**Assignment 3**

**Part 1**

**Plot Word Cloud with Illiad and Odyssey**

A word cloud is a text mining method that allows us to highlight the most frequently used keywords in a paragraph of texts. It is also referred to as a text cloud or tag cloud. A text mining package (tm) and word cloud generator package (wordcloud) are available in R for helping us to analyze texts and to quickly visualize the keywords words as a word cloud.

Steps-:

1. Install All the Packages Related to Word Cloud Library.

# Install

* 1. install.packages("tm") # for text mining
  2. install.packages("tm\_map")
  3. install.packages("NLP")
  4. install.packages("SnowballC") # for text stemming
  5. install.packages("wordcloud") # word-cloud generator
  6. install.packages("RColorBrewer") # color palettes.
  7. Needed <- c("tm", "SnowballCC", "RColorBrewer", "ggplot2", "wordcloud", "biclust", "cluster", "igraph", "fpc")
  8. install.packages(Needed, dependencies=TRUE)
  9. install.packages("Rcampdf", repos = "http://datacube.wu.ac.at/", type = "source")

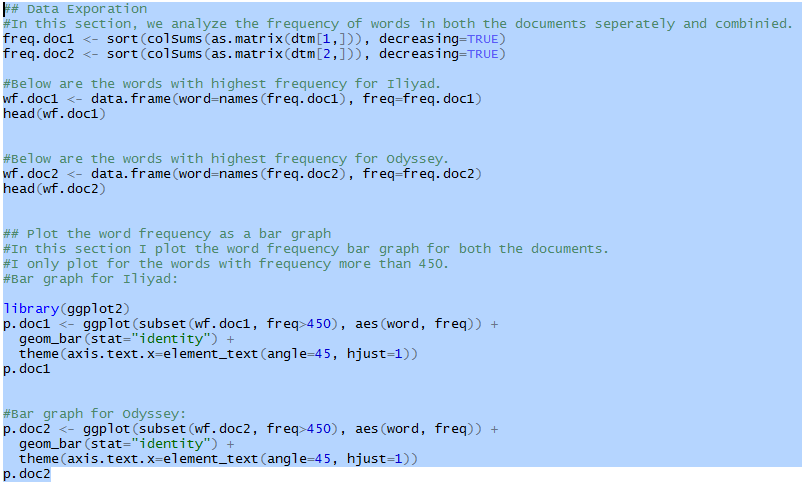
1. Load all the libraries that was installed.
   1. # Load
   2. library("NLP")
   3. library("tm")
   4. library("SnowballC")
   5. library("wordcloud")
   6. library("RColorBrewer")
2. First we are plotting for the file Illiad & Odyssey.

# Read the text file

* 1. filePath <- "C:/Users/rohit/Desktop/iliad.txt"
  2. text <- readLines(filePath)
  3. filePath <- "C:/Users/rohit/Desktop/odyssey.txt"
  4. text <- readLines(filePath)

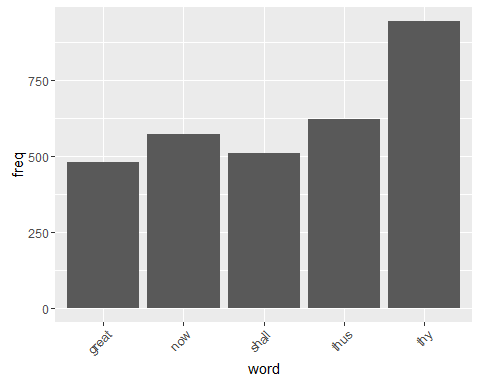
1. Load the Data to the corpus
   1. # Load the data as a corpus
   2. docs <- Corpus(VectorSource(text))
   3. inspect(docs)
2. Removing punctuation
   1. docs <- tm\_map(docs, removePunctuation)
   2. for(j in seq(docs))
   3. {
   4. docs[[j]] <- gsub("/", " ", docs[[j]])
   5. docs[[j]] <- gsub("@", " ", docs[[j]])
   6. docs[[j]] <- gsub("\\|", " ", docs[[j]])
   7. }
3. Remove Numbers
   1. docs <- tm\_map(docs, removeNumbers)
4. Convert to Lowercase
   1. docs <- tm\_map(docs, tolower)
5. Remove StopWords
   1. docs <- tm\_map(docs, removeWords, stopwords("english"))
6. Remove Words Ending Stem Words
   1. library(SnowballC)
   2. docs <- tm\_map(docs, stemDocument)
7. Remove WhiteSpaces
   1. docs <- tm\_map(docs, stripWhitespace)
8. Create a document term matrix (DTM) with Data Staging.
9. dtm <- DocumentTermMatrix(docs)
10. inspect(dtm)
11. dim(dtm)
12. tdm <- TermDocumentMatrix(docs)
13. tdm
14. freq <- colSums(as.matrix(dtm))
15. freq
16. ord <- order(freq)
17. Plot the word frequency as a bar graph

The plotting is done using the formula

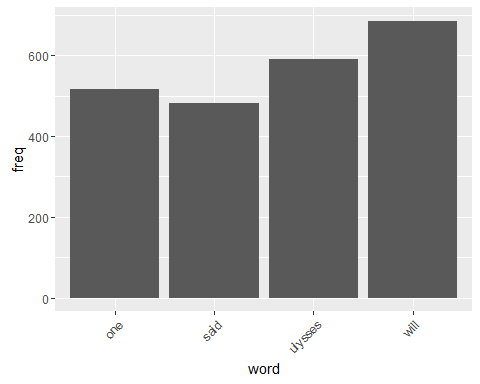


* 1. In this section I plot the word frequency bar graph for both the documents. I only plot for the words with frequency more than 450.

b- Illiad



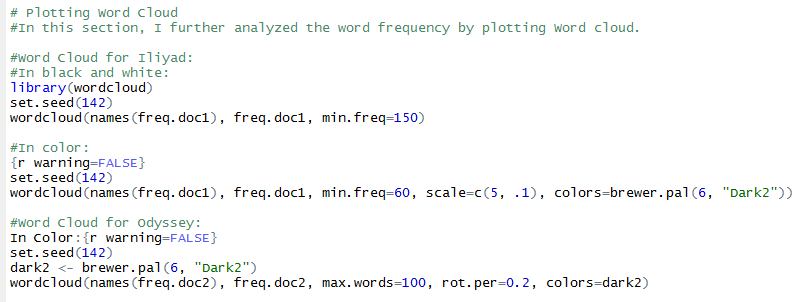
c- Odyssey



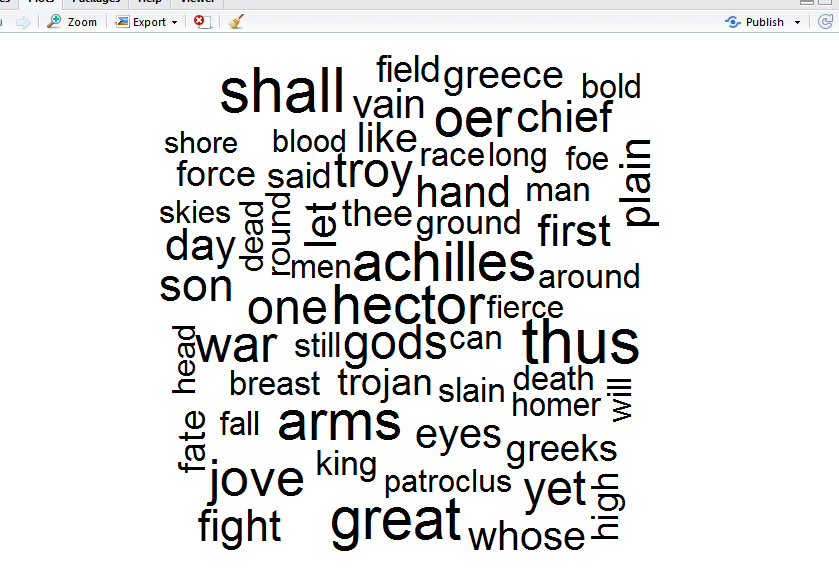
**Coloring**

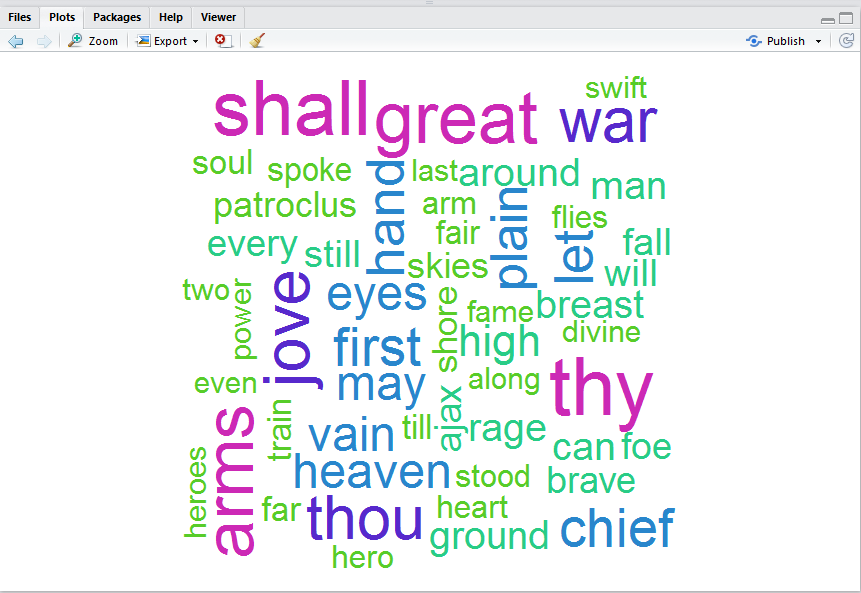
1. Plotting Word Cloud for Illiad and Odussey Black and White

This is done by using the formula -:



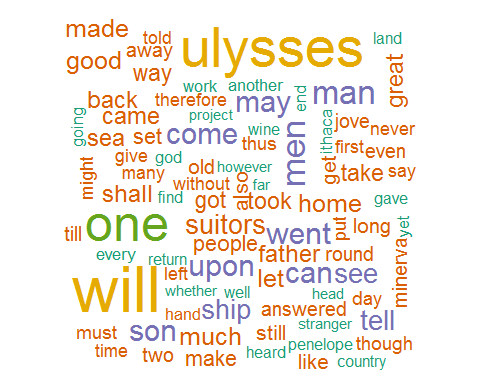
* 1. Illiad



1. Plotting Word Cloud for Illiad and Odussey for 400 words in Color.
   1. 

We can see that "thy" is the biggest word, hence it has the highest frequency.

* 1. Odyssey

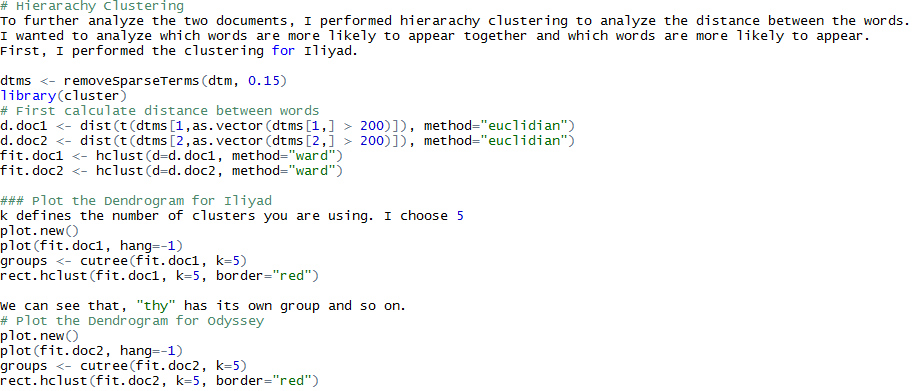


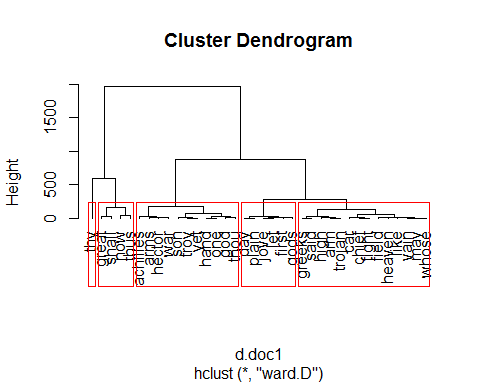
We can see that "ulysses" and "will" are the biggest words and of same color, hence they have the highest frequency.

**Hierarchy Clustering**

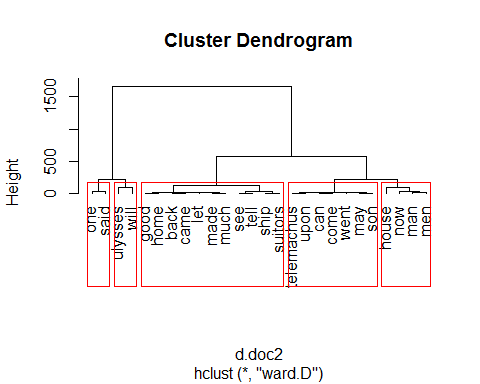
1. To further analyze the two documents, I performed hierarchy clustering to analyze the distance between the words. I wanted to analyze which words are more likely to appear together and which words are more likely to appear.
2. First, I performed the clustering for Iliad.
3. I plotted the Dindogram for illiad

By Using Formula -:





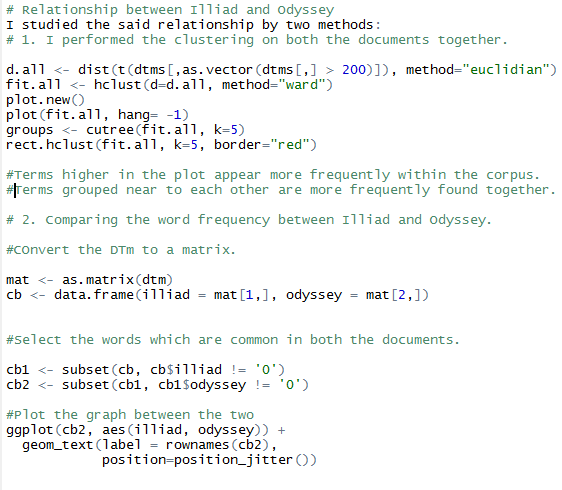
1. We can see that, "thy" has its own group and so on.
2. Now will do the same for Odyssey.



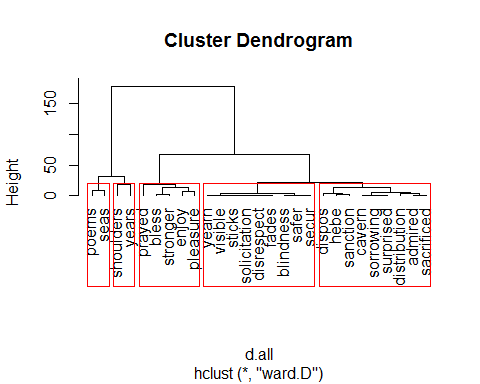
**Relationship b/w Iliad and Odyssey.**

1. Relationship is analyzed on the basis of two methods.

This calculated by using formula.



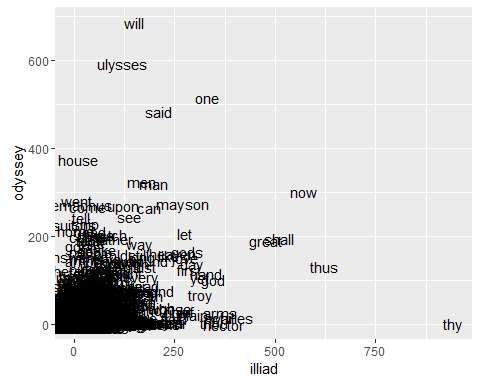
* 1. Cluster Dendogram



* 1. We can see
     1. Terms higher in the plot appear more frequently within the corpus.
     2. Terms grouped near to each other are more frequently found together.

1. Comparing the word Frequency between illiad and Odyssey.

This can be seen on the graph



We can observe that the

1. Word "thy" in the extreme right is occurring more in Iliad and significantly less in Odyssey.
2. the words "house", "ulysses" are occurring more in odyssey and less in Iliad.
3. The word jungle in the bottom left represents the words that are usually common in both the documents.

**Part 2**

1. Compute the factorial of 1,000,000 using Hadoop MapReduce. The numbers from 1 to 1,000,000 should be present in a file that you read in your code.

In this problem, I have created One Mapper and One Reducer.

Mapper –

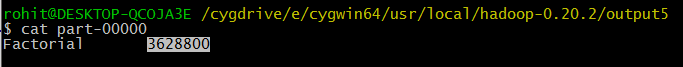
This read the number as value from the file and key is pre-defined.

Reducer-

Take key and value and perform multiplication on that numbers.

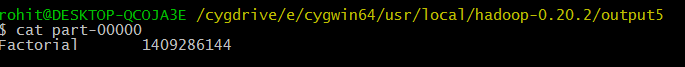
The file contains 1 to 1 million number but when I performed Multiplication on 1 million number the output was 0.

Then I limit my file to only 10 numbers and then tried to find the same factorial and the result was

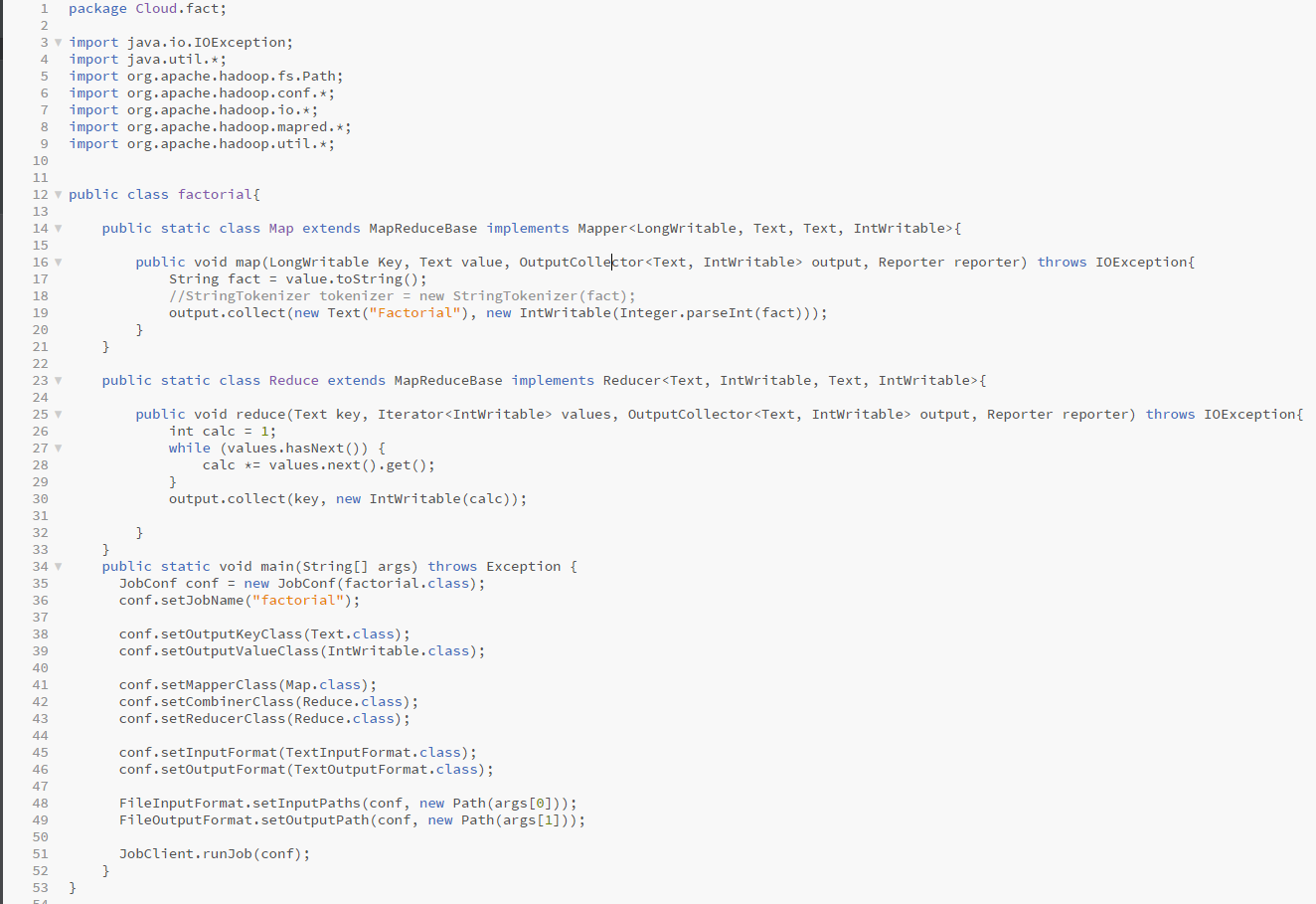


After I increase to 50 and result was the same 0

Then limiting it to 30 I found the result

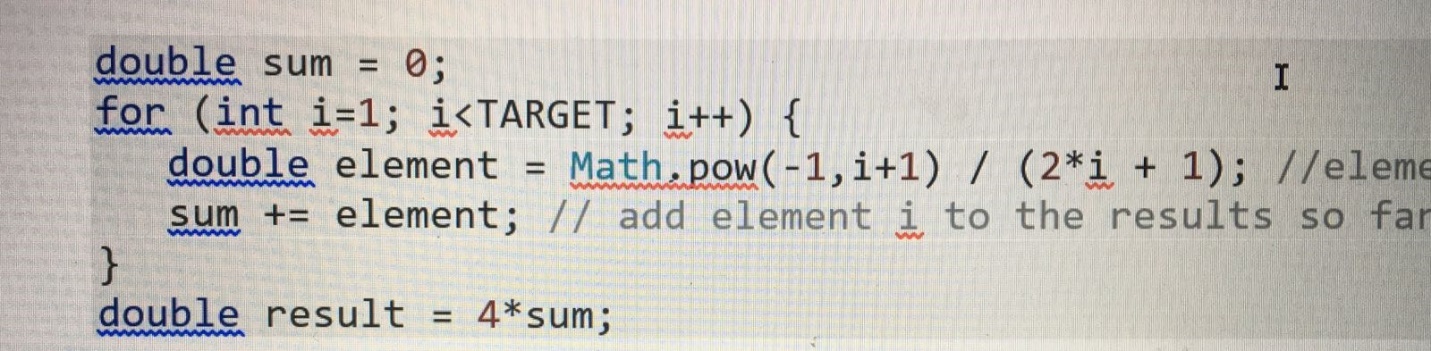


Attaching code



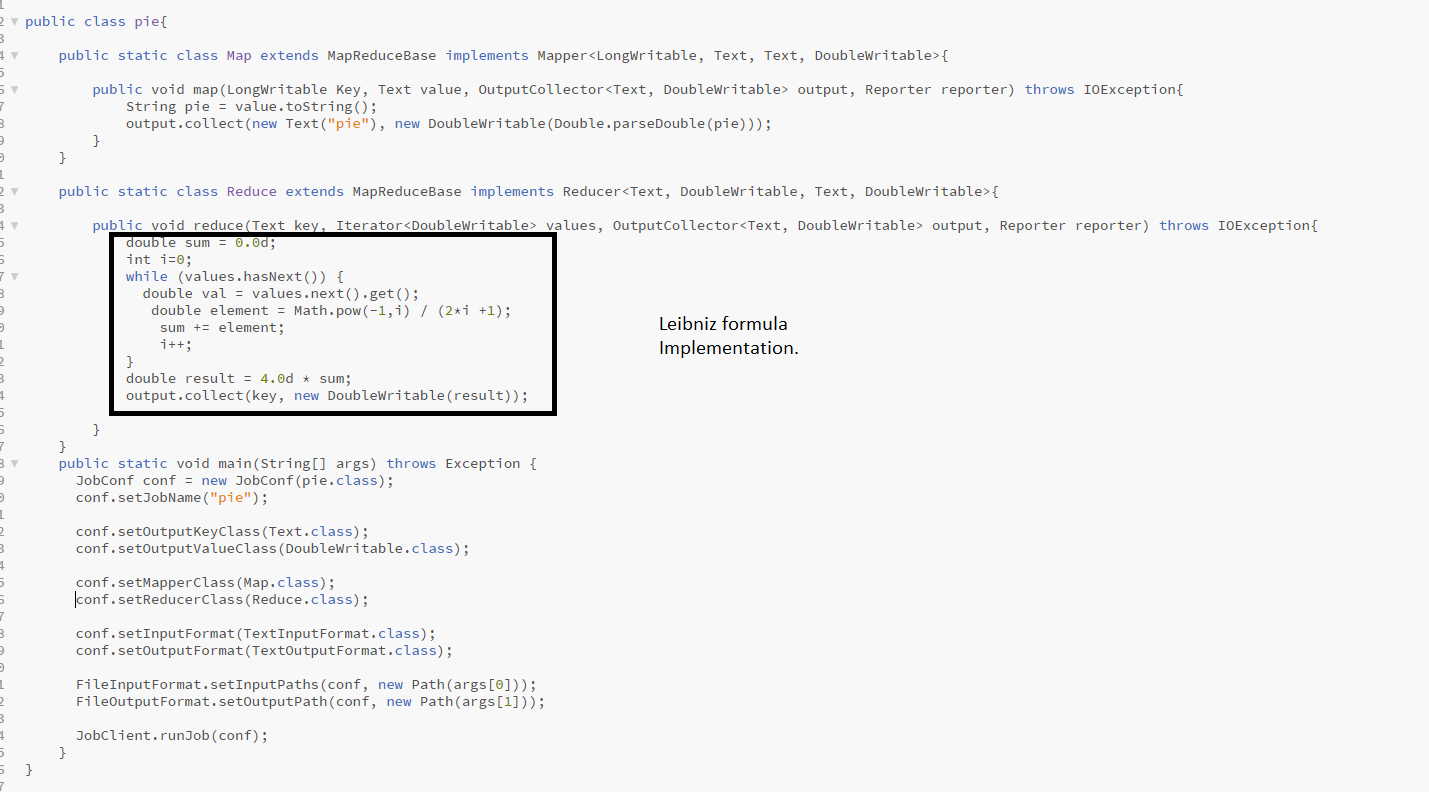
1. 2. Compute the value of pi using Hadoop MapReduce, leveraging the Leibniz formula. The numbers from 1 to 1,000,000 should be present in a file that you read in your code.

This can be calculated by using the Leibniz formula.



Iterating each value from file and calculating using this formula.

Attaching Code-:



O/P Generated-:

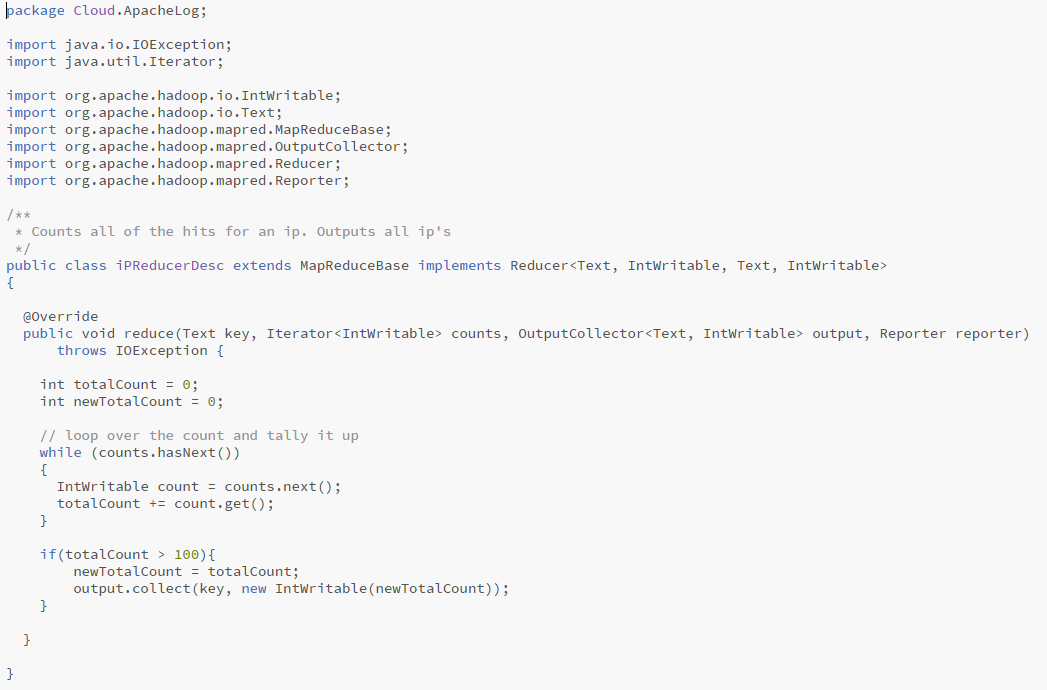


We can See value of pie = 3.14 in our output.

**Part 3**

Find all IP addresses with more than 100 hits ***in decreasing order.***You may *not*sort manually *nor*write a program to sort. You may need to run multiple map or reduce operations! Please refer to the slides to see the expected output format.

For the first part of the problem where IP address having count > 100 needs to be filtered, I modified the IpReducer.java and included the following condition.

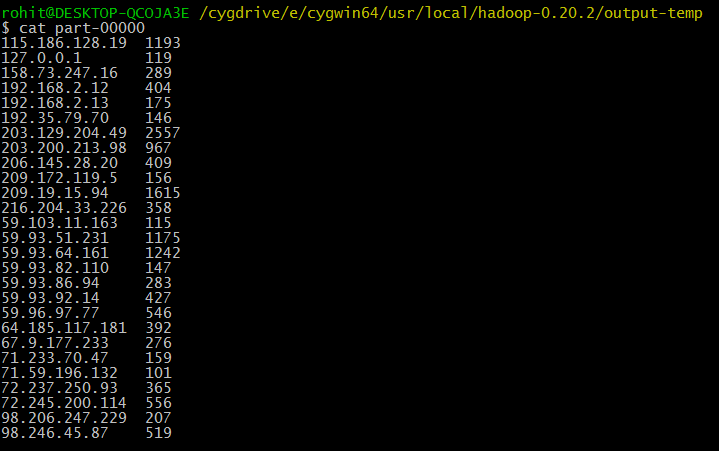


For the second part of the problem - the output needs to be sorted in decreasing order.

**Logic:**

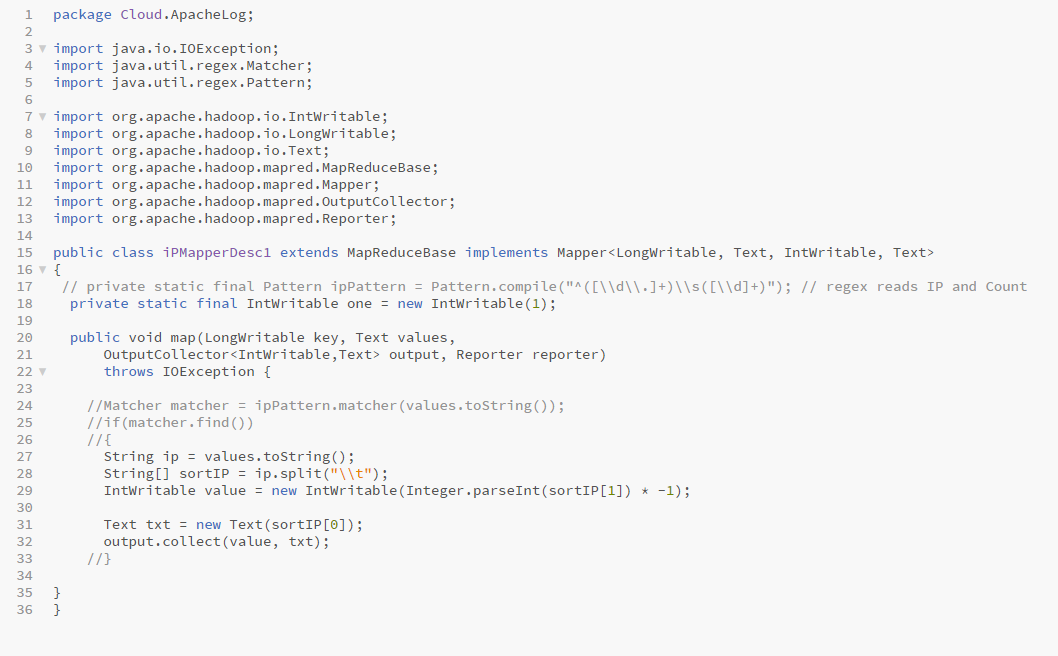
I created a second mapper and reducer. The output of the first reducer is input to the second mapper. The IP and count were passed as key-value pair in the second mapper. Then reversing the order and tricking the MapReduce, I passed the count as key and IP as value to the Reducer. This is necessary because MapReduce sorts the output based on key and not on value. Since by default it is sorted in ascending order, to sort it in descending order I made the count as negative before passing it to the second reducer. Finally, the count is made positive again before outputting.

Output of first mapper and Reducer



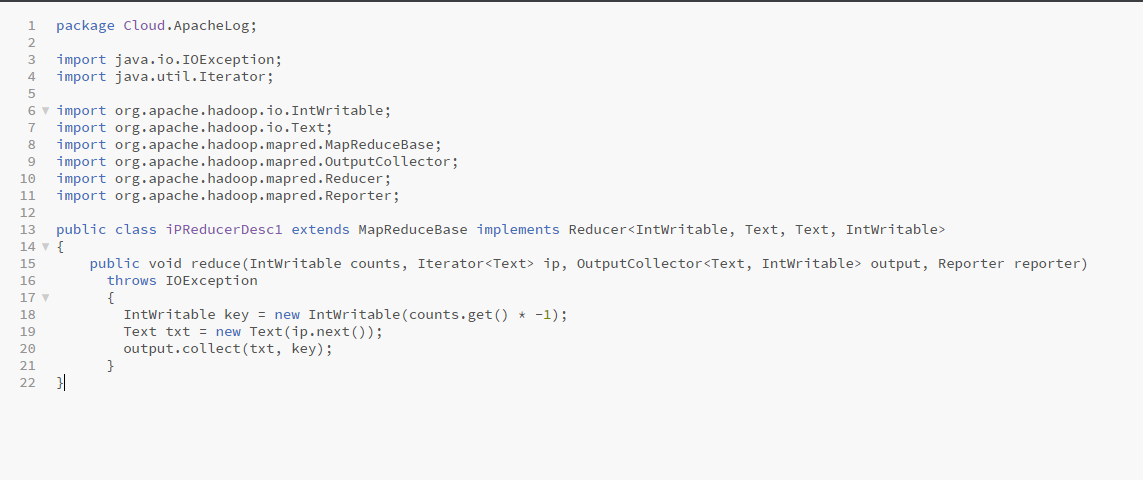
Now the task was to sort this IPS in Descending Order.

Second Mapper



This Mapper basically taking i/p and o/p of 1 reducer and multiplying the count of ip’s by -1.

Second Reducer

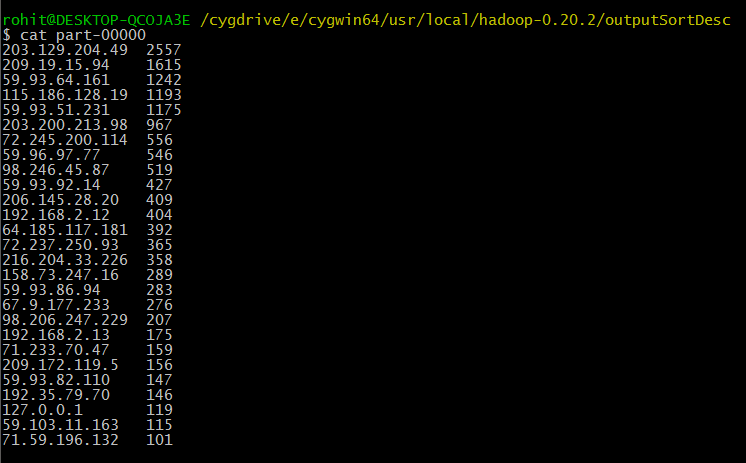


This Reducer taking i/p , o/p of second Mapper and again multiplying it by -1.

Now Runner which contains 2 jobs for each Mapper and Reducer.



After Running all these together the final output was



All IP’s get sorted in Descending Order.