

LAB 02 – Map Reduce

1. Thông tin nhóm

Tên sinh viên	Mã số sinh viên
Trần Đại Chí	18127070
Phan Tấn Đạt	18127078
Trần Minh Quang	18127192
Phan Quang Đại	18127073

2. Trình bày kết quả

1. Assignment 1 -Wordcount Program

```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    private long numRecords = 0;
    private static final Pattern WORD_BOUNDARY = Pattern.compile("\\s*\\b\\s*");

    public void map(LongWritable offset, Text lineText, Context context)
        throws IOException, InterruptedException {
        String line = lineText.toString();
        Text currentWord = new Text();
        for (String word : WORD_BOUNDARY.split(line)) {
            if (word.isEmpty()) {
                continue;
            }
            currentWord = new Text(word);
            context.write(currentWord, one);
        }
    }
}
```

Tạo lớp Map mở rộng lớp Mapper, sau đó định dạng hàm map():

- Đầu tiên chuyển input dạng Text sang dạng String (biến *line*)
- Sau đó thực hiện tách chuỗi (hàm split) theo Pattern tách được định nghĩa ở biến WORD_BOUNDARY từ chuỗi chứa trong biến *line* thành các chuỗi con. Sau đó thực hiện vòng lặp theo từng chuỗi con đó. Với mỗi vòng lặp, nếu chuỗi không là chuỗi trống (isEmpty) thì ghi vào chuỗi đó thành dạng Text vào *context* với giá trị “one” bằng context.write()

```

public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
    @Override
    public void reduce(Text word, Iterable<IntWritable> counts, Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable count : counts) {
            sum += count.get();
        }
        context.write(word, new IntWritable(sum));
    }
}

```

Ở override của hàm reduce():

- Đầu tiên tạo biến `sum = 0`
- Sau đó tạo vòng lặp qua tất cả các giá trị liên quan đến *word* và đếm tổng số lần xuất hiện của nó
- Cuối cùng, chúng ta sẽ sử dụng `context.write` để viết kết quả Text *word* và số lần xuất hiện của nó (sum)

```

public class WordCount extends Configured implements Tool {

    private static final Logger LOG = Logger.getLogger(WordCount.class);

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new WordCount(), args);
        System.exit(res);
    }

    public int run(String[] args) throws Exception {
        Job job = Job.getInstance(getConf(), jobName: "wordcount");
        job.setJarByClass(this.getClass());
        // Use TextInputFormat, the default unless job.setInputFormatClass is used
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        return job.waitForCompletion(verbose: true) ? 0 : 1;
    }
}

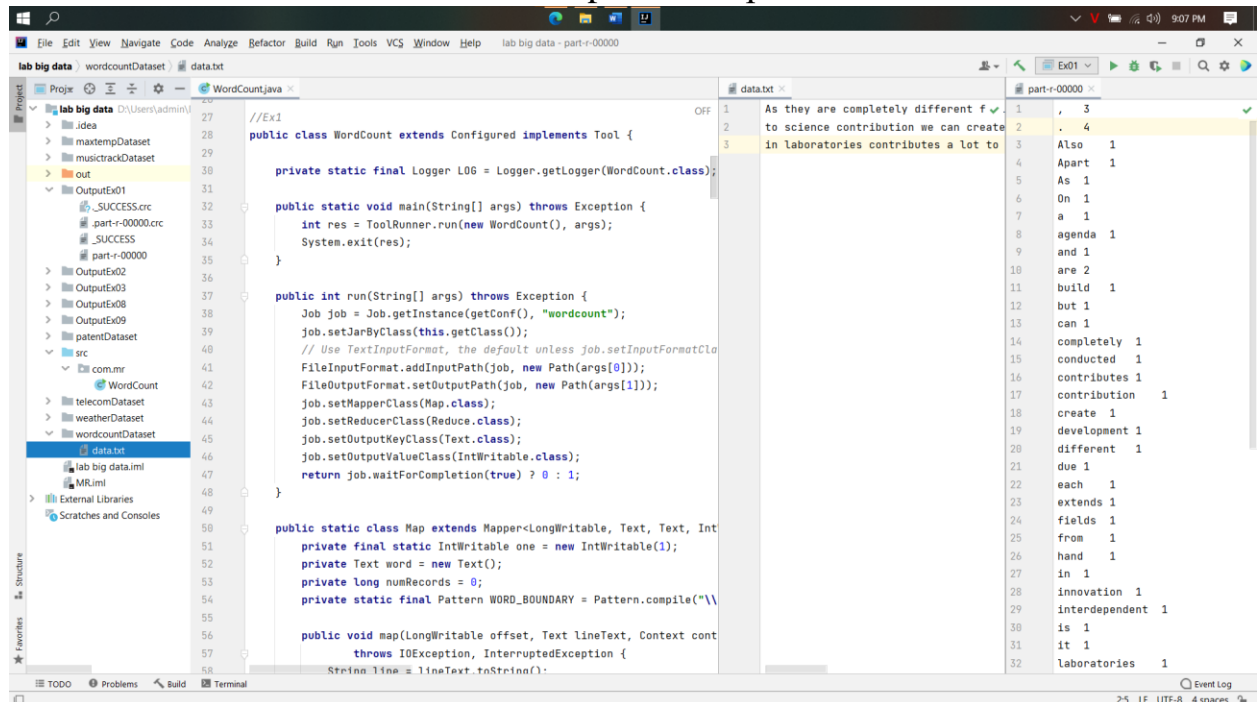
```

Ở hàm `main()` sẽ nhận `args` là một List chứa các String là vị trí của Input và Output được nhập vào để đọc dữ liệu và xuất kết quả như ý muốn

- Đầu tiên chúng ta sẽ tạo job với tên là “wordcount”

- Chúng ta sẽ để lần lượt tên hàm Map và Reduce của chúng ta tương ứng vào trong `setMapperClass` và `setReducerClass`
- Do ở pha Map, chúng ta dùng `context.write` để viết kết quả là dạng text cho word và số lần xuất hiện, nên `setMapOutputKeyClass` và `setMapOutputValueClass` của chúng ta là một `Text.class`

Hình dưới là kết quả có được sau khi chạy chương trình lấy args gồm input từ folder “wordcountDataset” và xuất output ở “OutputEx01”



2. Assignment 2 - WordSizeWordCount Program

Ở hàm `map()`:

- Đầu tiên chuyển input dạng `Text` thành dạng `String` và chứa ở biến `line`
- Sau đó dùng `StringTokenizer` để parse chuỗi `line` thành các chuỗi con
- Tạo vòng lặp theo từng chuỗi con đó. Trong đó biến `token` để lưu trữ chuỗi con ở mỗi vòng lặp. Ta có thể lấy được độ dài chuỗi con bằng `token.length()`

- Sau đó dùng `context.write()` để ghi lại độ dài chuỗi đó (ở dạng `IntWritable`) và chuỗi con đó (dạng `Text`)

```
public static class Map extends Mapper<LongWritable, Text, IntWritable, Text> {
    //https://stackoverflow.com/questions/26556972/mapreduce-find-word-length-frequency
    private IntWritable wordLength = new IntWritable();
    private Text word = new Text();

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            String token = tokenizer.nextToken();
            wordLength = new IntWritable(token.length());
            word.set(token);
            context.write(wordLength, word);
        }
    }
}
```

Ở hàm `reduce()`:

- Tạo biến đếm $cnt = 0$
- Tạo vòng lặp đi qua các giá trị tồn tại trong *values* có cùng độ dài *key* để cộng dồn biến *cnt* để đếm số lượng các chuỗi có độ dài *key*
- Sau đó dùng `context.write()` để lưu độ dài *key* (dạng `IntWritable`) và số lượng các chuỗi có độ dài *key* (dạng `IntWritable`)

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

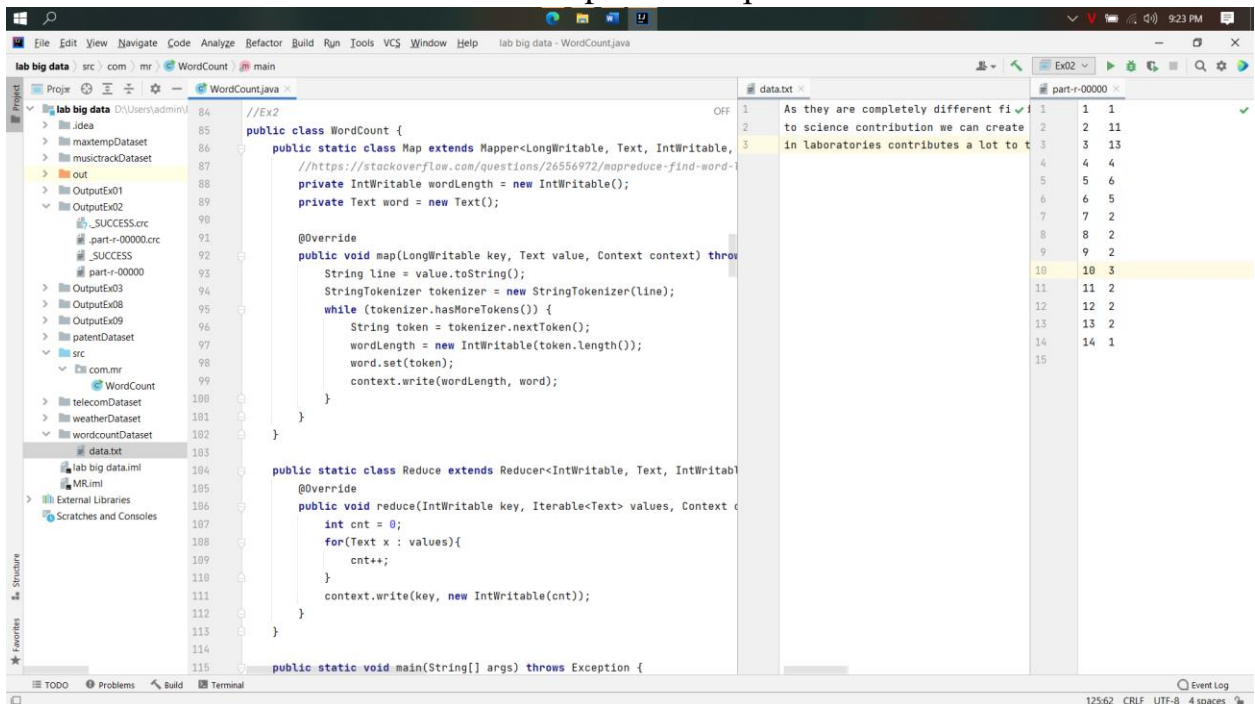
Hàm `main()` sẽ nhận `args` là một List chứa các String là vị trí của Input và Output được nhập vào để đọc dữ liệu và xuất kết quả như ý muốn

- Đầu tiên chúng ta sẽ tạo job với tên là “wordsize”, sau đó truyền tên lớp (`WordCount.class`) cho hàm `setJarByClass()` của job vừa tạo
- Chúng ta sẽ để lần lượt tên hàm Map và Reduce của chúng ta tương ứng vào trong `setMapperClass` và `setReducerClass`
- Do ở pha Map, chúng ta dùng `context.write` để viết kết quả là dạng `IntWritable` và `Text` cho word và số lần xuất hiện, nên

setMapOutputKeyClass sẽ là dạng IntWritable.class và setMapOutputValueClass của chúng ta là một Text.class

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = new Job(conf, "wordsize");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
    job.setMapOutputKeyClass(IntWritable.class);
    job.setMapOutputValueClass(Text.class);
    job.setOutputKeyClass(IntWritable.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);
}
```

Hình dưới là kết quả có được sau khi chạy chương trình lấy args gồm input từ folder “wordcountDataset” và xuất output ở “OutputEx02”



3. Assignment 3 - WeatherData Program

Dữ liệu thời tiết nhập vào sẽ có dạng:

weather_data.txt - Notepad																													
File	Edit	Format	View	Help																									
23/07	2015/01	2.423	-98.08	30.62	2.2	-0.6	0.8	0.9	6.2	1.47	3.7	1.1	2.5	99.9	85.4	97.2	0.369	0.388	-99.000	-99.000	-99.000	-99.000	7.0	8.1	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	3.5	1.3	2.4	2.2	9.0	1.43	4.9	2.3	3.1	100.0	98.8	99.8	0.391	0.327	-99.000	-99.000	-99.000	-99.000	7.1	7.9	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	15.9	2.3	9.1	7.5	2.9	11.00	16.4	2.9	7.3	100.0	34.8	73.7	0.450	0.397	-99.000	-99.000	-99.000	-99.000	7.6	7.9	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	9.2	-1.3	3.9	4.2	0.0	13.24	12.4	-0.5	4.9	82.0	40.6	61.7	0.414	0.352	-99.000	-99.000	-99.000	-99.000	7.3	7.9	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	10.9	-3.7	3.6	2.6	0.0	13.37	14.7	-3.0	3.8	77.9	33.3	57.4	0.399	0.340	-99.000	-99.000	-99.000	-99.000	6.3	7.0	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	20.2	2.9	11.6	10.9	0.0	12.90	22.0	1.6	9.9	67.7	30.2	49.3	0.395	0.335	-99.000	-99.000	-99.000	-99.000	8.0	8.0	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	10.9	-3.4	3.8	4.5	0.0	12.68	12.4	-2.1	5.5	82.7	36.5	55.7	0.387	0.328	-99.000	-99.000	-99.000	-99.000	7.6	8.3	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	0.6	-7.9	-3.6	-3.3	0.0	4.98	3.9	-4.8	-0.5	57.7	37.6	48.1	0.372	0.316	-99.000	-99.000	-99.000	-99.000	4.7	6.1	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	2.0	0.1	1.0	0.8	0.0	2.52	4.1	1.2	2.5	87.8	48.9	64.4	0.368	0.312	-99.000	-99.000	-99.000	-99.000	5.4	6.2	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	0.5	-2.0	-0.8	-0.6	3.3	2.11	2.5	-0.1	1.4	99.9	47.7	85.8	0.373	0.314	-99.000	-99.000	-99.000	-99.000	5.1	6.0	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	10.9	0.0	5.4	4.4	2.9	6.38	12.7	1.3	5.8	100.0	77.8	97.1	0.420	0.362	-99.000	-99.000	-99.000	-99.000	6.5	6.7	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	6.5	1.4	4.0	4.3	0.0	1.55	6.9	2.7	5.1	100.0	89.4	97.8	0.412	0.350	-99.000	-99.000	-99.000	-99.000	7.3	7.5	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	3.0	-0.7	1.1	1.2	0.0	3.26	5.6	0.7	2.9	99.7	80.7	90.7	0.401	0.337	-99.000	-99.000	-99.000	-99.000	6.1	6.8	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	2.9	0.9	1.9	1.8	0.0	1.88	4.7	2.0	3.1	99.6	90.8	97.9	0.395	0.331	-99.000	-99.000	-99.000	-99.000	6.1	6.7	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	13.2	1.2	7.2	6.4	0.0	13.37	16.4	1.4	6.7	98.9	46.7	73.4	0.395	0.333	-99.000	-99.000	-99.000	-99.000	6.7	7.0	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	16.7	3.5	10.1	9.9	0.0	13.68	19.2	1.3	8.7	80.2	38.1	58.2	0.391	0.330	-99.000	-99.000	-99.000	-99.000	7.3	7.4	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	19.5	5.0	12.2	12.3	0.0	10.96	20.9	3.3	10.6	87.7	30.4	55.7	0.388	0.327	-99.000	-99.000	-99.000	-99.000	8.7	8.4	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	20.9	7.6	14.3	13.7	0.0	15.03	23.4	3.5	11.9	45.9	14.6	31.4	0.383	0.325	-99.000	-99.000	-99.000	-99.000	9.5	9.2	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	23.9	6.7	15.3	14.3	0.0	14.10	25.6	3.8	12.6	65.3	26.8	45.6	0.376	0.321	-99.000	-99.000	-99.000	-99.000	9.9	9.5	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	26.0	9.5	17.8	15.9	0.0	14.57	27.9	6.5	14.5	88.4	16.1	50.2	0.373	0.320	-99.000	-99.000	-99.000	-99.000	10.9	10.4	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	11.0	6.9	8.9	8.9	1.7	2.71	13.1	6.8	9.7	99.2	68.0	88.1	0.369	0.317	-99.000	-99.000	-99.000	-99.000	10.7	10.6	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	8.6	3.5	6.1	5.6	39.6	1.28	9.1	4.1	6.3	99.6	95.2	98.0	0.546	0.418	-99.000	-99.000	-99.000	-99.000	9.0	9.3	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	9.4	2.2	5.8	4.2	7.4	6.58	11.1	2.0	4.8	98.4	58.8	86.5	0.554	0.409	-99.000	-99.000	-99.000	-99.000	7.6	8.1	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	16.0	1.4	8.7	8.0	0.0	14.26	18.8	0.4	7.7	92.0	33.0	63.0	0.494	0.381	-99.000	-99.000	-99.000	-99.000	7.7	7.9	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	20.2	6.4	13.3	12.7	0.0	14.99	22.0	4.4	11.0	69.2	18.9	43.8	0.456	0.357	-99.000	-99.000	-99.000	-99.000	9.1	8.9	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	21.5	7.2	14.4	14.1	0.0	12.01	22.9	5.5	12.2	56.8	23.7	40.6	0.433	0.349	-99.000	-99.000	-99.000	-99.000	10.0	9.7	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	26.5	10.7	18.6	17.5	0.0	15.18	28.9	8.1	15.5	52.2	21.4	38.8	0.420	0.344	-99.000	-99.000	-99.000	-99.000	11.4	10.8	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	26.3	13.3	19.8	19.1	0.0	15.11	28.1	7.9	16.3	54.9	19.4	35.5	0.410	0.339	-99.000	-99.000	-99.000	-99.000	12.1	11.5	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	23.1	9.8	16.5	16.4	0.0	13.74	27.4	9.7	16.4	87.0	34.2	55.6	0.402	0.334	-99.000	-99.000	-99.000	-99.000	13.1	12.4	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	13.0	6.9	10.0	9.0	0.0	7.19	19.2	8.3	11.0	67.6	48.4	58.8	0.389	0.328	-99.000	-99.000	-99.000	-99.000	11.6	11.8	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	15.1	7.4	11.3	10.2	8.5	1.18	14.5	8.4	10.7	100.0	63.1	90.3	0.401	0.337	-99.000	-99.000	-99.000	-99.000	11.2	11.3	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	18.3	3.9	11.1	13.3	0.0	8.69	22.1	4.1	13.8	98.8	53.6	79.1	0.450	0.386	-99.000	-99.000	-99.000	-99.000	12.9	12.6	-9999.0	-9999.0	-9999.0		
23/07	2015/01	2.423	-98.08	30.62	8.0	-1.9	3.1	3.3	0.0	12.48	15.2	-0.6	5.8	69.4	34.8	54.2	0.428	0.354	-99.000	-99.000	-99.000	-99.000	9.3	10.1	-9999.0	-9999.0	-9999.0		

Đầu tiên tạo class MaxTemperatureMapper là lớp Map sẽ sử dụng :

Trong đó override hàm map():

- Đầu tiên chuyển input dạng Text thành dạng String và chứa ở mỗi biến *line* là chuỗi chứa mỗi dòng trong dữ liệu
- Dùng hàm *line.substring(a,b)* để lấy chuỗi con trong *line* trong khoảng a - b
- Dữ liệu ngày tháng trong *line* sẽ ở ký tự thứ 6 đến 14
- Dữ liệu *temp_Max* sẽ được lưu ở dạng Float và nó được lấy từ ký tự thứ 39 đến 45
- Dữ liệu *temp_Min* sẽ được lưu ở dạng Float và nó được lấy từ ký tự thứ 47 đến 53
- Hàm trim() để loại bỏ dấu cách trong các chuỗi con lấy được từ *line*
- Nếu *temp_Max* > 40.0 thì dùng context.write để ghi ngày + “Hot Day” (dạng Text) và giá trị *temp_Max* (dạng Text)
- Nếu *temp_Min* < 10.0 thì dùng context.write để ghi ngày + “Cold Day” (dạng Text) và giá trị *temp_Min* (dạng Text)


```

public static class MaxTemperatureMapper extends Mapper<LongWritable, Text, Text, Text> {
    @Override
    public void map(LongWritable arg0, Text Value, Context context) throws IOException, InterruptedException {
        String line = Value.toString();
        try{
            if(!(line.length() == 0)){
                String date = line.substring(6, 14); //14, 22
                float temp_Max = Float.parseFloat(line.substring(39, 45).trim()); //104, 108
                float temp_Min = Float.parseFloat(line.substring(47, 53).trim()); //112, 116
                if (temp_Max > 40.0) {
                    context.write(new Text(date + "    Hot Day"), new Text(String.valueOf(temp_Max)));
                }
                if (temp_Min < 10.0) {
                    context.write(new Text(date + "    Cold Day"), new Text(String.valueOf(temp_Min)));
                }
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
}

```

Sau đó tạo lớp MaxTemperatureReducer là lớp Reduce sẽ dùng, trong đó override hàm reduce:

- Tạo vòng lặp qua các giá trị mang nội dung Key và lưu giá trị nhiệt độ dạng String
- Dùng context.write để ghi ra Key (gồm ngày và nhận định ngày đó nóng hay lạnh ở dạng Text) và nhiệt độ của nó (dạng Text)

```

public static class MaxTemperatureReducer extends Reducer<Text, Text, Text, Text> {
    @Override
    protected void reduce(Text Key, Iterable<Text> Values, Context context) throws IOException, InterruptedException {
        String temperature = "";
        for(Text val: Values){
            temperature = val.toString();
        }
        context.write(Key, new Text("    " + temperature));
    }
}

```

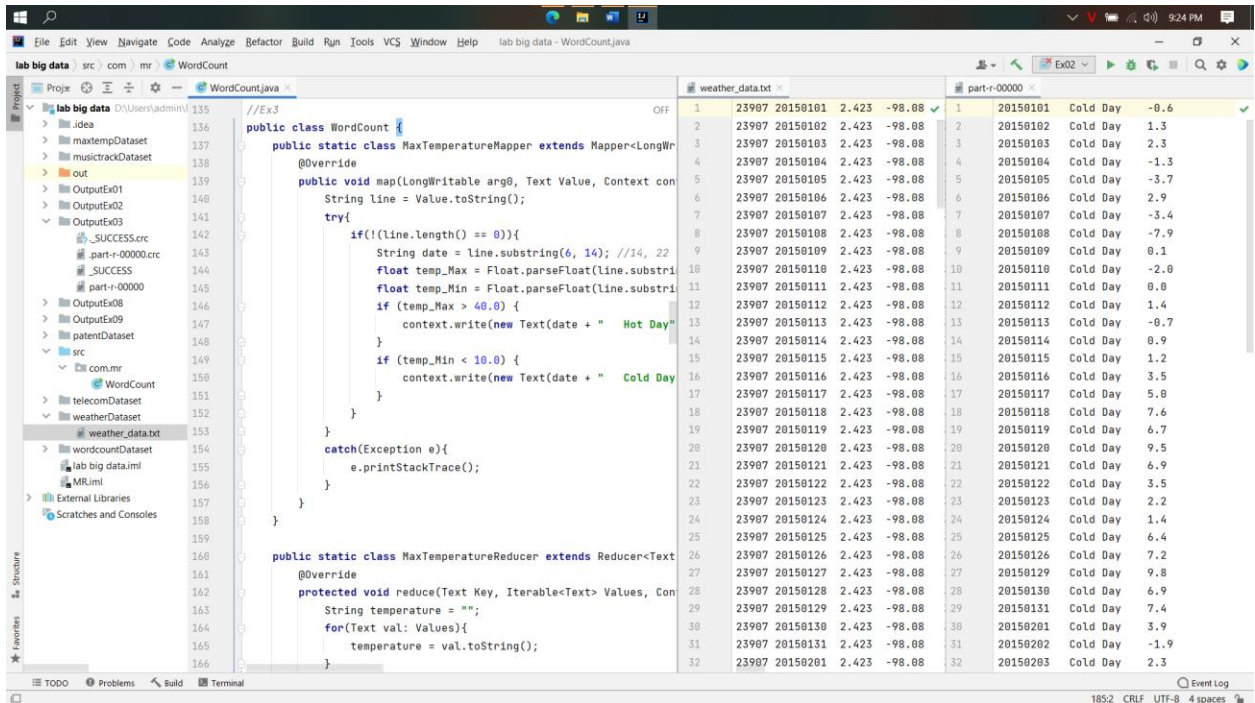
Hàm main() sẽ nhận args là một List chứa các String là vị trí của Input và Output được nhập vào để đọc dữ liệu và xuất kết quả như ý muốn

- Đầu tiên chúng ta sẽ tạo job với tên là “weather temperature”, sau đó truyền tên lớp (WordCount.class) cho hàm setJarByClass() của job vừa tạo
- Chúng ta sẽ chỉnh lần lượt tên lớp Map (MaxTemperatureMapper .class) và Reduce (MaxTemperatureReducer.class) của chúng ta tương ứng vào cho setMapperClass và setReducerClass

- Do ở pha Map, chúng ta dùng context.write để viết kết quả là ngày và nhận định ngày đó nóng hay lạnh (dạng Text) và nhiệt độ của nó (dạng Text) , nên setMapOutputKeyClass và setMapOutputValueClass của chúng ta là một Text.class

```
public static void main(String[] args) throws Exception {  
    Configuration conf = new Configuration();  
    Job job = new Job(conf, "weather temperature");  
    job.setJarByClass(WordCount.class);  
    job.setMapOutputKeyClass(Text.class);  
    job.setMapOutputValueClass(Text.class);  
    job.setMapperClass(MaxTemperatureMapper.class);  
    job.setReducerClass(MaxTemperatureReducer.class);  
    job.setInputFormatClass(TextInputFormat.class);  
    job.setOutputFormatClass(TextOutputFormat.class);  
    FileInputFormat.addInputPath(job, new Path(args[0]));  
    FileOutputFormat.setOutputPath(job, new Path(args[1]));  
    System.exit(job.waitForCompletion(true) ? 0 : 1);  
}
```

Hình dưới là kết quả có được sau khi chạy chương trình lấy args gồm input từ folder “weatherDataset” và xuất output ở “OutputEx03”



4. Assignment 4 – Patent Program

```
public static class Map extends Mapper<LongWritable, Text, Text, Text> {
    private Text word1 = new Text();
    private Text word2 = new Text();

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        StringTokenizer tokenizer = new StringTokenizer(value.toString());
        while (tokenizer.hasMoreTokens()) {
            word1.set(tokenizer.nextToken());
            word2.set(tokenizer.nextToken());
            context.write(word1, word2);
        }
    }
}
```

- Ở pha Map, đầu tiên chúng ta sẽ chuyển các dòng input dạng text sang string
- Tiếp theo, chúng ta sẽ sử dụng tokenizer để chia dòng thành các từ
- Sau đó, chúng ta sẽ lặp qua các từ và cặp giá trị của mỗi từ đó
- Vì ở mỗi dòng, mỗi patent sẽ có 1 sub-patent nên chúng ta sẽ gán từng công việc từ tokenizer (dạng string) tới word1 và word2 ở dạng text
- Cuối cùng, chúng ta sẽ sử dụng context.write để viết kết quả

```

public static class Reduce extends Reducer<Text, Text, Text, IntWritable> {
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        int cnt = 0;
        while(values.iterator().hasNext()){
            values.iterator().next();
            cnt++;
        }
        context.write(key, new IntWritable(cnt));
    }
}

```

- Ở pha Reduce, đầu tiên chúng ta sẽ tạo một biến count với giá trị bằng 0
- Tiếp theo, chúng ta sẽ lặp qua tất cả các giá trị liên quan đến khóa và tổng hợp tất cả chúng lại
- Cuối cùng, chúng ta sẽ sử dụng context.write để viết kết quả patent và số lượng của sub-patent liên quan

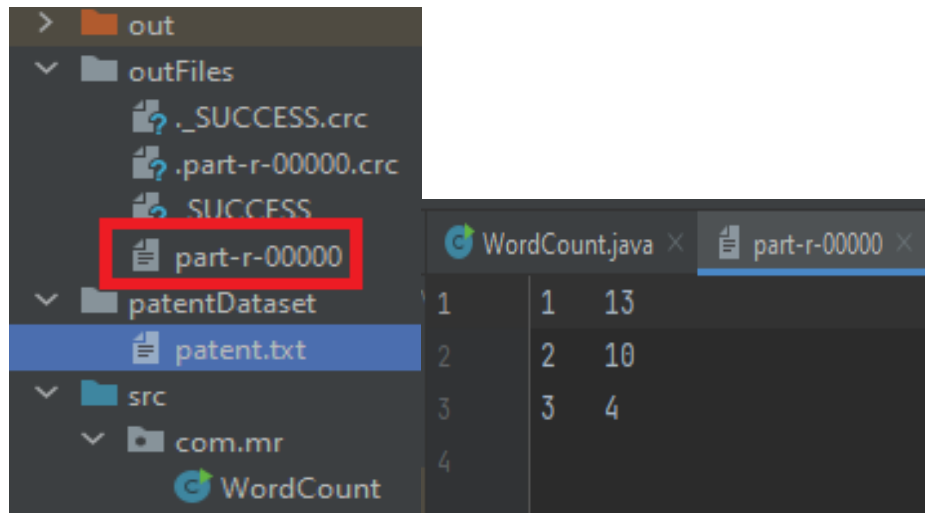
```

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = new Job(conf, "patent count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.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(verbose: true) ? 0 : 1);
}

```

- Tiếp theo, trong hàm Main chúng ta sẽ tạo job với tên là patent count và setJarByClass sẽ là tên lớp của chúng ta
- Chúng ta sẽ để lần lượt tên hàm Map và Reduce của chúng ta tương ứng vào trong setMapperClass và setReducerClass

- Do ở pha Map, chúng ta dùng context.write để viết kết quả là dạng text cho word1 và word2, nên setMapOutputKeyClass và setMapOutputValueClass của chúng ta là một Text.class
- Cuối cùng, chúng ta sẽ cần ghi ra patent là dạng text và biến count tương ứng với số lượng sub-patent liên quan ở dạng IntWritable trong hàm setOutputKeyClass và setOutputValueClass và sau đó ghi toàn bộ kết quả vào folder output



Và hình trên đây là kết quả sau khi chúng ta chạy chương trình thành công

5. Assignment 5 – MaxTemp Program

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

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        StringTokenizer tokenizer = new StringTokenizer(value.toString());
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            cnt = Integer.parseInt(tokenizer.nextToken());
            context.write(word, new IntWritable(cnt));
        }
    }
}
```

- Ở pha Map, đầu tiên chúng ta sẽ chuyển các dòng input dạng text sang dạng string
- Tiếp theo, chúng ta sẽ sử dụng tokenizer để chia dòng thành các từ

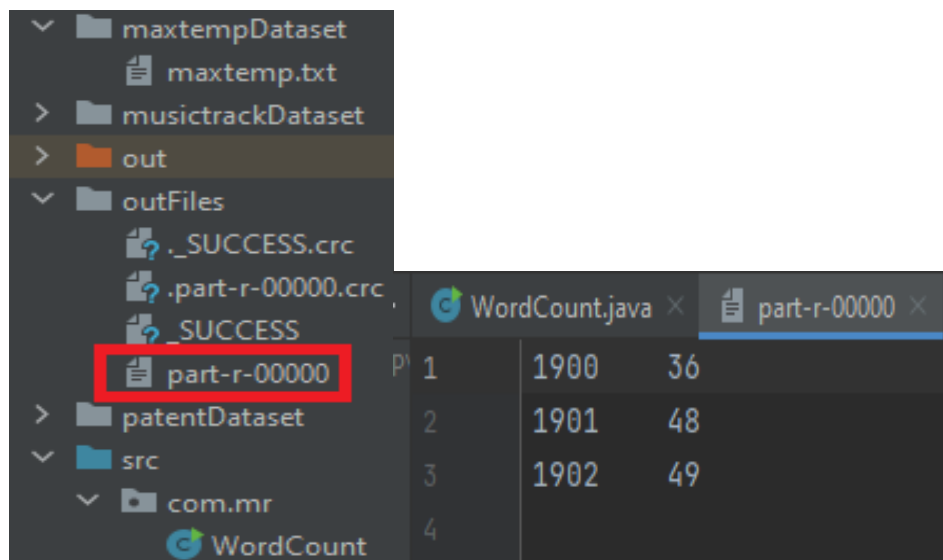
- Sau đó, chúng ta sẽ lặp qua các từ và cặp giá trị của mỗi từ đó
- Vì ở mỗi dòng chúng ta sẽ có năm nên chúng ta sẽ gán công việc từ tokenizer (dạng string) tới word dạng text và vì chúng ta muốn lấy ra nhiệt độ thuộc năm đó cao nhất nên chúng ta sẽ không dùng text cho nhiệt độ mà chúng ta sẽ chuyển nó từ string sang int để tiện cho việc so sánh các số
- Cuối cùng, chúng ta sẽ sử dụng context.write để viết kết quả

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

- Ở pha Reduce, đầu tiên chúng ta sẽ tạo một biến max với giá trị bằng 0
- Tiếp theo, chúng ta sẽ lặp qua toàn bộ giá trị liên quan đến khóa và lấy ra giá trị lớn nhất để gán cho biến max
- Cuối cùng, chúng ta sẽ sử dụng context.write để viết kết quả năm và nhiệt độ cao nhất trong năm đó

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

- Tiếp theo, trong hàm Main chúng ta sẽ tạo job với tên là max temperature và setJarByClass sẽ là tên lớp của chúng ta
- Chúng ta sẽ để lần lượt tên hàm Map và Reduce của chúng ta tương ứng vào trong setMapperClass và setReducerClass
- Do ở pha Map, chúng ta dùng context.write để viết kết quả là dạng text cho năm và dạng int cho nhiệt độ, nên setMapOutputKeyClass và setMapOutputValueClass của chúng ta lần lượt sẽ là Text.class và IntWritable.class
- Cuối cùng, chúng ta sẽ cần ghi ra năm là dạng text và biến max tương ứng với nhiệt độ cao nhất trong năm đó ở dạng IntWritable trong hàm setOutputKeyClass và setOutputValueClass và sau đó ghi toàn bộ kết quả vào folder output



Và hình trên đây là kết quả sau khi chúng ta chạy chương trình thành công

6. Assignment 6 - AverageSalary Program

```

410 import org.apache.hadoop.io.FloatWritable;
411
412 public class WordCount
413 {
414     public static class avgMapper extends Mapper<Object,Text,Text,FloatWritable>
415     {
416         private Text dept_id=new Text();
417         private FloatWritable salary = new FloatWritable();
418         public void map(Object key, Text value,Context context)throws IOException, InterruptedException{
419             String values[]=value.toString().split( regex: "\\t");
420             dept_id.set(values[0]);
421             salary.set(Float.parseFloat(values[2]));
422             context.write(dept_id,salary);
423         }
424     }

```

-ở hàm map, chúng ta sẽ split |t thành 1 mảng String values, sau đó tạo 2 object là dept_id và salary và set dữ liệu cho 2 object này là dữ liệu dc lấy từ file input, values[0] là cột 0, values[2] cột 2 của dòng thứ I trong file input, sau đó sẽ write dept_id và salary

```

425     public static class avgReducer extends Reducer<Text,FloatWritable,Text,FloatWritable>{
426         private FloatWritable result = new FloatWritable();
427         public void reduce(Text key, Iterable<FloatWritable>values, Context context)throws IOException,
428             InterruptedException{
429             float sum=0;
430             float count =0;
431             for(FloatWritable val: values){
432                 sum+=val.get();
433                 count++;
434             }
435             result.set(sum/count);
436             context.write(key,result);
437         }
438     }

```

Tiếp theo ở hàm map, chúng ta sẽ lặp qua tất cả các giá trị và tính sum cho giá trị salary , ID nào giống nhau thì sẽ được cộng dồn và dùng biến count để đếm số lượng ID , sau đó thì lấy sum/count để ra được lương trung bình và viết ra

```

439 ▶ @ public static void main(String[] args) throws Exception{
440     Configuration conf = new Configuration();
441     Job job=new Job(conf, jobName: "averagesal");
442     job.setJarByClass(WordCount.class);
443     job.setMapperClass(avgMapper.class);
444     job.setCombinerClass(avgReducer.class);
445     job.setReducerClass(avgReducer.class);
446     job.setOutputKeyClass(Text.class);
447     job.setOutputValueClass(FloatWritable.class);
448     Path p=new Path(args[0]);
449     Path p1=new Path(args[1]);
450     FileInputFormat.addInputPath(job,p);
451     FileOutputFormat.setOutputPath(job,p1);
452     job.waitForCompletion(verbose: true);
453 }
454 }

```

- Tiếp theo, trong hàm Main chúng ta sẽ tạo job với tên là averagesal và setJarByClass sẽ là tên lớp của chúng ta
- Chúng ta sẽ để lần lượt tên hàm Map và Reduce của chúng ta tương ứng vào trong setMapperClass và setReducerClass
- Do ở pha Map, chúng ta dùng context.write để viết kết quả là dạng text cho năm và dạng int cho nhiệt độ, nên setMapOutputKeyClass và setMapOutputValueClass của chúng ta lần lượt sẽ là Text.class và IntWritable.class
- Cuối cùng, chúng ta sẽ cần ghi ra năm là dạng text và biến max tương ứng với nhiệt độ cao nhất trong năm đó ở dạng IntWritable trong hàm setOutputKeyClass và setOutputValueClass và sau đó ghi toàn bộ kết quả vào folder output

Input:

1	1	NhanVienA	1000.000
2	2	NhanVienB	20000.000
3	3	NhanVienC	45000.25
4	4	NhanVienD	4000.1
5	5	NhanVienE	1000.142
6	1	NhanVienA	2000.00
7	2	NhanVienB	30000

Output:

1	1	1500.0
2	2	25000.0
3	3	45000.25
4	4	4000.1
5	5	1000.142

7. Assignment 7 - De Identify HealthCare Program

```

478 public class WordCount
479 {
480     static Logger logger = Logger.getLogger(WordCount.class.getName());
481     public static Integer[] encryptCol={2,3,4,5,6,8};
482     private static byte[] key1 = new String( "original: \"sampleKey1234567\"").getBytes();
483     public static class Map extends Mapper<Object, Text, NullWritable, Text> {
484         @Override
485         public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
486             //value = PatientID, Name,DOB,Phone Number,Email_Address,SSN,Gender,Disease,weight
487             StringTokenizer itr = new StringTokenizer(value.toString(), delim: ",");
488             List<Integer> list=new ArrayList<Integer>();
489             Collections.addAll(list, encryptCol);
490             //list=2,3,4,5,6,8
491             System.out.println("Mapper :: one :"+value);
492             String newStr="";
493             int counter=1;
494             while (itr.hasMoreTokens()) {
495                 String token=itr.nextToken();
496                 System.out.println("token"+token);
497                 System.out.println("i="+counter);
498                 if(list.contains(counter))
499                 {
500                     if(newStr.length()>0)
501                         newStr+=",";
502                     newStr+=encrypt(token, key1);
503                 }
504                 else
505                 {
506                     if(newStr.length()>0)
507                         newStr+=",";
508                     newStr+=token;
509                 }
510                 counter=counter+1;
511             }
512             context.write(NullWritable.get(), new Text(newStr.toString()));
513         }
514     }
515 }

```

Ban đầu sử hàm StringTokenizer để split “,”

Sau đó Collection.addAll(list,encryptcol) để add danh sách các cột split là 2, 3, 4, 5, 6, 8 đã khai bài trong mảng encryptcol

Bên trong vòng lặp, điều kiện là lặp cho tới khi nào không còn token

- Ban đầu đếm cột counter là 1 không có trong danh sách cột cần mã hóa thì sẽ rơi vào trường hợp else không mã hóa.
- Nếu cột counter là {2, 3, 4, 5, 6, 8} tức là chưa trong list thì sẽ encrypt, hàm encrypt sẽ được giải thích bên dưới.

-

```

514 ▶ @ public static void main(String[] args) throws Exception {
515     if (args.length != 2) {
516         System.out.println("usage: [input] [output]");
517         System.exit( status: -1);
518     }
519     Job job = Job.getInstance(new Configuration());
520     job.setOutputKeyClass(NullWritable.class);
521     job.setOutputValueClass(Text.class);
522     job.setMapperClass(Map.class);
523     job.setInputFormatClass(TextInputFormat.class);
524     job.setOutputFormatClass(TextOutputFormat.class);
525     FileInputFormat.setInputPaths(job, new Path(args[0]));
526     FileOutputFormat.setOutputPath(job, new Path(args[1]));
527     job.setJarByClass(WordCount.class);
528     job.waitForCompletion( verbose: true);
529 }

```

- Tiếp theo, trong hàm Main chúng ta sẽ getInstance job
- Chúng ta sẽ để lần lượt tên hàm Map và Reduce của chúng ta tương ứng vào trong setMapperClass và setReducerClass
- Do ở pha Map, chúng ta dùng context.write để viết kết quả là dạng text cho năm và dạng int cho nhiệt độ, nên setMapOutputKeyClass và setMapOutputValueClass của chúng ta lần lượt sẽ là Text.class và IntWritable.class
- Cuối cùng, chúng ta sẽ cần ghi ra năm là dạng text và biến max tương ứng với nhiệt độ cao nhất trong năm đó ở dạng IntWritable trong hàm setOutputKeyClass và setOutputValueClass và sau đó ghi toàn bộ kết quả vào folder output

```

530 @      public static String encrypt(String strToEncrypt, byte[] key)
531 {
532     try
533     {
534         Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
535         SecretKeySpec secretKey = new SecretKeySpec(key, "AES");
536         cipher.init(Cipher.ENCRYPT_MODE, secretKey);
537         //cipher.init(Cipher.DECRYPT_MODE, secretKey);
538         String encryptedString = Base64.encodeBase64String(cipher.doFinal(strToEncrypt.getBytes()));
539         //String decrypted = new String(cipher.doFinal(Base64.decodeBase64(strToEncrypt)));
540         return encryptedString.trim();
541         //return decrypted;
542     }
543     catch (Exception e)
544     {
545         logger.error("Error while encrypting", e);
546     }
547     return null;
548 }
549 }

```

- Tận dụng thư viện(`javax.crypto.Cipher`, `javax.crypto.spec.SecretKeySpec`,, `org.apache.commons.codec.binary.Base64`). Chúng ta sẽ mã hóa theo dạng AES 256-bit, ECB mode, PKCS5 Padding, với mã khóa bí mật sẽ là chuỗi danh sách list key mà ta đã xác định tham số ở trên dưới dạng AES.
- Sau đó chúng ta sử dụng thuật toán base64 để mã hóa các kí tự. Sau đó ta thu gọn chuỗi lại ta sẽ trả về chuỗi đã được mã hóa.
- Trường hợp nếu không mã hóa được chúng ta sẽ dùng lệnh catch để bắt lỗi xảy ra.

Input:

```

WordCount.java x part-r-00000 x samplekey1234567 x averagesal x CDRlog.txt x A
1 11111,bbb1,12/10/1950,1.23E+09,bbb1@xxx.com,1.11E+09,M,Diabetes,78
2 11112,bbb2,12/10/1984,1.23E+09,bbb2@xxx.com,1.11E+09,F,PCOS,67
3 11113,bbb3,712/11/1940,1.23E+09,bbb3@xxx.com,1.11E+09,M,Fever,90
4 11114,bbb4,12/12/1950,1.23E+09,bbb4@xxx.com,1.11E+09,F,Cold,88
5 11115,bbb5,12/13/1960,1.23E+09,bbb5@xxx.com,1.11E+09,M,Blood Pressure,76
6 11116,bbb6,12/14/1970,1.23E+09,bbb6@xxx.com,1.11E+09,F,Malaria,84

```

Output:

```

WordCount.java x part-r-00000 x samplekey1234567 x averagesal x CDRlog.txt x AverageSalary.java x
1 11116,MB10//Xw1NsUSLNNR987sw==,bu01jxC7FAP9GaLzWtjdA==,msgB4nawyTSRW0xS/DdxeQ==,PaNHJZoVYjqkPLI8L4Ju1A==,CLEExkCr20VeqEZKS0W+kA==,F,tWwJednnIQwt0CjV6YR7hA==,84
2 11115,w0wAJ4Jv0EuE03175CE3w==,c516Lk2XDFXcciML1oXeS0==,msgB4nawyTSRW0xS/DdxeQ==,90zBX060U5IzeYDAHLtb0Q==,CLEExkCr20VeqEZKS0W+kA==,M,Yt/3yK6kQE+/oRVDEResrA==,76
3 11114,70xnmfmg1kXGKrSkn3a3Q==,LlhfxWYbuvBKynXmhhX6A==,msgB4nawyTSRW0xS/DdxeQ==,/zSL1rFR5HBucWAKyw88cA==,CLEExkCr20VeqEZKS0W+kA==,F,C62Zskv9C7canR9HMqKwyg==,88
4 11113,d3fvTuErIdCjb8nZd9Yx4Q==,B04M0r06+nrLytWAXTv7MA==,msgB4nawyTSRW0xS/DdxeQ==,fMDg+phn865IHgpWcJgcVQ==,CLEExkCr20VeqEZKS0W+kA==,M,RLwXYW+QMqdJfQ4KH109Iw==,90
5 11112,4UuhbSjT11BaKzNtxSgaZw==,ESE09NsjybbgZJP4oUK7+Q==,msgB4nawyTSRW0xS/DdxeQ==,Se8fr+0Irrzh0j52w/a/1bQ==,CLEExkCr20VeqEZKS0W+kA==,F,CxnhbQJvMh9zUL7dHQImKw==,67
6 11111,DDR9LoE/50TF7xm00bpDQ==,/u1STvYwDT1NXQWSw+wy5A==,msgB4nawyTSRW0xS/DdxeQ==,E4NpafIA1rnrwb87vWEXqQ==,CLEExkCr20VeqEZKS0W+kA==,M,Ez3Ysk7Twbh3ztW2de9fQ==,78
7

```

8. Assignment 8- Music Track Program

Bài này sẽ đếm có bao nhiêu userID trả

```
public static final int user_id = 0, track_id = 1;

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

    @Override
    protected void map(Object key, Text value, Mapper<Object, Text, IntWritable, IntWritable>.Context context) throws IOException, Interrup
        String[] parts = value.toString().split( regex: "[|]");
        trackID.set(Integer.parseInt(parts[WordCount.track_id]));
        userID.set(Integer.parseInt(parts[WordCount.user_id]));
        if(parts.length == 5) {
            context.write(trackID, userID);
        }
        else{
            context.getCounter(COUNTERS.INVALID_RECORD_COUNT).increment(1L);
        }
    }
}
```

Ở hàm map, ban đầu sẽ split | và lưu về 1 String part, tách theo từng cột, ban đầu tạo ra 2 object trackID, userID, tiếp theo sẽ set trackID và userID là phần tử thứ i của dòng i tương ứng với số index ta đã xác định ở khai báo.

Sau khi đã split và set thành công sẽ write trackID và userID, file input có 5 cột nên sẽ có length sẽ bằng 5 sau khi tách

```
public static class Reduce extends Reducer<IntWritable, IntWritable, IntWritable, IntWritable>{
    @Override
    protected void reduce(IntWritable key, Iterable<IntWritable> values, Reducer<IntWritable, IntWritable, IntWritable, IntWritable>.Context context) throws IOException, Interrup
        Set<Integer> userIDSet = new HashSet<>();
        for(IntWritable userID: values) {
            userIDSet.add(userID.get());
        }
        IntWritable size = new IntWritable(userIDSet.size());
        context.write(key, size);
    }
}
```

Khai báo set<integer> biến là userIDSet, lặp qua từng giá trị của userID ở lúc trước mà ta đã set dữ liệu rồi, sau đó sẽ add vào set userIDSet, xuất hiện lần đầu thì count 1, xuất hiện lần 2 thì count là 2

Vd key size

222 1

223 1

225 1

225 2

Sau đó sẽ viết ra key và size

```

public static void main(String[] args) throws Exception{
    Configuration conf = new Configuration();
    Job job = new Job(conf, "music track");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(verbose: true) ? 0 : 1);
    org.apache.hadoop.mapreduce.Counters counters = job.getCounters();
    System.out.println("No. of Invalid Records : " + counters.findCounter(COUNTERS.INVALID_RECORD_COUNT).getValue());
}

```

- Tiếp theo, trong hàm Main chúng ta sẽ tạo job với tên là music track và setJarByClass sẽ là tên lớp của chúng ta
- Chúng ta sẽ để lần lượt tên hàm Map và Reduce của chúng ta tương ứng vào trong setMapperClass và setReducerClass
- Do ở pha Map, chúng ta dùng context.write để viết kết quả là dạng text cho năm và dạng int cho nhiệt độ, nên setMapOutputKeyClass và setMapOutputValueClass của chúng ta lần lượt sẽ là Text.class và IntWritable.class
- Cuối cùng, chúng ta sẽ cần ghi ra năm là dạng text và biến max tương ứng với nhiệt độ cao nhất trong năm đó ở dạng IntWritable trong hàm setOutputKeyClass và setOutputValueClass và sau đó ghi toàn bộ kết quả vào folder output

```

//Ex8
public class WordCount{
    private enum COUNTERS {
        INVALID_RECORD_COUNT
    }

    public static final int user_id = 0, track_id = 1;

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

        @Override
        protected void map(Object key, Text value, Mapper<Object, Text, IntWritable, IntWritable>.Context context) throws IOException, InterruptedException {
            String[] parts = value.toString().split("[|]");
            trackID.set(Integer.parseInt(parts[WordCount.track_id]));
            userID.set(Integer.parseInt(parts[WordCount.user_id]));
            if(parts.length == 5) {
                context.write(trackID, userID);
            }
            else{
                context.getCounter(COUNTERS.INVALID_RECORD_COUNT).increment(1);
            }
        }
    }

    public static class Reduce extends Reducer<IntWritable, IntWritable, IntWritable, IntWritable>{
        @Override
        protected void reduce(IntWritable key, Iterable<IntWritable> values, Reducer<IntWritable, IntWritable, IntWritable, IntWritable>.Context context) throws IOException, InterruptedException {
            Set<Integer> userIDSet = new HashSet<>();
            for(IntWritable userID: values) {
                userIDSet.add(userID.get());
            }
        }
    }
}

```

```

2021-04-17 17:00:00
part: 222
part: 225
part: 223
part: 225
2021-04-17 17:00:00

```

9. Assignment 9 - Telecom Call Data Record Program

```

public static class TelecomRecord{
    public static final int fromPhone = 0, toPhone = 1, callStart = 2, callEnd = 3, flag = 4;
}

public static class Map extends Mapper<Object, Text, Text, LongWritable>{
    Text phoneNumber = new Text();
    LongWritable durationInMinutes = new LongWritable();

    @Override
    protected void map(Object key, Text value, Mapper<Object, Text, Text, LongWritable>.Context context) throws IOException, InterruptedException {
        String[] parts = value.toString().split( regex: "[|]");
        if (parts[TelecomRecord.flag].equalsIgnoreCase( anotherString: "1")) {
            phoneNumber.set(parts[TelecomRecord.fromPhone]);
            String callStartTime = parts[TelecomRecord.callStart];
            String callEndTime = parts[TelecomRecord.callEnd];
            long duration = toMillis(callEndTime) - toMillis(callStartTime);
            durationInMinutes.set(duration / (1000 * 60));
            context.write(phoneNumber, durationInMinutes);
        }
    }
}

```

Ở hàm map, ban đầu split | và lưu thành 1 mảng String theo từng cột như bài 8

Chúng ta sẽ xét điều kiện, chỉ xét những flag bằng 1

Bên trong hàm if chúng ta sẽ set dữ liệu cho các biến, fromPhone cột 0, tophone cột 1, callstart cột 2 callEnd cột 3 của dòng thứ I (I từ 1->6)

-dùng hàm toMillis để chuyển dạng ngày sang số giây, thì duration sẽ bằng end-start, lấy số lượng giây / (1000*60) là sẽ ra số phút

```
public static class Reduce extends Reducer<Text, LongWritable, Text, LongWritable>{
    LongWritable result = new LongWritable();

    @Override
    protected void reduce(Text key, Iterable<LongWritable> values, Reducer<Text, LongWritable, Text, LongWritable> context) throws IOException, InterruptedException {
        long sum = 0;
        for(LongWritable val: values){
            sum += val.get();
        }
        result.set(sum);
        if(sum >= 60){
            context.write(key, this.result);
        }
    }
}
```

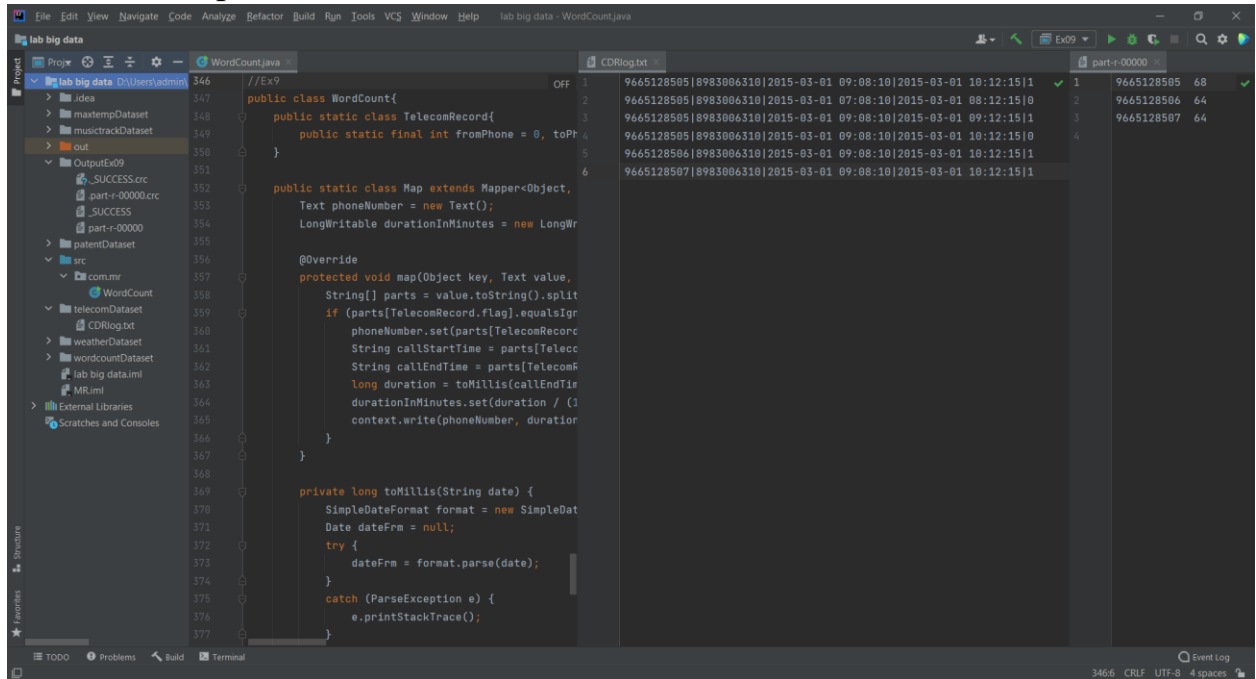
Ở hàm reduce này sẽ các số điện thoại giống nhau sẽ sum các giá trị durationminutes lại, nếu lớn hơn 60 thì viết ra

```
public static void main(String[] args) throws Exception{
    Configuration conf = new Configuration();
    Job job = new Job(conf, "telecom record");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(Map.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(verbose: true) ? 0 : 1);
}
```

- Tiếp theo, trong hàm Main chúng ta sẽ tạo job với tên là telecom record và setJarByClass sẽ là tên lớp của chúng ta
- Chúng ta sẽ để lần lượt tên hàm Map và Reduce của chúng ta tương ứng vào trong setMapperClass và setReducerClass
- Do ở pha Map, chúng ta dùng context.write để viết kết quả là dạng text cho năm và dạng int cho nhiệt độ, nên setMapOutputKeyClass và

setMapOutputValueClass của chúng ta lần lượt sẽ là Text.class và IntWritable.class

- Cuối cùng, chúng ta sẽ cần ghi ra năm là dạng text và biến max tương ứng với nhiệt độ cao nhất trong năm đó ở dạng IntWritable trong hàm setOutputKeyClass và setOutputValueClass và sau đó ghi toàn bộ kết quả vào folder output



```
//EX9
public class WordCount{
    public static class TelecomRecord{
        public static final int fromPhone = 0, toPhone = 1;
    }
    public static class Map extends Mapper<Object, Text>{
        Text phoneNumber = new Text();
        LongWritable durationInMinutes = new LongWritable();

        @Override
        protected void map(Object key, Text value,
            Context context) throws IOException, InterruptedException {
            String[] parts = value.toString().split("\\|");
            if (parts[TelecomRecord.flag].equals("Igr")){
                phoneNumber.set(parts[TelecomRecord.fromPhone]);
                String callStartTime = parts[TelecomRecord.toPhone];
                String callEndTime = parts[TelecomRecord.toPhone];
                long duration = toMillis(callEndTime) - toMillis(callStartTime);
                durationInMinutes.set(duration / 60);
                context.write(phoneNumber, durationInMinutes);
            }
        }

        private long toMillis(String date) {
            SimpleDateFormat format = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
            Date dateFrm = null;
            try {
                dateFrm = format.parse(date);
            } catch (ParseException e) {
                e.printStackTrace();
            }
        }
    }
}
```

key	value	count
9665128505 8983006310 2015-03-01 09:08:10 2015-03-01 10:12:15 1		1
9665128505 8983006310 2015-03-01 07:08:10 2015-03-01 08:12:15 0		2
9665128505 8983006310 2015-03-01 09:08:10 2015-03-01 09:12:15 1		3
9665128505 8983006310 2015-03-01 09:08:10 2015-03-01 10:12:15 0		4
9665128506 8983006310 2015-03-01 09:08:10 2015-03-01 10:12:15 1		5
9665128507 8983006310 2015-03-01 09:08:10 2015-03-01 10:12:15 1		6

3. Tự đánh giá:

Các assignment được giao đã hoàn thành và chạy ra kết quả

Khó khăn gặp phải là code chạy bằng ngôn ngữ Java nhưng sử dụng một số thư viện của Hadoop nên còn lạ lẫm, khó hiểu. Bên cạnh đó thì có bạn trong nhóm import các file jar của hadoop bị lỗi dẫn đến không chạy code được nhưng đã khắc phục được sau khi cài lại hadoop.

4. Nguồn tham khảo

Sriram Balasubramanian (2016), Hadoop-MapReduce Lab Map |Reduce |Driver, Cloudera, https://courses.ctda.hcmus.edu.vn/pluginfile.php/53002/mod_resource/content/4/Labs%20in%20Detail.pdf