Assignment 1 Inverted Index - Report

9961244 EE14 吳德霖

1 INSTRUCTION

The following instructions indicate how to compile and execute my program.

1.1 TO COMPILE

For Inverted Index part:

1. mkdir classFolder
2. javac -classpath ../hadoop-core-1.2.1.jar -d retFolder/ ./*.java
3. jar -cvf InvIndex.jar -C classFolder .

For Retrieval part:

1. mkdir retFolder
2. javac -classpath ../hadoop-core-1.2.1.jar -d retFolder/ ./Retrieval.java
3. jar -cvf Retrieval.jar -C retFolder .

1.2 TO EXECUTE

For Inverted Index part:

- hadoop jar InvIndex.jar hw1.InvIndex [input] [output]
- In this case we would set the input as /opt/HW1/input1/* and any specified output.

For Retrieval part:

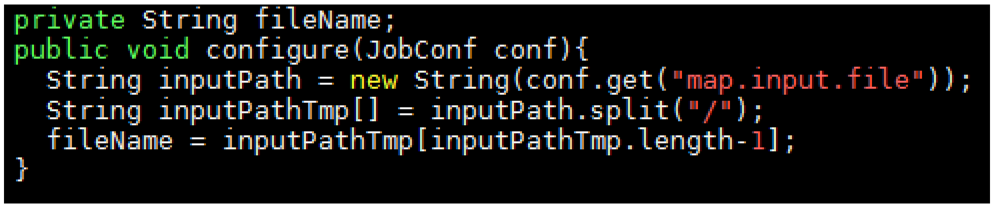
- hadoop jar Retrieval.jar hw1.Retrieval [input Table] [input file]
- In this case we would set the input table as “InvIndexTable” after extracting the output of interest from the output directory.
- Input file in this case would be “/opt/HW1/input1/”, the indicated file path of the 44 input files.

2 Design

2.1 Map Reduce

2.1.1 Mapper

For the mapper part, I used the following code to extract and store the file name of each input file among the 44 total files.

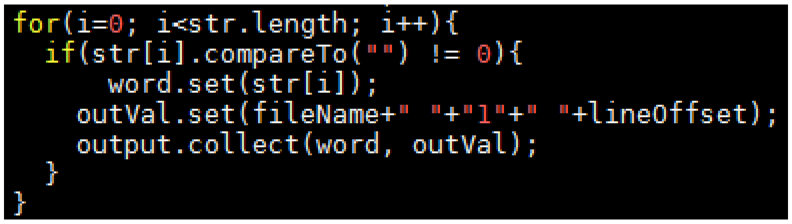


```
private String fileName;
public void configure(JobConf conf){
  String inputPath = new String(conf.get("map.input.file"));
  String inputPathTmp[] = inputPath.split("/");
  fileName = inputPathTmp[inputPathTmp.length-1];
}
```

Since the returned string of the configuration is its absolute position written in a path, I had to split it up with respect to “/” and extract the last string as the file name to be stored.

The default input text form is set to be the TextInputFormat, whose key is the content’s line offset. I first changed the type of the key from object to string for the convenience concern. As for the value which is the content, eliminating those useless notations and extracting the actual word of concern precisely in considered crucial. As a result, here I simply split them up with respect to “\\W”, this implies that I could dispose those useless characters.

By setting the output value to be of Text form, I could simply send the information within a long string that contains all the information needed with the key containing only the word.



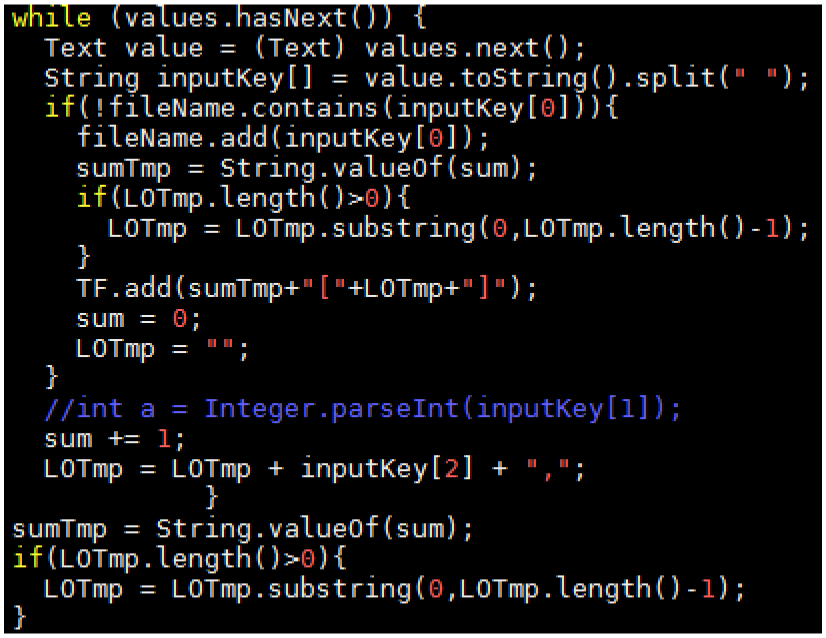
```
for(i=0; i<str.length; i++){
  if(str[i].compareTo("") != 0){
      word.set(str[i]);
    outVal.set(fileName+" "+"1"+" "+lineOffset);
    output.collect(word, outVal);
  }
}
```

As shown above, the outVal is of the text form, it contains the extracted file name, the term frequency which is 1, and the line offset. The outVal and the word would be sent successively in the key value pair format.

2.1.2 Reducer

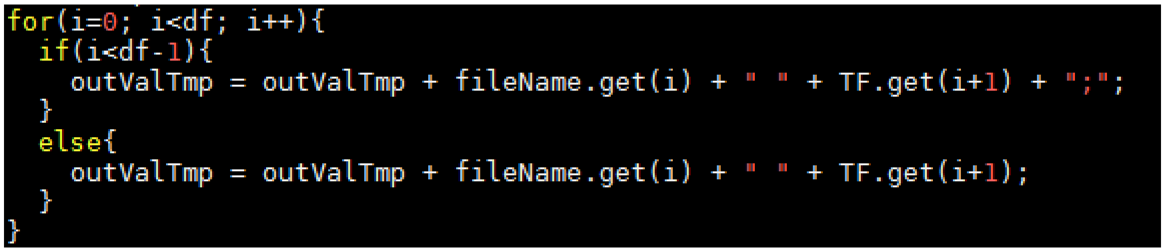
In the reducer, since the received key from the mapper is the keyword, the iterator would be set to those values. I declared two array lists to store the term frequency and the file name.

The code shown below clearly explain the method I adopted. If the array list of the file name doesn’t contain the current input file name, then it would add this file name as a new one. While counting the term frequency, at the same time the full string to be stored to the array list of term frequency would kept collecting the line offsets. Those offsets would be separated by a comma which would be easy to parse in the retrieval part.



```
while (values.hasNext()) {
  Text value = (Text) values.next();
  String inputKey[] = value.toString().split(" ");
  if(!fileName.contains(inputKey[0])){
    fileName.add(inputKey[0]);
    sumTmp = String.valueOf(sum);
    if(LOTmp.length()>0){
      LOTmp = LOTmp.substring(0,LOTmp.length()-1);
    }
    TF.add(sumTmp+"["+LOTmp+"]");
    sum = 0;
    LOTmp = "";
  }
  //int a = Integer.parseInt(inputKey[1]);
  sum += 1;
  LOTmp = LOTmp + inputKey[2] + ",";
          }
sumTmp = String.valueOf(sum);
if(LOTmp.length()>0){
  LOTmp = LOTmp.substring(0,LOTmp.length()-1);
}
```

This process would guarantee that the same set of information with respect to the keyword goes to the same place and could be collected at once of the form of string.



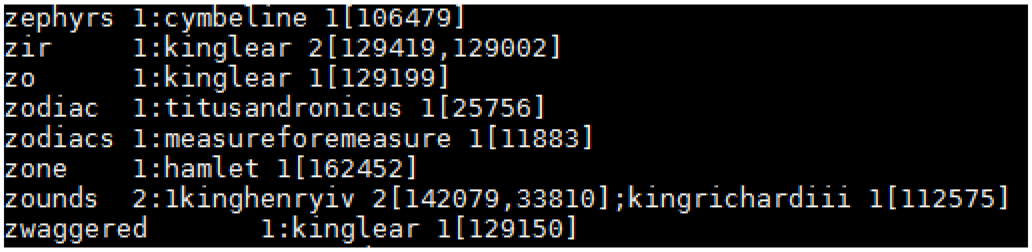
```
for(i=0; i<df; i++){
  if(i<df-1){
    outValTmp = outValTmp + fileName.get(i) + " " + TF.get(i+1) + ";";
  }
  else{
    outValTmp = outValTmp + fileName.get(i) + " " + TF.get(i+1);
  }
}
```

Finally, the output form is again of the form Text, the outValTmp here would contain all required information for retrieval, separating the different source file by a semi colon.

2.1.3 Sample Execution and output

The sample output is as shown below, with the format set to be:

word df:fileName1 tf1[offset11,offset12,…]; fileName2 tf2[offset21,offset22,…]; …

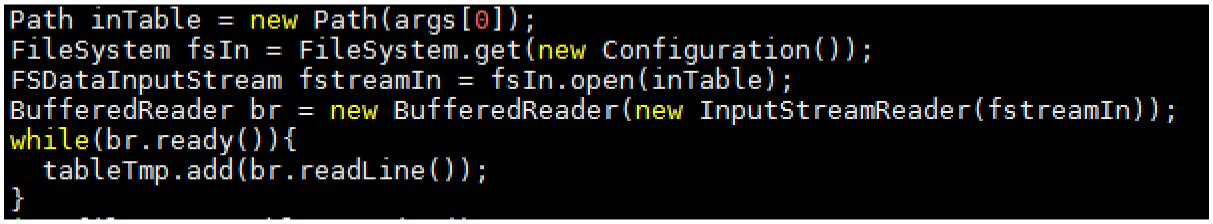


```
zephyrs 1:cymbeline 1[106479]
zir     1:kinglear 2[129419,129002]
zo      1:kinglear 1[129199]
zodiac  1:titusandronicus 1[25756]
zodiacs 1:measureforemeasure 1[11883]
zone    1:hamlet 1[162452]
zounds  2:1kinghenryiv 2[142079,33810];kingrichardiii 1[112575]
zwaggered       1:kinglear 1[129150]
```

2.2 Retrieval

2.2.1 Design

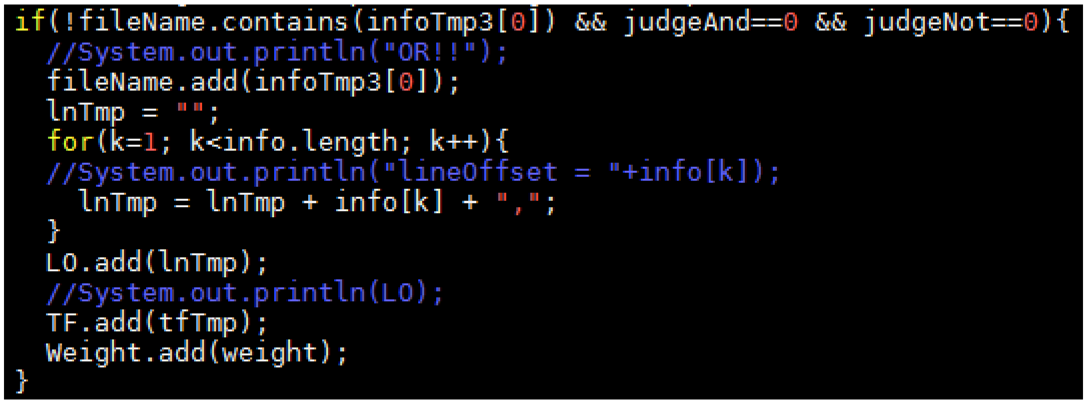
In the retrieval part, I first adopted several array lists to store and the input information and calculated results.



```java
Path inTable = new Path(args[0]);
FileSystem fsIn = FileSystem.get(new Configuration());
FSDataInputStream fstreamIn = fsIn.open(inTable);
BufferedReader br = new BufferedReader(new InputStreamReader(fstreamIn));
while(br.ready()){
  tableTmp.add(br.readLine());
}
```

As shown above, I used the FSDataInputStream's API to collect the input file from the Hadoop file system remotely. And each line would be read and stored using the buffer reader.

The retrieval started with matching the query keywords to the inverted index table, and the console will show the return order of the word if matched term is found, otherwise will display nothing. If the query string contains "&&" and "~", which represents AND and NOT respectively, the order would be set to -1 and -2 respectively for detecting.

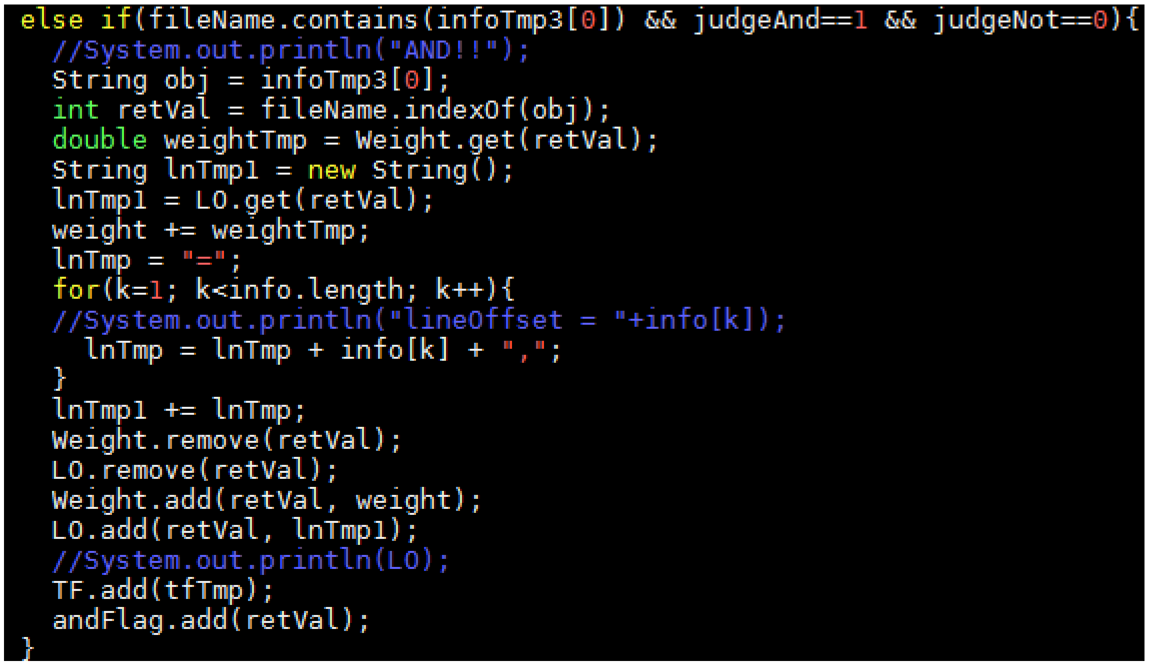


```java
if(!fileName.contains(infoTmp3[0]) && judgeAnd==0 && judgeNot==0){
  //System.out.println("OR!!");
  fileName.add(infoTmp3[0]);
  lnTmp = "";
  for(k=1; k<info.length; k++){
  //System.out.println("lineOffset = "+info[k]);
    lnTmp = lnTmp + info[k] + ",";
  }
  LO.add(lnTmp);
  //System.out.println(LO);
  TF.add(tfTmp);
  Weight.add(weight);
}
```

In the OR case, if the current file name array list doesn't contain the file name to be proceeded, the weight, file name and line offset array lists would add new information to their ends after parsing out useless separation characters added by the reducer. But if the current array list of file name does contain the input file name, those informations would be combined to the already existed ones with respect to the index of such file name, leaving the file name array list unchanged. The weights would be added up for the keywords of interest and ranked regarding the summations of the weights.

For the AND and NOT cases, I define the query string should be interpreted in order, that is, for example A OR B AND C would represent (A OR B) AND C, another example, A NOT B AND C would lead to files containing A AND C but not B.

Understanding the definition would made the algorithm clearer and could be inherit from the OR case. Snap shot of the code is as shown below.

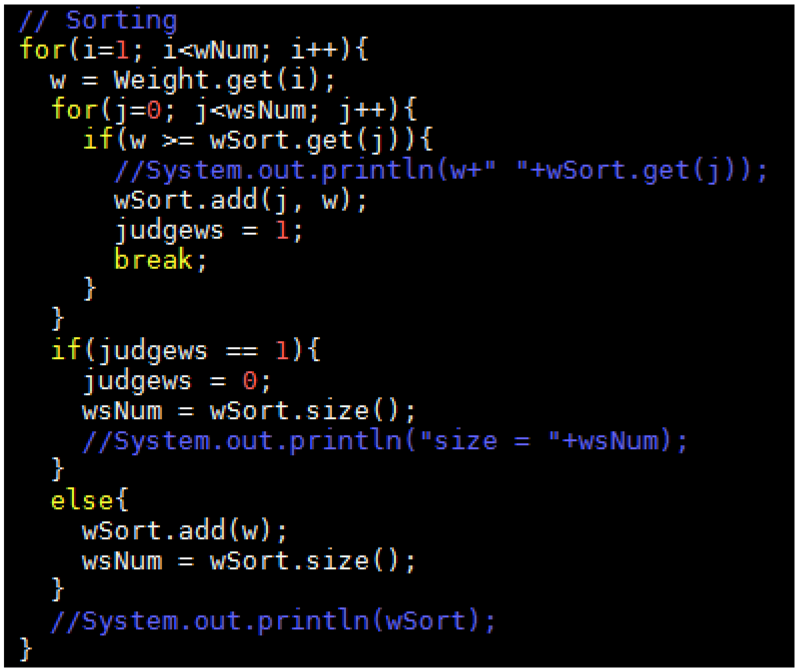


```java
else if(fileName.contains(infoTmp3[0]) && judgeAnd==1 && judgeNot==0){
  //System.out.println("AND!!");
  String obj = infoTmp3[0];
  int retVal = fileName.indexOf(obj);
  double weightTmp = Weight.get(retVal);
  String lnTmp1 = new String();
  lnTmp1 = LO.get(retVal);
  weight += weightTmp;
  lnTmp = "=";
  for(k=1; k<info.length; k++){
  //System.out.println("lineOffset = "+info[k]);
    lnTmp = lnTmp + info[k] + ",";
  }
  lnTmp1 += lnTmp;
  Weight.remove(retVal);
  LO.remove(retVal);
  Weight.add(retVal, weight);
  LO.add(retVal, lnTmp1);
  //System.out.println(LO);
  TF.add(tfTmp);
  andFlag.add(retVal);
}
```

So for the AND case, if the current input file couldn’t be found in the current array list, the input file and all its related information would be ignored.

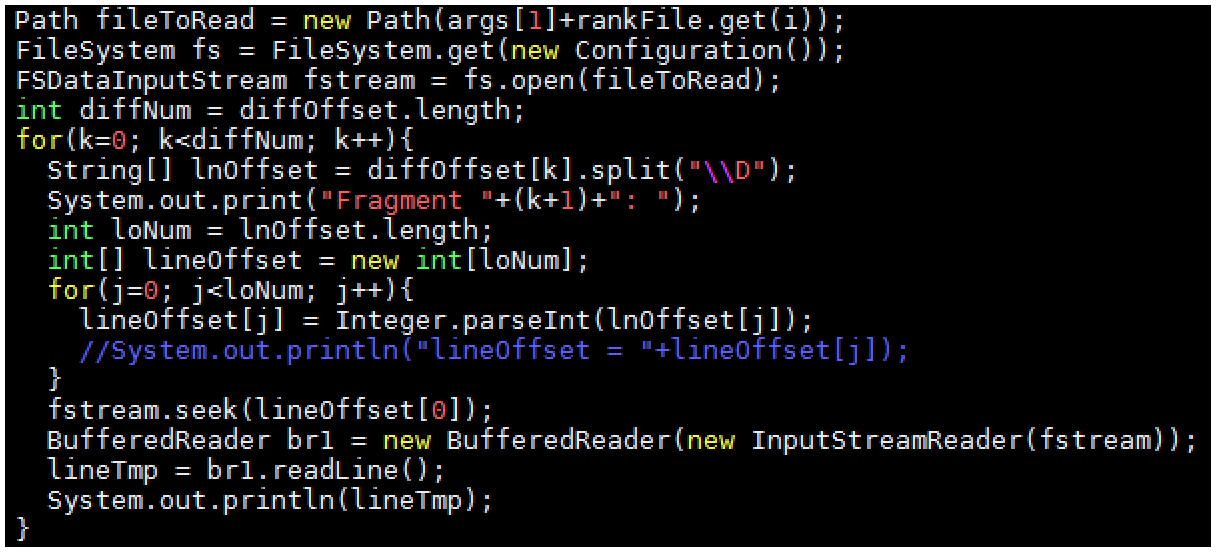
For the NOT case, the same situation simply lead to the elimination of the existing file name within the array list as long with the related information.

After the whole collecting process is done, the final array lists would all be rearranged and stored again, eliminating those file names and the relate information that are illegal to exist due to the user’s query. (This step is especially vital in the AND case.)



```java
// Sorting
for(i=1; i<wNum; i++){
  w = Weight.get(i);
  for(j=0; j<wsNum; j++){
    if(w >= wSort.get(j)){
      //System.out.println(w+" "+wSort.get(j));
      wSort.add(j, w);
      judgews = 1;
      break;
    }
  }
  if(judgews == 1){
    judgews = 0;
    wsNum = wSort.size();
    //System.out.println("size = "+wsNum);
  }
  else{
    wSort.add(w);
    wsNum = wSort.size();
  }
  //System.out.println(wSort);
}
```

Insertion sort is applied to sort the weightings and rank those terms from the first place to the last, if the length of the rank list is larger than 10, it would display only the top 10.

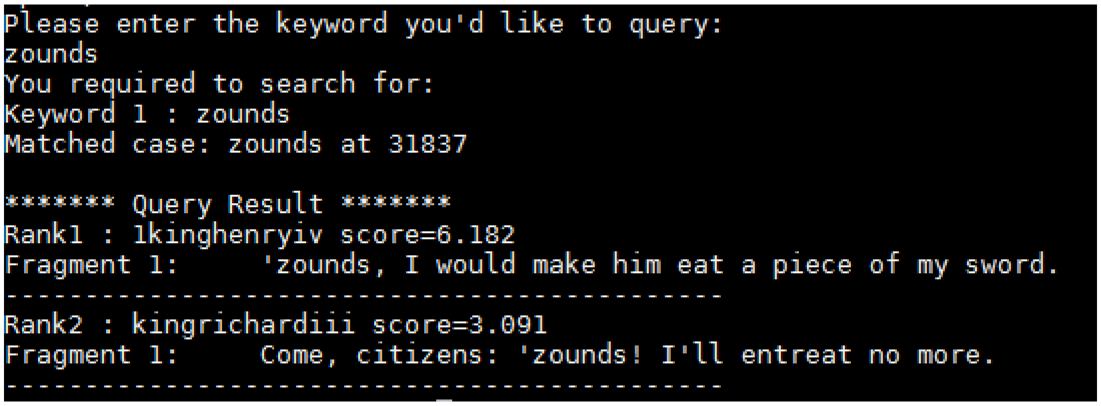

```
Path fileToRead = new Path(args[1]+rankFile.get(i));
FileSystem fs = FileSystem.get(new Configuration());
FSDataInputStream fstream = fs.open(fileToRead);
int diffNum = diffOffset.length;
for(k=0; k<diffNum; k++){
  String[] lnOffset = diffOffset[k].split("\\D");
  System.out.print("Fragment "+(k+1)+": ");
  int loNum = lnOffset.length;
  int[] lineOffset = new int[loNum];
  for(j=0; j<loNum; j++){
    lineOffset[j] = Integer.parseInt(lnOffset[j]);
    //System.out.println("lineOffset = "+lineOffset[j]);
  }
  fstream.seek(lineOffset[0]);
  BufferedReader br1 = new BufferedReader(new InputStreamReader(fstream));
  lineTmp = br1.readLine();
  System.out.println(lineTmp);
}
```

Again I applied the input stream for the fragment demonstration part, the API .seek() could initialize the read in of a file from the indicating line offset, which is exactly the required fragments to be displayed.

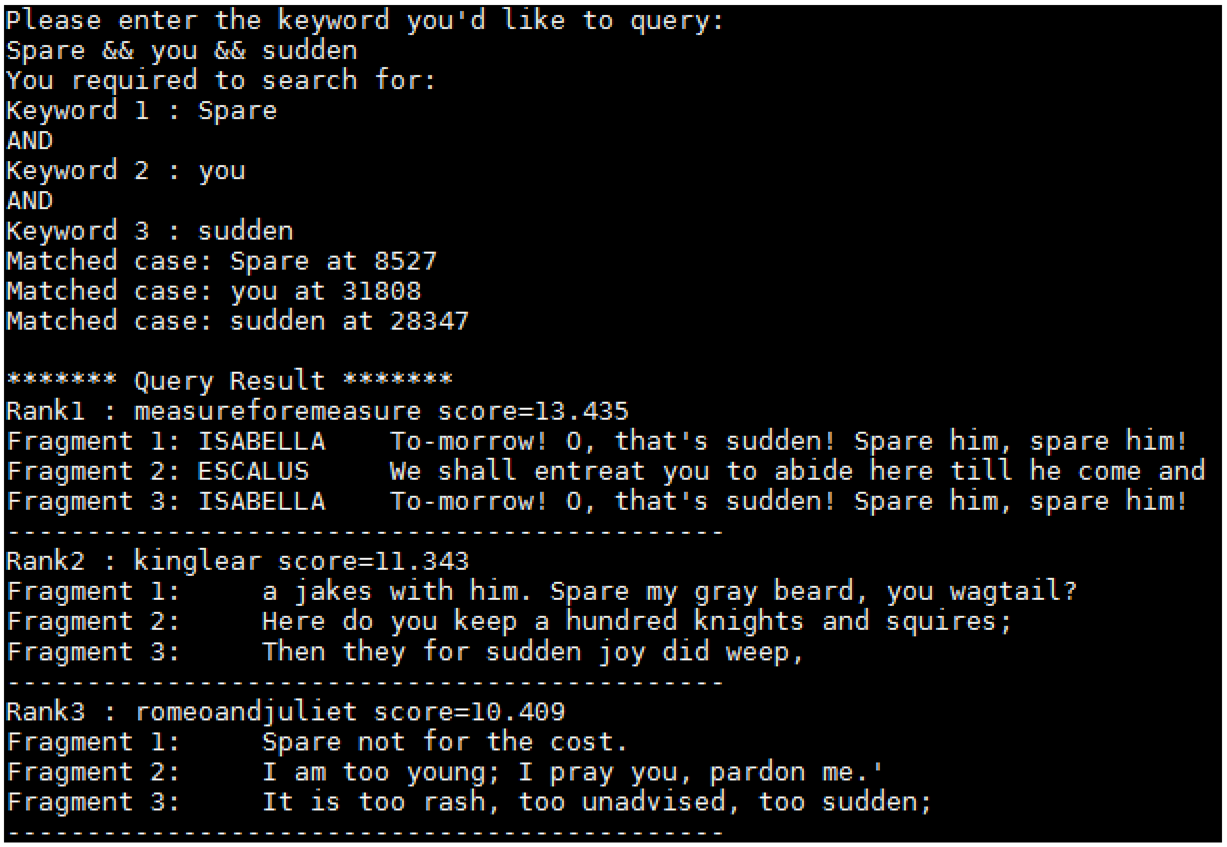
In the OR case, it would only display the first input keyword that follows the space bar “ ”, while in the AND case, it would display the fragments containing the first keyword and followed up by those with “&&” prior to them.

2.2.2 Sample execution and output

The below shows some test cases tried by myself, including a single word, AND, NOT, OR, and combination of them.

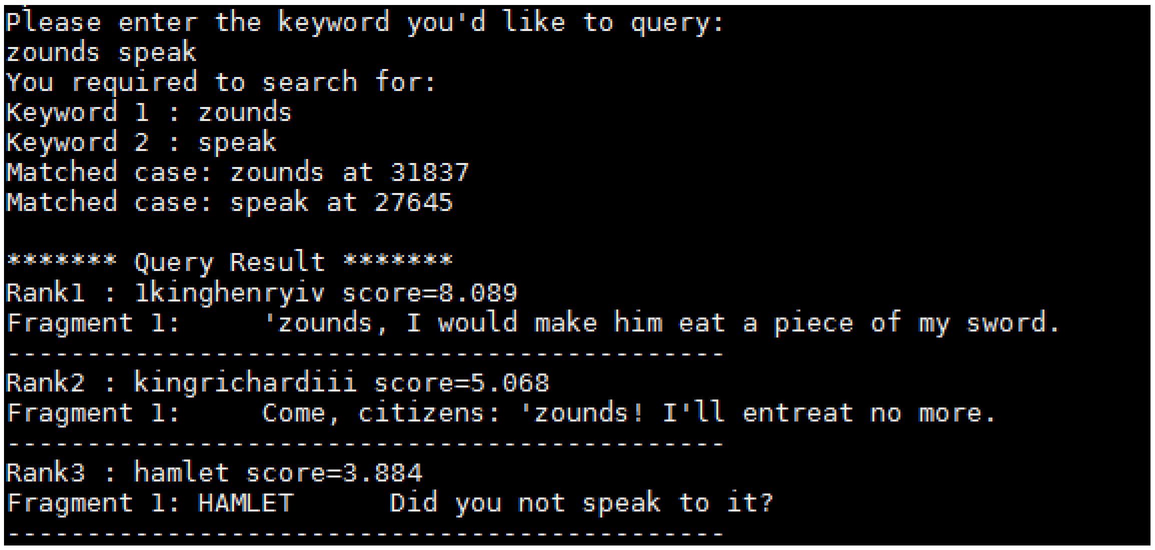

```
Please enter the keyword you'd like to query:
zounds
You required to search for:
Keyword 1 : zounds
Matched case: zounds at 31837

******* Query Result *******
Rank1 : 1kinghenryiv score=6.182
Fragment 1:     'zounds, I would make him eat a piece of my sword.
--------------------------------------------
Rank2 : kingrichardiii score=3.091
Fragment 1:     Come, citizens: 'zounds! I'll entreat no more.
--------------------------------------------
```



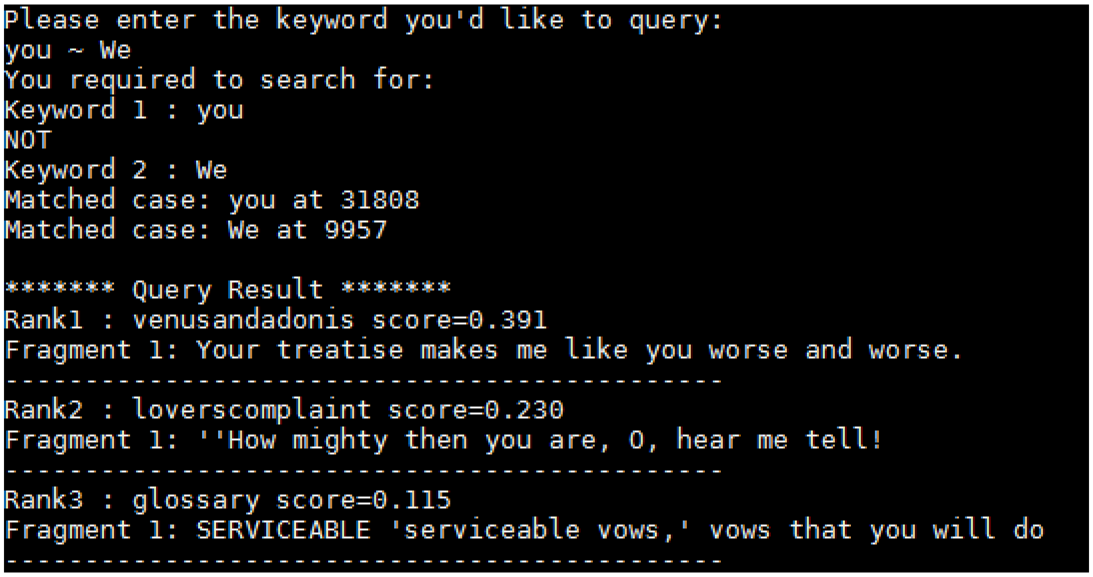
```
Please enter the keyword you'd like to query:
Spare && you && sudden
You required to search for:
Keyword 1 : Spare
AND
Keyword 2 : you
AND
Keyword 3 : sudden
Matched case: Spare at 8527
Matched case: you at 31808
Matched case: sudden at 28347

******* Query Result *******
Rank1 : measureforemeasure score=13.435
Fragment 1: ISABELLA    To-morrow! O, that's sudden! Spare him, spare him!
Fragment 2: ESCALUS     We shall entreat you to abide here till he come and
Fragment 3: ISABELLA    To-morrow! O, that's sudden! Spare him, spare him!
--------------------------------------------
Rank2 : kinglear score=11.343
Fragment 1:     a jakes with him. Spare my gray beard, you wagtail?
Fragment 2:     Here do you keep a hundred knights and squires;
Fragment 3:     Then they for sudden joy did weep,
--------------------------------------------
Rank3 : romeoandjuliet score=10.409
Fragment 1:     Spare not for the cost.
Fragment 2:     I am too young; I pray you, pardon me.'
Fragment 3:     It is too rash, too unadvised, too sudden;
--------------------------------------------
```



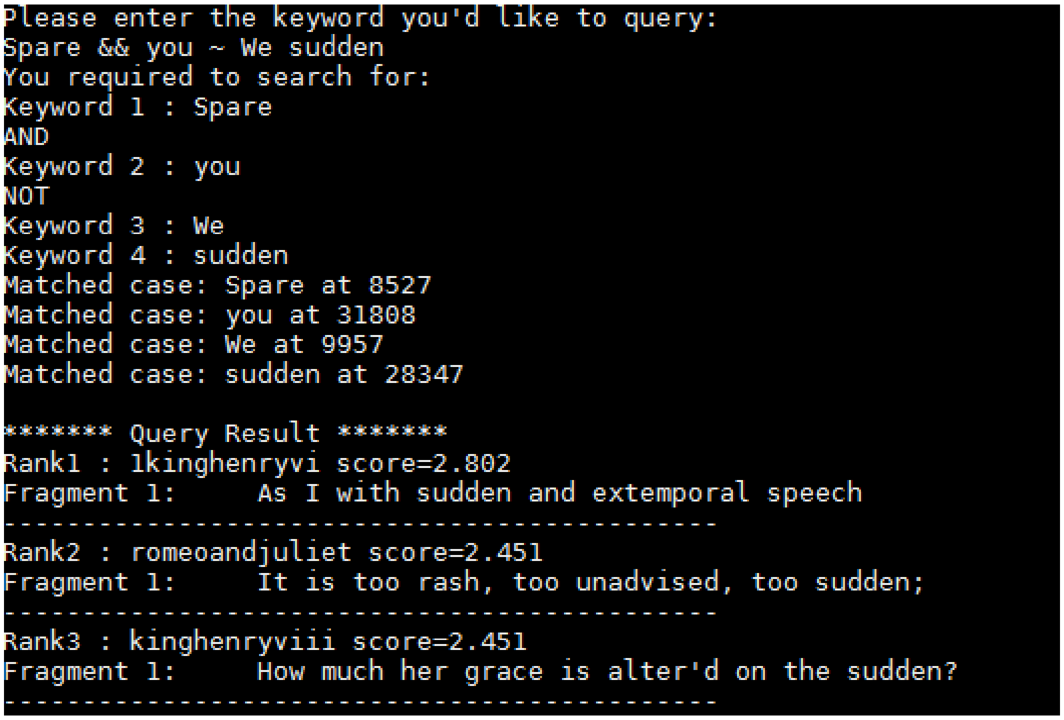
```
Please enter the keyword you'd like to query:
zounds speak
You required to search for:
Keyword 1 : zounds
Keyword 2 : speak
Matched case: zounds at 31837
Matched case: speak at 27645

******* Query Result *******
Rank1 : 1kinghenryiv score=8.089
Fragment 1:     'zounds, I would make him eat a piece of my sword.
--------------------------------------------
Rank2 : kingrichardiii score=5.068
Fragment 1:     Come, citizens: 'zounds! I'll entreat no more.
--------------------------------------------
Rank3 : hamlet score=3.884
Fragment 1: HAMLET      Did you not speak to it?
--------------------------------------------
```



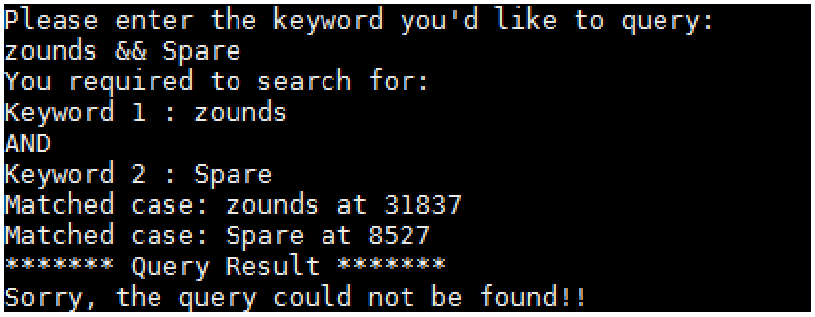
```
Please enter the keyword you'd like to query:
you ~ We
You required to search for:
Keyword 1 : you
NOT
Keyword 2 : We
Matched case: you at 31808
Matched case: We at 9957

******* Query Result *******
Rank1 : venusandadonis score=0.391
Fragment 1: Your treatise makes me like you worse and worse.
--------------------------------------------
Rank2 : loverscomplaint score=0.230
Fragment 1: ''How mighty then you are, O, hear me tell!
--------------------------------------------
Rank3 : glossary score=0.115
Fragment 1: SERVICEABLE 'serviceable vows,' vows that you will do
--------------------------------------------
```



```
Please enter the keyword you'd like to query:
Spare && you ~ We sudden
You required to search for:
Keyword 1 : Spare
AND
Keyword 2 : you
NOT
Keyword 3 : We
Keyword 4 : sudden
Matched case: Spare at 8527
Matched case: you at 31808
Matched case: We at 9957
Matched case: sudden at 28347

******* Query Result *******
Rank1 : 1kinghenryvi score=2.802
Fragment 1:     As I with sudden and extemporal speech
--------------------------------------------
Rank2 : romeoandjuliet score=2.451
Fragment 1:     It is too rash, too unadvised, too sudden;
--------------------------------------------
Rank3 : kinghenryviii score=2.451
Fragment 1:     How much her grace is alter'd on the sudden?
--------------------------------------------
```



```
Please enter the keyword you'd like to query:
zounds && Spare
You required to search for:
Keyword 1 : zounds
AND
Keyword 2 : Spare
Matched case: zounds at 31837
Matched case: Spare at 8527
******* Query Result *******
Sorry, the query could not be found!!
```

3 QUESTIONS

3.1 QUESTION 2

My extensions are clearly indicated with the design of the retrieval part, which are exactly the AND and NOT functions, with the display slightly modified to indicate the query word precisely if the AND function is triggered. The most difficult part would be the process dealing with those different functions, quite a few array lists should be initialized to gather the right information, avoiding mislead to the final output. The other obstacle was the demonstration of the fragments with respect to the line offset. Not until I learnt the .seek() API did I solve this problem with ease.

3.2 QUESTION 3

I stated that I applied the "\\W" splitting method to filter out all that is not a vocabulary, resulting in none of the useless notations could be sent by the mapper. If we need those characters, we should modify the splitting method with respect to the requirements. For example, if we need something like comma, we should add this requirement after \\W within the splitting API.