浙江大學

Advanced Data Structures

Roll Your Own Mini Search Engine

2021~2022春夏学期 2022 年 3 月 13 日

Roll Your Own Mini Search Engine

Categories

Chapter 1: Introduction

1.1 Background

In this project, you are supposed to create your own mini search engine which can handle inquiries over "The Complete Works of William Shakespeare" (http://shakespeare.mit.edu/).

You may download the functions for handling stop words and stemming from the Internet, as long as you add the source in your reference list.

1.2 Tasks specification

Our main tasks are:

1) Run a word count over the Shakespeare set and try to identify the stop words (also called the noisy 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 (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

2.1 Overall algorithms design

2.1.1

We got the data of the website(http://shakespeare.mit.edu/) .After that we use the text-Transform tool(https://www.iamwawa.cn/daxiaoxie.html)to transform the capital to small letter and recorded the frequency-collection and identify the stop words like "a" " it" from the list of stop words(https://blog.csdn.net/u012661010/article/details/70880847).

2.1.2 Indexing

Firstly,we used the post-list in std::map container of C++ to access the whole files and associated every word that appearing in the files.Secondly,we identify the frequency of the words and the files included.

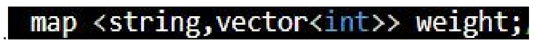
2.1.3 Searching

It is advised that we should make the index file available for search queries,thus speed of finding information related to the queries would get faster.

2.2 Algorithm Descriptions

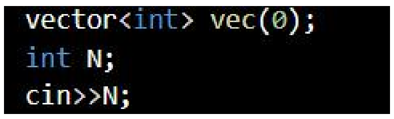
Our algorithm contains 3 main parts: Word counter, index generator, and query processor.Here we just put these three parts together to sketch a whole picture by pseudo-code as our program is highly integrated.

1. Create a new dictionary with key as the word and value as a vector containing the ids of the documents containing the word

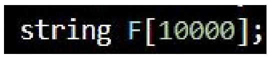


```
 map <string,vector<int>> weight;
```

2. Use N to record the number of the file we have input.Use F array to store the file.

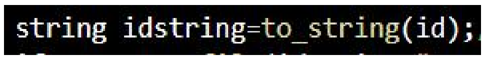


```
vector<int> vec(0);
int N;
cin>>N;
```

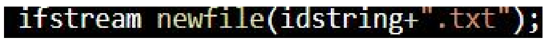


```
string F[10000];
```

3. The ID is converted to a string that can be attached to a file and input the file which is corresponding to the number.

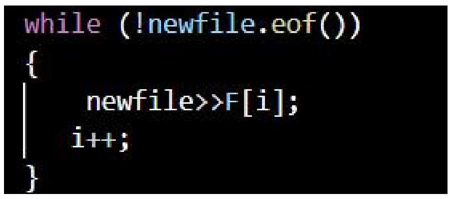


```
string idstring=to_string(id);
```



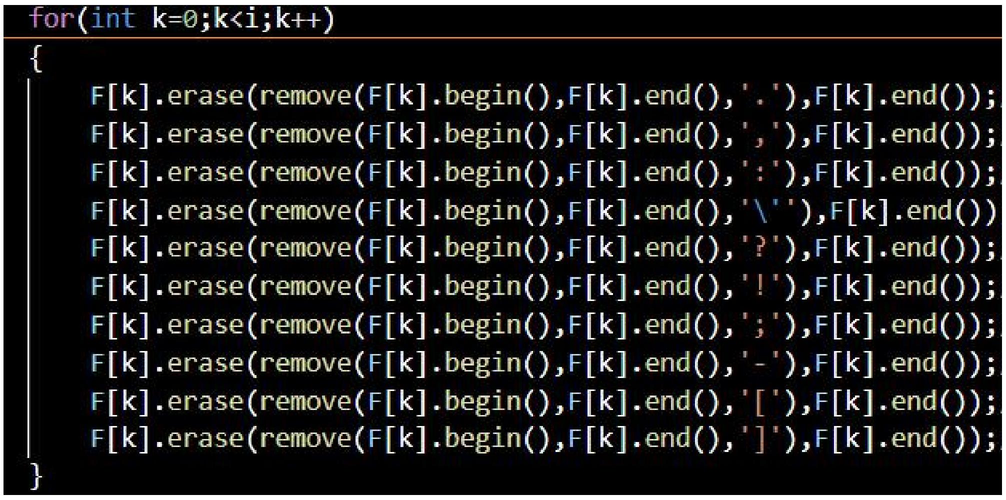
```
ifstream newfile(idstring+".txt");
```

4. Read each word into an array of strings



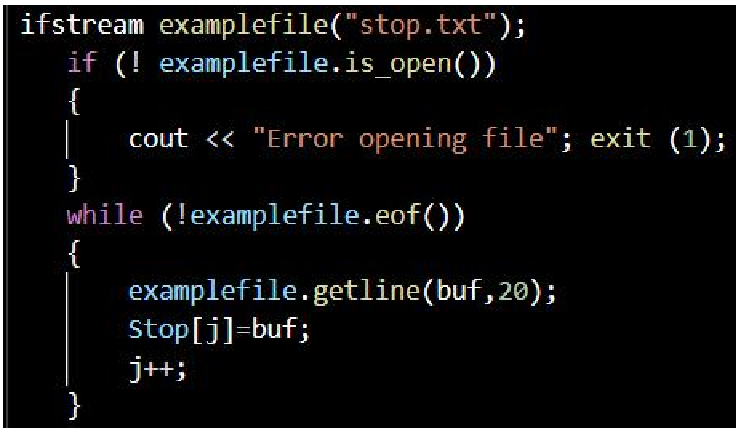
```
while (!newfile.eof())
{
    newfile>>F[i];
   i++;
}
```

5. Eliminate the tokens from the file.



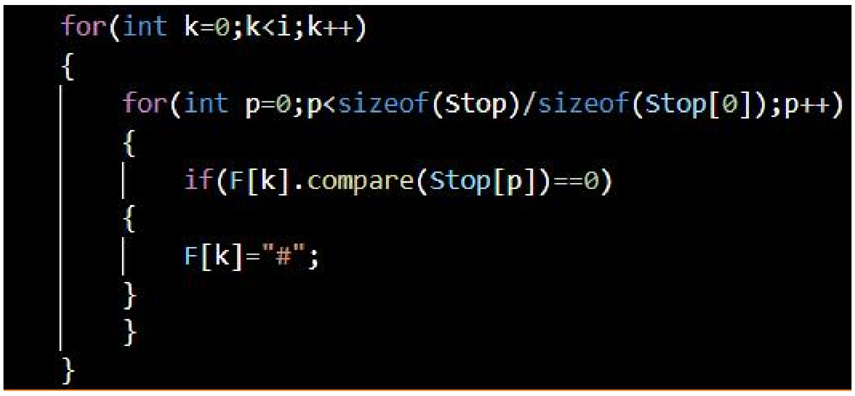
```
for(int k=0;k<i;k++)
{
    F[k].erase(remove(F[k].begin(),F[k].end(),'.'),F[k].end());
    F[k].erase(remove(F[k].begin(),F[k].end(),','),F[k].end());
    F[k].erase(remove(F[k].begin(),F[k].end(),':'),F[k].end());
    F[k].erase(remove(F[k].begin(),F[k].end(),'\''),F[k].end())
    F[k].erase(remove(F[k].begin(),F[k].end(),'?'),F[k].end());
    F[k].erase(remove(F[k].begin(),F[k].end(),'!'),F[k].end());
    F[k].erase(remove(F[k].begin(),F[k].end(),';'),F[k].end());
    F[k].erase(remove(F[k].begin(),F[k].end(),'-'),F[k].end());
    F[k].erase(remove(F[k].begin(),F[k].end(),'['),F[k].end());
    F[k].erase(remove(F[k].begin(),F[k].end(),']'),F[k].end());
}
```

6. Stop word input from the file.



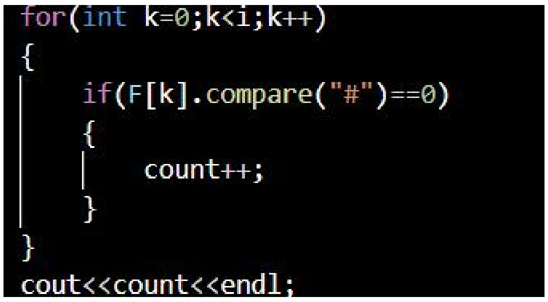
```
ifstream examplefile("stop.txt");
   if (! examplefile.is_open())
   {
       cout << "Error opening file"; exit (1);
   }
   while (!examplefile.eof())
   {
       examplefile.getline(buf,20);
       Stop[j]=buf;
       j++;
   }
```

If there exists stop word, then turn it into “#”



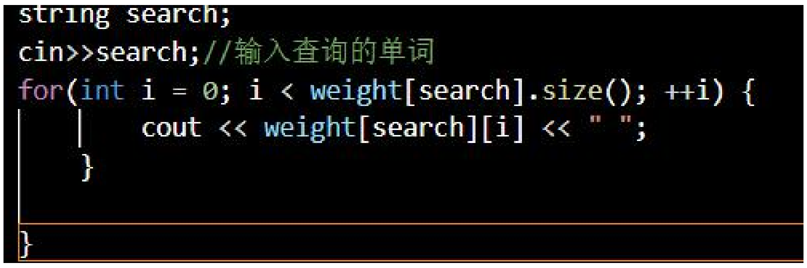
```
    for(int k=0;k<i;k++)
    {
        for(int p=0;p<sizeof(Stop)/sizeof(Stop[0]);p++)
        {
            if(F[k].compare(Stop[p])==0)
        {
            F[k]="#";
        }
        }
    }
```

7. Word counter



```
for(int k=0;k<i;k++)
{
    if(F[k].compare("#")==0)
    {
        count++;
    }
}
cout<<count<<endl;
```

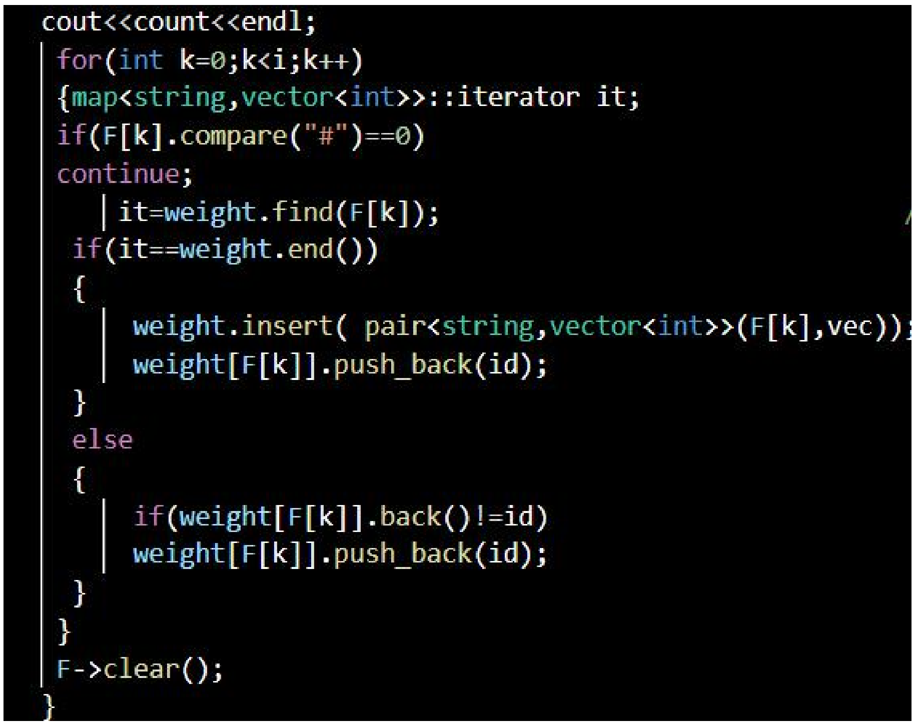
8. Index Generator



```
string search;
cin>>search;//输入查询的单词
for(int i = 0; i < weight[search].size(); ++i) {
        cout << weight[search][i] << " ";
    }

}
```

9. Query processor.

Create a new iterator to walk through the dictionary, skip the stop word, find out if the word is in the dictionary, if not, create a new line of the dictionary, Add the word to the dictionary and press the id of the document in the vector. If the vector does not have the ID of the document, press the ID of the document in the vector .



```
  cout<<count<<endl;
   for(int k=0;k<i;k++)
   {map<string,vector<int>>::iterator it;
   if(F[k].compare("#")==0)
   continue;
       it=weight.find(F[k]);
    if(it==weight.end())
    {
        weight.insert( pair<string,vector<int>>(F[k],vec));
        weight[F[k]].push_back(id);
    }
    else
    {
        if(weight[F[k]].back()!=id)
        weight[F[k]].push_back(id);
    }
   }
   F->clear();
  }
```

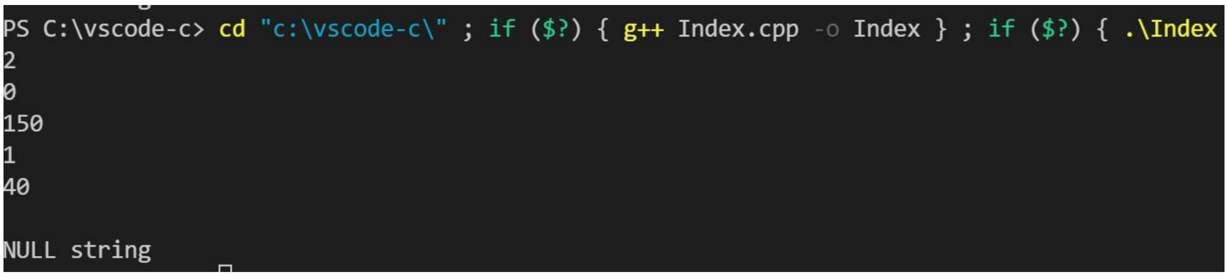
2.3 Main data structures

The data structure of our index is array.We first create a Inverted-File-Index and then use the Stop-Word tool to get stop words and store them in the set Stop-Word. After that, we scan all the documents and build our index file. Then we start searching words with the help of the index file, as well as test the correctness of the Inverted Index we built. If the given threshold is 1.0, we test if our inverted index works correctly.

Chapter 3: Testing results

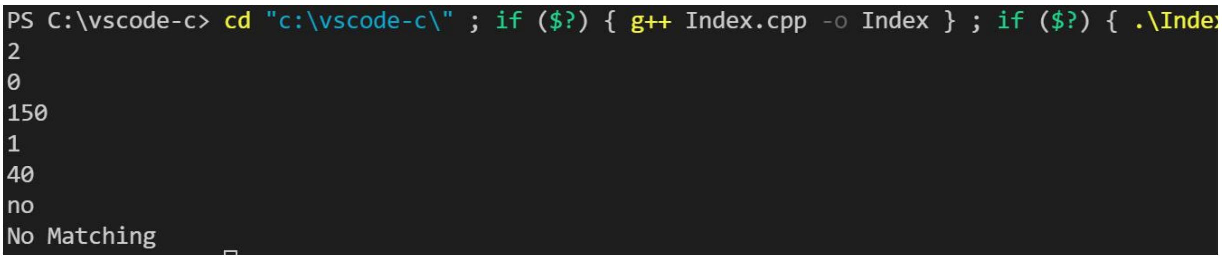
3.1 The correctness of the inverted index

1.empty query



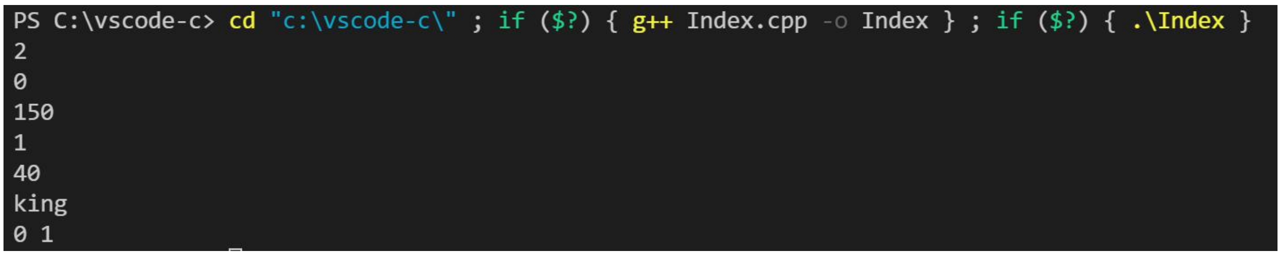
```
PS C:\vscode-c> cd "c:\vscode-c\" ; if ($?) { g++ Index.cpp -o Index } ; if ($?) { .\Index
2
0
150
1
40

NULL string
```

2. stop word



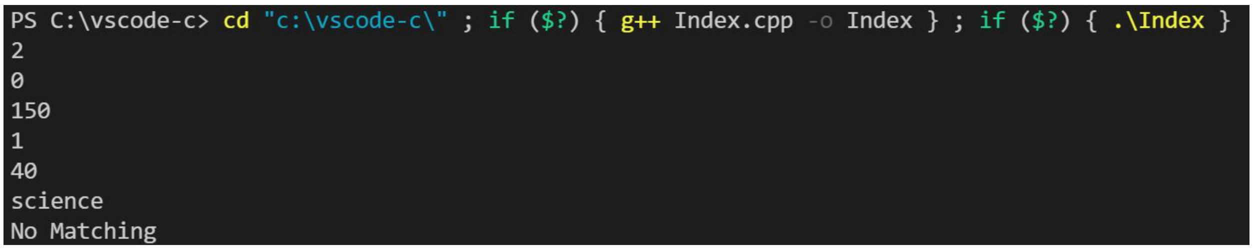
```
PS C:\vscode-c> cd "c:\vscode-c\" ; if ($?) { g++ Index.cpp -o Index } ; if ($?) { .\Inde
2
0
150
1
40
no
No Matching
```

3. word in doc



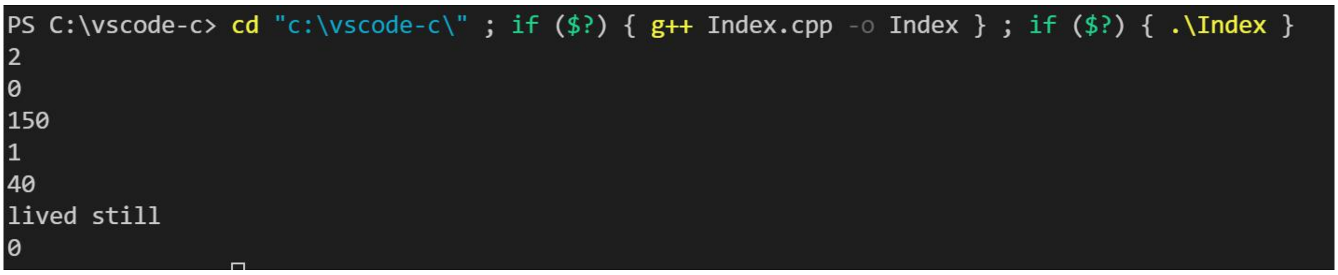
```
PS C:\vscode-c> cd "c:\vscode-c\" ; if ($?) { g++ Index.cpp -o Index } ; if ($?) { .\Index }
2
0
150
1
40
king
0 1
```

4. word not in doc



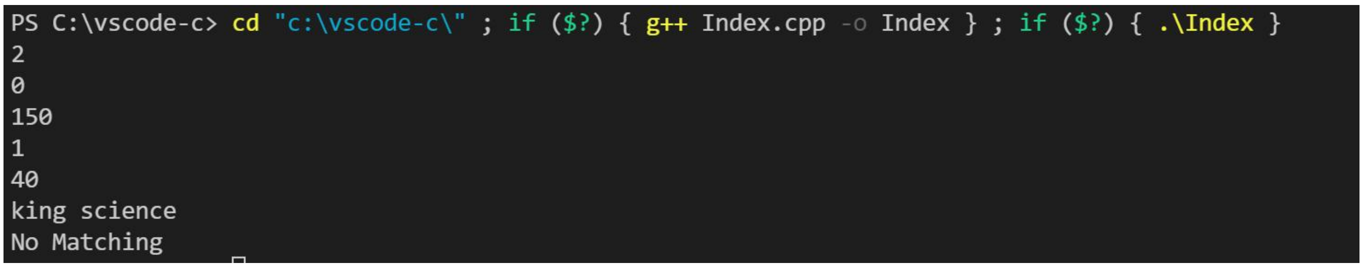
```
PS C:\vscode-c> cd "c:\vscode-c\" ; if ($?) { g++ Index.cpp -o Index } ; if ($?) { .\Index }
2
0
150
1
40
science
No Matching
```

5. phrase in doc



```
PS C:\vscode-c> cd "c:\vscode-c\" ; if ($?) { g++ Index.cpp -o Index } ; if ($?) { .\Index }
2
0
150
1
40
lived still
0
```

6. Phrase not in doc



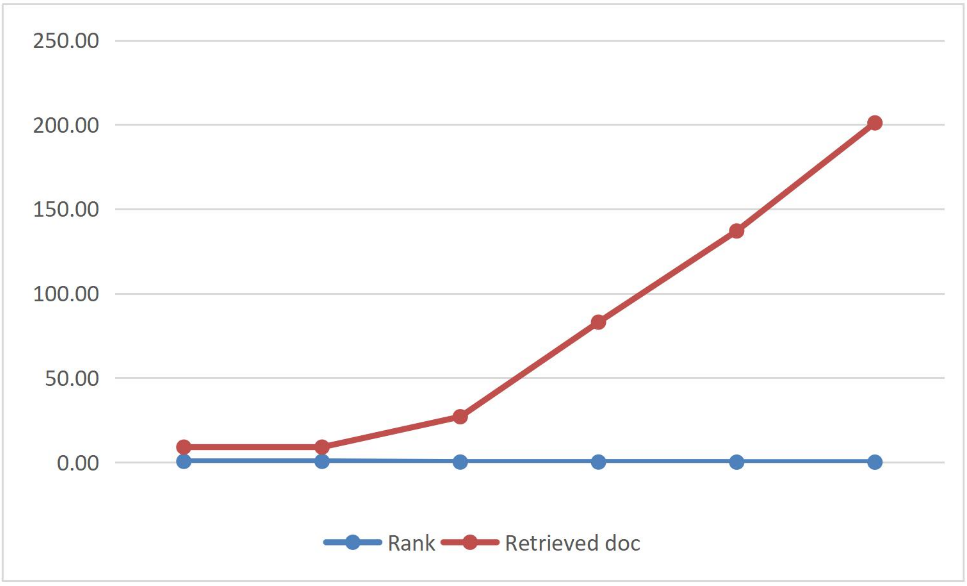
```
PS C:\vscode-c> cd "c:\vscode-c\" ; if ($?) { g++ Index.cpp -o Index } ; if ($?) { .\Index }
2
0
150
1
40
king science
No Matching
```

3.2 The thresholding for query

Our task is to test how the threshold of the query affects the results. According to our document retrieval method, the number of documents retrieved and the search time should decrease as the threshold decreases. At the meantime,the rank of the query phrase should be a line parallel to the x axis.

We choose the phrase " Merchant of Syracuse, plead no more" from "The Comedy of Errors.txt" .Then we compare the documents with different thresholds.

Testing Threshold	Rank	Retrieved doc
1	No.17/201	201
0.8	No.13/137	137
0.6	No.13/83	83
0.4	No.4/27	27
0.2	No.5/9	9
0	No.5/9	9



Chapter 4: Analysis and Comments

4.1 Time complexity

4.1.1 Stop word

O(NlogM)

where N is the total number of words among the documents, M is the number of different words in the collection.

4.1.2 Whole program

The upper bound is O(min(MK, N)),which is used to store the inverted index.

4.2 Space complexity

4.2.1 Stop word

O(M)

Where M is the number of different words in the collection.

4.1.2 Whole program

The upper bound is O(2*(sum of number of documents containing it for each term) + M).

4.3 Comments

The program we show here does not sort the retrieved documents. We did not record the word frequency in the list, only the intersection operation. What we can improve later is that we can retrieve and sort documents related to the query terms, since we assume that the highest-ranked documents tend to be those that contain all the query terms. Moreover, waiver of documents can be avoided.
In a practical case, we can only truncate the retrieved list to control the number of documents provided to the user, thereby improving accuracy and ensuring the security of the retrieval.

4.4 Bonus Question

Considering the typical memory capacity of a modern PC is 8GB, It would be impossible to store indexes.In our program , the numbers mean that K = 500,000 and M = 400,000.One alternative is to make full use of storage space. As long as N is not too large, the index can always be stored on hard disk, and when N is too large, we can use a distributed system. The index is first stored in blocks. When the temporary index size reaches the set block size, it is written to a block and memory is freed. Finally, we merge the blocks. When building an inverted index, we mark each term with an ID . Before writing, we first sort the index by ID when the block enters the disk. In the merge step, we open all the blocks on the hard disk at the same time and select the record with the lowest ID among all the blocks. We need to select one record from each block. Once the newly read record has a different ID from the last record, which means that there are no more records related to the last record in the block, we write it to the file that stores the entire index.

Appendix:Source Code

```
#include<iostream>
#include<string>
#include <cstring>
#include<math.h>
#include<algorithm>
#include <fstream>
#include <cstdlib>
#include<vector>
#include<map>
#include <sstream>
using namespace std;
void split(const string& s,vector<string>& sv,const char flag = ' ') {
    sv.clear();
    istringstream iss(s);
    string temp;

    while (getline(iss, temp, flag)) {
        sv.push_back(temp);
    }
    return;
}
map <string,vector<int>> weight;//新建字典，key为词语，value为一个vector，其中存放包含这个词的文档的id
int main()
{vector<int> vec(0);
int N;
cin>>N;//总共读入多少个文档
getchar();
for(int id=0;id<N;id++)
{   cout<<id<<endl;
    string F[10000];//用以存储文档的数组
    int i=0;
    char buf[20];
    string idstring=to_string(id);//id转换为可以连接到文件的字符串
    ifstream newfile(idstring+".txt");//读入序号对应的文件
    if (! newfile.is_open())
    {
        cout << "Error opening file"; exit (1);//如果找不到文件则退出
    }
    while (!newfile.eof())
    {
        newfile>>F[i];//将每个单词读入字符串数组中
       i++;
    }
    for(int k=0;k<i;k++)
    {
        F[k].erase(remove(F[k].begin(),F[k].end(),'.'),F[k].end());//从单词中清除标点符号
        F[k].erase(remove(F[k].begin(),F[k].end(),','),F[k].end());//从单词中清除标点符号
        F[k].erase(remove(F[k].begin(),F[k].end(),':'),F[k].end());//从单词中清除标点符号
        F[k].erase(remove(F[k].begin(),F[k].end(),'\''),F[k].end());//从单词中清除标点符号
        F[k].erase(remove(F[k].begin(),F[k].end(),'?'),F[k].end());//从单词中清除标点符号
        F[k].erase(remove(F[k].begin(),F[k].end(),'!'),F[k].end());//从单词中清除标点符号
        F[k].erase(remove(F[k].begin(),F[k].end(),';'),F[k].end());//从单词中清除标点符号
        F[k].erase(remove(F[k].begin(),F[k].end(),'-'),F[k].end());//从单词中清除标点符号
    }
    string Stop[10005];//停止词数组
    int j=0;
 ifstream examplefile("stop.txt");//从文件中读入停止词
    if (! examplefile.is_open())
    {
        cout << "Error opening file"; exit (1);//如果找不到文件则退出
    }
    while (!examplefile.eof())
    {
        examplefile.getline(buf,20);//从文件中读入停止词
        Stop[j]=buf;//从文件中读入停止词
        j++;//从文件中读入停止词
    }
    for(int k=0;k<i;k++)
    {
        for(int p=0;p<sizeof(Stop)/sizeof(Stop[0]);p++)
        {
            if(F[k].compare(Stop[p])==0)//如果读入文档中的单词有停止词
        {
            F[k]="#";//则将其变为#
        }
        }
    }
    int count=0;
    for(int k=0;k<i;k++)
    {
        if(F[k].compare("#")==0)
        {
            count++;//字数统计，有几个非停止词的单词
        }
    }
    cout<<count<<endl;//打印词数
     for(int k=0;k<i;k++)
     {map<string,vector<int>>::iterator it;//新建迭代器遍历字典
     if(F[k].compare("#")==0)//跳过停止词
     continue;
         it=weight.find(F[k]);//查找某词是否在字典中
      if(it==weight.end())//如果不在
      {
          weight.insert( pair<string,vector<int>>(F[k],vec));//新建字典的一行，将这个词加入字典
          weight[F[k]].push_back(id);//在对应的vector中压入这个词对应文档的id
      }
      else//如果在
      {
          if(weight[F[k]].back()!=id)//如果vector中本身没有这个词所在文档的的id
          weight[F[k]].push_back(id);//在对应的vector中压入这个词对应文档的id
      }
     }
     F->clear();
    }
string search;
getline(cin,search);//输入查询的单词或者phrase
vector<string> phrase;
split(search, phrase, ' ');//将短语中的单词取出存入string的vector中
map<string,vector<int>>::iterator iter;
if(phrase.size()>1)//如果vector中单词多于1个，则进入短语查找模式
{
    vector <int> res=weight[phrase[0]];
```

```
for(int i=0;i<phrase.size();i++)//遍历vector
{vector <int> temp;
if(weight.find(phrase[i])==weight.end())
{
cout<<"No Matching"<<endl;//如果有一个单词都没有找到，则退出
return 0;
}
temp=weight[phrase[i]];
set_intersection(temp.begin(),temp.end(),res.begin(),res.end(),back_inserter(res));//求出包含这些单词的篇目序号集合的交集
}
sort(res.begin(), res.end());//排序
    res.erase(unique(res.begin(), res.end()), res.end());//去重
if(res.empty())//如果没有交集
{
    cout<<"No Matching"<<endl;//则输出报错信息
    return 0;//并且退出
}
else//如果有交集，则定位到短语所在的篇目
{    for(int k = 0; k < res.size(); ++k)//输出所在的篇目
        cout << res[k] <<" ";
}
}
else//如果vector中单词不多于1个，则进入单词查找模式
{
if(search.empty())//如过输入的是空条目，则提醒输入空单词
{
cout<<"NULL string"<<endl;//空条目
return 0;//并且退出
}
iter = weight.find(search);//迭代器
if(iter==weight.end())//如果找不到该单词
{
    cout<<"No Matching"<<endl;//输出找不到的信息并退出
    return 0;//结束程序
}
for(int i = 0; i < weight[search].size(); ++i) {//如果找到了则遍历
        cout << weight[search][i] << " ";//并且输出所在的篇目序号
    }
}
}
```

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.