بسمه تعالی

ارائه دهنده : زهرا منصوری

عنوان : کارگاه اول _ تحلیل شبکه های اجتماعی _ بخش دوم (مطالعه یک و دو)

فهرست

مطالعه یک

شبکه انتخابی

فایل داده شبکه انتخاب شده با نام ’facebook_combined.txt‘ در پوشه پروژه وجود دارد. فایل انتخابی مربوط به یک گراف غیر جهت دار می باشد.

مشخصات گراف انتخاب شده

ابتدا گراف را در یک dataframe لود کرده و با استفاده از متد from_pandas_edgelist() از کتابخانه networkx گراف زیر را تشکیل دادیم.

```
In [1]: import pandas as pd
        import networkx as nx
        import random
        import operator
```

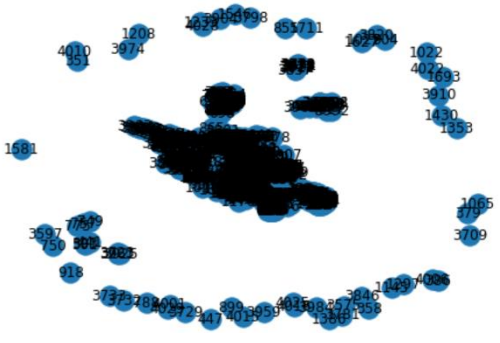
```
In [2]: df = pd.read_csv('facebook_combined.txt', delimiter=" ")
        df.columns =['FromNodeId', 'ToNodeId']
        g = nx.from_pandas_edgelist(df, 'FromNodeId', 'ToNodeId', create_using=nx.Graph())
        print(nx.info(g))
```

```
Name:
Type: Graph
Number of nodes: 4039
Number of edges: 88233
Average degree:  43.6905
```

نمونه برداری تصادفی از گره ها

۲۰۰۰ گره از گراف اصلی را به صورت رندوم انتخاب کرده و زیرگراف G1 را به شکل زیر رسم کرده ایم.

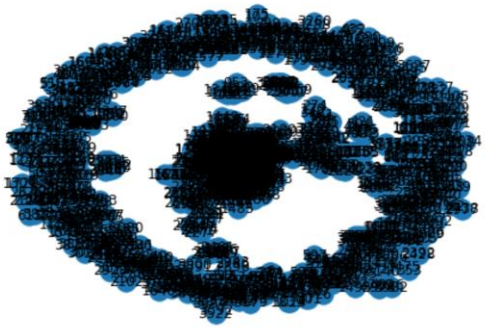
```
In [3]: k = 2000
        sampled_nodes = random.sample(g.nodes, k)
        G1 = g.subgraph(sampled_nodes)
        nx.draw(G1, with_labels=True)
```



نمونه برداری تصادفی از یال ها

۱۰۰۰ یال از گراف اصلی را به صورت رندوم انتخاب کرده و زیرگراف G2 را به شکل زیر رسم کرده ایم.

In [4]:
```
k = 1000
sampled_edgs = random.sample(g.edges(), k)
G2 = nx.Graph(sampled_edgs)
nx.draw(G2, with_labels=True)
```

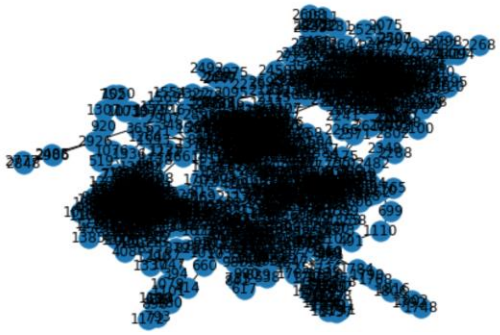


نمونه برداری بر اساس قدم زدن تصادفی

با استفاده از تابع random_walk_sampling_simple() که به عنوان ورودی یک گراف و یک گره برای شروع می گیرد؛ زیرگراف G3 را به شکل زیر رسم کرده ایم.

In [5]:
```
growth_size = 2
T = 100    # number of iterations
def random_walk_sampling_simple(complete_graph, nodes_to_sample):
        complete_graph = nx.convert_node_labels_to_integers(complete_graph, 0, 'default', True)
        # giving unique id to every node same as built-in function id
        for n, data in complete_graph.nodes(data=True):
            complete_graph.nodes[n]['id'] = n

        nr_nodes = len(complete_graph.nodes())
        upper_bound_nr_nodes_to_sample = nodes_to_sample
        index_of_first_random_node = random.randint(0, nr_nodes - 1)
        sampled_graph = nx.Graph()

        sampled_graph.add_node(complete_graph.nodes[index_of_first_random_node]['id'])

        iteration = 1
        edges_before_t_iter = 0
        curr_node = index_of_first_random_node
        while sampled_graph.number_of_nodes() != upper_bound_nr_nodes_to_sample:
            edges = [n for n in complete_graph.neighbors(curr_node)]
            index_of_edge = random.randint(0, len(edges) - 1)
            chosen_node = edges[index_of_edge]
            sampled_graph.add_node(chosen_node)
            sampled_graph.add_edge(curr_node, chosen_node)
            curr_node = chosen_node
            iteration = iteration + 1

            if iteration % T == 0:
                if ((sampled_graph.number_of_edges() - edges_before_t_iter) < growth_size):
                    curr_node = random.randint(0, nr_nodes - 1)
                edges_before_t_iter = sampled_graph.number_of_edges()
        return sampled_graph
```

```
In [6]: G3 = random_walk_sampling_simple(g,random.choice(list(g.nodes())))
        nx.draw(G3, with_labels=True)
```



گره های مهم هر گراف

متد find_important_nodes() به عنوان ورودی یک گراف گرفته و در دیکشنری dic گره و درجه را ذخیره می کند. سپس dic را بر اساس value (درجه هر گره) مرتب کردیم و ۱۰۰ گره اول با بیشترین درجه را به عنوان خروجی بر می گردانیم.

```
In [7]: def find_important_nodes(graph):
            dic = {}
            for n in graph.degree():
                dic[n[0]] = n[1]
            sorted_d = dict( sorted(dic.items(), key=operator.itemgetter(1),reverse=True))
            return [k for k in list(sorted_d)[:100]]
```

```
In [8]: most_important_nodes_G1 = find_important_nodes(G1)
        most_important_nodes_G1
```

```
Out[8]: [1684,
         1912,
         0,
         483,
         1985,
         2233,
         1993,
         1941,
         2123,
         1730,
         2240,
         2369,
         2507,
         2604,
         2611,
         1938,
         2047,
         2073,
         2131,
         1827
```

```
In [9]: most_important_nodes_G2 = find_important_nodes(G2)
        most_important_nodes_G2
```

```
Out[9]: [107,
         3437,
         1912,
         1684,
         2624,
         2200,
         1946,
         2654,
         2510,
         2619,
         2473,
         2188,
         2376,
         2212,
         2410,
         1574,
         1329,
         2093,
         3299,
         2104
```

```
In [10]: most_important_nodes_G3 = find_important_nodes(G3)
         most_important_nodes_G3
```

```
Out[10]: [106,
          351,
          700,
          838,
          2871,
          1312,
          2403,
          2334,
          2614,
          1284,
          1285,
          1149,
          1069,
          2892,
          2689,
```

```
In [11]: most_important_nodes_g = find_important_nodes(g)
         most_important_nodes_g
```

```
Out[11]: [107,
          1684,
          1912,
          3437,
          0,
          2543,
          2347,
          1888,
          1800,
          1663,
          1352,
          2266,
          483,
          348,
```

مقایسه لیست گره های مهم هر شبکه با لیست گره های مهم گراف اصلی

```
In [12]: set(most_important_nodes_g).intersection(most_important_nodes_G1)
```

```
Out[12]: {0,
          483,
          1078,
          1584,
          1613,
          1684,
          1714,
          1730,
          1746,
          1827,
          1833,
          1912,
          1917,
          1938,
          1941,
          1943,
```

```
In [13]: set(most_important_nodes_g).intersection(most_important_nodes_G2)
```

```
Out[13]: {0,
          107,
          1352,
          1377,
          1589,
          1613,
          1684,
          1714,
          1804,
          1888,
          1912,
          1946,
          1962,
          2088,
          2103,
          2118,
          2131,
          2188,
          2218,
          2266
```

```
In [14]: set(most_important_nodes_g).intersection(most_important_nodes_G3)
```

```
Out[14]: {1199, 2188, 2369, 2542}
```

مطالعه دو

شبکه انتخابی

فایل داده شبکه انتخاب شده با نام ’facebook_combined.txt‘ در پوشه پروژه وجود دارد. فایل انتخابی مربوط به یک گراف غیر جهت دار می باشد.

مشخصات گراف انتخاب شده

ابتدا گراف را در یک dataframe لود کرده و با استفاده از متد from_pandas_edgelist() از کتابخانه networkx گراف زیر را تشکیل دادیم.

```
In [1]: import pandas as pd
        import networkx as nx
        import random
        import matplotlib.pyplot as plt
```

```
In [2]: df = pd.read_csv('facebook_combined.txt', delimiter=" ")
        df.columns =['FromNodeId', 'ToNodeId']
        g = nx.from_pandas_edgelist(df, 'FromNodeId', 'ToNodeId', create_using=nx.Graph())
        print(nx.info(g))
```

```
Name:
Type: Graph
Number of nodes: 4039
Number of edges: 88233
Average degree:  43.6905
```

متد های تعریف شده در مسئله

ابتدا برای راحتی کار متد های مورد نیاز در مسئله را تعریف کرده ایم که به شرح آن ها می پردازیم:

در ابتدا دو لیست s و sr که اولی برای نگه داشتن گره های قرمز و دومی برای نگه داشتن همسایه های جدیدی که قرار است به s اضافه شوند؛ تعریف شده است. دیکشنری iterations نیز، درصد گره های قرمز را در هر iteration نگاه می دارد. لیست node_colors نیز رنگ هر گره را مشخص می کند. که در ابتدا همه گره ها آبی هستند.

- Color()

این متد رنگ هر گره را مشخص میکند. ابتدا به ازای هر گره در sr (که نشان دهنده همسایه های گره های موجود در s هستند)، آن گره را به s اضافه می کنیم و به ازای هر کدام از گره های گراف اگر آن گره در لیست s وجود داشت، رنگ آن را در node_colors به قرمز تغییر می دهیم.

```
In [3]: s = []
        sr = []
        iterations = {}
        node_colors = ['blue' for node in g.nodes()]
        def color():
            for x in sr:
                if x not in s:
                    s.append(x)
            for node in g.nodes():
                if node in s:
                    node_colors[node] = 'red'
```

- find_random_node()

این متد یک گره رندم را با استفاده از متد choice() از کتابخانه random انتخاب کرده و آن را به s اضافه می کند و متد color() را نیز برای تغییر رنگ آن گره از آبی به قرمز فراخوانی میکنیم.

```
In [4]: def find_random_node():
            random_node = random.choice(list(g.nodes()))
            if random_node not in s:
                s.append(random_node)
            color()
```

- report_percentage()

این متد به عنوان ورودی iteration را گرفته و درصد گره های قرمز به کل گره ها را محاسبه کرده و به دیکشنری iterations اضافه می کند.

```
In [5]: def report_percentage(iteration):
            print("iteration: ", iteration)
            p = round(((len(s) / len(g.nodes())) * 100), 2)
            print("percentage: ", p, "%")
            iterations[iteration] = p
```

- find_neighbor()

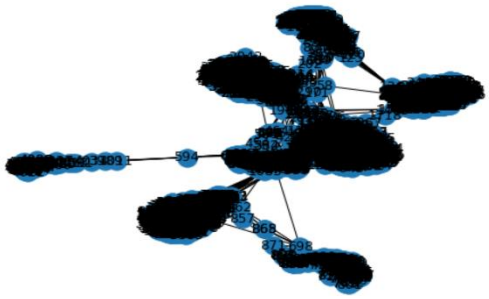
این متد همسایه های گره های موجود در لیست s را محاسبه می کند. برای این کار در ابتدا، هر دفعه لیست sr را خالی کرده و سپس به ازای هر گره در s، همسایه های آن گره را در گراف g با استفاده از متد neighbors() محاسبه کرده و به sr اضافه می کنیم. در نهایت نیز متد color() را فراخوانی کرده تا گره های جدید اضافه شده به s، قرمز شوند.

```
In [8]: def find_neighbor():
            sr[:] = []
            for i in s:
                for n in list(g.neighbors(i)):
                    if n not in s:
                        sr.append(n)
            color()
```

گزارش و نمودارها خروجی به ازای هر iteration

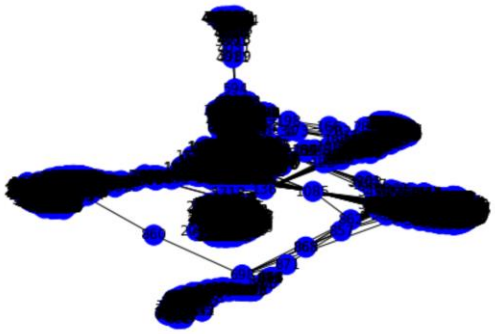
```
In [6]: iteration = 0
        report_percentage(iteration)
        nx.draw(g, with_labels=True)
```

```
iteration:  0
percentage:  0.0 %
```



In [7]:
```
iteration = 1
find_random_node()
report_percentage(iteration)
nx.draw(g, node_color = node_colors, with_labels=True)
```

```
iteration:  1
percentage:  0.02 %
```

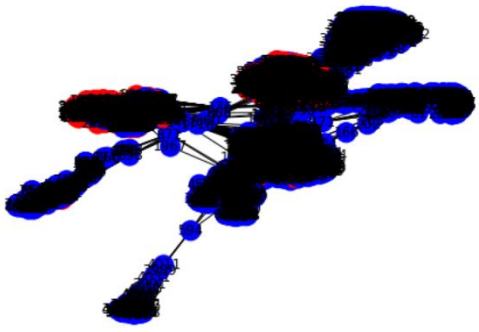


In [9]:
```
iteration = 2
find_neighbor()
report_percentage(iteration)
nx.draw(g, node_color = node_colors, with_labels=True)
```

```
iteration:  2
percentage:  0.74 %
```

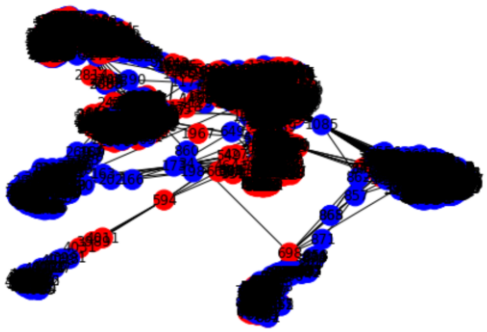
In [10]:
```
iteration = 3
find_neighbor()
report_percentage(iteration)
nx.draw(g, node_color = node_colors, with_labels=True)
```

```
iteration:  3
percentage:  20.35 %
```



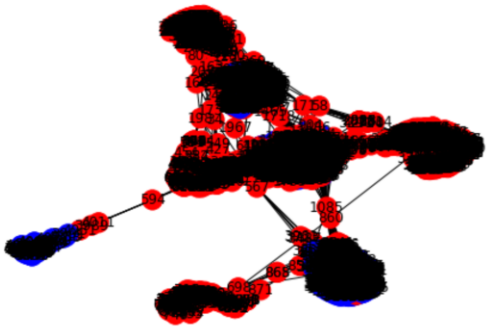
In [11]:
```
iteration = 4
find_neighbor()
report_percentage(iteration)
nx.draw(g, node_color = node_colors, with_labels=True)
```

```
iteration:  4
percentage:  45.33 %
```



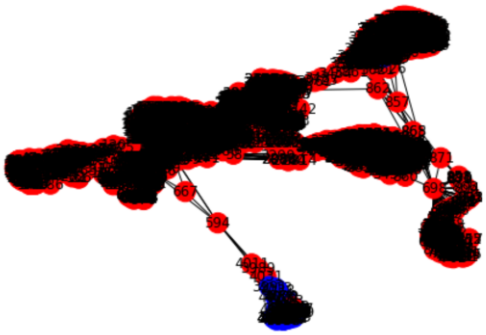
In [12]:
```
iteration = 5
find_neighbor()
report_percentage(iteration)
nx.draw(g, node_color = node_colors, with_labels=True)
```

```
iteration:  5
percentage:  82.37 %
```



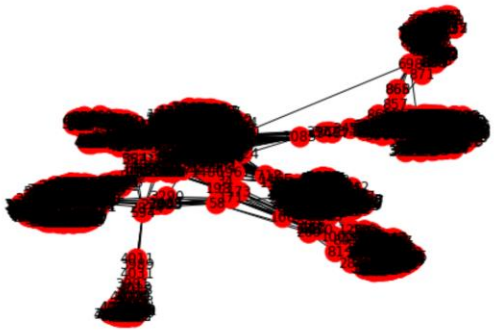
In [13]:
```
iteration = 6
find_neighbor()
report_percentage(iteration)
nx.draw(g, node_color = node_colors, with_labels=True)
```

```
iteration:  6
percentage:  98.64 %
```



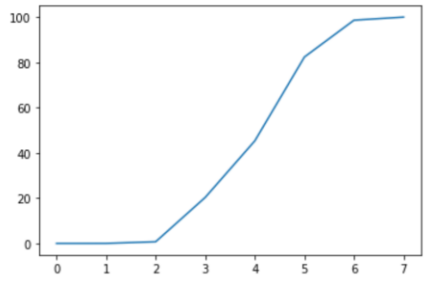
```
In [14]: iteration = 7
         find_neighbor()
         report_percentage(iteration)
         nx.draw(g, node_color = node_colors, with_labels=True)
```

```
iteration:  7
percentage:  100.0 %
```



نمودار برحسب تکرار_درصد گره های قرمز در گام اول

```
In [15]: x = iterations.keys()
         y = iterations.values()
         plt.plot(x, y)
         plt.show()
```



استفاده از مهمترین گره شبکه به جای گره رندم در iteration = 1

ابتدا لیست درجات گره ها را به دست آورده و در degrees ذخیره می کنیم. سپس ماکزیمم این لیست را که نشان دهنده بیشترین درجه است، پیدا کرده و اندیس آن را به عنوان مهم ترین گره شبکه در most_important_node ذخیره می کنیم.

```
In [5]: degrees = [val for (node, val) in sorted(g.degree(), key=lambda pair: pair[0])]
```

```
In [33]: max_degree = max(degrees)
         most_important_node = degrees.index(max_degree)
         print(max_degree)
         print(most_important_node)
```

```
1045
107
```

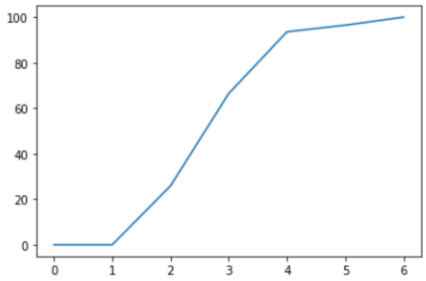
سپس با استفاده از متد output() همان کارهایی را که در iteration های گام اول گفته شد انجام می دهیم با این تفاوت که این بار در iteration = 1، most_important_node را به جای یک گره رندم به s اضافه می کنیم.

```
In [47]: s = []
         sr = []
         iterations = {}
         def output():
             iteration = 0
             while len(s) != len(list(g.nodes())):
                 if iteration == 0:
                     report_percentage(iteration)
                 elif iteration == 1:
                     s.append(most_important_node)
                     color()
                     report_percentage(iteration)
                 else:
                     find_neighbor()
                     report_percentage(iteration)
                 iteration += 1
```

```
In [48]: output()

iteration:  0
percentage:  0.0 %
iteration:  1
percentage:  0.02 %
iteration:  2
percentage:  25.9 %
iteration:  3
percentage:  66.5 %
iteration:  4
percentage:  93.59 %
iteration:  5
percentage:  96.48 %
iteration:  6
percentage:  100.0 %
```

نمودار برحسب تکرار_درصد گره های قرمز در گام دوم

```
In [49]: x = iterations.keys()
         y = iterations.values()
         plt.plot(x, y)
         plt.show()
```



تحلیل نهایی

در گام دوم که از most_important_node به جای گره رندم استفاده شد، پدیده انتشار سریع تر رشد کرد و همانطور که در نتایج مشاهده شد؛ در گام دوم با ۶ دور کلیه گره ها قرمز شدند اما در گام اول با هفت دور این اتفاق افتاد.

علاوه بر موارد ذکر شده، با توجه به شیب نمودار های " تکرار_درصد گره های قرمز" در گام اول و دوم، رشد پدیده انتشار پس از iteration = 1 در گام دوم بسیار بیشتر از گام اول است.