Lab 02: Map Reduce Programming

1. WordCount Program

Step 1: Program's solution

- Import:

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

- Mapper:

```
  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```

- Reducer:

```
  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }
```

- Main:

```
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

- Explain:
- In the mapper method: This mapper will take as input an Object (representing the key), and a Text (representing the value) and use the StringTokenizer to separate the words in the value. Then it sends each word to the Reducer with a value of 1.
- In the reducer metho: This reducer will take the words from the Mapper and calculate the total number of occurrences of each word by adding the values of 1s together.

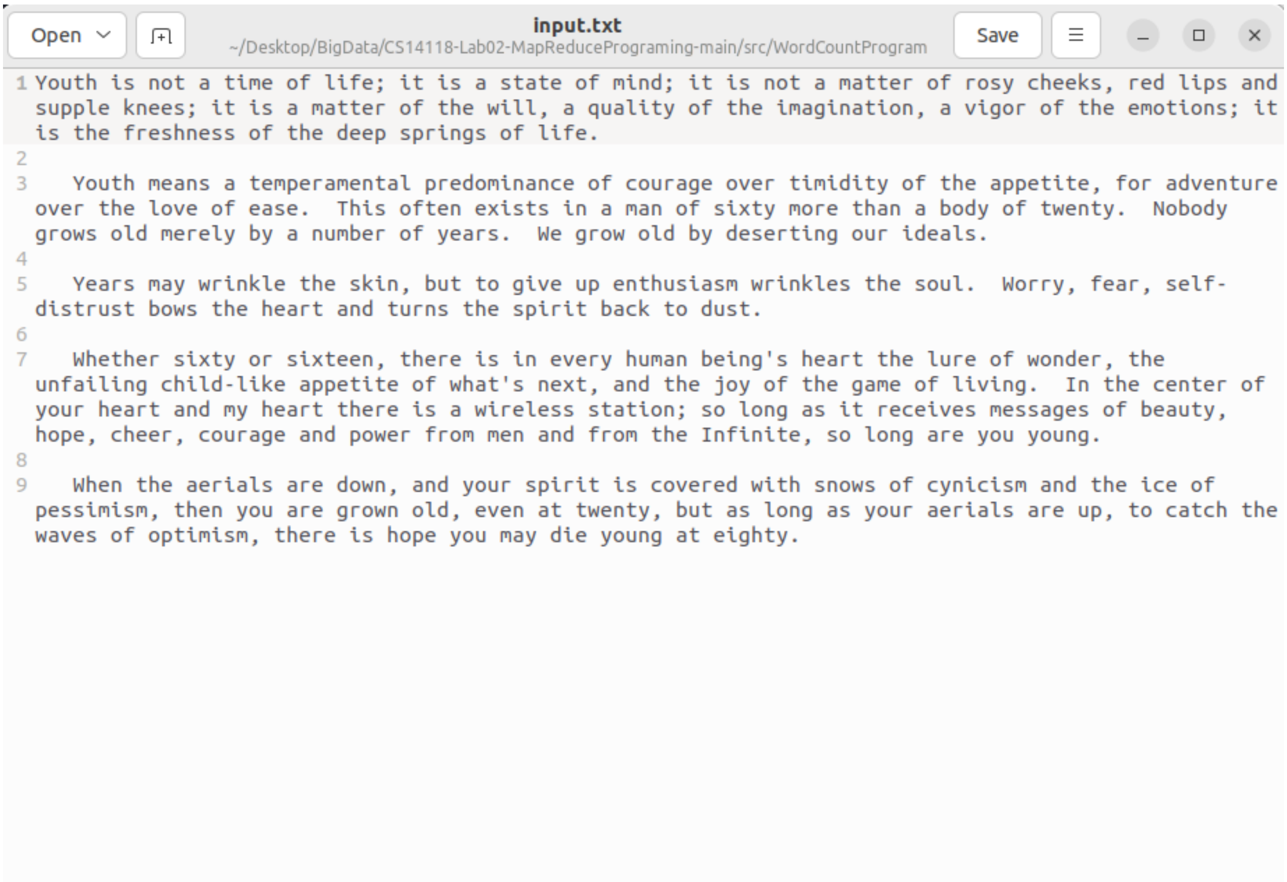
Step 2: Class Creation

```
jar -cvf WordCount.jar -C classes/ .
```

Step 3: Create directory structure for program in Hadoop

```
hadoop fs -mkdir /WordCount
hadoop fs -mkdir /WordCount/Input
hadoop fs -put 'local input file's path ' /WordCount/Input
```

- Example input:

- 
Open
input.txt
~/Desktop/BigData/CS14118-Lab02-MapReduceProgramming-main/src/WordCountProgram
Save

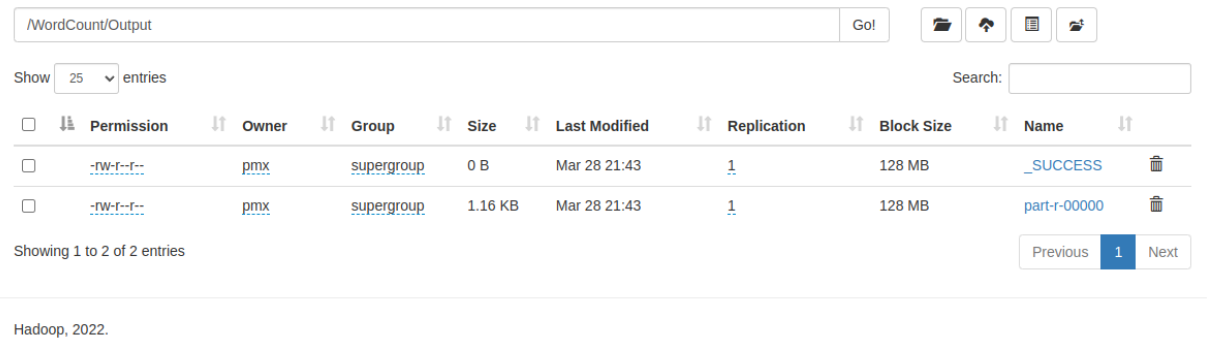
```
Youth is not a time of life; it is a state of mind; it is not a matter of rosy cheeks, red lips and supple knees; it is a matter of the will, a quality of the imagination, a vigor of the emotions; it is the freshness of the deep springs of life.

  Youth means a temperamental predominance of courage over timidity of the appetite, for adventure over the love of ease.  This often exists in a man of sixty more than a body of twenty.  Nobody grows old merely by a number of years.  We grow old by deserting our ideals.

  Years may wrinkle the skin, but to give up enthusiasm wrinkles the soul.  Worry, fear, self-distrust bows the heart and turns the spirit back to dust.

  Whether sixty or sixteen, there is in every human being's heart the lure of wonder, the unfailing child-like appetite of what's next, and the joy of the game of living.  In the center of your heart and my heart there is a wireless station; so long as it receives messages of beauty, hope, cheer, courage and power from men and from the Infinite, so long are you young.

  When the aerials are down, and your spirit is covered with snows of cynicism and the ice of pessimism, then you are grown old, even at twenty, but as long as your aerials are up, to catch the waves of optimism, there is hope you may die young at eighty.
```

Step 4: Create Jar File and deploy it to Hadoop

```
hadoop jar "Path to your local file .jar" WordCount /WordCount/Input
/WordCount/Output
```

Step 5: Final result

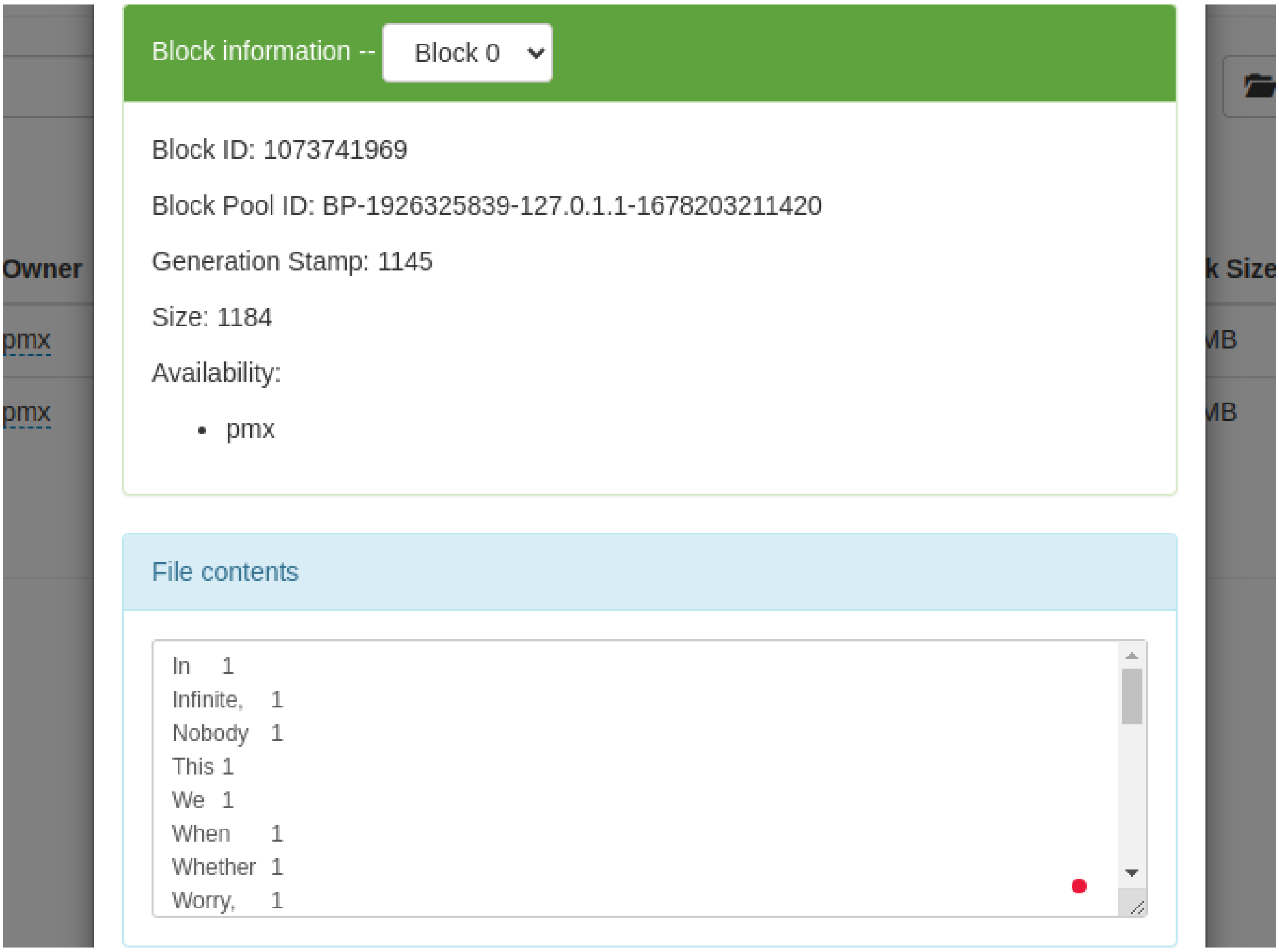
- After succesfully calculating, we can check our result in HDFS like below:

- 

☐	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
☐	-rw-r--r--	pmx	supergroup	0 B	Mar 28 21:43	1	128 MB	_SUCCESS	🗑
☐	-rw-r--r--	pmx	supergroup	1.16 KB	Mar 28 21:43	1	128 MB	part-r-00000	🗑

Showing 1 to 2 of 2 entries

Hadoop, 2022.

- 

Block ID: 1073741969

Block Pool ID: BP-1926325839-127.0.1.1-1678203211420

Generation Stamp: 1145

Size: 1184

Availability:

- pmx

File contents

```
In	1
Infinite,	1
Nobody	1
This 1
We	1
When	1
Whether	1
Worry,	1
```

---

2. WordSizeWordCount Program

Step 1: Program's solution

- Import:

```
import java.util.*;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

- Mapper:

```
  public static class Map extends Mapper<LongWritable, Text, Text,
IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {

                word.set(tokenizer.nextToken());
        String length = String.valueOf(word.getLength());
        Text len = new Text(length);
                context.write(len, one);
            }
        }
    }
```

- Reducer:

```
  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException

        {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }
```

- Main:

```
  public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
            Job job = new Job(conf, "WordSizeWordCount");
        job.setJarByClass(WordSizeWordCount.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.waitForCompletion(true);
    }
```

- Explain:
- In the mapper method: This mapper will take as input an Object (representing the key), and a Text (representing the value) and use the StringTokenizer to separate the words in the value. Then it sends each word's length to the Reducer with a value of 1.
- In the reducer metho: This reducer will take the word's length from the Mapper and calculate the total number of occurrences of each word's length by adding the values of 1s together.

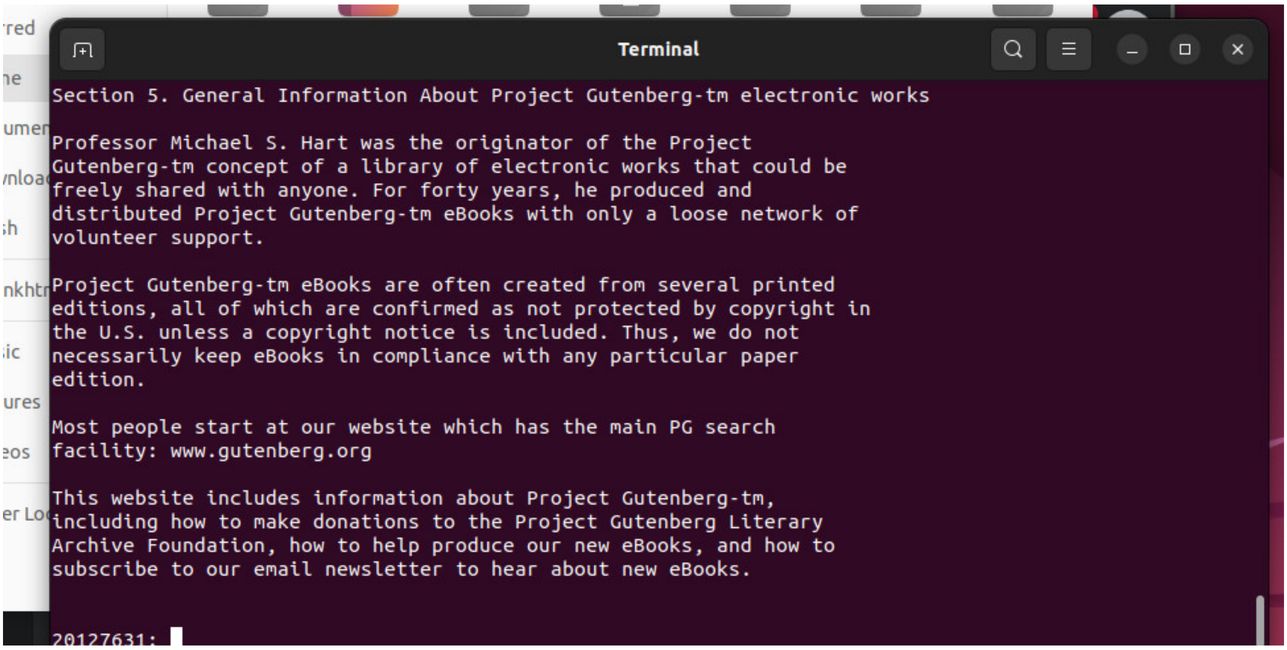
Step 2: Class Creation

```
jar -cvf WordSizeWordCount.jar -C classes/ .
```

Step 3: Create directory structure for program in Hadoop

```
hadoop fs -mkdir /WordSizeWordCount
hadoop fs -mkdir /WordSizeWordCount
hadoop fs -put 'local input file's path ' /WordSizeWordCount/Input
```

- Example input:



```
Section 5. General Information About Project Gutenberg-tm electronic works

Professor Michael S. Hart was the originator of the Project
Gutenberg-tm concept of a library of electronic works that could be
freely shared with anyone. For forty years, he produced and
distributed Project Gutenberg-tm eBooks with only a loose network of
volunteer support.

Project Gutenberg-tm eBooks are often created from several printed
editions, all of which are confirmed as not protected by copyright in
the U.S. unless a copyright notice is included. Thus, we do not
necessarily keep eBooks in compliance with any particular paper
edition.

Most people start at our website which has the main PG search
facility: www.gutenberg.org

This website includes information about Project Gutenberg-tm,
including how to make donations to the Project Gutenberg Literary
Archive Foundation, how to help produce our new eBooks, and how to
subscribe to our email newsletter to hear about new eBooks.

20127631:
```

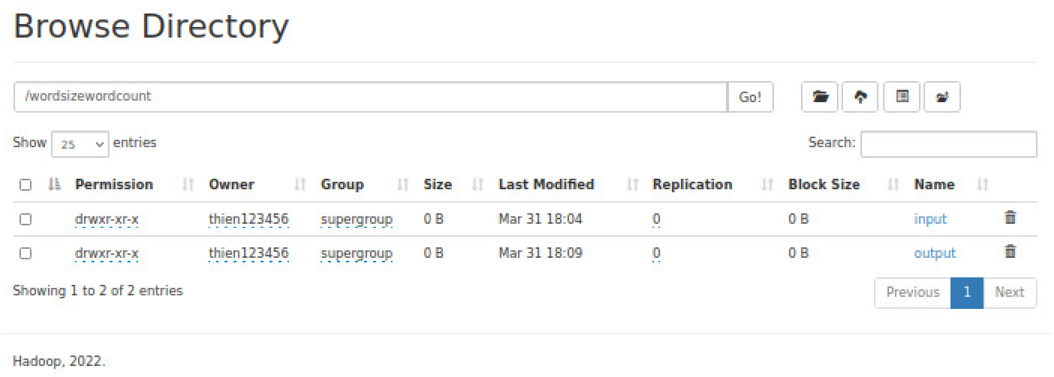
-

Step 4: Create Jar File and deploy it to Hadoop

```
hadoop jar "Path to your local file .jar" WordSizeWordCount
/WordSizeWordCount/Input /WordSizeWordCount/Output
```

Step 5: Final result

- After succesfully calculating, we can check our result in HDFS like below:



Browse Directory

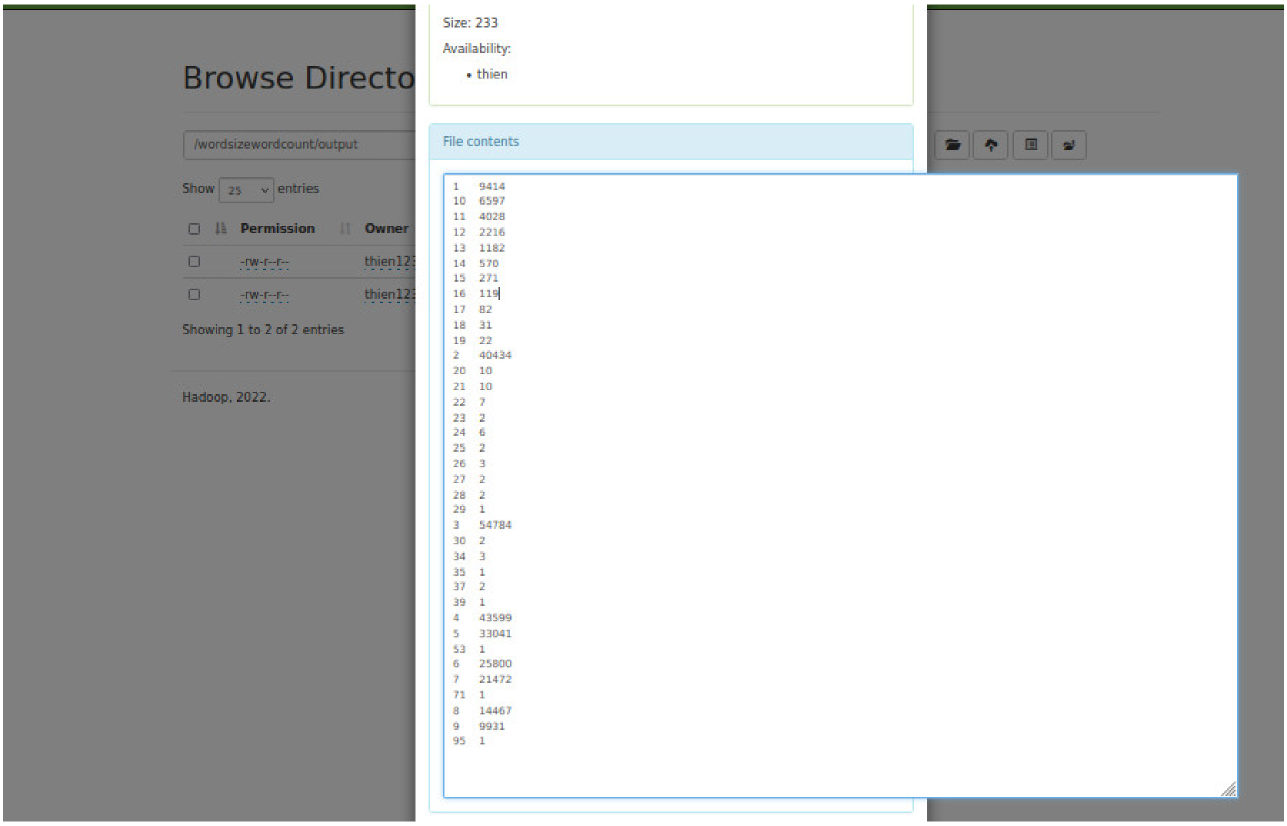
/wordsizewordcount

Show 25 entries Search:

☐	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
☐	drwxr-xr-x	thien123456	supergroup	0 B	Mar 31 18:04	0	0 B	input
☐	drwxr-xr-x	thien123456	supergroup	0 B	Mar 31 18:09	0	0 B	output

Showing 1 to 2 of 2 entries

Hadoop, 2022.

- 

Browse Directo

/wordsizewordcount/output

Show 25 entries

☐	Permission	Owner
☐	-rw-r--r--	thien123
☐	-rw-r--r--	thien123

Showing 1 to 2 of 2 entries

Hadoop, 2022.

Size: 233

Availability:

- thien

File contents

```
1	9414
10	6597
11	4028
12	2216
13	1182
14	570
15	271
16	119
17	82
18	31
19	22
2	40434
20	10
21	10
22	7
23	2
24	6
25	2
26	3
27	2
28	2
29	1
3	54784
30	2
34	3
35	1
37	2
39	1
4	43599
5	33041
53	1
6	25800
7	21472
71	1
8	14467
9	9931
95	1
```

-

---

3. WeatherData program

Step 1: Program's solution

- Import:

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

- Mapper:

```
  public static class Map
            extends Mapper<Object, Text, Text, Text>{

        public void map(Object key, Text value, Context context
        ) throws IOException, InterruptedException {
            String line = value.toString();
            String[] tokens = line.split("\\s+");
            String date = tokens[1];
            float tempMax = Float.parseFloat(tokens[6].trim());
            float tempMin = Float.parseFloat(tokens[7].trim());

            if(tempMax > 40.0) {
                context.write(new Text("Hot Day " + date), new
Text(String.valueOf(tempMax)));
            }
            if(tempMin < 10.0) {
                context.write(new Text("Cold Day " + date), new
Text(String.valueOf(tempMin)));
            }

        }
    }
```

- Reducer:

```
public static class Reduce
            extends Reducer<Text,Text,Text,Text>{

        public void reduce(Text key, Iterator<Text> values,
                           Context context
        ) throws IOException, InterruptedException {
            String temperature = values.next().toString();
            context.write(key, new Text(temperature));
        }
    }
```

- Main:

```
public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "weather");
        job.setJarByClass(WeatherData.class);
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
```

- Explain:
- In the mapper method: This mapper will take as input an Object (representing the key), and a Text (representing the value) and use the `split()` method to separate the fields in a line (inside is a regular expression indicate one or more whitespace). It then extract the date, minTemp and maxTemp (at index 1, 6, 7 respectively). Then, it check for the conditions (tempMax > 40 or tempMin < 10) and write the coressponding output to the context.
- In the reducer metho: This reducer will take the words from the Mapper and print out the result.

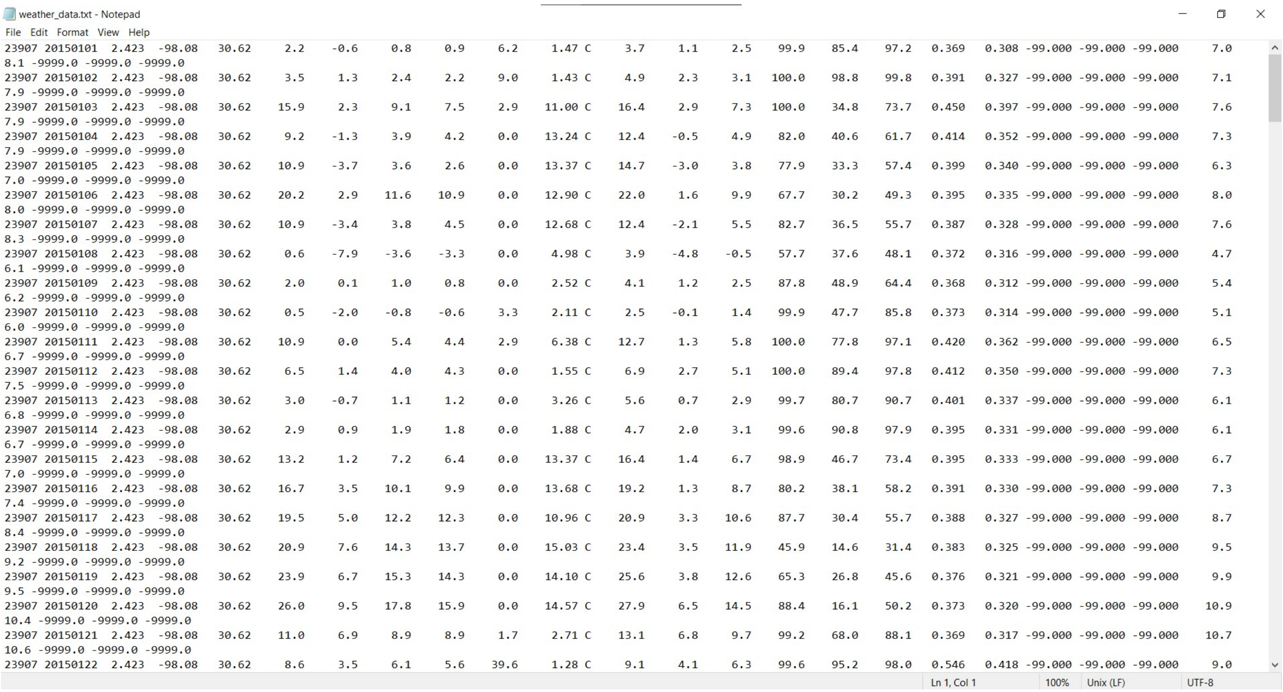
Step 2: Class Creation

```
jar -cvf WeatherData.jar -C classes/ .
```

Step 3: Create directory structure for program in Hadoop

```
hadoop fs -mkdir /Weather
hadoop fs -mkdir /Weather/Input
hadoop fs -put 'local input file's path ' /Weather/Input
```

- Example input:

- 

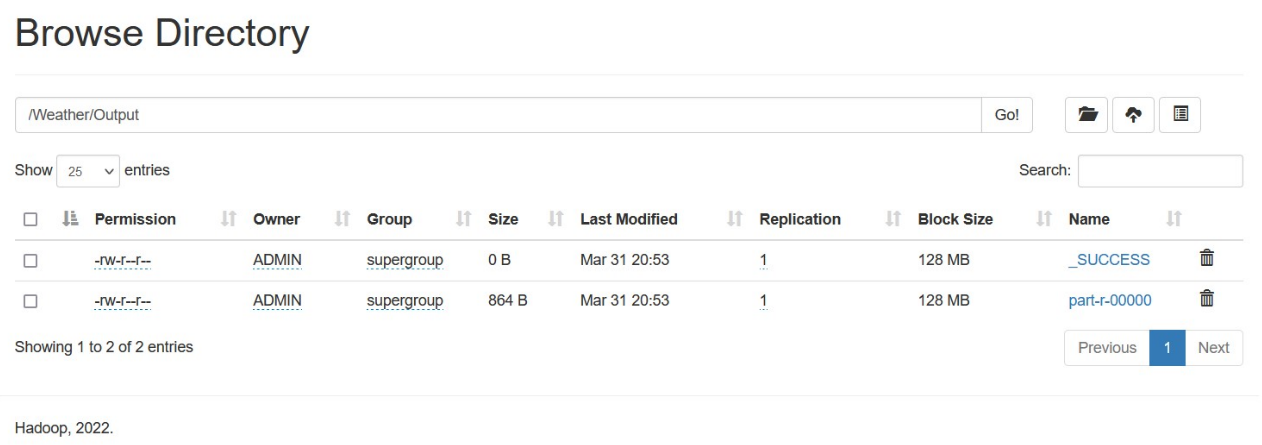
```
23907 20150101  2.423  -98.08   30.62     2.2    -0.6     0.8     0.9     6.2    1.47 C     3.7     1.1     2.5    99.9    85.4    97.2   0.369   0.308 -99.000 -99.000 -99.000     7.0
8.1 -9999.0 -9999.0 -9999.0
23907 20150102  2.423  -98.08   30.62     3.5     1.3     2.4     2.2     9.0    1.43 C     4.9     2.3     3.1   100.0    98.8    99.8   0.391   0.327 -99.000 -99.000 -99.000     7.1
7.9 -9999.0 -9999.0 -9999.0
23907 20150103  2.423  -98.08   30.62    15.9     2.3     9.1     7.5     2.9   11.00 C    16.4     2.9     7.3   100.0    34.8    73.7   0.450   0.397 -99.000 -99.000 -99.000     7.6
7.9 -9999.0 -9999.0 -9999.0
23907 20150104  2.423  -98.08   30.62     9.2    -1.3     3.9     4.2     0.0   13.24 C    12.4    -0.5     4.9    82.0    40.6    61.7   0.414   0.352 -99.000 -99.000 -99.000     7.3
7.9 -9999.0 -9999.0 -9999.0
23907 20150105  2.423  -98.08   30.62    10.9    -3.7     3.6     2.6     0.0   13.37 C    14.7    -3.0     3.8    77.9    33.3    57.4   0.399   0.340 -99.000 -99.000 -99.000     6.3
7.0 -9999.0 -9999.0 -9999.0
23907 20150106  2.423  -98.08   30.62    20.2     2.9    11.6    10.9     0.0   12.90 C    22.0     1.6     9.9    67.7    30.2    49.3   0.395   0.335 -99.000 -99.000 -99.000     8.0
8.0 -9999.0 -9999.0 -9999.0
23907 20150107  2.423  -98.08   30.62    10.9    -3.4     3.8     4.5     0.0   12.68 C    12.4    -2.1     5.5    82.7    36.5    55.7   0.387   0.328 -99.000 -99.000 -99.000     7.6
8.3 -9999.0 -9999.0 -9999.0
23907 20150108  2.423  -98.08   30.62     0.6    -7.9    -3.6    -3.3     0.0    4.98 C     3.9    -4.8    -0.5    57.7    37.6    48.1   0.372   0.316 -99.000 -99.000 -99.000     4.7
6.1 -9999.0 -9999.0 -9999.0
23907 20150109  2.423  -98.08   30.62     2.0     0.1     1.0     0.8     0.0    2.52 C     4.1     1.2     2.5    87.8    48.9    64.4   0.368   0.312 -99.000 -99.000 -99.000     5.4
6.2 -9999.0 -9999.0 -9999.0
23907 20150110  2.423  -98.08   30.62     0.5    -2.0    -0.8    -0.6     3.3    2.11 C     2.5    -0.1     1.4    99.9    47.7    85.8   0.373   0.314 -99.000 -99.000 -99.000     5.1
6.0 -9999.0 -9999.0 -9999.0
23907 20150111  2.423  -98.08   30.62    10.9     0.0     5.4     4.4     2.9    6.38 C    12.7     1.3     5.8   100.0    77.8    97.1   0.420   0.362 -99.000 -99.000 -99.000     6.5
6.7 -9999.0 -9999.0 -9999.0
23907 20150112  2.423  -98.08   30.62     6.5     1.4     4.0     4.3     0.0    1.55 C     6.9     2.7     5.1   100.0    89.4    97.8   0.412   0.350 -99.000 -99.000 -99.000     7.3
7.5 -9999.0 -9999.0 -9999.0
23907 20150113  2.423  -98.08   30.62     3.0    -0.7     1.1     1.2     0.0    3.26 C     5.6     0.7     2.9    99.7    80.7    90.7   0.401   0.337 -99.000 -99.000 -99.000     6.1
6.8 -9999.0 -9999.0 -9999.0
23907 20150114  2.423  -98.08   30.62     2.9     0.9     1.9     1.8     0.0    1.88 C     4.7     2.0     3.1    99.6    90.8    97.9   0.395   0.331 -99.000 -99.000 -99.000     6.1
6.7 -9999.0 -9999.0 -9999.0
23907 20150115  2.423  -98.08   30.62    13.2     1.2     7.2     6.4     0.0   13.37 C    16.4     1.4     6.7    98.9    46.7    73.4   0.395   0.333 -99.000 -99.000 -99.000     6.7
7.0 -9999.0 -9999.0 -9999.0
23907 20150116  2.423  -98.08   30.62    16.7     3.5    10.1     9.9     0.0   13.68 C    19.2     1.3     8.7    80.2    38.1    58.2   0.391   0.330 -99.000 -99.000 -99.000     7.3
7.4 -9999.0 -9999.0 -9999.0
23907 20150117  2.423  -98.08   30.62    19.5     5.0    12.2    12.3     0.0   10.96 C    20.9     3.3    10.6    87.7    30.4    55.7   0.388   0.327 -99.000 -99.000 -99.000     8.7
8.4 -9999.0 -9999.0 -9999.0
23907 20150118  2.423  -98.08   30.62    20.9     7.6    14.3    13.7     0.0   15.03 C    23.4     3.5    11.9    45.9    14.6    31.4   0.383   0.325 -99.000 -99.000 -99.000     9.5
9.2 -9999.0 -9999.0 -9999.0
23907 20150119  2.423  -98.08   30.62    23.9     6.7    15.3    14.3     0.0   14.10 C    25.6     3.8    12.6    65.3    26.8    45.6   0.376   0.321 -99.000 -99.000 -99.000     9.9
9.5 -9999.0 -9999.0 -9999.0
23907 20150120  2.423  -98.08   30.62    26.0     9.5    17.8    15.9     0.0   14.57 C    27.9     6.5    14.5    88.4    16.1    50.2   0.373   0.320 -99.000 -99.000 -99.000    10.9
10.4 -9999.0 -9999.0 -9999.0
23907 20150121  2.423  -98.08   30.62    11.0     6.9     8.9     8.9     1.7    2.71 C    13.1     6.8     9.7    99.2    68.0    88.1   0.369   0.317 -99.000 -99.000 -99.000    10.7
10.6 -9999.0 -9999.0 -9999.0
23907 20150122  2.423  -98.08   30.62     8.6     3.5     6.1     5.6    39.6    1.28 C     9.1     4.1     6.3    99.6    95.2    98.0   0.546   0.418 -99.000 -99.000 -99.000     9.0
```

Step 4: Create Jar File and deploy it to Hadoop

```
hadoop jar "Path to your local file .jar" WeatherData /Weather/Input
/Weather/Output
```

Step 5: Final result

- After succesfully calculating, we can check our result in HDFS like below:



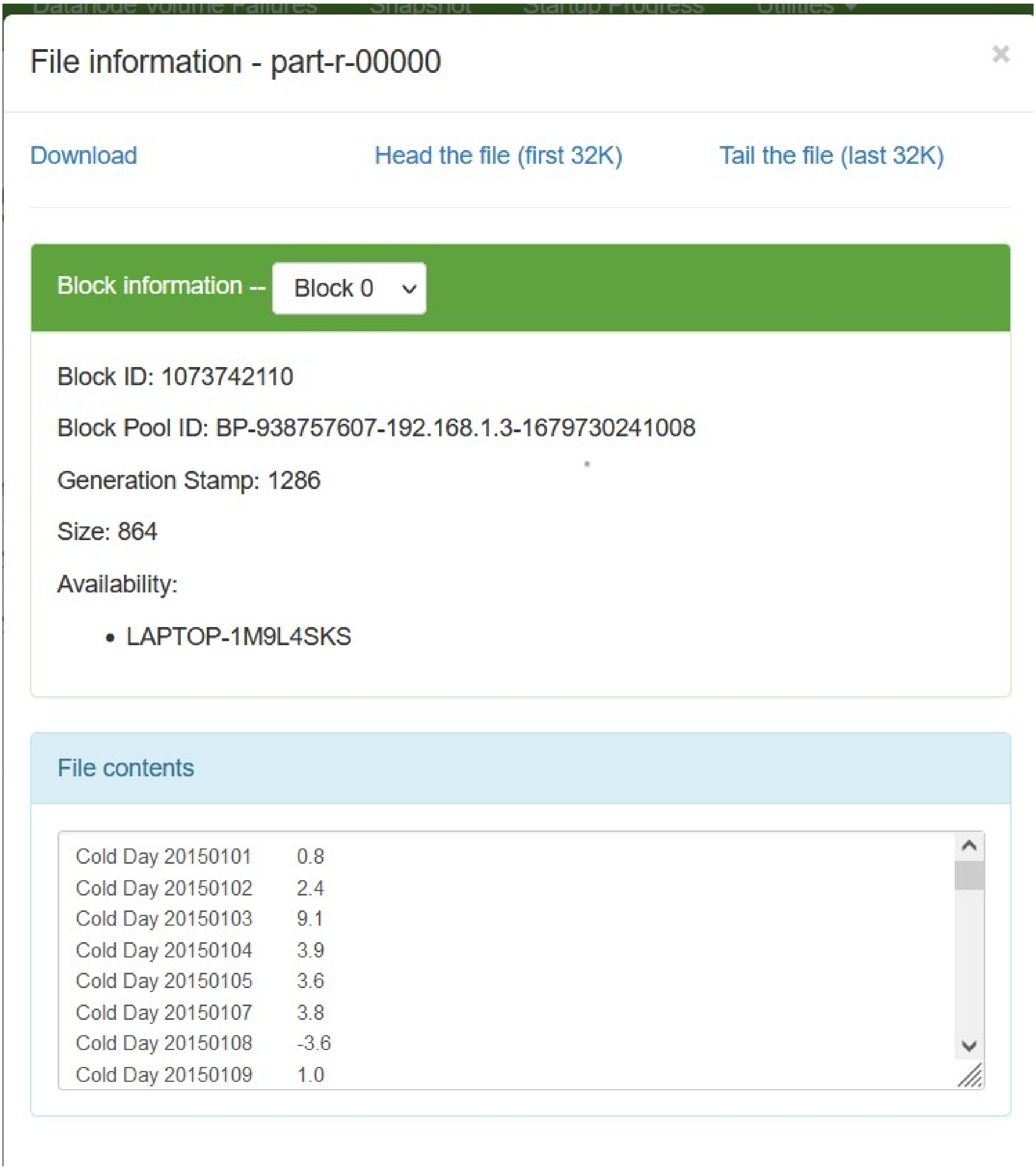
/Weather/Output

☐	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
☐	-rw-r--r--	ADMIN	supergroup	0 B	Mar 31 20:53	1	128 MB	_SUCCESS
☐	-rw-r--r--	ADMIN	supergroup	864 B	Mar 31 20:53	1	128 MB	part-r-00000

Showing 1 to 2 of 2 entries

Hadoop, 2022.

-

- 

File information - part-r-00000

Block ID: 1073742110

Block Pool ID: BP-938757607-192.168.1.3-1679730241008

Generation Stamp: 1286

Size: 864

Availability:

- LAPTOP-1M9L4SKS

File contents

```
Cold Day 20150101	0.8
Cold Day 20150102	2.4
Cold Day 20150103	9.1
Cold Day 20150104	3.9
Cold Day 20150105	3.6
Cold Day 20150107	3.8
Cold Day 20150108	-3.6
Cold Day 20150109	1.0
```

4. Patent Program

Step 1: Program's solution

- Mapper:

```
    public static class PatentMapper
            extends Mapper<Object, Text, Text, Text> {
        Text k = new Text();
        Text v = new Text();

        public void map(Object key, Text value, Context context) throws
```

```
IOException, InterruptedException {
            String line = value.toString();

            StringTokenizer tokenizer = new StringTokenizer(line, " ");
            while (tokenizer.hasMoreTokens()) {
                String token = tokenizer.nextToken();
                k.set(token);
                String token1 = tokenizer.nextToken();
                v.set(token1);
                context.write(k, v);
            }
        }
    }
```

- Reducer:

```
    public static class SumSubPatentReducer
            extends Reducer<Text, Text, Text, Text> {

        public void reduce(Text key, Iterable<Text> values,
                           Context context) throws IOException,
InterruptedException {
            int sum = 0;
            for (Text x : values) {
                sum++;
            }
            String result = Integer.toString(sum);
            context.write(key, new Text(result));
        }
    }
```

- Main:

```
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "patent program");
        job.setJarByClass(PatentProgram.class);

        job.setMapperClass(PatentMapper.class);
        job.setReducerClass(SumSubPatentReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
```

- The main idea for this program is that collecting pair of token in Map function, after combining, we count them through the their key and write in output file.

Step 2: Class Creation

- After complete code in Java, we need to generate file jar from builded classes by below command:

```
jar -cvf PatentProgram.jar -C classes/ .
```

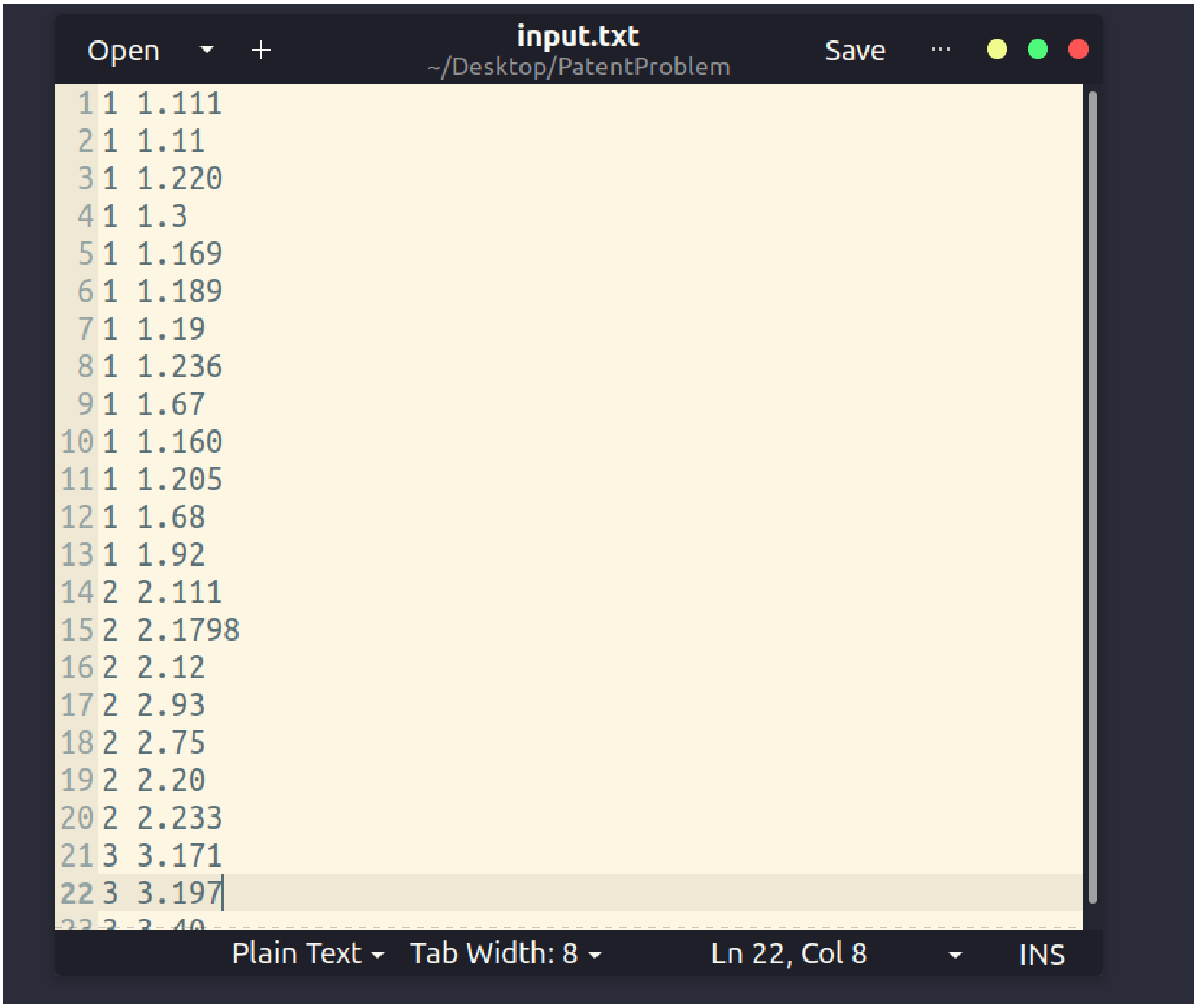
- Notice: Make sure that you export HADOOP_CLASSPATH before buiding file jar

Step 3: Create directory structure for program in Hadoop

- We need to create folder to store input data in HDFS by below command:

```
hadoop fs -mkdir /PatentProgram
hadoop fs -mkdir /PatentProgram/Input
hadoop fs -put "local input file's path" /PatentProgram/Input
```

- Example input:



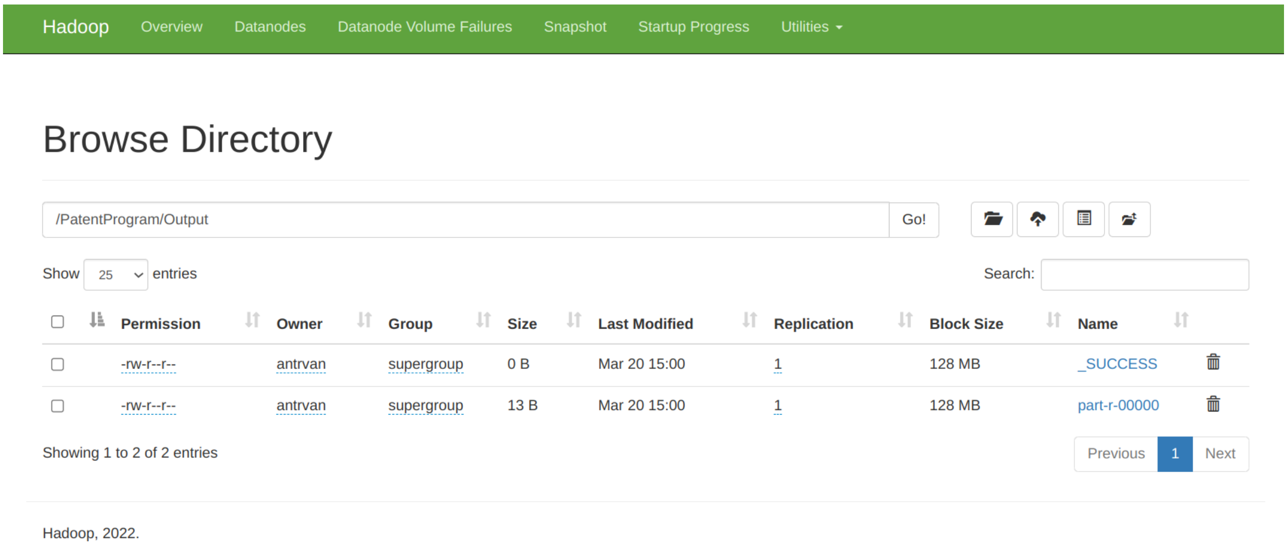
```
1 1.111
1 1.11
1 1.220
1 1.3
1 1.169
1 1.189
1 1.19
1 1.236
1 1.67
1 1.160
1 1.205
1 1.68
1 1.92
2 2.111
2 2.1798
2 2.12
2 2.93
2 2.75
2 2.20
2 2.233
3 3.171
3 3.197
3 3.40
```

Step 4: Create Jar File and deploy it to Hadoop

```
hadoop jar "Path to your local file .jar" PatentProgram
/PatentProgram/Input /PatentProgram/Output
```

Step 5: Final result

- After succesfully calculating, we can check our result in HDFS like below:



Browse Directory

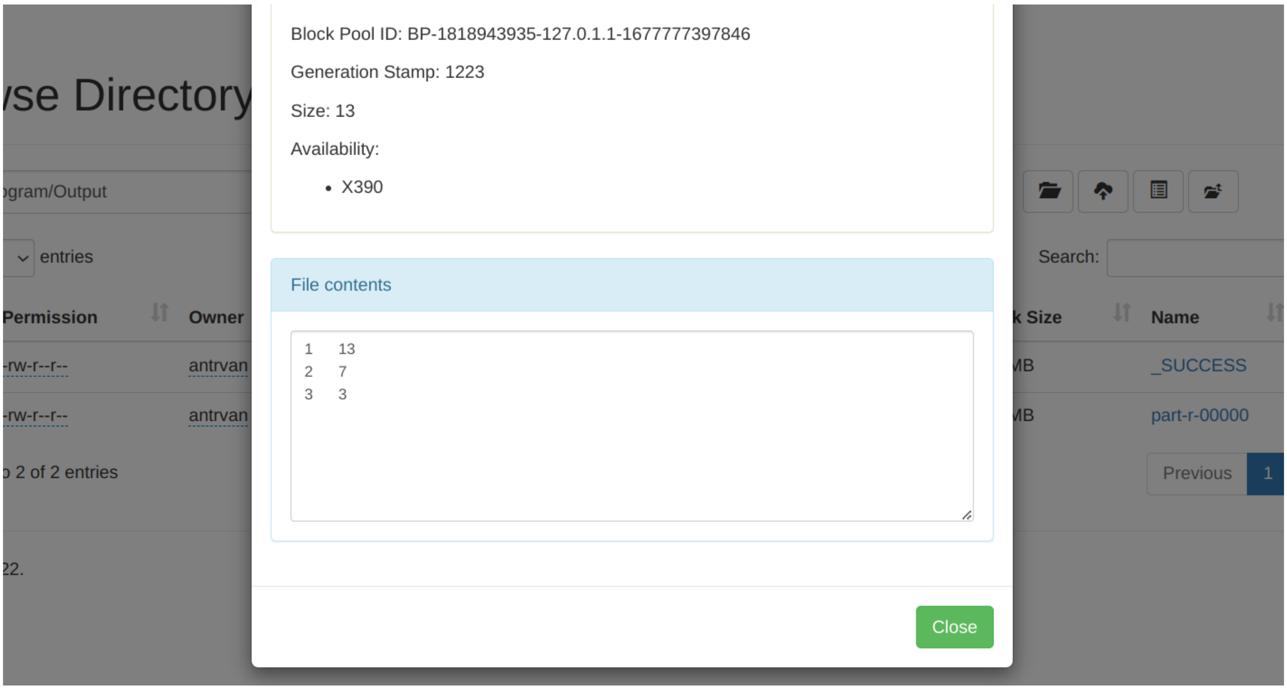
/PatentProgram/Output

□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	-rw-r--r--	antrvan	supergroup	0 B	Mar 20 15:00	1	128 MB	_SUCCESS
□	-rw-r--r--	antrvan	supergroup	13 B	Mar 20 15:00	1	128 MB	part-r-00000

Showing 1 to 2 of 2 entries

Hadoop, 2022.

-



Block Pool ID: BP-1818943935-127.0.1.1-1677777397846

Generation Stamp: 1223

Size: 13

Availability:

- X390

```
1	13
2	7
3	3
```

-

---

5. MaxTemp Program

Step 1: Program's solution

- Import:

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

- Mapper:

```
  public static class MaxTemperatureMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private static final int MISSING = 9999;

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      String line = value.toString();
      String year = line.substring(0, 4);
      int airTemperature = Integer.parseInt(line.substring(5));
      context.write(new Text(year), new IntWritable(airTemperature));
    }
  }
```

- Reducer:

```
  public static class MaxTemperatureReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int maxTemperature = Integer.MIN_VALUE;
      for (IntWritable value : values) {
        maxTemperature = Math.max(maxTemperature, value.get());
      }
      context.write(key, new IntWritable(maxTemperature));
    }
  }
```

- Main:

```
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "max temperature");
    job.setJarByClass(MaxTemp.class);
    job.setMapperClass(MaxTemperatureMapper.class);
    job.setCombinerClass(MaxTemperatureReducer.class);
    job.setReducerClass(MaxTemperatureReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

- Explain:
- In the mapper method, we extract the year and temperature from each input line and write them to the key/value pair. We do not need to verify the format of the input stream because in this case all the lines have the same format and we can simply use fixed indexes to extract the information.
- In the reduce method, we find the highest temperature for each year by traversing the list of pooled values for the same key.

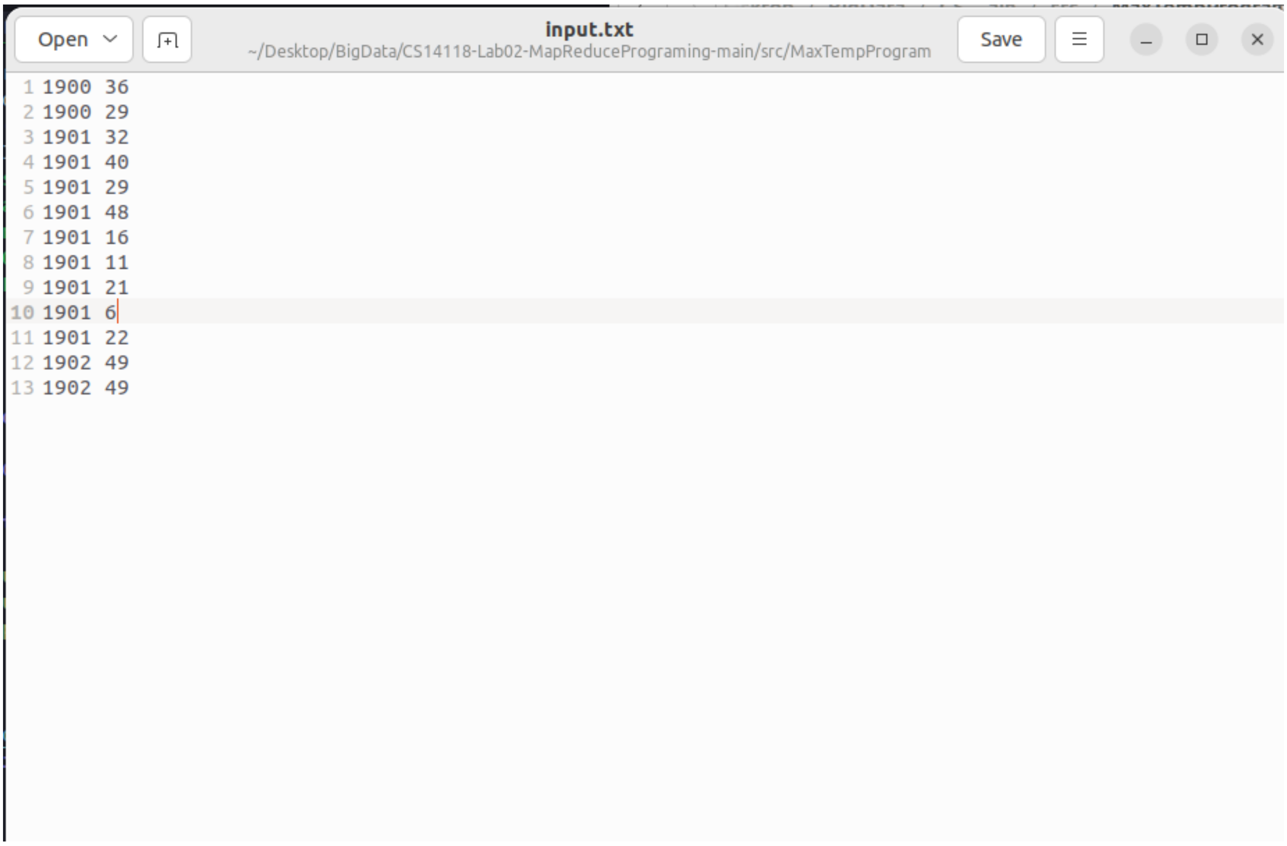
Step 2: Class Creation

```
jar -cvf MaxTemp.jar -C classes/ .
```

Step 3: Create directory structure for program in Hadoop

```
hadoop fs -mkdir /MaxTemp
hadoop fs -mkdir /MaxTemp/Input
hadoop fs -put 'local input file's path ' /MaxTemp/Input
```

- Example input:

- 
Open
input.txt
~/Desktop/BigData/CS14118-Lab02-MapReducePrograming-main/src/MaxTempProgram
Save

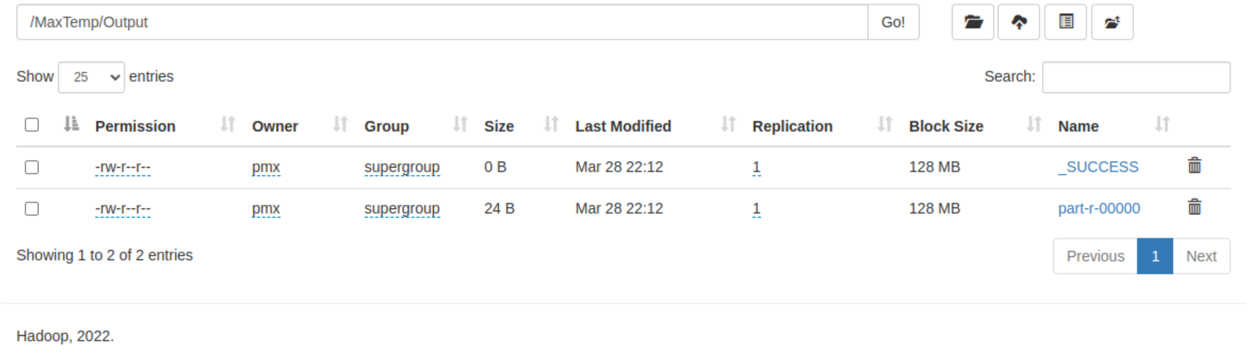
```
1900 36
1900 29
1901 32
1901 40
1901 29
1901 48
1901 16
1901 11
1901 21
1901 6
1901 22
1902 49
1902 49
```

Step 4: Create Jar File and deploy it to Hadoop

```
hadoop jar "Path to your local file .jar" MaxTemp /MaxTemp/Input
/MaxTemp/Output
```

Step 5: Final result

- After succesfully calculating, we can check our result in HDFS like below:

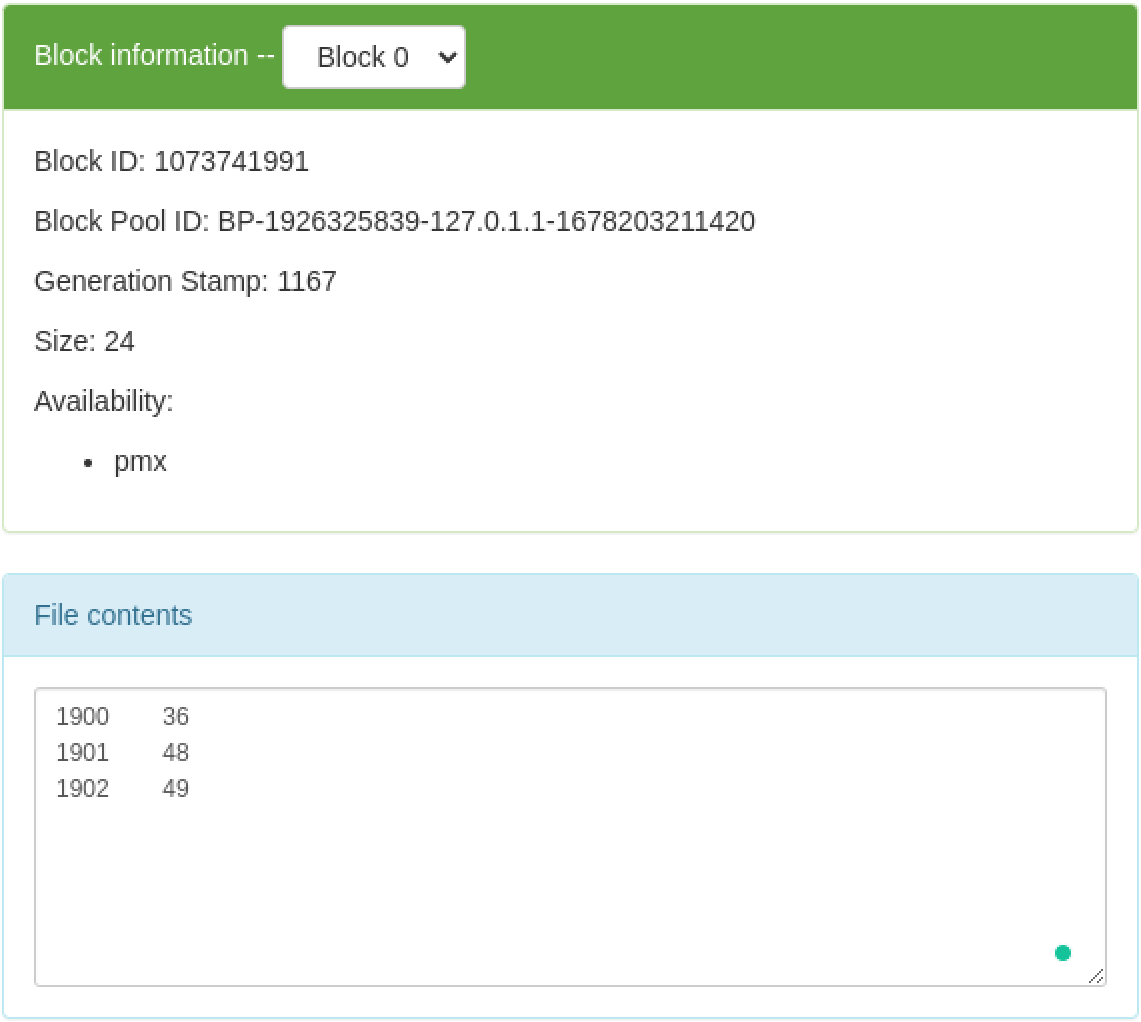


□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
□	-rw-r--r--	pmx	supergroup	0 B	Mar 28 22:12	1	128 MB	_SUCCESS	🗑
□	-rw-r--r--	pmx	supergroup	24 B	Mar 28 22:12	1	128 MB	part-r-00000	🗑

Showing 1 to 2 of 2 entries

Hadoop, 2022.

-

- 

Block ID: 1073741991

Block Pool ID: BP-1926325839-127.0.1.1-1678203211420

Generation Stamp: 1167

Size: 24

Availability:

- pmx

```
1900	36
1901	48
1902	49
```

6. AverageSalary Program

Step 1: Program's solution

- Mapper:

```
    public static class AvgMapper
            extends Mapper<Object, Text, Text, FloatWritable> {

        private Text id = new Text();
        private FloatWritable salary = new FloatWritable();

        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String[] values = value.toString().split("\t");
            id.set(values[0]);

            salary.set(Float.parseFloat(values[2]));
            context.write(id, salary);
        }
    }
```

- Reducer

```
    public static class AvgReducer
            extends Reducer<Text, FloatWritable, Text, FloatWritable> {

        private FloatWritable result = new FloatWritable();

        public void reduce(Text key, Iterable<FloatWritable> values,
                                 Context context) throws IOException,
InterruptedException {
            float totalSalary = 0;
            int numberPersons = 0;
            for (FloatWritable salary : values) {
                totalSalary += salary.get();
                numberPersons++;
            }

            result.set(totalSalary/numberPersons);
            context.write(key, result);
        }
    }
```

- Main

```
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "average salary");
        job.setJarByClass(AverageSalary.class);

        job.setMapperClass(AvgMapper.class);
        job.setReducerClass(AvgReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(FloatWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
```

- In Map function, we will collect employee's ID with their salary to make a pair. Then in Reducer, we will take average salary of each employee's ID to write in the output.

Step 2: Class Creation

- After complete code in Java, we need to generate file jar from builded classes by below command:

```
jar -cvf AverageSalary.jar -C classes/ .
```

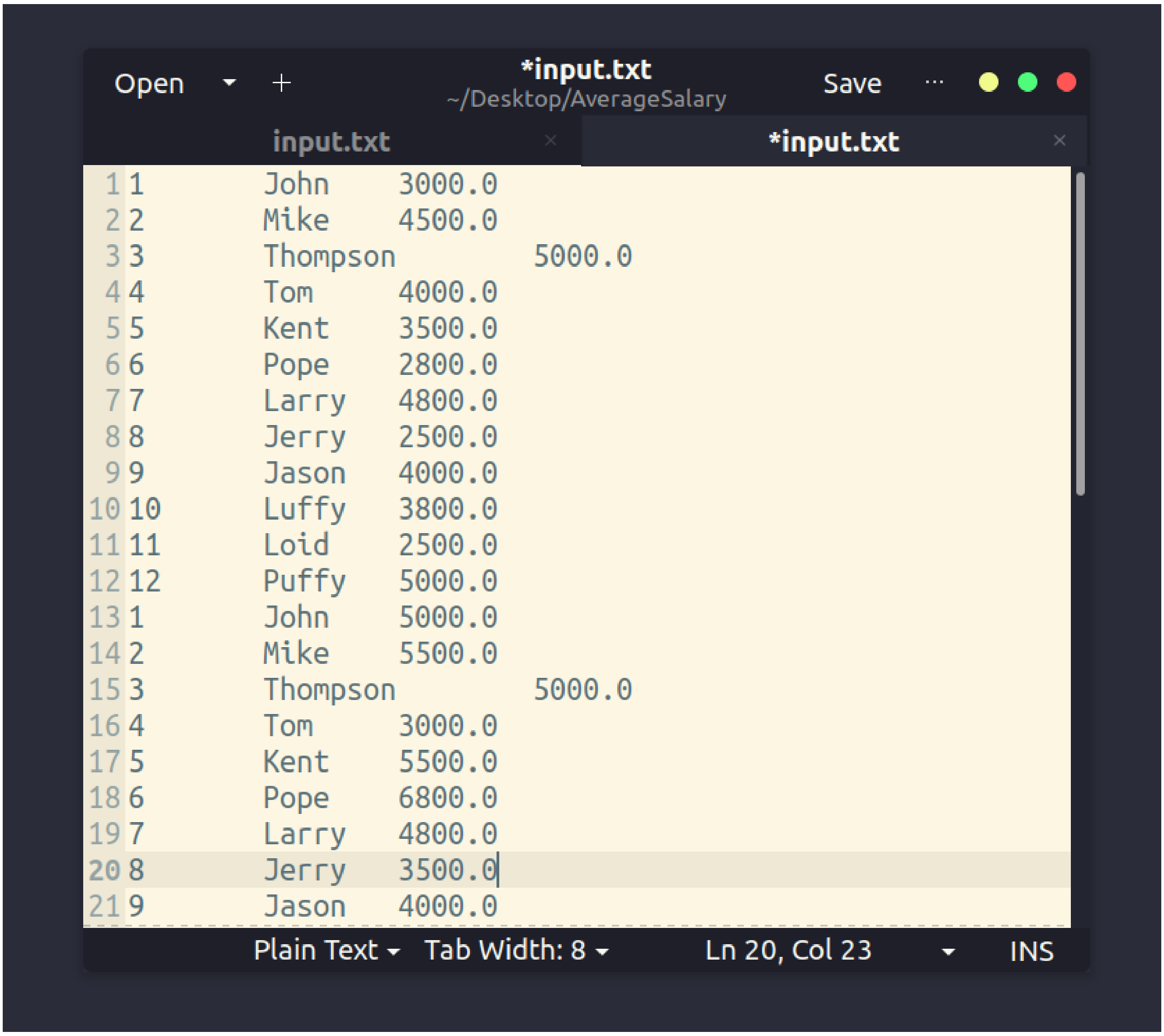
- Notice: Make sure that you export HADOOP_CLASSPATH before buiding file jar

Step 3: Create directory structure for program in Hadoop

- We need to create folder to store input data in HDFS by below command:

```
hadoop fs -mkdir /AverageSalary
hadoop fs -mkdir /AverageSalary/Input
hadoop fs -put "local input file's path" /AverageSalary/Input
```

- Example input:



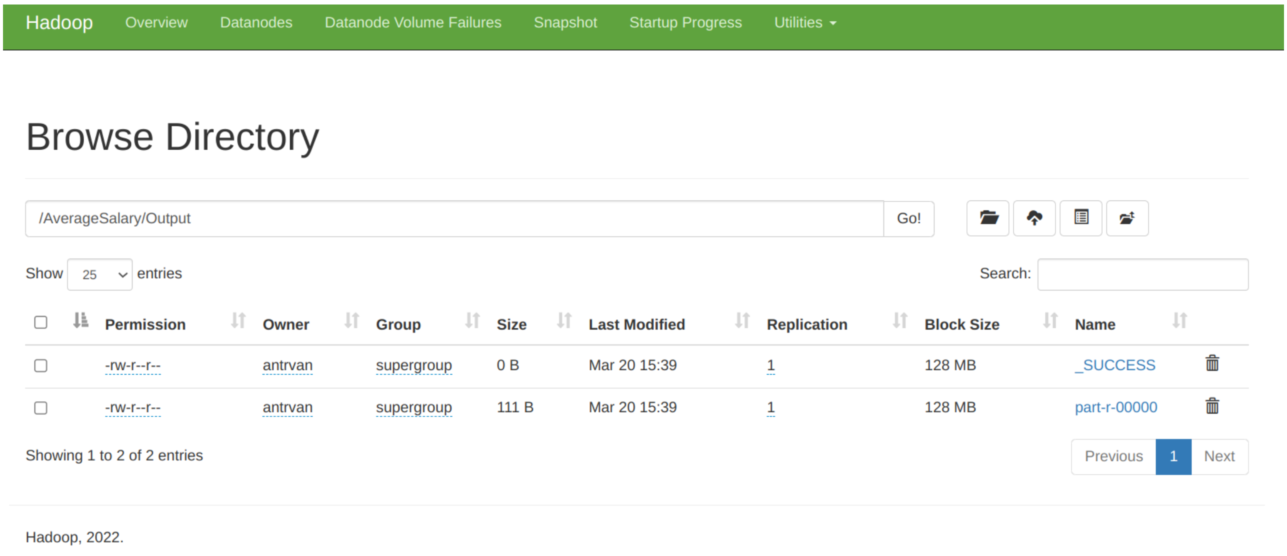
```
1       John    3000.0
2       Mike    4500.0
3       Thompson        5000.0
4       Tom     4000.0
5       Kent    3500.0
6       Pope    2800.0
7       Larry   4800.0
8       Jerry   2500.0
9       Jason   4000.0
10      Luffy   3800.0
11      Loid    2500.0
12      Puffy   5000.0
1       John    5000.0
2       Mike    5500.0
3       Thompson        5000.0
4       Tom     3000.0
5       Kent    5500.0
6       Pope    6800.0
7       Larry   4800.0
8       Jerry   3500.0
9       Jason   4000.0
```

Step 4: Create Jar File and deploy it to Hadoop

```
hadoop jar "Path to your local file .jar" AverageSalary
/AverageSalary/Input /AverageSalary/Output
```

Step 5: Final result

- After succesfully calculating, we can check our result in HDFS like below:



Browse Directory

/AverageSalary/Output

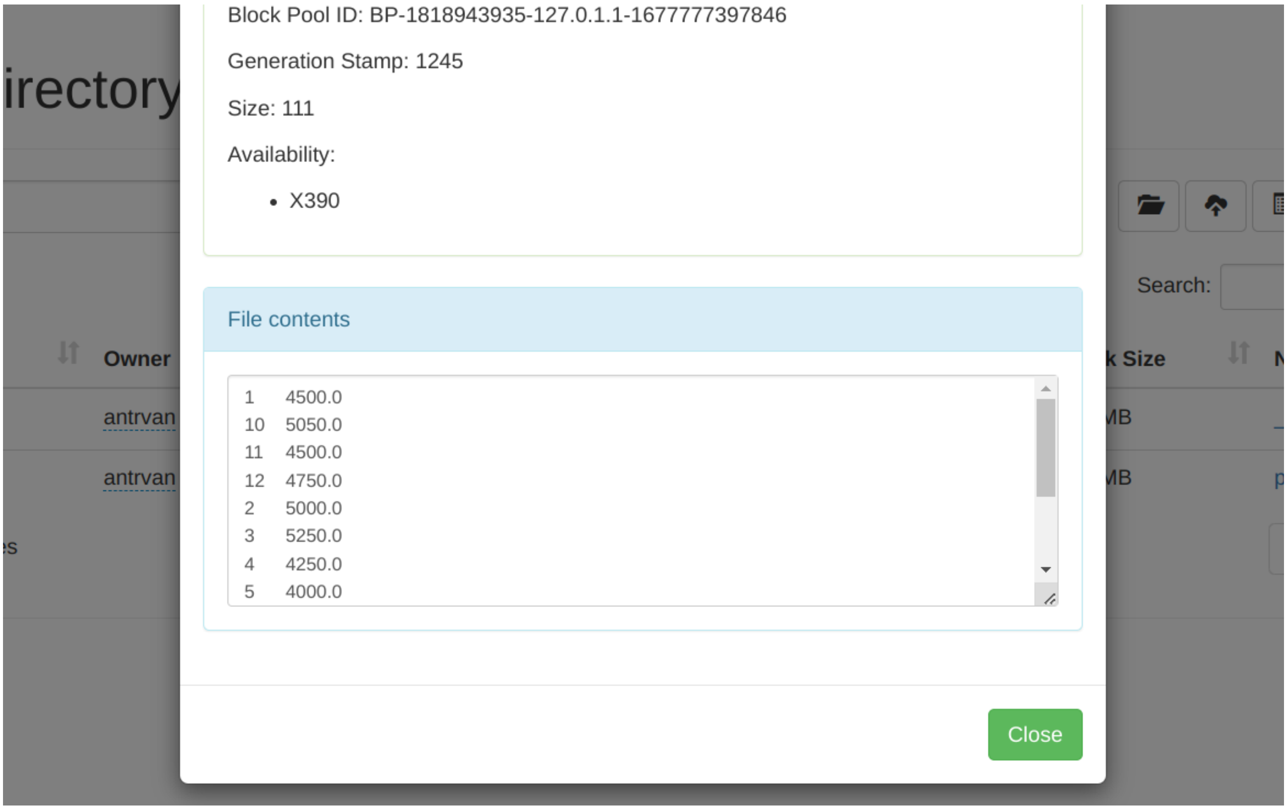
Show 25 entries Search:

□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	-rw-r--r--	antrvan	supergroup	0 B	Mar 20 15:39	1	128 MB	_SUCCESS
□	-rw-r--r--	antrvan	supergroup	111 B	Mar 20 15:39	1	128 MB	part-r-00000

Showing 1 to 2 of 2 entries

Previous 1 Next

Hadoop, 2022.

- 

Block Pool ID: BP-1818943935-127.0.1.1-1677777397846

Generation Stamp: 1245

Size: 111

Availability:

- X390

File contents

```
1	4500.0
10	5050.0
11	4500.0
12	4750.0
2	5000.0
3	5250.0
4	4250.0
5	4000.0
```

---

7. De Identify HealthCare Program

Step 3: Program's solution

- Mapper

```
    public static Integer[] encryptCol = {2, 3, 4, 5, 6, 7, 8};
    private static byte[] key1 = new String("sampleKey1234567").getBytes();
    public static class Map
            extends Mapper<Object, Text, NullWritable, Text> {

        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            StringTokenizer tokenizer = new
StringTokenizer(value.toString(), ",");
            List<Integer> list = new ArrayList<>();

            Collections.addAll(list, encryptCol);
            // list = {2, 3, 4, 5, 6, 7, 8}

            System.out.println("Mapper :: one" + value);
            String newStr = "";

            int counter = 1;

            while (tokenizer.hasMoreTokens()) {
                String token = tokenizer.nextToken();
                System.out.println("token" + token);
                System.out.println("i=" + counter);

                if (list.contains(counter)) {
                    if (newStr.length() > 0) {
                        newStr += ",";
                    }
                    newStr += encrypt(token, key1);
                }
                else {
                    if (newStr.length() > 0) {
                        newStr += ",";
                    }
                    newStr += token;
                }
                counter += 1;
            }

            context.write(NullWritable.get(), new Text(newStr.toString()));
        }
    }
```

- Encrypt function

```
    public static String encrypt(String strToEncrypt, byte[] key)
    {
        try
        {
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            SecretKeySpec secretKey = new SecretKeySpec(key, "AES");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);

            String encryptedString =
Base64.encodeBase64String(cipher.doFinal(strToEncrypt.getBytes()));

            return encryptedString.trim();
        }
        catch (Exception e)
        {
            logger.error("Error while encrypting", e);
        }
        return null;
    }
```

- Main

```
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.out.println("usage: [input] [output]");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "de identify data");
        job.setMapperClass(Map.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        job.setOutputKeyClass(NullWritable.class);
        job.setOutputValueClass(Text.class);

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

        job.setJarByClass(DeIdentifyData.class);
        job.waitForCompletion(true);
    }
```

- The idea to resolve this question is only using Map function and encrypt function to encrypt data in identified columns which need to be hidden.

Step 2: Class Creation

- After complete code in Java, we need to generate file jar from builded classes by below command:

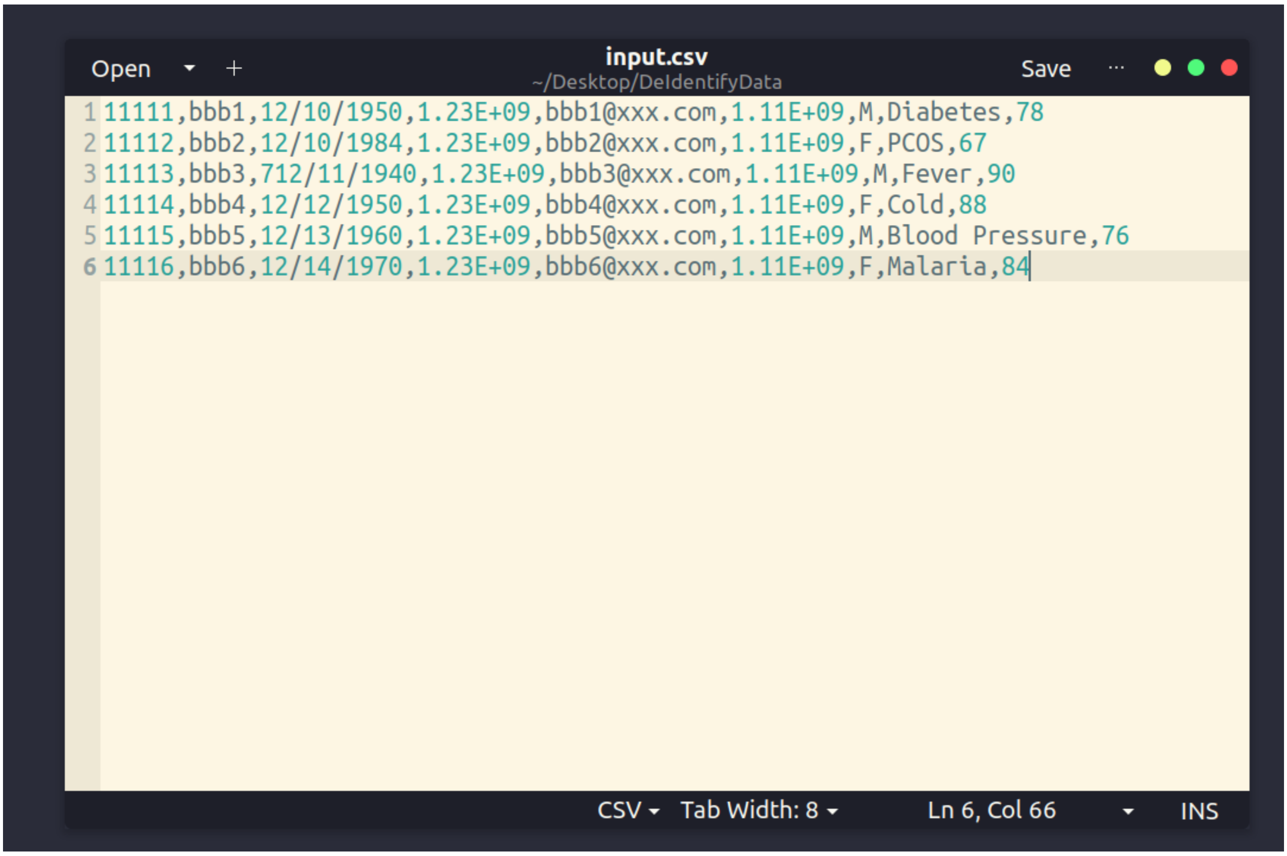
```
jar -cvf DeIdentifyData.jar -C classes/ .
```

- Notice: Make sure that you export HADOOP_CLASSPATH before buiding file jar

Step 3: Create directory structure for program in Hadoop

- We need to create folder to store input data in HDFS by below command:

```
hadoop fs -mkdir /DeIdentifyData
hadoop fs -mkdir /DeIdentifyData/Input
hadoop fs -put "local input file's path" /DeIdentifyData/Input
```

- Example input:
- 

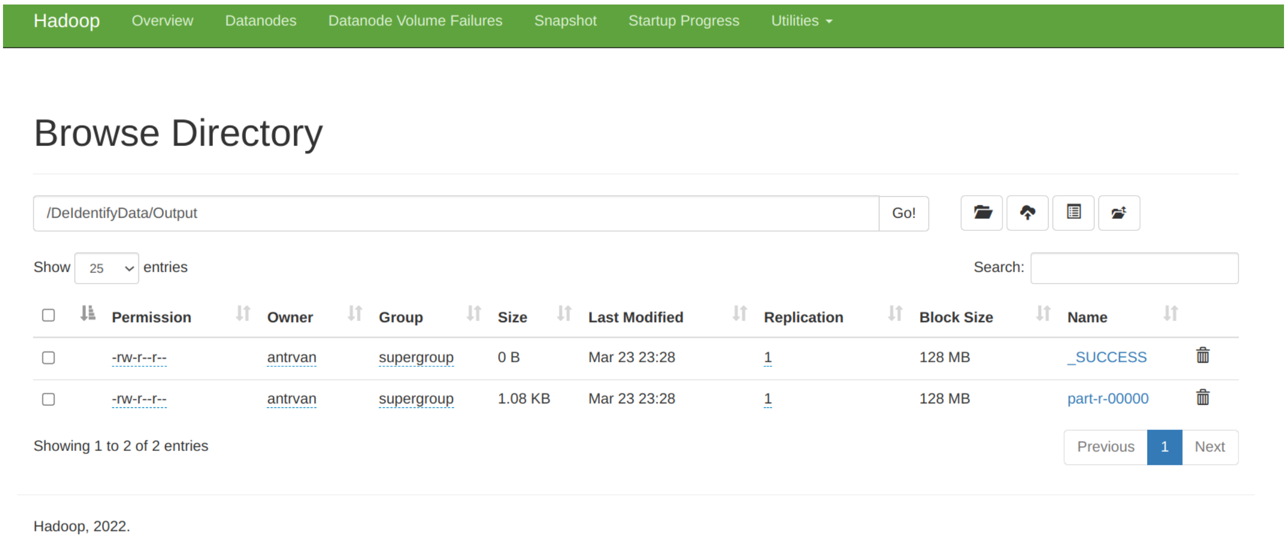
```
11111,bbb1,12/10/1950,1.23E+09,bbb1@xxx.com,1.11E+09,M,Diabetes,78
11112,bbb2,12/10/1984,1.23E+09,bbb2@xxx.com,1.11E+09,F,PCOS,67
11113,bbb3,712/11/1940,1.23E+09,bbb3@xxx.com,1.11E+09,M,Fever,90
11114,bbb4,12/12/1950,1.23E+09,bbb4@xxx.com,1.11E+09,F,Cold,88
11115,bbb5,12/13/1960,1.23E+09,bbb5@xxx.com,1.11E+09,M,Blood Pressure,76
11116,bbb6,12/14/1970,1.23E+09,bbb6@xxx.com,1.11E+09,F,Malaria,84
```

Step 4: Create Jar File and deploy it to Hadoop

```
hadoop jar "Path to your local file .jar" DeIdentifyData
/DeIdentifyData/Input /DeIdentifyData/Output
```

Step 5: Final result

- After succesfully calculating, we can check our result in HDFS like below:

- 

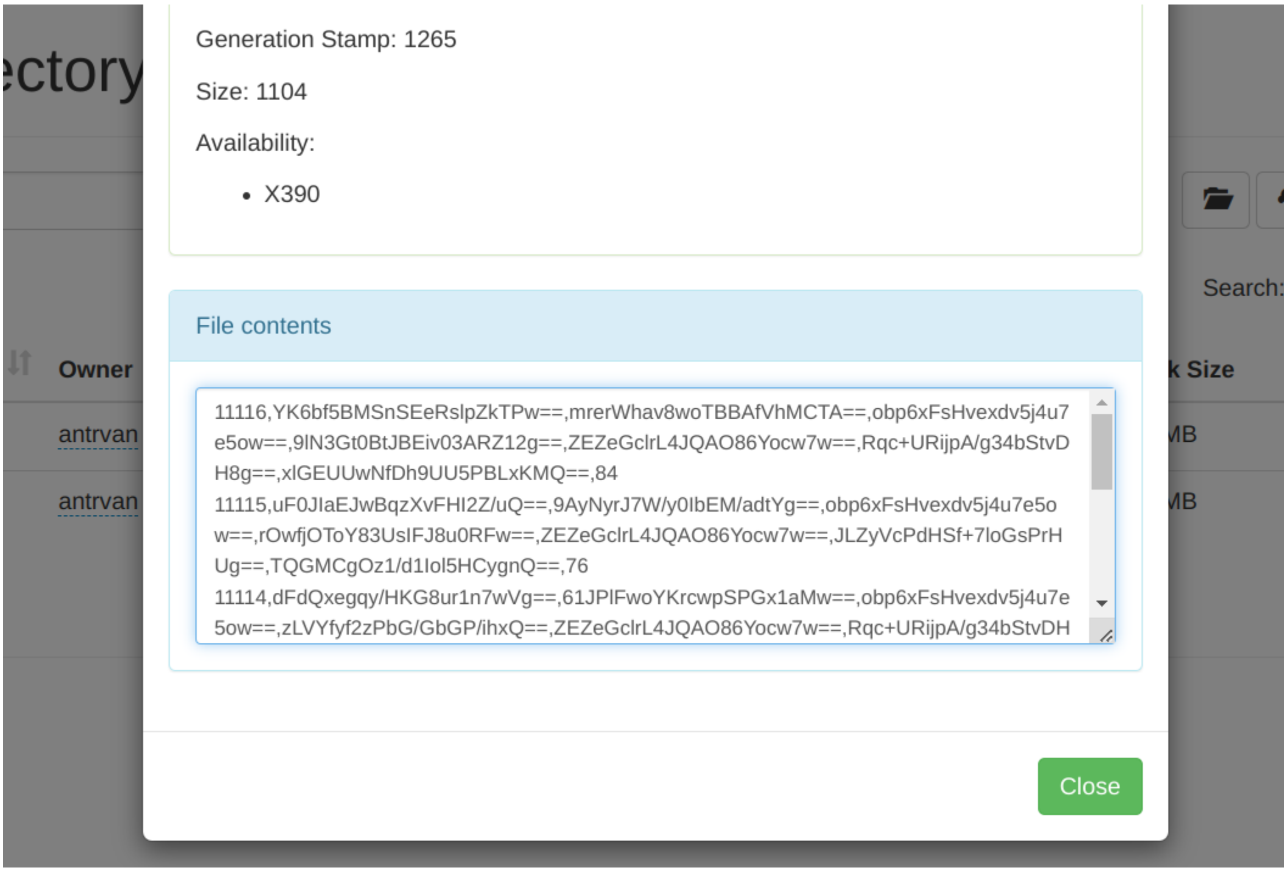
  Browse Directory

  /DeIdentifyData/Output

□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	-rw-r--r--	antrvan	supergroup	0 B	Mar 23 23:28	1	128 MB	_SUCCESS
□	-rw-r--r--	antrvan	supergroup	1.08 KB	Mar 23 23:28	1	128 MB	part-r-00000

  Showing 1 to 2 of 2 entries

  Hadoop, 2022.

- 

  Generation Stamp: 1265

  Size: 1104

  Availability:

  - X390

  File contents

  ```
  11116,YK6bf5BMSnSEeRslpZkTPw==,mrerWhav8woTBBAfVhMCTA==,obp6xFsHvexdv5j4u7e5ow==,9lN3Gt0BtJBEiv03ARZ12g==,ZEZeGclrL4JQAO86Yocw7w==,Rqc+URijpA/g34bStvDH8g==,xlGEUUwNfDh9UU5PBLxKMQ==,84
  11115,uF0JIaEJwBqzXvFHI2Z/uQ==,9AyNyrJ7W/y0IbEM/adtYg==,obp6xFsHvexdv5j4u7e5ow==,rOwfjOToY83UsIFJ8u0RFw==,ZEZeGclrL4JQAO86Yocw7w==,JLZyVcPdHSf+7loGsPrHUg==,TQGMCgOz1/d1Iol5HCygnQ==,76
  11114,dFdQxegqy/HKG8ur1n7wVg==,61JPlFwoYKrcwpSPGx1aMw==,obp6xFsHvexdv5j4u7e5ow==,zLVYfyf2zPbG/GbGP/ihxQ==,ZEZeGclrL4JQAO86Yocw7w==,Rqc+URijpA/g34bStvDH
  ```

---

8 Music Track Program

Step 1: Program's solution

- task1: Number of unique listeners
- Import:

```java
import java.util.*;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
```

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

- Mapper:

```
  public static class Map extends Mapper<Object, Text, IntWritable, IntWritable> {
        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            String line = value.toString();
            String[] data = line.split(",");
            IntWritable user = new IntWritable(Integer.parseInt(data[UserId]));
            IntWritable track = new IntWritable(Integer.parseInt(data[TrackId]));
            context.write(track, user);

        }
    }
```

- Reducer:

```
  public static class Reduce extends Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {
        public void reduce(IntWritable key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
            Set<Integer> users = new HashSet<Integer>();
            for (IntWritable val : values) {
                users.add(val.get());
            }
            IntWritable result = new IntWritable(users.size());
            context.write(key, result);
        }
    }
```

- Main:

```
  public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = new Job(conf, "Listener");
        job.setJarByClass(Listener.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(IntWritable.class);
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.waitForCompletion(true);
    }
```

- Explain:
- In the mapper method: This mapper will take as input an Object (representing the key), and a Text (representing the value) and use the String[] to separate the data in the value. Then it sends each pair<trackId,userId> to the Reducer.
- In the reducer method: This reducer will take the pair<trackId,userId> from the Mapper and add it into Set(HashSet) then return <key,Set.size()).

- task2: Number of times the track was shared with others

- Import:

```
import java.util.*;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

- Mapper:

```
  public static class Map extends Mapper<Object, Text, Text, IntWritable> {
        private Text track = new Text();
        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
            String[] data = line.split(",");
            track = new Text(data[TrackId]);
            context.write(track, new
IntWritable(Integer.valueOf(data[Shared])));
        }
    }
```

- Reducer:

```
  public static class Reduce extends Reducer<Text, IntWritable, Text,
IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context
context)
                throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum = sum + val.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }
```

- Main:

```
  public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = new Job(conf, "Shared");
        job.setJarByClass(Shared.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.waitForCompletion(true);
    }
```

- Explain:
- In the mapper method: This mapper will take as input an Object (representing the key), and a Text (representing the value) and use the String[] to separate the data in the value. Then it sends each pair<trackId,Shared> to the Reducer.

- In the reducer method: This reducer will take the pair<trackId,Shared> from the Mapper and calculate the total number of each track was shared with orthers by adding the values of 1s together..

- task3: Number of times the track was listened to on the radio

- Import:

```
import java.util.*;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

- Mapper:

```
   public static class Map extends Mapper<Object, Text, Text, IntWritable>
{
        private Text track = new Text();

        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
            String[] data = line.split(",");
            track = new Text(data[TrackId]);
            context.write(track, new
IntWritable(Integer.valueOf(data[Radio])));
        }
    }
```

- Reducer:

```
  public static class Reduce extends Reducer<Text, IntWritable, Text,
IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context
context)
                throws IOException, InterruptedException {
```

```
            int sum = 0;
            for (IntWritable val : values) {
                sum = sum + val.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }
```

- Main:

```
  public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = new Job(conf, "Radio");
        job.setJarByClass(Radio.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.waitForCompletion(true);
    }
```

- Explain:
- In the mapper method: This mapper will take as input an Object (representing the key), and a Text (representing the value) and use the String[] to separate the data in the value. Then it sends each pair<trackId,Radio> to the Reducer.
- In the reducer method: This reducer will take the pair<trackId,Radio> from the Mapper and calculate the total number of each track was listened to on radio by adding the values of 1s together..

- task4: Number of times the track was listened to in total

- Import:

```
import java.util.*;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

- Mapper:

```
   public static class Map extends Mapper<Object, Text, Text, IntWritable> {
        private Text track = new Text();

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            String line = value.toString();
            String[] data = line.split(",");
            track = new Text(data[TrackId]);
            context.write(track, new IntWritable(Integer.valueOf(data[Skip])));
        }
    }
```

- Reducer:

```
  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                if (val.get() == 0)
                    sum++;
            }
            context.write(key, new IntWritable(sum));
        }
    }
```

- Main:

```
  public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = new Job(conf, "Total");
        job.setJarByClass(Total.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.waitForCompletion(true);
    }
```

- Explain:
- In the mapper method: This mapper will take as input an Object (representing the key), and a Text (representing the value) and use the String[] to separate the data in the value. Then it sends each pair<trackId,Skip> to the Reducer.
- In the reducer method: This reducer will take the pair<trackId,Skip> from the Mapper and calculate the total number of each track was'n skipped by adding the values of 1s together..

- task5: Number of times the track was skipped on the radio

- Import:

```
import java.util.*;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

- Mapper:

```
   public static class Map extends Mapper<Object, Text, Text, IntWritable>
{
        private Text track = new Text();

        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
            String[] data = line.split(",");
            track = new Text(data[TrackId]);
            if (data[Radio].equals("0") || data[Skip].equals("0"))
                context.write(track, new IntWritable(0));
            else
                context.write(track, new IntWritable(1));

        }
    }
```

- Reducer:

```
  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum = sum + val.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }
```

- Main:

```
  public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = new Job(conf, "Skip_Radio");
        job.setJarByClass(Skip_Radio.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.waitForCompletion(true);
    }
```

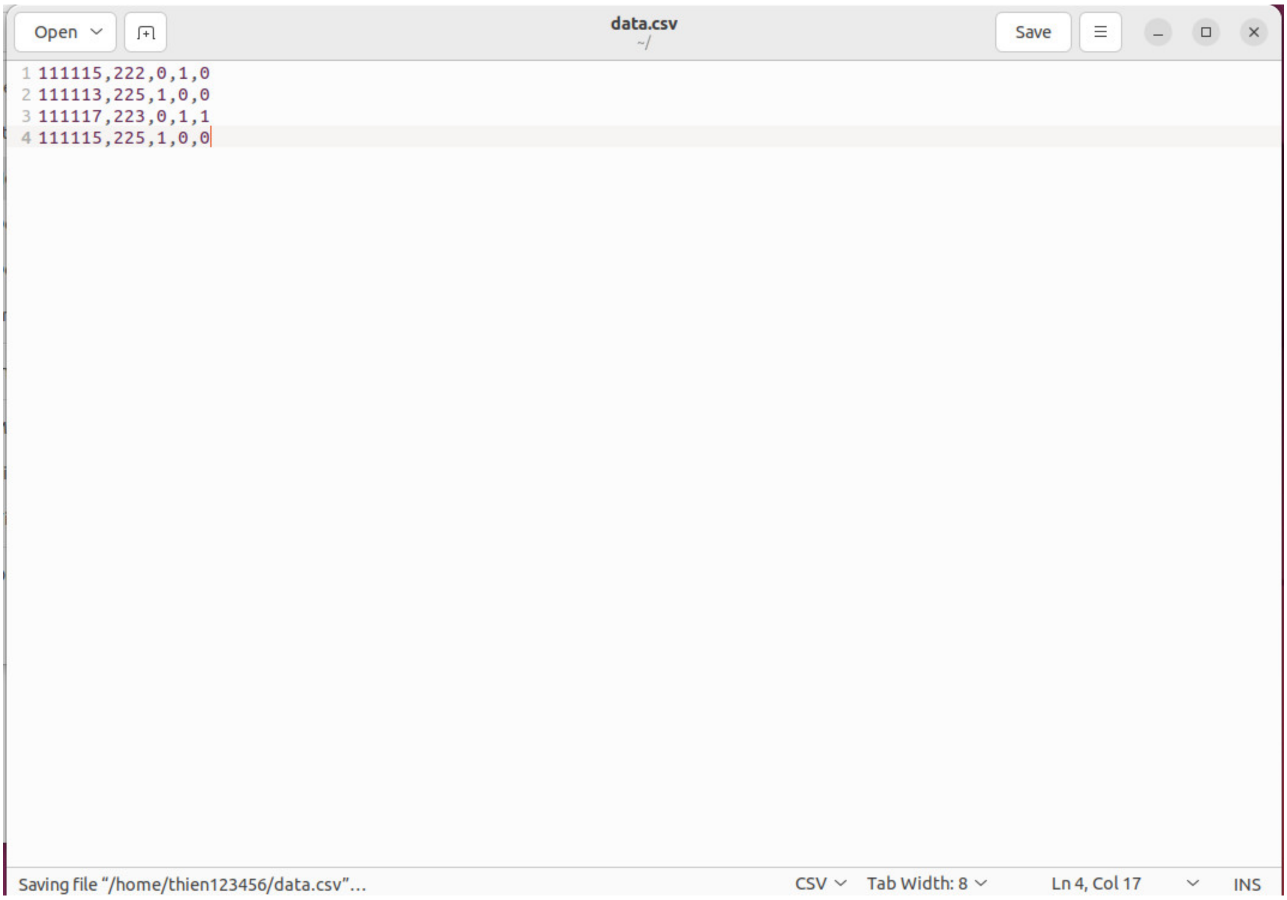
- Explain:
- In the mapper method: This mapper will take as input an Object (representing the key), and a Text (representing the value) and use the String[] to separate the data in the value. Then it sends each pair<trackId,Skip&Radio> to the Reducer.
- In the reducer method: This reducer will take the pair<trackId,Skip&Radio> from the Mapper and calculate the total number of each track was skipped on the radio by adding the values of 1s together..

Step 2: Class Creation

```
jar -cvf fileName.jar -C classes/ .
```

Step 3: Create directory structure for program in Hadoop

```
hadoop fs -mkdir /fileName
hadoop fs -mkdir /fileName/Input
hadoop fs -put 'local input file's path ' /fileName/Input
```

- Example input:
- 

```
111115,222,0,1,0
111113,225,1,0,0
111117,223,0,1,1
111115,225,1,0,0
```

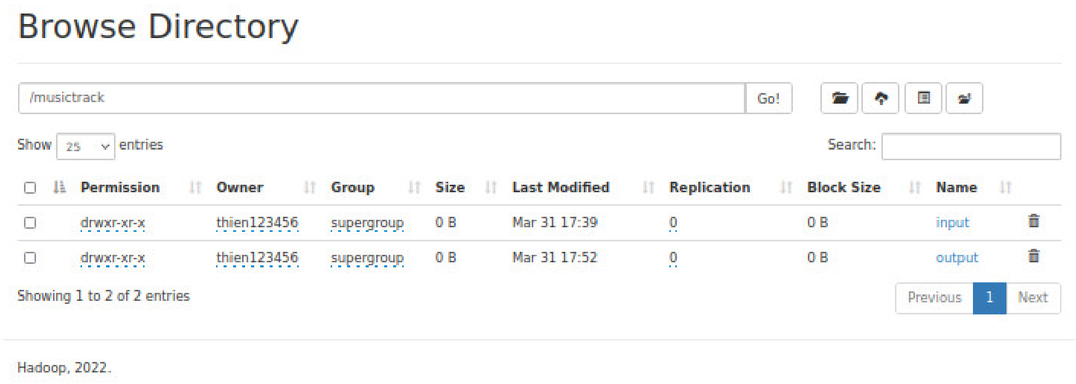
Saving file "/home/thien123456/data.csv"...

Step 4: Create Jar File and deploy it to Hadoop

```
hadoop jar "Path to your local file .jar" WordCount /WordCount/Input
/WordCount/Output
```

Step 5: Final result

- After succesfully calculating, we can check our result in HDFS like below:
- task1:



Browse Directory

/musictrack Go!

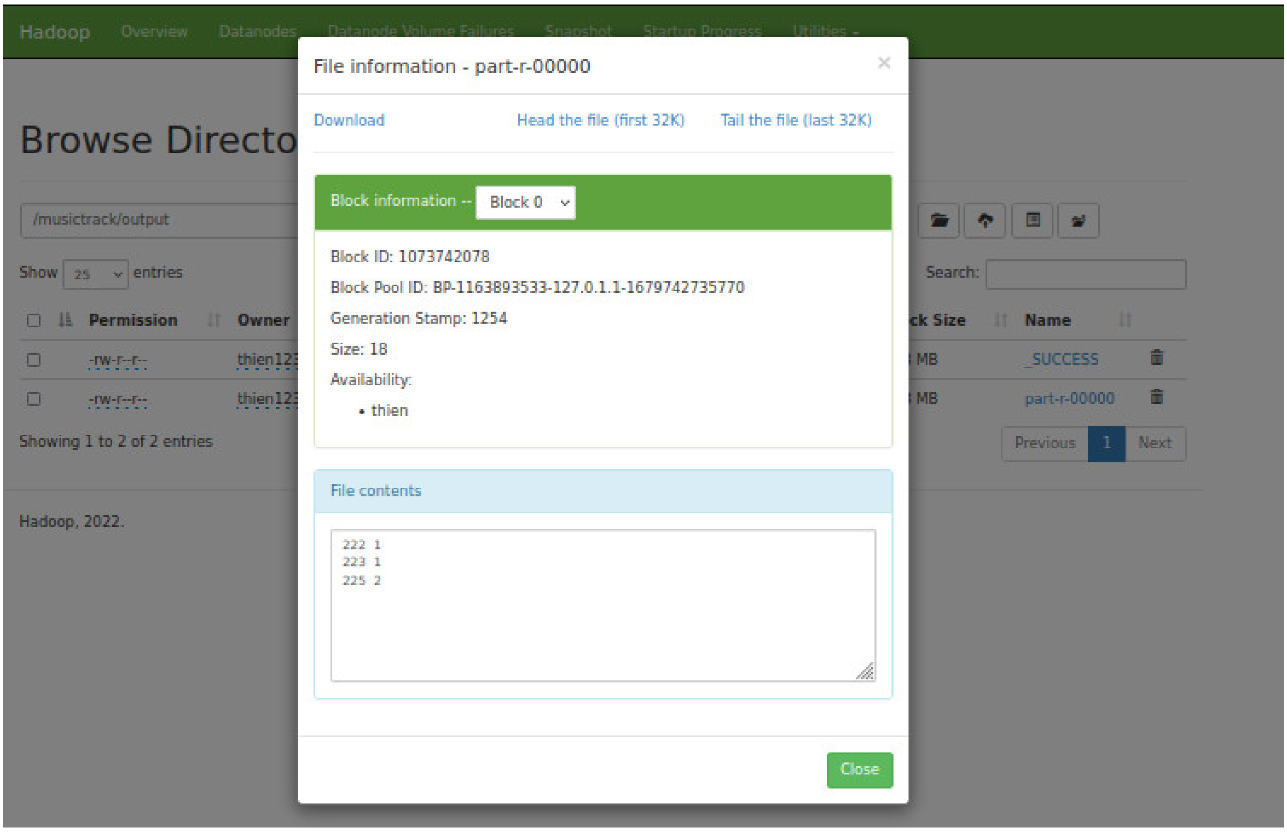
Show 25 entries Search:

☐	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
☐	drwxr-xr-x	thien123456	supergroup	0 B	Mar 31 17:39	0	0 B	input	🗑
☐	drwxr-xr-x	thien123456	supergroup	0 B	Mar 31 17:52	0	0 B	output	🗑

Showing 1 to 2 of 2 entries

Hadoop, 2022.

-

- 

Browse Directo

/musictrack/output

Show 25 entries Search:

☐	Permission	Owner	ck Size	Name	
☐	-rw-r--r--	thien123	MB	_SUCCESS	🗑
☐	-rw-r--r--	thien123	MB	part-r-00000	🗑

Showing 1 to 2 of 2 entries

Hadoop, 2022.

File information - part-r-00000

Block information -- Block 0

Block ID: 1073742078

Block Pool ID: BP-1163893533-127.0.1.1-1679742735770

Generation Stamp: 1254

Size: 18

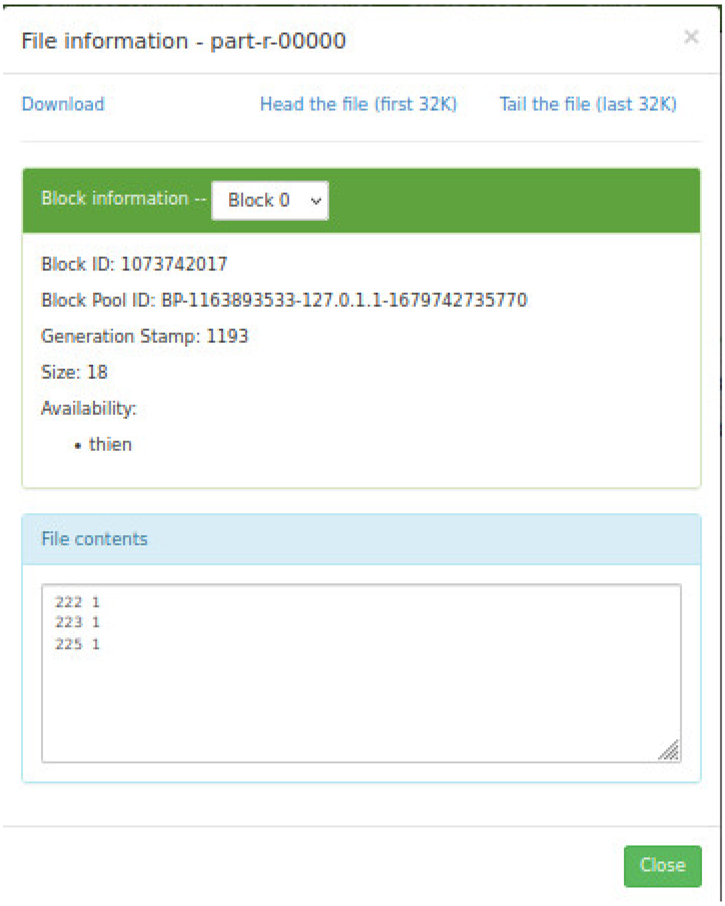
Availability:

- thien

File contents

```
222 1
223 1
225 2
```

- task2:

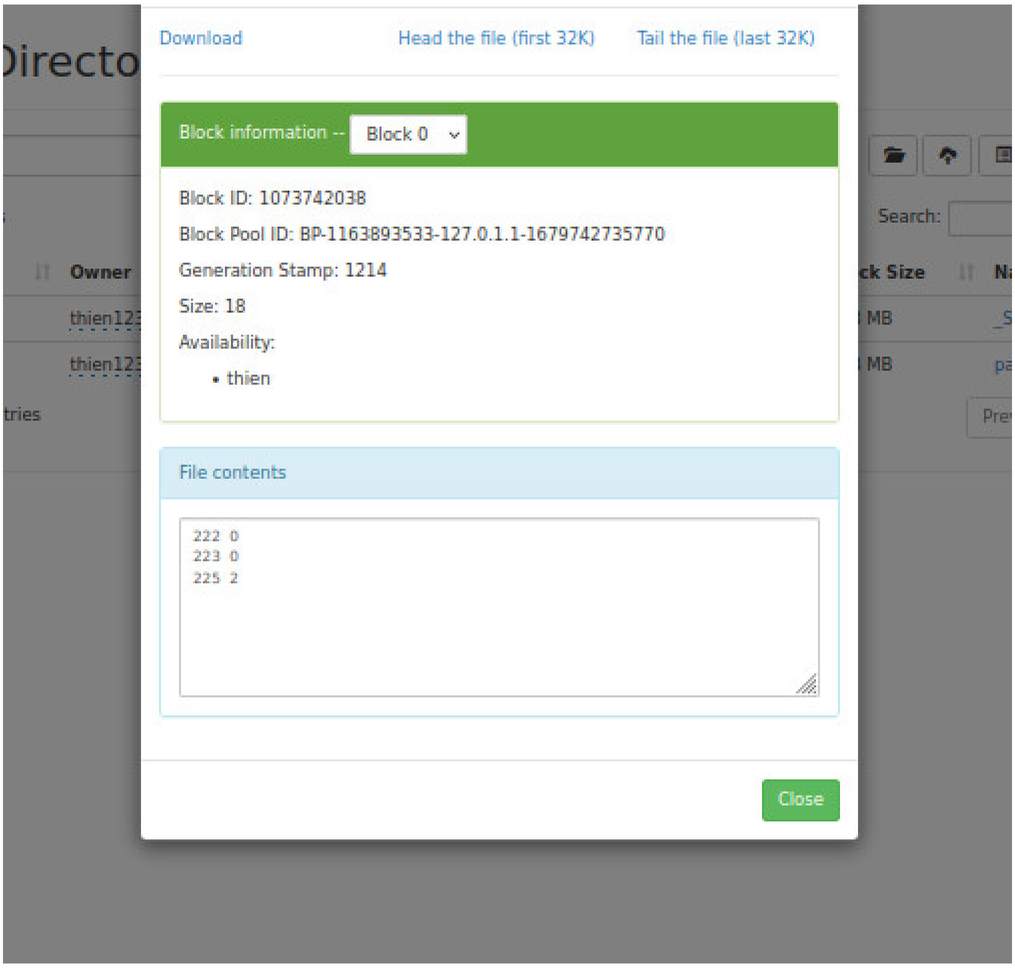
- 

  File information - part-r-00000

  Block ID: 1073742017
  Block Pool ID: BP-1163893533-127.0.1.1-1679742735770
  Generation Stamp: 1193
  Size: 18
  Availability:
  - thien

  File contents

  ```
  222 1
  223 1
  225 1
  ```

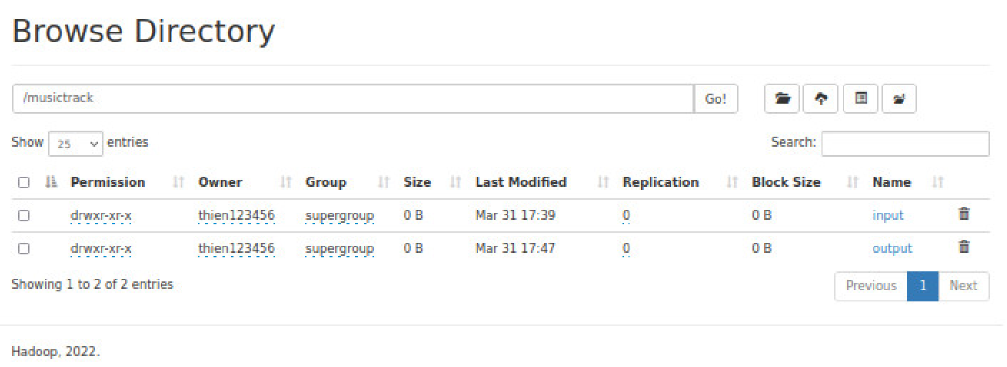
- 

  Block ID: 1073742038
  Block Pool ID: BP-1163893533-127.0.1.1-1679742735770
  Generation Stamp: 1214
  Size: 18
  Availability:
  - thien

  File contents

  ```
  222 0
  223 0
  225 2
  ```

- task3:



Browse Directory

/musictrack Go!

Show 25 entries Search:

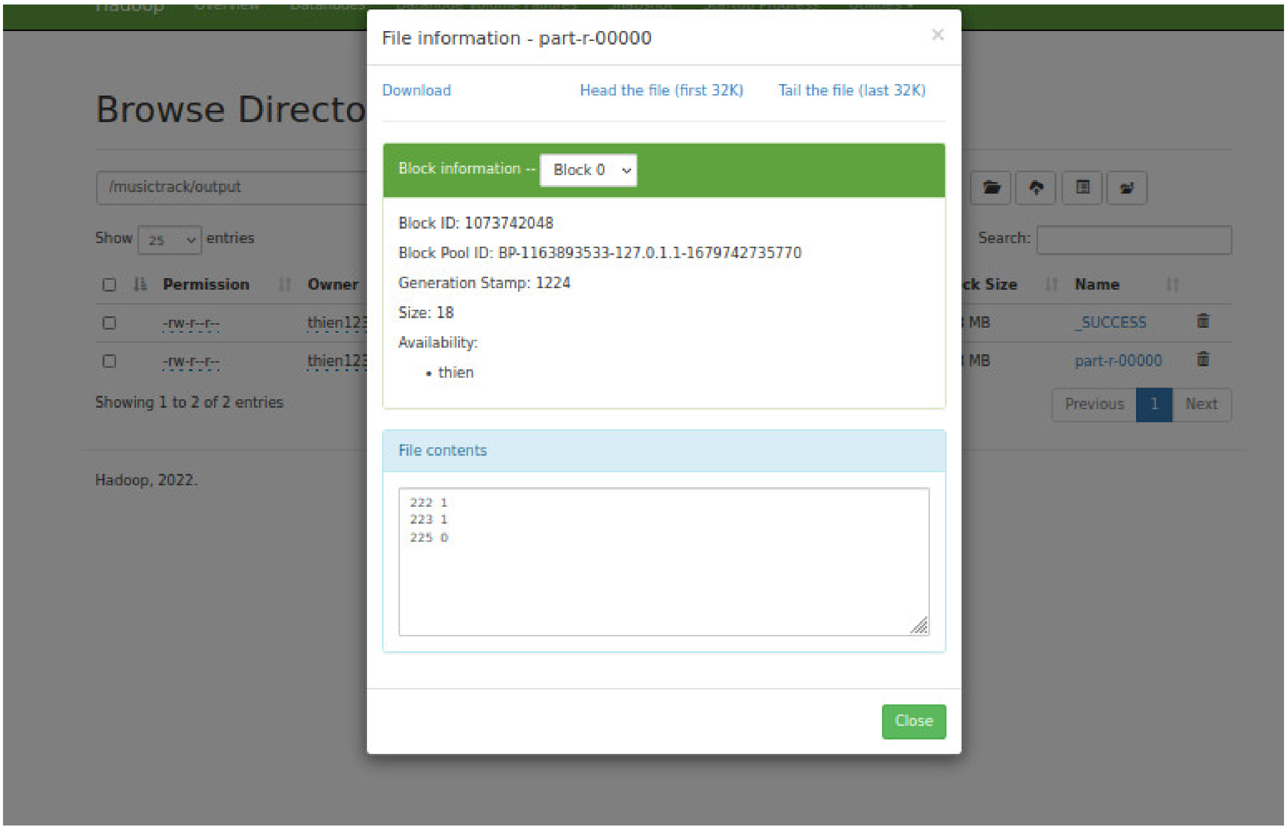
□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	drwxr-xr-x	thien123456	supergroup	0 B	Mar 31 17:39	0	0 B	input
□	drwxr-xr-x	thien123456	supergroup	0 B	Mar 31 17:47	0	0 B	output

Showing 1 to 2 of 2 entries

Previous 1 Next

Hadoop, 2022.

-



File information - part-r-00000

Download Head the file (first 32K) Tail the file (last 32K)

Block information -- Block 0

Block ID: 1073742048

Block Pool ID: BP-1163893533-127.0.1.1-1679742735770

Generation Stamp: 1224

Size: 18

Availability:

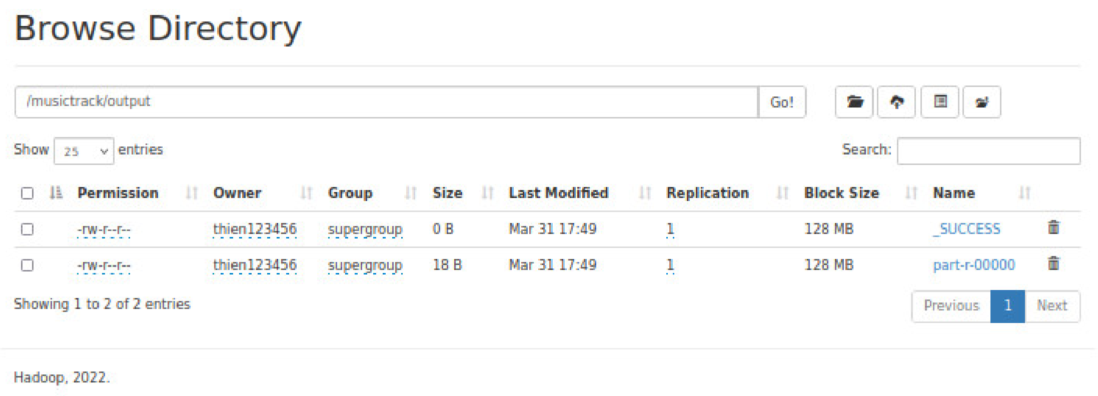
- thien

File contents

```
222 1
223 1
225 0
```

Close

-
- task4:



Browse Directory

/musictrack/output Go!

Show 25 entries Search:

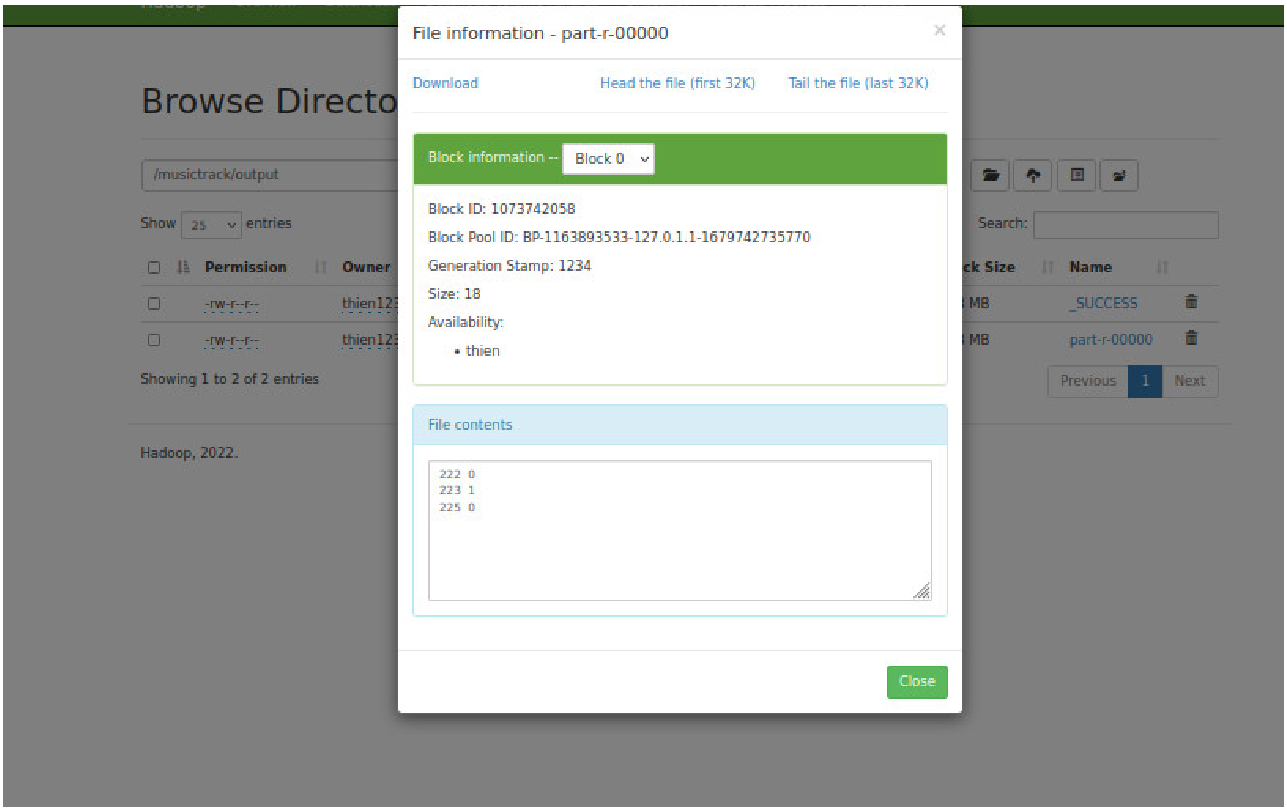
□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	-rw-r--r--	thien123456	supergroup	0 B	Mar 31 17:49	1	128 MB	_SUCCESS
□	-rw-r--r--	thien123456	supergroup	18 B	Mar 31 17:49	1	128 MB	part-r-00000

Showing 1 to 2 of 2 entries

Previous 1 Next

Hadoop, 2022.

-

- 

File information - part-r-00000

Download　Head the file (first 32K)　Tail the file (last 32K)

Block information -- Block 0

Block ID: 1073742058

Block Pool ID: BP-1163893533-127.0.1.1-1679742735770

Generation Stamp: 1234

Size: 18

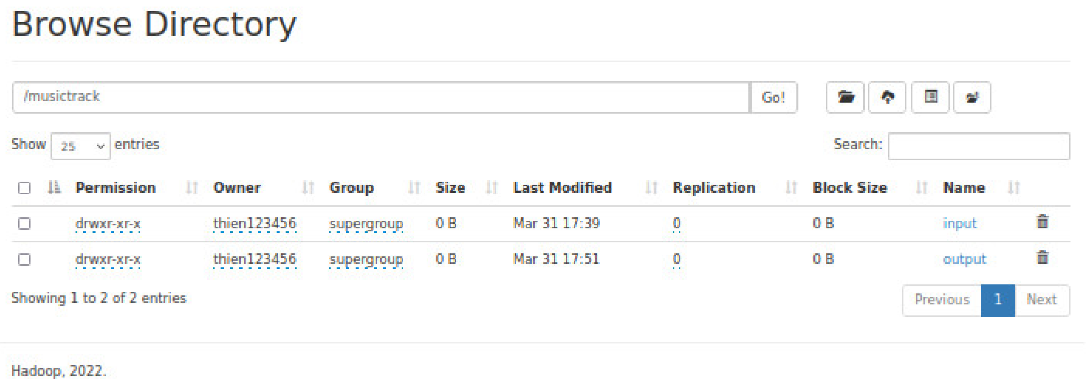
Availability:

- thien

File contents

```
222	0
223	1
225	0
```

- task5:



Browse Directory

/musictrack

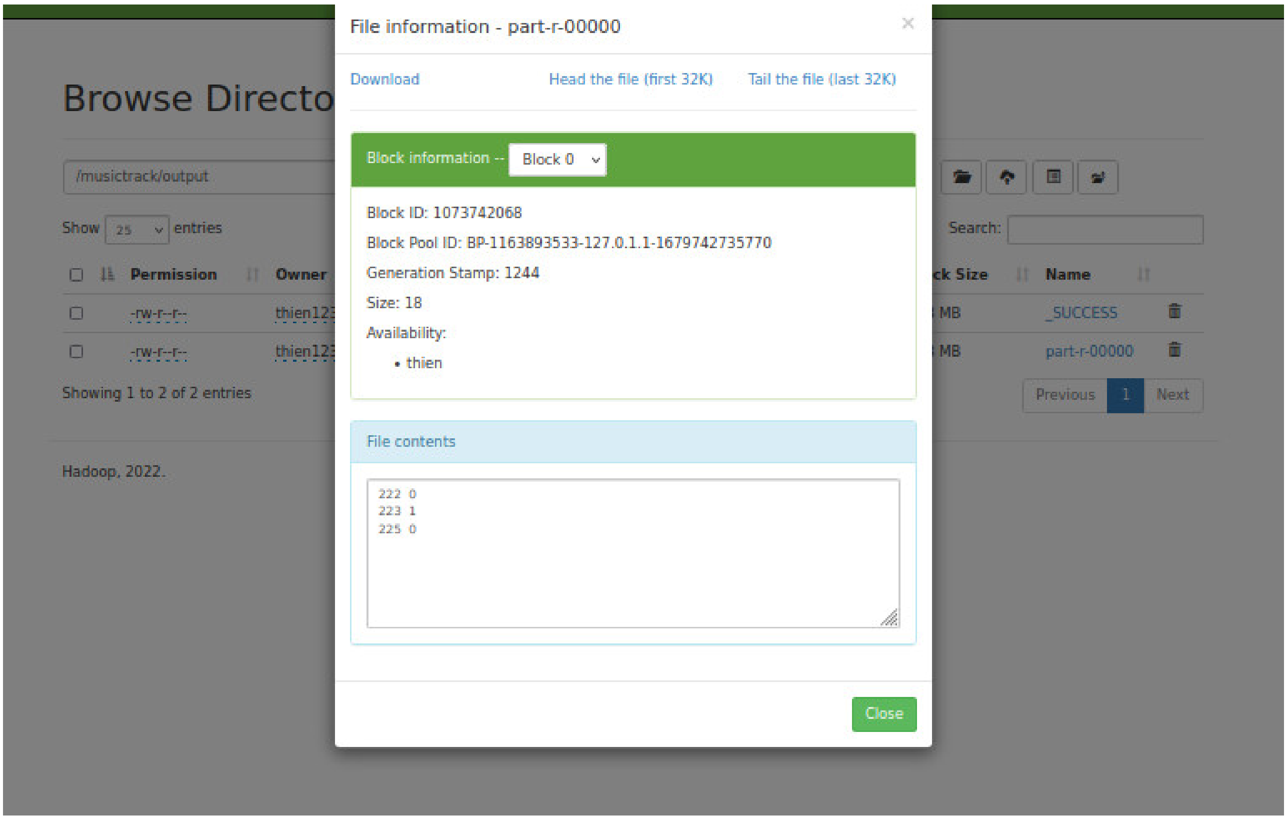
Show 25 entries　Search:

☐	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
☐	drwxr-xr-x	thien123456	supergroup	0 B	Mar 31 17:39	0	0 B	input
☐	drwxr-xr-x	thien123456	supergroup	0 B	Mar 31 17:51	0	0 B	output

Showing 1 to 2 of 2 entries

Hadoop, 2022.

-

- 

File information - part-r-00000

Block information -- Block 0

Block ID: 1073742068

Block Pool ID: BP-1163893533-127.0.1.1-1679742735770

Generation Stamp: 1244

Size: 18

Availability:

- thien

File contents

```
222	0
223	1
225	0
```

---

9. Telecom Call Data Record Program

Step 1: Program's solution

- Import:

```
import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

- Mapper:

```
public static class Map
            extends Mapper<Object, Text, Text, LongWritable>{

        Text phoneNumber = new Text();
        LongWritable minutes = new LongWritable();
        public void map(Object key, Text value, Context context
        ) throws IOException, InterruptedException {
            String line = value.toString();
            String[] tokens = line.split("\\|");
            if(tokens[4].equals("1")) {
                phoneNumber.set(tokens[0]);
                minutes.set(calculateTimeInMinutes(tokens[2], tokens[3]));
                context.write(phoneNumber, minutes);
            }
        }

        private long calculateTimeInMinutes(String start, String end) {
            SimpleDateFormat formatter = new SimpleDateFormat(("yyyy-MM-dd HH:mm:ss"));
            long minutes = -1; // if this value happen then there's an error
            try {
                // put code in try catch so that java is not angry
                Date startDate = formatter.parse(start);
                Date endDate = formatter.parse(end);
                long duration = endDate.getTime() - startDate.getTime();
                minutes = duration / (1000 * 60);
            } catch (ParseException e) {
                e.printStackTrace();
            }
            return minutes;
        }
    }
```

- Reducer:

```
public static class Reduce
            extends Reducer<Text,LongWritable,Text,LongWritable>{

        public void reduce(Text key, Iterable<LongWritable> values,
                           Context context
        ) throws IOException, InterruptedException {
            long totalMinutes = 0;
            for(LongWritable val: values) {
                totalMinutes += val.get();
            }
            if(totalMinutes > 60) {
                context.write(key, new LongWritable(totalMinutes));
            }

        }
    }
```

- Main:

```
public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "call data record");
        job.setJarByClass(CallDataRecord.class);
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(LongWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
```

- Explain:
- In the mapper method: This mapper will take as input an Object (representing the key), and a Text (representing the value) and use the `split()` method to separate the fields in a line (inside is a regular expression indicate a "|" character). It then check if the std is equal to 1. If true, it will write the FromPhoneNumber and the time in minute (calculate using the calculateTimeInMinutes() utility function) to the context.
- In the reducer metho: This reducer will take the words from the Mapper and add all the values together, it then write the result to the context if the total is greater than 60.

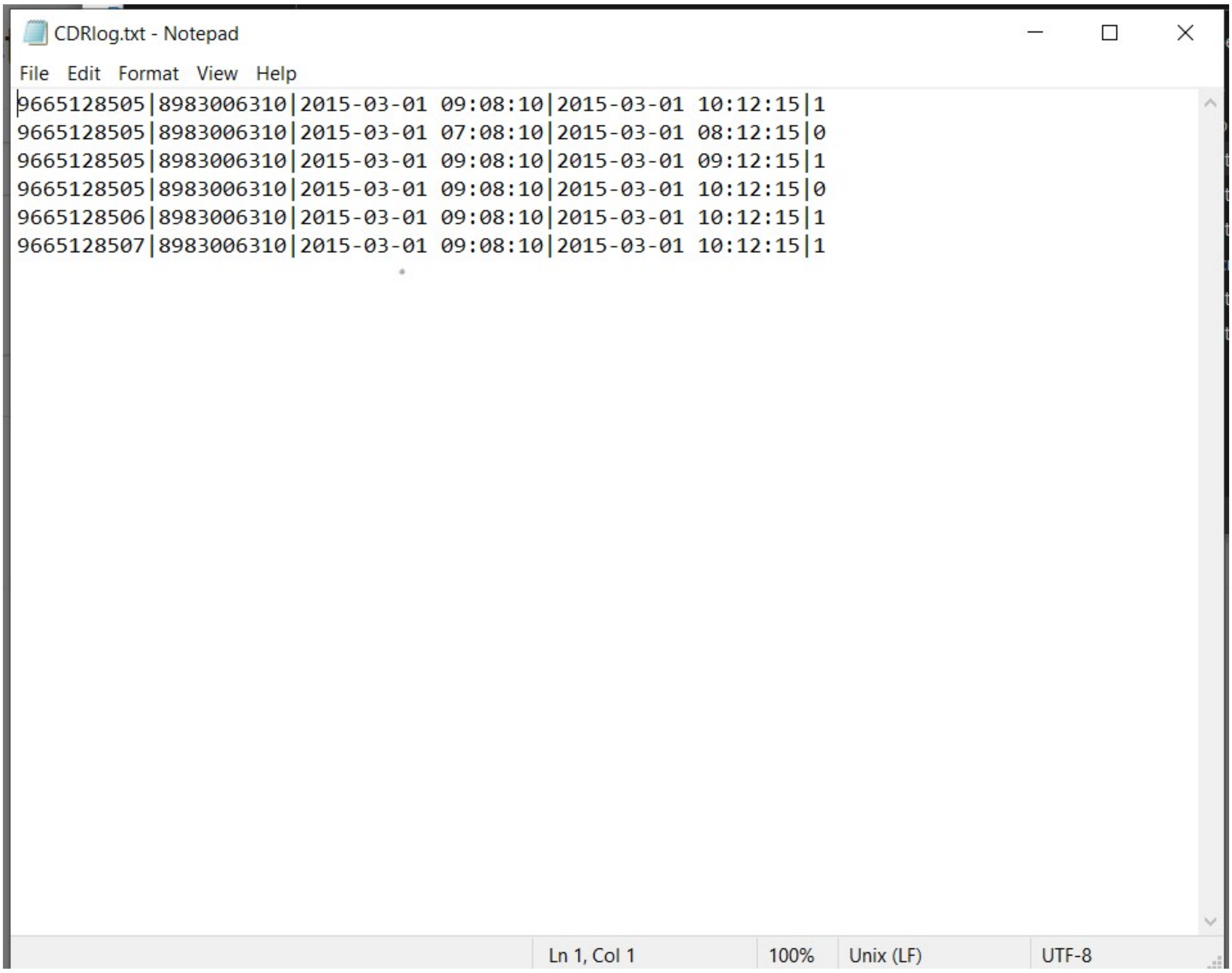
Step 2: Class Creation

```
jar -cvf CallDataRecord.jar -C classes/ .
```

Step 3: Create directory structure for program in Hadoop

```
hadoop fs -mkdir /Phone
hadoop fs -mkdir /Phone/Input
hadoop fs -put 'local input file's path ' /Phone/Input
```

- Example input:

- 

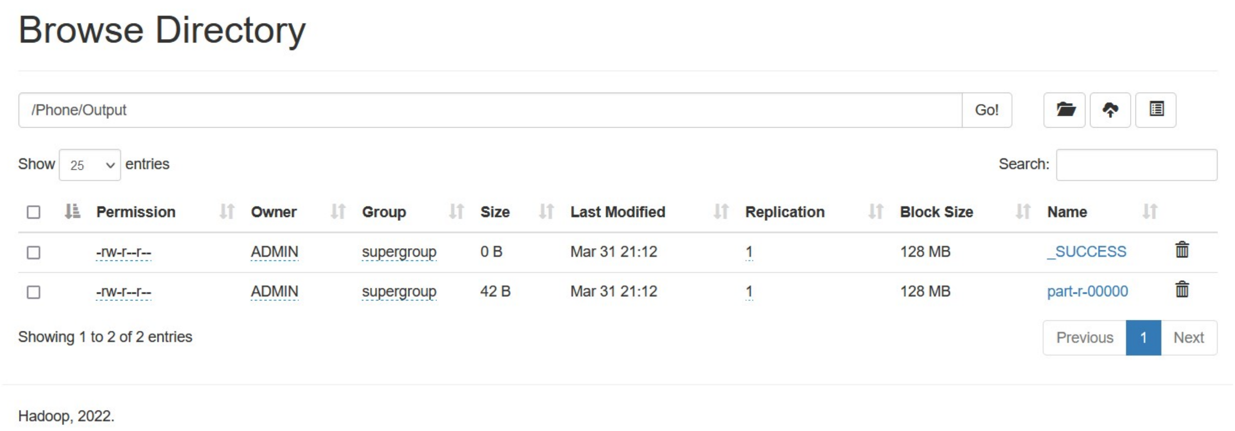
```
9665128505|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|1
9665128505|8983006310|2015-03-01 07:08:10|2015-03-01 08:12:15|0
9665128505|8983006310|2015-03-01 09:08:10|2015-03-01 09:12:15|1
9665128505|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|0
9665128506|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|1
9665128507|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|1
```

Step 4: Create Jar File and deploy it to Hadoop

```
hadoop jar "Path to your local file .jar" CallDataRecord /Phone/Input
/Phone/Output
```

Step 5: Final result

- After succesfully calculating, we can check our result in HDFS like below:

- 

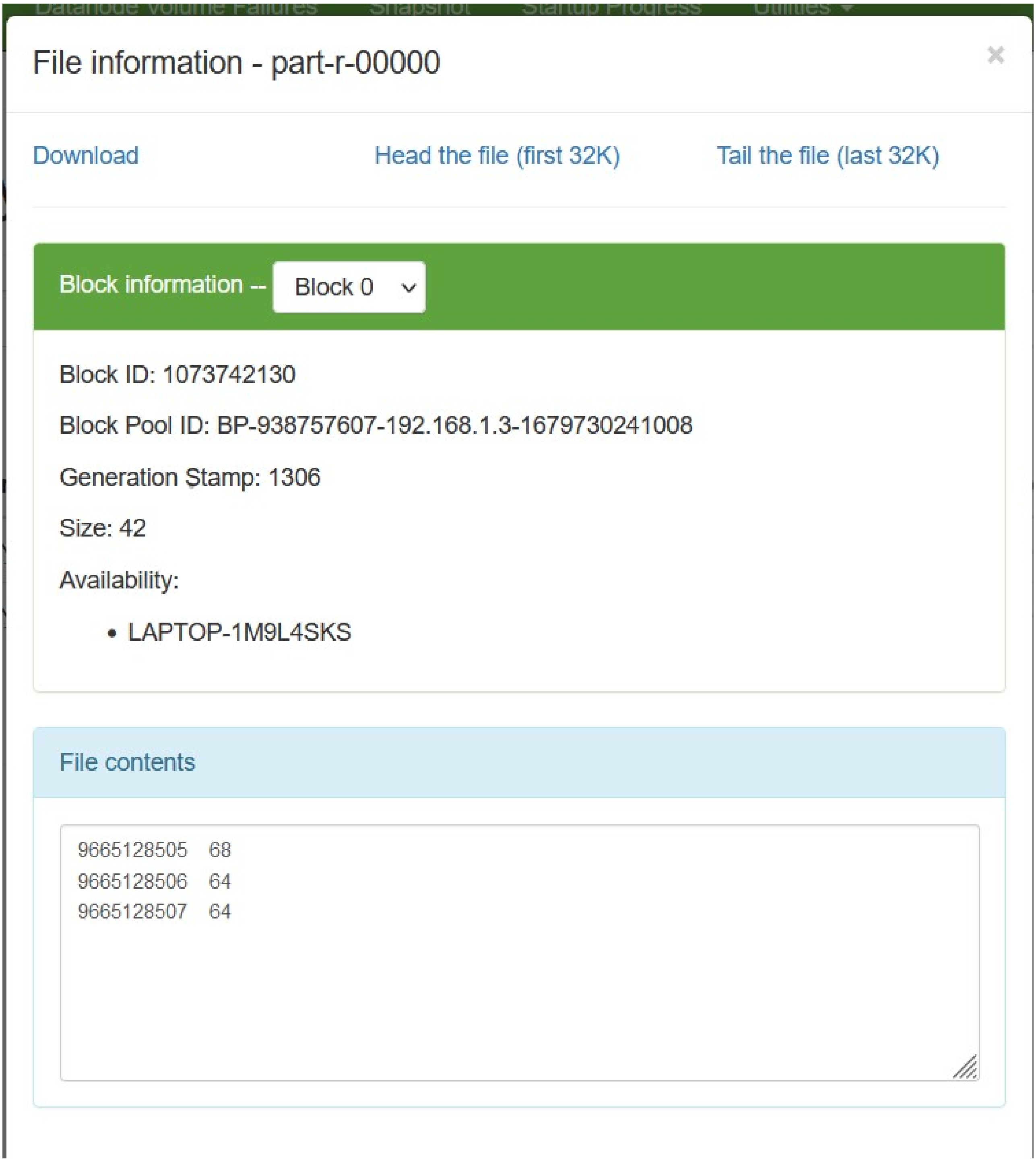
Browse Directory

/Phone/Output

☐	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
☐	-rw-r--r--	ADMIN	supergroup	0 B	Mar 31 21:12	1	128 MB	_SUCCESS
☐	-rw-r--r--	ADMIN	supergroup	42 B	Mar 31 21:12	1	128 MB	part-r-00000

Showing 1 to 2 of 2 entries

Hadoop, 2022.

- 

File information - part-r-00000

Block ID: 1073742130

Block Pool ID: BP-938757607-192.168.1.3-1679730241008

Generation Stamp: 1306

Size: 42

Availability:

- LAPTOP-1M9L4SKS

```
9665128505	68
9665128506	64
9665128507	64
```

---

10. Count Connected Components Program

Step 1: Program's solution

- Mapper

```
    public static class Map
            extends Mapper<Object, Text, Text, Text> {

        public void map(Object key, Text value, Context context
        ) throws IOException, InterruptedException {
            String[] tokens = value.toString().split(" ");
            String keyValue = tokens[0];
            Arrays.sort(tokens);

            int i = 0;
            while (i < tokens.length) {
                context.write(new Text("map"), new Text(keyValue + "," +
tokens[i]));
                i++;
            }
        }
    }
```

- Reducer

```
    public static class Reduce
            extends Reducer<Text, Text, Text, Text> {

        public void reduce(Text key, Iterable<Text> values,
                           Context context
        ) throws IOException, InterruptedException {

            TreeMap<Integer, ArrayList<Integer>> sortedMap = new TreeMap<>
();
            HashMap<Integer, Integer> result = new HashMap<>();
            for (Text value : values) {
                String[] pair = value.toString().split(",");

                int keyItem = Integer.parseInt(pair[0]);
                int valueItem = Integer.parseInt(pair[1]);

                ArrayList<Integer> tmp = sortedMap.getOrDefault(keyItem,
new ArrayList<Integer>());
                tmp.add(valueItem);
                Collections.sort(tmp);
                sortedMap.put(keyItem, tmp);
            }

            for (Integer k : sortedMap.keySet()) {
                Integer start = sortedMap.get(k).get(0);
                if (start.compareTo(k) == 0) {
                    result.put(k, k);
                }
                if (start.compareTo(k) < 0) {
                    result.put(k, result.get(start));
                    for (Integer v : sortedMap.get(k)) {
                        if (v.equals(start)) continue;
                        for (Integer j : result.keySet()) {
                            if (result.get(j).equals(v)) {
                                result.replace(j, v, start);
                            }
                        }
                    }
                }
            }

            HashSet<Integer> components = new HashSet<>();
            boolean b = components.addAll(result.values());

            if (b) {
                context.write(new Text(""), new
Text(String.valueOf(components.size())));
            }
        }
    }
```

- Main

```
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "count connected component
program");

        job.setJarByClass(CountConnectedComponentProgram.class);

        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
```

- This task is an intriguing question that calculate numbers of separated components in a graph. To resolve this problem, we put pair of source and destination point of every edges in graph to reducer. We put all pairs to TreeMap to sort them. Then in each components, we mark all connected vertices value to smallest vertex. Finally, the result equals numbers of different values in HashMap.

Step 2: Class Creation

- After complete code in Java, we need to generate file jar from builded classes by below command:

```
jar -cvf CountConnectedComponentProgram.jar -C classes/ .
```

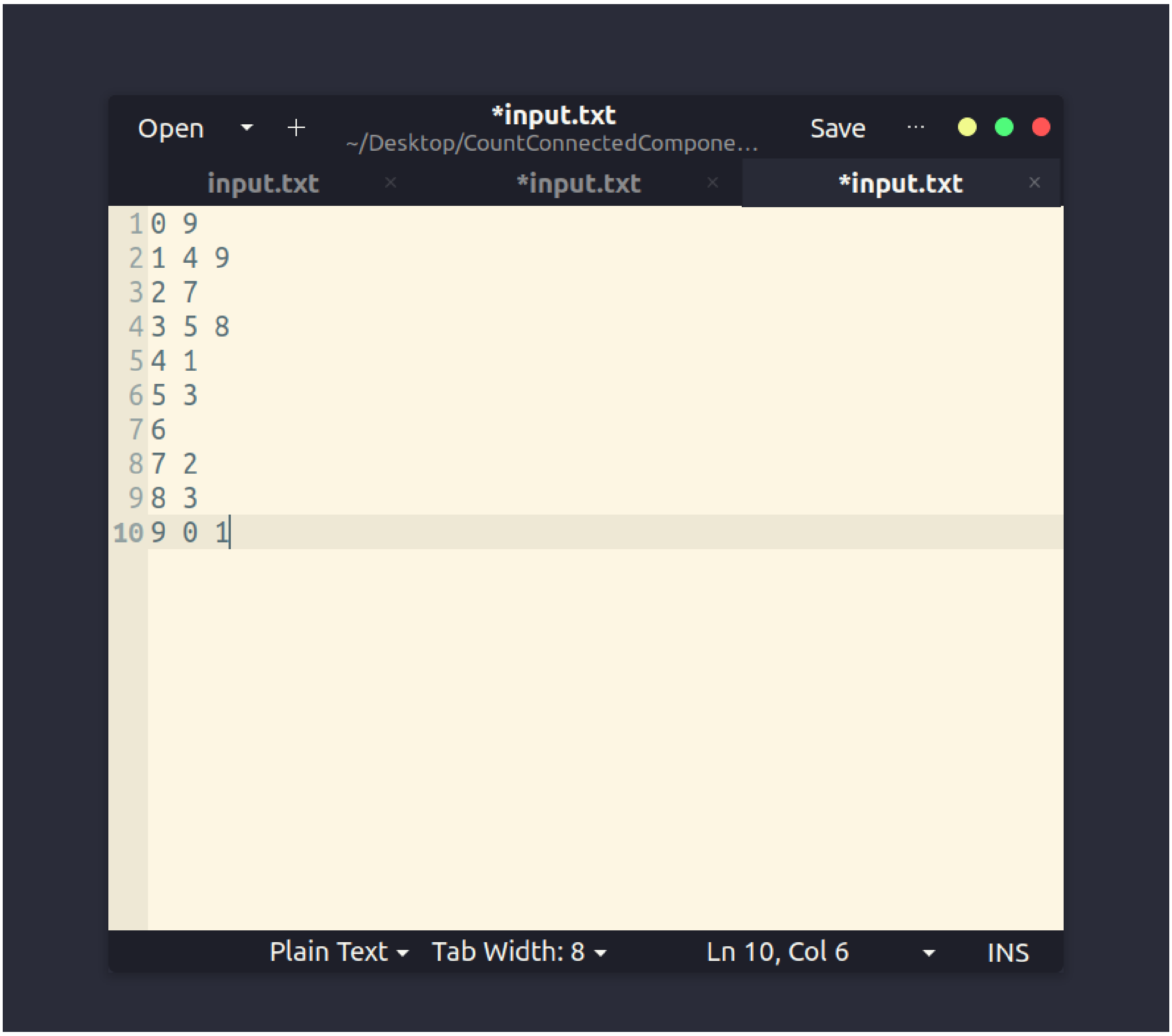
- Notice: Make sure that you export HADOOP_CLASSPATH before buiding file jar

Step 3: Create directory structure for program in Hadoop

- We need to create folder to store input data in HDFS by below command:

```
hadoop fs -mkdir /CountConnectedComponentProgram
hadoop fs -mkdir /CountConnectedComponentProgram/Input
hadoop fs -put "local input file's path"
/CountConnectedComponentProgram/Input
```

- Example input:



```
0 9
1 4 9
2 7
3 5 8
4 1
5 3
6
7 2
8 3
9 0 1
```

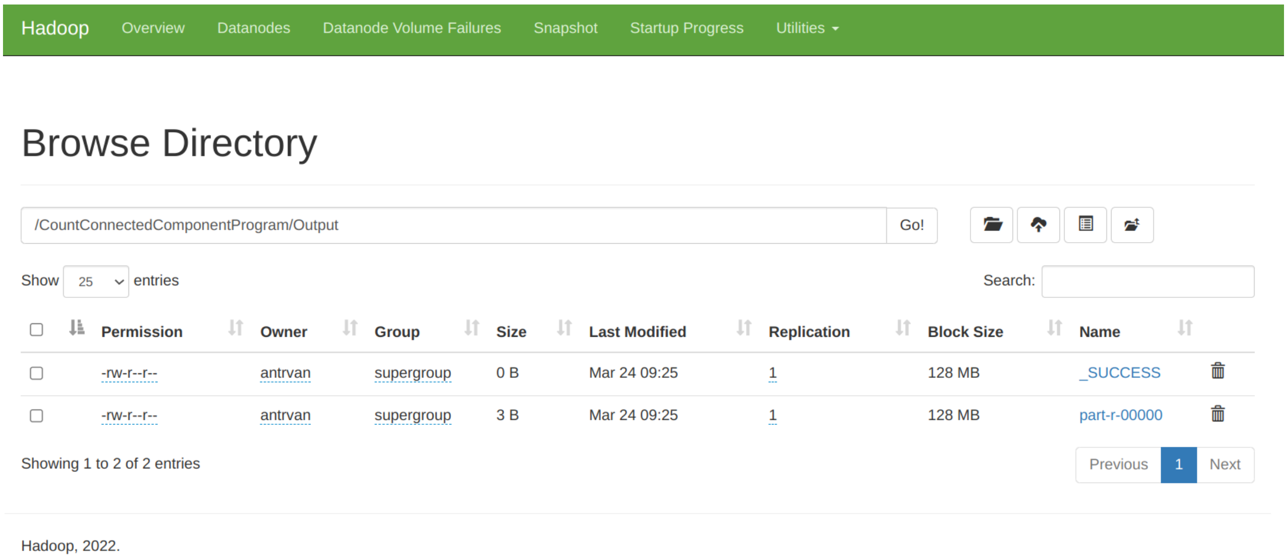
-

Step 4: Create Jar File and deploy it to Hadoop

```
hadoop jar "Path to your local file .jar" CountConnectedComponentProgram
/CountConnectedComponentProgram/Input
/CountConnectedComponentProgram/Output
```

Step 5: Final result

- After succesfully calculating, we can check our result in HDFS like below:

- 

Browse Directory

/CountConnectedComponentProgram/Output

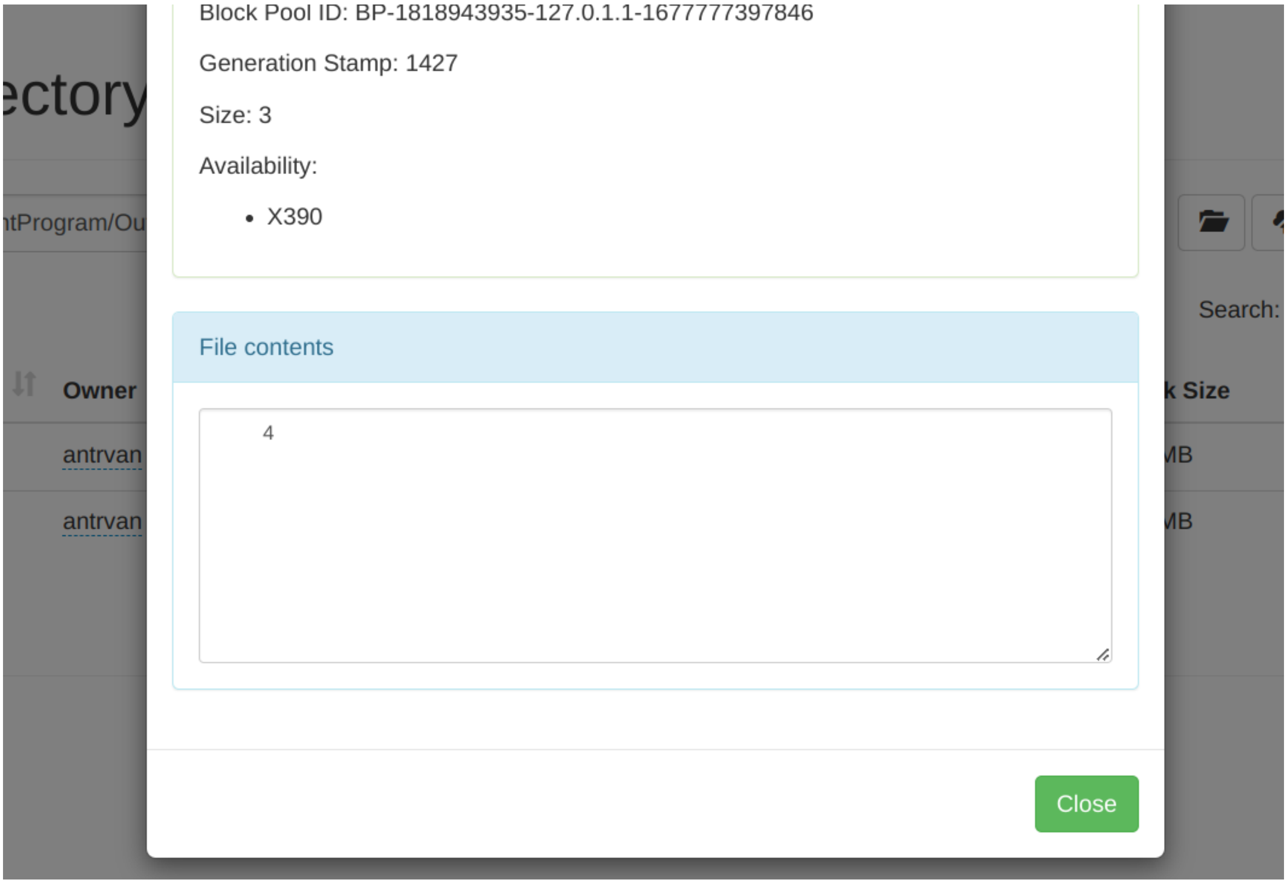
Show 25 entries

Search:

□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	-rw-r--r--	antrvan	supergroup	0 B	Mar 24 09:25	1	128 MB	_SUCCESS
□	-rw-r--r--	antrvan	supergroup	3 B	Mar 24 09:25	1	128 MB	part-r-00000

Showing 1 to 2 of 2 entries

Hadoop, 2022.

- 

Block Pool ID: BP-1818943935-127.0.1.1-1677777397846

Generation Stamp: 1427

Size: 3

Availability:

- X390

File contents

```
4
```

Self-reflection

20127435 - Tran Van An

- After completing above tasks, I know more about the useful of MapReduce in real-problems in many aspects as well as get experiences in MapReduce Programing for the midterm test.

20127395 - Phan Minh Xuan

- After completing above tasks, I understand how to store, process and manage large data sets, develop skills in the field of big data, especially know more about java language.

20127032 - Bui Gia Huy

- After completing above tasks, I know how to set up and manipulate a basic map reduce program, as well as transforming data using java utility class, as well as familiarize myself with java syntax.

20127631 - Thai Van Thien

- After completing the above tasks, I know how to set up and work with a basic map reduction program and have a preparation for the midterm exam.

Member's contribution

Task	Result
1.WordCount Program	100%
2.WordSizeWordCount Program	100%
3.Weather Data	100%
4.PatentProgram	100%
5.MaxTemp Program	100%
6.Average Salary	100%
7.De Identify Data	100%
8.Music Track Program	100%
9.Telecom Call Data Record Program	100%
10.Count Connected Components	100%

MSSV	Member	Contribution Percentage
20127435	Tran Van An	25%
20127395	Phan Minh Xuan	25%
20127032	Bui Gia Huy	25%
20127631	Thai Van Thien	25%

References