Project Report-P3

Group Members:
Vamsi Krishna Angajala
Teja Sasank Gorthi
Sudhir Kumar Ghandikota

Wikipedia questions

Problem 1:
Problem Statement:
To find basic article statistics.

Since the data set has large number of individual files and the size of the file is relatively small, we have created a customized input format(**WholeInputFormat**) which takes the entire content of the file as the value(instead of one single line) and the file name as a key.

i. Job 1 - finds largest and smallest articles w.r.t word count

Mapper:
- In the mapper, entire content of an article is read and its word count is found. ***StringTokenizer*** used to tokenize the words.
- A generic key "values" is emitted along with the combination of word count and file name as the value (Text).

Reducer:
- In reducer, the articles with the highest and lowest word count are found. The names of these articles along with their word counts are the final output of this job.

Output:

```
Max Article   article002738 Words---> 3390126
Min Article   article005969 Words---> 10420
```

ii. Job 2- finds largest and smallest articles w.r.t character count

Similar to the earlier job but instead of word count, the mapper emits the character count and the reducer finds article with largest and smallest character counts.

Output:

```
Max Article   article002738.txt Letters -> 4240348
Min Article   article005969.txt Letters-> 74107
```

iii. Job 3 - finds top ten and bottom ten words in each article w.r.t word count

Mapper:

- The mapper works on each article and emits the Top Ten (words having highest count) and bottom ten (words having least count) respectively.
- A HashMap with the word as key and their count as value is maintained in the mapper.
- This HashMap is then loaded onto a TreeMap with custom Comparator so that the words are sorted according to their word count
- Output from the mapper is "article name" and "word" as a composite key and "count

of the word", "total no of words in the article" and the "term frequency" of the word as a text.

Reducer:

- The reducer is just used to aggregate the outputs from all mappers (articles) into one single file.

Sample output: Output file can be found in the path https://github.uc.edu/ghandisv/P3Hadoop/tree/master/ArticleStatistics/SampleOutput

KEY		VALUE		
Article Name	Word Name	Frequency of word	Total Word count	Term Freq. (%)
article000000.txt	And (Highest Count)	66674	2536138	2.63
article000000.txt	repente(lowest count)	1	2536138	0.00003943

iv.**Job 4: finds top ten and bottom ten letters in each article w.r.t count.**

Similar to the above job but works on characters instead of words.

Sample output: Output file can be found in the path https://github.uc.edu/ghandisv/P3Hadoop/tree/master/ArticleStatistics/SampleOutput

KEY		VALUE		
Article Name	LetterName	Frequency of letter	Total letter count	letter Freq. (%)
article000000.txt	a(Highest Count)	1180133	13236705	8.91
article000000.txt	z(lowest count)	20017	13236705	0.151

Problem 2:
Problem Statement:

For each article to find top 5 words according to TF-IDF metric.

i. **Job 1 - Finds term frequencies of all the words**

Mapper:

- The mapper receives the entire article as a value and the article name as key
- It collects all the words, and the number of times they have occurred in the document and emits the "word name" as the key and combination of "article", "word

frequency”, “maximum frequency among all the words in the article”, “Total Number of words in the article” along with a “1” for every article.

- The discretion behind emitting the “maximum frequency among all the words in the article” is to **normalize** the term frequency value. So the word which occurs most times in the article will have a term frequency of ‘1’ and the rest of the words are normalised accordingly.
- The discretion behind sending “1” for every value emitted by mapper is, since the mapper works on the whole article, the sum of all the ones will give the number of articles in which the word has been found (used for IDF calculation).

Reducer:

- The reducer as mentioned above will add up all the ones and find the article count.
- The combination of “word” and “article” is emitted as key along with other stats mentioned above as value along the article count.

ii.**Job 2 - Finds IDF values for all the documents**

Mapper:

- The output from the first job is sent as an input to this job. “KeyValueTextInputFormat” used to access the same key-value pairs.
- In the mapper the above mentioned stats are accessed and tf, idf and tf-idf values are calculated for each word and article combination.
- The tf value is normalized here in the mapper(as described earlier).
- To calculate IDF value, log-base 2 has been used. The formula used is
IDF = log2 (N/m), where N= total no. of articles, m = no. of articles in which the word is found

Reducer:

- The reducer received all such word and article combinations along with the calculated tf-idf values and collects them in a **HashMap**.
- This data is stored onto **TreeMap** and a custom Comparator to sort all the word and article combinations based on the tf-idf values.
- After sorting them, the top 5 words are emitted.

Sample Output:

article000000.txt Word->difranco, Term Frequency->0.0006,IDF->1.87,TFIDF->0.00124
article000000.txt Word->amphipolis, Term Frequency->0.0003,IDF->3.46,TFIDF->0.00128
article000000.txt Word->poirot, Term Frequency->0.00154,IDF->0.87,TFIDF->0.00134
article000000.txt Word->amaranthus,Term Frequency->0.0005,IDF--->2.459,TFIDF->0.00140
…..

The sample output file can be found at
https://github.uc.edu/ghandisv/P3Hadoop/tree/master/TfIDF2/SampleOutput

<u>Google 1gram questions</u>

Problem 1:

Problem Statement :

For each year available, plot the size of the set of words used. Year on the x-axis, number of words on y-axis.

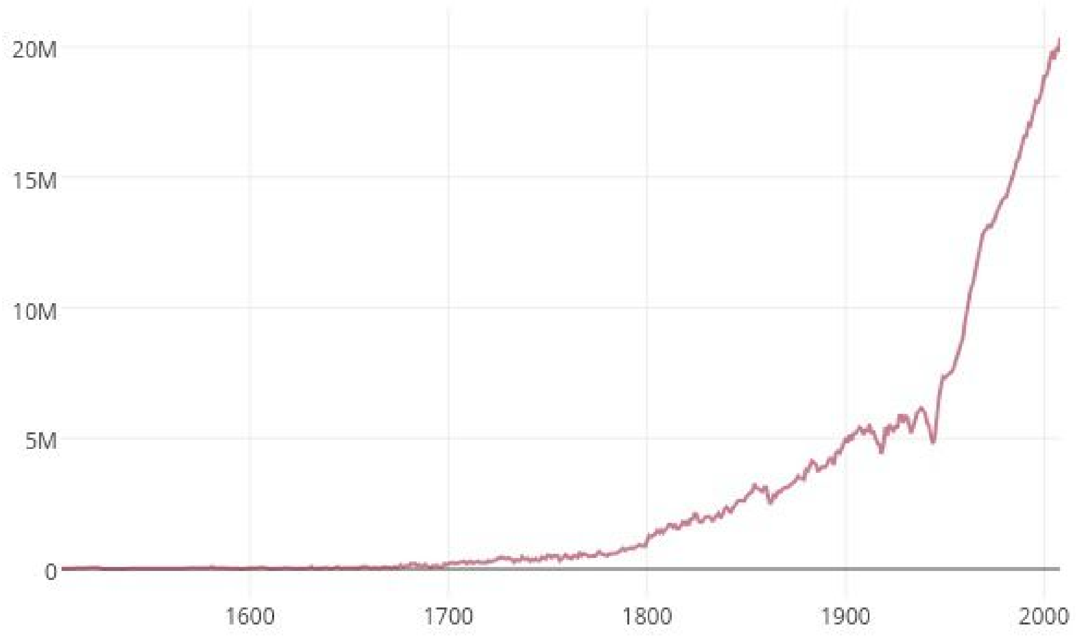
To draw a graph firstly, the number of words used per year need to be calculated for each year.
Each line in the data set is in the format of (‘word used’, year, number of times it appeared each year, number of books it appeared in that year). Since we need the set of words used, this implies we need only the distinct words used each year. Accordingly the map and reduce functions are implemented as below.

Map Function :- The map() function for this problem would read each line of the input files(Key (LongWritable), Value(LongWritable) - value of the string in each line). The mapper will split the line based on a delimiter(TAB or “\t”) and populate the year, word , counts (number of times, number of books) for each line. The mapper will output a map which has the key as the year and output the number 1, for every distinct word it encounters that year.

Reduce Function :- The reduce() function for this problem would sum the number of the values for each distinct key. This implies it calculates the number of 1’s for each year, which is the number of words used per year.

The output of reduce function can be found at
https://github.uc.edu/ghandisv/P3Hadoop/blob/master/Google%20ngram/output/first

_The data from the reducer can be used to plot a graph of Number of words used for that year vs Year



Problem 3 :

Problem Statement :

Plot the average word length for all unique words for all years available. Year on x-axis, average word-length on y-axis.

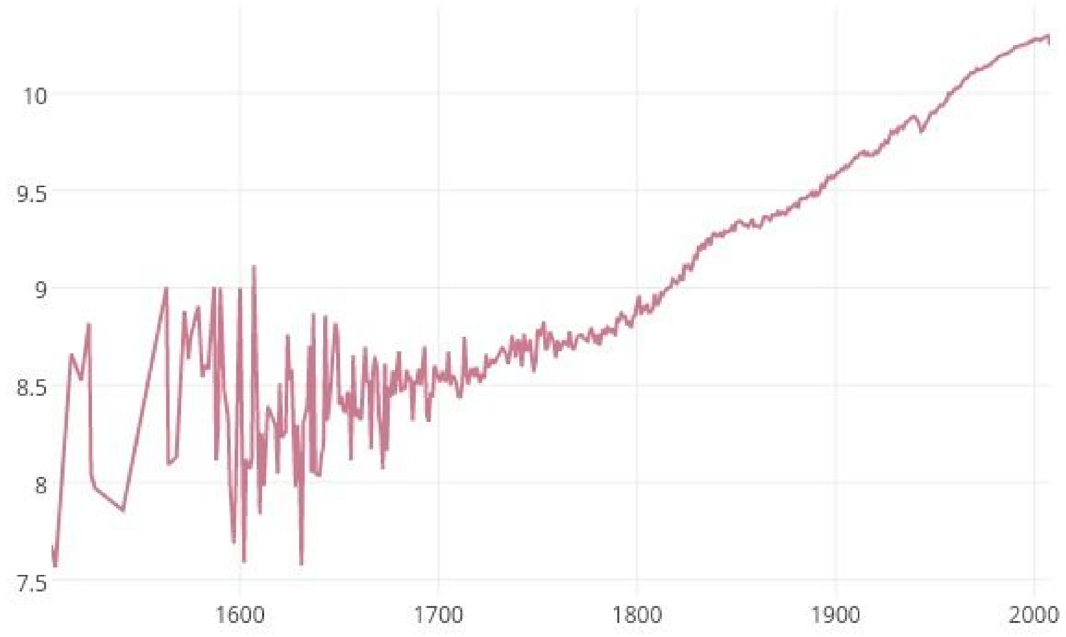
To draw a graph firstly, the average length of words used per year need to be calculated.

Each line in the data set is in the format of (‘word used’, year, number of times it appeared in that year, number of books it appeared in that year). Therefore, to calculate the average length we need to calculate the sum of lengths of all words used that year divided by the number of words used.

Map Function: The map() function for this problem would read each line of the input files(Key (LongWritable), Value(LongWritable) - value of the string in each line). The mapper will split the line based on a delimiter(TAB or “\t”) and populate the year, word , counts (number of times, number of books) for each line. The mapper will output a map which has the year as the key and the length of each word as value in the line.

Reduce Function :- The reduce() function for this problem will receive a map which has the year as the key and the length of each word as value, for each line. The reduce function will maintain two local integer variables, counter and sum, for each year. As it iterates through each value for a given key (year), it increments the counter by one and sum by the length of word. Finally, the counter has the number of words used for each year and the sum has the total length of words used. The reduce function will then output a map with key as the year and value of sum divided by counter, as value.
The output of reduce function can be found at
https://github.uc.edu/ghandisv/P3Hadoop/blob/master/Google%20ngram/output/second

The data from the reducer can be used to plot a graph of Average length vs Year



Problem 4:

Problem Statement :

Plot the average number of syllables on the y-axis, and time (year) on the x-axis.

To draw a graph firstly, the number of syllables used per year need to be calculated for each year.

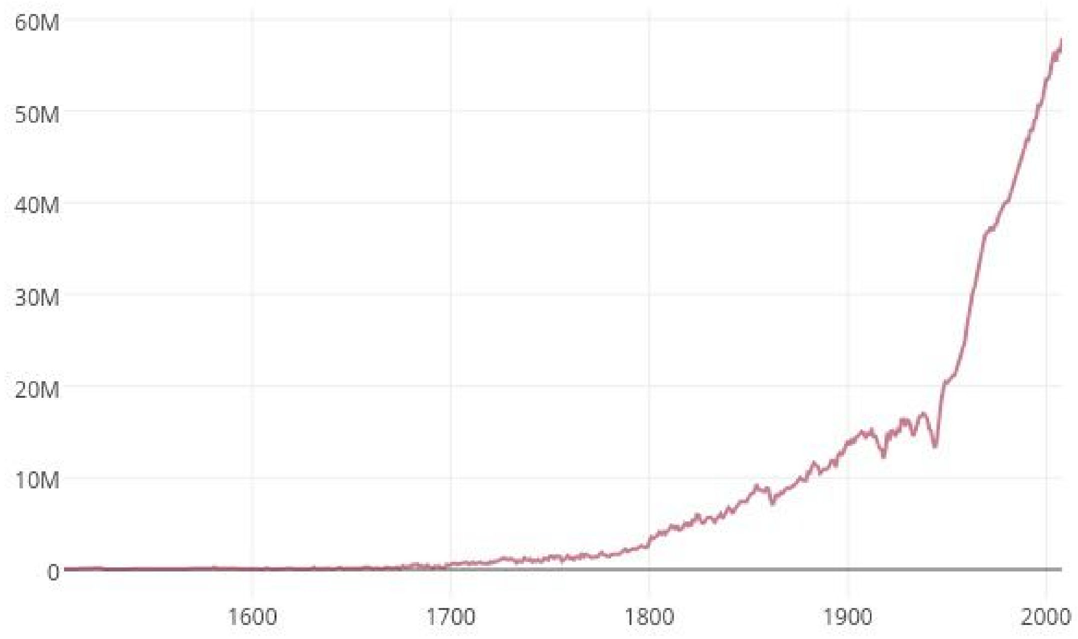
Each line in the data set is in the format of (‘word used’, year, number of times it appeared each year, number of books it appeared in that year). Since we need the number of syllables used per year. We need to calculate the number of syllables used in each word. Accordingly the map and reduce functions are implemented as below.

Map Function :- The map() function for this problem would read each line of the input files(Key - Long value generated by HDFS, Value - value of the string in each line). The mapper will split the line based on a delimiter and populate the year , word , counts (number of times, number of books) for each line. The mapper then uses an external library(click for the link to library), to calculate the number of syllables per each word in the line. Since we need to send the jar file over the network to reduce the size of the final jar, we have extracted the Syllable class from this library and added it to our jar, instead of adding the entire package to our library. The Syllable class provides us a static method that returns the number of syllables in a word. The map function ultimately outputs a map with the year as key and number of syllables in a given word, as the value.

Reduce Function :- The reduce() function for this problem would sum the number of the values for each distinct key(year). The final sum for each distinct key gives us the total number of syllables used per key(year). The reduce function would then output a map with year as the key and the sum of number of syllables used for that year as the value.

The output of reduce function can be found at
https://github.uc.edu/ghandisv/P3Hadoop/blob/master/Google%20ngram/output/third

The data from the reducer can be used to plot a graph of Number of Syllables used vs Year



60M
50M
40M
30M
20M
10M
0
1600
1700
1800
1900
2000