Information Retrieval and Web Search
Assignment 1
Vishal Singh
2018CS50426

Inverted Index Construction and Boolean Retrievals

Running Instructions and Directory Structure

Check README.md for it

Some good techniques used in the program for faster run times

1. Least Frequently Used Cache
2. Reverse bits using a lookup table in O(1) time.

Implementation Details

After indexing, I have a .dict file containing the dictionary and a .idx file containing all the posting lists.

```
.dict file contains
   1. Stop Words
            vector<string>
   2. Dictionary/Vocabulary:
            unordered_map<string, tuple<size_t, size_t, ll>>
            term -> posting_list_size, compressed_list_size,
position_in_idx_file
   3. Documents
            vector<string>
            It will have all the doc_ids.
            If DOCID has index i then its respective number used in the
      program will be i+1.
```

.idx file contains continuous bytes of chars representing the posting lists and nothing else.

Indexing

While making a dictionary/vocabulary I make sure that the dictionary remains in memory.
For each term, I have a safe size which means that how many bytes can I keep without any problem for each term. Even if all the terms consume full safe size it will be in memory. Currently, the safe size for each term is 2 MB.
Then there is the extra size (*MAX_EXTRA_SIZE_POSTING_LISTS*) which means how many bytes in the dictionary can I keep beyond the safe sizes. If the

dictionary ends up having an extra size more than this then I would push all the terms occupying more than safe memory to disk in temp files.
Each posting list can span across multiple files in the temp folder during indexing.
Then after all the indexing, I need to build the .dict file and .idx file. For each term, I will fetch the entire posting list in memory if not already present using files stored in temp. Then I compress it and push it to the .idx file.
Now, I push stop words, dictionary, document_indices_to_id vector to .dict file.

Boolean Search

I have implemented a **Least Frequently Used Cache** which will have the vocabulary with posting lists. If the current size becomes > capacity then the least frequent terms' posting lists will be removed. For each term in the query, I fetch the posting list one by one and keep intersecting them with the results.

Compression and Decompression

For compression and decompressions, I have made sure that each term is compressed in O(1) time. I have fully utilized bitwise operations for faster compressions.
For compression method 1 and 2, I needed to reverse the bits for the number. I did that too in O(1) using this method Reverse bits using lookup tables in O(1) time.
For compression method 2 and 4, I have developed a custom bti stream where you can keep writing bits and then later convert the whole to a compressed list.
I also implemented compression method 4 **(BONUS)**, for each number in the posting list, I first stored k using compression method 2 and then the required bits by the compression.

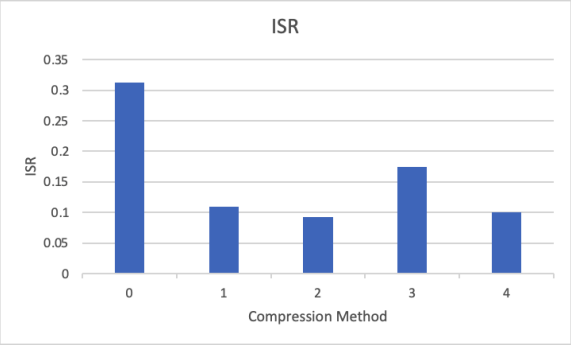
Metrics of Interest

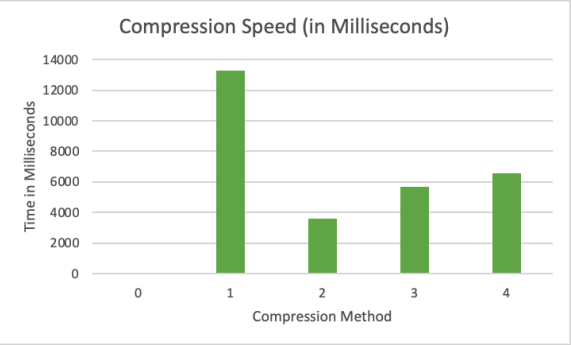
Index Size Ratio (ISR) ISR = (|D|+|P|)/|C|
|D| = the size of the dictionary file
|P| = the size of the postings file
|C| = the size of the entire collection = 516.7 MB in our case

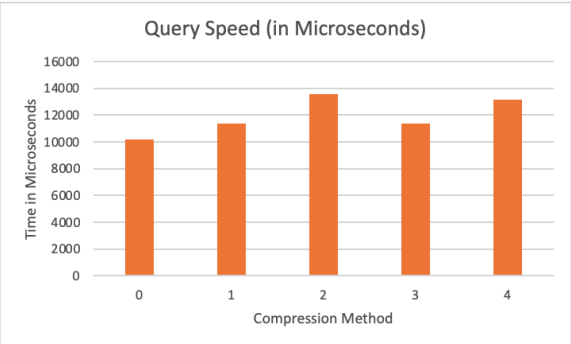
Compression Speed = Time difference between indexing with a compression strategy and indexing without any compression.

Query Speed = |TQ|/Q
where TQ is the time taken for answering (and writing the results to the file) for all given queries, and Q is the number of queries.
Q = 50 in our case

Compression Method	ISR	Compression Speed (in **Milliseconds**)	Query Speed (in **Microseconds**)
0	0.313	0	10200
1	0.110	13270	11400
2	0.092	3580	13600
3	0.174	5680	11400
4	0.101	6550	13134







Detailed Stats for given Documents and Queries

Following results are taken on a Ryzen 9 machine with 16 GB of RAM

Time taken for different Compressions Methods

Total Time Taken in Compression (method) = t(method) - t(0)

Compression Method	Total Time is Time taken in Indexing (t) in seconds	Total Time Time Taken in Compression (c) in seconds	Total Time is Time taken in Searching (s) in seconds	Total Time is Time taken in Decompression (d) in seconds
0	152.75	0	0.51	0
1	166.02	13.27	0.57	0.06
2	156.33	3.58	0.68	0.17
3	158.43	5.68	0.57	0.06
4	159.30	6.55	0.65671	0.14671

Note: Total time in compression is not exactly compression time since we are also writing data to the disks. If the compressed file has less size then the overall program will take less time even though compression might have taken a large part of time. But, since I was using an M.2 gen 3 SSD with 2GB of reading/write speeds this time is very similar to actual compress times.

File Sizes for different Compressions Methods

Compression Method	size(indexfile.dict) in MB	size(indexfile.idx) in MB
0	15.47 MB	146.56 MB
1	15.47 MB	41.79 MB
2	15.47 MB	32.17 MB
3	15.47 MB	74.59 MB
4	15.47 MB	36.8 MB