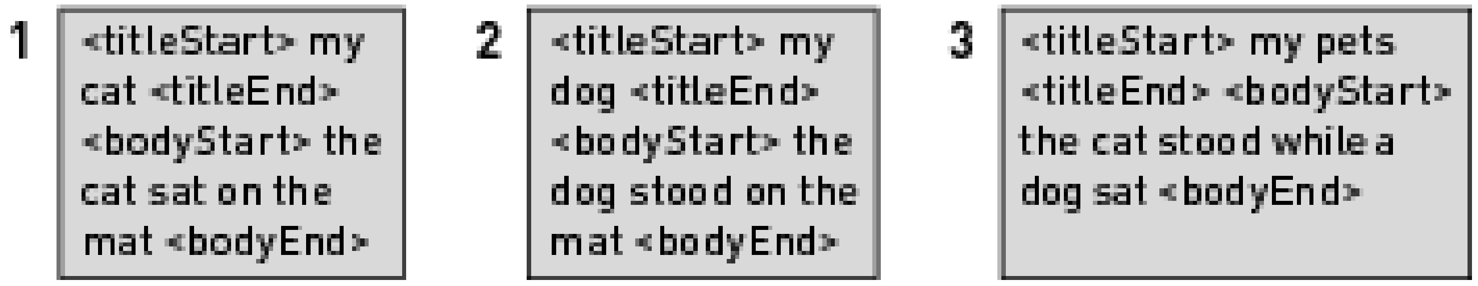
2 <titleStart> my dog <titleEnd> <bodyStart> the dog stood on the mat <bodyEnd>

3 <titleStart> my pets <titleEnd> <bodyStart> the cat stood while a dog sat <bodyEnd>

The pages as the browser and search engine see them.

When written in HTML the Web page typically has a **title** between the special tags <titleStart> and <titleEnd>. (The actual HTML tags are slightly different.) An intelligent search engine takes advantage of noticing the title!

HTML: Planning an improved improved tiny web index



1 <titleStart> my cat <titleEnd> <bodyStart> the cat sat on the mat <bodyEnd>

2 <titleStart> my dog <titleEnd> <bodyStart> the dog stood on the mat <bodyEnd>

3 <titleStart> my pets <titleEnd> <bodyStart> the cat stood while a dog sat <bodyEnd>

Our improved tiny web index will notice and include HTML tags.

Since the spiders see the actual text that generates the Web page, the first "word" to index is the HTML tag <titleStart>, the next words are the ones in the title, then the HTML tag <titleEnd>, then the HTML tag for starting the body, followed by the body of the page and finally the HTML tag for ending.

HTML: Indexing a single page, including HTML tags

For example, for the page entitled “My Cat” we have the following indexing.

<titleStart>	my	cats	<titleEnd>
1	2	3	4
<bodyStart>	the	cat	sat
5	6	7	8
on	the	mat	<bodyEnd>
9	10	11	12

HTML: The tiny web index #3

Word	Page, Position				
a	3-10				
cat	1-3	1-7	3-7		
dog	2-3	2-7	3-11		
mat	1-11	2-11			
my	1-2	2-2	3-2		
on	1-9	2-9			
pets	3-3				
sat	1-8	3-12			
stood	2-8	3-8			
the	1-6	1-10	2-6	2-10	3-6
while	3-9				
<titleStart>	1-1	2-1	3-1		
<titleEnd>	1-4	2-4	3-4		
<bodyStart>	1-5	2-5	3-5		
<bodyEnd>	1-12	2-12	3-13		

HTML: Using index #3

Suppose a user searches for **dog**. A page in which **dog** is in the title is probably a stronger match.

Each time a page is found containing the word **dog**, the engine can check whether this word is actually part of the web page title. It does this by comparing the positions of <**titleEnd**> and **dog** and <**titleStart**>. If the keyword falls between the two title markers, then this web page is more highly related than if it occurs elsewhere.

By looking at our index, we see the following cases:

Page	titleStart	dog	titleEnd	Start < dog < End?
2	1	3	4	yes
2	1	7	4	no
3	1	11	4	no

This technique is The Metaword Trick.

Maps and Indexes have solved the pagematch problem

From what we have seen, the impossible problem of quickly responding to a request to find keywords in all the webpages in the world has become the possible problem of intelligently searching a single index file.

Just as with card catalogs and an index at the back of a book, the creation of an index file for the web takes a great deal of time, and space.

Google, for instance, has created enormous collections of computer servers whose job is to collect all the information on all the web pages and create, update, and analyze the corresponding index file. This means that the index file is actually always out of date (like Google Street View) but regularly updated piece by piece.

Computational Thinking: Computer Tricks

The search engine may seem to be intelligent - it's answering your questions, after all. But actually, we have simply figured out a number of tricks that make it possible to come up with reasonable approximations of good answers:

- The Cache Trick
- The Map Trick
- The Index Trick
- The Nearness Trick
- The Word Location Trick
- The MetaWord Trick

These are examples of computational thinking in action: given a problem, how we can use the strengths of a computer (ability to store information, to gather and remember new information, look up information quickly, repeat operations) to simulate the abilities of a human agent (read all the web pages, and find the matching ones).

After finding matches, we need to do ranking!

Even with the tricks we have described, it is common for a search engine to discover thousands or millions of matching web pages.

The mapping and matching algorithm are only the beginning of the process of responding to your web search.

Next it will be necessary to consider the page rank algorithm, which considers all the matching pages that have been found, and sorts them in order of importance, so that even with millions of matches, most users know their best choice is a match on one of the first few pages.

Socrative Quiz PartII_Quiz2

IUZGAZ34E

Use the table below to answer the following questions.

Word	Page, Position				
a	3-10				
cat	1-3	1-7	3-7		
dog	2-3	2-7	3-11		
mat	1-11	2-11			
my	1-2	2-2	3-2		
on	1-9	2-9			
pets	3-3				
sat	1-8	3-12			
stood	2-8	3-8			
the	1-6	1-10	2-6	2-10	3-6
while	3-9				
<titleStart>	1-1	2-1	3-1		
<titleEnd>	1-4	2-4	3-4		
<bodyStart>	1-5	2-5	3-5		
<bodyEnd>	1-12	2-12	3-13		

1. The word pets is in only one of the three titles.

2. The word sat is adjacent to cat but not to dog.
3. The word mat only occurs in the body of the page and not in the title.
4. The word my only occurs in the title of the pages and not in the body.