Program – 6

AIM: Implement Flajolet-Martin Algorithm for counting distinct elements in stream data.

Introduction & Theory

Flajolet-Martin Algorithm

The Flajolet–Martin algorithm is an algorithm for approximating the number of distinct elements in a stream with a single pass and space-consumption which is logarithmic in the maximum number of possible distinct elements in the stream.

1. Create a bit vector (bit array) of sufficient length L, such that 2L>n, the number of elements in the stream. Usually a 64-bit vector is sufficient since 264 is quite large for most purposes.
2. The i-th bit in this vector/array represents whether we have seen a hash function value whose binary representation ends in 0i. So initialize each bit to 0.
3. Generate a good, random hash function that maps input (usually strings) to natural numbers.
4. Read input. For each word, hash it and determine the number of trailing zeros. If the number of trailing zeros is k, set the k-th bit in the bit vector to 1.
5. Once input is exhausted, get the index of the first 0 in the bit array (call this R). By the way, this is just the number of consecutive 1s (i.e. we have seen 0, 00, ..., 0^{R-1} as the output of the hash function) plus one.
6. Calculate the number of unique words as 2R/ϕ, where ϕ is 0.77351. A proof for this can be found in the original paper listed in the reference section.
7. The standard deviation of R is a constant: σ(R)=1.12. (In other words, R can be off by about 1 for 1-0.68=32% of the observations, off by 2 for about 1-0.95=5% of the observations, off by 3 for 1-0.997=0.3% of the observations using the Empirical rule of statistics). This implies that our count can be off by a factor of 2 for 32% of the observations, off by a factory of 4 for 5% of the observations, off by a factor of 8 for 0.3% of the observations and so on.

Below are the steps for MapReduce data flow:

- **Step 1:** One block is processed by one **mapper** at a time. In the mapper, a developer can specify his own business logic as per the requirements. In this manner, Map runs on all the nodes of the cluster and process the data blocks in parallel.
- **Step 2:** Output of Mapper also known as intermediate output is written to the local disk. An output of mapper is not stored on HDFS as this is temporary data and **writing on HDFS** will create unnecessary many copies.
- **Step 3:** Output of mapper is shuffled to **reducer** node (which is a normal slave node but reduce phase will run here hence called as reducer node). The shuffling/copying is a physical movement of data which is done over the network.

Program – 6

- **Step 4:** Once all the mappers are finished and their output is shuffled on reducer nodes then this intermediate output is merged & sorted. Which is then provided as input to reduce phase.
- **Step 5:** Reduce is the second phase of processing where the user can specify his own custom business logic as per the requirements. An input to a reducer is provided from all the mappers. An output of reducer is the final output, which is written on HDFS

Code

FM algo Class

```java
package algos;

import java.io.IOException;
import java.util.Collection;
import java.util.Collections;
import java.util.List;
import java.util.ArrayList;

public class FlajoletMartin {

    private static final double PHI = 0.77351D;
    private int hashFunctions;
    private HashFunction[] hashes;
    public FlajoletMartin(int numHashFunctions) {
        this.hashFunctions = numHashFunctions;
        hashes = new HashFunction[hashFunctions];
        generateHashes();

    }

    private void generateHashes() {
        for (int i = 0; i < hashFunctions; i++) {
            hashes[i] = new HashFunction(i);
        }
    }

    public int uniques(String str) {
        List<Integer> averageR = new ArrayList<Integer>();
        String[] stream = str.split(" ");
        for (int i = 0; i < hashFunctions; i++){
            int sumR = 0;
            for (String word : stream) {
                sumR =
Math.max(getFirstZero(hashes[i].getHash(word)), sumR);
            }
            averageR.add(sumR);
        }

        Collections.sort(averageR);
        double r = 0;
        int avgMid = averageR.size() / 2;
        if (averageR.size() % 2 == 0) {
            r = (averageR.get(avgMid) + averageR.get(avgMid+1))/2;
        } else {
```

Program – 6

```
                r = averageR.get(avgMid + 1);
            }
            return (int) (Math.pow(2, r));

        }

        private int getFirstZero(int val) {
            if (val == 0)
                return 0;
            int pos = 0;
            // counting the position of first set bit
            for (int i = 0; i < 64; i++) {
                if ((val & (1 << i))== 0)
                    pos++;

                else
                    break;
            }
            return pos;
        }
        private static class HashFunction {
            private int numLoop;

            public HashFunction(int loop) {
                this.numLoop = loop;
            }

            public int getHash(Object s) {
                int hash;
                int i = 0;
                do {
                    if (s instanceof String) {
                        hash = s.hashCode() * (i+1);
                        s = Integer.toString(hash);
                    } else if (s instanceof Number) {
                        hash = String.valueOf(s).hashCode() * (i+1);
                        s = Integer.toString(hash);
                    }
                    else {
                        String t = s.toString();
                        hash = t.hashCode() *(i+1);
                        s = Integer.toString(hash);
                    }
                    i += 1;
                } while (i < numLoop);
                return 13 + 7*Math.abs(hash);
            }

        }

    }

```

Program – 6

Mapper

```java
package hadoop;

import java.io.IOException;

import algos.*;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class FMMapper extends Mapper<LongWritable, Text, Text,
IntWritable>{
    private static final int EMPTY = 0;
    FlajoletMartin fm;

    @Override
    protected void setup(org.apache.hadoop.mapreduce.Mapper.Context
context)
            throws IOException, InterruptedException {
        super.setup(context);
        fm = new FlajoletMartin(4);
    }

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        String stream = value.toString();
        String id = stream.substring(0, 4);
        stream = stream.substring(5);
        int distinct = fm.uniques(stream);
        context.write(new Text(id), new IntWritable(distinct));

    }
}

```

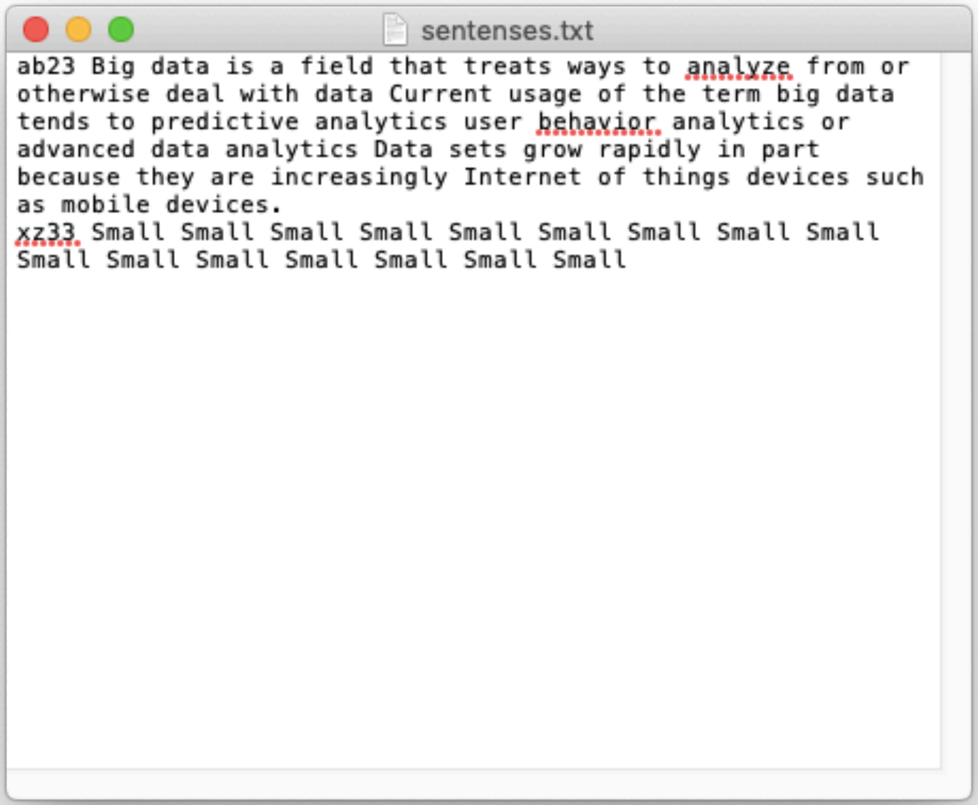
Main

```java
package hadoop;
import java.io.IOException;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Counter;
import org.apache.hadoop.mapreduce.CounterGroup;
import org.apache.hadoop.mapreduce.Counters;
import org.apache.hadoop.mapreduce.Job;

public class FMAlgo {
    public static void main(String[] args)

```

Program – 6

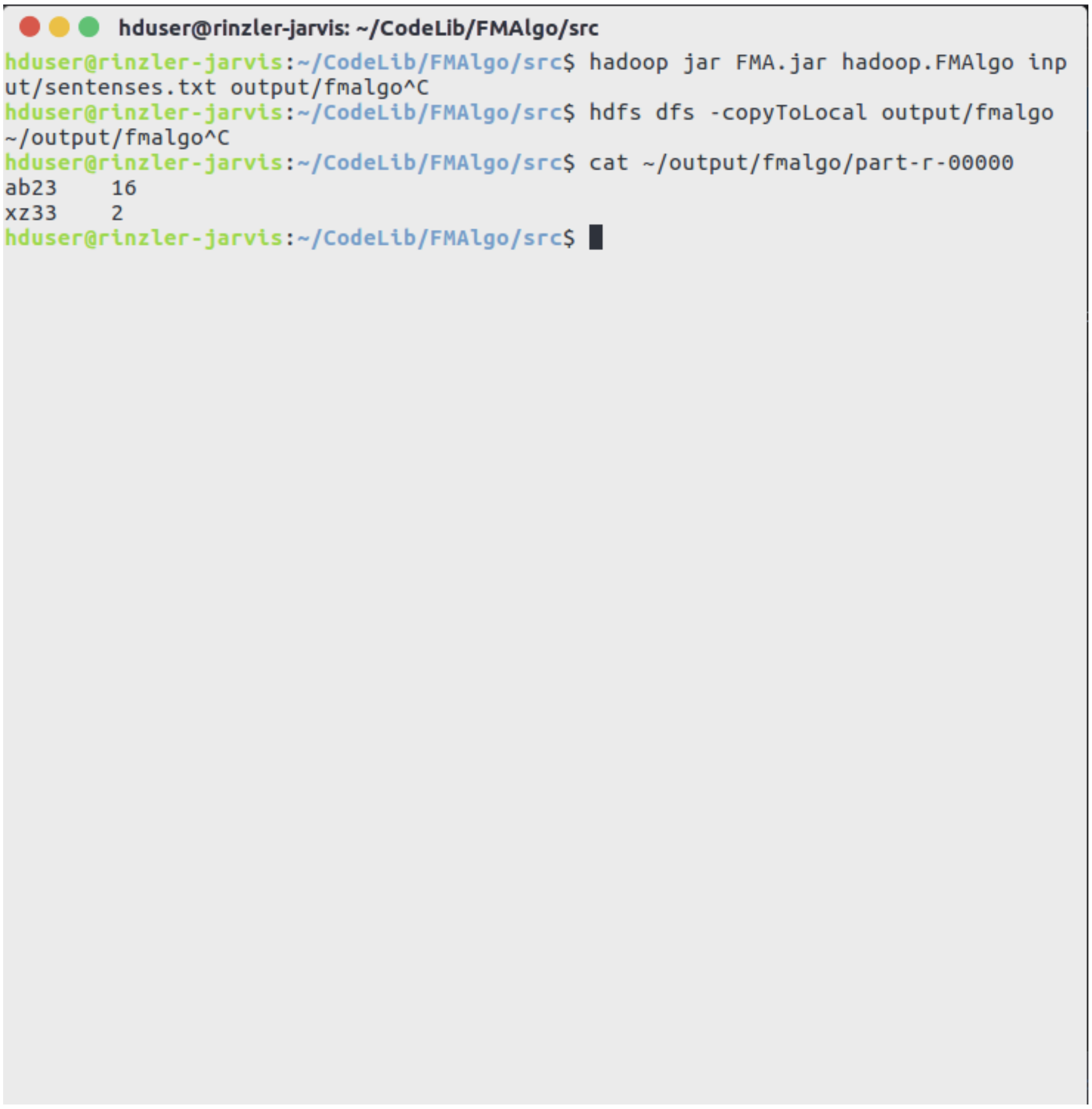
```
                throws IOException, ClassNotFoundException,
InterruptedException {
          if (args.length != 2) {
               System.err.println("Usage: FilterJob <input path> <output
path>");
               System.exit(-1);
          }
          Job job = new Job();
          job.setJarByClass(hadoop.FMMapper.class);
          job.setJobName("Multi stream FM algo");

          FileInputFormat.addInputPath(job, new Path(args[0]));
          FileOutputFormat.setOutputPath(job, new Path(args[1]));

          job.setMapperClass(hadoop.FMMapper.class);
          job.setOutputKeyClass(Text.class);
          job.setOutputValueClass(IntWritable.class);
          job.waitForCompletion(true);
     }
}
```

Output



```
ab23 Big data is a field that treats ways to analyze from or
otherwise deal with data Current usage of the term big data
tends to predictive analytics user behavior analytics or
advanced data analytics Data sets grow rapidly in part
because they are increasingly Internet of things devices such
as mobile devices.
xz33 Small Small Small Small Small Small Small Small Small
Small Small Small Small Small Small Small
```

Program – 6



```
hduser@rinzler-jarvis:~/CodeLib/FMAlgo/src$ hadoop jar FMA.jar hadoop.FMAlgo inp
ut/sentenses.txt output/fmalgo^C
hduser@rinzler-jarvis:~/CodeLib/FMAlgo/src$ hdfs dfs -copyToLocal output/fmalgo
~/output/fmalgo^C
hduser@rinzler-jarvis:~/CodeLib/FMAlgo/src$ cat ~/output/fmalgo/part-r-00000
ab23    16
xz33    2
hduser@rinzler-jarvis:~/CodeLib/FMAlgo/src$ 
```

Findings and Learnings:

1. We learned how map-reduce works.
2. We learned how to code using hadoop in Java.
3. We have learned the working of the flajolet martin algorithm.

We have successfully implemented flajolet martin algorithm.