Fundamentals of Data Structures

Laboratory Projects

Minimum Requirements on Writing a Project Report

Mini Search Engine

Author’s Name

Date: yyyy-mm-dd

Chapter 1: Introduction

This project aims to create a mini search engine that can handle queries over “The Complete Works of William Shakespeare”. The search engine will use an inverted index with word stemming to retrieve the relevant documents that contain the user-specified word or phrase. The search engine will also exclude the stop words or noisy words that are too common or irrelevant in the Shakespeare set. The project will consist of four tasks:

(1) Run a word count over the Shakespeare set and try to identify the stop words or noisy words. Explain how and where to draw the line between “interesting” and “noisy” words.

(2) Create an inverted index over the Shakespeare set with word stemming. The stop words identified in part (1) must not be included.

(3) Write a query program on top of the inverted file index, which will accept a user-specified word (or phrase) and return the IDs of the documents that contain that word.

(4) Run tests to show how the thresholds on query may affect the results.

Chapter 2: Algorithm Specification

Input:

A word or phrase to search for (string)

A directory containing Shakespeare's works in text format (string)

Output:

A list of names of the work where the search word or phrase appears.

Algorithm:

Pre-process Part:

Open the directory containing Shakespeare's works and read each file in the directory.

For each file, read its content, after removing stop words, we will deal interesting words with stem word library.

Enter the new words into an dictionary(hash table), count times of their appearance and calculate their frequency in each work and fill in the hash table. Add link list of work that the word appeared after each word. Also, we should use link list to store position.

Query Part:

For query word, firstly, deal it with stem word library, then look up the dictionary made in pre-process part and output the link list that content restrictions(like thresholds).

For phases, first handle it with stem word process and then search the dictionary for each word respectively, then take the intersection of the returned results, apply threshold filtering, and output.

Data structures:

Hash table (implemented as an undirected graph) and linked list

Stop words

At first, we suppose to some algorithm like K-means to select stop words according to their frequency, but it doesn’t come true. On the one hand, time doesn’t allow; on the other hand, we heard that stops words are selected manually. We choose an easier way, but possibly not safe: if a word occurs with a certain frequency in a certain number of articles，it is considered as stop words. We make a test set, test and choose good(but may not be the best) parameters.

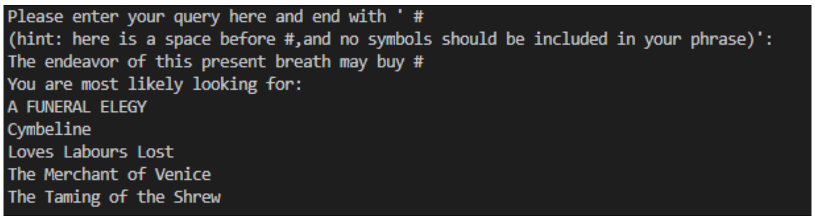
Chapter 3: Testing Results

Setence：

case 1:

The endeavor of this present breath may buy

from Loves Labours Lost

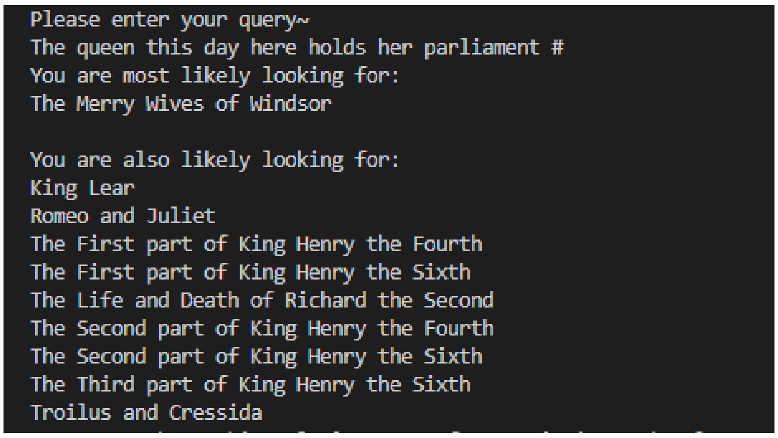


```
Please enter your query here and end with ' #
(hint: here is a space before #,and no symbols should be included in your phrase)':
The endeavor of this present breath may buy #
You are most likely looking for:
A FUNERAL ELEGY
Cymbeline
Loves Labours Lost
The Merchant of Venice
The Taming of the Shrew
```

case 2:

The queen this day here holds her parliament

from The Third part of King Henry the Sixth



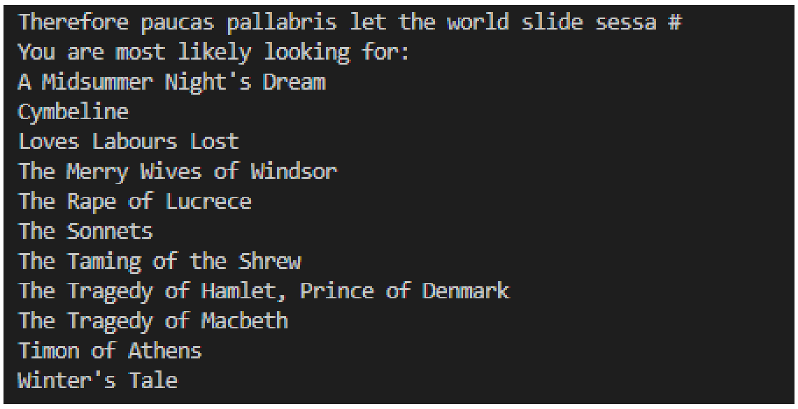
```
Please enter your query~
The queen this day here holds her parliament #
You are most likely looking for:
The Merry Wives of Windsor

You are also likely looking for:
King Lear
Romeo and Juliet
The First part of King Henry the Fourth
The First part of King Henry the Sixth
The Life and Death of Richard the Second
The Second part of King Henry the Fourth
The Second part of King Henry the Sixth
The Third part of King Henry the Sixth
Troilus and Cressida
```

case 3:

Therefore paucas pallabris let the world slide sessa

from The Taming of the Shrew



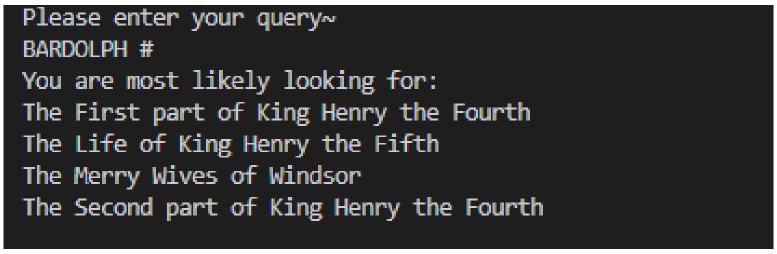
```
Therefore paucas pallabris let the world slide sessa #
You are most likely looking for:
A Midsummer Night's Dream
Cymbeline
Loves Labours Lost
The Merry Wives of Windsor
The Rape of Lucrece
The Sonnets
The Taming of the Shrew
The Tragedy of Hamlet, Prince of Denmark
The Tragedy of Macbeth
Timon of Athens
Winter's Tale
```

Word

case 4:

BARDOLPH

from The Second part of King Henry the Fourth

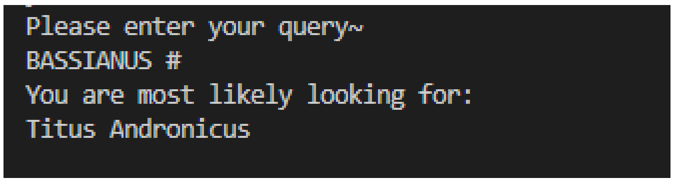


```
Please enter your query~
BARDOLPH #
You are most likely looking for:
The First part of King Henry the Fourth
The Life of King Henry the Fifth
The Merry Wives of Windsor
The Second part of King Henry the Fourth
```

case 5:

BASSIANUS

from Titus Andronicus



```
Please enter your query~
BASSIANUS #
You are most likely looking for:
Titus Andronicus
```

Chapter 4: Analysis and Comments

Comments:

Time complexity:

Pre-process: O(nlogn), for we build RB tree while traversing the words.

Searching: O(nlogn), for RB tree insertion’s time complexity is O(logn), when searching, we just build the map when traversing, so it should be O(nlogn).

Space complexity:

Pre-processing: Can’t be measured. That’s because we use map to store link lists. The list, contain the number of word’s appearance which is dependent on writing habits and the language. So it is logn times something, further step should continue with more information.

Searching: O(n), for RB tree’s space complexity is O(n), we use RB tree to save words, the number of article is countable, so it should be O(n).

For pre-process: n is for number of words.

For searching: n is for number of interesting words.

Appendix: Source Code

Create_word.cpp

```
#include<iostream>

#include<string>

#include<fstream>

#include<unordered_map>

#include<map>

#include<math.h>
```

```
#include<iomanip>

#include <algorithm>

#include<string.h>

#include<ctype.h>

using namespace std;


typedef struct _word_position{

    int position;

    struct _word_position *next;

}word_position;

// this structure is used for contain the position

typedef struct _title_node{

    word_position *p_head;

    word_position *p_tail;

    double cnt;

    string title;

    float persent;

    struct _title_node *next;

}title_node;

// to establish the inverted index, we need

// a title node, which include the title of

// article that contain the word, frequency
// of the word in this article, pointer to the
// next article that contain the word and the
// position of the word in the article
typedef struct _word_num{
    double num;
    string title;
}word_num;

typedef struct _word_cnt{
    double cnt;
    title_node *head;
    title_node *tail;
}*word_cnt;
// every word appears in all article, then we divide
// it into stop words and interesting words.
word_cnt create(){
    word_cnt block = (word_cnt)malloc(sizeof(struct
_word_cnt));
    block->cnt = 0;

    block->head = block->tail = NULL;
    return block;
```

```
}

// create a new linked list, set all members

//to their defult.

void append(word_cnt block,string name){

    block->cnt++;

    if(block->head){                                                      /

/不是空的

        if(block->tail->title != name){

            title_node *temp = new title_node;

            temp->p_head = temp->p_tail = NULL;

            temp->title = name;

            temp->persent = -1;

            temp->cnt = 0;

            temp->next = NULL;

            block->tail->next = temp;

            block->tail = temp;

        }

    } else

{                                              //是空的

        block->head = new title_node;

        block->head->title = name;

        block->head->persent = -1;
        block->head->p_head =block->head->p_tail = NULL;

        block->cnt = 0;

        block->head->next = NULL;

        block->tail = block->head;

    }

}


void p_append(word_cnt block,int position){


    if(block->tail->p_head){

        word_position *temp = new

word_position;                                                //不是空

的

        temp->position = position;

        temp->next = NULL;

        block->tail->p_tail->next = temp;

        block->tail->p_tail = temp;


    } else

{                                                       //是空的

        block->tail->p_head = new word_position;

        block->tail->p_head->position = position;
        block->tail->p_head->next = NULL;

        block->tail->p_tail = block->tail->p_head;

    }

}

// remove all 's or 'S, normally, only

// proper nouns have this form, they shouldn't

// be treated to be stop words.

string cut_tail(string word){

    if(word.length() >= 2){

        if((word[word.length()-2] == '\'') &&

(word[word.length()-1]=='S' ||

word[word.length()-1]=='s')){

            word = word.erase(word.length()-2,2);

        }

    }

    return word;

}

// this is designed for select proper nouns,

// according to the custom of English writing,

// we list cases to select specific nouns.

bool IS_name(string word){

    bool flag = true;
    if((word.length() < 5) ||
(!isalnum(word[word.length()-1]))){
        flag = false;
        return flag;
    }
    for(int i = 0; i < word.length();i++){
        if(islower(word[i])){
            flag = false;
            break;
        }
    }
    return flag;
}
// this function is designed for making all
// words have the same case
string lowerString(string s){
    transform(s.begin(), s.end(), s.begin(),[](unsigned
char c){ return tolower(c); });
    return s;
}
// this is designed for delete punctuation marks
// but punctuation marks inside don't have to be
// removed since when querying, they're reserved

string alphanumize(string a){

    while(!isalnum(a[0])){

        a = a.erase(0, 1);

        if(a == "")

            break;

    }

    if(a != ""){

        while(!isalnum(a[a.length()-1])){

        a = a.erase(a.length()-1, 1);

        if(a == "")

            break;

        }

    }

    return a;

}

// this function is designed for replacing

// all ' ' of the title with "_".

// Then we can have convenience in reading

// and writing titles in files.

string change_title(string title){

    for(int i = 0; i < title.length(); i++){
        if(title[i] == ' '){
            title[i] = '_';
        }
    }
    return title;
}

int main(){
    word_num  array[42];
    int count = 0;
    ifstream titles("titles.txt");
    ofstream fw("./result/interesting_word.txt");
    ofstream show("./result/show.txt");
    map<string, word_cnt> word_bag;
    map<string, word_cnt> name;
    // open the file that stores all
    // titles, then read article with
    // the title we have read.
    // word_count is for marking the position
    // of word appears in each article.
    if(titles){
        string line;
        string current_title;

        string word;

        while(getline(titles,line)){

            if(line != ""){

                current_title ="./articles/" + line +
".txt";

                ifstream article(current_title);

                array[count].title = line;

                array[count].num = 0;

                if(article){

                    int word_count = 0;

                    double current_sum = 0;

                    cout << line << " begin,sir" << endl;

                    while (article >> word){ // line 中不
包括每行的换行符

                        //for each word, the position get
increment.

                        word_count++;

                        word = cut_tail(word);

                        if(word != ""){

                            if(IS_name(word)){
```

```
                            if(name.find(word) ==
name.end()){      //如果 word 不在 name 中
                                name[word] = create();
                            }
                            // when empty create a new
link list
                            append(name[word],line);
                            p_append(name[word],word_
count);
                            name[word]->tail->cnt++;
                            array[count].num++;
                            // for word, no matter it is
name or the orther
                            // we append them into
corresponding map
                        } else {
                            word = alphanumize(word);
                            word = lowerString(word);
                            if(word != ""){
                                if(word_bag.find(word)
== word_bag.end()){      //如果 word 不在 name 中
```

```
                                                word_bag[word] =
create();
                                   }
                                   // when empty create a
new link list
                                   append(word_bag[word]
,line);
                                   p_append(word_bag[wor
d],word_count);
                                   word_bag[word]->tail-
>cnt++;
                                   array[count].num++;
                                   // for word, no matter
it is name or the orther
                                   // we append them into
corresponding map
                               }
                           }
                       }
                   }
                   article.close();
                   cout << line << " finish,sir" << endl;
```

```
                    count++;
                } else {
                    // these are used for notice,don't mind.
                    cout << "#" << line << "#" << "can't
found" << endl;
                }
            }
        }
    } else {
        // when the file can't open, printout error.
        cout << "No Such File With Titles,Sir" << endl;
    }

    string temp_line;

    map<string, word_cnt>::iterator pointer =
word_bag.begin( );
    // then we construct the file used for inverted index
and
    // searching. word_bag is used for store all distinct
words
    // now we use it to write into these files.
    while(pointer!=word_bag.end()){
        if((pointer->first != "") && 
(pointer->second->cnt > 2)){
            int ccount = 0;
            title_node *ptr = pointer->second->head;
            while(ptr){
                ccount++;
                ptr = ptr->next;
            }
            // we suppose the thresholding to be less than 
7
            // seven times appears in different article.
(here, 
            //if a word appear in a article, we take one 
time.)
            if(ccount < 7){
                show << pointer->first << ":\n";
                fw << pointer->first << " $";
                ptr = pointer->second->head;
                while(ptr){
                    for(int i = 0;i < 42; i++){
                        if(array[i].title == ptr->title){
                            ptr->persent =
ptr->cnt/array[i].num;
                            break;
                        }
                    }
                    temp_line = change_title(ptr->title);
                    show << " " << ptr->title << " cnt:" <<
ptr->cnt;
                    fw << " " << temp_line << " "
<<ptr->persent;
                    word_position *q = ptr->p_head;
                    while(q){
                        show << " " << q->position;
                        q = q->next;
                    }
                    // this is used for beautiful looking,
for
                    // your reading the content.
                    show << "\n";
                    ptr = ptr->next;
                }
                show << "\n";
                fw << " #\n";
            }
        }
        pointer++;
    }
    map<string, word_cnt>::iterator point = name.begin( );
    while(point!=name.end()){
        if((point->first != "") && (point->second->cnt >
2)){
            title_node *ptr = point->second->head;
            int ccount = 0;
            while(ptr){
                ccount++;
                ptr = ptr->next;
            }
            // we suppose the thresholding to be less than
7
            // seven times appears in different article.
(here,
            // if a word appear in a article, we take one
time.)
            // actually, the name and word_bag is so similar
            // just copy the same code and edit some varible.
            if(ccount < 7){
                show << point->first << ":\n";
                fw << point->first << " $";
                ptr = point->second->head;
                while(ptr){
                    for(int i = 0;i < 42; i++){
                        if(array[i].title == ptr->title){
                            ptr->persent =
ptr->cnt/array[i].num;
                            break;
                        }
                    }
                    // this is used for beautiful looking,
for
                    // your reading the content.
                    temp_line = change_title(ptr->title);
                    show << " " << temp_line << " cnt:"
<<ptr->cnt;
                    fw << " " << temp_line << " "
<<ptr->persent;
                    word_position *q = ptr->p_head;
```

```cpp
                    while(q){
                        show << " " << q->position;
                        q = q->next;
                    }
                    show << "\n";
                    ptr = ptr->next;
                }
                show << "\n";
                fw << " #\n";
            }
        }
        point++;
    }
    show.close();
    fw.close();
    // a sign for the completion.
    cout << "All tasks have been finished,sir";
    return 0;
}
```

Read_word.cpp

```cpp
#include <iostream>
#include <string>
#include <fstream>
#include <unordered_map>
#include<map>
#include<math.h>
#include<iomanip>
#include <algorithm>
#include<string.h>
#include<ctype.h>
using namespace std;

#define TRUE 1
#define FALSE 0
/*

------------------------------------------------
  WARNING   WARNING   WARNING   WARNING
------------------------------------------------
This .cpp file may print out many possible articles.
The strategy used in searching is mainly on searching on
interesting words, time is too short for establishing the
```

```
inverted index for searching.😭😭😭
Due to different standard of identifying stop
words, the strategy taught in class doesn't have
significant effect. So we finally choose to print the
article that has the biggest and the second biggest number
of words appears in.
Also, the project is searching on words not order, we don't
pay attention to the order of input




*/
int stem(char *p, int index, int position);

static char *b;

static int k;
static int k0;

/* j is a general offset into the string */
static int j;

/**
```

```
 * TRUE when `b[i]` is a consonant.

 */


static int

isConsonant(int index) {

  switch (b[index]) {

    case 'a':

    case 'e':

    case 'i':

    case 'o':

    case 'u':

      return FALSE;

    case 'y':

      return (index == k0) ? TRUE : !isConsonant(index - 1);

    default:

      return TRUE;

  }

}


/* Measure the number of consonant sequences between

 * `k0` and `j`.  If C is a consonant sequence and V

 * a vowel sequence, and <..> indicates arbitrary
 * presence:
 *
 *   <C><V>       gives 0
 *   <C>VC<V>     gives 1
 *   <C>VCVC<V>   gives 2
 *   <C>VCVCVC<V> gives 3
 *   ....
 */
static int
getMeasure() {
  int position;
  int index;

  position = 0;
  index = k0;

  while (TRUE) {
    if (index > j) {
      return position;
    }

    if (!isConsonant(index)) {
```

```
    break;
  }

  index++;
}

index++;

while (TRUE) {
  while (TRUE) {
    if (index > j) {
      return position;
    }

    if (isConsonant(index)) {
      break;
    }

    index++;
  }

  index++;
```

```
    position++;

    while (TRUE) {
      if (index > j) {
        return position;
      }

      if (!isConsonant(index)) {
        break;
      }

      index++;
    }

    index++;
  }
}

/* `TRUE` when `k0, ... j` contains a vowel. */
static int
vowelInStem() {
  int index;
```

```c
  index = k0 - 1;

  while (++index <= j) {
    if (!isConsonant(index)) {
      return TRUE;
    }
  }

  return FALSE;
}

/* `TRUE` when `j` and `(j-1)` are the same consonant. */
static int
isDoubleConsonant(int index) {
  if (b[index] != b[index - 1]) {
    return FALSE;
  }

  return isConsonant(index);
}
```

```
/* `TRUE` when `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.
 *
 * Such as:
 *
 * `cav(e)`, `lov(e)`, `hop(e)`, `crim(e)`, but `snow`,
 * `box`, `tray`.
 */
static int
cvc(int index) {
  int character;

  if (index < k0 + 2 || !isConsonant(index) ||
isConsonant(index - 1) || !isConsonant(index - 2)) {
    return FALSE;
  }

  character = b[index];

  if (character == 'w' || character == 'x' || character ==
'y') {
    return FALSE;
  }

  return TRUE;
}

/* `ends(s)` is `TRUE` when `k0, ...k` ends with `value`.
*/
static int
ends(const char *value) {
  int length = value[0];

  /* Tiny speed-up. */
  if (value[length] != b[k]) {
    return FALSE;
  }

  if (length > k - k0 + 1) {
    return FALSE;
  }
  if (memcmp(b + k - length + 1, value + 1, length) != 0)
{
    return FALSE;
  }

  j = k - length;

  return TRUE;
}

/* `setTo(value)` sets `(j + 1), ...k` to the characters
in
 * `value`, readjusting `k`. */
static void
setTo(const char *value) {
  int length = value[0];

  memmove(b + j + 1, value + 1, length);

  k = j + length;
}
```

```c
/* Set string. */
static void
replace(const char *value) {
  if (getMeasure() > 0) {
    setTo(value);
  }
}

/* `step1ab()` gets rid of plurals, `-ed`, `-ing`.
 *
 * Such as:
 *
 *   caresses  ->  caress
 *   ponies    ->  poni
 *   ties      ->  ti
 *   caress    ->  caress
 *   cats      ->  cat
 *
 *   feed      ->  feed
 *   agreed    ->  agree
 *   disabled  ->  disable
 *
 *   matting   ->  mat
 *   mating    ->  mate
 *   meeting   ->  meet
 *   milling   ->  mill
 *   messing   ->  mess
 *
 *   meetings  ->  meet
 */
static void
step1ab() {
  int character;
  if (b[k] == 's') {
    if (ends("\04" "sses")) {
      k -= 2;
    } else if (ends("\03" "ies")) {
      setTo("\01" "i");
    } else if (b[k - 1] != 's') {
      k--;
    }
  }
//   string _str = b;
//   cout << _str << endl;
  if (ends("\03" "eed")) {
    if (getMeasure() > 0) {
      k--;
    }
  } else if ((ends("\02" "ed") || ends("\03" "ing")) &&
vowelInStem()) {
    k = j;

    if (ends("\02" "at")) {
      setTo("\03" "ate");
    } else if (ends("\02" "bl")) {
      setTo("\03" "ble");
    } else if (ends("\02" "iz")) {
      setTo("\03" "ize");
    } else if (isDoubleConsonant(k)) {
      k--;

      character = b[k];

      if (character == 'l' || character == 's' || character
== 'z') {
        k++;

      }

    } else if (getMeasure() == 1 && cvc(k)) {

      setTo("\01" "e");

    }

  }

}


/* `step1c()` turns terminal `"y"` to `"i"` when there

 * is another vowel in the stem. */

static void

step1c() {

  if (ends("\01" "y") && vowelInStem()) {

    b[k] = 'i';

  }

}


/* `step2()` maps double suffices to single ones.

 * so -ization ( = -ize plus -ation) maps to -ize etc.

 * note that the string before the suffix must give

 * getMeasure() > 0. */

static void
```

```
step2() {

  switch (b[k - 1]) {

    case 'a':

      if (ends("\07" "ational")) {

        replace("\03" "ate");

        break;

      }


      if (ends("\06" "tional")) {

        replace("\04" "tion");

        break;

      }


      break;

    case 'c':

      if (ends("\04" "enci")) {

        replace("\04" "ence");

        break;

      }


      if (ends("\04" "anci")) {

        replace("\04" "ance");
        break;
      }

      break;
    case 'e':
      if (ends("\04" "izer")) {
        replace("\03" "ize");
        break;
      }

      break;
    case 'l':
      /* --DEPARTURE--: To match the published algorithm,
       * replace this line with:
       *
       * ```
       * if (ends("\04" "abli")) {
       *     replace("\04" "able");
       *
       *     break;
       * }
       * ```
```

```
       */
      if (ends("\03" "bli")) {
        replace("\03" "ble");
        break;
      }

      if (ends("\04" "alli")) {
        replace("\02" "al");
        break;
      }

      if (ends("\05" "entli")) {
        replace("\03" "ent");
        break;
      }

      if (ends("\03" "eli")) {
        replace("\01" "e");
        break;
      }

      if (ends("\05" "ousli")) {
        replace("\03" "ous");

        break;

      }


      break;

    case 'o':

      if (ends("\07" "ization")) {

        replace("\03" "ize");

        break;

      }


      if (ends("\05" "ation")) {

        replace("\03" "ate");

        break;

      }


      if (ends("\04" "ator")) {

        replace("\03" "ate");

        break;

      }


      break;
    case 's':
      if (ends("\05" "alism")) {
        replace("\02" "al");
        break;
      }

      if (ends("\07" "iveness")) {
        replace("\03" "ive");
        break;
      }

      if (ends("\07" "fulness")) {
        replace("\03" "ful");
        break;
      }

      if (ends("\07" "ousness")) {
        replace("\03" "ous");
        break;
      }

      break;
    case 't':
      if (ends("\05" "aliti")) {
        replace("\02" "al");
        break;
      }

      if (ends("\05" "iviti")) {
        replace("\03" "ive");
        break;
      }

      if (ends("\06" "biliti")) {
        replace("\03" "ble");
        break;
      }

      break;
    /* --DEPARTURE--: To match the published algorithm,
delete this line. */
    case 'g':
      if (ends("\04" "logi")) {
        replace("\03" "log");
```

```c
        break;
      }
  }

}

/* `step3()` deals with -ic-, -full, -ness etc.
 * similar strategy to step2. */
static void
step3() {
  switch (b[k]) {
    case 'e':
      if (ends("\05" "icate")) {
        replace("\02" "ic");
        break;
      }

      if (ends("\05" "ative")) {
        replace("\00" "");
        break;
      }
      if (ends("\05" "alize")) {
        replace("\02" "al");
        break;
      }

      break;
    case 'i':
      if (ends("\05" "iciti")) {
        replace("\02" "ic");
        break;
      }

      break;
    case 'l':
      if (ends("\04" "ical")) {
        replace("\02" "ic");
        break;
      }

      if (ends("\03" "ful")) {
        replace("\00" "");
        break;
      }

      break;
    case 's':
      if (ends("\04" "ness")) {
        replace("\00" "");
        break;
      }

      break;
  }
}

/* `step4()` takes off -ant, -ence etc., in
 * context <c>vcvc<v>. */
static void
step4() {
  switch (b[k - 1]) {
    case 'a':
      if (ends("\02" "al")) {
        break;
      }
      return;
    case 'c':
      if (ends("\04" "ance")) {
        break;
      }

      if (ends("\04" "ence")) {
        break;
      }

      return;
    case 'e':
      if (ends("\02" "er")) {
        break;
      }

      return;
    case 'i':
      if (ends("\02" "ic")) {
        break;
      }
      return;
    case 'l':
      if (ends("\04" "able")) {
        break;
      }

      if (ends("\04" "ible")) {
        break;
      }

      return;
    case 'n':
      if (ends("\03" "ant")) {
        break;
      }

      if (ends("\05" "ement")) {
        break;
      }

      if (ends("\04" "ment")) {
        break;
      }

      if (ends("\03" "ent")) {
        break;
      }

      return;
    case 'o':
      if (ends("\03" "ion") && j >= k0 && (b[j] == 's' ||
b[j] == 't')) {
        break;
      }

      /* takes care of -ous */
      if (ends("\02" "ou")) {
        break;
      }

      return;
    case 's':
      if (ends("\03" "ism")) {
```

```
      break;
    }

    return;
  case 't':
    if (ends("\03" "ate")) {
      break;
    }

    if (ends("\03" "iti")) {
      break;
    }

    return;
  case 'u':
    if (ends("\03" "ous")) {
      break;
    }

    return;
  case 'v':
    if (ends("\03" "ive")) {
        break;
      }

      return;
    case 'z':
      if (ends("\03" "ize")) {
        break;
      }

      return;
    default:
      return;
  }

  if (getMeasure() > 1) {
    k = j;
  }
}

/* `step5()` removes a final `-e` if `getMeasure()` is
 * greater than `1`, and changes `-ll` to `-l` if
 * `getMeasure()` is greater than `1`. */
```

```
static void
step5() {
  int a;

  j = k;

  if (b[k] == 'e') {
    a = getMeasure();

    if (a > 1 || (a == 1 && !cvc(k - 1))) {
      k--;
    }
  }

  if (b[k] == 'l' && isDoubleConsonant(k) && getMeasure() >
1) {
    k--;
  }
}

/* In `stem(p, i, j)`, `p` is a `char` pointer, and the
 * string to be stemmed is from `p[i]` to
 * `p[j]` (inclusive).
 *
 * Typically, `i` is zero and `j` is the offset to the
 * last character of a string, `(p[j + 1] == '\0')`.
 * The stemmer adjusts the characters `p[i]` ... `p[j]`
 * and returns the new end-point of the string, `k`.
 *
 * Stemming never increases word length, so `i <= k <= j`.
 *
 * To turn the stemmer into a module, declare 'stem' as
 * extern, and delete the remainder of this file. */
int
stem(char *p, int index, int position) {
  /* Copy the parameters into statics. */
  b = p;
  k = position;
  k0 = index;

  if (k <= k0 + 1) {
    return k; /* --DEPARTURE-- */
  }
  /* With this line, strings of length 1 or 2 don't
   * go through the stemming process, although no
   * mention is made of this in the published
   * algorithm. Remove the line to match the published
   * algorithm. */
  step1ab();

  if (k > k0) {
    step1c();
    step2();
    step3();
    step4();
    step5();
  }

  return k;
}


typedef struct _title_node{
  double cnt;
  string title;
  float persent;
  struct _title_node *next;
}title_node;
/*
struct store the frequency of word in each article
the link table link all articles that word appears
*/
typedef struct _word_cnt{
  title_node *head;
  title_node *tail;
}*word_cnt;
/*
include head and tail, standing for the beginning and the
end
*/
word_cnt create(){
  word_cnt block = (word_cnt)malloc(sizeof(struct
_word_cnt));
  block->head = block->tail = NULL;
  return block;
}

void append(word_cnt block,string name){
  if(block->head){                                //
不是空的
    if(block->tail->title != name){
    // if the article doesn't appear in the link
      title_node *temp = new title_node;
      temp->title = name;
      temp->persent = -1;
      temp->cnt = 0;
      temp->next = NULL;
      block->tail->next = temp;
      block->tail = temp;
    }
  } else {                                          //
是空的
    block->head = new title_node;
    block->head->title = name;
    block->head->persent = -1;
    block->head->next = NULL;
    block->tail = block->head;
  }
}
```

```
string change_back_title(string title){
  for(int i = 0;i < title.length(); i++){
    if(title[i] == '_'){
      title[i] = ' ';
    }
  }
  return title;
}

// turn string into Lowercase
string lowerString(string s)
{
    transform(s.begin(), s.end(), s.begin(),
                   [](unsigned char c){ return
tolower(c); });
    return s;
}

int main(){
  int len = 0;
    ifstream source("./result/interesting_word.txt");
    ofstream fw("./result/read_result.txt");
    map<string, word_cnt> word_bag;
  int count = 0;
    if(source){
        string get_word;
    string word;
        while(source >> get_word){
      if(get_word == "#"){
        count = 0;
      } else {
        if(count == 0){
          word = get_word;
          char *_c = (char*)word.data();
          int len = stem(_c, 0, word.length()-1);
          _c[len+1] = 0;
          word = _c;
          word_bag[word] = create();
          len++;
        } else {
          if(count == 1){
          } else {
            if(count % 2 == 0){   //title
            get_word = change_back_title(get_word);
```

```
              append(word_bag[word],get_word);
            } else {
              word_bag[word]->tail->persent =
stof(get_word);
            }
          }

        }
        count++;
      }
        }
    } else {
        cout << "No Such File With Titles,Sir";
    }

    map<string, word_cnt>::iterator pointer =
word_bag.begin( );
    int aaa = 0;
    while(pointer!=word_bag.end()){
    aaa++;
    fw << pointer->first;
    title_node *ptr = pointer->second->head;
```

```cpp
    while(ptr){
      fw << " " << ptr->title << " " << ptr->persent;
      ptr = ptr->next;
    }
    fw << "\n";
    pointer++;
  }
  cout << "I'm a mini search engine~" << endl;
  cout << "Please enter your query here and end with \' #\n(hint: here is a space before #,and no symbols should be included in your phrase)\':" << endl;
  while(1){
    string _str;
    map<string, int>Select_Result;
    int max_num = 0;
    int second_max_num = 0;
    cin >>_str;
    int cnt, word_len;
    cnt = word_len = 0;
    while(_str !="#"){
      _str = lowerString(_str);
      // turn the string into Lowercase
```

```
char *_c = (char*)_str.data();
// stem() is available only when
// it was Char* rather than string
int len = stem(_c, 0, _str.length()-1);
_c[len+1] = 0;
_str = _c;
if(word_bag.find(_str)!=word_bag.end()){
// if the word is in the interesting words
  word_len++;
  title_node *tmp = word_bag[_str]->head;
  while(tmp){
    Select_Result[tmp->title]++;
    if(max_num < Select_Result[tmp->title]){
      second_max_num = max_num;
      max_num = Select_Result[tmp->title];
    }
    // find the max and the second times that
    // words appear in each article
    // so why we choose the max and the second?
    // the search result is dependent on what stop
    // words we choose.
    // if we just take the max, it sometimes can't
```

```
      // return the true answer, also, if we take the
      // third, the fourth... the number of results
      // is too large.
      tmp = tmp->next;
    }
  }
  cin >>_str;
}
// here, it prints the most possible article out
map<string, int>::iterator it = Select_Result.begin();
int iswrite = 0;
while(it!=Select_Result.end()){
  if(it->second==max_num){
    if(iswrite==0){
      cout << "You are most likely looking for:" << endl;
      iswrite = 1;
    }
    cout << it->first << endl;


  }
  it++;
}
```

```
    cout <<endl;

    // and here, it prints the secondly possible article out

    if(second_max_num){

      int iswrite1 = 0;

      map<string, int>::iterator it_ =

Select_Result.begin();

      while(it_!=Select_Result.end()){

        if(it_->second==second_max_num){

          if(iswrite1==0){

            cout << "You are also likely looking for:" <<

endl;

            iswrite1 = 1;

          }

          cout << it_->first << endl;


        }

        it_++;

      }

    }

    // we just check if we prints the max number to

    // see whether the result is empty.

    if(!iswrite){
      cout << "Your query is too vague to get results" <<
endl;
    }
    cout << "Wanna search anything else? Enter Y for
continuing and N for stopping (^ ^)" << endl;
    string flag;
    cin >> flag;
    if( flag =="Y" || flag =="y"){
      cout << "Please enter your query~" << endl;
      continue;
    } else {
      break;
    }
  }
  cout << "Thanks for using,BYE~";
  return 0;
}
```

Declaration

I hereby declare that all the work done in this project titled " Mini Search Engine" is of my independent effort.