# REPORT

국민대학교
KOOKMIN UNIVERSITY


과목명 | 빅데이터최신기술

담당교수 | 박하명 교수님

학과 |

학년 |

학번 | 20181703

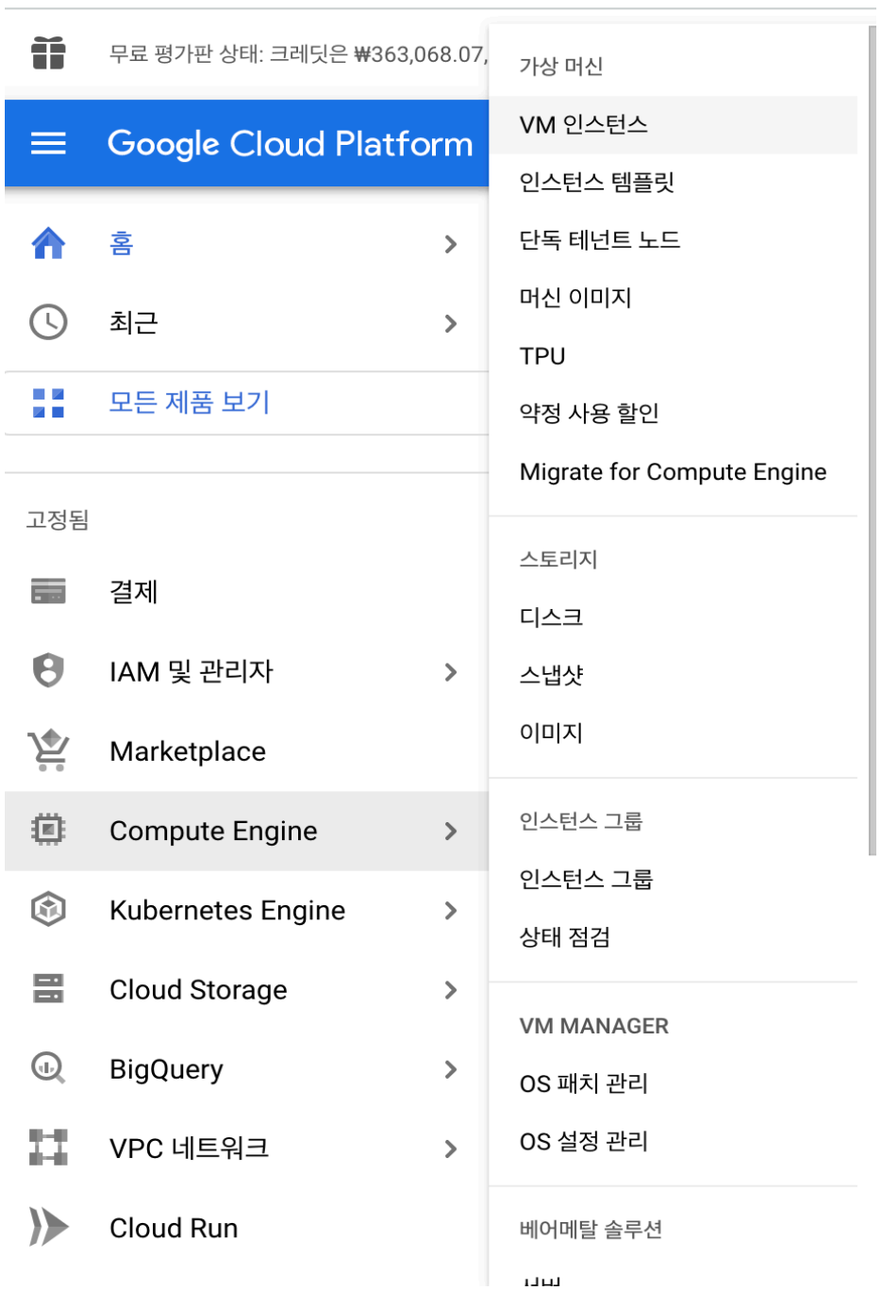
이름 | 평선호

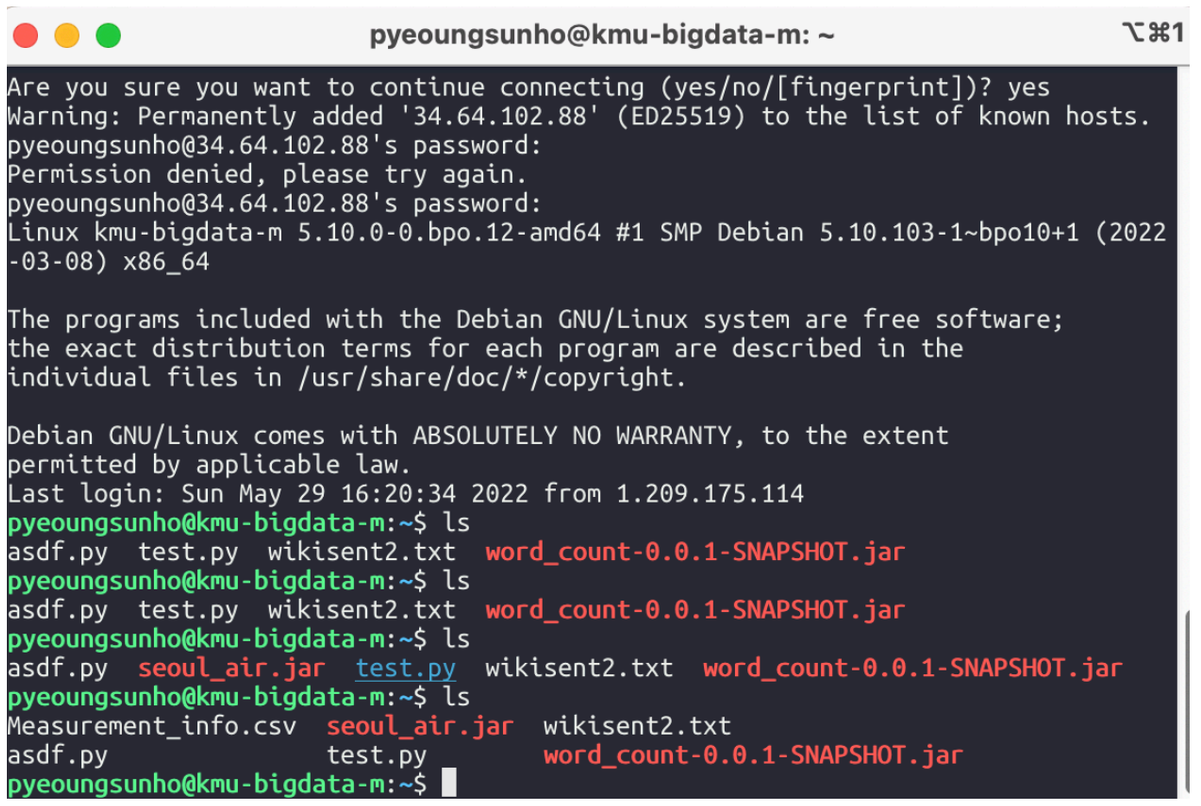
제출일 |

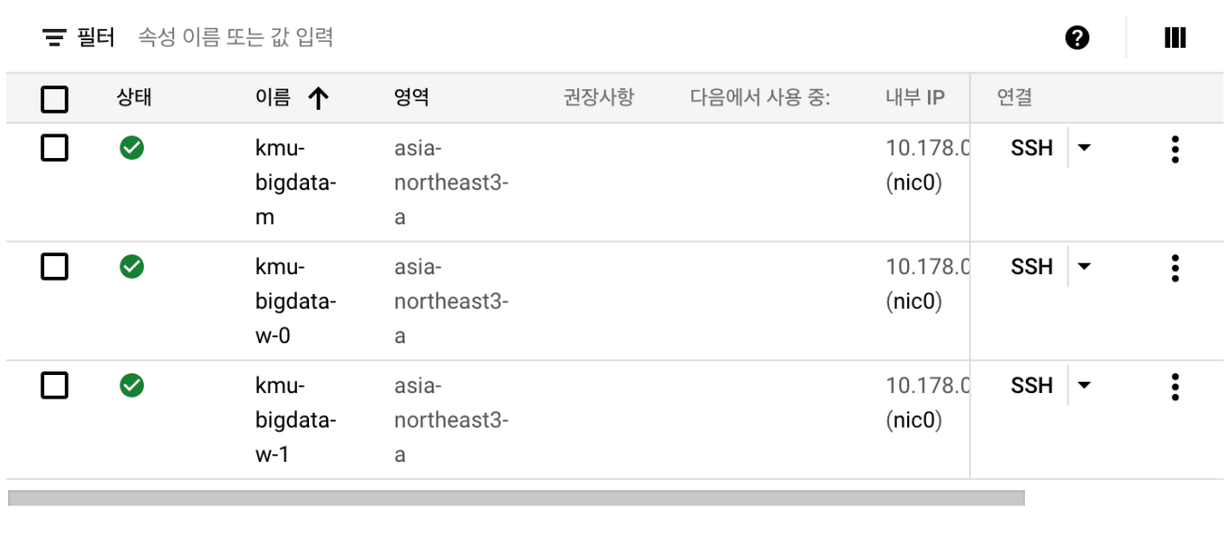
# <목차>



Google cloud platform을 이용하여 HDFS(Hadoop Distributed File System)을 사용하였습니다.
ssh명령어를 사용하여 해당 가상 머신의 인스턴스의 ip주소로 접근 하였고 scp 명령어를 통해 나의 PC에 있는 file들을 Master node로 전송하였습니다.
이후 접속후 $ >hdfs dfs -put 명령어를 통해 hdfs에 data 파일을 저장하였습니다.



# Task 1 해결 과정

-MAP

- input: < Key, Value> Type: <Object, Text>

key로 해당 파일을 읽어 들인 후 Value는 파일에서 한줄 씩 읽어온다.

이후 StringTokenizer를 통해 edge에 해당하는 node를 한개씩 읽어온다.
이때 간선 (u,v)와 (v,u)는 동일한 간선으로 생각하여 비교를 통해 값의 순서를 변경하여 reduce로 넘겨준다. 또한 loop를 제거하기 위해 u와 v를 비교해서 같으면 return처리를 진행해준다.

```java
public class Tritask1Mapper extends Mapper<Object, Text, Text, Text> {

    Text ok = new Text();
    Text ov = new Text();

    @Override
    protected void map(Object key, Text value, Mapper<Object, Text, Text, Text>.Context context)
            throws IOException, InterruptedException {

        StringTokenizer st = new StringTokenizer(value.toString());

        String u = st.nextToken();
        String v = st.nextToken();
        if (Integer.parseInt(u) == Integer.parseInt(v)) return;
        if (Integer.parseInt(u) < Integer.parseInt(v)) {
            ok.set(u+ " "+ v);
            ov.set("x");
            context.write(ok, ov);
        }
        else if (Integer.parseInt(u) > Integer.parseInt(v)) {
            ok.set(v + " " + v);
            ov.set("x");
            context.write(ok, ov);
        }

    }
}
```

# Task 1 해결 과정

-Reduce

- input: < Key, Value> Type: <Text, Text>

reduce에서 묶은 값들을 연산과정을 통해 처리합니다.

key와 value중에서 value에는 쓰레기 값이 들어있으므로 key를 Stringtokenizer를 이용하여 정제한 간선들의 값을 받아옵니다.

```
public class Tritask1Reducer extends Reducer<Text, Text, Text, Text>{
    Text ok = new Text();
    Text ov = new Text();

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


        StringTokenizer st = new StringTokenizer(key.toString());

        String u =st.nextToken();
        String v = st.nextToken();
        ok.set(u);
        ov.set(v);
        context.write(ok,ov);
    }
}
```

# Task 2 해결 과정

-MAP

- input: < Key, Value> Type: <Object, Text>

key로 해당 파일을 읽어 들인 후 Value는 파일에서 한줄 씩 읽어옵니다.

정제된 그래프에서 각각의 edge에 대해 degree를 구해야 합니다.
이때 degree를 구하는 과정 중 분산처리로 진행하기 때문에 한번에 degree를 추가하는 것이 아닌 2단계로 나눠서 map_reduce를 진행 해줍니다.
edge의 한쪽 node degree를 계산하기 위해 2번 반복하여 한번은 (u,v) 한번은 (v,u)로 값을 전달합니다.

```java
public class Tritask2_1Mapper extends Mapper<Object, Text, Text, Text> {

    Text ok = new Text();
    Text ov = new Text();

    @Override
    protected void map(Object key, Text value, Mapper<Object, Text, Text, Text>.Context context)
            throws IOException, InterruptedException {

        StringTokenizer st = new StringTokenizer(value.toString());

        String u = st.nextToken();
        String v = st.nextToken();
        for(int i =0;i <=1;++i) {
            if(i==0) {
                ok.set(u);
                ov.set(u+"\t"+v);
                context.write(ok, ov);
            }
            else {
                ok.set(v);
                ov.set(u+"\t"+v);
                context.write(ok, ov);
            }
        }
    }
}
```

# Task 2 해결 과정

## -Reduce

- input: < Key,values> Type: <Text, Iterable<Text>

  reduce에서 묶은 값들을 연산과정을 통해 처리합니다.

맨 처음엔 HashMap을 통해 들어오는 Reduce에 들어오는 value값을 저장하려 했습니다. Hashmap을 사용하여 value의 값들을 받아 온 후 value 값이 있으면 Hashmap 안의 key 의 value값을 증가시키고, 없으면 1로 추가해줄려했습니다. 그러나 해당 코드에서 문제점을 발견하였습니다. 바로 Map reduce는 한개의 key를 바탕으로 value들이 쭉 들어오는 과정에 있어서 value가 날라가는데 이를 저장을 해야 value를 사용 할 수 있었습니다.  또한 value의 형태가 Iterable한것도 있었습니다.

```java
public class Tritask2Reducer extends Reducer<Text, Text, Text, IntWritable>{
    Text ok = new Text();
    IntWritable ov = new IntWritable();

    @Override
    protected void reduce(Text key, Iterable<Text> values,
            Reducer<Text, Text, Text, IntWritable>.Context context) throws IOException, InterruptedException {


        StringTokenizer st = new StringTokenizer(values.toString());


        String u =st.nextToken();
        String v = st.nextToken();

        HashMap<String,Integer> save_degree = new HashMap<String,Integer>();
        Integer cnt = 0;
        for(Text val: values) {
            System.out.println(val.toString());
            if(save_degree.containsKey(val.toString())) {
                save_degree.replace(val.toString(),save_degree.get(val)+1);
                System.out.println("yes");
            }
            else // not contain
            {
                save_degree.put(val.toString(), 0);
                save_degree.replace(val.toString(), 1);
                System.out.println("no");
            }

        }
        ok.set(values.toString());
        ov.set(save_degree.get(u));
        context.write(ok,ov);
    }
}
```

처음 생각한 방법

따라서 해당 자료를 저장하고 탐색하고자 기존 triangle code에 있는 TriStep1Reducer.java 의 ArrayList개념을 도입하기로 했습니다.
해당 key에 대해 같이 들어오는 value값을 counting하며 key값의 degree를 구했습니다.

```java
public class Tritask2_1Reducer extends Reducer<Text, Text, Text, Text>{
    Text ok = new Text();
    Text ov = new Text();

//  HashMap<String,Integer> save_degree = new HashMap<String,Integer>();
    @Override
    protected void reduce(Text key, Iterable<Text> values,
            Reducer<Text, Text, Text, Text>.Context context) throws IOException, InterruptedException {

        int cnt = 0;

        ArrayList<String> neighbors = new ArrayList<String>();
        for(Text v : values) {
            cnt +=1 ;
            neighbors.add(v.toString());
        }

        ov.set(Integer.toString(cnt));
        for (String n: neighbors) {
            ok.set(n);
            context.write(ok, ov);
        }

    }
}
```

두번째 방법

이후 드라이버에서 Tritask2 Reducer의 결과물을 바탕으로 같은 방식으로 한번더 degree를 구해줍니다. 해당 2-1 task처리를 진행 한 후에는 degree가 있으므로 값을 3개를 받아와 이를 value로 넣어 준 후 진행합니다.

```java
public class Tritask2_2Mapper extends Mapper<Object, Text, Text, Text> {

    Text ok = new Text();
    Text ov = new Text();

    @Override
    protected void map(Object key, Text value, Mapper<Object, Text, Text, Text>.Context context)
            throws IOException, InterruptedException {

        StringTokenizer st = new StringTokenizer(value.toString());

        String u = st.nextToken();
        String v = st.nextToken();
        String degree_s = st.nextToken();


        ok.set(v);
        ov.set(u+"\t"+v + "\t" + degree_s);
        context.write(ok, ov);
    }
}
```

```java
public class Tritask2_2Reducer extends Reducer<Text, Text, Text, Text>{
    Text ok = new Text();
    Text ov = new Text();

//  HashMap<String,Integer> save_degree = new HashMap<String,Integer>();
    @Override
    protected void reduce(Text key, Iterable<Text> values,
            Reducer<Text, Text, Text, Text>.Context context) throws IOException, InterruptedException {

        int cnt = 0;

        ArrayList<String> neighbors = new ArrayList<String>();
        for(Text v : values) {
            cnt +=1 ;
            neighbors.add(v.toString());
        }

        ov.set(Integer.toString(cnt));
        for (String n: neighbors) {
            ok.set(n);
            context.write(ok, ov);
        }

    }
}
```

이후 2-1 task에서 reduce 처리를 해준 방식과 똑같이 진행하여 degree를 구해준다.  이러면 아래와 같이 해당 degree를 계산하여 결과값이 나오게 됩니다.

```
99849   100000  5
99863   100000  5
100000  1108125 5
1000001 1026895 20
1000001 1178389 20
1000001 1060102 20
1000001 1493134 20
999985  1000001 20
999988  1000001 20
999996  1000001 20
999998  1000001 20
1000001 1777925 20
1000001 1205255 20
1000001 1184471 20
1000001 1179890 20
1000001 1000004 20
1000001 1000017 20
1000001 1211433 20
1000001 1184562 20
1000001 1174785 20
999997  1000001 20
999997  1000004 23
999994  1000004 23
999995  1000004 23
999998  1000004 23
1000004 1176469 23
1000004 1179676 23
1000004 1180075 23
1000004 1023265 23
1000002 1000004 23
999985  1000004 23
```

Task2 step1 결과값 일부

```
1   99849   100000  91  6
2   99863   100000  5   6
3   99849   100000  5   6
4   99863   100000  9   6
5   999997  1000001 31  12
6   999985  1000001 24  12
7   999988  1000001 94  12
8   999998  1000001 26  12
9   999996  1000001 14  12
0   999988  1000001 20  12
1   999997  1000001 20  12
2   999996  1000001 20  12
3   999985  1000001 20  12
4   999998  1000001 20  12
5   999994  1000004 23  18
6   999995  1000004 23  18
7   999998  1000004 23  18
8   999997  1000004 23  18
9   1000001 1000004 23  18
0   999985  1000004 23  18
1   1000001 1000004 20  18
2   999988  1000004 23  18
```

Task2 step2 결과값 일부

# Task 2_3 해결 과정

-MAP

- input: < Key, Value> Type: <Object, Text>

  key로 해당 파일을 읽어 들인 후 Value는 파일에서 한줄 씩 읽어온다.

Node 2 개, 해당 node에 대한 degree 2개가 key value로 들어왔기때문에 Stringtokenizer를 4개를  만들어 해당 값을 읽어온다. 이후 과제에 맞는 조건문을 만들어 u,v 값을 정상적으로 reduce로 넘긴다.

```
    @Override
    protected void map(Object key, Text value, Mapper<Object, Text, Text, Text>.Context context)
            throws IOException, InterruptedException {

        StringTokenizer st = new StringTokenizer(value.toString());

        int u = Integer.parseInt(st.nextToken()); //id u
        int v = Integer.parseInt(st.nextToken()); //id v
        int s = Integer.parseInt(st.nextToken()); //degree s
        int t = Integer.parseInt(st.nextToken()); //degree v

        if (s<t) { //degree s < degree v
            ok.set(u + "\t" + v);
            ov.set("x");
            context.write(ok, ov);
        }
        else if (s>t) { //degree s > degree v
            ok.set(u + "\t" + v);
            ov.set("x");
            context.write(ok, ov);
        }
        else if (s==t) { // node num
            if (u < v) {
                ok.set(u + "\t" + v);
                ov.set("x");
                context.write(ok, ov);
            }
            else if (u > v) {
                ok.set(v + "\t" + u);
                ov.set("x");
                context.write(ok, ov);
            }
        }

    }
}
```

# Task 2_3 해결 과정

## -Reduce

- input: < Key,values, > Type: <Text, Text >

  map에서 받은 값들을 emit합니다.

```
package triangle.opt;

import java.io.IOException;

public class Tritask3Reducer extends Reducer<Text, Text, Text, Text>{
    Text ok = new Text();
    Text ov = new Text();

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

            ok.set(key);
            context.write(ok, ov);


    }
}
```

# 해당 Task 1, 2 결과

```
0   0
1   2
1   598775
10  1101387
100 433392
1000    1291934
100000  1108125
1000000 1000056
1000000 1184562
1000000 1184622
1000001 1026895
1000001 1060102
1000001 1178389
1000001 1493134
1000002 1000004
1000002 1000034
1000002 1179326
1000002 1179455
1000003 1029351
1000003 1056099
1000003 1175727
1000003 1179153
1000003 1179498
1000003 1179645
1000003 1184409
1000003 1186236
1000003 1186332
1000003 1187037
1000003 1599552
1000004 1023265
1000004 1176469
1000004 1179676
1000004 1180075
1000005 1062911
1000005 1178165
1000005 1184090
1000006 1178837
1000006 1184549
1000006 1212527
1000009 1000010
1000009 1179326
1000009 1179701
1000009 1179890
1000009 1185131
```

Task 1 결과값 일부

```
99849   100000  5
99863   100000  5
100000  1108125 5
1000001 1026895 20
1000001 1178389 20
1000001 1060102 20
1000001 1493134 20
999985  1000001 20
999988  1000001 20
999996  1000001 20
999998  1000001 20
1000001 1777925 20
1000001 1205255 20
1000001 1184471 20
1000001 1179890 20
1000001 1000004 20
1000001 1000017 20
1000001 1211433 20
1000001 1184562 20
1000001 1174785 20
999997  1000001 20
999997  1000004 23
999994  1000004 23
999995  1000004 23
999998  1000004 23
1000004 1176469 23
1000004 1179676 23
1000004 1180075 23
1000004 1023265 23
1000002 1000004 23
999985  1000004 23
```

Task2 step1 결과값 일부

```
99849   100000  91  6
99863   100000  5   6
99849   100000  5   6
99863   100000  9   6
999997  1000001 31  12
999985  1000001 24  12
999988  1000001 94  12
999998  1000001 26  12
999996  1000001 14  12
999988  1000001 20  12
999997  1000001 20  12
999996  1000001 20  12
999985  1000001 20  12
999998  1000001 20  12
999994  1000004 23  18
999995  1000004 23  18
999998  1000004 23  18
999997  1000004 23  18
1000001 1000004 23  18
999985  1000004 23  18
1000001 1000004 20  18
999988  1000004 23  18
```

Task2 step2 결과값 일부

```
1000002 1179326
1000002 1179455
1000003 1029351
1000003 1056099
1000003 1175727
1000003 1179153
1000003 1179498
1000003 1179645
1000003 1184409
1000003 1186236
1000003 1186332
1000003 1187037
1000003 1599552
1000004 1000014
1000004 1178802
1000004 1185030
1000004 1599771
1000005 1064173
1000005 1169041
1000005 1174783
1000006 1178837
1000006 1184549
1000006 1212527
1000009 1000010
1000009 1179326
1000009 1179701
1000009 1179890
1000009 1185131
1000009 1199402
1000009 1205255
```

Task2-3 결과값 일부

# Task 2_3 해결 과정

## -Reduce

- input: < Key,values, > Type: <Text, Text >

  map에서 받은 값들을 emit합니다.

```
package triangle.opt;

import java.io.IOException;

public class Tritask3Reducer extends Reducer<Text, Text, Text, Text>{
    Text ok = new Text();
    Text ov = new Text();

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

            ok.set(key);
            context.write(ok, ov);


    }
}
```

# TriCnt 드라이버

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

        String inputpath = args[0];
        System.out.println(inputpath);
        String tmppath = inputpath + ".tmp";
        String outpath = inputpath + ".out";

        String tmppath1 = inputpath + ".tmp1";
        String outpath1 = inputpath + ".out1";

        String tmppath2_1 = inputpath + ".tmp2_1";
        String outpath2_1 = inputpath + ".out1";

        String tmppath2_2 = inputpath + ".tmp2_2";
        String outpath2_2 = inputpath + ".out1";

        String tmppath3 = inputpath + ".tmp3";
        String outpath3 = inputpath + ".out1";

        runtask1(inputpath, tmppath1);
        runtask2_1(tmppath1, tmppath2_1);
        runtask2_2(tmppath2_1, tmppath2_2);
        runtask3(tmppath2_2,tmppath3);

        runStep1(tmppath1, tmppath);
        runStep2(tmppath1, tmppath, outpath);
```

```
  private void runtask1(String inputpath, String tmppath) throws Exception{

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

      job.setMapperClass(Tritask1Mapper.class);
      job.setReducerClass(Tritask1Reducer.class);

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

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

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

      //SequenceFileOutputForma
      FileInputFormat.addInputPath(job, new Path(inputpath));
      FileOutputFormat.setOutputPath(job, new Path(tmppath));

      job.waitForCompletion(true);

  }
```

runtask중 일부

TriCnt파일 안에서 task들을 순차적으로 진행하기 위해 첫번째 file의 결과값을 input으로 받고 다시 ouput으로 내보는 file들은 또 다른 file의 input으로 받는 작업을 진행하여 file들이 연속적으로 실행되게 만들었습니다.

# Hadoop 환경에서
# map_reduce 구현 과정

```
sunho99@pyeongseonhoui-MacBookPro ~/eclipse-workspace/wiki-topic  test  s
cp -p 22 src/test/resources/wiki-topcats.txt  pyeoungsunho@34.64.139.79:~/
pyeoungsunho@34.64.139.79's password:
```

```
✘ sunho99@pyeongseonhoui-MacBookPro ~/eclipse-workspace/wiki-topic  test
 scp -p 22 target/triangle-0.1.jar pyeoungsunho@34.64.139.79:~/
pyeoungsunho@34.64.139.79's password:
```

먼저 local file에 있는 txt file과 maven을 통해 jar 파일로
압축된 triangle-0.1.jar 을 scp 명령어를 통해 해당 port번호와 ip번호를
가지고 있는 서버로 보냅니다.

```
pyeoungsunho@kmu-bigdata-m:~$ hdfs dfs -put wiki-topcats.txt wiki-topcats.txt
pyeoungsunho@kmu-bigdata-m:~$ hdfs dfs -put triangle-0.1.jar triangle-0.1.jar
```

이후 server에 있는 file들을 Hadoop file system인 hdfs에 put을 통해
올려줍니다.

```
pyeoungsunho@kmu-bigdata-m:~$ hadoop jar triangle-0.1.jar triangle.opt.TriCnt -D
mapreduce.job.reduces=3 wiki-topcats.txt
```

이후 Hadoop환경에서 Map reduce를 진행합니다.

# Hadoop 환경에서
# map_reduce 결과물 도출

```
pyeoungsunho@kmu-bigdata-m:~$ hdfs dfs -ls wiki-topcats.txt.tmp
Found 4 items
-rw-r--r--   2 pyeoungsunho hadoop          0 2022-06-20 10:20 wiki-topcats.txt.
tmp/_SUCCESS
-rw-r--r--   2 pyeoungsunho hadoop 2177611327 2022-06-20 10:19 wiki-topcats.txt.
tmp/part-r-00000
-rw-r--r--   2 pyeoungsunho hadoop 2164375502 2022-06-20 10:20 wiki-topcats.txt.
tmp/part-r-00001
-rw-r--r--   2 pyeoungsunho hadoop 2263859358 2022-06-20 10:20 wiki-topcats.txt.
tmp/part-r-00002
```

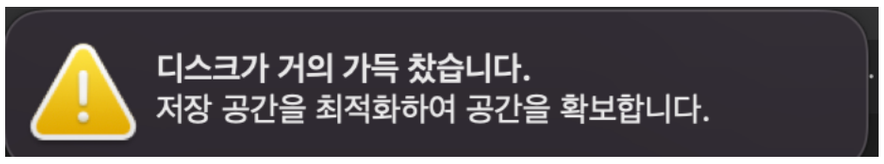
```
pyeoungsunho@kmu-bigdata-m:~$ hdfs dfs -ls wiki-topcats.txt.tmp1
Found 4 items
-rw-r--r--   2 pyeoungsunho hadoop          0 2022-06-20 10:58 wiki-topcats.txt.
tmp1/_SUCCESS
-rw-r--r--   2 pyeoungsunho hadoop   75011087 2022-06-20 10:57 wiki-topcats.txt.
tmp1/part-r-00000
-rw-r--r--   2 pyeoungsunho hadoop   75032028 2022-06-20 10:58 wiki-topcats.txt.
tmp1/part-r-00001
-rw-r--r--   2 pyeoungsunho hadoop   75012407 2022-06-20 10:58 wiki-topcats.txt.
tmp1/part-r-00002
```

```
pyeoungsunho@kmu-bigdata-m:~$ hdfs dfs -ls wiki-topcats.txt.tmp2_1
Found 4 items
-rw-r--r--   2 pyeoungsunho hadoop          0 2022-06-20 10:13 wiki-topcats.txt.
tmp2_1/_SUCCESS
-rw-r--r--   2 pyeoungsunho hadoop  181799133 2022-06-20 10:13 wiki-topcats.txt.
tmp2_1/part-r-00000
-rw-r--r--   2 pyeoungsunho hadoop  184165181 2022-06-20 10:13 wiki-topcats.txt.
tmp2_1/part-r-00001
-rw-r--r--   2 pyeoungsunho hadoop  182560465 2022-06-20 10:13 wiki-topcats.txt.
tmp2_1/part-r-00002
```

```
pyeoungsunho@kmu-bigdata-m:~$ hdfs dfs -ls wiki-topcats.txt.tmp3
Found 4 items
-rw-r--r--   2 pyeoungsunho hadoop          0 2022-06-20 10:17 wiki-topcats.txt.
tmp3/_SUCCESS
-rw-r--r--   2 pyeoungsunho hadoop   79542323 2022-06-20 10:17 wiki-topcats.txt.
tmp3/part-r-00000
-rw-r--r--   2 pyeoungsunho hadoop   79566772 2022-06-20 10:17 wiki-topcats.txt.
tmp3/part-r-00001
-rw-r--r--   2 pyeoungsunho hadoop   79637064 2022-06-20 10:17 wiki-topcats.txt.
tmp3/part-r-00002
```

```
pyeoungsunho@kmu-bigdata-m:~$ hdfs dfs -ls wiki-topcats.txt.tmp2_2
Found 4 items
-rw-r--r--   2 pyeoungsunho hadoop          0 2022-06-20 10:15 wiki-topcats.txt.
tmp2_2/_SUCCESS
-rw-r--r--   2 pyeoungsunho hadoop  217134364 2022-06-20 10:15 wiki-topcats.txt.
tmp2_2/part-r-00000
-rw-r--r--   2 pyeoungsunho hadoop  222404163 2022-06-20 10:15 wiki-topcats.txt.
tmp2_2/part-r-00001
-rw-r--r--   2 pyeoungsunho hadoop  219014430 2022-06-20 10:15 wiki-topcats.txt.
tmp2_2/part-r-00002
```

# Task 3  삼각형 MapReduce 알고리즘에 1) Task 1의 결과 그래프와 2) Task 2의 결과 그래프를 각각 입력했을 때, 성능 비교하기




해당 task들을 진행하면서 먼저 local에서 task1의 결과 값을 바탕으로 삼각형 알고리즘에 넣었을 때 메모리가 초과가 되면서 진행이 멈췄습니다. 이에 대해서 생각을 해본 결과 약 900MB정도의 데이터를 바탕으로 데이터를 처리하는데 해당 Data block 들을 Map Phase 단계에서 Mapper가 data를 shuffle하고 Reduce로 넘겨주는데 해당 과정에서 너무 많은 data가 처리가 되어 메모리가 오버가  발생하지 않았나 싶습니다. 또한 reducer 단계에서 이중 for문을 도는데 이 또한 원인 중 하나 일거 같습니다. 또한 text 파일을 입력으로 받는데, 이때 binary 파일로 입력을 받았으면 parsing이 필요없어 파일 크기가 줄어 들었을 거 같습니다.

그리고 task1과정을 거친 후 task2를 진행 하고난 후 결과값을 보니 삼각형의 개수가 기존 wiki-topcat 데이터의 삼각형 개수와 맞지가 않았습니다. 실제 삼각형의 갯수보다 코드를 통해 나온 결과값이 더 부족하였습니다. 이 부분에 있어서 생각을 해보니 기존 코드에 있는 IntPairWritable을 사용하지 않을것이 문제였던거 같습니다. 아직까지 정확하게 이유를 찾지 못하여 종강후에도 Map-reduce를 공부하여 좀 더 Map reduce에 대한 전반적인 이해를 높이고 싶습니다.

# YARN ResourceManager

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| application_1655716632054_0005 | pyeoungsunho | triangle-0.1.jar | MAPREDUCE | default | 0 | Mon Jun 20 19:17:51 +0900 2022 | Mon Jun 20 19:17:56 +0900 2022 | Mon Jun 20 19:21:01 +0900 2022 | FINISHED | SUCCEEDED | N/A | N/A | N/A | N/A | N/A | N/A | N/A | 0.0 | 0.0 | History | 0 |
| application_1655716632054_0004 | pyeoungsunho | triangle-0.1.jar | MAPREDUCE | default | 0 | Mon Jun 20 19:15:49 +0900 2022 | Mon Jun 20 19:15:54 +0900 2022 | Mon Jun 20 19:17:50 +0900 2022 | FINISHED | SUCCEEDED | N/A | N/A | N/A | N/A | N/A | N/A | N/A | 0.0 | 0.0 | History | 0 |
| application_1655716632054_0003 | pyeoungsunho | triangle-0.1.jar | MAPREDUCE | default | 0 | Mon Jun 20 19:13:47 +0900 2022 | Mon Jun 20 19:13:52 +0900 2022 | Mon Jun 20 19:15:47 +0900 2022 | FINISHED | SUCCEEDED | N/A | N/A | N/A | N/A | N/A | N/A | N/A | 0.0 | 0.0 | History | 0 |
| application_1655716632054_0002 | pyeoungsunho | triangle-0.1.jar | MAPREDUCE | default | 0 | Mon Jun 20 19:11:51 +0900 2022 | Mon Jun 20 19:11:55 +0900 2022 | Mon Jun 20 19:13:45 +0900 2022 | FINISHED | SUCCEEDED | N/A | N/A | N/A | N/A | N/A | N/A | N/A | 0.0 | 0.0 | History | 0 |
| application_1655716632054_0001 | pyeoungsunho | triangle-0.1.jar | MAPREDUCE | default | 0 | Mon Jun 20 19:09:59 +0900 2022 | Mon Jun 20 19:10:01 +0900 2022 | Mon Jun 20 19:11:49 +0900 2022 | FINISHED | SUCCEEDED | N/A | N/A | N/A | N/A | N/A | N/A | N/A | 0.0 | 0.0 | History | 0 |

이후 GCP에서 dataproc을 들어간 후 웹 인터페이스에 있는
YARN ResourceManager를 들어가면 Hadoop을 통한 분산 시스템 처리를
볼 수 있습니다.