

# Let's Search

## Scenario

MustangWiki's search engine is broken! MustangWiki is a collaborative, open place where students, faculty, staff and the community can create online, freely-accessible content. Please help MustangWiki get its search back by implementing a fast search engine for all of the books in the collection.

For this project, you'll be using a dump of the WikiBooks (<http://en.wikibooks.org>) as the basis for your search engine. Wiki Books contains approximately 161,000 entries and is marked up in XML.

## Search Engine Architecture

Search engines are designed to allow users to quickly locate the data they want. Input to a search engine is a set of documents and commonly referred to as a **corpus**. Typically, the user will enter a search predicate, and any documents that satisfy that predicate are returned. Another task of a search engine is to rank the results such that those that are identified to be more relevant are “higher” in the search results.

There are four major components<sup>1</sup> of a general search engine:

1. Document parser/processor,
2. Query processor,
3. Search processor, and
4. Ranking processor.

Figure 1 provides a general overview of a potential system architecture:

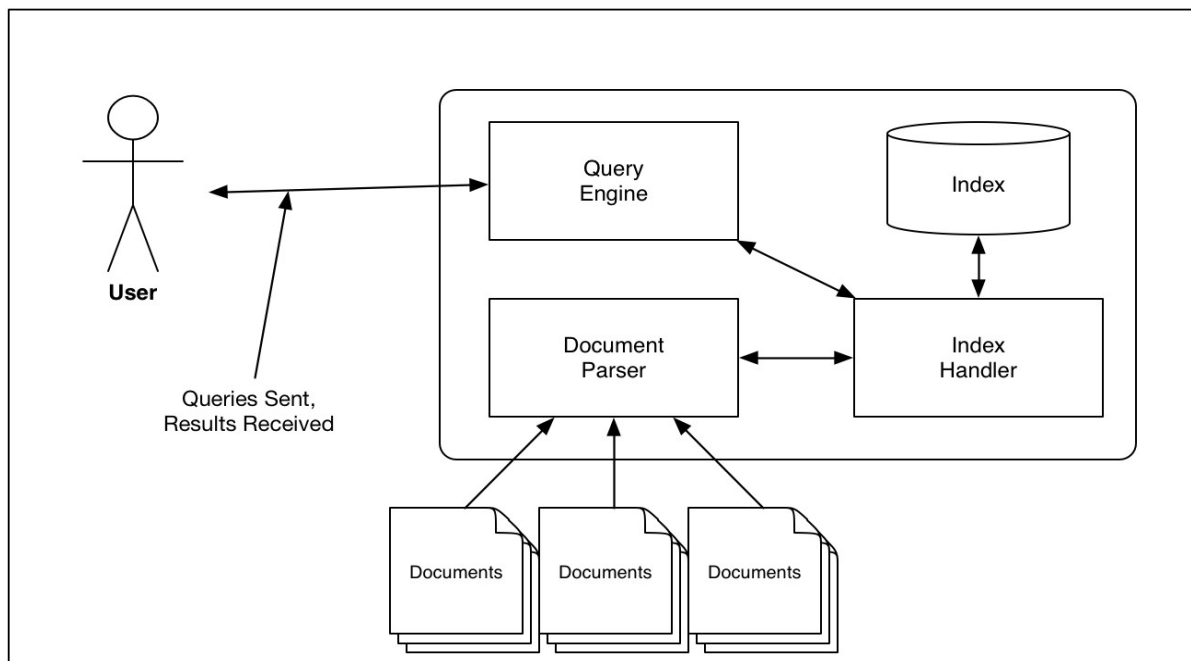


Figure 1 - Sample Search Engine System Architecture

<sup>1</sup> <http://www.infoday.com/searcher/may01/liddy.htm>

The basic document for this project is a page from WikiBooks. The dataset you will be using contains about 160,000 documents. The documents set is marked up with XML(<http://en.wikipedia.org/wiki/XML>). You'll be considering `<page>...</page>` as one document as you generate your search engine.

Below is an overview of the major tasks/responsibilities of each of the components of the search engine.

The **document parser** is responsible for the following tasks:

- *Processing each document in the document corpus.* The fact that the documents are marked up in XML allows you to use an XML parser to help parse the documents. However, you can brute-force parse the XML docs if you'd like. Note that the XML document contains other interesting information besides just the text of the document that you may want to include in you index and/or use for ranking.
- *Removing stop words from the documents.* Stop words are common words that appear in text but that provide little information in a search because they are so common. Example stop words include “a”, “the”, and “if”. One possible list of stop words to use for this project can be found at <http://www.webconfs.com/stop-words.php>. You may use other stop word lists you find online.
- *Stemming the remaining words.* Stemming<sup>2</sup> refers to removing certain endings of words in the English language. For instance, the stemmed version of “running” may be “run”. For this project, you may make use of any previously implemented stemming algorithm that you can find online. One such algorithm is the Porter Stemming algorithm. More information as well as implementations can be found at <http://tartarus.org/~martin/PorterStemmer/>. Another option is <http://www.oleandersolutions.com/stemming/stemming.html>. You may use others.
- *Computing/maintaining term frequencies.* Term frequency is how often a word appears in the document as well as in the overall corpus. This will be used in relevance ranking of documents returned from a search.

The **index handler** is responsible for:

- *Reading and writing to/from the main index.* You'll be creating an inverted file index which stores references from each element to be indexed to the corresponding document(s) in which those elements exist.
- *Searching the inverted file index based based on a request from the query processor.*
- *Storing other data with each word.*

This is the “workhorse” component of the search engine.

The **query processor** is responsible for:

- *Parsing of queries entered by the user of the search engine.* For this project, you'll implement functionality to handle **simple** prefix Boolean queries entered by the user. The Boolean expression will be prefixed with a Boolean operator of either AND or OR if there more than one word is of interest. Trailing search terms may be preceded with NOT to indicate documents including that term should be removed from the result set. For simple one-term searches, an operator is not required. Here are some examples:
  - **Boston**
    - This query should return all documents that contain the word Boston.
  - **AND programming computer**
    - This query should return all documents that contain the words programming **and** computer
  - **OR Boston Seattle**

---

<sup>2</sup> See <http://www.comp.lancs.ac.uk/computing/research/stemming/general/> for more information.

- This query should return all documents that contain either Boston **OR** Seattle **OR** both.
- AND book Boston NOT Seattle
  - This query should return all documents that contain book and Boston, but not Seattle.
- Boston NOT Seattle
  - This query should return all document that contain Boston, but not Seattle.

The **user interface** is responsible for:

- Receiving queries from the user
- Communicating with the Search Engine
- Formatting and displaying results in an organized, logical fashion

More info on the UI later.

## The Index Structure

---

The **inverted file index**<sup>3</sup> is a data structure that relates each unique word to the document(s) in which it appears. It allows the process which is executing a query to quickly determine in which documents a particular query term appears. For instance, let's assume we have the following documents with ascribed contents:

- d1 = Computer network security
- d2 = network cryptography
- d3 = database security

The inverted file index for these documents would contain, at a very minimum, the following:

- computer = d1
- network = d1, d2
- security = d1, d3
- cryptography = d2
- database = d3

The query “AND computer security” would find the intersection of the documents that contained *computer* and the documents that contained *security*.

- computer = d1
- security = d1, d3
- computer AND security = d1

**Relevancy ranking** refers to organizing the results so that “more relevant” documents are higher in the result set than less relevant documents. The difficulty here is determining what the concept of “more relevant” means. One way of handling relevancy is by using a basic **term frequency – inverse document frequency** (tf/idf) statistic<sup>4</sup>. tf/idf is used to determine how important a particular word is to a document which is from a collection (corpus) of documents. If a word appears frequently in document  $d_t$  but infrequently in other documents, then document  $d_t$  would be ranked higher than another document  $d_s$  in which a query term appears frequently, but it also appears frequently in other documents as well. You may review <http://en.wikipedia.org/wiki/Tf-idf> and <http://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html> to understand more about how to perform the calculations.

---

<sup>3</sup> See [http://en.wikipedia.org/wiki/Inverted\\_index](http://en.wikipedia.org/wiki/Inverted_index) for more information.

<sup>4</sup> <http://en.wikipedia.org/wiki/Tf-idf>

## ***Inverted File Index Implementation Details***

The heart of this project is the **inverted file index**. Notice from the example above that what is being stored is essentially words and a list of documents in which each appears with perhaps some other book keeping information. Once the index is created, retrieving the set of documents for various words will be the central task of the index. Therefore, we should use a data structure that will allow for efficient searching.

You will implement at least two different data structures to store the index.

- AVL Tree
- Hash Table with collisions handled by separate chaining

You may also include a list implementation of the index. This will provide a good starting place.

In your implementation you should strive to abstract the index data structure idea from the underlying storage data structure implementation. What this means is that the AVL tree and hash table implementation should share a common interface. In Figure 2, IndexInterface may contain abstract methods such as addWord(), getDocsForWord(word:string), etc.

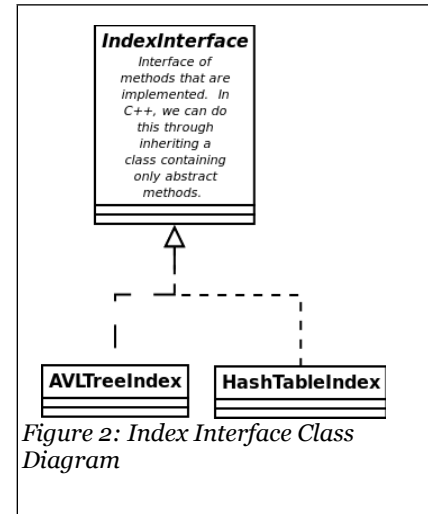


Figure 2: Index Interface Class Diagram

Each class that inherits from IndexInterface would be required to implement those methods. So that would mean that we could do something like:

```
IndexInterface* if = new AVLTreeIndex;
```

or

```
IndexInterface*if = new HashTableIndex;
```

## ***Index Persistence***

The index must also be persistent once it is created. This means that the contents of the index should be written to disk when the program ends and read when the program starts. The user should have the option of clearing the index and starting over.

## ***User Interface***

The user interface of the application should provide the following options:

- The program should have **three modes** (controlled by command line parameters)
  - maintenance mode –
    - allows the user to add documents to the index by supplying the path to a new file containing properly marked-up documents
    - allows the user to clear the index completely
  - interactive mode –
    - allow the user to indicate if they want the index loaded into an AVL structure or a hash table structure.
    - allow the user to enter a properly formatted Boolean query (as described above).

- The results should display the article/page title, contributor's username, date associated with the article, and the TF/IDF value (or other relevancy ranking statistic). The result set shown to the user need not contain any more than 15 articles. You may paginate all postings if you wish.
- The user should be allowed to choose one of the pages from the result set and have its contents displayed.
- Note that the query terms should have stop words removed and stemmed before querying the index.
- stress test mode –
  - allow the user to supply a “command file” based on a set of commands that you develop. The output of this mode will be the amount of time it takes to complete each of the commands in the command file.
  - stress test mode should command options should be able to access all functionality from maintenance mode and interactive mode.
  - the set of commands should be documented in the users manual.
- Any other options you deem appropriate and useful.

## ***Document Data Set***

---

We will be using a complete export of the WikiBooks website. The original data set can be found here: <http://dumps.wikimedia.org/enwikibooks/20141026/>. The main file that you'll be indexing is named enwikibooks-20141026-pages-meta-current.xml.bz2 and can be found under the section titled “All pages, current versions only.” The file contains roughly 171,000 individual pages. There are other export files on the web page above that you may find useful in your project.

## ***Mechanics of Implementation***

---

Some things to note:

- This project may be done individually, in teams of two students, or in teams of three students.
  - Individually: Finish all work on your own.
  - Team of 2 students:
    - Each team member must contribute to both the design AND implementation of the project.
    - Each class in the design must have an “owner”. The owner is a group member that is principally responsible for its design, implementation and integration into the overall project.
  - Team of 3 students:
    - Complete all work for this project and, additionally, the following:
      - implement simple date range queries as part of the query processor. Notice that each article in the corpus has a date stamp associated with it. You will use these dates and allow the user to constrain by date-range the documents that are returned to the user. Here are the additional operators you need to handle:
        - Operator DATEGT mm/dd/yyyy
          - The date on the posting is greater than the operand to DATEGT
          - Example:
            - **AND** computer programmer **DATEGT** 03/12/1987
        - Operator DATELT mm/dd/yyyy
          - The date on the posting is less than the operand to DATELT
          - Example:
            - **AND** Boston Seattle **DATELT** 03/12/1987

- The above two operands can be combined to provide a date range as in:
    - **AND** Boston Seattle **DATEGT** 03/12/1987 **DATELT** 06/14/1987
  - You are not required to do any pre-processing on the dates to make sure they are viable. For example if DATEGT operand is chronologically after DATELT operand, you may simply return an empty result set.
- implement searching for articles/pages by a particular user:
  - Operator USERNAME username
    - The results should be limited to articles by a particular user
- This project must be implemented using an object-oriented design methodology.
- You are free to use as much of the C++ standard library as you would like. In fact, I encourage you to make generous use of it. You may use other libraries as well.
- You may use implementations of other data structures from other sources (i.e. AVL Trees, etc.). Be sure to reference any code you use from other sources in the comments when appropriate.
  - Note that just because a website says that some code is the implementation of a particular data structure doesn't necessarily mean it is a good implementation or that it is in fact the data structure it claims to be. Review the code and understand it before using it.
- All of your code must be properly documented and formatted
  - Each class should be separated into interface and implementation (.h and .cpp) files unless templated.
  - Each file should have appropriate header comments to include owner of the class and a history of updates/modifications to the class

## ***Submission Schedule***

---

You must submit the following:

- **Design Documents: Due Monday November 17, 2014 at 11pm**
  - Submit to Blackboard a document containing the following (one submission per team):
    - Team member(s)
    - Class diagram indicating
      - Interface for each class
      - Role and responsibilities of each class
      - Owner for each class (if done in groups)
    - Initial set of commands you expect to implement for the “stress test” command mode.
- **Implementation Milestone 1: Due In Lab – Week of 11/17.**
  - Demonstrate to your TA that you've made substantial progress on the document processor and one of the index data structures.
- **Implementation Milestone 2: Due In Lab – Week of 12/1.**
  - Demonstrate that you've made adequate progress on query parser and have almost completed the indexer.
- **Final Project: Due Monday December 8, 2014 @ 6:00am (no extensions!)**
  - Complete project with full user interface
    - Your goal for parsing the entire wikibooks export file is 2 minutes. You should be able to implement parsing that can rip through all 171000 pages in 2 minutes or under.
  - Users Manual to include information on management piece as well as regular user piece
  - Documentation
    - updated UML diagrams

- documentation about each class in your project. Consider using Doxygen<sup>5</sup> for this.
- Report that compares the underlying functionality of the AVL implementation vs the Hash Table implementation. Use the stress test mode.
  - Which one is better?
  - How do you know?
  - Can you quantify this?
  - Is one better for small data sets compared to large data sets?
- Demonstration of project to Professor Fontenot and TAs on afternoon of Sunday December 8.

## ***Thoughts and Suggestions***

---

- If you wait even 1 week to start this project, you will very likely not finish.
- A significant portion of your grade will come from your demonstration of the project to Prof. Fontenot and the TAs. Be ready for this.
- Take an hour to read about the various parts of the C++ STL, particularly the container classes. They can help you immensely in the project.
- As mentioned previously, beware of code that you find on the Internet. It isn't always as good as it may seem upon initial inspection. Make sure that any code you use in the project is cited/referenced in the header comments of the project.
- Don't take the week of Thanksgiving off from the project. Work on it every day. Make use of a variety of options to communicate with your team. You can have virtual meetings through Google Hangout or use GroupMe to keep in touch with text messages.
- Take the large Wikibooks export file and examine it. Data is rarely beautiful and nicely formatted. Use the file to extract some sample test files of various sizes (10 pages, 100 pages, 1000 pages) to use during testing. Don't start out trying to index 171,000 pages.

---

<sup>5</sup> See <http://www.stack.nl/~dimitri/doxygen/> for more information.

**Grading:**

This project is worth ***25 percent of your final grade*** in this course (all other implementation projects are worth 30 percent of your final grade).

	<b><i>Points possible</i></b>	<b><i>Points awarded</i></b>
Early Design Documents	10	
Milestone 1	10	
Milestone 2	10	
<b><u>Completed Project</u></b>		
Completed Project	50	
Documentation	25	
Demonstration	25	



Sample “page” from the wikibooks xml file:

```
<page>
  <title>Talk:Scratch</title>
  <ns>1</ns>
  <id>123374</id>
  <revision>
    <id>877966</id>
    <parentid>877964</parentid>
    <timestamp>2007-05-27T15:22:13Z</timestamp>
    <contributor>
      <username>Robert Horning</username>
      <id>1227</id>
    </contributor>
    <minor />
    <comment>/* Central Discussion */</comment>
    <text xml:space="preserve">== Keep it simple ==
```

Please keep this page clean and simple. This is the “front page” that has links to other parts of this Wikibook, and should only have major sections listed. Individual chapters/modules and other parts of this book ought to be listed in the table of contents and as needed in other modules.

If there are ideas on how to improve this page, please feel free to add comments and suggestions below. --[[User:Robert Horning|Rob Horning]] 23:45, 26 May 2007 (UTC)

== Central Discussion ==

If you are making a comment about the general organization of this Wikibook, I would encourage you to use [[Scratch/Planning]] instead of this page for those discussion. This talk page should be limited to the appearance of this “title page” and any general organization thoughts to help improve this first impression people may have about this book. --[[User:Robert Horning|Rob Horning]] 15:21, 27 May 2007 (UTC)</text>

```
  <sha1>m7955bdj4askeelqqoriyeuv6sgw9lh</sha1>
  <model>wikitext</model>
  <format>text/x-wiki</format>
</revision>
</page>
```