**Group number 16**
**Group Members:**
**1) Rahul Aditya (18CS30032)**
**2) Aryaman Jain (18CS30007)**
**3) Maitrey Govind Ranade (18CS30026)**
**4) Abhishek Srivastava (18CS10068)**

## Task 2A - TF-IDF Vectorization

Algo:

- Loaded the inverted index.
- For each term length of the postings list was stored as document frequency.
- For each document we maintained a dictionary, with terms as keys and term frequencies as values. Only non zero term frequencies were stored to save space.
- This dictionary for each doc was copied three times and named ‘A’, ‘B’ and ‘C’ as per given in the assignment.
- For each copy a different variant was used (‘lnc’, ‘lnc’, ‘anc’). Accordingly the copies were modified and the final dictionary was the tf-idf vector (sparse) for the given doc and given variant.
- Opened the preprocessed query file.
- For the query tokens, the tf idf vector was determined using the same algorithm as used above.

- For every query, we iterate over the vectors of each document (of the matching variant) and calculate the tf-idf score, only for terms present in the sparse vector of the query.
- The documents are sorted in descending order based on score, and the top 50 documents are written to the output file for that query and that variant.

Assumptions:

- For each doc and query we used sparse vectors. There were around 89000 documents and more than 2.3 lakh terms in the vocabulary. Not using sparse vectors was leading to memory errors.

Libraries and versions:

- Python 3.9.7
- We used pickle, math, os, json, re, sys

## Task 2B - Evaluation

Algo:

- Opened the results file produced by our algo, and stored the ordered list of retrieved docs against the query number.
- Opened the golden standard file, and stored the docs and their relevance score against each query. This list is not sorted.

- For calculating precision for a query:
    - Iterate over the list of docs retrieved for the query.
    - If the doc is present in the golden standard for the query, it is relevant (since it has a relevance score) else non relevant.
    - Calculate the precision based on the number of relevant docs retrieved.
- For calculating normalised discounted cumulative gain:
    - Obtain the relevance scores of docs for that query, and sort them in descending order. Obtain the top 10 or 20 scores. This will give us our ideal ordering.
    - Iterate over the list of docs retrieved for the query.
    - If the doc is present in the golden standard, obtain its relevance score, and add to dcg. Parallely calculate the ideal dcg.
    - Finally divide it by the ideal dcg to get ndcg @10 or @20.
- Report the metrics for each query as well averaged over all queries.

Assumptions:

- Relevance score 2 means higher relevance, and 1 means lesser relevance. If no relevance is given to a document that means its relevance score is 0.
- For saving query text we need to provide the path to “raw_query.txt” as the last command line argument and parse it as well before writing to the final file.

Libraries and version:

- Python 3.9.7
- We used csv, math and sys.