PART A - TASK 1
Group 1

Krinal Patel (21CS60R39)
Sarvesh Gupta (21CS60R53)
Arkapravo Ghosh (21CS60R64)
Anima Prasad (21CS60R66)

Task 1A (Building the Index):

To Run this file the command is:
python PAT1_1_indexer.py ./Data/en_BDNews24

Libraries Required for this task are:
- Standard Python Libraries
- nltk

Python Version:
Python 3.9.6

Assumption: Here we are saving the term frequencies along with doc ids in the inverted index.

Implementation: Here the goal is to read the corpus and create an inverted index and store it as a pickle file.

We use the corpus list to store the doc_id, tokens pair for all documents in the corpus. We need to get the stopword list from nltk. After this, we go through each file in the given path and process their contents to extract the text in the file. We use string processing to extract the data from their corresponding tags. Then the text in each document is tokenized using the nltk RegexpTokenizer. Then we remove the stopwords from the tokenized data. After this step we apply lemmatization on the terms. We sort the corpus based on the document ids so that the documents get added in the postings list in increasing order of doc ids. This step concludes the preprocessing task of the corpus.

Now we construct the inverted index from the corpus. We are representing the inverted index as a Python dictionary where the keys are the vocabulary terms and the values are the postings list represented as Python lists. We go through each document and get the term frequencies of every term in the document. Then we update the inverted index by appending the document ids and term frequencies in the postings list for each term.

The we sort the inverted index based on the dictionary(vocabulary/terms). Finally, we store the inverted index as a pickle file.

Task 1A Output:
Output file (“model_queries_1.pth”) contains the inverted index for the given corpus.

Task 1B (Query Preprocessing):

To Run this file the command is:
python PAT1_1_parser.py ./Data/raw_query.txt

Libraries Required for this task are:

- Standard Python Libraries
- nltk

Python Version:
Python 3.9.6

Implementation: Here the goal is to read the query file and store the query ids along with the corresponding queries.

We use the queries file to store the query_id, query pair for all the given queries. We need to get the stopword list from nltk. After this, we go through each query in the given file and process their contents to extract the query id and the title tag. We use string processing to extract the data from their corresponding title and num tags. Then the title in each query is tokenized using the nltk RegexpTokenizer. Then we remove the stopwords from the tokenized data. After this step we apply lemmatization on the terms. Then we store the query id and query pairs in a dictionary, where the keys are the query ids and the values are the corresponding queries after processing. This step concludes the preprocessing task of the query file.

Finally, we store the processed queries in a file.

Task 1B Output:
Output file ("queries_1.txt") contains the query id along with the processed queries as shown below-
Query_ID, Query

Task 1C (Boolean Retrieval):

To Run this file the command is:
python PAT1_1_bool.py model_queries_1.pth queries_1.txt

Libraries Required for this task are:

- Standard Python Libraries

Python Version:
Python 3.9.6

We are using the output generated from the code of Task-1A (model_queries_1.pth) and Task-1B (queries_1.txt).

Implementation: Here we have to handle only one type of Boolean query **'AND'.**
AND query can be handled by finding intersection of posting list of each term occurring in a query.
e.g., query: Godhra train attack.
Here we can find a list of documents by finding intersection of posting list of terms 'godhra', 'train' and 'attack'.

We find an intersection of posting list of two terms at a time using below merge procedure.

```
MergingList(lista, listb):
  listc=<>
  i=0,j=0
  while i<length(lista) and j<length(listb):
        if lista[i]<listb[j]:
             i=i+1
        else if lista[i]>listb[j]:
             j=j+1
        else:
            listc.append(lista[i])
             i=i+1
             j=j+1
  return listc
```

Time complexity of above procedure: O(m+n), where m and n are the length of posting list lista and listb respectively.

Furthermore, we have used priority queue to store the posting lists along with its' priority. Priority is the length of the posting list. Thus, smaller size of the posting list will be selected first. First, we will select two smallest posting list from priority queue and then find intersection of them using MergingList procedure and resultant list is added into priority queue. And then again select the two smallest posting list from priority queue. This, procedure is repeated until only one list remains in the priority queue.

Final Output:
It is stored in 1 file:
Output file ("PAT1_1_results.txt") contains the list of retrieved documents for each query as shown below.
Query_ID : List of retrieved document ids.