CMSC 676
Information Retrieval
Project-IV

Bhushan Mahajan
m302@umbc.edu

Project Overview:

- This project is built on previous project's (Project-3) functionalities, i.e. Calculate inverted index and creation of Posting File and Dictionary File
- In this project, I have used hashmap in Python which is defaultdict from collections library. Lookup time of hashmap is O(1).
- Previously, I had created the Term Document Matrix (TDM) using defaultdict which maps a token to a list in which document-id and token's tf-idf.
- Dictionary and Posting files are also created with the help of defaultdict.

Development Environment:

- Processor: Apple Silicon M1 (ARM-64)
- Programming Language- Python-3.8
- Text Editor: VS Code

Project Execution:

1. Go to the project directory
2. Run the following command:
    - *python3 retreive.py <space-separated keywords>*

Information Retrieval Process Explanation:

- Pre-processing:
    - Preprocessing is the first step to calculate inverted indices. I have used pre-processing functionality from Project-1 and Project-2 in which I extracted tokens using "Gensim" tokenizer.
    - Furthermore, I removed stop-words, words having length 1, words that occur only once in entire corpus. Also, I removed punctuations, numbers, white spaces, etc.

- TF-IDF calculation:
  - Terminology:
    - t: term/token
    - d: document (set of tokens)
    - N: count of corpus (For this project, N = 503)
    - corpus: the total document set
    - tf: term frequency
    - df: count of documents in which token t occurs
    - idf: inverse document frequency
    - tf-idf(t,d): weights of a token t in document d.

  - To calculate TF-IDF, I used the formula: ***tf-idf(t,d) = tf(t,d) * idf(t)***, Where,
    tf(t,d) = occurrence of t in d / total numbers of tokens in d.
    idf(t) = log_e(N/df(t))
  - Since we have large corpus, to avoid explosion of *idf* value, I have taken *natural log* of *idf* to dampen the effect.
- Inverted index processing:
  - An inverted index is a data structure that stores a mapping between information, such as words or integers, and their places in a document or group of documents. In layman's terms, it's a hashmap-like data structure that guides you from a word to a text or a web page.
  - Position-hashmap:
    - This is a hashmap that contains array of hashmap as a value for a given token as key.
      i.e. MAP(Token, LIST(MAP(document_number, tf-idf)))
    - This is a global hashmap and values are inserted right after tf-idf calculation is done.
  - Dictionary-hashmap:
    - This is a hashmap that maps the token to another hashmap of number of documents in which that token occurs and document number of first occurrence of the token.
    - i.e MAP(Token, LIST(MAP(Number of documents, Document of first occurrence)))
- Retrieval Algorithm:
  - The input query is preprocessed using same preprocessor used while tokenization of HTML documents.
  - The query words are then matched against the inverted file to come up with document weights, using Posting file and Dictionary file hashmaps.
  - To do so, posting file and dictionary file is first loaded into memory.

- To fetch the top ten documents, with highest tf-idf score, a maxheap is used. Maxheap is constructed using tuples of format (tf-idf, document_number). In python there is no default implementation of maxheap since python3 provides only minheap implementation. Hence while constructing minheap, negative tf-idf is stored so that largest score is at the top of tf-idf (negative of larger number is smaller).
- Furthermore, to extend the retrieving processs, user can add weights while retrieving the query. Command line execution is as follows:
  - *python3 retreive.py --wt weight1 keyword1 weight2 keyword2*
  - *--wt flag enables program to consider the input weights given in command-line.*
- The retrieval function prints the token, top ten document name and their corresponding TF-IDF.
- If the input keyword is not found then appropriate system message is displayed.

- Data structures used:
  - Hashmap, MaxHeap, Tuples. No intermediate or output files are created while retrieving the data.
- Time complexity:
  - Let n be the number of key-value pairs in Posting Hashmap.
  - Lookup the input token in the hashmap takes average time **O(1).**
  - Constructing maxheap to retrieve first 10 documents for the given input keyword takes time **O(k.logk)** where k is the number of items pushed and fetched from the maxheap.
  - Therefore, approximate time complexity of retrieving the data is **O(1) + O(klogk).**

Output Sample:

```
(venv) Bhushans-MacBook-Air:Mahajan_Bhushan_Project_4 bhushan$ python3 retrieve.py diet international affairs Zimbabwe computer network hydrother
apy identity theft
diet

Document: 18.html  tf-idf: 0.2590715164269349
Document: 252.html  tf-idf: 0.020131931025326965
Document: 263.html  tf-idf: 0.019083392951091186
Document: 9.html  tf-idf: 0.01574467779390212
Document: 50.html  tf-idf: 0.013878831237157227
Document: 152.html  tf-idf: 0.005076817127131148
Document: 353.html  tf-idf: 0.002365622590506046


international

Document: 161.html  tf-idf: 0.09430751692973044
Document: 133.html  tf-idf: 0.07642374239384324
Document: 117.html  tf-idf: 0.0711048738755904
Document: 247.html  tf-idf: 0.06561040634884024
Document: 243.html  tf-idf: 0.056376007144218115
Document: 205.html  tf-idf: 0.04540142284623847
Document: 138.html  tf-idf: 0.04307314475155958
Document: 125.html  tf-idf: 0.03499693011064216
Document: 108.html  tf-idf: 0.03318227447527552
Document: 197.html  tf-idf: 0.03228545624621402


affairs

Document: 219.html  tf-idf: 0.07827284879919254
Document: 129.html  tf-idf: 0.020664032082986828
Document: 133.html  tf-idf: 0.020338614254908297
Document: 295.html  tf-idf: 0.01961775197751914
Document: 226.html  tf-idf: 0.0187174203650243
Document: 229.html  tf-idf: 0.016452254843142382
Document: 389.html  tf-idf: 0.015944469199835515
Document: 286.html  tf-idf: 0.009747184944805108
Document: 232.html  tf-idf: 0.009392741855903103
Document: 235.html  tf-idf: 0.009291381332278251
```