Indexer

Timotej Kovač

I. Introduction

In this paper we describe the results gathered while testing two approaches in finding relevant documents to a search query. The two approaches used were the invert index and a manual checking one. We describe the implementation of both of them, pre-processing steps that were taken before and the results that we got.

II. Indexer implementation

A. Data processing with indexing

All of the data pre-processing was done in file `processing.py`. There html content is read directly from file and a DOM tree is constructed with the help of 'lxml.html' library. This is then cleaned of all scripts and nav and footer sub trees. After that only the body is forwarded to the next step of the extraction process. The result is then tokenized with the help of 'nltk' library, special characters are removed with the help of regular expressions library 're' and the resulting words are checked with the stop words black list. After this process what remains are only the words that have some meaning and are deemed useful for indexing. This was done for all of the websites that were available to us.

After that the unique words were gathered and tuples were created in order to insert them into the database. First two tables `IndexWord` and `Posting` were created. After that all of the data was inserted with two insert statements. This was necessary as individual insert statements which were first executed during the word tokenization proved to be inefficient.

B. Data retrieval with inverted index

For the data retrieval part with the help of mySQL database we started by retrieving the user input query words. First we checked if the database was present and if necessary we ran the data processing step with indexing (as described in the previous section). After that multiple select statements were executed to gather the frequency, document and indices of particular words appearing in the previously retrieved html pages. After that all of the data for individual query words were merged and sorted from the most frequent ones to least. Snippets were extracted to be attached to the end result. At the end the retrieved data was formatted and sent to the standard output.

C. Data retrieval without inverted index

For this part we used some of the same approaches as when we used the inverted index database. The main difference here was that we didn't use the database so we didn't have to create it and fill it with data. Nevertheless we still had to gain information about the appearance of words in html pages. So we started with this extraction and then manually checked each result if any of the query words appeared in it. If this was the case we saved the frequency, document and indices and proceeded with the search. After searching all of the documents we proceeded with merging the results, adding snippets, sorting them, formatting the output and sending everything to the standard output.

III. Database description

IV. Results

From the results it is apparent that the approach using the inverted index data is far superior to the one that manually checks all of the documents. Time needed for each approach can be seen in table IV.

query/method	inverted index	manual checking
predelovalne dejavnosti		
trgovina		
social services		
-		
-		
-		

Table I
Comparison of time needed to gain a query result using the inverted index and manual checking methods

The actual results gathered using the inverted index for the above mentioned queries can be seen bellow:

A. *predelovalne dejavnosti*

B. *trgovina*

C. *social services*

D. -

E. -

F. -

V. Conclusion

We were successful in implementing the invert index approach as well as the manual checking one. The results we got are in favor to the invert index one as it is clearly much faster and far less demanding as the manual checking one. With further improvements to database structure and better pre-processing steps we could improve the time needed for a search result execution.