

# 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 .....	4
1.1 Problem Description .....	4
1.2 Purpose of Report .....	4
1.3 Background of Data Structures and Algorithms .....	4
Chapter 2: Data Structure / Algorithm Specification .....	5
2.1 Algorithm Architecture .....	5
2.2 The Main Program .....	6
2.3 Word Count & Stop Words .....	7
2.4 Word Stemming .....	8
2.5 Inverted Index .....	9
2.5.1 Overall Functions .....	10
2.5.2 B+ Tree Operations .....	14
2.5.3 Hashing Operations .....	21
2.5.4 Other Functions .....	24
2.6 Query .....	28
2.6.1 Search Function .....	28
2.6.2 Search Test .....	30
2.6.3 Some helper functions .....	30
Chapter 3: Testing Results .....	31
3.1 Inverted Index Tests .....	31
3.1.1 Word Insertion Test .....	32
3.1.2 Single File to Multiple Files Test .....	38
3.1.3 Stopwords Test .....	41
3.1.4 Speed Test .....	43
3.2 Query Tests .....	44
3.2.1 input0 .....	44
3.2.2 input1 .....	45
3.2.3 input2 .....	45
3.2.4 input3 .....	46
3.2.5 input4 .....	46
3.2.6 input5 .....	47
Chapter 4: Analysis and Comments .....	48

4.1 Space Complexity .....	48
4.2 Time Complexity .....	49
4.3 Further Improvement .....	51
Appendix: Source code .....	51
5.1 File Structure .....	51
5.2 getStopWord.cpp .....	52
5.3 invIndexHeader.h .....	55
5.4 invIndexFunc.cpp .....	61
5.5 invIndexTest.cpp .....	84
5.6 invIndexSearch.cpp .....	87
5.7 search_test.cpp .....	97
5.8 search_main.cpp .....	100
References .....	102
Declaration .....	102

# 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 Background 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 structures used in printing the B+ tree, storing the positions for terms, etc.

#### 4. **TF-IDF Algorithm:**

**tf-idf** (also **TF\*IDF**, **TFIDF**, **TF-IDF**, or **Tf-idf**), short for **term frequency-inverse document frequency**, is a measure of importance of a word to a document in a collection or corpus, adjusted for the fact that some words appear more frequently in general.

It was often used as a weighting factor in searches of information retrieval, text mining, and user modeling.

One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model.

There are many variations of how tf-idf is calculated. In our project, we use the following formula to calculate tf, idf and tf-idf.

$$\text{tf}(\text{word}, \text{doc}) = \frac{\text{num of word in the doc}}{\text{total word count of the doc}}$$

$$\text{idf}(\text{word}, \text{doc}) = \log \frac{\text{total num of documents}}{1 + |\text{num of documents that include the word}|}$$

$$\text{ti-idf}(\text{word}, \text{doc}) = \text{tf} \times \text{idf}$$

## Chapter 2: Data Structure / Algorithm Specification

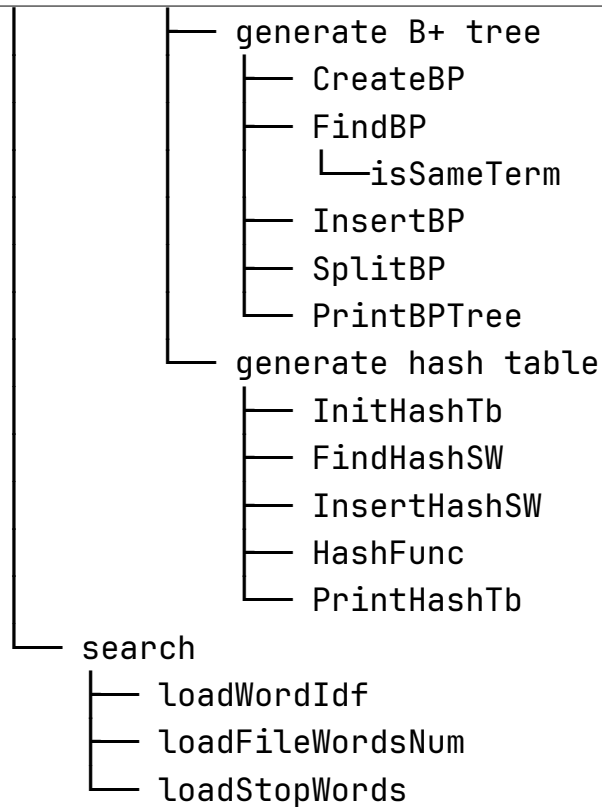
### 2.1 Algorithm Architecture

The overall algorithm architecture in the program is shown below:

```

the main program
├── word count
├── stop words
├── inverted index
│   ├── askforFilePos
│   └── fileTraversaler

```



## 2.2 The Main Program

### Inputs:

- All files of *The Complete Works of Shakespeare*
- User's queries

### Outputs:

- The results of queries

### Procedure: main()

```

1 Begin
2   load necessary resources and establish inverted index
3   set values for pageSize and threshold from user input
4   while search_allow do
5     get query from user input
6     Calling the search function
7     ask user if they want to continue searching or not and set
      search_allow accordingly
8   end
9 End
  
```

The main idea of the code is to serve as the entry point for a search application named “ShakespeareFinder.” It involves loading required resources, setting up an inverted index, and managing user queries.

## 2.3 Word Count & Stop Words

### Note

Because the original open resources of *The Works* are written in HTML format with too many markups, we have converted them into TXT format, which is more readable and convenient for us to handle. And all titles of these articles are extracted to a TXT file at the same time.

The word count and stop words detection are relatively simple, we combined their functions into one program to cope with problems together.

**Inputs:** No obvious input parameters, but the program will read in:

- A file(`code/source/txt_title.txt`) contains all titles of articles in *The Works*
- A directory(`code/source/shakespeare_work`) includes *The Works* in TXT format

**Outputs:** 3 files

- `code/source/stop_words.txt`: Record all selected stopwords
- `code/source/word_count.txt`: Count for each word in all files
- `code/source/word_docs.txt`: Count the words for each file

**Procedure:** `getStopWord()`

```
1 Begin
2   Read in the file txt_title.txt as infile
3   Prepare the output file (outfile) named file_word_count.txt
4   while reading in the content of infile do
5     // file: one line content in infile, i.e. the title of each file
6     Read in the file "shakespeare_works/" + file + ".txt" as in
7     while reading each line(line) in file in do
```

```

8         if find an English word (called word) then
9             Do word stemming
10            wordList[word]  $\leftarrow$  wordList[word] + 1
11            wordNumOfDoc[file]  $\leftarrow$  wordNumOfDoc[file] + 1
12            wordDocs[word]  $\leftarrow$  file
13        endif
14    end
15    Output the wordNumOfDoc to outfile
16    closefile(in)
17 end
18 closefile(infile)
19 Sort(wordDocs) Prepare the output file (out) named
    word_count.txt
20 Prepare the output file (out2) named stop_words.txt
21 Prepare the output file (out3) named word_docs.txt
22 for each item(word) in wordDocs do
23     Output the word $\rightarrow$ content and word $\rightarrow$ frequency to out3
24     if word $\rightarrow$ frequency > THRESHOLD then
25         Output the word $\rightarrow$ content to out2
26     endif
27     Output the word $\rightarrow$ content and wordList[word $\rightarrow$ content] to out
28 end
29 closefile(out)
30 closefile(out2)
31 closefile(out3)
32 End

```

## 2.4 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/scripts/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.

## 2.5 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 structruers 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:

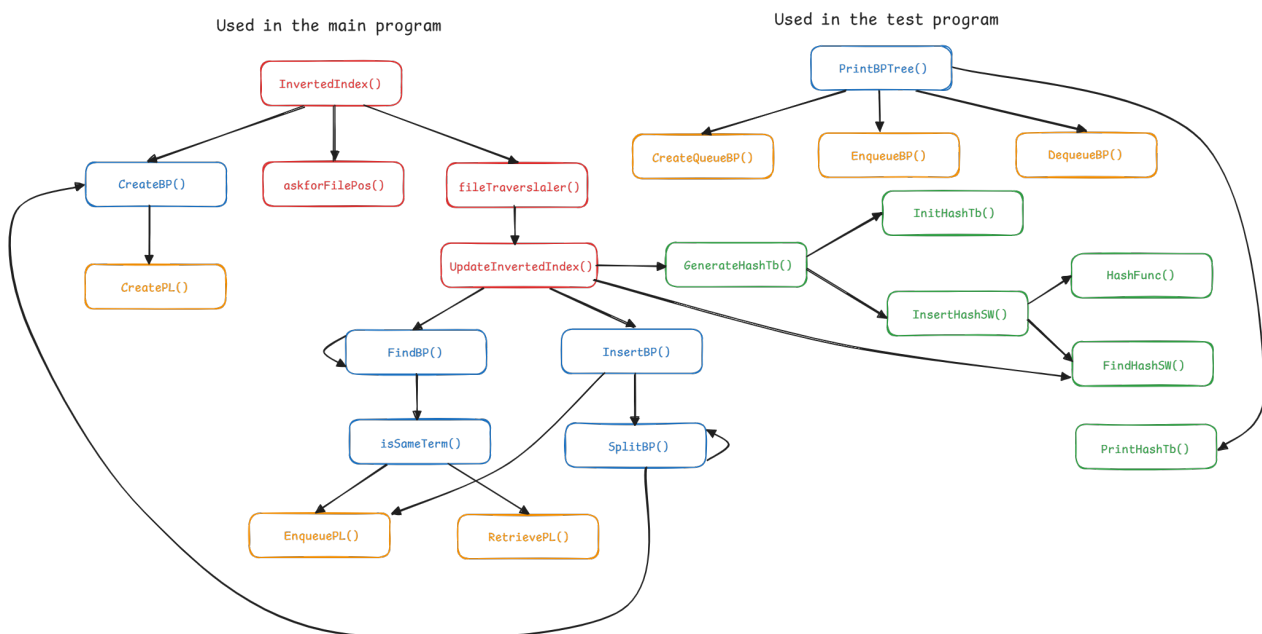


Figure 1: Relation diagram of all functions for inverted index

- **Red:** Overall Functions
- **Blue:** B+ Tree Operations

- **Green:** Hashing Functions
- **Yellow:** Other Functions

We'll introduce these functions in detail below.

### 2.5.1 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**)

```

1  Begin
2    InvIndex  $\leftarrow$  CreateBP()
3    askforFilePos(dir, fname, isTest)
4    // dir: directory name, fname: filename
5    InvIndex  $\leftarrow$  fileTraversaler(InvIndex, dir, fname, isTest,
6    containStopWords)
7    if InvIndex→size > 0 then
8      print("Build successfully!")
9    else
10     print("Fail to build an inverted index!")
11  endif
12  return InvIndex
13  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**)

```

1  Begin
2      if isTest is true then
3          print("Now testing the correctness of inverted Index:")
4          print("Please input the name of the input sample file:")
5          input("Name:", fname)
6      else
7          dir ← SHAKESPEAREDIR
8          print("Now building an inverted Index:")
9          print("Do you want to search something in the default
10             path({ dir })?")
11          print("Please input yes or no: ")
12          input(choice)
13          switch choice
14              case 'y': case 'Y':
15                  break
16              case 'n': case 'N':
17                  print("Please input the directory of the documents:")
18                  print("Path: ")
19                  input(dir)
20                  break
21              default:
22                  print("Choice Error!")
23                  exit(1)
24                  break
25          end
26      endif
27  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**)

```

1  Begin
2      docCnt = 0 // Count the number of documents and act as the
                    index of the documents at the same time
3      if isTest is false then
4          if dir exists then
5              for file in the dir directory do
6                  filename ← the name of file // string
7                  docNames[docCnt] ← filename
8                  // docNames: an array containing names of
                        documents(global variable)
9                  wholePath ← dir + "/" + filename
10                 // wholePath: the complete path of the file to be read
11                 end
12                 closefile(fp)
13             else
14                 Error("Could not open directory!")
15             endif
16         else
17             docNames[docCnt] ← fname
18             dir ← DEFAULTFILEPOS // Constant: "tests"(string)
19             wholePath ← dir + "/" + fname

```

```

20    endif
21     $fp \leftarrow \text{openfile}(wholePath, "r")$  // read mode
22    //  $fp$ : the pointer to the file
23     $T \leftarrow \text{UpdateInvertedIndex}(T, docCnt, fp, containStopWords)$ 
24    return  $T$ 
25 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)

```

1 Begin
2    $H \leftarrow \text{GenerateHashTb}()$ 
3   while reading texts in the file pointed by  $fp$  do
4     if find an English word then do
5        $term \leftarrow$  the English word
6       if  $containStopWords$  is false and  $\text{FindHashSW}(term, H,$ 
7         true)  $\geq 0$  then
8         continue
9       endif
10       $term \leftarrow \text{WordStemming}(term)$ 
11       $isDuplicated \leftarrow \text{false}$ 
12       $nodebp \leftarrow \text{FindBP}(term, docCnt, T, isDuplicated)$ 
13      if  $isDuplicated$  is false then
14         $T = \text{InsertBP}(term, docCnt, nodebp, T)$ 
15      endif

```

```

15      else
16          continue searching for next English word.
17      endif
18  end
19  return  $T$ 
20 End

```

## 2.5.2 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()

```

1  Begin
2      Allocate a memory block for new B+ tree  $T$ 
3      for all data and children in  $T$  do
4          Allocate memory blocks for term and poslist of the data, and
            children
5          // Use CreatePL() to initialize the poslist
6      end
7       $T \rightarrow size \leftarrow 0$ 
8       $T \rightarrow childrenSize \leftarrow 0$ 
9       $T \rightarrow parent \leftarrow NULL$ 
10     return  $T$ 
11 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)

```

1  Begin
2    if  $T \rightarrow \text{childrenSize} = 0$  then
3      isSameTerm(term, docCnt, T, flag, isSearch)
4      return T
5    endif
6     $pos \leftarrow -1$ 
7    for i in range(0,  $T \rightarrow \text{size}$ ) do // not contains  $T \rightarrow \text{size}$ 
8      if term has less lexicographical order than  $T \rightarrow \text{data}[i] \rightarrow \text{term}$ 
9      then
10        $pos \leftarrow i$ 
11       break
12     endif
13   end
14   if  $pos = -1$  then
15      $pos \leftarrow i$ 
16   endif return FindBP(term, docCnt,  $T \rightarrow \text{children}[pos]$ , isSearch)
17 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)`

```

1  Begin
2      if nodebp→size > 0 then
3          for i in range(0, T→size) do // not contains T→size
4              if term = nodebp→data[i]→term then
5                  if isSearch is false then
6                      EnqueuePL(docCnt, nodebp→data[i]→poslist)
7                  else
8                      poslist ← nodebp→data[i]→poslist
9                      size ← poslist→size
10                     cnt ← 0
11                     print("Successfully find the word!")
12                     print("The word was found in files below:")
13                     posArr ← RetrievePL(poslist)
14                     for j in range(0, size) do // not contains size
15                         if posArr[j][1] ≤ 1 then
16                             print("{ docNames[posArr[j][0]]: { posArr[j][1]}  
time")
17                         else
18                             print("{ docNames[posArr[j][0]]: { posArr[j][1]}  
times")
19                         endif
20                         cnt ← cnt + posArr[j][1]
21                     end
22                     print("Frequency: { cnt}")
23                     print("_____")
24                 endif

```



```

25          flag ← true
26          break
27      endif
28  end
29  endif
30 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**)

```

1  Begin
2      nodebp→data[nodebp→size]→term ← term
3      EnqueuePL(docCnt, nodebp→data[nodebp→size]→poslist)
4      nodebp→size ← nodebp→size + 1
5      Sort(nodebp→data)
6      Tree ← SplitBP(nodebp, Tree)
7      return Tree
8  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)

```

1  Begin
    // ORDER: (constant)the order of B+ trees if (nodebp→
2  childrenSize = 0 and nodebp→size ≤ ORDER) or (nodebp→
    childrenSize > 0 and nodebp→size < ORDER) then
3      return Tree
4  endif
5
6  // lnodebp, rnodebp: the left and right part of the split node
7  // tmpNodebp: store the node temporarily
8  // parent: the parent node of nodebp
9  // cut: the position of the middle data
10
11  parent ← nodebp→parent
12  if parent = NULL then
13      tmpNodebp ← CreateBP()
14      Allocate memory for parent
15      Tree ← parent ← tmpNodebp
16  endif
17  lnodebp ← CreateBP()
18  rnodebp ← CreateBP()
19  lnodebp→parent ← rnodebp→parent ← parent
20  if nodebp→childrenSize = 0 then
21      cut ← LEAFCUT // constant: (ORDER / 2 + 1)
22      for i in range(0, cut) do // not contains cut
23          lnodebp→data[i] ← nodebp→data[i]
24      end
25      lnodebp→size ← cut
26      for j in range(cut, nodebp→size) do // not contains nodebp→
          size
27          rnodebp→data[j - cut] ← nodebp→data[j]
28      end

```

```

29     rnodebp→size ← nodebp→size - cut
30  else
31     cut ← NONLEAFCUT // constant: (ORDER / 2)
32     for i in range(0, cut + 1) do // not contains cut + 1
33         if i ≠ cut then
34             lnodebp→data[i] ← nodebp→data[i]
35         endif
36         lnodebp→children[i] ← nodebp→children[i]
37         lnodebp→children→parent ← lnodebp
38     end
39     lnodebp→size ← cut
40     lnodebp→childrenSize ← cut + 1
41     for j in range(cut + 1, nodebp→size) do // not contains
42         nodebp→size
43         rnodebp→data[j - cut - 1] ← nodebp→data[j]
44     end
45     for j in range(cut + 1, nodebp→childrenSize) do // not
46         contains nodebp→childrenSize
47         rnodebp→children[j - cut - 1] ← nodebp→children[j]
48         rnodebp→children[j - cut - 1]→parent ← rnodebp
49     end
50     rnodebp→size ← nodebp→size - cut - 1 rnodebp→childrenSize
51     ← nodebp→childrenSize - cut - 1
52 end
53 parent→data[parent→size] ← nodebp→data[cut]
54 parent→size ← parent→size + 1
55 if parent→childrenSize > 0 then
56     for i in range(0, parent→childrenSize) do // not contains
57         parent→childrenSize
58         if parent→children[i] = nodebp then
59             parent→children[i] ← lnodebp
60             break
61         endif
62     end
63 end

```

```

60     parent→children[parent→childrenSize] ← lnodebp
61     parent→childrenSize ← parent→childrenSize + 1
62     endif
63     parent→children[parent→childrenSize] ← rnodebp
64     parent→childrenSize ← parent→childrenSize + 1
65     Sort(parent→data)
66     Sort(parent→children)
67     Tree ← SplitBP(parent, Tree)
68     return Tree
69 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**)

```

1  Begin
2    print("B+ Tree of Inverted Index:")
3    q ← CreateQueueBP()
4    EnqueueBP(T, q)
5    EnqueueBP(NULL, q)
6    while q→size > 0 do
7      nodebp ← DequeueBP(q)
8      if nodebp is NULL then
9        change to a newline
10     if q→size > 0 then
11       EnqueueBP(NULL, q)
12     endif
13   else
14     print("[")
15     for i in range(0, nodebp→size) do // not contains nodebp→
        size

```

```

16         if  $i = 0$  then
17             print( $nodebp \rightarrow data[i] \rightarrow term$ )
18         else
19             print(", { $nodebp \rightarrow data[i] \rightarrow term$ }")
20         endif
21     end
22     print("[")
23 endif
24 if  $nodebp$  is not NULL then
25     for  $i$  in range (0,  $nodebp \rightarrow childrenSize$ ) do // not contains
26          $nodebp \rightarrow childrenSize$ 
27         EnqueueBP( $nodebp \rightarrow children[i]$ ,  $q$ )
28     end
29 endif
30 end
31 End

```

### 2.5.3 Hashing Operations

#### (1) GenerateHashTb

**Function:** Build a hash table.

**Inputs:** None

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

**Procedure:** GenerateHashTb()

```

1  Begin
2       $H \leftarrow \text{InitHashTb}()$ 
3       $fname \leftarrow \text{STOPWORDPATH}$  // constant: "stop_words.txt"
4       $fp \leftarrow \text{openfile}(fname, "r")$  // read mode
5      if  $fp$  is NULL then
6          Error("Fail to open the file of stopwords!")
7      endif
8      while reading texts in the file pointed by  $fp$  do
9          if find an English word then do

```

```

10      term ← the English word
11      InsertHashSW(term, H)
12  endif
13  end
14  closefile(fp)
15  return H
16 End

```

## (2) InitHashTb

**Function:** Initialization of the hash table.

**Inputs:** None

**Outputs:** A new initialized hash table

**Procedure:** InitHashTb()

```

1 Begin
2   Allocate memory block for H // HashTb
3   if H is NULL then
4     Error("Fail to create a hash table for stopwords!")
5   end
6   H→size ← STOPWORDSUM // maximum size
7   for i in range(0, H→size) do // not contains H→size
8     Allocate memory block for H→data[i]
9     if H→data[i] is NULL then
10      Error("Fail to create a hash table for stopwords!")
11    end
12    H→data[i]→stopword
13    H→data[i]→info ← Empty // constant: 0
14  end
15  return H
16 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 stopwords, or just search the term in the hash table

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

```

1  Begin
2      collisionNum  $\leftarrow$  0
3      pos  $\leftarrow$  HashFunc(stopword, H $\rightarrow$ size)
4      if justSearch is true and (H $\rightarrow$ data[pos] $\rightarrow$ info = Empty or H $\rightarrow$ 
5          data[pos] $\rightarrow$ stopword = stopword) then
6          return -1
7      endif
8      while H $\rightarrow$ data[pos] $\rightarrow$ info  $\neq$  Empty and H $\rightarrow$ data[pos] $\rightarrow$ stopword
9          = stopword do
10         collisionNum  $\leftarrow$  collisionNum + 1 pos  $\leftarrow$  pos + 2 *
11         collisionNum - 1
12         if pos  $\geq$  H $\rightarrow$ size then
13             pos  $\leftarrow$  pos - H $\rightarrow$ size
14         endif
15     end
16     return pos
17 End

```

**(4) InsertHashSW**

**Function:** Insert a new stopwords 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)

```

1 Begin
2   pos  $\leftarrow$  FindHashSW(stopword, H, false)
3   // Legitimate: (constant) 1
4   if (H→data[pos]→info  $\neq$  Legitimate) then
5     H→data[pos]→info  $\leftarrow$  Legitimate
6     H→data[pos]→stopword  $\leftarrow$  stopword
7   endif
8 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)

```

1 Begin
2   val  $\leftarrow$  0
3   for each character ch in stopword do
4     val = (val << 5) + integer(ch)
5   end
6   return val % size
7 End

```

### 2.5.4 Other Functions

#### (1) CreateQueueBP

**Function:** Create the queue

**Inputs:** None

**Outputs:** A new queue *Q*



**Procedure:** CreateQueueBP()

```

1 Begin
2   Allocate memory block for the queue  $Q$ 
3    $Q \rightarrow size \leftarrow 0$ 
4    $Q \rightarrow front \leftarrow Q \rightarrow rear \leftarrow 0$ 
5   return  $Q$ 
6 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)

```

1 Begin
2   if  $Q \rightarrow size \geq SIZE$  then
3     Error("Full B+-tree-item queue!")
4   endif
5    $Q \rightarrow data[Q \rightarrow rear] \leftarrow nodebp$ 
6    $Q \rightarrow rear \leftarrow Q \rightarrow rear + 1$ 
7    $Q \rightarrow size \leftarrow Q \rightarrow size + 1$ 
8 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)

```

1 Begin
```

```

2   if  $Q \rightarrow size = 0$  then
3       Error("Empty B+-tree-item queue!")
4   endif
5    $returnNodeBP \leftarrow Q \rightarrow data[Q \rightarrow front]$ 
6    $Q \rightarrow front \leftarrow Q \rightarrow front + 1$ 
7    $Q \rightarrow size \leftarrow Q \rightarrow size - 1$ 
8   return  $returnNodeBP$ 
9 End

```

#### (4) CreatePL

**Function:** Create the poslist

**Inputs:** None

**Outputs:** A new PosList  $L$

**Procedure:** CreatePL()

```

1 Begin
2   Allocate memory blocks for  $L(\text{PosList})$ ,  $L \rightarrow front(\text{PosData})$ ,  $L \rightarrow$ 
    $rear(\text{PosData})$ 
3    $L \rightarrow size \leftarrow 0$ 
4    $L \rightarrow front \leftarrow L \rightarrow rear$ 
5    $L \rightarrow rear \rightarrow pos \leftarrow -1$ 
6   return  $L$ 
7 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)

```

1 Begin

```

```

2   if  $L \rightarrow rear \rightarrow pos \neq pos$  then
3       Allocate memory block for  $tmp(PosData)$ 
4       if  $tmp$  is NULL then
5           Error("Fail to create a new position data!")
6       endif
7        $tmp \rightarrow pos \leftarrow pos$ 
8        $tmp \rightarrow time \leftarrow 1$ 
9        $tmp \rightarrow next \leftarrow L \rightarrow rear \rightarrow next$ 
10       $L \rightarrow rear \rightarrow next \leftarrow tmp$ 
11       $L \rightarrow rear \leftarrow tmp$ 
12       $L \rightarrow size \leftarrow L \rightarrow size + 1$ 
13  else
14       $L \rightarrow rear \rightarrow time \leftarrow L \rightarrow rear \rightarrow time + 1$ 
15  endif
16  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)

```

1  Begin
2      if  $L \rightarrow size = 0$  then
3          Error("Empty position-data queue!")
4      endif
5      Allocate memory block for  $posArr$   $cur \leftarrow L \leftarrow front \leftarrow next$ 
6       $i \leftarrow 0$ 
7      while  $cur \neq NULL$  do
8           $posArr[i][0] \leftarrow cur \rightarrow pos$ 
9           $posArr[i][1] \leftarrow cur \rightarrow time$ 

```

```

10       $cur \leftarrow cur \rightarrow next$ 
11       $i \leftarrow i + 1$ 
12  end
13  return posArr
14 End

```

## 2.6 Query

### 2.6.1 Search Function

**Inputs:**

- *query*: User's query in string form
- *T*: Inverted index
- *pageSize*: Number of documents to be displayed per query
- *threshold*: The proportion of search terms to the total number of terms

**Outputs:** The query result

**Procedure:** search(*query*: string, *T*: BplusTree, *pageSize*: integer, *threshold*: double)

```

1  Begin
2    for read every character in query do
3      remove the Punctuation and Whitespace
4      if get the word then
5        store the word itself and the idf of word in a vector
6      endif
7    end
8    sort the vector in descending order
9    if no word then
10     return
11   else if one word then
12     search the word in T
13     sort the file name by tf
14     print(file, tf)
15   else
16     according to the threshold get the word needed to searched

```

```
17      for every word do
18          search the word in  $T$ 
19          store the result in a hash table
20          (the hash table store the file name and their  $tf-idf$ )
21      end
22      print( $file$ ,  $td-idf$ )
23  endif
24 End
```

The main idea of the function is to implement a search functionality that retrieves documents related to a given query from a B+ tree. Here is a more detailed summary of the function logic:

1. **Query parsing:** The code starts by loading and analyzing the query string, breaking it down into words while converting each to lowercase and stemming. It checks if each word exists in a stop words list or if it is present in the inverted index.
2. **Valid word collection:** Valid words are added to a vector, while warnings are displayed for invalid words (such as stop words or words not found in the inverted index).
3. **Search condition assessment:** Depending on the number of valid words, the processing is divided into two modes. If there is only one valid word, it searches directly in the inverted index and returns the matching documents with their TF values. For multiple valid words, the code calculates their TF-IDF values.
4. **Inverted index lookup:** For multi-word queries, the code traverses the inverted index for each valid word, using a hash table to store document IDs and their corresponding TF-IDF scores while ensuring it only retains documents that contain all the query words.
5. **Result sorting and output:** Finally, the results are sorted based on the TF-IDF values, and the document names along with their TF-IDF scores are printed, adhering to a specified limit on the number of results (page size).

Overall, this function provides an efficient document retrieval mechanism through valid word filtering and TF-IDF score calculation, suitable for the context of search engines or information retrieval systems.

### 2.6.2 Search Test

**Inputs:** Files of *The Works*

**Outputs:** The results of queries

**Procedure:** SearchTest()

```
1 Begin
2   load necessary resources and establish inverted index
3   while test_allow do
4     get serial number of the test input form user
5     get the query from the input file
6     get important parameters from user's input
7     call the search function with the query and parameters
8     ask user if they want to continue testing or not and set
      test_allow accordingly
9   end
10 End
```

The main idea of the code is to set up a testing framework for the “ShakespeareFinder” search application. This includes loading necessary resources, allowing users to input parameters from a test file, executing search queries multiple times to evaluate performance, and managing repeated tests.

### 2.6.3 Some helper functions

**(1) Load function: loadWordIdf & loadFileWordsNum & loadStopWords**

These functions have a similar structure. The main idea is to load resources from existing files. See the pseudocode for details.

**Inputs:** Files of *The Works*

**Outputs:** The results of queries

**Procedure:** SearchTest()

```
1 Begin
2   load the input FILE
3   read resource from the file
4   load the resource into a hash table
5 End
```

## (2) Search helper: FindBP2 & isSameTerm2 & RetrievePL2

These functions are very similar to their versions without the “2”, but in order to better cooperate with the search function, the type of data they return is changed to `vector<pair<docId, tf>>`, and the logic of getting data is changed accordingly.

# Chapter 3: Testing Results

## 3.1 Inverted Index Tests

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.
- Build an inverted index from a single file, or a directory with a bunch of files.
- Check if the inverted index can eliminate all stopwords.

### Warning

In the following tests, we will use the test programs to verify the correctness of our sub-programs separately, and we **won't tell you the usage of these instructions** we use below. If you are curious about it, please read the `README.md` file in the directory `code`, which will guide you how to run these instructions.

### 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/source/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

-----

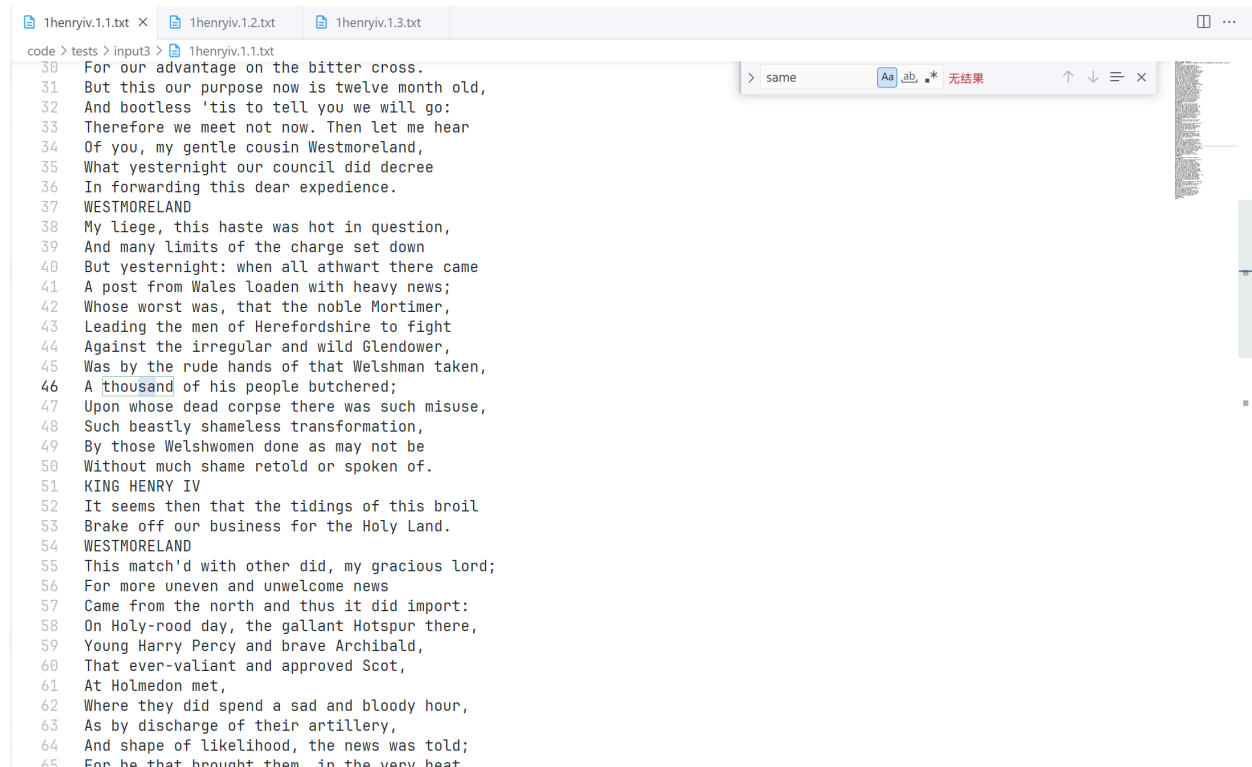
Find 3: **fantastic**

Sorry, no such word in the inverted index!

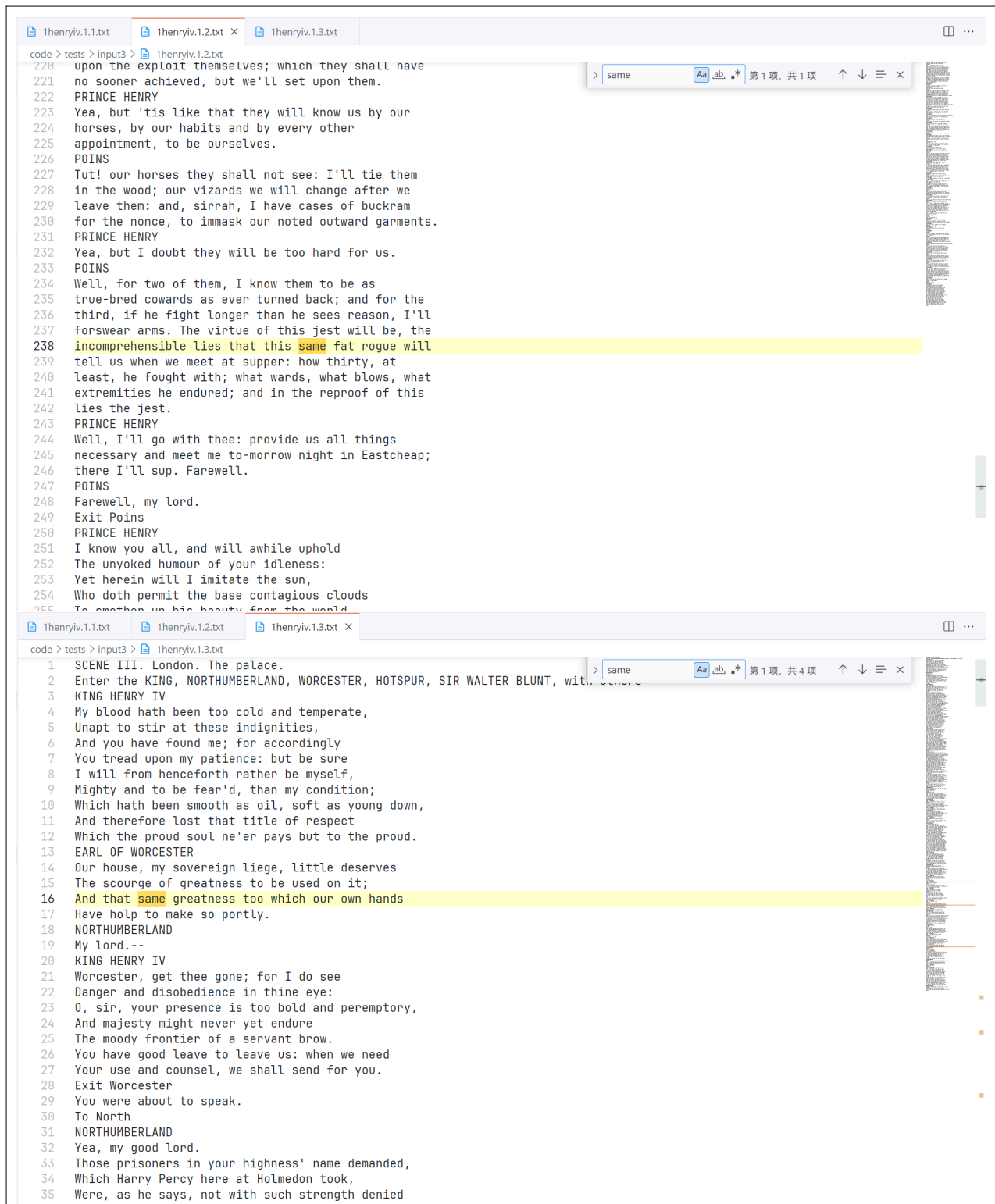
-----

Verification by using finding function in Visual Studio Code

Find 1: same



```
code > tests > input3 > 1henryiv.1.1.txt
30 For our advantage on the bitter cross.
31 But this our purpose now is twelve month old,
32 And bootless 'tis to tell you we will go:
33 Therefore we meet not now. Then let me hear
34 Of you, my gentle cousin Westmoreland,
35 What yesternight our council did decree
36 In forwarding this dear expedience.
37 WESTMORELAND
38 My liege, this haste was hot in question,
39 And many limits of the charge set down
40 But yesternight: when all athwart there came
41 A post from Wales loaden with heavy news;
42 Whose worst was, that the noble Mortimer,
43 Leading the men of Herefordshire to fight
44 Against the irregular and wild Glendower,
45 Was by the rude hands of that Welshman taken,
46 A thousand of his people butchered;
47 Upon whose dead corpse there was such misuse,
48 Such beastly shameless transformation,
49 By those Welshwomen done as may not be
50 Without much shame retold or spoken of.
51 KING HENRY IV
52 It seems then that the tidings of this broil
53 Brake off our business for the Holy Land.
54 WESTMORELAND
55 This match'd with other did, my gracious lord;
56 For more uneven and unwelcome news
57 Came from the north and thus it did import:
58 On Holy-rood day, the gallant Hotspur there,
59 Young Harry Percy and brave Archibald,
60 That ever-valiant and approved Scot,
61 At Holmedon met,
62 Where they did spend a sad and bloody hour,
63 As by discharge of their artillery,
64 And shape of likelihood, the news was told;
65 For he that brought them in the very heat
```



Find 2: star

The screenshot shows a web application with two panels. The top panel is titled 'code > tests > input3 > 1henryiv.1.1.txt' and displays a search result for the keyword 'star'. The search bar shows 'star' and the results are listed as '无结果' (No results). The text content is from Shakespeare's Henry IV, Part 1, Act 1, Scene 1, London. The bottom panel is titled 'code > tests > input3 > 1henryiv.1.2.txt' and displays the source text for the same file. The text content is from Shakespeare's Henry IV, Part 1, Act 1, Scene 2, London. Line 19 is highlighted: 'purses go by the moon and the seven stars, and not by Phoebus, he, 'that wandering knight so fair.' And,'.

1henryiv.1.1.txt 1henryiv.1.2.txt 1henryiv.1.3.txt

code > tests > input3 > 1henryiv.1.1.txt

> star Aa ab,\* 无结果

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

1henryiv.1.1.txt 1henryiv.1.2.txt 1henryiv.1.3.txt

code > tests > input3 > 1henryiv.1.2.txt

> star Aa ab,\* 第1项, 共1项

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

```

code > tests > input3 > 1henryiv.1.3.txt
84  To such a person and in such a place,
85  At such a time, with all the rest retold,
86  May reasonably die and never rise
87  To do him wrong or any way impeach
88  What then he said, so he unsay it now.
89  KING HENRY IV
90  Why, yet he doth deny his prisoners,
91  But with proviso and exception,
92  That we at our own charge shall ransom straight
93  His brother-in-law, the foolish Mortimer;
94  Who, on my soul, hath wilfully betray'd
95  The lives of those that he did lead to fight
96  Against that great magician, damn'd Glendower,
97  Whose daughter, as we hear, the Earl of March
98  Hath lately married. Shall our coffers, then,
99  Be emptied to redeem a traitor home?
100 Shall we but treason? and indent with fears,
101 When they have lost and forfeited themselves?
102 No, on the barren mountains let him starve;
103 For I shall never hold that man my friend
104 Whose tongue shall ask me for one penny cost
105 To ransom home revolted Mortimer.
106 HOTSPUR
107 Revolted Mortimer!
108 He never did fall off, my sovereign liege,
109 But by the chance of war; to prove that true
110 Needs no more but one tongue for all those wounds,
111 Those mouthed wounds, which valiantly he took
112 When on the gentle Severn's sedgy bank,
113 In single opposition, hand to hand,
114 He did confound the best part of an hour
115 In changing hardiment with great Glendower:
116 Three times they breathed and three times did
117 they drink,
118 Upon agreement, of swift Severn's flood;
119 Who then affrighted with their bloody looks

```

Find 3: fantastic

```

1henryiv.1.1.txt 1henryiv.1.2.txt 1henryiv.1.3.txt
code > tests > input3 > 1henryiv.1.1.txt
> fantastic 无结果
1 SCENE I. London. The palace.
2 Enter KING HENRY, LORD JOHN OF LANCASTER, the EARL of WESTMORELAND, SIR WALTER BLUNT, and others
3 KING HENRY IV
4 So shaken as we are, so wan with care,
5 Find we a time for frighted peace to pant,
6 And breathe short-winded accents of new broils
7 To be commenced in strands afar remote.
8 No more the thirsty entrance of this soil
9 Shall daub her lips with her own children's blood;
10 Nor more shall trenching war channel her fields,
11 Nor bruise her flowerets with the armed hoofs
12 Of hostile paces: those opposed eyes,
13 Which, like the meteors of a troubled heaven,
14 All of one nature, of one substance bred,
15 Did lately meet in the intestine shock
16 And furious close of civil butchery
17 Shall now, in mutual well-beseeming ranks,
18 March all one way and be no more opposed
19 Against acquaintance, kindred and allies:
20 The edge of war, like an ill-sheathed knife,
21 No more shall cut his master. Therefore, friends,
22 As far as to the sepulchre of Christ,
23 Whose soldier now, under whose blessed cross
24 We are impressed and engaged to fight,
25 Forthwith a power of English shall we levy;
26 Whose arms were moulded in their mothers' womb
27 To chase these pagans in those holy fields
28 Over whose acres walk'd those blessed feet
29 Which fourteen hundred years ago were nail'd
30 For our advantage on the bitter cross.
31 But this our purpose now is twelve month old,
32 And bootless 'tis to tell you we will go:
33 Therefore we meet not now. Then let me hear
34 Of you, my gentle cousin Westmoreland.

```

```

code > tests > input3 > 1henryiv.1.2.txt
1 SCENE 11. London. An apartment of the Prince's.
2 Enter the PRINCE OF WALES and FALSTAFF
3 FALSTAFF
4 Now, Hal, what time of day is it, lad?
5 PRINCE HENRY
6 Thou art so fat-witted, with drinking of old sack
7 and unbuttoning thee after supper and sleeping upon
8 benches after noon, that thou hast forgotten to
9 demand that truly which thou wouldst truly know.
10 What a devil hast thou to do with the time of the
11 day? Unless hours were cups of sack and minutes
12 capons and clocks the tongues of bawds and dials the
13 signs of leaping-houses and the blessed sun himself
14 a fair hot wench in flame-coloured taffeta, I see no
15 reason why thou shouldst be so superfluous to demand
16 the time of the day.
17 FALSTAFF
18 Indeed, you come near me now, Hal; for we that take
19 purses go by the moon and the seven stars, and not
20 by Phoebus, he, 'that wandering knight so fair.' And,
21 I prithee, sweet wag, when thou art king, as, God
22 save thy grace,--majesty I should say, for grace
23 thou wilt have none,--
24 PRINCE HENRY
25 What, none?
26 FALSTAFF
27 No, by my troth, not so much as will serve to
28 prologue to an egg and butter.
29 PRINCE HENRY
30 Well, how then? come, roundly, roundly.
31 FALSTAFF
32 Marry, then, sweet wag, when thou art king, let not
33 us that are squires of the night's body be called
34 thieves of the day's beauty: let us be Diana's
35 foresters, gentlemen of the shade, minions of the
36 moon: and let men say we be men of good government.

code > tests > input3 > 1henryiv.1.3.txt
86 May reasonably die and never rise
87 To do him wrong or any way impeach
88 What then he said, so he unsay it now.
89 KING HENRY IV
90 Why, yet he doth deny his prisoners,
91 But with proviso and exception,
92 That we at our own charge shall ransom straight
93 His brother-in-law, the foolish Mortimer;
94 Who, on my soul, hath wilfully betray'd
95 The lives of those that he did lead to fight
96 Against that great magician, damn'd Glendower,
97 Whose daughter, as we hear, the Earl of March
98 Hath lately married. Shall our coffers, then,
99 Be emptied to redeem a traitor home?
100 Shall we but treason? and indent with fears,
101 When they have lost and forfeited themselves?
102 No, on the barren mountains let him starve;
103 For I shall never hold that man my friend
104 Whose tongue shall ask me for one penny cost
105 To ransom home revolted Mortimer.
106 HOTSPUR
107 Revolted Mortimer!
108 He never did fall off, my sovereign liege,
109 But by the chance of war; to prove that true
110 Needs no more but one tongue for all those wounds,
111 Those mouthed wounds, which valiantly he took
112 When on the gentle Severn's sedgy bank,
113 In single opposition, hand to hand,
114 He did confound the best part of an hour
115 In changing hardiment with great Glendower:
116 Three times they breathed and three times did
117 they drink,
118 Upon agreement, of swift Severn's flood;
119 Who then, affrighted with their bloody looks,
120 Ran fearfully among the trembling reeds,
121 And hid his crested head in the hollow bank

```

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

### 3.1.2 Single File to Multiple Files Test

We executed our first test based on a single file and several files, but our ultimate goal is to let our mini search engine to search something from a dozens

of files(i.e. *the Complete Works of Shakespeare*). So it's necessary for us to test whether the inverted index can be built from tons of files. Note that the Works is in the directory called `code/source/shakespeare_works`.

Case 1: Search some dedicated words from piles of files

```
./invIndexTest -f=3
```

```
Now building an inverted Index:
```

```
Please input the directory of the documents:
```

```
Path: ../source/shakespeare_works
```

```
Build successfully!
```

```
Finding Words Mode(only supports single word finding):
```

```
Find 1: hamlet
```

```
Successfully find the word!
```

```
The word was found in files below:
```

```
hamlet.1.1.txt: 3 times
```

```
hamlet.1.2.txt: 42 times
```

```
# Deleberate omisssion
```

```
hamlet.5.1.txt: 45 times
```

```
hamlet.5.2.txt: 83 times
```

```
Frequency: 470
```

```
-----  
Find 2: juliet
```

```
Successfully find the word!
```

```
The word was found in files below:
```

```
measure.1.2.txt: 3 times
```

```
measure.1.4.txt: 1 time
```

```
# Deleberate omisssion
```

```
romeo_juliet.5.2.txt: 1 time
```

```
romeo_juliet.5.3.txt: 19 times
```

```
Frequency: 199
```

```
-----  
Find 3: macbeth
```

```
Successfully find the word!
```

```
The word was found in files below:
```

```

macbeth.1.1.txt: 1 time
macbeth.1.2.txt: 4 times

# Deleberate omisssion

macbeth.5.7.txt: 8 times
macbeth.5.8.txt: 7 times
Frequency: 285
-----

```

Case 2: Search some universal words from piles of files

```

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

Finding Words Mode(only supports single word finding):
Find 1: moon
Successfully find the word!
The word was found in files below:
1henryiv.1.2.txt: 5 times
1henryiv.1.3.txt: 1 time

# Deleberate omisssion

winters_tale.4.3.txt: 1 time
winters_tale.4.4.txt: 1 time
Frequency: 152
-----

Find 2: happy
Successfully find the word!
The word was found in files below:
1henryiv.2.2.txt: 1 time
1henryiv.4.3.txt: 1 time

# Deleberate omisssion

winters_tale.1.2.txt: 2 times
winters_tale.4.4.txt: 3 times

```



```

Frequency: 278
-----
Find 3: hit
Successfully find the word!
The word was found in files below:
1henryiv.2.4.txt: 1 time
2henryiv.1.1.txt: 1 time

# Deleberate omisssion
VenusAndAdonis.txt: 2 times
winters_tale.5.1.txt: 1 time
Frequency: 74
-----

```

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

### 3.1.3 Stopwords Test

Finally, 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 not included(default situation)

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

Finding Words Mode(only supports single word finding):
Find 1: much
Sorry, no such word in the inverted index!
-----
Find 2: you
Sorry, no such word in the inverted index!
-----
Find 3: great

```

Sorry, no such word in the inverted index!

-----

### Case 2: stopwords **included**

```
./invIndexTest -f=3 -s
```

Now building an inverted Index:

Please input the directory of the documents:

Path: `../source/shakespeare_works`

# Delebrate omission for the display of all stopwords

Build successfully!

Finding Words Mode(only supports single word finding):

Find 1: `much`

# Delebrate omission for the very long position list

winters\_tale.5.1.txt: 4 times

winters\_tale.5.2.txt: 2 times

winters\_tale.5.3.txt: 7 times

Frequency: 1070

-----

Find 2: `you`

# Delebrate omission for the very long position list

winters\_tale.5.1.txt: 40 times

winters\_tale.5.2.txt: 19 times

winters\_tale.5.3.txt: 29 times

Frequency: 14249

-----

Find 3: `great`

# Delebrate omission for the very long position list

winters\_tale.5.1.txt: 3 times

winters\_tale.5.2.txt: 1 time

winters\_tale.5.3.txt: 1 time

Frequency: 1032

-----

Actually, the inverted index can eliminate all stopwords, but due to space limitation, we won't list all tests about them.

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

Although we can't make a thorough test for the inverted index, but from the above tests, we can assure that our inverted index have no obvious error(maybe there're several small bugs existing).

### 3.1.4 Speed Test

#### Note

The specific analysis and comments about speed tests are written in **Chapter 4**.

To analyze the time complexity of the inverted index, especially the algorithms regarding the **finding** and **insertion** operations of B+ tree, we devise some timing tests for **different numbers of words** in *The Works*. The results are shown below:

Number of Words(roughly)	100,000	200,000	400,000	600,000	800,000	880,000
Iterations	10	10	10	10	5	5
Ticks	1988219	3704927	7317245	11594770	7985568	8438331
Total Time(s)	1.99	3.70	7.32	11.59	7.99	8.44
Duration(s)	0.199	0.370	0.732	1.159	1.598	1.688

Table 1: Speed Tests for Inverted Index

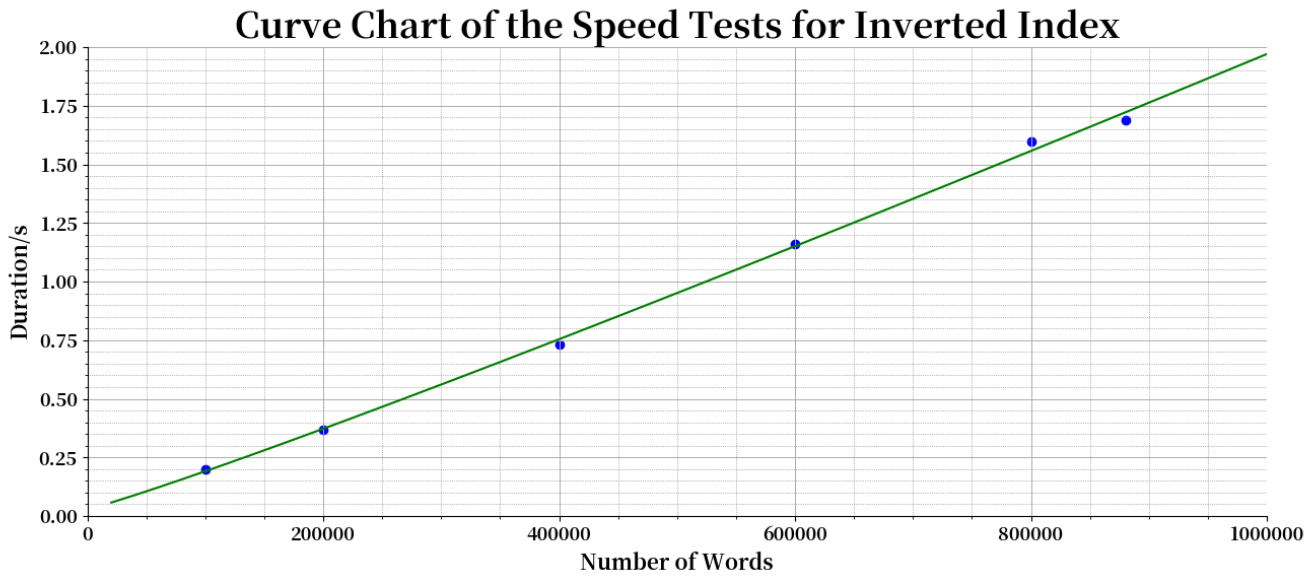


Figure 2: Curve Chart of the Speed Tests for Inverted Index

As you can see, these data points can be fitted with the product of linear function and **logarithmic function** (although the curve looks like a straight line due to the limitation of aspect ratio). So the result indicates that the time complexity of building inverted index approaches  $O(\log N)$ , and you will see the detailed explanation in Chapter 4.

## 3.2 Query Tests

Different from the oriented-procedure inverted index tests, the query tests are based on distinct and typical **input data**, which consider the performance, speed and threshold analysis simultaneously.

### 3.2.1 input0

- **content** : 1000\*'a'
- **purpose**: Test for illegal words of extreme length
- **expected results**:

Warning: "a...a" is not in the inverted index and will be ignored.  
 Ops, your query is in StopList or empty, so there are no documents retrieved.

- **test results**:

output	ticks(ticks/search)	time(s/search)
correct	155.835	1.6e-4

The time is averaged from 1000 runs. Since irrelevant words will not enter the search phase, the threshold is irrelevant to this test.

### 3.2.2 input1

- **content** : peas and beans
- **purpose**: One stop word, two rare words (each appearing in only two documents), but all appearing in one document. This tests the retrieval correctness of the program and its handling of stop words.
- **expected results**:

```
Warning: "and" is in the stop list and will be ignored.
Your query has multiple valid words, so we will search for them
in the inverted index.
The words were found in files below:
File name                Tf-Idf
1henryiv.2.1.txt         0.014780
```

- **test result**:

output	ticks(ticks/search)	time(s/search)
correct	79.439	8e-5

The time is averaged from 1000 runs. Since the test purpose does not include the threshold, the threshold is set to 1 here.

### 3.2.3 input2

- **content** : blank input
- **purpose**: Testing for blank input
- **expected results**:

```
Ops, your query is in StopList or empty, so there are no documents
retrieved.
```

- **Test result**

output	ticks(ticks/search)	time(s/search)
correct	15.246	2e-5

The time is averaged from 1000 runs. Since irrelevant words will not enter the search phase, the threshold is irrelevant to this test.

### 3.2.4 input3

- **content** : three stop words
- **purpose**: Testing for stop words
- **expected results**:

```
Warning: "call" is in the stop list and will be ignored.
Warning: "other" is in the stop list and will be ignored.
Warning: "man" is in the stop list and will be ignored.
Ops, your query is in StopList or empty, so there are no documents
retrieved.
```

- **test result**:

output	ticks(ticks/search)	time(s/search)
correct	51.027	5e-5

The time is averaged from 1000 runs. Since irrelevant words will not enter the search phase, the threshold is irrelevant to this test.

### 3.2.5 input4

- **content** : Complete 1henryiv.1.2
- **purpose**: Test detection of extreme length correct text. Test the performance of searches at different thresholds
- **expected results**(threshold=1.0):

```
Warning: ...
...(omit numerous warning about stop words)
Your query has multiple valid words, so we will search for them
in the inverted index.
The words were found in files below:
```

File name	Tf-Idf
1henryiv.1.2.txt	5.785513

- **test result:**

output	ticks(ticks/search)	time(s/search)
correct	129114.1	0.13

The time is averaged from 10 runs, and threshold=1

threshold	rank	tf-idf	search time
0.01	1/1	0.024940	0.02

No more thresholds were tested here because the situation was too extreme and the correct documents were already filtered out at a threshold of 0.01. Increasing the threshold will only increase the running time, so it is not tested here..

### 3.2.6 input5

- **content** : All sects, all ages smack of this vice; and he To die for't!
- **purpose**: This sentence is choose from measure.2.2. Select this to test the effect of the threshold on search results.
- **expected results**(threshold=1.0):

```
Warning: "all" is in the stop list and will be ignored.
Warning: "all" is in the stop list and will be ignored.
Warning: "of" is in the stop list and will be ignored.
Warning: "this" is in the stop list and will be ignored.
Warning: "and" is in the stop list and will be ignored.
Warning: "he" is in the stop list and will be ignored.
Warning: "to" is in the stop list and will be ignored.
Warning: "for" is in the stop list and will be ignored.
Warning: "t" is in the stop list and will be ignored.
Your query has multiple valid words, so we will search for them
in the inverted index.
The words were found in files below:
File name      Tf-Idf
measure.2.2.txt 0.017464    0.015054
```

- **test result:**

output	ticks(ticks/search)	time(s/search)
correct	992.903	9.9e-4

The time is averaged from 1000 runs, and `threshold=1`.

threshold	rank	tf-idf	search time
0.2	3/7	0.003003	2e-4
0.4	1/1	0.005808	2e-4
0.6	1/1	0.011469	3e-4
0.8	1/1	0.012638	5e-4
1.0	1/1	0.015054	9.9e-4

From the test results, we can see that the works can be correctly screened out with a threshold of about 0.4.

I also tested some sentences with similar results, which I will not list here.

## Chapter 4: Analysis and Comments

### Note

- We only care the memory space of **major** data structures
- It's undeniable that every word has different length, but we set the smallest unit to "word", not "character" for our convenience of analysis

### 4.1 Space Complexity

**Conclusion:**  $O(W + D + H + I + M + K)$

- $W$ : Word count all articles in *The Works*
- $D$ : The number of documents(files, or articles in *The Works*)
- $H$ : The size of the hash table which contains all stopwords
- $I$ : The size of the inverted index(notice: we have removed the duplicated words, so every two nodes contain distinct words)
- $M$ : The number of valid words
- $K$ : The number of found documents for a single word

**Analysis:**



We should analyze the space complexity step by step:

- **Word count:** We use C++ STL containers(`pair`, `map` and `set`) to store the essential information about words and files. Specifically, `wordList` records the word count in all files for every word; `wordNumOfDoc` involves the word count for every file; `wordDocs` contains the number of articles where words appear for every word. Consequently, the memory space in this step depends on both word count in all files ( $W$ ) and the number of files( $D$ ).
- **Stop words:** Actually, in the word count program, we also extracted the stop words from the variables mentioned above, then these words will be stored in a hash table for fast finding when building the inverted index. Therefore, the size of hash table( $H$ ) represents the space that stopwords use.
- **Inverted index:** The bulk of inverted index is stored in a B+ tree. It's universally acknowledged that the space complexity of B+ tree is  $O(N)$ , when  $N$  is the number of data. However, our program has some uncertain factors, for instance, we can't control the size of the position list for each word, because the specific frequency of words are different. But what we can assure is that the total size of all position list is proportional to unduplicated word count, which is less than  $W$ . As a consequence, we only care  $I = N \times \text{ORDER}$ , where `ORDER` means the order of the B+ tree, and we allocate `ORDER + 1` bytes for data and children in each node.
- **Query:** There are four data structures used in this part: `queryWord` stores up to  $m$  words, yielding  $O(M)$ ; `posVec` holds document positions, with a maximum size of  $k$ , contributing  $O(K)$ ; `freqMap` may contain up to  $d$  documents, leading to  $O(D)$ .; Temporary hash table `currentDocIdMap` also contributes  $O(D)$ . Consequently, the overall space complexity is  $O(M + K + D)$ .

Although our main program calls other functions which are not listed above, but their space complexity overlaps with the above operations or they are too trivial to be considered, so we don't care much about these function.

To sum up, the space complexity of our program is  $O(W + D + H + I + M + K)$ .

## 4.2 Time Complexity

**Conclusion:**  $O(W + I \log I + n + m + m \log m + X)$

- $W$ : Word count all articles in *The Works*
- $I$ : The size of the inverted index
- $n$ : The query string of length
- $m$ : The number of valid words
- $X$ : Vary in different situation
  - **single** valid word:  $k \log k$ , where  $k$  is the number of documents
  - **multiple** valid words:  $m \times (p \log p)$ , where  $p$  is the total document results in worst-case scenario

### Analysis:

- **Word count and stop words:** Apparently, we count and handle every word in all files to implement the functions of word count and stop words detection. Therefore, the time complexity of this part is proportional to  $W$ , which is mentioned above.
- **Inverted Index:** The most frequent operations we have run in the inverted index are insertion and finding, so we consider these operations mainly. It's proved that the efficiency is  $O(\log N)$  for both insertion and finding, and for every node we should execute these operations at least once. Consequently, the whole time complexity of building inverted index is  $O(I \log I)$ . Additionally, in our speed test above, we have drawn the curve chart of it, which can be fitted with  $a + b \cdot N \log N$  function, which proves the correctness of our analysis further.
- **Query:** We will analyze this part step by step:
  - **character iteration:** The function iterates through each character in the query string of length  $n$ , resulting in  $O(n)$ .
  - **word processing:** Each word is checked against the stop words list and the inverted index. If  $m$  is the number of valid words, this step is  $O(m)$  since both checks have an average time complexity of  $O(1)$ .
  - **sorting:** The `queryWord` vector containing valid words is sorted, taking  $O(m \log m)$ .
  - **document Search:** If there is one valid word, it retrieves the positions from the B+ tree (let's assume  $k$  documents) and sorts them:  $O(k \log k)$ . If there are multiple valid words, for a worst-case scenario of  $p$  total document results, it processes each word, resulting in  $O(m \times (p \log p))$ .

As a consequence:

- For a **single** valid word:  $O(n + m + m \log m + k \log k)$
- For **multiple** valid words:  $O(n + m + m \log m + m \times (p \log p))$

For the same reason above, we also ignore the time complexity of some functions which have slight impact on the whole time complexity of our main program.

In a nutshell, the time complexity of our program is  $O(W + I \log I + n + m + m \log m + X)$ .

### 4.3 Further Improvement

1. **Algorithm refinement:** So far, we have learned few of the efficient algorithms and data structures, which means that our implementation of the mini search engine might not be the best practice. However, it's possible for us to devised more ingenious and efficient procedure to cope with this problem after we systematically learned more excellent algorithms and data structures.
2. **Testing construction:** Although we come up with some testing cases, probably some crucial tests are still lost, and potential bugs may exists in our programs owing to our incomplete consideration. From our standpoint, it's difficult to find all typical cases for a program, but we're fully convinced that by delicate techniques and tricks for testing results, we can come up with tests as complete as possible.
3. **Complexity analysis:** As you can see, it's awkward to analyze the complexity of some programs such as the space complexity of position list in nodes of inverted index. As a consequence, our analysis on complexity isn't very accurate. We will study the systematic method of analyzing the complexity and improve the precision of our analysis in the foreseeable future.

## Appendix: Source code

### 5.1 File Structure

```

.
├── README.md
└── code

```

```
├── Makefile
├── README.pdf
├── build
├── scripts
│   ├── getStopWord
│   ├── getStopWord.cpp
│   ├── html2txt.py
│   ├── iist_diagram.py
│   ├── invIndexFunc.cpp
│   ├── invIndexHeader.h
│   ├── invIndexSearch.cpp
│   ├── invIndexTest.cpp
│   ├── search_main.cpp
│   ├── search_test.cpp
│   └── wordStem
├── source
│   ├── file_word_count.txt
│   ├── search_test
│   ├── shakespeare-master
│   ├── shakespeare_works
│   ├── stop_words.txt
│   ├── inverted_index_tests
│   ├── txt_title.txt
│   ├── word_count.txt
│   └── word_docs.txt
└── documents
    └── report-p1.pdf
```

## 5.2 getStopWord.cpp

```
/******
 * This program is used to get stop words from the Shakespeare
works.
 * At the same time, it also counts the number of occurrences of
 * each word in each document and then gets the stop words.
 * The output files are word_count.txt, stop_words.txt, and
word_docs.txt.
 * All of them are stored in the code/source directory.
```

```

*****/
#include <iostream>
#include <fstream>          // for file input/output
#include <string>
#include <map>
#include <vector>
#include <set>
#include <algorithm>        // for sort()
#include <cctype>
#include "wordStem/english_stem.h"

using namespace std;

typedef pair<wstring, set<string>> Pair;
map<wstring, int> wordList;          // word → count(The total
number of times a word appears in the corpus)
map<string, int> wordNumOfDoc;       // file → word_count(Total
word count of the file)
stemming::english_stem<> StemEnglish;
map<wstring, set<string>> wordDocs; // word → count(The
number of articles containing the word)

int main()
{
    string file;          // file name
    ifstream infile;      // input file
    ofstream outfile;     // output file

    infile.open("source/txt_title.txt"); // open the file
    containing the file names
    outfile.open("source/file_word_count.txt", ios::out);
    while(infile >> file)
    {
        string line;
        ifstream in;

        // Read in the file and stem each word
        in.open("source/shakespeare_works/"+file+".txt", ios::in);
        while(getline(in, line))

```

```

    {
        wstring word = L"";
        for(char &c : line)
        {
            // if the character is alpha or number, add it to
            the word. It solve the problem of the Punctuation and Whitespace
            if(isalnum(c))
                word += tolower(c);
            else if(word.length() > 0)
            {
                StemEnglish(word);                //stem the
word

                wordList[word]++;
                wordNumOfDoc[file]++;
                wordDocs[word].insert(file);        //get the
number of files by the size of the file name vector
                word = L"";
            }
        }
        outfile << file << " " << wordNumOfDoc[file] << endl;
        in.close();
    }
    infile.close();

    // Sort the words by their frequency
    // PS : Map does not have a built-in sorting algorithm, so I
    need to convert the map into a vector and then use sort().
    vector<Pair> vec(wordDocs.begin(), wordDocs.end());
    sort(vec.begin(), vec.end(), [](const Pair& a, const Pair& b)
    { return a.second.size() > b.second.size(); });

    wofstream out,out2,out3;    // output files
    out.open("source/word_count.txt", ios::out);
    out2.open("source/stop_words.txt", ios::out);
    out3.open("source/word_docs.txt", ios::out);
    for(Pair &word : vec)
    {
        out3 << word.first << " " << word.second.size() << endl;

```

```

        // To be honest, the threshold of stop words has no
        scientific basis.
        if(word.second.size() ≥ 334)
            out2 << word.first << endl; // stop words
            out << word.first << " " << wordList[word.first] <<
endl;    // word count
        }
        out.close();
        out2.close();
        out3.close();
    }

    // To compile the program, run the following command in the
    terminal:
    // Depends on where you are, maybe you need to change the path
    accordingly.
    // (I assume you are in the main dictionary of the project)
    // g++ -o code/scripts/getstopwords/getStopWord code/scripts/
    getstopwords/getStopWord.cpp -Werror -Wall -Wextra

```

### 5.3 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>
#include <vector>

#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 length of
string for one read
#define STOPWORDSUM 300 // The maximum number of
stop words
#define STOPWORDPATH "source/stop_words.txt" // The path
of the file storing stop words
#define DEFAULTFILEPOS "source/tests" // The default
position of the file(for test mode)
#define SHAKESPEAREDIR "source/shakespeare_works" // The path
of Shakespeare's Works
#define IISPEEDTESTDIR "source/tests/invIndex-
speedTest/880000" // Speed test files for inverted index
#define FILEWORDCOUNTPATH "source/
file_word_count.txt" // The path of word count
file
#define WORDDOCSPATH "source/
word_docs.txt" // The path of word-
>doc file
#define ITERATIONS 5 //
Iteration time used in speed test for inverted index

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

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

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

void loadWordIdf(std::string filePath);
void loadFileWordsNum(std::string filePath);
void loadStopWords(std::string filePath);
void search(std::string query, BplusTree T, int pageSize, double
threshold);
std::vector<std::pair<int,double>> FindBP2(string term, int
docCnt, BplusTree T) ;
std::vector<std::pair<int,double>> isSameTerm2(string term, int
docCnt, NodeBP nodebp) ;
```

```
std::vector<std::pair<int,double>> RetrievePL2(PosList L) ;

#endif
```

## 5.4 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
    strcpy(dir, SHAKESPEAREDIR);
    isTest = false;
    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); // we should input the filename
    } else { // Otherwise(in the main program or other
default situations),
        char choice[MAXWORDLEN];
        strcpy(dir, SHAKESPEAREDIR); // Default path
        printf("Now building an inverted Index:\n");
        printf("Do you want to search something in the default
path(%s)?\n", dir);
    }
}

```

```

        printf("Please input yes or no: ");
        scanf("%s", choice);          // Input your choice
        switch (choice[0]) {
            case 'y': case 'Y':        // Yes
                break;
            case 'n': case 'N':        // No
                printf("\nPlease input the directory of the
documents:\nPath: ");
                scanf("%s", dir);      // We should input the name
of the directory
                break;
            default:                    // Choice error
                printf("Choice Error!\n");
                exit(1);
                break;
        }
    }
}

// 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

                    // Word stemming
                    term = WordStem(term);

                    // 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
                    }

                    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
    term = WordStem(term);

    // 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
    }

    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 appropriate 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 splitting 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[MAXREADSTRLLEN]; // 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
// stopwords: 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; // Appropriate position
    int collisionNum = 0; // collision number,
    for quadratic probe
        pos = HashFunc(stopword, H->size); // Use hashing function
        first

        // 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;
            if (justSearch && H->data[pos]->info == Empty) {
                return -1;
            }
        }
        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 Stemming wrapper
string 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);

    return term;
}

// 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)
    int iterations;

```

```

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

```

## 5.5 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
    char dir[MAXREADSTRLEN];
    strcpy(dir, IISPEEDTESTDIR);
    start = clock();
    for (i = 0; i < ITERATIONS; i++) {
        InvIndex = CreateBP();
        InvIndex = fileTraversaler(InvIndex, dir, NULL, 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 = WordStem(word);

        // 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;
}

```

## 5.6 invIndexSearch.cpp

```

/*****
 * The following code is the implementation of the inverted index
 search algorithm.
 * The algorithm takes a query from the user and searches for the
 documents that contain all the valid query words.
 * The algorithm uses the B+ tree to search for the documents and
 the inverted index to get the word frequencies.
 * The algorithm also uses the stop list to ignore the common
 words and the stemming algorithm to reduce the words to their
 root form.
 *****/

#include <iostream>                // for input/output
#include <fstream>                 // for file input/output
#include <string>
#include <string.h>
#include <map>                     // for map data structure
#include <unordered_map>          // for unordered_map data
structure (hash)

```

```

#include <algorithm>                // for sorting
#include <cctype>
#include <cmath>                    // for log()
#include "wordStem/english_stem.h" // for stemming
#include "invIndexHeader.h"

#define DOCTOTALNUM 992            // total number of documents

//hash table: {word:wordIdf}
std::unordered_map<std::wstring, double> wordIdf;
//hash table: {docId:fileWordCount}
std::unordered_map<std::wstring, int> fileWordsNum;
//hash table: {word:bool} PS: Bool is just a placeholder and has
no actual function.
std::unordered_map<std::wstring, bool> stopWords;
stemming::english_stem<> StemEnglish;

/*****
 * This function is the main function for searching.
 * In this function, we analyzes the query
 * and search for the documents that contain all the valid query
 words.
 * Input: T: Inverted index (B+ tree)
 *      pageSize: number of documents to be displayed per query
 *      threshold: the proportion of search terms to the total
 number of terms
 *****/
void search(std::string query, BplusTree T, int pageSize, double
threshold)
{
    /*load the query and analyze it*/
    //vector: Elements are {word:wordIdf}
    std::vector<std::pair<std::wstring, double>> queryWord;
    std::wstring word=L"";
    //Iterate through each character in the string and extract
the words
    for(char &c : query)
    {
        // if the character is alpha or number, add it to the

```



```

word. It solve the problem of the Punctuation and Whitespace
    if(isalnum(c))
        word += tolower(c);
    else if (word.length() > 0)
    {
        std::wstring tmp = word;    // save original word
        StemEnglish(word);
        // check if the word is in the stop list
        auto it = stopWords.find(word);
        if(it != stopWords.end())
            // if the word is in the stop list, print a warning
message and ignore it.
            std::wcout << L"\033[1;31mWarning: \033[0m\"" << tmp
<< L"\n is in the stop list and will be ignored." << std::endl;
        else
        {
            // if the word is not in the stop list, check if
it is in the inverted index.
            auto it2 = wordIdf.find(word);
            if(it2 == wordIdf.end())
                // if the word is not in the inverted index,
print a warning message and ignore it.
                std::wcout << L"\033[1;31mWarning: \033[0m\""
<< tmp << L"\n is not in the inverted index and will be ignored."
<< std::endl;
            else
                // if the word is in the inverted index, add
it to the queryWord vector.
                queryWord.push_back({word,wordIdf[word]});
        }
        word = L"";    // reset the word
    }
}
// Sort the queryWord vector by wordIdf in descending order
    sort(queryWord.begin(),queryWord.end(),
[] (const std::pair<std::wstring,double>& a, const
std::pair<std::wstring,double>& b){return a.second > b.second;});

/* Search for documents that contain all the query words */

```

```

    int cntForDoc = 0; //
counter for pageSize
    int cntForWord = (int)(queryWord.size() * threshold); //
counter for threshold
    //control the size of cntForWord, to avoid bugs
    if (cntForWord == 0)
        cntForWord = 1;
    else if (cntForWord > queryWord.size())
        cntForWord = queryWord.size();

    char * wordForSearch; //
char array for search
    //if the query is empty or only has stop words, return
    if(queryWord.size() == 0)
    {
        std::cout << "\033[1;31mOps, your query is in StopList or
empty, so there are no documents retrieved. \033[0m" << std::endl;
        return;
    }
    //if the query has only one valid word, search for it in the
inverted index
    //here, we only use "Tf" to sort the documents in rusult
    else if(queryWord.size() == 1)
    {
        //vector: Elements are {docId:tf}
        std::vector<std::pair<int,double>> posVec;
        std::cout << "Your query has only one valid word, so we
will search for it in the inverted index." << std::endl;
        std::cout << "The word was found in files below:"<<
std::endl;
        printf("\033[1,32mFile name          \tTf\033[0m\n");
        //don't need to traverse here, but I wrote the case of
multiple words first, so I reused it directly
        for(auto &p : queryWord)
        {
            wordForSearch = wstringToChararr(p.first);
            //find all the positions of the word in the B+ tree
            posVec = FindBP2(wordForSearch, -1, T);
            //sort the positions by tf in descending order

```

```

        sort(posVec.begin(), posVec.end(), [](const
std::pair<int, double>& a, const std::pair<int, double>& b){return
a.second > b.second;});
        //print the file name and tf of the word
        for(auto &pos : posVec)
        {
            if(cntForDoc == pageSize)
                break;
            printf("\033[34m%-25s\t%lf\033[0m\n", docNames[pos.first],
cntForDoc++);
        }
    }
    // if the query has multiple valid words, search for them in
the inverted index
    // Here, we use "Tf-Idf" to sort the documents in result
    else
    {
        std::cout << "Your query has multiple valid words, so we
will search for them in the inverted index." << std::endl;
        std::cout << "The words were found in files below:"<<
std::endl;
        //hash table: {docId : sum of tf-idf}
        std::unordered_map<int, double> freqMap;
        //traverse the queryWord vector and search for each word
in the inverted index
        bool flag=false;
        for(auto &p : queryWord)
        {
            if(cntForWord == 0)
                break;

            wordForSearch = wstringToChararr(p.first);
            auto currentPosVec = FindBP2(wordForSearch, -1, T);

            //Transfer the values in the vector to a temporary
hash table, mainly for efficiency reasons.
            //The temporary hash table only contains the pair of
docId and idf.

```

```

        std::unordered_map<int, double> currentDocIdMap;
        if(flag == false)
        {
            for(auto &pos : currentPosVec)
                freqMap[pos.first] += pos.second * p.second; //
tf-idf = tf * idf
            flag = true;
        }
        else
        {
            for(auto &pos : currentPosVec)
                currentDocIdMap[pos.first] = pos.second *
p.second;
            if(freqMap.size()>0)
            {
                //remove "it++" in "for(;;)" to remove
duplicates
                for (auto it = freqMap.begin(); it !=
freqMap.end(); )
                {
                    if (currentDocIdMap.find(it->first) ==
currentDocIdMap.end())
                        it = freqMap.erase(it); //safe deletion
                    else
                    {
                        it->second += currentDocIdMap[it-
>first];
                        ++it;
                    }
                }
            }
            else
                break;
        }
        cntForWord--;
    }
    //if there is no document that contains all the query
words, return
    if(freqMap.size() == 0)

```

```

        {
            std::cout << "Ops, your query is not in the inverted
index, so there are no documents retrieved. " << std::endl;
            std::cout << "Please try again with different query."
<< std::endl;
            return;
        }
        //if there is at least one document that contains all the
query words, sort them by tf-idf in descending order
        else
        {
            printf("\033[1;32mFile name                \tTf-
Idf\033[0m\n");
            //convert the unordered_map to a vector and sort it
by tf-idf in descending order
            std::vector<std::pair<int, double>>
sortedFreqVec(freqMap.begin(), freqMap.end());
            std::sort(sortedFreqVec.begin(), sortedFreqVec.end(),
[] (const std::pair<int, double>& a, const std::pair<int, double>&
b) {return a.second > b.second; });
            //print the file name and tf-idf of the word
            for(auto &p : sortedFreqVec)
            {
                if(cntForDoc == pageSize)
                    break;
                printf("\033[34m%-25s\t%lf\033[0m\n", docNames[p.first], p.second);
                cntForDoc++;
            }
        }
    }
}

// similar to FindBP() in invIndexFunc.cpp, but desgined just for
searching
// return <word, tf>
std::vector<std::pair<int, double>> FindBP2(string term, int
docCnt, BplusTree T)
{

```

```

    int i;

    // if the tree is empty, return an empty vector
    if (!T)
        return std::vector<std::pair<int,double>>();
    else if (!T->childrenSize)
        return isSameTerm2(term, docCnt, T);

    int pos = -1;

    // find the position of the term in the current node
    for (i = 0; i < T->size; i++) {
        if (strcmp(term, T->data[i]->term) < 0) {
            pos = i;
            break;
        }
    }
    if (pos == -1) {
        pos = i;
    }

    return FindBP2(term, docCnt, T->children[pos]);
}

//similar to isSameTerm() in invIndexFunc.cpp, but designed just
for searching
//return <docId, tf>
std::vector<std::pair<int,double>> isSameTerm2(string term, int
docCnt, NodeBP nodebp)
{
    int i;
    // vector: Elements are {docId:idf}
    std::vector<std::pair<int,double>> posVec;

    if (nodebp->size)
    {
        for (i = 0; i < nodebp->size; i++) {
            if (!strcmp(term, nodebp->data[i]->term))
            {

```

```

        PosList poslist = nodebp→data[i]→poslist;
        posVec = RetrievePL2(poslist);
        break;
    }
}
}
return posVec;
}

//similar to RetrievePL() in invIndexFunc.cpp, but designed just
for searching
//return <docId, tf>
std::vector<std::pair<int,double>> RetrievePL2(PosList L)
{
    // if the position-data queue is empty, return an empty vector
    if (!L→size) {
        printf("Empty position-data queue!\n");
        exit(1);
    }

    double tf;
    std::vector<std::pair<int,double>> posVec;
    PosData cur = L→front→next;

    while (cur ≠ NULL) {
        std::wstring docName = chararrToWstring(docNames[cur-
>pos]);
        // calculate tf : tf = word freq/total words in file
        tf = (double)cur→time / (double)(fileWordsNum[docName]);
        posVec.push_back({cur→pos, tf});
        cur = cur→next;
    }

    return posVec;
}

/*The functions below load the wordIdf, fileWordsNum, and
stopWords from files*/

```

```
void loadWordIdf(std::string filePath)
{
    std::wifstream infile;
    std::wstring word;
    int freq;
    double idf;
    infile.open(filePath, std::ios::in);
    //if the file is empty, return
    if(!infile.is_open())
    {
        std::cout << "Error: unable to open file " << filePath <<
std::endl;
        return;
    }
    //read the word and its frequency from the file and calculate
its idf
    while(infile >> word >> freq)
    {
        //calculate idf :  $idf = \log(\text{total number of documents} / (1 + \text{number of documents contain the word}))$ 
        idf = log((double)DOCTOTALNUM/(double)(1+freq));
        wordIdf[word] = idf;
    }
    infile.close();
    std::cout << "Word Idf loaded successfully." << std::endl;
}

void loadFileWordsNum(std::string filePath)
{
    std::wifstream infile;
    std::wstring filename;
    int num;
    infile.open(filePath, std::ios::in);
    //if the file is empty, return
    if(!infile.is_open())
    {
        std::cout << "Error: unable to open file " << filePath <<
std::endl;
        return;
    }
}
```



```
    }
    //read the file name and its word count from the infile
    while(infile >> filename >> num)
    {
        filename += L".txt";
        fileWordsNum[filename] = num;
    }
    infile.close();
    std::cout << "The word count of every file loaded successfully."
    << std::endl;
}

void loadStopWords(std::string filePath)
{
    std::wifstream infile;
    std::wstring word;
    infile.open(filePath, std::ios::in);
    //if the file is empty, return
    if(!infile.is_open())
    {
        std::cout << "Error: unable to open file " << filePath <<
        std::endl;
        return;
    }
    while(infile >> word)
        stopWords[word] = true;
    std::cout << "Stop words loaded successfully." << std::endl;
}
```

## 5.7 search\_test.cpp

```
#include <iostream>
#include <fstream>           // for file input/output
#include <string>
#include <time.h>             // for clock()
#include "invIndexHeader.h"

const std::string root_path = "../source/search_test/input";
```

```

int main()
{
    clock_t start, end;

    //load inverted index and other necessary resources
    std::cout << "Please wait for resources to load and inverted
index to be established..."<<std::endl;
    loadWordIdf(WORDDOCSPATH);           //load word-idf
dictionary
    loadFileWordsNum(FILEWORDCOUNTPATH); //load file-word count
dictionary
    loadStopWords(STOPWORDPATH);        //load stop words
list
    BplusTree InvIndex = CreateBP();     //create B+ tree for
inverted index
    InvIndex = fileTraversaler(InvIndex, (char *)SHAKESPEAREDIR,
(char *)"", false, true);
    std::cout << "Inverted Index has been established
successfully!"<<std::endl;

                                std::cout <<
"-----" <<
std::endl;

    std::cout << "Welcome to ShakespeareFinder's test program!"
<< std::endl;
    std::cout << "The program tests the performance of the search
engine in different situations" << std::endl;

    //test loop begin
testbegin:
    std::cout << "Please enter the serial number of the input you
want to test: " << std::endl;
    int serial_num;           //serial number of input file
    std::cin >> serial_num;
        std::string input_file_path = root_path +
std::to_string(serial_num);    //input file path
    std::ifstream input_file;  //input
file object
    input_file.open(input_file_path, std::ios::in); //open

```

```
input file
// if input file is not found, return
if (!input_file.is_open())
    std::cout << "Invalid input serial number!" << std::endl;
// if input file is found, continue
else
{
    //read input parameters
    int pageSize, k;
    double threshold;
    std::cout << "Input file opened successfully!" << std::endl;
    std::cout << "Please enter the pageSize, threshold and
the times of cycles : " << std::endl;
    std::cin >> pageSize >> threshold >> k;
    std::string query;
    std::getline(input_file, query);
    query += " ";

    //test performance
    start = clock();
    for(int i=0; i<k; i++)
    {
        std::cout << "Cycle " << i+1 << " starts..." <<
std::endl; //print cycle number
        search(query, InvIndex , pageSize, threshold);
    }
    end = clock();

    PrintTime(start,end);
}

//ask user if they want to test another input
std::cout << "Do you want to test another input? (y/n)" <<
std::endl;
char choice;
std::cin >> choice;
if(choice == 'y')
{
    getchar();
}
```

```

        goto testbegin;           //jump to testbegin
    }
    else
        return 0;

}

```

## 5.8 search\_main.cpp

```

/*****
 * This is the main function of the search engine.
 * It first loads the inverted index and the necessary resources.
 * Then it enters a loop to repeatedly search for a query until
the user chooses to exit.
*****/
#include <iostream>
#include "invIndexHeader.h"

int main()
{
    //load necessary resources and inverted index
    std::cout << "Please wait for resources to load and inverted
index to be established..."<<std::endl;
    loadWordIdf(WORDDOCSPATH);           //load word-idf
dictionary
    loadFileWordsNum(FILEWORDCOUNTPATH); //load file-word count
dictionary
    loadStopWords(STOPWORDPATH);         //load stop words
list
    BplusTree InvIndex = CreateBP();      //create B+ tree for
inverted index
    InvIndex = InvertedIndex(); // fileTraversaler(InvIndex, (char
*)SHAKESPEAREDIR, (char *)"", false, true);
    std::cout << "Inverted Index has been established
successfully!"<<std::endl;

                                std::cout <<
"-----" <<
std::endl;

```

```

        //set values for pageSize and threshold from user input
        std::cout << "\033[1;32mWelcome to
        ***ShakespeareFinder***\033[0m" << std::endl;
        std::cout << "\033[34mPlease set the maximum number of records
to be displayed per query\033[0m" << std::endl;
        int pageSize = 10; //default value
        std::cout << "Size(int): ";
        std::cin >> pageSize;
        std::cout << "\033[34mPlease set the threshold (0-1) of the
query:\033[0m" << std::endl;
        std::cout << "Threshold(double): ";
        double threshold = 0.8; //default value
        std::cin >> threshold;
        getchar(); //clear the buffer to avoid input
error

        //start the search loop
        Searchbegin:
        std::cout << "Please enter your query:" << std::endl;
        std::string query;
        /* Read query from user and analyse it*/
        std::getline(std::cin, query);
        query += " "; // add a space at the end of
the query to avoid errors

        search(query, InvIndex, pageSize, threshold);
        //ask user if they want to search again
        std::cout << "Do you want to search again?\033[1;31m(y/
n)\033[0m: ";
        char choice;
        std::cin >> choice;
        if(choice == 'y' || choice == 'Y')
        {
            getchar();
            goto Searchbegin;
        }
        //exit the program
        else
        {

```

```
        std::cout << "\033[1;32mThank you for using  
ShakespeareFinder!" << std::endl << "Have a nice day~\033[0m" <<  
std::endl;  
        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.*