REPORT

Problem Statement
Analyze and find interesting information related to movies and users from the movie lens dataset.

Dataset Description: Below is the list of files that are part of the dataset

- movies.csv: This file contains information about movieId, title and genres. Movie title also includes the year it was released.
- ratings.csv: This file contains information about userId, movieId, rating, timestamp.
- tags.csv: This file contains information about userId, movieId, tag, timestamp
- links.csv: This file contains information about movieId, imdbId, tmdbId
- genome-scores.csv: This file contains information about movieId, tagId, relevance. Here relevance provides a score indicating how relevant is a tag provided by the user for a given movie.
- genome-tags.csv: This file contains information about tagId, tag

Dataset Link: https://grouplens.org/datasets/movielens/25m/

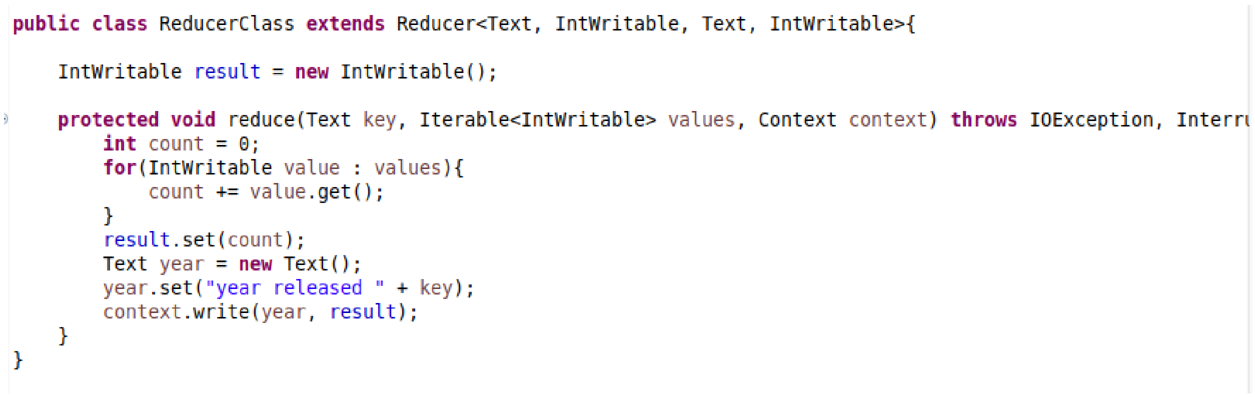
Analysis
All the analysis performed are from a user (logged into movieLens website) perspective

1. **Find number of movies released per year.**

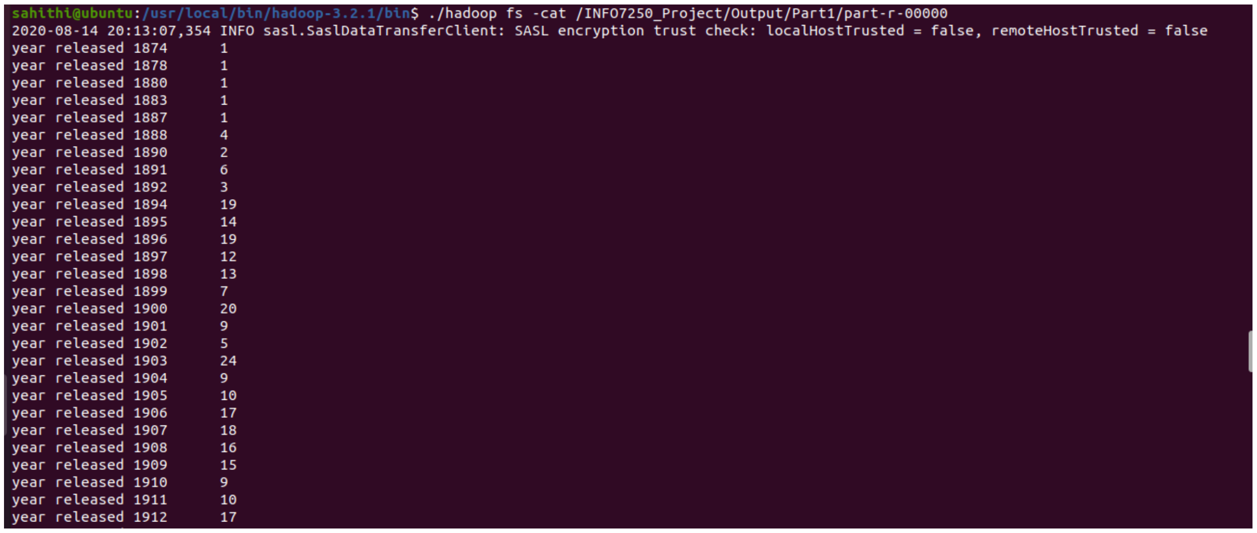
The analysis performed here is to find the total count of movies released per each year by analyzing movies.csv

In this analysis, we send all the years and in the reducer we accumulate the result sent from the mapper for each year and extract the count. The result would be in the form of <yearReleased, count>

Code Snippet:



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

    IntWritable result = new IntWritable();

    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, Interr
        int count = 0;
        for(IntWritable value : values){
            count += value.get();
        }
        result.set(count);
        Text year = new Text();
        year.set("year released " + key);
        context.write(year, result);
    }
}
```



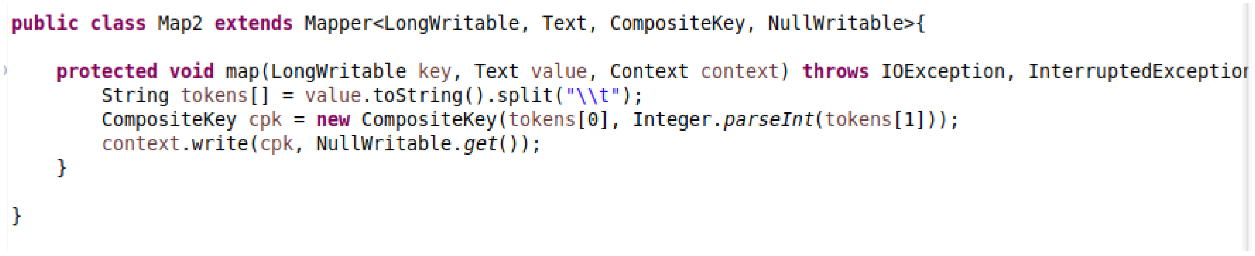
```
sahithi@ubuntu:/usr/local/bin/hadoop-3.2.1/bin$ ./hadoop fs -cat /INFO7250_Project/Output/Part1/part-r-00000
2020-08-14 20:13:07,354 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
year released 1874	1
year released 1878	1
year released 1880	1
year released 1883	1
year released 1887	1
year released 1888	4
year released 1890	2
year released 1891	6
year released 1892	3
year released 1894	19
year released 1895	14
year released 1896	19
year released 1897	12
year released 1898	13
year released 1899	7
year released 1900	20
year released 1901	9
year released 1902	5
year released 1903	24
year released 1904	9
year released 1905	10
year released 1906	17
year released 1907	18
year released 1908	16
year released 1909	15
year released 1910	9
year released 1911	10
year released 1912	17
```

As this analysis could not produce much useful information next analysis is to find which years have to the greatest number of movie releases.

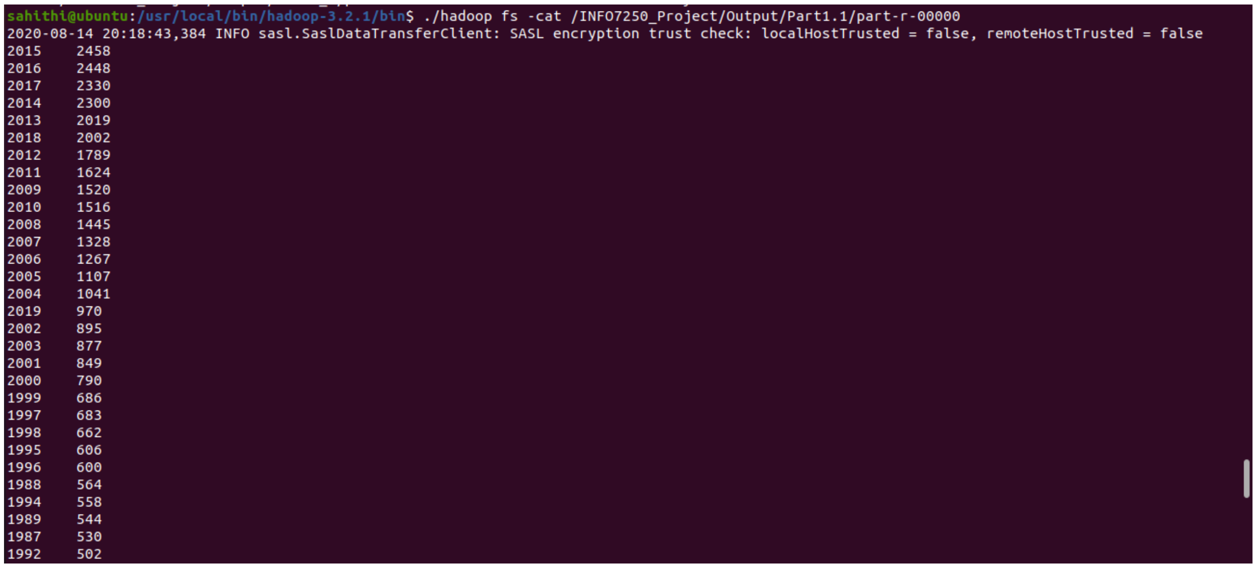
2. Find the years when most movies were released.

Here we run two map-reduce jobs. First job would produce the intermediate file containing year and accumulated count. The second job would take the intermediate file as input and produce the output in the form of <year, count> in decreasing order. In the second job we use secondary sorting to perform sorting based on value. We use movies.csv to perform analysis on.

Code Snippet:



```java
public class Map2 extends Mapper<LongWritable, Text, CompositeKey, NullWritable>{

    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedExceptior
        String tokens[] = value.toString().split("\\t");
        CompositeKey cpk = new CompositeKey(tokens[0], Integer.parseInt(tokens[1]));
        context.write(cpk, NullWritable.get());
    }

}
```

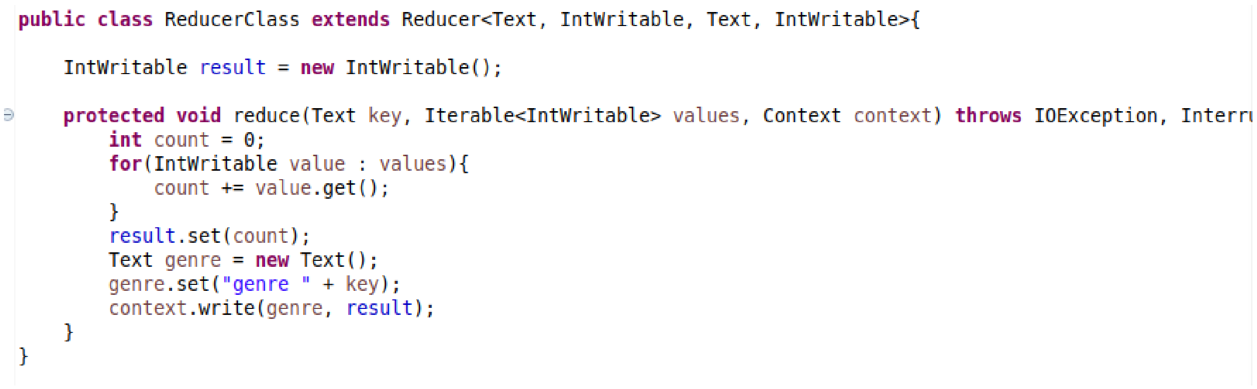


```
sahithi@ubuntu:/usr/local/bin/hadoop-3.2.1/bin$ ./hadoop fs -cat /INFO7250_Project/Output/Part1.1/part-r-00000
2020-08-14 20:18:43,384 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2015    2458
2016    2448
2017    2330
2014    2300
2013    2019
2018    2002
2012    1789
2011    1624
2009    1520
2010    1516
2008    1445
2007    1328
2006    1267
2005    1107
2004    1041
2019    970
2002    895
2003    877
2001    849
2000    790
1999    686
1997    683
1998    662
1995    606
1996    600
1988    564
1994    558
1989    544
1987    530
1992    502
```

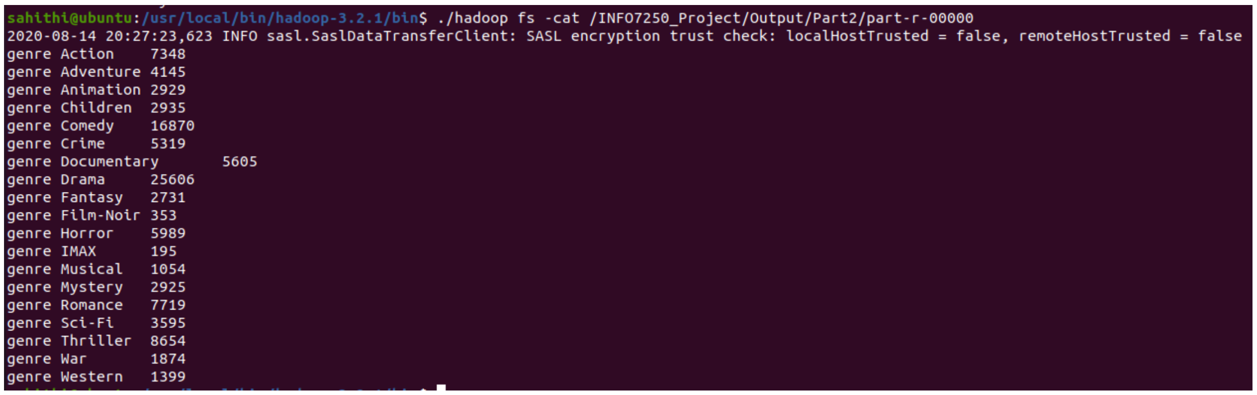
3. Find total number of movies per genre

A user might be interested in finding out how many were present in each genre in order to select some movie to watch. Movies.csv is been used to perform analysis on. Here we write a single map-reduce job which provides the output in the form of <genre, count>

Code Snippet:



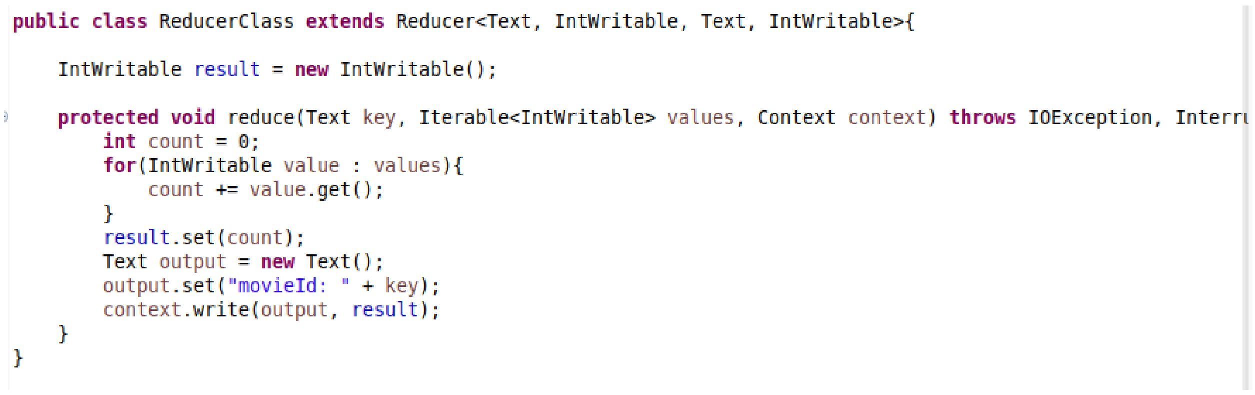
```
public class ReducerClass extends Reducer<Text, IntWritable, Text, IntWritable>{

    IntWritable result = new IntWritable();

    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, Interru
        int count = 0;
        for(IntWritable value : values){
            count += value.get();
        }
        result.set(count);
        Text genre = new Text();
        genre.set("genre " + key);
        context.write(genre, result);
    }
}
```



```
sahithi@ubuntu:/usr/local/bin/hadoop-3.2.1/bin$ ./hadoop fs -cat /INFO7250_Project/Output/Part2/part-r-00000
2020-08-14 20:27:23,623 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
genre Action    7348
genre Adventure 4145
genre Animation 2929
genre Children  2935
genre Comedy    16870
genre Crime     5319
genre Documentary       5605
genre Drama     25606
genre Fantasy   2731
genre Film-Noir 353
genre Horror    5989
genre IMAX      195
genre Musical   1054
genre Mystery   2925
genre Romance   7719
genre Sci-Fi    3595
genre Thriller  8654
genre War       1874
genre Western   1399
```

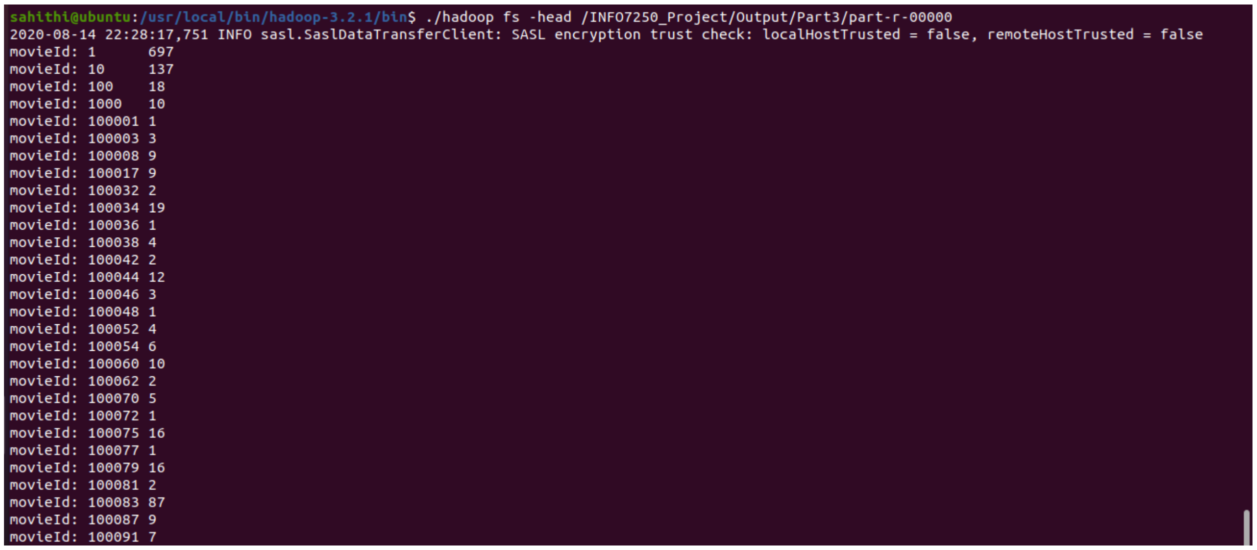
4. Find the number of tags for each movie

In this analysis, we would like to find out tag count for each movie. Analysis is performed on tags.csv



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

    IntWritable result = new IntWritable();

    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, Interru
        int count = 0;
        for(IntWritable value : values){
            count += value.get();
        }
        result.set(count);
        Text output = new Text();
        output.set("movieId: " + key);
        context.write(output, result);
    }
}
```

Output:



```
sahithi@ubuntu:/usr/local/bin/hadoop-3.2.1/bin$ ./hadoop fs -head /INFO7250_Project/Output/Part3/part-r-00000
2020-08-14 22:28:17,751 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
movieId: 1      697
movieId: 10     137
movieId: 100    18
movieId: 1000   10
movieId: 100001 1
movieId: 100003 3
movieId: 100008 9
movieId: 100017 9
movieId: 100032 2
movieId: 100034 19
movieId: 100036 1
movieId: 100038 4
movieId: 100042 2
movieId: 100044 12
movieId: 100046 3
movieId: 100048 1
movieId: 100052 4
movieId: 100054 6
movieId: 100060 10
movieId: 100062 2
movieId: 100070 5
movieId: 100072 1
movieId: 100075 16
movieId: 100077 1
movieId: 100079 16
movieId: 100081 2
movieId: 100083 87
movieId: 100087 9
movieId: 100091 7
```

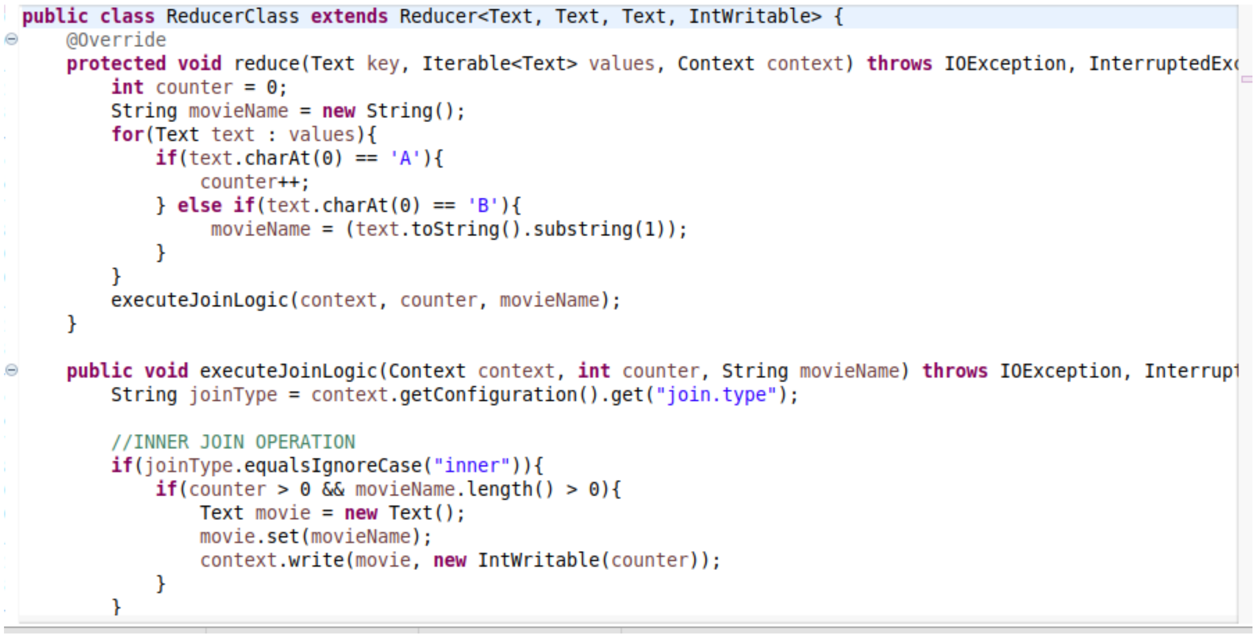
As the tags.csv has only movieId field in it, watching this result would not yield any information to a user interested in watching a new movie. Hence the next analysis performed would be to associate each count with its respective movieName.

5. Find movies that are most tagged by the users

This analysis is to join two files tags.csv with movies.csv as movies.csv is the only file containing movieName information. Also, if a user wants to watch some hit movie it would be useful to give results in decreasing order. Hence the output of the join has been sorted using secondary sort to get the output as <movieName, count>

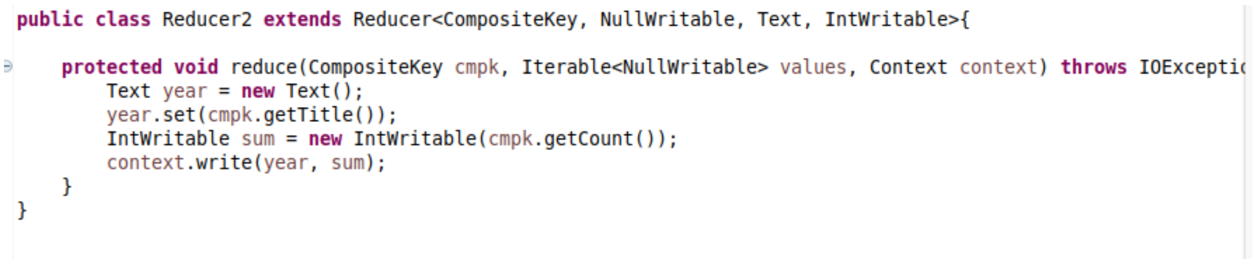
Code Snippet:

The below code snippet contains code to join two files (tags.csv and movies.csv)



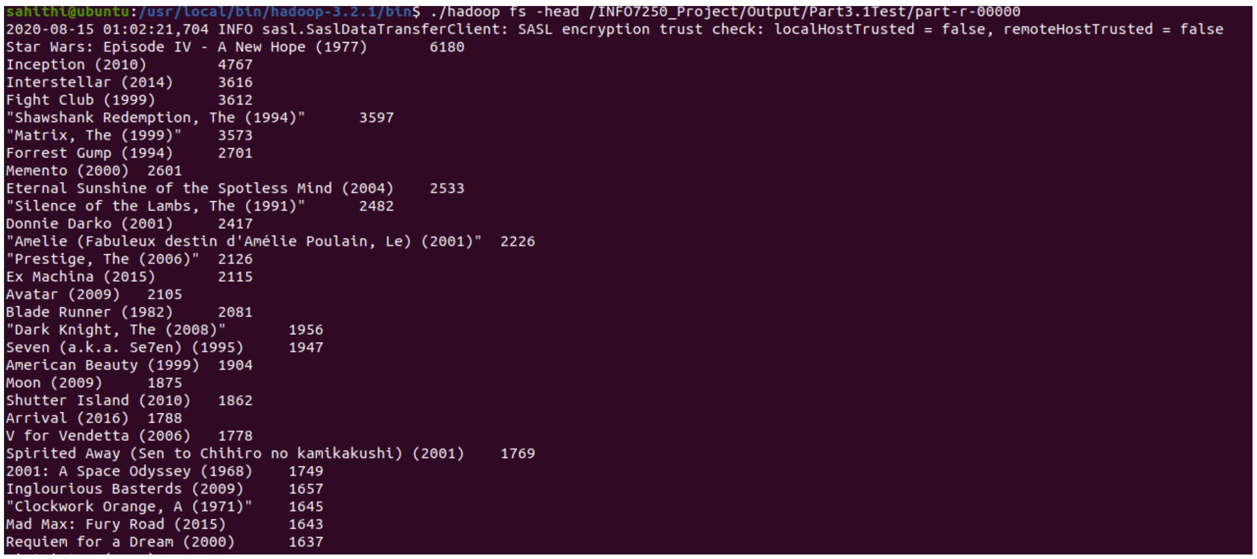
```
public class ReducerClass extends Reducer<Text, Text, Text, IntWritable> {
    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedExc
        int counter = 0;
        String movieName = new String();
        for(Text text : values){
            if(text.charAt(0) == 'A'){
                counter++;
            } else if(text.charAt(0) == 'B'){
                 movieName = (text.toString().substring(1));
            }
        }
        executeJoinLogic(context, counter, movieName);
    }

    public void executeJoinLogic(Context context, int counter, String movieName) throws IOException, Interrupt
        String joinType = context.getConfiguration().get("join.type");

        //INNER JOIN OPERATION
        if(joinType.equalsIgnoreCase("inner")){
            if(counter > 0 && movieName.length() > 0){
                Text movie = new Text();
                movie.set(movieName);
                context.write(movie, new IntWritable(counter));
            }
        }
```

Next, we write a job to sort the data



```
public class Reducer2 extends Reducer<CompositeKey, NullWritable, Text, IntWritable>{

    protected void reduce(CompositeKey cmpk, Iterable<NullWritable> values, Context context) throws IOExceptic
        Text year = new Text();
        year.set(cmpk.getTitle());
        IntWritable sum = new IntWritable(cmpk.getCount());
        context.write(year, sum);
    }
}
```

Output:



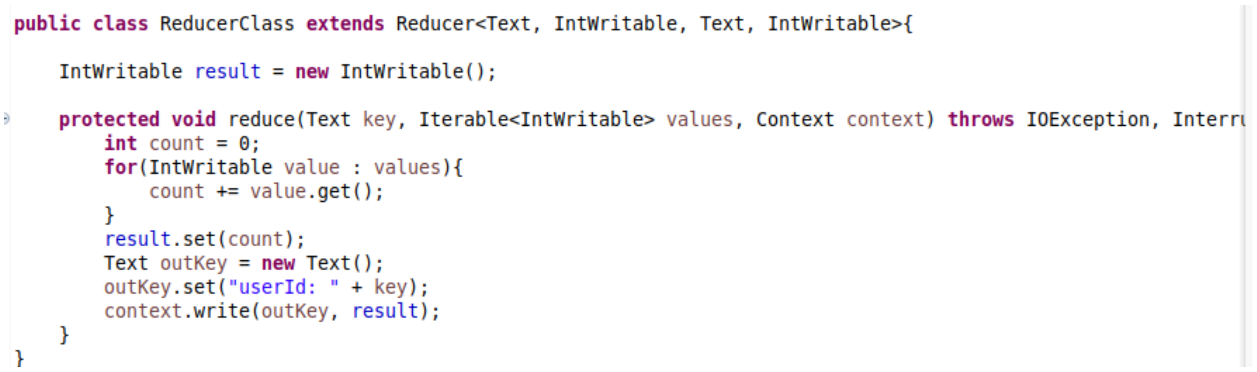
```
sahithi@ubuntu:/usr/local/bin/hadoop-3.2.1/bin$ ./hadoop fs -head /INFO7250_Project/Output/Part3.1Test/part-r-00000
2020-08-15 01:02:21,704 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
Star Wars: Episode IV - A New Hope (1977)       6180
Inception (2010)        4767
Interstellar (2014)     3616
Fight Club (1999)       3612
"Shawshank Redemption, The (1994)"      3597
"Matrix, The (1999)"    3573
Forrest Gump (1994)     2701
Memento (2000)  2601
Eternal Sunshine of the Spotless Mind (2004)    2533
"Silence of the Lambs, The (1991)"      2482
Donnie Darko (2001)     2417
"Amelie (Fabuleux destin d'Amélie Poulain, Le) (2001)"  2226
"Prestige, The (2006)"  2126
Ex Machina (2015)       2115
Avatar (2009)   2105
Blade Runner (1982)     2081
"Dark Knight, The (2008)"       1956
Seven (a.k.a. Se7en) (1995)     1947
American Beauty (1999)  1904
Moon (2009)     1875
Shutter Island (2010)   1862
Arrival (2016)  1788
V for Vendetta (2006)   1778
Spirited Away (Sen to Chihiro no kamikakushi) (2001)    1769
2001: A Space Odyssey (1968)    1749
Inglourious Basterds (2009)     1657
"Clockwork Orange, A (1971)"    1645
Mad Max: Fury Road (2015)       1643
Requiem for a Dream (2000)      1637
```

6. Find the number of times a user tagged movies.

In this analysis, we find out number of times a user tagged movies, by accumulating the count for each tag for a particular user.

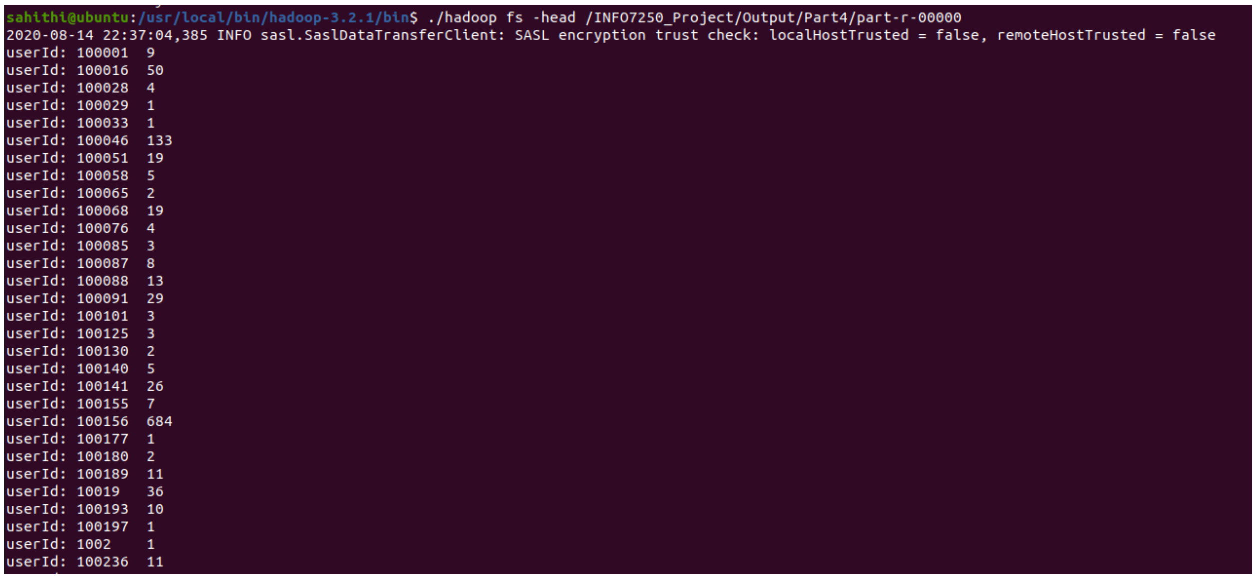
Code Snippet:

As we don't have any information regarding the user other than userId, we restrict all our analysis to only userId. Output of the analysis would be <userId, countOfTagsGiven>



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

    IntWritable result = new IntWritable();

    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, Interru
        int count = 0;
        for(IntWritable value : values){
            count += value.get();
        }
        result.set(count);
        Text outKey = new Text();
        outKey.set("userId: " + key);
        context.write(outKey, result);
    }
}
```

Output:



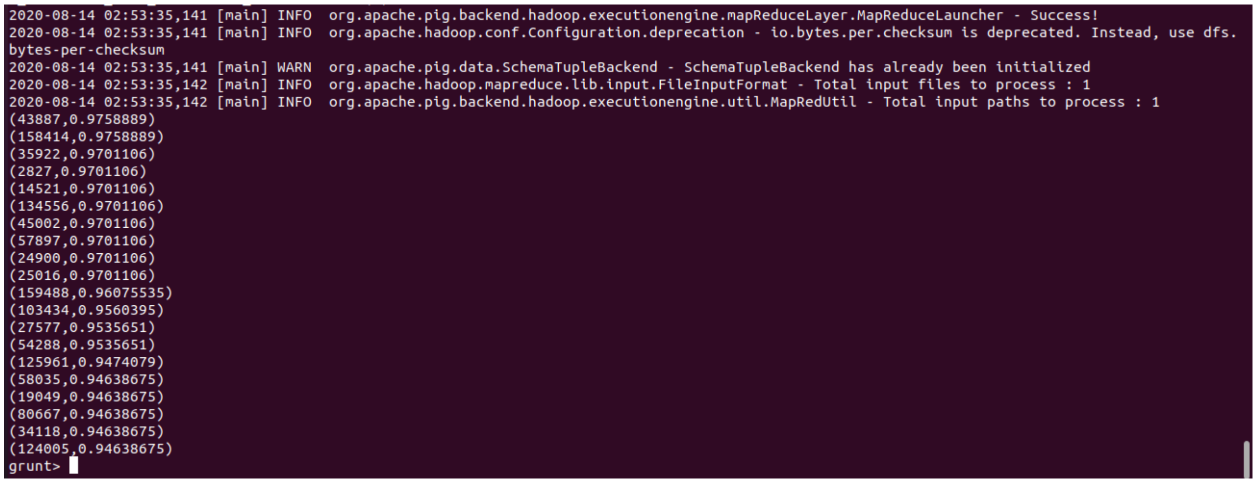
```
sahithi@ubuntu:/usr/local/bin/hadoop-3.2.1/bin$ ./hadoop fs -head /INFO7250_Project/Output/Part4/part-r-00000
2020-08-14 22:37:04,385 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
userId: 100001  9
userId: 100016  50
userId: 100028  4
userId: 100029  1
userId: 100033  1
userId: 100046  133
userId: 100051  19
userId: 100058  5
userId: 100065  2
userId: 100068  19
userId: 100076  4
userId: 100085  3
userId: 100087  8
userId: 100088  13
userId: 100091  29
userId: 100101  3
userId: 100125  3
userId: 100130  2
userId: 100140  5
userId: 100141  26
userId: 100155  7
userId: 100156  684
userId: 100177  1
userId: 100180  2
userId: 100189  11
userId: 10019   36
userId: 100193  10
userId: 100197  1
userId: 1002    1
userId: 100236  11
```

Seeing the result of the above analysis, we have many users who made just few tags and users with a lot of tags. We also have a file genome-scores.csv where the tag relevance score is given for each tag given by a user. Though a user makes more tags it doesn't mean that his/her tags are relevant to the movie. But using the relevance score we can find out the which user has given tags with high relevance. So, the next analysis performed is to find a non-spam user.

7. Find the probable non-spam users based on tag relevance.

To perform this analysis we need to join genome-tag.csv and genome-scores.csv to extract tags associated with each tagId and then filter out the tags which have relevance >= 0.7 (As our goal is to find non-spam users) and then join tags.csv to the filtered data to produce the result in the form <userId, avgRelevanceScore>

Output

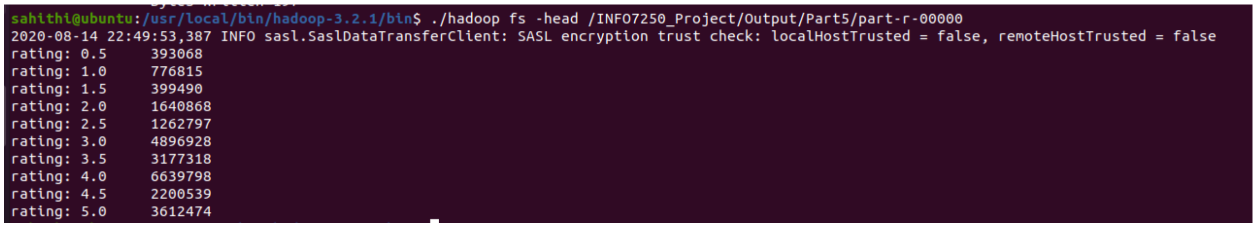


```
2020-08-14 02:53:35,141 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2020-08-14 02:53:35,141 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.
bytes-per-checksum
2020-08-14 02:53:35,141 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2020-08-14 02:53:35,142 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2020-08-14 02:53:35,142 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(43887,0.9758889)
(158414,0.9758889)
(35922,0.9701106)
(2827,0.9701106)
(14521,0.9701106)
(134556,0.9701106)
(45002,0.9701106)
(57897,0.9701106)
(24900,0.9701106)
(25016,0.9701106)
(159488,0.96075535)
(103434,0.9560395)
(27577,0.9535651)
(54288,0.9535651)
(125961,0.9474079)
(58035,0.94638675)
(19049,0.94638675)
(80667,0.94638675)
(34118,0.94638675)
(124005,0.94638675)
grunt>
```

8. Find the number of movies per rating

In this analysis on ratings.csv, the goal is to find the total number of movies per rating. Using this analysis, the user will understand how the movies listed in the website are distributed. For this analysis the data for is rating is accumulated in the reducer and produces the output in the form of <rating, count>

Output:



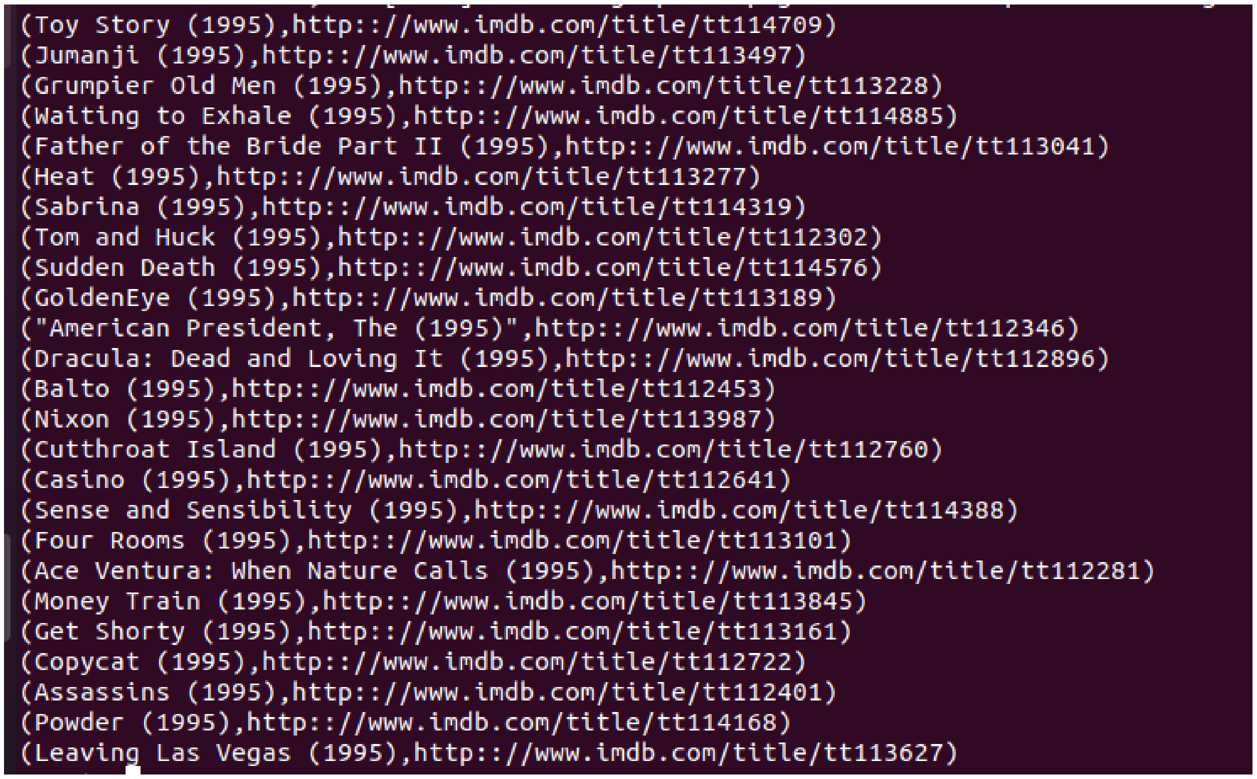
```
sahithi@ubuntu:/usr/local/bin/hadoop-3.2.1/bin$ ./hadoop fs -head /INFO7250_Project/Output/Part5/part-r-00000
2020-08-14 22:49:53,387 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
rating: 0.5     393068
rating: 1.0     776815
rating: 1.5     399490
rating: 2.0     1640868
rating: 2.5     1262797
rating: 3.0     4896928
rating: 3.5     3177318
rating: 4.0     6639798
rating: 4.5     2200539
rating: 5.0     3612474
```

9. Provide the IMDB links for each movie.

The data given to us has only Id's information regarding the movies present but let's say when a user goes through all the tags, rating information and finds some movie to be interesting and if he wants to watch it there is no direct link provided to the source. Hence this analysis is performed to generate IMDB links for each movie listed in the data.

For this analysis, a JOIN is performed on movies.csv and links.csv and then generate the link to each movie using the IMDB Id provided. Output produced would be in the form of <movieTitle, IMDBmovieLink>

Output:

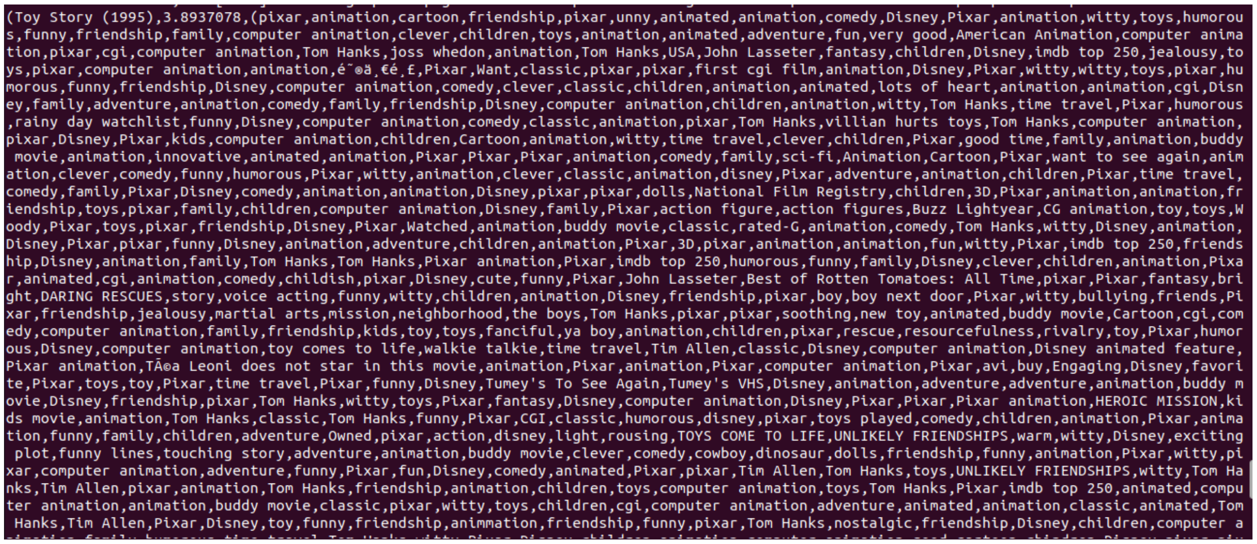


```
(Toy Story (1995),http:://www.imdb.com/title/tt114709)
(Jumanji (1995),http:://www.imdb.com/title/tt113497)
(Grumpier Old Men (1995),http:://www.imdb.com/title/tt113228)
(Waiting to Exhale (1995),http:://www.imdb.com/title/tt114885)
(Father of the Bride Part II (1995),http:://www.imdb.com/title/tt113041)
(Heat (1995),http:://www.imdb.com/title/tt113277)
(Sabrina (1995),http:://www.imdb.com/title/tt114319)
(Tom and Huck (1995),http:://www.imdb.com/title/tt112302)
(Sudden Death (1995),http:://www.imdb.com/title/tt114576)
(GoldenEye (1995),http:://www.imdb.com/title/tt113189)
("American President, The (1995)",http:://www.imdb.com/title/tt112346)
(Dracula: Dead and Loving It (1995),http:://www.imdb.com/title/tt112896)
(Balto (1995),http:://www.imdb.com/title/tt112453)
(Nixon (1995),http:://www.imdb.com/title/tt113987)
(Cutthroat Island (1995),http:://www.imdb.com/title/tt112760)
(Casino (1995),http:://www.imdb.com/title/tt112641)
(Sense and Sensibility (1995),http:://www.imdb.com/title/tt114388)
(Four Rooms (1995),http:://www.imdb.com/title/tt113101)
(Ace Ventura: When Nature Calls (1995),http:://www.imdb.com/title/tt112281)
(Money Train (1995),http:://www.imdb.com/title/tt113845)
(Get Shorty (1995),http:://www.imdb.com/title/tt113161)
(Copycat (1995),http:://www.imdb.com/title/tt112722)
(Assassins (1995),http:://www.imdb.com/title/tt112401)
(Powder (1995),http:://www.imdb.com/title/tt114168)
(Leaving Las Vegas (1995),http:://www.imdb.com/title/tt113627)
```

**10.	Find average rating and all tag information for each movie**

This analysis is performed to gather all the information about a movie in one place. For this analysis movies.csv, ratings.csv and tags.csv were joined and necessary actions were taken. The output is in the form of <movieTitle, avgRating, bagOfTuples(tags)>

Output:

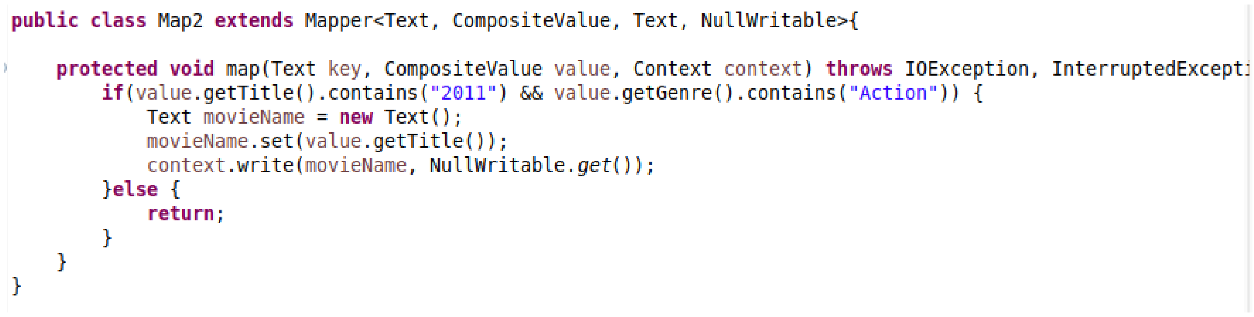


```
(Toy Story (1995),3.8937078,(pixar,animation,cartoon,friendship,pixar,unny,animated,animation,comedy,Disney,Pixar,animation,witty,toys,humorou
s,funny,friendship,family,computer animation,clever,children,toys,animation,animated,adventure,fun,very good,American Animation,computer anima
tion,pixar,cgi,computer animation,Tom Hanks,joss whedon,animation,Tom Hanks,USA,John Lasseter,fantasy,children,Disney,imdb top 250,jealousy,to
ys,pixar,computer animation,animation,é˜®ä¸€é¸£,Pixar,Want,classic,pixar,pixar,first cgi film,animation,Disney,Pixar,witty,witty,toys,pixar,hu
morous,funny,friendship,Disney,computer animation,comedy,clever,classic,children,animation,animated,lots of heart,animation,animation,cgi,Disn
ey,family,adventure,animation,comedy,family,friendship,Disney,computer animation,children,animation,witty,Tom Hanks,time travel,Pixar,humorous
,rainy day watchlist,funny,Disney,computer animation,comedy,classic,animation,pixar,Tom Hanks,villian hurts toys,Tom Hanks,computer animation,
pixar,Disney,Pixar,kids,computer animation,children,Cartoon,animation,witty,time travel,clever,children,Pixar,good time,family,animation,buddy
 movie,animation,innovative,animated,animation,Pixar,Pixar,Pixar,animation,comedy,family,sci-fi,Animation,Cartoon,Pixar,want to see again,anim
ation,clever,comedy,funny,humorous,Pixar,witty,animation,clever,classic,animation,disney,Pixar,adventure,animation,children,Pixar,time travel,
comedy,family,Pixar,Disney,comedy,animation,animation,Disney,pixar,pixar,dolls,National Film Registry,children,3D,Pixar,animation,animation,fr
iendship,toys,pixar,family,children,computer animation,Disney,family,Pixar,action figure,action figures,Buzz Lightyear,CG animation,toy,toys,W
oody,Pixar,toys,pixar,friendship,Disney,Pixar,Watched,animation,buddy movie,classic,rated-G,animation,comedy,Tom Hanks,witty,Disney,animation,
Disney,Pixar,pixar,funny,Disney,animation,adventure,children,animation,Pixar,3D,pixar,animation,animation,fun,witty,Pixar,imdb top 250,friends
hip,Disney,animation,family,Tom Hanks,Tom Hanks,Pixar animation,Pixar,imdb top 250,humorous,funny,family,Disney,clever,children,animation,Pixa
r,animated,cgi,animation,comedy,childish,pixar,Disney,cute,funny,Pixar,John Lasseter,Best of Rotten Tomatoes: All Time,pixar,Pixar,fantasy,bri
ght,DARING RESCUES,story,voice acting,funny,witty,children,animation,Disney,friendship,pixar,boy,boy next door,Pixar,witty,bullying,friends,Pi
xar,friendship,jealousy,martial arts,mission,neighborhood,the boys,Tom Hanks,pixar,pixar,soothing,new toy,animated,buddy movie,Cartoon,cgi,com
edy,computer animation,family,friendship,kids,toy,toys,fanciful,ya boy,animation,children,pixar,rescue,resourcefulness,rivalry,toy,Pixar,humor
ous,Disney,computer animation,toy comes to life,walkie talkie,time travel,Tim Allen,classic,Disney,computer animation,Disney animated feature,
Pixar animation,TÃ©a Leoni does not star in this movie,animation,Pixar,animation,Pixar,computer animation,Pixar,avi,buy,Engaging,Disney,favori
te,Pixar,toys,toy,Pixar,time travel,Pixar,funny,Disney,Tumey's To See Again,Tumey's VHS,Disney,animation,adventure,adventure,animation,buddy m
ovie,Disney,friendship,pixar,Tom Hanks,witty,toys,Pixar,fantasy,Disney,computer animation,Disney,Pixar,Pixar,Pixar animation,HEROIC MISSION,ki
ds movie,animation,Tom Hanks,classic,Tom Hanks,funny,Pixar,CGI,classic,humorous,disney,pixar,toys played,comedy,children,animation,Pixar,anima
tion,funny,family,children,adventure,Owned,pixar,action,disney,light,rousing,TOYS COME TO LIFE,UNLIKELY FRIENDSHIPS,warm,witty,Disney,exciting
 plot,funny lines,touching story,adventure,animation,buddy movie,clever,comedy,cowboy,dinosaur,dolls,friendship,funny,animation,Pixar,witty,pi
xar,computer animation,adventure,funny,Pixar,fun,Disney,comedy,animated,Pixar,pixar,Tim Allen,Tom Hanks,toys,UNLIKELY FRIENDSHIPS,witty,Tom Ha
nks,Tim Allen,pixar,animation,Tom Hanks,friendship,animation,children,toys,computer animation,toys,Tom Hanks,Pixar,imdb top 250,animated,compu
ter animation,animation,buddy movie,classic,pixar,witty,toys,children,cgi,computer animation,adventure,animated,animation,classic,animated,Tom
 Hanks,Tim Allen,Pixar,Disney,toy,funny,friendship,animmation,friendship,funny,pixar,Tom Hanks,nostalgic,friendship,Disney,children,computer a
```

11. Find the list of movies released in 2011 in “Action” Genre.

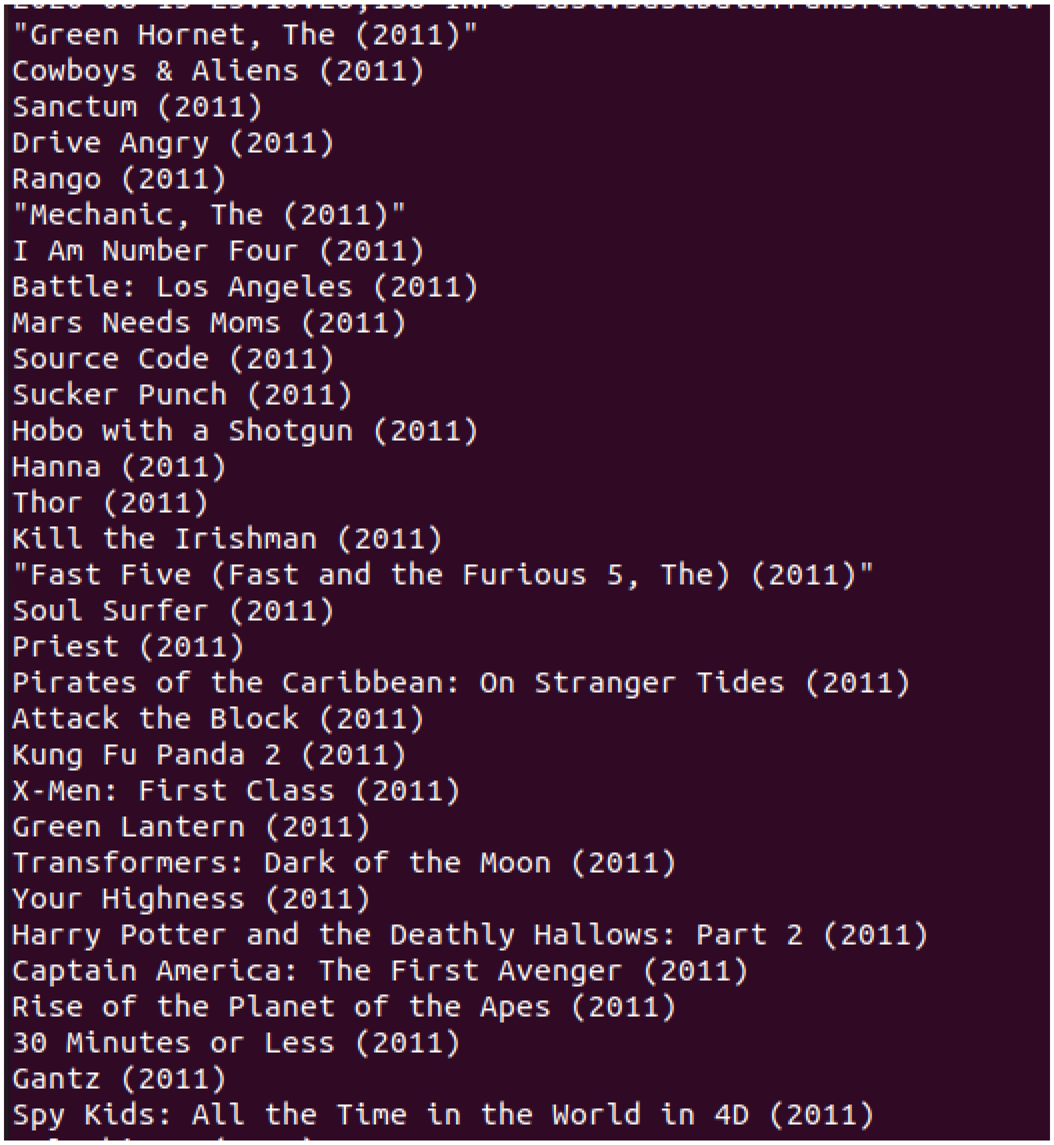
This analysis was performed as an example filter which can be changed according to the user’s choice. Let’s say a user is interested in watching a movie released in a particular year and with a particular genre, this analysis would be helpful. For this analysis, we use two mappers, one mapper to select the data using a composite value and the other mapper to filter the data according to the year and genre user wishes and generate the output in the form <movieTitle>. And this would be a map-only job.

Code Snippet:



```
public class Map2 extends Mapper<Text, CompositeValue, Text, NullWritable>{

    protected void map(Text key, CompositeValue value, Context context) throws IOException, InterruptedExcepti
        if(value.getTitle().contains("2011") && value.getGenre().contains("Action")) {
            Text movieName = new Text();
            movieName.set(value.getTitle());
            context.write(movieName, NullWritable.get());
        }else {
            return;
        }
    }
}
```

Output:



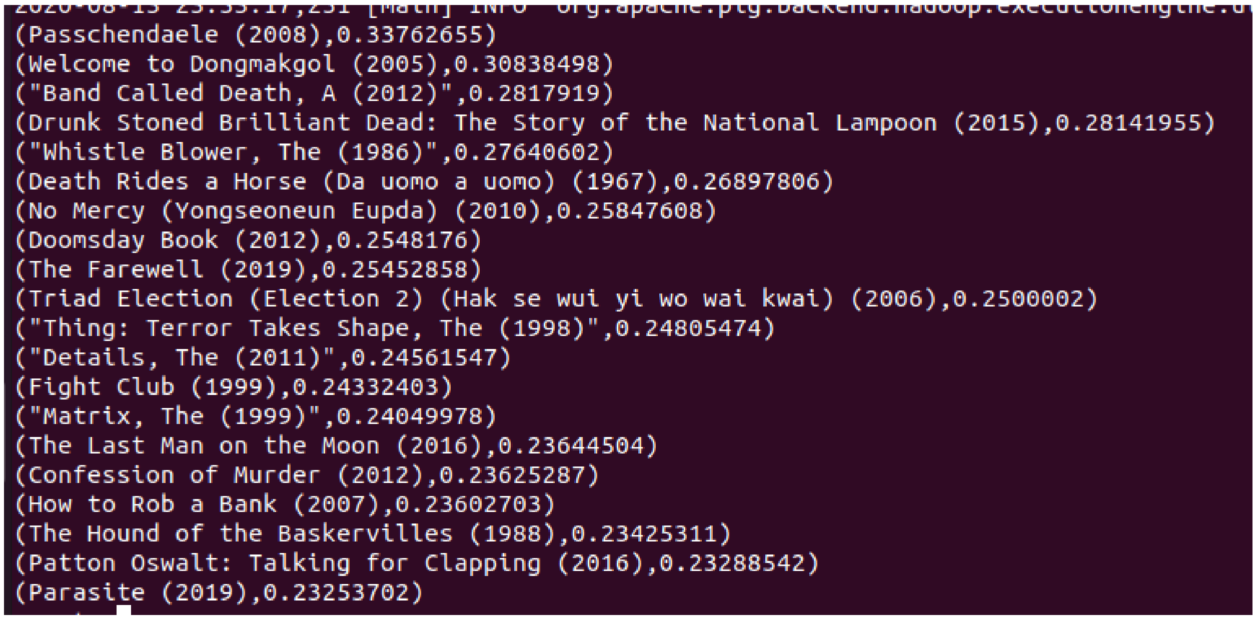
```
"Green Hornet, The (2011)"
Cowboys & Aliens (2011)
Sanctum (2011)
Drive Angry (2011)
Rango (2011)
"Mechanic, The (2011)"
I Am Number Four (2011)
Battle: Los Angeles (2011)
Mars Needs Moms (2011)
Source Code (2011)
Sucker Punch (2011)
Hobo with a Shotgun (2011)
Hanna (2011)
Thor (2011)
Kill the Irishman (2011)
"Fast Five (Fast and the Furious 5, The) (2011)"
Soul Surfer (2011)
Priest (2011)
Pirates of the Caribbean: On Stranger Tides (2011)
Attack the Block (2011)
Kung Fu Panda 2 (2011)
X-Men: First Class (2011)
Green Lantern (2011)
Transformers: Dark of the Moon (2011)
Your Highness (2011)
Harry Potter and the Deathly Hallows: Part 2 (2011)
Captain America: The First Avenger (2011)
Rise of the Planet of the Apes (2011)
30 Minutes or Less (2011)
Gantz (2011)
Spy Kids: All the Time in the World in 4D (2011)
```

12. Find the top movies based on user tags relevance.

This analysis was performed to find out movies which have tags high user tags relevance. This analysis would be useful if a user who has no knowledge about a movie and would want to know if he would be interested in watching the movie after reading through all the user tags. Unfortunately, from this analysis we find out that all the tags which are related to the movie have very less relevance score and would not help the user and provide him any gist about the movie.

Output:

Format < movieTitle, AvgRelevance>



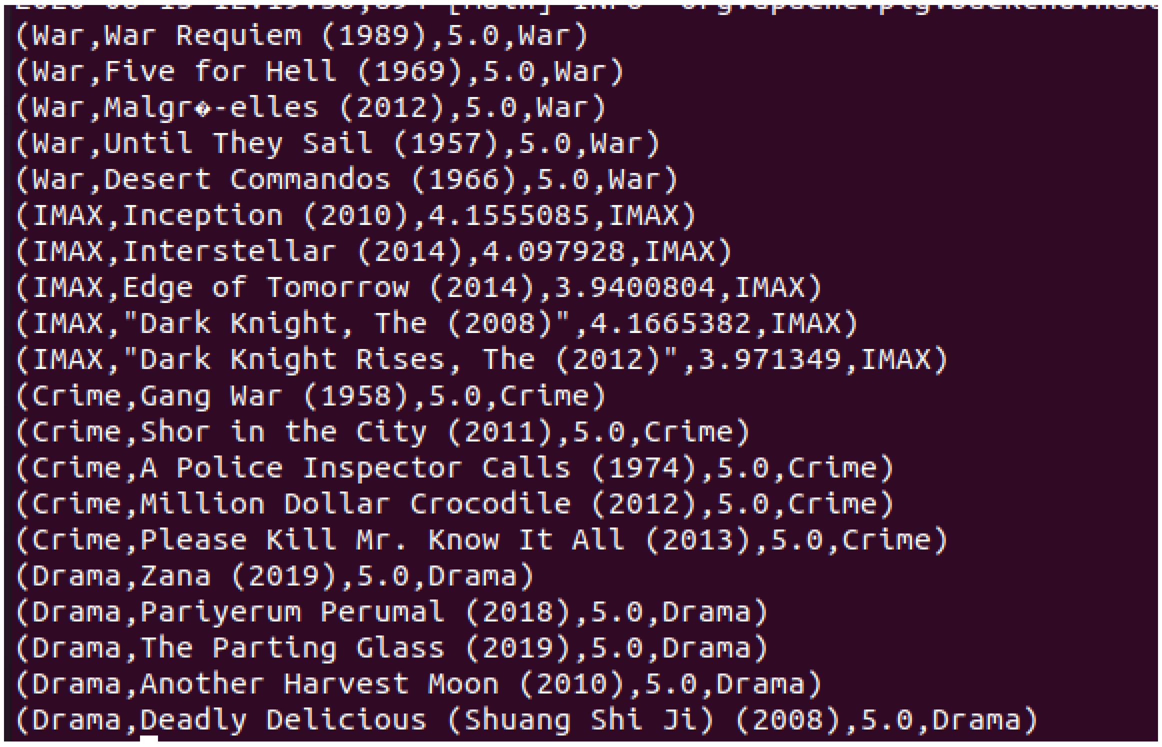
```
2020-08-13 23:33:17,231 [main] INFO  org.apache.pig.backend.hadoop.executionengine.u
(Passchendaele (2008),0.33762655)
(Welcome to Dongmakgol (2005),0.30838498)
("Band Called Death, A (2012)",0.2817919)
(Drunk Stoned Brilliant Dead: The Story of the National Lampoon (2015),0.28141955)
("Whistle Blower, The (1986)",0.27640602)
(Death Rides a Horse (Da uomo a uomo) (1967),0.26897806)
(No Mercy (Yongseoneun Eupda) (2010),0.25847608)
(Doomsday Book (2012),0.2548176)
(The Farewell (2019),0.25452858)
(Triad Election (Election 2) (Hak se wui yi wo wai kwai) (2006),0.2500002)
("Thing: Terror Takes Shape, The (1998)",0.24805474)
("Details, The (2011)",0.24561547)
(Fight Club (1999),0.24332403)
("Matrix, The (1999)",0.24049978)
(The Last Man on the Moon (2016),0.23644504)
(Confession of Murder (2012),0.23625287)
(How to Rob a Bank (2007),0.23602703)
(The Hound of the Baskervilles (1988),0.23425311)
(Patton Oswalt: Talking for Clapping (2016),0.23288542)
(Parasite (2019),0.23253702)
```

13.Find top (highly rated) 5 movies per each genre

Let’s say a user is interested in watching highly rated movie in Romance genre, this analysis would help. This analysis would produce the result of highly rated 5 movies for every genre present in the dataset.

Output:

Format<Genre, MovieTitle, AvgRating, Genre>



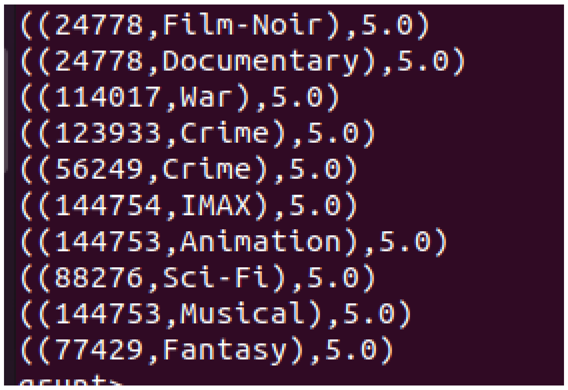
```
(War,War Requiem (1989),5.0,War)
(War,Five for Hell (1969),5.0,War)
(War,Malgr�-elles (2012),5.0,War)
(War,Until They Sail (1957),5.0,War)
(War,Desert Commandos (1966),5.0,War)
(IMAX,Inception (2010),4.1555085,IMAX)
(IMAX,Interstellar (2014),4.097928,IMAX)
(IMAX,Edge of Tomorrow (2014),3.9400804,IMAX)
(IMAX,"Dark Knight, The (2008)",4.1665382,IMAX)
(IMAX,"Dark Knight Rises, The (2012)",3.971349,IMAX)
(Crime,Gang War (1958),5.0,Crime)
(Crime,Shor in the City (2011),5.0,Crime)
(Crime,A Police Inspector Calls (1974),5.0,Crime)
(Crime,Million Dollar Crocodile (2012),5.0,Crime)
(Crime,Please Kill Mr. Know It All (2013),5.0,Crime)
(Drama,Zana (2019),5.0,Drama)
(Drama,Pariyerum Perumal (2018),5.0,Drama)
(Drama,The Parting Glass (2019),5.0,Drama)
(Drama,Another Harvest Moon (2010),5.0,Drama)
(Drama,Deadly Delicious (Shuang Shi Ji) (2008),5.0,Drama)
```

14. Find the genres liked by a user based on the ratings given

This analysis would be helpful in recommending movies to the user according to his likes. In this analysis we perform analysis to find out a user's most liked genres based on the previous ratings he/she has provided.

Output:

Format <userId, Genre,AvgRating>



```
((24778,Film-Noir),5.0)
((24778,Documentary),5.0)
((114017,War),5.0)
((123933,Crime),5.0)
((56249,Crime),5.0)
((144754,IMAX),5.0)
((144753,Animation),5.0)
((88276,Sci-Fi),5.0)
((144753,Musical),5.0)
((77429,Fantasy),5.0)
```

APPENDIX:

Part 1:

Mapper:

```
package info7250.bigData.Project.Part1;


import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;


public class Map extends Mapper<LongWritable, Text, Text, IntWritable>{


    protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException{

            String line = value.toString();

            if(line.equals("movieId,title,genres")){

                    return;

            }

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


            int startIndex = data[1].lastIndexOf("(");

            if(startIndex < 0 || !Character.isDigit(data[1].charAt(startIndex +1)))

                    return;

            String yearValue = new String();

            yearValue = data[1].substring(startIndex+1, startIndex+5);

            Text year = new Text();

            year.set(yearValue);

            context.write(year, new IntWritable(1));

    }

}
```

Reducer:

```
package info7250.bigData.Project.Part1;


import java.io.IOException;


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


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


    IntWritable result = new IntWritable();


    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, 
InterruptedException{
            int count = 0;
            for(IntWritable value : values){
                    count += value.get();
            }
            result.set(count);
            Text year = new Text();
            year.set("year released " + key);
            context.write(year, result);
    }
}
```

Driver:

```
package info7250.bigData.Project.Part1;


import java.io.IOException;


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.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


public class Driver {


    public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException{

        // Create a new Job

                Configuration config = new Configuration();

                Job job = Job.getInstance(config,"movieCountYear");

        job.setJarByClass(Driver.class);


        FileInputFormat.addInputPath(job, new Path(args[0]));

        FileOutputFormat.setOutputPath(job, new Path(args[1]));


        job.setInputFormatClass(TextInputFormat.class);

        job.setOutputFormatClass(TextOutputFormat.class);


        job.setMapOutputKeyClass(Text.class);

        job.setMapOutputValueClass(IntWritable.class);


        job.setMapperClass(Map.class);

        job.setReducerClass(ReducerClass.class);


        job.setOutputKeyClass(Text.class);

        job.setOutputValueClass(IntWritable.class);


        System.exit(job.waitForCompletion(true) ? 0:1);
        }
}
```

Part 2:

Map1:

```
package info7250.bigData.Project.Part1_1;


import java.io.IOException;


import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;


public class Map1 extends Mapper<LongWritable, Text, Text, IntWritable>{


        protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException{

                String line = value.toString();

                if(line.equals("movieId,title,genres")){

                        return;

                }

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


                int startIndex = data[1].lastIndexOf("(");

                if(startIndex < 0 || !Character.isDigit(data[1].charAt(startIndex +1)))

                        return;

                String yearValue = new String();

                yearValue = data[1].substring(startIndex+1, startIndex+5);

                Text year = new Text();

                year.set(yearValue);

                context.write(year, new IntWritable(1));
```

```
    }
}
```

Reducer1:

```
package info7250.bigData.Project.Part1_1;


import java.io.IOException;


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


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


    IntWritable result = new IntWritable();


    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException{
            int count = 0;
            for(IntWritable value : values){
                    count += value.get();
            }
            result.set(count);
            Text year = new Text();
            year.set(key);
            context.write(year, result);
    }
}
```

Map2:

```
package info7250.bigData.Project.Part1_1;


import java.io.IOException;
```

```
}


import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;


public class Map2 extends Mapper<LongWritable, Text, CompositeKey, NullWritable>{


	protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException{

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

			CompositeKey cpk = new CompositeKey(tokens[0], Integer.parseInt(tokens[1]));

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

	}

}
```

Reducer2:

```
package info7250.bigData.Project.Part1_1;


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.Reducer;


public class Reducer2 extends Reducer<CompositeKey, NullWritable, Text, IntWritable>{


	protected void reduce(CompositeKey cmpk, Iterable<NullWritable> values, Context context) throws
IOException, InterruptedException{

			Text year = new Text();

			year.set(cmpk.getYear());

			IntWritable sum = new IntWritable(cmpk.getCount());

			context.write(year, sum);

	}
}
```

CompsiteKey:

```
package info7250.bigData.Project.Part1_1;

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

import org.apache.hadoop.io.WritableComparable;

public class CompositeKey implements WritableComparable<CompositeKey>{
    private String year;
    private int count;
    public CompositeKey(){
    }
    public CompositeKey(String year, int count) {
            this.year = year;
            this.count = count;
    }
    public String getYear() {
            return year;
    }
    public void setYear(String year) {
            this.year = year;
    }
    public int getCount() {
            return count;
    }
    public void setCount(int count) {
            this.count = count;
    }
    @Override
```

```
	public void readFields(DataInput in) throws IOException {

		year = in.readUTF();

		count = in.readInt();

	}

	@Override

	public void write(DataOutput out) throws IOException {

		out.writeUTF(year);

		out.writeInt(count);

	}

	@Override

	public int compareTo(CompositeKey o) {

		return o.count - count;

	}

}
```

CompositeKeyComparator:

```
package info7250.bigData.Project.Part1_1;

import org.apache.hadoop.io.WritableComparator;

public class CompositeKeyComparator extends WritableComparator {

	public CompositeKeyComparator(){
		super(CompositeKey.class, true);
	}

	public int compare(CompositeKey a, CompositeKey b){
		return b.compareTo(a);
	}
}
```

NaturalGroupingKeyComparator:

```
package info7250.bigData.Project.Part1_1;


import org.apache.hadoop.io.WritableComparator;

public class NaturalGroupingKeyComparator extends WritableComparator {


	public NaturalGroupingKeyComparator(){
		super(CompositeKey.class, true);
	}
	public int compare(CompositeKey a, CompositeKey b){
		return a.getYear().compareTo(b.getYear());
	}

}
```

NaturalKeyPartitioner:

```
package info7250.bigData.Project.Part1_1;


import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.mapreduce.Partitioner;


public class NaturalKeyPartitioner extends Partitioner<CompositeKey, IntWritable>{


	@Override

	public int getPartition(CompositeKey key, IntWritable value, int partitionsCount) {

		return key.getYear().hashCode() % partitionsCount;

	}

}
```

Driver:

```
package info7250.bigData.Project.Part1_1;


import java.io.IOException;


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.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

```
public class Driver {

    public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException{
      // Create a new Job
            Configuration config = new Configuration();
            Job job = Job.getInstance(config,"movieCountYear-1");
      job.setJarByClass(Driver.class);

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

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

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

      job.setMapperClass(Map1.class);
      job.setReducerClass(ReducerClass.class);

      if (!job.waitForCompletion(true)) {
        System.exit(1);
      }

      Job job2 = Job.getInstance(config,"movieCountYear-2");
      job2.setJarByClass(Driver.class);
      job2.setNumReduceTasks(1);
      job2.setGroupingComparatorClass(NaturalGroupingKeyComparator.class);
      job2.setSortComparatorClass(CompositeKeyComparator.class);
      job2.setPartitionerClass(NaturalKeyPartitioner.class);

      FileInputFormat.addInputPath(job2, new Path("/out"));
```

```
            FileOutputFormat.setOutputPath(job2, new Path(args[1]));

            job2.setInputFormatClass(TextInputFormat.class);
            job2.setOutputFormatClass(TextOutputFormat.class);

            job2.setMapOutputKeyClass(CompositeKey.class);
            job2.setMapOutputValueClass(NullWritable.class);

            job2.setMapperClass(Map2.class);
            job2.setReducerClass(Reducer2.class);

             job2.setOutputKeyClass(Text.class);
             job2.setOutputValueClass(IntWritable.class);

             System.exit(job2.waitForCompletion(true) ? 0:1);
        }
}
```

Part 3:

Map:

```
package info7250.bigData.Project.Part2;

import java.io.IOException;

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

public class Map extends Mapper<LongWritable, Text, Text, IntWritable>{

        protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException{
```

```
            String line = value.toString();
            if(line.equals("movieId,title,genres")){
                return;
            }
            int genreStartIndex = line.lastIndexOf(",");
            String genreString = line.substring(genreStartIndex+1);
            if(genreString.equals("(no genres listed)")) {
                return;
            }
            String[] genreList = genreString.split("\\|");
            Text genre = new Text();
            for(int i=0; i< genreList.length; i++) {
                genre.set(genreList[i]);
                context.write(genre, new IntWritable(1));
            }
    }
}
```

Reducer:

```
package info7250.bigData.Project.Part2;

import java.io.IOException;

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

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

    IntWritable result = new IntWritable();

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

```
                int count = 0;
                for(IntWritable value : values){
                        count += value.get();
                }
                result.set(count);
                Text genre = new Text();
                genre.set("genre " + key);
                context.write(genre, result);
        }
}
```

Driver:

```
package info7250.bigData.Project.Part2;

import java.io.IOException;

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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class Driver {

    public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException{
      // Create a new Job
            Configuration config = new Configuration();
            Job job = Job.getInstance(config,"movieGenreCount");
      job.setJarByClass(Driver.class);
```

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

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

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

        job.setMapperClass(Map.class);
        job.setReducerClass(ReducerClass.class);

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

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

}
```

Part 4:

Driver:

```
package info7250.bigData.Project.Part3;

import java.io.IOException;

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.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


public class Driver {


    public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException{

        // Create a new Job

                Configuration config = new Configuration();

                Job job = Job.getInstance(config,"movieTagCount");

        job.setJarByClass(Driver.class);


        FileInputFormat.addInputPath(job, new Path(args[0]));

        FileOutputFormat.setOutputPath(job, new Path(args[1]));


        job.setInputFormatClass(TextInputFormat.class);

        job.setOutputFormatClass(TextOutputFormat.class);


        job.setMapOutputKeyClass(Text.class);

        job.setMapOutputValueClass(IntWritable.class);


        job.setMapperClass(Map.class);

        job.setReducerClass(ReducerClass.class);


        job.setOutputKeyClass(Text.class);

        job.setOutputValueClass(IntWritable.class);


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

    }

}
```

Mapper:

```
package info7250.bigData.Project.Part3;
```

```
import java.io.IOException;


import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;


public class Map extends Mapper<LongWritable, Text, Text, IntWritable>{


    protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException{

            String line = value.toString();

            if(line.equals("userId,movieId,tag,timestamp")){

                return;

            }

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

            Text movie = new Text();

            movie.set(data[1]);

            context.write(movie, new IntWritable(1));

    }

}
```

Reducer:

```
package info7250.bigData.Project.Part3;


import java.io.IOException;


import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;


public class ReducerClass extends Reducer<Text, IntWritable, Text, IntWritable>{
	IntWritable result = new IntWritable();

	protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException{
		int count = 0;
		for(IntWritable value : values){
			count += value.get();
		}
		result.set(count);
		Text output = new Text();
		output.set("movieId: " + key);
		context.write(output, result);
	}
}
```

Part 5:

CompositeKey:

```
package info7250.bigData.Project.Part3_1;

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

import org.apache.hadoop.io.WritableComparable;

public class CompositeKey implements WritableComparable<CompositeKey>{

	private String title;
	private int count;

	public CompositeKey(){
	}
```

```
public CompositeKey(String title, int count) {
        this.title = title;
        this.count = count;
}

public String getTitle() {
        return title;
}

public void setTitle(String title) {
        this.title = title;
}

public int getCount() {
        return count;
}

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

@Override
public void readFields(DataInput in) throws IOException {
        title = in.readUTF();
        count = in.readInt();
}

@Override
public void write(DataOutput out) throws IOException {
        out.writeUTF(title);
        out.writeInt(count);
```

```
	}


	@Override
	public int compareTo(CompositeKey o) {
		return o.count - count;
	}
}
```

CompositeKeyComparator:

```
package info7250.bigData.Project.Part3_1;

import org.apache.hadoop.io.WritableComparator;

public class CompositeKeyComparator extends WritableComparator {

	public CompositeKeyComparator(){
		super(CompositeKey.class, true);
	}

	public int compare(CompositeKey a, CompositeKey b){
		return b.compareTo(a);
	}
}
```

Driver:

```
package info7250.bigData.Project.Part3_1;

import java.io.IOException;

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.input.MultipleInputs;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class Driver {

	public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException{
	    // Create a new Job
		Configuration config = new Configuration();
		Job job = Job.getInstance(config,"movieNameTagCount");
	   job.setJarByClass(Driver.class);
```

```
        MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, Map1.class);
        MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class, Map2.class);

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

        job.getConfiguration().set("join.type","inner");
        job.setReducerClass(ReducerClass.class);

        job.setOutputFormatClass(TextOutputFormat.class);
        //FileOutputFormat.setOutputPath(job, new Path("/out2"));
        TextOutputFormat.setOutputPath(job, new Path("/out2"));

        if (!job.waitForCompletion(true)) {
          System.exit(1);
        }

        Job job2 = Job.getInstance(config,"movieNameTagCount-2");
        job2.setJarByClass(Driver.class);
        job2.setNumReduceTasks(1);
        job2.setGroupingComparatorClass(NaturalGroupingKeyComparator.class);
        job2.setSortComparatorClass(CompositeKeyComparator.class);
        job2.setPartitionerClass(NaturalKeyPartitioner.class);

        FileInputFormat.addInputPath(job2, new Path("/out2"));
        FileOutputFormat.setOutputPath(job2, new Path(args[2]));

        job2.setInputFormatClass(TextInputFormat.class);
        job2.setOutputFormatClass(TextOutputFormat.class);

        job2.setMapOutputKeyClass(CompositeKey.class);
        job2.setMapOutputValueClass(NullWritable.class);

        job2.setMapperClass(Map3.class);
        job2.setReducerClass(Reducer2.class);

        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(IntWritable.class);

        System.exit(job2.waitForCompletion(true) ? 0:1);
    }

}
```

Map1:

```
package info7250.bigData.Project.Part3_1;

import java.io.IOException;

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

```
public class Map1 extends Mapper<LongWritable, Text, Text, Text>{

	protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException{
			String line = value.toString();
			if(line.equals("userId,movieId,tag,timestamp")){
				return;
			}
			String[] data = line.split(",");
			Text movie = new Text();
			movie.set(data[1]);
			Text count = new Text();
			count.set("A");
			context.write(movie,count);
	}

}
```

Map2:

```
package info7250.bigData.Project.Part3_1;

import java.io.IOException;

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

public class Map2 extends Mapper<LongWritable, Text, Text, Text>{

	protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException{
			String line = value.toString();
			if(line.equals("movieId,title,genres")){
				return;
			}
			int firstIndex = line.indexOf(",");
			int lastIndex = line.lastIndexOf(",");
			String movieTitle = line.substring(firstIndex+1, lastIndex);
			String movieId = line.substring(0,firstIndex);
			Text movie = new Text();
			movie.set(movieId);
			Text title = new Text();
			title.set("B"+movieTitle);
			context.write(movie, title);
	}

}
```

Map3:

```
package info7250.bigData.Project.Part3_1;

import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class Map3 extends Mapper<LongWritable, Text, CompositeKey, NullWritable>{

	protected void map(LongWritable key, Text value, Context context) throws IOException, 
InterruptedException{
			String tokens[] = value.toString().split("\\t");
			CompositeKey cpk = new CompositeKey(tokens[0], Integer.parseInt(tokens[1]));
			context.write(cpk, NullWritable.get());
	}

}
```

NaturalKeyPartitioner:

```
package info7250.bigData.Project.Part3_1;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Partitioner;

public class NaturalKeyPartitioner extends Partitioner<CompositeKey, IntWritable>{

	@Override
	public int getPartition(CompositeKey key, IntWritable value, int partitionsCount) {
			return key.getTitle().hashCode() % partitionsCount;
	}
}
```

NaturalGroupingKeyComparator:

```
package info7250.bigData.Project.Part3_1;

import org.apache.hadoop.io.WritableComparator;

public class NaturalGroupingKeyComparator extends WritableComparator {


	public NaturalGroupingKeyComparator(){
			super(CompositeKey.class, true);
	}

	public int compare(CompositeKey a, CompositeKey b){
			return a.getTitle().compareTo(b.getTitle());
	}

}
```

Reducer1:

```
package info7250.bigData.Project.Part3_1;

import java.io.IOException;

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

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

   @Override
   protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
      int counter = 0;
      String movieName = new String();
      for(Text text : values){
         if(text.charAt(0) == 'A'){
            counter++;
         } else if(text.charAt(0) == 'B'){
             movieName = (text.toString().substring(1));
         }
      }
      executeJoinLogic(context, counter, movieName);
   }

   public void executeJoinLogic(Context context, int counter, String movieName) throws IOException,
InterruptedException {
      String joinType = context.getConfiguration().get("join.type");

      //INNER JOIN OPERATION
      if(joinType.equalsIgnoreCase("inner")){
         if(counter > 0 && movieName.length() > 0){
                  Text movie = new Text();
                  movie.set(movieName);
            context.write(movie, new IntWritable(counter));
         }
      }
   }
}
```

Reducer2:

```
package info7250.bigData.Project.Part3_1;

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.Reducer;

public class Reducer2 extends Reducer<CompositeKey, NullWritable, Text, IntWritable>{

      protected void reduce(CompositeKey cmpk, Iterable<NullWritable> values, Context context) throws
IOException, InterruptedException{
                  Text year = new Text();
                  year.set(cmpk.getTitle());
                  IntWritable sum = new IntWritable(cmpk.getCount());
                  context.write(year, sum);
      }
}
```

Part 6:
Driver:

```
package info7250.bigData.Project.Part4;

import java.io.IOException;

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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class Driver {

    public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException{
        // Create a new Job
            Configuration config = new Configuration();
            Job job = Job.getInstance(config,"userTagCount");
        job.setJarByClass(Driver.class);

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

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

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

        job.setMapperClass(Map.class);
        job.setReducerClass(ReducerClass.class);

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

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

}
```

Mapper:

```
package info7250.bigData.Project.Part4;

import java.io.IOException;

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

public class Map extends Mapper<LongWritable, Text, Text, IntWritable>{

	protected void map(LongWritable key, Text value, Context context) throws IOException, 
InterruptedException{
			String line = value.toString();
			if(line.equals("userId,movieId,tag,timestamp")){
				return;
			}
			String[] data = line.split(",");
			Text movie = new Text();
			movie.set(data[0]);
			context.write(movie, new IntWritable(1));
	}

}
```

Reducer:

```
package info7250.bigData.Project.Part4;

import java.io.IOException;

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

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

	IntWritable result = new IntWritable();

	protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, 
InterruptedException{
			int count = 0;
			for(IntWritable value : values){
				count += value.get();
			}
			result.set(count);
			Text outKey = new Text();
			outKey.set("userId: " + key);
			context.write(outKey, result);
	}
}
```

Part 7:

```
genomeScore = LOAD '/home/sahithi/Desktop/ml-25m/genome-scores.csv' USING PigStorage(',') AS 
(movieId:int, tagId:int, relevance:float);

genomeTag = LOAD '/home/sahithi/Desktop/ml-25m/genome-tags.csv' USING PigStorage(',') AS (tagId:int, 
tag:chararray);
```

```
genome = JOIN genomeTag by tagId, genomeScore by tagId;

genomeFilter = FILTER genome by relevance >= 0.7;

tagData = LOAD '/home/sahithi/Desktop/ml-25m/tags.csv' Using PigStorage(',') As
(userId:int,movieId:int,tag:chararray,timestamp:double);

genomeTagCombine = JOIN genomeFilter by tag, tagData by tag;

genomeRelevantUser = FOREACH genomeTagCombine GENERATE userId, relevance;

genomeTagCombineGroup = GROUP genomeRelevantUser by userId;

genomeTagAvg = FOREACH genomeTagCombineGroup GENERATE
group,AVG(genomeRelevantUser.relevance) AS avgRelevance:float;

genomeTagSort = ORDER genomeTagAvg by avgRelevance DESC;
```

Part 8:

Mapper:

```
package info7250.bigData.Project.Part5;


import java.io.IOException;


import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;


public class Map extends Mapper<LongWritable, Text, Text, IntWritable>{


    protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException{

            String line = value.toString();

            if(line.equals("userId,movieId,rating,timestamp")){

                    return;

            }

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

            Text rating = new Text();

            rating.set(data[2]);
```

```
                    context.write(rating, new IntWritable(1));
        }


}
```

Reducer:

```
package info7250.bigData.Project.Part5;


import java.io.IOException;


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


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


        IntWritable result = new IntWritable();


        protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException{
                    int count = 0;
                    for(IntWritable value : values){
                                count += value.get();
                    }
                    result.set(count);
                    Text outKey = new Text();
                    outKey.set("rating: " + key);
                    context.write(outKey, result);
        }
}
```

Driver:

```
package info7250.bigData.Project.Part5;
```

```
import java.io.IOException;


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.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


public class Driver {


    public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException{

        // Create a new Job

                Configuration config = new Configuration();

                Job job = Job.getInstance(config,"ratingCount");

        job.setJarByClass(Driver.class);


        FileInputFormat.addInputPath(job, new Path(args[0]));

        FileOutputFormat.setOutputPath(job, new Path(args[1]));


        job.setInputFormatClass(TextInputFormat.class);

        job.setOutputFormatClass(TextOutputFormat.class);


        job.setMapOutputKeyClass(Text.class);

        job.setMapOutputValueClass(IntWritable.class);


        job.setMapperClass(Map.class);
        job.setReducerClass(ReducerClass.class);


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


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


}
```

Part 9:

```
movieData = LOAD '/home/sahithi/Desktop/movies.txt' Using PigStorage() As(MovieId:int, Title:chararray,
Genre:chararray);

linksData = LOAD '/home/sahithi/Desktop/ml-25m/links.csv' Using PigStorage(',') As(MovieId:int, ImdbId:int,
TmdbId:int);

movie_Links = JOIN movieData by MovieId, linksData by MovieId;

movieLinkAdded = FOREACH movie_Links GENERATE *,
CONCAT('http:://www.imdb.com/title/tt',(chararray)ImdbId) AS IMDBLink:chararray;

outLinks = FOREACH movieLinkAdded GENERATE Title,IMDBLink;

lm = LIMIT outLinks 25;

dump lm;
```

Part 10:

```
ratingData = LOAD '/home/sahithi/Desktop/ml-25m/ratings.csv' USING PigStorage(',') AS
(userId:int,movieId:int,rating:float,timestamp:double);

groupRating = Group ratingData By movieId;

ratingAvg = FOREACH groupRating GENERATE group, AVG(ratingData.rating) AS average:float;

tagData = LOAD '/home/sahithi/Desktop/ml-25m/tags.csv' Using PigStorage(',') As
(userId:int,movieId:int,tag:chararray,timestamp:double);

groupTag = GROUP tagData by movieId;

tagConcat = FOREACH groupTag GENERATE group, BagToTuple(tagData.tag) as tagCollection;

movieData = LOAD '/home/sahithi/Desktop/movies.txt' Using PigStorage() As(MovieId:int, Title:chararray,
Genre:chararray);

tagMovieRatingJoin = JOIN ratingAvg by group,tagConcat by group,movieData by MovieId;

tagMovieRating = FOREACH tagMovieRatingJoin GENERATE Title,average,tagCollection;
```

```
lmi = LIMIT tagMovieRating 1;

dump lmi;
```

Part 11:

Driver:

```
package info7250.bigData.Project.Part8;


import java.io.IOException;


import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.chain.ChainMapper;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


public class Driver {


    public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException{

        // Create a new Job

                Configuration config = new Configuration();

                Job job = Job.getInstance(config,"filterYearGenre");

        job.setJarByClass(Driver.class);

        job.setNumReduceTasks(0);


        Configuration mapConf1 = new Configuration(false);

        ChainMapper.addMapper(job, Map.class, LongWritable.class, Text.class,

            Text.class, CompositeValue.class,  mapConf1);
        Configuration mapConf2 = new Configuration(false);
        ChainMapper.addMapper(job, Map2.class, Text.class, CompositeValue.class,
            Text.class, NullWritable.class, mapConf2);

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

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

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

}
```

Map1:

```
package info7250.bigData.Project.Part8;

import java.io.IOException;

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

public class Map extends Mapper<LongWritable, Text, Text, CompositeValue>{

        protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException{
                String line = value.toString();
                if(line.equals("movieId,title,genres")){
                        return;
                }
                int firstIndex = line.indexOf(",");

                int lastIndex = line.lastIndexOf(",");

                String movieTitle = line.substring(firstIndex+1, lastIndex);

                String movieId = line.substring(0,firstIndex);

                String genres = line.substring(lastIndex+1);

                CompositeValue cv = new CompositeValue(movieTitle, genres);

                Text id = new Text();

                id.set(movieId);

                context.write(id, cv);

        }


}
```

Map2:

```
package info7250.bigData.Project.Part8;


import java.io.IOException;




import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;


public class Map2 extends Mapper<Text, CompositeValue, Text, NullWritable>{


        protected void map(Text key, CompositeValue value, Context context) throws IOException,
InterruptedException{

                if(value.getTitle().contains("2011") && value.getGenre().contains("Action")) {

                        Text movieName = new Text();

                        movieName.set(value.getTitle());

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

                }else {
                        return;
                }
        }
}
```

CompositeValue:

```
package info7250.bigData.Project.Part8;

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

import org.apache.hadoop.io.Writable;

public class CompositeValue implements Writable{

    private String title;
    private String genre;

    public CompositeValue(){
    }

    public CompositeValue(String title, String genre) {
            this.title = title;
            this.genre = genre;
    }

    public String getTitle() {
            return title;
    }

    public void setTitle(String title) {
            this.title = title;
    }

    public String getGenre() {
            return genre;
    }

    public void setGenre(String genre) {
            this.genre = genre;
    }

    @Override
    public void readFields(DataInput in) throws IOException {
            title = in.readUTF();
            genre = in.readUTF();
    }

    @Override
```

```
	public void write(DataOutput out) throws IOException {
		out.writeUTF(title);
		out.writeUTF(genre);
	}

}
```

Part 12:

```
genomeData = LOAD '/home/sahithi/Desktop/ml-25m/genome-scores.csv' Using PigStorage(',')
As(movieId:int, tagId:int, relevance:float);

groupGenome = GROUP genomeData by movieId;

GenomeAvg = FOREACH groupGenome GENERATE group, AVG(genomeData.relevance) AS average:float;

movieData = LOAD '/home/sahithi/Desktop/movies.txt' Using PigStorage() As(MovieId:int, Title:chararray,
Genre:chararray);

movieGenome = JOIN movieData by MovieId, GenomeAvg by group;

movieRelevance = FOREACH movieGenome GENERATE Title, average;

orderMovieRelevance = ORDER movieRelevance BY average DESC;
```

Part 13:

```
ratingData = LOAD '/home/sahithi/Desktop/ml-25m/ratings.csv' USING PigStorage(',') AS
(userId:int,movieId:int,rating:float,timestamp:double);

grpByRating = GROUP ratingData by movieId;

grpByRatingAvg = FOREACH grpByRating GENERATE group, AVG(ratingData.rating) AS average:float;

movieData = LOAD '/home/sahithi/Desktop/movies.txt' Using PigStorage() As(MovieId:int, Title:chararray,
Genre:chararray);

movierating = JOIN movieData by MovieId , grpByRatingAvg by group;

movieGenreAvgRating = FOREACH movierating GENERATE Title,Genre,average;

movieGenreFlatten = FOREACH movieGenreAvgRating GENERATE Title,average, flatten(TOKENIZE(Genre, '|'));

movieGenreGroup = GROUP movieGenreFlatten by token;

movieGenreGroupDesc = FOREACH movieGenreGroup {

movieGenreOrder = ORDER movieGenreFlatten BY average DESC;

top = LIMIT movieGenreOrder 5;

GENERATE group, flatten(top);

}

lmi = LIMIT movieGenreGroupDesc 20;
```

```
dump lmi;
```

Part 14:

```
ratingData = LOAD '/home/sahithi/Desktop/ml-25m/ratings.csv' USING PigStorage(',') AS
(userId:int,movieId:int,rating:float,timestamp:double);

movieData = LOAD '/home/sahithi/Desktop/movies.txt' Using PigStorage() As(MovieId:int, Title:chararray,
Genre:chararray);

movieRating = JOIN movieData by MovieId, ratingData by movieId;

userRatingGenre = FOREACH movieRating GENERATE userId, rating,flatten(TOKENIZE(Genre, '|'));

userGenreGroup = Group userRatingGenre by (userId,token);

userGenreAvg = FOREACH userGenreGroup GENERATE group, AVG(userRatingGenre.rating) as average:float;

userGenreOrder = ORDER userGenreAvg BY average DESC;

lmi = LIMIT userGenreDesc 10;

dump lmi;
```