How Internet Search Engines Work

by Curt Franklin

The good news about the Internet and its most visible component, the World Wide Web, is that there are hundreds of millions of pages available, waiting to present information on an amazing variety of topics. The bad news about the Internet is that there are hundreds of millions of pages available, most of them titled according to the whim of their author, almost all of them sitting on servers with cryptic names. When you need to know about a particular subject, how do you know which pages to read? If you're like most people, you visit an **Internet search engine**.

Internet search engines are special sites on the Web that are designed to help people find information stored on other sites. There are differences in the ways various search engines work, but they all perform three basic tasks:

- They search the Internet -- or select pieces of the Internet -- based on important words.
- They keep an index of the words they find, and where they find them.
- They allow users to look for words or combinations of words found in that index.

Early search engines held an index of a few hundred thousand pages and documents, and received maybe one or two thousand inquiries each day. Today, a top search engine will index hundreds of millions of pages, and respond to tens of millions of queries per day. In this edition of **HowStuffWorks**, we'll tell you how these major tasks are performed, and how Internet search engines put the pieces together in order to let you find the information you need on the Web.

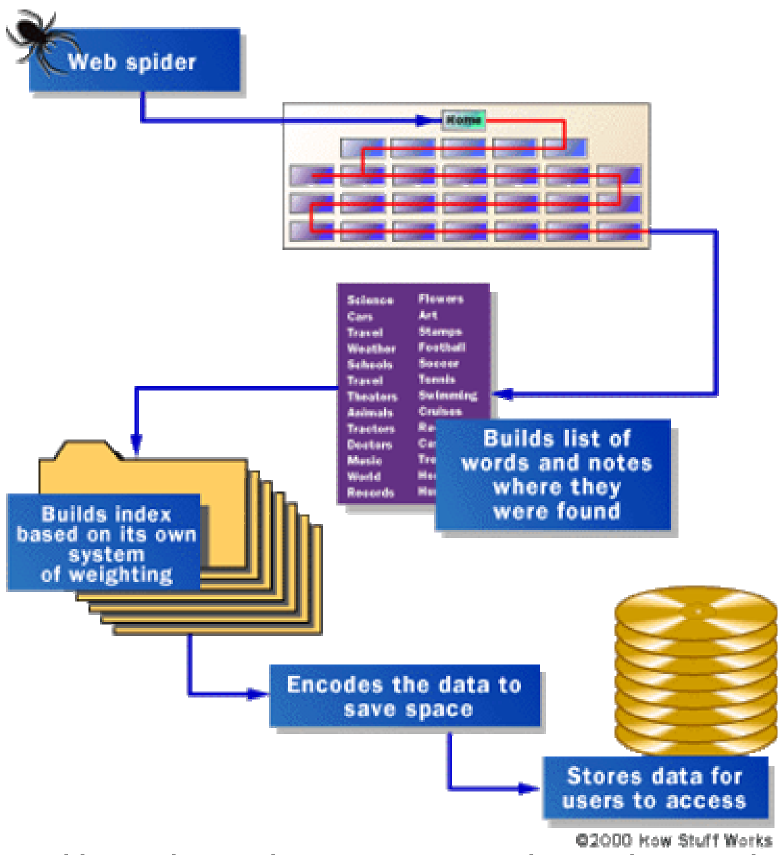
Looking at the Web

When most people talk about Internet search engines, they really mean World Wide Web search engines. Before the Web became the most visible part of the Internet, there were already search engines in place to help people find information on the Net. Programs with names like "gopher" and "Archie" kept indexes of files stored on servers connected to the Internet, and dramatically reduced the amount of time required to find programs and documents. In the late 1980s, getting serious value from the Internet meant knowing how to use gopher, Archie, Veronica and the rest. Today, most Internet users limit their searches to the Web, so we'll limit this article to search engines that focus on the contents of Web pages.

An Itsy-Bitsy Beginning

Before a search engine can tell you where a file or document is, it must be found. To find information on the hundreds of millions of Web pages that exist, a search engine employs special software robots, called **spiders**, to build lists of the words found on Web sites. When a spider is building its lists, the process is called **Web crawling**. (There are some disadvantages to calling part of the Internet the World Wide Web -- a large set of arachnid-centric names for tools is one of them.) In order to build and maintain a useful list of words, a search engine's spiders have to look at a lot of pages.

How does any spider start its travels over the Web? The usual starting points are lists of heavily used servers and very popular pages. The spider will begin with a popular site, indexing the words on its pages and following every link found within the site. In this way, the spidering system quickly begins to travel, spreading out across the most widely used portions of the Web.



"Spiders" take a Web page's content and create key search words that enable online users to find pages they're looking for.

Google.com began as an academic search engine. In the paper that describes how the system was built, Sergey Brin and Lawrence Page give an example of how quickly their spiders can work. They built their initial system to use multiple spiders, usually three at one time. Each spider could keep about 300 connections to Web pages open at a time. At its peak performance, using four spiders, their system could crawl over 100 pages per second, generating around 600 kilobytes of data each second.

Keeping everything running quickly meant building a system to feed necessary information to the spiders. The early Google system had a server dedicated to providing URLs to the spiders. Rather than depending on an Internet service provider for the domain name server (DNS) that translates a server's name into an address, Google had its own DNS, in order to keep delays to a minimum.

When the Google spider looked at an HTML page, it took note of two things:

- The words within the page
- Where the words were found

Words occurring in the title, subtitles, **meta tags** and other positions of relative importance were noted for special consideration during a subsequent user search. The Google spider was built to index every significant word on a page, leaving out the articles "a," "an" and "the." Other spiders take different approaches.

These different approaches usually attempt to make the spider operate faster, allow users to search more efficiently, or both. For example, some spiders will keep track of the words in the title, sub-headings and links, along with the 100 most frequently used words on the page and each word in the first 20 lines of text. Lycos is said to use this approach to spidering the Web.

Other systems, such as AltaVista, go in the other direction, indexing every single word on a page, including "a," "an," "the" and other "insignificant" words. The push to completeness in this approach is matched by other systems in the attention given to the unseen portion of the Web page, the **meta tags**.

Meta Tags

Meta tags allow the owner of a page to specify key words and concepts under which the page will be indexed. This can be helpful, especially in cases in which the words on the page might have double or triple meanings -- the meta tags can guide the search engine in choosing which of the several possible meanings for these words is correct. There is, however, a danger in over-reliance on meta tags, because a careless or unscrupulous page owner might add meta tags that fit very popular topics but have nothing to do with the actual contents of the page. To protect against this, spiders will correlate meta tags with page content, rejecting the meta tags that don't match the words on the page.

All of this assumes that the owner of a page actually wants it to be included in the results of a search engine's activities. Many times, the page's owner doesn't want it showing up on a major search engine, or doesn't want the activity of a spider accessing the page. Consider, for example, a game that builds new, active pages each time sections of the page are displayed or new links are followed. If a Web spider accesses one of these pages, and begins following all of the links for new pages, the game could mistake the activity for a high-speed human player and spin out of control. To avoid situations like this, the **robot exclusion protocol** was developed. This protocol, implemented in the meta-tag section at the beginning of a Web page, tells a spider to leave the page alone -- to neither index the words on the page nor try to follow its links.

Building the Index

Once the spiders have completed the task of finding information on Web pages (and we should note that this is a task that is never actually completed -- the constantly changing nature of the Web means that the spiders are always crawling), the search engine must store the information in a way that makes it useful. There are two key components involved in making the gathered data accessible to users:

- The **information stored with the data**
- The **method by which the information is indexed**

In the simplest case, a search engine could just store the word and the URL where it was found. In reality, this would make for an engine of limited use, since there would be no way of telling whether the word was used in an important or a trivial way on the page, whether the word was used once or many times or whether the page contained links to other pages containing the word. In other words, there would be no way of building the **ranking** list that tries to present the most useful pages at the top of the list of search results.

To make for more useful results, most search engines store more than just the word and URL. An engine might store the number of times that the word appears on a page. The engine might assign a **weight** to each entry, with increasing values assigned to words as they appear near the top of the document, in sub-headings, in links, in the meta tags or in the title of the page. Each commercial search engine has a different formula for assigning weight to the words in its index. This is one of the reasons that a search for the same word on different search engines will produce different lists, with the pages presented in different orders.

Regardless of the precise combination of additional pieces of information stored by a search engine, the data will be **encoded** to save storage space. For example, the original Google paper describes using 2 bytes, of 8 bits each, to store information on weighting -- whether the word was capitalized, its font size, position, and other information to help in ranking the hit. Each factor might take up 2 or 3 bits within the 2-byte grouping (8 bits = 1 byte). As a result, a great deal of information can be stored in a very compact form. After the information is compacted, it's ready for indexing.

An index has a single purpose: It allows information to be found as quickly as possible. There are quite a few ways for an index to be built, but one of the most effective ways is to build a **hash table**. In **hashing**, a formula is applied to attach a numerical value to each word. The formula is designed to evenly distribute the entries across a predetermined number of divisions. This numerical distribution is different from the distribution of words across the alphabet, and that is the key to a hash table's effectiveness.

In English, there are some letters that begin many words, while others begin fewer. You'll find, for example, that the "M" section of the dictionary is much thicker than the "X" section. This inequity means that finding a word beginning with a very "popular" letter could take much longer than finding a word that begins with a less popular one. Hashing evens out the difference, and reduces the average time it takes to find an entry. It also separates the index from the actual entry. The hash table contains the hashed number along with a pointer to the actual data, which can be sorted in whichever way allows it to be stored most efficiently. The combination of efficient indexing and effective storage makes it possible to get results quickly, even when the user creates a complicated search.

Building a Search

Searching through an index involves a user building a **query** and submitting it through the search engine. The query can be quite simple, a single word at minimum. Building a more complex query requires the use of **Boolean operators** that allow you to refine and extend the terms of the search.

The Boolean operators most often seen are:

- **AND** - All the terms joined by "AND" must appear in the pages or documents. Some search engines substitute the operator "+" for the word AND.
- **OR** - At least one of the terms joined by "OR" must appear in the pages or documents.
- **NOT** - The term or terms following "NOT" must not appear in the pages or documents. Some search engines substitute the operator "-" for the word NOT.
- **FOLLOWED BY** - One of the terms must be directly followed by the other.
- **NEAR** - One of the terms must be within a specified number of words of the other.
- **Quotation Marks** - The words between the quotation marks are treated as a phrase, and that phrase must be found within the document or file.

Searching for Sport

Search engines have become such an integral part of our lives that at least one organized game has evolved around this tool. In **Googlewhacking**, you type two words into the Google search engine in the hopes of receiving *exactly* one result -- a single Web page on which both of those words appear. This is a **pure whack**.

It's quite a difficult task -- you need to choose two completely unrelated words or else you'll get a whole lot more than one result, but with many completely unrelated words you get zero results.

If you achieve a pure whack, you can submit it to www.googlewhack.com, where it is posted in **The Whack Stack** (along with your name, or whatever you want to call yourself) for all to see. One pure whack currently in The Whack Stack is "ambidextrous scallywags."

Future Search

The searches defined by Boolean operators are **literal** searches -- the engine looks for the words or phrases exactly as they are entered. This can be a problem when the entered words have multiple meanings. "Bed," for example, can be a place to sleep, a place where flowers are planted, the storage space of a truck or a place where fish lay their eggs. If you're interested in only one of these meanings, you might not want to see pages featuring all of the others. You can build a literal search that tries to eliminate unwanted meanings, but it's nice if the search engine itself can help out.

One of the areas of search engine research is **concept-based** searching. Some of this research involves using statistical analysis on pages containing the words or phrases you search for, in order to find other pages you might be interested in. Obviously, the information stored about each page is greater for a concept-based search engine, and far more processing is required for each search. Still, many groups are working to improve both results and performance of this type of search engine. Others have moved on to another area of research, called **natural-language queries.**

The idea behind natural-language queries is that you can type a question in the same way you would ask it to a human sitting beside you -- no need to keep track of Boolean operators or complex query structures. The most popular natural language query site today is AskJeeves.com, which parses the query for keywords that it then applies to the index of sites it has built. It only works with simple queries; but competition is heavy to develop a natural-language query engine that can accept a query of great complexity.