Homework - 2

Contents of the folder -
1) Scala code file (*SaiSree_Kamineni_SON.scala*)
2) JAR files (*SaiSree_Kamineni_SON.jar*)
3) OutputFiles (*SaiSree_Kamineni_SON.case1_1200.txt, SaiSree_Kamineni_SON.case1_1300.txt, SaiSree_Kamineni_SON.case2_500.txt, SaiSree_Kamineni_SON.case2_600.txt*)
4) InputFiles (*ratings.dat, users.dat*)

Note - I haven't set up SPARK_HOME variable. So I run the scripts and commands from inside "*spark-1.6.3-bin-hadoop2.4*" directory.
Place this UnZipped folder in *spark-1.6.3-bin-hadoop2.4* directory

Steps to run the jar files -
Case – 1
S = 1300
./bin/spark-submit ./SaiSree_Kamineni_hw2/Solution/SaiSree_Kamineni_SON.jar 1 ./SaiSree_Kamineni_hw2/InputFiles/ratings.dat ./SaiSree_Kamineni_hw2/InputFiles/users.dat 1300

Output will be stored in current directory that is spark-1.6.3-bin-hadoop2.4 with name SaiSree_Kamineni_SON.case1_1300.txt.

S = 1200
./bin/spark-submit ./SaiSree_Kamineni_hw2/Solution/SaiSree_Kamineni_SON.jar 1 ./SaiSree_Kamineni_hw2/InputFiles/ratings.dat ./SaiSree_Kamineni_hw2/InputFiles/users.dat 1200

Output will be stored in current directory that is spark-1.6.3-bin-hadoop2.4 with name SaiSree_Kamineni_SON.case1_1200.txt.

Case – 2
S = 600
./bin/spark-submit ./SaiSree_Kamineni_hw2/Solution/SaiSree_Kamineni_SON.jar 2 ./SaiSree_Kamineni_hw2/InputFiles/ratings.dat ./SaiSree_Kamineni_hw2/InputFiles/users.dat 600

Output will be stored in current directory that is spark-1.6.3-bin-hadoop2.4 with name SaiSree_Kamineni_SON.case2_600.txt.

S = 500
./bin/spark-submit ./SaiSree_Kamineni_hw2/Solution/SaiSree_Kamineni_SON.jar 2 ./SaiSree_Kamineni_hw2/InputFiles/ratings.dat ./SaiSree_Kamineni_hw2/InputFiles/users.dat 500

Output will be stored in current directory that is *spark-1.6.3-bin-hadoop2.4* with name *SaiSree_Kamineni_SON.case2_500.txt*.

Output format

As mentioned in the problem statement.

```
(1), (50), (110), (260), (296), (318), (356), (457), (480), (527), (541), (589), (593),
(608), (780), (858), (924), (1036), (1097), (1127), (1196), (1197), (1198), (1200),
(1210), (1214), (1221), (1240), (1259), (1265), (1270), (1387), (1580), (1610), (1617),
(2000), (2028), (2396), (2571), (2628), (2716), (2762), (2791), (2858), (2916), (2987),
(2997), (3175), (3578)
(110, 480), (110, 589), (110, 1196), (110, 2028), (260, 480), (260, 589), (260, 858),
(260, 1097), (260, 1196), (260, 1198), (260, 1210), (260, 1214), (260, 1240), (260, 1270),
(260, 1580), (260, 2028), (260, 2571), (260, 2628), (260, 2858), (260, 2916), (480, 589),
(480, 1196), (480, 1210), (480, 1580), (480, 2571), (480, 2858), (480, 2916), (589, 1196),
(589, 1198), (589, 1210), (589, 1240), (589, 1270), (589, 1580), (589, 2028), (589, 2571),
(589, 2628), (589, 2858), (589, 2916), (593, 608), (593, 2858), (608, 2858), (1097, 1196),
(1196, 1197), (1196, 1198), (1196, 1200), (1196, 1210), (1196, 1214), (1196, 1240), (1196,
1270), (1196, 1580), (1196, 2028), (1196, 2571), (1196, 2628), (1196, 2858), (1196, 2916),
(1198, 1210), (1210, 1240), (1210, 1270), (1210, 1580), (1210, 2028), (1210, 2571), (1210,
2858), (1240, 2571), (1580, 2571), (1580, 2628), (1580, 2916), (2028, 2571), (2028, 2858),
(2571, 2628), (2571, 2858), (2571, 2916), (2762, 2858), (2858, 2997)
(260, 480, 1196), (260, 589, 1196), (260, 589, 1210), (260, 589, 2571), (260, 1196, 1198),
(260, 1196, 1210), (260, 1196, 1240), (260, 1196, 1270), (260, 1196, 2571), (260, 1210,
2571), (480, 589, 1580), (480, 589, 2571), (589, 1196, 1210), (589, 1196, 2571), (589,
1580, 2571), (1196, 1198, 1210), (1196, 1210, 2571)
```

Approach –

1) Read the files given in arguments and created users, ratings RDDs.
2) Based on the case given in argument, joined users and ratings RDD to create original baskets.
    a. For case 1 – userID is the key and value is a list of MovieIDs rated by the male user.
    b. For case 2 – MovieID is the key and value is a list of female urserIDs who rated the movie.
3) Repartitioned the basket into 10 partitions, created a temporary variable (n) to hold threshold for partitions (s/10).

    ######### SINGLETONS #######
4) In the 1st mapper, for each basket flattened the list of values and created (value,1) as the key value pair.
5) Using MapPartitions, performed group by over the new key and created a list as value. Compared size of the list with ‘n’ and filtered the values that didn’t exceed ‘n’. Returned RDD as (key, 1).
6) The 1st reducer identifies the distinct records of all (key, 1).
7) Created a list with all the keys from 1st reducer.
8) In 2nd mapper, flattened the basket again and filtered the keys which aren’t part of the list from step #7.

9) In 2nd reducer, counted the occurrence of each key using reduceByKey, sorted the keys and returned them.
10) Output from this step is stored in a string.

    ######## PAIRS ######
11) Output from Step #9 is stored in a list and from this the combinations of size 2 are generated and stored in another list.
12) Input basket is modified by filtering each basket to only contain the singletons from step #9.
13) Steps #5 - #10 are repeated except in mapper stage, pairs are generated using the combinations
    ######## TRIPLETS, QUATRIPLETS, ……#########
14) Generation of the next sets follows the same procedure.
15) All the pairs identified in Step #13 are flattened to a list and combinations of 3 are obtained.
16) By applying monotonicity, the triplets are split into pairs and compared with the pairs form Step #13. Only if all the possible pairs are present, the triplet makes the list which is used to filter unnecessary triplets in 1st mapper.
17) Whenever sets are empty, the loop breaks and output is written to file.