**Assignment no 3**

**CSL 436 Information Retrieval**

**Winter 2024**

**Maximum Marks: 20 (May be scaled later)**

**Deadline: 20th September 5.00 pm (Hard copy needs to be submitted in my cabin)**

1. Consider a lexicon (dictionary) in which there are no words of length 1 or 2. Assume the probability of a word in this lexicon having length i is proportional to 1/i2, for i > 2. Assume that the distribution is truncated so that the longest possible word length is 25. Further, the lexicon has 50,000 words in it. [Hint: The sum of the series 1/i2 is 2/6. If needed, you can approximate the sum of the series 1/i2 for i = 1, ........, 25 by 1.61.]

(i) Assume each character requires a byte of storage. Although inefficient, we could safely use 25 bytes per word for the dictionary, as in the first cut approach. How much storage is required for storing all the words in this fashion?

(ii) Now consider storing the words as one contiguous string, with a term pointer to the beginning of each word. How much space is used in total, including storage for the entire string as well as for the pointers that resolve the beginning of each word? Show a formula, and estimate the total as a number of bytes needed.

(iii) Next consider “blocking” the dictionary string so we point to the beginning of each eighth word rather than every word, storing the length of each word in one byte immediately preceding its appearance in the string. What is the total space used now? Show the formula, your estimates, and an estimated number of bytes.

1. We have 100 million documents containing a total of 9 million terms.

(i) How many posting entries are there using the simple Zipf approximation? You may assume that the natural log of 9 million is 16.

(ii) Assume 12 bytes per postings entry and a machine with sufficient main memory to hold all data in memory. Assume a cost of 1 microsecond per CPU operation. How much time would it take to sort the postings entries in memory? Assume we use Quicksort with running time of Θ(N ln N).

(iii) Next consider the case where we have only 8GB of main memory, and we use blocks with 200 million postings entries in a block. Assuming 0.5 milliseconds per disk seek, 0.5 microseconds per byte following a seek in block transfer mode and 1 microsecond for all other operations, estimate the time to create one such block of 200 million sorted postings on disk. Also give the total time needed to create all such (initial) sorted blocks.

(iv) Now consider merging these sorted blocks into a single inverted index by reading two sorted blocks from disk, merging them and writing back to disk. What is the time for such a single such read/merge/write?

(v) What is the total merge time for all such merges in fully sorting the data?

1. Compute variable byte and **γ**codes for the postings list (777, 17743, 294068, 31251336). Use gaps instead of docIDs where possible. Write binary codes in 8-bit blocks.
2. Consider the postings list (4, 10, 11, 12, 15, 62, 63, 265, 268, 270, 400) with a corresponding list of gaps (4, 6, 1, 1, 3, 47, 1, 202, 3, 2, 130). Assume that the length of the postings list is stored separately, so the system knows when a postings list is complete. Using variable byte encoding:
3. What is the largest gap you can encode in 1 byte?
4. What is the largest gap you can encode in 2 bytes?
5. How many bytes will the above postings list require under this encoding? (Count only space for encoding the sequence of numbers.)
6. A little trick is to notice that a gap cannot be of length 0 and that the stuff left to encode after shifting cannot be 0. Based on these observations:
7. Suggest a modification to variable byte encoding that allows you to encode slightly larger gaps in the same amount of space.
8. What is the largest gap you can encode in 1 byte?
9. What is the largest gap you can encode in 2 bytes?
10. How many bytes will the postings list in Question 4 require under this encoding? (Count only space for encoding the sequence of numbers.)
11. From the following sequence of **γ**-coded gaps, reconstruct first the gap sequence and then the postings sequence: 1110001110101011111101101111011.
12. **γ**codes are relatively inefficient for large numbers as they encode the length of the offset in inefficient unary code. **δ** *codes* differ from **γ**codes in that they encode the first part of the code (*length*) in **γ**code instead of unary code. The encoding of *offset* is the same. For example, the **δ**code of 7 is 10,0,11 (again, we add commas for readability). 10,0 is the **γ**code for *length* (2 in this case) and the encoding of *offset* (11) is unchanged.
13. Compute the **δ**codes for the other numbers (0, 1, 2, 3, 4, 9, 13, 24, 511, 1025). For what range of numbers is the **δ**code shorter than the **γ** code?
14. **γ**code beats variable byte code because the index contains stop words and thus many small gaps. Show that variable byte code is more compact if larger gaps dominate.
15. Compare the compression ratios of **δ**code and variable byte code for a distribution of gaps dominated by large gaps.
16. For a collection of your choosing, determine the number of documents and terms and the average length of a document.
17. How large is the inverted index predicted to be?
18. Implement an indexer that creates a **γ**-compressed inverted index for the collection. How large is the actual index?
19. Implement an indexer that uses variable byte encoding. How large is the variable byte encoded index?
20. To be able to hold as many postings as possible in main memory, it is a good idea to

compress intermediate index files during index construction.

1. This makes merging runs in blocked sort-based indexing more complicated. As an example, work out the  **γ**-encoded merged sequence of the gaps in

(a) 1110110111111001011111111110100011111001

(b) 11111010000111111000100011111110010000011111010101.

1. Index construction is more space efficient when using compression. Would you also expect it to be faster?
2. Answer the following:

(i) Show that the size of the vocabulary is finite according to Zipf’s law and infinite

according to Heaps’ law.

(ii) Can we derive Heaps’ law from Zipf’s law?