Project I:

Simple Search engine

Date: 2022/3/16

Chapter 1: Introduction

This project is aiming to realize a simple search engine on *The Complete Works of William Shakespeare*. The core of this program is **Inverted File Index (e.g.1)**, which simplifies the search work by dividing the whole sentence into single words and record their appearance frequency and positions in the sentence.

Meanwhile, considering different forms of verbs and words at extremely high frequency, **stemming words** and **stop words (e.g.2)** are necessary to be adopted to eliminate the redundancy work. After these steps, key words are left and stored in a B+ tree, which ranks key words in document number and their first letter.

To run the program, you should first enter an integer **m**, which is the number of operations you want to carry out, then enter letter "**s**" for searching function and key words. Program will print Top 10 files, which is determined by Threshold value, with the highest appearance frequency of key words.

Furthermore, if you want to check the word dictionary, please enter "**c**" after "**m**"; if you want to check total number of words, please enter "**n**" after "**m**".

e.g.1

A01: **I am a student of Zhejiang University.**

Keyword	Position
a	[3]
am	[2]
i	[1]
of	[5]
student	[4]
university	[7]
zhejiang	[6]

In this sentence, the frequency and document number of different key words are all the same.

e.g.2

I. Stemming word:

Words like "have", "having", "had" will be classified as "have"

II. Stop words:

Words such as "a", "an", "the"… have an extremely high frequency in English. They are labelled so that they will not be recorded by the program.

Chapter 2: Algorithm Specification

I. Stemming word

In this program, our group applies Porter Stemming Algorithm, which is coded up in ANSI C by the author. This algorithm summarizes and handle every situation of postfix, then finally help to rid them off the word.

The core of this algorithm is these functions below:

a) **int Cons(int i):** check if the i^{th} letter in the word is consonant.

b) **int m()**:calculate the number of consonants in certain part of a word.

c) **int vowelinstem()**: check if there is a vowel between k0 and j.

d) **int doublec(int i):** check if the letters on i^{th} and $(i-1)^{th}$ are the same consonant.

e) **int cvc(i)**: check if (i-2) - (i-1) - i has the form **consonant - vowel - consonant** and also if the second c is not w,x or y. this is used when trying to restore an e at the end of a short word.

e.g.

cav(e), lov(e), hop(e), crim(e)	**YES**
snow, box, tray	**NO**

f) **int ends(char* s):** check if the word is ended by char* s.

g) **int setto(char* s)**: sets (j+1) to k to the letters contained in the char* s.

After 5 steps of operation, which are combination of these functions, different kinds of postfix will be removed and words will return to primary forms.

II. B+ tree

In this program, B+ tree is used to establish the list of Stop words, create the document of Shakespeare.

There are 2 basic structures help to store the information.

```
struct WordNode
{
    Btree children;
    int frequency;
    string key_word;
};
//typedef struct Treenode* Btree;
//typedef struct WordNode* word;
```

```
struct Treenode
{
    word key[4];
    int key_num;
    //how much key word in this node
    int level;//used for BFS print
    Btree parent;
    Btree next;
    bool is_leaf;
    bool is_root;
};
```

After defining 2 basic structures, the relevant functions are present below.

a) **Btree Initial()**: this function is to initialize **struct Treenode**.

b) **void Index(Btree T,string val,Btree new_child)**: when inserting a new key with string value and its children into B+ tree, this function is to help order them.

c) **Btree find_leaf(Btree root, string val)**: when search node in the B+ tree, this function is called. It will first compare the target node value with the root node value, and decide which subtree should go. Then compare target value with every node in key[4]. According to B+ tree's properity, if we find equal or target value is smaller, then go to key[i-1]'s child node, then continue searching.

d) **word find_word(Btree T, string val)**: this function is simply visit every member in key[4] and return if finding target value, or return **NULL**.

e) **Btree insert(Btree root, string val, Btree leaf_contain)**: this function includes: (1) find the proper location to insert, (2) insert tree node, and (3) split up and down. For (1), we use function **find_leaf** to realize it. For (2), call **find_word** to check if the value of the node to be inserted has appeared in the B+ tree. For (3), check if the root node is full after inserting the node. If root node is full, create a new root and update.

f) **void Document_Insert()**: this function is called to read in words from .txt file and finish the establishment of dictionary, that is, call the stemming word functions to stem key words and add them into an initially empty document list, if word inserted is not in dictionary and not a stop word, which will mentioned below. Apart from inserting word into list, a global variable **total_word_num** increments when the function read in a word, which is used to calculate total number of Shakespeare's works.

g) **void search()**: read in words for std until "#", and call word stemming functions to get the stem of input.

For single word searching, we start by calling find_leaf & find_word, we can get to where the target word is in the dictionary list while excluding stop words, then the program will apply binary search to finish the rest work. Finally, the program prints out the highest 10, depending on threshold value, documents in descending order.

For multiple words input, the function will go through the file and find the word with lowest appearance frequency, which is the representative one among input words, then find the Top 10 document having highest frequency of the word and print them out in descending order.

III. Stop words

Stop words are set to eliminate redundancy in search work. In this program, we use “Stop words.txt” to list all of them, and a “#” is set at the end of the file. After that, stem function will be called to finish word stemming, then Insert function will be called to insert stemmed word into stop word list.

IV. Output

The output part is based on 2 functions called BFS and print:

For BFS: it is used to print out the B+ tree, including its index and leaves. The core in this function is to use queue to output B+ tree in height order. When going through the tree, by checking 3 different situations of current node, program will carry out different operations to output or give value to queue member.

And for print: we need to check if the current node is leaf node, then output every key word in the key[4] and go to next sibling and continue.

Chapter 3: Testing Result

Testing tips: change the value of threshold and test again

(1) search:

a) **input**: (threshold=10)

1

s love#

output:

```
1
s love#
Searching result: the document name that contain the search query (threshold=10)
As_You_Like_It
Loves_Labours_Lost
A_Midsummer_Night's_Dream
Much_Ado_About_Nothing
All's_Well_That_Ends_Well
King_Lear
Antony_and_Cleopatra
Cymbeline
Measure_for_Measure
Funeral_Elegy_by_W.S.
```

As you can see, program will first make sure that the threshold value, and print out corresponding number of document names. There are 10 works(documents) in total, so the program actually works.

b) **input**: (threshold=15)

keep input unchanged, but threshold value is changed to 15:

output:

```
1
s love#
Searching result: the document name that contain the search query (threshold=15)
Romeo_and_Juliet
As_You_Like_It
Loves_Labours_Lost
A_Midsummer_Night's_Dream
Much_Ado_About_Nothing
Othello_the_Moore_of_Venice
All's_Well_That_Ends_Well
King_Lear
Antony_and_Cleopatra
Cymbeline
The_First_part_of_King_Henry_the_Fourth
Pericles_Prince_of_Tyre
Measure_for_Measure
Funeral_Elegy_by_W.S.
The_Comedy_of_Errors
```

As you can see, the output lines change from 10 to 15, and program indicates that the threshold has been changed. So, the program functions well.

c) **input**: (threshold=10)

1

s Hamlet#(The hero of *The Tragedy of Hamlet Prince of Denmark*)

output:

```
1
s Hamlet#
Searching result: the document name that contain the search query (threshold=10)
The_Tragedy_of_Hamlet_Prince_of_Denmark
```

As you can see, because Hamlet is only appeared in the *The Tragedy of Hamlet Prince of Denmark*, it's impossible that Hamlet appears in other dramas. Therefore, there is only one output in the terminal.

d) **input**: (threshold=10)

1

s it will feed my revenge# (a sentence from *Merchants of Venice*)

output:

```
1
s it will feed my revenge#
Searching result: the document name that contain the search query (threshold=10)
As_You_Like_It
Antony_and_Cleopatra
Pericles_Prince_of_Tyre
Othello_the_Moore_of_Venice
Cymbeline
All's_Well_That_Ends_Well
A_Midsummer_Night's_Dream
Funeral_Elegy_by_W.S.
Loves_Labours_Lost
Measure_for_Measure
```

As you can see, when entering a complete sentence, the searching engine still works. Although we didn't get Merchants of Venice in output, it still makes sense. Because in our algorithm, the program will first find the rarest word among the sentence, then

search to find the top 10 documents containing most this word. In short, our search engine doesn't consider the input as a whole part but divides it and does its work.

e) **input:** (threshold=10)
2
s abcdsdes#
s a the #
output:

```
2
s abcdsdes#
Not Found!
s a the #
Not Found!
```

We group members first input letters randomly, make an input impossibly to be found, and the result is as expected: they were not found in the file! Then we tried stop words "a", "the", the result is still Not Found.

(2) output the word dictionary:
input:
1
c(\n)
output:

```
zani:Loves_Labours_Lost Twelfth_Night
zeal:Funeral_Elegy_by_W.S. Loves_Labours_Lost Much_Ado_About_Nothing The_First_part_of_King_Henry_the_Fourth
 The_Life_and_Death_of_King_John The_Life_and_Death_of_Richard_the_Second The_Life_and_Death_of_Richard_the_
Third The_Life_of_King_Henry_the_Eighth The_Life_of_King_Henry_the_Fifth The_Merchant_of_Venice The_Second_p
art_of_King_Henry_the_Fourth The_Second_part_of_King_Henry_the_Sixth The_Third_part_of_King_Henry_the_Sixth
Timon_of_Athens Titus_Andronicus Two_Gentlemen_of_Verona Winter's_Tale
zealou:All's_Well_That_Ends_Well Loves_Labours_Lost The_Life_and_Death_of_King_John The_Life_and_Death_of_Ri
chard_the_Third
zed:King_Lear
zenelophon:Loves_Labours_Lost
zenith:The_Tempest
zephyr:Cymbeline
zir:King_Lear
zo:King_Lear
zodiac:Measure_for_Measure Titus_Andronicus
zone:The_Tragedy_of_Hamlet_Prince_of_Denmark
zound:Othello_the_Moore_of_Venice Romeo_and_Juliet The_First_part_of_King_Henry_the_Fourth The_Life_and_Deat
h_of_King_John The_Life_and_Death_of_Richard_the_Third Titus_Andronicus
zwagger:King_Lear
```

As you can see, for the whole dictionary is too large to put here, here is a capture of its end. The output is apparently in a form of inverted files, that is "word: document1 document2…".

(3) print the total number:
input:
1
n(\n)
output:

```
1
n
The total words in Shakespeare works is 925356
```

Chapter 4: Analysis and Comments

I. Analysis

In our group's work, the algorithm concerning search work is mainly based on B+ tree, so the time complexity depends on that of the B+ tree established in this program.

As we know, the time complexity of inserting nodes into B+ tree is O (1). Therefore, our focus is searching through B+ tree.

For **B+ tree search time = the number of nodes visited * time of searching the node,** we can figure out the answer after calculating every value in the above equation. And we defined that B+ tree has **m** branches and **N** nodes in total.

(1) The number of the visited nodes equals the depth of B+ tree, which is apparently $\log_m N$.

(2) For there are m data in each node, and because our search is binary search towards the inner array of nodes, so the time complexity is O ($log_2 m$).

Therefore, the result of the equation is $\log_m N$* O ($log_2 m$). As for our program, m=4, so the final result is $log_2 N$.

II. Comments

After finishing the work of simple engine, our group discussed on how to improve our program, and we come to conclusion. The program writing format need to be improved.

For the program is finished in C++ language, but parts of the program such as Porter Stemming Algorithm coded in ANSI C which is mentioned in the program's comments, make our program kind of "strange" to some extent.

Besides, as for output of the program, we didn't unify all the output functions as "cout", there are still sentences finished in Basic C function like "printf".

Last but not least, in our group's work, C++ class was not applied, we still adopted form of "struct xxx" to finish data structure work. Although we are familiar with "struct", but using "class", keyword "private" can help us to protect the security of the data, and member function can also be adopted as a result to finish initializing operations.

Declaration

I hereby declare that all the work done in this project titled

"simple search engine" is of my independent effort.