The following **MapReduce** program in **Spark** that implements a simple “**Mutual/Common friend** list of two friends". The key idea is that if two people are friend then they have a lot of mutual/common friends. This question will give any two Users as **input**, **outpu**t the list of the user id of their mutual friends.

**Input:**

Input files : **soc-LiveJournal1Adj.txt**

The input contains the adjacency list and has multiple lines in the following format: <***User***><TAB><***Friends***>

Here, <**User**> is a unique **integer ID** corresponding to a unique user and <Friends> is a comma-separated list of unique **IDs** (<User> ID) corresponding to the friends of the user. Note that the friendships are mutual (i.e., edges are undirected): if **A** is friend with B then B is also friend with A. The data provided is consistent with that rule as there is an explicit entry for each side of each edge. So when you make the pair, always consider (A, B) or (B, A) for user A and B but not both.

**Output:**

The output should contain one line per user in the following format:

<**User\_A**><TAB><**User\_B**><TAB><**Mutual/Common Friend List**>

where <**User\_A**> & <**User\_B**> are **unique IDs** corresponding to a user A and B (A and B are friend). < **Mutual/Common Friend List** > is a comma-separated list of unique **IDs** corresponding to mutual friend list of User **A** and **B**.

1. ***Expliquer le rôle de la function pairs ?***

//generate **((user1, user2), friendList)** for pair counts

def **pairs**(str: Array[String]) = {

**val users = str(1).split(",")**

**val user=str(0)**

**val n = users.length**

**for(i <- 0 until n) yield {**

**val pair = if(user < users(i)) {**

**(user,users(i))**

**} else {**

**(users(i),user)**

**}**

**(pair, users) }** }

*2-* ***On considère le code Spark en Scala******(MutualFriends.scala) à completer :***

//Main of our program

**---1-- val data = sc.textFile("**soc-LiveJournal1Adj.txt**")**

**---2-- val data1= ……………………………………………………………………….**

**---3-- val pairCounts = ……………………………………………………………**

**---4-- val p1= …………………………………………………………………………….**

**---5-- p1.saveAsTextFile("**output**")**

*3-* ***Expliquer le rôle du code Sparks en Scala suivant :***

**var ans=""**

**-----1--- val p2=p1.map(x=>x.split("\t")).filter(x => (x.size == 3)).filter(x=>(x(0)=="0"&&x(1)=="4")).flatMap(x=>x(2).split(",")).collect()**

**-----2--- ans=ans+"0"+"\t"+"4"+"\t"+p2.mkString(",")+"\n"**

**-----3--- val p3=p1.map(x=>x.split("\t")).filter(x => (x.size == 3)).filter(x=>x(0)=="20"&&x(1)=="22939").flatMap(x=>x(2).split(",")).collect()**

**-----4--- ans=ans+"20"+"\t"+"22939"+"\t"+p3.mkString(",")+"\n"**

**-----5--- val answer=sc.parallelize(Seq(ans))**

**-----6--- answer.saveAsTextFile("output1")**

//Main of our program

**val data = sc.textFile("**soc-LiveJournal1Adj.txt**")**

**val data1=data.map(x=>x.split("\t")).filter(li => (li.size == 2))**

**val pairCounts = data1.flatMap(pairs).reduceByKey({ case (param1,param2) => (param1.intersect(param2)) })**

**val p1=pairCounts.map({case ((param1, param2),param3) => (param1+"\t"+param2+"\t"+param3.mkString(","))})**

**p1.saveAsTextFile("**output**")**

**var ans=""**

**val p2=p1.map(x=>x.split("\t")).filter(x => (x.size == 3)).filter(x=>(x(0)=="0"&&x(1)=="4")).flatMap(x=>x(2).split(",")).collect()**

**ans=ans+"0"+"\t"+"4"+"\t"+p2.mkString(",")+"\n"**

**val p3=p1.map(x=>x.split("\t")).filter(x => (x.size == 3)).filter(x=>x(0)=="20"&&x(1)=="22939").flatMap(x=>x(2).split(",")).collect()**

**ans=ans+"20"+"\t"+"22939"+"\t"+p3.mkString(",")+"\n"**

**val p4=p1.map(x=>x.split("\t")).filter(x => (x.size == 3)).filter(x=>x(0)=="1"&&x(1)=="29826").flatMap(x=>x(2).split(",")).collect()**

**ans=ans+"1"+"\t"+"29826"+"\t"+p4.mkString(",")+"\n"**

**val p5=p1.map(x=>x.split("\t")).filter(x => (x.size == 3)).filter(x=>x(0)=="19272"&&x(1)=="6222").flatMap(x=>x(2).split(",")).collect()**

**ans=ans+"6222"+"\t"+"19272"+"\t"+p5.mkString(",")+"\n"**

**val p6=p1.map(x=>x.split("\t")).filter(x => (x.size == 3)).filter(x=>x(0)=="28041"&&x(1)=="28056").flatMap(x=>x(2).split(",")).collect()**

**ans=ans+"28041"+"\t"+"28056"+"\t"+p6.mkString(",")+"\n"**

**val answer=sc.parallelize(Seq(ans))**

**answer.saveAsTextFile("output1")**