DS 501: Assignment 1

- ★ **Submit your assignment as a zip/tar.gz file in Piazza, with "IR_HW1_Group No._XYZ" as the filename.**
- ★ **Submission from any one group member would be sufficient.**
- ★ **Assignment is due on February 19, 23:59 hrs.**

Part 1: Index Construction

Write a code to crawl the top 20 webpages from Google search engine, given three queries - q1 = “Computer Science”, q2 = “IITs in India” and q3 = “Cities of Chattisgarh”. Extract the first two paragraphs from each of these webpages to create corresponding documents and name them as
[d1_q1, d2_q1,...,d20_q1],
[d1_q2, d2_q2,...,d20_q2], and
[d1_q3, d2_q3,...,d20_q3].

Write a program in Python to build an inverted index of these documents.

Submit your commented code, as well as print the postings for the three mostly used and three least used terms. Submit a separate file containing the document ids and corresponding contents (along with the webpage link)

Part 2: Merge posting-lists

Implement the merge algorithm for intersecting the postings of two terms, as well as code to use it to process Boolean queries. When there are multiple query terms, make sure that your algorithm uses the optimization of performing the most restrictive intersection first.

Using your algorithm and the index you built on the web documents, process the following queries:

- IIT AND Computer
- Computer AND Bhilai
- Bhilai AND Computer AND IIT

Submit your commented code, as well as the list of document IDs matching each query.

Part 3: Adding Skip-pointers

Re-index the same set of documents crawled in part 1 with skip-pointers. For a postings list of length P, use √P evenly-spaced skip pointers. Execute the same three queries mentioned above 100 times each and compare the absolute total time taken to run them for the index with skip-pointers and index without skip-pointers (created during part 1).

Submit your commented code, as well as time taken for both the indices.

Part 4: Scoring

Extend your system from part 2 to perform simple TF-IDF scoring of the retrieved results.

Submit your commented code and the sorted list of document IDs matching each of the queries from part 2. Mention the TF-IDF scheme used.

Part 5: Zone indexing

In this file, you will find a file containing data from approximately 100,000 PubMed Central articles. Each line of the file is an article, and each article is represented as a JSON object. Each article has a title, an abstract, a unique identifier (its PubMed ID, or PMID), and bibliographic metadata (a list of authors, journal publication info, etc.).

Run zone indexing technique on this data set, indexing both titles as well as abstracts and performing weighted zone scoring with title weight = 0.7 and abstract weight = 0.3.

Create two indices - In one of them, encode zones in the term list and in the other, encode them in posting-list.

Process the following simple Boolean queries:

- title:monkey AND abstract:development
- titletitle:schistosoma
- abstract:cholera AND title:risk
- title:mosquito AND abstract:filariasis

Submit your commented code, the number of document IDs (PMIDs, in this case) that your system found for each query, along with the list of results, and compare the total time it takes to execute these queries 100 times each on both indices (along with one line explanation of which index works better here and why).

Part 6: Create a REST API which uses ElasticSearch for indexing and searching.

Create an index for this Netflix dataset. Use it to implement a Netflix search API with the following functionalities.

A. *Auto Completion Endpoints*:
   a. *Endpoint for Adults:* Returns top 5 results(suggestions) for a random text search i.e. when a user types the text “to”, it should return top 5 suggestion(docs) containing today, tomorrow, told etc. You may assume that you are given the searched text and then continue with your design.
   b. *Child Proof Endpoint:* Same functionality as (a) with an additional constraint that it filters out all R, NC, PG rated Movies/TV shows from all search results.

B. *Pagination (Sorted by release_year) Endpoint:*

   Netflix dataset contains a field “release_year”. Your task is to return PS (page size) number of Movies/TV show documents for a given page number i.e. If page size is 10 and page number is 9, you will be returning Movies/TV show documents 81-90 sorted in descending order of release year. There will be separate end-points for Movies and TV Shows.

C. Custom Queries:
   a. *Exact match Endpoint:* Return the Movies/TV shows matching exact names for the specified fields. Example - For specified ”director” field in the dataset return all those docs which exactly match the director name. Use N-gram filter for analyzer where N = (Your Group Number % 4) + 2.
   b. *Prefix Match Endpoint:* Return the Movies/TV shows whose descriptions start with the specified query. Example - If the query is “after”, return Movies/TV shows with “description” starting with “After”.
   c. Genre Match Endpoint: Return the Movies/TV shows whose genre match the boolean query terms. Example - the query can be “Drama and (Comedy or Horror)”, and you should return Movies/TV shows with genre “Drama and Comedy” or “Drama and Horror”.

Submission Details:

a. Your Netflix Search API should be deployed remotely (For example, Heroku).
b. Note: You are designing backend APIs and need not design any front-end components.

c. Submit a PDF/Word/Latex document with your endpoint details, parameters, method (GET, PUT, POST etc.) with one example for each endpoint. Clearly state details of analyzers, tokenizers, filters, mapping, query type used for the same.