Roll your own mini search engine

September 29, 2024

2024-2025 Fall & Winter Semester

Contents

Chapter1: Introduction

Tasks:
(1) Run a word count over the Shakespeare set and try to identify the stop words (also called the noisy words) –How and where do you draw the line between "interesting" and "noisy" words?
(2) Create your 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 your 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.

Chapter2: Programming

Word counting

The word counting's part, we use the thresholding method to identy the noisy words and interesting words. To be concrete, we define that the term whose word frequency is over the thresholding is the noisy word, and the left part is the interesting one. We design a function called print_highfrequency_word, which will print the noisy word out, and we can check whether some words which ought to be interesting are misclassified.

Index generation

To generate the index, firstly, we seperate the words from the whole passage, and for each word, we use the word stemmer to get the stem.Also, we will check if it is a stop word. After that,For each term, if the term has already in the posting list, just insert it; If not, then create a new node. For the data scale is small, here we use the hash table to store the listing node, which contains the information of the word's location. The hash function we use is to regard the word as a base-31 number and get its value in decimal.(In the source code, we rotate the word and get its value) It's obviously a 1-1 correspondence.

Query processing

To realize the searching function, we firstly read the query in, and seperate the word just like what we do in the word counting part. Then we store these term in a searching list. Then we sort the searching list, because the less frequency word is more important in our searching. Then we define different thresholding for different length of the query and using the selected term to finish our searching. And the searching is easy bacause we just calculate the value of hash function and use the finding function to get the listing node. For a setence query, we have to AND the results.

Pseudo code

```
print_posting_list(Hash_Table, word_count, max_hash_value)
        for( i: from 0 to HASH_TABLE_SIZE )
                temp_Node = Hash_Table[i]
                if temp_Node is not NULL
                temp_List_Node = temp_Node->posting_list
                print word, hash_value, frequency
                while(temp_List_Node is not NULL)
                        Print index
                        temp_List_Node = temp_List_Node->next
        Print word_count
        Print max_hash_value
End


print_highfrequency_word(Hash_Table)
        initialize thershold and word_number
        input thershold
        for( i: from 0 to HASH_TABLE_SIZE)
                temp_Node = Hash_Table[i]
                if temp_Node is not NULL and frequency > thersnold
                Print word, frequency
                word_number++
        print word_number
End


read_file(Hash_Table, word_count, max_hash_value)
    Initialize
    for( i: from 0 to TXT_NUMBER)
        choose_file(filename, type_number, article_number)
        open file
        Initialize act_number, scene_number, line_number
        read information from script line by line
            line_number ++
            initialize orig_start_place and t
            while(don't finish reading)
                copy information from orig to temp word by word
                if temp is empty, go back and continue to read
```

```
                determine whether an update of act_number or scene_number is needed
                otherwise
                        word stemming
                    if temp is stop words, go back and continue to read
                    hash_value = HASH(temp)
                    if the word is in the dictionary
                        insert_index
                         otherwise
                        make new word node
                        insert_index
                        renew word_count and max_hash_value
                close file
End


find_word_in_hashtable(word, hash_value, Hash_Table)
        while Hash_Table[hash_value] is not NULL
                If Hash_Table[hash_value]->WORD is equal to word
                find it, return 1
                hash_value += MAX_HASH_VALUE
        not find, return 0
End

insert_Posting_List(Hash_Table, hash_value, type, article, act, scene, line)
        create a new node
        set indices of new node
        word_node = Hash_Table[hash_value]
        word_node->frequency++
        if word_node->posting_list is NULL
                word_node->posting_list = new_node
                word_node->tail_list_node = new_node
         otherwise
                word_node->tail_list_node->next = new_node
                word_node->tail_list_node = new_node
End


choose_file(filename, type_number, article_number)
        static flag = 0
```

```
        flag++
        choose file according to flag, and renew type_number and article_number
End


search_word(Hash_Table)
        initialize operation[1000]
        input the word to search
        while input operation
                if operation is equal to "--exit--"
                end
          else if  operation is equal to "--clear--"
                clear the terminal
                print related information
                else
                Search_function(Hash_Table, operation)
                print related information
        empty operation[1000]
End


Search_function(Hash_Table, operation)
        initialize ope_start_place = 0, t = 0
        initialize temp[100]
        initialize word
        Search_Word_List Search_List = NULL
        read the information in operation word by word
        word stemming
        make_Searching_List(Hash_Table, temp, Search_List)
        if Search_List is NULL
                print "Can't find the word. Please try again."
                end
        Search_List_Sort(Search_List)
        Query_Threshold(Search_List)
        Search_Result_Printing(Search_List)
        clear_Search_List(Search_List)
End
```

```
make_Searching_List(Hash_Table, word, Search_List)
        hash_value = HASH(word)
        if find_word_in_hashtable(word, &hash_value, Hash_Table) == 1
                create new_node
                if Search_List is NULL
                new_node->prev = NULL
                create Search_List
                 set head and tail
                return Search_List
                else
                new_node->prev = Search_List->tail
                add new_node to the tail of Search_List
                return Search_List
        else    return Search_List

End


Search_List_Sort(Search_List)
        if Search_List is empty or have only one node
                return Search_List
         save head_Node and tail_Node
        start_Node = head_Node->next
        while start_Node is not NULL
                move_Node = start_Node
//move_Node is the node where we start bubble sort
                while move_Node->prev is not NULL
                temp_Node = move_Node->prev
                if move_Node->frequency < temp_Node->frequency
                        renew head_Node, tail_Node and start_Node
                        swap two node
        start_Node = start_Node->next
        renew the head and tail of Search_List
        return Search_List
End


Query_Threshold(Search_List)
        if Search_List is NULL
```

```
            return NULL
        calculate total_frequency and total_count
        if total_count <= 3
         just return Search_List
        else if total_count <= 8
        set threshold = 0.8
        else
        set threshold = 0.7
        temp_Node = Search_List->head
        ans = temp_Node->frequency / total_frequency
        while ans <= threshold
        temp_Node = temp_Node->next
        renew ans
        clear excess nodes of Search_List
        return Search_List
End


Search_Result_Printing(Search_List)
        if Search_List is NULL
                error
        initialize first_Node = Search_List->head, Second_Node = first_Node->next
        initialize Search_flag and Result_flag
        if Second_Node is NULL
         print_single_posting_list(first_Node)
        return
        while target_index is not NULL
        search each temp_Node and see if it is correct
                print answer
        else    print not find
End
```

Chapter3: Testing

Testing for the inverted index

abandon, a normal word to test if the inverted index can work correctly.
Brevity is the soul of wit, a classic quote of Shakesphere, to test the sentence searching function.
Internet, to test if the searching engine can identify the irrelevant words.
a, stop words.
To be or not to be, a classic quote of Shakesphere containing lots of stop words.

Testing for the query processing

Love looks not with the eyes, but with the mind; and therefore is winged Cupid painted blind, a long and classic quote of Shakesphere to test the influence of thresholding.

Table 1: The query and the results

Query	**Searching Results**
abandon	(1,1,1,1,20)(1,2,2,1,880)(1,2,5,1,3168) (1,2,5,1,3171)...
Brevity is the soul of wit	(3,3,2,2,1535)
Internet	Can't find the word. Please try again.
a	Can't find the word. Please try again.
To be or not to be	Can't find the word. Please try again.
Love looks not with the eyes, but with the mind; and therefore is winged Cupid painted blind	(1,9,1,1,304) (1,14,3,2,2246)

Chapter4: Analysis

Pros

From the testing part, we can easily find that this project realize the fundemental function of a search engine.
First, it generates the inverted index through HASH properly.
Second, with the help of word stemmer, it uses the method of thresholding to identify the noisy words and stores the important terms in its posting list.
Third, the query program successfully returns the location of most of the words and sentences in the testing parts.

Cons

From the testing data, to be or not to be, which is one of the most classic quotes in Shakesphere's works,we can find that the classic quotes'query returns none.
So this search engine can't handle some important information consisting of all stop words.
Also, since we use the method of hashing when store the data of the listing node, we just can't handle the large scale data.One improvement may be to store those datas in a B+ tree.

Appendix: Cpp source code

Listing 1: Insert index

```
#include <iostream>
#include <fstream>
#include <string>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <malloc.h>
#include "porter2_stemmer.h"
#include "insert_index.h"

void print_posting_list(Table_Node *Hash_Table ,int *word_count ,int *max_hash_value)
{
    for(int i = 0 ; i < HASH_TABLE_SIZE ; ++ i)                    //遍历哈希表
    {
        Table_Node temp_Node = Hash_Table[i];
        if(temp_Node != NULL)
        {
        List_Node temp_List_Node = temp_Node -> posting_list;
        printf("Word: %s , hash_value: %d , frequency: %d" , temp_Node -> WORD ,
                    i , temp_Node -> frequency);
            while(temp_List_Node != NULL)
            {
                printf("-> (%d,%d,%d,%d,%d) " , temp_List_Node -> index_type ,
                temp_List_Node -> index_article , temp_List_Node -> index_act ,
                temp_List_Node -> index_scene , temp_List_Node -> index_line);
                temp_List_Node = temp_List_Node -> next;
            }
            printf("\n");
        }
    }
    printf("The number of words is: %d\n" , *word_count);
    printf("The maximum hash value is: %d\n" , *max_hash_value);
    return ;
}
```

```
void print_highfrequency_word(Table_Node *Hash_Table)      //打印高频词(测试用)
{
    int thershold , word_number = 0;
    printf("Please input the frequency threshold: ");
    scanf("%d" , &thershold);
    for(int i = 0 ; i < HASH_TABLE_SIZE ; ++ i)
    {
        Table_Node temp_Node = Hash_Table[i];
        if(temp_Node != NULL && temp_Node -> frequency >= thershold)
    {
    printf("Word: %s , frequency: %d \n" , temp_Node -> WORD , temp_Node -> frequency)
    ++ word_number;
    }
    }
    printf("Total number of words: %d\n" , word_number);
    return ;
}

void read_file(Table_Node *Hash_Table , int *word_count , int *max_hash_value)
//读入文件
{
    string word;
    char orig[2500] , temp[100] , filename[100];
    int t , orig_start_place , hash_value ,act_flag = 0 , scene_flag = 0 ,
    type_number = 0 , article_number = 0 , act_number = 0 ,
     scene_number = 0 , line_number = 0;
    ifstream script;
    for(int i = 1 ; i <= TXT_NUMBER ; ++ i)
    {
        choose_file(filename , &type_number , &article_number);
        //依次选择读入的文件, 同时更新type和article
        script.open(filename);
        //读文件
        act_number = scene_number = line_number = 0;
        while (script.getline(orig, 2500))
        //逐行读入
        {
            ++ line_number;
            t = orig_start_place = 0;
```

```
        while(orig[orig_start_place] != '\0')
         //逐字读入
        {
while(orig[orig_start_place + t] != '␣' && orig[orig_start_place + t] != '\r'
&& orig[orig_start_place + t] != '\n' && orig[orig_start_place + t] != '\0'
&& orig[orig_start_place + t] != '.' && orig[orig_start_place + t] != ','
&& orig[orig_start_place + t] != ';' && orig[orig_start_place + t] != ':'
&& orig[orig_start_place + t] != '!' && orig[orig_start_place + t] != '?'
&& orig[orig_start_place + t] != '–' && orig[orig_start_place + t] != '\'')
                  //将该单词逐字母复制给temp
                {
                    temp[t] = orig[orig_start_place + t];
                    t++;
                }
            orig_start_place += (t + 1);
            temp[t] = '\0';
            t = 0;
            //修改位置，便于下一次读写
            if(strcmp(temp , "") == 0)
            //若遇到空格或换行符则跳过
            {
                continue;
            }
            else if(strcmp(temp, "ACT") == 0)
            //若遇到ACT或SCENE则判断是否变成新的一章
            {
                act_flag = 1;
            }
            else if(strcmp(temp, "SCENE") == 0)
            {
                scene_flag = 1;
            }
            else if(isRomanNumDot(temp) || isRomanNumNodot(temp))
                 //若是，则更新act或scene的编号
                    {
                        if(act_flag == 1)
                        {
                            ++ act_number;
                            scene_number = 0;
```

```
                act_flag = 0;
            }
            else if(scene_flag == 1)
            {
                ++ scene_number;
                scene_flag = 0;
            }
        }
    else
    {
        act_flag = 0;
        scene_flag = 0;     //复原标志位
        word = temp;        //将temp转换为string类型
        Porter2Stemmer::trim(word);
        Porter2Stemmer::stem(word);
        strcpy(temp, word.c_str()); //将string类型转换成char类型，方便操作
        if(strlen(temp) == 1 || isStopWords(temp))//若是停用词则跳过
        {
            continue;
        }
        hash_value = HASH(temp);    //计算hash值
        if(find_word_in_hashtable(temp , &hash_value , Hash_Table))
        //若单词在Dictionary中，插入新的索引
        {
            insert_Posting_List(Hash_Table , hash_value , type_number ,
              article_number , act_number , scene_number , line_number);
//插入索引
        }
        else
        {
            make_table_node(Hash_Table , hash_value , temp);
           //建立新的单词节点
          insert_Posting_List(Hash_Table , hash_value , type_number ,
          article_number , act_number , scene_number , line_number);
//插入索引
          ++ *word_count;
         if(*max_hash_value < hash_value)      *max_hash_value = hash_value
        }
    }
```

```
            }
        }
        script.close();
    }
    return ;
}

int HASH(char *word)        //hash函数
{
    ULL p = 31 , hash_value = 0 , len = strlen(word);
    for(int i = 0 ; i <= len ; ++ i)
    {
        hash_value = hash_value * p + word[i];
    }
    return (int)(hash_value % (MAX_HASH_VALUE + 1));
}

int find_word_in_hashtable(char *word , int *hash_value , Table_Node *Hash_Table)
  //查找单词是否在Dictionary中
{
    while(Hash_Table[*hash_value] != NULL)
     //若没找到且还可以继续找，则一直搜索下去
    {
        if(strcmp(Hash_Table[*hash_value] -> WORD , word) == 0)
       //找到了
        {
            return 1;
        }
        *hash_value += MAX_HASH_VALUE;
      //继续找下一个位置
    }
    return 0;
}

void make_table_node(Table_Node * Hash_Table , int hash_value , char *word)
//建立新的单词节点
{
    Table_Node new_node = (Table_Node)malloc(sizeof(struct HashTableNode));
    new_node -> WORD = (char*)malloc(sizeof(char) * (strlen(word) + 1));
```

```
    strcpy(new_node -> WORD , word);
    new_node -> frequency = 0;
    new_node -> posting_list = NULL;
    new_node -> tail_list_node = NULL;
    Hash_Table[hash_value] = new_node;
    return ;
}

void insert_Posting_List(Table_Node *Hash_Table , int hash_value , int type ,
 int article , int act , int scene , int line) //插入新的索引
{
    List_Node new_node = (List_Node)malloc(sizeof(struct PostingList_Node));
    new_node -> index_type = type;
    new_node -> index_article = article;
    new_node -> index_act = act;
    new_node -> index_scene = scene;
    new_node -> index_line = line;
    new_node -> next = NULL;
    Table_Node word_node = Hash_Table[hash_value];              //找到单词节点
    ++ word_node -> frequency;                                  //更新词频
    if(word_node -> posting_list == NULL)                       //若链表为空
    {
        word_node -> posting_list = new_node;
        word_node -> tail_list_node = new_node;
    }
    else
    //否则直接改变尾节点
    {
        word_node -> tail_list_node -> next = new_node;
        word_node -> tail_list_node = new_node;
    }
    return ;
}

void clear_hash_table(Table_Node * Hash_Table)
//清空哈希表
{
    Table_Node temp_Node;
    for(int i = 0 ; i < HASH_TABLE_SIZE ; ++ i)
```

```
    //遍历哈希表
    {
        temp_Node = Hash_Table[i];
        if(temp_Node != NULL)
        //若节点不为空
        {
            List_Node temp_List_Node = temp_Node -> posting_list , next_List_Node;
            do
            {
                next_List_Node = temp_List_Node -> next;
                free(temp_List_Node);
                temp_List_Node = next_List_Node;
            } while (temp_List_Node != NULL);
             //释放Posting List
            free(temp_Node -> WORD);
            free(temp_Node);
           //释放单词节点
        }
    }
    return ;
}

void choose_file(char *filename , int *type_number , int *article_number)
         //选择文件
{
    static int flag = 0;
    ++ flag;
    switch (flag)
    {
        case 1:
            strcpy(filename, ".\\txt_source\\script1.txt");
            *type_number = 1;
            *article_number = 1;
            break;
        case 2:
            strcpy(filename, ".\\txt_source\\script2.txt");
            *type_number = 1;
            *article_number = 2;
            break;
```

```
case 3:
    strcpy(filename, ".\\txt_source\\script3.txt");
    *type_number = 1;
    *article_number = 3;
    break;
case 4:
    strcpy(filename, ".\\txt_source\\script4.txt");
    *type_number = 1;
    *article_number = 4;
    break;
case 5:
    strcpy(filename, ".\\txt_source\\script5.txt");
    *type_number = 1;
    *article_number = 5;
    break;
case 6:
    strcpy(filename, ".\\txt_source\\script6.txt");
    *type_number = 1;
    *article_number = 6;
    break;
case 7:
    strcpy(filename, ".\\txt_source\\script7.txt");
    *type_number = 1;
    *article_number = 7;
    break;
case 8:
    strcpy(filename, ".\\txt_source\\script8.txt");
    *type_number = 1;
    *article_number = 8;
    break;
case 9:
    strcpy(filename, ".\\txt_source\\script9.txt");
    *type_number = 1;
    *article_number = 9;
    break;
case 10:
    strcpy(filename, ".\\txt_source\\script10.txt");
    *type_number = 1;
    *article_number = 10;
```

```
        break;
    case 11:
        strcpy(filename, ".\\txt_source\\script11.txt");
        *type_number = 1;
        *article_number = 11;
        break;
    case 12:
        strcpy(filename, ".\\txt_source\\script12.txt");
        *type_number = 1;
        *article_number = 12;
        break;
    case 13:
        strcpy(filename, ".\\txt_source\\script13.txt");
        *type_number = 1;
        *article_number = 13;
        break;
    case 14:
        strcpy(filename, ".\\txt_source\\script14.txt");
        *type_number = 1;
        *article_number = 14;
        break;
    case 15:
        strcpy(filename, ".\\txt_source\\script15.txt");
        *type_number = 1;
        *article_number = 15;
        break;
    case 16:
        strcpy(filename, ".\\txt_source\\script16.txt");
        *type_number = 1;
        *article_number = 16;
        break;
    case 17:
        strcpy(filename, ".\\txt_source\\script17.txt");
        *type_number = 1;
        *article_number = 17;
        break;
    case 18:
        strcpy(filename, ".\\txt_source\\script18.txt");
        *type_number = 2;
```

```
        *article_number = 1;
        break;
    case 19:
        strcpy(filename, ".\\txt_source\\script19.txt");
        *type_number = 2;
        *article_number = 2;
        break;
    case 20:
        strcpy(filename, ".\\txt_source\\script20.txt");
        *type_number = 2;
        *article_number = 3;
        break;
    case 21:
        strcpy(filename, ".\\txt_source\\script21.txt");
        *type_number = 2;
        *article_number = 4;
        break;
    case 22:
        strcpy(filename, ".\\txt_source\\script22.txt");
        *type_number = 2;
        *article_number = 5;
        break;
    case 23:
        strcpy(filename, ".\\txt_source\\script23.txt");
        *type_number = 2;
        *article_number = 6;
        break;
    case 24:
        strcpy(filename, ".\\txt_source\\script24.txt");
        *type_number = 2;
        *article_number = 7;
        break;
    case 25:
        strcpy(filename, ".\\txt_source\\script25.txt");
        *type_number = 2;
        *article_number = 8;
        break;
    case 26:
        strcpy(filename, ".\\txt_source\\script26.txt");
```

```
        *type_number = 2;
        *article_number = 9;
        break;
    case 27:
        strcpy(filename, ".\\txt_source\\script27.txt");
        *type_number = 2;
        *article_number = 10;
        break;
    case 28:
        strcpy(filename, ".\\txt_source\\script28.txt");
        *type_number = 3;
        *article_number = 1;
        break;
    case 29:
        strcpy(filename, ".\\txt_source\\script29.txt");
        *type_number = 3;
        *article_number = 2;
        break;
    case 30:
        strcpy(filename, ".\\txt_source\\script30.txt");
        *type_number = 3;
        *article_number = 3;
        break;
    case 31:
        strcpy(filename, ".\\txt_source\\script31.txt");
        *type_number = 3;
        *article_number = 4;
        break;
    case 32:
        strcpy(filename, ".\\txt_source\\script32.txt");
        *type_number = 3;
        *article_number = 5;
        break;
    case 33:
        strcpy(filename, ".\\txt_source\\script33.txt");
        *type_number = 3;
        *article_number = 6;
        break;
    case 34:
```

```
        strcpy(filename, ".\\txt_source\\script34.txt");
        *type_number = 3;
        *article_number = 7;
        break;
    case 35:
        strcpy(filename, ".\\txt_source\\script35.txt");
        *type_number = 3;
        *article_number = 8;
        break;
    case 36:
        strcpy(filename, ".\\txt_source\\script36.txt");
        *type_number = 3;
        *article_number = 9;
        break;
    case 37:
        strcpy(filename, ".\\txt_source\\script37.txt");
        *type_number = 3;
        *article_number = 10;
        break;
    case 38:
        strcpy(filename, ".\\txt_source\\script38.txt");
        *type_number = 4;
        *article_number = 1;
        break;
    case 39:
        strcpy(filename, ".\\txt_source\\script39.txt");
        *type_number = 4;
        *article_number = 2;
        break;
    case 40:
        strcpy(filename, ".\\txt_source\\script40.txt");
        *type_number = 4;
        *article_number = 3;
        break;
    case 41:
        strcpy(filename, ".\\txt_source\\script41.txt");
        *type_number = 4;
        *article_number = 4;
        break;
```

```
        case 42:
            strcpy(filename, ".\\txt_source\\script42.txt");
            *type_number = 4;
            *article_number = 5;
            break;
        default:
            return ;
    }
    return ;
}
```

Listing 2: Search words

```
#include <iostream>
#include <fstream>
#include <string>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <malloc.h>
#include "porter2_stemmer.h"
#include "search_word.h"

using namespace std;

void search_word(Table_Node *Hash_Table) //搜索功能主函数
{
    printf("Finish reading the file!\n");
    char operation[1000];
    memset(operation, '\0', 1000);
    printf("Here is the format of the search result:(type,article,act,scene,line)\n");
    printf("If you want to exit the program, please enter the word '--exit--'.\n");
    printf("If you want to clear the terminal, please enter the word '--clear--'.\n");
    printf("Please enter the word you want to search:\n");
    while(cin.getline(operation, 1000))
    {
    if(strcmp(operation, "--exit--") == 0)
    {
        return ;
    }
```

```
    else if (strcmp(operation, "--clear--") == 0)
    {
    system("clear");
    printf("Here is the format of the search result:(type,article,act,scene,line)\n");
    printf("If you want to exit the program, please enter the word '--exit--'.\n");
    printf("If you want to clear the terminal, please enter the word '--clear--'.\n");
    printf("Please enter the word you want to search:\n");
    }
    else
    {
    printf("\n");
    Search_function(Hash_Table , operation);
    printf("\n");
    printf("Here is the format of the search result:(type,article,act,scene,line)\n");
    printf("If you want to exit the program, please enter the word '--exit--'.\n");
    printf("If you want to clear the terminal, please enter the word '--clear--'.\n");
    printf("Please enter the word you want to search:\n");
    }
        memset(operation, '\0', 1000);
    }
    return ;
}

void Search_function(Table_Node * Hash_Table , char * operation) //搜索词句
{
    int ope_start_place = 0, t = 0;
    char temp[100];
    string word;
    Search_Word_List Search_List = NULL;
    while(operation[ope_start_place] != '\0')
    {
    while(operation[ope_start_place + t] != ' ' && operation[ope_start_place + t] != '
    && operation[ope_start_place + t] != '\r' && operation[ope_start_place + t] != '\n
        {
            temp[t] = operation[ope_start_place + t];
            t++;
        }
        ope_start_place += (t + 1);
        temp[t] = '\0';
```

```
            t = 0;
            word = temp;
            Porter2Stemmer::trim(word);
            Porter2Stemmer::stem(word);
            strcpy(temp, word.c_str());
            Search_List = make_Searching_List(Hash_Table , temp , Search_List);
            if(Search_List == NULL)
            {
                printf("Can't find the word. Please try again.\n");
                return ;
            }
        }
        Search_List = Search_List_Sort(Search_List);
        Search_List = Query_Threshold(Search_List);
        Search_Result_Printing(Search_List);
        clear_Search_List(Search_List);
        return ;
}

Search_Word_List make_Searching_List(Table_Node *Hash_Table ,
char *word , Search_Word_List Search_List)//创建搜索链表
{
        int hash_value = HASH(word);
        if(find_word_in_hashtable(word , &hash_value , Hash_Table))
         //若单词在Dictionary中
        {
            Search_Node new_node = (Search_Node)malloc(sizeof(struct Node_for_Search));
            //创建新节点
            strcpy(new_node -> word , word);
            new_node -> frequency = Hash_Table[hash_value] -> frequency;
            new_node -> first_posting_list = Hash_Table[hash_value] -> posting_list;
            new_node -> last_posting_list = Hash_Table[hash_value] -> tail_list_node;
            new_node -> next = NULL;
            if(Search_List == NULL)
            {
                new_node -> prev = NULL;       //前驱节点也为空

                Search_List = (Search_Word_List)malloc(sizeof(struct Searching_Word_List))
                Search_List -> head = new_node;
```

```
            Search_List -> tail = new_node;
        }
        else
        {
            new_node -> prev = Search_List -> tail;      //前驱节点为尾节点

            Search_List -> tail -> next = new_node;
            Search_List -> tail = new_node;
        }
        return Search_List;
    }
    else     return Search_List;
}

Search_Word_List Search_List_Sort(Search_Word_List Search_List)
  //对搜索链表排序
{
    if(Search_List == NULL || Search_List -> head == Search_List -> tail)
     //若链表为空或只有一个节点
    {
        return Search_List;
    }
    Search_Node head_Node = Search_List -> head , tail_Node = Search_List -> tail;
    //保存头尾节点，方便操作
    Search_Node start_Node = head_Node -> next , move_Node , temp_Node;
    //排序开始节点
    while(start_Node != NULL)
    {
        move_Node = start_Node;    //从start_Node节点开始进行冒泡排序
        while(move_Node ->prev != NULL)
        {
            temp_Node = move_Node -> prev;  //对前驱节点进行比较
            if(move_Node -> frequency < temp_Node -> frequency)
            {
                if(move_Node == start_Node)
                {
                    start_Node = temp_Node;
                }
                if(move_Node == tail_Node)
```

```
                {
                    tail_Node = temp_Node;
                }
                else if(temp_Node == head_Node)
                 //遇到特殊情况，先将start_Node,tail_Node或者head_Node更新
                {
                    head_Node = move_Node;
                }
                move_Node -> prev = temp_Node -> prev;
                temp_Node -> prev = move_Node;
                temp_Node -> next = move_Node -> next;
                move_Node -> next = temp_Node;
            }
            else    break;
        }
        start_Node = start_Node -> next;
    }
    Search_List -> head = head_Node;
    Search_List -> tail = tail_Node;  //更新头尾节点
    return Search_List;
}

Search_Word_List Query_Threshold(Search_Word_List Search_List) //根据阈值进行筛选
{
    if(Search_List == NULL)
    {
        return NULL;
    }
    int all_frequency = 0 , num_word = 0 , temp_frequency = 0;
    float thershold = 0 , ans = 0;
    Search_Node temp_Node = Search_List -> head , next_Node;
    while(temp_Node != NULL)        //计算总词频和总词数
    {
        all_frequency += temp_Node -> frequency;
        temp_Node = temp_Node -> next;
        ++ num_word;
    }
    if(num_word <= 3)  //若总词数小于等于3，则不进行阈值筛选
    {
```

```
        return Search_List;
    }
    else if(num_word <= 8)      //若总词数小于等于8，则阈值为0.5
    {
        thershold = 0.8;
    }
    else
    {
        thershold = 0.7;
    }
    temp_Node = Search_List -> head;
    temp_frequency = temp_Node -> frequency;
    ans = (float)temp_frequency / (float)all_frequency;
    while(ans <= thershold)       //找到第一个大于阈值的节点
    {
        temp_Node = temp_Node -> next;
        temp_frequency += temp_Node -> frequency;
        ans = (float)temp_frequency / (float)all_frequency;
    }
    if(temp_Node == NULL)
    {
        printf("ERROR: in Query_Threshold function!\n");
        return NULL;
    }
    temp_Node -> prev -> next = NULL;
    Search_List -> tail = temp_Node -> prev;     //更新链表
    while(temp_Node != NULL)        //将多余的点全部删除
    {
        next_Node = temp_Node -> next;
        free(temp_Node);
        temp_Node = next_Node;
    }
    return Search_List;
}

void Search_Result_Printing(Search_Word_List Search_List) //结果输出函数
{
    if(Search_List == NULL)
    {
```

```
        printf("ERROR: in Search_Result_Printing function!\n");
        return ;
    }
    Search_Node first_Node = Search_List -> head
    , Second_Node = first_Node -> next , temp_Node;
    List_Node target_index = first_Node -> first_posting_list , temp_index;
    int Search_flag = 0 , Result_flag = 0;
    printf("The result of searching is:\n");
    if(Second_Node == NULL)
    {
        print_single_posting_list(first_Node);
        return;
    }
    while(target_index != NULL)
    {
        temp_Node = Second_Node;
        while(temp_Node != NULL)
        {
            Search_flag = 0;
            temp_index = temp_Node -> first_posting_list;
            while(temp_index != NULL)
            {
                if(Result_Compare(target_index , temp_index))
                {
                    Search_flag = 1;
                    break;
                }
                temp_index = temp_index -> next;
            }
            if(Search_flag == 0)     break;
            temp_Node = temp_Node -> next;
        }
        if(temp_Node == NULL && Search_flag == 1)
        {
            printf("(%d,%d,%d,%d,%d) " ,
             target_index -> index_type , target_index -> index_article
             , target_index -> index_act , target_index -> index_scene
             , target_index -> index_line);
            Result_flag = 1;
```

```
        }
        target_index = target_index -> next;
    }
    if(Result_flag == 0)
    {
        printf("Not find any result!\n");
    }
    else     printf("\n");
    return ;
}

int Result_Compare(List_Node node1 , List_Node node2)//结果比较函数
{
    if(node1 -> index_type != node2 -> index_type)
    {
        return 0;
    }
    if(node1 -> index_article != node2 -> index_article)
    {
        return 0;
    }
    if(node1 -> index_act != node2 -> index_act)
    {
        return 0;
    }
    if(node1 -> index_scene != node2 -> index_scene)
    {
        return 0;
    }
    if(node1 -> index_line != node2 -> index_line)
    {
        return 0;
    }
    return 1;
}

void print_single_posting_list(Search_Node word_index) //打印单个Posting List
{
    if(word_index == NULL)
```

```
    {
        printf("ERROR: in print_single_posting_list function!\n");
        return ;
    }
    List_Node temp_index = word_index -> first_posting_list;
    while(temp_index != NULL)
    {
        printf("(%d,%d,%d,%d,%d) " , temp_index -> index_type ,
        temp_index -> index_article , temp_index -> index_act ,
        temp_index -> index_scene , temp_index -> index_line);
        temp_index = temp_index -> next;
    }
    printf("\n");
    return ;
}

void clear_Search_List(Search_Word_List Search_List)//清空搜索链表
{
    Search_Node temp_Node = Search_List -> head , next_Node;
    while(temp_Node != NULL)
    {
        next_Node = temp_Node -> next;
        free(temp_Node);
        temp_Node = next_Node;
    }
    free(Search_List);
    return ;
}
```

Listing 3: main

```
#include <iostream>
#include <cstring>
#include "insert_index.h"
#include "search_word.h"

using namespace std;
Table_Node Hash_Table[HASH_TABLE_SIZE];                    //哈希表
int main()
{
    int word_count = 0;                                    //单词数量
```

```
    int max_hash_value = 0;                                    //最大哈希值

    printf("Welcome to the Dictionary Search System!\n");
    printf("The file is reading...\n");
    memset(Hash_Table, 0, sizeof(Table_Node) * HASH_TABLE_SIZE);
    read_file(Hash_Table , &word_count, &max_hash_value);
    //print_posting_list(Hash_Table , &word_count, &max_hash_value);
    //print_highfrequency_word(Hash_Table);
    search_word(Hash_Table);
    clear_hash_table(Hash_Table);                              //释放内存
    return 0;
}
```

Reference

Word stemmer function is from: https://github.com/smassung/porter2_stemmer

HASH function is from: https://blog.csdn.net/qq_40342400/article/details/127232662?ops_request_misc=&request_id=&biz_id=102&utm_term=%E5%AD%97%E7%AC%A6%E4%B8%B2%E6%9F%A5%E6%89%BE%20%E5%93%88%E5%B8%8C%E5%87%BD%E6%95%B0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-1-127232662.142^v100^pc_search_result_base1&spm=1018.2226.3001.4187