Text Retrieval and Search Engines – HW01

Dry Part – Positional Index

Ariel Suller – 324369412

Avi Ferdman – 316420132

Q1:

Consider the following documents:

Doc 1: I am a scientist, and I am currently enrolled to IR course. I am a student at Reichman University.

Doc 2: I was a student and scientist.

Let's say we want to find documents in which **"I"** and **"scientist"** are at most 2 words apart, but in the same sentence. How would you modify the positional index to support queries that demand the terms to be in the same sentence? You can assume that there is a pre-parsing step that identifies sentences in documents. Describe the structure of the index you would construct. Write the modified postings list for the words "I" and "scientist".

In order to be able to find documents where the words "I" and "Scientist" are at most 2 words apart, we suggest changing the positional index in the following way – for each word we insert to the index: the term, number of documents in which it appears, the document id, number of the sentence (1st in document, 2nd in document, 3rd in document, etc.) and position as shown at class (from document beginning). So, the suggested index is:

<**term**, number of docs containing **term;**
doc_id: {sentence number, position}, ...;
etc.>

In this way, we first continue to support all the ideas we saw in class (since we keep the "classic" position as shown in class), and we can use the new data added to look for words in the same sentence, in the distance wanted.

For the given documents example, the index will be (counting in 1-base):

<**I,** 2,

Doc1: {1, 1}, {1, 6}, {2, 13};

Doc2: {1,1}>

<**scientist,** 2,

Doc1: {1, 4};

Doc2: {1, 6}; >

So, If we want to see if the words appear withing 2 words apart from each other, we need to find in which documents they both appear (merging as shown in class) and for those documents check if for both words sentence id is equal, and difference between position is less or equal to 2.

Q2:

Malware indexing and retrieval: malware is a malicious software program with several characteristics. Malware has a **type** (virus, worm, ransomware..), **author** and a short **description**. A malware search engine retrieves malware information in response to a malicious activity (query). Describe the structure of the index you would construct for malware search given a collection of malware objects.

For this given task, we would build the following index:

< **term**, number of malwares containing **term**;

Malware_id: {type, position},..>

Where **term** is any word in the malware object (type, author or description), type is: 1 for type, 2 for author, and 3 for description, and position is the position in the flattened malware object (meaning – all the object as one long string, not separated to fields). Malware ID is a key number given to the malware object, starting from 1.

In a separate database we will keep the malware objects (with their unique ID).

When a Query arrives, the engine will search for each word in the index, and count for each malware_id how many times it was retrieved in this query (meaning – we will know how many words from the query appear in this malware object). We will return the objects by descending order of this count.

We could also use a dictionary for malwares, and skip inserting to the index common words that can appear in a query but won't indicate anything about a malware – for example words like “used”, “against”, “install” – could appear but they don't indicate about a query being a malware. A good way to test it will be using test benchmarks and see which words affect the most the false positives, and remove them.

Q3:

Explain the different characteristics of the above two sets of terms

---

The difference between the characteristics of the two sets of terms lies in their contribution to the text's context. The first set consists of words that are too general and provide little insight into the context of the text. In contrast, the second set includes words that carry significant contextual meaning, as they are likely to appear in documents discussing similar semantic topics. For this reason, the "idf" method assigns lower weights to the most frequent tokens and higher weights to the less frequent ones, emphasizing their contextual relevance.