Web Information Extraction and Retrieval Programming Assignment 3: Document indexing and querying

Bojan Vrangeloski

1. Pre-processing the data

For a more efficient indexing, we decided to preprocess the corpus. The corpus contains 1416 crawled web pages. Each web page is processed in the same manner as described in continuation. First, we extract the text data using the package inscriptis and unify the text into lowercase. Due to the poor performance of the dependent library which leaves behind some HTML tags among the content, we extracted them out with regex. Although still we had some complications. The next step of preprocessing the corpus is tokenization, i.e. splitting up the data into, not necessarily meaningful, words. We have performed it using the nltk.tokenize package. In order to minimize space complexity of our approach we removed all duplicates occurrences of the tokens. To keep the tokens meaningful, we also removed all tokens belonging to a predefined set of stopwords, and eliminated most of the special characters. To keep the data retrieval process as efficient as possible, we also tokenized the content of each document without removing the stopwords and special characters. The output of the pre-processing is a dictionary data structure which containing the source file names. The structure of our pre-processed corpus is available in the listing below. We store this structure on the disk in order to efficiently build the indices and speed-up the sequential data retrieval.

2. Index Building

To be able to benchmark the performance of the indices, we built two different indices: Inverted

Index and Non-Inverted Index. We use the cached pre-processing output in order to efficiently build

the indices. In the following subsections we describe how we build the indices.

- ### Inverted Index

In our implementation of the Inverted index, we used a database in order to simulate the inverted index structure. The database structure is consisted of three tables: IndexWord consists of all the words indexed from the documents in the corpus, i.e. our dictionary, D. Posting consists of a word from IndexWord, a document name in which the specific it appears, the frequency of its appearance within the document, and the indices where the word appears in the source document. Existing consists of a column doesExist which we have added in order to indicate whether the inverted index has already been built, as the rebuilding operation is rather costly. First, we store the list of keys from the Inverted Index dictionary in the table IndexWord. The keys represent the set of all unique words from the provided corpus. Then we construct a list which holds all postings in the format [(word, documentName, indexes)].

- ### Non-Inverted Index

The Non-inverted Index does not require a special procedure of preparing in order to perform the search. We simply use the pre-processed data.

3. Data Retrieval

The data retrieval process (search) is performed on a user provided query which is first pre-processed, to get it in the same form as the rest of the corpus, i.e. it is tokenized. Afterward we are performing the search based on which index has been chosen by the user (either sequential or inverted). Both methods return the results in the same format: a list of tuples containing the cumulative document, the document name, and the aggregated indices of all results.

4. Implementation Details

Further implementation details worth noting:

Output printing Each row from the obtained result is printed in a table as provided in the instructions. The printing is provided by texttable.

Mode where it is possible to enter in an interactive mode where the user is able to: query both types of indices, change the type of index being queried, force a recreation of the index and change the number of results printed in the table.

5. Conclusion

Throughout the paper we introduced our implementation of building inverted index and query against it. Key differences between naive approach of non-inverted file reading and inverted index are noted supporting with experiments and results which prove how much more efficient the inverted index approach is, which is the reason why that concept is widely used nowadays even though Inverted Index takes much more time in constructing the structure as it allows very efficient querying.