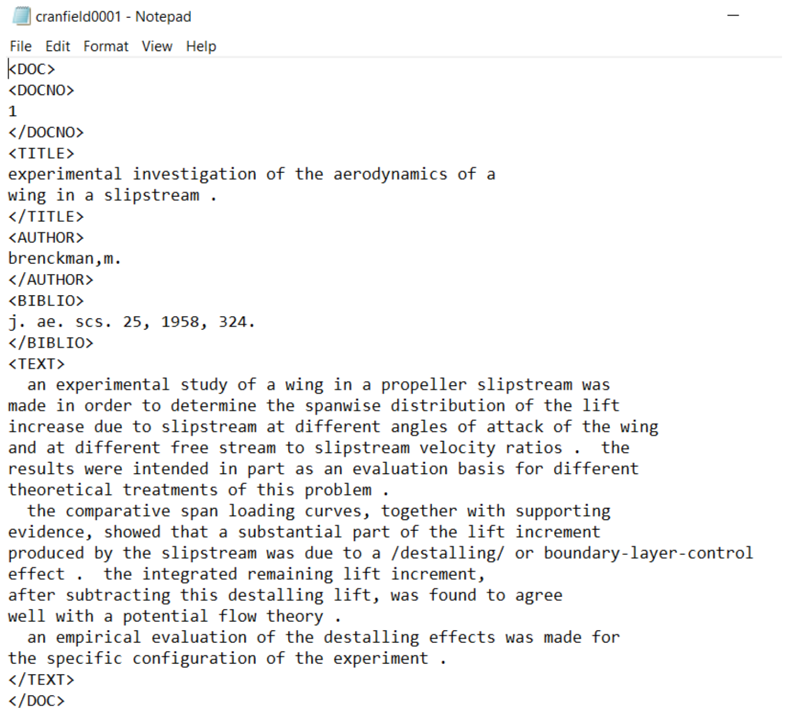
Group Number: 104
Group Members:
Akshita Gupta
Shivam Agrawal
Yash Kumar

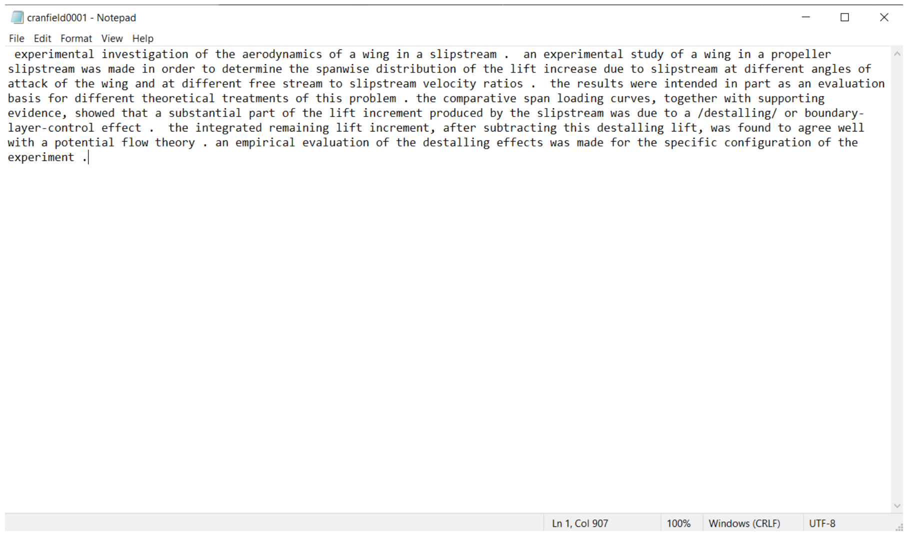
INFORMATION RETRIEVAL ASSIGNMENT 1

For this assignment, we were given a dataset of 1400 documents containing basic HTML format with different tags.



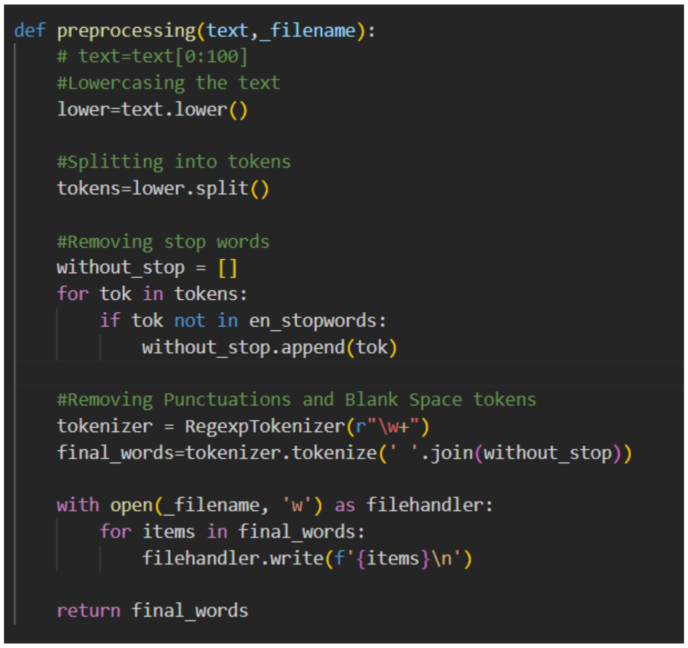
```
<DOC>
<DOCNO>
1
</DOCNO>
<TITLE>
experimental investigation of the aerodynamics of a
wing in a slipstream .
</TITLE>
<AUTHOR>
brenckman,m.
</AUTHOR>
<BIBLIO>
j. ae. scs. 25, 1958, 324.
</BIBLIO>
<TEXT>
  an experimental study of a wing in a propeller slipstream was
made in order to determine the spanwise distribution of the lift
increase due to slipstream at different angles of attack of the wing
and at different free stream to slipstream velocity ratios .  the
results were intended in part as an evaluation basis for different
theoretical treatments of this problem .
  the comparative span loading curves, together with supporting
evidence, showed that a substantial part of the lift increment
produced by the slipstream was due to a /destalling/ or boundary-layer-control
effect .  the integrated remaining lift increment,
after subtracting this destalling lift, was found to agree
well with a potential flow theory .
  an empirical evaluation of the destalling effects was made for
the specific configuration of the experiment .
</TEXT>
</DOC>
```

From this file we extracted the content under the “TITLE” and “TEXT” tag. For this we used simple python code which is mentioned in the “_text_extraction.py” file. First we find the file names and open them in a loop and find the required data, after which we overwrite the file with the extracted data in a single line.



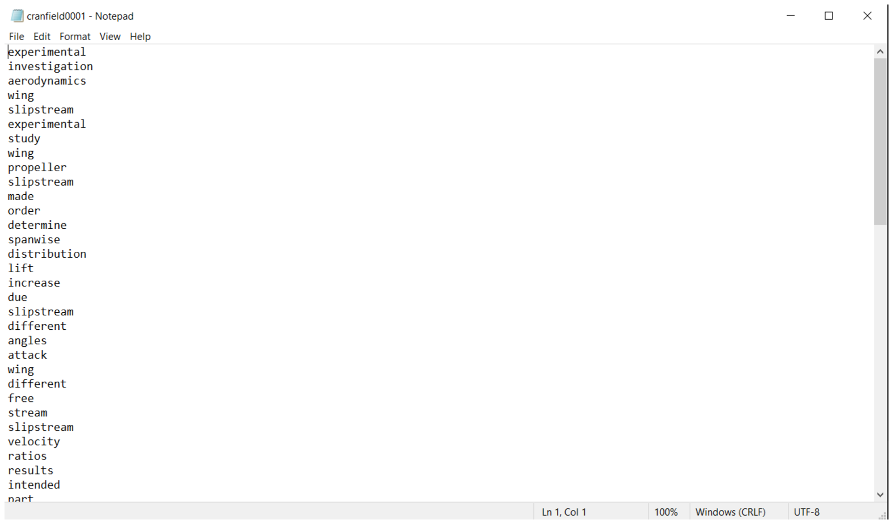
```
 experimental investigation of the aerodynamics of a wing in a slipstream .  an experimental study of a wing in a propeller
slipstream was made in order to determine the spanwise distribution of the lift increase due to slipstream at different angles of
attack of the wing and at different free stream to slipstream velocity ratios .  the results were intended in part as an evaluation
basis for different theoretical treatments of this problem . the comparative span loading curves, together with supporting
evidence, showed that a substantial part of the lift increment produced by the slipstream was due to a /destalling/ or boundary-
layer-control effect .  the integrated remaining lift increment, after subtracting this destalling lift, was found to agree well
with a potential flow theory . an empirical evaluation of the destalling effects was made for the specific configuration of the
experiment .
```

This is done for all the given 1400 files. This data is then preprocessed to remove the unnecessary words and blanks and punctuations. This is done in the file “preprocessing.ipynb”.



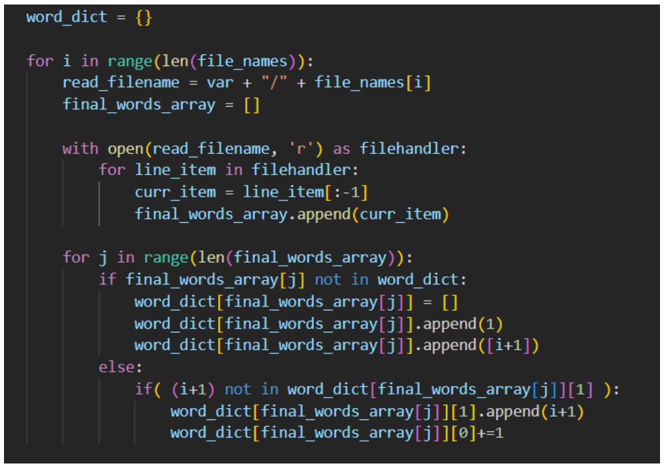
```python
def preprocessing(text,_filename):
    # text=text[0:100]
    #Lowercasing the text
    lower=text.lower()

    #Splitting into tokens
    tokens=lower.split()

    #Removing stop words
    without_stop = []
    for tok in tokens:
        if tok not in en_stopwords:
            without_stop.append(tok)

    #Removing Punctuations and Blank Space tokens
    tokenizer = RegexpTokenizer(r"\w+")
    final_words=tokenizer.tokenize(' '.join(without_stop))

    with open(_filename, 'w') as filehandler:
        for items in final_words:
            filehandler.write(f'{items}\n')

    return final_words
```

Once the data is processed we store this in a new Folder by the name “Processed_Files”. This data looks like:



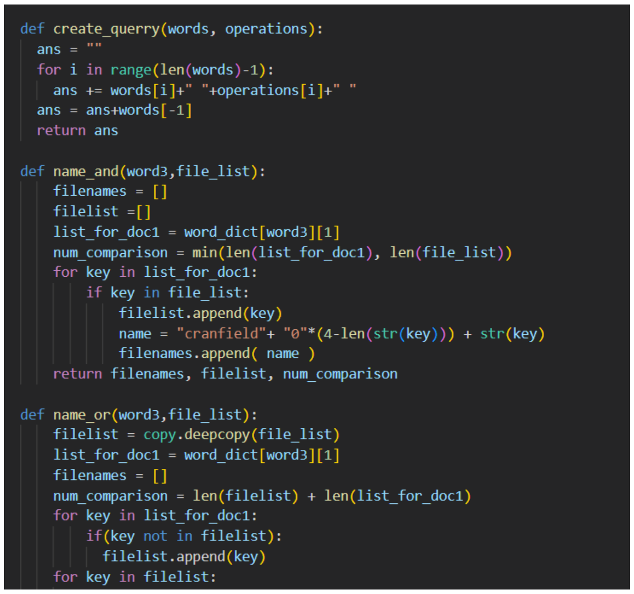
```
experimental
investigation
aerodynamics
wing
slipstream
experimental
study
wing
propeller
slipstream
made
order
determine
spanwise
distribution
lift
increase
due
slipstream
different
angles
attack
wing
different
free
stream
slipstream
velocity
ratios
results
intended
part
```

For Q2, we first have to create a unigram inverted index and then create a system to run the mentioned boolean queries. These include “AND”, “OR”, “AND NOT”, “OR NOT”. We have created the unigram inverted index from the processed files by the code written in file “inverted_index.ipynb”.



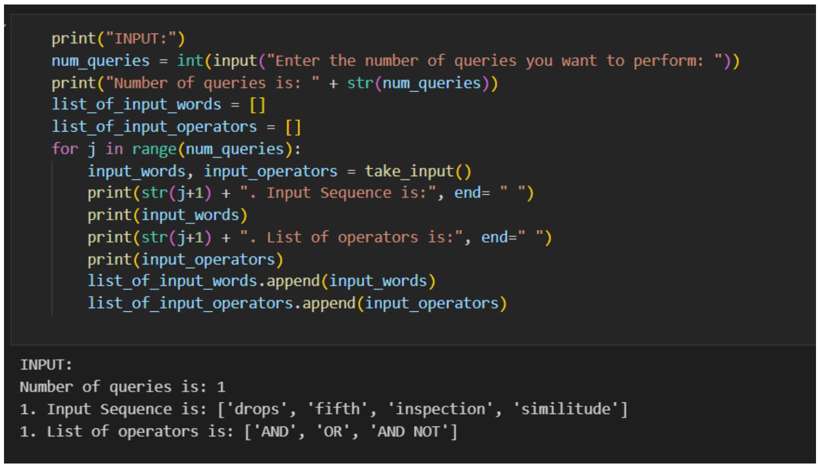
```python
word_dict = {}

for i in range(len(file_names)):
    read_filename = var + "/" + file_names[i]
    final_words_array = []

    with open(read_filename, 'r') as filehandler:
        for line_item in filehandler:
            curr_item = line_item[:-1]
            final_words_array.append(curr_item)

    for j in range(len(final_words_array)):
        if final_words_array[j] not in word_dict:
            word_dict[final_words_array[j]] = []
            word_dict[final_words_array[j]].append(1)
            word_dict[final_words_array[j]].append([i+1])
        else:
            if( (i+1) not in word_dict[final_words_array[j]][1] ):
                word_dict[final_words_array[j]][1].append(i+1)
                word_dict[final_words_array[j]][0]+=1
```

Using pickle module of python this inverted index is stored in “Unigram_Inverted_Index_File”. For boolean queries, load function of pickle is used to load the Unigram_Inverted_Index_File. The code for boolean queries is written in the file “boolean_query.ipynb”.



```python
def create_querry(words, operations):
  ans = ""
  for i in range(len(words)-1):
    ans += words[i]+" "+operations[i]+" "
  ans = ans+words[-1]
  return ans

def name_and(word3,file_list):
    filenames = []
    filelist =[]
    list_for_doc1 = word_dict[word3][1]
    num_comparison = min(len(list_for_doc1), len(file_list))
    for key in list_for_doc1:
        if key in file_list:
            filelist.append(key)
            name = "cranfield"+ "0"*(4-len(str(key))) + str(key)
            filenames.append( name )
    return filenames, filelist, num_comparison

def name_or(word3,file_list):
    filelist = copy.deepcopy(file_list)
    list_for_doc1 = word_dict[word3][1]
    filenames = []
    num_comparison = len(filelist) + len(list_for_doc1)
    for key in list_for_doc1:
        if(key not in filelist):
          filelist.append(key)
    for key in filelist:
```

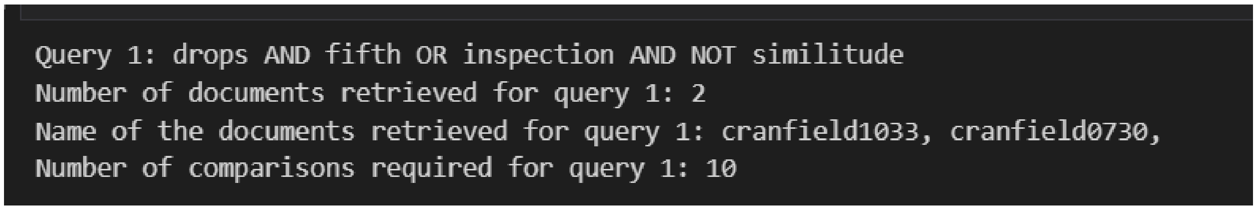
Input is taken in the following format:



```python
print("INPUT:")
num_queries = int(input("Enter the number of queries you want to perform: "))
print("Number of queries is: " + str(num_queries))
list_of_input_words = []
list_of_input_operators = []
for j in range(num_queries):
    input_words, input_operators = take_input()
    print(str(j+1) + ". Input Sequence is:", end= " ")
    print(input_words)
    print(str(j+1) + ". List of operators is:", end=" ")
    print(input_operators)
    list_of_input_words.append(input_words)
    list_of_input_operators.append(input_operators)
```

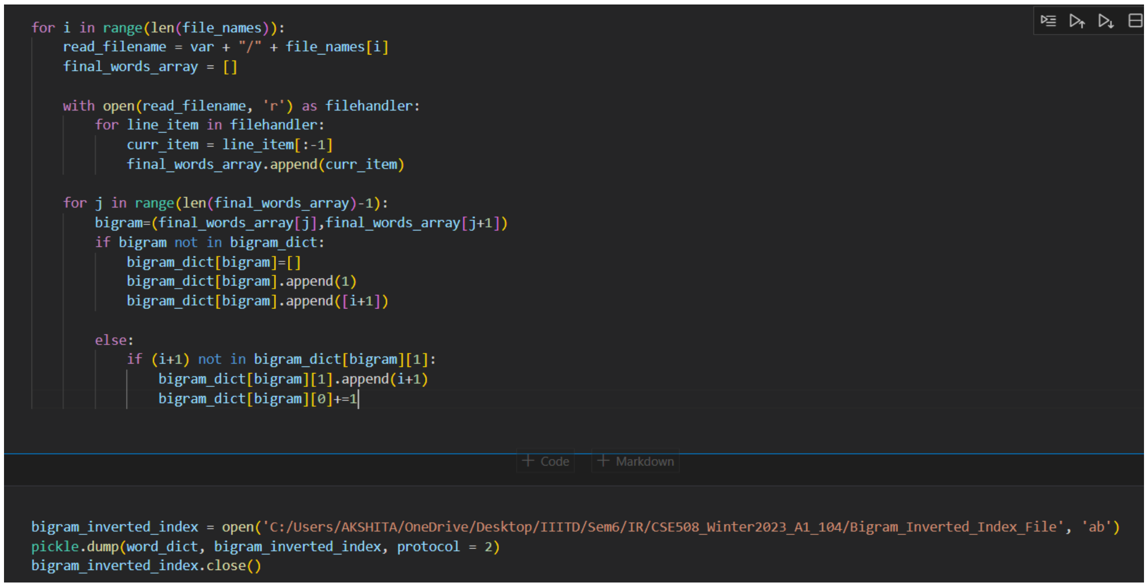
```
INPUT:
Number of queries is: 1
1. Input Sequence is: ['drops', 'fifth', 'inspection', 'similitude']
1. List of operators is: ['AND', 'OR', 'AND NOT']
```

And the output is printed in the following format:



```
Query 1: drops AND fifth OR inspection AND NOT similitude
Number of documents retrieved for query 1: 2
Name of the documents retrieved for query 1: cranfield1033, cranfield0730,
Number of comparisons required for query 1: 10
```

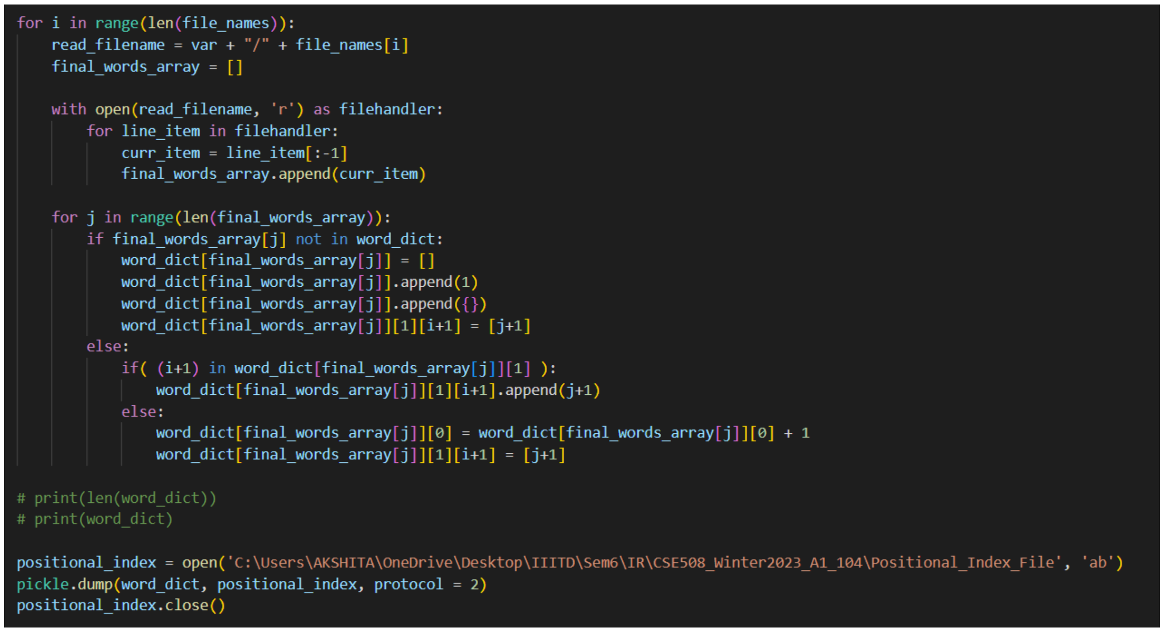
For question 3, we create 2 different indexes, Bigram Inverted Index and Positional Index. The code for bigram inverted index is in the file “inverted_index.ipynb”. After creating the index, it is stored using the pickle module of python in “Bigram_Inverted_Index_File”.



```python
for i in range(len(file_names)):
    read_filename = var + "/" + file_names[i]
    final_words_array = []

    with open(read_filename, 'r') as filehandler:
        for line_item in filehandler:
            curr_item = line_item[:-1]
            final_words_array.append(curr_item)

    for j in range(len(final_words_array)-1):
        bigram=(final_words_array[j],final_words_array[j+1])
        if bigram not in bigram_dict:
            bigram_dict[bigram]=[]
            bigram_dict[bigram].append(1)
            bigram_dict[bigram].append([i+1])

        else:
            if (i+1) not in bigram_dict[bigram][1]:
                bigram_dict[bigram][1].append(i+1)
                bigram_dict[bigram][0]+=1
```

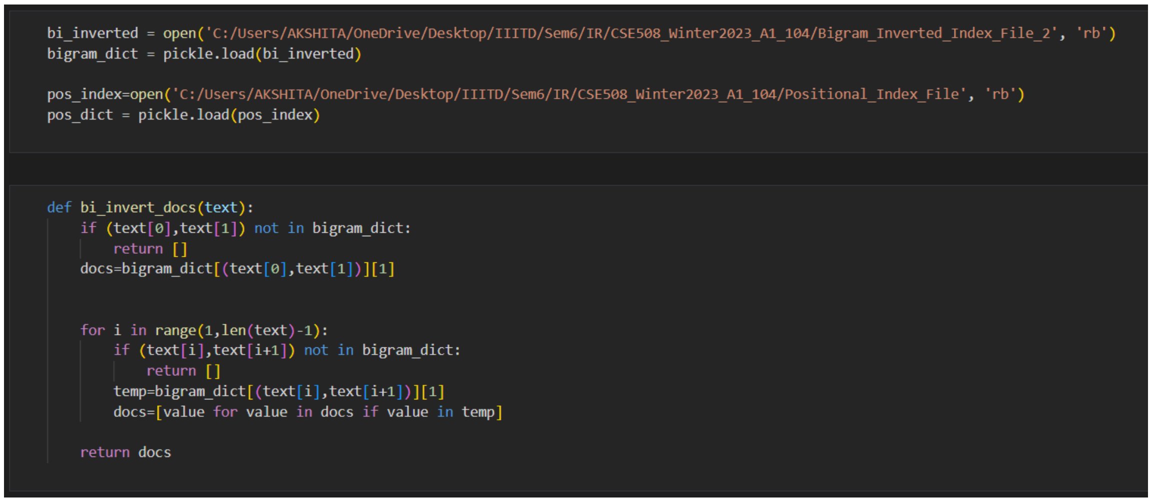
```python
bigram_inverted_index = open('C:/Users/AKSHITA/OneDrive/Desktop/IIITD/Sem6/IR/CSE508_Winter2023_A1_104/Bigram_Inverted_Index_File', 'ab')
pickle.dump(word_dict, bigram_inverted_index, protocol = 2)
bigram_inverted_index.close()
```

The positional index is coded in the file “positional_index.py”. After creating the index it is stored using the pickle module in “Positional_Index_File”.



```python
for i in range(len(file_names)):
    read_filename = var + "/" + file_names[i]
    final_words_array = []

    with open(read_filename, 'r') as filehandler:
        for line_item in filehandler:
            curr_item = line_item[:-1]
            final_words_array.append(curr_item)

    for j in range(len(final_words_array)):
        if final_words_array[j] not in word_dict:
            word_dict[final_words_array[j]] = []
            word_dict[final_words_array[j]].append(1)
            word_dict[final_words_array[j]].append({})
            word_dict[final_words_array[j]][1][i+1] = [j+1]
        else:
            if( (i+1) in word_dict[final_words_array[j]][1] ):
                word_dict[final_words_array[j]][1][i+1].append(j+1)
            else:
                word_dict[final_words_array[j]][0] = word_dict[final_words_array[j]][0] + 1
                word_dict[final_words_array[j]][1][i+1] = [j+1]

# print(len(word_dict))
# print(word_dict)

positional_index = open('C:\Users\AKSHITA\OneDrive\Desktop\IIITD\Sem6\IR\CSE508_Winter2023_A1_104\Positional_Index_File', 'ab')
pickle.dump(word_dict, positional_index, protocol = 2)
positional_index.close()
```

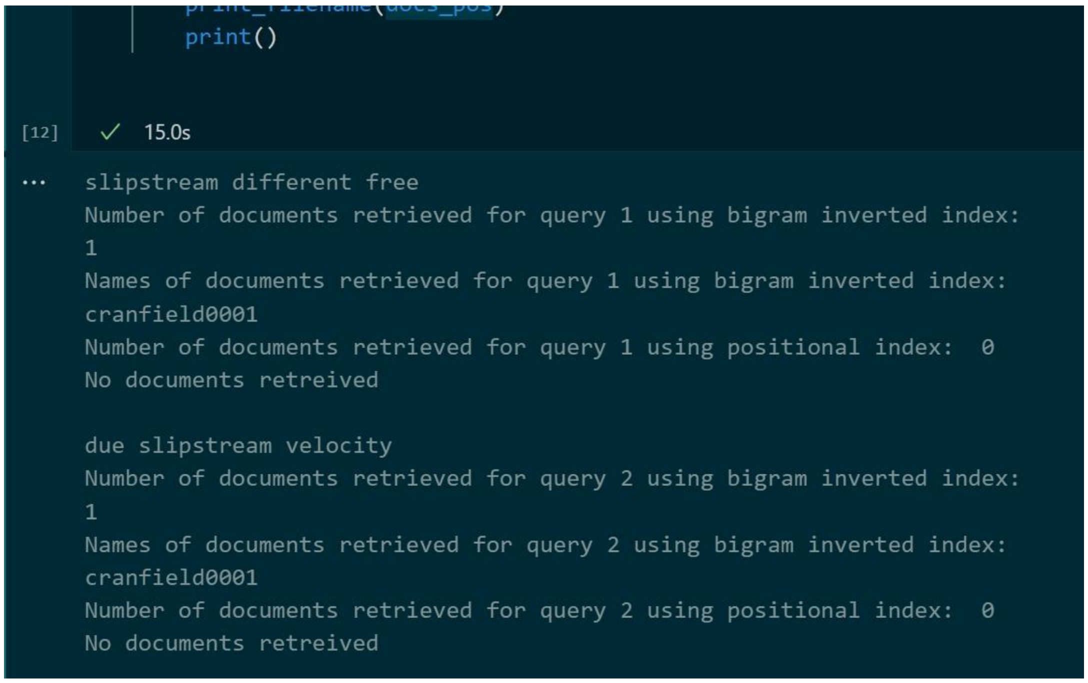
In the end we have to create a system to solve the phrase queries for both the indexes. The system is developed in the file “queries_bigram.ipynb”.



```python
bi_inverted = open('C:/Users/AKSHITA/OneDrive/Desktop/IIITD/Sem6/IR/CSE508_Winter2023_A1_104/Bigram_Inverted_Index_File_2', 'rb')
bigram_dict = pickle.load(bi_inverted)

pos_index=open('C:/Users/AKSHITA/OneDrive/Desktop/IIITD/Sem6/IR/CSE508_Winter2023_A1_104/Positional_Index_File', 'rb')
pos_dict = pickle.load(pos_index)
```

```python
def bi_invert_docs(text):
    if (text[0],text[1]) not in bigram_dict:
        return []
    docs=bigram_dict[(text[0],text[1])][1]


    for i in range(1,len(text)-1):
        if (text[i],text[i+1]) not in bigram_dict:
            return []
        temp=bigram_dict[(text[i],text[i+1])][1]
        docs=[value for value in docs if value in temp]

    return docs
```

Input and Output are taken in the following format:



```
        print()

[12]  ✓ 15.0s

...   slipstream different free
      Number of documents retrieved for query 1 using bigram inverted index:
      1
      Names of documents retrieved for query 1 using bigram inverted index:
      cranfield0001
      Number of documents retrieved for query 1 using positional index:  0
      No documents retreived

      due slipstream velocity
      Number of documents retrieved for query 2 using bigram inverted index:
      1
      Names of documents retrieved for query 2 using bigram inverted index:
      cranfield0001
      Number of documents retrieved for query 2 using positional index:  0
      No documents retreived
```

For the assignment we followed certain Assumptions,

1. While performing indexing we have started indexing from 1 instead of 0. Thus for phrases like “experimental study evaluation”, in positional indexing “experimental” is 1, “study” is 2 and “evaluation” is 3.
2. For stop words we have used the already provided stop words dataset available in python.
3. We have taken all the punctuations into account.
4. Number of keywords should always be more than equal to 2 for processing bigram queries, also number of keywords should be 1 more than number of operators for boolean queries.
5. In phrase queries, the position of words in queries is allotted after preprocessing of the phrase. Thus for a query “experimental is a study”, study is allotted the position 2, not 4.