**Documents and Data on the Web Coursework**

1. Introduction

This report presents the decisions taken while implementing a document indexing application with MapReduce.

1. Functionality
   1. **Removing characters and references**

After creating the initial version of the Inverted Index (passing each character sequence as a token), few tokens were a combination of random characters (‘7G02’), word(s) with special characters (‘(info/dl)’) or a word with reference(s) (‘chalk”[1]’).

Removing special characters may lead to losing some important tokens, such as dates (‘1992-02-20’), websites or words with hyphens (‘mass-energy’). However, in the given documents exist tokens containing special characters that are adding unnecessary information and make tokens much harder to be found (ex. ‘Fox.?’).

For this reason, only brackets, commas, references and few characters at the beginning and at the end of the token were removed. The removed characters add a valuable information only when the token is a website containing them which happens rarely.

* 1. **Case folding**

Two methods are implemented for case folding:

* If the word is at the beginning of the sentence, is capitalized and the word that is following is decapitalized, then we convert the token to lower case.
* If we have a capital case word that is in the middle of the sentence, it is a stopword and the following word is decapitalized, then we remove the token.

The problem that these methods are creating is that we may convert a name such as ‘Antarctic’ to lower case if it is at the beginning of the sentence and we may leave a token with a capital letter, if an end token has not been added for the previous sentence. However, most of the sentences start with a stopword and our second method will reduce such mistakes. Also, our first method will not convert names in the middle of the sentence into lower case.

* 1. **Multi-word terms**

Single terms can be very useful, when we are searching for a specific phrase, because tokens in that phrase will have close index positions. However, if we are searching only part of a name/phrase (ex. George) in a single-term application, we may find different phrase combinations of the word (ex. ‘George H’, ‘George Meyer’ (can be seen in the fragment output)). When we are having a multi-term application, we may only recognize phrases, but not single tokens (ex. ‘World War II’, but not ‘war’). This is the reason why a combination of multi and a single token indexing has been implemented.

Two techniques were used for making multi-tokens. The first one is recognizing names, when we have consecutive capitalized words that do not end with an end sign (ex. ‘The Simpsons’). The disadvantage of this one is that may create non-existing names if all words are capitalized (ex. ‘From Wikipedia’) or if initials are used (ex. ‘J’). The other method is making the content under quotes a multi-word token. This may detect some important phrases from someone’s speech (‘Yabba Dabba Doo’), but can include reported speech.

* 1. **Stemming**

If we are using stemming, we may receive different roots for words with the same root (ex. the stemmed version of ‘addition’ is ‘addit’ in ‘Bart the Mother’, instead of ‘add’). However, stemming reduce the number of tokens, which can increase the performance of the application. For instance, in the given documents, the word ‘add’ was written in 4 different forms (‘add’, ‘added’, ‘adding’ and ‘adds’). We can replace all of them with their root ‘add’. In the Inverted Index stemming was used, because we perform the operation only on the single term token which does not change the meaning and the context much.

* 1. **Stopword removal**

Stopwords take major part of the given documents. For instance, the word ‘the’ is appearing 206 times in ‘Bart the Genius.txt’. When someone is searching a stopword that is not in a phrase, then he should waste a lot of time to find what he is looking for. If we are searching into larger file, this task may take hours. Also, if we use stopwords as tokens, then their position indexing can contain thousands of positions, which will decrease the performance.

However, this removal may affect the content, especially if the phase contains only words with decapitalized words (ex. removing ‘into’ from the phrase ‘developed into’). The given documents do not contain many lower case phrases that are not part of a quote. For this reason, stopwords are not considered as tokens in the application.

1. Performance
   1. **In-mapper aggregation and position indexing**

A hash-map was used to store each token with an ArrayList of index positions. This hash-map enable the map operation to pass to the reducer only one copy of each token and its index positions. It reduced 5 to 10 seconds of the running time and reduce the complexity of the reduce function by not going several times through the same token.

The biggest issue with this technique is that if we tokenize a document consisting only of different tokens/words (which happens rarely), then we need to create a hash for all tokens, and we need to perform one additional check every time to ensure that the token does not already have a hash which may reduce the performance.

* 1. **Pairs and TFIDF**

Pairs were used for storing two or more types of information as one object. In the map function the location and the position indexes of a token are encapsulated as a pair. In the Reducer the pair was used as a part of another pair to create a triple of writables which store not only the information of the input pair but also the TFIDF. The triple enabled the program to visualize as one object the document location, token’s indexes and its TFIDF for the given location.

The downside of this implementation is that if we need to present more types of information, we need to go through multiple pairs for each term, which reduce the performance.

Fragment of output:









