
Lab 02: MapReduce Programming

CSC14118 Introduction to Big Data 20KHMT1

SmallData

2023-02-17

Contents

1	Lab 02: MapReduce Programming	1
1.1	List of team members	1
1.2	Team's result	1
1.3	Team reflection	2
1.4	Problems	2
1.4.1	Problem 1	2
1.4.1.1	Solution idea:	2
1.4.1.2	Explain:	2
1.4.1.3	Running process:	3
1.4.1.4	Result:	5
1.4.2	Problem 2	5
1.4.2.1	Solution idea:	5
1.4.2.2	Explain:	5
1.4.2.3	Running process:	6
1.4.2.4	Result:	8
1.4.3	Problem 3	8
1.4.3.1	Solution idea:	8
1.4.3.2	Explain:	8
1.4.3.3	Running process:	9
1.4.3.4	Result:	12
1.4.4	Problem 4	12
1.4.4.1	Solution idea:	12
1.4.4.2	Explain:	12
1.4.4.3	Running process:	13
1.4.4.4	Result:	16
1.4.5	Problem 5	16
1.4.5.1	Solution idea:	16
1.4.5.2	Explain:	17
1.4.5.3	Running process:	17
1.4.5.4	Result:	19

1.4.6	Problem 6	19
1.4.6.1	Solution idea:	19
1.4.6.2	Explain:	19
1.4.6.3	Running process:	20
1.4.6.4	Result:	21
1.4.7	Problem 7	21
1.4.7.1	Solution idea:	22
1.4.7.2	Explain:	22
1.4.7.3	Running process:	22
1.4.7.4	Result:	25
1.4.8	Problem 8	26
1.4.8.1	Solution idea:	26
1.4.8.2	Explain:	26
1.4.8.3	Running process:	26
1.4.8.4	Result:	29
1.4.9	Problem 9	29
1.4.9.1	Solution idea:	29
1.4.9.2	Explain:	29
1.4.9.3	Running process:	30
1.4.9.4	Result:	33
1.4.10	Problem 10	33
1.4.10.1	Solution idea:	33
1.4.10.2	Explain:	33
1.4.10.3	Running process:	34
1.4.10.4	Result:	36
1.5	References	37

1 Lab 02: MapReduce Programming

1.1 List of team members

ID	Full Name
20120366	Pham Phu Hoang Son
20120391	Ha Xuan Truong
20120393	Huynh Minh Tu
20120468	Nguyen Van Hai

1.2 Team's result

Problem	Complete
1	100%
2	100%
3	100%
4	100%
5	100%
6	100%
7	100%
8	100%
9	100%
10	100%

1.3 Team reflection

Does your journey to the deadline have any bugs? How have you overcome it?

During our MapReduce Hadoop Java project, we faced a few logical bugs. One example was when we were implementing a custom reducer function that was not producing the expected output. After investigating, we discovered that the issue was with our implementation of the function. Specifically, we had made an error in the logic of our code that resulted in incorrect output.

To overcome this bug, we first used debugging tools to identify the source of the issue. Once we had identified the function as the problem area, we took a closer look at our code and worked to identify the error in our logic. We also discussed the issue as a team and reviewed each other's code to find potential solutions.

Ultimately, we were able to solve the issue by correcting the logical error in our reducer function. We also added additional unit tests to ensure that the function was producing the correct output. Through this process, we learned the importance of careful code review and testing, as well as the importance of understanding the logic of our code in-depth.

What have you learned after this process?

We gained a deeper understanding of the MapReduce Hadoop Java platform by working through the bugs and exploring different solutions. This experience helped us improve our knowledge of the platform and be better prepared to tackle similar challenges in the future.

1.4 Problems

We coded some of the problems ourselves, while for other problems, we referred to the solutions.

1.4.1 Problem 1

1.4.1.1 Solution idea:

Read each line of data and split it into single words, then count the number of occurrences of each word

1.4.1.2 Explain:

1. The Mapper class: This class extends the Mapper abstract class and overrides the map() method. The map() method reads each line of the input text file, splits it into individual words,

and emits a key-value pair of (word, 1). The key is the individual word, and the value is a constant integer 1. This class also defines the data types of the input key-value pairs and the output key-value pairs.

2. The Reducer class: This class extends the Reducer abstract class and overrides the reduce() method. The reduce() method receives a key-value pair of (word, list of values), where the key is a word and the value is a list of integers (each integer represents a count of the word). The reduce() method sums up the values in the list and emits a key-value pair of (word, sum of counts).

1.4.1.3 Running process:

Step 1: Create a input folder, output folder and put input file

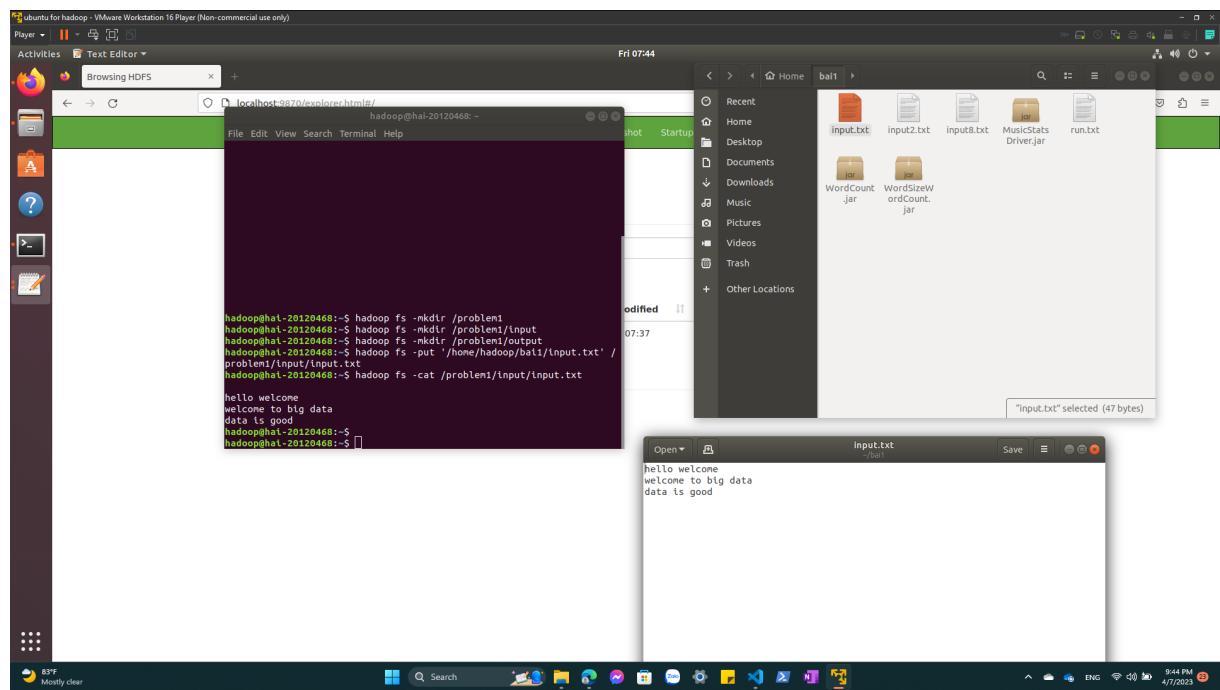


Figure 1.1: input of assignment 1

Step 2: Export file jar

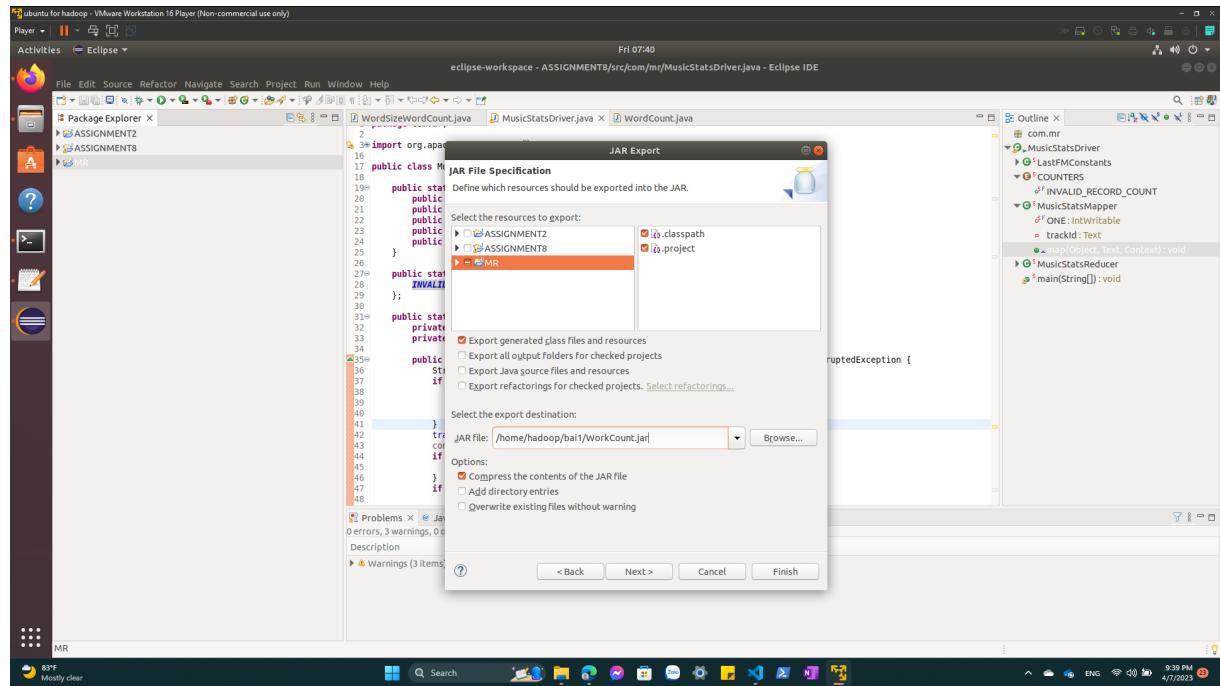


Figure 1.2: input of assignment 1

Step 3: Running jar

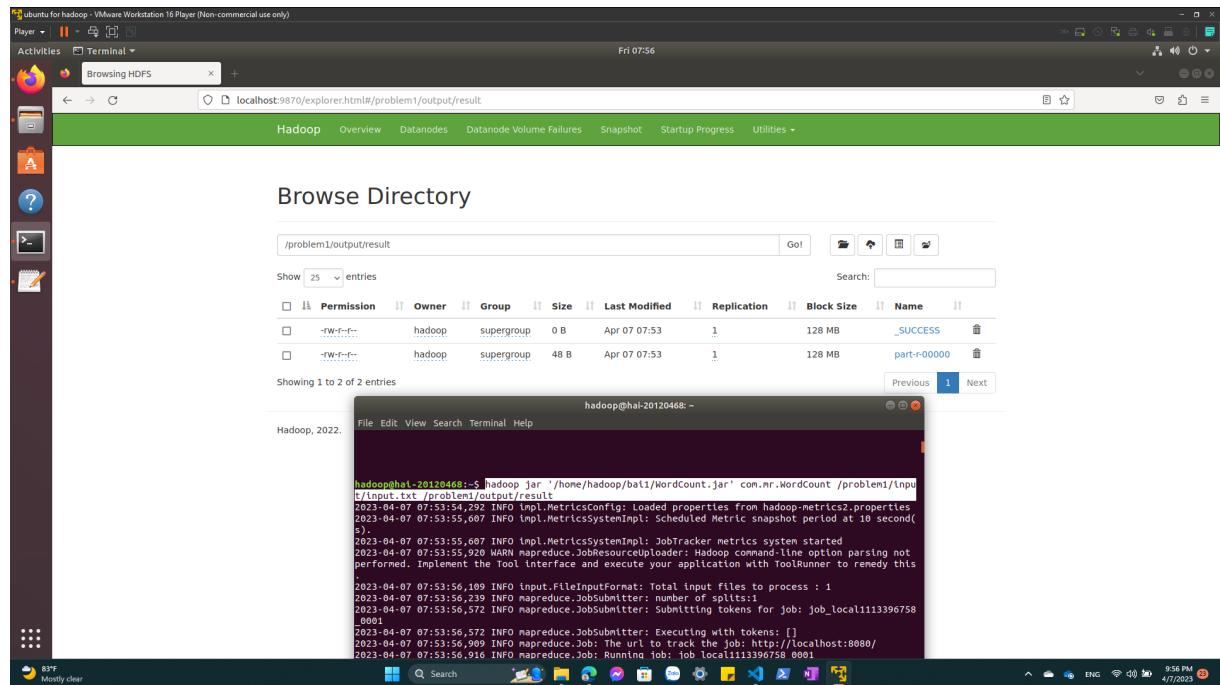
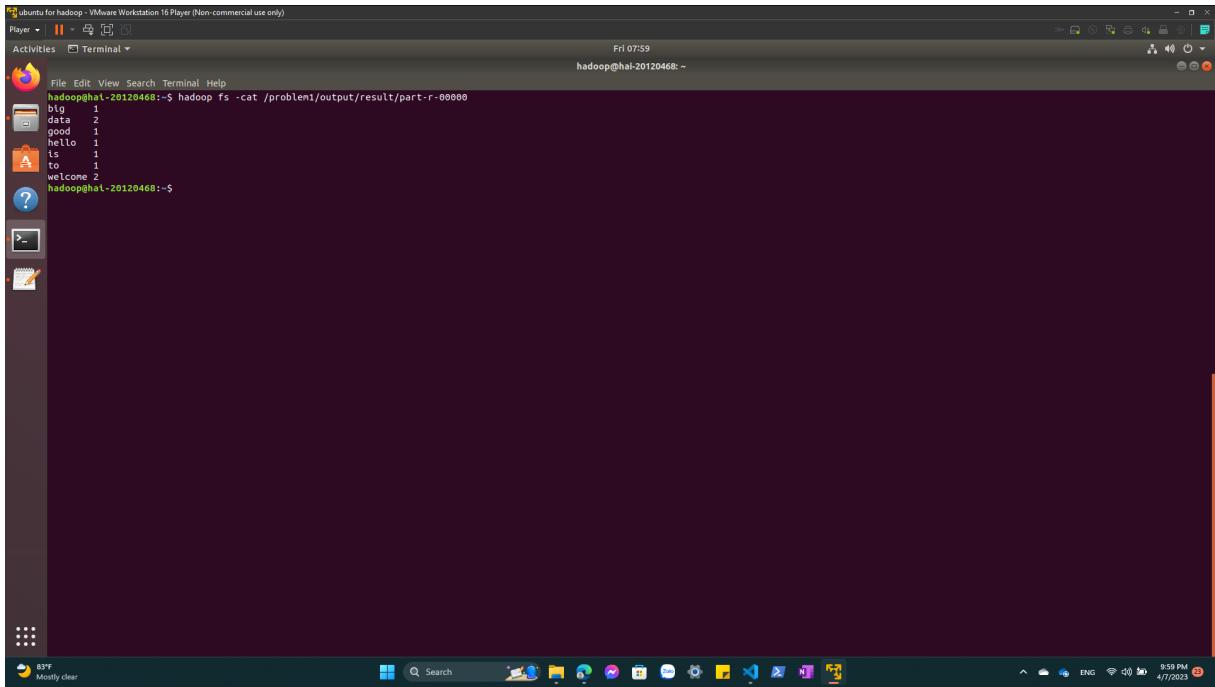


Figure 1.3: input of assignment 1

1.4.1.4 Result:



```
ubuntu for hadoop - VMware Workstation 16 Player (Non-commercial use only)
Player | || Activities Terminal Fri 07:59
Activities Terminal hadoop@hal-20120468: ~
File Edit View Search Terminal Help
hadoop@hal-20120468:~$ hadoop fs -cat /problem1/output/result/part-r-00000
big 2
data 2
good 1
hello 1
i 1
to 1
welcome 2
hadoop@hal-20120468:~$
```

Figure 1.4: input of assignment 1

1.4.2 Problem 2

1.4.2.1 Solution idea:

The solution is to count the number of words of each length in a given text document using the MapReduce programming model.

1.4.2.2 Explain:

1. The Mapper class, the map() method takes in a key-value pair consisting of a byte offset and a line of text, and emits intermediate key-value pairs consisting of the length of each word in the line and the value 1, and the same length with a value of 0. The IntWritable class is used to represent the integer keys and values.
2. The Reducer class, the reduce() method takes in intermediate key-value pairs and sums up the values associated with each key, giving a final count of the number of words of each length. The results are written to output using the context.write method.

3. The WordSizeWordCount class, the main() method sets up and runs the MapReduce job, configuring input and output paths, setting up the mapper and reducer classes, and specifying input and output formats. Finally, the job is executed using job.waitForCompletion(true).

1.4.2.3 Running process:

Step 1: Create a input folder, output folder and put input file

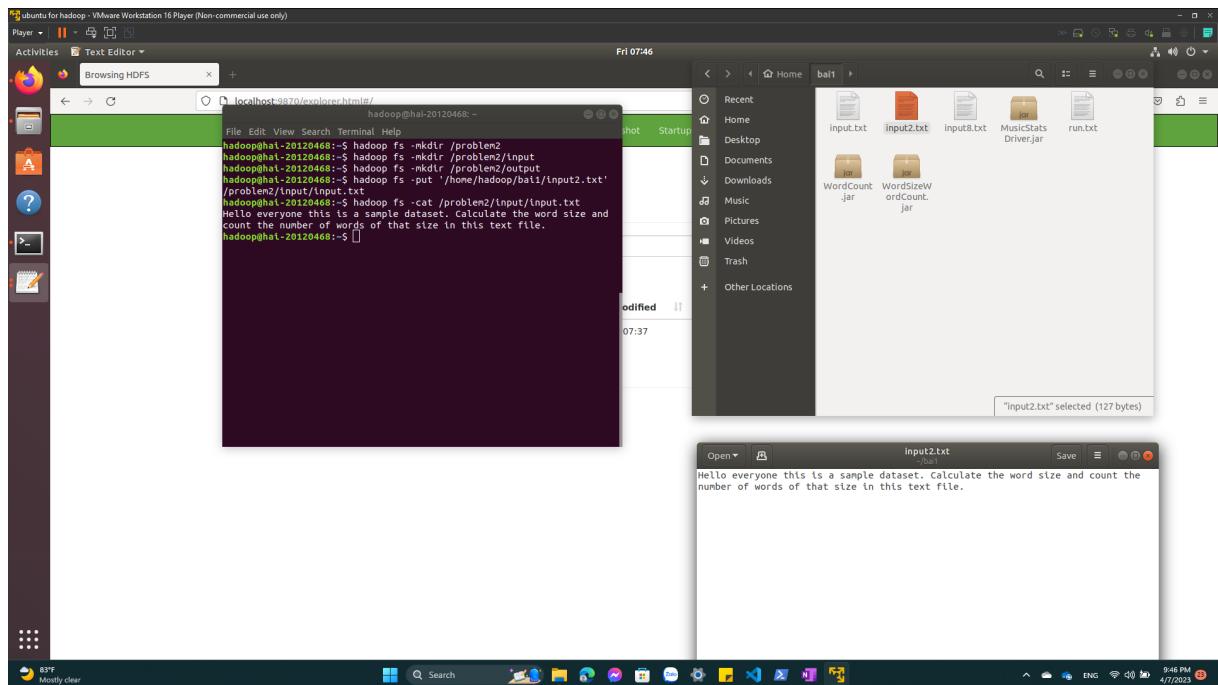
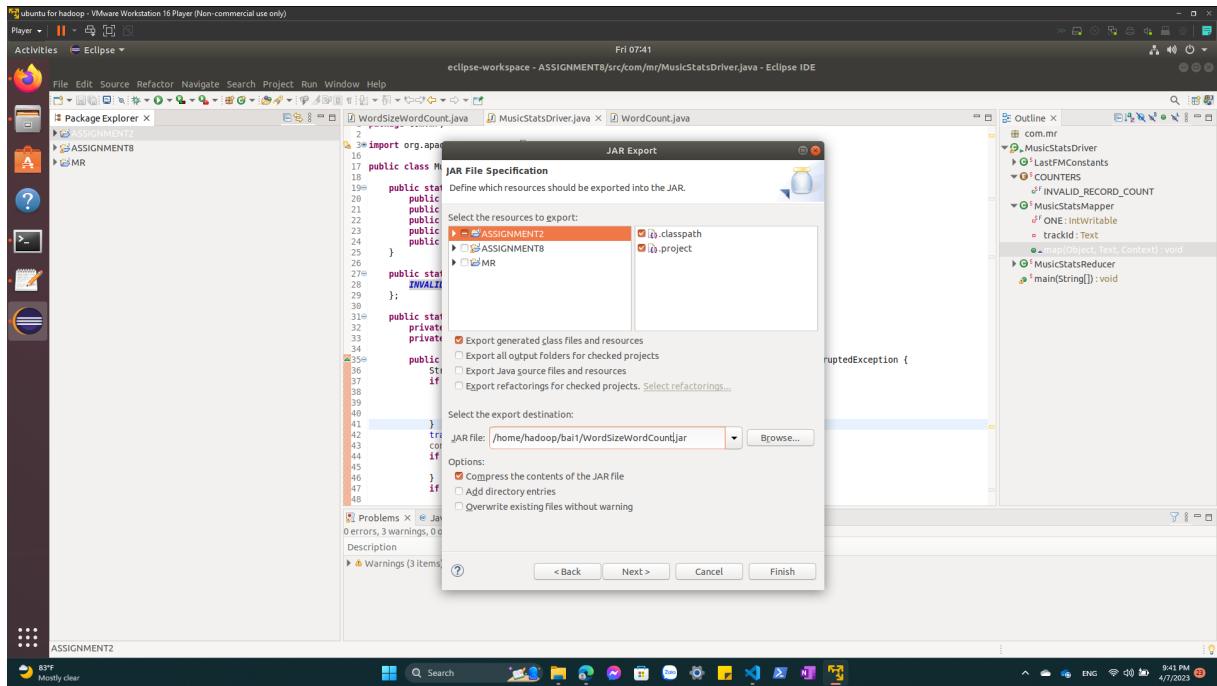
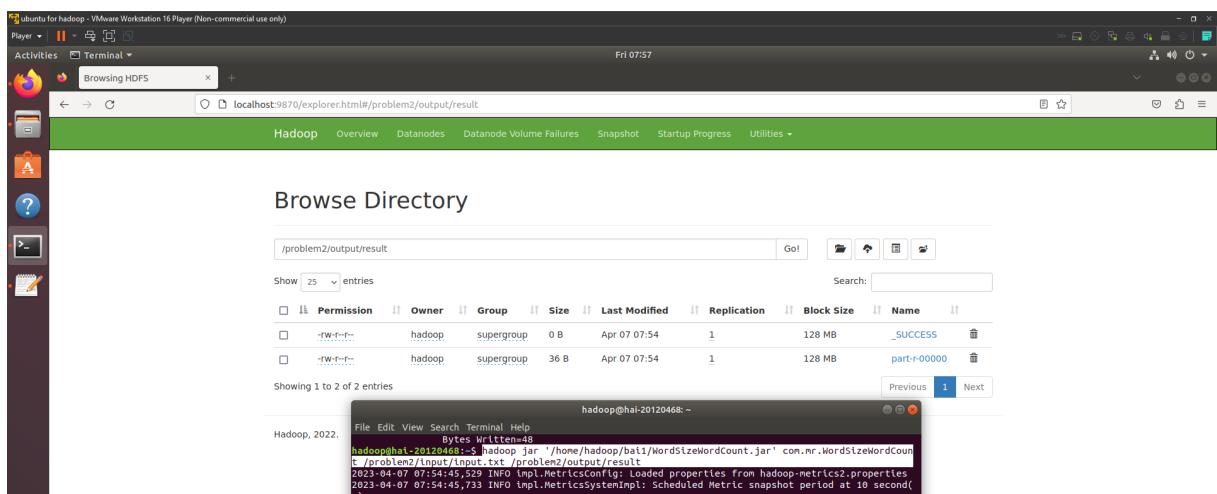


Figure 1.5: input of assignment 2

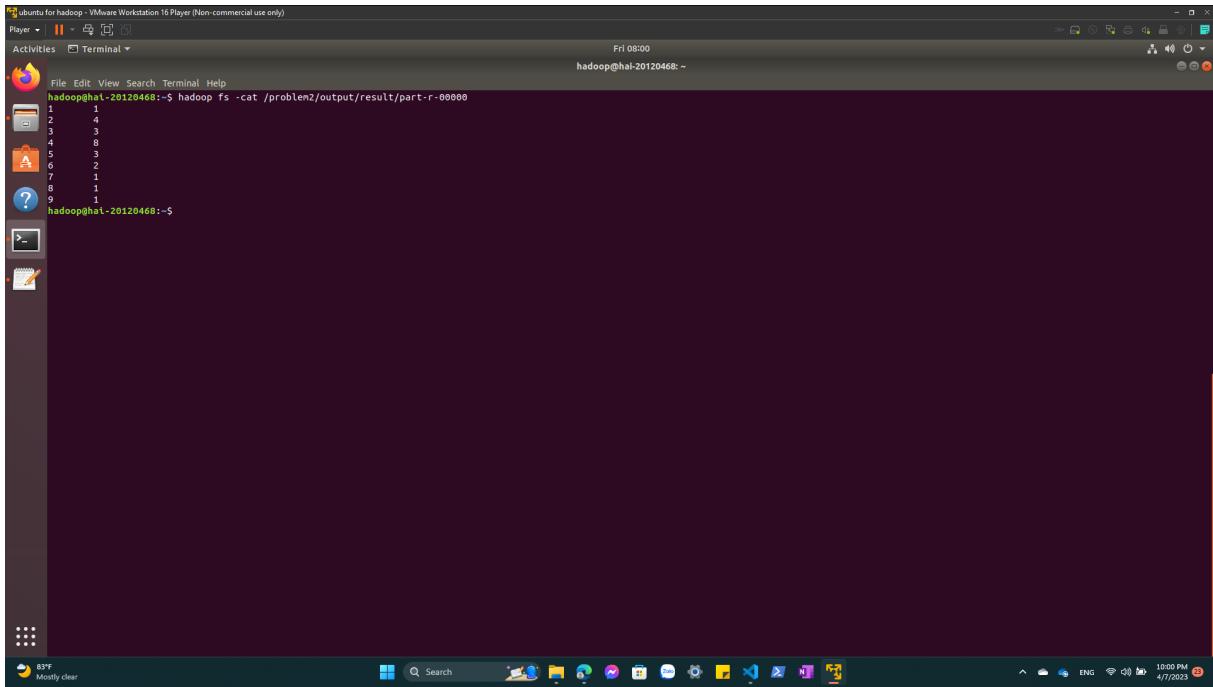
Step 2: Export file jar

**Figure 1.6:** input of assignment 2

Step 3: Running jar

**Figure 1.7:** input of assignment 2

1.4.2.4 Result:



The screenshot shows a terminal window on an Ubuntu desktop environment. The terminal window title is "ubuntu for hadoop - VMware Workstation 16 Player (Non-commercial use only)". The terminal content displays the output of a Hadoop command: "hadoop fs -cat /problem2/output/result/part-r-00000". The output consists of the following numbers:
2
4
3
8
5
3
0
2
7
1
8
1
9
1

Figure 1.8: input of assignment 2

1.4.3 Problem 3

1.4.3.1 Solution idea:

Reads the weather data line by line and extracts the date, maximum temperature, and minimum temperature. Then compare them with the given min max temperature.

1.4.3.2 Explain:

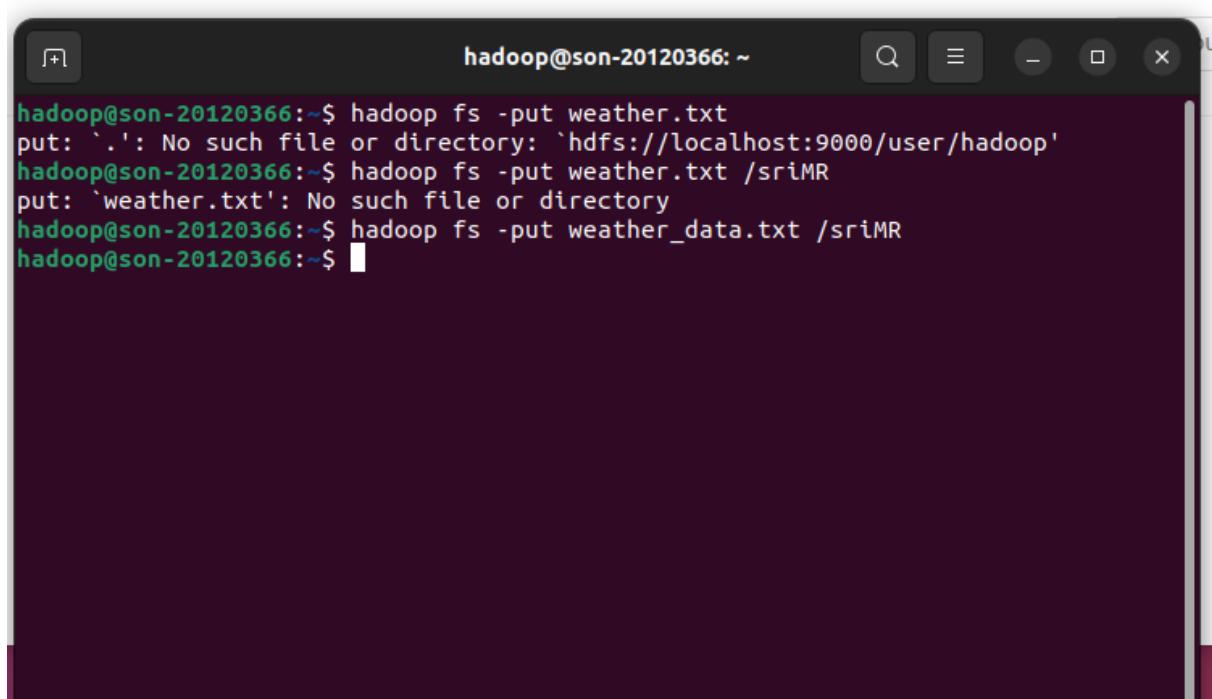
MaxTemperatureMapper class is responsible for reading the weather data and emitting key-value pairs for hot and cold days. The mapper reads each line of the input file, extracts the date, maximum temperature, and minimum temperature, and checks if the maximum temperature is above a predefined HOT_THRESHOLD or if the minimum temperature is below a predefined COLD_THRESHOLD. If either of these conditions is true, the mapper emits a key-value pair with the date as the key and the temperature as the value.

MaxTemperatureReducer class receives the key-value pairs emitted by the mapper and finds the maximum temperature for each date. The reducer iterates over the values for each key and finds the

maximum value. It then emits a key-value pair with the date and maximum temperature.

1.4.3.3 Running process:

Step 1: Put file weather_data.txt



```
hadoop@son-20120366:~$ hadoop fs -put weather.txt
put: '.'': No such file or directory: 'hdfs://localhost:9000/user/hadoop'
hadoop@son-20120366:~$ hadoop fs -put weather.txt /sriMR
put: 'weather.txt': No such file or directory
hadoop@son-20120366:~$ hadoop fs -put weather_data.txt /sriMR
hadoop@son-20120366:~$
```

Figure 1.9: put weather.txt

Step 2: Export Weather.jar

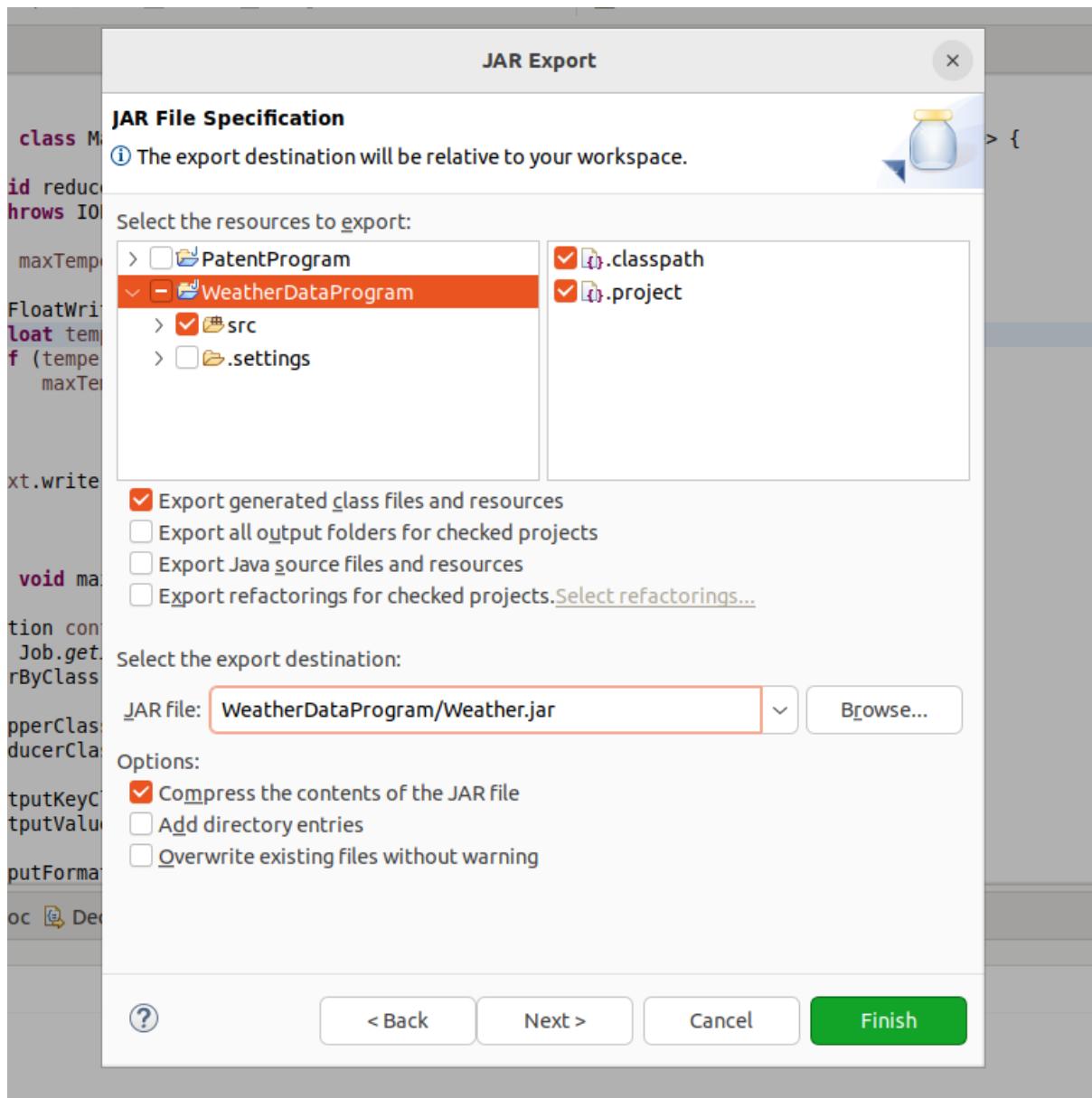


Figure 1.10: export Weather.jar

Step 3: Run Weather.jar

The screenshot shows a terminal window titled "hadoop@son-20120366: ~". The window displays the output of a Java application named WeatherData.java. The application prints various performance metrics and error counts. The metrics include Combine input records=0, Combine output records=0, Reduce input groups=70, Reduce shuffle bytes=1756, Reduce input records=70, Reduce output records=70, Spilled Records=140, Shuffled Maps =1, Failed Shuffles=0, Merged Map outputs=1, GC time elapsed (ms)=17, Total committed heap usage (bytes)=497025024, Shuffle Errors, BAD_ID=0, CONNECTION=0, IO_ERROR=0, WRONG_LENGTH=0, WRONG_MAP=0, WRONG_REDUCE=0, File Input Format Counters, Bytes Read=41881, File Output Format Counters, Bytes Written=1710. The command "Weather.jar" was run at the prompt.

```
weatherData.java x
package weather.project;

import java.io.IOException;

public class WeatherData {

    public static void main(String[] args) throws IOException {
        System.out.println("Combine input records=0");
        System.out.println("Combine output records=0");
        System.out.println("Reduce input groups=70");
        System.out.println("Reduce shuffle bytes=1756");
        System.out.println("Reduce input records=70");
        System.out.println("Reduce output records=70");
        System.out.println("Spilled Records=140");
        System.out.println("Shuffled Maps =1");
        System.out.println("Failed Shuffles=0");
        System.out.println("Merged Map outputs=1");
        System.out.println("GC time elapsed (ms)=17");
        System.out.println("Total committed heap usage (bytes)=497025024");
        System.out.println("Shuffle Errors");
        System.out.println("BAD_ID=0");
        System.out.println("CONNECTION=0");
        System.out.println("IO_ERROR=0");
        System.out.println("WRONG_LENGTH=0");
        System.out.println("WRONG_MAP=0");
        System.out.println("WRONG_REDUCE=0");
        System.out.println("File Input Format Counters");
        System.out.println("Bytes Read=41881");
        System.out.println("File Output Format Counters");
        System.out.println("Bytes Written=1710");
    }
}

hadoop@son-20120366:~$ Weather.jar
```

Figure 1.11: run Weather.jar

1.4.3.4 Result:

```
hadoop@son-20120366: ~
20150226 Cold Day      1.4E-45
20150227 Cold Day      1.4E-45
20150228 Cold Day      1.4E-45
20150301 Cold Day      1.4E-45
20150302 Cold Day      1.5
20150306 Cold Day      1.4E-45
20150307 Cold Day      4.4
20150308 Cold Day      6.8
20150309 Cold Day      8.1
20150310 Cold Day      8.4
20150312 Cold Day      9.4
20150315 Cold Day      9.5
20150327 Cold Day      7.3
20150404 Cold Day      9.4
20150420 Cold Day      9.4
20150428 Cold Day      9.1
20150429 Cold Day      8.0
20150608 Cold Day      1.4E-45
20150609 Cold Day      1.4E-45
20150613 Cold Day      1.4E-45
20150615 Cold Day      1.4E-45
20150617 Cold Day      1.4E-45
20150618 Cold Day      1.4E-45
hadoop@son-20120366:~$
```

Figure 1.12: output weather**1.4.4 Problem 4****1.4.4.1 Solution idea:**

Extract the patent id and the sub-patent id in Map. Then send it to Reduce for counting.

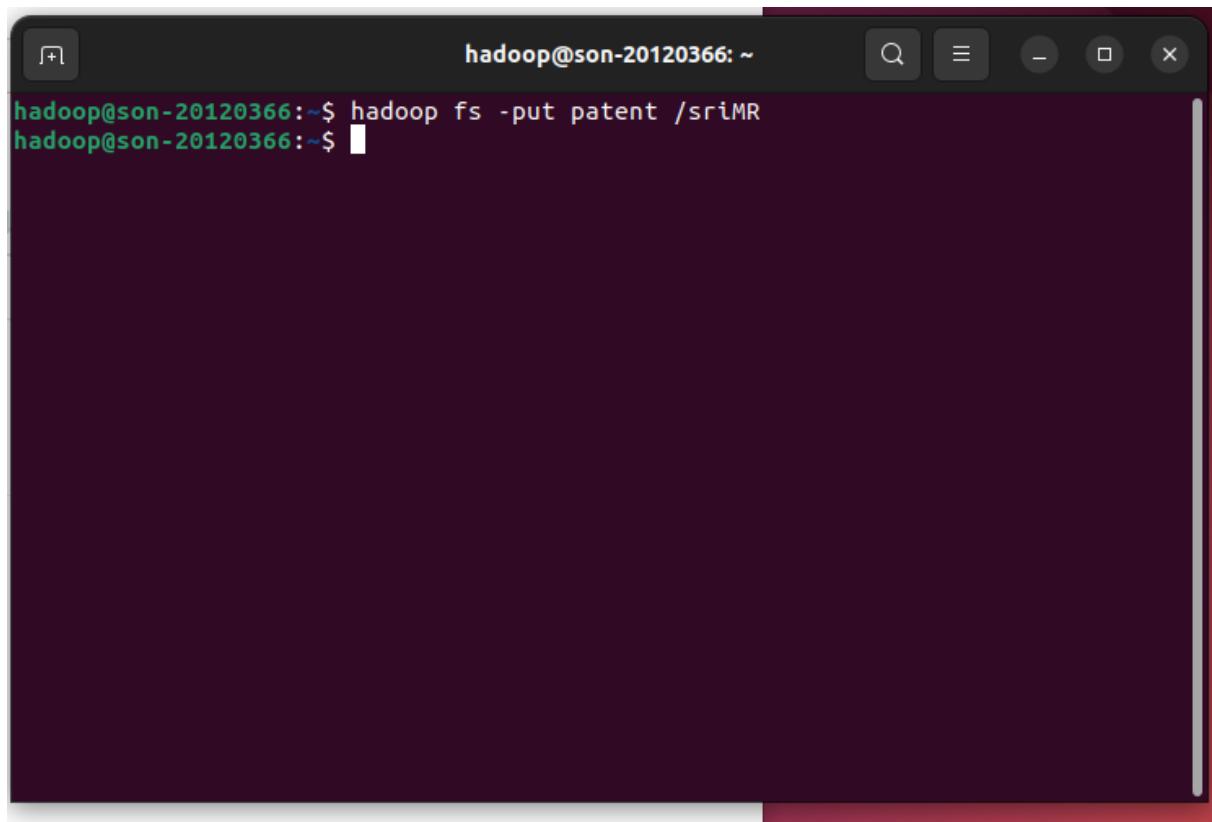
1.4.4.2 Explain:

In the class map, the input line is first converted to a String using the `toString()` method of the `Text` class. Then, a `StringTokenizer` object is created to split the line into tokens using space as the delimiter. The first token represents the patent ID, which is set as the key of the output tuple. The second token represents the sub-patent ID, which is set as the value of the output tuple. Finally, the output tuple (key, value) is written to the context using the `write()` method. In the class `reduce`, the number of sub-patents for each patent is counted by iterating through the list of sub-patent IDs and incrementing a

counter variable for each sub-patent ID

1.4.4.3 Running process:

Step 1: Put file patent



A screenshot of a terminal window titled "hadoop@son-20120366: ~". The window shows a command being entered: "hadoop fs -put patent /sriMR". The terminal has a dark background and light-colored text. The command is partially visible at the top of the terminal window.

Figure 1.13: put patent

Step 2: Export Patent.jar

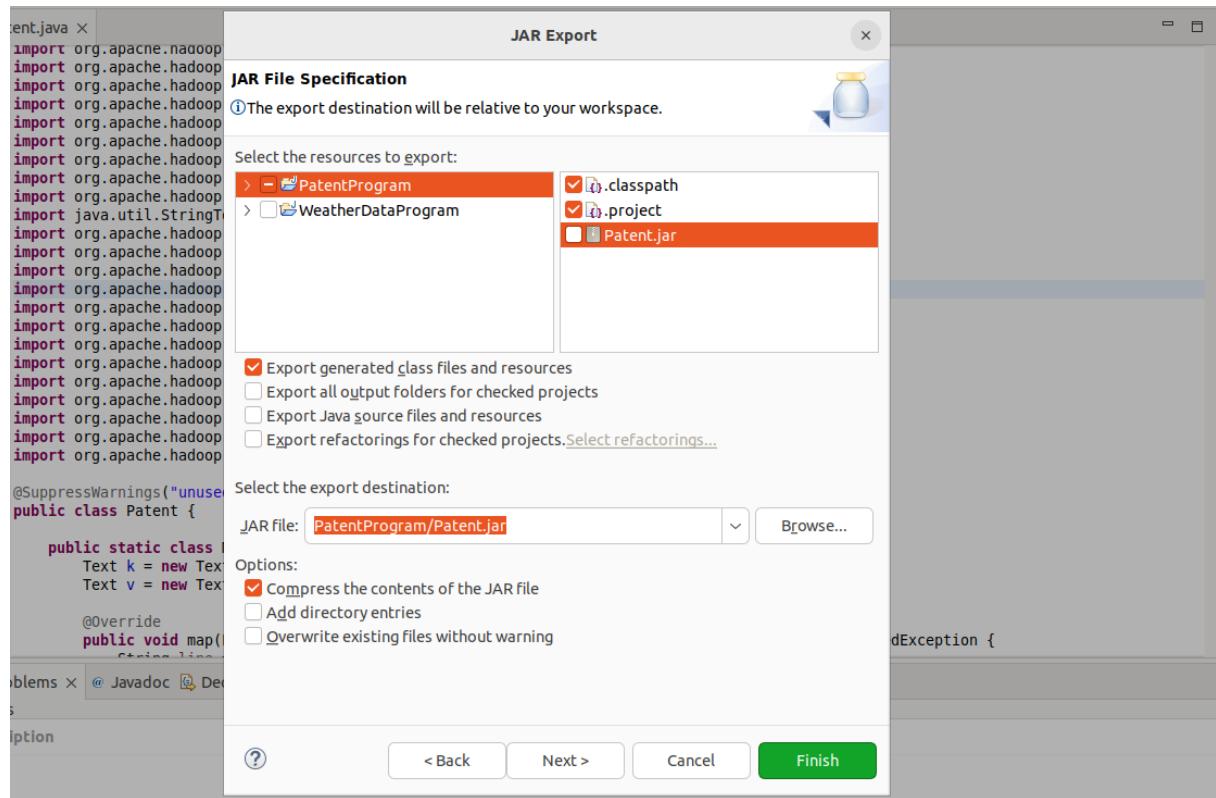
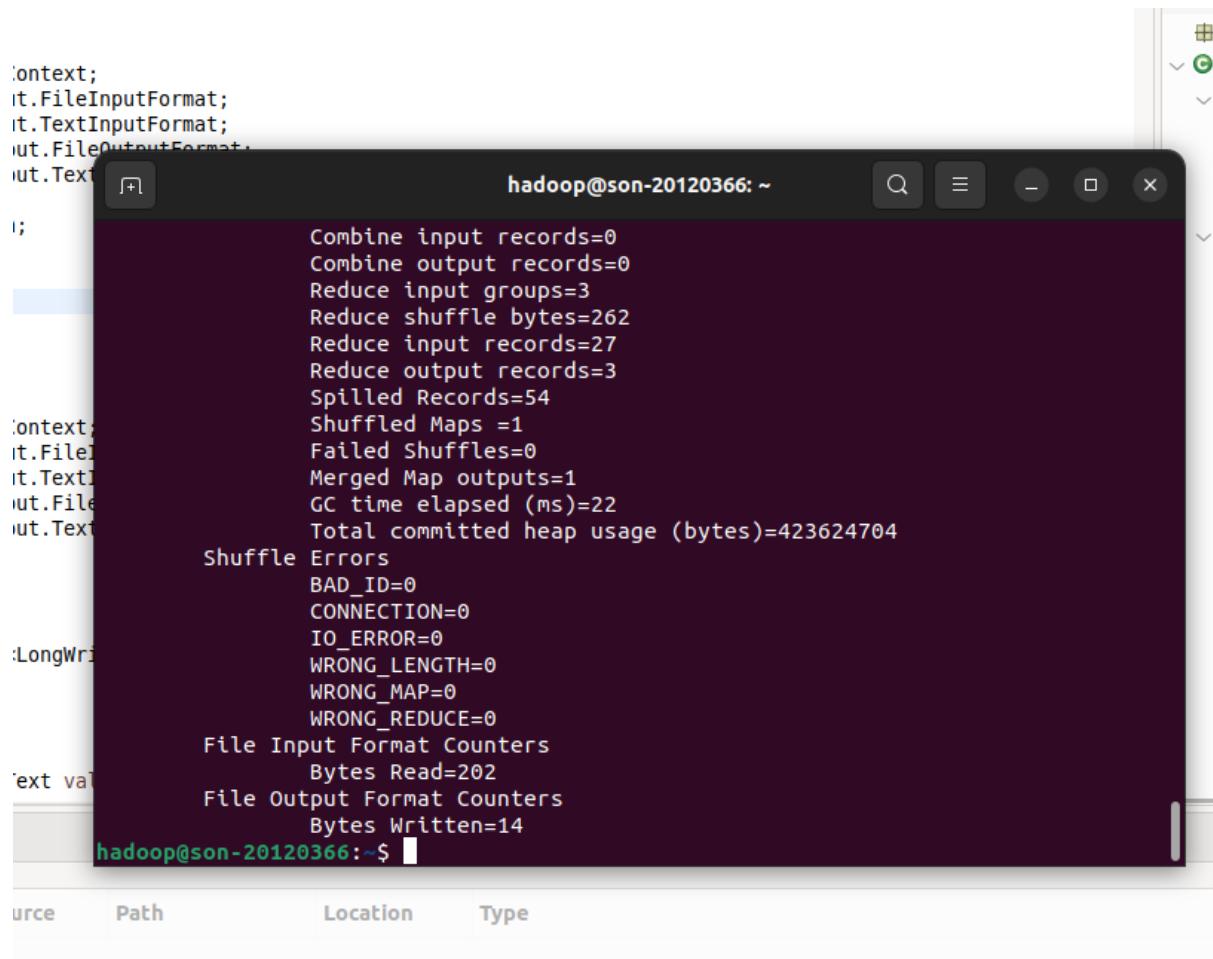


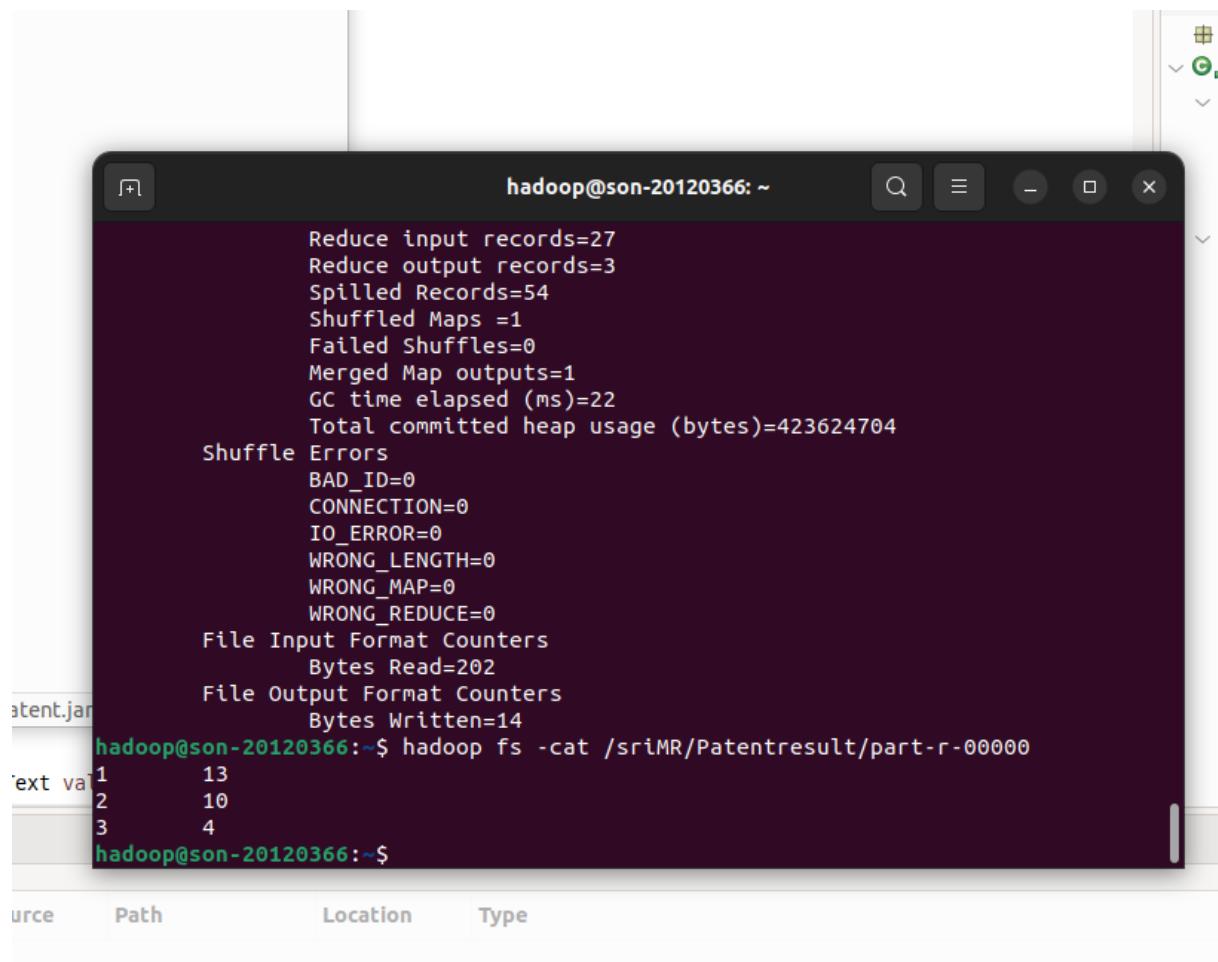
Figure 1.14: export Patent.jar

Step 3: Run Patent.jar



```
hadoop@son-20120366: ~
Combine input records=0
Combine output records=0
Reduce input groups=3
Reduce shuffle bytes=262
Reduce input records=27
Reduce output records=3
Spilled Records=54
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=22
Total committed heap usage (bytes)=423624704
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=202
File Output Format Counters
Bytes Written=14
hadoop@son-20120366:~$
```

Figure 1.15: run Patent.jar

1.4.4.4 Result:

The screenshot shows a terminal window titled "hadoop@son-20120366: ~". The window displays the following output from a Hadoop job:

```
Reduce input records=27
Reduce output records=3
Spilled Records=54
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=22
Total committed heap usage (bytes)=423624704
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=202
File Output Format Counters
  Bytes Written=14
hadoop@son-20120366:~$ hadoop fs -cat /sriMR/Patentresult/part-r-00000
1 13
2 10
3 4
hadoop@son-20120366:~$
```

Below the terminal window, there is a table with columns: Source, Path, Location, and Type. The table has three rows, but only the first row is visible.

Source	Path	Location	Type
Text	Patent.jar		
Text	var		

Figure 1.16: output patent**1.4.5 Problem 5****1.4.5.1 Solution idea:**

The mapper extracts the year and temperature values and emits them as key-value pairs, where the year is the key and the temperature is the value. The reducer receives all key-value pairs within the same year and calculates the maximum temperature for that year.

1.4.5.2 Explain:

The mapper reads the input dataset line by line and tokenizes each line using whitespace as the delimiter. It checks whether the first token is a 4-digit number and sets the “year” variable accordingly. If the token is not a year, it assumes it is a temperature and sets the “temperature” variable accordingly. It then emits a (key, value) pair, where the key is the year and the value is the temperature.

The reducer receives these (key, value) pairs, where the key is the year and the value is a list of temperatures. It iterates over the list of temperatures and finds the maximum temperature for that year. It then emits a (key, value) pair, where the key is the year and the value is the maximum temperature.

1.4.5.3 Running process:

Step 1: Put file Temperature.txt into HDFS

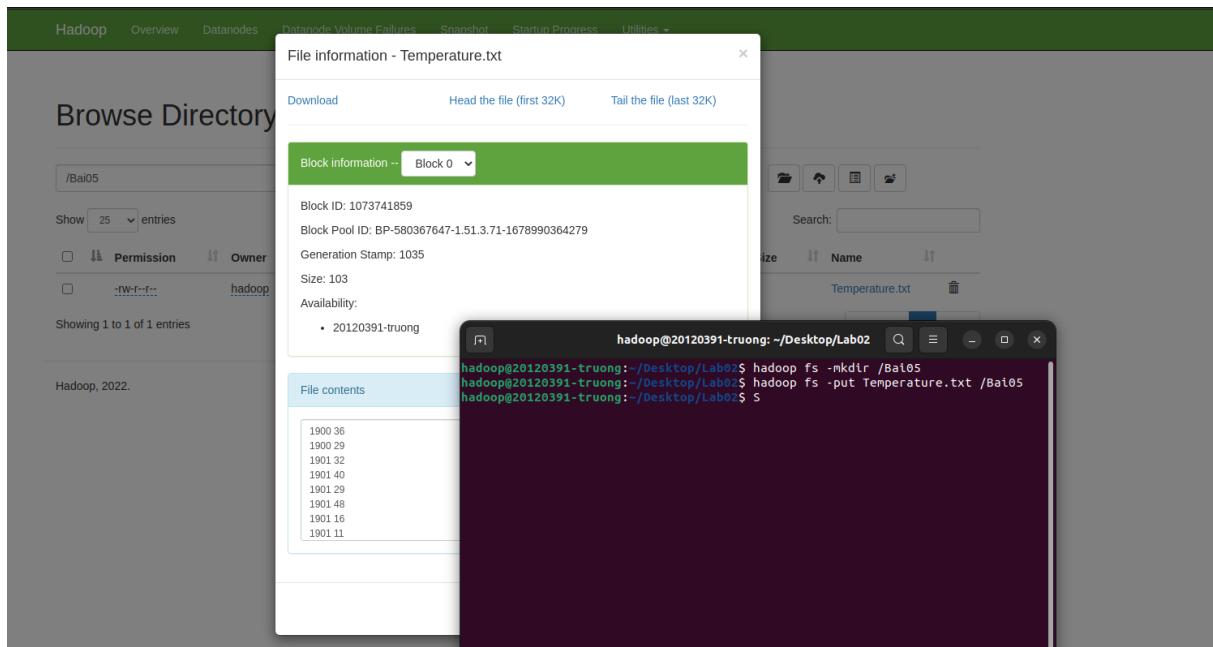
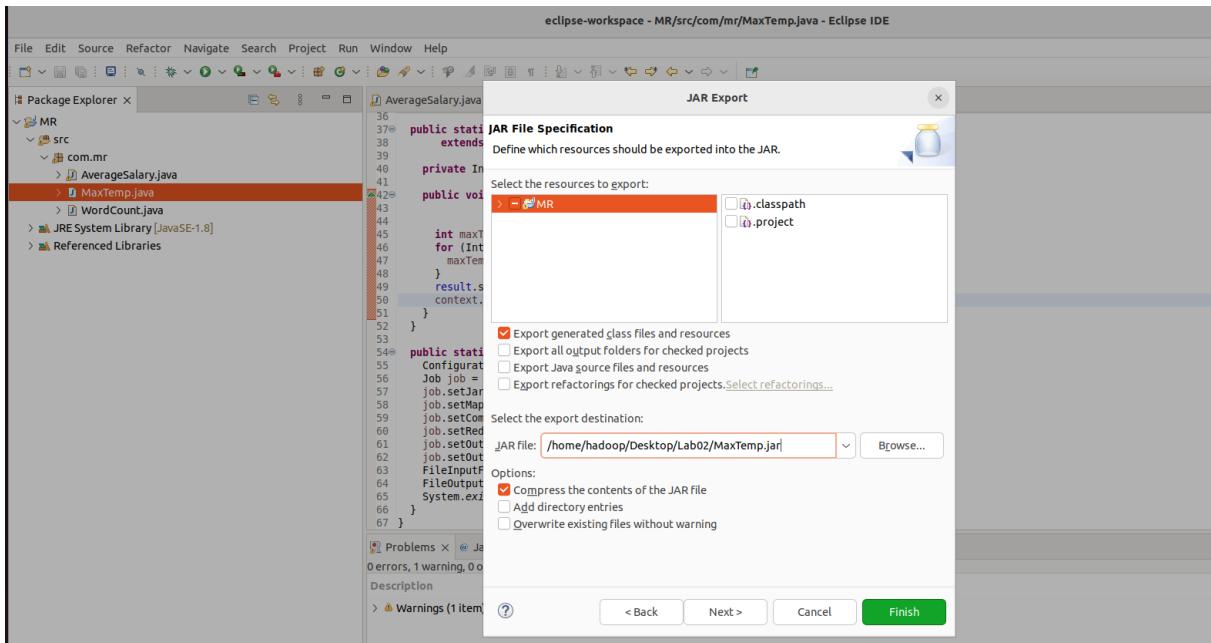
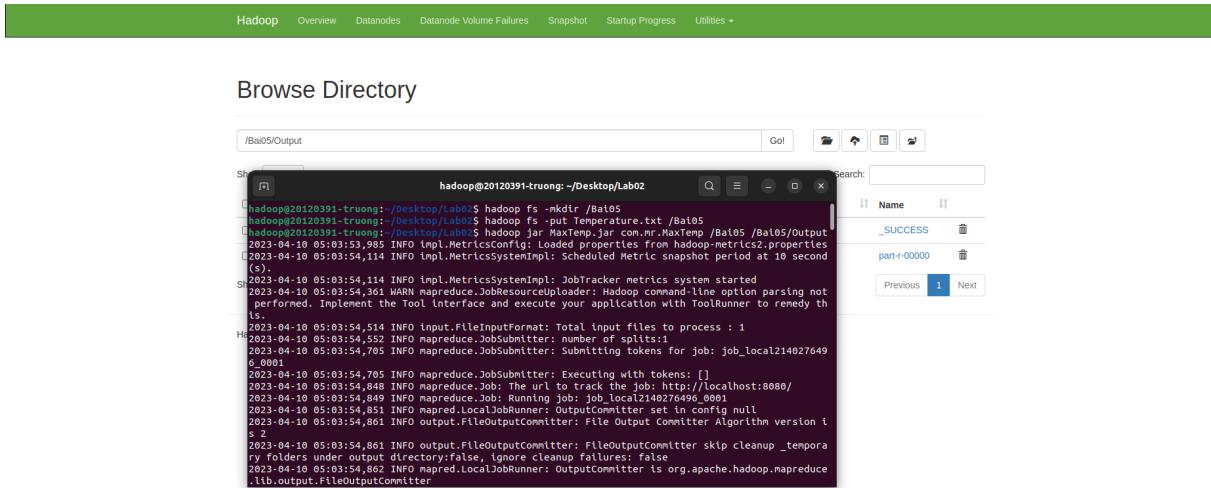


Figure 1.17: Put Temperature.txt into HDFS

Step 2: Export MaxTemp.jar

**Figure 1.18:** Export MaxTemp.jar**Step 3: Run MapReduce program****Figure 1.19:** Run MapReduce program

1.4.5.4 Result:

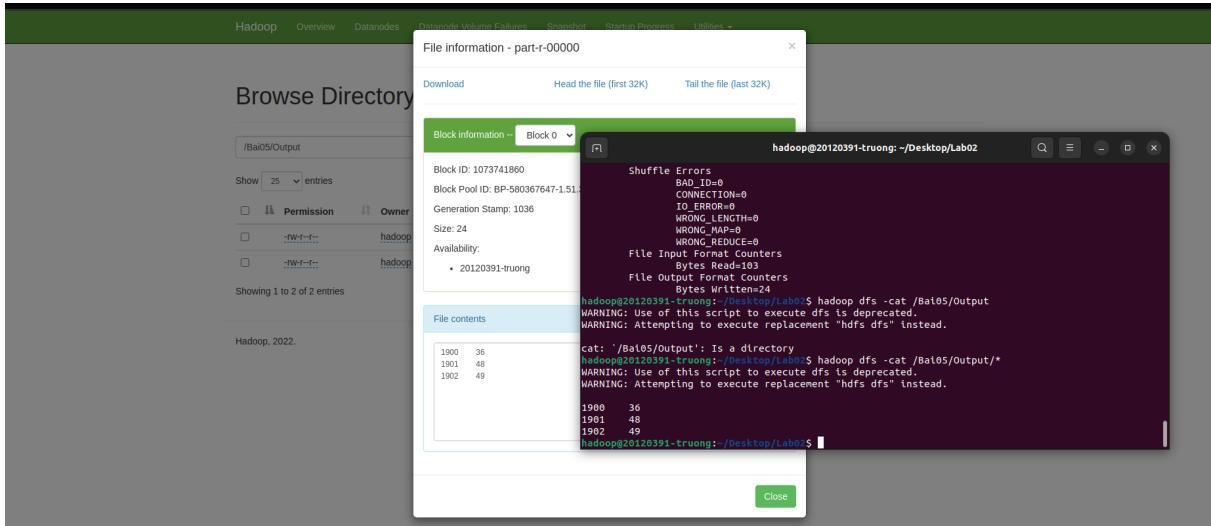


Figure 1.20: Output

1.4.6 Problem 6

This code is taken from file: LabRequirement.pdf

1.4.6.1 Solution idea:

The program consists of a mapper class, which reads input data and emits key-value pairs of department IDs and salaries, and a reducer class, which receives the key-value pairs from the mapper, calculates the average salary for each department, and outputs the result.

1.4.6.2 Explain:

In the mapper class `avgMapper`, each line of input is split into three fields separated by tabs. The first field represents the department id, the second field represents the employee name, and the third field represents the salary. The department id and salary are extracted and stored in `Text` and `FloatWritable` objects, respectively, and emitted as key-value pairs.

In the reducer class `avgReducer`, the input key-value pairs are grouped by department id, and the average salary is calculated for each department. The `reduce()` method iterates over the values and calculates the sum and count of salaries for each department. Finally, the average salary is calculated as the sum divided by the count, and the result is emitted as a key-value pair.

1.4.6.3 Running process:

Step 1: Put file salary.txt into HDFS

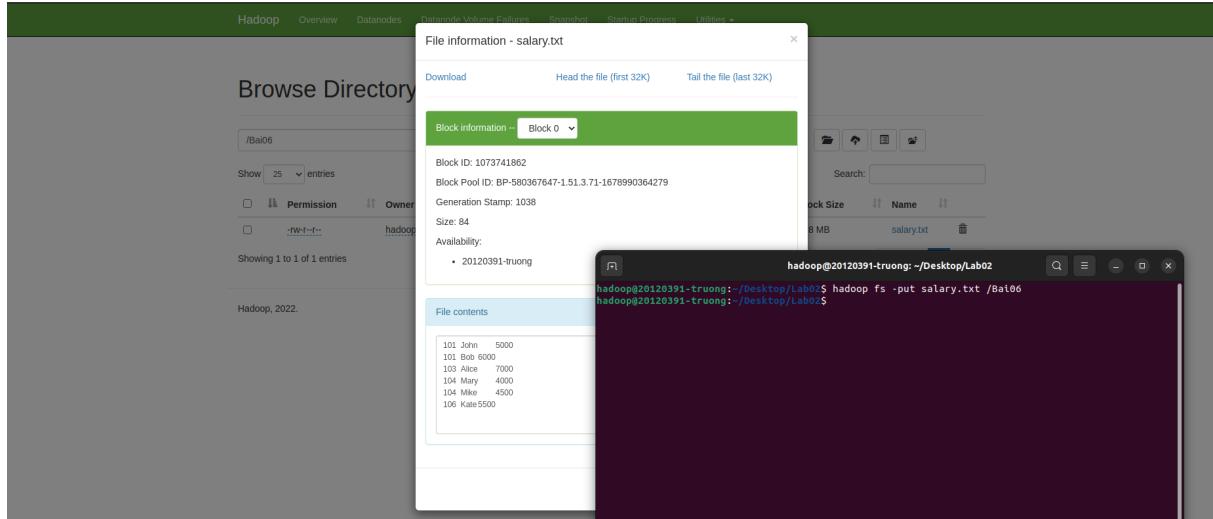


Figure 1.21: Put salary.txt into HDFS

Step 2: Export AverageSalary.jar

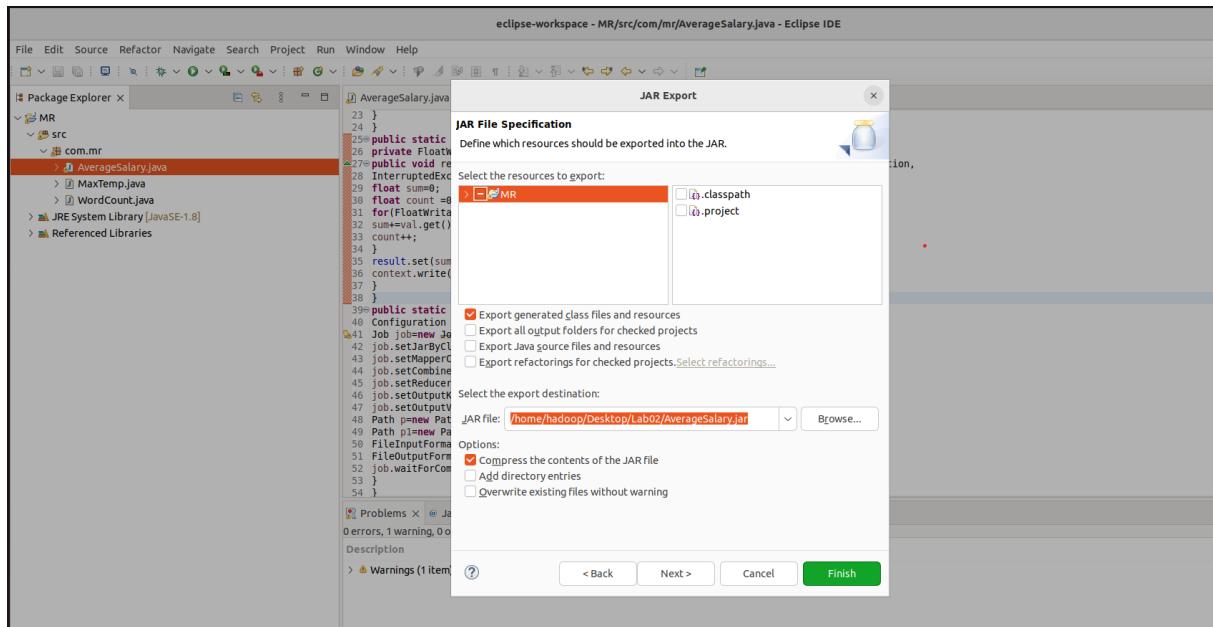


Figure 1.22: Export AverageSalary.jar

Step 3: Run MapReduce program

```

hadoop@20120391-truong:~/Desktop/Lab02$ hadoop fs -put salary.txt /Bal06
hadoop@20120391-truong:~/Desktop/Lab02$ hadoop jar AverageSalary.jar com.mr.AverageSalary /Bal06 /
hadoop@20120391-truong:~/Desktop/Lab02$ hadoop@20120391-truong:~/Desktop/Lab02$ hadoop jar AverageSalary.jar com.mr.AverageSalary /Bal06 /
2023-04-10 05:11:36,286 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2023-04-10 05:11:36,468 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second
(s).
2023-04-10 05:11:36,461 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2023-04-10 05:11:36,756 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not
performed. Implement the Tool interface and execute your application with ToolRunner to remedy th
is.
2023-04-10 05:11:36,947 INFO input.FileInputFormat: Total input files to process : 1
2023-04-10 05:11:36,998 INFO mapreduce.JobSubmitter: number of splits:1
2023-04-10 05:11:37,185 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local392747964
_0001
2023-04-10 05:11:37,185 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-04-10 05:11:37,424 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
2023-04-10 05:11:37,424 INFO mapreduce.Job: Running local
2023-04-10 05:11:37,452 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2023-04-10 05:11:37,476 INFO output.FileOutputCommitter: File Output Committer Algorithm version i

```

Figure 1.23: Run MapReduce program

1.4.6.4 Result:

```

hadoop@20120391-truong:~/Desktop/Lab02$ hadoop dfs -cat /Bal06/Output/*
WARNING: Use of this script to execute dfs is deprecated!
WARNING: Attempting to execute replacement "hdfs dfs" instead.

101 6500.0
103 7000.0
104 4250.0
105 5500.0

```

Figure 1.24: Output

1.4.7 Problem 7

The problem statement in the book was unclear, and I had to conduct some internet research to understand the nature of the problem.

1.4.7.1 Solution idea:

Create a function called `stringToEncrypt` to encrypt a given string. **Mapper:** Extract fields from input string. For each fields that requires encryption, apply the `stringToEncrypt` function.

1.4.7.2 Explain:

This program is a Java-based implementation of MapReduce that aims to de-identify data by encrypting specific fields. It accepts a CSV input file and encrypts the specified fields using the AES algorithm. The resulting encrypted data is written to an output file, where each field is separated by a comma. The program leverages Hadoop to solve this problem efficiently by processing the input data in parallel, which is particularly useful for encrypting large datasets.

The program defines a static class named `DeIdentifyData` that contains a `Map` class, a `stringToEncrypt` function, and a `main` function.

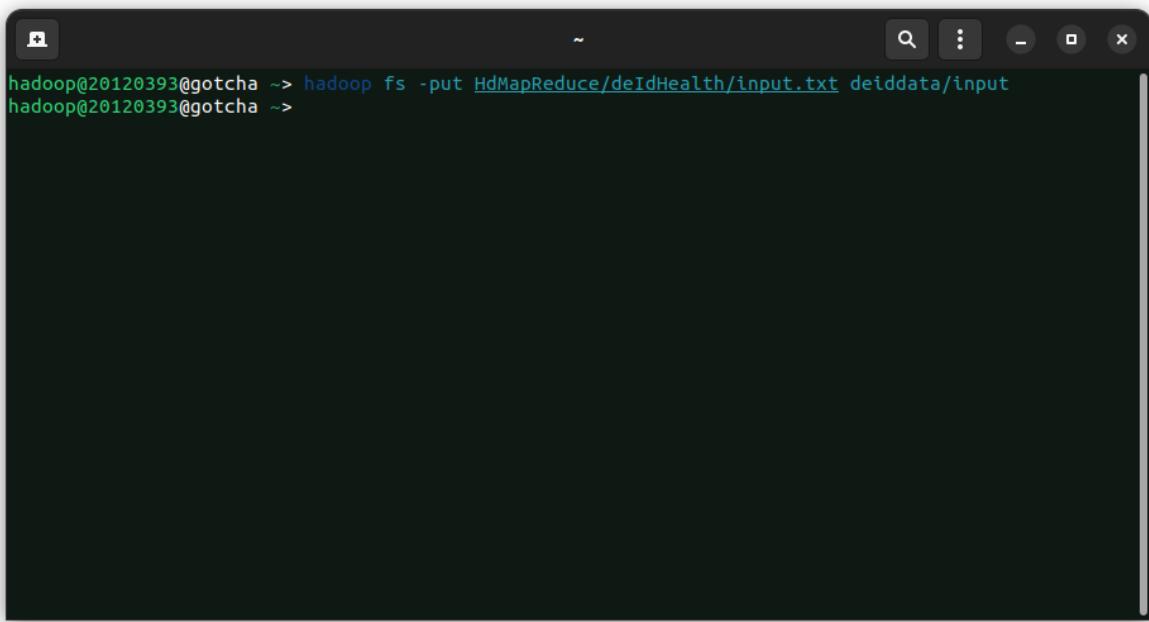
The `Map` class extends the `Mapper` class, which is a component of Hadoop MapReduce that processes the input data in a parallel and distributed manner. The input data is split into lines, and each line is further split into fields based on the CSV format.

The program uses a list of integers `ENCRYPT_COLS` to specify which fields need to be encrypted. The `Map` class checks whether a field is in this list by calling the `encryptColList.contains(i + 1)` method. If the field needs to be encrypted, the program calls the `stringToEncrypt` function, which encrypts the input string using the AES algorithm and a secret key. The encrypted string is then returned to the `Map` class, where it is appended to a `StringBuilder` object. After all the fields in a line have been processed, the `StringBuilder` object is converted to a `Text` object and written to the output file.

The `stringToEncrypt` function takes a string as input and returns an encrypted string. The function first creates a `SecretKeySpec` object using the `SECRET_KEY` and the AES algorithm. It then creates a `Cipher` object using the same algorithm and initializes it in encryption mode with the secret key. The function then encrypts the input string using the `doFinal` method and encodes the result using Base64 encoding.

1.4.7.3 Running process:

Step 1: Put file `input.txt` into HDFS



A screenshot of a terminal window with a dark background. At the top, there are standard window control icons (minimize, maximize, close) and a search bar. The terminal prompt shows the user's session: "hadoop@20120393@gotcha ~>". Below the prompt, the command "hadoop fs -put HdMapReduce/deIdHealth/input.txt deiddata/input" is entered and executed. The output shows the command was successful, indicated by the green text "hadoop@20120393@gotcha ~>".

Figure 1.25: Put input.txt into HDFS

Step 2: Export DeIdData.jar

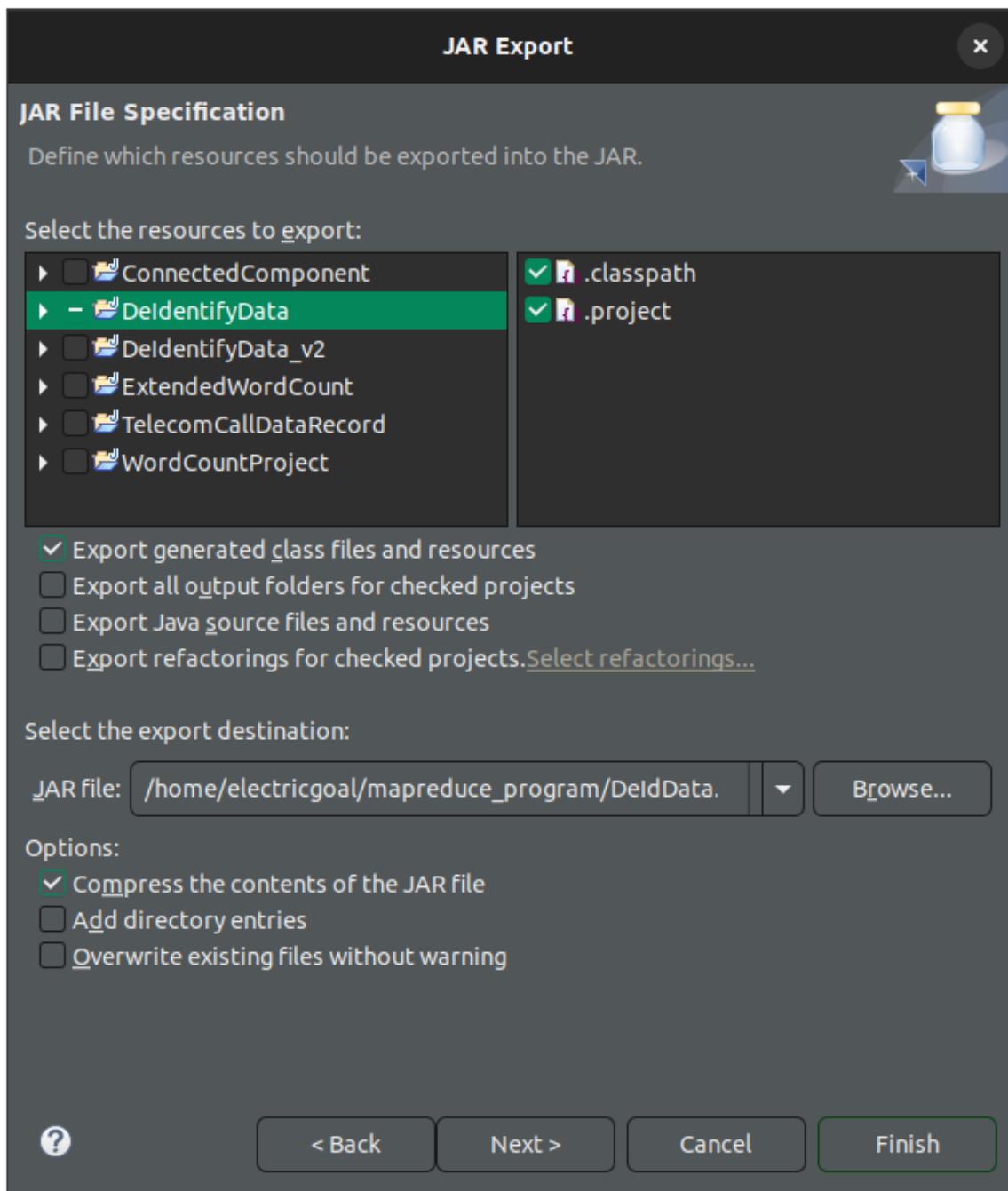


Figure 1.26: Export DeldData.jar

Step 3: Run MapReduce program

```

hadoop@20120393@gotcha -> hadoop jar HdMapReduce/deIdHealth/DeIdData.jar com.deidata.DeIdentifyData deiddata/input/input.txt deiddata/output
2023-04-05 16:45:34,622 INFO client.DefaultNoharmFallbackProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-04-05 16:45:35,129 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2023-04-05 16:45:35,161 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/hadoop/.staging/job_1680687371580_0001
2023-04-05 16:45:35,482 INFO input.FileInputFormat: Total input files to process : 1
2023-04-05 16:45:35,589 INFO mapreduce.JobSubmitter: number of splits:1
2023-04-05 16:45:36,357 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1680687371580_0001
2023-04-05 16:45:36,580 INFO conf.Configuration: resource-types.xml not found
2023-04-05 16:45:36,580 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-04-05 16:45:36,919 INFO impl.YarnClientImpl: Submitted application application_1680687371580_0001
2023-04-05 16:45:37,020 INFO mapreduce.Job: The url to track the job: http://gotcha:8088/proxy/application_1680687371580_0001/
2023-04-05 16:45:37,021 INFO mapreduce.Job: Running job: job_1680687371580_0001
2023-04-05 16:45:47,319 INFO mapreduce.Job: Job job_1680687371580_0001 running in uber mode : false
2023-04-05 16:45:47,320 INFO mapreduce.Job: map 0% reduce 0%
2023-04-05 16:45:54,560 INFO mapreduce.Job: map 100% reduce 0%
2023-04-05 16:45:54,579 INFO mapreduce.Job: Job job_1680687371580_0001 completed successfully
2023-04-05 16:45:54,685 INFO mapreduce.Job: Counters: 33
  File System Counters
    FILE: Number of bytes read=0
    FILE: Number of bytes written=275440
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=520
    HDFS: Number of bytes written=966
    HDFS: Number of read operations=7
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
    HDFS: Number of bytes read erasure-coded=0
  Job Counters
    Launched map tasks=1
    Data-local map tasks=1
    Total time spent by all maps in occupied slots (ms)=4008
    Total time spent by all reduces in occupied slots (ms)=0
    Total time spent by all map tasks (ms)=4008

```

Figure 1.27: Run MapReduce program

1.4.7.4 Result:

```

input.txt
admin:///home/hadoop/HdMapReduce/deIdHealth
1 1111,bbb1,12/10/1950,1.23E+09,bbb1@xxx.com,1.11E+09,M,Diabetes,78
2 1112,bbb2,12/10/1984,1.23E+09,bbb2@xxx.com,1.11E+09,F,PCOS,67
3 1113,bbb3,7/12/11/1940,1.23E+09,bbb3@xxx.com,1.11E+09,M,Fever,90
4 1114,bbb4,12/12/1950,1.23E+09,bbb4@xxx.com,1.11E+09,F,Cold,88
5 1115,bbb5,12/13/1960,1.23E+09,bbb5@xxx.com,1.11E+09,M,Blood Pressure,76
6 1116,bbb6,12/14/1970,1.23E+09,bbb6@xxx.com,1.11E+09,F,Malaria,84

hadoop@20120393@gotcha -> hadoop fs -cat deiddata/output/part-m-00000
1111,anGpkoej8t bv+YRBj9ILCw==,stvKAXcCPYu8optqhETnCA==,92+HjLdrXhZDmk+CJ9C7Ew==,PEzo2egr4iY6QRBBzx6LA==,W8ZvunKVWXuMnC5f9M0HFg==,M,rq4fO2Zduny5P2qjX
/nY0g==,.78
11112,Vn66zEn0fEWwxE+Hn/xUQ==,pkrN8585d13L0SrkozMRbg==,92+HjLdrXhZDmk+CJ9C7Ew==,yoLQQVemvqSlZ0hm9K8gA==,W8ZvunKVWXuMnC5f9M0HFg==,F,NH1fn+ugtA3ZFdEZT
oMz0A==,.67
11113,Ieq/XHU0ONuZsjdhP3UffA==,0FQkvbltvk5Mb0m0xzZL/A==,92+HjLdrXhZDmk+CJ9C7Ew==,NpBRIYWFs+DUwObFBByIKxQ==,W8ZvunKVWXuMnC5f9M0HFg==,M,uXB+yV3211GD8t3VD
kMOGA==,.90
11114,qvJ0SbDTVG/3GOwyS9NIWA==,zsSm09ZgAv7HFoSh+bXjA==,92+HjLdrXhZDmk+CJ9C7Ew==,BsJPiE7mLFWHHINjqqeZNg==,W8ZvunKVWXuMnC5f9M0HFg==,F,qmQ+oIbdN7EI//gGk
r6KQg==,.88
11115,4lf2YmmA7JDztUKjHnoegw==,RSwnUi6fHu9NTe0ip13t8Q==,92+HjLdrXhZDmk+CJ9C7Ew==,1I4RvllocxAnKjpIMMQz9g==,W8ZvunKVWXuMnC5f9M0HFg==,M,Dd+h3BGFXYXxmUvUx
+UDyA==,.76
11116,1eYUtv-h70nsV+zL0k9C0g==,4uPgxtT/IYmdTRrpkTRA==,92+HjLdrXhZDmk+CJ9C7Ew==,i7mt7Fhut2SOBLJiu2XpkGg==,W8ZvunKVWXuMnC5f9M0HFg==,F,L5wtPkGhn8oaNsgJY
Rch+g==,.84
hadoop@20120393@gotcha -> []

```

Figure 1.28: Output

1.4.8 Problem 8

1.4.8.1 Solution idea:

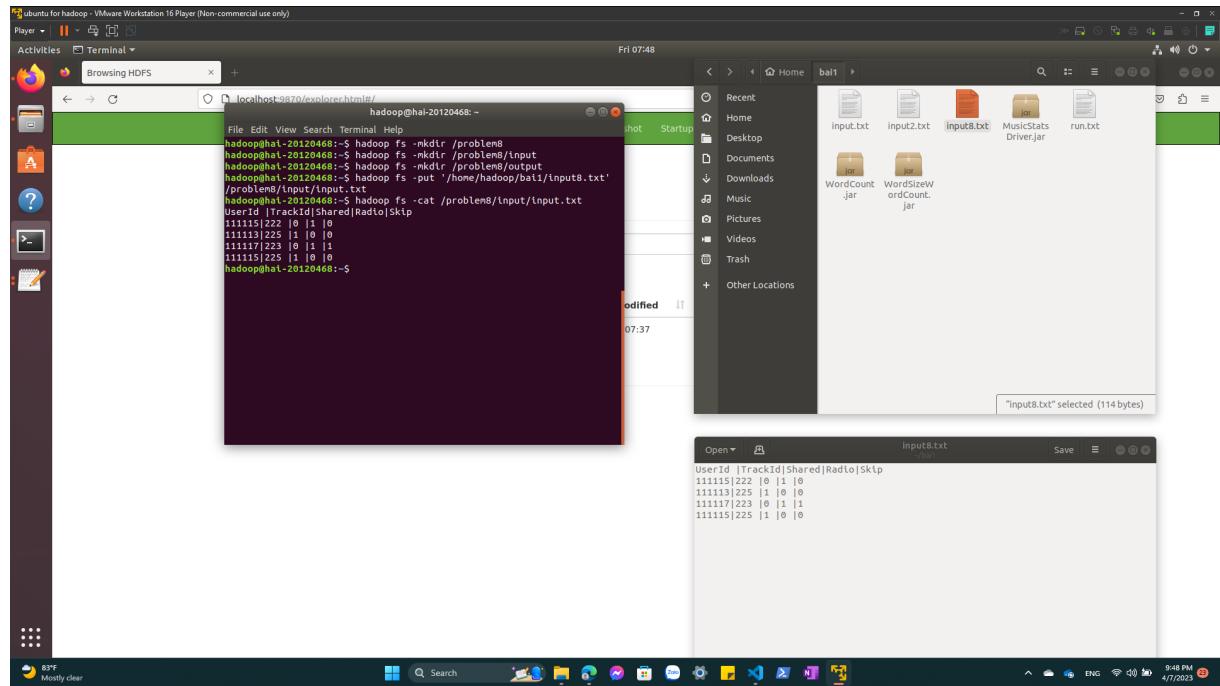
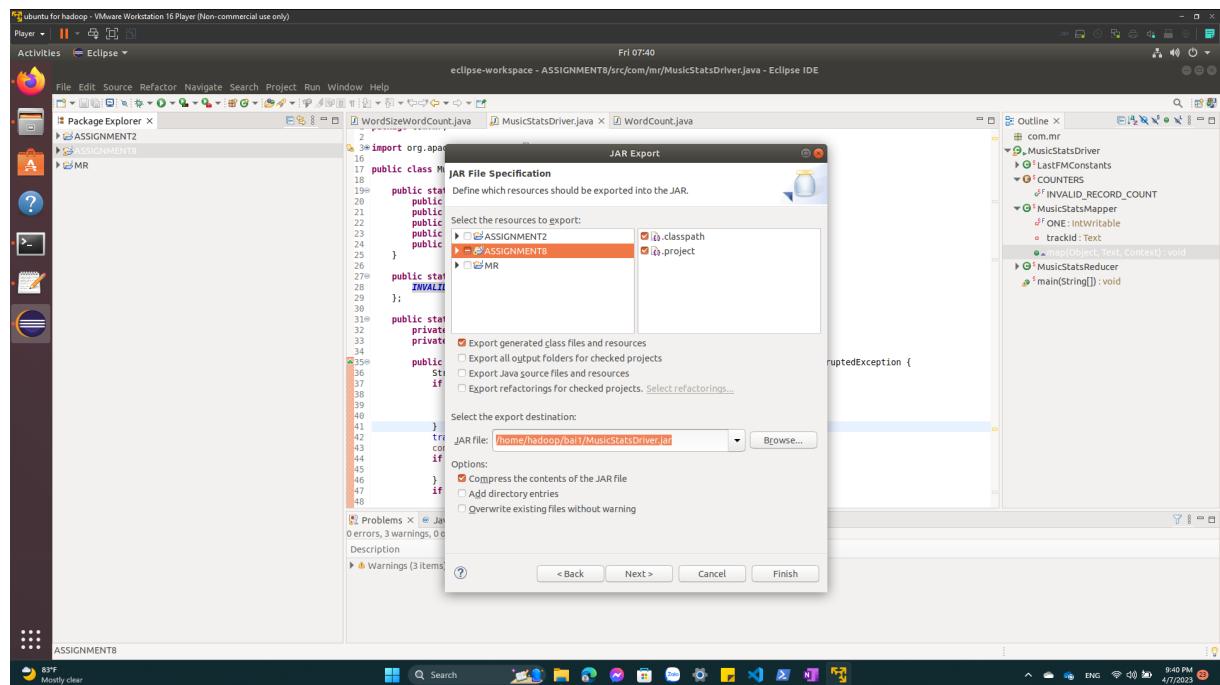
The program takes a dataset of music listening events and computes various statistics related to the number of times each song was listened to, the number of unique listeners, whether a song was shared, listened to on the radio, and skipped on the radio. The program also handles invalid records in the dataset by counting them and outputting the count at the end.

1.4.8.2 Explain:

1. The Mapper class: The code defines a Mapper class called `MusicStatsMapper` which extends the Hadoop MapReduce Mapper class. In the mapper, each record is split into its constituent parts and checked for validity, and if it is valid, the mapper emits key-value pairs where the key is a string representing the type of statistic to be calculated and the value is always
 1. The keys are formed by concatenating the relevant prefix with the track ID. For example, `unique_listeners_123` indicates that the track with ID 123 has been played by a unique listener.
2. The Reducer class: The code also defines a Reducer class called `MusicStatsReducer` which extends the Hadoop MapReduce Reducer class. In the reducer, the values for each key are summed up to calculate the total number of times the track was played or interacted with in the given way.
3. The Main method of the code is responsible for setting up the Hadoop job by configuring it with the necessary classes and input/output paths. The job is executed and the counters are retrieved to print out the number of invalid records encountered during processing.

1.4.8.3 Running process:

Step 1: Create a input folder, output folder and put input file

**Figure 1.29:** input of assignment 8**Step 2: Export file jar****Figure 1.30:** input of assignment 8

Step 3: Running jar

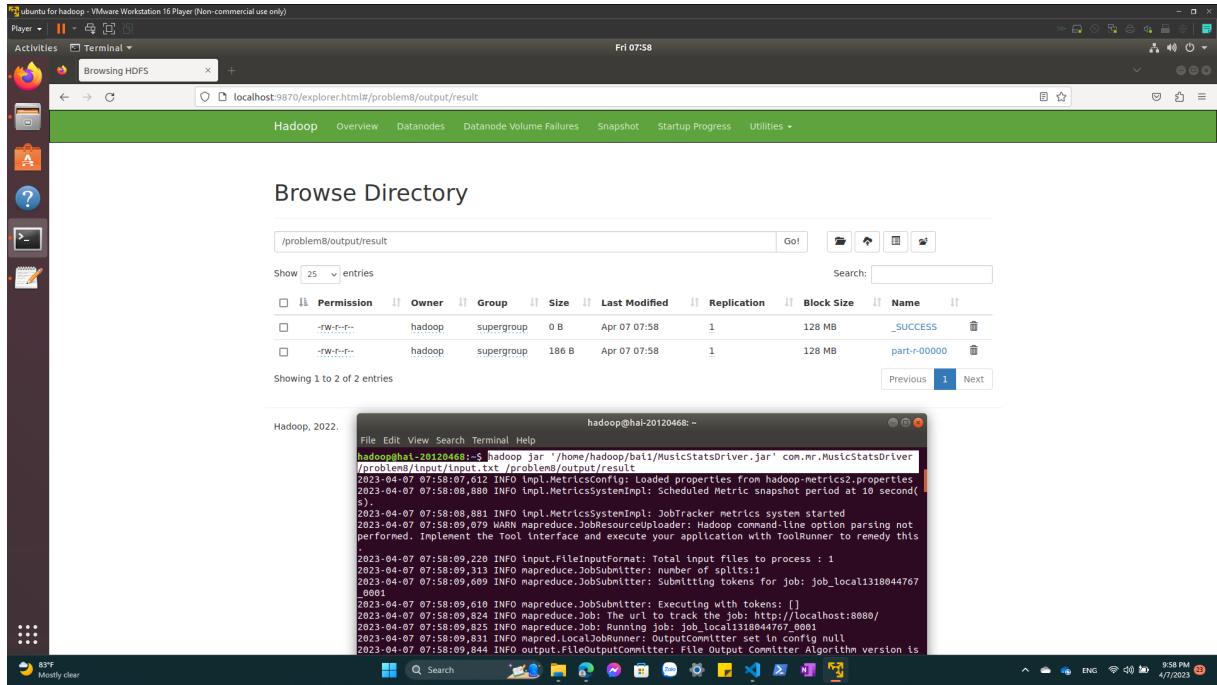
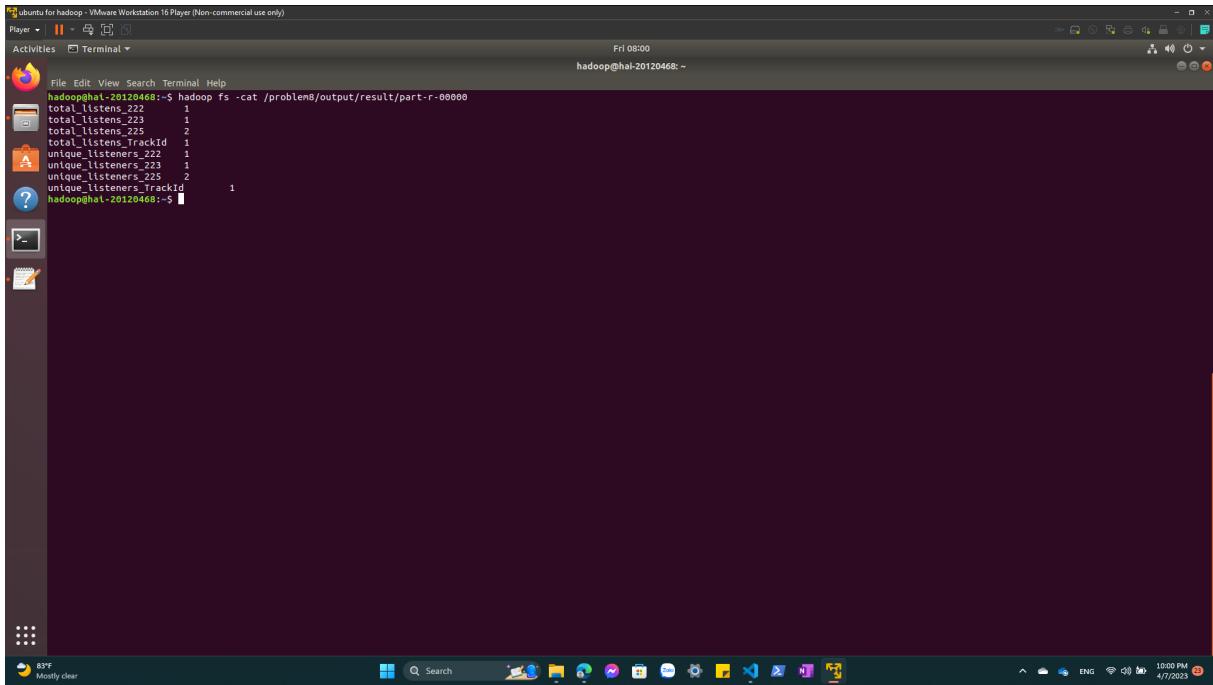


Figure 1.31: input of assignment 8

1.4.8.4 Result:



```
total_listens_222 1
total_listens_223 1
total_listens_22 2
total_listens_TrackId 1
unique_listeners_22 1
unique_listeners_223 1
unique_listeners_225 2
unique_listeners_TrackId 1
```

Figure 1.32: input of assignment 8

1.4.9 Problem 9

1.4.9.1 Solution idea:

Mapper: Extract these fields from input string: fromPhoneNumber, callStartTime, callEndTime, and stdFlag. If the value of stdFlag is equal to 1, then calculate the call duration by subtracting the callStartTime from the callEndTime.

Reducer: Group the calls by the phone number and calculate the total duration of each phone number to identify which phone number has total duration is greater than or equal to one hour.

1.4.9.2 Explain:

This is a MapReduce program written in Java that processes Call Data Records (CDR) and outputs the total duration of each phone number that has made a call lasting for more than one hour. The input data is assumed to be in a pipe-separated (|) format.

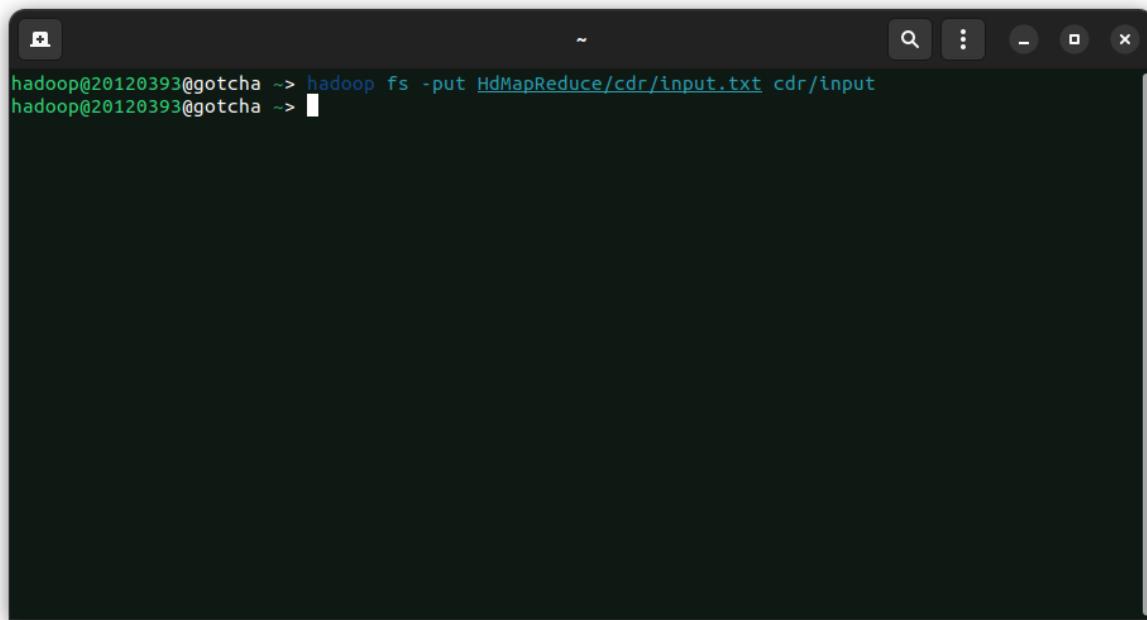
The program has two classes: CDRMapper, CDRReducer and main function. The CDRMapper class reads in each line of input data, splits it into fields, and extracts the fromPhoneNumber, call-

`StartTime`, `callEndTime`, and `stdFlag` fields. The `stdFlag` indicates whether the call is a long-distance (1) call or not. If the call is long-distance, the duration of the call is computed by subtracting the `callEndTime` from the `callStartTime`, dividing the result by 1000 to convert it from milliseconds to seconds, and emitting a key-value pair with the phone number as the key and the duration as the value.

The `CDRReducer` class receives the key-value pairs from the mapper, groups them by the phone number, and computes the total duration of each phone number. If the total duration of a phone number is greater than or equal to one hour (3600 seconds), it emits the phone number and the total duration as a key-value pair.

1.4.9.3 Running process:

Step 1: Put file input.txt into HDFS



A screenshot of a terminal window with a dark background. The window title bar shows a small icon followed by the text "hadoop@20120393@gotcha ~>". The main area of the terminal shows the command "hadoop fs -put HdMapReduce/cdr/input.txt cdr/input" being typed in. The command has been partially completed, with the first part "hadoop fs -put HdMapReduce/cdr/input.txt" visible. The cursor is positioned at the end of the command line, indicated by a small vertical bar.

Figure 1.33: Put input.txt into HDFS

Step 2: Export CDR.jar

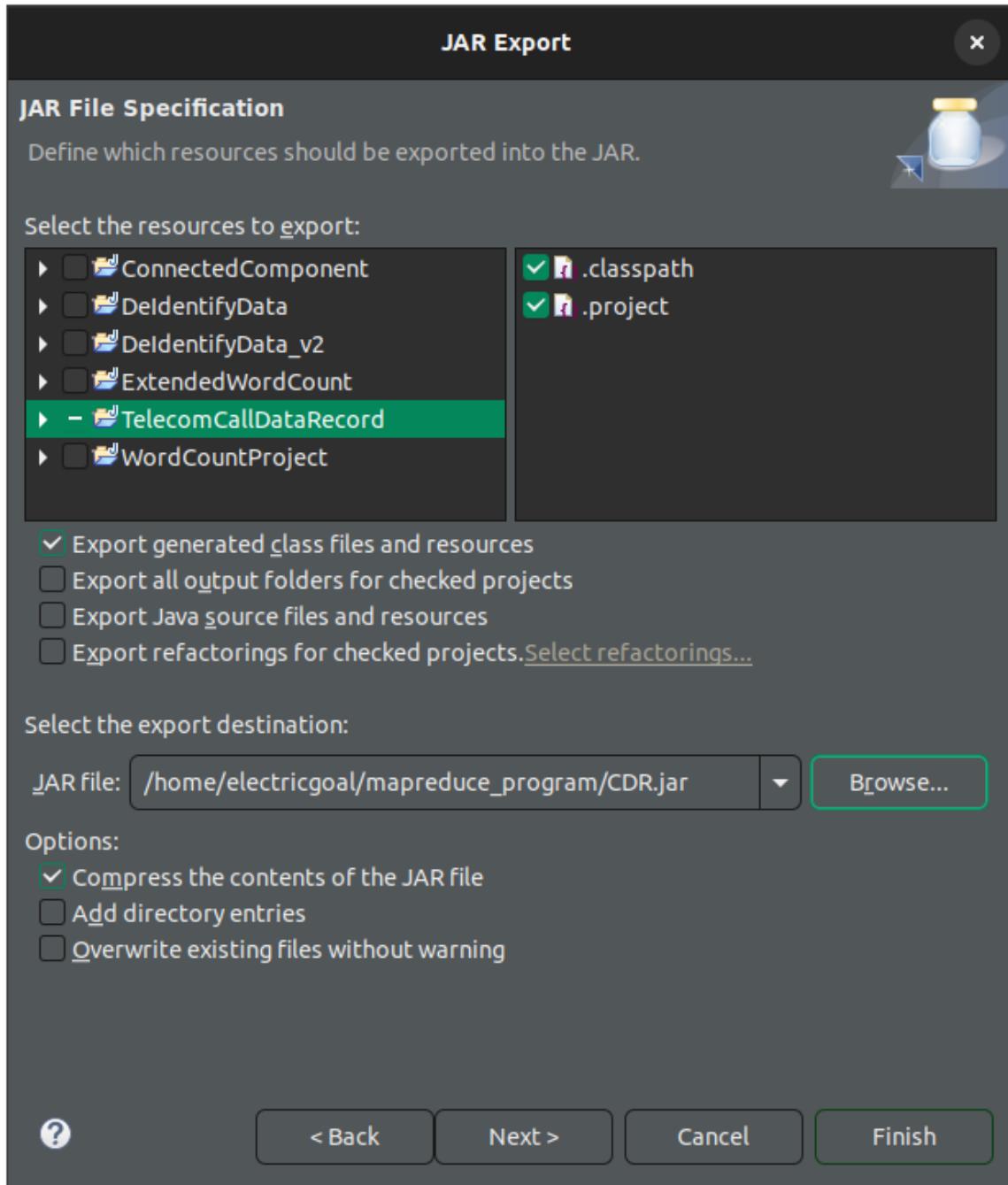
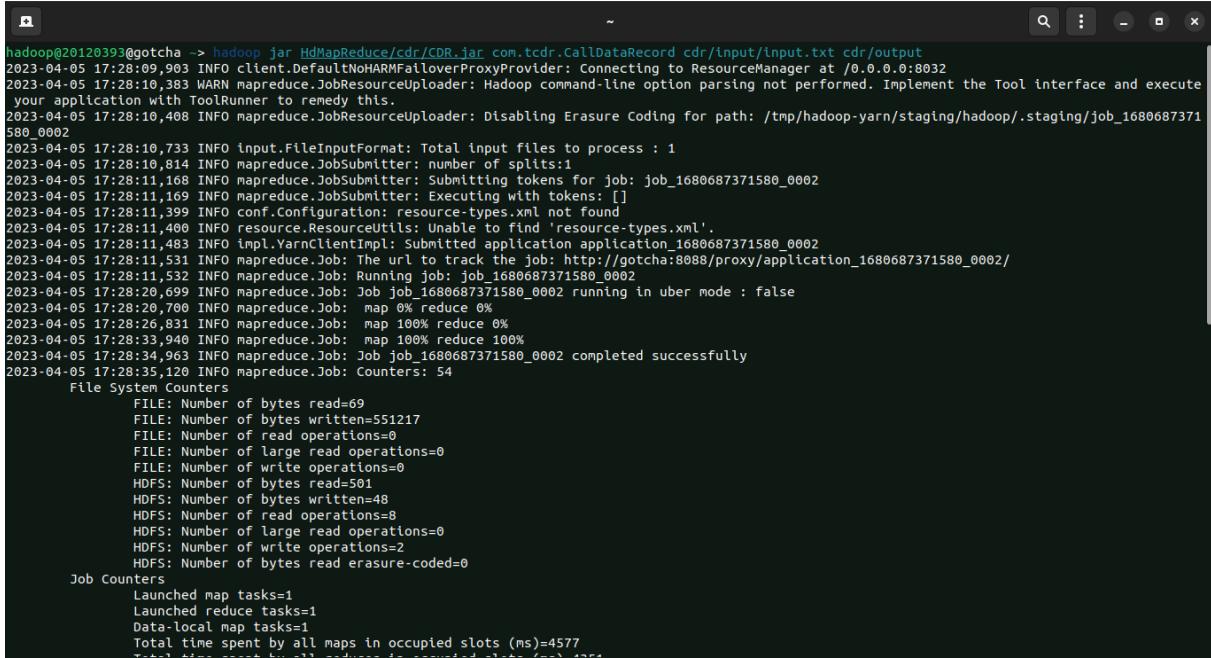


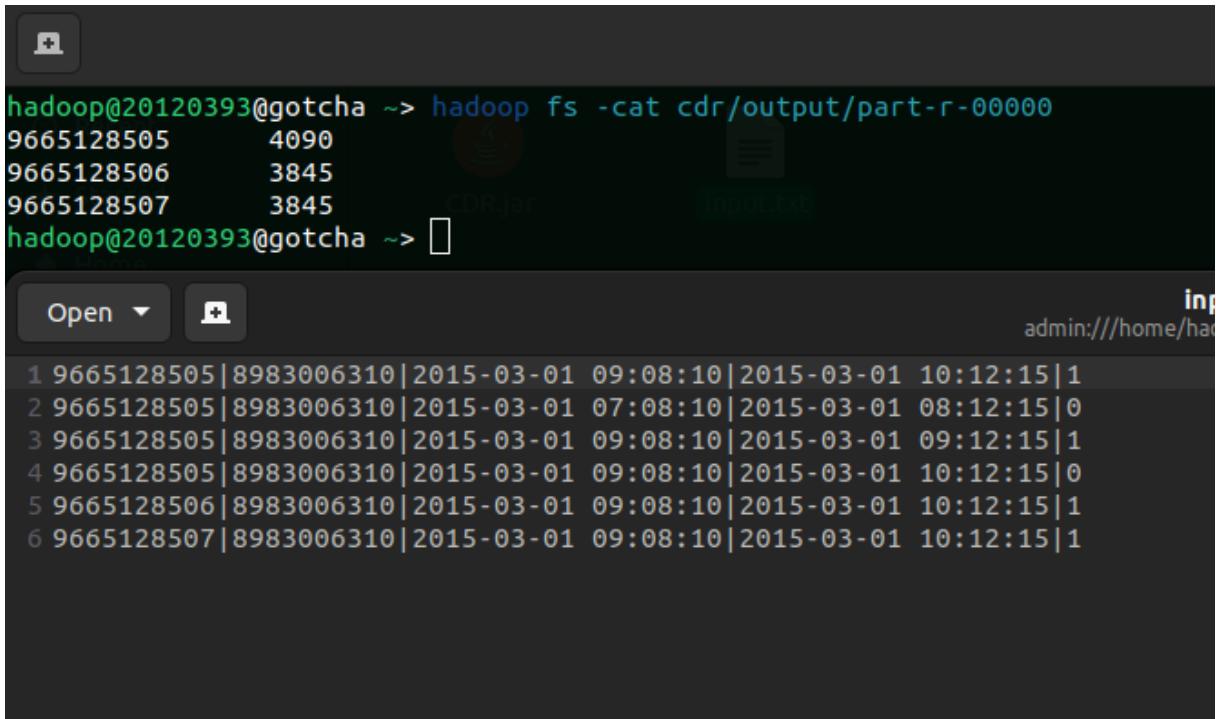
Figure 1.34: Export CDR.jar

Step 3: Run MapReduce program



```
hadoop@20120393@gotcha -> hadoop jar HdMapReduce/cdr/CDR.jar com.tcdr.CallDataRecord cdr/input/input.txt cdr/output
2023-04-05 17:28:09,903 INFO client.DefaultNoharmFallbackProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-04-05 17:28:10,383 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2023-04-05 17:28:10,408 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/hadoop/.staging/job_1680687371580_0002
2023-04-05 17:28:10,733 INFO input.FileInputFormat: Total input files to process : 1
2023-04-05 17:28:10,814 INFO mapreduce.JobSubmitter: number of splits:1
2023-04-05 17:28:11,168 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1680687371580_0002
2023-04-05 17:28:11,169 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-04-05 17:28:11,399 INFO conf.Configuration: resource-types.xml not found
2023-04-05 17:28:11,400 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-04-05 17:28:11,483 INFO impl.YarnClientImpl: Submitted application application_1680687371580_0002
2023-04-05 17:28:11,531 INFO mapreduce.Job: The url to track the job: http://gotcha:8088/proxy/application_1680687371580_0002/
2023-04-05 17:28:11,532 INFO mapreduce.Job: Job: Running job: job_1680687371580_0002
2023-04-05 17:28:20,699 INFO mapreduce.Job: Job job_1680687371580_0002 running in uber mode : false
2023-04-05 17:28:20,700 INFO mapreduce.Job: map 0% reduce 0%
2023-04-05 17:28:26,831 INFO mapreduce.Job: map 100% reduce 0%
2023-04-05 17:28:33,940 INFO mapreduce.Job: map 100% reduce 100%
2023-04-05 17:28:34,963 INFO mapreduce.Job: Job job_1680687371580_0002 completed successfully
2023-04-05 17:28:35,120 INFO mapreduce.Job: Counters: 54
  File System Counters
    FILE: Number of bytes read=69
    FILE: Number of bytes written=551217
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=501
    HDFS: Number of bytes written=48
    HDFS: Number of read operations=8
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
    HDFS: Number of bytes read erasure-coded=0
  Job Counters
    Launched map tasks=1
    Launched reduce tasks=1
    Data-local map tasks=1
    Total time spent by all maps in occupied slots (ms)=4577
```

Figure 1.35: Run MapReduce program

1.4.9.4 Result:

The screenshot shows a terminal window with the following content:

```
hadoop@20120393@gotcha ~> hadoop fs -cat cdr/output/part-r-00000
9665128505      4090
9665128506      3845
9665128507      3845
hadoop@20120393@gotcha ~>
```

Below the terminal, there is a file viewer window titled "CDRjar" showing the contents of "input.txt". The file contains the following data:

Index	Node ID	Neighbors	Timestamp	Timestamp	Timestamp	Timestamp
1	9665128505	8983006310	2015-03-01 09:08:10	2015-03-01 10:12:15	1	
2	9665128505	8983006310	2015-03-01 07:08:10	2015-03-01 08:12:15	0	
3	9665128505	8983006310	2015-03-01 09:08:10	2015-03-01 09:12:15	1	
4	9665128505	8983006310	2015-03-01 09:08:10	2015-03-01 10:12:15	0	
5	9665128506	8983006310	2015-03-01 09:08:10	2015-03-01 10:12:15	1	
6	9665128507	8983006310	2015-03-01 09:08:10	2015-03-01 10:12:15	1	

Figure 1.36: Output**1.4.10 Problem 10****1.4.10.1 Solution idea:**

Mapper: The input string should be converted to a map structure that defines the edges of the graph.

Reducer: The map structure should be used to save the key as the node of the graph, and the value should be a list of its neighbors. Then, the cleanup function can use that map structure to perform a DFS algorithm to count the connected components.

1.4.10.2 Explain:

This is a MapReduce program for finding the number of connected components in an undirected graph. The input to the program is a text file containing one line per node, with each line containing a node ID followed by the IDs of its neighbors, separated by whitespace.

The program consists of three classes: CCMapper, CCCombiner, and CCReducer, each of which extends the Mapper or Reducer class.

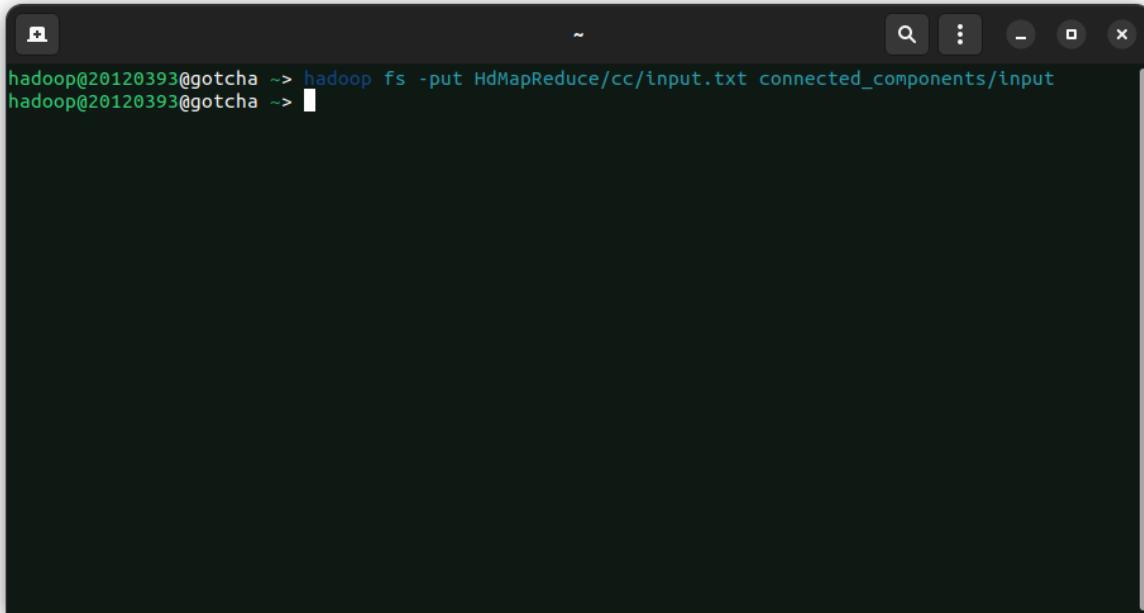
CCMapper is responsible for reading each line of the input file and emitting key-value pairs that represent the edges in the graph. The mapper extracts the node ID and its neighbor IDs from each line, and emits a pair for each edge in the graph. Specifically, for each node ID, the mapper emits a pair with the node ID as the key and the neighbor IDs as the value. Additionally, for each neighbor ID, the mapper emits a pair with the neighbor ID as the key and the node ID as the value.

CCCombiner is a local reducer that receives the output of the mapper and aggregates the values for each key by eliminating duplicates. The output of the combiner is a set of unique edges.

CCReducer is responsible for computing the connected components of the graph using depth-first search (DFS). The reducer builds a map of each node and its neighbors, and initializes a visited map to keep track of which nodes have been visited. It then iterates through the nodes and performs DFS starting from each unvisited node. The DFS function updates the visited map to mark the nodes that have been visited. After DFS has been performed on all unvisited nodes, the reducer emits a single pair with a key of 0 and a value of the number of connected components found.

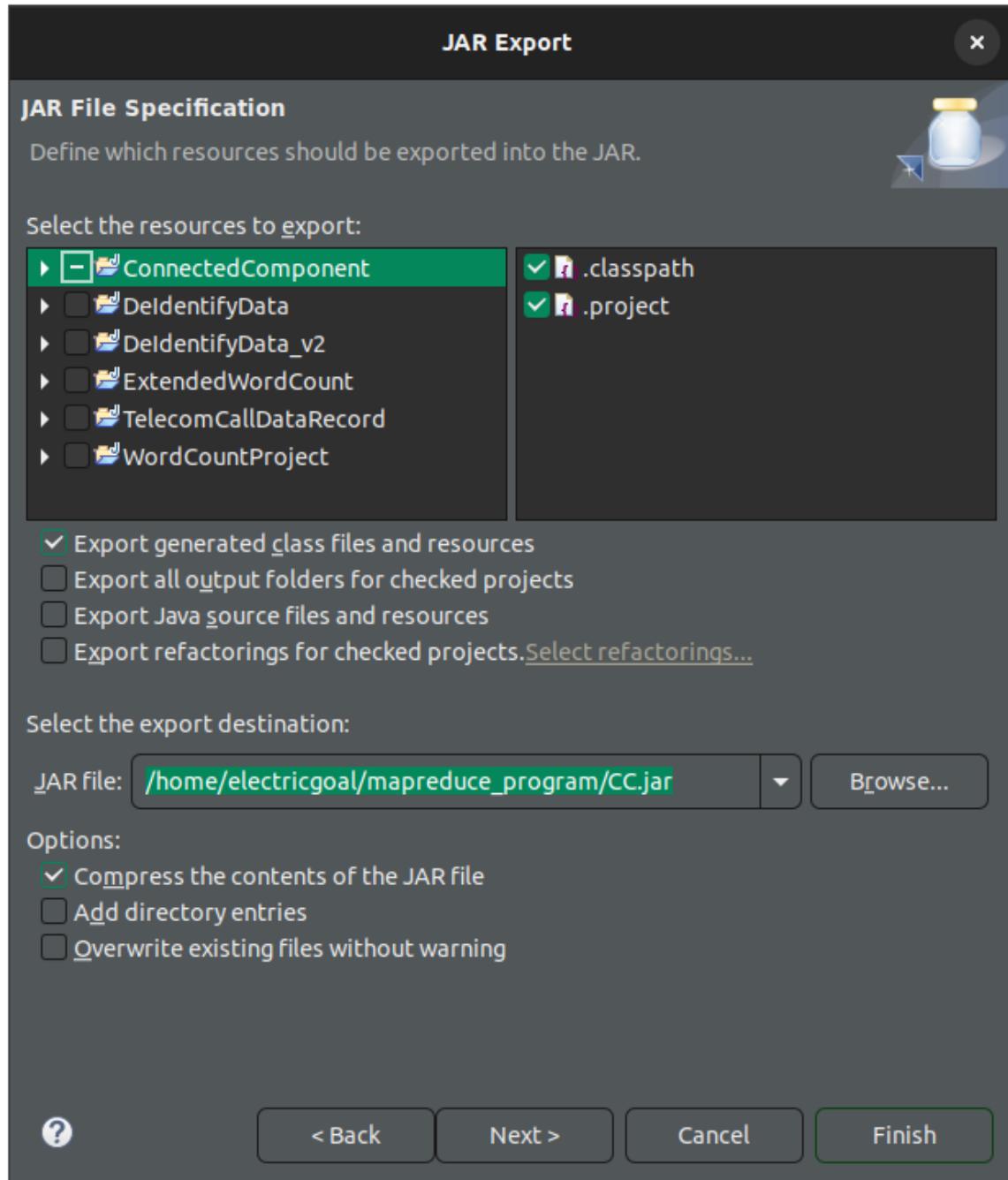
1.4.10.3 Running process:

Step 1: Put file input.txt into HDFS



```
hadoop@20120393@gotcha ~> hadoop fs -put HdMapReduce/cc/input.txt connected_components/input
hadoop@20120393@gotcha ~>
```

Figure 1.37: Put input.txt into HDFS

Step 2: Export CC.jar**Figure 1.38:** Export CC.jar**Step 3: Run MapReduce program**

```

hadoop@20120393@gotcha ~> hadoop jar HdMapReduce/cc/CC.jar com.cc.ConnectedComponent connected_components/input/input.txt connected_components/output
2023-04-05 17:37:35,094 INFO client.DefaultNoharmFallbackProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-04-05 17:37:35,577 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2023-04-05 17:37:35,606 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/hadoop/.staging/job_1680687371580_0003
2023-04-05 17:37:35,924 INFO input.FileInputFormat: Total input files to process : 1
2023-04-05 17:37:36,004 INFO mapreduce.JobSubmitter: number of splits:1
2023-04-05 17:37:36,341 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1680687371580_0003
2023-04-05 17:37:36,341 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-04-05 17:37:36,589 INFO conf.Configuration: resource-types.xml not found
2023-04-05 17:37:36,589 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-04-05 17:37:36,663 INFO impl.YarnClientImpl: Submitted application application_1680687371580_0003
2023-04-05 17:37:36,712 INFO mapreduce.Job: The url to track the job: http://gotcha:8088/proxy/application_1680687371580_0003/
2023-04-05 17:37:36,712 INFO mapreduce.Job: Job: job_1680687371580_0003 running in uber mode : false
2023-04-05 17:37:46,953 INFO mapreduce.Job: Job job_1680687371580_0003 running in uber mode : false
2023-04-05 17:37:46,955 INFO mapreduce.Job: map 0% reduce 0%
2023-04-05 17:37:55,108 INFO mapreduce.Job: map 100% reduce 0%
2023-04-05 17:38:01,174 INFO mapreduce.Job: map 100% reduce 100%
2023-04-05 17:38:01,190 INFO mapreduce.Job: Job job_1680687371580_0003 completed successfully
2023-04-05 17:38:01,318 INFO mapreduce.Job: Counters: 54
  File System Counters
    FILE: Number of bytes read=240
    FILE: Number of bytes written=551673
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=177
    HDFS: Number of bytes written=4
    HDFS: Number of read operations=8
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
    HDFS: Number of bytes read erasure-coded=0
  Job Counters
    Launched map tasks=1
    Launched reduce tasks=1
    Data-local map tasks=1
    Total time spent by all maps in occupied slots (ms)=5550
    Total time spent by all reduces in occupied slots (ms)=0
    Total time spent by all map tasks=5550
    Total time spent by all reduce tasks=0

```

Figure 1.39: Run MapReduce program

1.4.10.4 Result:

```

hadoop@20120393@gotcha ~> hadoop fs -cat connected_components/output/part-r-00000
0 4
hadoop@20120393@gotcha ~>

```

The terminal window also shows a file editor with the file ***input1.txt** containing the input data:

```

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

```

Figure 1.40: Output

1.5 References

- Example: WordCount v1.0: https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example:_WordCount_v1.0
- Reducer (Apache Hadoop Main 2.7.5 API): <https://hadoop.apache.org/docs/r2.7.5/api/org/apache/hadoop/mapreduce/org/apache/hadoop/mapreduce.Reducer.Context>
- Sriram Balasubramanian, Hadoop-MapReduce Lab, 2016