

Index (search engine)

From Wikipedia, the free encyclopedia

Search engine indexing entails how data is collected, parsed, and stored to facilitate fast and accurate retrieval. Index design incorporates interdisciplinary concepts from linguistics, cognitive psychology, mathematics, informatics, physics, and computer science. An alternate name for the process is Web indexing, within the context of search engines designed to find web pages on the Internet.

Popular engines focus on the full-text indexing of online, natural language documents^[1], yet there are other searchable media types such as video, audio^[2], and graphics^{[3][4]}. Meta search engines reuse the indices of other services and do not store a local index, whereas cache-based search engines permanently store the index along with the corpus. Unlike full text indices, partial text services restrict the depth indexed to reduce index size. Larger services typically perform indexing at a predetermined interval due to the required time and processing costs, whereas agent-based search engines index in real time.

Contents

- 1 Indexing
 - 1.1 Index Design Factors
 - 1.2 Index Data Structures
 - 1.3 Challenges in Parallelism
 - 1.4 Inverted indices
 - 1.5 Index Merging
 - 1.6 The Forward Index
 - 1.7 Compression
- 2 Document Parsing
 - 2.1 Challenges in Natural Language Processing
 - 2.2 Tokenization
 - 2.3 Language Recognition
 - 2.4 Format Analysis
 - 2.5 Section Recognition
 - 2.6 Meta Tag Indexing
- 3 See also
- 4 Further reading
- 5 References

Indexing

The goal of storing an index is to optimize the speed and performance of finding relevant documents for a search query. Without an index, the search engine would scan every document in the corpus, which would take a considerable amount of time and computing power. For example, an index of 10,000 documents can be queried within milliseconds, where a sequential scan of every word in 10,000 large documents could take hours. No search engine user would be comfortable waiting several hours to get search results. The trade off for the time saved during retrieval is that additional storage is required to store the index and that it takes a considerable amount of time to update.

Index Design Factors

Major factors in designing a search engine's architecture include:

Merge factors

How data enters the index, or how words or subject features are added to the index during corpus traversal, and whether multiple indexers can work asynchronously. The indexer must first check whether it is updating old content or adding new content. Traversal typically correlates to the data collection policy. Search engine index merging is similar in concept to the SQL Merge command and other merge algorithms.^[5]

Storage techniques

How to store the index data - whether information should be compressed or filtered

Index size

How much computer storage is required to support the index

Lookup speed

How quickly a word can be found in the inverted index. How quickly an entry in a data structure can be found, versus how quickly it can be updated or removed, is a central focus of computer science

Maintenance

Maintaining the index over time^[6]

Fault tolerance

How important it is for the service to be reliable, how to deal with index corruption, whether bad data can be treated in isolation, dealing with bad hardware, partitioning schemes such as hash-based or composite partitioning^[7], data replication

Index Data Structures

Search engine architectures vary in how indexing is performed and in index storage to meet the various design factors. Types of indices include:

Suffix trees

Figuratively structured like a tree, supports linear time lookup. Built by storing the suffixes of words. Used for searching for patterns in DNA sequences and clustering. A major drawback is that the storage of a word in the tree may require more storage than storing the word itself.^[8] An alternate representation is a suffix array, which is considered to require less memory and supports compression like BWT.

Trees

An ordered tree data structure that is used to store an associative array where the keys are strings. Regarded as faster than a hash table, but are less space efficient. The suffix tree is a type of trie. Tries support extendible hashing, which is important for search engine indexing.^[9]

Inverted indices

Stores a list of occurrences of each atomic search criterion^[10], typically in the form of a hash table or binary tree^{[11][12]}.

Citation indices

Stores the existence of citations or hyperlinks between documents to support citation analysis, a subject of Bibliometrics.

Ngram indices

For storing sequences of n length of data to support other types of retrieval or text mining.^[13]

Term document matrices

Used in latent semantic analysis, stores the occurrences of words in documents in a two dimensional sparse matrix.

Challenges in Parallelism

A major challenge in the design of search engines is the management of parallel processes. There are many opportunities for race conditions and coherence faults. For example, a new document is added to the corpus and the index must be updated, but the index simultaneously needs to continue responding to search queries. This is a collision between two competing tasks. Consider that authors are producers of information, and a crawler is the consumer of this information, grabbing the text and storing it in a cache (or corpus). The forward index is the consumer of the information produced by the corpus, and the inverted index is the consumer of information produced by the forward index. This is commonly referred to as a **producer-consumer model**. The indexer is the producer of searchable information and users are the consumers that need to search. The challenge is magnified when working with distributed storage and distributed processing. In an effort to scale with larger amounts of indexed information, the search engine's architecture may involve distributed computing, where the search engine consists of several machines operating in unison. This increases the possibilities for incoherency and makes it more difficult to maintain a fully-synchronized, distributed, parallel architecture.^[14]

Inverted indices

Many search engines incorporate an inverted index when evaluating a search query to quickly locate the documents which contain the words in a query and rank these documents by relevance. The inverted index stores a list of the documents for each word. The search engine can retrieve the matching documents quickly using direct access to find the documents for a word. The following is a simplified illustration of the inverted index:

Inverted Index	
Word	Documents
the	Document 1, Document 3, Document 4, Document 5
cow	Document 2, Document 3, Document 4
says	Document 5
moo	Document 7

The above figure is a simplified form of a Boolean index. Such an index would only serve to determine whether a document matches a query, but would not contribute to ranking matched documents. In some designs the index includes additional information such as the frequency of each word in each document or the positions of the word in each document.^[15] With position, the search algorithm can identify word proximity to support searching for phrases. Frequency can be used to help in ranking the relevance of documents to the query. Such topics are the central research focus of information retrieval.

The inverted index is a sparse matrix given that words are not present in each document. It is stored differently than a two dimensional array to reduce memory requirements. The index is similar to the term document matrices employed by latent semantic analysis. The inverted index can be considered a form of a hash table. In some cases the index is a form of a binary tree, which requires additional storage but may reduce the lookup time. In larger indices the architecture is typically distributed.^[16]

Inverted indices can be programmed in several computer programming languages.^{[17][18]}

Index Merging

The inverted index is filled via a merge or rebuild. A rebuild is similar to a merge but first deletes the contents of the inverted index. The architecture may be designed to support incremental indexing^[19], where a merge involves identifying the document or documents to add into or update in the index and parsing each document into words. For technical accuracy, a merge involves the unison of newly indexed documents, typically residing in virtual memory, with the index cache residing on one or more computer hard drives.

After parsing, the indexer adds the containing document to the document list for the appropriate words. The process of finding each word in the inverted index in order to denote that it occurred within a document may be too time consuming when designing a larger search engine, and so this process is commonly split up into the development of a forward index and the process of sorting the contents of the forward index for entry into the inverted index. The inverted index is named inverted because it is an inversion of the forward index.

The Forward Index

The forward index stores a list of words for each document. The following is a simplified form of the forward index:

Document	Words
Document 1	the,cow,says,moo
Document 2	the,cat,and,the,hut
Document 3	the,dish,ran,away,with,the,spoon

The rationale behind developing a forward index is that as documents are parsed, it is better to immediately store the words per document. The delineation enables asynchronous processing, which partially circumvents the inverted index update bottleneck.^[20] The forward index is sorted to transform it to an inverted index. The forward index is essentially a list of pairs consisting of a document and a word, collated by the document. Converting the forward index to an inverted index is only a matter of sorting the pairs by the words. In this regard, the inverted index is a word-sorted forward index.

Compression

Generating or maintaining a large-scale search engine index represents a significant storage and processing challenge. Many search engines utilize a form of compression to reduce the size of the indices on disk.^[21] Consider the following scenario for a full text, Internet, search engine.

- An estimated 2,000,000,000 different web pages exist as of the year 2000^[22]
- A fictitious estimate of 250 words per webpage on average, based on the assumption of being similar to the pages of a novel.^[23]
- It takes 8 bits (or 1 byte) to store a single character. Some encodings use 2 bytes per character [24][25]
- The average number of characters in any given word on a page can be estimated at 5 (Wikipedia:Size comparisons)
- The average personal computer comes with about 20 gigabytes of usable space^[26]

Given these estimates, generating an uncompressed index (assuming a non-conflated, simple, index)

for 2 billion web pages would need to store 5 billion word entries. At 1 byte per character, or 5 bytes per word, this would require 25 gigabytes of storage space alone, more than the average size a personal computer's free disk space. This space is further increased in the case of a distributed storage architecture that is fault-tolerant. Using compression, the index size can be reduced to a portion of its size, depending on which compression techniques are chosen. The trade off is the time and processing power required to perform compression.

Notably, large scale search engine designs incorporate the cost of storage, and the costs of electricity to power the storage. Compression, in this regard, is a measure of cost as well.

Document Parsing

Document parsing involves breaking apart the components (words) of a document or other form of media for insertion into the forward and inverted indices. For example, if the full contents of a document consisted of the sentence "Hello World", there would typically be two words found, the token "Hello" and the token "World". In the context of search engine indexing and natural language processing, parsing is more commonly referred to as tokenization, and sometimes word boundary disambiguation, tagging, text segmentation, content analysis, text analysis, text mining, concordance generation, Speech segmentation, lexing, or lexical analysis. The terms 'indexing', 'parsing', and 'tokenization' are used interchangeably in corporate slang.

Natural language processing, as of 2006, is the subject of continuous research and technological improvement. There are a host of challenges in tokenization, in extracting the necessary information from documents for indexing to support quality searching. Tokenization for indexing involves multiple technologies, the implementation of which are commonly kept as corporate secrets.

Challenges in Natural Language Processing

Word Boundary Ambiguity - native English speakers can at first consider tokenization to be a straightforward task, but this is not the case with designing a multilingual indexer. In digital form, the text of other languages such as Chinese, Japanese or Arabic represent a greater challenge as words are not clearly delineated by whitespace. The goal during tokenization is to identify words for which users will search. Language specific logic is employed to properly identify the boundaries of words, which is often the rationale for designing a parser for each language supported (or for groups of languages with similar boundary markers and syntax).

Language Ambiguity - to assist with properly ranking matching documents, many search engines collect additional information about words, such as its language or lexical category (part of speech). These techniques are language-dependent as the syntax varies among languages. Documents do not always clearly identify the language of the document or represent it accurately. In tokenizing the document, some search engines attempt to automatically identify the language of the document.

Diverse File Formats - in order to correctly identify what bytes of a document represent characters, the file format must be correctly handled. Search engines which support multiple file formats must be able to correctly open and access the document and be able to tokenize the characters of the document.

Faulty Storage - the quality of the natural language data is not always assumed to be perfect. An unspecified amount of documents, particular on the Internet, do not always closely obey proper file protocol. Binary characters may be mistakenly encoded into various parts of a document. Without recognition of these characters and appropriate handling, the index quality or indexer performance

could degrade.

Tokenization

Unlike literate human adults, computers are not inherently aware of the structure of a natural language document and do not instantly recognize words and sentences. To a computer, a document is only a big sequence of bytes. Computers do not know that a space character between two sequences of characters means that there are two separate words in the document. Instead, a computer program is developed by humans which trains the computer, or instructs the computer, how to identify what constitutes an individual or distinct word, referred to as a token. This program is commonly referred to as a tokenizer or parser or lexer. Many search engines, as well as other natural language processing software, incorporate specialized programs for parsing, such as YACC OR Lex.

During tokenization, the parser identifies sequences of characters, which typically represent words. Commonly recognized tokens include punctuation, sequences of numerical characters, alphabetical characters, alphanumerical characters, binary characters (backspace, null, print, and other antiquated print commands), whitespace (space, tab, carriage return, line feed), and entities such as email addresses, phone numbers, and URLs. When identifying each token, several characteristics may be stored such as the token's case (upper, lower, mixed, proper), language or encoding, lexical category (part of speech, like 'noun' or 'verb'), position, sentence number, sentence position, length, and line number.

Language Recognition

If the search engine supports multiple languages, a common initial step during tokenization is to identify each document's language, given that many of the later steps are language dependent (such as stemming and part of speech tagging). Language recognition is the process by which a computer program attempts to automatically identify, or categorize, the language of a document. Other names for language recognition include language classification, language analysis, language identification, and language tagging. Automated language recognition is the subject of ongoing research in natural language processing. Finding which language the words belongs to may involve the use of a language recognition chart.

Format Analysis

Depending on whether the search engine supports multiple document formats, documents must be prepared for tokenization. The challenge is that many document formats contain, in addition to textual content, formatting information. For example, HTML documents contain HTML tags, which specify formatting information, like whether to start a new line, or display a word in **bold**, or change the font size or family. If the search engine were to ignore the difference between content and **markup**, the segments would also be included in the index, leading to poor search results. Format analysis involves the identification and handling of formatting content embedded within documents which control how the document is rendered on a computer screen or interpreted by a software program. Format analysis is also referred to as structure analysis, format parsing, tag stripping, format stripping, text normalization, text cleaning, or text preparation. The challenge of format analysis is further complicated by the intricacies of various file formats. Certain file formats are proprietary and very little information is disclosed, while others are well documented. Common, well-documented file formats that many search engines support include:

- Microsoft Word

- Microsoft Excel
- Microsoft Powerpoint
- IBM Lotus Notes
- HTML
- ASCII text files (a text document without any formatting)
- Adobe's Portable Document Format (PDF)
- PostScript (PS)
- LaTeX
- The UseNet archive (NNTP) and other deprecated bulletin board formats
- XML and derivatives like RSS
- SGML (this is more of a general protocol)
- Multimedia meta data formats like ID3

Techniques for dealing with various formats include:

- Using a publicly available commercial parsing tool that is offered by the organization which developed, maintains, or owns the format
- Writing a custom parser

Some search engines support inspection of files that are stored in a compressed, or encrypted, file format. If working with a compressed format, then the indexer first decompresses the document, which may result in one or more files, each of which must be indexed separately. Commonly supported compressed file formats include:

- ZIP - Zip File
- RAR - Archive File
- CAB - Microsoft Windows Cabinet File
- Gzip - Gzip file
- BZIP - Bzip file
- TAR, GZ, and TAR.GZ - Unix Gzip'ped Archives

Format analysis can involve quality improvement methods to avoid including 'bad information' in the index. Content can manipulate the formatting information to include additional content. Examples of abusing document formatting for spamdexing:

- Including hundreds or thousands of words in a section which is hidden from view on the computer screen, but visible to the indexer, by use of formatting (e.g. hidden "div" tag in HTML, which may incorporate the use of CSS or Javascript to do so).
- Setting the foreground font color of words to the same as the background color, making words hidden on the computer screen to a person viewing the document, but not hidden to the indexer.

Section Recognition

Some search engines incorporate section recognition, the identification of major parts of a document, prior to tokenization. Not all the documents in a corpus read like a well-written book, divided into organized chapters and pages. Many documents on the web contain erroneous content and side-sections which do not contain primary material, that which the document is about, such as newsletters and corporate reports. For example, this article may display a side menu with words inside links to other web pages. Some file formats, like HTML or PDF, allow for content to be displayed in columns. Even though the content is displayed, or rendered, in different areas of the view, the raw markup content may store this information sequentially. Words that appear in the raw source content sequentially are indexed sequentially, even though these sentences and paragraphs are

rendered in different parts of the computer screen. If search engines index this content as if it were normal content, a dilemma ensues where the quality of the index is degraded and search quality is degraded due to the mixed content and improper word proximity. Two primary problems are noted:

- Content in different sections is treated as related in the index, when in reality it is not
- Organizational 'side bar' content is included in the index, but the side bar content does not contribute to the meaning of the document, and the index is filled with a poor representation of its documents, assuming the goal is to go after the meaning of each document, a sub-goal of providing quality search results.

Section analysis may require the search engine to implement the rendering logic of each document, essentially an abstract representation of the actual document, and then index the representation instead. For example, some content on the Internet is rendered via Javascript. Viewers of web pages in web browsers see this content. If the search engine does not render the page and evaluate the Javascript within the page, it would not 'see' this content in the same way, and index the document incorrectly. Given that some search engines do not bother with rendering issues, many web page designers avoid displaying content via Javascript or use the Noscript tag to ensure that the web page is indexed properly. At the same time, this fact is also exploited to cause the search engine indexer to 'see' different content than the viewer.

Meta Tag Indexing

Specific documents offer embedded meta information such as the author, keywords, description, and language. For HTML pages, the meta tag contains keywords which are also included in the index. During earlier growth periods in the Internet and search engine technology (more so, the hardware on which it runs) would only index the keywords in the meta tags for the forward index (and still applying techniques such as stemming and stop words). The full document would not be parsed. At this time, full-text indexing was not as well established, nor was the hardware to support such technology. The design of the HTML markup language initially included support for meta tags for this very purpose of being properly and easily indexed, without requiring tokenization.^[27]

As the Internet grew (the number of users capable of browsing the web and the number of websites increased and the technology for making websites and hosting websites improved), many brick-and-mortar corporations went 'online' in the mid 1990s and established corporate websites. The keywords used to describe webpages (many of which were corporate-oriented webpages similar to product brochures) changed from descriptive keywords to marketing-oriented keywords designed to drive sales by placing the webpage high in the search results for specific search queries. The fact that these keywords were subjectively-specified was leading to spamdexing, which drove many search engines to adopt full-text indexing technologies in the 1990s. Search engine designers and companies could only place so many 'marketing keywords' into the content of a webpage before draining it of all interesting and useful information. Given that conflict of interest with the business goal of designing user-oriented websites which were 'sticky', the customer lifetime value equation was changed to incorporate more useful content into the website in hopes of retaining the visitor. In this sense, full-text indexing was more objective and increased the quality of search engine results, as it was one more step away from subjective control of search engine result placement, which in turn furthered research of full-text indexing technologies.

In the context of Desktop search, many solutions incorporate meta tags to provide a way for authors to further customize how the search engine will index content from various files that is not evident from the file content. Desktop search is under the control of the user and the changes in that context only serve to help, unlike Internet search engines which must focus more on the full text index.

See also

- Information retrieval
- Document Retrieval
- Information extraction
- Search engine
- Vertical search
- Desktop search
 - Windows indexing service^[28]
- List of search engines
- Text Retrieval
- Latent Semantic Indexing
- Keyword In Context Indexing
- Concordance
- Controlled vocabulary
- Natural language processing
- Text mining
- Content analysis

Further reading

- R. Bayer and E. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 173-189, 1972.
- Donald E. Knuth. *The art of computer programming, volume 1 (3rd ed.): fundamental algorithms*, Addison Wesley Longman Publishing Co. Redwood City, CA, 1997.
- Donald E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*, Addison Wesley Longman Publishing Co. Redwood City, CA, 1998.
- Gerald Salton. *Automatic text processing*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1988.
- Gerard Salton. Michael J. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill, Inc., New York, NY, 1986.
- Gerard Salton. Lesk, M.E.: Computer evaluation of indexing and text processing. *Journal of the ACM*. January 1968.
- Gerard Salton. *The SMART Retrieval System - Experiments in Automatic Document Processing*. Prentice Hall Inc., Englewood Cliffs, 1971.
- Gerard Salton. *The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley, Reading, Mass., 1989.
- Baeza-Yates, R., Ribeiro-Neto, B.: *Modern Information Retrieval*. Chapter 8. ACM Press 1999.
- G. K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.
- Adelson-Velskii, G.M., Landis, E. M.: An information organization algorithm. *DANSSSR*, 146, 263-266 (1962).
- Edward H. Sussenguth, Jr., Use of tree structures for processing files, *Communications of the ACM*, v.6 n.5, p.272-279, May 1963
- Harman, D.K., et al: Inverted files. In *Information Retrieval: Data Structures and Algorithms*, Prentice-Hall, pp 28-43, 1992.
- Lim, L., et al: Characterizing Web Document Change, LNCS 2118, 133–146, 2001.
- Lim, L., et al: Dynamic Maintenance of Web Indexes Using Landmarks. *Proc. of the 12th W3 Conference*, 2003.
- Moffat, A., Zobel, J.: Self-Indexing Inverted Files for Fast Text Retrieval. *ACM TIS*, 349–379, October 1996, Volume 14, Number 4.
- Mehlhorn, K.: *Data Structures and Efficient Algorithms*, Springer Verlag, EATCS

Monographs, 1984.

- Mehlhorn, K., Overmars, M.H.: Optimal Dynamization of Decomposable Searching Problems. IPL 12, 93–98, 1981.
- Mehlhorn, K.: Lower Bounds on the Efficiency of Transforming Static Data Structures into Dynamic Data Structures. Math. Systems Theory 15, 1–16, 1981.
- Koster, M.: ALIWEB: Archie-Like indexing in the Web. Computer Networks and ISDN Systems, Vol. 27, No. 2 (1994) 175-182 (also see Proc. First Int'l World Wide Web Conf., Elsevier Science, Amsterdam, 1994, pp. 175-182)
- Serge Abiteboul and Victor Vianu. Queries and Computation on the Web (<http://dbpubs.stanford.edu:8090/pub/showDoc.Fulltext?lang=en&doc=1996-20&format=text&compression=&name=1996-20.text>). Proceedings of the International Conference on Database Theory. Delphi, Greece 1997.
- Ian H Witten, Alistair Moffat, and Timothy C. Bell. Managing Gigabytes: Compressing and Indexing Documents and Images. New York: Van Nostrand Reinhold, 1994.
- A. Emtage and P. Deutsch, "Archie--An Electronic Directory Service for the Internet." Proc. Usenix Winter 1992 Tech. Conf., Usenix Assoc., Berkeley, Calif., 1992, pp. 93-110.
- M. Gray, World Wide Web Wanderer (<http://www.mit.edu/people/mkgray/net/>).
- D. Cutting and J. Pedersen. "Optimizations for Dynamic Inverted Index Maintenance." Proceedings of the 13th International Conference on Research and Development in Information Retrieval, pp. 405-411, September 1990.
- Searching semantic information (<http://motoresrecuperacion.iespana.es/>)

References

1. ^ Clarke, C., Cormack, G.: Dynamic Inverted Indexes for a Distributed Full-Text Retrieval System. TechRep MT-95-01, University of Waterloo, February 1995.
2. ^ Stephen V. Rice, Stephen M. Bailey. Searching for Sounds (<http://www.comparisonics.com/SearchingForSounds.html>). Comparisonics Corporation. May 2004. Verified Dec 2006
3. ^ Charles E. Jacobs, Adam Finkelstein, David H. Salesin. Fast Multiresolution Image Querying (<http://grail.cs.washington.edu/projects/query/mrquery.pdf>). Department of Computer Science and Engineering, University of Washington. 1995. Verified Dec 2006
4. ^ Lee, James. Software Learns to Tag Photos (http://www.technologyreview.com/read_article.aspx?id=17772&ch=infotech). MIT Technology Review. November 09, 2006. Pg 1-2. Verified Dec 2006. Commercial external link
5. ^ Brown, E.W.: Execution Performance Issues in Full-Text Information Retrieval. Computer Science Department, University of Massachusetts at Amherst, Technical Report 95-81, October 1995.
6. ^ Cutting, D., Pedersen, J.: Optimizations for dynamic inverted index maintenance. Proceedings of SIGIR, 405-411, 1990.
7. ^ Linear Hash Partitioning (<http://dev.mysql.com/doc/refman/5.1/en/partitioning-linear-hash.html>). MySQL 5.1 Reference Manual. Verified Dec 2006
8. ^ Gusfield, Dan [1997] (1999). *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. USA: Cambridge University Press. ISBN 0-521-58519-8. .
9. ^ trie (<http://www.nist.gov/dads/HTML/trie.html>), Dictionary of Algorithms and Data Structures (<http://www.nist.gov/dads>), U.S. National Institute of Standards and Technology (<http://www.nist.gov/>).
10. ^ Black, Paul E., inverted index (<http://www.nist.gov/dads/HTML/invertedIndex.html>), Dictionary of Algorithms and Data Structures (<http://www.nist.gov/dads>), U.S. National Institute of Standards and Technology (<http://www.nist.gov/>) Oct 2006. Verified Dec 2006.
11. ^ C. C. Foster, Information retrieval: information storage and retrieval using AVL trees, Proceedings of the 1965 20th national conference, p.192-205, August 24-26, 1965, Cleveland,

Ohio, United States

12. ^ Landauer, W. I.: The balanced tree and its utilization in information retrieval. IEEE Trans. on Electronic Computers, Vol. EC-12, No. 6, December 1963.
13. ^ Google Ngram Datasets (<http://www ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2006T13>) for sale at LDC (<http://www ldc.upenn.edu/>) Catalog
14. ^ Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. Google, Inc. OSDI. 2004.
15. ^ Grossman, Frieder, Goharian. IR Basics of Inverted Index (<http://www.eng.auburn.edu/~gilbert/Comp7120/Concept-50/IR-Basics-of-Inverted-Index.pdf>). 2002. Verified Dec 2006.
16. ^ Tang, Hunqiang. Dwarkadas, Sandhya. "Hybrid Global Local Indexing for Efficient Peer to Peer Information Retrieval". University of Rochester. Pg 1. <http://www.cs.rochester.edu/u/sarmor/publications/eSearch-NSDI.ps>
17. ^ [1] (<http://www.kimbly.com/code/invidx/haskell/InvIdx.lhs>) - inverted index written in Haskell
18. ^ [2] (<http://www.kimbly.com/code/invidx/lisp/invidx.cl>) - inverted index written in Lisp
19. ^ Tomasic, A., et al: Incremental Updates of Inverted Lists for Text Document Retrieval. Short Version of Stanford University Computer Science Technical Note STAN-CS-TN-93-1, December, 1993.
20. ^ Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine (<http://infolab.stanford.edu/~backrub/google.html>). Stanford University. 1998. Verified Dec 2006.
21. ^ H.S. Heaps. Storage analysis of a compression coding for a document database. INFOR, 10 (i):47-61, February 1972.
22. ^ Murray, Brian H. Sizing the Internet (http://www.cyveillance.com/web/downloads/Sizing_the_Internet.pdf). Cyveillance, Inc. Pg 2. July 2000. Verified Dec 2006.
23. ^ Blair Bancroft. Word Count:A Highly Personal-and Probably Controversial-Essay on Counting Words (http://www.blairbancroft.com/word_count.htm). Personal Website. Verified Dec 2006.
24. ^ The Unicode Standard - Frequently Asked Questions (http://www.unicode.org/faq/basic_q.html#15). Verified Dec 2006.
25. ^ Storage estimates (<http://www.uplink.freeuk.com/data.html>). Verified Dec 2006.
26. ^ PC Technology Guide: Guides/Storage/Hard Disks (<http://www.pctechguide.com/31HardDisk.htm>). PC Technology Guide. 2003. Verified Dec 2006.
27. ^ Berners-Lee, T., "Hypertext Markup Language - 2.0", RFC 1866 (<http://tools.ietf.org/html/rfc1866>), Network Working Group, November 1995.
28. ^ Krishna Nareddy. Indexing with Microsoft Index Server (<http://msdn2.microsoft.com/en-us/library/ms951558.aspx>). MSDN Library. Microsoft Corporation. January 30, 1998. Verified Dec 2006. Note that this is a commercial, external link.

Retrieved from "http://en.wikipedia.org/wiki/Index_%28search_engine%29"

Categories: Wikipedia articles needing copy edit from January 2007 | All articles needing copy edit | Information retrieval | Searching | Indexing

-
- This page was last modified 01:06, 30 June 2007.
 - All text is available under the terms of the GNU Free Documentation License. (See **Copyrights** for details.) Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a US-registered 501(c)(3) tax-deductible nonprofit charity.