서울 공기질 분석

이름 : 임현진 학번 : 20181684

Map reduce의 개념

Map reduce는 거대한 input data를 쪼개어 수많은 머신들에 분산시켜서 로직을 수행시키고, 로직을 수행한 다음 결과를 하나로 합치는 것입니다. 여기에서는 split, map, reduce순서로 Map reduce가 진행됩니다.

1. Split : 크기가 큰 input파일을 작은 단위의 청크들로 나누어 분산 파일 시스템에 저장합니다.
2. Map : 잘게 쪼개어진 input을 받아와서 데이터를 분석하는 로직을 수행합니다.
3. Reduce : 처리된 데이터를 다시 합치는 작업을 수행합니다.

Map reduce는 이러한 과정을 통해서 대용량의 데이터를 효과적을 처리할 수 있게됩니다.

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

WeatherCount파일

이 파일에서는 run함수를 오버라이드 하여 hadoop에 필요한 설정값을 전달해줍니다. 내가 현재 작업하려는 작업, Mapper에서 실행할 mapper파일, Reducer에서 실행할 reducer파일, mapper함수의 output key,value 타입, inputformat과 outputformat, 마지막으로 input으로 들어올 경로와 결과값을 저장할 경로 등과 같은 설정값들을 지정해줍니다.

Code)

```
Job job = Job.getInstance(getConf());
        job.setJarByClass(WeatherCount.class);

        job.setMapperClass(WeatherCountMapper.class);
        job.setReducerClass(WeatherCountReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(DoubleWritable.class);

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

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

WeatherCountMapper파일

이 파일에서는 날씨 측적 정보 내용을 담은 파일을 읽어와서 PM10에 해당하는 값에 대해서만 key = 지역 코드, value = PM10 측정값으로 하여 context에 써줍니다.

이때 context는 reduce에서 다시 읽게됩니다.

```
public class WeatherCountMapper extends Mapper<Object, Text, Text, DoubleWritable>
```

이 부분은 들어오는값이 Text 형식이고 내보낼 key값과 value값의 타입을 각 각, Text와 double형으로 할 것을 뜻합니다.

우선, csv파일의 첫째줄은 실제 데이터 값이 아닌 각 열의 이름이 저장 되어있으므로 첫째줄은 건너뛰고 두번째줄부터 읽어들입니다.
그리고 들어오는 값을 콤마를 기준으로 split하여 값을 저장해야 하기 때문에 StrigTokenizer(value.toString(),",")을 사용하여 split 해주었습니다.
마지막으로 PM10은 들어오는 값 내에서는 8로 되어있기 때문에 if문을 이용하여 PM10인 값에 대해서만 key, value값을 설정해주었습니다.

```
public class WeatherCountMapper extends Mapper<Object, Text, Text, DoubleWritable>
{

    Text word = new Text();
    DoubleWritable one = new DoubleWritable();
    int n=0;
    @Override
    protected void map(Object key, Text value, Context context)
              throws IOException, InterruptedException {
        if(n>0)     //첫째줄을 건너뛰기 위해서 조건문 사용.
        {
              StringTokenizer st = new StringTokenizer(value.toString(),",");
              String Mueaure_date = st.nextToken();
              String station_code = st.nextToken();
              String item_code = st.nextToken();
              double item_value = Double.parseDouble(st.nextToken());

              if(item_code.compareTo("8")==0){
                    word.set(station_code);
                    one.set(item_value);
                    context.write(word,one);
              }
        }
        n++;
    }
}
```

WeatherCountReducer파일

이 파일에서는 Mapper파일에서 key와 value값으로 설정한 값들을 읽어와서 하나로 합치는 작업을 합니다.
우선 들어오는 key,value 타입은 각 각 text와 double형이고 결과값으로 보여질 key와 value값은 모두 text타입이기 때문에 다음과 같이 했습니다.

```
public class WeatherCountReducer extends Reducer<Text, DoubleWritable, Text, Text>
```

들어오는 KEY값이 지역코드로 되어 있기 때문에 Reducer에서는 지역 코드 별로 PM10값의 최댓값, 최솟값을 if 문을 통해서 구할 수 있고, 모든 값들을 더하여 평균도 구할수 있게 됩니다.

여기서 주의할 점은 이상한 값이 들어올때는 -1로 측정되기 때문에, 0보다 작은 값이 들어오는 경우에는 무시해주어야 합니다.

이렇게 max,min,avg값을 구했으면 key는 지역 코드, value는 평균, 최대, 최소값을 string으로 만들어서 output으로 보내주면 됩니다.

```
public class WeatherCountReducer extends Reducer<Text, DoubleWritable, Text, Text>
{

    Text a = new Text();
    @Override
    protected void reduce(Text key, Iterable<DoubleWritable> values,
                          Reducer<Text, DoubleWritable, Text, Text>.Context context)
throws IOException, InterruptedException {
        double max = 0;
        double min = 100000;
        double sum = 0;
        int n = 0;
        for(DoubleWritable v : values)
        {
            if(v.get()<0)
                continue;
            if(v.get()>max)
            {
                max = v.get();
            }
            if(v.get()<min)
            {
                min = v.get();
            }
            sum += v.get();
            n++;
        }
        double avg = sum/n;
        a.set(avg + "\t" + max + "\t" + min);
        context.write(key, a);
    }
}
```

WeatherCountTest파일

값이 제대로 잘 나오는지 test파일을 만들어서 실행해보았습니다.

실행 결과:

```
101     38.01078260869565      516.0   0.0
102     37.988529718456725     296.0   0.0
103     35.82778880199214      330.0   0.0
104     42.75509885738798      985.0   0.0
105     42.494438066584856     985.0   0.0
106     47.85673691688373      985.0   0.0
107     49.91897431930693      985.0   0.0
108     45.842131724484666     1661.0  0.0
109     39.08231214783876      329.0   0.0
110     38.152979693650856     693.0   0.0
111     45.938372857750785     3403.0  0.0
112     38.85219105031301      333.0   0.0
113     43.474774110986154     985.0   0.0
114     40.43205251979147      389.0   0.0
115     42.70005792624059      357.0   0.0
```

```
116     54.86369812492721     3586.0 0.0
117     51.60438837506288     1985.0 0.0
118     39.980223260844376    695.0  0.0
119     50.56395101788542     985.0  0.0
120     41.78763461687456     594.0  0.0
121     46.016322346633025    2172.0 0.0
122     54.086435551419264    1985.0 0.0
123     39.99357776739841     561.0  0.0
124     45.945995051801454    985.0  0.0
125     45.36289169661141     622.0  0.0
```

위와 같이 지역 코드값, 평균, 최댓값, 최솟값이 순서대로 잘 나타나는 것을 볼 수 있습니다.

문제 2) PM10, PM2.5 기준으로 공기의 질이 '좋음' 수준이 가장 많이 측정된 지역은 어디인지 찾기

WeatherCount2파일

이 파일은 WeatherCount파일과 똑같은 기능을 하기 때문에 크게 차이가 없습니다. 달라지는 부분이 있다면 hadoop에 필요한 설정값들과 결과값을 저장할 경로 등에 차이가 있습니다.

```
public static void main(String[] args) throws Exception {

        ToolRunner.run(new WeatherCount2(), args);
    }

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


        Job job = Job.getInstance(getConf());
        job.setJarByClass(WeatherCount2.class);

        job.setMapperClass(WeatherCountMapper2.class);
        job.setReducerClass(WeatherCountReducer2.class);

        job.setMapOutputKeyClass(IntWritable.class);
        job.setMapOutputValueClass(Text.class);

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

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

        job.waitForCompletion(true);

        return 0;
    }
```

WeatherCountMapper2파일

이 파일에서는 날씨정보를 담고있는 csv파일을 읽어와서, PM10과 PM2.5가 각 각, 30과 15 이하라면 key=지역 코드, value=측정 날짜로 하여 context에 써줍니다. 이때 context는

reduce에서 다시 읽게됩니다.

public class WeatherCountMapper2 extends Mapper<Object, Text, IntWritable, Text>
이 부분은 들어오는값이 Text 형식이고 내보낼 key값과 value값의 타입을 각 각, Int형과 text형임을 나타냅니다.
우선, 여기서도 csv파일 첫째줄은 건너뛰고 읽어야 하기 때문에 조건문을 넣어줬습니다. 그리고 들어오는 값을 콤마를 기준으로 split하여 값을 저장해야 하기 때문에 StrigTokenizer(value.toString(),",")을 사용하여 split 해주었습니다. 또한 PM10은 csv파일 내에서 8로 나와있고, PM2.5는 파일 내에서 9로 나와있기 때문에, 조건문을 사용하여 PM10과 PM2.5에 대해서만 값을 추출하였습니다. 또한 PM10의 아이템 값은 30이하이고, PM2.5의 아이템 값은 15이하 인것만 뽑도록 조건문을 추가했습니다.

```
public class WeatherCountMapper2 extends Mapper<Object, Text, IntWritable, Text>{

            IntWritable a = new IntWritable();
        Text b = new Text();
        int n=0;

        @Override
        protected void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            if(n>0)
            {
                    StringTokenizer st = new
StringTokenizer(value.toString(),",");
                    String Measure_date = st.nextToken();
                    int Station_code = Integer.parseInt(st.nextToken());
                    String Item_code = st.nextToken();
                    double Item_value = Double.parseDouble(st.nextToken());

                    if(Item_code.compareTo("8")==0 && Item_value<=30)
                    {
                            a.set(Station_code);
                            b.set(Measure_date);
                            context.write(a,b);
                    }
                    else if(Item_code.compareTo("9")==0 && Item_value<=15)
                    {
                            a.set(Station_code);
                            b.set(Measure_date);
                            context.write(a,b);
                    }
            }
            n++;
        }

}
```

WeatherCountReducer2파일

이 파일에서는 Mapper파일에서 key와 value값으로 설정한 값들을 읽어와서 공기 질의 수준이 '좋음'이 가장 많이 측정된 지역을 찾는 역할을 합니다.
우선 들어오는 key,value 타입은 각 각 text와 double형이고 결과값으로 보여질 key와

value값은 모두 int타입이기 때문에 다음과 같이 했습니다.

```
public class WeatherCountReducer2 extends Reducer<IntWritable, Text, IntWritable,
IntWritable>
```

우선, 측정 날짜가 한번 측정되면 array에 넣어줍니다. 그리고 같은 날짜가 한번 더 나왔다면 PM10과 PM2.5가 모두 좋음 수준이기 때문에 num변수를 증가 시켜주고 array에서 제거합니다. 이를 반복하면서 해당 지역의 특정 날짜가 PM10과 PM2.5가 모두 '좋음' 수준일때마다 num값을 증가시킵니다. 이것을 지역별로 값을 구하여 가장 큰 값을 구하여 key=지역 코드, value= '좋음'수준이 측정된 수로 하여 output으로 내보내줍니다.

```
public class WeatherCountReducer2 extends Reducer<IntWritable, Text, IntWritable,
IntWritable> {
    IntWritable a = new IntWritable();
    IntWritable b = new IntWritable();

    int k = 0;    //key값을 가지는 변수
    int v = 0;    //max값을 가지는 변수

    @Override
    protected void reduce(IntWritable key, Iterable<Text> values,
                          Context context) throws IOException, InterruptedException
{
        ArrayList<String> d= new ArrayList<String>();
        int num=0;

        for(Text t : values)
        {
            String date = t.toString();
            boolean check = d.contains(date);
            if(check==false)
            {
                d.add(date);
            }
            else
            {
                d.remove(date);
                num++;
            }
        }
        if(v<num)
        {
                k = key.get();
                v = num;
        }
        if(key.get()==125){
            a.set(k);
            b.set(v);
            context.write(a, b);
        }
    }
}
```

WeatherCountTest2파일

값이 제대로 잘 나오는지 test파일을 만들어서 실행해보았습니다.

실행 결과:
112	9872

다음과 같이 지역코드 : 112 좋음이라고 측정된 수 : 9872 라고 결과값이 잘 나옵니다.

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

WeatherCount3파일

이 파일은 WeatherCount1, WeatherCount2 파일과 똑같은 기능을 하기 때문에 크게 차이가 없습니다. 달라지는 부분이 있다면 hadoop에 필요한 설정값들과 결과값을 저장할 경로 등에 차이가 있습니다.

```
public class WeatherCount3 extends Configured implements Tool{

    public static void main(String[] args) throws Exception {
        ToolRunner.run(new WeatherCount3(), args);
    }

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


        Job job = Job.getInstance(getConf());
        job.setJarByClass(WeatherCount3.class);

        job.setMapperClass(WeatherCountMapper3.class);
        job.setReducerClass(WeatherCountReducer3.class);

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

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

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

        job.waitForCompletion(true);

        return 0;
    }

}
```

WeatherCountMapper3파일

이 파일에서는 날씨정보를 담고있는 csv파일을 읽어와서, key = 측정날짜 + 지역코드, value = 아이템 코드 + 아이템 값으로 하여 넘겨줍니다.

```
public class WeatherCountMapper3 extends Mapper<Object,Text,Text,Text>
```

이 부분은 들어오는값이 Text 형식이고 내보낼 key값과 value값도 Text형식인 것을 나타냅니다.

우선, 여기서도 csv파일 첫째줄은 건너뛰고 읽어야 하기 때문에 조건문을 넣어줬습니다. 그리고

들어오는 값을 콤마를 기준으로 split하여 값을 저장해야 하기 때문에 StrigTokenizer(value.toString(),",")을 사용하여 split 해주었습니다.
key로는 측정 날짜와 지역 코드를 콤마 묶어서 보내주고, value는 item code와 item값을 띄어쓰기 형태로 묶어서 reducer로 보내주었습니다.

```
public class WeatherCountMapper3 extends Mapper<Object,Text,Text,Text>{

            Text a = new Text();
          Text b = new Text();
          int n=0;

          @Override
          protected void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
              if(n>0)
              {
                      StringTokenizer st = new
StringTokenizer(value.toString(),",");
                      String Measure_date = st.nextToken();
                      String Station_code = st.nextToken();
                      String Item_code = st.nextToken();
                      String Item_value = st.nextToken();

                      a.set("<"+Measure_date+", "+Station_code+">");
                      b.set(Item_code+" "+Item_value);
                      context.write(a,b);
              }
              n++;
          }
}
```

WeatherCountReducer3파일

이 파일에서는 Mapper파일에서 key와 value값으로 설정한 값들을 읽어와서 key값을 기준으로 아이템 코드에 따른 아이템 값을 구하여 한줄에 다 보여주도록 해줍니다.
우선 들어오는 key,value 타입은 text형이고 결과값으로 보여질 타입도 texd형 이기 때문에 다음과 같이 했습니다.

```
public class WeatherCountReducer3  extends Reducer<Text, Text, Text, Text>
```

우선, value값을 읽어와서 StringTokenizer를 이용하여 값을 띄어쓰기 기준으로 나눕니다. 그렇게 되면 item code값과 item value값을 얻을 수 있게됩니다. 그리하여 얻은 값을 바탕으로 item code가 1,3,4,5,8,9인지 확인하여 item value값을 각 각의 변수에 저장하여 마지막에 하나의 string으로 모아줍니다.
마지막으로 key값은 mapper의 key값과 동일한 값으로 하고, value 값은 하나의 string으로 모아준 값으로 하여 output으로 내보내줍니다.

WeatherCountTest3파일

값이 제대로 잘 나오는지 test파일을 만들어서 실행해보았습니다.
실행 결과:

```
<2017-01-01 00:00, 101>      1: 0.004      3: 0.059000000000000004      5: 1.2 6:
0.002  8: 73.09: 57.0
```

```
<2017-01-01 00:00, 102>	1: 0.006	3: 0.068	5: 1.3 6: 0.002	8:
77.0	9: 63.0
<2017-01-01 00:00, 103>	1: 0.005	3: 0.039	5: 1.4 6: 0.002	8:
70.0	9: 68.0
<2017-01-01 00:00, 104>	1: 0.005	3: 0.045	5: 0.6 6: 0.003	8:
73.0	9: 46.0
<2017-01-01 00:00, 105>	1: 0.005	3: 0.044000000000000004	5: 1.0 6:
0.004	8: 81.09: 44.0
<2017-01-01 00:00, 106>	1: 0.005	3: 0.066	5: 1.5 6: 0.003	8:
71.0	9: 62.0
<2017-01-01 00:00, 107>	1: 0.005	3: 0.049	5: 0.9 6: 0.002	8:
64.0	9: 40.0
<2017-01-01 00:00, 108>	1: 0.004	3: 0.045	5: 0.8 6: 0.003	8:
68.0	9: 63.0
<2017-01-01 00:00, 109>	1: 0.006	3: 0.052000000000000005	5: 1.1 6:
0.002	8: 76.09: 50.0
<2017-01-01 00:00, 110>	1: 0.005	3: 0.045: 0.8 6: 0.002	8: 91.09:
50.0
```

이 외에도 많지만 결과값이 너무 많이 나와서 몇 개의 값만 가져왔습니다.

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

WeatherCount4파일

이 파일은 WeatherCount1, WeatherCount2, WeatherCount3 파일과 똑같은 기능을 하기 때문에 크게 차이가 없습니다. 달라지는 부분이 있다면 hadoop에 필요한 설정값들과 결과값을 저장할 경로 등에 차이가 있습니다.

```
public class WeatherCount4 extends Configured implements Tool{

    public static void main(String[] args) throws Exception {
        ToolRunner.run(new WeatherCount4(), args);
    }

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


        Job job = Job.getInstance(getConf());
        job.setJarByClass(WeatherCount4.class);

        job.setMapperClass(WeatherCountMapper4.class);
        job.setReducerClass(WeatherCountReducer4.class);

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

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

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

```
        job.waitForCompletion(true);

        return 0;
    }

}
```

WeatherCountMapper4파일

이 파일에서는 날씨정보를 담고있는 csv파일을 읽어와서 key = 측정 시간, value는 item code + item value를 합친 값으로 하여 넘겨줍니다.

```
public class WeatherCountMapper4 extends Mapper<Object,Text,Text,Text>
```

이 부분은 들어오는값이 Text 형식이고 내보낼 key값과 value값도 Text형식인 것을 나타냅니다.

우선, 여기서도 csv파일 첫째줄은 건너뛰고 읽어야 하기 때문에 조건문을 넣어줬습니다. 그리고 들어오는 값을 콤마를 기준으로 split하여 값을 저장해야 하기 때문에 StrigTokenizer(value.toString(),",")을 사용하여 split 해주었습니다.

그리고 시간을 key값으로 넘기기 위해서는 측정 시간에서 시간 값을 추출해주어야 합니다. 그렇게 하기 위해서 측정 날짜를 StringTokenizer를 이용하여 띄어쓰기 별로 나누어서 시간 값을 구해줍니다. 이렇게 하여 key로는 시간 값을 value값으로는 item code와 item value를 이어 붙인 값으로 하여 reducer로 넘겨줍니다.

```
public class WeatherCountMapper4 extends Mapper<Object,Text,Text,Text>{
            Text a = new Text();
            Text b = new Text();
            int n=0;

            @Override
            protected void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
                if(n>0)
                {
                        StringTokenizer st = new
StringTokenizer(value.toString(),",");
                        String Measure_date = st.nextToken();
                        String Station_code = st.nextToken();
                        String Item_code = st.nextToken();
                        String Item_value = st.nextToken();

                        StringTokenizer t = new
StringTokenizer(Measure_date.toString()," ");
                        String date = t.nextToken();
                        String tt = t.nextToken();

                        a.set(tt);
                        b.set(Item_code+" "+Item_value);
                        context.write(a,b);
                }
                n++;
            }
```

}

WeatherCountReducer4파일

이 파일에서는 Mapper파일에서 key와 value값으로 설정한 값들을 읽어와서 key값을 기준으로 즉, 시간대별을 기준으로 각 아이템 코드의 평균값을 구합니다.
우선 들어오는 key,value 타입은 text형이고 결과값으로 보여질 타입은 key = text형 value = double형 이기 때문에 다음과 같이 했습니다.

```
public class WeatherCountReducer4  extends Reducer<Text, Text, Text,
DoubleWritable>
```

우선, value값을 읽어와서 StringTokenizer를 이용하여 값을 띄어쓰기 기준으로 나눕니다. 그렇게 되면 item code값과 item value값을 얻을 수 있게됩니다. 그리하여 얻은 값을 바탕으로 item code가 1,3,4,5,8,9인지 확인하고 item value값을 각 각의 변수에 더하고, 그 수를 세어 각 각의 변수에 저장합니다. 이때 item value값이 -1이라면 잘못된것이기 때문에 측정값에서 제외하는 조건문을 넣었습니다.
마지막으로 아이템 코드가 1인 경우, 아이템 코드가 3인 경우, 아이템 코드가 5인 경우, 아이템 코드가 6인 경우, 아이템 코드가 8인 경우, 아이템 코드가 9인 경우 각 각에 key 값은 mapper의 key값과 동일하게 해주고 value 값은 평균 값으로 하여 output으로 내보내 줍니다.

```
public class WeatherCountReducer4  extends Reducer<Text, Text, Text,
DoubleWritable> {

	Text b = new Text();
	DoubleWritable a = new DoubleWritable();
	@Override
    protected void reduce(Text key, Iterable<Text> values,
                          Context context) throws IOException, InterruptedException
{

		double num1 = 0.0;
		double num3 = 0.0;
		double num5 = 0.0;
		double num6 = 0.0;
		double num8 = 0.0;
		double num9 = 0.0;

		int n1=0;
		int n3=0;
		int n5=0;
		int n6=0;
		int n8=0;
		int n9=0;

		for(Text t : values)
		{
			StringTokenizer st = new StringTokenizer(t.toString()," ");
			String item_code = st.nextToken();
			double item_value = Double.parseDouble(st.nextToken());
			if(item_code.compareTo("1")==0 && item_value>=0)
			{
				num1 += item_value;
				n1++;
```

```
                        }
                        if(item_code.compareTo("3")==0 && item_value>=0)
                        {
                                num3 = item_value;
                                n3++;
                        }
                        if(item_code.compareTo("5")==0 && item_value>=0)
                        {
                                num5 = item_value;
                                n5++;
                        }
                        if(item_code.compareTo("6")==0 && item_value>=0)
                        {
                                num6 = item_value;
                                n6++;
                        }
                        if(item_code.compareTo("8")==0 && item_value>=0)
                        {
                                num8 = item_value;
                                n8++;
                        }
                        if(item_code.compareTo("9")==0 && item_value>=0)
                        {
                                num9 = item_value;
                                n9++;
                        }
                }
                b.set(key +"\t"+" 1 : ");
                a.set(num1/n1);
                context.write(b, a);

                b.set(key+"\t"+ " 3 : ");
                a.set(num3/n3);
                context.write(b, a);

                b.set(key +"\t"+ " 5 : ");
                a.set(num5/n5);
                context.write(b, a);

                b.set(key +"\t"+ " 6 : ");
                a.set(num6/n6);
                context.write(b, a);

                b.set(key+"\t"+ " 8 : ");
                a.set(num8/n8);
                context.write(b, a);

                b.set(key +"\t"+ " 9 : ");
                a.set(num9/n9);
                context.write(b, a);
        }
}
```

WeatherCountTest4파일

값이 제대로 잘 나오는지 test파일을 만들어서 실행해보았습니다.

실행 결과:

```
00:00    1 :    0.004177678838671187
00:00    3 :    2.2442491116513934E-7
```

```
00:00   5 :   7.483349547257353E-6
00:00   6 :   7.859575582918523E-7
00:00   8 :   1.4959982048021541E-4
00:00   9 :   2.244081235740734E-4
01:00   1 :   0.004141655450875018
01:00   3 :   8.970956528239824E-7
01:00   5 :   1.869438420698422E-5
01:00   6 :   4.4881624714814676E-7
01:00   8 :   5.232666791253971E-4
01:00   9 :   1.1212857409829938E-4
02:00   1 :   0.004102853301464117
02:00   3 :   1.5308789485475317E-6
02:00   5 :   2.6145743846412428E-5
02:00   6 :   9.714178965066318E-7
02:00   8 :   0.0014560931899641578
02:00   9 :   0.0011574506216629951
03:00   1 :   0.004064416031105242
03:00   3 :   1.6072963779763015E-6
03:00   5 :   2.2433261048381066E-5
03:00   6 :   7.479151864178602E-8
03:00   8 :   0.001756877990430622
03:00   9 :   0.00127074301091344
04:00   1 :   0.004031226654714034
04:00   3 :   1.1949662048620189E-6
```

이 외에도 많지만 결과값이 너무 많이 나와서 몇 개의 값만 가져왔습니다

느낀점.

처음 해보는 map reduce여서 그런지 굉장히 하는데 오래 걸렸지만, 하나하나 해결해가면서 뿌듯함을 느꼈고, map reduce가 뭔지에 대해서 조금은 알게 된 것 같아서 기분이 좋습니다.