**TFIDF calculation**

TF(t) = (Number of times(n) term t appears in a document) / (Total number of terms(N) in the document).

IDF(t) = log(Total number of documents / Number of documents with term t in it).

a) This program has three map reduce phases. First phase output is served as input to second phase and second phase output is served as input to third phase.

**FIRST PHASE**:

This phase reads input from text file and emits < (document number, word), n>

n 🡪 number of times word occurred in the document

**Pseudo code for Mapper first phase:**

Mapper emits < (document number, word), 1>

*Class Mapper(LongWritable, Text, Text, IntWritable)*

*Method map(LongWritable key, Text value)*

*String docPart [] = value.split(" "); //splitting input string to get individual words*

*`*

*String tempStr=""; //temp string to construct the key part*

*for(all words in docpart)*

*emit([word, document name], one);*

**Pseudo code for Reducer first phase**

Reducer emits < (document number, word), n>

*Class Mapper(LongWritable, Text, Text, IntWritable)*

*Method reduce(Text key, Int values)*

*for(all ones in values)*

*sum<- sum + one //sums all ones*

*emit (key,sum)*

key has [word, document name]

**SECOND PHASE**:

This phase input **< (document number, word), n>**

This phase calculates N, tf and normalised tf and emits them.

So output of this phase is **< (document number, word), (N, tf, normalised tf) >**

**Pseudo code for Mapper:**

Mapper input : < (document number, word), n>

Mapper emits < document number, (word, n)>

*Class Mapper(LongWritable, Text, Text, Text)*

*Method map(LongWritable key, Text value)*

*String docPart [] = value.toString.split("\t "); //splitting according to /t and,according to comma*

*emit(docpart[0], [docpart[1], docpart[2]]);*

**Pseudo code for Reducer second phase**

Reducer emits < (document number, word), n>

*Class Reducer(LongWritable, Text, Text, Text)*

*Method reduce(Text key, Text)*

*//copy keys mapallvalues hashmap and calculate N*

*for (Text text : values)*

*Map\_allValues<- values ;*

*N = N + values[1];*

*//calculate tf*

*for (text : all\_Map values)*

*tf\_arr<-text[1]/ N*

*//normalize tf and emit*

*for (text : all\_Map values)*

*emit(*(document number, word)**, (**N, tf, ,tf/max(tf\_arr))

**THIRD PHASE**:

This phase input **< (document number, word), (N, tf, normalised tf)>**

This phase calculates m, idf and tf\*idf

So output of this phase is **< (document number, word), (m, normalised tf, idf, tf\*normalised idf) >**

**Pseudo code for Mapper:**

Mapper input : < (document number, word), (N, tf, normalised tf) >

Mapper emits < word, (document number, tf, normalised tf ,1)>

*Class Mapper(LongWritable, Text, Text, Text)*

*Method map(LongWritable key, Text value)*

*String docPart [] = value.toString.split("\t "); //splitting according to /t and,according to comma*

*emit(*word, [document number, tf, normalised tf ,1]*);*

**Pseudo code for Reducer third phase**

Reducer emits < (document number, word), (m, normalised tf, idf, tf\*normalised idf) >

*Class Reducer(LongWritable, Text, Text, Text)*

*Method reduce(Text key, Text)*

*//copy keys mapallvalues hashmap and calculate m*

*for (Text text : values)*

*m = m + values[3];*

*//calculate idf*

*for (text : all\_Map values)*

*idf<- log(document count/ m)*

*//emit normalized tf, idf tf\*idf*

*for (text : all\_Map values)*

*emit(*([key**,** text[0], **(**text[1],text[2], tf\*idf)