**Advanced Computing Concepts**

**Search Engine Report**

We confirm that we will keep the content of this project description confidential. We confirm that we will work as a group, in equal parts, and that we have not received any unauthorized assistance in preparing for or writing this project. We acknowledge that a mark of 0 may be assigned for copied work.

Xuan He 104466911

Yating Su 104443714

Xinyi Zhou 104534524

Rui Wang 104491717

**Introduction**

In this project we need to create a web searching engine. We divide this project into five parts translate HTML to TXT, create word dictionary, page ranking, Knuth-Morris-Pratt (KMP) and Fragment picking, and UI. We use four methods (jsoup, regular expression, hash map , and KPM) to complete the project. Our searching engine can search the key word and list the related page based on the frequency. The result will show the page name, the website, the frequency of searching words and it will be highlighted in the text. We use the eclipse as our Compile environment.

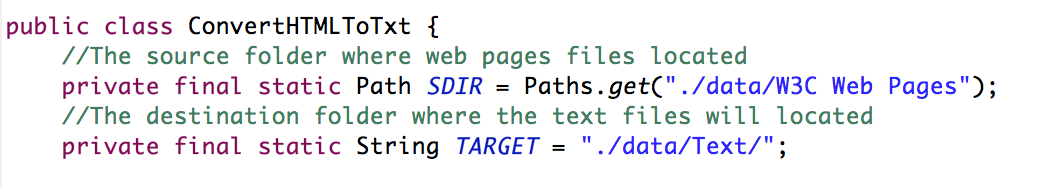
**Translate HTML to TXT**

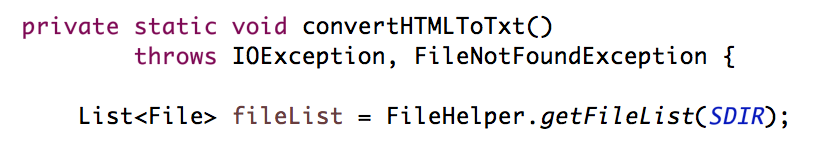
**Introduction:**

In this part, we would like to choose the main tool is jsoup, which is a open source that can parse HTML. In the class, we learned that jsoup is a Java library for working with real-world HTML, and it provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods, this is why we choose jsoup to help us achieve goals. In implement aspects, jsoup has several below advantages. First of all, jsoup implements the [WHATWG HTML5](http://whatwg.org/html) specification, and parses HTML to the same DOM as modern browsers do. Secondly, jsoup can scrape and [parse](https://jsoup.org/cookbook/input/parse-document-from-string) HTML from a URL, file, or string. Additionally, jsoup find and extract data, using DOM traversal or CSS selectors. Finally, jsoup can manipulate the HTML elements, attributes, and text and clean user-submitted content against a safe white-list, to prevent XSS attacks, then output tidy HTML. So jsoup is designed to deal with all varieties of HTML found in the wild, to invalid tag-soup from pristine and validating, and jsoup will create a sensible parse tree.

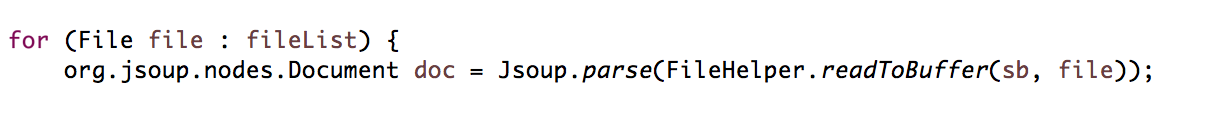
**Implementation:**

During the project, we have some static web pages in W3C Web Pages files, what we need to do is that change these file with extension “.html” to “.txt”. The first thing is that read all page files with extension “.htm” into a steam, and add each element of steam to file array list. We use Path.get() to get all page files by their location.



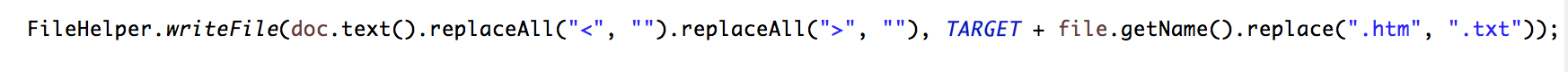


Then, the second step is that parse html in array list for loop one by one. We would like to use Jsoup.parse() to parse html page files, after this step, we will get files with extension “.txt”. To be more specific, these “.txt” files’ name equal to files’ name with extension “.html”.



Another problem needs to be noticed that we need to remove html tags from web pages when these web pages converting to txt files. So, according to this situation, Jsoup.parse() can help us solve this problem.

For example, jsoup can parse html code from <head>W3C Standards Web Design</head> to <head><W3C Standards Web Design></head>. Then the rest thing we need to do is remove all “<” “>”.



**Create word dictionary**

**Introduction:**

In create word dictionary part, we need to prepare the data first and then we need to choose a good method to store the files and create a dictionary. After previses step, we get hundreds text files. We need to deal with these files because people would not use the dictionary to search for the symbols such as “, . / ? ! @ # $ % & \* ( )”. The first step is removing these symbols. Second, we need to choose a perpetrate method to stock the information. There are 100 text files we need to store in the method, and there must have some words in each file may appears many times.

The techniques we use are regular expression and hash map. Regular expression is a way to deal with the data, which we can get the clearly data. The data structure of hash map is suitable for what we need. It is an array which can map keys and the value. We use the hash function to compute an index into an array of buckets or slots, form which the desired value can be found. In hashing, the hash functions are used to link key which is the key words and value which will be a page information object in HashMap.

The reason why we use the hash map as the method to build the dictionary is that the average time complexity of HashMap is O(1) when retrieving or storing the objects. When we pass key and value object to the put () method in Java HashMap, HashMap implement the hashCode method to the object and applies returned hashCode into its own hashing function to find a bucket location for storing.

Bellow figure is the HashMap data structure.

Keyword 1

Keyword 2

Keyword 3

Keyword 4

Hashing function

001

List<PageInfo, frequency>

002

List<PageInfo, frequency>

003

List<PageInfo, frequency>

012

List<PageInfo, frequency>

**Implementation:**

In the implementation part, there are three steps to complete a dictionary. First, deal with the data. Second, edit the value of hash map. Third, combine them together.

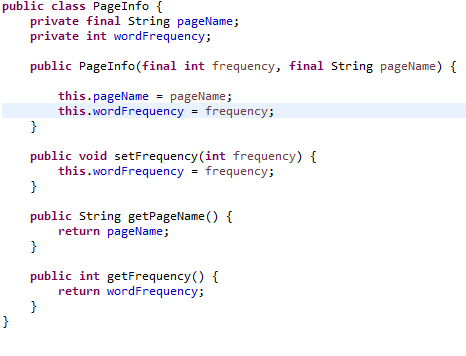
In the first step, all the files will be read to a stream again, at same time an array list named by its title will be generated. Then read through the file by using the read buffer and each character is read to a string builder in this file. Once encountering a character which is not the letters and digit, a splitter will be placed. After all the data was processing which means only left the digit and alphabets, it will be store in a string array named with page name. There are 100 files, so there will be 100 arrays. We use the regular expression to pick the digit and alphabets.



The next step, which is the preparation before hash the arrays into hash map. We need to hash the data into the hash map we need to edit the Value of the HashMap. For every word we need to know its frequency in each array and there are more than 100 arrays which means we have to know it belongs to which page. Therefore, the Value of the HashMap contains two parts the page name and the frequency of the word. We create a class to make sure the Value of the HashMap contains them. The Value can be created based on the key word and its frequency so page information object is:

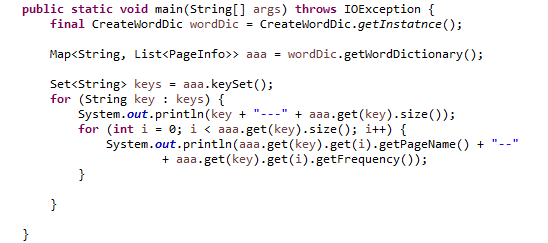
PageInfo pi = new PageInfo(table1.get(key), file.getName());

The page information build detials is showing below:



The last step is combining them together, and creates a hasp map. In this step we get the string arrays of all key words. Create a hash map which contains the Key and the Value. In this time, the key is the string array we ready to read, and the Value will be the PageInfo which is an array list contains the page name and the frequency of each page. We use Set<String>to get the keys and then hash them into the HashMap. Then if the key occurs in the wordDictionary, the page information object will be added to the existing page info list which is the value of the word dictionary. On the contrary, if the key does not exist in word dictionary, a new page information list will be created and all the new page information objects in. So the new entry of wordDictionary will be added.

After 100 files are done, the final word dictionary will be completed. There is one thing need to mention is that in order to implement this process once, the singleton pattern in Java is used, which means the word dictionary will be generated with initializing the program and end when the whole project is over. Please refer to bellow code for details:



**Page ranking**

**Introduction:**

Once a user input several key words to search corresponding webpages, the first step is to split key words into several single words, and collect all the corresponding webpages together. The second step is to calculate key words’ sum occurrence frequency in each page, and ranking webpages according to the keywords’ frequency. The process can be showed as the figure below.



**Implementation:**

There two main steps to implement page ranking.

The first step is to search key words from word dictionary and count the sum of frequency of key words in each page. For example, suppose a user input several key words, the input key words will be split and searched separately in the word dictionary. We also create an array list of type page information, as the figure ( 1 ) shows.



figure ( 1 ) create a list of type page information

If the word is found, a page information object list which contains the corresponding page will be returned. After getting all key words’ corresponding lists, the addAll() method is used to join them into a whole page information list, as the figure ( 2 ) shows.



figure ( 2 ) search each key word and add them together

Then the frequency of key words of this whole page information list is converted into a hash map. In the process of converting array list into hash map, the sum frequency of key words of each page is calculated. And the hash map takes the page name as key and the sum frequency of page as value. As the figure ( 3 ) shows, the sum frequency of each page is calculated.



figure ( 3 ) convert array list into hash map

The second step is to rank pages. After calculating the sum frequency of each page, we store each page’s name and sum frequency into a priority queue. According to pass a comparator to the priority queue and override compare method, pages are stored in descending order, which means page with highest frequency is on the top of the queue, as the figure ( 4 ) shows.



figure ( 4 ) creating an priority queue and passing comparator

Each page can be get one by one using poll() method. Before pages are printed, they are stored into an array list again in descending order using poll() method, as the figure ( 5 ) shows.



figure ( 5 ) using poll() method to get pages

**KMP and Fragment picking**

**Introduction:**

After the pages have been ranked, the pages will return to end-user. The result of the search will contain page name, repeat time, and a short content that search word located.

We used the KMP algorithm to do this because this algorithm has efficient running time and it compares quickly. This algorithm does not need to start comparing from the next position but directly jump to the search position.

As the first graph shows when pattern mismatch D with C. We actually know that the first six characters are "ABCDAB". The idea of the KMP algorithm is to try to use this information instead of moving back to the positon that has been compared before.It will increase the efficiency.As the second graph shows, the pattern will not move 1 position but move 4 position to D the move will realize by using partial match table. It let KMP can avoid redundant comparison when pattern string has repeat word. In this case, ABCDABD, which AB has repeated.Next step is the process of picking. At first, the function will read the content of top 10 ranked pages into string. And then it will use KMP algorithm to find position

To implement KMP algorithm, first of all, the names of the top 10 ranked pages for that searched words are already prepared from previous step. After that, the contents of the pages would be extracted from the pages and stored into a String one by one. Then a function that used to implement KMP algorithm would get the position of the word. While, unlike the search function provided for the assignment which will find the only first match position, a SearchAll function is created to search for up to first 5 match positions. As for SearchAll function, after the first match occurs, with substring function, the part that before the current match position can be deleted and the SearchAll function would keep moving forward to find the next match position. Additional, the SearchAll function would run until the 5 first matched positions are all found or the searched string reach the end. After get the positions, a map called resultList would be created which is used to store the searched words in String and the first 5 match positions in integer where the searched word occurs.

After preparing to locate the searched words in the pages, the fragments around the searched words could be extracted. To implement it, a function called FragmentPicker is created. FragmentPicker would calculate the start and end position around on the matched position, and extract the content totally for length of 300 between the start and end position. When 300 length string of content is extracted or the FragmentPicker extracts the content for the last match position, the function would be finished and return the extracted content to a String. For example, the first element in resultList map is a key word with up to 5 positions that key word occurred in that page. Based on the first position in resultList, the FragmentPicker function calculate the start position as position minus an integer and the end position as position plus an integer. Then the function extract the characters between the start and end position and store this in a String which is create to store the result of the fragment. Then the function would run for the next position until totally 300 length of characters have been extracted into the result String or the function reaches the last position for the first searched word. The FragmentPicker run for all elements in the resultList map and return strings for each elements. Finally, both of the resultList map and the string of the content will be returned to the UI. The user interface would use this information to display the search result to the user in an intelligent way.

**Implementation:**

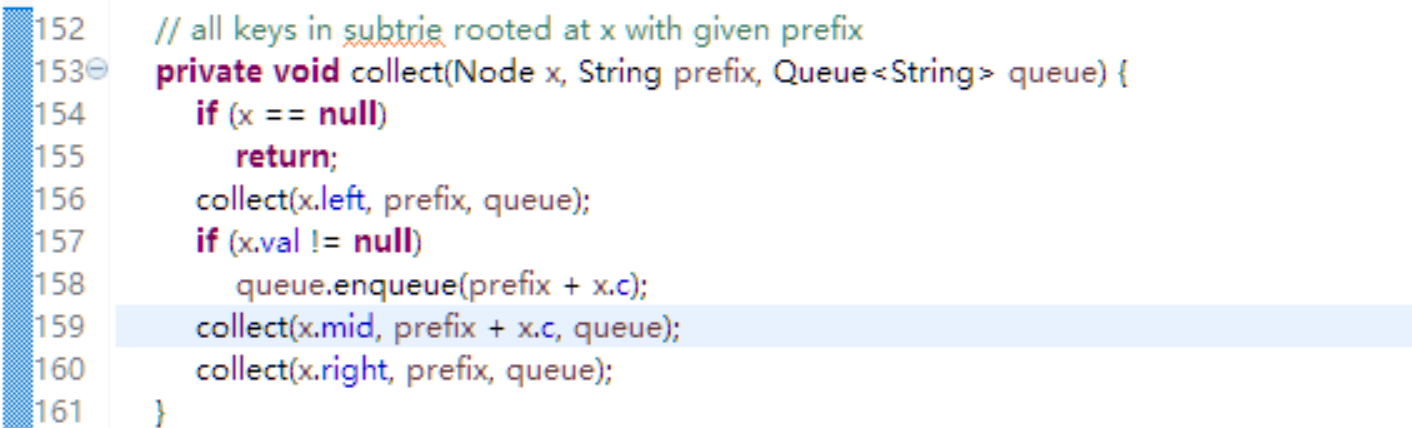
1. **Methodology**

In the application, we used prefix search in the TST to realized search assistance function. Due to prefix search is called every time the input is changed, the speed to show result is more important than accuracy. The original method of TST will return all the possible words that starting with the pattern and if the TST is getting bigger, the operation could lead to user wait for a long time, so we limit the size of the result queue and just return the first 5 matches. It may not return the most related word at first, but as the user keep typing, it is getting closer to a specific word that relevant to the input.

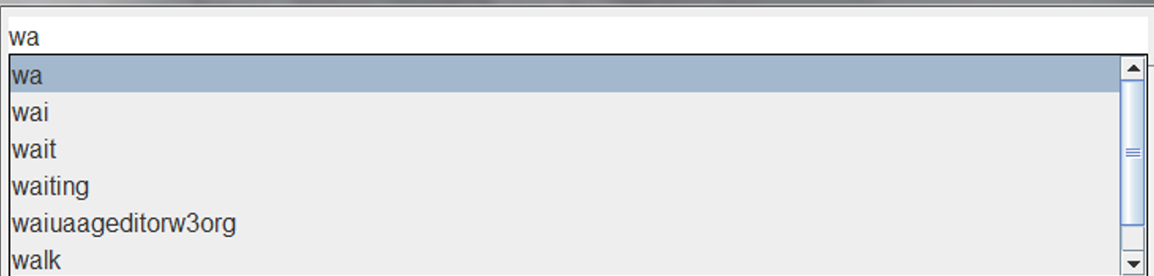
1. **Implementation of search algorithm**
   1. First create a method called prefixMatchLm, which takes a int parameter count to limit the search result.



* 1. Second, create a method called collect. It is used to count to limit the search result.



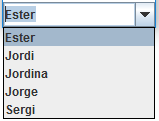
* 1. Using if to control the length of the queue, if queue grows longer than count, escape the method and return.

The result of the searching is: 

The first ad is the input, and count from wai to walk, The result is limited to show 5 words in prefix search.

1. **Implementation of UI of Search assistance**

The UI of the search assistance is mainly using the features of JComboBox,. The original appearance of the JComboBox is like:



It included an editor, a popup and a button. In order to implemented the search assistance appears the same as auto completed function on web search engine such as google, the button should be hided or deleted. Here is the code to delete the button:



1. **Implementation of UI of Search assistance**

In order to show the result, we used “JTextArea” to possess the result.If user need the feature like clicking the title of the Search result an open up the file, use JTextPane instead. In the search class, method listTopPages is called firstly to obtain the result pages as target. Secondly, call the pickFragment method to each result pages and return a fragment of the page content where searched words appeared. Finally, append the page information and the fragments to the textarea and highlight the searched words.

Highlight method is a native class of swing and only need to be apply to the textarea. A patter matcher is used to search the words.

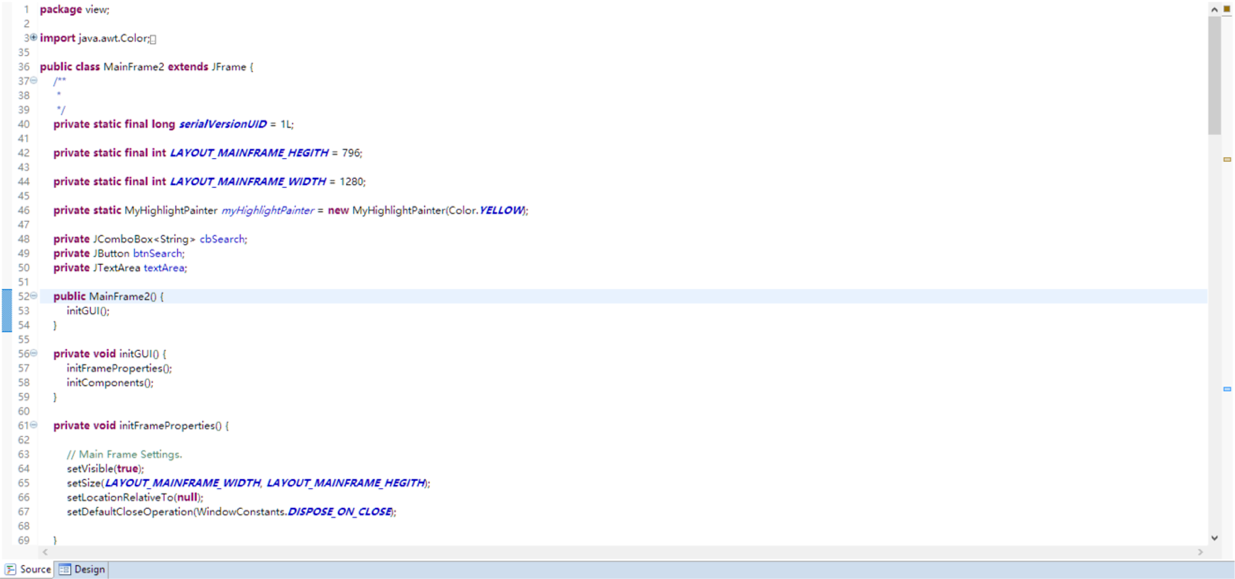
**UI design**

The main tools we use to create GUI is Swing. It is an API for providing a graphical user interface (GUI) for Java programs.

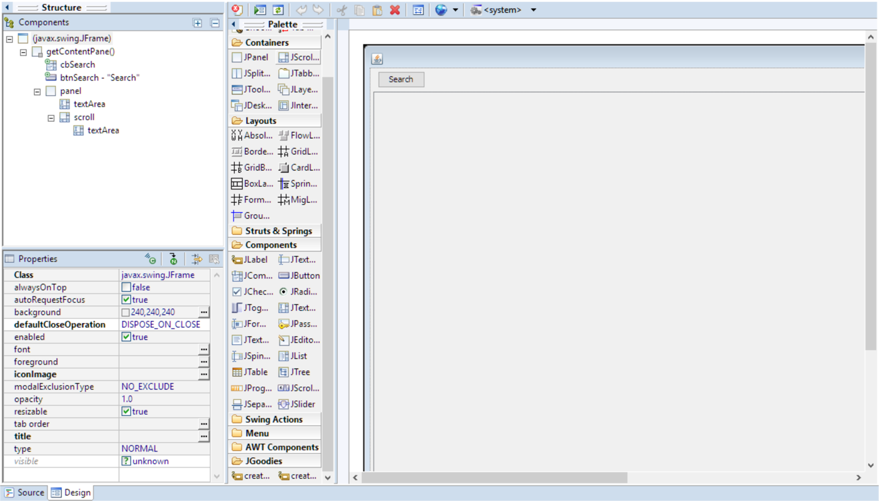
We use the “Window Builder” extension in eclipse to help us visually design the Swing frame, create the listener and set the layout and bounder.

Our mainframe design:

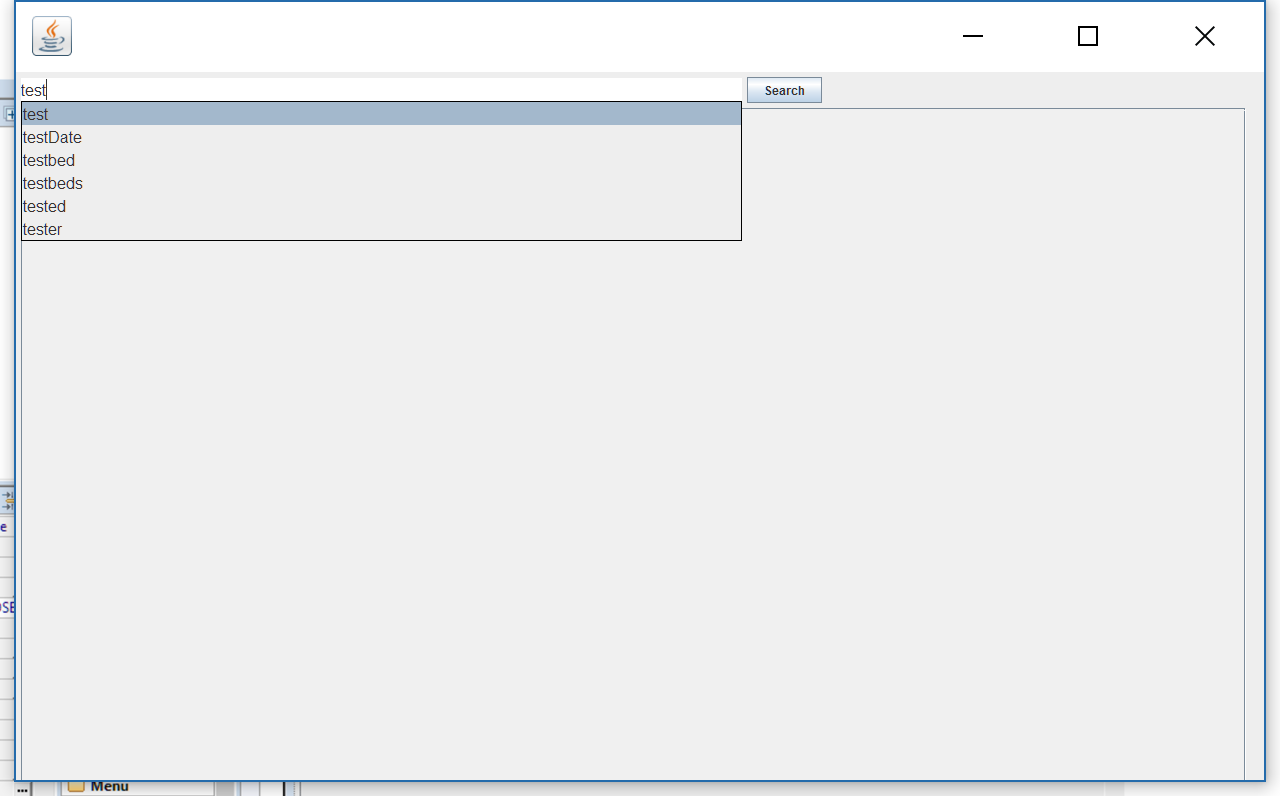
Part of source code:



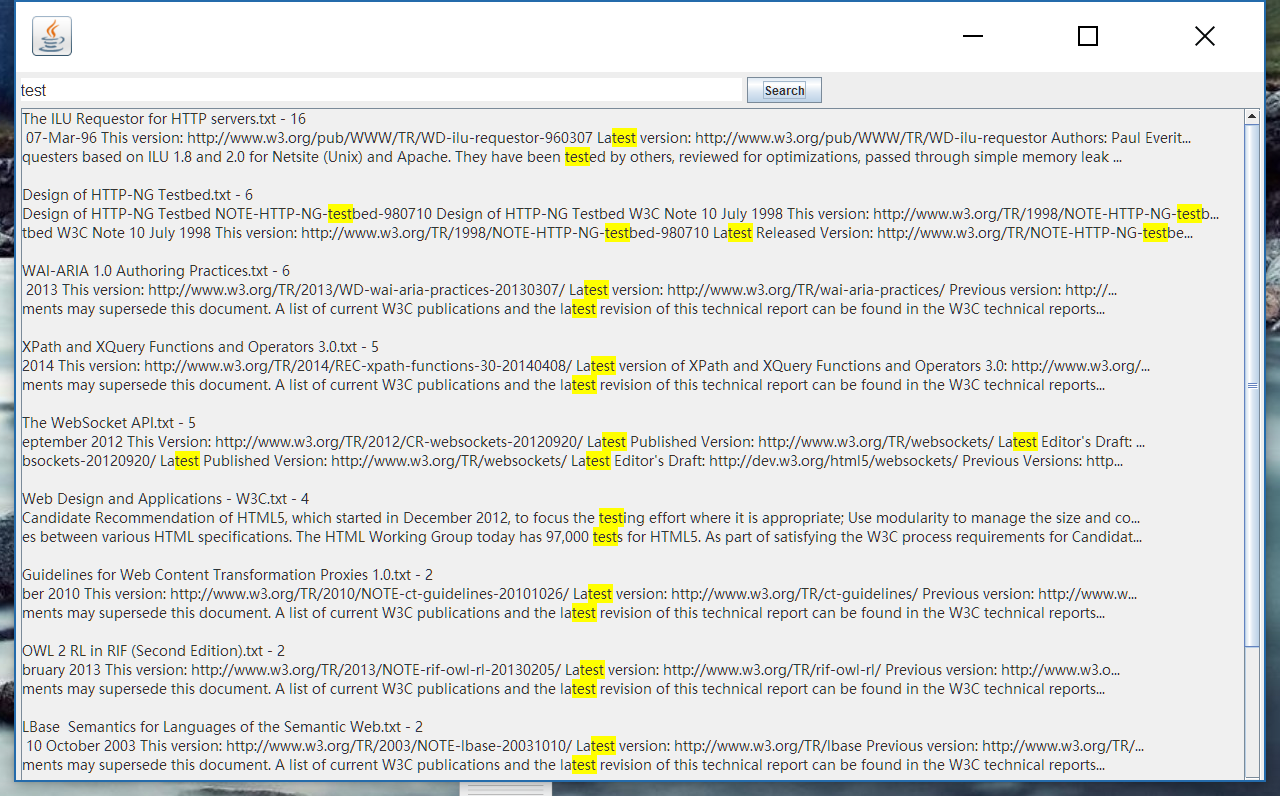
Design interface:



The result is like:



After do search:



The yellow part is the highlight of the key words in the contents

**Conclusion**

In this project, we learn a lot about the main function of search engine. We think that it is a great a way to help us to enhance the knowledge that what we learn in the class, especially about Based on the main function of search engine, the HTML conversion, the word dictionary, TST and ranging of the pages are designed. Meanwhile, based on the requirement that we analysis that user may have, the fragment extractor is designed as well as the UI design. This function means a lot to us, because it helps us to improve us by learning knowledge in the class. We really thank for this final project to influence our career in the future.

**Reference List**

Class slides of chapter 2(Hash table, Priory Queue)

Class slides of chapter 9(KMP, TST)

Class slides of chapter 10(Regex and Jsoup)