b) What's the average number of connections-of-connections at google? You will be surprised to find out that even with only 3,000 employees, and just a few projects, cruises, and sports in common, the number of connections^2 is huge. So, it would take forever to iterate through all possible employees. DON'T DO THIS. Instead, just use a random sampling technique to *sample* through a fraction of employees, throw in some debugging print statements in your loop(s) to see how fast your sampling is progressing, produce a few random samplings and average over these samplings. We leave it up to you how many samples you decide to choose, and will be lenient with our grading as long as your procedure seems reasonable. **(10 points)**

c) BONUS: What's the average distance that separates any two googlers? You may want to read this (http://blogs.cornell.edu/info2040/2011/09/26/networks-in-hollywood-the-six-degrees-of-kevin-bacon/) article about Kevin Bacon. This (https://en.wikipedia.org/wiki/Erd%C5%91s%E2%80%93Bacon_number) is a funny article, too. **(5 points)**

Congratulations! A successful first day at Google! You've made both Sergei and Larry happy. You're on your way to the top of the ladder! Exhausted, you escape from your desk before any other VP of soemthing can ask anything else from you, and stop for some sushi and a little jug of warm sake. Tomorrow is new employee orientation, and you're looking forward to introduce yourself to the most connected googlers. Now *that*'s taking advantage of insider information!

Solutions: Part 1

***Note:* This solution analyzes Question 1 in full (not partitioning the datasets). See partitioned version for solutions to both Question 1 and Question 2.**

a) Read in the three excel files provided as panda dataframes. Hint: you might need to install the xlrd package to get excel support. List Ian Isla's projects, cruises, and sport clubs at Google. Find *one* google employee that is connected with Isla because he or she has been on the same project, cruise, or sport club. **(10 points)**

In [341]:
```
import pandas as pd

#gprojects are the projects googlers have worked on. Encoded as sentence
#Notes: The first column is the name of google employees. The subsequent
gprojects = pd.read_excel('gprojects.xlsx')

#gcruises are the cruises that googlers have gone on. Encoded as sentenc
#Notes: Each row has the name of the google employee in the first column
gcruises = pd.read_excel('gcruises.xlsx')

#gsports are the sport clubs that googlers belong to. Encoded as sentenc
#Notes: The first column contains googler employee name, subsequent colu
gsports = pd.read_excel('gsports.xlsx')

display(gprojects.head(), gcruises.head(), gsports.head())
display(gprojects.head(), gcruises.head(), gsports.head())
```

	0	1	2	3	4	5
Ian Isla	me opening of of	or emtheem intend melancholy	of the may color	may about may is	there lean Another the	NaN
Julian Isabel	brush rousing my truth:	no of my is	there music always-new to	that teacher em—after seems	dark no of seems	will talking the truth:
Roman Valentina	melancholy One teacher writing	person house cluster set	NaN	NaN	NaN	NaN
Diego Lyla	better cluster range at	and ground where The	have or the gray	NaN	NaN	NaN
Kaleb Lauren	em—after set poems warren	of urgency color only	poems intend an then	remembering One small one	brush is is of	of it it am

	0	1	2	3	4
Ian Isla	NaN	NaN	NaN	NaN	NaN
Julian Isabel	and me door	front on the	lean words but	NaN	NaN
Roman Valentina	me Another she	NaN	NaN	NaN	NaN
Diego Lyla	gray the boulders	gray the boulders	front on the	may of and	instead idleness to
Kaleb Lauren	your To gray	the place rooms	image her falling	then over chosen	the instead addressing

	0	1	2	3
Ian Isla	Maybe When	of of	NaN	NaN
Julian Isabel	NaN	NaN	NaN	NaN
Roman Valentina	of of	NaN	NaN	NaN
Diego Lyla	NaN	NaN	NaN	NaN
Kaleb Lauren	am door	dawn Maybe	that to	is Another

In [39]:

```
#Ian Isla's Google Cruises: None
gcruises.T.loc[:, ['Ian Isla']]
```

Out[39]:

	Ian Isla
0	NaN
1	NaN
2	NaN
3	NaN
4	NaN

In [38]:

```
#Ian Isla's Google Projects: Five
gprojects.T.loc[:,['Ian Isla']]
```

Out[38]:

	Ian Isla
0	me opening of of
1	or emtheem intend melancholy
2	of the may color
3	may about may is
4	there lean Another the
5	NaN

In [40]:

```
#Ian Isla's Google Sport Clubs: Two
gsports.T.loc[:,['Ian Isla']]
```

Out[40]:

	Ian Isla
0	Maybe When
1	of of
2	NaN
3	NaN

In [41]:

```
#Ian Isla is the first row and he shares the same first project as each
gprojects.loc[gprojects[0] == 'me opening of of']

#Ian Isla is the first row and he shares the same Sport Club as many peo
gsports.loc[gsports[0] == 'Maybe When'].head(10)
```

Out[41]:

	0	1	2	3
Ian Isla	Maybe When	of of	NaN	NaN
Angel Addison	Maybe When	NaN	NaN	NaN
Jake Arianna	Maybe When	the touch	the touch	NaN
Diego Ruby	Maybe When	am door	NaN	NaN
Bradley Serenity	Maybe When	is Another	NaN	NaN
Bradley Eleanor	Maybe When	is of	dawn Maybe	over addressing
Lucas Chloe	Maybe When	towards over	NaN	NaN
Jonathan Alana	Maybe When	gray me	from the	NaN
Isaiah Eleanor	Maybe When	NaN	NaN	NaN
Cayden Adalyn	Maybe When	that to	NaN	NaN

b) Find the total number of connections at Google. We define connections as pairs of googlers that worked on a project together, had a cruise together or participated in a sports club together. For example, if Ian Isla and Kaleb Lauren went on a cruise together, then (Ian Isla, Kaleb Lauren) represents a connection. Similarly, if Kaleb Lauren and Roman Valentina played a sport together, then (Kaleb Lauren, Roman Valentina) represents a connection. Furthermore, if Ian Isla and Kaleb Lauren worked on a project together, they have another connection as well. Thus, we have another (Ian Isla, Kaleb Lauren) connection. For this problem, it might help to assign ID's to all your google employees and then loop through projects, cruises and sports to find all possible connections. **(15 points)**

In [342]:

```
#Move the current index to a new column and reindex (to create IDs), the
#Bring the Index over to create EmployeeID where every Employee has a un
gsports['employee'] = gsports.index
gsports = gsports.reset_index(drop=True)
gsports['employeeID'] = gsports.index
gsports = gsports[['employeeID','employee', 0, 1, 2, 3]]

gcruises['employee'] = gcruises.index
gcruises = gcruises.reset_index(drop=True)
gcruises['employeeID'] = gcruises.index
gcruises = gcruises[['employeeID','employee', 0, 1, 2, 3, 4]]

gprojects['employee'] = gprojects.index
gprojects = gprojects.reset_index(drop=True)
gprojects['employeeID'] = gprojects.index
gprojects = gprojects[['employeeID','employee', 0, 1, 2, 3, 4, 5]]

#Merge files together on Employee
#google = pd.merge(gprojects, pd.merge(gcruises, gsports, on = 'employee
#google
```

In [282]:

```
#Run time for this program is overwhelmingly slow... So I have truncated
#gsports = gsports.truncate(before=0, after=100)
#gcruises = gcruises.truncate(before=0, after=100)
#gprojects = gprojects.truncate(before=0, after=100)
```

In [343]:

```
#Create a function to sort through and return True/False based on matche
def valueMatch(val, list):
    for col in list:
        if col == val:
            return True
    return False
```

In [344]:

```
r this problem, we will need to count the amount of matches in each colum
sumptions:
 EmployeeIDs are created from employee index. It has been verified that t
e exact same order in EACH dataset. Therefore, the same EmployeeID repres
 Matched employee pairs can count for more than one groups (so the same e
n count in up to three groups - Sports, Cruises or Projects)
 There is only one possible match per Employee pair per group (regardless
uises or Projects shared)

t = {} #Dictionary to hold name matches (Employee 1, Employee 2) and (Emp
tID = {} #Dictionary to hold a list of matches per Employee

orts Club Connections
rtsCount = 0
erate through all rows
 i in range(len(gsports.index)-1):
 #Compare row being iterated to the next rows
 for j in range(i+1, len(gsports.index)):
     #Transpose the first record and provide the array into the valueMatc
     #Transpose the first record and provide the array into the valueMatc
     for value in gsports.T[i].values:
         if valueMatch(value, gsports.T[j].values):
             #Add a count to the match!
             sportsCount += 1
             #We have to add the matches for both Employees, but the coun
             #Add in (i, j)
             try:
                 #If the key DOES exist, append the pair to the existing
                 dict[gsports['employee'][i]].append([gsports['employee']
                 dictID[gsports['employeeID'][i]].append(gsports['employe
             except KeyError:
                 #If a key DOES NOT exist, create it and assign the first
                 dict[gsports['employee'][i]] = [[gsports['employee'][i],
                 dictID[gsports['employeeID'][i]] = [gsports['employeeID'
             #Add in (j, i)
             try:
                 dict[gsports['employee'][j]].append([gsports['employee']
                 dictID[gsports['employeeID'][j]].append(gsports['employe
             except KeyError:
                 dict[gsports['employee'][j]] = [[gsports['employee'][j],
                 dictID[gsports['employeeID'][j]] = [gsports['employeeID'
             #If there is any match at all, these employees are a matched
             break

uise Connections (Follows same logic as above)
isesCount = 0
 i in range(len(gcruises.index)-1):
 for j in range(i+1, len(gcruises.index)):
     for value in gcruises.T[i].values:
         if valueMatch(value, gcruises.T[j].values):
             cruisesCount += 1
             try:
                 dict[gcruises['employee'][i]].append([gcruises['employee
                 dictID[gcruises['employeeID'][i]].append(gcruises['emplo
             except KeyError:
                 dict[gcruises['employee'][i]] = [[gcruises['employee'][i
                 dictID[gcruises['employeeID'][i]] = [gcruises['employeeI
             try:
                 dict[gcruises['employee'][j]].append([gcruises['employee
                 dictID[gcruises['employeeID'][j]].append(gcruises['emplo
             except KeyError:
                 dict[gcruises['employee'][j]] = [[gcruises['employee'][j
                 dictID[gcruises['employeeID'][j]] = [gcruises['employeeI
             break

oject Connections (Follows same logic as above)
jectsCount = 0
 i in range(len(gprojects.index)-1):
 for j in range(i+1, len(gprojects.index)):
     for value in gprojects.T[i].values:
         if valueMatch(value, gprojects.T[j].values):
             projectsCount += 1
             try:
                 dict[gprojects['employee'][i]].append([gprojects['employ
                 dict[gprojects['employee'][i]].append([gprojects['employ
                 dictID[gprojects['employeeID'][i]].append(gprojects['emp
             except KeyError:
                 dict[gprojects['employee'][i]] = [[gprojects['employee']
                 dictID[gprojects['employeeID'][i]] = [gprojects['employe
             try:
                 dict[gprojects['employee'][j]].append([gprojects['employ
                 dictID[gprojects['employeeID'][j]].append(gprojects['emp
             except KeyError:
                 dict[gprojects['employee'][j]] = [[gprojects['employee']
                 dictID[gprojects['employeeID'][j]] = [gprojects['employe
             break

nt("Count of Connections from Sports Clubs: "+str(sportsCount))
nt("Count of Connections from Cruises: "+str(cruisesCount))
nt("Count of Connections from Projects: "+str(projectsCount))
```

```
Count of Connections from Sports Clubs: 743092
Count of Connections from Cruises: 140781
Count of Connections from Projects: 84156
```

c) Create a connections dictionary where for each googler, we store a list of all connections they participate in. Using this dictionary, determine the average number of connections per googler. **(15 points)**

In [108]:

```
#Dictionary created as a part of Part B
dict
```

Out[108]:

```
{'Ian Isla': [['Ian Isla', 'Roman Valentina'],
  ['Ian Isla', 'Christian Alaina'],
  ['Ian Isla', 'Emmett Aurora'],
  ['Ian Isla', 'Angel Addison'],
  ['Ian Isla', 'Jake Arianna'],
  ['Ian Isla', 'George Bella'],
  ['Ian Isla', 'Hudson Melanie'],
  ['Ian Isla', 'Emmett Riley'],
  ['Ian Isla', 'Wesley Athena'],
  ['Ian Isla', 'Ashton Quinn'],
  ['Ian Isla', 'Luca Adriana'],
  ['Ian Isla', 'Aiden Sarah']],
 'Roman Valentina': [['Roman Valentina', 'Emmett Aurora'],
  ['Roman Valentina', 'Hudson Melanie'],
  ['Roman Valentina', 'Emmett Riley'],
  ['Roman Valentina', 'Wesley Athena'],
  ['Roman Valentina', 'Ashton Quinn'],
  ['Roman Valentina', 'Timothy Mckenzie']],
 'Kaleb Lauren': [['Kaleb Lauren', 'Bryce Isla'],
```

In [345]:

```
total = sportsCount + cruisesCount + projectsCount
count = len(dict.keys())
average = total/count
print("Average Connections: "+str(average))
```

```
Average Connections: 336.82289491997216
```

d) Make use of this dictionary to find the most connected googler. What about the 10 *most* connected googlers? What about the 10 *least* connected googlers? What's the total number of connections, and the average number of connections at Google? Note that connections only counts number of projects, cruises, or sports in common. Who one is connected to makes no difference. A connection to Sergey Brin himself counts the same as a connection to a student co-op. **(10 points)**

In [346]:

```
print("Total Connections: "+str(total))
print("Average Connections: "+str(average))
```

```
Total Connections: 968029
Average Connections: 336.82289491997216
```

In [347]:

```
#Use bubble sort from our lectures, but modify it to work with a list of
#we can order key-value pairs from greatest to least
def bubble_sort(alist):
    for passnum in range(len(alist)-1, -1, -1):
        swapped = False
        for i in range(passnum):
            if alist[i][1] < alist[i+1][1]:
                alist[i], alist[i+1] = alist[i+1], alist[i]
                swapped = True
        if not swapped:
            break
```

In [*]:

```
#Pass the key AND # of contents into a list of tuples
sorted_list = []
for key, value in sorted(dict.items()):
    sorted_list.append((key, len([item for item in value if item])))
bubble_sort(sorted_list)
```

```
In [*]:  #Top Ten Connected Googlers!
         sorted_list[:10]
```

```
Out[349]: [('Jason Josephine', 1597),
           ('Jeremy Leilani', 1529),
           ('Joel Ava', 1527),
           ('Brandon Melanie', 1517),
           ('Bryce Michelle', 1511),
           ('Roman Jennifer', 1494),
           ('Nicolas Sofia', 1467),
           ('Chase Clara', 1464),
           ('Max Madelyn', 1462),
           ('Dominic Kennedy', 1461)]
```

```
In [*]:  #Bottom Ten Connected Googlers!
         sorted_list[-10:]
```

```
Out[350]: [('Joshua Khloe', 17),
           ('Maximus Camila', 16),
           ('Micah Lillian', 14),
           ('Weston Daisy', 14),
           ('Kaden Mila', 13),
           ('Luke Avery', 11),
           ('Brantley Valerie', 9),
           ('Victor Sophia', 7),
           ('Brady Hailey', 6),
           ('Everett Katelyn', 4)]
```

Now, what if having important connections is of significance? In other words, being connected to Sergey Brin makes a huge difference: if a googler acquaintance you are connected to is *very* connected, that automatically makes you very ***central*** to google. To answer these questions, we need our old friend PageRank.

e) One possible way to tap into PageRank is to use the now-familiar package `networkx` . We want you to create a graph from our connections dictionary. Hint: There are many ways of doing this. One possible way involves hooking into the networkx method `parse_adjlist` that takes in an adjacency list(connections). Look up documentation on this function to learn more. **(10 points)**

In [336]:

```
#Dictionary of EmployeeIDs created as a part of Part B
#Note: This is due to parse_adjust needing numbers to split nodes/edges
#Names ('First Last') would be two entities 'First' as a node and 'Last'
print(dictID)
```

```
43, 46, 50, 59, 67, 74, 78, 86, 94, 96], 59: [4, 54, 20, 43, 54, 14, 3
4, 52], 60: [4, 5, 7, 8, 10, 13, 14, 16, 19, 20, 24, 25, 31, 36, 38, 3
9, 40, 41, 48, 50, 52, 55, 61, 65, 66, 67, 69, 71, 75, 84, 86, 87, 90,
93, 95, 100, 12, 49, 76, 80, 91, 73], 70: [4, 5, 10, 11, 12, 17, 21, 2
3, 30, 39, 46, 48, 50, 58, 62, 92, 22, 64, 77, 0], 71: [4, 5, 7, 8, 10
, 14, 16, 19, 20, 24, 25, 29, 31, 34, 38, 39, 40, 48, 50, 52, 55, 60,
65, 67, 69, 75, 84, 86, 87, 90, 95, 100, 4, 37, 38, 51, 55, 14], 85: [
4, 97, 37, 63, 75, 76, 9, 77], 86: [4, 5, 10, 16, 19, 25, 38, 39, 52,
60, 65, 71, 87, 90, 95, 100, 3, 11, 13, 18, 31, 33, 54, 74, 78, 11], 8
7: [4, 5, 7, 8, 10, 11, 16, 19, 25, 27, 28, 30, 36, 38, 39, 52, 58, 60
, 62, 65, 66, 71, 79, 83, 84, 86, 90, 92, 93, 95, 100], 90: [4, 5, 10,
16, 19, 25, 38, 39, 52, 60, 65, 71, 86, 87, 95, 100, 40, 43], 92: [4,
5, 8, 9, 10, 11, 12, 14, 18, 25, 27, 30, 36, 39, 58, 62, 66, 67, 69, 7
0, 78, 79, 83, 87, 88, 93, 27, 31], 95: [4, 5, 10, 16, 19, 25, 38, 39,
52, 60, 65, 71, 86, 87, 90, 100, 1, 3, 11, 31, 96, 61], 97: [4, 85, 21
, 23, 25, 34], 100: [4, 5, 10, 16, 19, 25, 38, 39, 52, 60, 65, 71, 86,
87, 90, 95, 22, 43, 63, 67, 98, 99, 47, 88], 40: [5, 7, 8, 12, 14, 16,
20, 24, 25, 30, 31, 42, 48, 50, 51, 52, 54, 55, 56, 60, 61, 67, 69, 71
, 84, 46, 65, 77, 90, 15, 19, 81], 42: [5, 6, 12, 21, 23, 24, 25, 30,
```

In [*]:

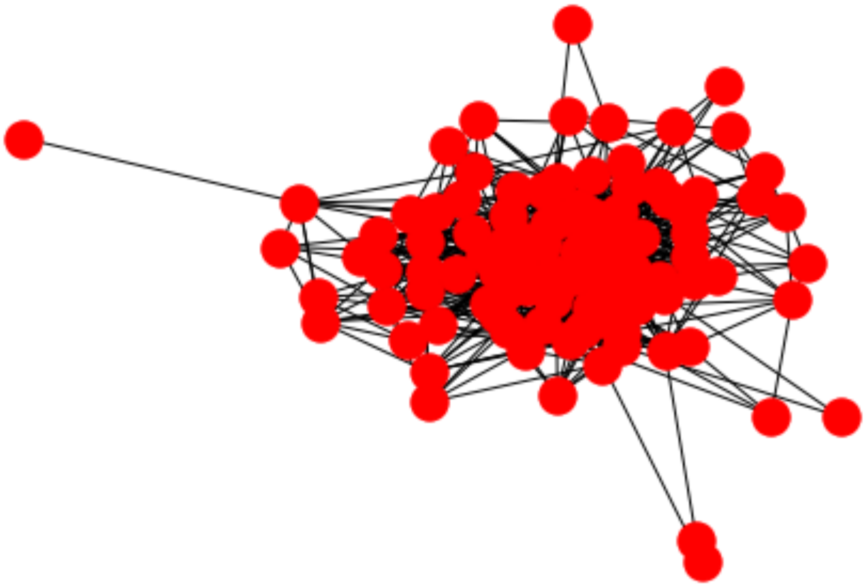
```
import networkx as nx

#Append the employeeIDs to create a list of strings (e.g. ['1 2 3', '2 2
test = []
for key, value in sorted(dictID.items()):
    test.append(str(key)+" "+" ".join([str(item) for item in value if it

#https://networkx.github.io/documentation/networkx-1.10/reference/genera
ggraph = nx.parse_adjlist(test, nodetype=str)
```

In [303]:

```
#Graph of Connections Drawn Out!
nx.draw(ggraph)
```



f) Use the template from the midterm and previous assignments, or just use networkx's `pagerank()` API that professor mentionned in class to calculate pageRank for google employees. Using this information, find the 10 most central googlers, and the 10 least central googlers. Are they the same as the 10 most and 10 least connected googlers? **(10 points)**

In [*]:

```
import numpy as np

#Apply PageRank to the graph of Googlers
ggraph_pr = nx.pagerank(ggraph) #Note: Default damping is 0.85!

#Convert the dictionary to a tuple and sort
ggraph_ranked = [(k, v) for k, v in ggraph_pr.items()]
bubble_sort(ggraph_ranked)
```

In [*]:

```
#Redo Part D to sort with EmployeeID rather than Employee names
empID_connections = []
for key, value in sorted(dictID.items()):
    empID_connections.append((key, len([item for item in value if item])
bubble_sort(empID_connections)
```

In [*]:
```
#Top Ten Googlers by PageRank! Note that only EmployeeID is displayed -
ggraph_ranked[:10]
```

Out[354]:
```
[('2377', 0.0006995799063534159),
 ('2544', 0.0006852014311075433),
 ('50', 0.0006843891733069788),
 ('994', 0.0006816870649398268),
 ('1756', 0.0006739167478446553),
 ('500', 0.0006730812847642394),
 ('1737', 0.0006706125276646455),
 ('1911', 0.0006696527517649552),
 ('1680', 0.0006692185427543316),
 ('2570', 0.0006676229650879282)]
```

In [*]:
```
#Top Ten Connected Googlers (but by EmployeeID, # of Connections)
empID_connections[:10]
```

Out[355]:
```
[(2377, 1597),
 (1756, 1529),
 (50, 1527),
 (2544, 1517),
 (994, 1511),
 (500, 1494),
 (2278, 1467),
 (1680, 1464),
 (341, 1462),
 (1911, 1461)]
```

In [*]:
```
#Bottom Ten Googlers by PageRank!
ggraph_ranked[-10:]
```

Out[356]:
```
[('2749', 6.101756721293558e-05),
 ('2041', 6.027257132876336e-05),
 ('2500', 5.986731195891479e-05),
 ('2273', 5.958149576583753e-05),
 ('1330', 5.89903753238994e-05),
 ('1790', 5.794241196596471e-05),
 ('82', 5.732606732746668e-05),
 ('526', 5.606907957757992e-05),
 ('1998', 5.544951783856664e-05),
 ('135', 5.418699300202199e-05)]
```

In [*]:

```
#Bottom Ten Connected Googlers (but by EmployeeID, # of Connections)
empID_connections[-10:]
```

Out[357]:

```
[(925, 17),
 (2273, 16),
 (1330, 14),
 (2041, 14),
 (2500, 13),
 (1790, 11),
 (82, 9),
 (526, 7),
 (1998, 6),
 (135, 4)]
```

Note: We can see that the top and bottom pagerank and connected employee list are different (as we have noticed in class before) in that the top members of pagerank are not the same as the top members with the most connections. Those with connections indirectly others with MORE connections often increases the pagerank more - although there is much overlap between the top ten groups. Given the analogy of Sergey Brin previously - a connection to Sergey Brin at this point raises a person's pagerank more than a connection to a co-op student (assuming the co-op student has very few connections in turn) so it is more of a measure of importance!

g) BONUS: Can you plot google's connectome? That connectome should easily reveal the least central googlers, since they represent isolated nodes, far away from network traffic. You have some latitude on how you choose to do this. Can you do something to upend that graph to also easily reveal the most central googlers? **(5 points)**

In [ ]:

```

```

Solutions: Part 2

a) Write a list comprehension (ideally) or some type of loop to determine connections of connections. Essentially, you should already have a list of connections, but now search the connections of a specific googler to find connections of connections. For example, if (Ian Isla, Kaleb Lauren) were connected in Part 1, and (Kalen Lauren, Roman Valentina) were also connected, then Roman Valentina would be a "connection of a connection" of Ian Isla. While you do this, perform a check to ensure that the connection of a connection should not already be a connection itself. **(20 points)**