Homework2

1.

(1) code screenshots

```
def getData(sc, filename):
    """
    Load data from raw text file into RDD and transform.
    Hint: transfromation you will use: map(<lambda function>).
    Args:
        sc (SparkContext): spark context.
        filename (string): hw2.txt cloud storage URI.
    Returns:
        RDD: RDD list of tuple of (<User>, [friend1, friend2, ... ]),
        each user and a list of user's friends
    """
    # read text file into RDD
    data = sc.textFile(filename).map(lambda line: line.split('\t'))

    # TODO: implement your logic here
    data = data.map(lambda tmp: (tmp[0], [num for num in tmp[1].split(',')]))

    return data
```

```
def mapFriends(line):
    """
    List out every pair of mutual friends, also record direct friends.
    Hint:
    For each <User>, record direct friends into a list:
    [(<User>, (friend1, 0)),(<User>, (friend2, 0)), ...],
    where 0 means <User> and friend are already direct friend,
    so you don't need to recommand each other.

    For friends in the list, each of them has a friend <User> in common,
    so for each of them, record mutual friend in both direction:
    (friend1, (friend2, 1)), (friend2, (friend1, 1)),
    where 1 means friend1 and friend2 has a mutual friend <User> in this "line"

    There are possibly multiple output in each input line,
    we applied flatMap to flatten them when using this function.
    Args:
        line (tuple): tuple in data RDD
    Yields:
        RDD: rdd like a list of (A, (B, 0)) or (A, (C, 1))
    """
    friends = line[1]
    directFriend = []
    mutualFriend = []
    for i in range(len(friends)):
        # Direct friend
        # TODO: implement your logic here
        yield (line[0], (friends[i], 0))

        for j in range(i+1, len(friends)):
            # Mutual friend in both direction
            # TODO: implement your logic here
            yield (friends[i], (friends[j], 1))
```

```python
def findMutual(line):
    """
    Find top 10 mutual friend for each person.
    Hint: For each <User>, input is a list of tuples of friend relations,
    whether direct friend (count = 0) or has friend in common (count = 1)

    Use friendDict to store the number of mutual friend that the current <User>
    has in common with each other <User> in tuple.
    Input:(User1, [(User2, 1), (User3, 1), (User2, 1), (User3, 0), (User2, 1)])
    friendDict stores: {User2:3, User3:1}
    directFriend stores: User3

    If a user has many mutual frineds and is not a direct frined, we recommend
    them to be friends.

    Args:
        line (tuple): a tuple of (<User1>, [(<User2>, 0), (<User3>, 1)....])
    Returns:
        RDD of tuple (line[0], returnList),
        returnList is a list of recommended friends
    """
    # friendDict, Key: user, value: count of mutual friends
    friendDict = defaultdict(int)
    # set of direct friends
    directFriend = set()
    # initialize return list
    returnList = []

    # TODO: Iterate through input to aggregate counts
    # save to friendDict and directFriend
    # friendList = line.map(lambda x : list(x[1])).collect()
    for i in line[1]:
        if i[1] == 0:
            directFriend.add(i[0])
        else:
            friendDict[i[0]] += i[1]
```

```python
    # TODO: Formulate output
    tmp = sorted(friendDict.items(), key=itemgetter(1,0))

    i = 0
    while len(returnList) < 10 and i < len(tmp):
        if tmp[i][0] not in directFriend:
            returnList.append(tmp[i][0])
        i += 1

    return (int(line[0]), returnList)

def main():
    # Configure Spark
    conf = SparkConf()
    sc = pyspark.SparkContext.getOrCreate(conf=conf)
    # The directory for the file
    filename = "gs://big_data_hw/hw2/q1.txt"

    # Get data in proper format
    data = getData(sc, filename)

    # Get set of all mutual friends
    mapData = data.flatMap(mapFriends).groupByKey()

    # For each person, get top 10 mutual friends
    getFriends = mapData.map(findMutual)

    # Only save the ones we want
    wanted = [924, 8941, 8942, 9019, 49824, 13420, 44410, 8974, 5850, 9993]
    result = getFriends.filter(lambda x: x[0] in wanted).collect()

    sc.stop()

if __name__ == "__main__":
    main()
```

(2) result screenshots

```
In [81]: for i in result:
             print(i)

(5850, [u'13283', u'13286', u'13289', u'13291', u'13292', u'13293', u'13295', u'13296', u'13299', u'13302'])
(9993, [u'13134', u'13478', u'13877', u'34299', u'34485', u'34642', u'37941'])
(44410, [u'10328', u'10370', u'10579', u'14052', u'15356', u'15731', u'16663', u'16680', u'16910', u'16965'])
(8974, [u'10318', u'10350', u'10471', u'10942', u'11030', u'11645', u'12109', u'12405', u'12430', u'12582'])
(8941, [u'8943', u'8944'])
(924, [u'11860', u'15416', u'2409', u'43748', u'45881', u'6995'])
(49824, [u'49825', u'49826', u'49827', u'49828', u'49829', u'49830', u'49831', u'49832', u'49833', u'49835'])
(9019, [u'9023', u'9022'])
(13420, [u'10370', u'10454', u'10523', u'10526', u'107', u'10985', u'11181', u'11214', u'11369', u'11880'])
(8942, [u'8943', u'8944'])
```

2.

(1) How many clusters / connected components in total for this dataset?

```
In [9]: result.select("component").distinct().count()

Out[9]: 917
```

(2) How many users in the top 10 clusters? There are different number of users in each clusters, so rank them and give the top 10 clusters with the largest amount of users.

```
In [14]: result.groupBy("component").count().sort("count", ascending = False).show(10)

+------------+-----+
|   component|count|
+------------+-----+
|           0|48860|
|         161|   66|
| 42949673000|   31|
|103079215141|   25|
| 34359738423|   19|
| 17179869446|   16|
|         231|   13|
|146028888124|    6|
| 51539607798|    5|
|618475290697|    4|
+------------+-----+
only showing top 10 rows
```

(3) What are the user ids for the cluster which has 25 users? Basically, list out all the 25 user IDs in that cluster.

```
In [18]: result.filter(result["component"] == 103079215141).show(25)

+-----+------------+
|   id|   component|
+-----+------------+
|18233|103079215141|
|18234|103079215141|
|18235|103079215141|
|18236|103079215141|
|18237|103079215141|
|18238|103079215141|
|18239|103079215141|
|18240|103079215141|
|18241|103079215141|
|18242|103079215141|
|18243|103079215141|
|18244|103079215141|
|18245|103079215141|
|18246|103079215141|
|18247|103079215141|
|18248|103079215141|
|18249|103079215141|
|18250|103079215141|
|18251|103079215141|
|18252|103079215141|
|18253|103079215141|
|18254|103079215141|
|18255|103079215141|
|18256|103079215141|
|18257|103079215141|
+-----+------------+
```

(4) Provide a list of 10 important users (User ID) in this network. Who is the most important one? Order by the "PageRank" value. Provide screenshots of this answer.

```
In [6]: PR = G.pageRank(resetProbability=0.15, maxIter=10)
        PR.vertices.orderBy("pagerank", ascending = False).show(10)

+-----+------------------+
|   id|          pagerank|
+-----+------------------+
|10164|17.932265192905074|
|15496|15.411551200953372|
|14689|13.533609243188113|
|24966|13.153795397889654|
| 7884| 12.62068681450054|
|  934| 12.22837914431175|
|45870|11.959292799812276|
|20283|11.830120761106159|
| 5148|11.739997940417085|
|46039|11.657811775224305|
+-----+------------------+
only showing top 10 rows
```

(5) By using different parameter settings for PageRank, is there any difference? This is an open question, you can try as many as you want. Provide the screenshots of your tests

By using different parameters such as '*resetProbability*', the result is different:

```
In [7]: PR = G.pageRank(resetProbability=0.3, maxIter=10)
        PR.vertices.orderBy("pagerank", ascending = False).show(10)

+-----+------------------+
|   id|          pagerank|
+-----+------------------+
|10164|20.289348477101946|
|15496| 17.35780416676058|
|14689|15.014338627419084|
|24966|14.007757357243849|
| 7884|13.421221274573075|
| 5148|13.210821740359101|
|  934|13.202398410093826|
|38123|13.086062601040028|
|45870|12.459702216067162|
|44815| 12.30306639081296|
+-----+------------------+
only showing top 10 rows
```

By using different parameters such as '*maxIter*', the result is also different:

```
In [8]: PR = G.pageRank(resetProbability=0.15, maxIter=20)
        PR.vertices.orderBy("pagerank", ascending = False).show(10)

+-----+------------------+
|   id|          pagerank|
+-----+------------------+
|10164| 18.05327786620178|
|15496|15.437961938815786|
|14689|13.521391484636682|
|24966|  13.0758131881052|
| 7884|12.501190157597295|
|  934|12.177788449130206|
|45870|11.865389614747667|
| 5148|11.731889619503333|
|20283|11.716038674581434|
|38123|11.577887045651268|
+-----+------------------+
only showing top 10 rows
```

(6) Why this user become the most important one? What are the possible reasons? This is an open question, basically, understand how PageRank works. You can also use the result from the connected component to explain it

A user become more important when they have huge number of followers compare to the number of their followings, and that their followers shall also be 'important' too. So when the user '*10164*' becomes the most important one, it must mean that this person have many important followers.

(7) Given the graph and formula below, calculate 5 ID's PageRank until convergence. (For each iteration, the values keep 2 decimals.)

Incoming edges:

- ID1: ID2
- ID2: ID3, ID5
- ID3: ID1, ID2, ID4, ID5
- ID4: ID2
- ID5: ID1, ID2

Outgoing edges L():

- L(ID1) = 2
- L(ID2) = 4
- L(ID3) = 1
- L(ID4) = 1
- L(ID5) = 2

iteration 1:

$$PR(ID_1) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.2}{4} \approx 0.07$$
$$PR(ID_2) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.2}{1} + \frac{0.2}{2}) \approx 0.29$$
$$PR(ID_3) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.2}{2} + \frac{0.2}{4} + \frac{0.2}{1} + \frac{0.2}{2}) \approx 0.41$$
$$PR(ID_4) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.2}{4} \approx 0.07$$
$$PR(ID_5) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.2}{2} + \frac{0.2}{4}) \approx 0.16$$

iteration 2:

$$PR(ID_1) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.029}{4} \approx 0.09$$
$$PR(ID_2) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.41}{1} + \frac{0.16}{2}) \approx 0.45$$
$$PR(ID_3) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.07}{2} + \frac{0.29}{4} + \frac{0.07}{1} + \frac{0.16}{2}) \approx 0.25$$
$$PR(ID_4) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.29}{4} \approx 0.09$$
$$PR(ID_5) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.07}{2} + \frac{0.29}{4}) \approx 0.12$$

iteration 3:

$$PR(ID_1) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.45}{4} \approx 0.13$$
$$PR(ID_2) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.25}{1} + \frac{0.12}{2}) \approx 0.29$$
$$PR(ID_3) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.09}{2} + \frac{0.45}{4} + \frac{0.09}{1} + \frac{0.12}{2}) \approx 0.29$$
$$PR(ID_4) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.45}{4} \approx 0.13$$
$$PR(ID_5) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.09}{2} + \frac{0.45}{4}) \approx 0.16$$

iteration 4:

$$PR(ID_1) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.29}{4} \approx 0.09$$
$$PR(ID_2) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.29}{1} + \frac{0.16}{2}) \approx 0.34$$
$$PR(ID_3) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.13}{2} + \frac{0.29}{4} + \frac{0.13}{1} + \frac{0.16}{2}) \approx 0.33$$
$$PR(ID_4) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.29}{4} \approx 0.09$$
$$PR(ID_5) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.13}{2} + \frac{0.45}{4}) \approx 0.15$$

iteration 5:

$$PR(ID_1) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.34}{4} \approx 0.1$$
$$PR(ID_2) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.33}{1} + \frac{0.15}{2}) \approx 0.37$$
$$PR(ID_3) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.09}{2} + \frac{0.34}{4} + \frac{0.09}{1} + \frac{0.15}{2}) \approx 0.28$$
$$PR(ID_4) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.34}{4} \approx 0.1$$
$$PR(ID_5) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.09}{2} + \frac{0.33}{4}) \approx 0.14$$

iteration 6:

$$PR(ID_1) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.37}{4} \approx 0.11$$
$$PR(ID_2) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.28}{1} + \frac{0.14}{2}) \approx 0.33$$
$$PR(ID_3) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.1}{2} + \frac{0.37}{4} + \frac{0.1}{1} + \frac{0.14}{2}) \approx 0.3$$
$$PR(ID_4) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.37}{4} \approx 0.11$$
$$PR(ID_5) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.1}{2} + \frac{0.28}{4}) \approx 0.13$$

iteration 7:

$$PR(ID_1) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.33}{4} \approx 0.1$$
$$PR(ID_2) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.3}{1} + \frac{0.13}{2}) \approx 0.34$$
$$PR(ID_3) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.11}{2} + \frac{0.33}{4} + \frac{0.11}{1} + \frac{0.13}{2}) \approx 0.3$$
$$PR(ID_4) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.33}{4} \approx 0.1$$
$$PR(ID_5) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.11}{2} + \frac{0.33}{4}) \approx 0.14$$

iteration 8:

$$PR(ID_1) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.34}{4} \approx 0.1$$
$$PR(ID_2) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.3}{1} + \frac{0.14}{2}) \approx 0.34$$
$$PR(ID_3) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.1}{2} + \frac{0.34}{4} + \frac{0.11}{1} + \frac{0.14}{2}) \approx 0.29$$
$$PR(ID_4) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.34}{4} \approx 0.1$$
$$PR(ID_5) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.1}{2} + \frac{0.34}{4}) \approx 0.14$$

iteration 9:

$$PR(ID_1) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.34}{4} \approx 0.1$$
$$PR(ID_2) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.29}{1} + \frac{0.14}{2}) \approx 0.34$$
$$PR(ID_3) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.1}{2} + \frac{0.34}{4} + \frac{0.11}{1} + \frac{0.14}{2}) \approx 0.29$$
$$PR(ID_4) = \frac{1-0.85}{5} + 0.85 \cdot \frac{0.34}{4} \approx 0.1$$
$$PR(ID_5) = \frac{1-0.85}{5} + 0.85 \cdot (\frac{0.1}{2} + \frac{0.34}{4}) \approx 0.14$$

Then it converges, given the final output.