Question 1

Convert text file to Adjacency Matrix and Edge List

In [30]:
```python
f = open("US_AIR_DATA.txt")
```

In [2]:
```python
list_ =  f.readlines()
```

In [3]:
```python
u = []
v = []

for i in list_:
    getArray = i.split("\t")

    getU = getArray[0]
    getV = getArray[1][:-1]

    u.append(int(getU)-1)
    v.append(int(getV)-1)
```

In [4]:
```python
nodes = max(max(u),max(v)) + 1

print('The total number of Nodes are : ',nodes)
print('The Number of edges are : ',len(u))

# AdjacencyMatrix is created.
m = nodes
n = nodes
adjacencyMatrix = [[0 for x in range(m)] for y in range(n) ]

for i in range(len(u)):
    adjacencyMatrix[u[i]][v[i]] = 1
    adjacencyMatrix[v[i]][u[i]] = 1
```

```
The total number of Nodes are :  332
The Number of edges are :  2126
```

In [5]:
```python
count = 0
for i in range(m):
    for j in range(n):
        if adjacencyMatrix[i][j] == 1:
            count = count + 1

print('Number of 1''s present are : ',count)
```

```
Number of 1s present are :  4252
```

In [6]:
```python
#Creating an Edge List
#Edge List is created as an Dictionary where key is the Node and Value is a list consists of all the neighbours nodes

edgeList = {}

for i in range(nodes):
    getList = []
    # j is for Columns
    for j in range(nodes):
        if adjacencyMatrix[i][j] == 1 :
            getList.append(j)
    edgeList[i] = getList

print('Edge List of the given data is : \n',edgeList)
```

```
Edge List of the given data is :
 {0: [1, 3, 7], 1: [0, 3, 7], 2: [4, 7], 3: [0, 1, 7, 25, 46], 4: [2, 7], 5: [6, 7, 12], 6: [5, 7, 12], 7: [0, 1, 2, 3, 4, 5, 6, 12, 15, 22, 23, 25, 26, 27, 29, 33, 34, 35, 36, 37, 46, ...
[illegible]
... 326], 330: [247, 312], 331: [326]}
```

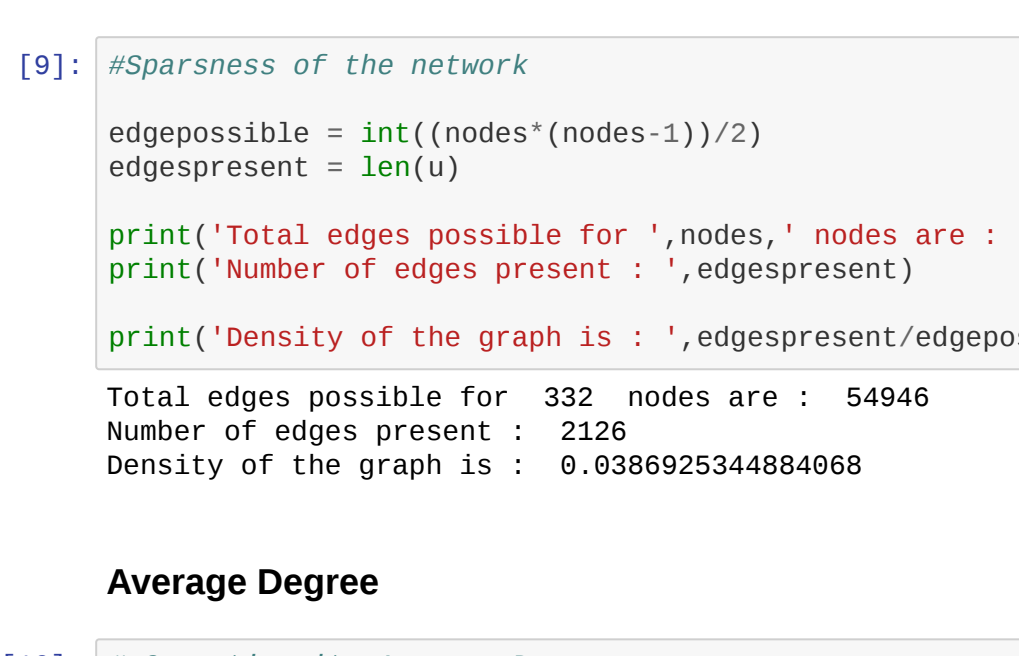
Visualizing the network

Note : This is also done in Cytoscape. Cytoscape visualization is in report file

In [7]:
```python
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
```

In [8]:
```python
G = nx.from_numpy_matrix(np.array(adjacencyMatrix))
nx.draw(G, with_labels=True)
```

```
C:\Users\Lakshmi Praffulla\Anaconda3\lib\site-packages\networkx\drawing\nx_pylab.py:579: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if not cb.iterable(width):
```



Sparseness of the network

In [9]:
```python
#Sparseness of the network

edgepossible = int((nodes*(nodes-1))/2)
edgespresent = len(u)

print('Total edges possible for ',nodes,' nodes are : ',edgepossible )
print('Number of edges present : ',edgespresent)

print('Density of the graph is : ',edgespresent/edgepossible)
```

```
Total edges possible for  332  nodes are :  54946
Number of edges present :  2126
Density of the graph is :  0.03869253448844068
```

Average Degree

In [10]:
```python
# Computing its Average Degree

nodeDegree = {}

for i in edgeList:
    nodeDegree[i] = len(edgeList[i])

print('Degree of each node is : ')
print(nodeDegree)

sumOfDegree = 0

for i in nodeDegree:
    sumOfDegree += nodeDegree[i]

print()
print('Average Degree of the given graph is : ',sumOfDegree/nodes)
```

```
Degree of each node is :
{0: 3, 1: 3, 2: 2, 3: 5, 4: 2, 5: 3, 6: 3, 7: 29, 8: 4, 9: 4, 10: 3, 11: 3, 12: 14, 13: 1, ...
[illegible]
... 326: 4, 327: 2, 328: 3, 329: 2, 330: 2, 331: 1}

Average Degree of the given graph is :  12.80722891566265
```

'scaled degree distribution'

In [11]:
```python
# Plot Scaled Degree Distribution

maxDegree = 0

for i in nodeDegree:
    if maxDegree < nodeDegree[i]:
        maxDegree = nodeDegree[i]

degreeDistribution = {}

for i in range(maxDegree+1):
    if i == 0:
        continue;
    degreeDistribution[i] = []

for i in nodeDegree:
    degreeDistribution[nodeDegree[i]].append(i)

print('Degree Distribution for each node is : ',degreeDistribution)
```

```
Degree Distribution for each node is :  {1: [13, 14, 17, 20, 24, 26, 41, 42, 43, 51, 87, 88, ...
[illegible]
... 137: [], 138: [], 139: [117]}
```

In [12]:
```python
#Plotting the graphs for different distributions

def plotGraph(xValues,yValues,xLabel,yLabel,graphTitle,defaultSize='n'):
    if defaultSize=='n':
        f,ax=plt.subplots(figsize=(20,5))
        ax.legend(fontsize = 14)

    plt.bar(xValues,yValues,color='green')
    plt.xlabel(xLabel)
    plt.ylabel(yLabel)
    plt.title(graphTitle)
    plt.show()
```

In [14]:
```python
max_degree = 0

for i in degreeDistribution:
    if max_degree < len(degreeDistribution[i]):
        max_degree = len(degreeDistribution[i])
```

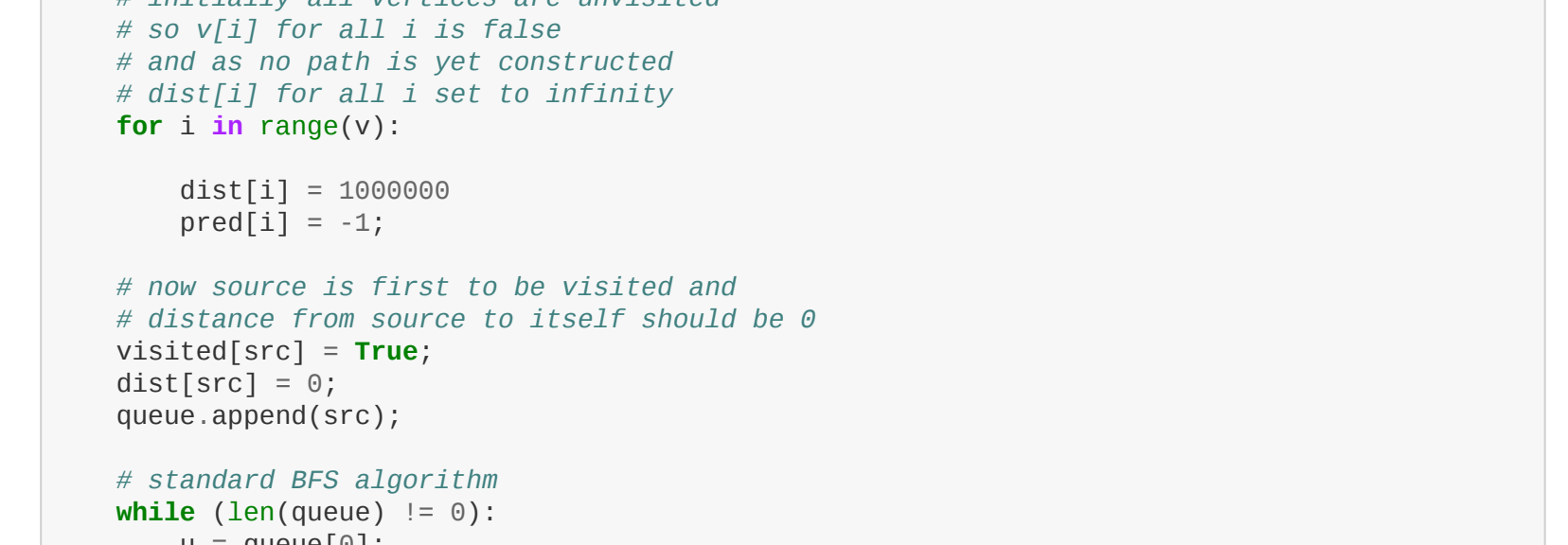
In [17]:
```python
# Graph Plot

Xval = []
Yval = []

for i in degreeDistribution:
    Xval.append(i)
    Yval.append(len(degreeDistribution[i])/max_degree)

plotGraph(Xval,Yval,'degree (k)','pk','Scaled degree distribution')
```

```
No handles with labels found to put in legend.
```



Average Path Length , Diameter and Average Clustering Coefficient.

In [18]:
```python
# Average Path Length = Average of the shortest path between all the pair of nodes
edgeListForBFS = []

for i in edgeList:
    edgeListForBFS.append(edgeList[i])
```

In [19]:
```python
def BFS(adj, src, dest, v, pred, dist):

    # a queue to maintain queue of vertices whose
    # adjacency list is to be scanned as per normal
    # DFS algorithm
    queue = []

    # boolean array visited[] which stores the
    # information whether ith vertex is reached
    # at least once in the Breadth first search
    visited = [False for i in range(v)];

    # initially all vertices are unvisited
    # so v[i] for all i is false
    # and as no path is yet constructed
    # dist[i] for all i set to infinity
    for i in range(v):

        dist[i] = 1000000
        pred[i] = -1;

    # now source is first to be visited and
    # distance from source to itself should be 0
    visited[src] = True;
    dist[src] = 0;
    queue.append(src);

    # standard BFS algorithm
    while (len(queue) != 0):
        u = queue[0];
        queue.pop(0);
        for i in range(len(adj[u])):

            if (visited[adj[u][i]] == False):
                visited[adj[u][i]] = True;
                dist[adj[u][i]] = dist[u] + 1;
                pred[adj[u][i]] = u;
                queue.append(adj[u][i]);

                # We stop BFS when we find
                # destination.
                if (adj[u][i] == dest):
                    return True;

    return False;
```

In [21]:
```python
def getShortestDistance(adj, s, dest, v):

    # predecessor[i] array stores predecessor of
    # i and distance array stores distance of i
    # from s
    pred=[0 for i in range(v)]
    dist=[0 for i in range(v)];

    if (BFS(adj, s, dest, v, pred, dist) == False):
        print("Given source and destination are not connected")

    # vector path stores the shortest path
    path = []
    crawl = dest;
    crawl = dest;
    path.append(crawl);

    while (pred[crawl] != -1):
        path.append(pred[crawl]);
        crawl = pred[crawl];

    return dist[dest]
```

In [22]:
```python
ShortestPathBetweenNodes = {}
for i in range(len(edgeList)):
    for j in range(i+1,len(edgeList)):
        ShortestPathBetweenNodes[(i,j)] = getShortestDistance(edgeListForBFS,i,j,nodes)
```

In [31]:
```python
# ShortestPathBetweenNodes
```

In [24]:
```python
#Average Path Length

ShortestPathSum = 0

for i in ShortestPathBetweenNodes:
    ShortestPathSum += ShortestPathBetweenNodes[i]

print('Average Shortest Path is ',ShortestPathSum/len(ShortestPathBetweenNodes))
```

```
Average Shortest Path is  2.7421104357005506
```

In [26]:
```python
#Diameter - It is the longest shortest path between 2 nodes

max_ = 0
for i in ShortestPathBetweenNodes:
    if max_ < ShortestPathBetweenNodes[i]:
        max_ = ShortestPathBetweenNodes[i]

print('Longest Shortest Path is : ',max_)
print('Diameter nodes are : ')

for i in ShortestPathBetweenNodes:
    if ShortestPathBetweenNodes[i] == max_:
        print(i)
```

```
Longest Shortest Path is :  6
Diameter nodes are :
(8, 329)
(8, 331)
(9, 329)
(9, 331)
(10, 329)
(10, 331)
(11, 329)
(11, 331)
(13, 329)
(13, 331)
(14, 329)
(14, 331)
(16, 329)
(16, 331)
(17, 329)
(17, 331)
(18, 329)
(18, 331)
(19, 329)
(19, 331)
(20, 329)
(20, 331)
(21, 331)
(24, 331)
(39, 331)
(54, 331)
(60, 331)
(63, 331)
(65, 331)
(86, 331)
(88, 329)
(88, 331)
(96, 331)
(97, 331)
(102, 331)
(103, 331)
(109, 331)
(112, 331)
(114, 331)
(115, 331)
(116, 331)
(120, 331)
(121, 329)
(121, 331)
(134, 329)
(134, 331)
(140, 331)
(155, 331)
(169, 331)
(170, 329)
(170, 331)
(186, 331)
(187, 329)
(187, 331)
(189, 331)
(193, 329)
(193, 331)
(199, 329)
(199, 331)
(209, 331)
(226, 331)
(234, 331)
(237, 331)
(242, 331)
(251, 331)
(261, 331)
(284, 331)
(301, 331)
(319, 331)
(322, 329)
(322, 331)
(323, 331)
(324, 331)
```

In [27]:
```python
#Clustering Coefficient

clusterCoef = {}

for k in edgeList:
    neighbours = edgeList[k]
    neighbLength = len(neighbours)
    denominator = (neighbLength*(neighbLength-1))/2

    numerator = 0
    for i in range(neighbLength):
        for jj in range(i+1,neighbLength):
            if adjacencyMatrix[neighbours[i]][neighbours[jj]] == 1:
                numerator += 1

    if denominator == 0.0:
        clusterCoef[k] = 0
    else:
        clusterCoef[k] = numerator/denominator

print('Clustering Coefficient of each nodes are : ',clusterCoef)
print('\n')


sumCC = 0
for i in clusterCoef:
    sumCC += clusterCoef[i]

print('Average Clustering Coefficient is ',sumCC/nodes)
```

```
Clustering Coefficient of each nodes are :  {0: 1.0, 1: 1.0, 2: 1.0, 3: 0.6, 4: 1.0, 5: 1.0, 6: 1.0, 7: 0.120689655172413 8, 8: 0.8333333333333334, 9: 0.8333333333333334, 10: 1.0, 11: 1.0, ...
[illegible]
```

Average clustering coefficient is: 0.017826285578653

In [ ]:

In [ ]: