Mustang Wiki Search Engine User Manual

1 Introduction

MustangWiki's Search Engine provides a basic, yet user-friendly environment for the maintenance, testing, and searching of WikiBooks. The interface allows simple boolean queries of the underlying data within the WikiBooks archive.

2 User Interface

2.1 Maintenance

```
MAINTENANCE MODE
1: Add Documents
2: Clear Index(added docs will need to be re-added)
3: Go Back
```

2.1.1 Add Documents: allows the user to supply an .xml file to be added to current index.

2.1.2 Clear Index: allows the user to completely clear the index, all data will be permanently removed from underlying data structures.

2.2 Interactive Search

```
INTERACTIVE SEARCH
1: Search Hash Map
2: Search AVL Tree
3: Go Back
```

2.2.1 Search Hash Map: simple boolean queries (AND, OR, NOT) on the underlying hash map data structure

2.2.2 Search AVL Tree: simple boolean queries (AND, OR, NOT) on the underlying AVL tree data structure

2.3 Stress Testing

```
STRESS TEST
1: Clear and Load Index
2: Iterative Search Time
3: Time to Index
4: Go Back
```

2.3.1 Clear and Load Index: Allows the user to supply a .txt file formatted the following way: the first line the number of clear/loads, followed by the .xml file locations to be loaded and cleared. (Example file format below.)

example.txt

```
5
enwikibooks_1.xml
enwikibooks_2.xml
enwikibooks_3.xml
enwikibooks_4.xml
enwikibooks_5.xml
```

2.3.2 Iterative Search Time: Allows the user to supply a .txt formatted the following way: the first line the number of searches to execute, followed by those searches. This mode reports the time for every search iteration and the time for all searches to be completed. (Example file format below.)

example.txt

```
5
AND help time
AND firefighter plan
OR chemistry math
science
mathematics
wikibooks NOT help
```

2.3.3 Time to Index: Allows the user to perform more in-depth analysis of loading each index. The user supplies a single .xml file properly formatted and the time to index for each structure is displayed.

3 Analysis of Data Structures: Using Stress Test Mode

3.1 Introduction

In grand scheme of maximizing efficiency, several different implementations of varying data structures come to mind. However as the size and complexity of the data set reaches a threshold, a few obvious solutions surface. Hashes, maps and keys all offer a fast and efficient way of storing and accessing large data.

3.2 Testing

Of the thousands of implementations out there, a dense hash map implemented by Google and an AVL tree implemented by Amanjit Gill, a software developer at PSI AG, were chosen. They both offered their own hashing functions and a fully implemented data structure. By utilizing the stress test mode implemented within the search engine, some valuable data and conclusions can be drawn on the efficiency of these two data structures.

3.3 Data

To test the time it takes to index a parsed .xml file, the Time to Index method and 4 different data sets were used: small.xml(~1.3mb), medium.xml(~7.5mb), large.xml(~24mb), enwikibooks_main.xml(~718.5mb).

Indexing Time		
Data Set	**Hash Map Time** (ms)	**AVL Tree Time** (ms)
small.xml(~1.3mb)	555	265
medium.xml(~7.5mb)	2998	1364
large.xml(~24mb)	9402	4327
enwikibooks_main.xml(~718.5mb)	352909	153866

Next to test query and retrieval time, the four data sets above, the Iterative Search Time method, and an iterativeSearch.txt command file were used.

iterativeSearch.txt

```
12
AND help time
AND firefighter plan
AND data set programming
OR chemistry math science
OR bird crow hawk
OR chronic disease
mathematics
temper
judical
wikibooks NOT help
AND time calendar NOT birthday
OR suit tie jacket NOT wedding
```

Searching Time (12 Boolean Searches)		
Data Set	**Hash Map Time** (ms)	**AVL Tree Time** (ms)
small.xml(~1.3mb)	< 1	< 1
medium.xml(~7.5mb)	1	2
large.xml(~24mb)	3	5
enwikibooks_main.xml(~718.5mb)	473	447

3.4 Conclusion

After reviewing the data, the initial assumption that the hash map structure would be faster for indexing contradicted the results. This can be attributed to the underlying implementation of the structure from the source from which it was obtained. As far as indexing from parsed .xml to functional data structure, the AVL tree was almost twice as fast compared to the hash map. When it comes to query and retrieval, the AVL tree was still the fastest but only marginally. After coming to these conclusions, the AVL tree implementation is the obvious winner. Not only did it excel in indexing but searching as well. In future development of this search engine, this critical fact will be taken into account.