Q1. Data Preprocessing [20 Marks]

1. Perform the following preprocessing steps on each of the text files in the dataset linked Above.

Approach:
The python code takes all the text files and uses libraries like nltk to perform all the preprocessing operations on the files and is storing all the preprocessed files in one folder.

Methodologies:

```
import os
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):
    print("Original Text:")
    print(text)
    print("--------------------")

    # Lowercase the text
    print("Text after lowercasing:")
    text = text.lower()
    print(text)
    print("--------------------")

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

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

    # 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("--------------------")

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

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

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

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

    # Join tokens back into a string
    preprocessed_text = ' '.join(tokens)

    print("Text after joining tokens:")
    print(preprocessed_text)
    print("--------------------")

    return preprocessed_text

def preprocess_files(input_folder, output_folder, num_files=5):
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    processed_files = 0
    for filename in os.listdir(input_folder):
        if filename.endswith(".txt"):
            input_file_path = os.path.join(input_folder, filename)
            output_file_path = os.path.join(output_folder, filename)
```

```
            with open(input_file_path, 'r') as input_file:
                text = input_file.read()
                print(f"File: {filename}")
                preprocess_text(text)
                print("--------------------")

            processed_files += 1
            if processed_files >= num_files:
                break

# Provide the folder paths for input and output files
input_folder = "C:\\Users\\sainiisha2619\\IR_A1\\text_files"
output_folder = "C:\\Users\\sainiisha2619\\IR_A1\\preprocessed_files"

preprocess_files(input_folder, output_folder)
```

a. Lowercase the text:

```
# Lowercase the text
print("Text after lowercasing:")
text = text.lower()
print(text)
print("--------------------")
```

The above code snippet reads the text from the file, and makes the text lowercase.

Contents of a file before and after this step:

```
File: file1.txt
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 o
f 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 o
f your springs than these are the way to go.
--------------------
```

b. Perform tokenization

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

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

The above code snippet performs tokenisation after lowercasing the text files.

Contents of the file before and after this step:

```
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 o
f 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', '.']
--------------------
```

c. 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 after tokenization the text files.

Contents of a file before and after this step:

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

d. 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 after removing stop words.

Contents of a file before and after this step:

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

e. 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 removes blank space tokens after removing punctuations.

Contents of a file before and after this step:

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

2. Print contents of 5 sample files before and after performing each operation. Remember to save each file after preprocessing to use the preprocessed file for the following tasks.

```python
def preprocess_files(input_folder, output_folder, num_files=5):
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    processed_files = 0
    for filename in os.listdir(input_folder):
        if filename.endswith(".txt"):
            input_file_path = os.path.join(input_folder, filename)
            output_file_path = os.path.join(output_folder, filename)

            with open(input_file_path, 'r') as input_file:
                text = input_file.read()
                print(f"File: {filename}")
                preprocess_text(text)
                print("----------------------")

            processed_files += 1
            if processed_files >= num_files:
                break
```

The above code snippet saves all the preprocessed files in one folder for further operations in the following questions.

Output for one of the files out of 5 samples outputs:

```
File: file1.txt
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 o
f 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 o
f 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', '.']
--------------------
Tokens after removing stopwords:
['loving', 'vintage', 'springs', 'vintage', 'strat', '.', 'good', 'tension', 'great', 'stability', '.', 'floating', 'bridge', 'want', 'springs', 'way',
 'go', '.']
--------------------
Tokens after removing punctuations:
['loving', 'vintage', 'springs', 'vintage', 'strat', 'good', 'tension', 'great', 'stability', 'floating', 'bridge', 'want', 'springs', 'way', 'go']
--------------------
Tokens after removing blank spaces:
['loving', 'vintage', 'springs', 'vintage', 'strat', 'good', 'tension', 'great', 'stability', 'floating', 'bridge', 'want', 'springs', 'way', 'go']
--------------------
Text after joining tokens:
loving vintage springs vintage strat good tension great stability floating bridge want springs way go
--------------------
--------------------
```

Q2. Unigram Inverted Index and Boolean Queries [40 Marks]

1. Create a unigram inverted index (from scratch; No library allowed) of the dataset obtained from Q1 (after preprocessing).

```
import os

def create_inverted_index_from_folder(folder_path):
    inverted_index = {}
    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
            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 = "C:\\Users\\sainiisha2619\\IR_A1\\preprocessed_files"
unigram_inverted_index = create_inverted_index_from_folder(folder_path)

# Print the inverted index
for word, doc_ids in sorted(unigram_inverted_index.items()):
    print(f"{word}: {doc_ids}\n")
```

This code creates a unigram index of the dataset from the folder “preprocessed_files” obtained from Q1.

Results for some sample tokes from the unigram inverted index:

```
usefull: ['file23.txt']

usefulness: ['file228.txt']

useless: ['file143.txt', 'file235.txt', 'file373.txt', 'file564.txt']

user: ['file196.txt', 'file270.txt', 'file444.txt', 'file491.txt', 'file711.txt', 'file722.txt', 'file969.txt']

uses: ['file286.txt', 'file355.txt', 'file564.txt', 'file591.txt', 'file631.txt', 'file720.txt', 'file769.txt', 'file872.txt', 'file880.txt', 'file885.
txt', 'file941.txt']

using: ['file108.txt', 'file111.txt', 'file116.txt', 'file118.txt', 'file124.txt', 'file144.txt', 'file18.txt', 'file186.txt', 'file195.txt', 'file196.
txt', 'file20.txt', 'file204.txt', 'file212.txt', 'file220.txt', 'file251.txt', 'file258.txt', 'file263.txt', 'file267.txt', 'file28.txt', 'file292.txt
', 'file299.txt', 'file30.txt', 'file33.txt', 'file331.txt', 'file344.txt', 'file382.txt', 'file384.txt', 'file404.txt', 'file434.txt', 'file438.txt',
'file440.txt', 'file454.txt', 'file465.txt', 'file467.txt', 'file468.txt', 'file469.txt', 'file511.txt', 'file516.txt', 'file523.txt', 'file524.txt', '
file533.txt', 'file539.txt', 'file557.txt', 'file564.txt', 'file569.txt', 'file591.txt', 'file61.txt', 'file627.txt', 'file631.txt', 'file632.txt', 'fi
le637.txt', 'file647.txt', 'file654.txt', 'file659.txt', 'file663.txt', 'file677.txt', 'file684.txt', 'file690.txt', 'file703.txt', 'file708.txt', 'fil
e716.txt', 'file718.txt', 'file739.txt', 'file749.txt', 'file759.txt', 'file805.txt', 'file808.txt', 'file810.txt', 'file821.txt', 'file828.txt', 'file
837.txt', 'file839.txt', 'file847.txt', 'file851.txt', 'file858.txt', 'file863.txt', 'file871.txt', 'file882.txt', 'file885.txt', 'file910.txt', 'file9
28.txt', 'file937.txt', 'file942.txt', 'file946.txt', 'file949.txt', 'file959.txt', 'file962.txt', 'file974.txt', 'file987.txt', 'file994.txt', 'file99
5.txt']

usingthis: ['file843.txt']

ussr: ['file394.txt']

usual: ['file233.txt', 'file260.txt', 'file299.txt', 'file550.txt', 'file742.txt', 'file787.txt', 'file839.txt']

usually: ['file302.txt', 'file428.txt', 'file478.txt', 'file859.txt']

usuals: ['file281.txt']

utf8: ['file12.txt', 'file24.txt', 'file269.txt', 'file465.txt', 'file490.txt', 'file608.txt', 'file609.txt', 'file855.txt', 'file928.txt', 'file949.tx
t', 'file981.txt']
```

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

```python
import os
import pickle

def create_inverted_index_from_folder(folder_path):
    inverted_index = {}
    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
            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

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

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

# Example usage:
folder_path = "C:\\Users\\sainiisha2619\\IR_A1\\preprocessed_files"
unigram_inverted_index = create_inverted_index_from_folder(folder_path)
# print(unigram_inverted_index)
# Save the inverted index to a file
save_inverted_index(unigram_inverted_index, "inverted_index.pkl")

# Load the inverted index from the file
loaded_inverted_index = load_inverted_index("inverted_index.pkl")

# Print the loaded inverted index
#for word, doc_ids in sorted(loaded_inverted_index.items()):
   # print(f"{word}: {doc_ids}")
```

This code is using pickle module to save and load the unigram inverted index created above into “inverted_index.pkl” file

3. Provide support for the following operations:
a. T1 AND T2
b. T1 OR T2
c. T1 AND NOT T2
d. T1 OR NOT T2

```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}' 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])
```

Output for all 4 operations:

```
Enter the number of queries: 4
Enter the query: greatest benefit
Enter the operations separated by comma: AND
Enter the query: greatest benefit
Enter the operations separated by comma: OR
Enter the query: greatest benefit
Enter the operations separated by comma: AND NOT
Enter the query: greatest benefit
Enter the operations separated by comma: OR NOT
Query after preprocessing: greatest AND benefit
Number of documents retrieved for query: 4
Names of the documents retrieved for query: ['file3.txt', 'file320.txt', 'file766.txt', 'file96.txt']

Query after preprocessing: greatest OR benefit
Number of documents retrieved for query: 4
Names of the documents retrieved for query: ['file3.txt', 'file320.txt', 'file766.txt', 'file96.txt']

Query after preprocessing: greatest AND NOT benefit
Number of documents retrieved for query: 4
Names of the documents retrieved for query: ['file3.txt', 'file320.txt', 'file766.txt', 'file96.txt']

Query after preprocessing: greatest OR NOT benefit
Number of documents retrieved for query: 4
Names of the documents retrieved for query: ['file3.txt', 'file320.txt', 'file766.txt', 'file96.txt']
```

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

Desired output for long generalized queries:

```
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,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']

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

Q3. Positional Index and Phrase Queries [40 Marks]

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

```python
import os

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

def main():
```

```
    folder_path = input("Enter the path to the folder containing
preprocessed files: ").strip()

    if not os.path.isdir(folder_path):
        print("Invalid folder path.")
        return

    positional_index = create_positional_index(folder_path)

    # Print the positional index
    for term, postings in positional_index.items():
        print(f"Term: {term}")
        print("Postings:")
        for posting in postings:
            print(f"Document ID: {posting[0]}, Position: {posting[1]}")
        print()

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

This code creates a positional index of the dataset from the folder “preprocessed_files” obtained from Q1.

Desired output:

```
Term: john
Postings:
Document ID: file995, Position: 61

Term: mayer
Postings:
Document ID: file995, Position: 62

Term: importantly
Postings:
Document ID: file995, Position: 69

Term: toneprint
Postings:
Document ID: file997, Position: 9

Term: artists
Postings:
Document ID: file997, Position: 15

Term: biggie
Postings:
Document ID: file998, Position: 53

Term: according
Postings:
Document ID: file999, Position: 13

Term: screenshot
Postings:
Document ID: file999, Position: 16
```

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

```python
import os
import pickle

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

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

def main():
    folder_path = input("Enter the path to the folder containing
preprocessed files: ").strip()

    if not os.path.isdir(folder_path):
        print("Invalid folder path.")
        return

    positional_index = create_positional_index(folder_path)

    save_file_name = input("Enter the filename to save the positional
index (with .pkl extension): ").strip()
    save_positional_index(positional_index, save_file_name)
    print("Positional index saved successfully.")

    # Load positional index from saved file and print
    load_file_name = input("Enter the filename to load the positional
index from: ").strip()
    loaded_index = load_positional_index(load_file_name)
```

```
    # Print the loaded positional index
    print("\nLoaded Positional Index:")
    for term, postings in loaded_index.items():
        print(f"Term: {term}")
        print("Postings:")
        for posting in postings:
            print(f"Document ID: {posting[0]}, Position: {posting[1]}")
        print()

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

This code is using pickle module to save and load the positional index created above into “positional_index.pkl” file

Desired output:

```
Term: howl
Postings:
Document ID: file911, Position: 24

Term: contends
Postings:
Document ID: file911, Position: 33

Term: brute
Postings:
Document ID: file911, Position: 34

Term: wp
Postings:
Document ID: file912, Position: 4
Document ID: file912, Position: 49
Document ID: file912, Position: 63

Term: gray
Postings:
Document ID: file912, Position: 15

Term: cast
Postings:
Document ID: file912, Position: 17

Term: awp
Postings:
Document ID: file912, Position: 22
Document ID: file912, Position: 52
Document ID: file912, Position: 72

Term: tbhey
Postings:
Document ID: file912, Position: 28
```

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
```

```python
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)
```

```
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 = "C:\\Users\\sainiisha2619\\IR_A1\\positional_index.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()
```

Desired output as per the question:

```
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: ['file893.txt', 'file994.txt', 'file208.txt', 'file748.txt', 'file440.txt', 'file602.txt', 'file387.txt
', 'file802.txt', 'file839.txt', 'file14.txt', 'file112.txt', 'file524.txt', 'file298.txt', 'file804.txt', 'file934.txt', 'file294.txt', 'file541.txt', 'file770.txt', '
file6.txt', 'file343.txt', 'file537.txt', 'file174.txt', 'file502.txt', 'file777.txt', 'file631.txt', 'file382.txt', 'file412.txt', 'file182.txt', 'file653.txt', 'file6
79.txt', 'file45.txt', 'file488.txt', 'file923.txt', 'file74.txt', 'file189.txt', 'file415.txt', 'file571.txt', 'file102.txt', 'file758.txt', 'file921.txt', 'file3.txt'
, 'file838.txt', 'file464.txt', 'file710.txt', 'file986.txt', 'file159.txt', 'file325.txt', 'file503.txt', 'file169.txt', 'file845.txt', 'file396.txt', 'file672.txt', '
file302.txt', 'file536.txt', 'file575.txt', 'file108.txt', 'file823.txt', 'file37.txt', 'file559.txt', 'file44.txt', 'file2.txt', 'file982.txt', 'file598.txt', 'file424
.txt', 'file388.txt', 'file689.txt', 'file43.txt', 'file509.txt', 'file542.txt', 'file339.txt', 'file961.txt', 'file366.txt', 'file711.txt', 'file904.txt', 'file361.txt
', 'file567.txt', 'file949.txt', 'file380.txt', 'file190.txt', 'file187.txt', 'file968.txt', 'file360.txt', 'file118.txt', 'file211.txt', 'file384.txt', 'file972.txt',
'file824.txt', 'file105.txt', 'file347.txt', 'file762.txt', 'file401.txt', 'file301.txt', 'file126.txt', 'file796.txt', 'file193.txt', 'file400.txt', 'file890.txt', 'fi
le67.txt', 'file31.txt', 'file309.txt', 'file549.txt', 'file124.txt', 'file406.txt', 'file981.txt', 'file590.txt', 'file813.txt', 'file35.txt', 'file738.txt', 'file519.
txt', 'file610.txt', 'file903.txt', 'file214.txt', 'file276.txt', 'file351.txt', 'file558.txt', 'file854.txt', 'file648.txt', 'file774.txt', 'file886.txt', 'file5.txt',
 'file100.txt', 'file830.txt', 'file861.txt', 'file147.txt', 'file461.txt', 'file18.txt', 'file953.txt', 'file754.txt', 'file731.txt', 'file643.txt', 'file579.txt', 'fi
le183.txt', 'file454.txt', 'file891.txt', 'file425.txt', 'file450.txt', 'file913.txt', 'file404.txt', 'file268.txt', 'file941.txt', 'file317.txt', 'file56.txt', 'file85
0.txt', 'file803.txt', 'file467.txt', 'file216.txt', 'file566.txt', 'file927.txt', 'file103.txt', 'file819.txt', 'file669.txt', 'file512.txt', 'file245.txt', 'file279.t
xt', 'file556.txt', 'file71.txt', 'file422.txt', 'file915.txt', 'file469.txt', 'file54.txt', 'file170.txt', 'file951.txt', 'file494.txt', 'file665.txt', 'file713.txt',
'file505.txt', 'file766.txt', 'file333.txt', 'file858.txt', 'file89.txt', 'file789.txt', 'file922.txt', 'file618.txt', 'file992.txt', 'file490.txt', 'file316.txt', 'fil
e821.txt', 'file342.txt', 'file407.txt', 'file859.txt', 'file608.txt', 'file307.txt', 'file263.txt', 'file592.txt', 'file880.txt', 'file889.txt', 'file684.txt', 'file46
5.txt', 'file825.txt', 'file983.txt', 'file272.txt', 'file8.txt', 'file977.txt', 'file257.txt', 'file764.txt', 'file750.txt', 'file860.txt', 'file926.txt', 'file772.txt
', 'file911.txt', 'file390.txt', 'file253.txt', 'file582.txt', 'file432.txt', 'file576.txt', 'file468.txt', 'file753.txt', 'file833.txt', 'file11.txt', 'file884.txt', '
file286.txt', 'file974.txt', 'file843.txt', 'file942.txt', 'file163.txt', 'file786.txt', 'file335.txt', 'file957.txt', 'file624.txt', 'file254.txt', 'file769.txt', 'fil
e998.txt', 'file470.txt', 'file22.txt', 'file70.txt', 'file634.txt', 'file157.txt', 'file57.txt', 'file663.txt', 'file847.txt', 'file19.txt', 'file28.txt', 'file746.txt
', 'file458.txt', 'file489.txt', 'file993.txt', 'file952.txt', 'file247.txt', 'file64.txt', 'file282.txt', 'file901.txt', 'file578.txt', 'file244.txt', 'file686.txt', '
file326.txt', 'file78.txt', 'file707.txt', 'file234.txt', 'file143.txt', 'file417.txt', 'file616.txt', 'file995.txt', 'file47.txt', 'file848.txt', 'file712.txt', 'file1
66.txt', 'file363.txt', 'file761.txt', 'file588.txt', 'file887.txt', 'file256.txt', 'file595.txt', 'file790.txt', 'file849.txt', 'file871.txt', 'file172.txt', 'file368.
txt', 'file603.txt', 'file736.txt', 'file29.txt', 'file106.txt', 'file539.txt', 'file569.txt', 'file477.txt', 'file635.txt', 'file523.txt', 'file109.txt', 'file916.txt'
, 'file939.txt', 'file196.txt', 'file815.txt', 'file281.txt', 'file186.txt', 'file73.txt', 'file718.txt', 'file349.txt', 'file345.txt', 'file544.txt', 'file248.txt', 'f
ile218.txt', 'file659.txt', 'file729.txt', 'file353.txt', 'file623.txt', 'file964.txt', 'file521.txt', 'file677.txt']
Number of documents retrieved for query 2 using positional index: 93
Names of documents retrieved for query 2 using positional index: ['file25.txt', 'file41.txt', 'file627.txt', 'file145.txt', 'file778.txt', 'file66.txt', 'file502.txt',
'file658.txt', 'file95.txt', 'file679.txt', 'file638.txt', 'file45.txt', 'file818.txt', 'file767.txt', 'file74.txt', 'file325.txt', 'file44.txt', 'file711.txt', 'file65
.txt', 'file373.txt', 'file836.txt', 'file735.txt', 'file290.txt', 'file647.txt', 'file347.txt', 'file68.txt', 'file5.txt', 'file324.txt', 'file830.txt', 'file666.txt',
 'file534.txt', 'file332.txt', 'file404.txt', 'file491.txt', 'file633.txt', 'file467.txt', 'file927.txt', 'file49.txt', 'file245.txt', 'file71.txt', 'file525.txt', 'fil
e410.txt', 'file435.txt', 'file89.txt', 'file992.txt', 'file973.txt', 'file821.txt', 'file784.txt', 'file684.txt', 'file859.txt', 'file307.txt', 'file880.txt', 'file273
.txt', 'file872.txt', 'file375.txt', 'file937.txt', 'file257.txt', 'file706.txt', 'file912.txt', 'file844.txt', 'file17.txt', 'file359.txt', 'file154.txt', 'file413.txt
', 'file468.txt', 'file265.txt', 'file11.txt', 'file611.txt', 'file753.txt', 'file974.txt', 'file163.txt', 'file760.txt', 'file160.txt', 'file862.txt', 'file910.txt', '
file252.txt', 'file938.txt', 'file573.txt', 'file686.txt', 'file781.txt', 'file143.txt', 'file47.txt', 'file171.txt', 'file790.txt', 'file308.txt', 'file29.txt', 'file3
65.txt', 'file920.txt', 'file718.txt', 'file248.txt', 'file625.txt', 'file329.txt', 'file677.txt']
```