ANUSHA GUPTA: aag488@ nyu.edu
SHIVANI GUPTA: sg3725@nyu.edu

CS 6913: WEB SEARCH ENGINES

ASSIGNMENT 2 &3: INVERTED INDEX AND QUERY PROCESSING

INTRODUCTION:

Many search engines incorporate an inverted index when evaluating a search query to quickly locate documents containing the words in a query and then rank these documents by relevance. Because the inverted index stores a list of the documents containing each word, the search engine can use direct access to find the documents associated with each word in the query in order to retrieve the matching documents quickly. The inverted index is generally a huge file and would not fit into the main memory at once. Therefore we load the terms in the main memory one by one as we get them in the query. Techniques such as document at a time query processing; caching and block wise uncompressing is used to optimize the lookup in the inverted index file and find out the pages relevant to the query. Relevance of a query is calculated using the BM25 score.

PROBLEM DEFINITION:

To write a program that creates an inverted index structure from a set of downloaded web pages. Since a basic web crawler which does not use an inverted index are probably all fairly slow, we will be using a larger set of pages for this part of the assignment. The data will be in the form of a compressed gzip file containing a few ten thousand up to a few million pages. The assignment 2 has been built up using python unix shell script where the call to the scripts is made using the python script.

Data (Inverted Index and Lexicon) developed in assignment 2 are used to process a query and find out the pages relevant to the query efficiently in assignment 3. Assignment 3, asks to write a program that will input a user's query and return the top ten results (URLs of webpages) which are relevant to the query terms. "AND" semantics is taken into consideration for the processing of the query terms. Similarly assignment 3 has been made using python and unix shell script. Assignment 3 makes use of the unix shell script to uncompress the file in parts.

HIGH-LEVEL SOLUTION:

Initially we parse all the data files based on the information given in the respective index files and filter out all the required words and important context related to the words from all the webpages. This data is then used to generate the intermediate postings files which will contain the document Id, frequency, position and context of the word for all the words present in the webpage and writing these postings out on the disk. Now all the temporary postings files need to be sorted and merged into one postings file. But the sorting and merging operations have to be I/O-efficient, i.e., run fast even when the data set is much larger than the available main memory. Unix-based sort algorithm is used here to perform I/O-efficient sorting.

After sorting and merging all temporary postings into one posting file, this final postings file is compressed using GZIP compression technique and written to the disk. Using the final postings file, an inverted index is made; this contains all the postings for all the words. A lexicon which is a dictionary data structure is also created which contains the file pointer and offset for each posting. The lexicon is stored in main memory. A mapping of the URLs and the document ID is also maintained using a simple list data structure to present the output of the search results. This file is also kept in main memory.

All these data structures contain important and meaningful information which will be required to search for webpages related to a particular query in an efficient / optimized manner.

For assignment 3, all these data structures i.e. the URL to doc id mapping and lexicon dictionary are loaded into main memory and the “SearchingApp” is called. A new window pops up where the user can enter his/her search query in a text box and click on the “search now” button. The On-Click event of the button takes the query as input to the program and filters the query for stop words. We search each of these query terms in the lexicon to retrieve the corresponding file offset in the inverted index. Here, the inverted index is present on the secondary memory in a compressed format.

To achieve good performance in terms of memory utilization and computational time, the entire inverted index file is not uncompressed at a time. Once the offset is retrieved, only that particular line (block) is uncompressed into the main memory in a list data structure. This is done for all the query terms. Thus we have all the document ids where the query terms occur and their corresponding frequencies in the document. Using an algorithm explained in detail below, we find the most relevant documents (webpages) for the search query terms and rank them in an order with the most relevant document appearing first. Lastly, we store the top ten results presented as URLs of webpages in a heap data structure and display them in the window as web accessible click-able links along with the respective score (rank). The user can click anyone of these links and the URL will open up in the default browser. All of this is achieved in less than **A SECOND** of the Button-click event.

SOLUTION DETAILS:

Basic definitions:

Main Memory (RAM) - Random access memory (or simply RAM) is the memory or information storage in a computer that is used to store running programs and data for the programs. Data (information) in the RAM can be read and written quickly in any order.

Web search engines- A web search engine is a software system that is designed to search for information on the World Wide Web. The search results are generally presented in a line of results. The information may be a specialist in web pages, images, information and other types of files. In our assignment we present the results as a list of URLs of the most relevant web pages.

Crawler- A Web crawler is an Internet bot / program that systematically browses the World Wide Web, typically for the purpose of Web indexing. A Web crawler may also be called a Web spider. A Web crawler starts with a list of URLs to visit, called the seeds. As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to visit, called the crawl frontier. URLs from the frontier are recursively visited according to a set of policies.

Focused Crawling -Web crawlers that attempt to download pages that are similar to each other are called focused crawler or topical crawlers.

Indexing-Search engine indexing collects, parses, and stores data to facilitate fast and accurate information retrieval. Index design incorporates interdisciplinary concepts from linguistics, cognitive psychology, mathematics, informatics, physics, and computer science. An alternate name for the process in the context of search engines designed to find web pages on the Internet is web indexing. Popular engines focus on the full-text indexing of online, natural language documents. Media types such as video and audio and graphics are also searchable. In our assignment we are considering only text-based indexing. The purpose of storing an index is to optimize speed and performance in finding relevant documents for a search query. Without an index, the search engine would scan every document in the corpus, which would require considerable time and computing power. For example, while an index of 10,000 documents can be queried within milliseconds, a sequential scan of every word in 10,000 large documents could take hours. The additional computer storage required to store the index, as well as the considerable increase in the time required for an update to take place, are traded off for the time saved during information retrieval.

Caching- Accessing the original data may take a long time, or it may be expensive to do. For this reason, it is much "cheaper" to simply use the copy of the data from the cache. Put differently, a cache is a temporary storage area that has copies of data that is used often. When a copy of the data is in this cache, it is faster to use this copy rather than re-fetching or re-calculating the original data. This will make the average time needed to access the data shorter. In our assignment 3, we cache the Inverted Index in main memory in a block of 64KB so that accessing this file is faster during a query lookup.

BM25 ranking function-In information retrieval, Okapi BM25 is a ranking function used by search engines to rank matching documents according to their relevance to a given search query. It is based on the probabilistic retrieval framework developed in the 1970s and 1980s by Stephen E. Robertson, Karen Spärck Jones.

$$BM25(q,d) = \sum_{t \in q} \log\left(\frac{N - f_t + 0.5}{f_t + 0.5}\right) \times \frac{(k_1 + 1) f_{d,t}}{K + f_{d,t}}$$

N: total number of documents in the collection;
f_t: number of documents that contain term t;
$f_{d,t}$: frequency of term t in document d;
$|d|$: length of document d;
$|d|_{avg}$: the average length of documents in the collection;
k_1 and b: constants, usually $k_1 = 1.2$ and $b = 0.75$

$$K = k_1 \times \left((1 - b) + b \times \frac{|d|}{|d|_{avg}}\right)$$

Var-Byte Encoding standard- This is a simple byte oriented method for encoding data. It takes an integer as input and converts it to a binary format.

Simple Method: vbyte

- simple byte-oriented method for encoding data
- encode number as follows:
  - if < 128, use one byte (highest bit set to 0)
  - if < 128*128 = 16384, use two bytes (first has highest bit 1, the other 0)
  - if < 128^3, then use three bytes, and so on …
- examples: 14169 = 110*128 + 89 = 11101110 01011001
  33549 = 2*128*128 + 6*128 + 13 = 10000010 10000110 00001101
- example for a list of 4 docIDs: after taking differences
  (34) (178) (291) (453) … becomes (34) (144) (113) (162)
- this is then encoded using six bytes total:

34 = 00100010
144 = 10000001 00010000
113 = 01110001
162 = 10000001 00100010

Assignment 2:

Class Parser that parses to eliminate the tags and find out the words present in the cached web pages in order to build up the lexicon. The class parser has methods handle_starttag that gets the start tag, handle_endtag that handles the end tag and handle_data that handles the data, finds out the word and store it in the temporary word list which is being maintained for every data page we get.

createIndex function parses the index file associated with the data file and get all the urls in the index page for that particular data page which is appended in the url list.

buildLexi function builds up the lexicon after all the words and their postings have been built. The postings have redundant words from which only distinct words are taken and they are used to build up the lexicon. The final posting file is compressed and written in a separate file to disk.

writeBinary / readBinary function writes /reads the file in binary format.

Modules are script files unzip.sh, sort.sh and zip.sh. The script unzip.sh unzips the file passed in as an argument. The script sort.sh sorts the unsorted files by the word . It uses unix sort to sort the files and merge them in a single file. The script zip.sh cleans up all the temporary postings files and compresses the final file conatining the inverted index data.

Assignment 3:

Class searchingApp is the class that initializes the gui for the assignment. The gui contains a text box, text area and a search button. The text box is the place where the query is entered. After the query is entered the user hits the search button. The program then executes the query and calculates the output links. Once the links have been found they are displayed using the text box.

Class hyperlinkmanager is used to make the links web accessible. It makes use of the module web browser to open up the page once the link is clicked on the textbox.

exceuteQuery is the function which handles all the communications between the functions. It has the cached data, lexicon and the doc to url mapping. These data sets are loaded before the query is executed.

Once these have been loaded in the main memory the query is fetched from the textbox of the searchingApp application. After the query has been fetched separate terms are got by splitting the query on whitespace using the function getTerms.

Uncompress function then is called to uncompress the part of the inverted index for the terms present in the query if they are not found in the cached data. The uncompression is done using shell script. The shell script is named as unzip_part.sh. The shell is passed the skip count which is the number of blocks that need to be skipped, size of the block that needs to be uncompressed

and the count of the blocks that need to be get from the compressed inverted index file. The uncompressed data is in binary mode.

Then the inverted lists for all the terms are opened using the openList method. The openList method.

Handling memory usage:

Making sure that the data structures do not take up the entire main memory space and at the same time making sure that the computations are performed at the best speed and accuracy possible is achieved by using techniques such as block wise compression and un-compression, making the right choices for the data structures that need to be used, using I/O efficient search and sort algorithms.

Main memory-

Inverted Index- The inverted index file created in the assignment 2, is required in assignment 3 for processing the query. The problem that occurs here is that the Inverted Index file is huge and will not fit entirely into the main memory.

Disk memory-

The temporary postings files created are stored on disk in binary format in compressed format using gzip to save space and optimize lookups. Once the final postings file is made, it stored in compressed format on the disk. This

Temporary files-

During the creation of the final postings file from all the downloaded web pages, the temporary postings files that are created are stored on the disk in a compressed format, these files are not required once the final file is created and are deleted to free up memory.

CONFIGURATION DETAILS:

1. All the input files should be stored in the same folder and the path of the this folder should be mentioned in the "build_index.py" file (Line no : 139)
2. Make sure all three shell script files have the correct access rights for the user, if not use the command chmod 777 filename.sh
3. Save the "build_index.py", "unzip.sh", "sort.sh", "zip.sh" in the same folder and run the command python build_index.py to run the program. The output file appears in the same folder.

STATISTICS:

Assignment 2:

Time required to execute the code for the NZ2 dataset is approximately 60 mins.

The size of the output file i.e. the 'InvertedIndex.txt' is 31.3 MB (after compression)

Assignment 3:

Time to return the search results are approximately 1 second.

Time to return the search results when caching is implemented is less than a second.

LIMITATIONS:

Currently the program is supported for Unix-based operating systems only as we are using unix shell scripting.

FUTURE SCOPE:

The process of looking up the inverted index to find the relevant documents can be optimized further by using pruning techniques.

A more rigorous ranking function can be used to determine the relevance of a page with respect to the query entered by the user like the PageRank by Google.

Proximity analysis is the process of considering the location of the query terms in the documents and the distance between the terms. This is a contributing factor in the determining the relevance of the page to the query. In our solution the location of the terms with respect to the beginning of the document are stored, but the proximity analysis algorithm is a part of future implementations.