Advanced Data Structures and Algorithm Analysis

Project 1: Roll Your Own Mini Search Engine

Date: 2024-09-30

2024-2025 Autumn&Winter Semester

Table of Content

Chapter 1: Introduction

1.1 Problem Description

The project required us to create a **mini search engine** which can handle inquiries over “The Complete Works of William Shakespeare”.

Here are some specific requirements:

- Run a word count over the Shakespeare set, extract all words from the documents by word stemming and try to identify the stop words.
- Create a customized inverted index over the Shakespeare set with word stemming. The stop words identified must not be included.
- Write a query program on top of the inverted file index, which will accept a user-specified word (or phrase) and return the IDs of the documents that contain that word.
- Run tests to show how the thresholds on query may affect the results.

1.2 Purpose of Report

- Show the details of the implementation of the mini search engine by showcasing essential data structures and algorithms.
- Demonstrate the correctness and efficiency of the program by analysis based on testing data and diagrams.
- Summarize the whole project, analyze the pros and cons of the mini search engine, and put forward the prospect of further improvement.

1.3 Backgound of Data Structures and Algorithms

1. **B+ Trees**: It’s an improved version of search trees, widely used in the relational database and file management in operating systems. We will use this data structure to store and access to the inverted index.
2. **Hashing**: Hash tables have an excellent performance in searching data(only cost O(1) time), hence we take advantage of this data structure for finding stopwords when building an inverted index.

3. **Queue**: The Queue ADT is one of the most basic data structrues used in printing the B+ tree, storing the positions for terms, etc.

Chapter 2: Data Structure / Algorithm Specification

2.1 Algorithm Architecture

The overall algorithm architecture in the program is shown below:

In the following sections, I will introduce these algorithms from top to down, but with some slight adjustement, in the hope that you can gain a deeper insight into my whole program.

2.2 The Main Program

2.3 Word Count

2.4 Stop Words

2.5 Word Stemming

We tap into the codes from a GitHub repository called "OleanderStemmingLibrary" by the author Blake-Madden. The codes are stored in the directory `code/wordStem`, and the link of repository is listed in the **References** section below.

> Warning
>
> We have to admitted that this word stemming program is kind of clumsy, especially for nouns, because the program will continue doing word stemming even though the word is in the simplest and the most common form. For example, for a simple English word "orange", it will convert it to another word "orang", which means "gorilla".
>
> Owing to the time and capability limitation, we couldn't find a better word stemming programs in C/C++ version or convert other languages version to C/C++ version. We hope that we will use a smarter word stemming program in the foreseeable future.

2.6 Inverted Index

Maybe this is the most complicated part of the whole program, because in this part we have a relatively complex algorithm architecture, and we use a couple of data structrues and algorithms, such as B+ trees, implicit queue ADT and linked list ADT. Here is the diagram of the functions used in the inverted index:

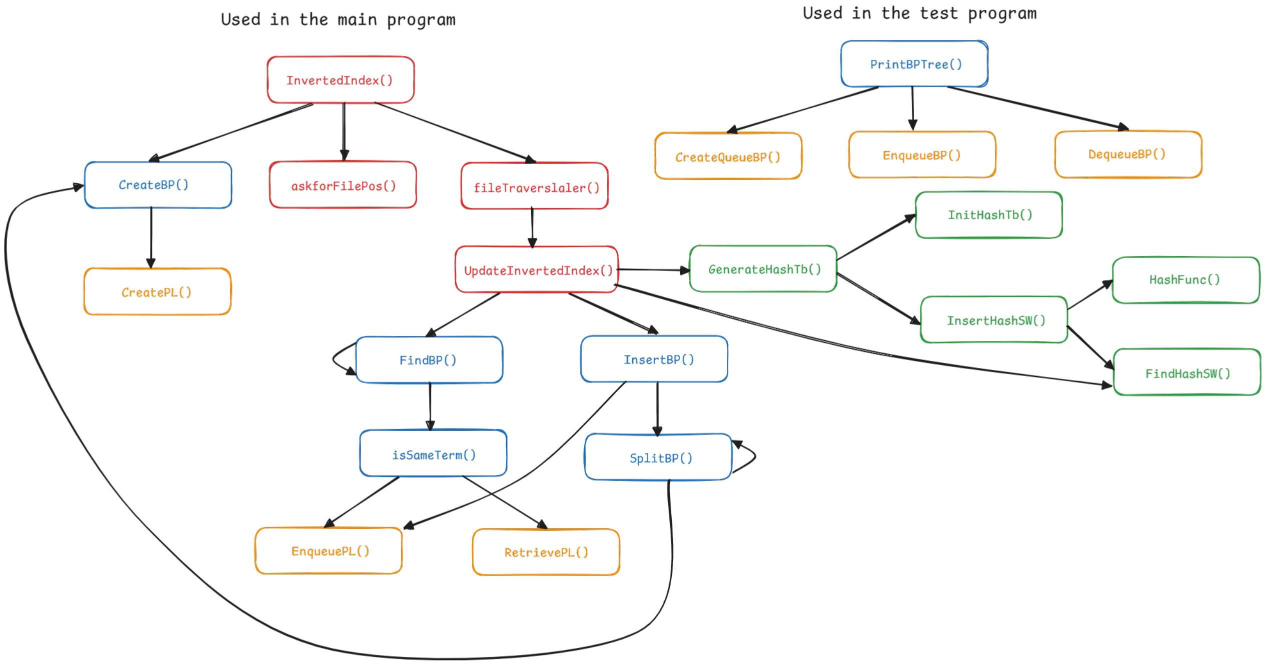


Figure 1: Relation diagram of all functions for inverted index

- Red: Overall Functions
- Blue: B+ Tree Operations
- Green: Hashing Functions
- Yellow: Other Functions

We'll introduce these functions in detail below.

2.6.1 Overall Functions

(1) InvertedIndex

Function: The highest-level function, which users can call it directly.

Inputs:

- *isTest*: `-t` or `--test` mode, just use one particular file

- *containStopWords*: `-s` or `--stopwords` mode, contain stop words when building inverted index

Outputs:

- *InvIndex*: A B+ tree containing the inverted index

Procedure: InvertedIndex(*isTest*: **bool**, *containStopWords*: **bool**)

```
Begin
    InvIndex ← CreateBP()
    askforFilePos(dir, fname, isTest)
    // dir: directory name, fname: filename
    InvIndex ← fileTraversaler(InvIndex, dir, fname, isTest, containStopWords)
    if InvIndex→size > 0 then
        print("Build successfully!")
    else
        print("Fail to build an inverted index!")
    endif
    return InvIndex
End
```

(2) askforFilePos

Function: Ask for the position of the directory or file.

Inputs:

- *dir*: directory name
- *fname*: file name
- *isTest*: `-t` or `--test` mode, just use one particular file

Outputs: None, but will update either *dir* or *fname*

Procedure: askforFilePos(dir: **string**, fname: **string**, isTest: **boolean**)

```
Begin
    if isTest is true then
        print("Now testing the correctness of inverted Index:")
        print("Please input the name of the input sample file:")
        input("Name:", fname)
    else
        print(“Now building an inverted Index:”)
        print(“Please input the directory of the documents:”)
        input(“Path:”, dir)
    endif
End
```

(3) fileTraversaler

Function: Make a traversal of all files(or a single file) and build the inverted index from them(or it).

Inputs:

- *T*: A B+ tree containing the inverted index
- *dir*: directory name
- *fname*: file name
- *isTest*: `-t` or `--test` mode, just use one particular file
- *containStopWords*: `-s` or `--stopwords` mode, contain stop words when building inverted index

Outputs: An updated B+ tree *T*

Procedure: fileTraversaler(T: **BplusTree**, *dir*: **string**, *fname*: **string**, *isTest*: **boolean**, *containStopWords*: **boolean**)

```
Begin
    docCnt = 0 // Count the number of documents and act as the index of the documents at the same time
    if isTest is false then
        if dir exists then
            for file in the dir directory do
                filename ← the name of file // string
                docNames[docCnt] ← filename
                // docNames: an array containing names of documents(global variable)
                wholePath ← dir + “/” + filename
            // wholePath: the complete path of the file to be read
            end
            closefile(fp)
        else
            Error("Could not open directory!")
        endif
    else
        docNames[docCnt] ← fname
        dir ← DEFAULTFILEPOS // Constant: "tests"(string)
        wholePath ← dir + "/" + fname
    endif
    fp ← openfile(wholePath, "r") // read mode
    // fp: the pointer to the file
    T ← UpdateInvertedIndex(T, docCnt, fp, containStopWords)
    return T
End
```

(4) UpdateInvertedIndex

Function: Update the Inverted Index while reading a new document.

Inputs:

- *T*: A B+ tree containing the inverted index
- *docCnt*: the index of the document
- *fp*: pointer to the file
- *containStopWords*: `-s` or `--stopwords` mode, contain stop words when building inverted index

Outputs: An updated B+ tree *T*

Procedure: UpdateInvertedIndex(T: **BplusTree**, *docCnt*: **integer**, *fp*: **filePointer**, *containStopWords*: **boolean**)

```
Begin
    H ← GenerateHashTb()
    while reading texts in the file pointed by fp do
        if find an English word then do
            term ← the English word
            if containStopWords is false and FindHashSW(term, H,
            true) >= 0 then
```

```
                continue
            endif
            term ← WordStemming(term)
            isDuplicated ← false
            nodebp ← FindBP(term, docCnt, T, isDuplicated)
            if isDuplicated is false then
                T = InsertBP(term, docCnt, nodebp, T)
            endif
        else
            continue searching for next English word.
        endif
    end
    return T
End
```

2.6.2 B+ Tree Operations

Note

- The order of our B+ tree is 4.

(1) CreateBP

Function: Create a B+ tree.

Inputs: None

Outputs: A new and initialized B+ Tree

Procedure: CreateBP()

```
Begin
    Allocate a memory block for new B+ tree T
    for all data and children in T do
        Allocate memory blocks for term and poslist of the data, and
        children
        // Use CreatePL() to intialize the poslist
    end
    $T \rightarrow size \leftarrow 0$
    $T \rightarrow childrenSize \leftarrow 0$
    $T \rightarrow parent \leftarrow NULL$
    return $T$
End
```

(2) FindBP

Function: Find a term in B+ tree.

Inputs:

- *term*: term
- *docCnt*: the index of the document
- *T*: inverted index
- *flag*: true if the term is found, false otherwise
- *isSearch*: mark the find mode(-f or –find)

Outputs: the (possibly updated) B+ tree T or recursively call itself again

Procedure: FindBP(*term*: **string**, *docCnt*: **integer**, *T*: **BplusTree**, *flag*: **booleanPointer**, *isSearch*: **boolean**)

```
Begin
    if $T \rightarrow$childrenSize $= 0$ then
        isSameTerm($term$, $docCnt$, $T$, $flag$, $isSearch$)
        return $T$
    endif
    $pos \leftarrow -1$
    for $i$ in range(0, $T \rightarrow size$) do // not contains $T \rightarrow size$
        if $term$ has less lexicographical order than $T \rightarrow data[i] \rightarrow term$ then
            $pos \leftarrow i$
            break
        endif
    end
    if $pos = -1$ then
        $pos \leftarrow i$
    endif return FindBP($term$, $docCnt$, $T \rightarrow children[pos]$, $isSearch$)
```

```
End
```

(3) isSameTerm

Function: Check if the term exists in the B+ tree.

Inputs:

- *term*: term
- *docCnt*: the index of the document
- *nodebp*: the appropriate node where the term may exists or will exists after insertion
- *flag*: true if the term is found, false otherwise
- *isSearch*: mark the find mode(-f or –find)

Outputs: None, but may update the flag and print some information regarding *term*

Procedure: isSameTerm(*term*: **string**, *docCnt*: **integer**, *nodebp*: **NodeBP**, *flag*: **booleanPointer**, *isSearch*: **boolean**)

```
Begin
    if $nodebp \to size > 0$ then
        for $i$ in range(0, $T \to size$) do // not contains $T \to size$
            if $term = nodebp \to data[i] \to term$ then
                if $isSearch$ is false then
                    EnqueuePL($docCnt$, $nodebp \to data[i] \to poslist$)
                else
                    $poslist \leftarrow nodebp \to data[i] \to poslist$
                    $size \leftarrow poslist \to size$
                    $cnt \leftarrow 0$
                    print("Successfully find the word!")
                    print("The word was found in files below:")
                    $posArr \leftarrow$ RetrievePL($poslist$)
                    for $j$ in range(0, $size$) do // not contains $size$
                        if $posArr[j][1] <= 1$ then
                            print("{$docNames[posArr[j][0]]$}: {$posArr[j][1]$} time")
                        else
```

```
                            print("{docNames[posArr[j][0]]}: {posArr[j][1]} times")
                        endif
                        cnt ← cnt + posArr[j][1]
                    end
                    print("Frequency: {cnt}")
                    print("——————————————————")
                endif
                flag ← true
                break
            endif
        end
    endif
End
```

(4) InsertBP

Function: Insert a term into the B+ tree.

Inputs:

- *term*: term
- *docCnt*: the index of the document
- *nodebp*: the appropriate node where the term will be inserted
- *Tree*: B+ tree containing the inverted index

Outputs: the updated B+ tree *Tree*

Procedure: InsertBP(*term*: **string**, *docCnt*: **integer**, *nodebp*: **NodeBP**, *Tree*: **BplusTree**)

```
Begin
    nodebp→data[nodebp→size]→term ← term
    EnqueuePL(docCnt, nodebp→data[nodebp→size]→poslist)
    nodebp→size ← nodebp→size + 1
    Sort(nodebp→data)
    Tree ← SplitBP(nodebp, Tree)
    return Tree
End
```

(5) SplitBP

Function: Split the node when the node is full.

Inputs:

- *nodebp*: the appropriate node where the term will be inserted
- *Tree*: B+ tree containing the inverted index

Outputs: The updated B+ tree *Tree*, or recursively call itself to split *nodebp*'s parent node

Procedure: SplitBP(*nodebp*: **NodeBP**, *Tree*: **BplusTree**)

```
Begin
    // ORDER: (constant)the order of B+ trees if (nodebp→childrenSize = 0 and nodebp→size <= ORDER) or (nodebp→childrenSize > 0 and nodebp→size < ORDER) then
        return Tree
    endif

    // lnodebp, rnodebp: the left and right part of the split node
    // tmpNodebp: store the node temporarily
    // parent: the parent node of nodebp
    // cut: the position of the middle data

    parent ← nodebp→parent
    if parent = NULL then
        tmpNodebp ← CreateBP()
        Allocate memory for parent
        Tree ← parent ← tmpNodebp
    endif
    lnodebp ← CreateBP()
    rnodebp ← CreateBP()
    lnodebp→parent ← rnodebp→parent ← parent
    if nodebp→childrenSize = 0 then
        cut ← LEAFCUT // constant: (ORDER / 2 + 1)
        for i in range(0, cut) do // not contains cut
```

```
            lnodebp→data[i] ← nodebp→data[i]
        end
        lnodebp→size ← cut
        for j in range(cut, nodebp→size) do // not contains nodebp→size
            rnodebp→data[j - cut] ← nodebp→data[j]
        end
        rnodebp→size ← nodebp→size - cut
    else
        cut ← NONLEAFCUT // constant: (ORDER / 2)
        for i in range(0, cut + 1) do // not contains cut + 1
            if i ≠ cut then
                lnodebp→data[i] ← nodebp→data[i]
            endif
            lnodebp→children[i] ← nodebp→children[i]
            lnodebp→children→parent ← lnodebp
        end
        lnodebp→size ← cut
        lnodebp→childrenSize ← cut + 1
        for j in range(cut + 1, nodebp→size) do // not contains nodebp→size
            rnodebp→data[j - cut - 1] ← nodebp→data[j]
        end
        for j in range(cut + 1, nodebp→childrenSize) do // not contains nodebp→childrenSize
            rnodebp→children[j - cut - 1] ← nodebp→children[j]
            rnodebp→children[j - cut - 1]→parent ← rnodebp
        end
        rnodebp→size ← nodebp→size - cut - 1 rnodebp→childrenSize ← nodebp→childrenSize - cut - 1
    end
    parent→data[parent→size] ← nodebp→data[cut]
    parent→size ← parent→size + 1
    if parent→childrenSize > 0 then
```

```
        for $i$ in range(0, $parent \to childrenSize$) do // not contains $parent \to childrenSize$
            if $parent \to children[i] = nodebp$ then
                $parent \to children[i] \leftarrow lnodebp$
                break
            endif
        end
    else
        $parent \to children[parent \to childrenSize] \leftarrow lnodebp$
        $parent \to childrenSize \leftarrow parent \to childrenSize + 1$
    endif
    $parent \to children[parent \to childrenSize] \leftarrow rnodebp$
    $parent \to childrenSize \leftarrow parent \to childrenSize + 1$
    Sort($parent \to data$)
    Sort($parent \to children$)
    $Tree \leftarrow$ SplitBP($parent$, $Tree$)
    return $Tree$
End
```

(6) PrintBPTree

Function: Print the B+ tree(level-order traversal).

Inputs:

- T: B+ tree containing the inverted index

Outputs: None, but will print the whole B+ tree

Procedure: PrintBPTree(T: **BplusTree**)

```
Begin
    print("B+ Tree of Inverted Index:")
    $q \leftarrow$ CreateQueueBP()
    EnqueueBP($T$, $q$)
    EnqueueBP($NULL$, $q$)
    while $q \to size > 0$ do
        $nodebp \leftarrow$ DequeueBP(q)
        if $nodebp$ is $NULL$ then
```

```
            change to a newline
            if $q \to size > 0$ then
                EnqueueBP($NULL$, $q$)
            endif
        else
            print("[")
            for $i$ in range(0, $nodebp \to size$) do // not contains $nodebp \to size$
                if $i = 0$ then
                    print($nodebp \to data[i] \to term$)
                else
                    print(", {$nodebp \to data[i] \to term$}")
                endif
            end
            print("]")
        endif
        if nodebp is not $NULL$ then
            for $i$ in range (0, $nodebp \to childrenSize$) do // not contains $nodebp \to childrenSize$
                EnqueueBP($nodebp \to children[i]$, $q$)
            end
        endif
    end
End
```

2.6.3 Hashing Operations

(1) GenerateHashTb

Function: Build a hash table.

Inputs: None

Outputs: A new hash table H, containing stopwords from the file

Procedure: GenerateHashTb()

```
Begin
```

```
    H ← InitHashTb()
    fname ← STOPWORDPATH // constant: “stop_words.txt"
    fp ← openfile(fname, “r”) // read mode
    if fp is NULL then
        Error(“Fail to open the file of stopwords!”)
    endif
    while reading texts in the file pointed by fp do
        if find an English word then do
            term ← the English word
            InsertHashSW(term, H)
        endif
    end
    closefile(fp)
    return H
End
```

(2) InitHashTb

Function: Initialization of the hash table.

Inputs: None

Outputs: A new initialized hash table

Procedure: InitHashTb()

```
Begin
    Allocate memory block for H // HashTb
    if H is NULL then
        Error(“Fail to create a hash table for stopwords!”)
    end
    H→size ← STOPWORDSUM // maximum size
    for i in range(0, H→size) do // not contains H→size
        Allocate memory block for H→data[i]
        if H→data[i] is NULL then
            Error(“Fail to create a hash table for stopwords!”)
        end
        H→data[i]→stopword
```

```
            $H \to data[i] \to info \leftarrow Empty$ // constant: 0
        end
        return $H$
End
```

(3) FindHashSW

Function: Find the stopwords or other words in the hash table.

Inputs:

- *stopword*: stop word
- *H*: hash table containing the stop words
- *justSearch*: find the term without subsequent insertion

Outputs: A appropriate position *pos* in hash table for stopword, or just search the term in the hash table

Procedure: FindHashSW(stopword: **string**, H: **HashTb**, justSearch: **boolean**)

```
Begin
    $collisionNum \leftarrow 0$
    $pos \leftarrow$ HashFunc($stopword$, $H \to size$)
    if $justSearch$ is true and ($H \to data[pos] \to info = Empty$ or $H \to data[pos] \to stopword = stopword$) then
        return $-1$
    endif
    while $H \to data[pos] \to info \neq Empty$ and $H \to data[pos] \to stopword = stopword$ do
        $collisionNum \leftarrow collisionNum + 1$ $pos \leftarrow pos + 2$ * $collisionNum$ - 1
        if $pos >= H \to size$ then
            $pos \leftarrow pos$ - $H \to size$
        endif
    end
    return $pos$
End
```

(4) InsertHashSW

Function: Insert a new stopword in hash table.

Inputs:

- *stopword*: stop word
- *H*: hash table containing the stop words

Outputs: None, but will update the hash table *H*

Procedure: InsertHashSW(*stopword*: **string**, *H*: **HashTb**)

```
Begin
    $pos \leftarrow$ FindHashSW($stopword$, $H$, false)
    // Legitimate: (constant) 1
    if ($H \rightarrow data[pos] \rightarrow info \neq Legitimate$) then
        $H \rightarrow data[pos] \rightarrow info \leftarrow Legitimate$
        $H \rightarrow data[pos] \rightarrow stopword \leftarrow stopword$
    endif
End
```

(5) HashFunc

Function: Hashing function.

Inputs:

- *stopword*: stop word
- *size*: the maximum size of the hash table

Outputs: A hash value to *stopword*

Procedure: HashFunc(*stopword*: **string**, *size*: **integer**)

```
Begin
    $val \leftarrow 0$
    for each character $ch$ in $stopword$ do
        $val = (val << 5) +$ integer($ch$)
    end
    return $val \% size$
End
```

2.6.4 Other Functions

(1) CreateQueueBP

Function: Create the queue

Inputs: None

Outputs: A new queue Q

Procedure: CreateQueueBP()

```
Begin
    Allocate memory block for the queue $Q$
    $Q \rightarrow size \leftarrow 0$
    $Q \rightarrow front \leftarrow Q \rightarrow rear \leftarrow 0$
    return $Q$
End
```

(2) EnqueueBP

Function: Put the node of B+ tree into the queue.

Inputs:

- *nodebp*: the newly added node
- Q: the queue

Outputs: None, but will update Q

Procedure: EnqueueBP(*nodebp*: **NodeBP**, Q: **QueueBP**)

```
Begin
    if $Q \rightarrow size >= SIZE$ then
        Error("Full B+-tree-item queue!")
    endif
    $Q \rightarrow data[Q \rightarrow rear] \leftarrow nodebp$
    $Q \rightarrow rear \leftarrow Q \rightarrow rear + 1$
    $Q \rightarrow size \leftarrow Q \rightarrow size + 1$
End
```

(3) DequeueBP

Function: Get the front node and delete it from the queue.

Inputs:

- Q: the queue

Outputs: the front node *returnNodeBP*

Procedure: DequeueBP(Q: **QueueBP**)

```
Begin
    if $Q \to size = 0$ then
        Error(“Empty B+-tree-item queue!”)
    endif
    $returnNodeBP \leftarrow Q \to data[Q \to front]$
    $Q \to front \leftarrow Q \to front + 1$
    $Q \to size \leftarrow Q \to size$ - 1
    return $returnNodeBP$
End
```

(4) CreatePL

Function: Create the poslist

Inputs: None

Outputs: A new PosList L

Procedure: CreatePL()

```
Begin
    Allocate memory blocks for $L$(PosList), $L \to front$(PosData), $L \to rear$(PosData)
    $L \to size \leftarrow 0$
    $L \to front \leftarrow L \to rear$
    $L \to rear \to pos \leftarrow -1$
    return $L$
    End
```

(5) EnqueuePL

Function: Add new position.

Inputs:

- *pos*: the position
- *L*: the position list

Outputs: None, but will update *L*

Procedure: EnqueuePL(*pos*: **integer**, *L*: **PosList**)

```
Begin
    if $L \to rear \to pos \neq pos$ then
        Allocate memory block for $tmp$(PosData)
        if $tmp$ is $NULL$ then
            Error("Fail to create a new position data!")
        endif
        $tmp \to pos \leftarrow pos$
        $tmp \to time \leftarrow 1$
        $tmp \to next \leftarrow L \to rear \to next$
        $L \to rear \to next \leftarrow tmp$
        $L \to rear \leftarrow tmp$
        $L \to size \leftarrow L \to size + 1$
    else
        $L \to rear \to time \leftarrow L \to rear \to time + 1$
    endif
End
```

(6) RetrievePL

Function: Retrieve all position in the list.

Inputs:

- *L*: the position list

Outputs: An 2D array *posArr* containing the all position in *L*, and each data contains two attributes: document index and the frequency in that document

Procedure: RetrievePL(*L*: **PosList**)

```
Begin
    if $L \to size = 0$ then
        Error("Empty position-data queue!")
    endif
    Allocate memory block for $posArr$ $cur \leftarrow L{\leftarrow}front{\leftarrow}next$
    $i \leftarrow 0$
    while $cur \neq NULL$ do
        $posArr[i][0] \leftarrow cur{\rightarrow}pos$
        $posArr[i][1] \leftarrow cur{\rightarrow}time$
        $cur \leftarrow cur{\rightarrow}next$
        $i \leftarrow i + 1$
    end
    return $posArr$
End
```

2.7 Query

Chapter 3: Testing Results

3.1 Inverted Index

To verify the correctness of our inverted index, we have devised several tests from different aspects. Here is the **purpose** of each test:

- Check if every word in document(s) is inserted into the inverted index correctly.
- Check if the inverted index can kick off all stopwords.
- Build an inverted index from a single file, or a directory with a bunch of files.

3.1.1 Word Insertion Test

We have two method to accomplish the first purpose: printing the whole inverted index(when the size is small), and finding the words existing in the inverted index(if words were correctly inserted).

(1) Printing the inverted index

Case 1: very simple example

```
$ ./invIndexTest -t -p
Now testing the correctness of inverted Index:
Please input the name of the input sample file:
```

```
Name: input1.txt
Build successfully!
B+ Tree of Inverted Index:
[beauti, ice, peach]
[appl, are, banana][beauti, cherri][ice, icecream, orang][peach,
pear, strawberri, watermelon]
```

input1.txt

```
ice
strawberry
orange
banana
peach
apple
pear
watermelon
cherry icecream you are beautiful
```

Case 2: simple example

```
$ ./invIndexTest -t -p
Now testing the correctness of inverted Index:
Please input the name of the input sample file:
Name: input2.txt
Build successfully!
B+ Tree of Inverted Index:
[et, lorem, nullam]
[consectetur, dolor, elit][id, ipsum][nec][pretium, sed, ut]
[adipisc, amet, at, congu][consectetur, consequat, dapibus, diam]
[dolor, e, eget][elit, erat][et, etiam, facilisi, fringilla][id,
interdum][ipsum, lacus, lectus][lorem, metus, mi][nec, nulla]
[nullam, orci, pellentesqu][pretium, purus, rhoncus][sed, sit,
sollicitudin, tincidunt][ut, vita]
```

input2.txt

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec erat sed nulla rhoncus dapibus et at lectus. Etiam in congue diam, ut interdum metus. Nullam pretium orci id mi pellentesque, vitae consequat lacus tincidunt. Pellentesque fringilla purus eget nulla facilisis sollicitudin.

(2) Finding words in the inverted index

Note

- This test is just used in checking the correctness of word insertion, which is similar to a simple query function, but the implementation is totally different from our formal query program, so you shouldn't mix them together.
- The texts in the following tests are too long, therefore we won't show these text in our report, but you can see them in the files positioned in the directory called `code/tests`.

Case 3: intermediate-level example

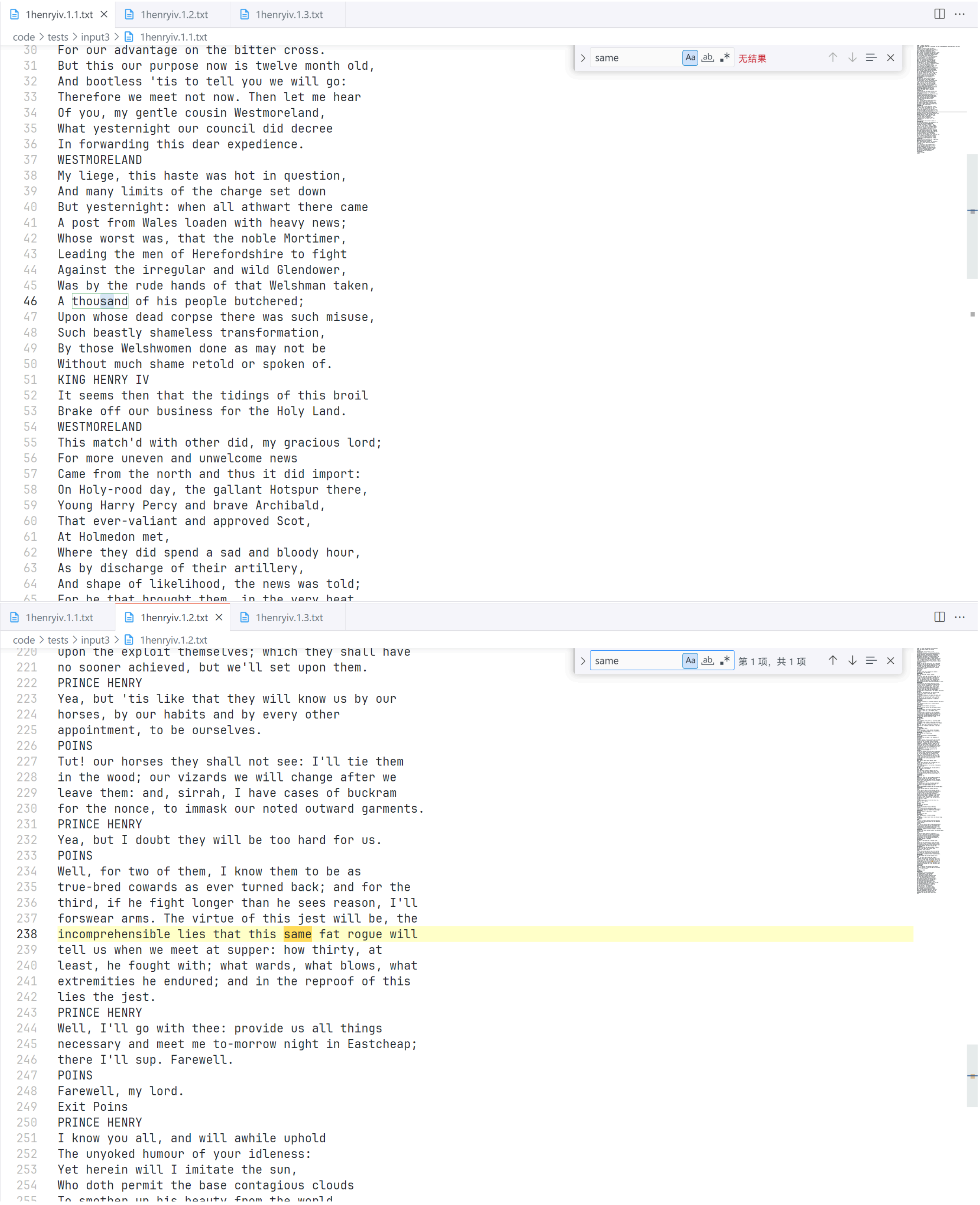
```
 ./invIndexTest -f=3
Now building an inverted Index:
Please input the directory of the documents:
Path: tests/input3
Build successfully!

Finding Words Mode(only supports single word finding):
Find 1: same
Successfully find the word!
The word was found in files below:
1henryiv.1.2.txt: 1 time
1henryiv.1.3.txt: 4 times
Frequency: 5
-----------------------------------
Find 2: star
Successfully find the word!
The word was found in files below:
1henryiv.1.2.txt: 1 time
Frequency: 1
-----------------------------------
```

```
Find 3: fantastic
Sorry, no such word in the inverted index!
-----------------------------------
```

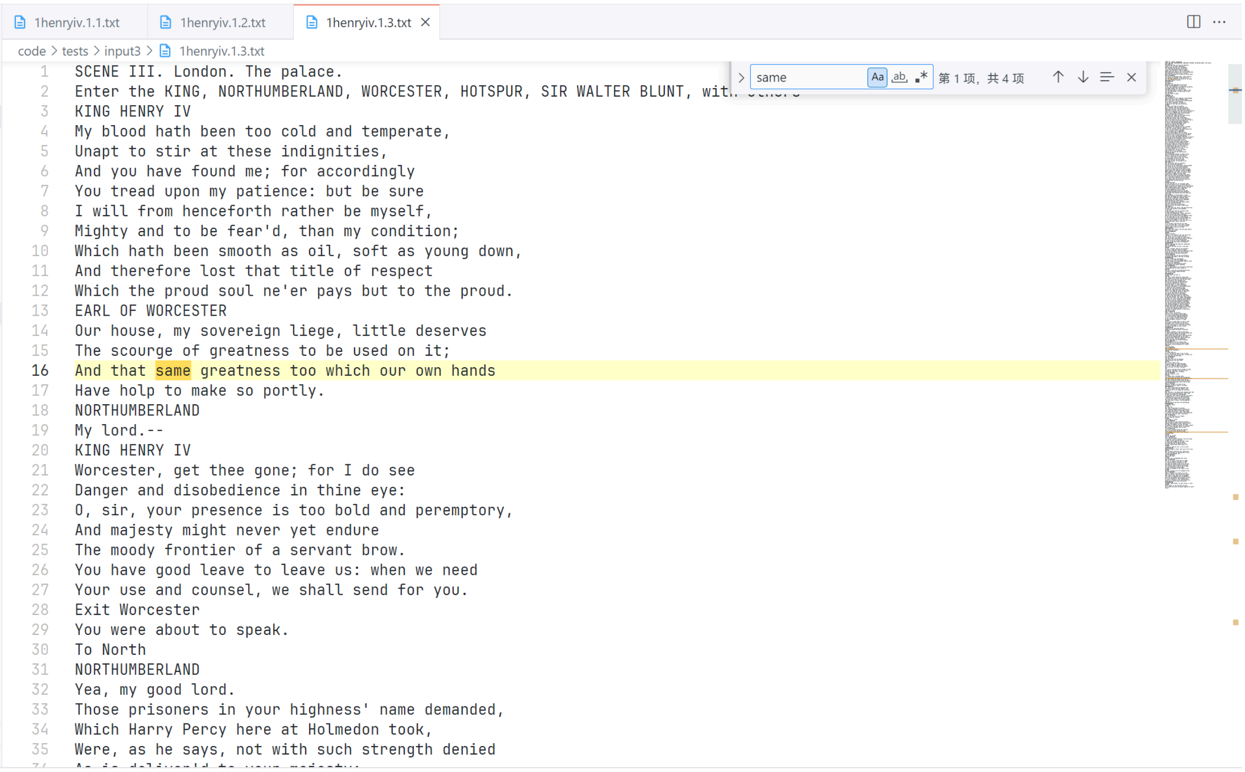
Verification by using finding function in Visual Studio Code

Find 1: same



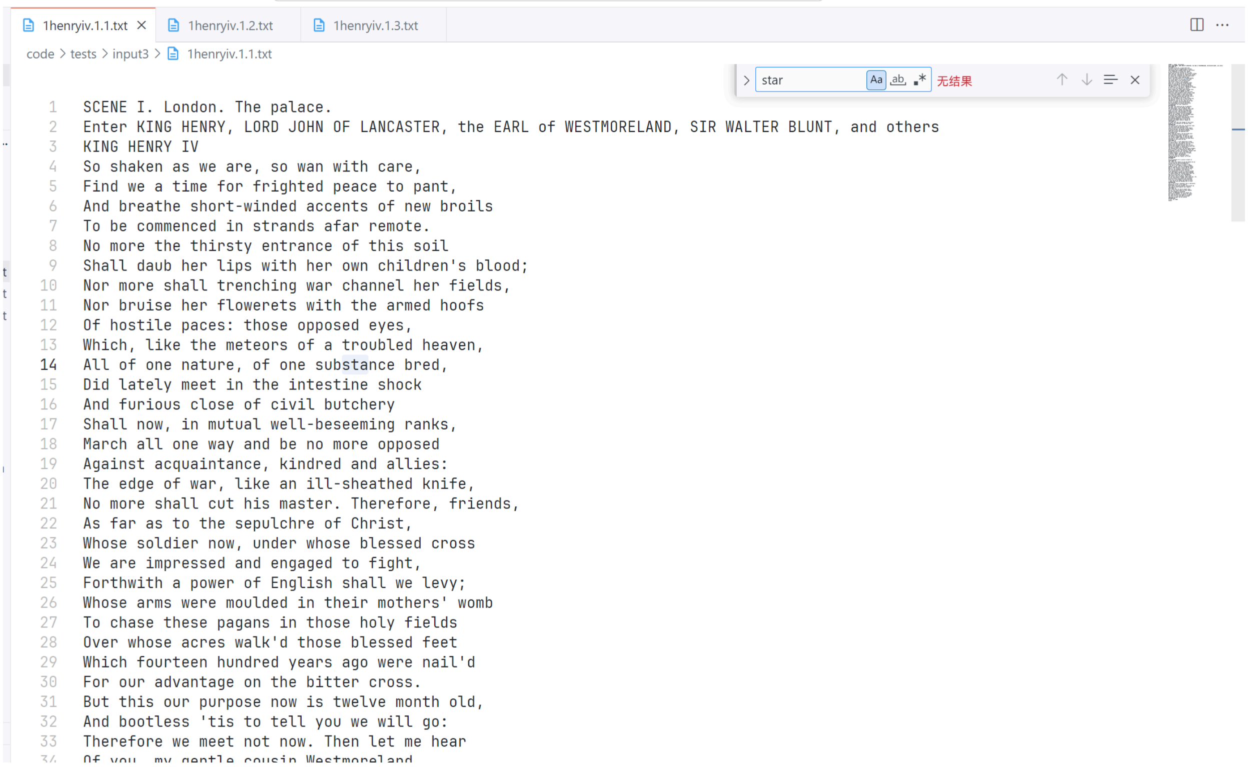
```
For our advantage on the bitter cross.
But this our purpose now is twelve month old,
And bootless 'tis to tell you we will go:
Therefore we meet not now. Then let me hear
Of you, my gentle cousin Westmoreland,
What yesternight our council did decree
In forwarding this dear expedience.
WESTMORELAND
My liege, this haste was hot in question,
And many limits of the charge set down
But yesternight: when all athwart there came
A post from Wales loaden with heavy news;
Whose worst was, that the noble Mortimer,
Leading the men of Herefordshire to fight
Against the irregular and wild Glendower,
Was by the rude hands of that Welshman taken,
A thousand of his people butchered;
Upon whose dead corpse there was such misuse,
Such beastly shameless transformation,
By those Welshwomen done as may not be
Without much shame retold or spoken of.
KING HENRY IV
It seems then that the tidings of this broil
Brake off our business for the Holy Land.
WESTMORELAND
This match'd with other did, my gracious lord;
For more uneven and unwelcome news
Came from the north and thus it did import:
On Holy-rood day, the gallant Hotspur there,
Young Harry Percy and brave Archibald,
That ever-valiant and approved Scot,
At Holmedon met,
Where they did spend a sad and bloody hour,
As by discharge of their artillery,
And shape of likelihood, the news was told;
For he that brought them, in the very heat
```

```
upon the exploit themselves; which they shall have
no sooner achieved, but we'll set upon them.
PRINCE HENRY
Yea, but 'tis like that they will know us by our
horses, by our habits and by every other
appointment, to be ourselves.
POINS
Tut! our horses they shall not see: I'll tie them
in the wood; our vizards we will change after we
leave them: and, sirrah, I have cases of buckram
for the nonce, to immask our noted outward garments.
PRINCE HENRY
Yea, but I doubt they will be too hard for us.
POINS
Well, for two of them, I know them to be as
true-bred cowards as ever turned back; and for the
third, if he fight longer than he sees reason, I'll
forswear arms. The virtue of this jest will be, the
incomprehensible lies that this same fat rogue will
tell us when we meet at supper: how thirty, at
least, he fought with; what wards, what blows, what
extremities he endured; and in the reproof of this
lies the jest.
PRINCE HENRY
Well, I'll go with thee: provide us all things
necessary and meet me to-morrow night in Eastcheap;
there I'll sup. Farewell.
POINS
Farewell, my lord.
Exit Poins
PRINCE HENRY
I know you all, and will awhile uphold
The unyoked humour of your idleness:
Yet herein will I imitate the sun,
Who doth permit the base contagious clouds
To smother up his beauty from the world
```

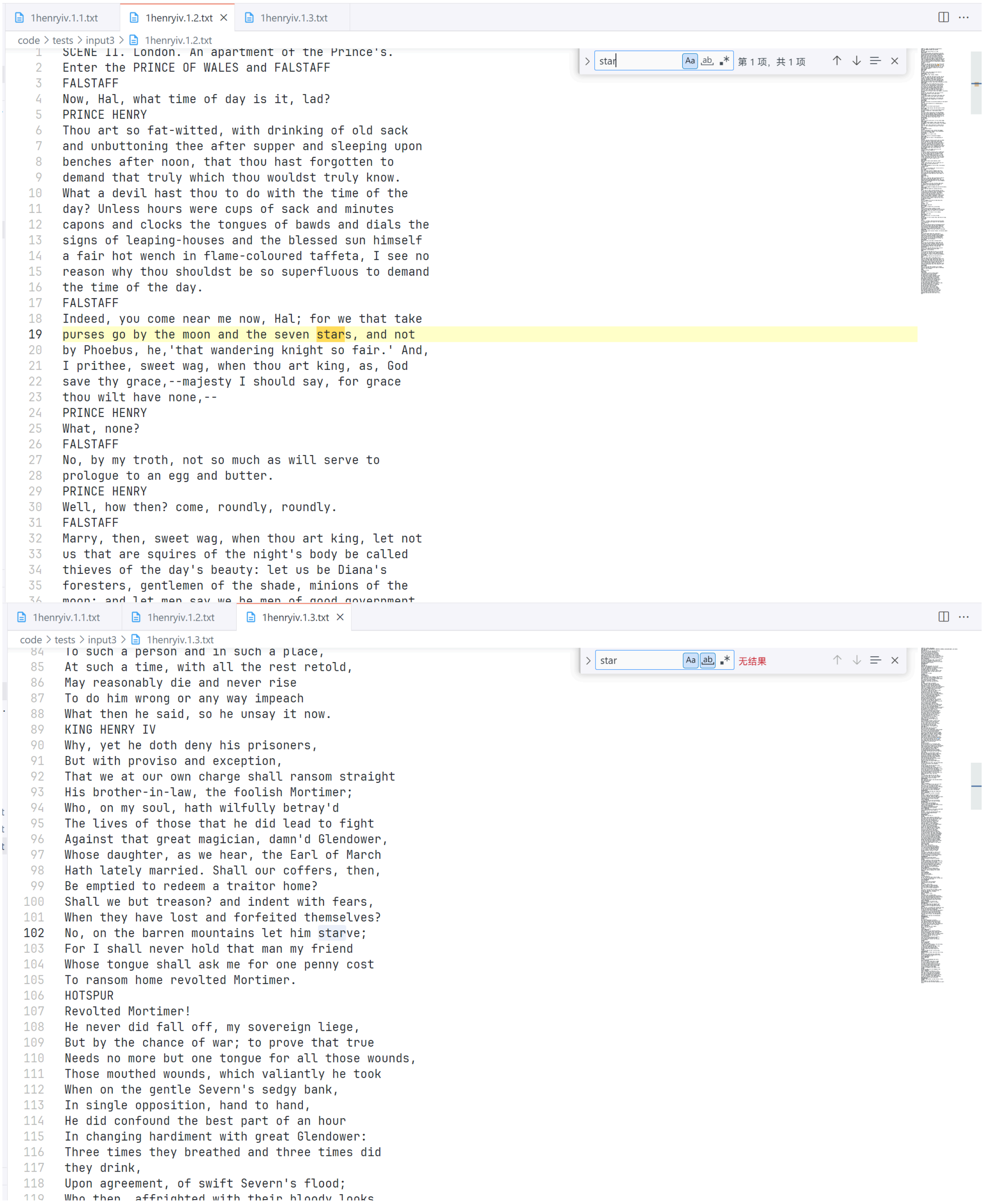


```
SCENE III. London. The palace.
Enter the KING, NORTHUMBERLAND, WORCESTER, HOTSPUR, SIR WALTER BLUNT, wit
KING HENRY IV
My blood hath been too cold and temperate,
Unapt to stir at these indignities,
And you have found me; for accordingly
You tread upon my patience: but be sure
I will from henceforth rather be myself,
Mighty and to be fear'd, than my condition;
Which hath been smooth as oil, soft as young down,
And therefore lost that title of respect
Which the proud soul ne'er pays but to the proud.
EARL OF WORCESTER
Our house, my sovereign liege, little deserves
The scourge of greatness to be used on it;
And that same greatness too which our own hands
Have holp to make so portly.
NORTHUMBERLAND
My lord.--
KING HENRY IV
Worcester, get thee gone; for I do see
Danger and disobedience in thine eye:
O, sir, your presence is too bold and peremptory,
And majesty might never yet endure
The moody frontier of a servant brow.
You have good leave to leave us: when we need
Your use and counsel, we shall send for you.
Exit Worcester
You were about to speak.
To North
NORTHUMBERLAND
Yea, my good lord.
Those prisoners in your highness' name demanded,
Which Harry Percy here at Holmedon took,
Were, as he says, not with such strength denied
As is deliver'd to your majesty:
```

Find 2: star



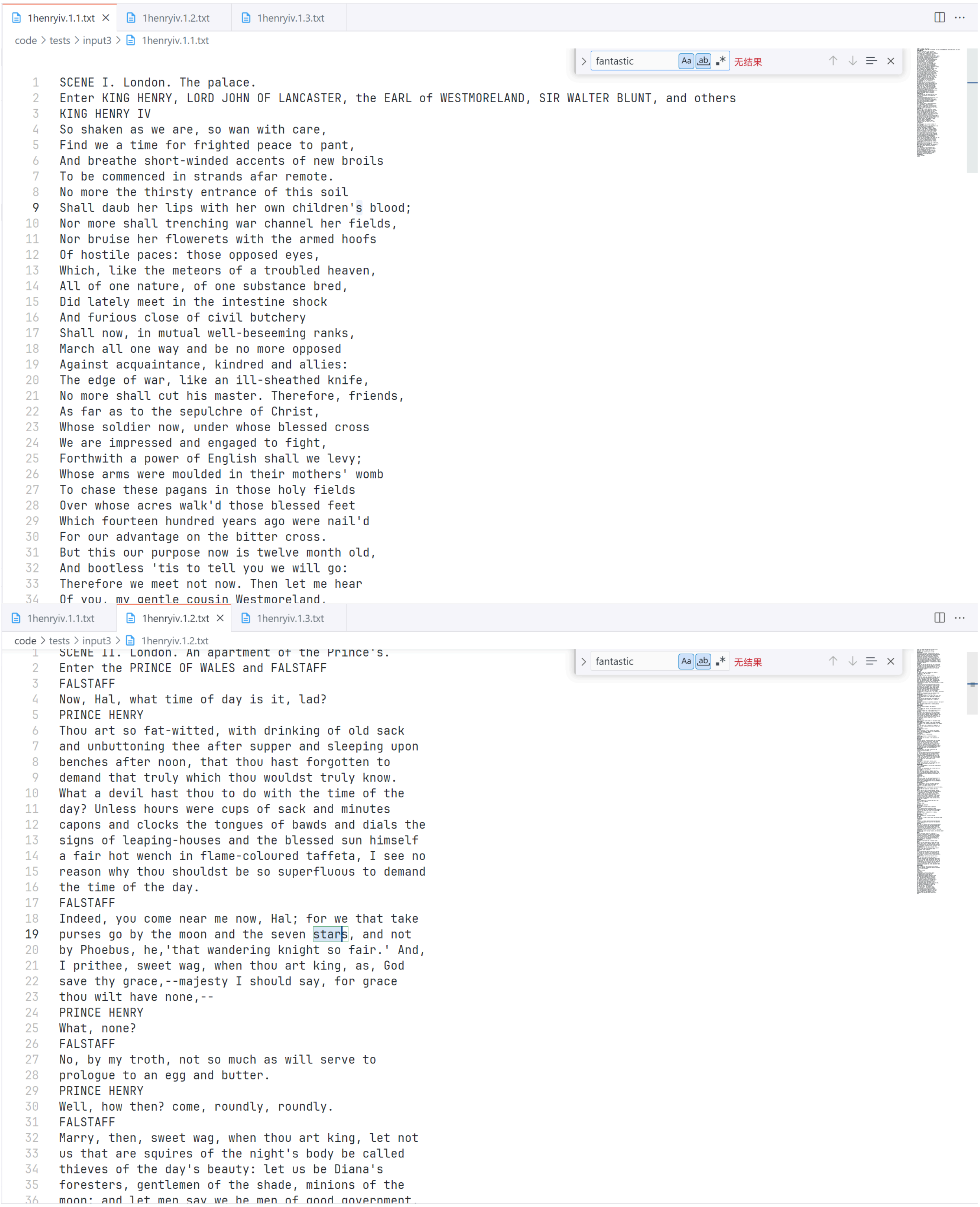
```
SCENE I. London. The palace.
Enter KING HENRY, LORD JOHN OF LANCASTER, the EARL of WESTMORELAND, SIR WALTER BLUNT, and others
KING HENRY IV
So shaken as we are, so wan with care,
Find we a time for frighted peace to pant,
And breathe short-winded accents of new broils
To be commenced in strands afar remote.
No more the thirsty entrance of this soil
Shall daub her lips with her own children's blood;
Nor more shall trenching war channel her fields,
Nor bruise her flowerets with the armed hoofs
Of hostile paces: those opposed eyes,
Which, like the meteors of a troubled heaven,
All of one nature, of one substance bred,
Did lately meet in the intestine shock
And furious close of civil butchery
Shall now, in mutual well-beseeming ranks,
March all one way and be no more opposed
Against acquaintance, kindred and allies:
The edge of war, like an ill-sheathed knife,
No more shall cut his master. Therefore, friends,
As far as to the sepulchre of Christ,
Whose soldier now, under whose blessed cross
We are impressed and engaged to fight,
Forthwith a power of English shall we levy;
Whose arms were moulded in their mothers' womb
To chase these pagans in those holy fields
Over whose acres walk'd those blessed feet
Which fourteen hundred years ago were nail'd
For our advantage on the bitter cross.
But this our purpose now is twelve month old,
And bootless 'tis to tell you we will go:
Therefore we meet not now. Then let me hear
Of you, my gentle cousin Westmoreland
```



```
SCENE II. London. An apartment of the Prince's.
Enter the PRINCE OF WALES and FALSTAFF
FALSTAFF
Now, Hal, what time of day is it, lad?
PRINCE HENRY
Thou art so fat-witted, with drinking of old sack
and unbuttoning thee after supper and sleeping upon
benches after noon, that thou hast forgotten to
demand that truly which thou wouldst truly know.
What a devil hast thou to do with the time of the
day? Unless hours were cups of sack and minutes
capons and clocks the tongues of bawds and dials the
signs of leaping-houses and the blessed sun himself
a fair hot wench in flame-coloured taffeta, I see no
reason why thou shouldst be so superfluous to demand
the time of the day.
FALSTAFF
Indeed, you come near me now, Hal; for we that take
purses go by the moon and the seven stars, and not
by Phoebus, he,'that wandering knight so fair.' And,
I prithee, sweet wag, when thou art king, as, God
save thy grace,--majesty I should say, for grace
thou wilt have none,--
PRINCE HENRY
What, none?
FALSTAFF
No, by my troth, not so much as will serve to
prologue to an egg and butter.
PRINCE HENRY
Well, how then? come, roundly, roundly.
FALSTAFF
Marry, then, sweet wag, when thou art king, let not
us that are squires of the night's body be called
thieves of the day's beauty: let us be Diana's
foresters, gentlemen of the shade, minions of the
moon; and let men say we be men of good government
```

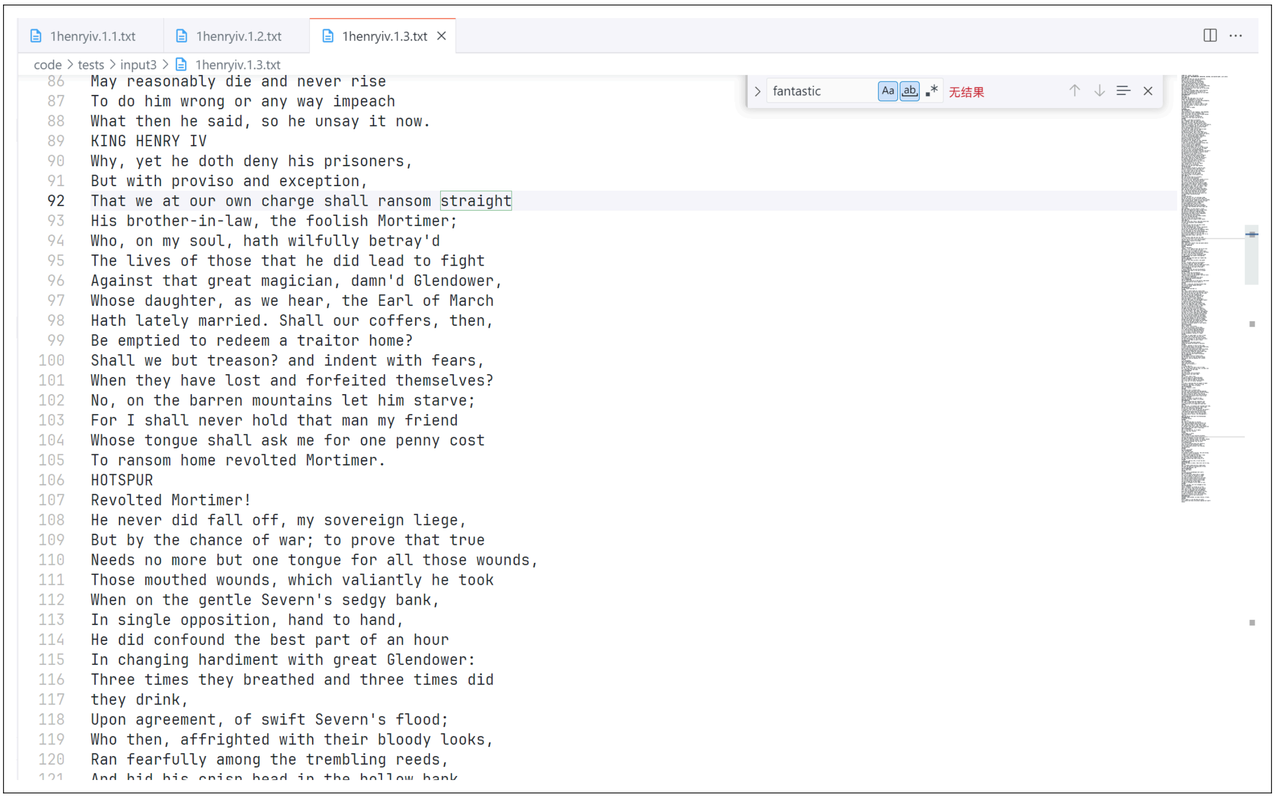
```
To such a person and in such a place,
At such a time, with all the rest retold,
May reasonably die and never rise
To do him wrong or any way impeach
What then he said, so he unsay it now.
KING HENRY IV
Why, yet he doth deny his prisoners,
But with proviso and exception,
That we at our own charge shall ransom straight
His brother-in-law, the foolish Mortimer;
Who, on my soul, hath wilfully betray'd
The lives of those that he did lead to fight
Against that great magician, damn'd Glendower,
Whose daughter, as we hear, the Earl of March
Hath lately married. Shall our coffers, then,
Be emptied to redeem a traitor home?
Shall we but treason? and indent with fears,
When they have lost and forfeited themselves?
No, on the barren mountains let him starve;
For I shall never hold that man my friend
Whose tongue shall ask me for one penny cost
To ransom home revolted Mortimer.
HOTSPUR
Revolted Mortimer!
He never did fall off, my sovereign liege,
But by the chance of war; to prove that true
Needs no more but one tongue for all those wounds,
Those mouthed wounds, which valiantly he took
When on the gentle Severn's sedgy bank,
In single opposition, hand to hand,
He did confound the best part of an hour
In changing hardiment with great Glendower:
Three times they breathed and three times did
they drink,
Upon agreement, of swift Severn's flood;
Who then, affrighted with their bloody looks
```

Find 3: fantastic



```
SCENE I. London. The palace.
Enter KING HENRY, LORD JOHN OF LANCASTER, the EARL of WESTMORELAND, SIR WALTER BLUNT, and others
KING HENRY IV
So shaken as we are, so wan with care,
Find we a time for frighted peace to pant,
And breathe short-winded accents of new broils
To be commenced in strands afar remote.
No more the thirsty entrance of this soil
Shall daub her lips with her own children's blood;
Nor more shall trenching war channel her fields,
Nor bruise her flowerets with the armed hoofs
Of hostile paces: those opposed eyes,
Which, like the meteors of a troubled heaven,
All of one nature, of one substance bred,
Did lately meet in the intestine shock
And furious close of civil butchery
Shall now, in mutual well-beseeming ranks,
March all one way and be no more opposed
Against acquaintance, kindred and allies:
The edge of war, like an ill-sheathed knife,
No more shall cut his master. Therefore, friends,
As far as to the sepulchre of Christ,
Whose soldier now, under whose blessed cross
We are impressed and engaged to fight,
Forthwith a power of English shall we levy;
Whose arms were moulded in their mothers' womb
To chase these pagans in those holy fields
Over whose acres walk'd those blessed feet
Which fourteen hundred years ago were nail'd
For our advantage on the bitter cross.
But this our purpose now is twelve month old,
And bootless 'tis to tell you we will go:
Therefore we meet not now. Then let me hear
Of you, my gentle cousin Westmoreland,
```

```
SCENE II. London. An apartment of the Prince's.
Enter the PRINCE OF WALES and FALSTAFF
FALSTAFF
Now, Hal, what time of day is it, lad?
PRINCE HENRY
Thou art so fat-witted, with drinking of old sack
and unbuttoning thee after supper and sleeping upon
benches after noon, that thou hast forgotten to
demand that truly which thou wouldst truly know.
What a devil hast thou to do with the time of the
day? Unless hours were cups of sack and minutes
capons and clocks the tongues of bawds and dials the
signs of leaping-houses and the blessed sun himself
a fair hot wench in flame-coloured taffeta, I see no
reason why thou shouldst be so superfluous to demand
the time of the day.
FALSTAFF
Indeed, you come near me now, Hal; for we that take
purses go by the moon and the seven stars, and not
by Phoebus, he,'that wandering knight so fair.' And,
I prithee, sweet wag, when thou art king, as, God
save thy grace,--majesty I should say, for grace
thou wilt have none,--
PRINCE HENRY
What, none?
FALSTAFF
No, by my troth, not so much as will serve to
prologue to an egg and butter.
PRINCE HENRY
Well, how then? come, roundly, roundly.
FALSTAFF
Marry, then, sweet wag, when thou art king, let not
us that are squires of the night's body be called
thieves of the day's beauty: let us be Diana's
foresters, gentlemen of the shade, minions of the
moon; and let men say we be men of good government,
```



```
May reasonably die and never rise
To do him wrong or any way impeach
What then he said, so he unsay it now.
KING HENRY IV
Why, yet he doth deny his prisoners,
But with proviso and exception,
That we at our own charge shall ransom straight
His brother-in-law, the foolish Mortimer;
Who, on my soul, hath wilfully betray'd
The lives of those that he did lead to fight
Against that great magician, damn'd Glendower,
Whose daughter, as we hear, the Earl of March
Hath lately married. Shall our coffers, then,
Be emptied to redeem a traitor home?
Shall we but treason? and indent with fears,
When they have lost and forfeited themselves?
No, on the barren mountains let him starve;
For I shall never hold that man my friend
Whose tongue shall ask me for one penny cost
To ransom home revolted Mortimer.
HOTSPUR
Revolted Mortimer!
He never did fall off, my sovereign liege,
But by the chance of war; to prove that true
Needs no more but one tongue for all those wounds,
Those mouthed wounds, which valiantly he took
When on the gentle Severn's sedgy bank,
In single opposition, hand to hand,
He did confound the best part of an hour
In changing hardiment with great Glendower:
Three times they breathed and three times did
they drink,
Upon agreement, of swift Severn's flood;
Who then, affrighted with their bloody looks,
Ran fearfully among the trembling reeds,
And hid his crisp head in the hollow bank
```

In a nutshell, our inverted index program successfully passes the first test.

3.1.2 Stopwords Test

Then, we should confirm whether our program can eliminate the stopwords we have selected in advance. So we can make a comparison with two test program: one includes the stopwords, while the other doesn't include them.

Case 1: stopwords included

3.1.3 Single File to Multiple Files Test

3.2 Thresholds for Queries

3.3 Speed Test

3.4 (Maybe)Debug Mode

Chapter 4: Analysis and Comments

4.1 Time Complexity

4.2 Space Complexity

Appendix: Source code

5.1 File Structure

5.2 invIndexHeader.h

```
// Use B+ tree to store and access to the inverted index
// Declaration of properties, methods and some constants related to B+ tree
#include <stdbool.h>
#include <stdio.h>
#include <string>
#include <time.h>

#ifndef INVINDEX_H
#define INVINDEX_H    // In case of re-inclusion of this header file

#define ORDER 4                               // The order of B+ Tree
#define LEAFCUT (ORDER / 2 + 1)               // The position of the middle data in the leaf node of B+ Tree
#define NONLEAFCUT (ORDER / 2)                // The position of the middle data in the non-leaf node of B+ Tree
#define SIZE 1000000                         // The maximum size of the queue used in printing the B+ Tree
#define MAXWORDLEN 31                        // The maximum length of a single word(the longest word is about 27 or 28 in Shakespeare's works)
#define MAXDOCSUM 500000                      // The maximum number of documents(files)
#define MAXREADSTRLEN 101                     // The maximum lenght of string for one read
#define STOPWORDSUM 300                       // The maximum number of stop words
#define STOPWORDPATH "../sources/stop_words.txt"  // The path of the file storing stop words
#define DEFAULTFILEPOS "tests"               // The default position of the file(for test mode)

// alias
```

```
typedef char * string;
typedef struct data * Data;
typedef struct nodebp * NodeBP;
typedef struct nodebp * BplusTree;
typedef struct poslist * PosList;
typedef struct posdata * PosData;
typedef struct queuebp * QueueBP;
typedef struct hashtb * HashTb;
typedef struct hashsw * HashSW;

enum Kind {Legitimate, Empty};         // The state of the cells
in hash table
extern string docNames[MAXDOCSUM];     // Array containing names
of documents(global variable)

// Nodes in B+ Trees
struct nodebp {
    int size;                          // The size of the data in
the node
    int childrenSize;                  // The size of the children
nodes of the node

    Data data[ORDER + 1];              // The data of the node
    NodeBP children[ORDER + 1];        // The children nodes
    NodeBP parent;                     // The parent node(for split
operation)
};

// Data of the node in B+ Trees
struct data {
    string term;                       // The term
    PosList poslist;                   // All position where the
term appears
};

// List of the position of terms(similar to the queue, but not
same)
struct poslist {
    int size;                          // The size of list
```

```
     PosData front;                         // The front node of
list(dummy node)
    PosData rear;                       // The rear node of list
};

// The specific position info
struct posdata {
    int pos;                           // Position, i.e. the index
of the document
    int time;                          // the frequency in a single
document
    PosData next;                       // the next pointer
};

// The queue of nodes in B+ tree(array implementation)
struct queuebp {
    int size;                           // The current size of queue
    int front;                           // The index of the front
node
    int rear;                             // The index of the rear
node
    NodeBP data[SIZE];                  // Data
};

// The hash table for stop words
struct hashtb {
    int size;                           // The maximum size of the
hash table
    HashSW data[STOPWORDSUM];           // Data
};

// The cells in hash table
struct hashsw {
    string stopword;                    // Stop word
    enum Kind info;                    // State. either legitimate
or empty
};

// All methods are listed here. The explanation of parameters are
```

```
in the file "invIndexFunc.cpp"
// Methods for building inverted index
// The highest-level function, which users can call it directly
BplusTree InvertedIndex(bool isTest = false, bool
containStopWords = false);
// Ask for the position of the directory or file
void askforFilePos(char * dir, char * fname, bool isTest);
// Make a traversal of all files(or a single file) and build the
inverted index from them(or it)
BplusTree fileTraversaler(BplusTree T, char * dir, char * fname,
bool isTest, bool containStopWords);
// Update the Inverted Index while reading a new document
BplusTree UpdateInvertedIndex(BplusTree T, int docCnt, FILE * fp,
bool containStopWords);

// Methods about B+ tree
// Create a B+ tree
BplusTree CreateBP();
// Find a term in B+ tree
NodeBP FindBP(string term, int docCnt, BplusTree T, bool * flag,
bool isSearch = false);
// Check if the term is in the B+ tree
void isSameTerm(string term, int docCnt, NodeBP nodebp, bool *
flag, bool isSearch = false);
// Insert a term into the B+ tree
BplusTree InsertBP(string term, int docCnt, NodeBP nodebp,
BplusTree Tree);
// Split the node when the node is full
BplusTree SplitBP(NodeBP nodebp, BplusTree Tree);
// Print the B+ tree(level-order traversal)
void PrintBPTree(BplusTree T);

// Methods about the queue
// Create the queue
QueueBP CreateQueueBP();
// Put the node of B+ tree into the queue
void EnqueueBP(NodeBP nodebp, QueueBP Q);
// Get the front node and delete it from the queue
NodeBP DequeueBP(QueueBP Q);
```

```
// Methods about poslist
// Create the poslist
PosList CreatePL();
// Add new position
void EnqueuePL(int pos, PosList L);
// Retrieve all position in the list
int ** RetrievePL(PosList L);

// Methods about hash table
// Build a hash table
HashTb GenerateHashTb();
// Initialization of the hash table
HashTb InitHashTb();
// Find the stopwords or other words in the hash table
int FindHashSW(string stopword, HashTb H, bool justSearch);
// Insert a new stopword in hash table
void InsertHashSW(string stopword, HashTb H);
// Hashing function
int HashFunc(string stopword, int size);
// Print hash table
void PrintHashTb(HashTb H);

// Comparison functions used in qsort()
int cmpData(const void * a, const void * b);     // Compare data
of the node in B+ tree
int cmpNodeBP(const void * a, const void * b);   // Compare the
node by their data

// wstring ⟷ char *, for word stemming
std::wstring chararrToWstring(char * st);
char * wstringToChararr(std::wstring wst);

// Word Stmming wrapper
void WordStem(string term);

// Print the ticks and duration, for -tr or --time function
void PrintTime(clock_t start, clock_t end);
```

```
#endif
```

5.3 invIndexFunc.cpp

```
// Implementation of methods related to B+ tree in invIndex.h
#include "invIndexHeader.h"
#include "wordStem/english_stem.h"
#include <algorithm>
#include <codecvt>
#include <filesystem>
#include <locale>
#include <string>
// To avoid unexpected import problems
extern "C" {
    #include <ctype.h>
    #include <stdbool.h>
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
}
// file system namespace
namespace fs = std::filesystem;

string docNames[MAXDOCSUM];        // Array containing names of
documents(global variable)
HashTb H;                          // Hash table storing the stop
words

// The highest-level function, which users can call it directly
// isTest: -t or --test mode, just use one particular file
// containStopWords: -s or --stopwords mode, contain stop words
when building inverted index
BplusTree InvertedIndex(bool isTest, bool containStopWords) {
    char dir[MAXREADSTRLEN];                 // Directory name
    char fname[MAXREADSTRLEN];               // File name
    BplusTree InvIndex = CreateBP();       // Inverted index, stored
in B+ tree

    askforFilePos(dir, fname, isTest);    // Ask for the position
of file or directory
     // Make a traversal in the directory(or a single file) and
build the inverted index from it
      InvIndex = fileTraversaler(InvIndex, dir, fname, isTest,
containStopWords);
    if (InvIndex->size) {                // If the inverted index
contains the data, it indicates the success of building
        printf("Build successfully!\n");
    } else {                             // Otherwise, it fails
        printf("Fail to build an inverted index!\n");
    }

     return InvIndex;                        // Return the final
inverted index
}

// Ask for the position of the directory or file
// dir: directory name
// fname: file name
// isTest: -t or --test mode, just use one particular file
void askforFilePos(char * dir, char * fname, bool isTest) {
     if (isTest) {     // If we choose the test mode in the test
file,
         printf("Now testing the correctness of inverted Index:
\n");
        printf("Please input the name of the input sample file:
\nName: ");
        scanf("%s", fname);   // then we should input the filename
     } else {          // Otherwise(in the main program or other
default situations),
        printf("Now building an inverted Index:\n");
          printf("Please input the directory of the documents:
\nPath: ");
        scanf("%s", dir);     // we should input the name of the
directory
    }
}

// Make a traversal of all files(or a single file) and build the
```

```
inverted index from them(or it)
// T: B+ tree containing the inverted index
// dir: directory name
// fname: file name
// isTest: -t or --test mode, just use one particular file
// containStopWords: -s or --stopwords mode, contain stop words
when building inverted index
BplusTree fileTraversaler(BplusTree T, char * dir, char * fname,
bool isTest, bool containStopWords) {
    int docCnt = 0;       // Count the number of documents and act
as the index of the documents at the same time
    char * wholePath;     // The whole path name
    FILE * fp = NULL;     // File pointer

    H = GenerateHashTb();                  // Build a hash table
for stop words
     if (containStopWords) {                   // If in stopwords
mode, print the hash table
        PrintHashTb(H);
    }

    wholePath = new char[MAXREADSTRLEN];
     if (!isTest) {         // If we choose the test mode in the
test file,
        fs::path dirPath(dir);
          if (fs::exists(dirPath) && fs::is_directory(dirPath))
{   // Make a traversal in the directory
          for (const auto& entry : fs::directory_iterator(dirPath))
{
                if (fs::is_regular_file(entry)) {     // entry: a
single file
                                        std::string  filename  =
entry.path().filename().string();  // Get the file name
                       docNames[docCnt] = new char[filename.length()
+ 1];
                 strcpy(docNames[docCnt], filename.c_str());   //
Store the filename
                    strcpy(wholePath, (dirPath.string() + "/" +
filename).c_str());  // Get the whole path name
```

```
                    // Open the file
                    fp = fopen(wholePath, "r");
                    if (!fp) {  // Error handler
                        printf("Couldn't open the file!\n");
                        exit(1);
                    }
                    // Update the inverted index
                       T = UpdateInvertedIndex(T, docCnt++, fp,
containStopWords);
                }
            }
            if (fp)  // Don't forget close the file pointer
                fclose(fp);
        } else {  // Input wrong directory
            perror("Could not open directory");

        }
    } else {
        strcpy(dir, DEFAULTFILEPOS);       // The file is in the
default position
        std::string sdir(dir);
        std::string sfname(fname);
        strcpy(wholePath, (sdir + "/" + sfname).c_str());  // Get
the whole path name

              docNames[docCnt] = (string)malloc(sizeof(char) *
(strlen(fname) + 1));
         strcpy(docNames[docCnt], fname);                      //
Store the filename
        // Open the file
        fp = fopen(wholePath, "r");
        if (!fp) {  // Error handler
            printf("Couldn't open the file!\n");
            exit(1);
        }
        // Update the inverted index
                  T = UpdateInvertedIndex(T, docCnt++, fp,
containStopWords);
        fclose(fp);    // Don't forget close the file pointer
    }

    return T;
}

// Update the Inverted Index while reading a new document
// T: B+ tree containing the inverted index
// docCnt: the index of the document
// fp: file pointer
// containStopWords: -s or --stopwords mode, contain stop words
when building inverted index
BplusTree UpdateInvertedIndex(BplusTree T, int docCnt, FILE * fp,
bool containStopWords) {
    int i;
    int pre, cur;                          // Mark the start and
the end of one word
    char tmp[MAXREADSTRLEN];               // Memory space storing
the reading data temporarily
    string term;                           // Term(or word)
     bool isDuplicated;                        // A flag, record
whether the term exists in the B+ tree
    NodeBP nodebp;                         // Node in B+ tree


     while (fgets(tmp, MAXREADSTRLEN - 1, fp) != NULL) {   //
Continue reading the file, until arrive at the end of file
        pre = cur = 0;                        // Initialization
         for (i = 0; i < strlen(tmp); i++) {    // Retrieve all
characters in the tmp string
            if (!isalpha(tmp[i])) {           // Maybe it's time
to record a word
                cur = i;
                  if (cur > pre) {                // Legitimate
situation
                      term = (char *)malloc(sizeof(char) * (cur -
pre + 1));
                     strncpy(term, tmp + pre, cur - pre);
                      term[cur - pre] = '\0';    // Don't forget
this step
```

```
                    // If we consider the stop words(default) and
assure the term is a stop word,
                    if (!containStopWords && FindHashSW(term, H,
true) >= 0) {
                        pre = cur + 1;
                        continue;   // then we should ignore it
                    }

                    // Word stemming
                    WordStem(term);

                    isDuplicated = false;
                nodebp = FindBP(term, docCnt, T, &isDuplicated);  //
Find the appropriate position for the term
                    // If isDuplicated is true, then the time of
the term will +1 in function isSameTerm()
                    if (!isDuplicated) {  // If it's a new term,
insert it!
                        T = InsertBP(term, docCnt, nodebp, T);
                    }
                }

                pre = cur + 1;     // Move the start position for
possible new word
            }
        }

        // Handle the last possible word in the tmp string
        if (!cur || pre > cur && pre != i) {
            cur = i;
              term = (char *)malloc(sizeof(char) * (cur - pre +
1));
            strncpy(term, tmp + pre, cur - pre);
            term[cur - pre] = '\0';

            // Word stemming
            WordStem(term);
```

```
             isDuplicated = false;
             nodebp = FindBP(term, docCnt, T, &isDuplicated); //
Find the appropriate position for the term
             // If isDuplicated is true, then the time of the term
will +1 in function isSameTerm()
             if (!isDuplicated) {  // If it's a new term, insert
it!
                 T = InsertBP(term, docCnt, nodebp, T);
             }
         }
    }

    return T;
}

// Create a B+ tree
BplusTree CreateBP() {
    BplusTree T = (BplusTree)malloc(sizeof(struct nodebp));  //
Allocate the memory space for new B+ tree
    if (T == NULL) {    // Allocation failure
        printf("Failed to create a B+ Tree!\n");
        return T;
    }

    int i;

    // Memory allocation and initialization of data and children
    for (i = 0; i <= ORDER; i++) {
        T->data[i] = (Data)malloc(sizeof(struct data));
              T->data[i]->term  =  (string)malloc(sizeof(char)  *
MAXWORDLEN);
        T->data[i]->poslist = CreatePL();
        T->children[i] = (NodeBP)malloc(sizeof(struct nodebp));
    }
    // Initialization of other fields
    T->size = 0;
    T->childrenSize = 0;
    T->parent = NULL;
    return T;
}

// Find a term in B+ tree
// term: term
// docCnt: the index of the document
// T: inverted index
// flag: true if the term is found, false otherwise
// isSearch: mark the find mode(-f or --find)
NodeBP FindBP(string term, int docCnt, BplusTree T, bool * flag,
bool isSearch) {
    int i;

    if (!T) {    // If the tree is empty, return the tree(actually,
it's impossible in our program)
        return T;
    } else if (!T->childrenSize) {  // If we arrive at the leaf
node, search its data
        isSameTerm(term, docCnt, T, flag, isSearch);
        return T;
    }

    int pos = -1;  // The index of the appopriate non-leaf node
    for (i = 0; i < T->size; i++) {
         if (strcmp(term, T->data[i]->term) < 0) { // Find the
first node which have term with higher lexicographic number
            pos = i;
            break;
        }
    }
     if (pos == -1) {  // If no position found in above loop,
choose the last node
        pos = i;
    }

        return  FindBP(term,  docCnt,  T->children[pos],  flag,
isSearch);  // Continue finding in the children node
}
```

```
// Check if the term exists in the B+ tree
// term: term
// docCnt: the index of the document
// nodebp: the appropriate node where the term may exists or will
exists after insertion
// flag: true if the term is found, false otherwise
// isSearch: mark the find mode(-f or --find)
void isSameTerm(string term, int docCnt, NodeBP nodebp, bool *
flag, bool isSearch) {
    int i;

    if (nodebp->size) {      // If it's not an empty node, start
searching
        for (i = 0; i < nodebp->size; i++) {
            if (!strcmp(term, nodebp->data[i]->term)) {     // If
the term exists in the inverted index
                if (!isSearch) {     // If it's not in the find
mode
                 EnqueuePL(docCnt, nodebp->data[i]->poslist);  //
Update the poslist of the term
                } else {    // Otherwise, print all info of the
term
                 PosList poslist = nodebp->data[i]->poslist;     //
Position list
                  int size = poslist->size;                      //
The number of all documents where the term appears
                  int cnt = 0;                                   //
Record the total frequency of the term

                 printf("Successfully find the word!\n");        //
Some banners
                     printf("The word was found in files below:
\n");

                    int j;
                    int ** posArr = (int **)malloc(sizeof(int *)
* size);  // Allocation of a 2D array
                    for (j = 0; j < size; j++) {
                         posArr[i] = (int *)malloc(sizeof(int) *
2);
                    }

                  posArr = RetrievePL(poslist);                //
Put the poslist in a 2D array

                     for (j = 0; j < size; j++) {  // Print the
name of documents and their frequency respectively
                        if (posArr[j][1] <= 1)  // Singular
                        printf("%s: %d time\n", docNames[posArr[j]
[0]], posArr[j][1]);
                        else   // Plural
                       printf("%s: %d times\n", docNames[posArr[j]
[0]], posArr[j][1]);
                        cnt += posArr[j][1];
                    }
                  printf("Frequency: %d\n", cnt);              //
The total frequency
                     printf("-----------------------------------
\n");
                }

                *flag = true;      // mark the flag, indicating
we find the term
                break;
            }
        }
    }
}

// Insert a term into the B+ tree
// term: term
// docCnt: the index of the document
// nodebp: the appropriate node where the term will be inserted
// Tree: B+ tree containing the inverted index
BplusTree  InsertBP(string  term,  int  docCnt,  NodeBP  nodebp,
BplusTree Tree) {
    int i;
    strcpy(nodebp->data[nodebp->size]->term, term);
   EnqueuePL(docCnt, nodebp->data[nodebp->size++]->poslist);   //
Add the data info
     qsort(nodebp->data, nodebp->size, sizeof(nodebp->data[0]),
cmpData);   // Sort the data in time

    Tree = SplitBP(nodebp, Tree);    // Split the node
    return Tree;
}

// Split the node when the node is full
// nodebp: the appropriate node where the term will be inserted
// Tree: B+ tree containing the inverted index
BplusTree SplitBP(NodeBP nodebp, BplusTree Tree) {
    if (!nodebp->childrenSize && nodebp->size <= ORDER     // If
the node is not full
        || nodebp->childrenSize && nodebp->size < ORDER) {    //
(consider both leaf node and non-leaf node),
        return Tree;                                         // do
nothing!
    }

     // lnodebp, rnodebp: the left and right part of the split
node
    // tmpNodebp: store the node temporarily
    // parent: the parent node of nodebp
    NodeBP lnodebp, rnodebp, tmpNodebp, parent;
    int cut;    // The position of the middle data
    int i, j;

    parent = nodebp->parent;

    if (!parent) {  // If the node has no parent(i.e. this node
is the root),
          tmpNodebp = CreateBP();   // create a new node as the
parent(and also the root of the tree)
        parent = (NodeBP)malloc(sizeof(struct nodebp));
        Tree = parent = tmpNodebp;
    }
    lnodebp = CreateBP();
    rnodebp = CreateBP();
    lnodebp->parent = rnodebp->parent = parent;   // Connect the
two parts with the parent node

     if (!nodebp->childrenSize) {    // If the node is the leaf
node
        cut = LEAFCUT;

        for (i = 0; i < cut; i++) {     // Assign the data in the
left part of original node to lnodebp
            lnodebp->data[i] = nodebp->data[i];
        }
        lnodebp->size = cut;

         for (j = cut; j < nodebp->size; j++) {    // Assign the
data in the right part of original node to rnodebp
            rnodebp->data[j - cut] = nodebp->data[j];
        }
        rnodebp->size = nodebp->size - cut;

    } else {                       // If the node is the non-leaf
node
        cut = NONLEAFCUT;

         for (i = 0; i <= cut; i++) {     // Assign the data and
children in the left part of original node to lnodebp
            if (i != cut)
                lnodebp->data[i] = nodebp->data[i];
            lnodebp->children[i] = nodebp->children[i];
            lnodebp->children[i]->parent = lnodebp;
        }
        lnodebp->size = cut;
        lnodebp->childrenSize = cut + 1;

          // Assign the data and children in the right part of
original node to rnodebp
        for (j = cut + 1; j < nodebp->size; j++) {
            rnodebp->data[j - cut - 1] = nodebp->data[j];
        }
        for (j = cut + 1; j < nodebp->childrenSize; j++) {
            rnodebp->children[j - cut - 1] = nodebp->children[j];
            rnodebp->children[j - cut - 1]->parent = rnodebp;
        }
        rnodebp->size = nodebp->size - cut - 1;
        rnodebp->childrenSize = nodebp->childrenSize - cut - 1;
    }

    // Assign the middle data in the original node to its parent
    parent->data[parent->size++] = nodebp->data[cut];
   if (parent->childrenSize) {    // If the parent has children(not
be created newly)
        for (i = 0; i < parent->childrenSize; i++) {
             if (parent->children[i] == nodebp) {     // Replace
the original node with lnodebp
                parent->children[i] = lnodebp;
                break;
            }
        }
    } else {     // newly created parent
        parent->children[parent->childrenSize++] = lnodebp;   //
Insert the lnodebp
    }
    parent->children[parent->childrenSize++] = rnodebp;       //
Insert the rnodebp

    // Sort the data and children of the parent
     qsort(parent->data, parent->size, sizeof(parent->data[0]),
cmpData);
    qsort(parent->children, parent->childrenSize, sizeof(parent-
>children[0]), cmpNodeBP);

    free(nodebp); // Free the memory of the original node

    Tree = SplitBP(parent, Tree);  // Continue spliting the upper
node
```

```
    return Tree;
}

// Print the B+ tree(level-order traversal)
// T: B+ tree containing the inverted index
void PrintBPTree(BplusTree T) {
    int i;
    NodeBP nodebp;          // The node obtained from the queue
    QueueBP q;              // The queue containing the nodes from
B+ tree

    printf("B+ Tree of Inverted Index:\n");

    q = CreateQueueBP();  // Create an empty queue
    EnqueueBP(T, q);       // Put the root of the tree into the
queue first
    EnqueueBP(NULL, q);   // Put the NULL pointer, for creation
of newline

    while (q->size) {     // If the queue isn't empty, repeat the
following steps
        nodebp = DequeueBP(q);  // Get the front node
        if (!nodebp) {          // If it's an NULL pointer, it's
time to add a newline
            printf("\n");
              if (q->size) {       // If the queue isn't empty,
continue add a new NULL pointer
                EnqueueBP(NULL, q);
            }
        } else {
            printf("[");        // Print the node's data(just the
term)
            for (i = 0; i < nodebp->size; i++) {
                if (!i) {
                    printf("%s", nodebp->data[i]->term);
                } else {
                    printf(", %s", nodebp->data[i]->term);
                }
            }
            printf("]");
        }

        if (nodebp) {          // If nodebp isn't a NULL pointer,
then put its children into the queue
            for (i = 0; i < nodebp->childrenSize; i++) {
                EnqueueBP(nodebp->children[i], q);
            }
        }
    }
}

// Create the queue
QueueBP CreateQueueBP() {
    QueueBP Q = (QueueBP)malloc(sizeof(struct queuebp));
    Q->size = 0;
    Q->front = Q->rear = 0;

    return Q;
}

// Put the node of B+ tree into the queue
// nodebp: the newly added node
// Q: the queue
void EnqueueBP(NodeBP nodebp, QueueBP Q) {
     if (Q->size >= SIZE) {   // If the queue is full, enqueue
operation fails
        printf("Full B+-tree-item queue!\n");
        exit(1);
    }
    Q->data[Q->rear++] = nodebp;   // Add new node
    Q->size++;
}

// Get the front node and delete it from the queue
// Q: the queue
NodeBP DequeueBP(QueueBP Q) {
    if (!Q->size) {          // If the queue is empty, dequeue
operation fails
```

```
        printf("Empty B+-tree-item queue!\n");
        exit(1);
    }
    NodeBP returnNodeBP = Q->data[Q->front++];  // Get the front
node
    Q->size--;      // Delete the node from queue
    return returnNodeBP;
}

// Create the poslist
PosList CreatePL() {
    PosList L;

    L = (PosList)malloc(sizeof(struct poslist));
    L->size = 0;
    L->front = (PosData)malloc(sizeof(struct posdata));
    L->rear = (PosData)malloc(sizeof(struct posdata));
    L->front = L->rear;
    L->rear->pos = -1;   // Distinguish from other nodes

    return L;
}

// Add new position
// pos: the position
// L: the position list
void EnqueuePL(int pos, PosList L) {
    if (L->rear->pos != pos) {  // If it's a new position
        PosData tmp = (PosData)malloc(sizeof(struct posdata));
        if (!tmp) {
            printf("Fail to create a new position data!\n");
            exit(1);
        }  // Insert the new one in the position list
        tmp->pos = pos;
        tmp->time = 1;
        tmp->next = L->rear->next;
        L->rear->next = tmp;
        L->rear = tmp;
        L->size++;
    } else {    // Otherwise, just increment the frequency
        L->rear->time++;
    }

}

// Retrieve all position in the list
// L: the position list
int ** RetrievePL(PosList L) {
     if (!L->size) {        // If the list is empty, retrieve operation fails
        printf("Empty position-data queue!\n");
        exit(1);
    }

    int i = 0, j;
    int ** posArr = (int **)malloc(sizeof(int *) * L->size);
    for (j = 0; j < L->size; j++) {  // Memory Allocation for 2D array
        posArr[j] = (int *)malloc(sizeof(int) * 2);
    }

    PosData cur = L->front->next;

    while (cur != NULL) {  // Make a traversal in the position list
        posArr[i][0] = cur->pos;  // Get the specific info of the position
        posArr[i][1] = cur->time;
        cur = cur->next;
        i++;
    }

    return posArr;
}

// Build a hash table
HashTb GenerateHashTb() {
    int i;
    int pre, cur;                  // Mark the start and the end
of one word
    HashTb H;                      // The hash table containing
the stop words
    FILE * fp;                     // File pointer
    char fname[MAXWORDLEN];        // File name
    char tmp[MAXREADSTRLEN];       // Memory space storing the
reading data temporarily
    char * term;                   // Term(or word)

    H = InitHashTb();              // Initialization

    strcpy(fname, STOPWORDPATH);   // Use default
path(stop_words.txt)
    fp = fopen(fname, "r");        // Open the file
    if (!fp) {
        printf("Fail to open the file of stopwords!\n");
        exit(1);
    }

    while (fgets(tmp, MAXREADSTRLEN - 1, fp) != NULL) { //
Continue reading the file, until arrive at the end of file
        pre = cur = 0;                           // Initialization
        for (i = 0; i < strlen(tmp); i++) {      // Retrieve
all characters in the tmp string
            if (!isalpha(tmp[i])) {              // Maybe it's
time to record a word
                cur = i;
                if (cur > pre) {                 // Legitimate
situation
                    term = (char *)malloc(sizeof(char) * (cur -
pre + 1));
                    strncpy(term, tmp + pre, cur - pre);
                    term[cur - pre] = '\0';
                    InsertHashSW(term, H);       // Insert the
new term
                }

                pre = cur + 1;
            }
        }

        // Handle the last possible word in the tmp string
        if (!cur || pre > cur && pre != i) {
            cur = i;
            term = (char *)malloc(sizeof(char) * (cur - pre + 1));
            strncpy(term, tmp + pre, cur - pre);
            term[cur - pre] = '\0';
            InsertHashSW(term, H);                    // Insert the new term
        }
    }
    fclose(fp);
    return H;
}

// Initialization of the hash table
HashTb InitHashTb() {
    HashTb H;                // Hash table
    int i;

    H = (HashTb)malloc(sizeof(struct hashtb));    // Memory allocation for the whole table
    if (H == NULL) {
        printf("Fail to create a hash table for stopwords!\n");
        exit(1);
    }

    H->size = STOPWORDSUM;    // maxixum size

    for (i = 0; i < H->size; i++) {
        H->data[i] = (HashSW)malloc(sizeof(hashsw));
        if (H->data[i] == NULL) {     // Memory allocation for cells
            printf("Fail to create a hash table for stopwords!\n");
            exit(1);
        }
           H->data[i]->stopword = (string)malloc(sizeof(char) *
MAXWORDLEN);
        H->data[i]->info = Empty;
    }

    return H;
}

// Find the stopwords or other words in the hash table
// stopword: stop word
// H: hash table containing the stop words
// justSearch: find the term without subsequent insertion
int FindHashSW(string stopword, HashTb H, bool justSearch) {
    int pos;                               // Appropraite position
    int collisionNum = 0;                   // collision number,
for quadratic probe
    pos = HashFunc(stopword, H->size);    // Use hashing function
first

    // If we just search a term in the hash table and assure it's
not a stop word, then return
    if (justSearch && (H->data[pos]->info == Empty || strcmp(H-
>data[pos]->stopword, stopword))) {
        return -1;
    }

    // Collision occurs!
     while (H->data[pos]->info != Empty && strcmp(H->data[pos]-
>stopword, stopword)) {
        pos += 2 * ++collisionNum - 1;  // Quadratic probe
        if (pos >= H->size)
            pos -= H->size;
    }
    return pos;
}

// Insert a new stopword in hash table
// stopword: stop word
// H: hash table containing the stop words
void InsertHashSW(string stopword, HashTb H) {
    int pos;
    pos = FindHashSW(stopword, H, false);    // Find the correct
position
    if (H->data[pos]->info != Legitimate)    // Insert the stop
word
    {
        H->data[pos]->info = Legitimate;
        strcpy(H->data[pos]->stopword, stopword);
    }
}

// Hashing function
// stopword: stop word
// size: the maximum size of the hash table
int HashFunc(string stopword, int size) {
    unsigned int val = 0;
    while (*stopword != '\0')
        val = (val << 5) + *stopword++;     // Generate the hash
value from every character in the string
    return val % size;
}

// Print hash table
void PrintHashTb(HashTb H) {
    int i;

    printf("Stopwords in hash table:\n");
    for (i = 0; i < H->size; i++) {
        if (H->data[i]->info != Empty) {
            printf("%d: %s\n", i, H->data[i]->stopword);
        }
    }
    printf("\n");
}

// Comparison functions used in qsort()
int cmpData(const void * a, const void * b) {
    const Data dataA = *(const Data*)a;
    const Data dataB = *(const Data*)b;

    return strcmp(dataA->term, dataB->term);
}

int cmpNodeBP(const void * a, const void * b) {
    const NodeBP nodebpA = *(const NodeBP*)a;
    const NodeBP nodebpB = *(const NodeBP*)b;

        return  strcmp(nodebpA->data[0]->term,  nodebpB->data[0]-
>term);
}

// wstring <-> char *, for word stemming
std::wstring chararrToWstring(char * st) {
    std::string tmp(st);
    std::wstring_convert<std::codecvt_utf8<wchar_t>> converter;
    std::wstring wstr = converter.from_bytes(tmp);

    return wstr;
}

char * wstringToChararr(std::wstring wst) {
    std::wstring_convert<std::codecvt_utf8<wchar_t>> converter;
    std::string tmp = converter.to_bytes(wst);
    char * st = new char[tmp.size() + 1];
    strcpy(st, tmp.c_str());

    return st;
}

// Word Stmming wrapper
void WordStem(string term) {
    std::wstring term_wstr;                // the wstring form of
the term
      stemming::english_stem<>  StemEnglish;    //  Word  stemming
function(a little clumsy)
```

```
    term_wstr = chararrToWstring(term);
                transform(term_wstr.begin(),    term_wstr.end(),
term_wstr.begin(), ::tolower);
    StemEnglish(term_wstr);
    term = wstringToChararr(term_wstr);
}

// Print the ticks and duration, for -tr or --time function
void PrintTime(clock_t start, clock_t end) {
    clock_t tick;       // ticks
    double duration;    // duration(unit: seconds)

    tick = end - start;
    duration = ((double)(tick)) / CLOCKS_PER_SEC;
    printf("Ticks: %lu\n", (long)tick);    // Print the info
    printf("Duration: %.2fs\n", duration);
}
```

5.4 invIndexTest.cpp

```
#include "invIndexHeader.h"
#include "wordStem/english_stem.h"
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <time.h>

int main(int argc, char * argv[]) {         // Use command line
parameters
    int i;
    bool isTest = false;                    // Whether open test
mode(-t or --test)
    bool Print = false;                     // Whether open print
mode(-p or --print)
    bool isFound;                           // Whether the term is
found in the inverted index
    bool containStopWords = false;        // Whether open stopword
```

```
mode(-s or --stopwords)
    bool timeRecord = false;                // Whether open time
record mode(-tr or --time)
    int findCnt = 0;                        // The time of finding
a single word for find mode(-f=n or --find=n)
    char * pos;                             // Get the number after
`=` in parameters
    char tmp[MAXWORDLEN];                     // Store the input
string temporarily
    char * word;                               // The word to be
searched
    double duration;                        // Duration of running
a function
    std::wstring word_wstr;                   // wstring form of
word
      stemming::english_stem<> StemEnglish;    // Word stemming
function
    BplusTree InvIndex;                     // Inverted Index
    clock_t start, end, tick;                 // Record the start
and the end of the clock

    for (i = 1; i < argc; i++) {            // Read the parameters
        if (!strcmp(argv[i], "--test") || !strcmp(argv[i], "-t"))
{  // Test mode
            isTest = true;
        } else if (strstr(argv[i], "--find") || strstr(argv[i],
"-f")) {  // Find mode
            if ((pos = strchr(argv[i], '='))) {
                findCnt = atoi(pos + 1);       // Get the number
behind `=`
                if (!findCnt) {
                    printf("Wrong Number!\n");
                    exit(1);
                }
            } else {
                findCnt = 1;    // Use default number
            }
        } else if (!strcmp(argv[i], "--print") || !strcmp(argv[i],
"-p")) {  // Print mode
```

```
            Print = true;
            } else if (!strcmp(argv[i], "--stopwords") || !
strcmp(argv[i], "-s")) {  // Stopword mode
            containStopWords = true;
        } else if (!strcmp(argv[i], "--time") || !strcmp(argv[i],
"-tr")) {  // Time record mode
            timeRecord = true;
        } else {   // Error
            printf("Wrong Parameter!\n");
            exit(1);
        }
    }

    if (!timeRecord) {   // No time record
        InvIndex = InvertedIndex(isTest, containStopWords);
    } else {  // Time record
        start = clock();
        InvIndex = InvertedIndex(isTest, containStopWords);
        end = clock();
        PrintTime(start, end);
    }

    if (Print) {  // Print the B+ tree
        PrintBPTree(InvIndex);
    }

    if (InvIndex->size && findCnt) {  // Search the single word
        word = (char *)malloc(sizeof(char) * MAXWORDLEN);
        printf("\nFinding Words Mode(only supports single word
finding):\n");
        for (i = 0; i < findCnt; i++) {  // For every search
            isFound = false;
            printf("Find %d: ", i + 1);
            scanf("%s", tmp);
            strcpy(word, tmp);

            // Word stemming
            word_wstr = chararrToWstring(word);
            StemEnglish(word_wstr);
            word = wstringToChararr(word_wstr);

            // Find the word
            if (!timeRecord) {  // No time record
                FindBP(word, -1, InvIndex, &isFound, findCnt);
            } else {  // Time record
                start = clock();
                FindBP(word, -1, InvIndex, &isFound, findCnt);
                end = clock();
                PrintTime(start, end);
            }

            // If not found, then give relevant information
            if (!isFound) {
               printf("Sorry, no such word in the inverted index!
\n");
                printf("-----------------------------------\n");
            }
        }
    }

    return 0;
}
```

References

[1] Blake-Madden, OleanderStemmingLibrary, https://github.com/Blake-Madden/OleanderStemmingLibrary

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.