Documentation for TinySearchEngine: Query

Chua Zheng Fu Edrei

Version 1.0: last update: 15 August 2015

I Introduction

We will discuss the Requirement Specifications and the command line execution of the query engine.

I.I Requirement Specification

Query is designed with the following requirement specifications:

The query engine should perform the following operations

1. It should load a previously generated index from the file system
2. It should receive user queries from input;
3. It should convert capital letters of words of the query to lower case (i.e., we do not distinguish between dog and Dog;
4. For each query, it should check the index and retrieve the results matching the query;
5. It should rank the list of results, returning the document with the highest number of matches to the query string, then the second highest, and so on; and
6. It should display the results to the user.

I.II Command line execution

The design takes the following 2 options, described in the listing below

Listing 1: Command line option

```
/* command line option*/

./query [INDEX FILE] [HTML DIR]

 // Example command input:
 // ./query ../indexer/index.dat ../data
```

$[INDEXFILE]$ is the file name of the file with the inverted index and $[HTMLDIR]$ is the directory with all the html web pages.

II Design Specification

In the design specification, we will talk about the Top level module (which includes the input, output and data flow specifications), the data structure that can be used and the pseudocode for query.

II.I Top level module

In the following Design Module we describe the input, data flow, and output specification for the query module.

Now, lets look at how we define the inputs, outputs and the major data flow through the module:

Input

./query [INDEX FILE] [HTML DIR]
// Example command input: // ./query ../indexer/index.dat ../data
[$INDEXFILE$] ../indexer/index.dat
Requirements: The inverted index file must exist and be in the same format as described in the Indexer specification. The below is an example to illustrate the format in which the inverted index file should take:

cat 2 3 4 7 6
moose 1 5 7...

Let us consider the entry cat 2 2 3 4 5 . cat is the word, the number 2 is the number of documents containing the word cat , the following 3 4 means the document with identifier 3 has 4 occurrences of cat in it; the following 7 6 means that the document with identifier 7 has 6 occurrences of cat in it.

Usage: The query engine needs to inform the user if the file cannot be found, is empy, cannot be opened or is not in the correct format

[$HTMLDIR$] ../data
Requirements: The directory should exist and the html files in the directory should be in the format specified in crawler i.e. the first line is the URL address, the second line is the depth and the thir line marks the beginning of the html file.
Usage: The query engine needs to inform the user if the directory does not exist of if the html file is not in the correct format

User Input

The query engine should prompt for user input and generate the result repeatedly until the user hits CTRL-D or CTRL-C i.e. there should be a while loop.

The queries have the following structure:

Listing 2: Structure of queries

```
word

word1 AND word2

word1 OR word2

word1 word2 //(equivalent to "word1 AND word2")

word1 AND word2 OR word 3 // (equivalent to "(word1 AND word2) OR word3")
```

A space is assumed to be an AND (e.g., ”village vanguard”).

Just like in C, AND has higher precedence than OR.

The ranking module should use word frequency in an HTML page to rank the results. Lets suppose that the query is dog cat. The context score is defined as the sum of the occurrences of the words in the documents taken into consideration. The search engine should assume that an AND logic operator is used. Then the query processor module consults a context index for dog and cat. For example, assume the resulting set is composed of document 1 (containing 3 dogs and 5 cats) and document 2 (containing 2 dogs and 5 cats). The rank is derived by simply calculating the sum of the occurrences of both words in a document. In this example, the score for document 1 is 8 (3+5) and the score for document 2 is 7 (2+5). Therefore, document 1 is ranked first.

For OR, the importance score is calculated based on the highest of the two words in the document. Using the previous example with document 1 (containing 3 dogs and 5 cats) and document 2 (containing 2 dogs and 5 cats). The rank is derived by taking the highest of the two occurrences. In this example, the score for document 1 is 5 and the score for document 2 is 5 as well. Both documents are ranked equally.

Output

The output should be take the form of Doucment ID, followed by a number indicating the importance of the webpage, and then the URL. The webpages should be ranked in descending order of importance.
A sample of the output could be:

Listing 3: Sample output

```
QUERY :> Dartmouth College AND Hanlon OR Mathematics AND Computer Science AND Philosophy OR
    incredibles Pixar

Printing the top 5 results or less for testing:
Document ID:5 Importance:659 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Dartmouth_College.html
Document ID:97 Importance:427 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Philosophy.html
Document ID:2 Importance:422 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Computer_science.html
Document ID:148 Importance:210
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Philosophy_of_mind.html
Document ID:64 Importance:179 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Computer_graphics.html
```

Data flow

The user input will be stored in an array and be transversed once (single transversal). If the word is not an operator, it will be stored in an array called “intersect”. If the next word is not an operator, it will be added to a temporary array and “intersect” will be updated to be the intersection between the temporary array and the previous intersect array. If the next word is an AND operator, then the following word will be added to the temporary array and “intersect” will be updated to be the intersection between the temporary array and the previous intersect array too .

When the word is the OR operator, then we will take the union of the intersect array with the previous union array (it can be empty).

The above operations are performed until we have finish transversing the user input.

The final array will be sorted based on the importance score and will be printed to standard output as described in the previous section.

II.II Data Structure

The data structure are described below:

int wordarray[MAXLEN][MAXLEN]: This is the array used to store the user input. The user is allowed to type in up to MAXLEN characters, which is 1000 by default as in the lab instructions.

int temparray[MAXSIZE][2]: This is the array used to temporarily store the document ID together with its corresponding frequency. The first column stores the document ID and the second column stores the frequency.

int intersect[MAXSIZE][2]: This is the array used to store the document ID togther with its corresponding frequncy when calculating the intersection of two words

final[MAXSIZE][2]: This is the array used to store the document ID togther with its corresponding frequncy when calculating the union of two words. It is also the final array that will be returned

The inverted index will be a hashtable with WordNodes, and each WordNode will contain the head to a single-linked list of DocumentNodes as described in the Indexer documentation.

The detailed data structure will be described in the functional specification section.

II.III Query Pseudocode

Here, we will described the pseudocode for query. The pseudocode for each function will be described in the Functional Specification section.

Listing 4: Pseudocode for query

```
/**
 * Check input parameters
 * Reconstruct inverted index from [INDEX FILE] using the ReadFile function
 * while (true){
 *     unionflag <- 0; flag <- 0;
 *     scan for words and operators (AND, OR) and store in an array
 *
 *     for all elements i in the array
 *             if the first element in the array is not an operator
 *                     normalize word and store it in the intersect array; continue;
 *             else if first element is an operator
 *                     input is invalid; flag = 1; break;
 *             else if unionflag == 1 (the previous word is a OR)
 *                     if the word is not an operator
 *                             normalize word and store it in intersect; unionflag <-0; continue;
 *                     else
 *                             input is invalid; flag = 1; break;
 *
 *             if the operator is AND
 *                     if operator is valid
 *                             normalize next word and store it in the intersect array; i++; continue;
 *                     else
 *                             input is invalid; flag = 1; break;
 *             else if operator is OR
 *                     if operator is valid
 *                             find the union of the final array and the intersect array; continue;
 *                     else
 *                            input is invalid; flag = 1; break;
 *             else (is not an operator, therefore acts as AND)
 *                    normalize word and store it in the intersect array; continue;
 *
 *             find the union of the final array and the intersect array
 *
 *             if flag == 1
 *                    invalid syntax;
 *             if size of final array == 0
 *                    no results found;
 *             else
 *                    sort final array by frequency
 *                    print results
 * Free memory
 **/
```

III Functional Specification

The functional specification section will detailed the data structure and functions employed in the implementation of the query engine.

III.I Data Structure

The data structure outined in the previous section will be implemented as shown below:

Forthe array used to store the document ID and frequency

Listing 5: Data structure: array to store document ID and frequency

```
/* Each array cotains MAXSIZE rows, MAXSIZE is defaulted to 1705, which is the maximum number of
    unique URLs. It should be adjusted according to the number of URLs crawled by crawler. The array
    also contain 2 columns, the first is used to store the document ID while the second is used to
    store the frequency or importance score*/

int temparray[MAXSIZE][2];                 // use to store the information temporarily

int intersect[MAXSIZE][2];                 // use to calculate the intersect

int final[MAXSIZE][2];                     // use to calculate the union
```

For the inverted index:

Listing 6: Data structure: WordNode

```
typedef struct WordNode {
  struct WordNode *next;                   // pointer to the next word (for collisions)
  char *word;                              // the word
  int docfreq;                             // number of documents with the word
  struct DocumentNode *page;               // pointer to the first element of the page list.
} WordNode;
```

Listing 7: Data structure: DocumentNode

```
typedef struct DocumentNode {
  struct DocumentNode *next;      // pointer to the next member of the list.
  int doc_id;                     // document identifier
  int freq;                       // number of occurrences of the word
} DocumentNode;
```

Listing 8: Data structure: Hash Table

```
typedef struct HashTable {
    struct WordNode* table[MAX_HASH_SLOT]; // actual hashtable
} HashTable;
```

III.II Files in the package and function prototype

The files contained in the package are described below. For the .c and .h file, function prototypes are also included.

Listing 9: Files in package lab6/query

```
Files in package lab6

1) lab6/query/src

        a) query.c
                functions:      char* getFileName(int fileName, char* dir);        // function from Indexer
                                int ReadFile(HashTable* index, char *file);         // function from Indexer
                                int FindHash(char* word, int array[][2], HashTable index);
                                int FindIntersection(int array1[][2], int size1, int array2[][2], int
                                    size2);
                                int FindUnion(int array1[][2], int size1, int array2[][2], int size2);
                                void Sort(int array[][2], int size, int criteria);
                                int BinarySearch(int array[][2], int min, int max, int doc);
                                int StringToWord(char wordarray[][MAXLEN], char text[MAXLEN]);
                                int CheckOperator(char wordarray[][MAXLEN], int check, int count);
                                int ReturnURL(char url[MAXLEN], int doc_id, char* dir);
        b) header.h

2) lab6/utils

        a) file.c and file.h
                functions:      int IsDir(const char *path);
                                int IsFile(const char *path);
                                int GetFilenamesInDir(const char *dir, char ***filenames);

        b) list.c and list.h
                functions:      int ListInsert(int doc, DocumentNode** dnode);
                                void CleanUpList(DocumentNode* head);

        c) hashtable.c and hashtable.h
                functions:      unsigned long JenkinsHash(const char *str, unsigned long mod);
                                int HashTableInsert(char* word, int doc, HashTable* index);
                                void InitialiseHashTable(HashTable* hashTable);
                                void CleanUpHash(HashTable* hashtable);

        d) web.c and web.h
                functions:      int GetNextWord(const char* doc, int pos, char **word);
                                void NormalizeWord(char *word);
        e) common.h                                // common functionality

        f) Makefile                                       // Makefile for the library

3) lab6/query/test

        a) QEBATS.sh                               // Automated testing

        b) queryengine_test.c                      // c script to support QEBATS.sh

        c) Makefile                                // Makefile for test

        d) header.h                                // header.h for queryengine_test.c

        e) Sun_Aug_16_11:07:59_2015.log            // log file

4) README                                          // README for query

5) Makefile                                        // make and make clean functionality for query

6) query_doc.pdf                                   // Documentation
```

III.III Function description

Description of each function will be outlined here:

Functions in query.c

Listing 10: Functions in query.c

```
/*
 * getFileName: function to get file name combined with directory path
 * @fileName: the file name
 * @dir: the directory to store the file
 *
 * Returns the new file name (file name combined with directory path)
 * Special consideration: It is the caller's responsibility to free the filename
 *
 * Pseudocode: 1. If directory ends with a '\', append filename with directory path
 *                          2. If directory does not end with a '\', append filename with directory
 *                                 path and include a '\' in between
 *
 */
char* getFileName(int fileName, char* dir);



/*
 * ReadFile: function to read the file and reconstruct the inverted index
 * @index: the reconstructed inverted index (should be NULL when passed into the function)
 * @file: the file name of the file with the inverted index
 *
 * Returns the reconstructed inverted index
 * Special consideration: index should be declared NULL before being passed into the function
 * Assumptions: the file is in the format specified in the documentation
 *
 * Pseudocode: 1. Open the file and check if it exist
 *                          2. Scan for the word, the number of documents with the word, the document
    id
 *                                 and the frequency of word occurrences
 *                          3. Input the parameters to reconstruct the index using HashTableInsert
 *
 */
int ReadFile(HashTable* index, char *file);



/*
 * FindHash: function to find the word in the inverted index and return with an array of
 *                   the document ID with the corresponding frequency of documents that contain the
    word
 *
 * @word: the word to be searched
 * @array: the array to be filled in
 * @index: the inverted index
 *
 * Returns an array of document ID with the corresponding frequency of documents that contain the word
 * Returns the size of the array
 *
 * Pseudocode: 1. Calculate the hashnum for the word, look through the table
 *                          2. Find the WordNode that matches the word using a while loop
 *                          3. Record all the Document ID in the WordNode in the array
 *                          4. Sort the array by document ID in ascending order
 *                          5. Return the size of the array if documents are found, return 0 otherwise
 *
 */
int FindHash(char* word, int array[][2], HashTable index);



/*
 * FindIntersection: function to find the intersection between 2 arrays
 *
 * @array1: the first array
 * @size1: the size of the first array
 * @array2: the second array
 * @size2: the size of the second array
 *
 * Modify array1 to be the intersection between array1 and array2
 * Returns the size of the modified array that is the intersection between array1 and array2
 *
 * Special Consideration: It is the caller's responsibility to ensure that array1 and
 *                                 array2 have already been sorted before passing them into the
    function
 *
 * Pseudocode: for all element i in array1
 *                                 for all elements j in array2 that are less than i
 *                                        if doc_id of i is same as j
 *                                               store the doc_id and combined freq in a dummy array
    intersect
 *                                               increment the number of element k in the intersect
    array
 *                                               remember the previous j value so we can start from
    that value
 *                                               break
```

```c
 *                             Transfer the content of the dummy array to array1
 *                             Sort array1
 *                             Returns the size of array1
 *
 */
int FindIntersection(int array1[][2], int size1, int array2[][2], int size2);



/*
 * FindUnion: function to find the union between 2 arrays
 *
 * @array1: the first array
 * @size1: the size of the first array
 * @array2: the second array
 * @size2: the size of the second array
 *
 * Modify array1 to be the union between array1 and array2
 * Returns the size of the modified array that is the union between array1 and array2
 *
 * Special Consideration: It is the caller's responsibility to ensure that array1 and
 *                                    array2 have already been sorted before passing them into the
    function
 *
 * Pseudocode: size <- size1
 *                             for all element i in array2
 *                                    if element i cannot be found in array1 (using a binary search)
 *                                                  store the doc_id and freq as a new element in array
    1
 *                                                  increment size
 *                                    else
 *                                           ensure that the element i is storing the greater frequency
 *
 *                             Sort array1
 *                             Returns the size of array1
 *
 */
int FindUnion(int array1[][2], int size1, int array2[][2], int size2);



/*
 * Sort: function to sort the array in ascending order of document ID or document frequency
 *
 * @array: the array to be sorted
 * @size: the size of the array
 * @criteria: the criteria to sort the array. 0 to sort by document ID and 1 to sort by frequency
 *
 * Modify array to be the sorted array
 *
 * Pseudocode: Classic bubble sort
 *
 */
void Sort(int array[][2], int size, int criteria);



/*
```

```c
 * BinarySearch: function to search for an element in an array using binary search
 *
 * @array: the array
 * @min: the minimum index to start from
 * @max: the maximum index to end with
 * @doc: the document ID to be searched for
 *
 * Returns the corresponding index of the search entry in the array if it is found, return
 *     -1 otherwise
 *
 * Pseudocode: Classic binary search using recursion
 *
 */
int BinarySearch(int array[][2], int min, int max, int doc);



/*
 * StringToWord: function to convert a string of words to an array of words
 *
 * @wordarray: the array of words to be created
 * @text: the string of words to be converted
 *
 * Returns the size of the word array
 *
 * Pseudocode: Read the word from the string using the function GetNextWord from indexer
 *                                 then store it in the array and free the memory
 *
 */
int StringToWord(char wordarray[][MAXLEN], char text[MAXLEN]);



/*
 * CheckOperator: function to check if an operator (AND, OR) is being used correctly
 *
 * @wordarray: the array of words
 * @check: the index of the operator in wordarray
 * @count: the size of wordarray
 *
 * Returns 1 if operator is used correctly, returns 0 otherwise
 *
 * Pseudocode: Check if the word before the operator and the word after is an operator
 *                                 if that is the case, return 1
 *                                 else return 0
 *
 */
int CheckOperator(char wordarray[][MAXLEN], int check, int count);



/*
 * ReturnURL: function to return the url by reading the html doc
 *
 * @url: the url to be returned
 * @doc_id: the document ID
 * @dir: the directory
 *
 * Returns 1 if URL is found, returns 0 otherwise
 *
 * Pseudocode: get file name from the dir and doc_id using the getFileName function
 *                          if url is obtained, return 1
 *                          else return 0
 *
 *                          file is closed and memory is freed at the same time
 */
int ReturnURL(char url[MAXLEN], int doc_id, char* dir);
```

IV Summary of error conditions

There are no major error conditions observed at the time of writing of this documentation. Boundary cases are tested for (and described in the Testing section). There are also no memory leaks.

V Test cases and expected results

Testing was performed using the command line, the automated testing script (QEBATS.sh) and Valgrind

1) From running ./query ../indexer/index.dat ../data

Listing 11: Runningquery using command line

```
$ ./indexer ../../data index.dat

Reading file: ../../data/1
Reading file: ../../data/10
Reading file: ../../data/100
Reading file: ../../data/1000
Reading file: ../../data/1001
Reading file: ../../data/1002
...
...
...
Reading file: ../../data/998
Reading file: ../../data/999
Inverted index saved in ../../data/index.dat
```

2) From running the test option: ./indexer ../../data index.dat new_index.dat

Listing 12: Running indexer using test option

```
$ ./query ../indexer/index.dat ../data

Please wait while the query engine is loading. It might take a few minutes...
QUERY :> AND
Error: Invalid input syntax
```

```
QUERY :> OR
Error: Invalid input syntax

QUERY :> Apple AND Orange

Printing the top 50 results or less:
Document ID:719 Importance:726 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Apple%2c_Inc.html
Document ID:1016 Importance:16 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Grep.html
Document ID:1206 Importance:10 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Manhattan.html
Document ID:827 Importance:9 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/FreeBSD.html
Document ID:1241 Importance:8 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Yahoo%21.html
Document ID:1328 Importance:7 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/EBay.html
Document ID:37 Importance:7 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/IBM.html
Document ID:836 Importance:6
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Linux_distributions.html
Document ID:1162 Importance:5
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Booz_Allen_Hamilton.html
Document ID:1113 Importance:4 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Dartmouth_green.html
Document ID:1110 Importance:4 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/United_States.html
Document ID:1040 Importance:3 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/UNIX_System_V.html
Document ID:786 Importance:3 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/OpenGL.html
Document ID:758 Importance:3 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/SVR4.html
Document ID:753 Importance:3 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/System_V.html
Document ID:760 Importance:2 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Sun_Microsystems.html
```

3) Logfile from a successful test using QEBATS.sh

Listing 13: Log file from QEBATS.sh

```
Test started at: Sun_Aug_16_11:07:59_2015
Operating system and host name: Linux tahoe.cs.dartmouth.edu 4.0.4-303.fc22.x86_64 #1 SMP Thu May 28
    12:37:06 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
start testing query


1. make clean and build query engine
rm -f *~
rm -f *#
rm -f *.o
rm -f queryengine_test
cd ../../util/; make clean
make[1]: Entering directory '/net/tahoe3/edrei/cs50/labs/cs50-X15-edrei/util'
rm -f *.o
rm -f *.a
make[1]: Leaving directory '/net/tahoe3/edrei/cs50/labs/cs50-X15-edrei/util'
cd ../../util/; make
make[1]: Entering directory '/net/tahoe3/edrei/cs50/labs/cs50-X15-edrei/util'
gcc -Wall -c *.c
ar -cvq libtseutil.a *.o
a - file.o
a - hashtable.o
a - list.o
a - web.o
make[1]: Leaving directory '/net/tahoe3/edrei/cs50/labs/cs50-X15-edrei/util'
gcc -Wall -pedantic -std=c11 -o queryengine_test ./queryengine_test.c -L../../util/ -ltseutil -lm
```

```
build query engine successfully


2.1 Test incorrect number of argument
./query ../../indexer/index.dat
Output: Error: Incorrect number of input argument
test input parameters successfully


2.2 Test invalid index file
./query invalidindex ../../data
Output: Error: File invalidindex is invalid
test input parameters successfully


2.3 Test invalid directory
./query ../../indexer/index.dat invaliddir
Output: Error: Directory invaliddir cannot be found
test input parameters successfully


3. Test query engine
./query ../../indexer/index.dat ../../data
Please wait while the query engine is loading. It might take a few minutes...

3.1 Test invalid input syntax
QUERY :> AND dog
Error: Invalid input syntax

Test 3.1 for query engine successfully

3.2 Test invalid input syntax
QUERY :> cat OR AND dog
Error: Invalid input syntax

Test 3.2 for query engine successfully

3.3 Test no result
QUERY :> thisisrandom
No results found.

Test 3.3 for query engine successfully

3.4 Test single entry
QUERY :> incredible

Printing the top 5 results or less for testing:
Document ID:1443 Importance:1 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Columbia_Lions.html
Document ID:1355 Importance:1
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Mister_Rogers%27_Neighborhood.html
Document ID:1148 Importance:1
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Supreme_Court_of_the_United_States.html
Document ID:1090 Importance:1 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/OS/2.html
Document ID:830 Importance:1 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Linus_Torvalds.html

Test 3.4 for query engine successfully

3.5 Test uppercase
QUERY :> Incredible
```

```
Printing the top 5 results or less for testing:
Document ID:1443 Importance:1 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Columbia_Lions.html
Document ID:1355 Importance:1
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Mister_Rogers%27_Neighborhood.html
Document ID:1148 Importance:1
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Supreme_Court_of_the_United_States.html
Document ID:1090 Importance:1 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/OS/2.html
Document ID:830 Importance:1 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Linus_Torvalds.html

Test 3.5 for query engine successfully

3.6 Test AND
QUERY :> Dartmouth AND College AND Computer AND Science

Printing the top 5 results or less for testing:
Document ID:5 Importance:668 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Dartmouth_College.html
Document ID:2 Importance:407 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Computer_science.html
Document ID:1155 Importance:381
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/College_admissions_in_the_United_States.html
Document ID:1522 Importance:300
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Saint_Anselm_College.html
Document ID:1498 Importance:244 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Union_Dutchmen.html

Test 3.6 for query engine successfully

3.7 Test space as AND
QUERY :> Dartmouth College Computer Science

Printing the top 5 results or less for testing:
Document ID:5 Importance:668 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Dartmouth_College.html
Document ID:2 Importance:407 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Computer_science.html
Document ID:1155 Importance:381
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/College_admissions_in_the_United_States.html
Document ID:1522 Importance:300
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Saint_Anselm_College.html
Document ID:1498 Importance:244 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Union_Dutchmen.html

Test 3.7 for query engine successfully

3.8 Test OR
QUERY :> Dartmouth OR Computer

Printing the top 5 results or less for testing:
Document ID:5 Importance:416 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Dartmouth_College.html
Document ID:1429 Importance:284
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/The_Tabard_(fraternity).html
Document ID:1127 Importance:284
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Dartmouth_College_Greek_organizations.html
Document ID:1440 Importance:254
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Dartmouth_Forensic_Union.html
Document ID:1227 Importance:250
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Dartmouth_College_student_groups.html

Test 3.8 for query engine successfully

3.9 Test combined
QUERY :> Dartmouth College AND Hanlon OR Mathematics AND Computer Science AND Philosophy OR
    incredibles Pixar
```

```
Printing the top 5 results or less for testing:
Document ID:5 Importance:659 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Dartmouth_College.html
Document ID:97 Importance:427 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Philosophy.html
Document ID:2 Importance:422 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Computer_science.html
Document ID:148 Importance:210
    URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Philosophy_of_mind.html
Document ID:64 Importance:179 URL:http://old-www.cs.dartmouth.edu/~cs50/tse/wiki/Computer_graphics.html

Test 3.9 for query engine successfully

9/9 tests are successful
All tests are successful. Query engine test success!


Test ended at: Sun_Aug_16_11:08:17_2015
```

3) Valgrind test for memory leak

Listing 14: Valgrind test for memory leak

```
QUERY :> ==786==
==786== HEAP SUMMARY:
==786==    in use at exit: 0 bytes in 0 blocks
==786==  total heap usage: 2,405,319 allocs, 2,405,319 frees, 39,405,504 bytes allocated
==786==
==786== All heap blocks were freed -- no leaks are possible
==786==
==786== For counts of detected and suppressed errors, rerun with: -v
==786== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

4) Boundary cases

The check for boundary cases are included in the QEBATS.sh logfile.

VI Special considerations

1. query returns the top 50 results (or less if there are fewer URLs that meet the search criteria). The user can change the maximum number of URLs return by changing the constant MAXRESULT (the default is 50).

2. query assumes that there are less than MAXSIZE unique urls. The default for MAXSIZE is 1705. If there are more urls, the value of MAXSIZE should be changed accordingly.

3. query allows the user to key in a maximum of MAXLEN characters for the search criteria and operators. Entries that contain more than MAXLEN characters will be truncated. The default for MAXLEN is 1000 and can be adjusted accordingly.

VII References

Dartmouth Canvas lab instruction for query. Accessed on 16 August 2015. Retrieved from:
`https://canvas.dartmouth.edu/courses/9618/assignments/46577`

Dartmouth Canvas template on Design and Implementation specifications for Crawler. Accessed on 16 August 2015. Retrieved from:
`https://canvas.dartmouth.edu/courses/9618/pages/lecture-13-tinysearchengine-crawler-design`