hw1

September 13, 2017

1 Part 3 Coding: Shortest Path

2 Q8

```
In [1]: import numpy as np
        import random
        import matplotlib.pyplot as plt
        import networkx as nx
```

2.1 Q8a

A procedure that produces a graph (represented by a matrix) of N nodes, each pair connected with probability p input: number of nodes N, probability p output: a graph represented by a matrix, start node, end node

```
In [2]: def generate_graph(N, p):
            graph = (np.random.rand(N, N) < p).astype(int)
            # make the graph undirected/ matrix symmetric
            i_lower = np.tril_indices(N, -1)
            graph[i_lower] = graph.T[i_lower]
            np.fill_diagonal(graph, 0)
            return graph
```

```
In [3]: graph = generate_graph(10, 0.5)
        print (graph)

        # Print AdjacencyList representation of graph
        def converGraph(graph, N):
            result = [[] for i in range(N)]
            for i in range(N):
                for j in range(i, N):
                    if graph[i,j] == 1:
                        result[i].append(j)
                        result[j].append(i)
            return result

        print (converGraph(graph, 10))
```

```
[[0 1 0 1 1 1 0 0 1 1]
 [1 0 1 1 0 1 1 0 0 1]
 [0 1 0 1 1 0 0 0 1 0]
 [1 1 1 0 0 1 0 0 1 0]
 [1 0 1 0 0 1 1 1 1 0]
 [1 1 0 1 1 0 0 1 0 1]
 [0 1 0 0 1 0 0 1 0 1]
 [0 0 0 0 1 1 1 0 1 1]
 [1 0 1 1 1 0 0 1 0 1]
 [1 1 0 0 0 1 1 1 1 0]]
[[1, 3, 4, 5, 8, 9], [0, 2, 3, 5, 6, 9], [1, 3, 4, 8], [0, 1, 2, 5, 8], [0, 2, 5, 6, 7, 8], [0,
```

2.2 Q8b

Generalized Shortest Path Algorithm: BFS (Breadth First Search) Input: a graph represented by a matrix, start node, end node Output: list of nodes leading from start node to end node

```
In [4]: def bfs(graph, start, end):
            # maintain a queue of paths
            queue = []
            explored = []
            # push the first path into the queue
            queue.append([start])
            while queue:
                # get the first path from the queue
                path = queue.pop(0)
                node = path[-1]
                if node not in explored:
                    explored.append(node)
                    if node == end:
                        return path
                    for adjacent in graph[node]:
                        new_path = list(path)
                        new_path.append(adjacent)
                        queue.append(new_path)
```

2.3 Q8c

Generate a graph w/ p = 0.1; Pick any two nodes and compute their shortest dist 10000 times; 100 sample output is printed; Average distance is 1.9

```
In [154]: N = 1000
          p = 0.1
          maxIter = 10000
          total = 0;
          pCount = 100;
          graph = generate_graph(N, p)
```

```
        myGraph = converGraph(graph, N)
        numDC = 0;
        for i in range(maxIter):
            start,end = random.sample(range(N), 2)
            path = bfs(myGraph, start, end)
            if (path == "infinity"):
                dist = sys.maxint
                numDC = numDC + 1
            else:
                dist = len(path)-1
                total = total + dist
            if pCount > 0:
                print ("(node A " + repr(start) + ", node B "+ repr(end) + "): path length " +
                pCount = pCount - 1

        print ("Average distance is " + repr(float(total/(maxIter-numDC))))
```

```
(node A 988, node B 882): path length 2
(node A 753, node B 30): path length 2
(node A 792, node B 288): path length 2
(node A 990, node B 473): path length 2
(node A 78, node B 46): path length 2
(node A 401, node B 500): path length 1
(node A 978, node B 210): path length 2
(node A 379, node B 780): path length 2
(node A 391, node B 267): path length 2
(node A 30, node B 919): path length 2
(node A 673, node B 342): path length 2
(node A 516, node B 162): path length 2
(node A 336, node B 163): path length 2
(node A 803, node B 59): path length 2
(node A 436, node B 991): path length 2
(node A 565, node B 784): path length 2
(node A 641, node B 216): path length 2
(node A 621, node B 708): path length 2
(node A 450, node B 615): path length 2
(node A 121, node B 970): path length 2
(node A 416, node B 661): path length 2
(node A 114, node B 197): path length 2
(node A 137, node B 200): path length 2
(node A 452, node B 319): path length 2
(node A 777, node B 835): path length 2
(node A 748, node B 879): path length 2
(node A 408, node B 786): path length 2
(node A 196, node B 558): path length 1
(node A 80, node B 821): path length 2
(node A 651, node B 573): path length 1
(node A 625, node B 108): path length 2
```

```
(node A 59, node B 775): path length 2
(node A 190, node B 186): path length 2
(node A 334, node B 520): path length 1
(node A 520, node B 387): path length 1
(node A 435, node B 6): path length 2
(node A 425, node B 718): path length 2
(node A 36, node B 816): path length 2
(node A 986, node B 553): path length 2
(node A 360, node B 842): path length 2
(node A 952, node B 25): path length 2
(node A 150, node B 488): path length 2
(node A 701, node B 841): path length 2
(node A 106, node B 28): path length 2
(node A 991, node B 725): path length 2
(node A 521, node B 285): path length 2
(node A 790, node B 932): path length 2
(node A 683, node B 791): path length 2
(node A 79, node B 247): path length 2
(node A 719, node B 42): path length 2
(node A 279, node B 103): path length 1
(node A 194, node B 86): path length 1
(node A 620, node B 449): path length 2
(node A 668, node B 993): path length 2
(node A 586, node B 35): path length 2
(node A 645, node B 849): path length 2
(node A 653, node B 694): path length 2
(node A 359, node B 795): path length 2
(node A 883, node B 990): path length 2
(node A 256, node B 968): path length 2
(node A 215, node B 934): path length 2
(node A 281, node B 478): path length 2
(node A 160, node B 643): path length 2
(node A 983, node B 135): path length 2
(node A 211, node B 944): path length 2
(node A 189, node B 884): path length 2
(node A 691, node B 295): path length 2
(node A 883, node B 101): path length 2
(node A 27, node B 198): path length 2
(node A 219, node B 381): path length 1
(node A 606, node B 479): path length 2
(node A 418, node B 104): path length 2
(node A 834, node B 371): path length 2
(node A 82, node B 434): path length 2
(node A 450, node B 892): path length 2
(node A 213, node B 113): path length 2
(node A 34, node B 22): path length 2
(node A 645, node B 763): path length 2
(node A 175, node B 357): path length 2
```

```
(node A 438, node B 797): path length 2
(node A 82, node B 928): path length 2
(node A 396, node B 338): path length 2
(node A 611, node B 283): path length 2
(node A 521, node B 688): path length 2
(node A 818, node B 600): path length 2
(node A 215, node B 603): path length 2
(node A 872, node B 212): path length 2
(node A 911, node B 527): path length 2
(node A 370, node B 238): path length 2
(node A 980, node B 112): path length 2
(node A 129, node B 787): path length 2
(node A 100, node B 448): path length 2
(node A 965, node B 827): path length 2
(node A 434, node B 209): path length 2
(node A 882, node B 899): path length 2
(node A 160, node B 23): path length 2
(node A 54, node B 92): path length 2
(node A 62, node B 838): path length 1
(node A 995, node B 2): path length 2
(node A 450, node B 830): path length 2
Average distance is 1.9027
```

2.4 Q8d

Run the shortest-path algorithm on data sets constructed with many values of p Numerical data: [3.244, 2.637, 2.388, 2.14, 2.031, 1.901, 1.852, 1.801, 1.752, 1.689, 1.634, 1.585, 1.544, 1.502] for p values: [0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.15000000000000002, 0.2, 0.25, 0.3, 0.35000000000000003, 0.4, 0.45, 0.5] Relation plotted below

```
In [5]: def avgDist(N, p, maxIter):
            total = 0;
            graph = generate_graph(N, p)
            myGraph = converGraph(graph, N)
            numDC = 0;
            for i in range(maxIter):
                start,end = random.sample(range(N), 2)
                path = bfs(myGraph, start, end)
                if (path == "infinity"):
                    numDC = numDC + 1
                else:
                    dist = len(path) - 1
                total = total + dist
            return float(total/(maxIter-numDC))

In [168]: p = [0.01, 0.02, 0.03, 0.04]
          for i in range(10):
```

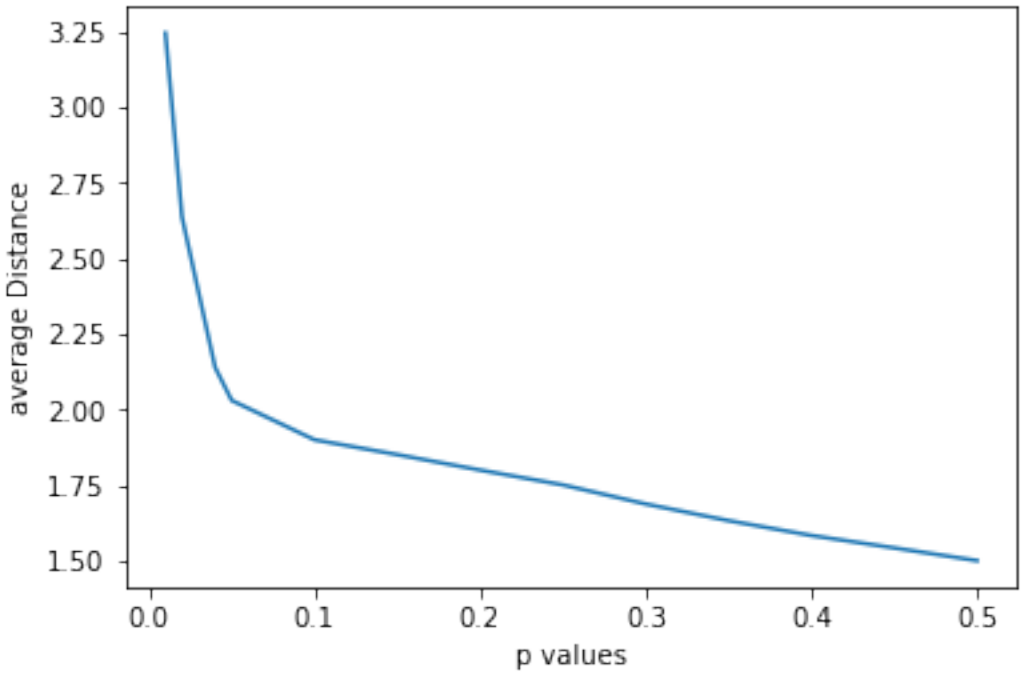
```
            p.append(0.05+0.05*i)
        distList = []
        for pvalue in p:
            distList.append(avgDist(1000, pvalue, 1000))
        print (distList)
```

```
[3.244, 2.637, 2.388, 2.14, 2.031, 1.901, 1.852, 1.801, 1.752, 1.689, 1.634, 1.585, 1.544, 1.502
```

```
In [169]: print (p)
```

```
[0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.15000000000000002, 0.2, 0.25, 0.3, 0.35000000000000003, 0.
```

```
In [170]: plt.plot(p, distList)
          plt.xlabel("p values")
          plt.ylabel("average Distance")
          plt.show()
```



2.5 Q8e

As p increases, the average distance decreases. Intuitively, the more likely two nodes are connected, the shorter their shortest distance becomes. As p gets closer to 1, the shortest distance should converge to 1 as well because two nodes are highly like to be connected directly. The asymptotic line is "avg Dist = 1".

3 Q9

Load data and convert graph into adjacency list representation

```
In [136]: # Q9
          # Load fb data
          fb =  np.loadtxt(fname = 'facebook_combined.txt', delimiter = ' ', dtype = 'int')
```

```
In [141]: # Convert fb data to graph
          N = 4039
          fb_graph = np.zeros((N, N))
          for i in range(fb.shape[0]):
              fb_graph[fb[i,0], fb[i,1]] = 1
              fb_graph[fb[i,1], fb[i,0]] = 1
          print (fb_graph)
```

```
[[ 0.  1.  1. ...,  0.  0.  0.]
 [ 1.  0.  0. ...,  0.  0.  0.]
 [ 1.  0.  0. ...,  0.  0.  0.]
 ...,
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]]
```

```
In [147]: myfb = converGraph(fb_graph, N)
          print (bfs(myfb, 0, 1))
```

```
[0, 1]
```

3.1 Q9a

Below is the code for simulating Q8c on facebook graph; 100 sample results are printed; Average distance is 3.64

```
In [157]: # Q9a
          N = 4039
          maxIter = 100
          total = 0;
          pCount = 100;
          numDC = 0;
          myfb = converGraph(fb_graph, N)
          for i in range(maxIter):
              start,end = random.sample(range(N), 2)
              path = bfs(myfb, start, end)
              if (path == "infinity"):
                  dist = sys.maxint
                  numDC = numDC + 1
```

```
            else:
                dist = len(path)-1
                total = total + dist
            if pCount > 0:
                print ("(node A " + repr(start) + ", node B "+ repr(end) + "): path length " +
                pCount = pCount - 1

        print ("Average distance is " + repr(float(total/(maxIter-numDC))))
```

```
1
(node A 517, node B 873): path length 5
(node A 2208, node B 940): path length 4
(node A 1599, node B 2273): path length 3
(node A 701, node B 1048): path length 6
(node A 2076, node B 1255): path length 4
(node A 697, node B 729): path length 2
(node A 261, node B 1536): path length 3
(node A 1068, node B 3615): path length 4
(node A 3639, node B 3203): path length 5
(node A 2172, node B 939): path length 4
(node A 1980, node B 1638): path length 4
(node A 2767, node B 2782): path length 2
(node A 1960, node B 293): path length 4
(node A 1557, node B 552): path length 3
(node A 3793, node B 3421): path length 5
(node A 443, node B 2668): path length 4
(node A 1618, node B 1336): path length 2
(node A 3465, node B 91): path length 5
(node A 2460, node B 3469): path length 5
(node A 1024, node B 3600): path length 4
(node A 2763, node B 1964): path length 4
(node A 2431, node B 544): path length 3
(node A 2193, node B 3468): path length 5
(node A 3143, node B 2125): path length 4
(node A 1514, node B 2877): path length 3
(node A 3964, node B 3821): path length 2
(node A 1015, node B 1411): path length 2
(node A 1950, node B 1231): path length 4
(node A 3903, node B 1678): path length 4
(node A 1893, node B 2812): path length 3
(node A 3727, node B 1139): path length 4
(node A 136, node B 1869): path length 3
(node A 2333, node B 207): path length 4
(node A 4037, node B 2533): path length 5
(node A 3251, node B 2236): path length 4
(node A 2164, node B 1088): path length 4
(node A 4033, node B 234): path length 6
(node A 1686, node B 2982): path length 3
```

```
(node A 3426, node B 1361): path length 3
(node A 1290, node B 1467): path length 2
(node A 2480, node B 1301): path length 4
(node A 1507, node B 816): path length 6
(node A 871, node B 134): path length 6
(node A 2123, node B 545): path length 3
(node A 3135, node B 1046): path length 3
(node A 1023, node B 3675): path length 4
(node A 2464, node B 284): path length 4
(node A 1389, node B 3184): path length 3
(node A 3622, node B 3553): path length 2
(node A 2249, node B 401): path length 4
(node A 2312, node B 2032): path length 2
(node A 3636, node B 3398): path length 5
(node A 1324, node B 278): path length 3
(node A 3501, node B 3250): path length 4
(node A 3580, node B 3410): path length 5
(node A 245, node B 164): path length 2
(node A 1524, node B 113): path length 3
(node A 3536, node B 3013): path length 5
(node A 3294, node B 3281): path length 2
(node A 1702, node B 719): path length 5
(node A 3309, node B 1220): path length 3
(node A 977, node B 3121): path length 3
(node A 2378, node B 3993): path length 5
(node A 1450, node B 3797): path length 3
(node A 2559, node B 2304): path length 2
(node A 4001, node B 1919): path length 5
(node A 3937, node B 3565): path length 2
(node A 3551, node B 1390): path length 4
(node A 3866, node B 2470): path length 5
(node A 1271, node B 3012): path length 3
(node A 1593, node B 1831): path length 2
(node A 3391, node B 2399): path length 4
(node A 1960, node B 2322): path length 1
(node A 1677, node B 636): path length 3
(node A 1858, node B 1753): path length 2
(node A 107, node B 2538): path length 3
(node A 1070, node B 604): path length 2
(node A 2703, node B 520): path length 4
(node A 2281, node B 2312): path length 2
(node A 3416, node B 3912): path length 5
(node A 225, node B 1697): path length 3
(node A 3104, node B 3473): path length 5
(node A 264, node B 1807): path length 3
(node A 196, node B 409): path length 4
(node A 792, node B 2036): path length 7
(node A 3828, node B 2148): path length 5
```

```
(node A 426, node B 2277): path length 3
(node A 1048, node B 85): path length 3
(node A 976, node B 130): path length 3
(node A 3699, node B 345): path length 5
(node A 3374, node B 2345): path length 4
(node A 2987, node B 3213): path length 2
(node A 1987, node B 2106): path length 2
(node A 2548, node B 3855): path length 5
(node A 1405, node B 2584): path length 4
(node A 2112, node B 2396): path length 2
(node A 204, node B 2779): path length 4
(node A 63, node B 3666): path length 5
(node A 2813, node B 1362): path length 3
(node A 1986, node B 2688): path length 4
Average distance is 3.64
```

3.2 Q9b

p = number of edges/ total possible edges = 0.0108

```
In [172]: # Q9b
          p = fb.shape[0]/ ((N*(N-1))/2)
          print ("Facebook data has p value: " + repr(p))
```

```
Facebook data has p value: 0.010819963503439287
```

3.3 Q9C

Average shortest path from Facebook data is about 3.6; Average shortest path from constructed data is about 2.6, which is smaller than the actual data. The reason may be: in the constructed data, the graph is uniformly generated with probability p. Each person has the same expected number of friends. All are equal. However, in the real world, the distribution of number of friends is not uniform. Some people tend to have more friends, some less. The distribution should be more like gaussian. In the graph view, some nodes in the actual Facebook data is more dense than others. That's why the average shortest path is larger in the real data set.

```
In [7]: # Q9c
        avgDist(4039, 0.01081996, 1000)
```

```
Out[7]: 2.58
```