Q1. Data Preprocessing

(i) **Relevant Text Extraction [2 Marks]**

For each file, extract the contents between the <TITLE>...</TITLE> and <TEXT>...</TEXT> tags and concatenate the 2 strings using blank space. Discard the rest of the text and save the string obtained above in the same file. [Do NOT change filename].

Assumptions :

The code would work perfectly only when it finds the <TITLE>...</TITLE> and <TEXT>...</TEXT> tags in the raw input file, once it finds the tags.

Methodologies :

```
In [5]: title_flag=0
        text_flag=0

        for file_index in range(len(file_list)):
            file_name = f'{folder_name}/{file_list[file_index]}'

            lines_between_title_tag=''
            lines_between_text_tag=''

            with open(file_name) as file:
                file = open(file_name)
                all_lines = file.readlines()
                file_content = ''.join(all_lines)

            if file_index<5:
                print(f'File - {file_index+1}')
                print('File contents before preprocessing\n')
                print(file_content)

            for line in all_lines:
                if line=='<TITLE>\n':
                    title_flag=1
                if line=='</TITLE>\n':
                    title_flag=0

                if line=='<TEXT>\n':
                    text_flag=1
                if line=='</TEXT>\n':
                    text_flag=0

                if title_flag==1 and line!='<TITLE>\n':
                    lines_between_title_tag=lines_between_title_tag+line

                if text_flag==1 and line!='<TEXT>\n':
                    lines_between_text_tag=lines_between_text_tag+line
```

The logic above iterates over a file line by line and then adds the lines after <TITLE> tag to lines_between_title_tag variable till </TITLE> tag is encountered in the file.

The logic above iterates over a file line by line and then adds the lines after <TEXT> tag to lines_between_text_tag variable till </TEXT> tag is encountered in the file.

Results :

Contents of a file before relevant text extraction:

```
File contents before preprocessing

<DOC>
<DOCNO>
5
</DOCNO>
<TITLE>
one-dimensional transient heat conduction into a double-layer
slab subjected to a linear heat input for a small time
internal .
</TITLE>
<AUTHOR>
wasserman,b.
</AUTHOR>
<BIBLIO>
j. ae. scs. 24, 1957, 924.
</BIBLIO>
<TEXT>
  analytic solutions are presented for the transient heat conduction
in composite slabs exposed at one surface to a
```

Contents of a file after relevant text extraction:

```
</BIBLIO>
<TEXT>
  analytic solutions are presented for the transient heat conduction
in composite slabs exposed at one surface to a
triangular heat rate .  this type of heating rate may occur, for
example, during aerodynamic heating .
</TEXT>
</DOC>

File contents after preprocessing

one-dimensional transient heat conduction into a double-layer
slab subjected to a linear heat input for a small time
internal .
   analytic solutions are presented for the transient heat conduction
in composite slabs exposed at one surface to a
triangular heat rate .  this type of heating rate may occur, for
example, during aerodynamic heating .
```

(ii) Preprocessing

Carry out the following preprocessing steps on the dataset obtained above: [8 Marks]

Assumptions :

The code responsible for making the text lowercase,performing tokenisation,removing stopwords,removing punctuation,remove blank space tokens runs perfectly when the relevant text extraction has been done already i.e. only when text between <TITLE> and <TEXT> tags.

Methodologies :

1) Lowercase the text

ii)

In [6]:
```
from nltk.tokenize import word_tokenize,RegexpTokenizer
from nltk.corpus import stopwords
import string
```

In [7]:
```
for file_index in range(0,len(file_list)):
    file_name = f'{folder_name}/{file_list[file_index]}'
    with open(file_name,'r') as file:
        file_contents = file.read()

    if file_index<5:
        print(f'\n\n\nFile - {file_index+1}')
        print('\nContents of file before making them lowercase',end='\n\n')
        print_file_contents(file_name)
    updated_words = file_contents.lower()
    write_to_file(file_name,updated_words)
    if file_index<5:
        print('\nContents of file after making them lowercase',end='\n\n')
        print_file_contents(file_name)
```

The above code reads and file, makes the text to lowercase and writes the file back

Results :

Contents of the file before and after this step:

```
File - 3

Contents of file before making them lowercase

the boundary layer in simple shear flow past a flat plate .
 the boundary-layer equations are presented for steady
incompressible flow with no pressure gradient .


Contents of file after making them lowercase

the boundary layer in simple shear flow past a flat plate .
 the boundary-layer equations are presented for steady
incompressible flow with no pressure gradient .
```

2) Perform tokenization

```
In [8]: for file_index in range(0,len(file_list)):
            file_name = f'{folder_name}/{file_list[file_index]}'
            with open(file_name,'r') as file:
                file_contents = file.read()
            if file_index<5:
                print(f'\n\n\nFile - {file_index+1}')
                print('\nContents of file before tokenisation',end='\n\n')
                print_file_contents(file_name)
            updated_word_list = word_tokenize(file_contents)
            updated_words = ' '.join(updated_word_list)
            write_to_file(file_name,updated_words)
            if file_index<5:
                print('\nContents of file after tokenisation',end='\n\n')
                print_file_contents(file_name)
```

The above code reads and file,performs tokenisation and writes the file back

Results:

Contents of the file before and after this step:

```
File - 3

Contents of file before tokenisation

the boundary layer in simple shear flow past a flat plate .
 the boundary-layer equations are presented for steady
incompressible flow with no pressure gradient .


Contents of file after tokenisation

the boundary layer in simple shear flow past a flat plate . the boundary-layer equations are presented for steady incompressi
ble flow with no pressure gradient .



File - 4

Contents of file before tokenisation
```

3) Remove stopwords

```
In [9]: for file_index in range(0,len(file_list)):
            file_name = f'{folder_name}/{file_list[file_index]}'
            with open(file_name,'r') as file:
                file_contents = file.read()
            if file_index<5:
                print(f'\n\n\nFile - {file_index+1}')
                print('\nContents of file before removing stopwords',end='\n\n')
                print_file_contents(file_name)
            stop_words = stopwords.words('english')
            updated_word_list = file_contents.split()
            updated_word_list = [word for word in updated_word_list if word not in stop_words]
            updated_words = ' '.join(updated_word_list)
            write_to_file(file_name,updated_words)
            if file_index<5:
                print('\nContents of file after removing stopwords',end='\n\n')
                print_file_contents(file_name)
```

The above code reads and file, removes stopwords and writes the file back

Results:

Contents of the file before and after this step

```
File - 1

Contents of file before removing stopwords

experimental investigation of the aerodynamics of a wing in a slipstream . an experimental study of a wing in a propeller slips
tream was made in order to determine the spanwise distribution of the lift increase due to slipstream at different angles of at
tack of the wing and at different free stream to slipstream velocity ratios . the results were intended in part as an evaluatio
n basis for different theoretical treatments of this problem . the comparative span loading curves , together with supporting e
vidence , 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 w
ell with a potential flow theory . an empirical evaluation of the destalling effects was made for the specific configuration of
the experiment .

Contents of file after removing stopwords

experimental investigation aerodynamics wing slipstream . experimental study wing propeller slipstream made order determine spa
nwise distribution lift increase due slipstream different angles attack wing different free stream slipstream velocity ratios .
results intended part evaluation basis different theoretical treatments problem . comparative span loading curves , together su
pporting evidence , showed substantial part lift increment produced slipstream due /destalling/ boundary-layer-control effect .
integrated remaining lift increment , subtracting destalling lift , found agree well potential flow theory . empirical evaluati
on destalling effects made specific configuration experiment .
```

4) Remove punctuations

```
In [10]: for file_index in range(0,len(file_list)):
             file_name = f'{folder_name}/{file_list[file_index]}'
             with open(file_name,'r') as file:
                 file_contents = file.read()
             if file_index<5:
                 print(f'\n\n\nFile - {file_index+1}')
                 print('\nContents of file before removing punctuations',end='\n\n')
                 print_file_contents(file_name)

             updated_word_list = file_contents.split()

             token = RegexpTokenizer(r"\w+")

             updated_word_list = token.tokenize(' '.join(updated_word_list))
             updated_words = ' '.join(updated_word_list)

             write_to_file(file_name,updated_words)
             if file_index<5:
                 print('\nContents of file after removing punctuations',end='\n\n')
                 print_file_contents(file_name)
```

The above code reads and file, removes punctuations and writes the file back

Results:

Contents of the file before and after this step

```
File - 1

Contents of file before removing punctuations

experimental investigation aerodynamics wing slipstream . experimental study wing propeller slipstream made order determine spa
nwise distribution lift increase due slipstream different angles attack wing different free stream slipstream velocity ratios .
results intended part evaluation basis different theoretical treatments problem . comparative span loading curves , together su
pporting evidence , showed substantial part lift increment produced slipstream due /destalling/ boundary-layer-control effect .
integrated remaining lift increment , subtracting destalling lift , found agree well potential flow theory . empirical evaluati
on destalling effects made specific configuration experiment .

Contents of file after removing punctuations

experimental investigation aerodynamics wing slipstream experimental study wing propeller slipstream made order determine spanw
ise distribution lift increase due slipstream different angles attack wing different free stream slipstream velocity ratios res
ults intended part evaluation basis different theoretical treatments problem comparative span loading curves together supportin
g evidence showed substantial part lift increment produced slipstream due destalling boundary layer control effect integrated r
emaining lift increment subtracting destalling lift found agree well potential flow theory empirical evaluation destalling effe
cts made specific configuration experiment


File - 2
```

5) Remove blank space tokens

```
In [11]: for file_index in range(0,len(file_list)):
             file_name = f'{folder_name}/{file_list[file_index]}'
             with open(file_name,'r') as file:
                 file_contents = file.read()
             if file_index<5:
                 print(f'\n\n\nFile - {file_index+1}')
                 print('\nContents of file before removing blank spaces',end='\n\n')
                 print_file_contents(file_name)

             word_list = file_contents.split()
             word_list = [word.strip() for word in word_list]
             file_contents = ' '.join(word_list)
             write_to_file(file_name,file_contents)
             if file_index<5:
                 print('\nContents of file after removing blank spaces',end='\n\n')
                 print_file_contents(file_name)
```

The above code reads and file, removes blank space tokens and writes the file back

Results:

Contents of the file before and after this step

File - 2

Contents of file before removing blank spaces

simple shear flow past flat plate incompressible fluid small viscosity study high speed viscous flow past two dimensional body usually necessary consider curved shock wave emitting nose leading edge body consequently exists inviscid rotational flow regio n shock wave boundary layer situation arises instance study hypersonic viscous flow past flat plate situation somewhat differen t prandtl s classical boundary layer problem prandtl s original problem inviscid free stream outside boundary layer irrotationa l hypersonic boundary layer problem inviscid free stream must considered rotational possible effects vorticity recently discuss ed ferri libby present paper simple shear flow past flat plate fluid small viscosity investigated shown problem treated boundar y layer approximation novel feature free stream constant vorticity discussion restricted two dimensional incompressible steady flow

Contents of file after removing blank spaces

simple shear flow past flat plate incompressible fluid small viscosity study high speed viscous flow past two dimensional body usually necessary consider curved shock wave emitting nose leading edge body consequently exists inviscid rotational flow regio n shock wave boundary layer situation arises instance study hypersonic viscous flow past flat plate situation somewhat differen t prandtl s classical boundary layer problem prandtl s original problem inviscid free stream outside boundary layer irrotationa l hypersonic boundary layer problem inviscid free stream must considered rotational possible effects vorticity recently discuss ed ferri libby present paper simple shear flow past flat plate fluid small viscosity investigated shown problem treated boundar y layer approximation novel feature free stream constant vorticity discussion restricted two dimensional incompressible steady flow

Q2

(i) Create a unigram inverted index(from scratch; No library allowed) of the dataset obtained from Q1

Reading the data after preprocessing from Q1.

In [5]:
```
# creating the list of names of files in which a particular word occured along with frequency
# and creating a list with all the unique words
file_names = []
s = []
all_words = ""
for file in files:
    f = open(f'{folder}/{file}','r')
    file_contents = f.read()
    file_names.append(file)
    words = list(file_contents.split(" "))
    words = list(set(words))
    s.append(words)
    temp = " ".join(words)
    all_words = all_words + " "+ temp
    f.close()
all_words = list(all_words.split(" "))
all_words = list(set(all_words))
for ele in all_words:
    if ele == '':
        all_words.remove(ele)
```

Implementing the unigram inverted index from scratch

In [9]:
```
# creating unigram inverted_index
inverted_index = {}
for word in all_words:
    values = [0,[]]
    count = 0
    for j in range(len(s)):
        if word in s[j]:
            values[1].append(file_names[j])
            count += 1
    values[0] = count
    values[1] = sorted(values[1])
    inverted_index[word] = values

# inverted_index
```

Using pickle module to dump the unigram inverted index created above into “inverted_index.pkl” file.

In [10]:
```
# dumping the inverted index using pickle
pickle.dump(inverted_index,open("inverted_index.pkl",'wb'))
```

Using pickle module to load the unigram inverted index from “inverted_index.pkl” file.

In [11]:
```
#loading the inverted index
loaded_inverted_index = pickle.load(open('inverted_index.pkl','rb'))
```

Implementing intersection(AND) from scratch. The method takes two lists as arguments and returns the list after performing AND/ NOT AND on the two terms along with number of comparisons.

In [14]:
```
# AND and NOT AND logic
# taking two lists as parameters and returning a list and number of comparisions
# l1 is smaller than l2
def intersection(l1,l2):
    comparisions = 0;
    i=0
    j=0
    l1_and_l2 = []
    while(i < len(l1) and j < len(l2)):
        if(l1[i] == l2[j]):
            l1_and_l2.append(l1[i])
            i += 1
            j += 1
            comparisions += 1
        elif(l1[i] < l2[j]):
            # update smaller value i.e i
            i+= 1
            comparisions += 2
        else:
            j+= 1
            comparisions += 2
    return (l1_and_l2,comparisions)
```

Implementing union (OR) from scratch. The method takes two lists as arguments and returns the list after performing OR/ NOT OR on the two terms along with number of comparisons.

In [15]:
```
# OR and NOT OR logic
# taking two lists as parameters and returning a list and number of comparisions
# l1 is smaller than l2
def union(l1,l2):
    comparisions = 0
    i=0
    j=0
    l1_or_l2 = []
    while(i<len(l1) and j<len(l2)):
        if(l1[i] == l2[j]):
            l1_or_l2.append(l1[i])
            i+=1
            j+=1
            comparisions += 1
        elif(l1[i]<l2[j]):
            l1_or_l2.append(l1[i])
            i+=1
            comparisions += 2
        else:
            l1_or_l2.append(l2[j])
            j+=1
            comparisions += 2
    if(j < len(l2)):
        l1_or_l2.extend(l2[j:])
    return (l1_or_l2,comparisions)
```

T1 AND T2

In [24]:
```
# T1 AND T2
# t1 = 'dissociating'
# t2 = 'gas'
print("T1 AND T2")
t1 = input("Plese enter the first term : ")
t2 = input("please enter the second term : ")
t1_list = loaded_inverted_index[t1][1]
t1_list_len = len(loaded_inverted_index[t1][1])
t2_list = loaded_inverted_index[t2][1]
t2_list_len = len(loaded_inverted_index[t2][1])

# using function build from scratch
if(len(t1_list) < len(t2_list)):
    t1_and_t2, comparisions = intersection(t1_list,t2_list)
else:
    t1_and_t2, comparisions = intersection(t2_list,t1_list)
print(f'List of files containing {t1} AND {t2} : {t1_and_t2}')
print(f'numeber of comparisions : {comparisions}')

# #using set functions
# t1_and_t2 = list(set(t1_list) & set(t2_list))

# print(min(t1_list_len,t2_list_len))
# print(sorted(t1_and_t2))
```

```
T1 AND T2
Plese enter the first term : dissociating
please enter the second term : gas
List of files containing dissociating AND gas : ['cranfield0110', 'cranfield0166', 'cranfield0167', 'cranfield0236',
'cranfield0317', 'cranfield0517', 'cranfield0575', 'cranfield0656', 'cranfield1295']
numeber of comparisions : 235
```

T1 OR T2

In [25]:
```
# T1 OR T2
# t1 = 'positive'
# t2 = 'letter'
print("T1 OR T2")
t1 = input("Plese enter the first term : ")
t2 = input("please enter the second term : ")
t1_list = loaded_inverted_index[t1][1]
t1_list_len = len(loaded_inverted_index[t1][1])
t2_list = loaded_inverted_index[t2][1]
t2_list_len = len(loaded_inverted_index[t2][1])
if(len(t1_list) < len(t2_list)):
    t1_or_t2, comparisions = union(t1_list,t2_list)
else:
    t1_or_t2, comparisions = union(t2_list,t1_list)
print(f'List of files containing {t1} OR {t2} : {t1_or_t2}')
print(f'numeber of comparisions : {comparisions}')
```

```
T1 OR T2
Plese enter the first term : positive
please enter the second term : letter
List of files containing positive OR letter : ['cranfield0133', 'cranfield0150', 'cranfield0371', 'cranfield0392', 'c
ranfield0447', 'cranfield0515', 'cranfield0636', 'cranfield0693', 'cranfield0694', 'cranfield0721', 'cranfield0988',
'cranfield1005', 'cranfield1035', 'cranfield1147', 'cranfield1201', 'cranfield1218', 'cranfield1302', 'cranfield134
7', 'cranfield1365']
numeber of comparisions : 6
```

T1 AND NOT T2

In [27]:
```
# T1 AND NOT T2
# t1 = 'positive'
# t2 = 'letter'
print("T1 AND NOT T2")
t1 = input("Plese enter the first term : ")
t2 = input("please enter the second term : ")
t1_list = loaded_inverted_index[t1][1]
t1_list_len = len(loaded_inverted_index[t1][1])
t2_list = loaded_inverted_index[t2][1]
t2_list_len = len(loaded_inverted_index[t2][1])
not_t2_list = [ele for ele in files if ele not in t2_list]

if(len(t1_list) < len(not_t2_list)):
    t1_and_not_t2, comparisions = intersection(t1_list,not_t2_list)
else:
    t1_and_not_t2, comparisions = intersection(not_t2_list,t1_list)
print(f'List of files containing {t1} AND NOT {t2} : {t1_and_not_t2}')
print(f'numeber of comparisions : {comparisions}')

# t1_and_not_t2 = list(set(t1_list) & set(not_t2_list))
# print(min(t1_list_len,len(not_t2_list)))
# print(sorted(t1_and_not_t2))
```

```
T1 AND NOT T2
Plese enter the first term : dissociating
please enter the second term : gas
List of files containing dissociating AND NOT gas : []
numeber of comparisions : 2364
```

T1 OR NOT T2

In [29]:
```
# T1 OR NOT T2
# t1 = 'positive'
# t2 = 'letter'
print("T1 OR NOT T2")
t1 = input("Plese enter the first term : ")
t2 = input("please enter the second term : ")
t1_list = loaded_inverted_index[t1][1]
t1_list_len = len(loaded_inverted_index[t1][1])
t2_list = loaded_inverted_index[t2][1]
t2_list_len = len(loaded_inverted_index[t2][1])
not_t2_list = [ele for ele in files if ele not in t2_list]

if(len(t1_list) < len(not_t2_list)):
    t1_or_not_t2, comparisions = union(t1_list,not_t2_list)
else:
    t1_or_not_t2, comparisions = union(not_t2_list,t1_list)
print(f'List of files containing {t1} OR NOT {t2} : {t1_or_not_t2}')
print(f'numeber of comparisions : {comparisions}')
# t1_or_not_t2 = list(set(t1_list).union(set(not_t2_list)))
# print(max(t1_list_len,len(not_t2_list)))
# print(len(t1_or_not_t2))
# print(sorted(t1_or_not_t2))
```

```
T1 OR NOT T2
Plese enter the first term : statistical
please enter the second term : made
List of files containing statistical OR NOT made : ['cranfield0002', 'cranfield0003', 'cranfield0005', 'cranfield00
06', 'cranfield0007', 'cranfield0008', 'cranfield0011', 'cranfield0012', 'cranfield0015', 'cranfield0016', 'cranfie
ld0017', 'cranfield0018', 'cranfield0019', 'cranfield0020', 'cranfield0021', 'cranfield0022', 'cranfield0023', 'cra
nfield0025', 'cranfield0026', 'cranfield0027', 'cranfield0029', 'cranfield0030', 'cranfield0031', 'cranfield0032',
'cranfield0033', 'cranfield0034', 'cranfield0035', 'cranfield0036', 'cranfield0037', 'cranfield0038', 'cranfield003
9', 'cranfield0040', 'cranfield0041', 'cranfield0042', 'cranfield0043', 'cranfield0044', 'cranfield0045', 'cranfiel
d0047', 'cranfield0048', 'cranfield0050', 'cranfield0051', 'cranfield0053', 'cranfield0054', 'cranfield0057', 'cran
field0058', 'cranfield0059', 'cranfield0060', 'cranfield0061', 'cranfield0062', 'cranfield0063', 'cranfield0065',
```

Generalised query such as T1 AND T2 OR T3 OR T4 ....

```
Enter the number of test cases : 2
Enter the input sequence : it is a positive letter
enter the comma seperated operationsAND
['positive', 'letter']
hit and
Query :  positive AND letter
Number of documents retrieved 0
Names of the documents retrieved for query : []
Number of comparisons required for query : 6
Enter the input sequence : the gas is dissociating from the machine
enter the comma seperated operationsOR,NOT AND
['gas', 'dissociating', 'machine']
hit or
Query :  gas OR dissociating NOT AND machine
Number of documents retrieved 134
Names of the documents retrieved for query : ['cranfield0018', 'cranfield0022', 'cranfield0027', 'cranfield0033', 'cr
anfield0034', 'cranfield0049', 'cranfield0068', 'cranfield0073', 'cranfield0085', 'cranfield0101', 'cranfield0108',
'cranfield0110', 'cranfield0125', 'cranfield0161', 'cranfield0166', 'cranfield0167', 'cranfield0168', 'cranfield016
9', 'cranfield0170', 'cranfield0183', 'cranfield0185', 'cranfield0190', 'cranfield0193', 'cranfield0208', 'cranfield0
213', 'cranfield0215', 'cranfield0217', 'cranfield0236', 'cranfield0237', 'cranfield0240', 'cranfield0262', 'cranfiel
d0310', 'cranfield0317', 'cranfield0318', 'cranfield0319', 'cranfield0328', 'cranfield0329', 'cranfield0332', 'cranfi
eld0337', 'cranfield0340', 'cranfield0341', 'cranfield0342', 'cranfield0343', 'cranfield0344', 'cranfield0349', 'cran
field0353', 'cranfield0355', 'cranfield0365', 'cranfield0366', 'cranfield0370', 'cranfield0375', 'cranfield0421', 'cr
anfield0427', 'cranfield0435', 'cranfield0437', 'cranfield0456', 'cranfield0480', 'cranfield0481', 'cranfield0488',
'cranfield0493', 'cranfield0494', 'cranfield0511', 'cranfield0517', 'cranfield0518', 'cranfield0529', 'cranfield053
6', 'cranfield0540', 'cranfield0541', 'cranfield0571', 'cranfield0574', 'cranfield0575', 'cranfield0576', 'cranfield0
583', 'cranfield0591', 'cranfield0595', 'cranfield0625', 'cranfield0629', 'cranfield0645', 'cranfield0646', 'cranfiel
d0656', 'cranfield0665', 'cranfield0767', 'cranfield0868', 'cranfield0942', 'cranfield0967', 'cranfield0974', 'cranfi
eld0975', 'cranfield0998', 'cranfield1003', 'cranfield1006', 'cranfield1007', 'cranfield1009', 'cranfield1072', 'cran
field1077', 'cranfield1105', 'cranfield1139', 'cranfield1143', 'cranfield1147', 'cranfield1160', 'cranfield1180', 'cr
anfield1185', 'cranfield1189', 'cranfield1190', 'cranfield1199', 'cranfield1203', 'cranfield1204', 'cranfield1221',
'cranfield1226', 'cranfield1230', 'cranfield1237', 'cranfield1244', 'cranfield1245', 'cranfield1248', 'cranfield124
9', 'cranfield1252', 'cranfield1255', 'cranfield1264', 'cranfield1268', 'cranfield1274', 'cranfield1286', 'cranfield1
288', 'cranfield1295', 'cranfield1296', 'cranfield1310', 'cranfield1312', 'cranfield1313', 'cranfield1315', 'cranfiel
d1318', 'cranfield1319', 'cranfield1335', 'cranfield1347', 'cranfield1373', 'cranfield1374', 'cranfield1375']
Number of comparisons required for query : 235
```

Q3 -

(i) Bigram inverted index [30 Marks]

1. Create a bigram inverted index (from scratch; No library allowed) of the dataset obtained from Q1.
2. Use Python’s pickle module to save and load the bigram inverted index.

In [105]:
```
positional_index = {}

for file_index in range(len(file_list)):

    f = open(f'{folder_name}/{file_list[file_index]}', 'r')
    all_words = f.read().split()

    file_number = file_index + 1

    unique_words=list(set(all_words))

    for unique_word in unique_words:
        no_of_occurences,indices = find_occurences(all_words,unique_word)

        if unique_word in positional_index.keys():
            positional_index[unique_word][0] = positional_index[unique_word][0] + 1
            document_level_data_structure = positional_index[unique_word][1]
            document_level_data_structure[file_number] = [no_of_occurences,indices]
        else:
            positional_index[unique_word] = [1,{file_number:[no_of_occurences,indices]}]
```

In [106]:
```
import pickle

pickle.dump(positional_index,open('positional_index.pkl','wb'))
positional_index = pickle.load(open('positional_index.pkl','rb'))
```

We have iterated over all the files and extracted each word. For each word, we have created a new entry if the word is not in the positional index data structure and if the entry is available, we have added that file in the positional index data structure.

Final positional index -

```
In [117]: for i in positional_index:
              print(i, positional_index[i], "\n\n")
```

```
agree [32, {1: [1, [65]], 14: [1, [97]], 72: [1, [79]], 165: [2, [80, 92]], 177: [1, [108]], 233: [1, [53]], 289: [1, [11
2]], 318: [1, [72]], 330: [1, [42]], 363: [1, [62]], 383: [1, [63]], 417: [1, [216]], 648: [1, [50]], 666: [1, [74]], 68
8: [1, [79]], 728: [1, [34]], 830: [1, [96]], 869: [1, [98]], 887: [1, [61]], 903: [1, [76]], 923: [1, [61]], 950: [1, [5
4]], 959: [1, [55]], 996: [1, [158]], 1118: [1, [40]], 1185: [1, [81]], 1191: [1, [72]], 1199: [1, [107]], 1204: [1, [18
3]], 1269: [1, [25]], 1303: [1, [95]], 1367: [1, [39]]}]


comparative [6, {1: [1, [39]], 77: [1, [1]], 173: [1, [83]], 503: [1, [24]], 905: [2, [1, 5]], 1336: [1, [70]]}]


effect [269, {1: [1, [56]], 4: [1, [42]], 7: [3, [1, 23, 102]], 8: [3, [2, 11, 21]], 9: [1, [91]], 11: [1, [36]], 21: [1,
[8]], 23: [3, [37, 57, 69]], 26: [2, [3, 14]], 29: [1, [46]], 36: [1, [70]], 39: [1, [43]], 42: [3, [2, 18, 27]], 43: [2,
[4, 76]], 44: [2, [87, 118]], 53: [1, [80]], 55: [1, [15]], 58: [1, [92]], 60: [1, [70]], 61: [1, [35]], 73: [1, [58]], 7
9: [1, [61]], 80: [4, [1, 29, 41, 137]], 81: [1, [45]], 87: [1, [31]], 94: [5, [3, 14, 23, 181, 236]], 96: [3, [4, 14, 12
5]], 105: [2, [53, 71]], 117: [1, [71]], 123: [2, [34, 56]], 125: [2, [73, 80]], 129: [2, [114, 130]], 131: [1, [110]], 1
33: [1, [72]], 145: [1, [34]], 156: [3, [1, 11, 59]], 163: [1, [199]], 168: [2, [62, 130]], 170: [1, [71]], 173: [2, [1,
146]], 174: [1, [129]], 176: [1, [13]], 179: [2, [130, 134]], 182: [2, [1, 19]], 186: [1, [80]], 187: [1, [8]], 188: [2,
[114, 118]], 189: [1, [134]], 197: [4, [7, 38, 133, 149]], 201: [2, [61, 73]], 208: [3, [2, 21, 58]], 211: [1, [1]], 217:
[1, [29]], 220: [1, [25]], 230: [1, [81]], 233: [1, [27]], 239: [2, [69, 92]], 240: [1, [125]], 245: [3, [2, 15, 42]], 24
```

Querying based on positional index

(ii) Positional index [35 Marks]
1. Create a positional index (from scratch; No library allowed) of the dataset obtained from Q1.
2. Use Python's pickle module to save and load the positional index.

```
In [113]: bigram_index = {}


for file in file_list:

    s = []

    f = open(f'{folder_name}/{file}','r')
    file_contents = f.read()

    words = list(file_contents.split(" "))

    for i in range(len(words)-1):
        s.append(words[i]+" "+words[i+1])

    for i in s:
        if i in bigram_index:
            val = bigram_index[i]
            val[0]+=1
            val[1].append(file)
            bigram_index[i] = val

        else:
            bigram_index[i] = [1, [file]]

    f.close()
```

This is the code used for creating the bigram inverted index.

Final bigram inverted index -

```
    f.close()


for i in bigram_index:
    print(i, bigram_index[i], "\n\n")
```

```
slipstream experimental [2, ['cranfield0001', 'cranfield0484']]


experimental study [17, ['cranfield0001', 'cranfield0074', 'cranfield0256', 'cranfield0256', 'cranfield0334', 'cranfield0
420', 'cranfield0464', 'cranfield0544', 'cranfield0549', 'cranfield0760', 'cranfield0772', 'cranfield0801', 'cranfield084
7', 'cranfield0911', 'cranfield1019', 'cranfield1167', 'cranfield1264']]


study wing [1, ['cranfield0001']]


wing propeller [1, ['cranfield0001']]


propeller slipstream [7, ['cranfield0001', 'cranfield0453', 'cranfield0453', 'cranfield0453', 'cranfield1064', 'cranfield
1094', 'cranfield1164']]


slipstream made [1, ['cranfield0001']]
```

(iii) Compare and comment on your results using (i) and (ii). [5 Marks]

```
In [118]: print(len(positional_index))
          print(len(bigram_index))
```

```
8996
85209
```

As we can see here, there are 8996 unique words in the case of positional index. This is because if the word is occuring in multiple files, those will be updated.

But in the case of bigram inverted index, there are 85209 entries. this makes sense because we look for all the bigrams. there are more number of bigrams existing in the file

1. Input Format:

a. The first line contains N denoting the number of queries to execute
b. The next N lines contain phrase queries

2. Output Format:

a. 4N lines consisting of the results in the following format:
i. Number of documents retrieved for query X using bigram inverted index
ii. Names of documents retrieved for query X using bigram inverted index
iii. Number of documents retrieved for query X using positional index
iv. Names of documents retrieved for query X using positional index 3.

Perform preprocessing steps (from Q1) on the input sequence as well. Assume the length of the input sequence to be <=5. 4.

Code -

In [115]:
```
n = int(input("Enter number of phrase queries to execute:"))
print(n, "\n")
query_no=1
while n>0:
    input_query = input("Enter the input phrase query: ")

    input_tokens = preprocess_input_query(input_query)

    doc_indices_of_tokens = []
    positional_indices_of_tokens = []

    bigram_query_docs = []

    for i in range(len(input_tokens)-1):
        word = input_tokens[i]+" "+input_tokens[i+1]

        intersection = []
        if word in bigram_index:
            #print(word, " occur in ", bigram_index[word][0], " documents and the documents are ", bigram_index[word][1], "\
            intersection.append(bigram_index[word][1])

    print("Number of documents retrieved for query ", 3-n, " using bigram inverted index: ", len(intersection))
    if len(intersection)!=0:
        print("Names of documents retrieved for query ", 3-n, " using bigram inverted index: ", list(set.intersection(*map(s
    else:
        print("Names of documents retrieved for query ", 3-n, " using bigram inverted index: ", [])
```

```
    common_documents = intersection_of_multiple_list(doc_indices_of_tokens)

    relevant_doc_ids = {}

    for doc in common_documents:

        resultant_list = []
        for i in range(len(positional_indices_of_tokens)):
            if resultant_list==[]:
                resultant_list = positional_indices_of_tokens[i][doc][1]
            else:
                resultant_list = doc_position_list_of_tokens(resultant_list,positional_indices_of_tokens[i][doc][1])

            if len(resultant_list)==0:
                continue

        if len(resultant_list)>0:
            relevant_doc_ids[doc] = len(resultant_list)


    relevant_doc_ids = dict(sorted(relevant_doc_ids.items(),reverse=True,key=lambda x:x[1]))
    print_postional_index_query_results(relevant_doc_ids,query_no)

    query_no+=1

    n-=1
```

Output -

```
    n-=1
```

```
Enter number of phrase queries to execute:2
2

Enter the input phrase query: dissociating in a gas
Number of documents retrieved for query  1  using bigram inverted index:  1
Names of documents retrieved for query  1  using bigram inverted index:  ['cranfield0167', 'cranfield0517', 'cranfield129
5', 'cranfield0166', 'cranfield0656', 'cranfield0575', 'cranfield0110', 'cranfield0317']

Number of documents retrieved for query 1 query using positional index is 8
Names of documents retrieved for query 1 query using positional index is ['cranfield0110', 'cranfield0166', 'cranfield016
7', 'cranfield0317', 'cranfield0517', 'cranfield0575', 'cranfield0656', 'cranfield1295']


Enter the input phrase query: dissociating in a gas for lucid
Number of documents retrieved for query  2  using bigram inverted index:  1
Names of documents retrieved for query  2  using bigram inverted index:  ['cranfield0110']

Number of documents retrieved for query 2 query using positional index is 1
Names of documents retrieved for query 2 query using positional index is ['cranfield0110']
```