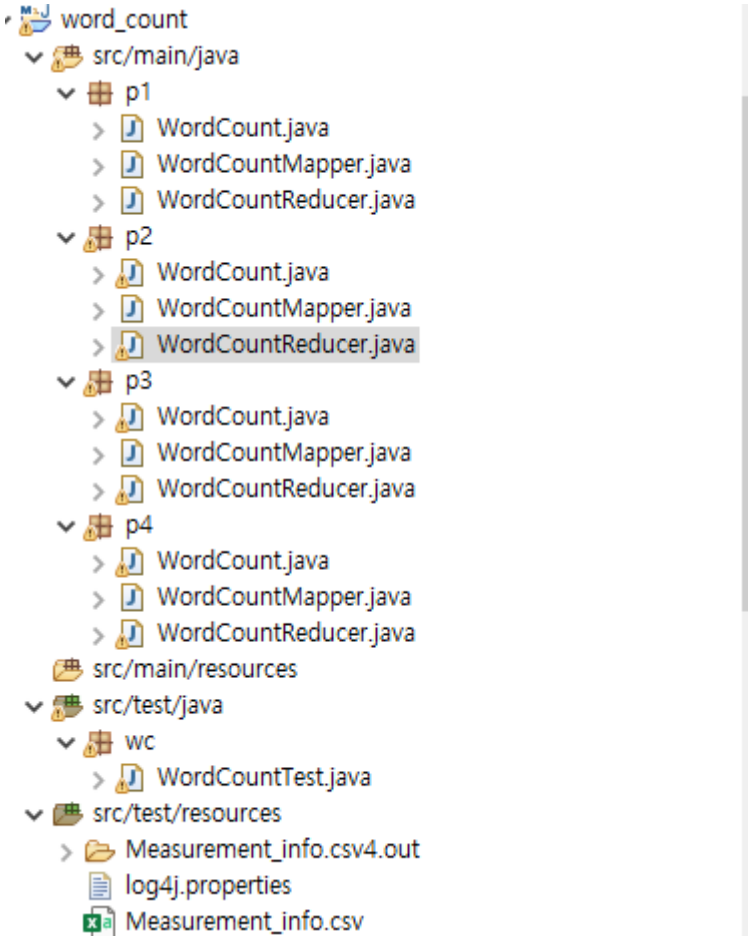
서울 공기질 분석

<맵리듀스와 분산 클러스터 활용하여 서울 공기질을 분석>

학번 : 20181612

이름 : 박병규

구성



문제 수에 맞게 package를 생성하였고 패키지 안에서 각각 Driver, Mapper, Reducer을 구현하였다.

문제 1) 각 지역(station code)별로 평균, 최대, 최소 PM10 측정치 구하기

결과

```
pbk5485@kmubigdata-cluster-m:~$ hdfs dfs -cat /Measurement_info.csv1.out/* | sort
101     average: 37 max: 289 min: 3
102     average: 38 max: 296 min: 3
103     average: 35 max: 330 min: 3
104     average: 42 max: 423 min: 1
105     average: 42 max: 401 min: 3
106     average: 43 max: 389 min: 3
107     average: 44 max: 411 min: 3
108     average: 41 max: 340 min: 3
109     average: 39 max: 326 min: 3
110     average: 39 max: 414 min: 3
111     average: 44 max: 421 min: 3
112     average: 39 max: 322 min: 2
113     average: 40 max: 354 min: 1
114     average: 40 max: 289 min: 3
115     average: 42 max: 293 min: 3
116     average: 43 max: 389 min: 3
117     average: 43 max: 405 min: 3
118     average: 39 max: 329 min: 3
119     average: 47 max: 351 min: 3
120     average: 41 max: 321 min: 3
121     average: 44 max: 385 min: 3
122     average: 44 max: 470 min: 1
123     average: 39 max: 302 min: 1
124     average: 42 max: 426 min: 1
125     average: 45 max: 443 min: 1
```

쉘에서 sort를 이용하여 지역(station code)별로 평균, 최대, 최소 값을 출력하였다.

Driver

```java
package p1;

import org.apache.hadoop.conf.Configured;

public class WordCount extends Configured implements Tool {
    // Configured -> Hadoop에 설정 값을 전달해준다.
    // Tool : command 라인에서 실행할 때 옵션값을 받기 편하게 해준다.
    public int run(String[] args) throws Exception {

        Job myjob = Job.getInstance(getConf()); // my job -> 내가 지금 돌릴려고 하는 일이다. configured의 getConf()에서 디폴트값들을 가지고온다.
        myjob.setJarByClass(WordCount.class);   // 내가 실행하고 있는 파일이 무엇인지 넘겨준다.

        myjob.setMapperClass(WordCountMapper.class);   // 내가 구현한 매퍼클래스를 지정해준다
        myjob.setReducerClass(WordCountReducer.class); // 내가 구현한 리듀서클래스를 지정해준다.

        myjob.setMapOutputKeyClass(Text.class);           // output 타입을 지정해준다
        myjob.setMapOutputValueClass(IntWritable.class);  // output 타입을 지정해준다.

        myjob.setOutputFormatClass(TextOutputFormat.class);  //  OutPutFomat이 Text이면 안써줘도 되긴하다
        myjob.setInputFormatClass(TextInputFormat.class);    // TextInputFormat을 지정해준다. SequenceFileInputFormat -> binary 파일 읽을 수 있다.

        FileInputFormat.addInputPath(myjob, new Path(args[0]));                    // 입력파일을 어디서 불러올 것인가
        FileOutputFormat.setOutputPath(myjob, new Path(args[0]).suffix("1.out"));  // 출력파일을 어디에 어떻게 저장을 할 것인가

        myjob.waitForCompletion(true);    // Hadoop를 실행하는 코드 true로 하면 로그들이 많이 찍힌다.

        return 0;
        }
    // command line에서 실행시키기 위해 구현한 코드
    public static void main(String[] args) throws Exception{
        ToolRunner.run(new WordCount(), args);
    }

}
```

수업에서 실습으로 진행했던 WordCount driver 스켈레톤 코드를 사용했다. 라인에 대한 설명은

주석으로 적었다

driver에서는 구현한 클래스들을 지정해 주었고 타입들을 지정해 줬다.

Mapper 함수 타입 <Object, Text, Text, IntWritable>

Reducer 함수 타입 <Text, IntWritable, Text, Text>

Mapper

```java
package p1;


import java.io.IOException;

//item code가 8인 경우PM10
//9인 경우 PM2.5
public class WordCountMapper extends Mapper<Object, Text, Text, IntWritable> {

    Text word = new Text();
    IntWritable num = new IntWritable();
    String date = "";
    String station = "";
    String item_code = "";
    String avg = "";
    String status = "";
    int k;
    @Override
    protected void map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context context)
            throws IOException, InterruptedException{

        StringTokenizer st = new StringTokenizer(value.toString(), ","); //split해준다

        if (st.hasMoreTokens()) {date = st.nextToken();}
        if (st.hasMoreTokens()) {station = st.nextToken();}
        if (st.hasMoreTokens()) {item_code = st.nextToken();}
        if (st.hasMoreTokens()) {avg = st.nextToken();}
        if (st.hasMoreTokens()) {status = st.nextToken();}

        if (item_code.equals("8") && status.equals("0")) {
            k = (int) Double.parseDouble(avg);
            num.set(k);
            word.set(station);
            context.write(word, num);
            }
    }
}
```

Mapper 함수에서는 먼저 이상치에 대해 제거를 다 해주었다.

제거하는 방식으로는 분석해야하는 csv 파일인 Measurement_info.csv파일을 보게되면 status 값이 0이 아닌 값들은 이상치 임을 확인하고 status가 0인 값들로 과제를 진행하였다.

Item_code가 8(PM10)이고 status 값이 0인 값에 대해서

Key 를 지역코드로 하고, 이때 의 측정값을 value로 reducer에게 전달했다.

Reducer

```java
package p1;

import java.io.IOException;

public class WordCountReducer extends Reducer<Text, IntWritable, Text, Text>{
    Text ret = new Text();
    Text total = new Text();

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values,
            Reducer<Text, IntWritable, Text, Text>.Context context)
            throws IOException, InterruptedException{
        int sum = 0;
        int min = 99999999;
        int max = 0;
        int cnt = 0;

        for(IntWritable value : values) {
            sum += value.get();
            cnt ++;
            if (value.get() > max) max = value.get();
            if (value.get() < min) min = value.get();
        }
        int avg = Math.round(sum/cnt);
        ret.set("average: "+ Integer.toString(avg) + " max: " + Integer.toString(max) + " min: " + Integer.toString(min));
        context.write(key, ret);
    }
}
```

Reducer 함수에서는 평균, 최대, 최소 값을 구해야 하므로 이에 맞는 변수를 선언해 주고.

Values에 있는 value를 연산하여 출력 값들을 구했다

<Text, Text> 타입으로 출력을 했다

문제 2) PM10, PM2.5 기준으로 공기의 질이 '좋음' 수준이 가장 많이 측정된 지역은 어디인지 찾기(공기질 좋음 = PM10 기준: 30 이하, PM2.5 기준 15 이하)

결과

```
pbk5485@kmubigdata-cluster-m:~$ hdfs dfs -cat /Measurement_info.csv2.out/* | sort -k 2
119     6069
106     6345
121     6398
111     6929
125     7020
104     7025
115     7333
116     7396
107     7411
108     7575
120     7685
114     7728
117     7728
122     7742
118     7775
105     7836
124     7970
109     8101
123     8425
110     8517
113     8768
101     8778
103     8884
102     8978
112     9669
pbk5485@kmubigdata-cluster-m:~$ hdfs dfs -cat /Measurement_info.csv2.out/* | sort -k 2 | tail -1
112     9669
pbk5485@kmubigdata-cluster-m:~$
```

맵리듀스 함수를 이용하여 지역코드별 공기질이 좋았던 횟수를 카운트했다. 그리고 나서 sort와 tail을 사용하여 shell 상에서 가장 많이 미세먼지가 좋았던 지역을 출력하였다.

Driver

```java
package p2;

import org.apache.hadoop.conf.Configured;

public class WordCount extends Configured implements Tool {


    public int run(String[] args) throws Exception {

        Job myjob = Job.getInstance(getConf());
        myjob.setJarByClass(WordCount.class);

        myjob.setMapperClass(WordCountMapper.class);
        myjob.setReducerClass(WordCountReducer.class);

        myjob.setMapOutputKeyClass(Text.class);
        myjob.setMapOutputValueClass(Text.class);

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

        FileInputFormat.addInputPath(myjob, new Path(args[0]));
        FileOutputFormat.setOutputPath(myjob, new Path(args[0]).suffix("2.out"));

        myjob.waitForCompletion(true);

        return 0;
        }

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

}
```

Driver의 코드 내용은 문제 1번과 유사하며 Mapper이 value Output 타입을 Text로 바꾼 것과 출력 파일의 이름을 "2.out"을 붙인 것 이외에는 동일하다.

Mapper

```java
package p2;

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

//item code가 8인 경우PM10
//9인 경우 PM2.5
public class WordCountMapper extends Mapper<Object, Text, Text, Text>{
    Text word = new Text();
    Text word2 = new Text();
    IntWritable num = new IntWritable(1);
    String date = "";
    String station = "";
    String item_code = "";
    String avg = "";
    String status = "";
    int k;
    int area;
    @Override
    protected void map(Object key, Text value, Mapper<Object, Text, Text, Text>.Context context)
            throws IOException, InterruptedException{
        StringTokenizer st = new StringTokenizer(value.toString(), ","); //split해준다
        if (st.hasMoreTokens()) {date = st.nextToken();}
        if (st.hasMoreTokens()) {station = st.nextToken();}
        if (st.hasMoreTokens()) {item_code = st.nextToken();}
        if (st.hasMoreTokens()) {avg = st.nextToken();}
        if (st.hasMoreTokens()) {status = st.nextToken();}

        if (item_code.equals("8") && status.equals("0")) {
            k = (int) Double.parseDouble(avg);
            if ( k <= 30) {
                word.set(station);
                word2.set(date);
                context.write(word, word2);
            }
        }
        else if (item_code.equals("9") && status.equals("0")) {
            k = (int) Double.parseDouble(avg);
            if ( k <= 15) {
                word.set(station);
                word2.set(date);
                context.write(word, word2);
            }
        }
```

Mapper의 입출력 타입으로는 <Object, Text, Text, Text>타입이다.

문제 2에서도 문제 1과 동일하게 이상치를 제거해줬다.

Key로는 지역 코드로 설정해 두었고 Value로는 이상치가 아니고 PM10(item_code==8),

PM2.5(item_code==9)인 측정치 값들이 각각 30, 15 이하인 경우로 설정했다.

Reducer

```java
package p2;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.Collections;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.util.HashSet;

public class WordCountReducer extends Reducer<Text, Text, Text, Text>{
    Text ret = new Text();
    Text total = new Text();
    Map<Text, Integer> pick_good = new HashMap<Text, Integer>();
    IntWritable n = new IntWritable(1);

    @Override
    protected void reduce(Text key, Iterable<Text> values,
            Reducer<Text, Text, Text, Text>.Context context)
            throws IOException, InterruptedException{
        int sum = 0;

        Set<Text> list = new HashSet<Text>();

        for(Text value : values) {
            if (list.contains(value)) {
                sum ++;
                list.remove(value);
            }
            else {
                list.add(value);
            }
        }

        ret.set(Integer.toString(sum));
        context.write(key, ret);
    }
}
```

Reducer에서는 set을 이용해서 PM10과 PM2.5가 둘 다 좋은 수치인지 확인했다.

PM10과 PM2.5가 같은 날에 좋은 수치를 보내야 공기질이 좋은 날이라고 볼 수 있다. 따라서 집합 안에 먼저 전달받은 date값들을 저장을 해 두었고 만약 같은 date가 들어오게 된다면 sum을 1씩 증가시켰다.

그리고 지역코드와 공기질이 좋은 날들을 카운트 한 값을 리턴해주었다.

문제 3) 데이터 변환하기 → 각 <시간, 지역>별로 모든 종류의 측정치 모아서 저장하기

결과

```
pbk5485@kmubigdata-cluster-m:~$ hdfs dfs -cat /Measurement_info.csv3.out/* | head
<2017-01-01 00:00, 101> SO2: 0.004 NO2: 0.059000000000000004 CO: 1.2 O3: 0.002 PM10: 73.0 PM2: 57.0
<2017-01-01 00:00, 106> SO2: 0.005 NO2: 0.066 CO: 1.5 O3: 0.003 PM10: 71.0 PM2: 62.0
<2017-01-01 00:00, 110> SO2: 0.005 NO2: 0.04 CO: 0.8 O3: 0.002 PM10: 91.0 PM2: 50.0
<2017-01-01 00:00, 115> SO2: 0.005 NO2: 0.055 CO: 1.3 O3: 0.002 PM10: 75.0 PM2: 48.0
<2017-01-01 00:00, 124> SO2: 0.006 NO2: 0.04 CO: 1.4 O3: 0.003 PM10: 60.0 PM2: 51.0
<2017-01-01 01:00, 101> SO2: 0.004 NO2: 0.057999999999999996 CO: 1.2 O3: 0.002 PM10: 71.0 PM2: 59.0
<2017-01-01 01:00, 106> SO2: 0.004 NO2: 0.064 CO: 1.5 O3: 0.003 PM10: 70.0 PM2: 62.0
<2017-01-01 01:00, 110> SO2: 0.005 NO2: 0.043 CO: 0.8 O3: 0.002 PM10: 75.0 PM2: 52.0
<2017-01-01 01:00, 115> SO2: 0.005 NO2: 0.053 CO: 1.3 O3: 0.002 PM10: 74.0 PM2: 49.0
<2017-01-01 01:00, 124> SO2: 0.005 NO2: 0.036000000000000004 CO: 1.3 O3: 0.003 PM10: 65.0 PM2: 43.0
cat: Unable to write to output stream.
cat: Unable to write to output stream.
```

Driver

```java
package p3;

import org.apache.hadoop.conf.Configured;

public class WordCount extends Configured implements Tool {


    public int run(String[] args) throws Exception {

        Job myjob = Job.getInstance(getConf());
        myjob.setJarByClass(WordCount.class);

        myjob.setMapperClass(WordCountMapper.class);
        myjob.setReducerClass(WordCountReducer.class);

        myjob.setMapOutputKeyClass(Text.class);
        myjob.setMapOutputValueClass(Text.class);

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

        FileInputFormat.addInputPath(myjob, new Path(args[0]));
        FileOutputFormat.setOutputPath(myjob, new Path(args[0]).suffix("3.out"));

        myjob.waitForCompletion(true);

        return 0;
        }

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

}
```

출력 파일의 이름을 제외하고는 문제 2의 Driver 코드와 동일합니다.

Mapper

```java
package p3;

import java.io.IOException;

public class WordCountMapper extends Mapper<Object, Text, Text, Text>{

    Text word = new Text();
    Text word2 = new Text();
    IntWritable num = new IntWritable(1);
    String date = "";
    String station = "";
    String item_code = "";
    String avg = "";
    String status = "";
    int k;
    int area;
    @Override
    protected void map(Object key, Text value, Mapper<Object, Text, Text, Text>.Context context)
            throws IOException, InterruptedException{

        StringTokenizer st = new StringTokenizer(value.toString(), ","); //split해준다

        if (st.hasMoreTokens()) {date = st.nextToken();}
        if (st.hasMoreTokens()) {station = st.nextToken();}
        if (st.hasMoreTokens()) {item_code = st.nextToken();}
        if (st.hasMoreTokens()) {avg = st.nextToken();}
        if (st.hasMoreTokens()) {status = st.nextToken();}
        if (status.equals("0")) {
            String por_key = "<" + date + ", " + station + ">";
            word.set(por_key);
            String val = "(" + item_code + avg + ")";
            word2.set(val);
            context.write(word, word2);
        }
    }
}
```

Key 값으로 date와 지역 코드가 합친 Text 타입으로 설정하였다.

Value 값으로는 item_code와 측정치를 합친 Text타입으로 설정하였다.

Reducer

```java
package p3;

import java.io.IOException;

public class WordCountReducer extends Reducer<Text, Text, Text, Text>{
    Text ret = new Text();
    Text total = new Text();
    Map<Text, Integer> pick_good = new HashMap<Text, Integer>();
    IntWritable n = new IntWritable(1);

    @Override
    protected void reduce(Text key, Iterable<Text> values,
            Reducer<Text, Text, Text, Text>.Context context)
            throws IOException, InterruptedException{
        //String ret_string = "";
        String s1 = "";
        String s2 = "";
        String s3 = "";
        String s4 = "";
        String s5 = "";
        String s6 = "";
        String v = "";

        for(Text value : values) {
            v = value.toString();
            if (v.equals("1")) {
                s1 = v.substring(3);
            }
            else if (v.equals("3")) {
                s2 = v.substring(3);
            }
            else if (v.equals("5")) {
                s3 = v.substring(3);
            }
            else if (v.equals("6")) {
                s4 = v.substring(3);
            }
            else if (v.equals("8")) {
                s5 = v.substring(3);
            }
            else if (v.equals("9")) {
                s6 = v.substring(3);
            }
        }
        String ret_string = "SO2: " + s1 +  " NO2: " + s2 + " CO: "  +  s3 + " O3: " + s4 + " PM10: " + s5 + " PM2: " + s6;
        ret.set(ret_string);
        context.write(key, ret);
    }
}
```

시간 지역 코드를 Key 로 받고 아이템 코드와 측정값을 합친 Text 타입을 입력으로 받았다.

Substring을 이용하여 아이템 코드를 뽑았고 아이템 코드에 맞춰 미세먼지의 측정치들을 개별적으로 분리를 하였다.

분리한 측정치들을 보기 쉽게 아이템들의 이름에 따라 구별하는 Text 타입의 문자열을 만들었고 출력을 해주었다

문제 4) 시간대를 기준으로 평균 공기질 구하기 (SO2, NO2, CO, O3, PM10, PM2.5 한꺼번에 구하기)

결과

```
pbk5485@kmubigdata-cluster-m:~$ hdfs dfs -cat /Measurement_info.csv4.out/* | sort
00:00    SO2: 0.0041977675 NO2: 0.030757977 CO: 0.54184663 O3: 0.018234747 PM10: 41.089653 PM2: 24.470604
01:00    SO2: 0.0041624783 NO2: 0.027941374 CO: 0.53645694 O3: 0.018936202 PM10: 40.189247 PM2: 24.040787
02:00    SO2: 0.004124109 NO2: 0.02584021 CO: 0.5284036 O3: 0.019200742 PM10: 39.853836 PM2: 24.10432
03:00    SO2: 0.0040828963 NO2: 0.024584183 CO: 0.5210749 O3: 0.01891138 PM10: 38.598225 PM2: 23.494148
04:00    SO2: 0.0040511196 NO2: 0.024580104 CO: 0.5180735 O3: 0.017839154 PM10: 38.66395 PM2: 23.62679
05:00    SO2: 0.004044188 NO2: 0.027083615 CO: 0.52651876 O3: 0.015178583 PM10: 38.270653 PM2: 23.312805
06:00    SO2: 0.0040890058 NO2: 0.03119647 CO: 0.5481072 O3: 0.012429276 PM10: 39.250732 PM2: 23.637861
07:00    SO2: 0.0042216913 NO2: 0.033589832 CO: 0.57450205 O3: 0.012328027 PM10: 39.89085 PM2: 23.418598
08:00    SO2: 0.004381715 NO2: 0.033379197 CO: 0.5883516 O3: 0.0149618825 PM10: 42.249924 PM2: 24.184484
09:00    SO2: 0.0045221914 NO2: 0.03155316 CO: 0.57159364 O3: 0.019093461 PM10: 43.253498 PM2: 24.24625
10:00    SO2: 0.0045734188 NO2: 0.02925248 CO: 0.5374733 O3: 0.023924103 PM10: 44.573174 PM2: 24.76568
11:00    SO2: 0.004562792 NO2: 0.026755877 CO: 0.50322074 O3: 0.029342322 PM10: 43.945175 PM2: 24.291567
12:00    SO2: 0.0045209187 NO2: 0.024349429 CO: 0.47724584 O3: 0.03471876 PM10: 43.734142 PM2: 24.276747
13:00    SO2: 0.0044727977 NO2: 0.023043275 CO: 0.45902652 O3: 0.038606394 PM10: 42.953014 PM2: 23.685593
14:00    SO2: 0.004444101 NO2: 0.02255363 CO: 0.4467627 O3: 0.040877443 PM10: 43.550022 PM2: 23.754595
15:00    SO2: 0.004433785 NO2: 0.022958063 CO: 0.44070673 O3: 0.041308492 PM10: 43.271736 PM2: 23.46515
16:00    SO2: 0.0044283327 NO2: 0.024397653 CO: 0.4474078 O3: 0.039357025 PM10: 43.627678 PM2: 23.813921
17:00    SO2: 0.0044006193 NO2: 0.027084535 CO: 0.461888 O3: 0.035168953 PM10: 42.83119 PM2: 23.53553
18:00    SO2: 0.00438601 NO2: 0.030430758 CO: 0.4904904 O3: 0.029895931 PM10: 42.95337 PM2: 23.943842
19:00    SO2: 0.004372101 NO2: 0.032740302 CO: 0.5168117 O3: 0.025593797 PM10: 42.666885 PM2: 24.115837
20:00    SO2: 0.004360192 NO2: 0.03329877 CO: 0.53107667 O3: 0.022653352 PM10: 43.164486 PM2: 25.227777
21:00    SO2: 0.0043321582 NO2: 0.03326272 CO: 0.53713244 O3: 0.020641102 PM10: 42.20289 PM2: 25.143055
22:00    SO2: 0.004293159 NO2: 0.03340281 CO: 0.5412082 O3: 0.018994197 PM10: 41.79941 PM2: 24.991611
23:00    SO2: 0.004244554 NO2: 0.032781668 CO: 0.5425181 O3: 0.018129194 PM10: 40.83499 PM2: 24.406862
```

시간대를 기준으로 공기질의 각각의 측정요소들의 평균을 출력하였다.

Driver

```
package p4;

import org.apache.hadoop.conf.Configured;

public class WordCount extends Configured implements Tool {


    public int run(String[] args) throws Exception {

        Job myjob = Job.getInstance(getConf());
        myjob.setJarByClass(WordCount.class);

        myjob.setMapperClass(WordCountMapper.class);
        myjob.setReducerClass(WordCountReducer.class);

        myjob.setMapOutputKeyClass(Text.class);
        myjob.setMapOutputValueClass(Text.class);

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

        FileInputFormat.addInputPath(myjob, new Path(args[0]));
        FileOutputFormat.setOutputPath(myjob, new Path(args[0]).suffix("4.out"));

        myjob.waitForCompletion(true);

        return 0;
        }

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

}
```

출력 파일을 제외하고는 문제 2 Driver 코드와 동일합니다.

Mapper

```java
package p4;

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class WordCountMapper extends Mapper<Object, Text, Text, Text>{

    Text word = new Text();
    Text word2 = new Text();
    IntWritable num = new IntWritable(1);
    String date = "";
    String station = "";
    String item_code = "";
    String avg = "";
    String status = "";
    int k;
    int area;
    @Override
    protected void map(Object key, Text value, Mapper<Object, Text, Text, Text>.Context context)
            throws IOException, InterruptedException{

        StringTokenizer st = new StringTokenizer(value.toString(), ","); //split해준다

        if (st.hasMoreTokens()) {date = st.nextToken();}
        if (st.hasMoreTokens()) {station = st.nextToken();}
        if (st.hasMoreTokens()) {item_code = st.nextToken();}
        if (st.hasMoreTokens()) {avg = st.nextToken();}
        if (st.hasMoreTokens()) {status = st.nextToken();}

        if (status.equals("0")) {
            word.set(date.substring(11));
            String word2_string = item_code + " " + avg;
            word2.set(word2_string);
            context.write(word, word2);
        }
    }
}
```

Key 값을 date값에서 시간을 뽑아주었다 설정을 했다.

value값으로는 아이템 코드와 이에 맞는 측정값으로 설정하였다.

Reducer

```java
package p4;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Collections;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class WordCountReducer extends Reducer<Text, Text, Text, Text>{
    Text ret = new Text();
    Text total = new Text();
    Map<Text, Integer> pick_good = new HashMap<Text, Integer>();
    IntWritable n = new IntWritable(1);
    @Override
    protected void reduce(Text key, Iterable<Text> values,
            Reducer<Text, Text, Text, Text>.Context context)
            throws IOException, InterruptedException{
        //1 3 5 6 8 9
        String kf = "";
        String tmp = ""; //일단 분리시키자
        String estimate = "";
        float so2 = 0;
        int so2_cnt = 0;

        float no2 = 0;
        int no2_cnt = 0;

        float co = 0;
        int co_cnt = 0;

        float o3 = 0;
        int o3_cnt = 0;

        float PM10 = 0;
        int PM10_cnt = 0;

        float PM2 = 0;
        int PM2_cnt = 0:
```

```java
        for(Text value : values) {
            tmp = value.toString();
            kf = tmp.substring(0, 1);
            estimate = tmp.substring(2);

            if (kf.equals("1")) {
                so2 += Float.parseFloat(estimate);
                so2_cnt++;
            }
            else if (kf.equals("3")) {
                no2 += Float.parseFloat(estimate);
                no2_cnt++;
            }
            else if (kf.equals("5")) {
                co += Float.parseFloat(estimate);
                co_cnt++;
            }
            else if (kf.equals("6")) {
                o3 += Float.parseFloat(estimate);
                o3_cnt++;
            }
            else if (kf.equals("8")) {
                PM10 += Float.parseFloat(estimate);
                PM10_cnt++;
            }
            else if (kf.equals("9")) {
                PM2 += Float.parseFloat(estimate);
                PM2_cnt++;
            }
        }
        float avg_so2 = so2/so2_cnt;
        float avg_no2 = no2/no2_cnt;
        float avg_co = co/co_cnt;
        float avg_o3 = o3/o3_cnt;
        float avg_PM10 = PM10/PM10_cnt;
        float avg_PM2 = PM2/PM2_cnt;

        String result = "SO2: " + String.valueOf(avg_so2) +  " NO2: " + String.valueOf(avg_no2) + " CO: "  +
        String.valueOf(avg_co) + " O3: " + String.valueOf(avg_o3) + " PM10: " + String.valueOf(avg_PM10) + " PM2: " + String.valueOf(avg_PM2);
        ret.set(result);
        context.write(key, ret);
    }
}
```

분제 3과 동일하게 Mapper에게 넘겨받은 value인 문자열 값을 슬라이싱을 했다.

그로 인해 아이템 코드 값과 측정값을 얻었다. 그리고 이 값을 이용하여 아이템 별로의 측정값을 얻을 수 있었으며 이 값을 각각의 아이템 들이 측정값을 모아둔 변수에 저장을 해주고 카운트를 했다. 평균을 구하였고 시간을 key로 설정하고 각각의 아이템들이 갖는 평균 공기질을 value로 하여 출력하였다.