Advanced Data Structures and Algorithm Analysis

Laboratory Projects

P1: Roll Your Own Mini Search Engine

Group 1(Wang Haoyuan/Ren feiyang/Chen yihang)

Date: 2024/3/21

Chapter 1:Introduction

1. Problem description:

In this project, we need to design a **mini search engine** using the inverted file index, targetted at the wetsite:"The Complete Workss of William Shakespeare".

Our main tasks are:

1. Run a word count over the Shakespeare set and try to identify the stop words.
2. Create our inverted index over the Shakespeare set with word stemming.
3. Write a query program on top of our inverted file index, which will accept a user-specified word 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.

2. Content of this Report:

In this report, we will explain the structure of the program, the algorithm and method we use in implementing this program, list the testing results, and finally analysis the time/space complexity of the program.

3. The Preprocession of the Program: Self-designed Spider

To get all the content from the website, we design a spider to get the information from the html data.

In the spider, we use Requests & Beautifulsoup to decode the html text.

The spider's structure is as follows:

1. The first layer: get the address of all the books

```
# import......
def maindata():
    url = 'http://shakespeare.mit.edu/'
    response = requests.get(url, headers = headers)
```

```
    if (response.status_code != 200):
        print("request declined")#异常，有反爬装甲
    else:
        dstring = response.text
    soup = BeautifulSoup(dstring, 'lxml')
    passagesoup = soup.body
    tablesoup = passagesoup.find_all("table")
    targettable = tablesoup[1]
    targettr = targettable.find_all("tr")[1]#这里与网页本身有关，第一个tr是分类
    targeta = targettr.find_all("a")

    nameset = ["string"]
    nameset += [None] * 100
    f = open("firstaddress.txt", "w")
    for i in range(len(targeta)):
        f.write(url)
        f.write(targeta[i].attrs['href'])
        f.write("\n")
    f.write("#")
    f.close()
    for i in range(len(targeta)):
        nameset[i] = targeta[i].string
        nameset[i] = nameset[i].replace('\n', ' ').strip()
    articlenumber = len(targeta)
    namedata = open("namedata.txt", "w")
    for i in range(articlenumber):
        namedata.write(nameset[i])
        namedata.write('\n')
    namedata.close()
    print("the files are initialized succesfully.")
```

2. Set the bookspider function for the 37 books:

```
def bookspider():
    for i in range(37):#一共有37个可以直接爬的书
        content = open(listofname[i].strip() + ".txt", "w")
        content.write(listofname[i])
        url = listofaddress[i][:-11] + "full.html"
        print(url)
        response = requests.get(url, headers = headers)
        if (response.status_code != 200):
            print("request declined")#异常，有反爬装甲
        else:
            dstring = response.text
        wholesoup = BeautifulSoup(dstring, 'lxml')
        body_soup = wholesoup.body
        a_soup = body_soup.find_all(["a", "i"])#a后是剧本，i后是出场/入场
        num_of_line = len(a_soup)
        for j in range(2, num_of_line):
            content.write(a_soup[j].string)
            content.write('\n')
```

3. Because of the specialty of "The Sonnets", set the sonnetspider for this address:

```
def sonnetspider():
    index = 37
    content = open(listofname[index].strip() + ".txt", "w")
    content.write(listofname[index])
    url = listofaddress[index].strip()
    print(listofname[index])
    print(url)
    response = requests.get(url, headers = headers)
    if (response.status_code != 200):
        print("request declined")
    else:
        dstring = response.text
    sonnetsoup = BeautifulSoup(dstring, 'lxml')
    secondaddress = sonnetsoup.body.find_all("a")
    for i in range(1, len(secondaddress)):
        url = "http://shakespeare.mit.edu/Poetry/" + secondaddress[i].attrs['href']
        response = requests.get(url, headers = headers)
        if (response.status_code != 200):
            print("request declined")
        else:
            dstring = response.text
        partsoup = BeautifulSoup(dstring, 'lxml')
        partstring = partsoup.body.find_all("blockquote")
        print(secondaddress[i].attrs['href'][:-5])
        content.write(secondaddress[i].attrs['href'][:-5])
        content.write('\n')
        for j in range(len(partstring)):
            content.write(partstring[j].text)
```

4. Set the poemspider for other poems:

```
def poemspider():
    index = 38
    for i in range(index, len(listofname)):
        content = open(listofname[i].strip() + ".txt", "w")
        content.write(listofname[i])
        url = listofaddress[i].strip()
        print(url)
        response = requests.get(url, headers= headers)
        if (response.status_code != 200):
            print("request declined")
        else:
            dstring = response.text
        poetsoup = BeautifulSoup(dstring, 'lxml')
        poetcontent = poetsoup.body.find_all(['p', 'blockquote'])
        for j in range(len(poetcontent)):
            content.write(poetcontent[j].text)
```

You can also try to run `spider.py` to get the whole process of the spider.

note: the spider will put all the texts in the file where it is.

Chapter 2:Data Structure / Algorithm Specification

1.Data Structure

The programming language we use is C++. The data of our program is mainly based on $map, vector, and\ pair$.

```
vector<string> txt_name;
map<string,vector<pair<int,int>>> SearchEngine;
map<string,int> Frequency[50];
```

- **vector txt_name** is the vector used to store all the text in the database.
- **map<string, vector<pair<int, int>>>** SearchEngine is a data structure used to store inverted indexes, in which the two ints in the pair store the book target number and the corresponding number of rows, respectively.
- **map<string,int> Frequency[50]** is a data structure used to store the frequency of lookup words in each book.

2. Descriptions of all the key algorithms:

- **Overview of the algorithm**:We first perform root processing and case conversion of the input words, and then use the inverted index to query the input words, store the results, and store the query results in a local txt file (the path is ./data./**word**_data.txt, where **word** is the name of the query word), so as to shorten the time for searching for the word for the second time.

Database reads : Read the title of each article in the database and push it into **txt_name**.

```
    string name="./docspider/namedata.txt";/*We stored the title of the article in
a txt text via a crawler*/
        ifstream file_name(name);
        if (!file_name.is_open()){
                cout << "open file error!" << endl;
                return 0;
```

```
        }
        string str_read;
    while(getline(file_name,str_read)){
        txt_name.push_back(str_read);
        }
```

Word_Stem : function is used to process the root word and convert it to case.We processed some of the roots by calling an external stem library.

```
string Word_Stem(string x){
        transform(x.begin(),x.end(),x.begin(),::tolower);
        char*tmp=&x[0];
        int end = stem(tmp, 0, strlen(tmp)-1);
        tmp[end + 1] = 0;
        x=tmp;
        return x;
}
```

Input & Query : We first read the **word** and the **ratio** to be queried, and then look for the existence of the relevant data file locally.At the same time, we first judge the query word to determine whether it is a **stop word**. We use a more standard deactivation thesaurus for comparison, which already exists locally and can be read from the file.

```
    cin>>str>>rate;
    string tmpl;
        ifstream stopword("StopWord.txt");
        int flag=1;
        while(getline(stopword,tmpl)){
            if(str.compare(tmpl)==0) flag=0;
        }
        if(flag==0){
                cout<<"This is a stop word"<<endl;
                return 0;
        }
    str=Word_Stem(str);
        string Exist_file="./data/";
        Exist_file+=str;
        Exist_file+="_data";
        Exist_file+=".txt";
        ifstream check_name(Exist_file);
    if (!check_name.is_open()){
        Traverse the database for queries and store inverted index data locally.
    }
    else{
        Read the data file for data processing.
    }
```

Query method(Words that have not been searched) : First, we read the name of the document locally, then open the corresponding document, demarcate each string with a space, and store the read words in the map and record it.The document index number and the position of the word in the document (number of lines). If the word Repeat, the number of repeated words in the word list is increased by one, and The result of the final read is stored in the map.

```
for(int i=0;i<txt_name.size();i++){
                    string now_txt;
                    now_txt="./docspider/";
                    now_txt+=txt_name[i];
                    now_txt+=".txt";
                    ifstream file_name(now_txt);
                    if (!file_name.is_open()){
                        cout << "open file error!" << endl;
                        return 0;
                }
                string tmpn;
                int line=0;
                while(getline(file_name,tmpn)){/*Read by line, chunk by string*/
                        line++;
                        tmpn=Word_Stem(tmpn);/*Root processing for each word*/
                        int index=tmpn.find(str);
                    if(index!=string::npos){
                            SearchEngine[str].push_back(make_pair(i,line));/*push
into the data structure of the search results*/
                            Frequency[i][str]++;/*Frequency plus one*/
                            }
                    }
            }
        map<string,int> BookName;
                for (const auto& entry : SearchEngine) {/*Write to local txt file*/
                        ofstream ofs;
                        string NewTxt="./data/";
                        NewTxt+=str;
                        NewTxt+="_data.txt";
                    ofs.open(NewTxt,ios::out);
                    string w=entry.first;
                    ofs<<entry.first;
                    for (const auto& subEntry : entry.second) {
                        string book=txt_name[subEntry.first];
                        if(BookName[book]==0){
                                ofs<<endl;
                                BookName[book]=1;
                                ofs<<book<<" "<<Frequency[subEntry.first][w]<<endl;
                                }
                    ofs<<subEntry.second<<" ";
                    }
                    ofs<<endl;
                        BookName.clear();
                ofs.close();
        }
        priority_queue<int> q;//*Use the small root heap to find the k-th largest
```

```
element, k=rate*total number of articles*/
        for(int i=0;i<txt_name.size();i++){
                q.push(Frequency[i][str]);
                }
                for(int i=0;i<rate*txt_name.size();i++){
                        q.pop();
                }
                int boundary=q.top();
            for (const auto& entry : SearchEngine) {/*Output at terminal*/
                string w=entry.first;
                cout<<entry.first<<endl;
                for(int i=0;i<txt_name.size();i++){
                    if(Frequency[i][w]>=boundary) {
                        cout<<txt_name[i]<<" "<<Frequency[i][w]<<" The first
appearance is in the ";
                                        for (const auto& subEntry : entry.second) {
                                            if(subEntry.first==i){
                                                cout<<subEntry.second<<" line"
<<endl;
                                break;
                                        }
                    }
                        }
                }
                BookName.clear();
            }
```

Query method(Words that have been searched) :Reads the txt file where the word is stored locally. Since the storage format of the data file is fixed, the first line of each book block stores the title of the book and the frequency in the book, and the next few lines store the number of lines that appear, so we only need to iterate through each line of the txt file, extract the book title and its corresponding frequency, and output the book title and the number of lines where the word appears for the first time in the book.

```
priority_queue<int> q;/*Use the small root heap to find the k-th largest element,
k=rate*total number of articles*/
                int total=0;
                string tmpn;
                getline(check_name,tmpn);
                while(getline(check_name,tmpn)){
                        int k=atoi(&tmpn[0]);
                        if(k==0) {
                                total++;
                                int sum,i;
                                sum=0;
                                i=tmpn.size()-1;
                                string temp=tmpn;
                                while(i>0){
                                        if(tmpn[i]==' ') break;
                                        i--;
                                }
                                temp=temp.substr(i+1, temp.length()-i);
```

```
                int fre=atoi(&temp[0]);
                q.push(fre);
        }
}
for(int i=0;i<rate*total;i++){
        q.pop();
}
int boundary=q.top();
ifstream check2_name(Exist_file);
getline(check2_name,tmpn);
while(getline(check2_name,tmpn)){/*to output*/
        int k=atoi(&tmpn[0]);
        if(k==0) {
                total++;
                int sum,i;
                sum=0;
                i=tmpn.size()-1;
                string temp=tmpn;
                while(i>0){
                        if(tmpn[i]==' ') break;
                        i--;
                }
                temp=temp.substr(i+1, temp.length()-i);
                int fre=atoi(&temp[0]);
                if(fre>boundary) {
                    cout<<tmpn<<" The first appearance is in the ";
                    string line;
                    getline(check2_name,line,' ');
                    cout<<line<<" line"<<endl;
                }
        }
}
```

Chapter 3: Testing Results

word + rates

Test Cases	Purpose	Result	Status
well 0.1	stop word	This is a stop word	Pass
apple 0.5	normal word	1.Title:" As You Like It" ,Number of occurrences:3 The first appearance is in:571 line 2.Title:" Love's Labours Lost" ,Number of occurrences:3 The first appearance is in:1037 line 3.Title:" Measure for Measure" ,Number of occurrences:3 The first appearance is in:90 line 4.Title:" Taming of the Shrew" ,Number of occurrences:4 The first appearance is in:394 line 5.Title:" Troilus and Cressida" ,Number of occurrences:6 The first appearance is in:663 line ...(ignore 10 articals)	Pass
apples 0.5	stemming	1.Title:"As You Like It" ,Number of occurrences:3 The first appearance is in:571 line 2.Title:"Love's Labours Lost" ,Number of occurrences:3 The first appearance is in:1037 line 3.Title:"Measure for Measure" ,Number of occurrences:3 The first appearance is in:90 line 4.Title:"Taming of the Shrew" ,Number of occurrences:4 The first appearance is in:394 line	Pass

Test Cases	Purpose	Result	Status
		5.Title:"Troilus and Cressida" ,Number of occurrences:6 The first appearance is in:663 line ...(ignore 10 articals)	
apples 1.0	rate search	1.Title:"All's Well That Ends Well" ,Number of occurrences:2 The first appearance is in:396 line 2.Title:"As You Like It" ,Number of occurrences:3 The first appearance is in:571 line 3.Title:"Love's Labours Lost" ,Number of occurrences:3 The first appearance is in:1037 line 4.Title:"Measure for Measure" ,Number of occurrences:3 The first appearance is in:90 line 5.Title:"The Merchant of Venice" ,Number of occurrences:2 The first appearance is in:521 line 6.Title:"A Midsummer Night's Dream" ,Number of occurrences:2 The first appearance is in:1383 line 7.Title:"Much Ado About Nothing" ,Number of occurrences:2 The first appearance is in:468 line 8.Title:"Pericles, Prince of Tyre" ,Number of occurrences:2 The first appearance is in:1304 line 9.Title:"Taming of the Shrew" ,Number of occurrences:4 The first appearance is in:394 line 10.Title:"Troilus and Cressida" ,Number of occurrences:6 ...(ignore 17 articals)	Pass

Chapter4 Analysis and Comments

1.Complexity analysis

1) For words that have not been searched:

- **Time Complexity:** $O(M * L + \Phi + K) = O(M * L)$
  K is the sum of the number of times the file is read and the number of articles, **M** is the number of articles,**L** is the number of lines in the article,Φ is the sum of the time it takes to build the root heap and the number of outputs.
- **Space Complexity:** $O(M * L + \Phi)$
  M is the number of articles,**L** is the number of lines in the article,Φ is the sum of the space it takes to build the root heap and the number of outputs.

2) For words that have been searched:

- **Time Complexity:** $O(N)$
  N is the number of lines in the **word_data.txt**,and the $O(N)$ here is much smaller than the $O(M * L)$ above.Because all the desired results are already in the local text, it is only necessary to iterate through each line of the text and extract the data.
- **Space Complexity:** $O(\Phi)$
  Φ is the sum of the space it takes to build the root heap and the number of outputs. Because all the desired results are already in the local text, it is only necessary to create a small root heap to filter the text of the output.

2.Program Highlights

It can be seen that when a word is searched for obsolete, the time and space cost of searching for the word again is greatly reduced, and in the face of massive data, the search speed will be faster and faster after the user's continuous search.

Appendix: Source Code(if required)

1. spider.py

```
#注：本程序主要通过文件流的形式进行爬取，从而避免某些全局变量的出现
#注：本程序爬取得到的所有信息均与本程序处于同名文件夹下

import requests
from bs4 import BeautifulSoup

headers = {"user-agent":'Mozilla/5.0(Windows NT 10.0; Win64; x64)
AppleWebKit/537.36(KHTML,like Gecko) Chrome/86.0.4240.198 Safari/537.36'}
#上句为反反爬的一种操作，通过模拟浏览器访问实现html获取

#函数一：爬取基本信息，这个函数必须第一个执行，以更新文件流
def maindata():
    url = 'http://shakespeare.mit.edu/'
    response = requests.get(url, headers = headers)
    if (response.status_code != 200):
        print("request declined")#异常，有反爬装甲
    else:
        dstring = response.text

    f = open("data.txt", "w")#把主页的html存到文件里对比下
    f.write(dstring)
    f.close()
    f = open("data.txt", "r")
    dstring = f.read()
    f.close()
    soup = BeautifulSoup(dstring, 'lxml')
    passagesoup = soup.body
    tablesoup = passagesoup.find_all("table")
    targettable = tablesoup[1]
    targettr = targettable.find_all("tr")[1]#这里与网页本身有关，第一个tr是分类
    targeta = targettr.find_all("a")

    nameset = ["string"]
    nameset += [None] * 100
    f = open("firstaddress.txt", "w")
    for i in range(len(targeta)):
        f.write(url)
        f.write(targeta[i].attrs['href'])
        f.write("\n")
    f.write("#")
    f.close()
    for i in range(len(targeta)):
        nameset[i] = targeta[i].string
        nameset[i] = nameset[i].replace('\n', ' ').strip()
    articlenumber = len(targeta)
    namedata = open("namedata.txt", "w")
```

```
    for i in range(articlenumber):
        namedata.write(nameset[i])
        namedata.write('\n')
    namedata.close()
    print("the files are initialized succesfully.")

#现在已经获取了第一层的所有地址，接下来需要对这些地址继续爬取(即，章节名字)
#second = open("secondaddress.txt", "w")
#first = open("firstaddress.txt", "r")
#listofname = first.readlines()#这里面的字符串后面有回车，记得strip一下
#first.close()
#for i in range(articlenumber):
#url = listofname[0].strip()
#response = requests.get(url, headers = headers)
#if (response.status_code != 200):
#    print("request declined")#异常，有反爬装甲
#else:
#    sec_string = response.text
#sec_soup = BeautifulSoup(sec_string, 'lxml')
#sec_body = sec_soup.body
#print(list(enumerate(sec_body.p.next_siblings)))
#结果经过以上代码，发现其实二层地址下有一个full.html，加这个后缀就可以爬全书了，偷个懒ovo
#打完所有代码回来，偷懒成功，上面的东西基本没啥用了，但也可以留着

#为了方便，以下全局变量需被声明并定义
maindata()
namedata = open("namedata.txt", "r")
first = open("firstaddress.txt", "r")
listofname = namedata.readlines()#因为这里存文件流，所以还是保存\n了，避免排版太抽象
listofaddress = first.readlines()
namedata.close()
first.close()

def bookspider():
    for i in range(37):#一共有37个可以直接爬的书
        content = open(listofname[i].strip() + ".txt", "w")
        content.write(listofname[i])
        url = listofaddress[i][:-11] + "full.html"
        print(url)
        response = requests.get(url, headers = headers)
        if (response.status_code != 200):
            print("request declined")#异常，有反爬装甲
        else:
            dstring = response.text
        wholesoup = BeautifulSoup(dstring, 'lxml')
        body_soup = wholesoup.body
        a_soup = body_soup.find_all(["a", "i"])#a后是剧本，i后是出场/入场
        num_of_line = len(a_soup)
        for j in range(2, num_of_line):
            content.write(a_soup[j].string)
            content.write('\n')
    # 此时运行程序，会生成37个txt文档，每个文档都是一本书

#以下是十四行诗的爬虫，因为它本身结构不同，又要与其他诗歌区别开来
def sonnetspider():
    index = 37
    content = open(listofname[index].strip() + ".txt", "w")
```

```
    content.write(listofname[index])
    url = listofaddress[index].strip()
    print(listofname[index])
    print(url)
    response = requests.get(url, headers = headers)
    if (response.status_code != 200):
        print("request declined")
    else:
        dstring = response.text
    sonnetsoup = BeautifulSoup(dstring, 'lxml')
    secondaddress = sonnetsoup.body.find_all("a")
    for i in range(1, len(secondaddress)):
        url = "http://shakespeare.mit.edu/Poetry/" + secondaddress[i].attrs['href']
        response = requests.get(url, headers = headers)
        if (response.status_code != 200):
            print("request declined")
        else:
            dstring = response.text
        partsoup = BeautifulSoup(dstring, 'lxml')
        partstring = partsoup.body.find_all("blockquote")
        print(secondaddress[i].attrs['href'][:-5])
        content.write(secondaddress[i].attrs['href'][:-5])
        content.write('\n')
        for j in range(len(partstring)):
            content.write(partstring[j].text)


#以下是其他诗歌的爬虫
def poemspider():
    index = 38
    for i in range(index, len(listofname)):
        content = open(listofname[i].strip() + ".txt", "w")
        content.write(listofname[i])
        url = listofaddress[i].strip()
        print(url)
        response = requests.get(url, headers= headers)
        if (response.status_code != 200):
            print("request declined")
        else:
            dstring = response.text
        poetsoup = BeautifulSoup(dstring, 'lxml')
        poetcontent = poetsoup.body.find_all(['p', 'blockquote'])
        for j in range(len(poetcontent)):
            content.write(poetcontent[j].text)

print("please input the part you want to update: book/sonnet/poem/(exit: others)")
query = input()
if query == "book":
    bookspider()
elif query == "sonnet":
    sonnetspider()
elif query == "poem":
    poemspider()
```

2. searchdata.cpp

```
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <string.h>
#include <vector>
#include <map>
#include <algorithm>
#include <queue>
#define TRUE 1
#define FALSE 0
using namespace std;
typedef string filename;
typedef int line;
int stem(char *p, int index, int position);
using namespace std;
string Word_Stem(string x);
int main(){
        vector<string> txt_name;/*used to store all the text in the database*/
        map<string,vector<pair<int,int>>> SearchEngine;/*used to store inverted
indexes, in which the two ints in the pair store the book target number and the
corresponding number of rows, respectively*/
        map<string,int> Frequency[50];/*used to store the frequency of lookup words
in each book.*/
        string name="./docspider/namedata.txt";//*Read the local database file to
get the individual article titles*/
        ifstream file_name(name);
        if (!file_name.is_open()){
                cout << "open file error!" << endl;
                return 0;
        }
        string str_read;
    while(getline(file_name,str_read)){
        txt_name.push_back(str_read);
        }
        /*Enter and determine whether it is stopword, and perform root processing
at the same time*/
        string str;
        double rate;
        cout<<"Please input your word and rate:"<<endl;
        cin>>str>>rate;
        string tmpl;
        ifstream stopword("StopWord.txt");
        int flag=1;
        while(getline(stopword,tmpl)){
            if(str.compare(tmpl)==0) flag=0;
        }
        if(flag==0){
                cout<<"This is a stop word"<<endl;
                return 0;
        }
```

```
        str=Word_Stem(str);

        /*Check whether the local data has been queried*/
        string Exist_file="./data/";
        Exist_file+=str;
        Exist_file+="_data";
        Exist_file+=".txt";
        ifstream check_name(Exist_file);
        if (!check_name.is_open()){/*If there is no corresponding data locally,
start querying the database*/
                int ifsee=0;
                for(int i=0;i<txt_name.size();i++){
                    string now_txt;
                    now_txt="./docspider/";
                    now_txt+=txt_name[i];
                    now_txt+=".txt";
                    ifstream file_name(now_txt);
                    if (!file_name.is_open()){
                        cout << "open file error!" << endl;
                        return 0;
                }
                string tmpn;
                int line=0;
                while(getline(file_name,tmpn)){/*Read by line, chunk by string*/
                        line++;
                        tmpn=Word_Stem(tmpn);/*Root processing for each word*/
                        int index=tmpn.find(str);
                    if(index!=string::npos){
                            SearchEngine[str].push_back(make_pair(i,line));/*push
into the data structure of the search results*/
                            Frequency[i][str]++;/*Frequency plus one*/
                            ifsee=1;
                            }
                    }
            }
        map<string,int> BookName;
                for (const auto& entry : SearchEngine) {/*Write to local txt file*/
                        ofstream ofs;
                        string NewTxt="./data/";
                        NewTxt+=str;
                        NewTxt+="_data.txt";
                    ofs.open(NewTxt,ios::out);
                    string w=entry.first;
                    ofs<<entry.first;
                    for (const auto& subEntry : entry.second) {
                        string book=txt_name[subEntry.first];
                        if(BookName[book]==0){
                                ofs<<endl;
                                BookName[book]=1;
                                ofs<<book<<" "<<Frequency[subEntry.first][w]<<endl;
                                }
                    ofs<<subEntry.second<<" ";
                    }
                    ofs<<endl;
                        BookName.clear();
                ofs.close();
        }
```

```
        priority_queue<int> q;//*Use the small root heap to find the k-th largest
element, k=rate*total number of articles*/
        for(int i=0;i<txt_name.size();i++){
                q.push(Frequency[i][str]);
                }
                for(int i=0;i<rate*txt_name.size();i++){
                        q.pop();
                }
                int boundary=q.top();
                int sta=0;
            for (const auto& entry : SearchEngine) {/*Output at terminal*/
                    string w=entry.first;
//                  cout<<entry.first<<endl;
                    for(int i=0;i<txt_name.size();i++){
                        if(Frequency[i][w]>=boundary) {
                                sta++;
                            cout<<sta<<"."<<"Title:"<<"\""<<txt_name[i]<<"\"
,Number of occurrences:"<<Frequency[i][w]<<endl<<"The first appearance is in the ";
                                        for (const auto& subEntry : entry.second) {
                                                if(subEntry.first==i){
                                                    cout<<subEntry.second<<" line"
<<endl;
                                break;
                                                }
                            }
                                }
                        }
                        BookName.clear();
            }
            if(ifsee==0) cout<<"The word does not appear"<<endl;
        }
        else{/*Existing local data, read local data directly*/
            priority_queue<int> q;/*Use the small root heap to find the k-th
largest element, k=rate*total number of articles*/
                int total=0;
                string tmpn;
                getline(check_name,tmpn);
                while(getline(check_name,tmpn)){
                        int k=atoi(&tmpn[0]);
                        if(k==0) {
                                total++;
                                int sum,i;
                                sum=0;
                                i=tmpn.size()-1;
                                string temp=tmpn;
                                while(i>0){
                                        if(tmpn[i]==' ') break;
                                        i--;
                                }
                                temp=temp.substr(i+1, temp.length()-i);
                                int fre=atoi(&temp[0]);
                                q.push(fre);
                        }
                }
                for(int i=0;i<rate*total;i++){
                        q.pop();
                }
```

```
                int boundary=q.top();
                int sta=0;
                ifstream check2_name(Exist_file);
                getline(check2_name,tmpn);
                while(getline(check2_name,tmpn)){/*to output*/
                        int k=atoi(&tmpn[0]);
                        if(k==0) {
                                total++;
                                int sum,i;
                                sum=0;
                                i=tmpn.size()-1;
                                string temp=tmpn;
                                while(i>0){
                                        if(tmpn[i]==' ') break;
                                        i--;
                                }
                                temp=temp.substr(i+1, temp.length()-i);
                                tmpn=tmpn.substr(0,i);
                                int fre=atoi(&temp[0]);
                                if(fre>boundary) {
                                        sta++;
                                    cout<<sta<<"."<<"Title:"<<"\""<<tmpn<<"\"
,Number of occurrences:"<<temp<<""<<endl;
                                        cout<<"The first appearance is in:";
                                    string line;
                                    getline(check2_name,line,' ');
                                    cout<<line<<" line"<<endl;
                                }
                        }
                }
        }
        return 0;
}
string Word_Stem(string x){
        transform(x.begin(),x.end(),x.begin(),::tolower);
        char*tmp=&x[0];
        int end = stem(tmp, 0, strlen(tmp)-1);
        tmp[end + 1] = 0;
        x=tmp;
        return x;
}

//NOTE: Because of the length of the report, here won't give out the code of
stemming code from the extern lib.
```

References

- Stemmer lib: *https://github.com/wooorm/stmr.c*

Declaration

We hereby declare that all the work done in this project titled "roll your own mini search engine" is of our independent effort as a group.