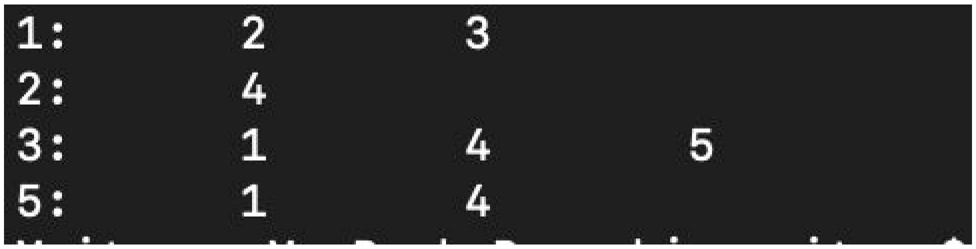
CS549 Assignment - 5

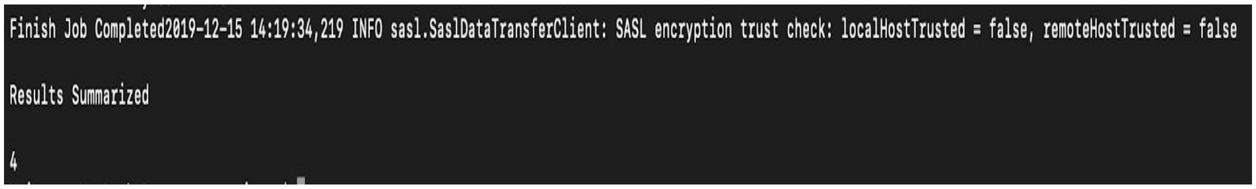
Maitrey Prajapati
10445262

Input Graph:



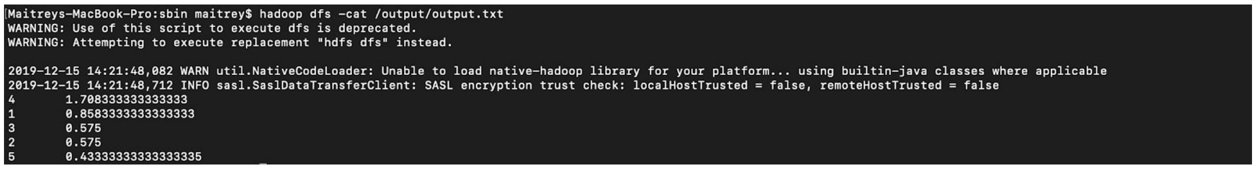
```
1:      2       3
2:      4
3:      1       4       5
5:      1       4
```

Output:



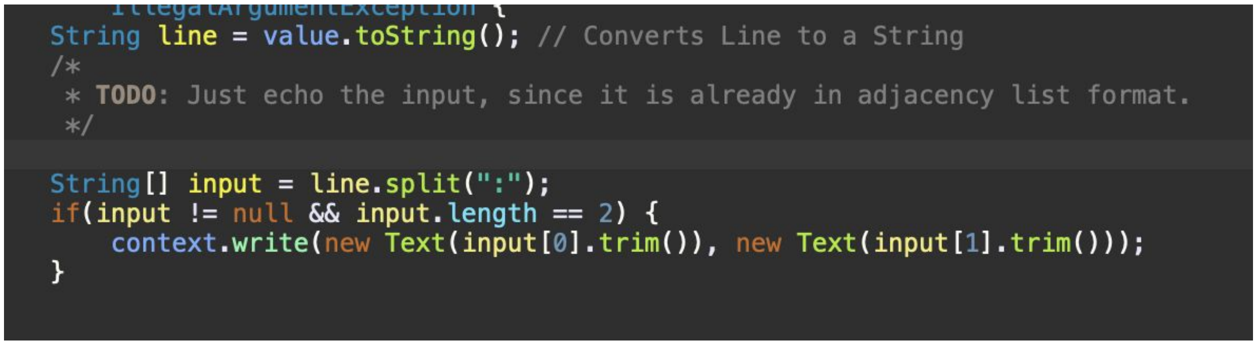
```
Finish Job Completed2019-12-15 14:19:34,219 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false

Results Summarized

4
```

Output.txt:



```
Maitreys-MacBook-Pro:sbin maitrey$ hadoop dfs -cat /output/output.txt
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.

2019-12-15 14:21:48,082 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2019-12-15 14:21:48,712 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
4       1.708333333333333
1       0.8583333333333333
3       0.575
2       0.575
5       0.43333333333333335
```

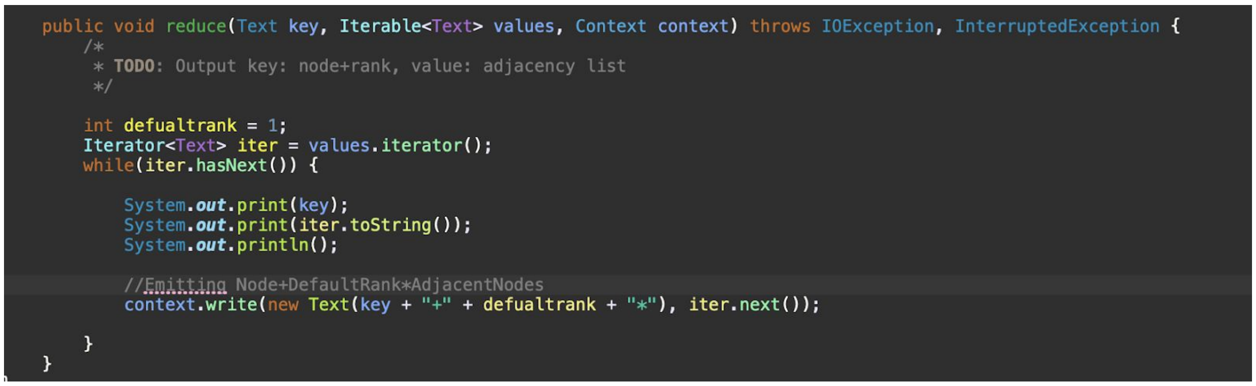
Init Mapper :



```
        IllegalArgumentException {
    String line = value.toString(); // Converts Line to a String
    /*
     * TODO: Just echo the input, since it is already in adjacency list format.
     */

    String[] input = line.split(":");
    if(input != null && input.length == 2) {
        context.write(new Text(input[0].trim()), new Text(input[1].trim()));
    }
```

Splitting the given input by colon, so input[0] will be node and input[1] will be nodes where the node is pointing towards

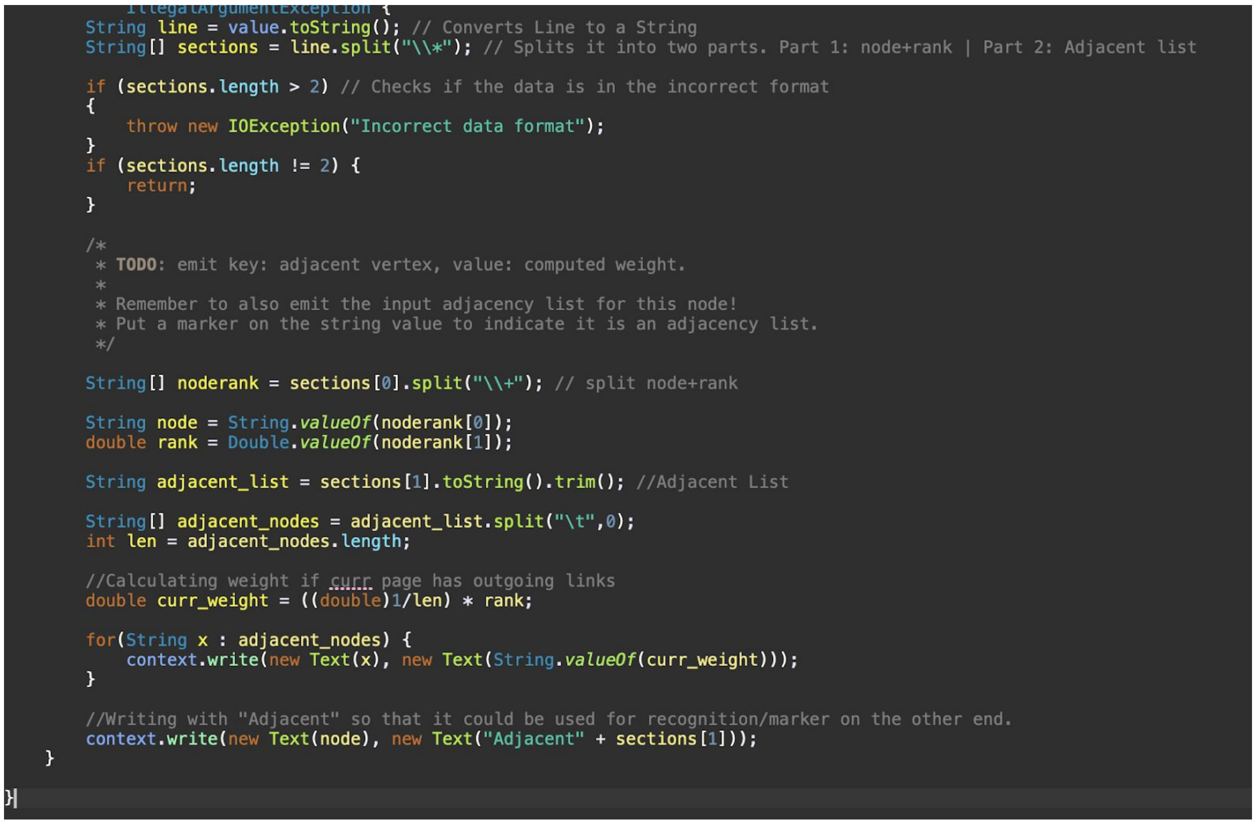
Init Reducer:



```
public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
    /*
     * TODO: Output key: node+rank, value: adjacency list
     */

    int defualtrank = 1;
    Iterator<Text> iter = values.iterator();
    while(iter.hasNext()) {

        System.out.print(key);
        System.out.print(iter.toString());
        System.out.println();

        //Emitting Node+DefaultRank*AdjacentNodes
        context.write(new Text(key + "+" + defualtrank + "*"), iter.next());

    }
}
```

Input in format of key and value(iterator form)
Emitting the output in form of (Key+DefaultRank*Adjacency list)

Iter Mapper:

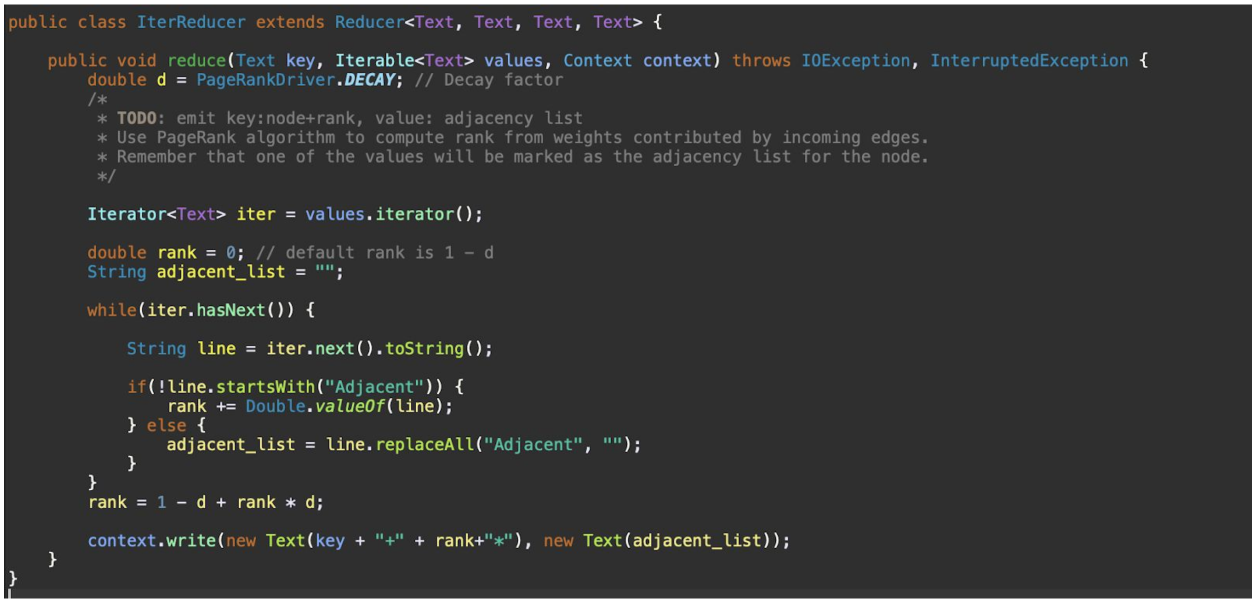


```
            IllegalArgumentException {
        String line = value.toString(); // Converts Line to a String
        String[] sections = line.split("\\*"); // Splits it into two parts. Part 1: node+rank | Part 2: Adjacent list

        if (sections.length > 2) // Checks if the data is in the incorrect format
        {
            throw new IOException("Incorrect data format");
        }
        if (sections.length != 2) {
            return;
        }

        /*
         * TODO: emit key: adjacent vertex, value: computed weight.
         *
         * Remember to also emit the input adjacency list for this node!
         * Put a marker on the string value to indicate it is an adjacency list.
         */

        String[] noderank = sections[0].split("\\+"); // split node+rank

        String node = String.valueOf(noderank[0]);
        double rank = Double.valueOf(noderank[1]);

        String adjacent_list = sections[1].toString().trim(); //Adjacent List

        String[] adjacent_nodes = adjacent_list.split("\t",0);
        int len = adjacent_nodes.length;

        //Calculating weight if curr page has outgoing links
        double curr_weight = ((double)1/len) * rank;

        for(String x : adjacent_nodes) {
            context.write(new Text(x), new Text(String.valueOf(curr_weight)));
        }

        //Writing with "Adjacent" so that it could be used for recognition/marker on the other end.
        context.write(new Text(node), new Text("Adjacent" + sections[1]));
    }

}
```

Splitting the input by (*) sign. The first part will be key-rank and second part will be Adjacency nodes

Output of this mapper is node and the list of adjacency nodes but we add the word “Adjacent” to use it as a marker to identify the list as list of adjacent nodes in reducer

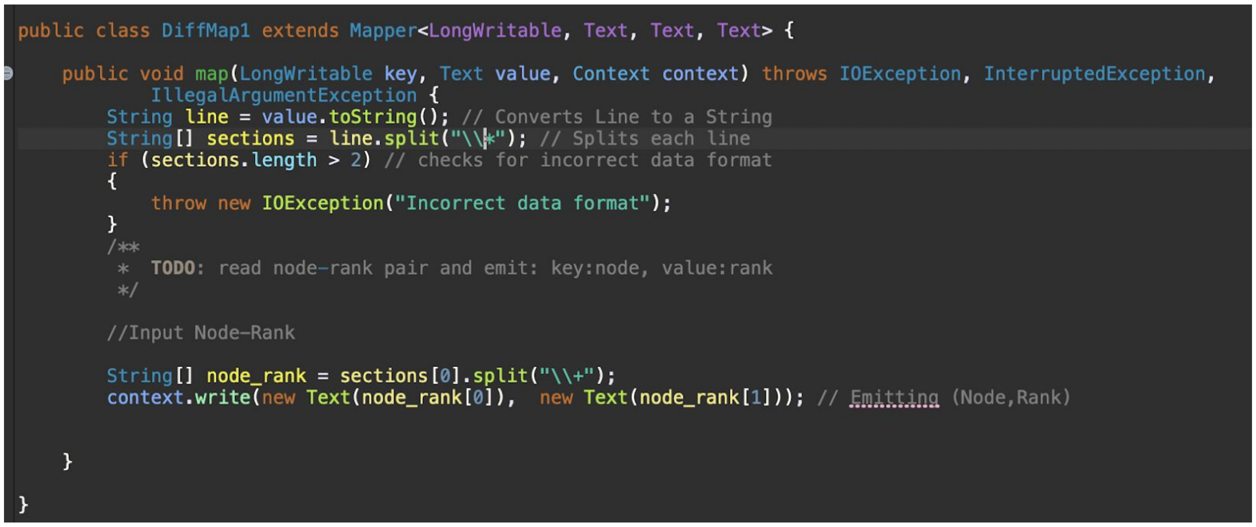
Iter Reducer:



```java
public class IterReducer extends Reducer<Text, Text, Text, Text> {

    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        double d = PageRankDriver.DECAY; // Decay factor
        /*
         * TODO: emit key:node+rank, value: adjacency list
         * Use PageRank algorithm to compute rank from weights contributed by incoming edges.
         * Remember that one of the values will be marked as the adjacency list for the node.
         */

        Iterator<Text> iter = values.iterator();

        double rank = 0; // default rank is 1 - d
        String adjacent_list = "";

        while(iter.hasNext()) {

            String line = iter.next().toString();

            if(!line.startsWith("Adjacent")) {
                rank += Double.valueOf(line);
            } else {
                adjacent_list = line.replaceAll("Adjacent", "");
            }
        }
        rank = 1 - d + rank * d;

        context.write(new Text(key + "+" + rank+"*"), new Text(adjacent_list));
    }
}
```

Output of the reducer : Key “+” Rank “*” Adjacent nodes

DiffMap1:



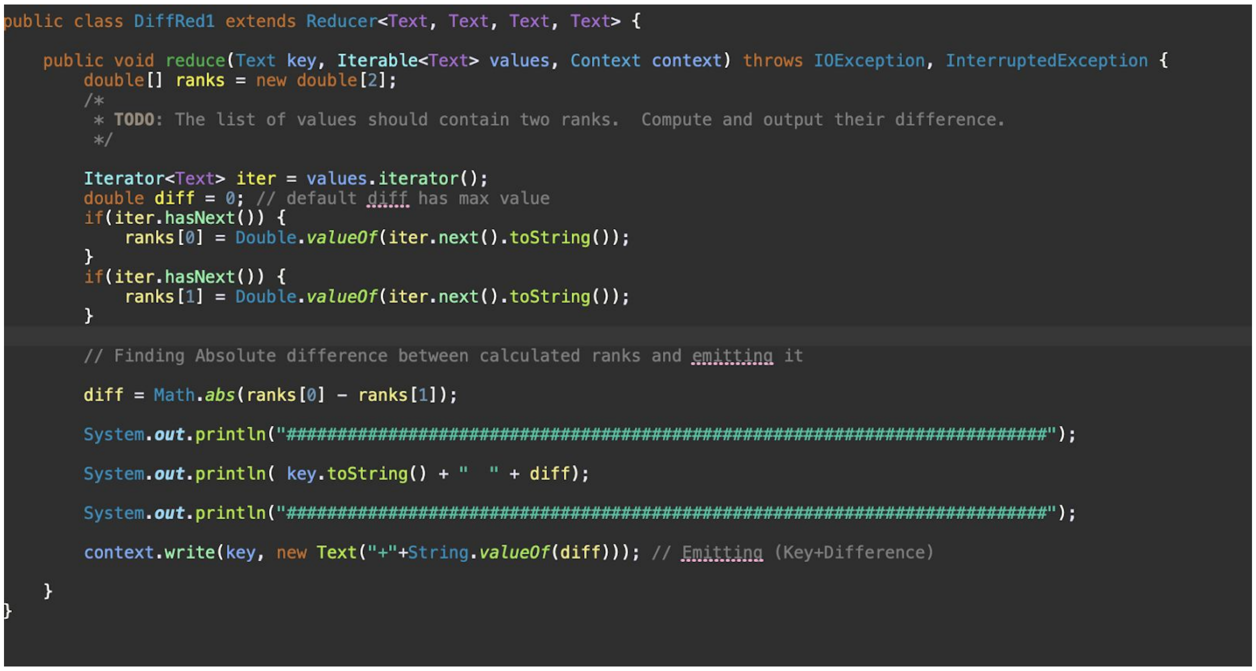
```java
public class DiffMap1 extends Mapper<LongWritable, Text, Text, Text> {

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException,
            IllegalArgumentException {
        String line = value.toString(); // Converts Line to a String
        String[] sections = line.split("\\*"); // Splits each line
        if (sections.length > 2) // checks for incorrect data format
        {
            throw new IOException("Incorrect data format");
        }
        /**
         *  TODO: read node-rank pair and emit: key:node, value:rank
         */

        //Input Node-Rank

        String[] node_rank = sections[0].split("\\+");
        context.write(new Text(node_rank[0]),  new Text(node_rank[1])); // Emitting (Node,Rank)


    }
}
```

Input = Node+Rank*AdjacentNodes

Split it by * and then + and then emit (Node,Rank)

Output = Node,Rank

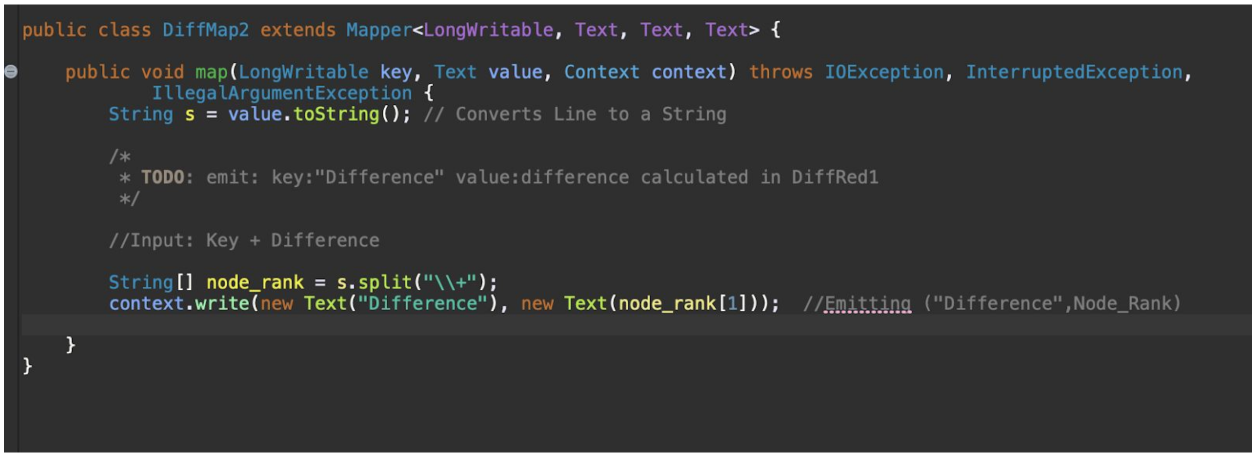
DiffRed1:



```java
public class DiffRed1 extends Reducer<Text, Text, Text, Text> {

    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        double[] ranks = new double[2];
        /*
         * TODO: The list of values should contain two ranks.  Compute and output their difference.
         */

        Iterator<Text> iter = values.iterator();
        double diff = 0; // default diff has max value
        if(iter.hasNext()) {
            ranks[0] = Double.valueOf(iter.next().toString());
        }
        if(iter.hasNext()) {
            ranks[1] = Double.valueOf(iter.next().toString());
        }

        // Finding Absolute difference between calculated ranks and emitting it

        diff = Math.abs(ranks[0] - ranks[1]);

        System.out.println("###########################################################################");

        System.out.println( key.toString() + "  " + diff);

        System.out.println("###########################################################################");

        context.write(key, new Text("+"+String.valueOf(diff))); // Emitting (Key+Difference)

    }
}
```

Calculating difference for the key and emitting it in form of (Key+”+”+Difference)

DiffMap2:

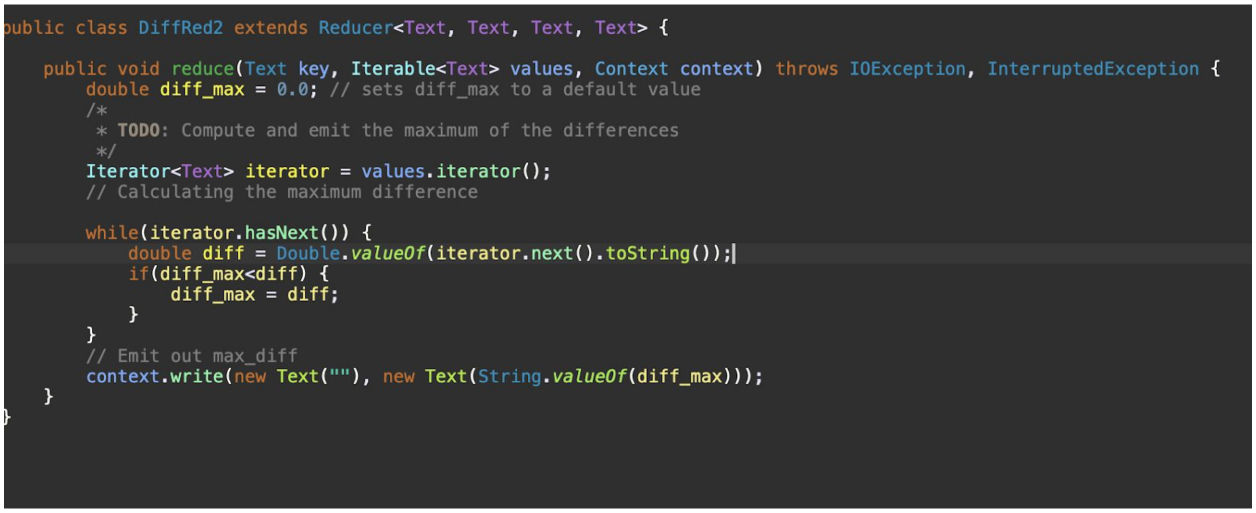


```java
public class DiffMap2 extends Mapper<LongWritable, Text, Text, Text> {

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException,
            IllegalArgumentException {
        String s = value.toString(); // Converts Line to a String

        /*
         * TODO: emit: key:"Difference" value:difference calculated in DiffRed1
         */

        //Input: Key + Difference

        String[] node_rank = s.split("\\+");
        context.write(new Text("Difference"), new Text(node_rank[1]));  //Emitting ("Difference",Node_Rank)

    }
}
```

Splitting the Key and difference by “+” sign.

Just outputting the difference in form of (“Difference”,difference) where “Difference” is the key and difference is the actual difference and value

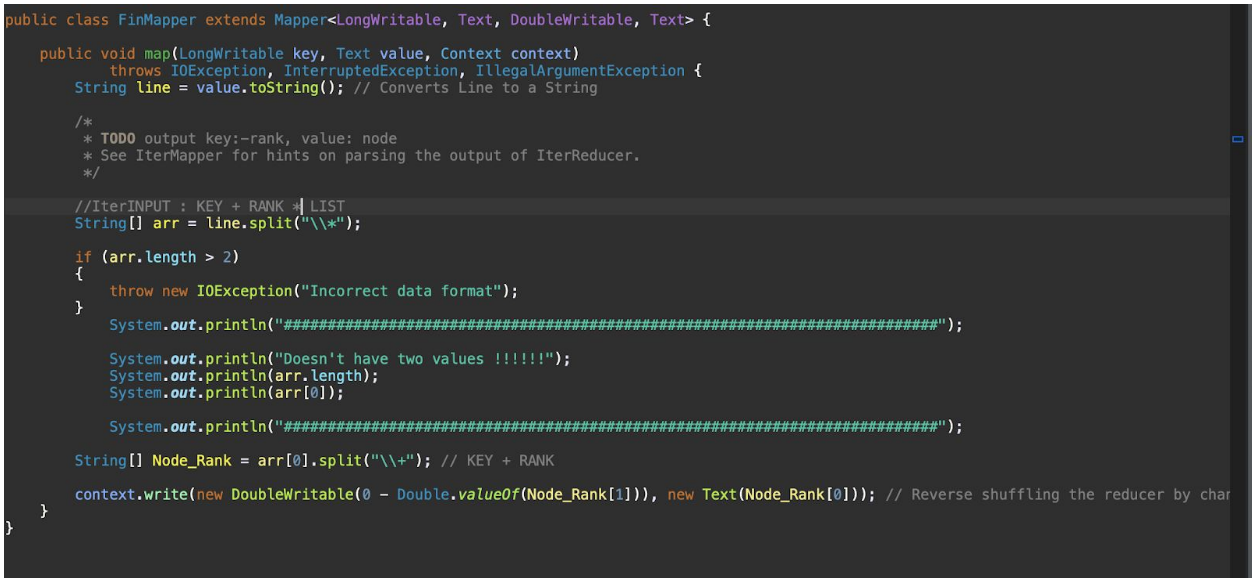
DiffRed2:



```
ublic class DiffRed2 extends Reducer<Text, Text, Text, Text> {

    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        double diff_max = 0.0; // sets diff_max to a default value
        /*
         * TODO: Compute and emit the maximum of the differences
         */
        Iterator<Text> iterator = values.iterator();
        // Calculating the maximum difference

        while(iterator.hasNext()) {
            double diff = Double.valueOf(iterator.next().toString());
            if(diff_max<diff) {
                diff_max = diff;
            }
        }
        // Emit out max_diff
        context.write(new Text(""), new Text(String.valueOf(diff_max)));
    }
}
```

Iterating over the differences and emitting out the max difference with empty key and max difference as value

FinMapper:

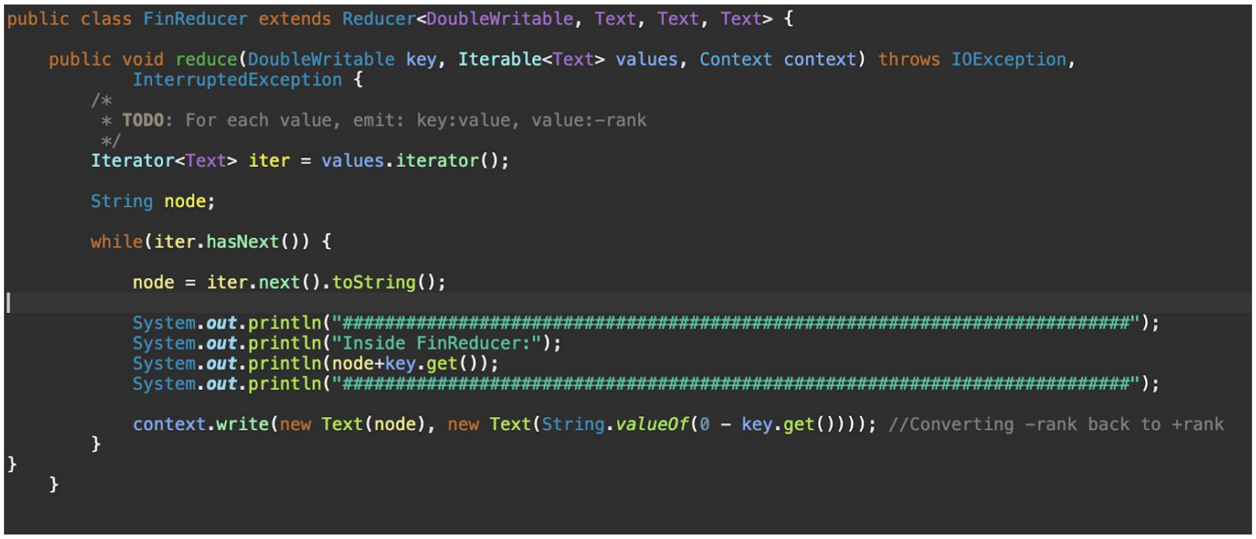


```java
public class FinMapper extends Mapper<LongWritable, Text, DoubleWritable, Text> {

    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException, IllegalArgumentException {
        String line = value.toString(); // Converts Line to a String

        /*
         * TODO output key:-rank, value: node
         * See IterMapper for hints on parsing the output of IterReducer.
         */

        //IterINPUT : KEY + RANK * LIST
        String[] arr = line.split("\\*");

        if (arr.length > 2)
        {
            throw new IOException("Incorrect data format");
        }
            System.out.println("###########################################################################");

            System.out.println("Doesn't have two values !!!!!!");
            System.out.println(arr.length);
            System.out.println(arr[0]);

            System.out.println("###########################################################################");

        String[] Node_Rank = arr[0].split("\\+"); // KEY + RANK

        context.write(new DoubleWritable(0 - Double.valueOf(Node_Rank[1])), new Text(Node_Rank[0])); // Reverse shuffling the reducer by char
    }
}
```

Input in form of key+rank*nodes and then splitting it with "*" first and then splitting it by "+" to get node as argument 0 and rank as argument 1

Outputting it in form of (-rank,node)

FinReducer:



```java
public class FinReducer extends Reducer<DoubleWritable, Text, Text, Text> {

    public void reduce(DoubleWritable key, Iterable<Text> values, Context context) throws IOException,
            InterruptedException {
        /*
         * TODO: For each value, emit: key:value, value:-rank
         */
        Iterator<Text> iter = values.iterator();

        String node;

        while(iter.hasNext()) {

            node = iter.next().toString();

            System.out.println("###########################################################################");
            System.out.println("Inside FinReducer:");
            System.out.println(node+key.get());
            System.out.println("###########################################################################");

            context.write(new Text(node), new Text(String.valueOf(0 - key.get()))); //Converting -rank back to +rank
        }
}
    }
```

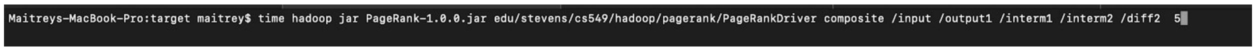
Outputting in form of (Node,rank). For this we will have to convert the -rank into rank.

Different numbers of reducers:

With modern systems it is very difficult to see the difference between the two results when there isn't much difference between number of reducers.

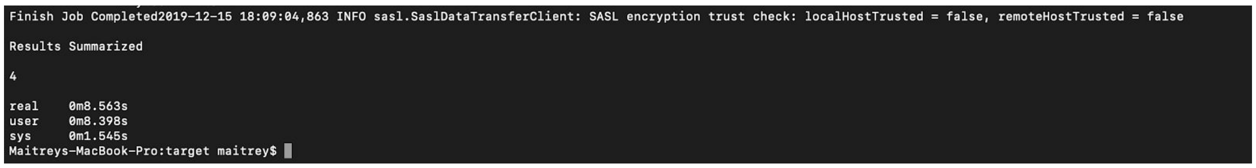
But a significant difference can be seen when you try to run the same program with the same input but big difference between number of reducers.

With 5 Reducers : 8.563 seconds
Command:



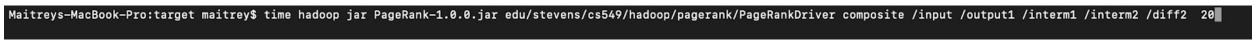
```
Maitreys-MacBook-Pro:target maitrey$ time hadoop jar PageRank-1.0.0.jar edu/stevens/cs549/hadoop/pagerank/PageRankDriver composite /input /output1 /interm1 /interm2 /diff2  5
```

Execution time:



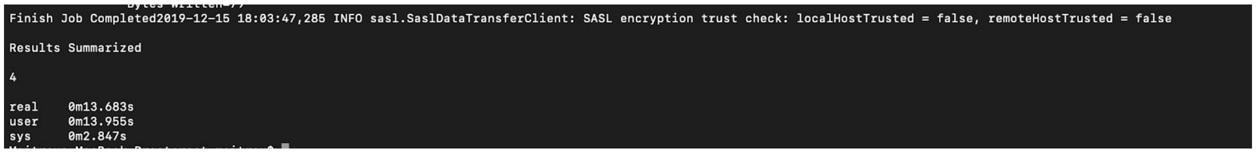
```
Finish Job Completed2019-12-15 18:09:04,863 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false

Results Summarized

4

real    0m8.563s
user    0m8.398s
sys     0m1.545s
Maitreys-MacBook-Pro:target maitrey$
```

With 20 Reducers: 13.683 seconds
Command:



```
Maitreys-MacBook-Pro:target maitrey$ time hadoop jar PageRank-1.0.0.jar edu/stevens/cs549/hadoop/pagerank/PageRankDriver composite /input /output1 /interm1 /interm2 /diff2  20
```

Execution time:



```
Finish Job Completed2019-12-15 18:03:47,285 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false

Results Summarized

4

real    0m13.683s
user    0m13.955s
sys     0m2.847s
```

This can be explained by the fact that it would take more execution time is required for more number of reducers.