**Bi-directional friend recommendation Algorithm**

Algorithm:

* Suppose we have the data in following format: A:B C
* Here A represents the user who has friends B and C.
* Mapper: In this stage we emit list of possible combinations among the friends. Ex (B,C),1
* In this mapper stage we also emit for every pair of existing friends -1 value. (A,B)-1, (A,C),-1
* We then do the groupByKey() and check if for any combination of friends, is there -1 value present in the list. If so, ignore that pair of friends.
* Then we use the reduceByKey method which gives us the list of common friends present between those 2 friends.
* We need to sort the data in descending order and give it as recommendation to the respective friends.

***r1 = sc.textFile('Documents/FriendRec\_IP')  
r2= r1.map(lambda x:(x.split(':')[0],x.split(':')[1].split(' ')))  
from itertools import combinations  
r3 = r2.map(lambda x:list(combinations(x[1],2)))  
r33 = r3.flatMap(lambda x:[(x[i],1) for i in range (0,len(x))])  
r33.collect()  
r4 = r2.map(lambda x: (x[0], [((x[1][i]),-1) for i in range (0, len(x[1]))]))  
def f(r4):return r4  
  
r5 = r4.flatMapValues(f)  
r6 = r5.map(lambda x:((x[0],x[1][0]),x[1][1]))  
r6.collect()  
r66 = r6.map(lambda x:(sorted(x[0]),x[1]))  
r66.collect()***[([u'A', u'B'], -1), ([u'A', u'C'], -1), ([u'A', u'B'], -1), ([u'B', u'D'], -1), ([u'B', u'C'], -1), ([u'A', u'C'], -1), ([u'B', u'C'], -1), ([u'B', u'D'], -1)]

***r8 = r6+r33  
r8.collect()***[((u'A', u'B'), -1), ((u'A', u'C'), -1), ((u'B', u'A'), -1), ((u'B', u'D'), -1), ((u'B', u'C'), -1), ((u'C', u'A'), -1), ((u'C', u'B'), -1), ((u'D', u'B'), -1), ((u'B', u'C'), 1), ((u'A', u'D'), 1), ((u'A', u'C'), 1), ((u'D', u'C'), 1), ((u'A', u'B'), 1)]  
  
***r9 = r8.groupByKey()***  
***r10 = r9.filter(lambda x: -1 not in x[1])  
r10.collect()***[((u'A', u'D'), <pyspark.resultiterable.ResultIterable object at 0x7f40dcf94490>), ((u'D', u'C'), <pyspark.resultiterable.ResultIterable object at 0x7f40dcf94750>)]  
  
***r11 = r10.map(lambda x:(x[0],sum(x[1])))  
r11.collect()***[((u'A', u'D'), 1), ((u'D', u'C'), 1)]  
  
***r12 = r11.map(lambda x:(x[0][0],(x[1],x[0][1])))  
r13 = r11.map(lambda x:(x[0][1],(x[1],x[0][0])))  
rFinal = r12+r13  
rFinal = rFinal.sortBy(lambda x:x[1][0], False)  
rFinal.collect()***

[(u'A', (1, u'D')), (u'D', (1, u'C')), (u'D', (1, u'A')), (u'C', (1, u'D'))]