Hw3 - Report

Yifu He(190003956), Jiahang Guo(198000356)

Nov 3th, 2019

Contents

1 Function and Test

1.1 Interface

To use the program first the user need to input the path of fileName.txt and path of dataset.

```
Welcome to Inverted Index created by Jason He.
Enter the path of fileName.txt:
For example: /Users/yifuhe/Desktop/hw3/fileName.txt
Input here:/Users/yifuhe/Desktop/hw3/fileName.txt

Enter the path of datasets of file
For example: /Users/yifuhe/Desktop/hw3/fin/
Input here:/Users/yifuhe/Desktop/hw3/fin/

------------------------------ Hide the Debug area
Instructions:
This program will not differentiate capitalized letter.
-------------------------------------------
1.Search a word,
2.Search a phrase, e.g. "financial industry"
3.Search two words in the following format, e.g. "future and option"
4.Wide card query, e.g."fin*"
5.Quit.
-------------------------------------------
Enter your choice:
```

1.2 One word search

1.2.1 treat lower and upper letter the same

finance v.s. FiNanCe

```
-------------------------------------------
Enter your choice: 1
Enter the word (e.g. "finance"): finance
finance is in our system.
t10.txt  2
t8.txt  1
t6.txt  1
t5.txt  1
t3.txt  1
t12.txt  1
t11.txt  1
-------------------------------------------
```

```
-----------------------------------------------
Enter your choice: 1
Enter the word (e.g. "finance"): FiNanCe
finance is in our system.
t10.txt  2
t8.txt  1
t6.txt  1
t5.txt  1
t3.txt  1
t12.txt  1
t11.txt  1
```

is

```
-----------------------------------------------
Enter your choice: 1
Enter the word (e.g. "finance"): is
is is in our system.
t3.txt  14
t6.txt  12
t9.txt  11
t12.txt  9
t8.txt  8
t10.txt  7
t5.txt  5
t7.txt  4
t2.txt  4
t11.txt  3
t1.txt  3
t4.txt  2
-----------------------------------------------
```

interpolation

```
-----------------------------------------------
Enter your choice: 1
Enter the word (e.g. "finance"): interpolation
interpolation is in our system.
t12.txt  1
-----------------------------------------------
```

1.3 Bonus

1.3.1 Bonus1: Phrase Search

Brownian motion

```
----------------------------------------
Enter your choice: 2
Enter a phrase, e.g. "financial industry":brownian motion
The phrase of "brownian motion " is in our system:
t1.txt  2
```

is a

```
----------------------------------------
Enter your choice: 2
Enter a phrase, e.g. "financial industry":is a
The phrase of "is a " is in our system:
t6.txt  3
t5.txt  3
t9.txt  1
t8.txt  1
t7.txt  1
t3.txt  1
----------------------------------------
```

financial industry

```
----------------------------------------
Enter your choice: 2
Enter a phrase, e.g. "financial industry":financial industry
There is no financial industry in our system.
----------------------------------------
```

1.3.2 Bonus2: A and B

In this part, if A exists 5 times in file1 and B exists 3 times in file1, we will choose the smaller one. It means there are only 3 pairs of A and B in file1.

"future and option" v.s. "option and future".
They should have the same result.

```
-------------------------------------------
Enter your choice: 3
Enter two words in the following format, e.g. "future and option":future and option
The pair of "future and option" exists in our system:t9.txt  1
t7.txt  1
-------------------------------------------
```

```
-------------------------------------------
Enter your choice: 3
Enter two words in the following format, e.g. "future and option":option and future
The pair of "option and future" exists in our system:t9.txt  3
t7.txt  1
-------------------------------------------
```

is and and

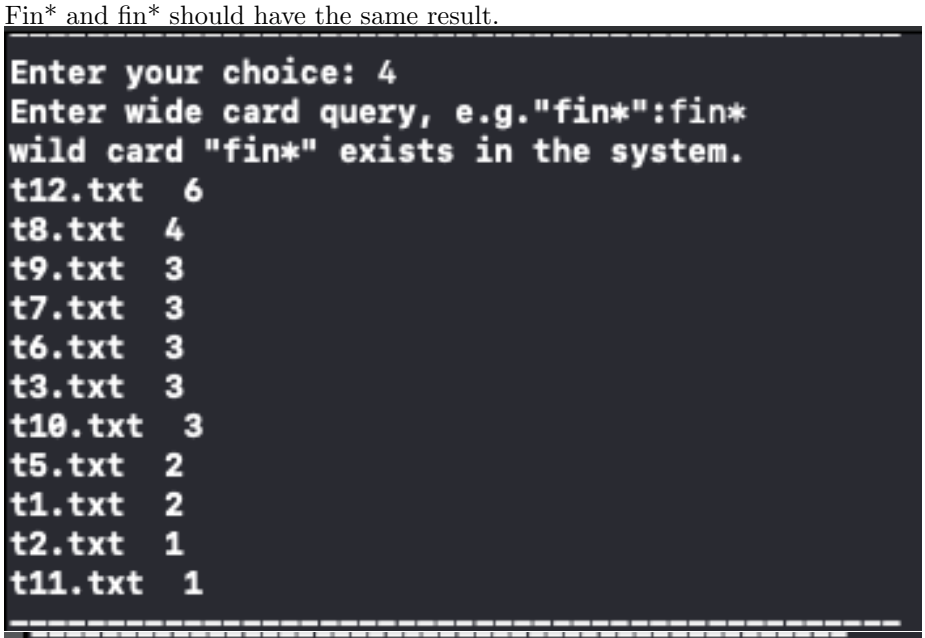
```
-------------------------------------------
Enter your choice: 3
Enter two words in the following format, e.g. "future and option":is and and
The pair of "is and and" exists in our system:t9.txt  11
t8.txt  8
t5.txt  5
t3.txt  5
t2.txt  4
t11.txt  3
t10.txt  3
t1.txt  3
t7.txt  2
t6.txt  2
t12.txt  2
t4.txt  1
```

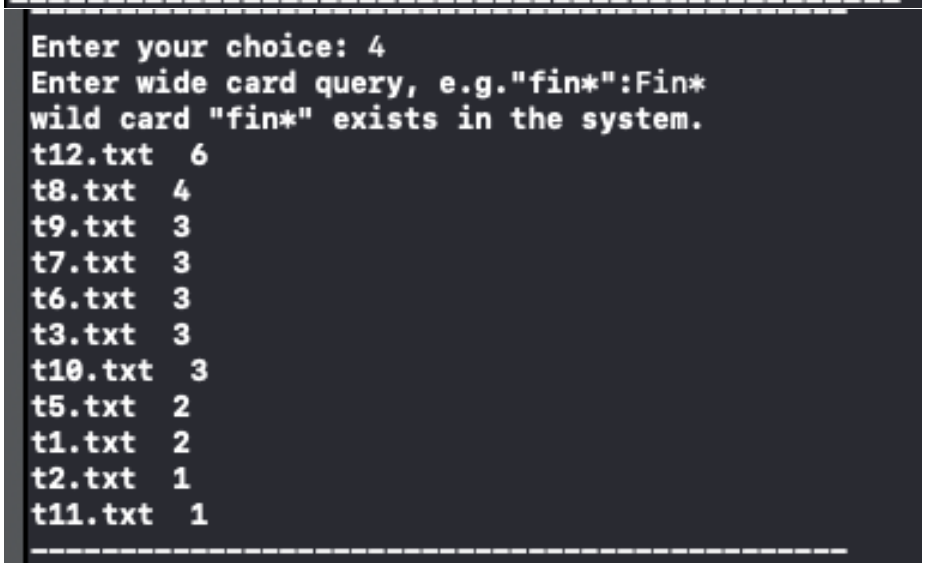
is and the

```
-------------------------------------------
Enter your choice: 3
Enter two words in the following format, e.g. "future and option":is and the
The pair of "is and the" exists in our system:
t3.txt  14
t6.txt  12
t9.txt  11
t12.txt  9
t8.txt  8
t10.txt  7
t5.txt  5
t7.txt  4
t2.txt  4
t11.txt  3
t1.txt  3
t4.txt  2
-------------------------------------------
```

1.3.3 Bonus 3: wild card search

Fin* v.s. fin*
Fin* and fin* should have the same result.





interpolation* v.s. interpo*
Cause the word interpolation only appeared once, so the result should be the

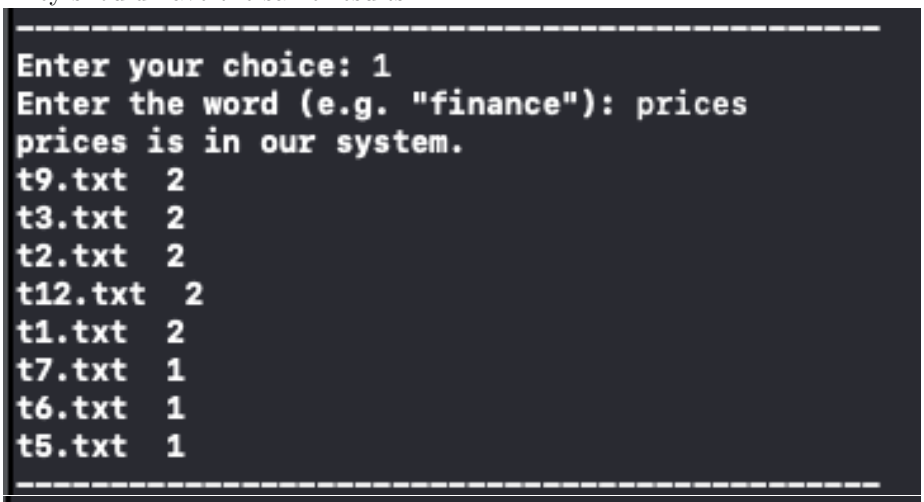
same.

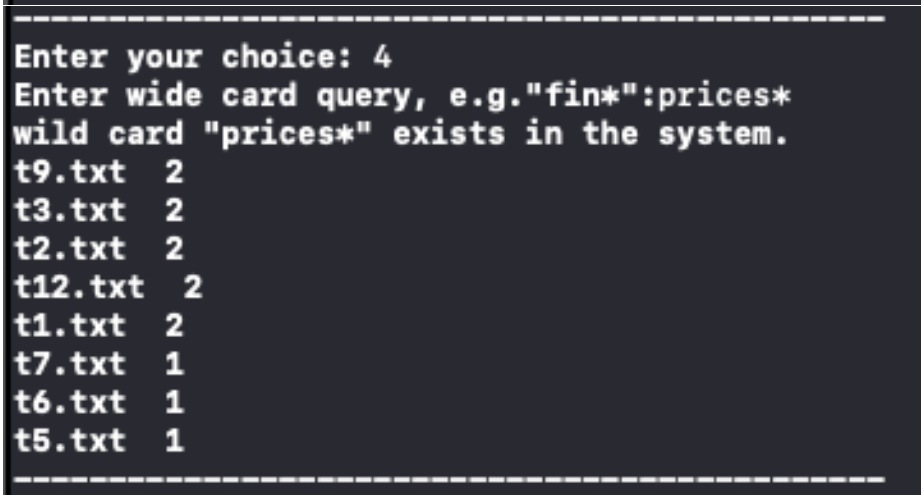
```
-----------------------------------------
Enter your choice: 4
Enter wide card query, e.g."fin*":interpolation*
wild card "interpolation*" exists in the system.
t12.txt  1
```

```
-----------------------------------------
Enter your choice: 4
Enter wide card query, e.g."fin*":interpo*
wild card "interpo*" exists in the system.
t12.txt  1
-----------------------------------------
```

prices* in wild card v.s. prices in one world search

They should have the same results.





2 Intro about the program

2.1 getFileName

Use the function getFileName(string path) to read the file of fileName.txt and save it in a vector of string.

2.2 class Node

1.Use Node to demonstrate the specific word, it has 4 attributes: the fileName of file which it saves in, the word itself and the word after it (for bonus 2) and the count if the former 3 attribute are the same.

2.Overloading the operator == equal, which will be necessary in the later iterator part.
3.Add 1 in count when the former 3 attributes are the same.
4.Use accessors to separate interface and implementation.

2.3 class InvertedIndex

1.InvertedIndex only have one private attribute of map, the key is the word, the value is the vector of Node.
2.It has only one private member function which will not be accessible out of the definition of class. This function will contribute to the public function addNewFile and would be used in the user's interface.
3.It only has one default constructor.
4.The addNewFile member function will be used when the program read file from the file list one by one.
5-8. The four implement of search function by requirement.

3 Source Code

3.1 main.cpp

```
//
//  main.cpp
//  hw3
//
//  Created by Jason Ha on 10/23/19.
//  Copyright © 2019 Jason He. All rights reserved.
//
#include <string>
#include <iostream>
#include <fstream>
#include <filesystem>
#include <cstdlib>
#include "FileAccess.hpp"
#include "InvertedIndex.hpp"
using namespace std;

int main() {
        // Please, don't use this function !!!!!
        //        generateFileName();

//        char pp;
//        ifstream infile("/Users/yifuhe/Desktop/hw3/fin/t9.txt");
```

```
//        if(infile.fail()){
//        cerr << "File cound not be opened in addNewfile" << endl;
//        exit(EXIT_FAILURE);
//    }
//        while(!infile.eof()){
//                infile >> pp;
//                if(!isalpha(pp)){
//                        cout << pp<<endl;
//                }
//
//        }

        // generate the InvertedIndex when running the program
        InvertedIndex tempIndex;

        // need user to input the path of fileName.txt
        cout << "Welcome to Inverted Index created by Jason He."<<endl;
        string path;
        // /Users/yifuhe/Desktop/hw3/fileName.txt
    cout << "Enter the path of fileName.txt:" << endl;
    cin >> path;
        vector<string> fileName = getFileName(path);
        string path2;
        // /Users/yifuhe/Desktop/hw3/fin/
        cout << "Enter the path of datasets of file:" << endl;
        cin >> path2;
        for(string item: fileName){
                //cout << item << endl;
                tempIndex.addNewFile(path2,item);
        }


        cout << "-----------------------------------------------"<<endl;
        cout << "Instructions:"<<endl;
        cout << "This program will not differentiate capitalized letter."<<endl;
        int choice;
        while (true){
                cout << "-----------------------------------------------"<<endl;
                cout <<"1.Search a word, "<<endl;
                cout <<"2.Search a phrase, e.g. \"financial industry\""<<endl;
                cout <<"3.Search two words in the following format, e.g. \"future and optio
                cout <<"4.Wide card query, e.g.\"fin*\""<<endl;
                cout <<"5.Quit."<<endl;
                cout <<"-----------------------------------------------"<<endl;
                cout <<"Enter your choice: ";
                cin >> choice;
```

```
                if(choice == 1){
                        string word;
                        cout << "Enter the word (e.g. \"finance\"): ";
                        cin >> word;
                        tempIndex.searchWord(word);
                }else if(choice == 2){
                        cout << "Enter a phrase, e.g. \"financial industry\":";
                        string word,after;
                        cin >> word >> after;
                        tempIndex.searchPhrase(word, after);
                }else if(choice == 3){
                        cout << "Enter two words in the following format, e.g. \"future and
                        string word1,AND,word2;
                        cin >> word1 >> AND >> word2;
                        tempIndex.searchTwo(word1,word2);
                }else if(choice == 4){
                        cout << "Enter wide card query, e.g.\"fin*\":";
                        string word;
                        cin >> word;
                        tempIndex.searchWide(word);
                }else if(choice == 5){
                        cout << "Thanks for choosing the program. See you!" << endl;
                        break;
                }else{
                        cout << "Wrong choice! Enter the choice agaain" << endl;
                }
        }
        return 0;
}
```

3.2 FileAccess.hpp

```
//
//  FileAccess.hpp
//  hw3
//
//  Created by Jason Ha on 10/24/19.
//  Copyright © 2019 Jason He. All rights reserved.
//

#ifndef FileAccess_hpp
#define FileAccess_hpp

#include <stdio.h>
#include <vector>
#include <iostream>
```

```cpp
#include <string>
#include <filesystem>
#include <cstdlib>
#include <fstream>
using namespace std;

// read the directory of the folder and get all the file name
void generateFileName();
// read the fileName.txt and return a vector of paths.
vector<string> getFileName(string path);

#endif /* FileAccess_hpp */
```

3.3 FileAccess.cpp

```cpp
//
//  FileAccess.cpp
//  hw3
//
//  Created by Jason Ha on 10/24/19.
//  Copyright © 2019 Jason He. All rights reserved.
//

#include "FileAccess.hpp"
using namespace std;


void generateFileName(){
    string path;
    cout << "Enter the path of data file:" << endl;
    cin >> path;
    ofstream fileOutput{path+"/../fileName.txt",ios::out};
    if(!fileOutput){
        cerr << "File cannot be opened during generateFileName." << endl;
        exit(EXIT_FAILURE);
    }

    for (const auto& entry : std::__fs::filesystem::directory_iterator(path)){
        cout << entry.path().c_str() <<endl;
        //fileOutput << entry.path().c_str() << endl;
    }
    fileOutput.close();
}

// version 1
```

```
//vector<string> getFileName(){
//    string path;
//    cout << "Enter the path of fileName:" << endl;
//    cin >> path;
//    ifstream fileInput{path, ios::in};
//    string file;
//    vector<string> fileNames;
//    if(!fileInput){
//        cerr << "File cannot be opened during getFileName." << endl;
//        exit(EXIT_FAILURE);
//    }
//    while(!fileInput.eof()){
//        fileInput >> file;
//        fileNames.push_back(file);
//        //cout << file <<endl;
//    }
//    fileInput.close();
//    return fileNames;
//}

// version 2
vector<string> getFileName(string path){
    ifstream fileInput{path, ios::in};
    string file;
    vector<string> fileNames;
    if(!fileInput){
        cerr << "File cannot be opened during getFileName." << endl;
        exit(EXIT_FAILURE);
    }
    while(!fileInput.eof()){
        fileInput >> file;
        fileNames.push_back(file);
        //cout << file <<endl;
    }
    fileInput.close();
    return fileNames;
}
```

3.4 InvertedIndex.hpp

```
//
//  InvertedIndex.hpp
//  hw3
//
//  Created by Jason Ha on 10/26/19.
//  Copyright © 2019 Jason He. All rights reserved.
```

```cpp
//

#ifndef InvertedIndex_hpp
#define InvertedIndex_hpp

#include <iterator>
#include <string>
#include <stdio.h>
#include <vector>
#include <map>
#include "FileAccess.hpp"
using namespace std;


class Node{
private:
    string fileName;
    string word;
    string after;
    unsigned int count;

public:
    // constructor
    Node(const string& fileName_,const string& word_,const string& after_);

    // overload the operator == for class Node
    bool operator==(const Node& node2);

    // addcount
    void addcount(){
        count +=1;
    }
    // accessor
    string getfileName(){
        return fileName;
    }
    string getWord(){
        return word;
    }
    string getAfter(){
        return after;
    }
    unsigned int getCount(){
        return count;
    }
```

```
};

class InvertedIndex{
private:
    map<string,vector<Node>> Index;
    // add a Node in the Index, which will be used in addNewFile function
    void addNode(const string& fileName_,const string& word,const string& after);

public:
    // default constructor
    InvertedIndex(){};

    // member function
    // add a new file
    void addNewFile(const string& path,const string& filename);

    //search fingle word
    void searchWord(const string& word);

    // extra credits
    // search phrase(only two)
    void searchPhrase(const string& word,const string& after);
    // search two word not need to be adjacent
    void searchTwo(const string& word1,const string& word2);
    // wild card search
    void searchWide(const string& word);
};

#endif /* InvertedIndex_hpp */
```

3.5 InvertedIndex.cpp

```
//
//  InvertedIndex.cpp
//  hw3
//
//  Created by Jason Ha on 10/26/19.
//  Copyright © 2019 Jason He. All rights reserved.
//

#include "InvertedIndex.hpp"


Node::Node(const string& fileName_,const string& word_,const string& after_)
: fileName(fileName_)
, word(word_)
```

```
, after(after_)
, count(1)
{
}

bool Node::operator==(const Node& node2){
    if((fileName == node2.fileName) &&(word == node2.word) &&(after == node2.after)){
        return true;
    }
    return false;
}


void InvertedIndex::addNode(const string& path, const string& word, const string& after){
    if (Index.find(word) != Index.end()){//word exists in the Index
        // find whether the Node exist in the vector<Node>
        vector<Node>::iterator iter;
        Node temp_node(path,word,after);
        iter = find(Index[word].begin(),Index[word].end(),temp_node);
        if(iter != Index[word].end()){ // the temp_node exist in the vector
            (*iter).addcount();
        }else{ // the temp_node doesn't exist in the vector
            Index[word].push_back(temp_node);
        }
    }else{ // word doesn't exist in the Index, add word - vector into Index
        vector<Node> position{Node(path,word,after)};
        Index[word] = position;
    }
}


void InvertedIndex::addNewFile(const string& path, const string& filename){
    //cout << path<<endl;

    ifstream infile;
    infile.open(path + filename, ios::in);
    // set the ifstream to make it not skip the whitespace
    infile >> noskipws;

    // exit program if ifstream could not open file
    if(infile.fail()){
        cerr << "File cound not be opened in addNewfile" << endl;
        exit(EXIT_FAILURE);
    }

    // read char-by-char in order to check the type of char
```

```
    char letter;
    string word{""};
    string after{""};
    // put the first word into word
    while(!infile.eof()){
        infile >>letter;
        if(islower(letter)){
            word += letter;
        //}else if(ispunct(letter) || isspace(letter)){
        }else if(!isalpha(letter)){
            break;
        }else if(isupper(letter)){
            word += tolower(letter);
        }else{

        }
    }

    // access the word after then add the word into index
    while(!infile.eof()){
        infile >> letter;
        if(islower(letter)){
            after += letter;
        //}else if(ispunct(letter) || isspace(letter)){
        }else if(!isalpha(letter)){
        // punctuation followed by white space
            if(after == ""){
                continue;
            }
        //add the word in index
            addNode(filename, word, after);
        // reset after and assign after to word
            word = after;
            after = "";
            //cout <<". ";
        }else if(isupper(letter)){
            after += tolower(letter);
        }
    }
    //cout <<"E ";

    // access the last word and save it
    addNode(filename, word, after);
//    if (isalpha(letter)){
//        addNode(filename, word, after);
//        //cout << "1";
```

```
//    }else{
//        addNode(filename, word, after);
//        //cout <<"2";
//    }

}

void InvertedIndex::searchWord(const string& word){
    // use temp list to count
    map<string,int> temp;
    // use the rank list to cout
    vector<pair<int,string>> rank;

    if(Index.find(word) != Index.end()){ // find the word in the Index, begin to count and r
        cout << word << " is in our system."<< endl;
        // access the vector<Node> and count the word according to filename
        for(Node item : Index[word]){
            // check whether fileName already exist in the
            if(temp.find((item.getfileName())) != temp.end()){
                temp[item.getfileName()] += item.getCount();
            }else{
                temp[item.getfileName()] = item.getCount();
            }
        }
        // search the temp and convert it into rank list
        map<string,int>:: iterator iter;
        iter = temp.begin();
        while(iter!=temp.end()){
            //cout << iter->second<<iter->first;
            rank.push_back(make_pair(iter->second, iter->first));
            iter++;
        }
        // search the rank list and cout
        sort(rank.begin(), rank.end());
        // ascending
//        for(pair<int,string> item: rank){
//            cout << item.second <<"  "<< item.first<<endl;
//        }
        // descending
        for(int i = static_cast<int>(rank.size())-1 ; i >= 0 ; i--){
            cout << rank[i].second <<"  "<< rank[i].first<<endl;
        }
    }else{ // the word no in the Index
        cout << "There is no "<< word <<" in our system. "<< endl;
    }
}
```

```
// bonus 1
void InvertedIndex::searchPhrase(const string& word,const string& after){
    // use temp list to count
    map<string,int> temp;
    // use the rank list to cout
    vector<pair<int,string>> rank;


    if(Index.find(word) != Index.end()){ // find the word in the Index, begin to count and t
        // access the vector<Node> and count the word according to filename
        for(Node item : Index[word]){
            if(item.getAfter() == after){
                // check whether fileName already exist in the
                if((temp.find((item.getfileName())) != temp.end()) ){
                    temp[item.getfileName()] +=item.getCount();
                }else{
                    temp[item.getfileName()] = item.getCount();
                }
            }
        }
        // check whether temp is empty
        if(temp.empty()){
            cout << "There is no "<< word <<" " << after <<" in our system. "<< endl;
        }else{
            cout << "The phrase of \""<<word <<" " << after << " \" is in our system:"<< enc
            // search the temp and convert it into rank list
            map<string,int>:: iterator iter;
            iter = temp.begin();
            while(iter!=temp.end()){
                //cout << iter->second<<iter->first;
                rank.push_back(make_pair(iter->second, iter->first));
                iter++;
            }
            // search the rank list and cout
            sort(rank.begin(), rank.end());
            // ascending
        //        for(pair<int,string> item: rank){
        //            cout << item.second <<"  "<< item.first<<endl;
        //        }
            // descending
            for(int i = static_cast<int>(rank.size())-1 ; i >= 0 ; i--){
                cout << rank[i].second <<"  "<< rank[i].first<<endl;
            }
        }
```

```
    }else{ // the word no in the Index
        cout << "There is no "<< word <<" " << after <<" in our system. "<< endl;
    }
}

// bouns 2
void InvertedIndex::searchTwo(const string& word1,const string& word2){
    // use temp list to count
    map<string,int> temp1;
    map<string,int> temp2;
    // use the rank list to cout
    vector<pair<int,string>> rank;
    // count word1 in each file
    if(Index.find(word1) != Index.end()){ // find the word in the Index, begin to count and
        // access the vector<Node> and count the word according to filename
        for(Node item : Index[word1]){
            // check whether fileName already exist in the
            if(temp1.find((item.getfileName())) != temp1.end()){
                temp1[item.getfileName()] += item.getCount();
            }else{
                temp1[item.getfileName()] = item.getCount();
            }
        }
    }else{
        cout << "There is no pair of \""<< word1 <<" and " << word2<<" \" in our system. "<<
    }
    // use temp2 to compare the number of word1 and word2 in the same file and choose the s
    temp2 = temp1;
    // count word2 in each file
    if(Index.find(word2) != Index.end()){ // find the word in the Index, begin to count and
    // access the vector<Node> and count the word according to filename
        for(Node item : Index[word2]){
            // check whether fileName already exist in the
            if(temp2.find((item.getfileName())) != temp2.end()){
                temp2[item.getfileName()] -= item.getCount();
            }
        }
    }else{
        cout << "There is no pair of \""<< word1 <<" and " << word2<<"\" in our system. "<<
    }

    // search the temp and convert it into rank list
    map<string,int>:: iterator iter;
    iter = temp2.begin();
    while(iter!=temp2.end()){
        // N(word2) is more than or equal to N(word1), then choose N(word1)
```

```
            string cursor = iter->first;
            if(iter->second <= 0){
                rank.push_back(make_pair(temp1[cursor], cursor));
            }else if(iter->second < temp1[cursor]){ // N(word2) is less than or equal to N(word
                rank.push_back(make_pair(temp2[cursor], cursor));
            }else{//no N(word2) equal to 0, eliminate this case
            }
            iter++;
        }
        if(rank.empty()){
            cout << "There is no pair of \""<< word1 <<" and " << word2<<"\" in our system. "<<
        }else{
            cout << "The pair of \"" << word1 <<"and"<<word2<<"\" exists in our system:";
            // search the rank list and cout
            sort(rank.begin(), rank.end());
            // ascending
//          for(pair<int,string> item: rank){
//              cout << item.second <<"  "<< item.first<<endl;
//          }
            // descending
            for(int i = static_cast<int>(rank.size())-1 ; i >= 0 ; i--){
                cout << rank[i].second <<"  "<< rank[i].first<<endl;
            }
        }
    }
    // bonus 3 : wild card search
    void InvertedIndex::searchWide(const string& word){
        // target is the wild card word
        int length =static_cast<int>(word.size())-1;
        string target = word.substr(0,length);

        map<string,int> temp;
        vector<pair<int,string>> rank;

        // search the Index to find wild card word
        map<string,vector<Node>>::iterator iter;
        iter=Index.begin();
        while(iter!=Index.end()){
            string cursor = iter->first;
            // check whether the word cursor is satisfied with the wild card condition
            if(cursor.size()<length){
                iter++;
                continue;
            }
            if(cursor.substr(0,length) == target){
                for(Node item : Index[cursor]){
```

```
                // check whether fileName already exist in the
                if(temp.find((item.getfileName())) != temp.end()){
                    temp[item.getfileName()] += item.getCount();
                }else{
                    temp[item.getfileName()] = item.getCount();
                }
            }
        }else{
            // the word cursor is not satisfied with the wild card condition, go to next one
        }
        iter++;
    }
    if(temp.empty()){
        cout <<"wild card \""<<word<<"\" is not in the system."<<endl;
    }else{
        cout <<"wild card \""<<word<<"\" exists in the system."<<endl;
        // search the temp and convert it into rank list
        map<string,int>:: iterator iter2;
        iter2 = temp.begin();
        while(iter2!=temp.end()){
            //cout << iter->second<<iter->first;
            rank.push_back(make_pair(iter2->second, iter2->first));
            iter2++;
        }
        // search the rank list and cout
        sort(rank.begin(), rank.end());
        // ascending
//        for(pair<int,string> item: rank){
//            cout << item.second <<"  "<< item.first<<endl;
//        }
        // descending
        for(int i = static_cast<int>(rank.size())-1 ; i >= 0 ; i--){
            cout << rank[i].second <<"  "<< rank[i].first<<endl;
        }
    }
}
```