CS251 Data Structures

Lab3: Building a Search Engine

This is the grading form that will be used to grade lab2 and lab3

Lab 3 Frequently Asked Questions

Sample url.txt and word.txt

In this lab you will build a search engine that will search over the information you harvested in lab2.

Description:

You will build a search server that will search for the URLs and descriptions that correspond to a set of words. The networking code necessary for this project is given to you.

When the search server starts, it will read the information collected by web-crawler in the files url.txt and word.txt. It will create a word array with all the words and URL's. When the request arrives it will use a dictionary to find the URL's and descriptions that match this word and the result will be returned to the browser. If more than one word is passed, only the URL's that are common to all the words will be printed

Step 1. Download Files and Build

Download the file lab3-src.tar, uncompress it

```
tar -xvf lab3-src.tar
cd lab3-src
make
```

To run the server type

```
search-engine port (array | hash | avl | bsearch)
```

Where the first argument gives a port number, and the second argument tells the type of dictionary that will be used, which will be explained later.

Example:

```
./search-engine 8888 array
```

Where port is any number larger than 1024 that you can choose. This command is going to start a small web server that you will use as the search engine. Run a browser and connect to the URL that is printed by the search-engine command. If you see that there is a bind error try another port number.

Currently the search-engine command returns a hardwired document. You will modify the search-engine to do real searches.

Step 2. Extracting the Words from the Request.

First study the file search-engine.cpp, specially the method SearchEngine::dispatch(). This method is called when a request arrives from the browser. The argument *documentRequested* has the form "/search?word=word1+word2+word3..." You will need to parse documentRequested to extract the words *word1, word2, word3* that will be searched by your search engine.

Step 3. Implement the ArrayDictionary

Initially we recommend the you to implement the ArrayDictionary that is the most simple of all. In the file array-dictionary.cpp add the code that will implement the array-dictionary. See that in array-dictionary.cpp the class ArrayDictionary inherits from the class Dictionary. In the addRecord method you will add a record into the array. Your implementation will extend the length of the array if it has reached the maximum. Also the search for the words will be sequential.

Step 4. Loading the word file

In the file search-engine.cpp you will find the constructor SearchEngine::SearchEngine(port, dictionaryType). This constructor performs the initialization of the search engine. The argument *dictionaryType* tells the type of dictionary that should be used: ArrayDictionary, HashDictionary, AVLDictionary, or BinarySearchDictionary. You will create here, for now, the ArrayDictionary you implemented in step 3. In the header file search-engine.h you will find the class SearchEngine a member variable **_wordToURLList** This member variable will point to the dictionary that will map words to URLRecordList's. Initially this dictionary will be an ArrayDicitonary but you will later use other dictionary implementations depending on the dictionaryType. Once you create the dictionary, you will populate it with the files url.txt and word.txt.

Note: The key in the dictionary is a word (const char *) and the data is the URLRecordList *. You will need to cast the data to from (void*) to (URLRecordList *) before adding to the dictionary. The structure URLRecordList is defined in webcrawl.h but you may use your own header file if you desire.

Step 5. Searching the dictionary

You will modify ***SearchEngine::dispatch()*** to return the results of your search.

```
void SearchEngine::dispatch( FILE * fout, const char * documentRequested)
```

The "fout" argument is a pointer to a file that can be used in fprintf to format the output that will be sent to the client. The documentRequested argument is used to extract the word to be searched from Step 2. You will use the **_wordToURLList** dictionary to lookup the URL record that match the word. If more than one word is passed, you should print only the URL's that are common to all words.

Step 6. Implement the other Dictionaries

You will implement the other dictionaries as well. Modify the files hash-dictionary.*, avl-dictionary.*, and bsearch-dictionary.* with your implementation. Also modify the constructor SearchEngine::SearchEngine to create the dictionary that corresponds to the dictionaryType.

Step 7. Adding Other Information to the Search Output

In the search output also include the type of the dictionary used, and also the time that the search took and the average so far. Use the function **clock_gettime** that can measure the time in nanoseconds. Use CLOCK_REALTIME as the clock id in this function. See the man pages of **clock_gettime.**

Notes

You are allowed to do any modifications to the data structures, classes, and add any methods as long as you print a correct search output and use the correct type of dictionary. Also, you are invited to improve the appearance of the output.
Also, it is important that you run the "testall" script to check your data structures.

Turnin

Follow these instructions to turnin lab3:

1. Create in lab3-src a file called README with the following:

- Problems you have in your implementation and known bugs.
- Extra features.

Make sure that your search-engine can be built by typing "make". Make sure it builds and runs on data.

2. Type:

```
make clean
turnin -c cs251 -p lab3 lab3-src
```

3. Type:

```
turnin -c cs251 -p lab3 -v
```

to make sure you have submitted the right files

the deadline of this project is 11:59PM Tuesday July 25th 2017.
The project will be presented during your PSO on Wednesday July 26th and Thursday July 27th. Both lab2 and lab3 will be presented the same day. A draft of the grading form that we will be using is at the top of the handout.