GLOBAL TERRORISM DATASET

INFO7275 Advanced Database Management System

Final Project

This project is about implementing different MapReduce patterns to analyze the database to answer various questions. This includes some of the basic, summarization, filtering, and complex analysis patterns.

Jahnvi Gandhi
001665157

1. INTRODUCTION

1.1 About the Dataset:

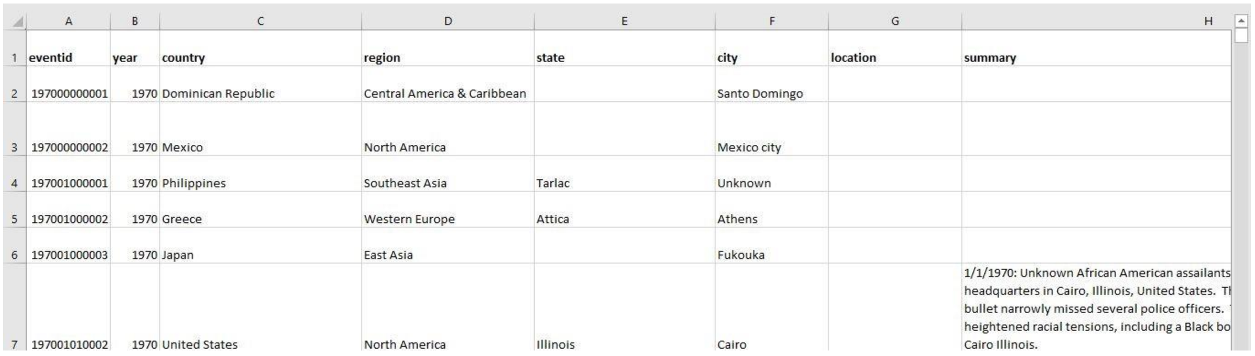
- ✓ The Global Terrorism Database (GTD) is a database containing information about incidences of terrorism from year 1970 to 2015.
- ✓ It has been maintained by the National Consortium for the Study of Terrorism and Responses to Terrorism (START) at the University of Maryland, College Park, United States.
- ✓ It is useful as basis in identifying terrorism-related measures, such as, Global Terrorism Index (GTI).
- ✓ One can try to deploy Recurrent Neural Network and other machine learning techniques to prognosticate terrorist attacks in the future.

1.2 Characteristics of the Dataset:

- ✓ It is an unclassified dataset having mentions for over 150,000 terrorist attacks around the world.
- ✓ The records have more than 75,000 bombings, 17,000 assassinations, 9,000 kidnappings and so on.
- ✓ In each case, 29 attributes are identified. They are:

1. Event Id	2. Year	3. Country
4. Region	5. State	6. City
7. Location	8. Summary	9. Multiple
10. Success	11. Suicide	12. Attack Type
13. Target	14. Nationality	15. Group Name
16. Target Type	17. Target Subtype	18. Corporation
19. Motive	20. Claim Mode	21. Weapon Type
22. Weapon Subtype	23. Weapon Details	24. Number Killed
25. Number Wounded	26. Ransom Amount	27. Ransom Paid
28. Database Source	29. Related Events	

✓ A sample of the dataset is shown in the screenshot below:



	A	B	C	D	E	F	G	H
1	eventid	year	country	region	state	city	location	summary
2	197000000001	1970	Dominican Republic	Central America & Caribbean		Santo Domingo		
3	197000000002	1970	Mexico	North America		Mexico city		
4	197001000001	1970	Philippines	Southeast Asia	Tarlac	Unknown		
5	197001000002	1970	Greece	Western Europe	Attica	Athens		
6	197001000003	1970	Japan	East Asia		Fukouka		
7	197001010002	1970	United States	North America	Illinois	Cairo		1/1/1970: Unknown African American assailants headquarters in Cairo, Illinois, United States. Th bullet narrowly missed several police officers. heightened racial tensions, including a Black bo Cairo Illinois.

1.3 Analysis performed to answer the following questions:

1. Finding out total attacks per year.
2. Finding out total attacks in each country.
3. Finding out total attacks of different attack types.
4. Finding out distinct types of attacks.
5. Finding out distinct attacking groups.
6. Sorting the attacks in each country in descending order.
7. Finding out Top N countries with higher casualties.
8. Finding out average casualties for each country.
9. Performing Market Basket Analysis based on the countries and attack types.
10. Performing Market Basket Analysis based on the countries and attacking groups.

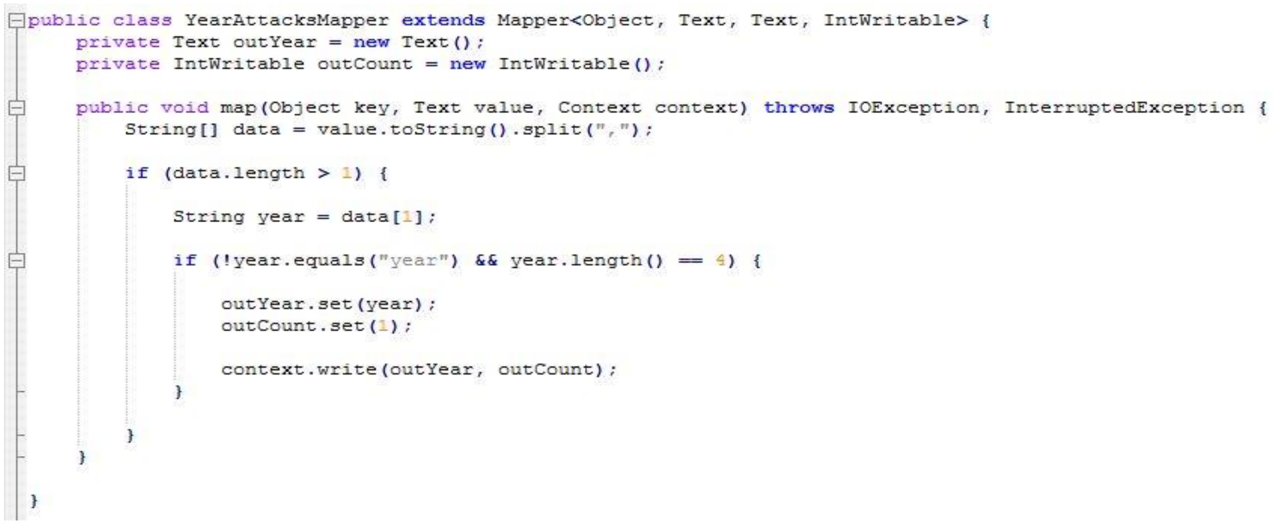
2. LINK TO THE DATASET:

https://www.kaggle.com/START-UMD/gtd

3. DETAILED DESCRIPTION:

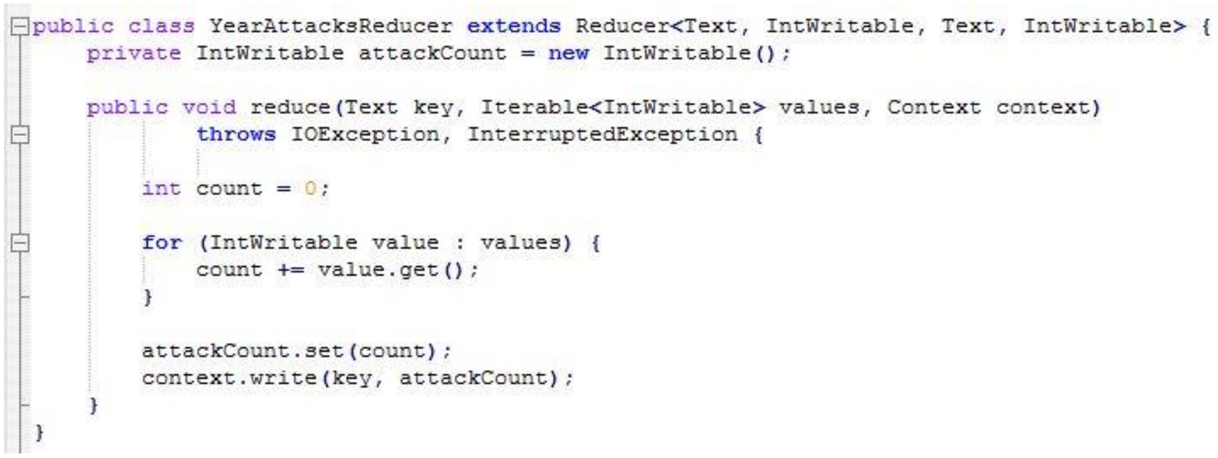
3.1 Finding out total attacks per year.

a. This implementation is a basic map reduce algorithm. The map function separates the input in <Key, Value> pairs. Here, the year is taken as the key as we want to find out the number of attacks per year. The count 1 is the value for each output <Key, Value> pair and given as the input to the reduce function.
b. The map function is shown below:



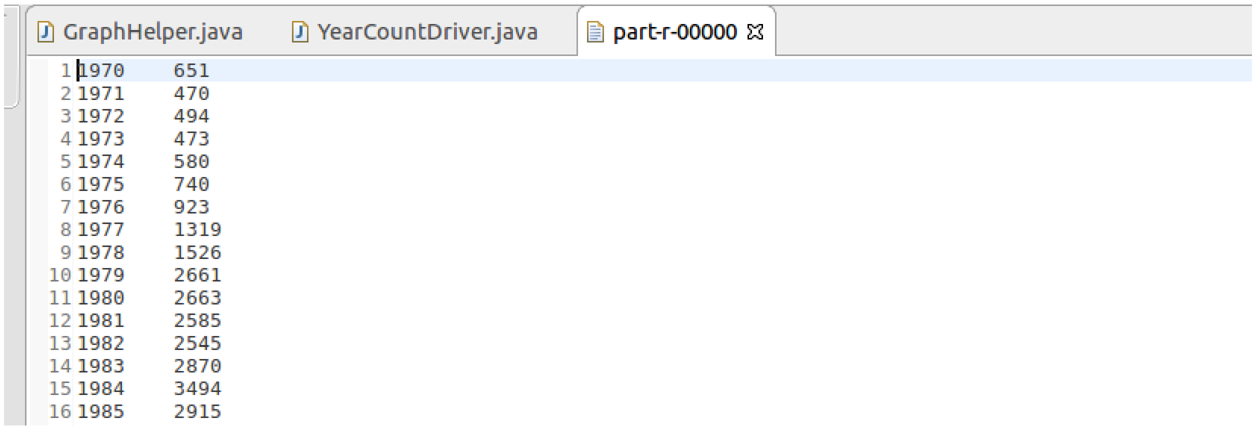
```java
public class YearAttacksMapper extends Mapper<Object, Text, Text, IntWritable> {
    private Text outYear = new Text();
    private IntWritable outCount = new IntWritable();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String[] data = value.toString().split(",");

        if (data.length > 1) {

            String year = data[1];

            if (!year.equals("year") && year.length() == 4) {

                outYear.set(year);
                outCount.set(1);

                context.write(outYear, outCount);
            }

        }
    }

}
```

c. The reduce function is shown below:



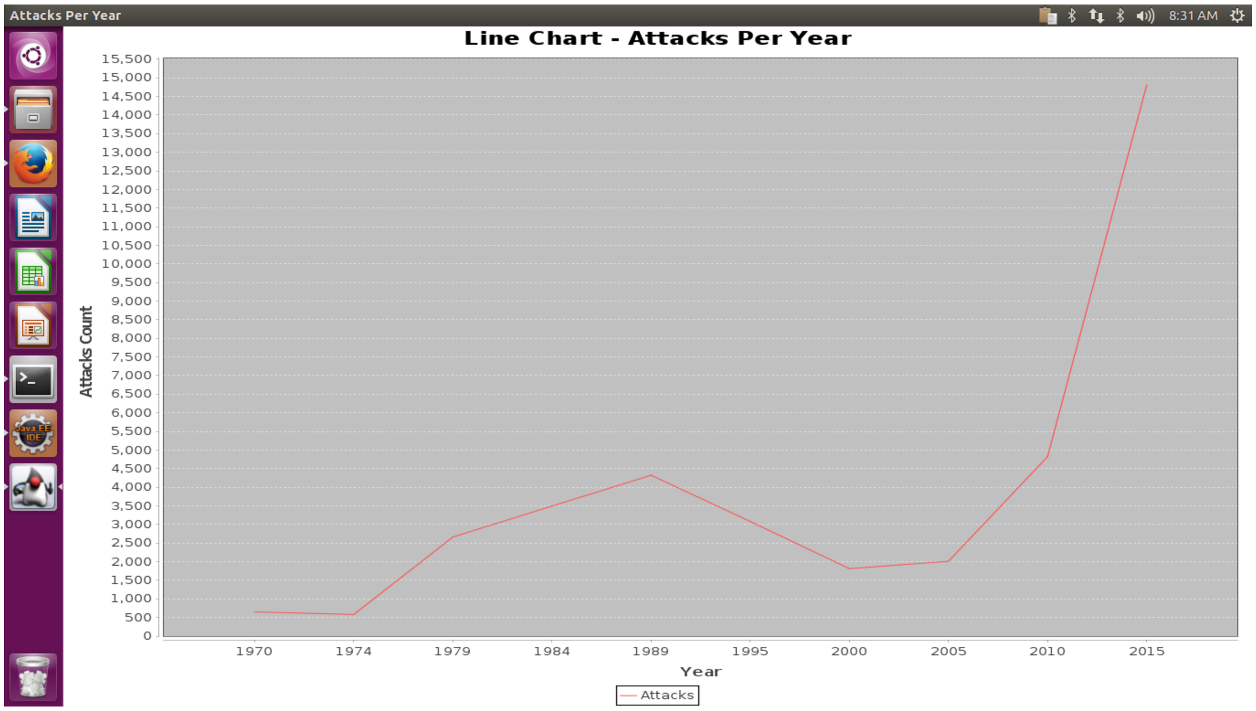
```java
public class YearAttacksReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable attackCount = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {

        int count = 0;

        for (IntWritable value : values) {
            count += value.get();
        }

        attackCount.set(count);
        context.write(key, attackCount);
    }
}
```

d. The output of the map-reduce is displayed in the screen shot below:



```
1970    651
1971    470
1972    494
1973    473
1974    580
1975    740
1976    923
1977    1319
1978    1526
1979    2661
1980    2663
1981    2585
1982    2545
1983    2870
1984    3494
1985    2915
```

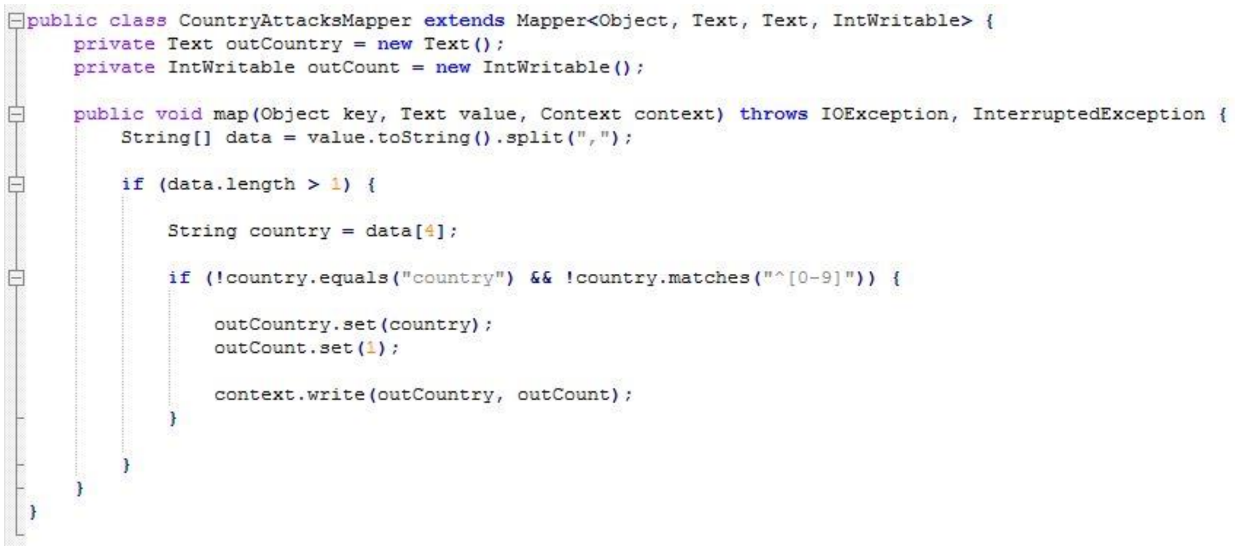
e. The output can be graphically represented as shown below:



f. From the graphical plotting of the values, the spike suggests that the number of attacks has increased to a large number by the year 2015.

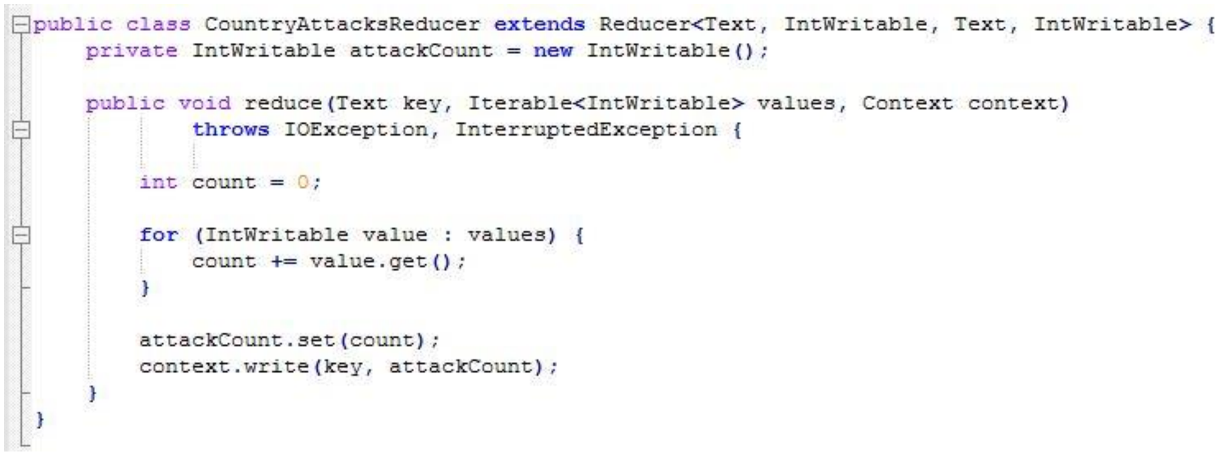
3.2 Finding out total attacks in each country.

a. This analysis is similar to the above mentioned pattern. The only difference in this implementation is the key. Here, the Country is used as the key and the number of attacks are calculated based on the countries.
b. The map function is shown below:



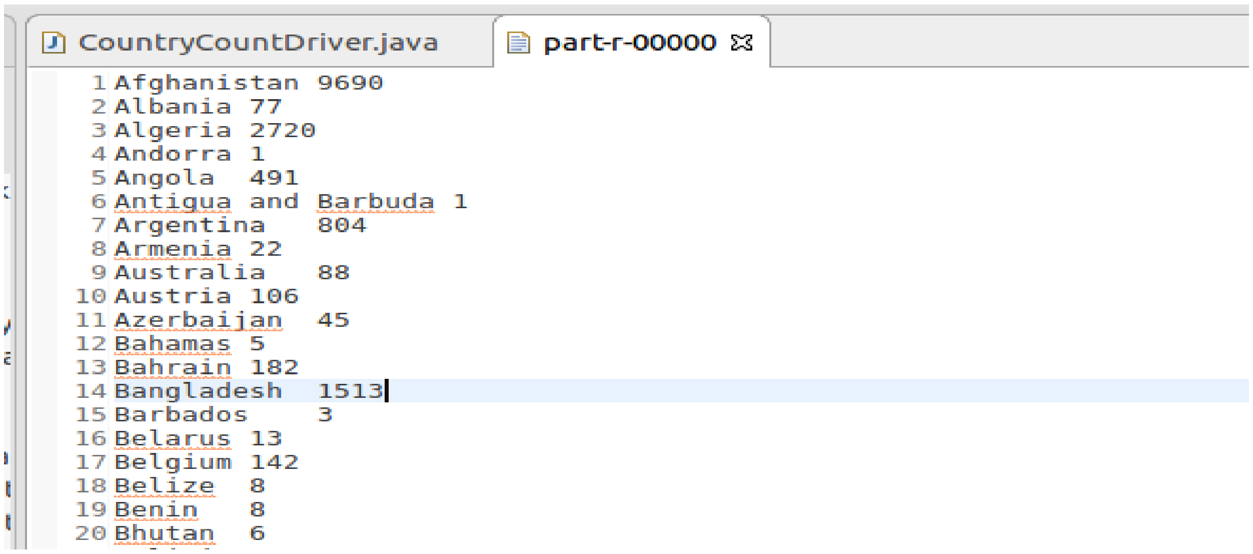
```
public class CountryAttacksMapper extends Mapper<Object, Text, Text, IntWritable> {
    private Text outCountry = new Text();
    private IntWritable outCount = new IntWritable();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String[] data = value.toString().split(",");

        if (data.length > 1) {

            String country = data[4];

            if (!country.equals("country") && !country.matches("^[0-9]")) {

                outCountry.set(country);
                outCount.set(1);

                context.write(outCountry, outCount);
            }

        }
    }
}
```

c. The reduce function is shown below:



```
public class CountryAttacksReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable attackCount = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {

        int count = 0;

        for (IntWritable value : values) {
            count += value.get();
        }

        attackCount.set(count);
        context.write(key, attackCount);
    }
}
```

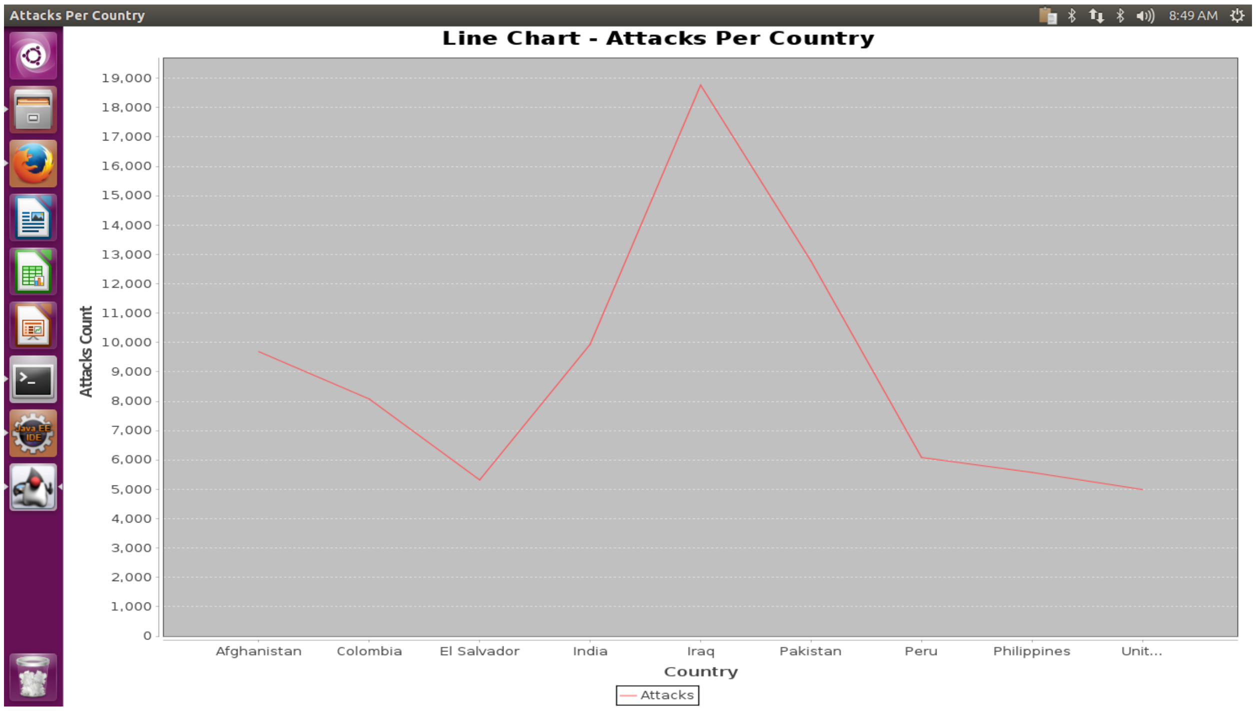
d. A piece of the output of the map-reduce is displayed in the screen shot below:



CountryCountDriver.java | part-r-00000

```
Afghanistan 9690
Albania 77
Algeria 2720
Andorra 1
Angola  491
Antigua and Barbuda 1
Argentina    804
Armenia 22
Australia    88
Austria 106
Azerbaijan   45
Bahamas 5
Bahrain 182
Bangladesh   1513
Barbados     3
Belarus 13
Belgium 142
Belize   8
Benin    8
Bhutan   6
```

e. The output can be graphically represented by Line Chart as shown below:

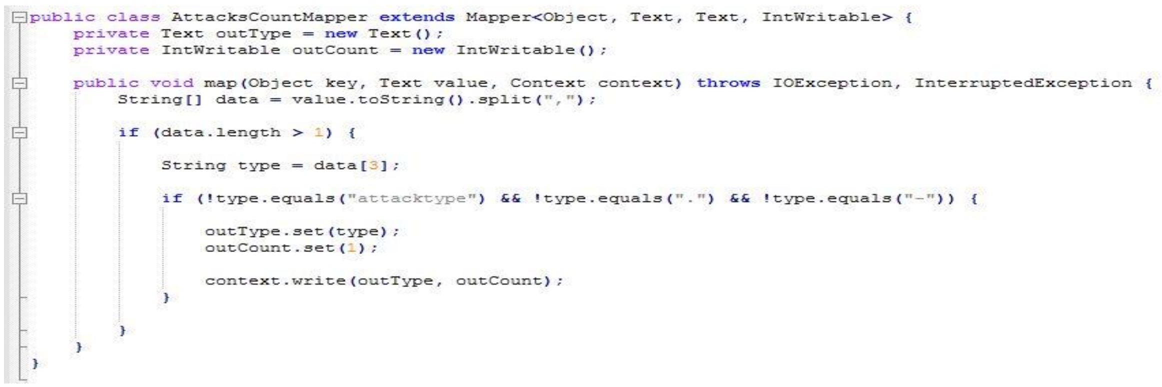


f. Here, the graph is plotted for the countries which have a number of attacks higher than 5000.

g. From the graphical plotting of the values, the spike suggests that the number of the attacks is way too large in the countries like Iraq.

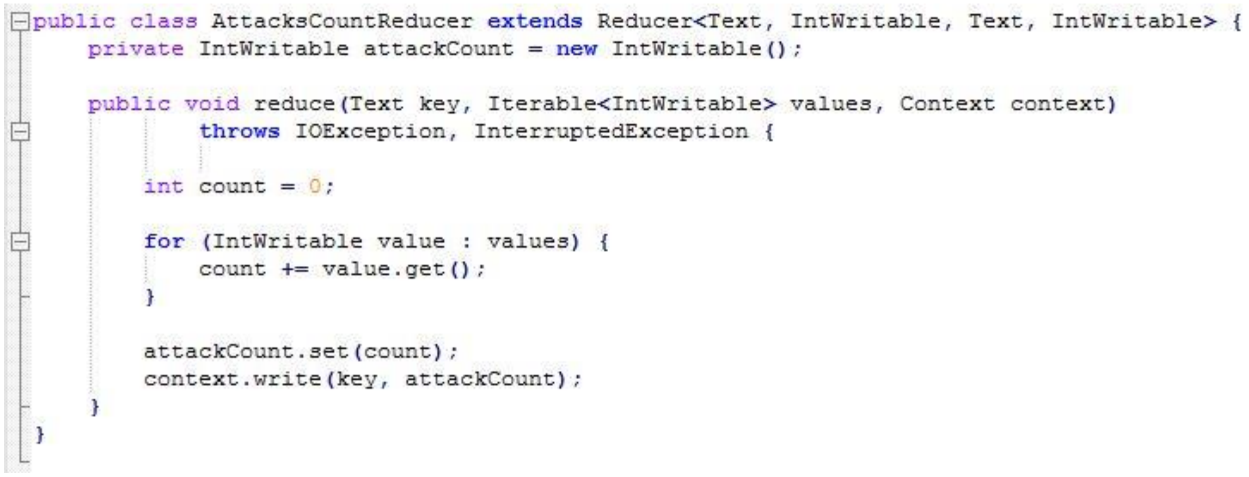
3.3 Finding out total attacks of different attack types.

a. The idea behind this analysis is to identify different types of attacks that have been taking place quite often.
b. The implementation here follows the similar concept and is basic. The difference here is the key. The attack type is used as the key and the number of attacks are calculated.
c. The map function looks like:



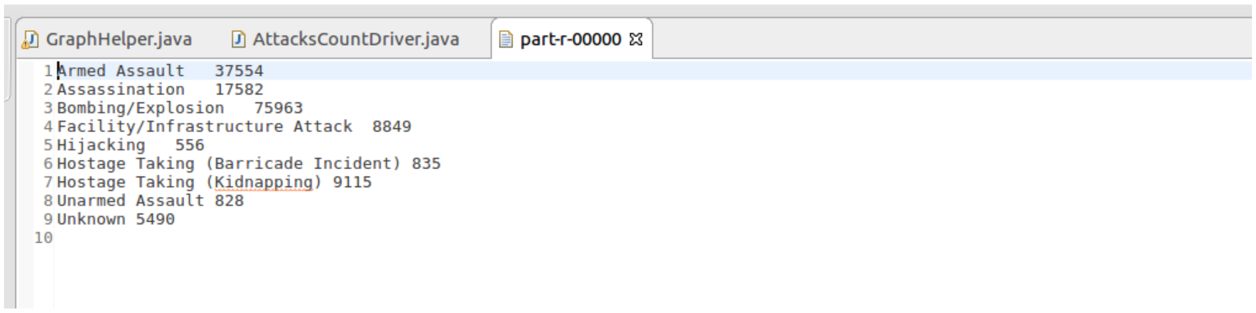
```
public class AttacksCountMapper extends Mapper<Object, Text, Text, IntWritable> {
    private Text outType = new Text();
    private IntWritable outCount = new IntWritable();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String[] data = value.toString().split(",");

        if (data.length > 1) {

            String type = data[3];

            if (!type.equals("attacktype") && !type.equals(".") && !type.equals("-")) {

                outType.set(type);
                outCount.set(1);

                context.write(outType, outCount);
            }

        }
    }
}
```

d. The reduce function looks like:



```
public class AttacksCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable attackCount = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {

        int count = 0;

        for (IntWritable value : values) {
            count += value.get();
        }

        attackCount.set(count);
        context.write(key, attackCount);
    }
}
```

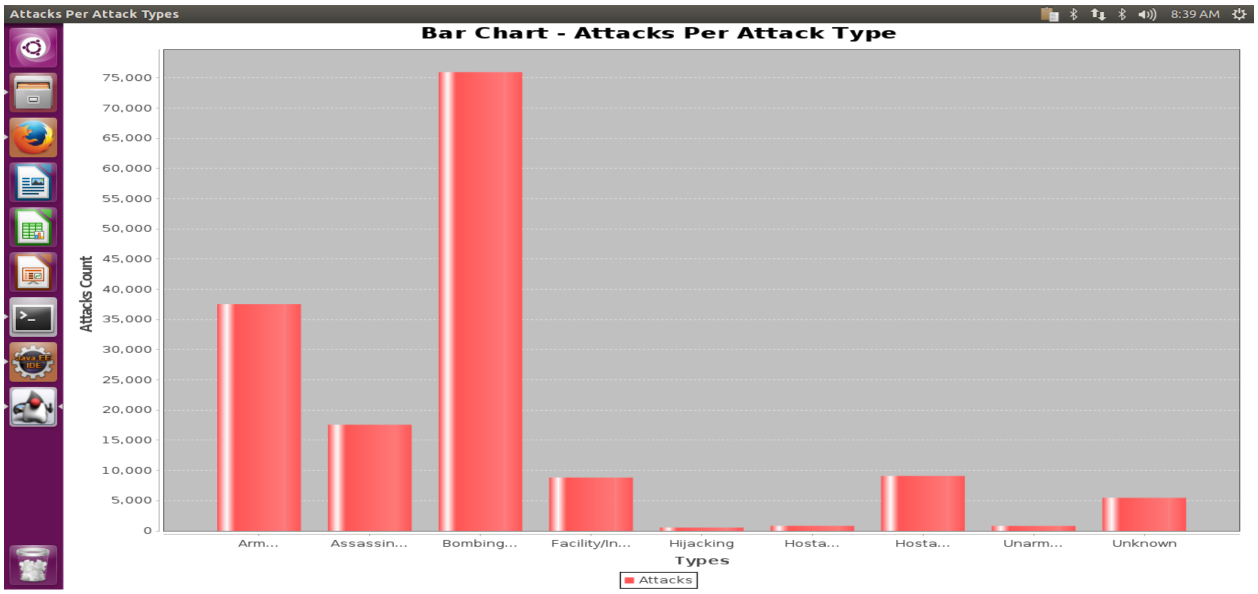
e. The output of the implementation is presented in the following screen shot:



GraphHelper.java | AttacksCountDriver.java | part-r-00000

```
Armed Assault   37554
Assassination   17582
Bombing/Explosion   75963
Facility/Infrastructure Attack  8849
Hijacking   556
Hostage Taking (Barricade Incident) 835
Hostage Taking (Kidnapping) 9115
Unarmed Assault 828
Unknown 5490
```

f. The graphical representation is shown below:

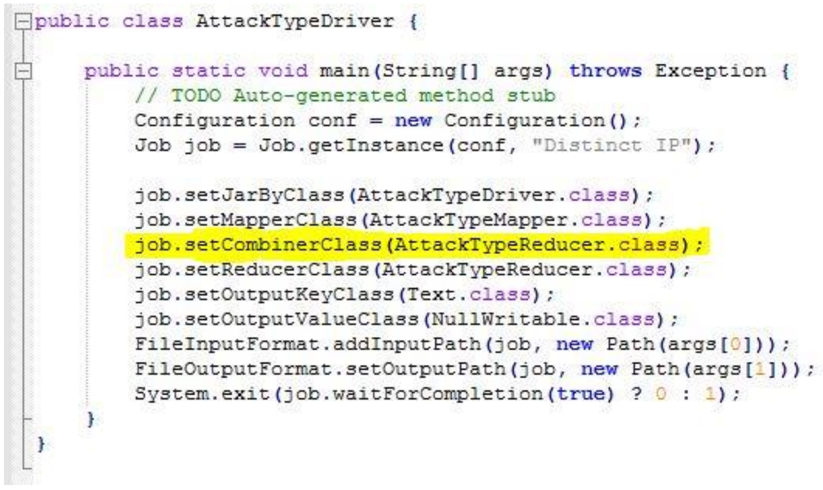


g. From the plot we can identify that the Bombing/Explosions have taken place the most.

3.4 Finding out the distinct types of attacks.

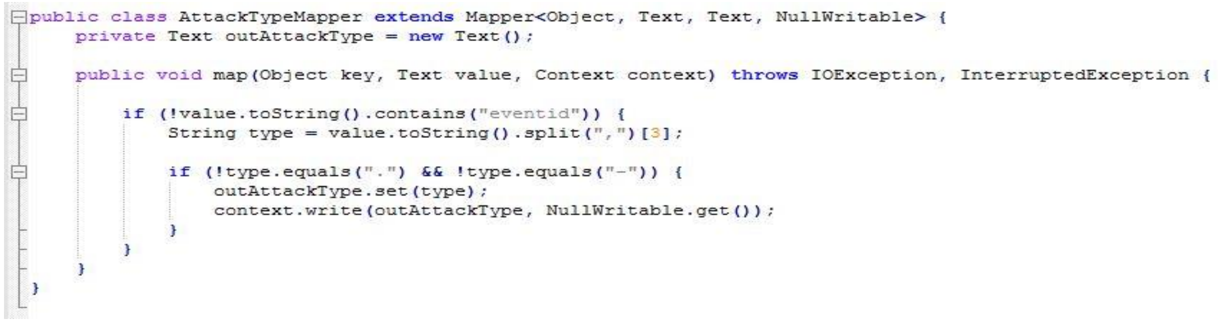
a. To find out the distinct types of attacks, a simple map-reduce job is executed.
b. In this implementation map function will output only the key, here the attack type, and NullWritable as the value.
c. These <Key, Value> ➔ <AttackType, NullWritable> will be given as the inputs to the combiner, which will reduce the load on the reducer.
d. The combiner and reducer implementations are same in this case, as we have to scrap the repeating values and generate only the distinct key.

e. At the end, the reducer will generate output of distinct attack types.
f. Using the combiner optimizes the execution time.
g. The driver function looks like shown in the screen shot when a combiner is used:

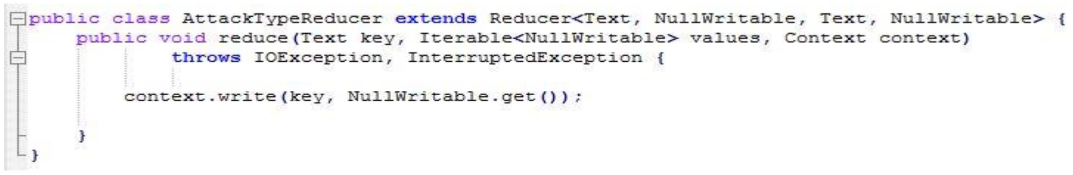


```
public class AttackTypeDriver {

    public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Distinct IP");

        job.setJarByClass(AttackTypeDriver.class);
        job.setMapperClass(AttackTypeMapper.class);
        job.setCombinerClass(AttackTypeReducer.class);
        job.setReducerClass(AttackTypeReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(NullWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

h. The map function and the reduce function are shown below:

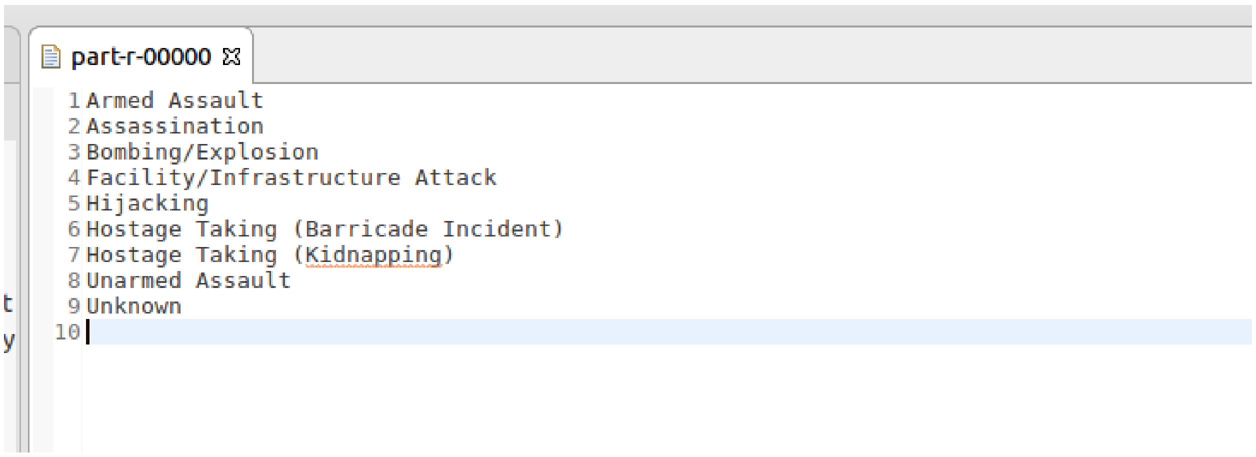


```
public class AttackTypeMapper extends Mapper<Object, Text, Text, NullWritable> {
    private Text outAttackType = new Text();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

        if (!value.toString().contains("eventid")) {
            String type = value.toString().split(",")[3];

            if (!type.equals(".") && !type.equals("-")) {
                outAttackType.set(type);
                context.write(outAttackType, NullWritable.get());
            }
        }
    }
}
```



```
public class AttackTypeReducer extends Reducer<Text, NullWritable, Text, NullWritable> {
    public void reduce(Text key, Iterable<NullWritable> values, Context context)
            throws IOException, InterruptedException {

        context.write(key, NullWritable.get());

    }
}
```

i. The output of the map-reduce job is:

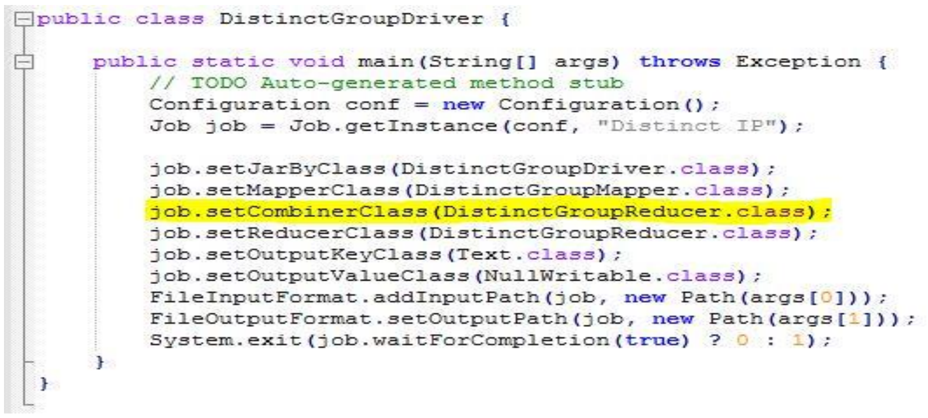


part-r-00000

```
Armed Assault
Assassination
Bombing/Explosion
Facility/Infrastructure Attack
Hijacking
Hostage Taking (Barricade Incident)
Hostage Taking (Kidnapping)
Unarmed Assault
Unknown
```

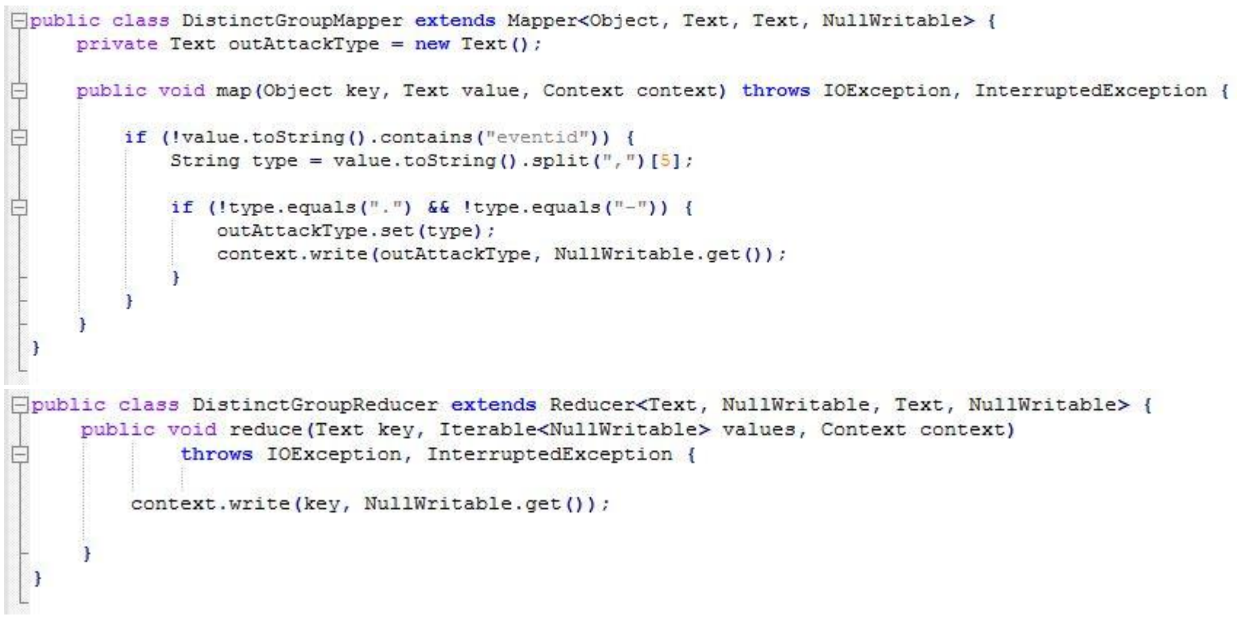
3.5 Finding out the distinct attacking groups:

a. This analysis is carried out to identify the distinct groups, which carry out the attacks.
b. In this implementation map function will output only the key, here the attack type, and NullWritable as the value.
c. These <Key, Value> ➔ <AttackType, NullWritable> will be given as the inputs to the combiner, which will reduce the load on the reducer.
d. The combiner and reducer implementations are same in this case, as we have to scrap the repeating values and generate only the distinct key.
e. At the end, the reducer will generate output of distinct attack types.
f. Using the combiner optimizes the execution time.
g. The driver function looks like shown in the screen shot when a combiner is used:



```
public class DistinctGroupDriver {

    public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Distinct IP");

        job.setJarByClass(DistinctGroupDriver.class);
        job.setMapperClass(DistinctGroupMapper.class);
        job.setCombinerClass(DistinctGroupReducer.class);
        job.setReducerClass(DistinctGroupReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(NullWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

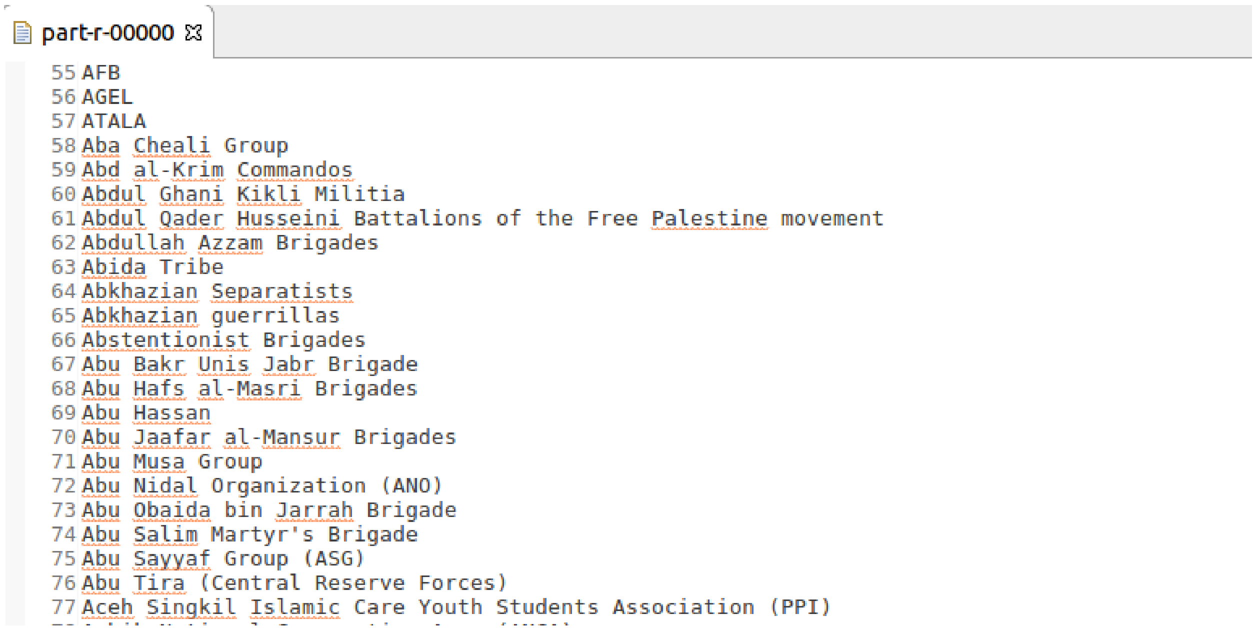
h. The map function and the reduce function are shown below:



```
public class DistinctGroupMapper extends Mapper<Object, Text, Text, NullWritable> {
    private Text outAttackType = new Text();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

        if (!value.toString().contains("eventid")) {
            String type = value.toString().split(",")[5];

            if (!type.equals(".") && !type.equals("-")) {
                outAttackType.set(type);
                context.write(outAttackType, NullWritable.get());
            }
        }
    }
}
```

```
public class DistinctGroupReducer extends Reducer<Text, NullWritable, Text, NullWritable> {
    public void reduce(Text key, Iterable<NullWritable> values, Context context)
            throws IOException, InterruptedException {

        context.write(key, NullWritable.get());

    }
}
```

i. The output of the map-reduce job is:



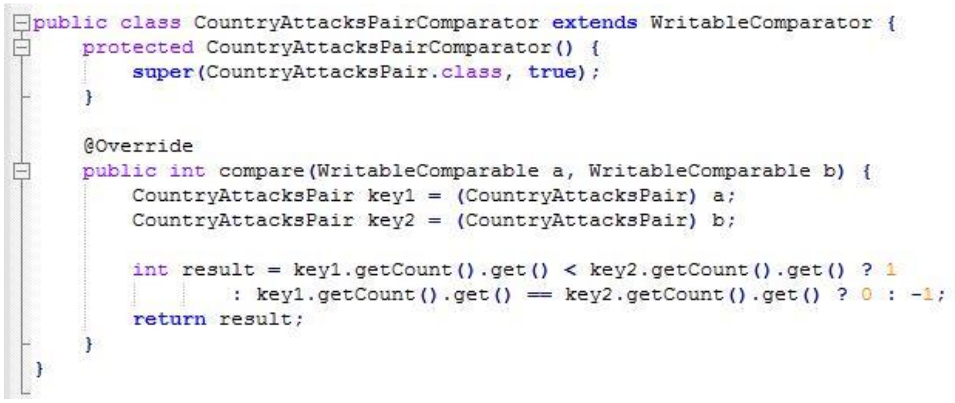
part-r-00000

```
AFB
AGEL
ATALA
Aba Cheali Group
Abd al-Krim Commandos
Abdul Ghani Kikli Militia
Abdul Qader Husseini Battalions of the Free Palestine movement
Abdullah Azzam Brigades
Abida Tribe
Abkhazian Separatists
Abkhazian guerrillas
Abstentionist Brigades
Abu Bakr Unis Jabr Brigade
Abu Hafs al-Masri Brigades
Abu Hassan
Abu Jaafar al-Mansur Brigades
Abu Musa Group
Abu Nidal Organization (ANO)
Abu Obaida bin Jarrah Brigade
Abu Salim Martyr's Brigade
Abu Sayyaf Group (ASG)
Abu Tira (Central Reserve Forces)
Aceh Singkil Islamic Care Youth Students Association (PPI)
```

3.6 Sorting the attacks per country in descending order:

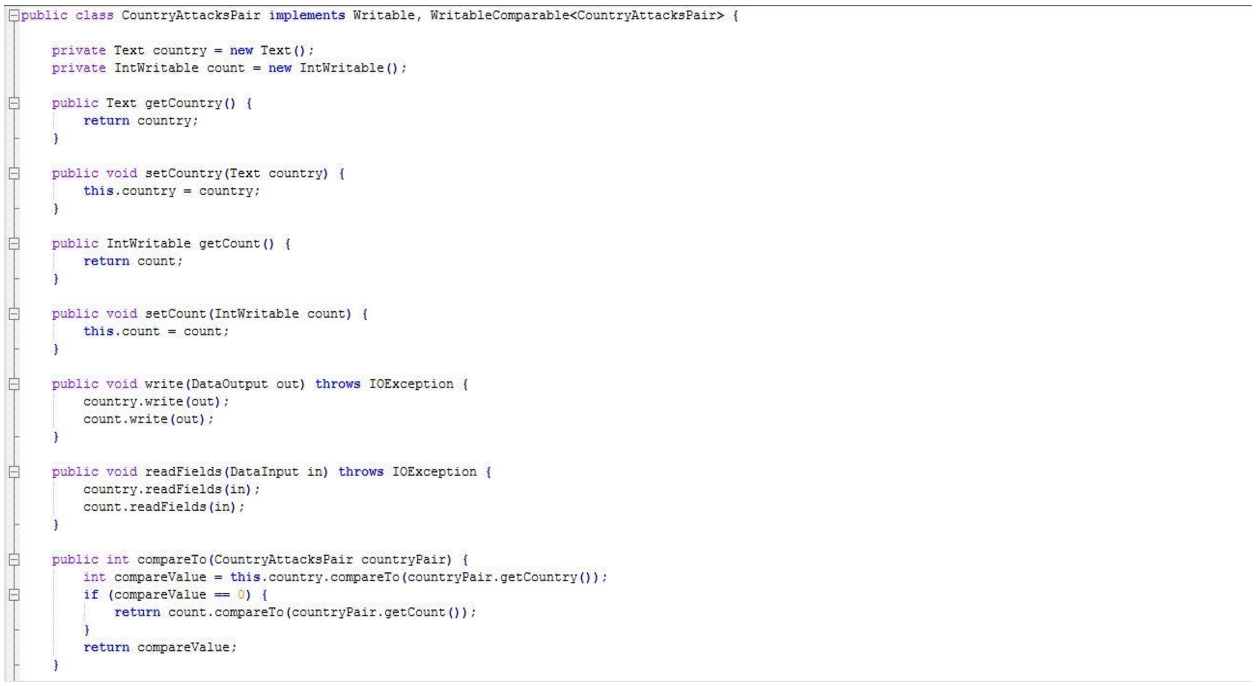
a. The analysis is used to determine the most attacked countries at the top and then followed by other countries.

b. The sorting of the countries is achieved using a Secondary Sorting algorithm.
c. There are two map-reduce jobs used to perform the task.
d. First set of map-reduce functions will calculate the total number of attacks for each country. The output of this map-reduce implementation will give the data sorted based on the Key, i.e. Country in this case.
e. As, we need to achieve sorting based on the Values, i.e. count of attacks, we need to implement a second set of map-reduce functions.
f. The second set will take the composite key and makes use of the WritableComparator class, and will perform the sorting using the part of the composite key.



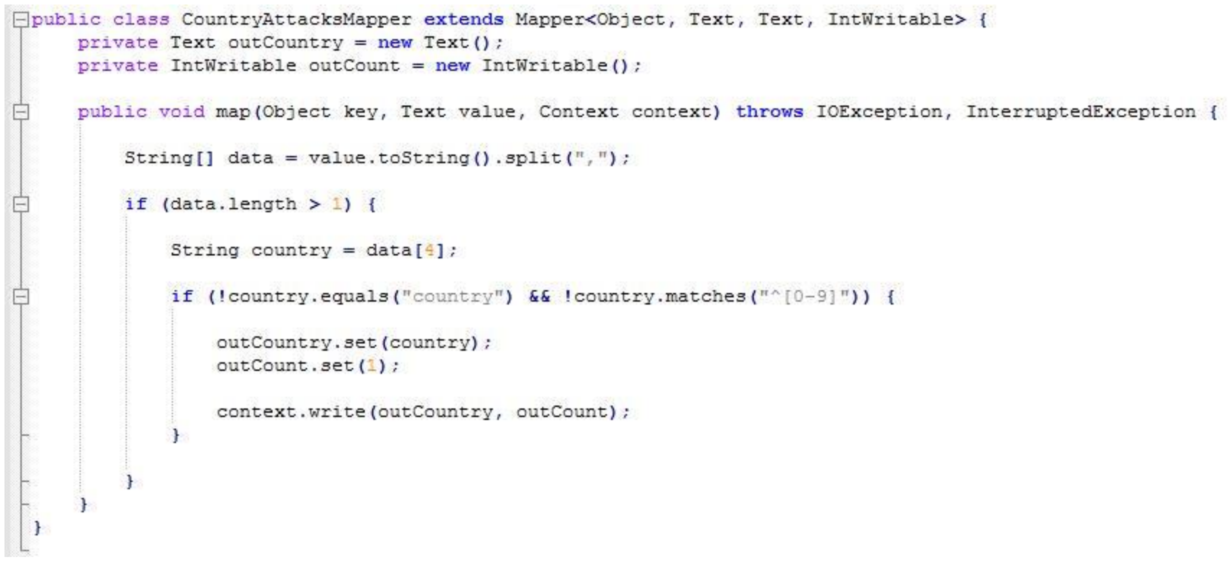
```
public class CountryAttacksPairComparator extends WritableComparator {
    protected CountryAttacksPairComparator() {
        super(CountryAttacksPair.class, true);
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {
        CountryAttacksPair key1 = (CountryAttacksPair) a;
        CountryAttacksPair key2 = (CountryAttacksPair) b;

        int result = key1.getCount().get() < key2.getCount().get() ? 1
                : key1.getCount().get() == key2.getCount().get() ? 0 : -1;
        return result;
    }
}
```

g. In this case, the composite key is made up of, the country and the attack count, where the country is the Natural Key.
h. The composite key is created using a Writable/WritableComparable class, that is displayed in the screenshot below:

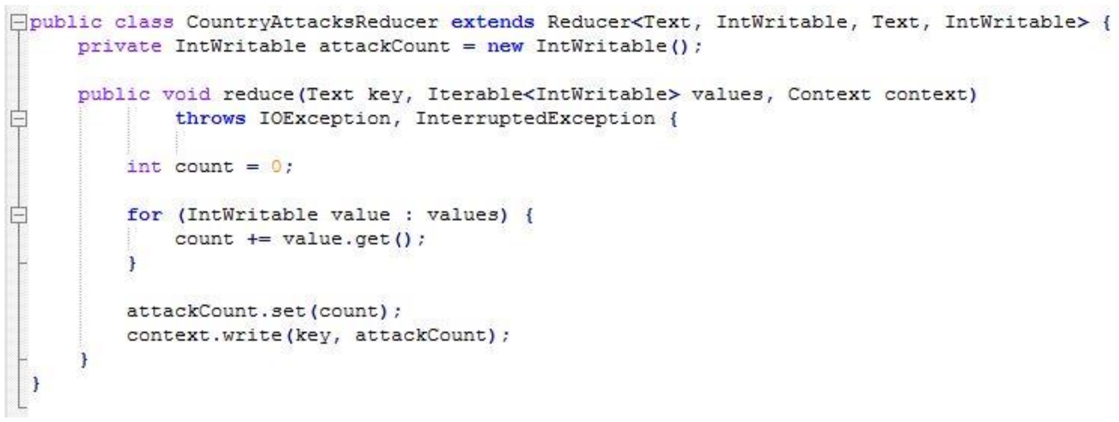


```
public class CountryAttacksPair implements Writable, WritableComparable<CountryAttacksPair> {

    private Text country = new Text();
    private IntWritable count = new IntWritable();

    public Text getCountry() {
        return country;
    }

    public void setCountry(Text country) {
        this.country = country;
    }

    public IntWritable getCount() {
        return count;
    }

    public void setCount(IntWritable count) {
        this.count = count;
    }

    public void write(DataOutput out) throws IOException {
        country.write(out);
        count.write(out);
    }

    public void readFields(DataInput in) throws IOException {
        country.readFields(in);
        count.readFields(in);
    }

    public int compareTo(CountryAttacksPair countryPair) {
        int compareValue = this.country.compareTo(countryPair.getCountry());
        if (compareValue == 0) {
            return count.compareTo(countryPair.getCount());
        }
        return compareValue;
    }
```

i. The first set of map-reduce functions are as shown below:

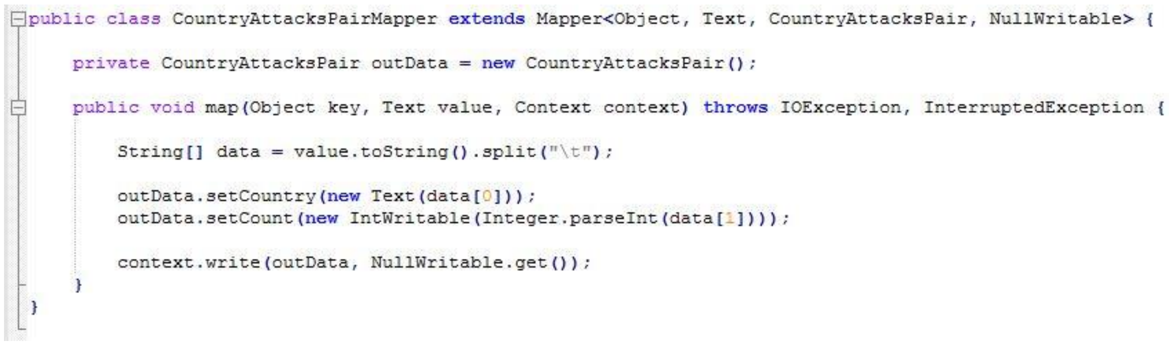


```
public class CountryAttacksMapper extends Mapper<Object, Text, Text, IntWritable> {
    private Text outCountry = new Text();
    private IntWritable outCount = new IntWritable();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

        String[] data = value.toString().split(",");

        if (data.length > 1) {

            String country = data[4];

            if (!country.equals("country") && !country.matches("^[0-9]")) {

                outCountry.set(country);
                outCount.set(1);

                context.write(outCountry, outCount);
            }

        }
    }
}
```

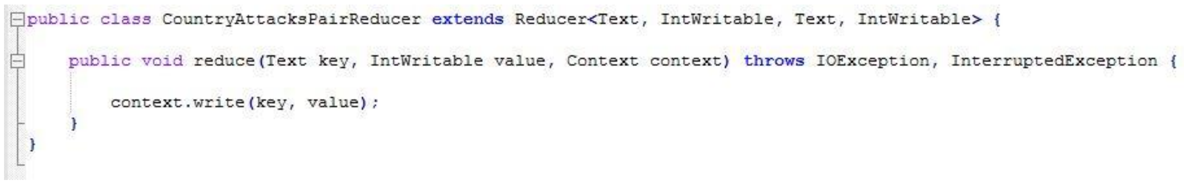


```java
public class CountryAttacksReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable attackCount = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {

        int count = 0;

        for (IntWritable value : values) {
            count += value.get();
        }

        attackCount.set(count);
        context.write(key, attackCount);
    }
}
```

j. The second set of map-reduce functions are as shown below:

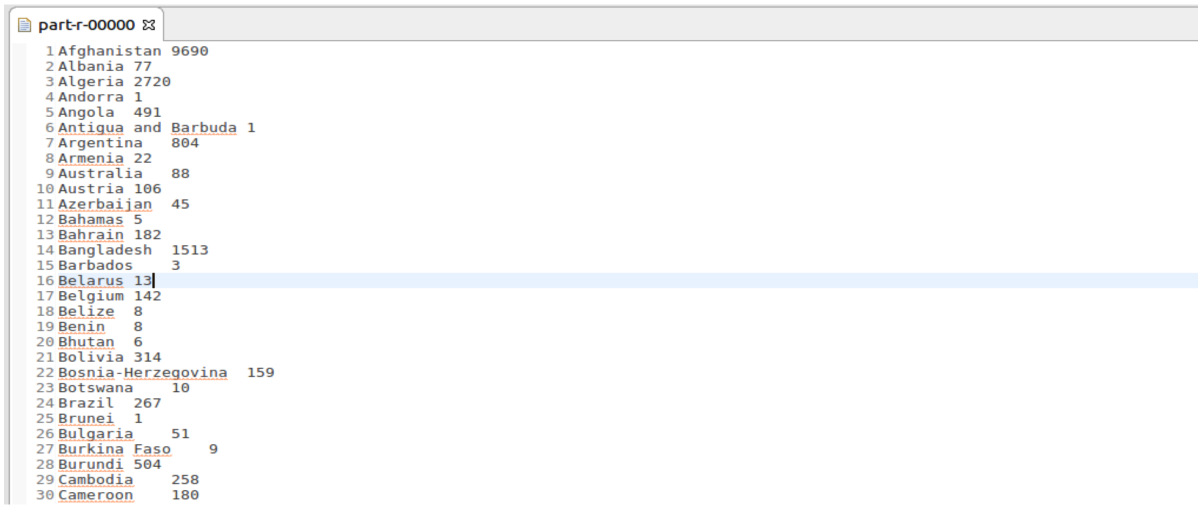


```java
public class CountryAttacksPairMapper extends Mapper<Object, Text, CountryAttacksPair, NullWritable> {

    private CountryAttacksPair outData = new CountryAttacksPair();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

        String[] data = value.toString().split("\t");

        outData.setCountry(new Text(data[0]));
        outData.setCount(new IntWritable(Integer.parseInt(data[1])));

        context.write(outData, NullWritable.get());
    }
}
```



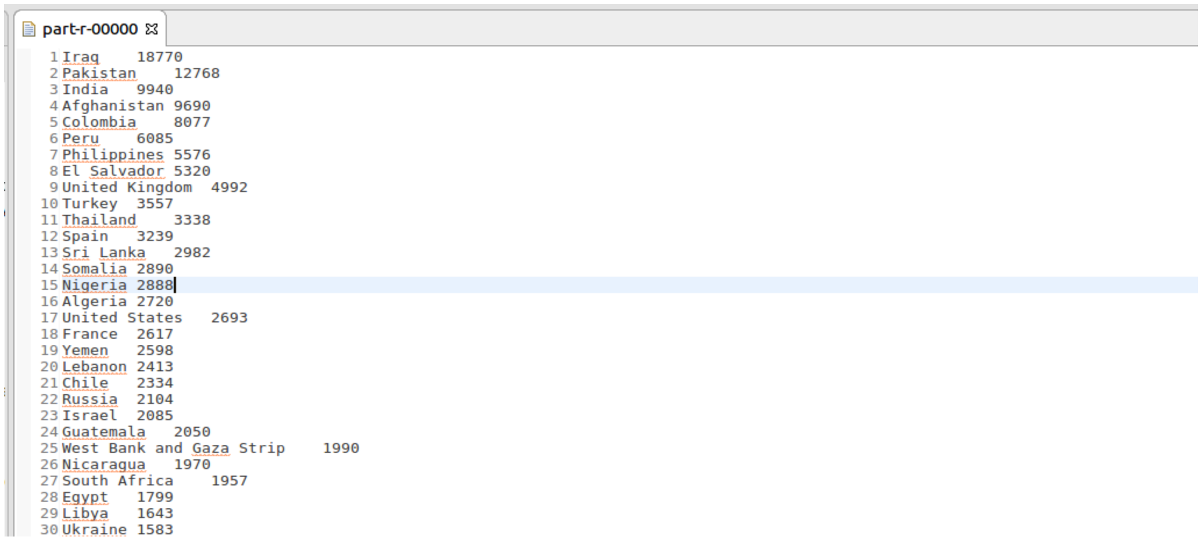
```java
public class CountryAttacksPairReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, IntWritable value, Context context) throws IOException, InterruptedException {

        context.write(key, value);
    }
}
```

k. The intermediate output and the sorted output can be compared using following screen shots:



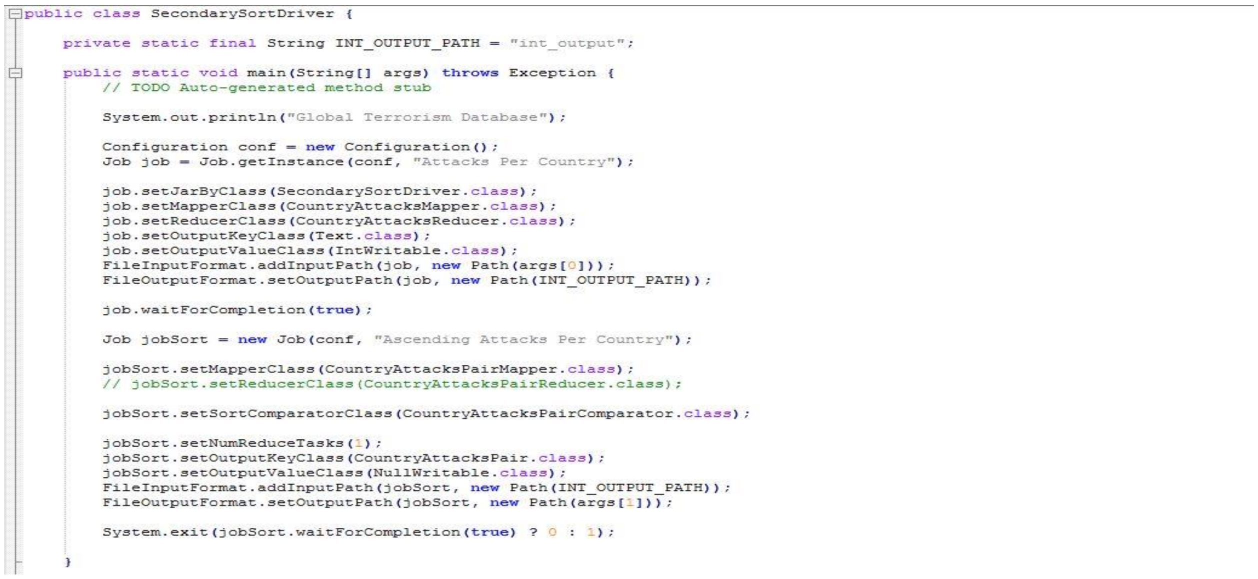
part-r-00000

```
Afghanistan	9690
Albania	77
Algeria	2720
Andorra	1
Angola	491
Antigua and Barbuda	1
Argentina	804
Armenia	22
Australia	88
Austria	106
Azerbaijan	45
Bahamas	5
Bahrain	182
Bangladesh	1513
Barbados	3
Belarus	13
Belgium	142
Belize	8
Benin	8
Bhutan	6
Bolivia	314
Bosnia-Herzegovina	159
Botswana	10
Brazil	267
Brunei	1
Bulgaria	51
Burkina Faso	9
Burundi	504
Cambodia	258
Cameroon	180
```



```
Iraq	18770
Pakistan	12768
India	9940
Afghanistan	9690
Colombia	8077
Peru	6085
Philippines	5576
El Salvador	5320
United Kingdom	4992
Turkey	3557
Thailand	3338
Spain	3239
Sri Lanka	2982
Somalia	2890
Nigeria	2888
Algeria	2720
United States	2693
France	2617
Yemen	2598
Lebanon	2413
Chile	2334
Russia	2104
Israel	2085
Guatemala	2050
West Bank and Gaza Strip	1990
Nicaragua	1970
South Africa	1957
Egypt	1799
Libya	1643
Ukraine	1583
```

l. The chaining of both the map-reduce jobs in the driver class looks like:

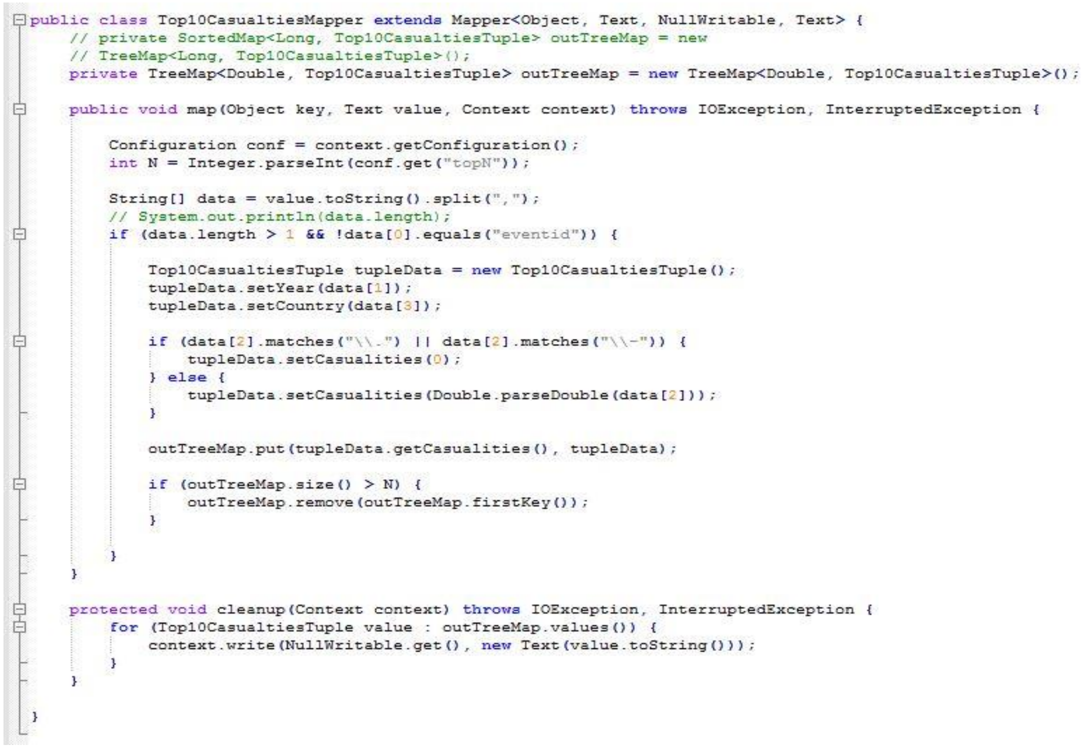


```java
public class SecondarySortDriver {

    private static final String INT_OUTPUT_PATH = "int_output";

    public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub

        System.out.println("Global Terrorism Database");

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Attacks Per Country");

        job.setJarByClass(SecondarySortDriver.class);
        job.setMapperClass(CountryAttacksMapper.class);
        job.setReducerClass(CountryAttacksReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(INT_OUTPUT_PATH));

        job.waitForCompletion(true);

        Job jobSort = new Job(conf, "Ascending Attacks Per Country");

        jobSort.setMapperClass(CountryAttacksPairMapper.class);
        // jobSort.setReducerClass(CountryAttacksPairReducer.class);

        jobSort.setSortComparatorClass(CountryAttacksPairComparator.class);

        jobSort.setNumReduceTasks(1);
        jobSort.setOutputKeyClass(CountryAttacksPair.class);
        jobSort.setOutputValueClass(NullWritable.class);
        FileInputFormat.addInputPath(jobSort, new Path(INT_OUTPUT_PATH));
        FileOutputFormat.setOutputPath(jobSort, new Path(args[1]));

        System.exit(jobSort.waitForCompletion(true) ? 0 : 1);

    }
```

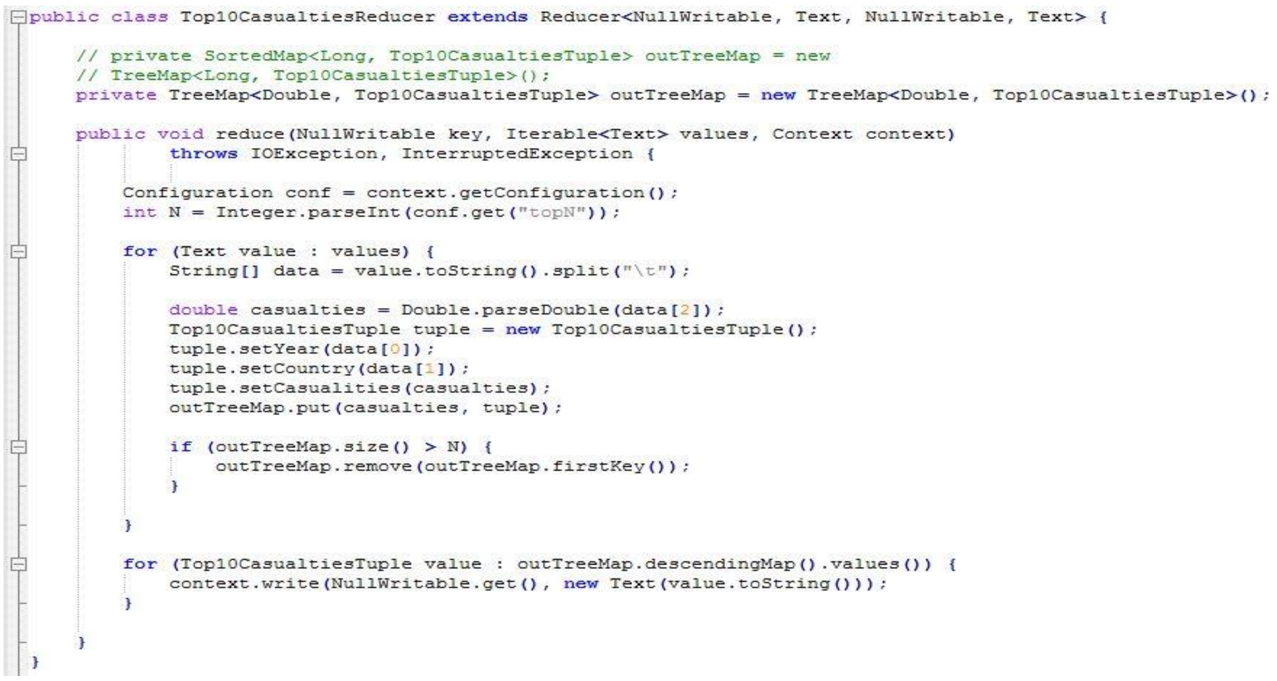
3.7 Finding out Top N countries based on casualties:

a. The terror attacks have affected the population and casualties are mentioned in the dataset.
b. To find out the top N casualties that occurred through 1970 to 2015, this analysis is performed.
c. In this implementation, the number N is taken from the user.
d. Based on the N, the map function uses a cleanup method to clean the TreeMap, which is used to store the <Key, Value> pairs, and removes the smallest pairs after size N.

e. Once the mapper function outputs the N <Key, Value> pairs, they are given to the reducer function.
f. Here, reduce function also works similar to the mapper function. It iterates over the values and creates a TreeMap of size N, and emits the values in descending order.
g. The mapper and reducer functions are as shown below:

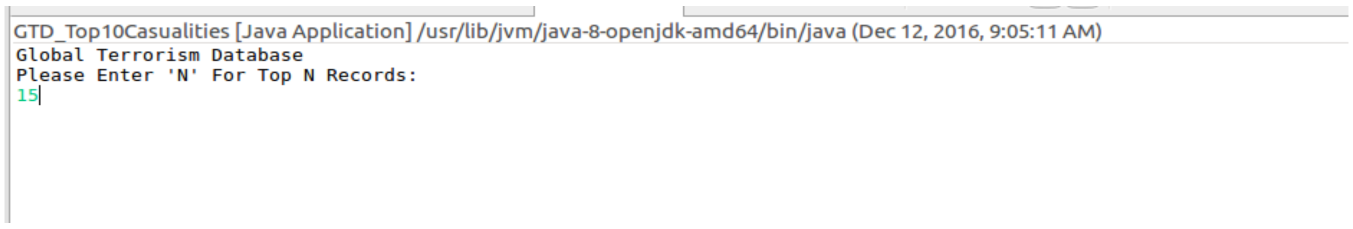


```java
public class Top10CasualtiesMapper extends Mapper<Object, Text, NullWritable, Text> {
    // private SortedMap<Long, Top10CasualtiesTuple> outTreeMap = new
    // TreeMap<Long, Top10CasualtiesTuple>();
    private TreeMap<Double, Top10CasualtiesTuple> outTreeMap = new TreeMap<Double, Top10CasualtiesTuple>();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

        Configuration conf = context.getConfiguration();
        int N = Integer.parseInt(conf.get("topN"));

        String[] data = value.toString().split(",");
        // System.out.println(data.length);
        if (data.length > 1 && !data[0].equals("eventid")) {

            Top10CasualtiesTuple tupleData = new Top10CasualtiesTuple();
            tupleData.setYear(data[1]);
            tupleData.setCountry(data[3]);

            if (data[2].matches("\\.") || data[2].matches("\\-")) {
                tupleData.setCasualities(0);
            } else {
                tupleData.setCasualities(Double.parseDouble(data[2]));
            }

            outTreeMap.put(tupleData.getCasualities(), tupleData);

            if (outTreeMap.size() > N) {
                outTreeMap.remove(outTreeMap.firstKey());
            }

        }
    }

    protected void cleanup(Context context) throws IOException, InterruptedException {
        for (Top10CasualtiesTuple value : outTreeMap.values()) {
            context.write(NullWritable.get(), new Text(value.toString()));
        }
    }

}
```

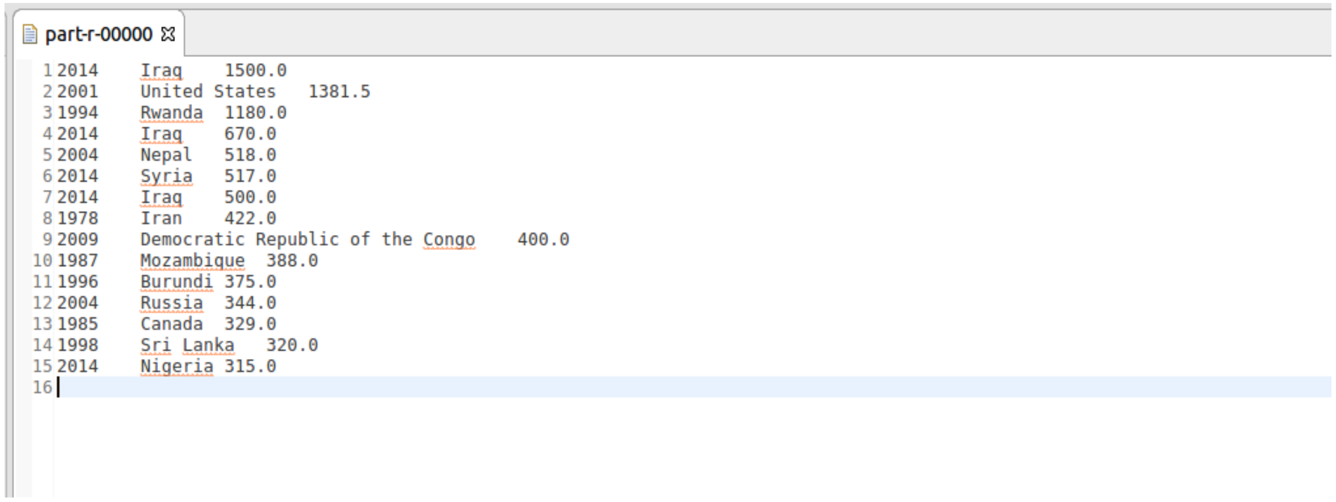


```java
public class Top10CasualtiesReducer extends Reducer<NullWritable, Text, NullWritable, Text> {

    // private SortedMap<Long, Top10CasualtiesTuple> outTreeMap = new
    // TreeMap<Long, Top10CasualtiesTuple>();
    private TreeMap<Double, Top10CasualtiesTuple> outTreeMap = new TreeMap<Double, Top10CasualtiesTuple>();

    public void reduce(NullWritable key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

        Configuration conf = context.getConfiguration();
        int N = Integer.parseInt(conf.get("topN"));

        for (Text value : values) {
            String[] data = value.toString().split("\t");

            double casualties = Double.parseDouble(data[2]);
            Top10CasualtiesTuple tuple = new Top10CasualtiesTuple();
            tuple.setYear(data[0]);
            tuple.setCountry(data[1]);
            tuple.setCasualities(casualties);
            outTreeMap.put(casualties, tuple);

            if (outTreeMap.size() > N) {
                outTreeMap.remove(outTreeMap.firstKey());
            }

        }

        for (Top10CasualtiesTuple value : outTreeMap.descendingMap().values()) {
            context.write(NullWritable.get(), new Text(value.toString()));
        }

    }
}
```

h. The output generated is:



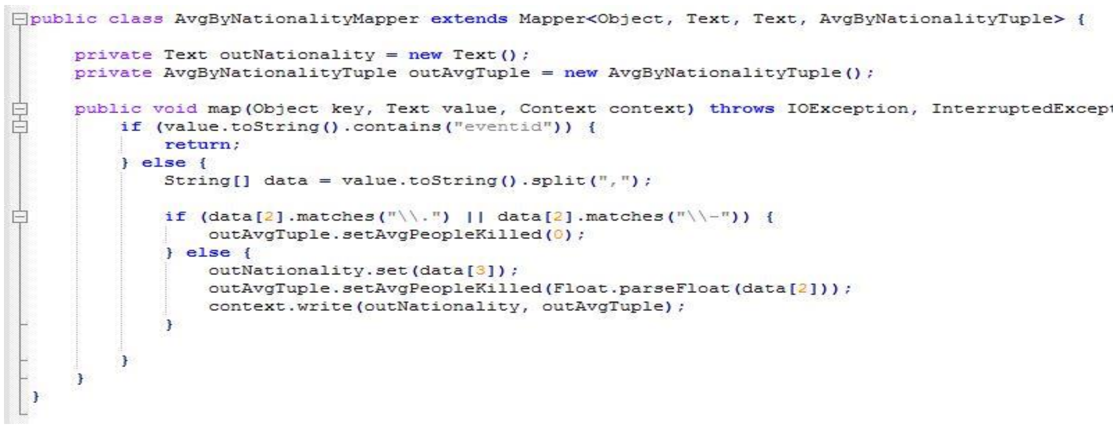
```
GTD_Top10Casualities [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (Dec 12, 2016, 9:05:11 AM)
Global Terrorism Database
Please Enter 'N' For Top N Records:
15
```



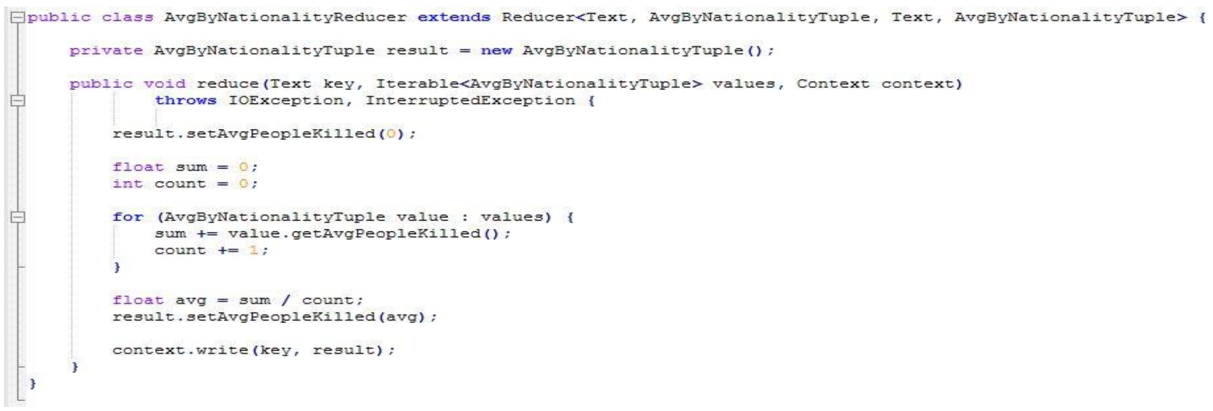
```
part-r-00000
2014    Iraq    1500.0
2001    United States   1381.5
1994    Rwanda  1180.0
2014    Iraq    670.0
2004    Nepal   518.0
2014    Syria   517.0
2014    Iraq    500.0
1978    Iran    422.0
2009    Democratic Republic of the Congo    400.0
1987    Mozambique  388.0
1996    Burundi 375.0
2004    Russia  344.0
1985    Canada  329.0
1998    Sri Lanka   320.0
2014    Nigeria 315.0
```

3.8 Finding out average casualties by nationality/country:

a. To get a brief idea about the average casualties per country in all the attacks this analysis is performed.
b. A map-reduce job is implemented, where the map function outputs the <Key, Value> pairs as <Country, Casualties>.
c. This output values are iterated to calculate the total casualties per country, a count is also maintained along with that, and at the end the average is calculated.
d. The map and reduce functions are as shown below:

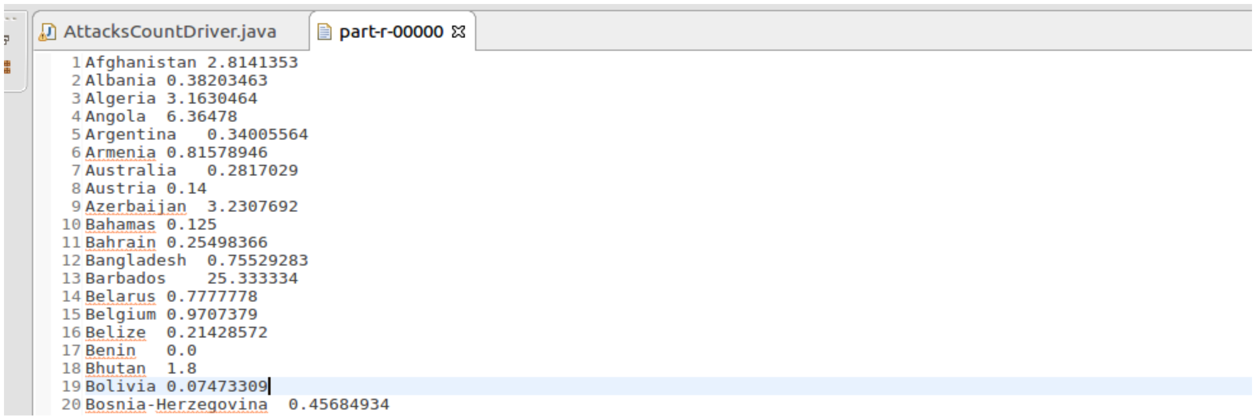


```
public class AvgByNationalityMapper extends Mapper<Object, Text, Text, AvgByNationalityTuple> {

    private Text outNationality = new Text();
    private AvgByNationalityTuple outAvgTuple = new AvgByNationalityTuple();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        if (value.toString().contains("eventid")) {
            return;
        } else {
            String[] data = value.toString().split(",");

            if (data[2].matches("\\.") || data[2].matches("\\-")) {
                outAvgTuple.setAvgPeopleKilled(0);
            } else {
                outNationality.set(data[3]);
                outAvgTuple.setAvgPeopleKilled(Float.parseFloat(data[2]));
                context.write(outNationality, outAvgTuple);
            }

        }
    }
}
```



```
public class AvgByNationalityReducer extends Reducer<Text, AvgByNationalityTuple, Text, AvgByNationalityTuple> {

    private AvgByNationalityTuple result = new AvgByNationalityTuple();

    public void reduce(Text key, Iterable<AvgByNationalityTuple> values, Context context)
            throws IOException, InterruptedException {

        result.setAvgPeopleKilled(0);

        float sum = 0;
        int count = 0;

        for (AvgByNationalityTuple value : values) {
            sum += value.getAvgPeopleKilled();
            count += 1;
        }

        float avg = sum / count;
        result.setAvgPeopleKilled(avg);

        context.write(key, result);
    }
}
```

e. The output of the implementation is:



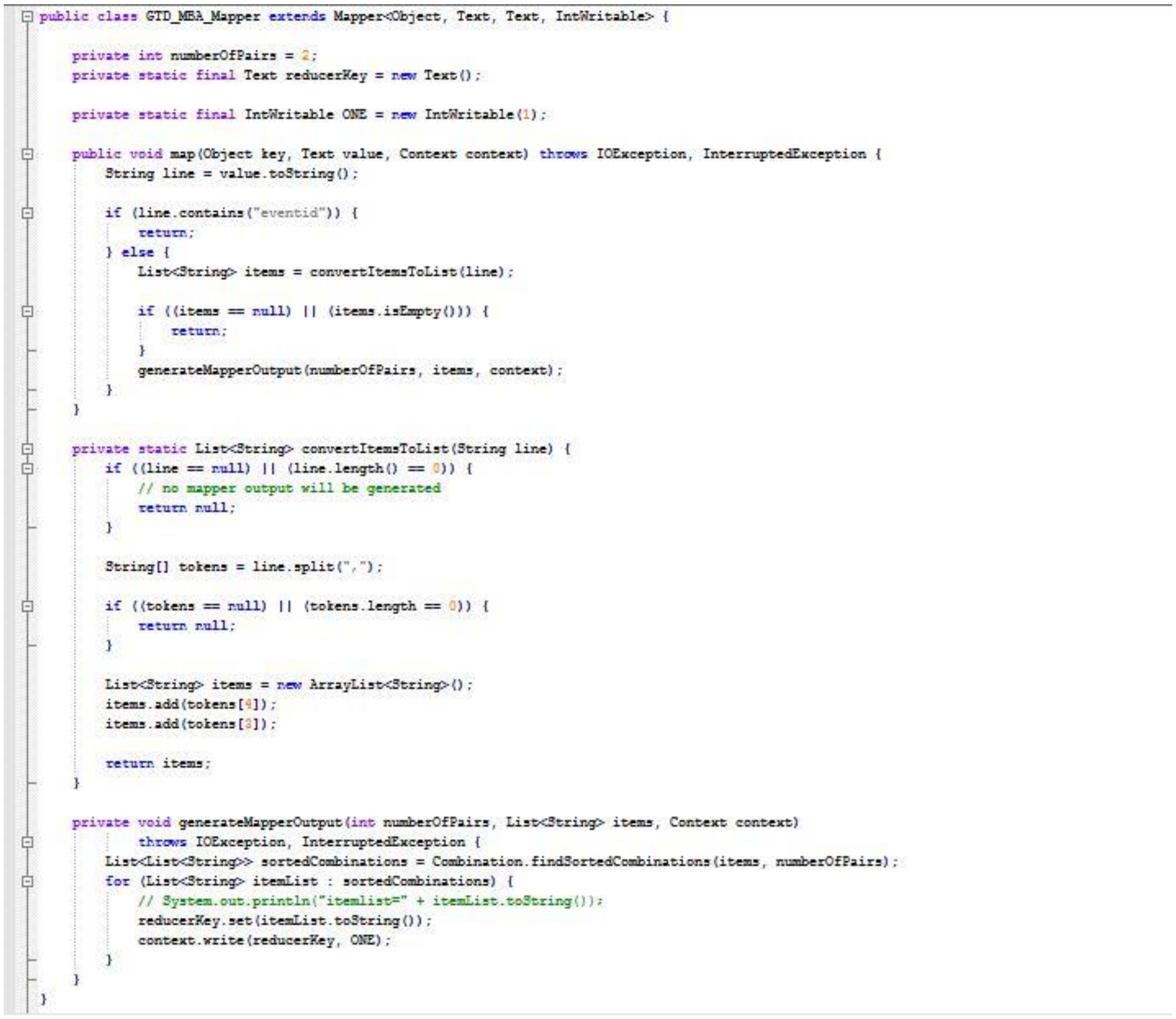
AttacksCountDriver.java part-r-00000

```
Afghanistan 2.8141353
Albania 0.38203463
Algeria 3.1630464
Angola  6.36478
Argentina    0.34005564
Armenia 0.81578946
Australia    0.2817029
Austria 0.14
Azerbaijan  3.2307692
Bahamas 0.125
Bahrain 0.25498366
Bangladesh  0.75529283
Barbados     25.333334
Belarus 0.7777778
Belgium 0.9707379
Belize  0.21428572
Benin   0.0
Bhutan  1.8
Bolivia 0.07473309
Bosnia-Herzegovina  0.45684934
```

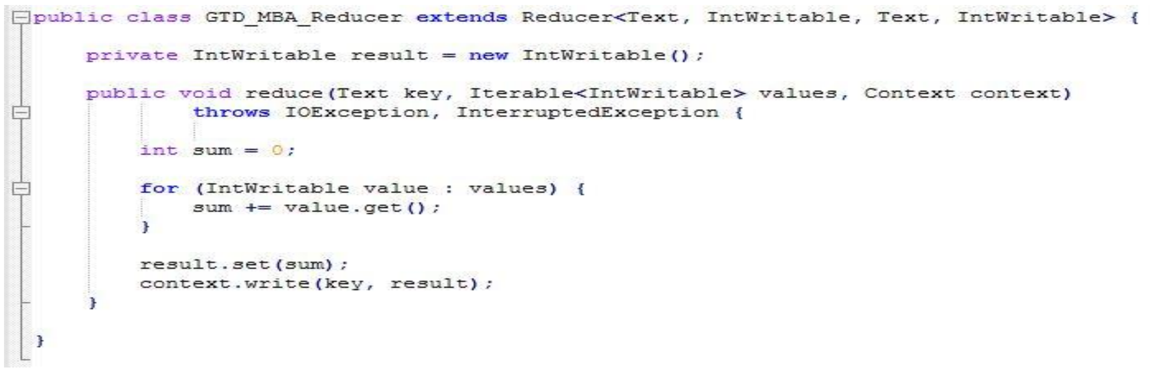
3.9 Performing Market Basket Analysis based on the attack types and countries:

a. Market Basket Analysis (MBA) is a data mining technique, used to reveal affinities between individual items or item groupings.
b. The map-reduce solution in this case, tuples for the order of 2.

c. Here, it is used to identify the counts of the attacks based on a combination of country and attack type, so the tuple will be [Country, AttackType].
d. The mapper function will create a list of items we want to use to create tuples. The helper method from Combination class will be used to generate all the combinations and the output will be generated with count one.
e. The reducer function will simply calculate the different values for each key and emit the total for each key.
f. The mapper and reducer functions for the MBA are as shown below:

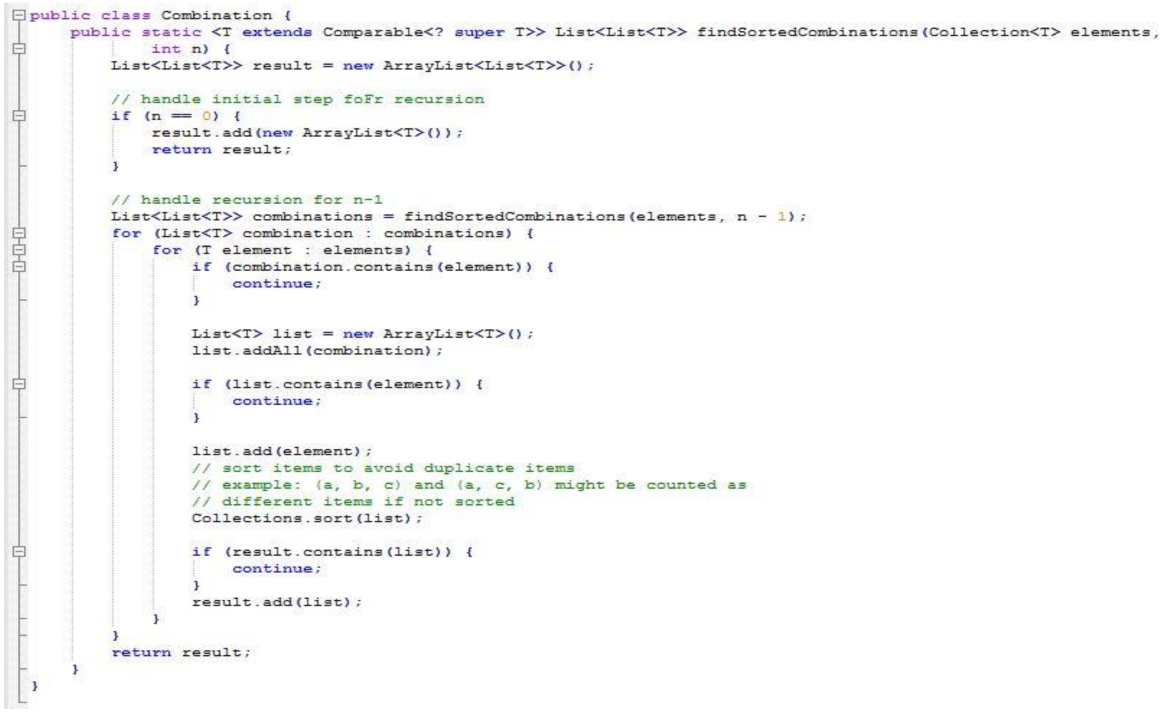


```
public class GTD_MBA_Mapper extends Mapper<Object, Text, Text, IntWritable> {

    private int numberOfPairs = 2;
    private static final Text reducerKey = new Text();

    private static final IntWritable ONE = new IntWritable(1);

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();

        if (line.contains("eventid")) {
            return;
        } else {
            List<String> items = convertItemsToList(line);

            if ((items == null) || (items.isEmpty())) {
                return;
            }
            generateMapperOutput(numberOfPairs, items, context);
        }
    }

    private static List<String> convertItemsToList(String line) {
        if ((line == null) || (line.length() == 0)) {
            // no mapper output will be generated
            return null;
        }

        String[] tokens = line.split(",");

        if ((tokens == null) || (tokens.length == 0)) {
            return null;
        }

        List<String> items = new ArrayList<String>();
        items.add(tokens[4]);
        items.add(tokens[3]);

        return items;
    }

    private void generateMapperOutput(int numberOfPairs, List<String> items, Context context)
            throws IOException, InterruptedException {
        List<List<String>> sortedCombinations = Combination.findSortedCombinations(items, numberOfPairs);
        for (List<String> itemList : sortedCombinations) {
            // System.out.println("itemlist=" + itemList.toString());
            reducerKey.set(itemList.toString());
            context.write(reducerKey, ONE);
        }
    }
}
```



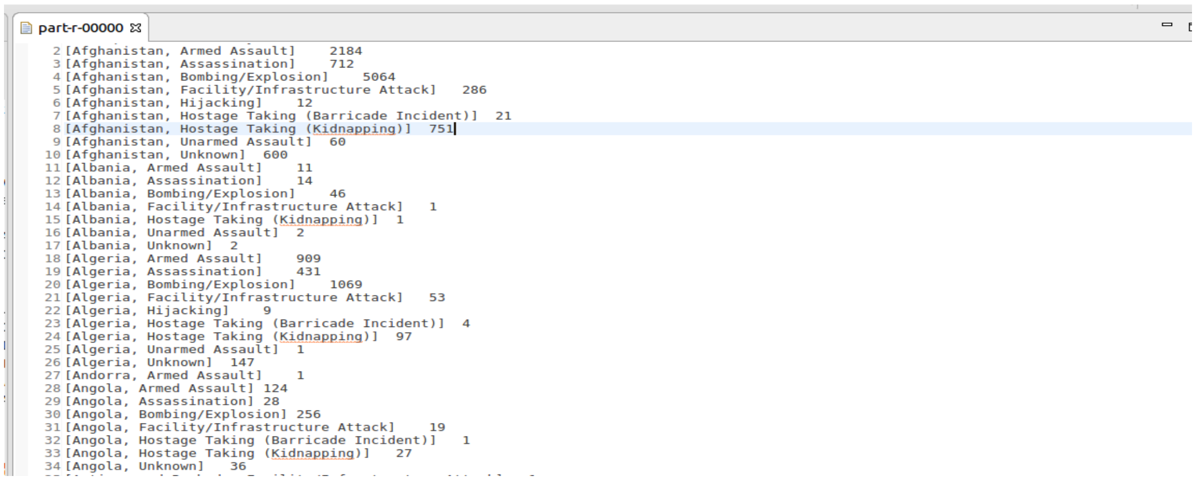
```java
public class GTD_MBA_Reducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {

        int sum = 0;

        for (IntWritable value : values) {
            sum += value.get();
        }

        result.set(sum);
        context.write(key, result);
    }

}
```

g. The helper class Combination is:



```java
public class Combination {
    public static <T extends Comparable<? super T>> List<List<T>> findSortedCombinations(Collection<T> elements,
            int n) {
        List<List<T>> result = new ArrayList<List<T>>();

        // handle initial step foFr recursion
        if (n == 0) {
            result.add(new ArrayList<T>());
            return result;
        }

        // handle recursion for n-1
        List<List<T>> combinations = findSortedCombinations(elements, n - 1);
        for (List<T> combination : combinations) {
            for (T element : elements) {
                if (combination.contains(element)) {
                    continue;
                }

                List<T> list = new ArrayList<T>();
                list.addAll(combination);

                if (list.contains(element)) {
                    continue;
                }

                list.add(element);
                // sort items to avoid duplicate items
                // example: (a, b, c) and (a, c, b) might be counted as
                // different items if not sorted
                Collections.sort(list);

                if (result.contains(list)) {
                    continue;
                }
                result.add(list);
            }
        }
        return result;
    }
}
```

h. The output of MBA looks like:

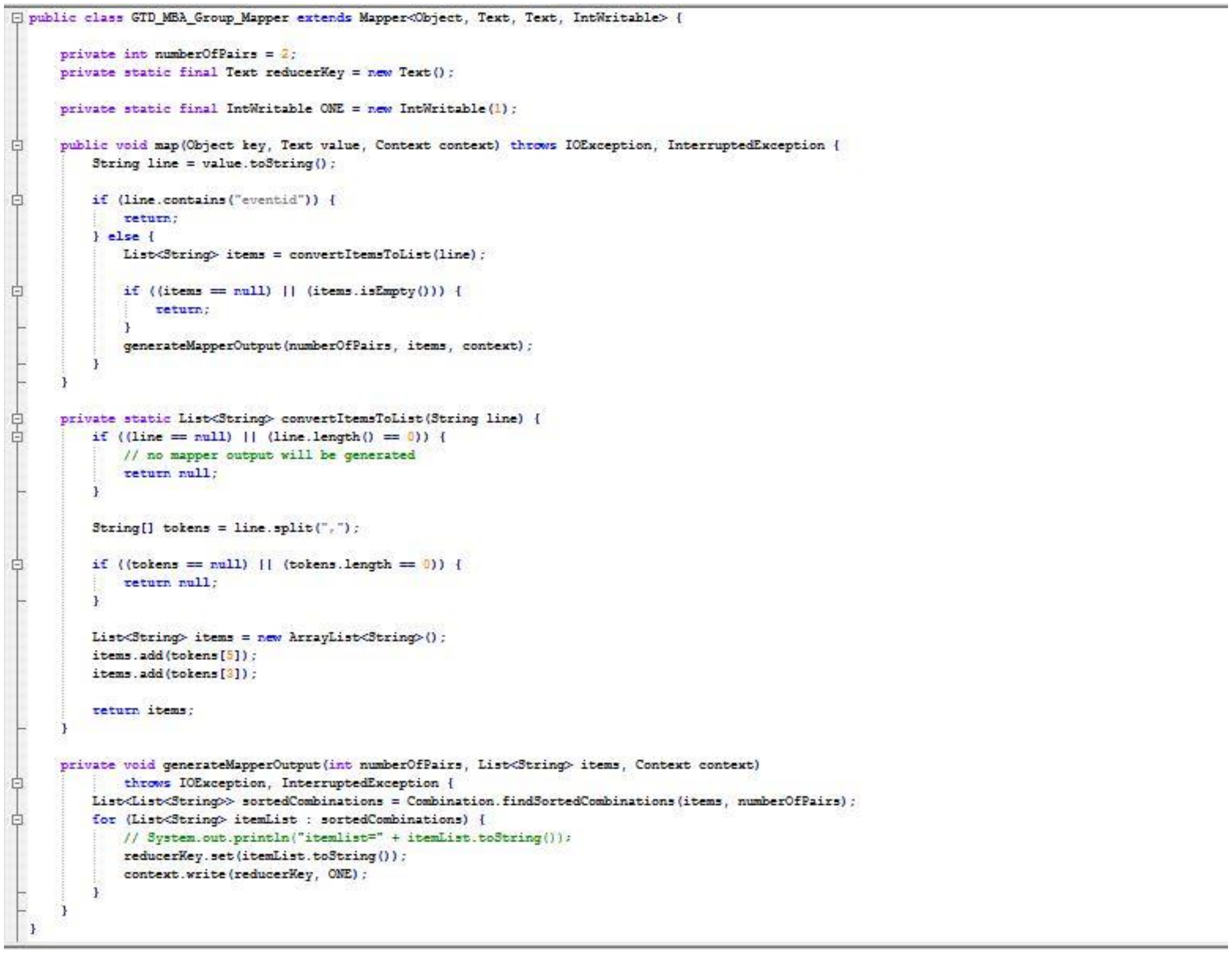


part-r-00000

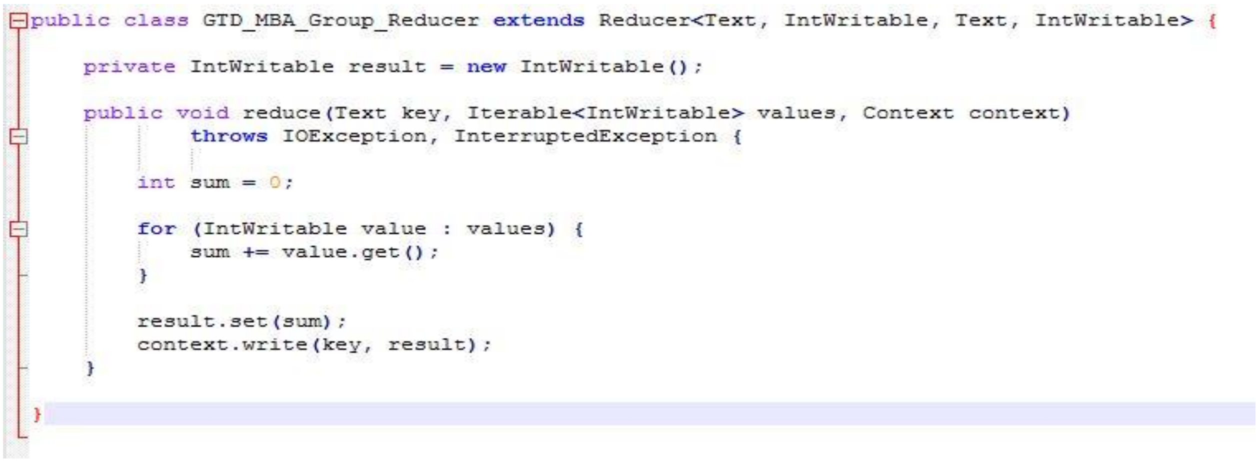
```
[Afghanistan, Armed Assault]    2184
[Afghanistan, Assassination]    712
[Afghanistan, Bombing/Explosion]    5064
[Afghanistan, Facility/Infrastructure Attack]   286
[Afghanistan, Hijacking]    12
[Afghanistan, Hostage Taking (Barricade Incident)]  21
[Afghanistan, Hostage Taking (Kidnapping)]  751
[Afghanistan, Unarmed Assault]  60
[Afghanistan, Unknown]  600
[Albania, Armed Assault]    11
[Albania, Assassination]    14
[Albania, Bombing/Explosion]    46
[Albania, Facility/Infrastructure Attack]   1
[Albania, Hostage Taking (Kidnapping)]  1
[Albania, Unarmed Assault]  2
[Albania, Unknown]  2
[Algeria, Armed Assault]    909
[Algeria, Assassination]    431
[Algeria, Bombing/Explosion]    1069
[Algeria, Facility/Infrastructure Attack]   53
[Algeria, Hijacking]    9
[Algeria, Hostage Taking (Barricade Incident)]  4
[Algeria, Hostage Taking (Kidnapping)]  97
[Algeria, Unarmed Assault]  1
[Algeria, Unknown]  147
[Andorra, Armed Assault]    1
[Angola, Armed Assault] 124
[Angola, Assassination] 28
[Angola, Bombing/Explosion] 256
[Angola, Facility/Infrastructure Attack]    19
[Angola, Hostage Taking (Barricade Incident)]   1
[Angola, Hostage Taking (Kidnapping)]   27
[Angola, Unknown]   36
```

3.10 Performing Market Basket Analysis based on the attack types and the attacking group:

a. Similar to the MBA mentioned above, this analysis is used to find out the frequencies for combinations of attacking groups and attack types.
b. This can be used to analyze and the types of attacks a group is more likely to plan.
c. The Mapper and reducer functions are:



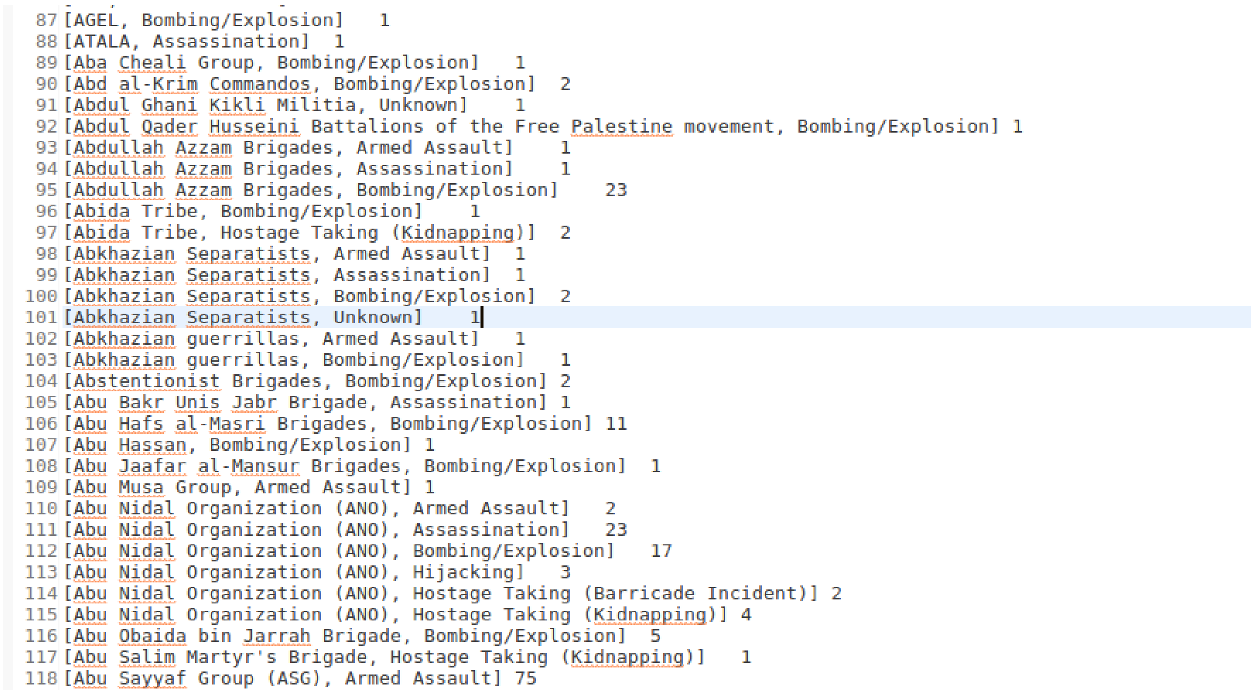
```java
public class GTD_MBA_Group_Mapper extends Mapper<Object, Text, Text, IntWritable> {

    private int numberOfPairs = 2;
    private static final Text reducerKey = new Text();

    private static final IntWritable ONE = new IntWritable(1);

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();

        if (line.contains("eventid")) {
            return;
        } else {
            List<String> items = convertItemsToList(line);

            if ((items == null) || (items.isEmpty())) {
                return;
            }
            generateMapperOutput(numberOfPairs, items, context);
        }
    }

    private static List<String> convertItemsToList(String line) {
        if ((line == null) || (line.length() == 0)) {
            // no mapper output will be generated
            return null;
        }

        String[] tokens = line.split(",");

        if ((tokens == null) || (tokens.length == 0)) {
            return null;
        }

        List<String> items = new ArrayList<String>();
        items.add(tokens[5]);
        items.add(tokens[3]);

        return items;
    }

    private void generateMapperOutput(int numberOfPairs, List<String> items, Context context)
            throws IOException, InterruptedException {
        List<List<String>> sortedCombinations = Combination.findSortedCombinations(items, numberOfPairs);
        for (List<String> itemList : sortedCombinations) {
            // System.out.println("itemlist=" + itemList.toString());
            reducerKey.set(itemList.toString());
            context.write(reducerKey, ONE);
        }
    }
}
```



```
public class GTD_MBA_Group_Reducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {

        int sum = 0;

        for (IntWritable value : values) {
            sum += value.get();
        }

        result.set(sum);
        context.write(key, result);
    }

}
```

d. The Combination class is same as the one shown in the previous technique.

e. The output of this analysis looks like:



```
[AGEL, Bombing/Explosion]    1
[ATALA, Assassination]  1
[Aba Cheali Group, Bombing/Explosion]    1
[Abd al-Krim Commandos, Bombing/Explosion]   2
[Abdul Ghani Kikli Militia, Unknown]     1
[Abdul Qader Husseini Battalions of the Free Palestine movement, Bombing/Explosion] 1
[Abdullah Azzam Brigades, Armed Assault]     1
[Abdullah Azzam Brigades, Assassination]     1
[Abdullah Azzam Brigades, Bombing/Explosion]    23
[Abida Tribe, Bombing/Explosion]     1
[Abida Tribe, Hostage Taking (Kidnapping)]   2
[Abkhazian Separatists, Armed Assault]   1
[Abkhazian Separatists, Assassination]   1
[Abkhazian Separatists, Bombing/Explosion]   2
[Abkhazian Separatists, Unknown]     1
[Abkhazian guerrillas, Armed Assault]    1
[Abkhazian guerrillas, Bombing/Explosion]    1
[Abstentionist Brigades, Bombing/Explosion] 2
[Abu Bakr Unis Jabr Brigade, Assassination] 1
[Abu Hafs al-Masri Brigades, Bombing/Explosion] 11
[Abu Hassan, Bombing/Explosion] 1
[Abu Jaafar al-Mansur Brigades, Bombing/Explosion]   1
[Abu Musa Group, Armed Assault] 1
[Abu Nidal Organization (ANO), Armed Assault]   2
[Abu Nidal Organization (ANO), Assassination]   23
[Abu Nidal Organization (ANO), Bombing/Explosion]   17
[Abu Nidal Organization (ANO), Hijacking]   3
[Abu Nidal Organization (ANO), Hostage Taking (Barricade Incident)] 2
[Abu Nidal Organization (ANO), Hostage Taking (Kidnapping)] 4
[Abu Obaida bin Jarrah Brigade, Bombing/Explosion]  5
[Abu Salim Martyr's Brigade, Hostage Taking (Kidnapping)]   1
[Abu Sayyaf Group (ASG), Armed Assault] 75
```

4. APPENDIX:

4.1 Finding out the total attacks per year

a. YearAttacksMapper.java

package adbms.finalproj;

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

public class YearAttacksMapper extends Mapper<Object, Text, Text, IntWritable> {

    private Text outYear = new Text();
    private IntWritable outCount = new IntWritable();

    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {

            String[] data = value.toString().split(",");
            if (data.length > 1) {
                    String year = data[1];
                    if (!year.equals("year") && year.length() == 4) {
                            outYear.set(year);
                            outCount.set(1);
                            context.write(outYear, outCount);

                    }
            }
    }
}
```

b. YearAttacksReducer.java

```
package adbms.finalproj;

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

public class YearAttacksReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable attackCount = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
                    throws IOException, InterruptedException {
```

```
		int count = 0;
		for (IntWritable value : values) {
			count += value.get();
		}
		attackCount.set(count);
		context.write(key, attackCount);
	}
}
```

c. YearAttacksDriver.java

```
package adbms.finalproj;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class YearCountDriver {

	public static void main(String[] args) throws Exception {
		// TODO Auto-generated method stub
		System.out.println("Global Terrorism Database");

		Configuration conf = new Configuration();
		Job job = Job.getInstance(conf, "Attacks Per Year");

		job.setJarByClass(YearCountDriver.class);
		job.setMapperClass(YearAttacksMapper.class);
		job.setReducerClass(YearAttacksReducer.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(true) ? 0 : 1);
		job.waitForCompletion(true);
```

```
        System.out.println("Graph Generation.");

        Path outputPath = new Path(args[1] + "/part-r-00000");
        FileSystem hdfs = FileSystem.get(conf);

        GraphHelper graphHelper = new GraphHelper();
        graphHelper.createGraph(hdfs, outputPath);
    }
}
```

d. GraphHelper.java

```
package adbms.finalproj;

import java.awt.Color;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;

public class GraphHelper {

    public void createGraph(FileSystem fileSystem, Path outputPath) {
        try {
            CategoryDataset dataset = createDataset(fileSystem, outputPath);
            JFreeChart chart = createChart(dataset);

            ChartFrame frame = new ChartFrame("Attacks Per Year", chart);
            frame.setVisible(true);
            frame.setSize(650, 650);

            System.out.print("Graph Created.");
```

```
		} catch (Exception ex) {
			// TODO: handle exception
			System.out.println("Graph cannot be created at this time. Please 
run on stand alone mode.");
		}
	}

	public CategoryDataset createDataset(FileSystem fileSystem, Path outputPath) {
		DefaultCategoryDataset graphData = new DefaultCategoryDataset();
		String seriesName = "Attacks";

		try (BufferedReader br = new BufferedReader(new 
InputStreamReader(fileSystem.open(outputPath)))) {

			String sCurrentLine;
			int i = 1;
			while ((sCurrentLine = br.readLine()) != null) {
				if (i == 1 || i % 5 == 0) {
					String[] data = sCurrentLine.split("\t");
					graphData.addValue(Integer.parseInt(data[1]), 
seriesName, data[0]);
				}
				i++;
			}
		} catch (IOException e) {
			e.printStackTrace();
		}

		return graphData;
	}

	private JFreeChart createChart(CategoryDataset dataset) {
		// create the chart...
		JFreeChart chart = ChartFactory.createLineChart("Line Chart - Attacks Per 
Year", // chart title
				"Year", // domain axis label
				"Attacks Count", // range axis label
				dataset, // data
				PlotOrientation.VERTICAL, // orientation
				true, // include legend
				true, // tooltips
				false // urls
		);
```

```
                chart.setBackgroundPaint(Color.white);

                CategoryPlot plot = (CategoryPlot) chart.getPlot();
                plot.setBackgroundPaint(Color.lightGray);
                plot.setRangeGridlinePaint(Color.white);

                return chart;
        }
}
```

4.2 Finding out the total attacks in each country

CountryAttacksMapper.java

```
package adbms.finalproj;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class CountryAttacksMapper extends Mapper<Object, Text, Text, IntWritable>
{
        private Text outCountry = new Text();
        private IntWritable outCount = new IntWritable();

        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
                String[] data = value.toString().split(",");
                if (data.length > 1) {
                        String country = data[4];
                        if (!country.equals("country") && !country.matches("^[0-9]")) {
                                outCountry.set(country);
                                outCount.set(1);
                                context.write(outCountry, outCount);

                        }
                }
        }
}
```

a. CountryAttacksReducer.java

```
package adbms.finalproj;
```

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class CountryAttacksReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
    private IntWritable attackCount = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {
        int count = 0;
        for (IntWritable value : values) {
                count += value.get();
        }

        attackCount.set(count);
        context.write(key, attackCount);
    }
}
```

b. CountryAttacksDriver.java

```
package adbms.finalproj;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class CountryCountDriver {

    public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub
        System.out.println("Global Terrorism Database");

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Attacks Per Country");
```

```
		job.setJarByClass(CountryCountDriver.class);
		job.setMapperClass(CountryAttacksMapper.class);
		job.setReducerClass(CountryAttacksReducer.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(true) ? 0 : 1);

		// job.waitForCompletion(true);
		//
		// System.out.println("Graph Generation.");
		//
		// Path outputPath = new Path(args[1] + "/part-r-00000");
		// FileSystem hdfs = FileSystem.get(conf);
		//
		// GraphHelper graphHelper = new GraphHelper();
		// graphHelper.createGraph(hdfs, outputPath);
	}
}
```

GraphHelper.java

```
package adbms.finalproj;

import java.awt.Color;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;
```

c. public class GraphHelper {

```
	public void createGraph(FileSystem fileSystem, Path outputPath) {
		CategoryDataset dataset = createDataset(fileSystem, outputPath);
		JFreeChart chart = createChart(dataset);

		ChartFrame frame = new ChartFrame("Attacks Per Country", chart);
		frame.setVisible(true);
		frame.setSize(650, 650);
	}

	public CategoryDataset createDataset(FileSystem fileSystem, Path outputPath) {
		System.out.println("Create DataSet Called");
		DefaultCategoryDataset graphData = new DefaultCategoryDataset();
		String seriesName = "Attacks";

		try (BufferedReader br = new BufferedReader(new
InputStreamReader(fileSystem.open(outputPath)))) {

			String sCurrentLine;
			int i = 1;
			while ((sCurrentLine = br.readLine()) != null) {

				String[] data = sCurrentLine.split("\t");
				if (Integer.parseInt(data[1]) > 4000) {
					graphData.addValue(Integer.parseInt(data[1]),
seriesName, data[0]);
				}
			}

		} catch (IOException e) {
			e.printStackTrace();
		}

		return graphData;
	}

	private JFreeChart createChart(CategoryDataset dataset) {
		// create the chart...
		JFreeChart chart = ChartFactory.createLineChart("Line Chart - Attacks Per
Country", // chart title
				"Country", // domain axis label
				"Attacks Count", // range axis label
				dataset, // data
				PlotOrientation.VERTICAL, // orientation
```

```
                        true, // include legend
                        true, // tooltips
                        false // urls
                );

                chart.setBackgroundPaint(Color.white);

                CategoryPlot plot = (CategoryPlot) chart.getPlot();
                plot.setBackgroundPaint(Color.lightGray);
                plot.setRangeGridlinePaint(Color.white);

                return chart;
    }
}
```

4.3 Finding out the total attack per attack type

AttacksCountMapper.java

```
package adbms.finalproj;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class AttacksCountMapper extends Mapper<Object, Text, Text, IntWritable> {
    private Text outType = new Text();
    private IntWritable outCount = new IntWritable();

    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
                String[] data = value.toString().split(",");
                if (data.length > 1) {
                        String type = data[3];
                        if (!type.equals("attacktype") && !type.equals(".") && !type.equals("-
")) {
                                outType.set(type);
                                outCount.set(1);

                                context.write(outType, outCount);
                        }
```

```
        }
    }
}
```

AttacksCountReducer.java

```
package adbms.finalproj;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class AttacksCountReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
    private IntWritable attackCount = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {

        int count = 0;
        for (IntWritable value : values) {
            count += value.get();
        }
        attackCount.set(count);
        context.write(key, attackCount);
    }
}
```

AttacksCountDriver.java

```
package adbms.finalproj;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class AttacksCountDriver {
    public static void main(String[] args) throws Exception {
            // TODO Auto-generated method stub
            System.out.println("Global Terrorism Database");

            Configuration conf = new Configuration();
            Job job = Job.getInstance(conf, "Attacks Per Attack Type");

            job.setJarByClass(AttacksCountDriver.class);
            job.setMapperClass(AttacksCountMapper.class);
            job.setReducerClass(AttacksCountReducer.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(true) ? 0 : 1);

            // job.waitForCompletion(true);
            //
            // System.out.println("Graph Generation.");
            //
            // Path outputPath = new Path(args[1] + "/part-r-00000");
            // FileSystem hdfs = FileSystem.get(conf);
            //
            // GraphHelper graphHelper = new GraphHelper();
            // graphHelper.createGraph(hdfs, outputPath);
    }
}
```

GraphHelper.java

```
package adbms.finalproj;

import java.awt.Color;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;

public class GraphHelper {

    public void createGraph(FileSystem fileSystem, Path outputPath) {
        try {
            CategoryDataset dataset = createDataset(fileSystem, outputPath);
            JFreeChart chart = createChart(dataset);

            ChartFrame frame = new ChartFrame("Attacks Per Attack Types",
chart);
            frame.setVisible(true);
            frame.setSize(650, 650);

            System.out.print("Graph Created.");
        } catch (Exception ex) {
            // TODO: handle exception
            System.out.println("Graph cannot be created at this time. Please
run on stand alone mode.");
        }
    }

    public CategoryDataset createDataset(FileSystem fileSystem, Path outputPath) {
        DefaultCategoryDataset graphData = new DefaultCategoryDataset();
        String seriesName = "Attacks";

        try (BufferedReader br = new BufferedReader(new
InputStreamReader(fileSystem.open(outputPath)))) {

            String sCurrentLine;
            int i = 1;
            while ((sCurrentLine = br.readLine()) != null) {

                String[] data = sCurrentLine.split("\t");
                graphData.addValue(Integer.parseInt(data[1]), seriesName,
data[0]);

            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return graphData;
    }

    private JFreeChart createChart(CategoryDataset dataset) {
            // create the chart...
            JFreeChart chart = ChartFactory.createBarChart("Bar Chart - Attacks Per
Attack Type", // chart title
                            "Types", // domain axis label
                            "Attacks Count", // range axis label
                            dataset, // data
                            PlotOrientation.VERTICAL, // orientation
                            true, // include legend
                            true, // tooltips
                            false // urls
            );

            chart.setBackgroundPaint(Color.white);

            CategoryPlot plot = (CategoryPlot) chart.getPlot();
            plot.setBackgroundPaint(Color.lightGray);
            plot.setRangeGridlinePaint(Color.white);

            return chart;
    }
}
```

4.4 Finding out the distinct types of attacks

AttackTypeMapper.java

```
package adbms.finalproj;

import java.io.IOException;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class AttackTypeMapper extends Mapper<Object, Text, Text, NullWritable> {
    private Text outAttackType = new Text();

    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {

            if (!value.toString().contains("eventid")) {
                    String type = value.toString().split(",")[3];
                    if (!type.equals(".") && !type.equals("-")) {
                            outAttackType.set(type);
                            context.write(outAttackType, NullWritable.get());
                    }
            }
    }
}
```

AttackTypeReducer.java

```
package adbms.finalproj;

import java.io.IOException;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class AttackTypeReducer extends Reducer<Text, NullWritable, Text,
NullWritable> {
    public void reduce(Text key, Iterable<NullWritable> values, Context context)
                    throws IOException, InterruptedException {

            context.write(key, NullWritable.get());
    }
}
```

AttackTypeDriver.java

```
package adbms.finalproj;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class AttackTypeDriver {

    public static void main(String[] args) throws Exception {
            // TODO Auto-generated method stub
```

```
		Configuration conf = new Configuration();
		Job job = Job.getInstance(conf, "Distinct IP");

		job.setJarByClass(AttackTypeDriver.class);
		job.setMapperClass(AttackTypeMapper.class);
		job.setCombinerClass(AttackTypeReducer.class);
		job.setReducerClass(AttackTypeReducer.class);
		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(NullWritable.class);
		FileInputFormat.addInputPath(job, new Path(args[0]));
		FileOutputFormat.setOutputPath(job, new Path(args[1]));
		System.exit(job.waitForCompletion(true) ? 0 : 1);
	}
}
```

4.5 Finding out the distinct attacking groups

DistinctGroupMapper.java

```
package adbms.finalproj;

import java.io.IOException;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class DistinctGroupMapper extends Mapper<Object, Text, Text, NullWritable>
{
	private Text outAttackType = new Text();

	public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {

		if (!value.toString().contains("eventid")) {
			String type = value.toString().split(",")[5];

			if (!type.equals(".") && !type.equals("-")) {
				outAttackType.set(type);
				context.write(outAttackType, NullWritable.get());
			}
		}
	}
}
```

DistinctGroupReducer.java

```
package adbms.finalproj;

import java.io.IOException;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class DistinctGroupReducer extends Reducer<Text, NullWritable, Text,
NullWritable> {
    public void reduce(Text key, Iterable<NullWritable> values, Context context)
                throws IOException, InterruptedException {

        context.write(key, NullWritable.get());

    }
}
```

DistinctGroupDriver.java

```
package adbms.finalproj;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class DistinctGroupDriver {

    public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Distinct IP");

        job.setJarByClass(DistinctGroupDriver.class);
        job.setMapperClass(DistinctGroupMapper.class);
        job.setCombinerClass(DistinctGroupReducer.class);
        job.setReducerClass(DistinctGroupReducer.class);
```

```
		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(NullWritable.class);
		FileInputFormat.addInputPath(job, new Path(args[0]));
		FileOutputFormat.setOutputPath(job, new Path(args[1]));
		System.exit(job.waitForCompletion(true) ? 0 : 1);
	}
}
```

4.6 Sorting the attacks per country in descending order

CountryAttacksPair.java

```
package adbms.finalproj;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;

public class CountryAttacksPair implements Writable,
WritableComparable<CountryAttacksPair> {

	private Text country = new Text();
	private IntWritable count = new IntWritable();

	public Text getCountry() {
		return country;
	}

	public void setCountry(Text country) {
		this.country = country;
	}

	public IntWritable getCount() {
		return count;
	}

	public void setCount(IntWritable count) {
		this.count = count;
	}
```

```
    public void write(DataOutput out) throws IOException {
            country.write(out);
            count.write(out);
    }

    public void readFields(DataInput in) throws IOException {
            country.readFields(in);
            count.readFields(in);
    }

    public int compareTo(CountryAttacksPair countryPair) {
            int compareValue = this.country.compareTo(countryPair.getCountry());
            if (compareValue == 0) {
                    return count.compareTo(countryPair.getCount());
            }
            return compareValue;
    }

    public String toString() {
            return country.toString() + "\t" + count.get();
    }
}
```

CountryAttacksPairComaprator.java

```
package adbms.finalproj;

import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;

public class CountryAttacksPairComparator extends WritableComparator {
    protected CountryAttacksPairComparator() {
            super(CountryAttacksPair.class, true);
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {
            CountryAttacksPair key1 = (CountryAttacksPair) a;
            CountryAttacksPair key2 = (CountryAttacksPair) b;

            int result = key1.getCount().get() < key2.getCount().get() ? 1
                            : key1.getCount().get() == key2.getCount().get() ? 0 : -1;
            return result;
    }
}
```

CountryAttacksMapper.java

```
package adbms.finalproj;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class CountryAttacksMapper extends Mapper<Object, Text, Text, IntWritable>
{
    private Text outCountry = new Text();
    private IntWritable outCount = new IntWritable();

    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {

            String[] data = value.toString().split(",");

            if (data.length > 1) {

                    String country = data[4];

                    if (!country.equals("country") && !country.matches("^[0-9]")) {

                            outCountry.set(country);
                            outCount.set(1);

                            context.write(outCountry, outCount);
                    }

            }
    }
}
```

CountryAttacksReducer.java

```
package adbms.finalproj;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class CountryAttacksReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
    private IntWritable attackCount = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {

        int count = 0;

        for (IntWritable value : values) {
            count += value.get();
        }

        attackCount.set(count);
        context.write(key, attackCount);
    }
}
```

CountryAttacksPairMapper.java

```
package adbms.finalproj;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class CountryAttacksPairMapper extends Mapper<Object, Text,
CountryAttacksPair, NullWritable> {

    private CountryAttacksPair outData = new CountryAttacksPair();

    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {

        String[] data = value.toString().split("\t");
        outData.setCountry(new Text(data[0]));
        outData.setCount(new IntWritable(Integer.parseInt(data[1])));
```

```
            context.write(outData, NullWritable.get());
    }
}
```

CountryAttacksPairReducer.java

```
package adbms.finalproj;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class CountryAttacksPairReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {

    public void reduce(Text key, IntWritable value, Context context) throws
IOException, InterruptedException {

            context.write(key, value);
    }
}
```

SecondarySortDriver.java

```
package adbms.finalproj;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class SecondarySortDriver {

    private static final String INT_OUTPUT_PATH = "int_output";

    public static void main(String[] args) throws Exception {
            // TODO Auto-generated method stub
```

```
		System.out.println("Global Terrorism Database");

		Configuration conf = new Configuration();
		Job job = Job.getInstance(conf, "Attacks Per Country");

		job.setJarByClass(SecondarySortDriver.class);
		job.setMapperClass(CountryAttacksMapper.class);
		job.setReducerClass(CountryAttacksReducer.class);
		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(IntWritable.class);
		FileInputFormat.addInputPath(job, new Path(args[0]));
		FileOutputFormat.setOutputPath(job, new Path(INT_OUTPUT_PATH));

		job.waitForCompletion(true);

		Job jobSort = new Job(conf, "Ascending Attacks Per Country");

		jobSort.setMapperClass(CountryAttacksPairMapper.class);
		// jobSort.setReducerClass(CountryAttacksPairReducer.class);

		jobSort.setSortComparatorClass(CountryAttacksPairComparator.class);

		jobSort.setNumReduceTasks(1);
		jobSort.setOutputKeyClass(CountryAttacksPair.class);
		jobSort.setOutputValueClass(NullWritable.class);
		FileInputFormat.addInputPath(jobSort, new Path(INT_OUTPUT_PATH));
		FileOutputFormat.setOutputPath(jobSort, new Path(args[1]));

		System.exit(jobSort.waitForCompletion(true) ? 0 : 1);

	}
}
```

4.7 Finding out Top N casualties based on nationality/country:

Top10CasultiesTuple.java

```
package adbms.finalproj;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.Writable;
```

```
public class Top10CasualtiesTuple implements Writable {
    private String year;
    private String country;
    private double casualities;

    public String getYear() {
            return year;
    }

    public void setYear(String year) {
            this.year = year;
    }

    public String getCountry() {
            return country;
    }

    public void setCountry(String country) {
            this.country = country;
    }

    public double getCasualities() {
            return casualities;
    }

    public void setCasualities(double casualities) {
            this.casualities = casualities;
    }

    public void readFields(DataInput in) throws IOException {
            year = in.readUTF();
            country = in.readUTF();
            casualities = in.readDouble();
    }

    public void write(DataOutput out) throws IOException {
            out.writeUTF(year);
            out.writeUTF(country);
            out.writeDouble(casualities);
    }

    public String toString() {
            return year + "\t" + country + "\t" + casualities;
    }
}
```

Top10CasualtiesMapper.java

```
package adbms.finalproj;

import java.io.IOException;
import java.util.TreeMap;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class Top10CasualtiesMapper extends Mapper<Object, Text, NullWritable,
Text> {
    // private SortedMap<Long, Top10CasualtiesTuple> outTreeMap = new
    // TreeMap<Long, Top10CasualtiesTuple>();
    private TreeMap<Double, Top10CasualtiesTuple> outTreeMap = new
TreeMap<Double, Top10CasualtiesTuple>();

    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {

            Configuration conf = context.getConfiguration();
            int N = Integer.parseInt(conf.get("topN"));

            String[] data = value.toString().split(",");
            // System.out.println(data.length);
            if (data.length > 1 && !data[0].equals("eventid")) {

                    Top10CasualtiesTuple tupleData = new Top10CasualtiesTuple();
                    tupleData.setYear(data[1]);
                    tupleData.setCountry(data[3]);

                    if (data[2].matches("\\.") || data[2].matches("\\-")) {
                            tupleData.setCasualities(0);
                    } else {
                            tupleData.setCasualities(Double.parseDouble(data[2]));
                    }

                    outTreeMap.put(tupleData.getCasualities(), tupleData);
            if (outTreeMap.size() > N) {
                outTreeMap.remove(outTreeMap.firstKey());
            }

        }
    }

    protected void cleanup(Context context) throws IOException,
InterruptedException {
        for (Top10CasualtiesTuple value : outTreeMap.values()) {
            context.write(NullWritable.get(), new Text(value.toString()));
        }
    }

}
```

Top10CasualtiesReducer.java

```
package adbms.finalproj;

import java.io.IOException;
import java.util.TreeMap;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class Top10CasualtiesReducer extends Reducer<NullWritable, Text,
NullWritable, Text> {

    // private SortedMap<Long, Top10CasualtiesTuple> outTreeMap = new
    // TreeMap<Long, Top10CasualtiesTuple>();
    private TreeMap<Double, Top10CasualtiesTuple> outTreeMap = new
TreeMap<Double, Top10CasualtiesTuple>();

    public void reduce(NullWritable key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

        Configuration conf = context.getConfiguration();
        int N = Integer.parseInt(conf.get("topN"));

        for (Text value : values) {
            String[] data = value.toString().split("\t");
```

```
			double casualties = Double.parseDouble(data[2]);
			Top10CasualtiesTuple tuple = new Top10CasualtiesTuple();
			tuple.setYear(data[0]);
			tuple.setCountry(data[1]);
			tuple.setCasualities(casualties);
			outTreeMap.put(casualties, tuple);

			if (outTreeMap.size() > N) {
				outTreeMap.remove(outTreeMap.firstKey());
			}

		}

		for (Top10CasualtiesTuple value : outTreeMap.descendingMap().values())
{
			context.write(NullWritable.get(), new Text(value.toString()));
		}

	}
}
```

Top10CasualtiesDriver.java

```
package adbms.finalproj;

import java.util.Scanner;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Top10CasualtiesDriver {

	public static void main(String[] args) throws Exception {
		// TODO Auto-generated method stub
		System.out.println("Global Terrorism Database");

		Scanner scan = new Scanner(System.in);
```

```
            System.out.println("Please Enter 'N' For Top N Records: ");
            int N = scan.nextInt();

            Configuration conf = new Configuration();
            conf.set("topN", String.valueOf(N));

            Job job = Job.getInstance(conf, "Top 10 Casualities");

            job.setJarByClass(Top10CasualtiesDriver.class);
            job.setMapperClass(Top10CasualtiesMapper.class);
            job.setReducerClass(Top10CasualtiesReducer.class);
            job.setOutputKeyClass(NullWritable.class);
            job.setOutputValueClass(Text.class);
            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));
            System.exit(job.waitForCompletion(true) ? 0 : 1);

            // job.waitForCompletion(true);
            //
            // System.out.println("Graph Generation.");
            //
            // Path outputPath = new Path(args[1] + "/part-r-00000");
            // FileSystem hdfs = FileSystem.get(conf);
            //
            // GraphHelper graphHelper = new GraphHelper();
            // graphHelper.createGraph(hdfs, outputPath, N);
    }
}
```

GraphHelper.java

```
package adbms.finalproj;

import java.awt.Color;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;

public class GraphHelper {

    public void createGraph(FileSystem fileSystem, Path outputPath, int N) {
        try {
            CategoryDataset dataset = createDataset(fileSystem, outputPath);
            JFreeChart chart = createChart(dataset);

            String name = "Top " + N + " Casualities";
            ChartFrame frame = new ChartFrame(name, chart);
            frame.setVisible(true);
            frame.setSize(650, 650);

            System.out.print("Graph Created.");
        } catch (Exception ex) {
            // TODO: handle exception
            ex.printStackTrace();
            System.out.println(ex);
            System.out.println("Graph cannot be created at this time. Please
run on stand alone mode.");
        }
    }

    public CategoryDataset createDataset(FileSystem fileSystem, Path outputPath) {
        DefaultCategoryDataset graphData = new DefaultCategoryDataset();
        String seriesName = "Casualities";

        try (BufferedReader br = new BufferedReader(new
InputStreamReader(fileSystem.open(outputPath)))) {

            String sCurrentLine;

            while ((sCurrentLine = br.readLine()) != null) {

                String[] data = sCurrentLine.split("\t");
                graphData.addValue(Double.parseDouble(data[2]),
seriesName, data[1]);

            }

        } catch (IOException e) {
```

```
				e.printStackTrace();
			}

			return graphData;
		}

		private JFreeChart createChart(CategoryDataset dataset) {
			// create the chart...
			JFreeChart chart = ChartFactory.createLineChart("Line Chart - Top 10
Casaulities", // chart title
					"Country", // domain axis label
					"Casuality", // range axis label
					dataset, // data
					PlotOrientation.VERTICAL, // orientation
					true, // include legend
					true, // tooltips
					false // urls
			);

			chart.setBackgroundPaint(Color.white);

			CategoryPlot plot = (CategoryPlot) chart.getPlot();
			plot.setBackgroundPaint(Color.lightGray);
			plot.setRangeGridlinePaint(Color.white);

			return chart;
		}
}
```

4.8 Finding out average casualties by nationality/country:

AvgByNationalityTuple.java

```
package adbms.finalproj;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.Writable;

public class AvgByNationalityTuple implements Writable {
	private float avgPeopleKilled = 0;
	public float getAvgPeopleKilled() {
		return avgPeopleKilled;
	}

	public void setAvgPeopleKilled(float avgPeopleKilled) {
		this.avgPeopleKilled = avgPeopleKilled;
	}

	public void readFields(DataInput in) throws IOException {
		avgPeopleKilled = in.readFloat();
	}

	public void write(DataOutput out) throws IOException {
		out.writeFloat(avgPeopleKilled);
	}

	public String toString() {
		return String.valueOf(avgPeopleKilled);
	}
}
```

AvgByNationalityMapper.java

```
package adbms.finalproj;

import java.io.IOException;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class AvgByNationalityMapper extends Mapper<Object, Text, Text,
AvgByNationalityTuple> {

	private Text outNationality = new Text();
	private AvgByNationalityTuple outAvgTuple = new AvgByNationalityTuple();

	public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
		if (value.toString().contains("eventid")) {
			return;
		} else {
			String[] data = value.toString().split(",");

			if (data[2].matches("\\.") || data[2].matches("\\-")) {
```

```
                    outAvgTuple.setAvgPeopleKilled(0);
                } else {
                    outNationality.set(data[3]);
                    outAvgTuple.setAvgPeopleKilled(Float.parseFloat(data[2]));
                    context.write(outNationality, outAvgTuple);
                }

            }
    }
}
```

AvgByNationalityReducer.java

```
package adbms.finalproj;

import java.io.IOException;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class AvgByNationalityReducer extends Reducer<Text,
AvgByNationalityTuple, Text, AvgByNationalityTuple> {

    private AvgByNationalityTuple result = new AvgByNationalityTuple();

    public void reduce(Text key, Iterable<AvgByNationalityTuple> values, Context
context)
                throws IOException, InterruptedException {

            result.setAvgPeopleKilled(0);

            float sum = 0;
            int count = 0;

            for (AvgByNationalityTuple value : values) {
                sum += value.getAvgPeopleKilled();
                count += 1;
            }

            float avg = sum / count;
            result.setAvgPeopleKilled(avg);

            context.write(key, result);
    }
}
```

AvgByNaitionalityDriver.java

```
package adbms.finalproj;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class AvgByNationalityDriver {

    public static void main(String[] args) throws Exception {
            // TODO Auto-generated method stub

            System.out.println("Global Terrorism Database");

            Configuration conf = new Configuration();
            Job job = Job.getInstance(conf, "Attacks Per Year");

            job.setJarByClass(AvgByNationalityDriver.class);
            job.setMapperClass(AvgByNationalityMapper.class);
            job.setCombinerClass(AvgByNationalityReducer.class);
            job.setReducerClass(AvgByNationalityReducer.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(AvgByNationalityTuple.class);
            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));
            System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

4.9 Performing Market Basket Analysis based on the attack types and countries:

Combination.java

```
package adbms.finalproj;

import java.util.ArrayList;
import java.util.Collection;
```

}

```
import java.util.Collections;
import java.util.List;

public class Combination {
    public static <T extends Comparable<? super T>> List<List<T>>
findSortedCombinations(Collection<T> elements,
                    int n) {
            List<List<T>> result = new ArrayList<List<T>>();

            // handle initial step foFr recursion
            if (n == 0) {
                    result.add(new ArrayList<T>());
                    return result;
            }

            // handle recursion for n-1
            List<List<T>> combinations = findSortedCombinations(elements, n - 1);
            for (List<T> combination : combinations) {
                    for (T element : elements) {
                            if (combination.contains(element)) {
                                    continue;
                            }

                            List<T> list = new ArrayList<T>();
                            list.addAll(combination);

                            if (list.contains(element)) {
                                    continue;
                            }

                            list.add(element);
                            // sort items to avoid duplicate items
                            // example: (a, b, c) and (a, c, b) might be counted as
                            // different items if not sorted
                            Collections.sort(list);

                            if (result.contains(list)) {
                                    continue;
                            }
                            result.add(list);
                    }
            }
            return result;
    }
}
```

GTD_MBA_Mapper.java

```
package adbms.finalproj;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class GTD_MBA_Mapper extends Mapper<Object, Text, Text, IntWritable> {

    private int numberOfPairs = 2;
    private static final Text reducerKey = new Text();

    private static final IntWritable ONE = new IntWritable(1);

    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
        String line = value.toString();

        if (line.contains("eventid")) {
            return;
        } else {
            List<String> items = convertItemsToList(line);

            if ((items == null) || (items.isEmpty())) {
                return;
            }
            generateMapperOutput(numberOfPairs, items, context);
        }
    }

    private static List<String> convertItemsToList(String line) {
        if ((line == null) || (line.length() == 0)) {
            // no mapper output will be generated
            return null;
        }

        String[] tokens = line.split(",");
        if ((tokens == null) || (tokens.length == 0)) {
            return null;
        }

        List<String> items = new ArrayList<String>();
        items.add(tokens[4]);
        items.add(tokens[3]);

        return items;
    }

    private void generateMapperOutput(int numberOfPairs, List<String> items,
Context context)
            throws IOException, InterruptedException {
        List<List<String>> sortedCombinations =
Combination.findSortedCombinations(items, numberOfPairs);
        for (List<String> itemList : sortedCombinations) {
            // System.out.println("itemlist=" + itemList.toString());
            reducerKey.set(itemList.toString());
            context.write(reducerKey, ONE);
        }
    }
}
```

GTD_MBA_Reducer.java

```
package adbms.finalproj;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class GTD_MBA_Reducer extends Reducer<Text, IntWritable, Text, IntWritable>
{

    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {

        int sum = 0;
```

```
            for (IntWritable value : values) {
                  sum += value.get();
            }

            result.set(sum);
            context.write(key, result);
      }

}
```

GTD_MBA_Driver.java

```
package adbms.finalproj;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class GTD_MBA_Driver {

      public static void main(String[] args) throws Exception {
            // TODO Auto-generated method stub
            System.out.println("Global Terrorism Database");

            Configuration conf = new Configuration();
            Job job = Job.getInstance(conf, "Attacks Per Country");

            job.setJarByClass(GTD_MBA_Driver.class);
            job.setMapperClass(GTD_MBA_Mapper.class);
            job.setReducerClass(GTD_MBA_Reducer.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(true) ? 0 : 1);
      }
}
```

4.10 Performing Market Basket Analysis based on the attack types and the attacking group:

Combination.java

```
package adbms.finalproj;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.List;

public class Combination {
	public static <T extends Comparable<? super T>> List<List<T>>
findSortedCombinations(Collection<T> elements,
				int n) {
		List<List<T>> result = new ArrayList<List<T>>();

		// handle initial step foFr recursion
		if (n == 0) {
			result.add(new ArrayList<T>());
			return result;
		}

		// handle recursion for n-1
		List<List<T>> combinations = findSortedCombinations(elements, n - 1);
		for (List<T> combination : combinations) {
			for (T element : elements) {
				if (combination.contains(element)) {
					continue;
				}

				List<T> list = new ArrayList<T>();
				list.addAll(combination);

				if (list.contains(element)) {
					continue;
				}

				list.add(element);
				// sort items to avoid duplicate items
				// example: (a, b, c) and (a, c, b) might be counted as
				// different items if not sorted
				Collections.sort(list);

				if (result.contains(list)) {
```

```
                                        continue;
                                }
                                result.add(list);
                        }
                }
                return result;
        }
}
```

GTD_MBA_Group_Mapper.java

```
package adbms.finalproj;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class GTD_MBA_Group_Mapper extends Mapper<Object, Text, Text, IntWritable>
{

        private int numberOfPairs = 2;
        private static final Text reducerKey = new Text();

        private static final IntWritable ONE = new IntWritable(1);

        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
                String line = value.toString();

                if (line.contains("eventid")) {
                        return;
                } else {
                        List<String> items = convertItemsToList(line);

                        if ((items == null) || (items.isEmpty())) {
                                return;
                        }
                        generateMapperOutput(numberOfPairs, items, context);
                }
        }
```

```
	private static List<String> convertItemsToList(String line) {
		if ((line == null) || (line.length() == 0)) {
			// no mapper output will be generated
			return null;
		}

		String[] tokens = line.split(",");

		if ((tokens == null) || (tokens.length == 0)) {
			return null;
		}

		List<String> items = new ArrayList<String>();
		items.add(tokens[5]);
		items.add(tokens[3]);

		return items;
	}

	private void generateMapperOutput(int numberOfPairs, List<String> items,
Context context)
			throws IOException, InterruptedException {
		List<List<String>> sortedCombinations =
Combination.findSortedCombinations(items, numberOfPairs);
		for (List<String> itemList : sortedCombinations) {
			// System.out.println("itemlist=" + itemList.toString());
			reducerKey.set(itemList.toString());
			context.write(reducerKey, ONE);
		}
	}
}
```

GTD_MBA_Group_Reducer.java

```
package adbms.finalproj;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
```

```
public class GTD_MBA_Group_Reducer extends Reducer<Text, IntWritable, Text,
IntWritable> {

	private IntWritable result = new IntWritable();

	public void reduce(Text key, Iterable<IntWritable> values, Context context)
			throws IOException, InterruptedException {

		int sum = 0;

		for (IntWritable value : values) {
			sum += value.get();
		}

		result.set(sum);
		context.write(key, result);
	}

}
```

GTD_MBA_Group_Driver.java

```
package adbms.finalproj;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class GTD_MBA_Group_Driver {

	public static void main(String[] args) throws Exception {
		// TODO Auto-generated method stub
		System.out.println("Global Terrorism Database");

		Configuration conf = new Configuration();
		Job job = Job.getInstance(conf, "Attacks Per Country");

		job.setJarByClass(GTD_MBA_Group_Driver.class);
		job.setMapperClass(GTD_MBA_Group_Mapper.class);
		job.setReducerClass(GTD_MBA_Group_Reducer.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(true) ? 0 : 1);
	}
}
```