Q1. Data Preprocessing

(i) Preprocessing
Perform the following preprocessing steps on each of the text files in the dataset linked above.
Approach:
It is using nltk library for preprocessing it take Input text file and save each after preprocess.

Methodologies :
1) Lowercase the text

```
# Lowercase the text
text = text.lower()
```

The above code snippet makes the text to lowercase
Results :
Contents of the file before and after this step:

```
Original Text:
Loving these vintage springs on my vintage strat. They have a good tension and great stability. If you are floating your bridge and want the most out of your springs than these are the way to go.
--------------------
Text after lowercasing:
loving these vintage springs on my vintage strat. they have a good tension and great stability. if you are floating your bridge and want the most out of your springs than these are the way to go.
--------------------
```

2) Perform tokenization

```
# Tokenization with WordPunctTokenizer
tokenizer = WordPunctTokenizer()
tokens = tokenizer.tokenize(text)

print("Tokens after tokenization:")
print(tokens)
print("--------------------")
```

The above code Snippet,performs tokenisation
Results:
Contents of the file before and after this step:

```
Original Text:
Loving these vintage springs on my vintage strat. They have a good tension and great stability. If you are floating your bridge and want the most out of your springs than these are the way to go.
--------------------
```

```
Tokens after tokenization:
['loving', 'these', 'vintage', 'springs', 'on', 'my', 'vintage', 'strat', '.', 'they', 'have', 'a', 'good', 'tension', 'and', 'great', 'stability', '.', 'if', 'you', 'are', 'floating', 'your', 'bridge', 'and', '
want', 'the', 'most', 'out', 'of', 'your', 'springs', 'than', 'these', 'are', 'the', 'way', 'to', 'go', '.']
```

3) Remove stopwords

```
# Remove stopwords
stop_words = set(stopwords.words('english'))
tokens = [word for word in tokens if word.lower() not in stop_words]

print("Tokens after removing stopwords:")
print(tokens)
print("--------------------")
```

The above code snippet removes stopwords

Results:

Contents of the file before and after this step

```
Original Text:
Loving these vintage springs on my vintage strat. They have a good tension and great stability. If you are floating your bridge and want the most out of your springs than these are the way to go.
--------------------
```

```
Tokens after removing stopwords:
['loving', 'vintage', 'springs', 'vintage', 'strat', '.', 'good', 'tension', 'great', 'stability', '.', 'floating', 'bridge', 'want', 'springs', 'way', 'go', '.']
--------------------
```

4) Remove punctuations

```
# Remove punctuations
tokens = [word for word in tokens if word not in string.punctuation]

print("Tokens after removing punctuations:")
print(tokens)
print("--------------------")
```

The above code snippet removes punctuations

Results:

Contents of the file before and after this step

```
Original Text:
Loving these vintage springs on my vintage strat. They have a good tension and great stability. If you are floating your bridge and want the most out of your springs than these are the way to go.
--------------------
```

```
Tokens after removing punctuations:
['loving', 'vintage', 'springs', 'vintage', 'strat', 'good', 'tension', 'great', 'stability', 'floating', 'bridge', 'want', 'springs', 'way', 'go']
--------------------
```

5) Remove blank space tokens

```
# Remove blank space tokens
tokens = [word for word in tokens if word.strip()]

print("Tokens after removing blank spaces:")
print(tokens)
print("--------------------")
```

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

Results:

Contents of the file before and after this step

```
Original Text:
Loving these vintage springs on my vintage strat. They have a good tension and great stability. If you are floating your bridge and want the most out of your springs than these are the way to go.
--------------------
```

```
Tokens after removing blank spaces:
['loving', 'vintage', 'springs', 'vintage', 'strat', 'good', 'tension', 'great', 'stability', 'floating', 'bridge', 'want', 'springs', 'way', 'go']
--------------------
```

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.

```
for filename in os.listdir(folder_path):
    file_path = os.path.join(folder_path, filename)
    with open(file_path, 'r') as file:
        text = file.read()
        words = text.split()
        doc_id = filename  # Assuming filename serves as the document identifier
```

Implementing the unigram inverted index from scratch

```
            for word in words:
                if word in inverted_index:
                    if doc_id not in inverted_index[word]:
                        inverted_index[word].append(doc_id)
                else:
                    inverted_index[word] = [doc_id]
    inverted_index = {word: sorted(doc_ids) for word, doc_ids in inverted_index.items()}
    return inverted_index

# Example usage:
folder_path = "D:\\python\\IR\\ans1"
unigram_inverted_index = create_inverted_index_from_folder(folder_path)
```

Output:

```
yard: ['file394.txt', 'file706.txt']
yas: ['file82.txt']
yeah: ['file679.txt', 'file978.txt']
year: ['file11.txt', 'file163.txt', 'file351.txt', 'file42.txt', 'file429.txt', 'file483.txt', 'file487.txt', 'file490.txt', 'file522.txt', 'file532.txt', 'file542.txt', 'file573.txt', 'file588.txt', 'file600.tx
t', 'file665.txt', 'file689.txt', 'file729.txt', 'file741.txt', 'file875.txt', 'file878.txt', 'file935.txt', 'file968.txt']
years: ['file116.txt', 'file127.txt', 'file150.txt', 'file155.txt', 'file157.txt', 'file160.txt', 'file177.txt', 'file196.txt', 'file214.txt', 'file240.txt', 'file265.txt', 'file270.txt', 'file28.txt', 'file29.t
xt', 'file327.txt', 'file346.txt', 'file35.txt', 'file375.txt', 'file379.txt', 'file382.txt', 'file400.txt', 'file404.txt', 'file412.txt', 'file532.txt', 'file534.txt', 'file557.txt', 'file56.txt', 'file591.txt'
, 'file596.txt', 'file654.txt', 'file660.txt', 'file672.txt', 'file684.txt', 'file686.txt', 'file691.txt', 'file71.txt', 'file722.txt', 'file736.txt', 'file759.txt', 'file770.txt', 'file839.txt', 'file851.txt',
'file858.txt', 'file871.txt', 'file896.txt', 'file898.txt', 'file928.txt', 'file955.txt']
yellow: ['file228.txt', 'file429.txt', 'file712.txt']
yellowish: ['file803.txt']
yep: ['file413.txt', 'file721.txt']
yes: ['file397.txt', 'file413.txt', 'file514.txt', 'file549.txt', 'file564.txt', 'file638.txt', 'file692.txt', 'file789.txt', 'file840.txt', 'file845.txt', 'file898.txt']
yesterday: ['file404.txt', 'file55.txt', 'file77.txt']
yet: ['file111.txt', 'file222.txt', 'file281.txt', 'file3.txt', 'file304.txt', 'file322.txt', 'file328.txt', 'file349.txt', 'file400.txt', 'file404.txt', 'file405.txt', 'file409.txt', 'file430.txt', 'file501.txt
', 'file513.txt', 'file559.txt', 'file609.txt', 'file716.txt', 'file745.txt', 'file838.txt', 'file964.txt']
yeti: ['file113.txt', 'file320.txt', 'file449.txt']
yhe: ['file221.txt']
yield: ['file340.txt']
ymtake: ['file83.txt']
yngwie: ['file995.txt']
yoke: ['file824.txt']
youll: ['file399.txt', 'file422.txt', 'file694.txt']
```

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

```
def save_inverted_index(index, filename):
    with open(filename, 'wb') as f:
        pickle.dump(index, f)
```

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

```python
def load_inverted_index(filename):
    with open(filename, 'rb') as f:
        index = pickle.load(f)
    return index
```

Provide support for the following operations:

a. T1 AND T2

```
C:\Users\shinchan\Desktop\IR\Assigment>python Q2_3.py
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\shinchan\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\shinchan\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Enter the number of queries: 1
Enter the query: greatest benefit
Enter the operations separated by comma: AND
Query after preprocessing: greatest AND benefit
Number of documents retrieved for query: 4
Names of the documents retrieved for query: ['file3.txt.txt', 'file320.txt.txt', 'file766.txt.txt', 'file96.txt.txt']
```

b. T1 OR T2

```
C:\Users\shinchan\Desktop\IR\Assigment>python Q2_3.py
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\shinchan\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\shinchan\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Enter the number of queries: 1
Enter the query: greatest benefit
Enter the operations separated by comma: OR
Query after preprocessing: greatest OR benefit
Number of documents retrieved for query: 4
Names of the documents retrieved for query: ['file3.txt.txt', 'file320.txt.txt', 'file766.txt.txt', 'file96.txt.txt']
```

c. T1 AND NOT T2

```
C:\Users\shinchan\Desktop\IR\Assigment>python Q2_3.py
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\shinchan\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\shinchan\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Enter the number of queries: 1
Enter the query: greatest benefit
Enter the operations separated by comma: AND NOT
Query after preprocessing: greatest AND NOT benefit
Number of documents retrieved for query: 4
Names of the documents retrieved for query: ['file3.txt.txt', 'file320.txt.txt', 'file766.txt.txt', 'file96.txt.txt']
```

d. T1 OR NOT T2

```
C:\Users\shinchan\Desktop\IR\Assigment>python Q2_3.py
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\shinchan\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\shinchan\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Enter the number of queries: 1
Enter the query: greatest benefit
Enter the operations separated by comma: OR NOT
Query after preprocessing: greatest OR NOT benefit
Number of documents retrieved for query: 4
Names of the documents retrieved for query: ['file3.txt.txt', 'file320.txt.txt', 'file766.txt.txt', 'file96.txt.txt']
```

```python
import os
import pickle
import nltk
from nltk.tokenize import WordPunctTokenizer
from nltk.corpus import stopwords
import string

# Download NLTK resources
nltk.download('punkt')
nltk.download('stopwords')

def preprocess_text(text):
    """Preprocess the input text."""
    # Lowercase the text
    text = text.lower()
    # Tokenization with WordPunctTokenizer
    tokenizer = WordPunctTokenizer()
    tokens = tokenizer.tokenize(text)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word.lower() not in stop_words]
    # Remove punctuations
    tokens = [word for word in tokens if word not in string.punctuation]
    # Remove blank space tokens
    tokens = [word for word in tokens if word.strip()]
    # Join tokens back into a string
    preprocessed_text = ' '.join(tokens)
    return preprocessed_text

def preprocess_query(query):
    """Preprocess the input query."""
```

```
    return preprocess_text(query)

def load_inverted_index(filename):
    """Load the inverted index from the file."""
    with open(filename, 'rb') as f:
        index = pickle.load(f)
    return index

def intersect_posting_lists(posting_list1, posting_list2):
    """Compute the intersection of two posting lists."""
    return sorted(list(set(posting_list1).intersection(posting_list2)))

def union_posting_lists(posting_list1, posting_list2):
    """Compute the union of two posting lists."""
    return sorted(list(set(posting_list1).union(posting_list2)))

def subtract_posting_lists(posting_list1, posting_list2):
    """Compute the subtraction of posting list2 from posting list1."""
    return sorted(list(set(posting_list1).difference(posting_list2)))

def perform_operation(operation, result, next_term, inverted_index):
    """Perform the specified operation on the given terms."""
    posting_list = search_query(next_term, inverted_index)
    if operation == 'AND':
        result = intersect_posting_lists(result, posting_list)
    elif operation == 'OR':
        result = union_posting_lists(result, posting_list)
    elif operation == 'AND NOT':
        result = subtract_posting_lists(result, posting_list)
    elif operation == 'OR NOT':
        result = union_posting_lists(result,
subtract_posting_lists(inverted_index.keys(), posting_list))
    return result

def process_queries(N, queries, operations, inverted_index):
    """Process the list of queries and return results."""
    results = []
    for i in range(N):
        query = preprocess_query(queries[i])
        terms = query.split()
```

```
        ops = operations[i]
        result = search_query(terms[0], inverted_index)
        j = 1
        op_index = 0
        while j < len(terms):
            if j < len(ops):
                operation = ops[op_index]
                next_term = terms[j]
                result = perform_operation(operation, result, next_term,
inverted_index)
                j += 1
                op_index += 1
            else:
                break
        results.append(result)
    return results

def search_query(term, inverted_index):
    """Retrieve the posting list for the given term."""
    return inverted_index.get(term, [])

def print_output(query, result, operations):
    """Print the output in the specified format."""
    preprocessed_terms = query.split()
    preprocessed_query = " AND ".join([f"{preprocessed_terms[i]}" for i in
range(len(preprocessed_terms) - 1) if preprocessed_terms[i] not in
stopwords.words('english')])
    preprocessed_query += f" {operations[-1]} {preprocessed_terms[-1]}"  #
Use the last operation with the last term
    print(f"Query after preprocessing: {preprocessed_query}")
    print(f"Number of documents retrieved for query: {len(result)}")
    print(f"Names of the documents retrieved for query: {[f'{x}.txt' for x
in result]}")
    print()


# Example usage:
index_file = 'inverted_index.pkl'

# Load the inverted index from the file
```

```
loaded_inverted_index = load_inverted_index(index_file)

# Input
N = int(input("Enter the number of queries: "))
queries = []
operations = []
for i in range(N):
    query = input("Enter the query: ")
    ops = input("Enter the operations separated by comma: ")
    queries.append(query)
    operations.append(ops.split(','))

# Process queries
results = process_queries(N, queries, operations, loaded_inverted_index)

# Output
# Output
for i in range(N):
    print_output(queries[i], results[i], operations[i])
```

4. Queries should be generalized i.e., you should provide support for queries like T1 AND T2 OR T3 AND T4 ....

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\shinchan\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\shinchan\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Enter the number of queries: 2
Enter the query: fun to play with small and easy to carry around
Enter the operations separated by comma: AND,OR,AND,AND,OR
Enter the query: Split time as my primary
Enter the operations separated by comma: AND,NOTAND,OR
Query after preprocessing: fun AND play AND small AND easy AND carry OR around
Number of documents retrieved for query: 1
Names of the documents retrieved for query: ['file6.txt.txt']

Query after preprocessing: Split AND time OR primary
Number of documents retrieved for query: 2
Names of the documents retrieved for query: ['file347.txt.txt', 'file5.txt.txt']
```

Q3 -

1. Create a positional index (from scratch; No library allowed) of the dataset obtained from Q1.

```
def create_positional_index(folder_path):
    positional_index = {}

    # Iterate over all files in the folder
    for file_name in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file_name)
        if os.path.isfile(file_path):  # Ensure it's a file, not a directory
            with open(file_path, 'r') as file:
                content = file.read()
                terms = content.split()
                doc_id = file_name.split('.')[0]  # Extract document ID from file name

                for position, term in enumerate(terms, start=1):
                    if term not in positional_index:
                        positional_index[term] = []
                    positional_index[term].append((doc_id, position))

    return positional_index
```

2. Use Python's pickle module to save and load the positional index.

```
def save_positional_index(positional_index, file_name):
    with open(file_name, 'wb') as file:
        pickle.dump(positional_index, file)

def load_positional_index(file_name):
    with open(file_name, 'rb') as file:
        positional_index = pickle.load(file)
    return positional_index
```

3. Input Format:

a. The first line contains N denoting the number of queries to execute
b. The next N lines contain phrase queries
4. Output Format:
a. 2N lines consisting of the results in the following format:
i. Number of documents retrieved for query X using positional index
ii. Names of documents retrieved for query X using positional index
5. Perform preprocessing steps (from Q1) on the input sequence as well. Assume the length of the input sequence to be <=5.

```
import os
```

```
import string
import pickle
from nltk.tokenize import WordPunctTokenizer
from nltk.corpus import stopwords

def preprocess_text(text):
    """Preprocess the input text."""
    # Lowercase the text
    text = text.lower()
    # Tokenization with WordPunctTokenizer
    tokenizer = WordPunctTokenizer()
    tokens = tokenizer.tokenize(text)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word.lower() not in stop_words]
    # Remove punctuations
    tokens = [word for word in tokens if word not in string.punctuation]
    # Remove blank space tokens
    tokens = [word for word in tokens if word.strip()]
    # Join tokens back into a string
    preprocessed_text = ' '.join(tokens)
    return preprocessed_text

def load_positional_index(index_file):
    with open(index_file, 'rb') as f:
        positional_index = pickle.load(f)
    return positional_index

def search_query(query, positional_index):
    # Implement your search algorithm here using the positional index
    # This is a placeholder function; you need to replace it with your
actual search logic
    unique_results = set()
    query_terms = query.split()  # Split query into terms
    for term in query_terms:
        if term in positional_index:
            for doc_id, _ in positional_index[term]:
                unique_results.add(doc_id)
    return list(unique_results)
```

```python
def main():
    num_queries = int(input("Enter the number of queries: "))
    queries = []

    # Take input for queries
    for _ in range(num_queries):
        query = input("Enter query: ").strip()
        queries.append(query)

    # Load positional index from PKL file
    index_file = "D:\\python\\IR\\positinal.pkl"
    if not os.path.isfile(index_file):
        print("Positional index file not found.")
        return
    positional_index = load_positional_index(index_file)

    # Process queries
    for i, query in enumerate(queries, 1):
        preprocessed_query = preprocess_text(query)
        result = search_query(preprocessed_query, positional_index)
        print(f"Number of documents retrieved for query {i} using
positional index: {len(result)}")
        print(f"Names of documents retrieved for query {i} using
positional index: {[f'{doc_id}.txt' for doc_id in result]}")

if __name__ == "__main__":
    main()
```

```
Enter the number of queries: 2
Enter query: fun play small easy carry around
Enter query: split time as my primary
Number of documents retrieved for query 1 using positional index: 297
Names of documents retrieved for query 1 using positional index: ['file712.txt', 'file981.txt', 'file762.txt', 'file11.txt', 'file964.txt', 'file977.txt', 'file968.txt', 'file813.txt', 'file635.txt', 'file957.tx
t', 'file105.txt', 'file541.txt', 'file37.txt', 'file618.txt', 'file56.txt', 'file366.txt', 'file729.txt', 'file244.txt', 'file803.txt', 'file467.txt', 'file539.txt', 'file502.txt', 'file821.txt', 'file347.txt',
 'file566.txt', 'file923.txt', 'file187.txt', 'file494.txt', 'file196.txt', 'file983.txt', 'file790.txt', 'file710.txt', 'file707.txt', 'file764.txt', 'file190.txt', 'file913.txt', 'file18.txt', 'file934.txt',
file335.txt', 'file404.txt', 'file126.txt', 'file349.txt', 'file915.txt', 'file815.txt', 'file549.txt', 'file44.txt', 'file109.txt', 'file70.txt', 'file503.txt', 'file903.txt', 'file544.txt', 'file407.txt', 'fil
e942.txt', 'file559.txt', 'file786.txt', 'file170.txt', 'file893.txt', 'file248.txt', 'file830.txt', 'file961.txt', 'file665.txt', 'file524.txt', 'file804.txt', 'file847.txt', 'file461.txt', 'file254.txt', 'file
245.txt', 'file663.txt', 'file761.txt', 'file860.txt', 'file624.txt', 'file590.txt', 'file74.txt', 'file926.txt', 'file490.txt', 'file182.txt', 'file28.txt', 'file401.txt', 'file512.txt', 'file54.txt', 'file183.
txt', 'file193.txt', 'file218.txt', 'file802.txt', 'file901.txt', 'file301.txt', 'file78.txt', 'file89.txt', 'file713.txt', 'file616.txt', 'file342.txt', 'file412.txt', 'file45.txt', 'file415.txt', 'file294.txt'
, 'file368.txt', 'file100.txt', 'file824.txt', 'file216.txt', 'file887.txt', 'file57.txt', 'file982.txt', 'file470.txt', 'file29.txt', 'file825.txt', 'file858.txt', 'file588.txt', 'file575.txt', 'file3.txt', 'fi
le489.txt', 'file2.txt', 'file31.txt', 'file253.txt', 'file951.txt', 'file659.txt', 'file157.txt', 'file147.txt', 'file571.txt', 'file974.txt', 'file777.txt', 'file758.txt', 'file880.txt', 'file71.txt', 'file536
.txt', 'file679.txt', 'file711.txt', 'file22.txt', 'file602.txt', 'file891.txt', 'file939.txt', 'file854.txt', 'file279.txt', 'file941.txt', 'file186.txt', 'file384.txt', 'file738.txt', 'file309.txt', 'file47.tx
t', 'file35.txt', 'file884.txt', 'file631.txt', 'file949.txt', 'file8.txt', 'file333.txt', 'file770.txt', 'file5.txt', 'file986.txt', 'file998.txt', 'file272.txt', 'file952.txt', 'file276.txt', 'file64.txt', 'fi
le454.txt', 'file916.txt', 'file731.txt', 'file505.txt', 'file380.txt', 'file112.txt', 'file579.txt', 'file214.txt', 'file282.txt', 'file889.txt', 'file169.txt', 'file871.txt', 'file542.txt', 'file686.txt', 'fil
e904.txt', 'file268.txt', 'file992.txt', 'file748.txt', 'file425.txt', 'file772.txt', 'file736.txt', 'file567.txt', 'file595.txt', 'file839.txt', 'file677.txt', 'file746.txt', 'file317.txt', 'file465.txt', 'file
634.txt', 'file648.txt', 'file789.txt', 'file468.txt', 'file286.txt', 'file325.txt', 'file469.txt', 'file623.txt', 'file166.txt', 'file537.txt', 'file523.txt', 'file921.txt', 'file838.txt', 'file124.txt', 'file4
40.txt', 'file103.txt', 'file823.txt', 'file598.txt', 'file718.txt', 'file263.txt', 'file256.txt', 'file14.txt', 'file993.txt', 'file432.txt', 'file850.txt', 'file576.txt', 'file669.txt', 'file143.txt', 'file890
.txt', 'file556.txt', 'file450.txt', 'file861.txt', 'file592.txt', 'file382.txt', 'file845.txt', 'file343.txt', 'file684.txt', 'file390.txt', 'file994.txt', 'file6.txt', 'file351.txt', 'file643.txt', 'file849.tx
t', 'file345.txt', 'file569.txt', 'file477.txt', 'file400.txt', 'file833.txt', 'file610.txt', 'file750.txt', 'file886.txt', 'file422.txt', 'file339.txt', 'file211.txt', 'file558.txt', 'file174.txt', 'file582.txt
', 'file521.txt', 'file302.txt', 'file424.txt', 'file67.txt', 'file911.txt', 'file672.txt', 'file774.txt', 'file819.txt', 'file189.txt', 'file307.txt', 'file363.txt', 'file995.txt', 'file19.txt', 'file361.txt',
'file608.txt', 'file769.txt', 'file281.txt', 'file927.txt', 'file298.txt', 'file106.txt', 'file859.txt', 'file353.txt', 'file316.txt', 'file689.txt', 'file922.txt', 'file603.txt', 'file172.txt', 'file388.txt',
file972.txt', 'file118.txt', 'file653.txt', 'file766.txt', 'file159.txt', 'file387.txt', 'file208.txt', 'file509.txt', 'file163.txt', 'file464.txt', 'file843.txt', 'file406.txt', 'file848.txt', 'file360.txt',
ile396.txt', 'file417.txt', 'file519.txt', 'file578.txt', 'file102.txt', 'file326.txt', 'file247.txt', 'file754.txt', 'file234.txt', 'file953.txt', 'file796.txt', 'file458.txt', 'file108.txt', 'file43.txt', 'fil
e257.txt', 'file488.txt', 'file753.txt', 'file73.txt']
Number of documents retrieved for query 2 using positional index: 93
Names of documents retrieved for query 2 using positional index: ['file373.txt', 'file11.txt', 'file65.txt', 'file525.txt', 'file467.txt', 'file573.txt', 'file502.txt', 'file821.txt', 'file347.txt', 'file375.txt
', 'file790.txt', 'file404.txt', 'file160.txt', 'file41.txt', 'file44.txt', 'file413.txt', 'file248.txt', 'file830.txt', 'file760.txt', 'file154.txt', 'file332.txt', 'file625.txt', 'file245.txt', 'file74.txt',
file611.txt', 'file171.txt', 'file324.txt', 'file627.txt', 'file435.txt', 'file359.txt', 'file844.txt', 'file912.txt', 'file89.txt', 'file45.txt', 'file68.txt', 'file29.txt', 'file534.txt', 'file880.txt', 'file9
74.txt', 'file71.txt', 'file679.txt', 'file711.txt', 'file290.txt', 'file666.txt', 'file920.txt', 'file308.txt', 'file47.txt', 'file25.txt', 'file329.txt', 'file5.txt', 'file735.txt', 'file910.txt', 'file686.txt
', 'file638.txt', 'file992.txt', 'file836.txt', 'file677.txt', 'file468.txt', 'file325.txt', 'file66.txt', 'file767.txt', 'file706.txt', 'file265.txt', 'file781.txt', 'file718.txt', 'file49.txt', 'file633.txt',
'file658.txt', 'file143.txt', 'file684.txt', 'file973.txt', 'file17.txt', 'file95.txt', 'file252.txt', 'file365.txt', 'file778.txt', 'file307.txt', 'file647.txt', 'file927.txt', 'file859.txt', 'file273.txt', 'fi
le937.txt', 'file163.txt', 'file145.txt', 'file818.txt', 'file938.txt', 'file410.txt', 'file862.txt', 'file491.txt', 'file872.txt', 'file784.txt', 'file257.txt', 'file753.txt']

D:\python\IR>python 01.py
```