Web Information Retrieval (67782) Ex1: Index Structure Analysis

Submitted by: **Yanir Elfassy** **308111830**

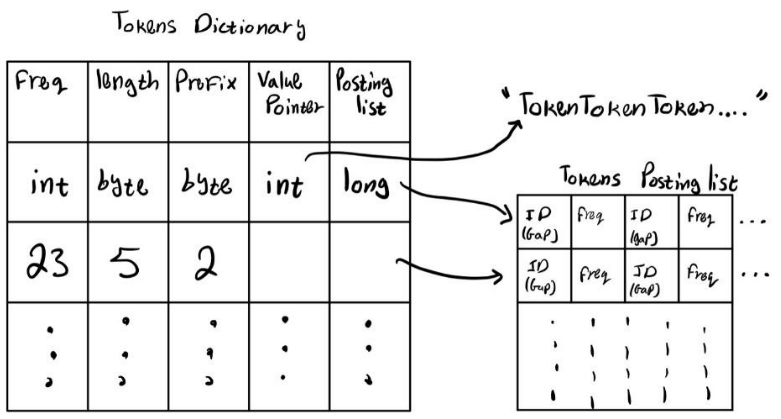
1 General Explanation and Diagram

My implementation consists of two index data structures and a separate data structure that holds the reviews meta-data. The main index class is the *Dictionary* abstract class, and both index data structures (*TokensDictionary* and *Products Dictionary*) inherit from it.

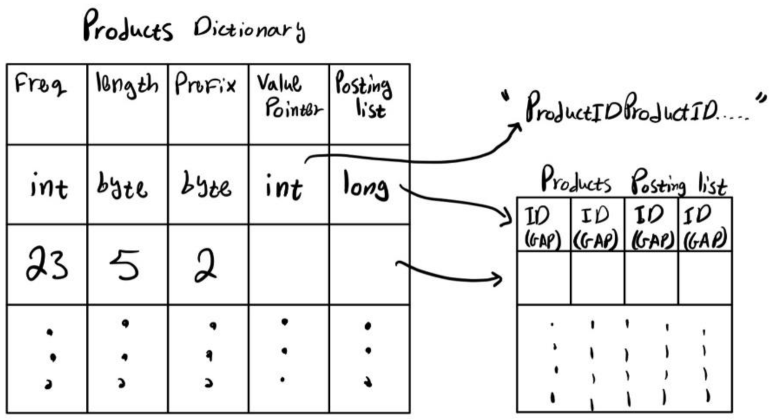
- Dictionary
  An abstract class that represents a general index data structure that uses the k-1 to k front method where k is set to 100. This class consists of the following properties:
  - *String* dictValues - Stores the concated string of the dictionary.
  - *int[]* freq – Stores the frequencies of each value in the dictionary.
  - *byte[]* length – Stores the length of each value in the dictionary.
  - *byte[]* prefix – Stores the length of the common prefix of the dictionary value.
  - *int[]* valuesPtr – Stores the starting position of the token. We store one pointer every 100 values.
  - *long[]* postingListPrt – Stores the starting index of the posting list in the file.

  Each class that extends this abstract class, should implement the *buildDictionary* method.

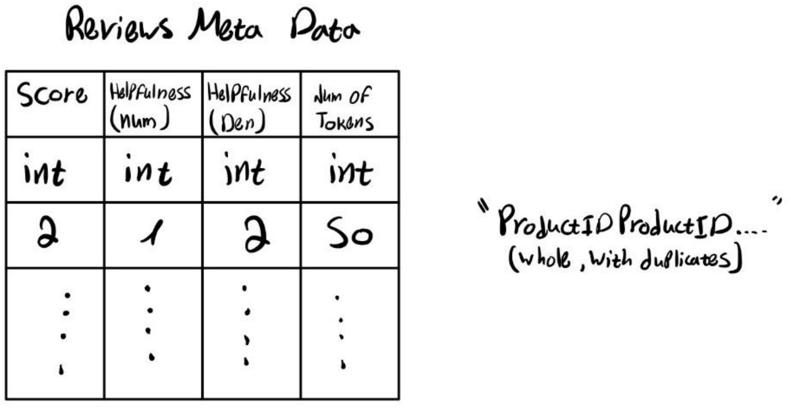
- TokensDictionary
  Extends the *Dictionary* class and represents an index data structure that stores data about tokens from the imported reviews. A single entry in the posting list contains two data elements, and the content is compressed using Group Varint:
  - reviewID – The ID of the review that contains the relevant token.
  - Frequency – The frequency of the relevant token inside the specific review.



- ProductsID
  Extends the *Dictionary* class and represents the index data structure that stores the data about each product ID. A single entry in the posting list contains a single data element, *reviewID*, which is the ID of the review written about the relevant productid. the content is compressed using Group Varint.



- ReviewsMetaData
  A data structure that contains all the meta data of the parsed reviews. This class consists of the following properties:
  - *int[]* score – contains the score given in the review.
  - *int[]* firstHelpfulness – contains the first part of the helpfulness.
  - *int[]* secondHelpfulness – contains the second part of the helpfulness.
  - *int[]* numOfTokens – contains the number of tokens in the review.
  - *int* numOfReviews – The total number of reviews.
  - *String* productIDs – All the product IDs, concated to each other with duplicates.



2 Main Memory Versus Disk

Attached a table describing which portions of the index are read into the memory and which will be read as needed:

Main Memory	Disk (Read as needed)
TokensDictionary	TokensDictionary posting list
ProductsDictionary	ProductsDictionary posting list
ReviewsMetaData	

3 Theoretical Analysis of Size

Denote:

- *N Number of reviews*
- *M Total number of tokens (counting duplicates as many times as they appear)*
- *D Number of different tokens (counting duplicates once)*
- *L Average token length (counting each token once)*
- *F Average token frequency, i.e., number of reviews containing a token.*
- *K The k-1 to k front method parameter (Here K=100)*
- *S The Average size of suffix without mutual prefix (specifically: S=L – avg(prefix length))*
- *B The Average number of Reviews containing a given token.*
- *A The Average frequency of a token in a single review.*

- <u>ReviewsMetaData size</u>

$$\underbrace{N \cdot 4 \cdot 4}_{4\ int\ arrays\ with\ length\ of\ N} + \underbrace{4}_{numOfReviews} + \underbrace{8+4+4+4+12+10*2*N}_{ProductIDs\ string}$$

$$= 16N + 36 + 20N = 36 + 36N = 36(N+1)\ Bytes$$

- TokensDictionary

The size of the props (without the posting list) is as follows:

$$\underbrace{D \cdot 4}_{frequencies\ array} + \underbrace{2 \cdot D}_{2\ byte\ arrays\ with\ length\ of\ D} + \underbrace{\left\lceil \frac{D}{K} \right\rceil \cdot 4}_{valuesPtr-arry\ of\ int}$$

$$+ \underbrace{D \cdot 8}_{postingListPtr-array\ of\ long} \underbrace{+ 36 + \left(\left\lceil \frac{D}{K} \right\rceil \cdot L + \left(D - \left\lceil \frac{D}{K} \right\rceil \right) \cdot S \right) \cdot 2}_{dictValues\ string}$$

$$= 14D + \left\lceil \frac{D}{K} \right\rceil \cdot 4 + \left(\left\lceil \frac{D}{K} \right\rceil \cdot L + \left(D - \left\lceil \frac{D}{K} \right\rceil \right) \cdot S \right) \cdot 2 + 36$$

$$= \left(14D + (1 + L + S) \cdot 2 \cdot \left\lceil \frac{D}{K} \right\rceil + 2DS + 36 \right) Bytes$$

The size of the posting list (using Group Varint) is as follows:

$$\left(\underbrace{B \cdot 4}_{bytes\ for\ ReviewIDs} + \underbrace{B \cdot \left\lceil \frac{\log_2 A}{8} \right\rceil}_{bytes\ for\ frequencies} + \underbrace{\frac{2B}{4} \cdot 4}_{bytes\ for\ the\ size\ indicator} \right) \cdot \underbrace{D}_{for\ each\ token\ entry}$$

$$= D \cdot \left(6B + B \cdot \left\lceil \frac{\log_2 A}{8} \right\rceil \right) Bytes$$

This is a rough estimation since the number of bytes for each encoded value, depends on the value itself (we do not use a fixed number of bytes per value).

In total, the estimated size of the Tokens Dictionary in bytes is:

$$\left(\left(14D + (1 + L + S) \cdot 2 \cdot \left\lceil \frac{D}{K} \right\rceil + 2DS + 36 \right) + D \cdot \left(6B + B \cdot \left\lceil \frac{\log_2 A}{8} \right\rceil \right) \right) Bytes$$

- <u>ProductsDictionary</u>

  The size of the props (without the posting list) is the same as the TokensDictionary.

  The size of the posting list (using Group Varint) is as follows:

$$\left(\underbrace{B \cdot 4}_{bytes\ for\ ReviewIDs} + \underbrace{\frac{B}{4} \cdot 4}_{bytes\ for\ the\ size\ indicator} \right) \cdot \underbrace{D}_{for\ each\ token\ entry}$$

$$= 5 \cdot B \cdot D\ Bytes$$

  This is a rough estimation since the number of bytes for each encoded value, depends on the value itself (we do not use a fixed number of bytes per value).

  In total, the estimated size of the Products Dictionary in bytes is:

$$\left(\left(14D + (1 + L + S) \cdot 2 \cdot \left\lceil \frac{D}{K} \right\rceil + 2DS + 36\right) + 5 \cdot B \cdot D\right) Bytes$$

4 Theoretical Analysis of Runtime

Using the same variables, theoretically analyze the runtime of the functions of IndexReader. Include both the runtime and a short explanation.

- IndexReader(String dir):
  For each data structure we save the properties in a different file. In addition, we encode the data using Group Varint. So, for each data structure, we read from the file and decode the data. In this function we create instance of TokenDictionary, Products Dictionary and ReviewsMetaData.

- getProductId(int reviewId):
  Since we are saving the product IDs in a concated string as they are (without prefix / suffix calculations), we use the substring method to get the product ID, and each product ID has a fixed length of 10, hence the runtime is the same as the substring method. So the runtime is O(1).

- getReviewScore(int reviewId):
  We are accessing the array in a specific index, hence the runtime is O(1)

- getReviewHelpfulnessNumerator(int reviewId):
  We are accessing the array in a specific index, hence the runtime is O(1)

- getReviewHelpfulnessDenominator(int reviewId):
  We are accessing the array in a specific index, hence the runtime is O(1)

- getReviewLength(int reviewId):
  We are accessing the array in a specific index, hence the runtime is O(1)

- getTokenFrequency(String token):
  In order to get the frequency there are few steps:
  - Find the token in the dictionary: binary search + linear search in range – $O(D \cdot \log(D))$
  - Get the posting list pointer: access to index in array – $O(1)$
  - Decoding the posting list:
    Recall B is the average number of reviews per token.
    - Accessing the file in specific position: O(1)
    - Decoding the bytes we read: O(B)
  - Select only the even indexes from the array: $O(B)$
  In total the runtime is: $O(D \cdot \log(D)) + O(B)$

- getTokenCollectionFrequency(String token):
  - Find the token in the dictionary: binary search + linear search in range – $O(D \cdot \log(D))$
  - Accessing the array in specific index: O(1)
  In total the runtime is: $O(D \cdot \log(D))$

- getReviewsWithToken(String token):
  - Find the token in the dictionary: binary search + linear search in range – $O(D \cdot \log(D))$
  - Get the posting list pointer: access to index in array – $O(1)$
  - Decoding the posting list:
    Recall B is the average number of reviews per token.
    - Accessing the file in specific position: O(1)
    - Decoding the bytes we read: O(B)
  In total the runtime is: $O(D \cdot \log(D)) + O(B)$

- getNumberOfReviews():
  Returning a class property: O(1)

- getTokenSizeOfReviews():
  Going over the array numOfTokens which is N long.
  The runtime is: O(N)

- getProductReviews(String productId):
  In order to get the frequency there are few steps:
  - Find the token in the dictionary: binary search + linear search in range – $O(D \cdot \log(D))$
  - Get the posting list pointer: access to index in array – $O(1)$
  - Decoding the posting list:
    Recall B is the average number of reviews per token.
    - Accessing the file in specific position: O(1)

- Decoding the bytes we read: O(B)

In total the runtime is: $O(D \cdot \log(D)) + O(B)$