

**College of Computer and Information Sciences**

**Computer Science Department**

**“Web Search”**

*CSC 361: Artificial Intelligence*

**Prepared by:**

Student 1 Name Student 1 ID

Student 2 Name Student 2 ID

Second Semester 2018/2019

# Abstract

In this paper, we present Google, a prototype of a large-scale search engine which makes heavy use of the structure present in hypertext. Google is designed to crawl and index the Web efficiently and produce much more satisfying search results than existing systems. The prototype with a full text and hyperlink database of at least 24 million pages is available at http://google.stanford.edu/ To engineer a search engine is a challenging task. Search engines index tens to hundreds of millions of web pages involving a comparable number of distinct terms. They answer tens of millions of queries every day. Despite the importance of large-scale search engines on the web, very little academic research has been done on them. Furthermore, due to rapid advance in technology and web proliferation, creating a web search engine today is very different from three years ago. This paper provides an in-depth description of our large-scale web search engine -- the first such detailed public description we know of to date. Apart from the problems of scaling traditional search techniques to data of this magnitude, there are new technical challenges involved with using the additional information present in hypertext to produce better search results. This paper addresses this question of how to build a practical large-scale system which can exploit the additional information present in hypertext. Also we look at the problem of how to effectively deal with uncontrolled hypertext collections where anyone can publish anything they want.

# Introduction

The web creates new challenges for information retrieval. The amount of information on the web is growing rapidly, as well as the number of new users inexperienced in the art of web research. People are likely to surf the web using its link graph, often starting with high quality human maintained indices such as Yahoo! or with search engines. Human maintained lists cover popular topics effectively but are subjective, expensive to build and maintain, slow to improve, and cannot cover all esoteric topics. Automated search engines that rely on keyword matching usually return too many low quality matches. To make matters worse, some advertisers attempt to gain people’s attention by taking measures meant to mislead automated search engines. We have built a large-scale search engine which addresses many of the problems of existing systems. It makes especially heavy use of the additional structure present in hypertext to provide much higher quality search results. We chose our system name, Google, because it is a common spelling of googol, or 10100 and fits well with our goal of building very large-scale search engines.

# Background

**2.1 Web Search Engines**

Search engine technology has had to scale dramatically to keep up with the growth of the web. In 1994, one of the first web search engines, the World Wide Web Worm (WWWW) [McBryan 94] had an index of 110,000 web pages and web accessible documents.

As of November, 1997, the top search engines claim to index from 2 million (WebCrawler) to 100 million web documents (from Search Engine Watch).

It is foreseeable that by the year 2000, a comprehensive index of the Web will contain over a billion documents.

At the same time, the number of queries search engines handle has grown incredibly too.

In March and April 1994, the World Wide Web Worm received an average of about 1500 queries per day.

In November 1997, Altavista claimed it handled roughly 20 million queries per day. With the increasing number of users on the web, and automated systems which query search engines, it is likely that top search engines will handle hundreds of millions of queries per day by the year 2000. The goal of our system is to address many of the problems, both in quality and scalability, introduced by scaling search engine technology to such extraordinary numbers.

**2.2. Google: Scaling with the Web**

Creating a search engine which scales even to today’s web presents many challenges.

Fast crawling technology is needed to gather the web documents and keep them up to date.

Storage space must be used efficiently to store indices and, optionally, the documents themselves.

The indexing system must process hundreds of gigabytes of data efficiently. Queries must be handled quickly, at a rate of hundreds to thousands per second.

These tasks are becoming increasingly difficult as the Web grows. However, hardware performance and cost have improved dramatically to partially offset the difficulty.

There are, however, several notable exceptions to this progress such as disk seek time and operating system robustness.

In designing Google, we have considered both the rate of growth of the Web and technological changes.

Google is designed to scale well to extremely large data sets. It makes efficient use of storage space to store the index.

Its data structures are optimized for fast and efficient access (see section 4.2). Further, we expect that the cost to index and store text or HTML will eventually decline relative to the amount that will be available (see Appendix B).

This will result in favorable scaling properties for centralized systems like Google.

**2.3 Design Goals**

**2.3.1 Improved Search Quality**

Our main goal is to improve the quality of web search engines.

In 1994, some people believed that a complete search index would make it possible to find anything easily.

According to Best of the Web 1994 -- Navigators, "The best navigation service should make it easy to find almost anything on the Web (once all the data is entered)." However, the Web of 1997 is quite different.

Anyone who has used a search engine recently, can readily testify that the completeness of the index is not the only factor in the quality of search results. "Junk results" often wash out any results that a user is interested in. In fact, as of November 1997, only one of the top four commercial search engines finds itself (returns its own search page in response to its name in the top ten results).

One of the main causes of this problem is that the number of documents in the indices has been increasing by many orders of magnitude, but the user’s ability to look at documents has not. People are still only willing to look at the first few tens of results.

Because of this, as the collection size grows, we need tools that have very high precision (number of relevant documents returned, say in the top tens of results). Indeed, we want our notion of "relevant" to only include the very best documents since there may be tens of thousands of slightly relevant documents.

This very high precision is important even at the expense of recall (the total number of relevant documents the system is able to return).

There is quite a bit of recent optimism that the use of more hypertextual information can help improve search and other applications [Marchiori 97] [Spertus 97] [Weiss 96] [Kleinberg 98].

In particular, link structure [Page 98] and link text provide a lot of information for making relevance judgments and quality filtering. Google makes use of both link structure and anchor text (see Sections 2.1 and 2.2).

**2.3.2 Academic Search Engine Research**

Aside from tremendous growth, the Web has also become increasingly commercial over time.

Another important design goal was to build systems that reasonable numbers of people can actually use.

Usage was important to us because we think some of the most interesting research will involve leveraging the vast amount of usage data that is available from modern web systems. For example, there are many tens of millions of searches performed every day.

Another goal we have is to set up a Spacelab-like environment where researchers or even students can propose and do interesting experiments on our large-scale web data.

# System Features

The Google search engine has two important features that help it produce high precision results.

First, it makes use of the link structure of the Web to calculate a quality ranking for each web page.

This ranking is called PageRank and is described in detail in [Page 98]. Second, Google utilizes link to improve search results.

**3.1 PageRank: Bringing Order to the Web**

The citation (link) graph of the web is an important resource that has largely gone unused in existing web search engines.

We have created maps containing as many as 518 million of these hyperlinks, a significant sample of the total.

These maps allow rapid calculation of a web page’s "PageRank", an objective measure of its citation importance that corresponds well with people’s subjective idea of importance. Because of this correspondence, PageRank is an excellent way to prioritize the results of web keyword searches.

For most popular subjects, a simple text matching search that is restricted to web page titles performs admirably when PageRank prioritizes the results (demo available at google.stanford.edu).

For the type of full text searches in the main Google system, PageRank also helps a great deal.

**3.1.1 Description of PageRank Calculation**

Academic citation literature has been applied to the web, largely by counting citations or backlinks to a given page.

This gives some approximation of a page’s importance or quality.

PageRank extends this idea by not counting links from all pages equally, and by normalizing by the number of links on a page.

PageRank is defined as follows: We assume page A has pages T1...Tn which point to it (i.e., are citations).

The parameter d is a damping factor which can be set between 0 and 1. We usually set d to 0.85. There are more details about d in the next section. Also C(A) is defined as the number of links going out of page A.

The PageRank of a page A is given as follows: PR(A) = (1-d) + d (PR(T1)/C(T1) + ... + PR(Tn)/C(Tn)) Note that the PageRanks form a probability distribution over web pages, so the sum of all web pages’ PageRanks will be one.

PageRank or PR(A) can be calculated using a simple iterative algorithm, and corresponds to the principal eigenvector of the normalized link matrix of the web. Also, a PageRank for 26 million web pages can be computed in a few hours on a medium size workstation.

There are many other details which are beyond the scope of this paper.

**3.1.2 Intuitive Justification**

PageRank can be thought of as a model of user behavior.

We assume there is a "random surfer" who is given a web page at random and keeps clicking on links, never hitting "back" but eventually gets bored and starts on another random page.

The probability that the random surfer visits a page is its PageRank. And, the d damping factor is the probability at each page the "random surfer" will get bored and request another random page.

One important variation is to only add the damping factor d to a single page, or a group of pages.

This allows for personalization and can make it nearly impossible to deliberately mislead the system in order to get a higher ranking.

We have several other extensions to PageRank, again see [Page 98]. Another intuitive justification is that a page can have a high PageRank if there are many pages that point to it, or if there are some pages that point to it and have a high PageRank. Intuitively, pages that are well cited from many places around the web are worth looking at.

**3.2 Anchor Text**

The text of links is treated in a special way in our search engine.

Most search engines associate the text of a link with the page that the link is on. In addition, we associate it with the page the link points to. This has several advantages.

First, anchors often provide more accurate descriptions of web pages than the pages themselves.

Second, anchors may exist for documents which cannot be indexed by a text-based search engine, such as images, programs, and databases.

This makes it possible to return web pages which have not actually been crawled. Note that pages that have not been crawled can cause problems, since they are never checked for validity before being returned to the user.

In this case, the search engine can even return a page that never actually existed, but had hyperlinks pointing to it. However, it is possible to sort the results, so that this particular problem rarely happens.

In our current crawl of 24 million pages, we had over 259 million anchors which we indexed.

**3.3 Other Features**

Aside from PageRank and the use of anchor text, Google has several other features. First, it has location information for all hits and so it makes extensive use of proximity in search. Second, Google keeps track of some visual presentation details such as font size of words.

Words in a larger or bolder font are weighted higher than other words. Third, full raw HTML of pages is available in a repository.

**4 Related Work**

Search research on the web has a short and concise history.

The World Wide Web Worm (WWWW) [McBryan 94] was one of the first web search engines.

It was subsequently followed by several other academic search engines, many of which are now public companies.

In the next two sections, we discuss some areas where this research needs to be extended to work better on the web.

* 1. **Information Retrieval**

Work in information retrieval systems goes back many years and is well developed [Witten 94].

However, most of the research on information retrieval systems is on small well controlled homogeneous collections such as collections of scientific papers or news stories on a related topic.

If a user issues a query like "Bill Clinton" they should get reasonable results since there is a enormous amount of high quality information available on this topic. Given examples like these, we believe that the standard information retrieval work needs to be extended to deal effectively with the web.

* 1. **Differences Between the Web and Well Controlled Collections**

The web is a vast collection of completely uncontrolled heterogeneous documents. Documents on the web have extreme variation internal to the documents, and also in the external meta information that might be available.

For example, documents differ internally in their language (both human and programming), vocabulary (email addresses, links, zip codes, phone numbers, product numbers), type or format (text, HTML, PDF, images, sounds), and may even be machine generated (log files or output from a database). On the other hand, we define external meta information as information that can be inferred about a document, but is not contained within it.

There are even numerous companies which specialize in manipulating search engines for profit.

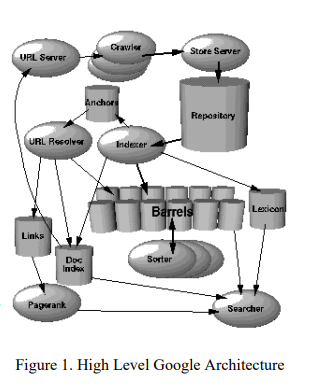
**5 System Anatomy**

First, we will provide a high level discussion of the architecture. Then, there is some in-depth descriptions of important data structures.

Finally, the major applications: crawling, indexing, and searching will be examined in depth.

**5.1 Google Architecture Overview**

In this section, we will give a high level overview of how the whole system works as pictured in Figure 1.



Further sections will discuss the applications and data structures not mentioned in this section. Most of Google is implemented in C or C++ for efficiency and can run in either Solaris or Linux.

In Google, the web crawling (downloading of web pages) is done by several distributed crawlers. There is a URLserver that sends lists of URLs to be fetched to the crawlers.

The web pages that are fetched are then sent to the storeserver. The storeserver then compresses and stores the web pages into a repository.

Every web page has an associated ID number called a docID which is assigned whenever a new URL is parsed out of a web page. The indexing function is performed by the indexer and the sorter.

The indexer performs a number of functions. It reads the repository, uncompresses the documents, and parses them.

Each document is converted into a set of word occurrences called hits. The hits record the word, position in document, an approximation of font size, and capitalization.

The indexer distributes these hits into a set of "barrels", creating a partially sorted forward index.

The indexer performs another important function. It parses out all the links in every web page and stores important information about them in an anchors file.

This file contains enough information to determine where each link points from and to, and the text of the link.

The URLresolver reads the anchors file and converts relative URLs into absolute URLs and in turn into docIDs.

It puts the anchor text into the forward index, associated with the docID that the anchor points to. It also generates a database of links which are pairs of docIDs.

The links database is used to compute PageRanks for all the documents. The sorter takes the barrels, which are sorted by docID and resorts them by wordID to generate the inverted index.

This is done in place so that little temporary space is needed for this operation.

The sorter also produces a list of wordIDs and offsets into the inverted index. A program called DumpLexicon takes this list together with the lexicon produced by the indexer and generates a new lexicon to be used by the searcher. The searcher is run by a web server and uses the lexicon built by DumpLexicon together with the inverted index and the PageRanks to answer queries.

**5.2 Major Data Structures**

Google’s data structures are optimized so that a large document collection can be crawled, indexed, and searched with little cost. Although, CPUs and bulk input output rates have improved dramatically over the years, a disk seek still requires about 10 ms to complete. Google is designed to avoid disk seeks whenever possible, and this has had a considerable influence on the design of the data structures.

**5.2.1 BigFiles**

BigFiles are virtual files spanning multiple file systems and are addressable by 64 bit integers.

The allocation among multiple file systems is handled automatically. The BigFiles package also handles allocation and deallocation of file descriptors, since the operating systems do not provide enough for our needs.

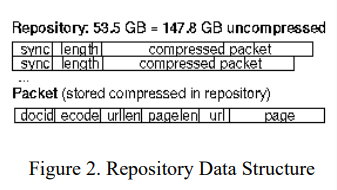
BigFiles also support rudimentary compression options.

**5.2.2 Repository**

The repository contains the full HTML of every web page. Each page is compressed using zlib (see RFC1950).

The choice of compression technique is a tradeoff between speed and compression ratio. We chose zlib’s speed over a significant improvement in compression offered by bzip. The compression rate of bzip was approximately 4 to 1 on the repository as compared to zlib’s 3 to 1 compression. In the repository, the documents are stored one after the other and are prefixed by docID, length, and URL as can be seen in Figure 2.

The repository requires no other data structures to be used in order to access it. This helps with data consistency and makes development much easier; we can rebuild all the other data structures from only the repository and a file which lists crawler errors.



**5.2.3 Document Index**

The document index keeps information about each document. It is a fixed width ISAM (Index sequential access mode) index, ordered by docID.

The information stored in each entry includes the current document status, a pointer into the repository, a document checksum, and various statistics.

If the document has been crawled, it also contains a pointer into a variable width file called docinfo which contains its URL and title. Otherwise the pointer points into the URLlist which contains just the URL.

This design decision was driven by the desire to have a reasonably compact data structure, and the ability to fetch a record in one disk seek during a search Additionally, there is a file which is used to convert URLs into docIDs. It is a list of URL checksums with their corresponding docIDs and is sorted by checksum.

In order to find the docID of a particular URL, the URL’s checksum is computed and a binary search is performed on the checksums file to find its docID. URLs may be converted into docIDs in batch by doing a merge with this file.

This is the technique the URLresolver uses to turn URLs into docIDs. This batch mode of update is crucial because otherwise we must perform one seek for every link which assuming one disk would take more than a month for our 322 million link dataset.

**5.2.4 Lexicon**

The lexicon has several different forms. One important change from earlier systems is that the lexicon can fit in memory for a reasonable price.

In the current implementation we can keep the lexicon in memory on a machine with 256 MB of main memory.

For various functions, Figure 2. Repository Data Structure the list of words has some auxiliary information which is beyond the scope of this paper to explain fully.

**5.2.5 Hit Lists**

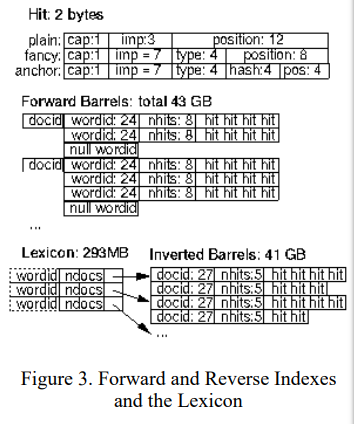
A hit list corresponds to a list of occurrences of a particular word in a particular document including position, font, and capitalization information.

Hit lists account for most of the space used in both the forward and the inverted indices.

Because of this, it is important to represent them as efficiently as possible.

We considered several alternatives for encoding position, font, and capitalization -- simple encoding (a triple of integers), a compact encoding (a hand optimized allocation of bits), and Huffman coding.

In the end we chose a hand optimized compact encoding since it required far less space than the simple encoding and far less bit manipulation than Huffman coding. The details of the hits are shown in Figure 3.



**5.2.6 Forward Index**

The forward index is actually already partially sorted. It is stored in a number of barrels (we used 64). Each barrel holds a range of wordID’s.

If a document contains words that fall into a particular barrel, the docID is recorded into the barrel, followed by a list of wordID’s with hitlists which correspond to those words. This scheme requires slightly more storage because of duplicated docIDs but the difference is very small for a reasonable number of buckets and saves considerable time and coding complexity in the final indexing phase done by the sorter.

**5.2.7 Inverted Index**

The inverted index consists of the same barrels as the forward index, except that they have been processed by the sorter. For every valid wordID, the lexicon contains a pointer into the barrel that wordID falls into. It points to a doclist of docID’s together with their corresponding hit lists.

This doclist represents all the occurrences of that word in all documents.

**5.3 Crawling the Web**

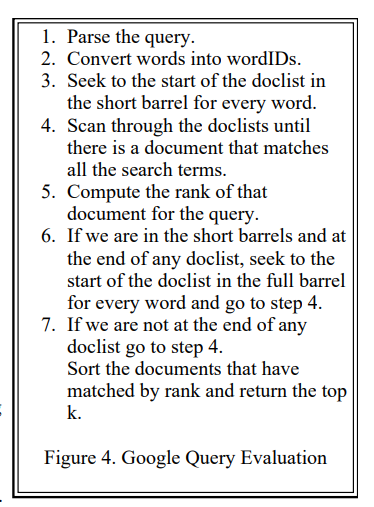
Running a web crawler is a challenging task.

There are tricky performance and reliability issues and even more importantly, there are social issues.

Crawling is the most fragile application since it involves interacting with hundreds of thousands of web servers and various name servers which are all beyond the control of the system.

* 1. **Indexing the Web**
* **Parsing** -- Any parser which is designed to run on the entire Web must handle a huge array of possible errors. These range from typos in HTML tags to kilobytes of zeros in the middle of a tag, non-ASCII characters, HTML tags nested hundreds deep, and a great variety of other errors that challenge anyone’s imagination to come up with equally creative ones. For maximum speed, instead of using YACC to generate a CFG parser, we use flex to generate a lexical analyzer which we outfit with its own stack. Developing this parser which runs at a reasonable speed and is very robust involved a fair amount of work.
* **Indexing Documents into Barrels** -- After each document is parsed, it is encoded into a number of barrels. Every word is converted into a wordID by using an in-memory hash table -- the lexicon. New additions to the lexicon hash table are logged to a file. Once the words are converted into wordID’s, their occurrences in the current document are translated into hit lists and are written into the forward barrels.
* **Sorting** -- In order to generate the inverted index, the sorter takes each of the forward barrels and sorts it by wordID to produce an inverted barrel for title and anchor hits and a full text inverted barrel.
  1. **Searching**

The goal of searching is to provide quality search results efficiently. Many of the large commercial search engines seemed to have made great progress in terms of efficiency. Therefore, we have focused more on quality of search in our research, although we believe our solutions are scalable to commercial volumes with a bit more effort. The google query evaluation process is show in Figure 4.



* + 1. **The Ranking System**

Google maintains much more information about web documents than typical search engines. Every hitlist includes position, font, and capitalization information. Additionally, we factor in hits from anchor text and the PageRank of the document. Combining all of this information into a rank is difficult.

We designed our ranking function so that no particular factor can have too much influence.

First, consider the simplest case -- a single word query. In order to rank a document with a single word query, Google looks at that document’s hit list for that word. Google considers each hit to be one of several different types (title, anchor, URL, plain text large font, plain text small font, ...), each of which has its own type-weight. The type-weights make up a vector indexed by type.

Google counts the number of hits of each type in the hit list. Then every count is converted into a count-weight. Count-weights increase linearly with counts at first but quickly taper off so that more than a certain count will not help.

We take the dot product of the vector of count-weights with the vector of type-weights to compute an IR score for the document.

Finally, the IR score is combined with PageRank to give a final rank to the document.

* + 1. **Feedback**

The ranking function has many parameters like the type-weights and the type-prox-weights.

Figuring out the right values for these parameters is something of a black art. In order to do this, we have a user feedback mechanism in the search engine.

A trusted user may optionally evaluate all of the results that are returned. This feedback is saved.

Then when we modify the ranking function, we can see the impact of this change on all previous searches which were ranked.

Although far from perfect, this gives us some idea of how a change in the ranking function affects the search results.

1. **Conclusions**

Google is designed to be a scalable search engine.

The primary goal is to provide high quality search results over a rapidly growing World Wide Web. Google employs a number of techniques to improve search quality including page rank, anchor text, and proximity information.

Furthermore, Google is a complete architecture for gathering web pages, indexing them, and performing search queries over them.

**6.1 Future Work**

A large-scale web search engine is a complex system and much remains to be done. Our immediate goals are to improve search efficiency and to scale to approximately 100 million web pages. Some simple improvements to efficiency include query caching, smart disk allocation, and subindices.

We have far too many to list here so we do not expect this Future Work section to become much shorter in the near future.

**6.2 High Quality Search**

The biggest problem facing users of web search engines today is the quality of the results they get back. While the results are often amusing and expand users’ horizons, they are often frustrating and consume precious time.

The analysis of link structure via PageRank allows Google to evaluate the quality of web pages. The use of link text as a description of what the link points to helps the search engine return relevant (and to some degree high quality) results.

Finally, the use of proximity information helps increase relevance a great deal for many queries.

**6.3 Scalable Architecture**

Aside from the quality of search, Google is designed to scale. It must be efficient in both space and time, and constant factors are very important when dealing with the entire Web.

In implementing Google, we have seen bottlenecks in CPU, memory access, memory capacity, disk seeks, disk throughput, disk capacity, and network IO. Google has evolved to overcome a number of these bottlenecks during various operations. Google’s major data structures make efficient use of available storage space. Furthermore, the crawling, indexing, and sorting operations are efficient enough to be able to build an index of a substantial portion of the web -- 24 million pages, in less than one week. We expect to be able to build an index of 100 million pages in less than a month.

**6.4 A Research Tool**

In addition to being a high quality search engine, Google is a research tool. The data Google has collected has already resulted in many other papers submitted to conferences and many more on the way.

Recent research such as [Abiteboul 97] has shown a number of limitations to queries about the Web that may be answered without having the Web available locally. This means that Google (or a similar system) is not only a valuable research tool but a necessary one for a wide range of applications. We hope Google will be a resource for searchers and researchers all around the world and will spark the next generation of search engine technology.

**References**

* Best of the Web 1994 -- Navigators <http://botw.org/1994/awards/navigators.html>
* Bill Clinton Joke of the Day: April 14, 1997. <http://www.io.com/~cjburke/clinton/970414.html>.
* Bzip2 Homepage <http://www.muraroa.demon.co.uk/>
* Google Search Engine <http://google.stanford.edu/>
* Harvest <http://harvest.transarc.com/>
* Mauldin, Michael L. Lycos Design Choices in an Internet Search Service, IEEE Expert Interview <http://www.computer.org/pubs/expert/1997/trends/x1008/mauldin.htm>
* The Effect of Cellular Phone Use Upon Driver Attention <http://www.webfirst.com/aaa/text/cell/cell0toc.htm>
* Search Engine Watch <http://www.searchenginewatch.com/>
* RFC 1950 (zlib) <ftp://ftp.uu.net/graphics/png/documents/zlib/zdoc-index.html>
* Robots Exclusion Protocol: <http://info.webcrawler.com/mak/projects/robots/exclusion.htm>
* Web Growth Summary: <http://www.mit.edu/people/mkgray/net/web-growth-summary.html>
* Yahoo! <http://www.yahoo.com/>
* [Abiteboul 97] Serge Abiteboul and Victor Vianu, Queries and Computation on the Web. Proceedings of the International Conference on Database Theory. Delphi, Greece 1997.
* [Bagdikian 97] Ben H. Bagdikian. The Media Monopoly. 5th Edition. Publisher: Beacon, ISBN: 0807061557
* [Cho 98] Junghoo Cho, Hector Garcia-Molina, Lawrence Page. Efficient Crawling Through URL Ordering. Seventh International Web Conference (WWW 98). Brisbane, Australia, April 14-18, 1998.
* [Gravano 94] Luis Gravano, Hector Garcia-Molina, and A. Tomasic. The Effectiveness of GlOSS for the Text-Database Discovery Problem. Proc. of the 1994 ACM SIGMOD International Conference On Management Of Data, 1994.
* [Kleinberg 98] Jon Kleinberg, Authoritative Sources in a Hyperlinked Environment, Proc. ACM-SIAM Symposium on Discrete Algorithms, 1998.
* [Marchiori 97] Massimo Marchiori. The Quest for Correct Information on the Web: Hyper Search Engines. The Sixth International WWW Conference (WWW 97). Santa Clara, USA, April 7-11, 1997.
* [McBryan 94] Oliver A. McBryan. GENVL and WWWW: Tools for Taming the Web. First International Conference on the World Wide Web. CERN, Geneva (Switzerland), May 25-26-27 1994. <http://www.cs.colorado.edu/home/mcbryan/mypapers/www94.ps>

**Tasks division**

|  |  |  |
| --- | --- | --- |
| Name | ID | Tasks |
| Student 1 Name | Student 1 ID |  |
| Student 2 Name | Student 2 ID |  |