Web Search Engines: Part 2

David Hawking
CSIRO ICT Centre

A data processing "miracle" provides responses to hundreds of millions of Web searches each day.

Part 1 of this two-part series (How Things Work, June 2006, pp. 86-88) described search engine infrastructure and algorithms for crawling the Web. Part 2 reviews the algorithms and data structures required to index 400 terabytes of Web page text and deliver high-quality results in response to hundreds of millions of queries each day.

INDEXING ALGORITHMS

Search engines use an inverted file to rapidly identify indexing terms—the documents that contain a particular word or phrase (J. Zobel and A. Moffat, "Inverted Files for Text Search Engines," to be published in *ACM Computing Surveys*, 2006). An inverted file is a concatenation of the postings lists for each distinct term. In its simplest form, each postings list comprises a sorted list of the ID numbers of the documents that contain it. A fast-lookup term dictionary references the postings lists for each term.

An indexer can create an inverted file in two phases. In the first phase, *scanning*, the indexer scans the text of each input document. For each indexable term it encounters, the indexer writes a posting consisting of a document number and a term number to a temporary file. Because of the scanning process, this file will naturally be in document number order.

In the second phase, *inversion*, the indexer sorts the temporary file into term number order, with the document number as the secondary sort key. It also records the starting point and length of the lists for each entry in the term dictionary.

REAL INDEXERS

As Figure 1 shows, for high-quality rankings, real indexers store additional information in the postings, such as term frequency or positions. (For additional information, see S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine;" www-db.stanford.edu/pub/papers/google.pdf.)

Scaling up. The scale of the inversion problem for a Web-sized crawl is enormous: Estimating 500 terms in each of 20 billion pages, the temporary file might contain 10 trillion entries.

An obvious approach, document partitioning, divides up the URLs between machines in a cluster in the same way as the crawler. If the system uses 400 machines to index 20 billion pages, and the machines share the load evenly, then each machine manages a partition of 50 million pages.

Even with 400-fold partitioning, each inverted file contains around 25 billion entries, still a significant indexing challenge. An efficient indexer builds a partial inverted "file" in main memory as it scans documents and stops when available memory is exhausted. At that point, the indexer writes a partial inverted file to disk, clears memory, and starts indexing the next document. It then repeats this process until it has scanned all documents. Finally, it merges all the partial inverted files.

Term lookup. The Web's vocabulary is unexpectedly large, containing hundreds of millions of distinct terms. How can this be, you ask, when even the largest English dictionary lists only about a million words? The answer is that the Web includes documents in all languages, and that human authors have an apparently limitless propensity to create new words such as acronyms, trademarks, e-mail addresses, and proper names. People certainly want to search for R2-D2 and C-3PO as well as IBM, B-52, and, yes, Yahoo! and Google. Many nondictionary words are of course misspellings and typographical errors, but there is no safe way to eliminate them.

Search engines can choose from various forms of tries, trees, and hash tables for efficient term lookup (D.E. Knuth, *The Art of Computer Programming: Sorting and Searching*, Addison-Wesley, 1973). They can use a two-level structure to reduce disk seeks, an important consideration because modern CPUs can execute many millions of instructions in the time taken for one seek.

Compression. Indexers can reduce demands on disk space and memory by using compression algorithms for key data structures. Compressed data structures mean fewer disk accesses and can lead to faster indexing and faster query processing, despite the CPU cost of compression and decompression.

Phrases. In principle, a query processor can correctly answer phrase queries such as "National Science Foundation" by intersecting postings lists containing word position infor-

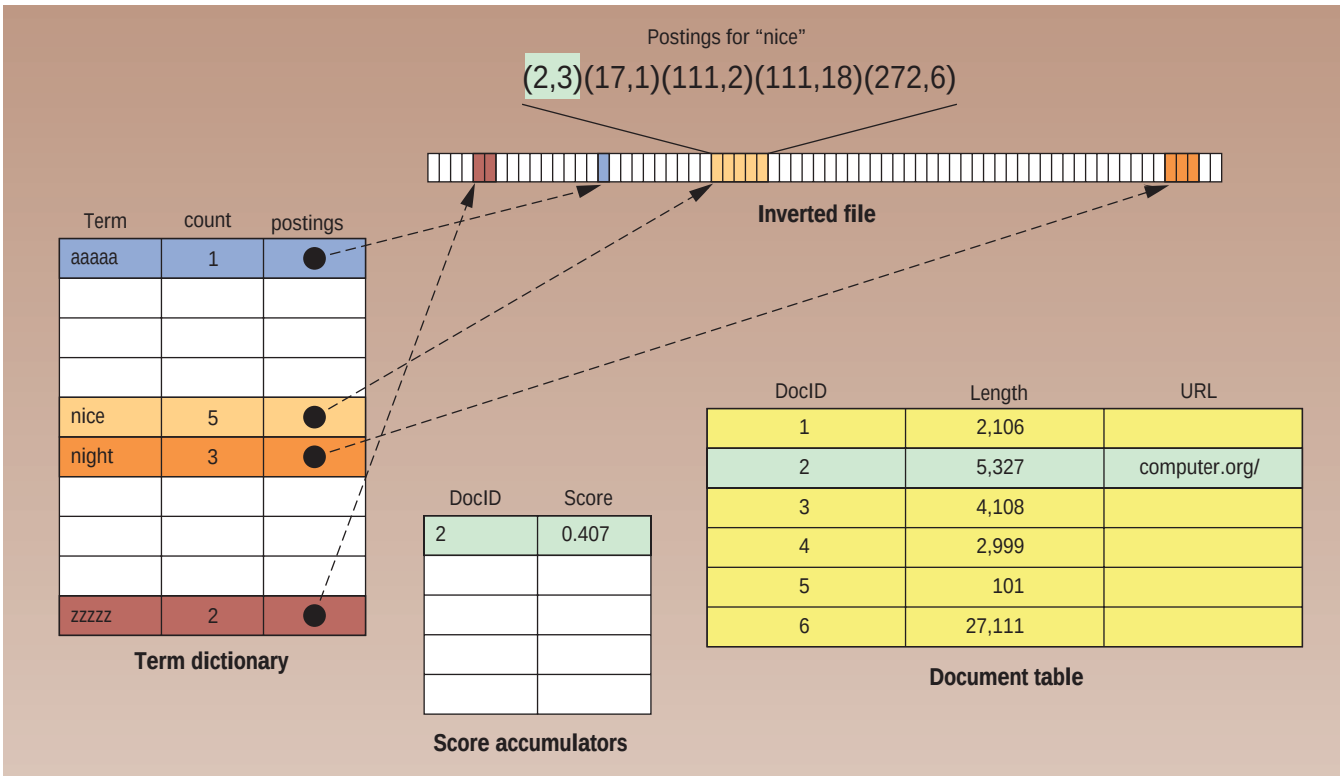

Figure 1. Inverted file index and associated data structures. In this simplified example, the alphabetically sorted term dictionary allows fast access to a word's postings list within the inverted file. Postings contain both a document number and a word position within the document. Note that "nice" occurs five times all told, twice in document 111. On the basis of the first posting, the query processor has calculated a relevance score for document 2.

mation. In practice, this is much too slow when postings lists are long.

Special indexing tricks permit a more rapid response. One trick is to precompute postings lists for common phrases. Another is to subdivide the postings list for a word into sublists, according to the word that follows the primary word. For example, postings for the word "electrical" might be divided into sublists for "electrical apparatus," "electrical current," "electrical engineering," and so on.

Anchor text. Web browsers highlight words in a Web page to indicate the presence of a link that users can click on. These words are known as *link anchor text*. Web search engines index anchor text with a link's target—as well as its source—because anchor text provides useful descriptions of the target, except "click here," of course. Pages that have many incoming links accumulate a variety of anchor text descriptions. The most useful descriptions can be repeated thousands of times, providing a strong signal of what the page is about. Anchor text contributes strongly to the quality of search results.

Link popularity score. Search engines assign pages a link popularity score derived from the frequency of incoming links. This can be a simple count or it can exclude links from within the same site. PageRank, a more sophisticated link popularity score, assigns different weights to links depending on the source's page rank. PageRank computation is an eigenvector calculation on the page-page link connectivity matrix.

Processing matrices of rank 20 billion is computationally impractical, and researchers have invested a great deal of brainpower in determining how to reduce the scale of PageRank-like problems. One approach is to compute HostRanks from the much smaller host-host connectivity matrix and distribute PageRanks to individual pages within each site afterwards. PageRank is a Google technology, but other engines use variants of this approach.

Query-independent score. Internally, search engines rank Web pages independently of any query, using a combination of query-independent factors such as link popularity, URL brevity, spam score, and perhaps the frequency with which users click them. A page with a high query-independent score has a higher a priori probability of retrieval than others that match the query equally well.

QUERY PROCESSING ALGORITHMS

By far the most common type of query that search engines receive consists of a small number of query words, without operators—for example, "Katrina" or "secretary of state." Several researchers have reported that the average query length is around 2.3 words.

By default, current search engines return only documents containing all

the query words. To achieve this, a simple-query processor looks up each query term in the term dictionary and locates its postings list. The processor simultaneously scans the postings lists for all the terms to find documents in common. It stops once it has found the required number of matching documents or when it reaches the end of a list.

In a document-partitioned environment, each machine in a cluster must answer the query on its subset of Web pages and then send the top-ranked results to a coordinating machine for merging and presentation.

REAL QUERY PROCESSORS

The major problem with the simple-query processor is that it returns poor results. In response to the query "the Onion" (seeking the satirical newspaper site), pages about soup and gardening would almost certainly swamp the desired result.

Result quality

Result quality can be dramatically improved if the query processor scans to the end of the lists and then sorts the long list of results according to a relevance-scoring function that takes into account the number of query term occurrences, document length, inlink score, anchor text matches, phrase matches, and so on. The MSN search engine reportedly takes into account more than 300 ranking factors.

Unfortunately, this approach is too computationally expensive to allow processing queries like "the Onion" quickly enough. The postings list for "the" contains billions of entries, and the number of documents needing to be scored and sorted—those that contain both "the" and "onion"—is on the order of tens of millions.

Speeding things up

Real search engines use many techniques to speed things up.

Skipping. Scanning postings lists one entry at a time is very slow. If the next document containing "Onion" is number 2,000,000 and the current position in the "the" list is 1,500,000, the search obviously should skip half a million postings in the latter list as fast as possible. A small amount of additional structure in postings lists permits the query processor to skip forward in steps of hundreds or thousands to the required document number.

Early termination. The query processor can save a great deal of computation if the indexer creates indexes in which it sorts postings lists in order of decreasing value. It can usually stop processing after scanning only a small fraction of the lists because later results are less likely to be valuable than those already seen. At first glance, early termination seems to be inconsistent with skipping and compression techniques, which require postings to be in document number order. But there is a solution.

Clever assignment of document numbers. Instead of arbitrarily numbering documents, the crawler or indexer can number them to reflect their decreasing query-independent score. In other words, document number 1 (or 0 if you are a mathematician!) is the document with the highest a priori probability of retrieval.

This approach achieves a win-win-win solution: effective postings compression, skipping, and early termination.

Caching. There is a strong economic incentive for search engines to use caching to reduce the cost of answering queries. In the simplest case, the search engine precomputes and stores HTML results pages for thousands of the most popular queries. A dedicated machine can use a simple in-memory lookup to answer such queries. The normal query processor also uses caching to reduce the cost of accessing commonly needed parts of term dictionaries and inverted files.

Limited space prevents discussion of many other fascinating aspects of search engine operation, such as estimating the number of hits for a query when it has not been fully evaluated, generating advertisements targeted to the search query, searching images and videos, merging search results from other sources such as news, generating spelling suggestions from query logs, creating query-biased result snippets, and performing linguistic operations such as word-stemming in a multilingual environment.

A high priority for search engine operation is monitoring the search quality to ensure that it does not decrease when a new index is installed or when the search algorithm is modified. But that is a story in itself.

David Hawking is a principal research scientist at CSIRO ICT Centre, Canberra, Australia, and chief scientist at funnelback.com. Contact him at david.hawking@csiro.au.

Computer welcomes your submissions to this bimonthly column. For additional information, contact Alf Weaver, the column editor, at weaver@cs.virginia.edu.