REPORT- Assignment 1
Group-111

Group members:

- Aayush Singh (aayush20009@iiitd.ac.in)
- Hrishav Basu Choudhury (hrishav20067@iiitd.ac.in)
- Ipsita R (ipsita20379@iiitd.ac.in)

Q1.

Methodology:

First we extracted the contents between the <TITLE>...</TITLE> and <TEXT>...</TEXT> tags using BeautifulSoup library.
Then we stored that data in the same file with an updated template where we implemented our data preprocessing steps as follows:

(We used NLTK library for tokenization)

1. Convert all text to lowercase.
2. Tokenize the text.
3. Remove stopwords.
4. Remove punctuations.
5. Remove blank space tokens.

Assumptions:

1. No spelling detection is done on tokens.
2. The format would be [op1,op2..].

Q2.

Methodology:

Creation of unigram inverted index

1. We created a dictionary to store inverted index for a single file
2. Then we used a token as key to assign a set as its value where the name of document is stored.
3. Then we created a dictionary to store the inverted index for all files (tokens/terms as key and a set of all document names as value)
4. Then we finally dumped this dictionary to a file using python's pickle module.

User queries for various operations

1. Firstly, we took the inputs as the number of queries, then the input statement, and then the different operations.

2. Then we preprocessed the given statement and turned it into a list of words.
3. Now we select the first two words and perform the first operation and save the results in a set, which is then used as a list to perform the next operation with the next word.
4. We have performed four operations, namely ‘OR’, ‘AND’, ‘OR NOT’, and ‘AND NOT’.

Assumptions:

1. Did not perform stemming or lemmatization on the list.
2. Input query will have a max size of 5

Q3.

Methodology:

Creation of bigram inverted index

1. On the preprocessed data, we took 2 tokens to a list named bigram.
2. We created a list of all such biagrams.
3. Then we created an inverted dictionary to store a biagram as its key and a set of file names as its value.
4. Finally, we dumped this data into a file using python’s pickle module.

the positional index:

1. On the preprocessed data we first created tokens.
2. Then we used defaultdict() which is basically a dictionary containing a list(positions of token in a particular file) and a set(file name of that file).
3. Then we used this method to load such dictionaries for the whole directory.
4. Then we finally used this dictionary to dump it into the file using python’s pickle module.

User queries

1. Preprocessing is done on the input query
2. Tokens are collected
3. If the length of the list is 0, then empty is printed
4. If the length of the list is 1 normal position of the word, the documents found are printed along with the frequency.

5. For the dictionary of the first token in the form of { doc id, position array}, iterate over all the doc IDs of the first token, and for all positions in that list as start position index and see for the subsequent query tokens. That very doc ID would be the keys and the next start as the starting position. 6) With these conditions, the docid is added to the list.

Assumptions:

1. Did not perform stemming or lemmatization on the list.
2. Input query will have a max size of 5.

Comparison on results of (i) and (ii):

1. For biagram inverted index queries we got the file numbers and file names where the queries result were present (tuple of 2 tokens).

2. While for the positional index we got the number of documents where the token is present and as well as its position in each document(but we are printing only the number and names of docs).

Below are the attached screenshots of working boolean and phrase queries:

1. Boolean queries:

```
1
Enter query : control system and analysis and design
Enter the operators: OR, AND, AND
Query 1 control OR system  AND analysis  AND design
Number of documents for query 1: 4
Number of comparisons for query 1: 3
Number of documents for query 1: {'cranfield0368', 'cranfield0367', 'cranfield0836', 'cranfield0650'}
```

2. Phrase queries:

```
{'cranfield1173', 'cranfield1295', 'cranfield0028', 'cranfield0429', 'cranfield1047', 'cranfield1030', 'cranfield1108', 'cranfield0486', 'cran
{'cranfield1173', 'cranfield1295', 'cranfield0028', 'cranfield0429', 'cranfield1047', 'cranfield1030', 'cranfield1108', 'cranfield0486', 'cran
{'cranfield1173', 'cranfield1295', 'cranfield0028', 'cranfield0429', 'cranfield1047', 'cranfield1030', 'cranfield1108', 'cranfield0486', 'cran
{'cranfield1173', 'cranfield1295', 'cranfield0028', 'cranfield0429', 'cranfield1047', 'cranfield1030', 'cranfield1108', 'cranfield0486', 'cran
{'cranfield1173', 'cranfield1295', 'cranfield0028', 'cranfield0429', 'cranfield1047', 'cranfield1030', 'cranfield1108', 'cranfield0486', 'cran
{'cranfield1173', 'cranfield1295', 'cranfield0028', 'cranfield0429', 'cranfield1047', 'cranfield1030', 'cranfield1108', 'cranfield0486', 'cran
{'cranfield1173', 'cranfield1295', 'cranfield0028', 'cranfield0429', 'cranfield1047', 'cranfield1030', 'cranfield1108', 'cranfield0486', 'cran
Number of documents for query 1: 2
Names of documents for query 1: {'cranfield0542', 'cranfield0833'}
Number of documents for query using pos1: 146
Names of documents for query using pos 1: {'cranfield1173', 'cranfield1295', 'cranfield0115', 'cranfield0028', 'cranfield1190', 'cranfield0220
```

1. flight conditions where substantial nonequilibrium
   AND, OR, NOT, OR
2. general expressions are derived
   AND, OR NOT
3. control system and analysis and design
   OR, AND, AND
4. geometry should be avoided in design
   OR NOT, AND NOT
5. infinite number of identical equally
   OR, OR NOT, OR