Inverted index

Kacper Zaleski, Michał Dzienisik, Kamil Janowski

October 2022

1 Abstract

An inverted index is an index data structure storing a mapping from content, such as words or numbers, to its locations in a document or a set of documents. In simple words, it is a hashmap like data structure that directs you from a word to a document or a web page. It allows quick searching of text documents. The inverted index is a structure where, for each word, it is indicated the documents that contain the word. Thus, when a user enters a specific search term, it is very fast to know the documents that contain that term.

There are two types of inverted indexes: A record-level inverted index contains a list of references to documents for each word. A word-level inverted index additionally contains the positions of each word within a document. The letter form offers more functionality, but needs more processing power and space to be created.

Advantage of Inverted Index are:

- Inverted index is to allow fast full text searches, at a cost of increased processing when a document is added to the database.
- It is easy to develop.
- It is the most popular data structure used in document retrieval systems, used on a large scale for example in search engines.

A web crawler, crawler or web spider, is a computer program that's used to search and automatically index website content and other information over the internet. These programs, or bots, are most commonly used to create entries for a search engine index.

Web crawlers systematically browse webpages to learn what each page on the website is about, so this information can be indexed, updated and retrieved when a user makes a search query. Other websites use web crawling bots while updating their own web content.

2 Introduction

In order to have efficient search engine we need inverted index is to allow fast full text searches, at a cost of increased processing when a document is added to the database. This document consist of our implementation of word-level inverted index. It describes our solution to the problem and experiments done to test efficiency of script. Program is developed fully in C. Instructions how to use are in separate file. Program is deployed on github.

Given the vast number of webpages on the Internet that could be indexed for search, this process could go on almost indefinitely. However, a web crawler will follow certain policies that make it

more selective about which pages to crawl, in what order to crawl them, and how often they should crawl them again to check for content updates.

3 Solution

The solution is built for with an assumption for future modules as a crawler, searcher ex. In the main directory there is only Readme file and gitignore. going further in to the SearchEngine folder there is 'src' folder. Inside it there would be more services, but for now there is only 'SearchEngine.Calculation' that is a console application with purpose to index given files.

Indexer as a input value receive file and base information about it as: URL, title and data to index. Before indexing the data is analyzed by analyzer and filtered by character filter. Also there is a process of tokenization with is separating test in to words. And then tockenFilters are put in to data. Purpose of it is to remove all unnecessary data in the word. For example small letters, stop words or non letter or number characters.

Web crawler is activated by running process/demon every one minnute and starts crawling Project Gutenberg website. It's crawling through website code looking for specific parts of a website with important information to it. to be more specific it is looking for list for books. And then when it finds one it is looking for: Author, Comment, Title, Link to book, And ranking. Then information are mapped in to object that's can be processed by inverted index.

Last part of crawler is to write pre-processed data into files with corresponding dates to crawled time. When file is written and save the main program is activating indexer to index the files and add to inverted index. Every loop the process is tempted by loop meaning given period a time.

4 Experiments

Testing are conducted on file with size 125kb and 20 000 words. And on the end of each one there is summary of total indexing time and number of indexes.

Crawler was tested on Project Gutenberg website with several sub sides. First was test of accuracy. Crawler scraped the same web side ten times and we compared the results of it if all the books and class object match.

Second test was test of modified content on the website to achive that we hosted one page of Gutenberg web side locally and changes source code. The results weren't perfect because in web crawler there is specified the pattern of data and if it doesn't match data can be wrongly interpreted. Data of name of the author was taken from wrong column. After this test we added security gap to prevent it from happening. If one parameter doesn't exist on the web side we are logging it and spitting this site.

Third test was to focused on measuring duration of whole process. Fetching the web takes on average taken form 100 crawls 70 percent of web crawl time. Average fetching time from 100 test was 412 ms and time of processing on av. was 124ms. It is making it really important to achieve the shortest download time .

5 Conclusion

Index consisting with about 10000 characters translates to 600 word indexes and it gets calculated in 200ms, which is good result and more then enough to meet our search engine development goals.

The most important parameter in web crawling is fetching time it is about 70 percent of all process time. Crawler after acquisition the data is just mapping it to internal object witch is simple and fast apart from input output actions as a writing it to files in the system do enable Indexer to index new data and update database.