CS276: Information Retrieval and Web Search **Spring 2013**

Programming Assignment 1

Botao Hu (botaohu), Jiayuan Ma (jiayuanm) Completed Date: *April 18, 2013*

1 Design and Implementation

To make the code more structured in all the subsequent tasks, we implement a unified framework which encapsulated all corpus I/O operations (both uncompressed or compressed) in the `Corpus` class, which can be found in `corpus.py`.

1.1 Task 1

We implemented blocked sort-based indexing (BSBI) without compressing the postings lists. To help with the merge process, we store term id with each postings list. We also keep the length of each postings list on file so that we don't need any separators between postings lists. BSBI is implemented in `index.py`, and retrieval with query optimization is implemented in `query.py`. The indexing time for this task is 121 seconds, and the corpus index size is 58 Mb (in total 72Mb with dictionaries). The retrieval time for dev-queries is in Table 1.

Query	1	2	3	4	5	6	7	8
Time(s)	3	2	2	3	2	3	2	3

Table 1: Retrieval time without compression

1.2 Task 2

We use variable byte (VB) encoding to compress the size of postings lists on file. Specifically, we apply VB encodings on gaps between postings. The size of the compressed index is 19 Mb (in total 35Mb with uncompressed dictionaries), and the time for indexing is 326 seconds. The retrieval time for dev-queries is in Table 2.

Query	1	2	3	4	5	6	7	8
Time(s)	2	3	2	2	3	2	3	2

Table 2: Retrieval time with VB compression

1.3 Extra credit

We apply γ encoding on on gaps between postings. Because γ encoding of 0 is empty bit-string, and γ encoding of 1 is "0", which we want to use as a padding symbol. So we increment every number to be encoded by two before encoding and use 0's as a padding symbol to pad in the end of bitstream to make the bitstring of the posting aligned to a byte. The alignment makes us convenient to locate the data in the index file via `file.seek()` operation. We compressed the index to 16 megabytes in 1194 seconds (33Mb in total with uncompressed dictionaries). The statistics of answering dev-queries are in Table 3.

Query	1	2	3	4	5	6	7	8
Time(s)	3	2	3	2	3	2	3	3

Table 3: Retrieval time with γ compression

2 Answers

(a) In this PA we asked you to use each sub-directory as a block and build index for one block at a time. Can you discuss the tradeoff of different sizes of blocks? Is there a general strategy when we are working with limited memory but want to minimize indexing time?

If the block is too small, it may incur more times of file I/O, which is unfavorable to efficiency. If the block is too large to fit into the memory, the per-block operation would take a much longer time. The general strategy when working with limited memory is to make effective use of memory as much as possible. Therefore, we may want the size of each block to be nearly the same as the size of memory.

(b) Is there a part of your indexing program that limits its scalability to larger datasets? Describe all the other parts of the indexing process that you can optimize for indexing time/scalability and retrieval time.

If the dataset is so large that we can no longer keep the entire dictionary in the memory, we may want to compress the dictionary and have part of it on disk as well. On the other hand, we can use single-pass in-memory indexing (SPIMI) to accelerate BSBI when compressing the postings list. When merging blocks of postings, we can use a multi-way merge instead of pairwise merge to improve efficiency.

(c) Any more ideas of how to improve indexing or retrieval performance?

During indexing, we prefer to use C++ to implement bitwise operations. As the experiment shows, Python implementation performs worse in the gamma encoding corpus due to a large volume of bitwise operations although the size of database gets samller.

We can make use of skip list to further speed up the merging process in retrieval.