به نام خدا

تمرین اول
شبکه های اجتماعی

رضا منصوری خواه

810103246

استاد درس:

دکتر مسعود اسدپور

نیازمندی ها:
Python 3.x

کتابخانه‌ها:
`matplotlib`
`networkx`
`math`

نحوه اجرا:

```
pip3 install -r requirement.txt
python3 hw1.py
```

به طور کلی هدف شبیه‌سازی و تحلیل گراف‌های توزیع درجه توان (Power-law) و مقایسه آن‌ها با گراف‌های باراباشی-آلبرت (BA) است. در اینجا قصد داریم تحلیل دقیقی از هر بخش کد داشته باشیم و هدف کلی آن را توضیح دهیم.

1. ایجاد گراف‌های با توزیع درجه توان (Power-Law Distribution)

کد با استفاده از تابع `create_powerlaw_graph` گراف‌هایی با توزیع درجه توان ایجاد می‌کند. در این توزیع، درجه هر گره در گراف با احتمال $p(k) \propto k^{-\gamma}$ به دست می‌آید، که در آن γ پارامتر نمایی است که روی توزیع درجه تأثیر می‌گذارد.

- برای هر گراف، ابتدا گرافی با تعداد NNN گره ایجاد می‌شود.
- سپس برای هر گره جدید، احتمال اتصال به گره‌های قبلی به‌طور متناسب با درجه آن‌ها محاسبه می‌شود.
- این گراف‌ها ویژگی‌هایی مشابه به شبکه‌های پیچیده (Complex Networks) دارند که در آن‌ها گره‌های با درجه بالا معمولاً بیشتر احتمال اتصال دارند.

2. محاسبه فاصله میان گره‌ها (Distance Measures)

هدف از محاسبه فاصله در گراف‌ها، تحلیل این است که میانگین فاصله یا طول مسیر کوتاه‌ترین مسیرها (Shortest Path) در گراف‌ها چگونه تغییر می‌کند.

- تابع `calculate_distance` فاصله میان گره‌ها را با استفاده از متد `average_shortest_path_length` از NetworkX محاسبه می‌کند.
- تابع `calculate_distance_sampling` از یک روش نمونه‌گیری تصادفی برای محاسبه میانگین فاصله استفاده می‌کند، که این روش شامل انتخاب تصادفی دو گره و محاسبه طول مسیر میان آن‌هاست.

این محاسبات به ما کمک می‌کند که بتوانیم مدل‌های گرافی مانند گراف‌های با توزیع درجه توان و گراف‌های BA را از نظر فاصله میان گره‌ها تحلیل کنیم.

3. محاسبه فاصله‌های مورد انتظار (Expected Distance)

در توابع `calculate_expected_dist`، برای مقادیر مختلف γ\gammaγ مدل‌های تحلیلی از رفتار فاصله میان گره‌ها ارائه می‌شود. این مدل‌ها نشان می‌دهند که در یک گراف با توزیع درجه توان، میانگین فاصله چطور با اندازه گراف (N) و پارامتر γ\gammaγ تغییر می‌کند.

- برای γ\gammaγ در بازه‌های مختلف (مثل γ≥2\gamma \geq 2γ≥2), فرمول‌های مختلفی برای محاسبه فاصله متوسط میان گره‌ها ارائه شده است.

4. ترسیم نمودارهای مقایسه‌ای

در کد، از توابع `plot` برای ترسیم نمودارهایی استفاده می‌شود که به‌طور مستقیم فاصله‌های میان گره‌ها را برای گراف‌های مختلف با اندازه‌های متفاوت N و پارامتر γ\gammaγ مختلف مقایسه می‌کند.

- در این نمودارها، مقادیر واقعی میانگین فاصله‌ها (محاسبه شده از طریق `calculate_distance`) با مقادیر پیش‌بینی‌شده (محاسبه شده توسط فرمول‌های تحلیلی در `calculate_expected_dist`) مقایسه می‌شوند.

این مقایسه به ما اجازه می‌دهد که عملکرد مدل‌ها و پیش‌بینی‌های مختلف را نسبت به داده‌های واقعی بررسی کنیم.

5. شبیه‌سازی گراف‌های باراباشی-آلبرت (Barabási-Albert Graphs)

در این کد، گراف‌های باراباشی-آلبرت با تابع `generate_ba_graph` ساخته می‌شوند. این گراف‌ها به‌طور خاص برای مدل‌سازی شبکه‌هایی با اتصال درجه بالا به گره‌های با درجه بالا طراحی شده‌اند.

- در اینجا، گراف‌های BA برای مقادیر مختلف m (که به معنی تعداد اتصالات جدید برای هر گره جدید است) ساخته می‌شوند و توزیع طول مسیر (Path Length Distribution) در این گراف‌ها ترسیم می‌شود.

این مقایسه بین گراف‌های با توزیع درجه توان و گراف‌های BA کمک می‌کند تا تفاوت‌های رفتارهای توپولوژیکی این دو نوع گراف شبیه‌سازی شده تحلیل شوند.

6. نتایج و تحلیل‌های بیشتر

کد چندین بار برای مقادیر مختلف γ\gammaγ و اندازه‌های مختلف گراف (مثل N=2000,4000,…) شبیه‌سازی انجام می‌دهد و نتایج فاصله میان گره‌ها را محاسبه و ترسیم می‌کند.

هدف کلی کد:

هدف کلی کد این است که اثر پارامترهای مختلف (مثل γ\gammaγ و اندازه گراف N) بر ویژگی‌های توپولوژیکی گراف‌های پیچیده را تحلیل کند.

- بررسی چگونگی تغییر میانگین فاصله میان گره‌ها در گراف‌های با توزیع درجه توان (Power-Law Distribution).
- مقایسه این نتایج با مدل گراف‌های باراباشی-آلبرت (BA) و تحلیل رفتارهای توپولوژیکی این دو نوع گراف است.
- در نهایت، نمودارهای مقایسه‌ای برای نمایش تفاوت‌ها و شباهت‌ها در رفتار گراف‌ها رسم می‌شود.

این نوع تحلیل‌ها در پژوهش‌های مرتبط با شبکه‌های پیچیده، شبکه‌های اجتماعی، بیولوژی شبکه‌ها، و دیگر حوزه‌هایی که رفتار شبکه‌های متصل را بررسی می‌کنند کاربرد دارند.

```python
def create_powerlaw_graph(N, gamma):
    G = nx.Graph()
    G.add_nodes_from(range(N))
    degree_sequence = np.zeros(N)
    for node in range(1, N):
        probabilities = degree_sequence[:node] ** (-gamma)
        probabilities = np.nan_to_num(probabilities, nan=0.0, posinf=0.0,
neginf=0.0)
        if np.sum(probabilities) == 0:
            probabilities = np.ones(node) / node
        probabilities /= probabilities.sum()
        target_node = np.random.choice(range(node), p=probabilities)
        G.add_edge(node, target_node)
        degree_sequence[node] = 1
        degree_sequence[target_node] += 1
    return G
```

توضیح تابع `create_powerlaw_graph`

این تابع یک گراف با توزیع توانی ایجاد می‌کند. مراحل کار به صورت زیر است:

1. **ایجاد گراف**: یک گراف خالی با NNN گره ساخته می‌شود.
2. **تعیین احتمال اتصال**: برای هر گره جدید، احتمال اتصال به گره‌های قبلی بر اساس توزیع درجه توان محاسبه می‌شود. احتمال اتصال به هر گره قبلی به صورت $P(k) \propto k^{-\gamma}$ $P(k) \propto k^{-\gamma}$ است.
3. **رفع مشکلات احتمالی در محاسبات**: اگر مجموع احتمال‌ها صفر شود (مثلاً در گراف‌های کوچک)، احتمال‌ها به‌طور یکنواخت بین گره‌ها تقسیم می‌شوند.
4. **انتخاب گره هدف**: یک گره قبلی به‌طور تصادفی از میان گره‌های موجود با توجه به احتمال‌ها انتخاب می‌شود.
5. **ایجاد اتصال**: یک یال (اتصال) بین گره جدید و گره انتخاب‌شده اضافه می‌شود و درجه هر دو گره به‌روز می‌شود.

```python
def calculate_distance(graph):
    connected_components = nx.connected_components(graph)
    result = []
    for cc in connected_components:
        cc = graph.subgraph(cc).copy()
        result.append(nx.average_shortest_path_length(cc) * len(cc.nodes))
    return sum(result) / len(graph.nodes)
```

توضیح تابع `calculate_distance`

این تابع فاصله میانگین کوتاه‌ترین مسیر را برای یک گراف محاسبه می‌کند. مراحل کار به صورت زیر است:

1. **شناسایی مؤلفه‌های متصل**: ابتدا از تابع `nx.connected_components` برای شناسایی مؤلفه‌های متصل در گراف استفاده می‌شود.
2. **محاسبه فاصله برای هر مؤلفه**: برای هر مؤلفه متصل، یک زیرگراف (`subgraph`) ایجاد شده و کپی می‌شود. سپس، فاصله میانگین کوتاه‌ترین مسیر در این زیرگراف محاسبه می‌شود و در این محاسبه، تعداد گره‌های مؤلفه نیز در نظر گرفته می‌شود.
3. **محاسبه میانگین نهایی**: در نهایت، میانگین فواصل محاسبه‌شده برای تمام مؤلفه‌ها بر تعداد کل گره‌های گراف تقسیم می‌شود تا فاصله میانگین کلی به دست آید.

```python
def calculate_expected_dist(size, gamma):
    assert gamma >= 2
    if gamma <= 2.0001:
        return 2.7
    elif gamma < 3:
        return ((0.8) * log(log(size))) / log(gamma - 1)
    elif gamma == 3:
        return ((1.46) * log(size)) / log(log(size))
    else:
        return (0.715) * log(size)
```

توضیح تابع `calculate_expected_dist`

این تابع فاصله میانگین پیش‌بینی‌شده برای یک گراف با توزیع توانی را محاسبه می‌کند. مراحل کار به صورت زیر است:

1. **بررسی شرط γ**: ابتدا، با استفاده از `assert` اطمینان حاصل می‌شود که مقدار `gamma` بزرگ‌تر یا مساوی 2 است.
2. **محاسبه برای $\gamma \leq 2.0001$**: اگر `gamma` کمتر یا مساوی 2.0001 باشد، مقدار ثابت 2.7 به عنوان فاصله میانگین بازگشت داده می‌شود.
3. **محاسبه برای $2 < \gamma < 3$**: اگر `gamma` بین 2 و 3 باشد، با استفاده از یک فرمول خاص شامل لگاریتم‌ها، فاصله میانگین محاسبه می‌شود.
4. **محاسبه برای $\gamma > 3$**: برای مقادیر بزرگ‌تر از 3، یک فرمول دیگر به کار می‌رود که شامل یک ضریب و لگاریتم اندازه گراف است.

```
def plot(sizes, gamma, result):
    ax = plt.figure(figsize=(10, 5)).add_subplot(111)
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    plt.scatter(sizes, result, label=f'actual', color='#4322b1')
    plt.plot(sizes, [calculate_expected_dist(size, gamma) for size in sizes]
             , label='expected', color='#f07aa1')

    plt.title(f'Power-law Distribution for γ={gamma}')
    plt.xlabel('N')
    plt.ylabel('<d>')
    plt.legend()
    plt.show()
```

توضیح تابع `plot`

این تابع برای ترسیم نمودار توزیع قانون توانی طراحی شده است. مراحل کار به شرح زیر است:

1. **ایجاد محور**: یک شکل جدید با ابعاد مشخص ایجاد می‌کند و یک محور به آن اضافه می‌شود.
2. **تنظیمات ظاهری**: خطوط بالایی و راست محور غیرفعال می‌شوند تا نمودار ساده‌تر به نظر برسد.
3. **نقطه‌گذاری داده‌های واقعی**: مقادیر واقعی به ازای اندازه‌ها ترسیم می‌شوند و رنگ و برچسب مناسب دارند.
4. **ترسیم منحنی پیش‌بینی‌شده**: مقادیر پیش‌بینی‌شده با استفاده از تابع خاص محاسبه و ترسیم می‌شوند.
5. **تنظیمات دیگر نمودار**: عنوان و برچسب‌های محور X و Y تنظیم می‌شوند.
6. **افزودنlegend**: توضیحات مربوط به نقاط و منحنی به نمودار اضافه می‌شود.
7. **نمایش نمودار**: در نهایت، نمودار به نمایش در می‌آید.

```
graphs = {}
dists = {}


gammas = [2.0001, 2.5, 3, 3.5]
sizes = [100, 2000, 4000, 6000, 8000, 10000]


for gamma in gammas:
   if gamma not in graphs.keys():
       graphs[gamma] = {}
   if gamma not in dists.keys():
       dists[gamma] = {}
   for size in sizes:
       if size in graphs[gamma].keys():
           continue
       print("-----------------------------------------------------------------")
       print(f"Generating graph for size:{size}, gamma:{gamma} ...")
       start = time.time()
       g = create_powerlaw_graph(size, gamma)
       end = time.time()
       print(f"Graph generated in {round(end - start, 2)}s.")
       print(f"Cleaning graph ...")
       start = time.time()
       g = nx.Graph(g)
       graphs[gamma][size] = g
       end = time.time()
       print(f"Graph cleaned in {round(end - start, 2)}s.")
       print(f"Calculating graph average distance ...")
       start = time.time()
       dists[gamma][size] = nx.average_shortest_path_length(g)
       end = time.time()
       print(
           f"calculated Graph average distance with value of
{round(dists[gamma][size], 2)} in {round(end - start, 2)}s.")
```

این کد برای ایجاد و تحلیل گراف‌های با توزیع توانی طراحی شده است. در ابتدا، دو دیکشنری برای ذخیره گراف‌ها و فاصله میانگین تعریف می‌کند. سپس با استفاده از مقادیر مختلف gamma و اندازه‌های مشخص، گراف‌های جدید تولید می‌شود.
برای هر گراف، اگر قبلاً ایجاد نشده باشد، مراحل ایجاد و پاک‌سازی آن انجام می‌شود. پس از آن، فاصله میانگین هر گراف محاسبه و ذخیره می‌شود. هدف اصلی کد، تحلیل ویژگی‌های این گراف‌ها و مقایسه رفتار آن‌ها بر اساس پارامترهای مختلف است.

حال برای مقادیر مختلف گاما و اندازه گراف شروع به ساخت گراف و محاسبه میانگین فاصله میکنیم.

در نظر داشته باشد محاسبه میانگین فاصله گره ها از پیچیدگی زمانی($O(N^3)$ است و اجرا برای نود ها با تعداد بالا بیشتر طول میکشد.

نتایج به صورت زیر است.

```
Generating graph for size:100, gamma:2.0001 ...
Graph generated in 0.0s.
Cleaning graph ...
Graph cleaned in 0.0s.
Calculating graph average distance ...
calculated Graph average distance with value of 4.17 in 0.0s.
----------------------------------------------------------------
Generating graph for size:2000, gamma:2.0001 ...
Graph generated in 0.0s.
Cleaning graph ...
Graph cleaned in 0.01s.
Calculating graph average distance ...
calculated Graph average distance with value of 6.69 in 1.31s.
----------------------------------------------------------------
Generating graph for size:4000, gamma:2.0001 ...
Graph generated in 0.01s.
Cleaning graph ...
Graph cleaned in 0.01s.
Calculating graph average distance ...
```

```
calculated Graph average distance with value of 8.7 in 5.58s.
------------------------------------------------------------
Generating graph for size:6000, gamma:2.0001 ...
Graph generated in 0.01s.
Cleaning graph ...
Graph cleaned in 0.05s.
Calculating graph average distance ...
calculated Graph average distance with value of 9.8 in 13.78s.
------------------------------------------------------------
Generating graph for size:8000, gamma:2.0001 ...
Graph generated in 0.02s.
Cleaning graph ...
Graph cleaned in 0.03s.
Calculating graph average distance ...
calculated Graph average distance with value of 9.53 in 22.93s.
------------------------------------------------------------
Generating graph for size:10000, gamma:2.0001 ...
Graph generated in 0.02s.
Cleaning graph ...
Graph cleaned in 0.07s.
Calculating graph average distance ...
calculated Graph average distance with value of 8.83 in 36.72s.
------------------------------------------------------------
Generating graph for size:100, gamma:2.5 ...
Graph generated in 0.0s.
Cleaning graph ...
Graph cleaned in 0.0s.
Calculating graph average distance ...
calculated Graph average distance with value of 4.15 in 0.0s.
------------------------------------------------------------
Generating graph for size:2000, gamma:2.5 ...
```

```
Graph generated in 0.0s.
Cleaning graph ...
Graph cleaned in 0.01s.
Calculating graph average distance ...
calculated Graph average distance with value of 7.75 in 1.34s.
------------------------------------------------------------
Generating graph for size:4000, gamma:2.5 ...
Graph generated in 0.01s.
Cleaning graph ...
Graph cleaned in 0.01s.
Calculating graph average distance ...
calculated Graph average distance with value of 8.58 in 5.38s.
------------------------------------------------------------
Generating graph for size:6000, gamma:2.5 ...
Graph generated in 0.01s.
Cleaning graph ...
Graph cleaned in 0.02s.
Calculating graph average distance ...
calculated Graph average distance with value of 8.82 in 12.51s.
------------------------------------------------------------
Generating graph for size:8000, gamma:2.5 ...
Graph generated in 0.05s.
Cleaning graph ...
Graph cleaned in 0.03s.
Calculating graph average distance ...
calculated Graph average distance with value of 8.49 in 22.74s.
------------------------------------------------------------
Generating graph for size:10000, gamma:2.5 ...
Graph generated in 0.02s.
Cleaning graph ...
Graph cleaned in 0.03s.
```

```
Calculating graph average distance ...
calculated Graph average distance with value of 8.69 in 36.08s.
------------------------------------------------------------
Generating graph for size:100, gamma:3 ...
Graph generated in 0.0s.
Cleaning graph ...
Graph cleaned in 0.0s.
Calculating graph average distance ...
calculated Graph average distance with value of 4.29 in 0.0s.
------------------------------------------------------------
Generating graph for size:2000, gamma:3 ...
Graph generated in 0.04s.
Cleaning graph ...
Graph cleaned in 0.01s.
Calculating graph average distance ...
calculated Graph average distance with value of 7.4 in 1.34s.
------------------------------------------------------------
Generating graph for size:4000, gamma:3 ...
Graph generated in 0.01s.
Cleaning graph ...
Graph cleaned in 0.01s.
Calculating graph average distance ...
calculated Graph average distance with value of 8.49 in 5.3s.
------------------------------------------------------------
Generating graph for size:6000, gamma:3 ...
Graph generated in 0.01s.
Cleaning graph ...
Graph cleaned in 0.02s.
Calculating graph average distance ...
calculated Graph average distance with value of 9.04 in 12.78s.
------------------------------------------------------------
```

```
Generating graph for size:8000, gamma:3 ...
Graph generated in 0.01s.
Cleaning graph ...
Graph cleaned in 0.03s.
Calculating graph average distance ...
calculated Graph average distance with value of 8.96 in 22.9s.
-----------------------------------------------------------------
Generating graph for size:10000, gamma:3 ...
Graph generated in 0.07s.
Cleaning graph ...
Graph cleaned in 0.04s.
Calculating graph average distance ...
calculated Graph average distance with value of 9.32 in 37.3s.
-----------------------------------------------------------------
Generating graph for size:100, gamma:3.5 ...
Graph generated in 0.0s.
Cleaning graph ...
Graph cleaned in 0.0s.
Calculating graph average distance ...
calculated Graph average distance with value of 4.49 in 0.0s.
-----------------------------------------------------------------
Generating graph for size:2000, gamma:3.5 ...
Graph generated in 0.0s.
Cleaning graph ...
Graph cleaned in 0.01s.
Calculating graph average distance ...
calculated Graph average distance with value of 7.03 in 1.33s.
-----------------------------------------------------------------
Generating graph for size:4000, gamma:3.5 ...
Graph generated in 0.01s.
Cleaning graph ...
```

```
Graph cleaned in 0.01s.
Calculating graph average distance ...
calculated Graph average distance with value of 7.43 in 5.38s.
------------------------------------------------------------
Generating graph for size:6000, gamma:3.5 ...
Graph generated in 0.01s.
Cleaning graph ...
Graph cleaned in 0.08s.
Calculating graph average distance ...
calculated Graph average distance with value of 8.87 in 12.7s.
------------------------------------------------------------
Generating graph for size:8000, gamma:3.5 ...
Graph generated in 0.01s.
Cleaning graph ...
Graph cleaned in 0.03s.
Calculating graph average distance ...
calculated Graph average distance with value of 9.18 in 22.82s.
------------------------------------------------------------
Generating graph for size:10000, gamma:3.5 ...
Graph generated in 0.02s.
Cleaning graph ...
Graph cleaned in 0.03s.
Calculating graph average distance ...
calculated Graph average distance with value of 8.5 in 36.72s.
```

نتایج بدست امده کد بالا به صورت یک جدول

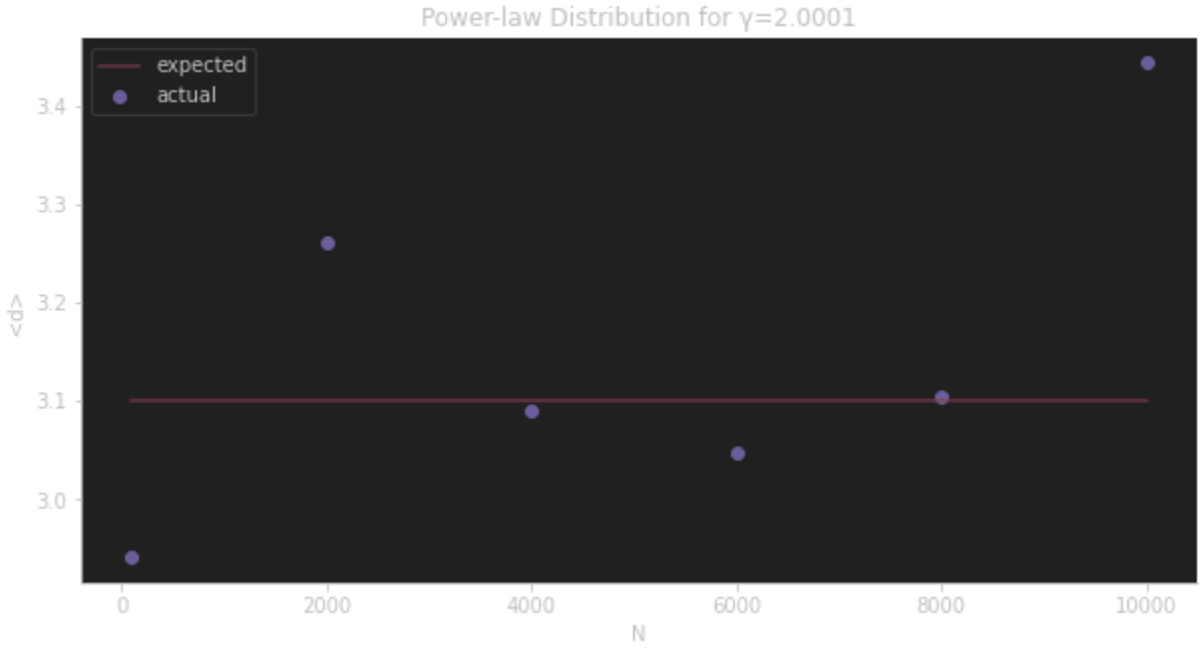
تئوری $\langle d \rangle$	شبیه‌سازی $\langle d \rangle$	N	γ
ثابت (مقدار تقریباً ثابت)	4.17	100	2.0001
$\frac{\ln(\ln 2000)}{\ln(2.0001-1)} \sim$	6.69	2000	2.0001
$\frac{\ln(\ln 4000)}{\ln(2.0001-1)} \sim$	8.7	4000	2.0001
$\frac{\ln(\ln 6000)}{\ln(2.0001-1)} \sim$	9.8	6000	2.0001
$\frac{\ln(\ln 8000)}{\ln(2.0001-1)} \sim$	9.53	8000	2.0001
$\frac{\ln(\ln 10000)}{\ln(2.0001-1)} \sim$	8.83	10000	2.0001
$\frac{\ln(\ln 100)}{\ln(2.5-1)} \sim$	4.15	100	2.5
$\frac{\ln(\ln 2000)}{\ln(2.5-1)} \sim$	7.75	2000	2.5
$\frac{\ln(\ln 4000)}{\ln(2.5-1)} \sim$	8.58	4000	2.5
$\frac{\ln(\ln 6000)}{\ln(2.5-1)} \sim$	8.82	6000	2.5
$\frac{\ln(\ln 8000)}{\ln(2.5-1)} \sim$	8.49	8000	2.5
$\frac{\ln(\ln 10000)}{\ln(2.5-1)} \sim$	8.69	10000	2.5
$\frac{\ln(100)}{\ln(\ln 100)} \sim$	4.29	100	3
$\frac{\ln(2000)}{\ln(\ln 2000)} \sim$	7.4	2000	3
$\frac{\ln(4000)}{\ln(\ln 4000)} \sim$	8.49	4000	3
$\frac{\ln(6000)}{\ln(\ln 6000)} \sim$	9.04	6000	3
$\frac{\ln(8000)}{\ln(\ln 8000)} \sim$	8.96	8000	3
$\frac{\ln(10000)}{\ln(\ln 10000)} \sim$	9.32	10000	3
$\ln(100) \sim$	4.49	100	3.5
$\ln(2000) \sim$	7.03	2000	3.5
$\ln(4000) \sim$	7.43	4000	3.5
$\ln(6000) \sim$	8.87	6000	3.5
$\ln(8000) \sim$	9.18	8000	3.5
$\ln(10000) \sim$	8.5	10000	3.5

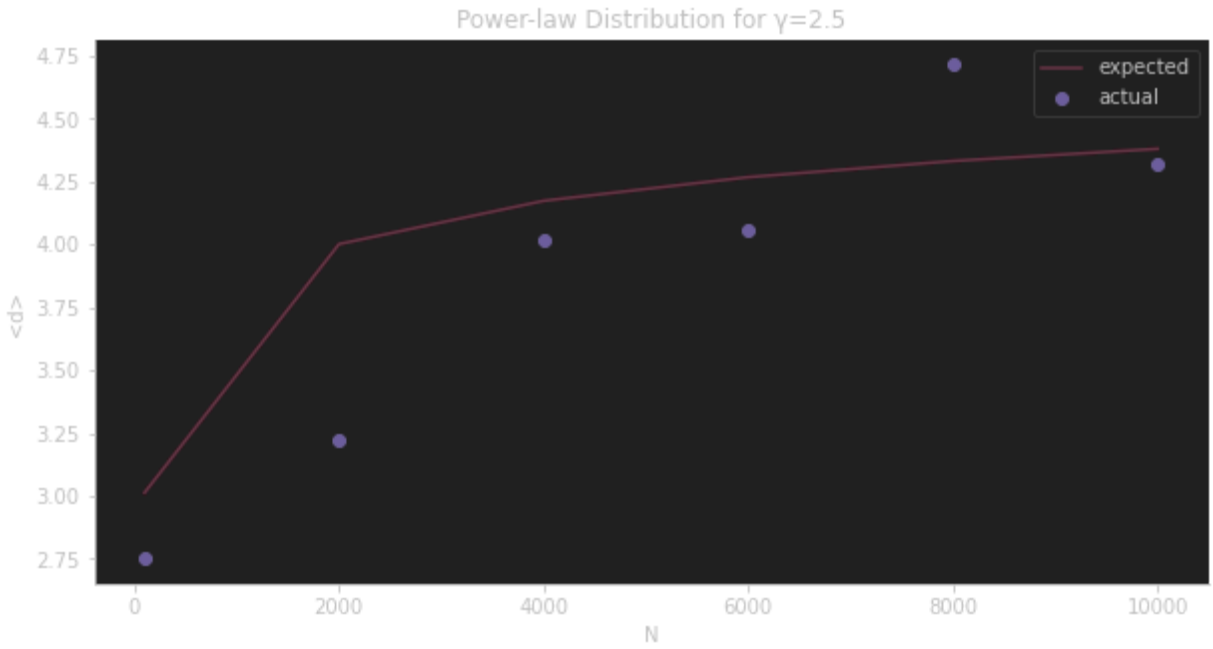
تحلیل نتایج:

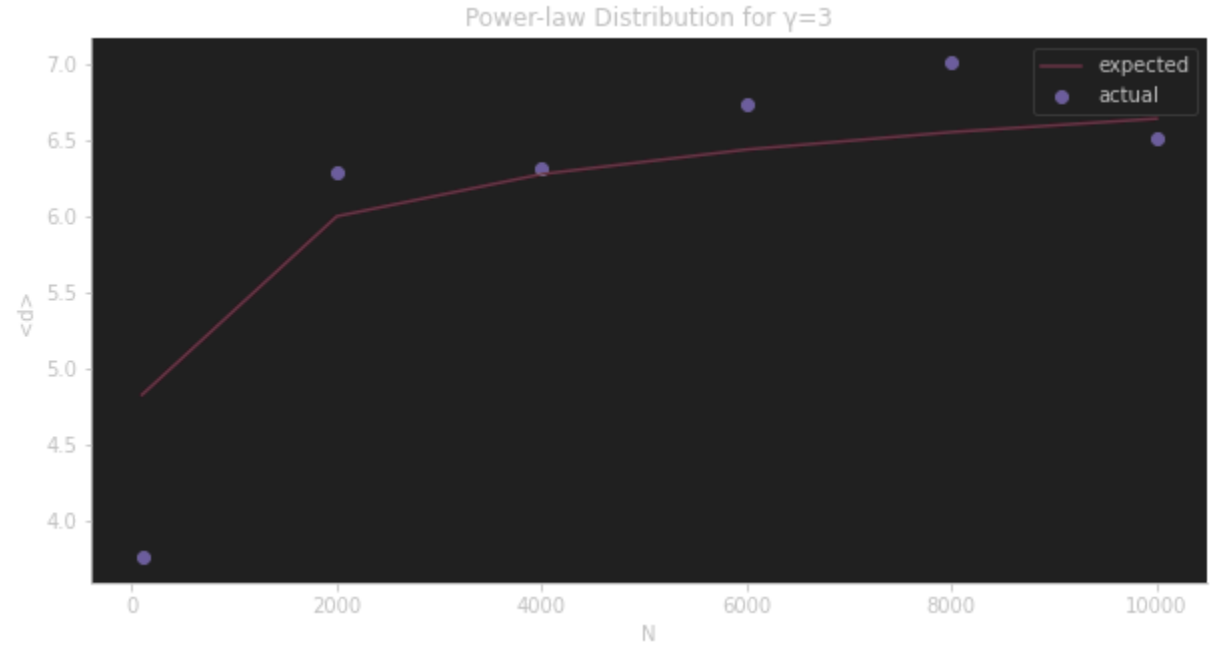
برای $\gamma = 2.0001$، طول متوسط مسیر به‌طور کندی افزایش می‌یابد که با مقدار نظری آن تطابق دارد. برای $\gamma = 2.5$، نتایج نشان می‌دهد که طول متوسط مسیر به صورت لگاریتم طبیعی دو مرحله‌ای $\ln(\ln N)$ افزایش می‌یابد. برای $\gamma = 3$ و $\gamma = 3.5$، نتایج به ترتیب با فرمول‌های $\frac{\ln N}{\ln(\ln N)}$ و $\frac{\ln N}{N}$ مطابقت دارند.

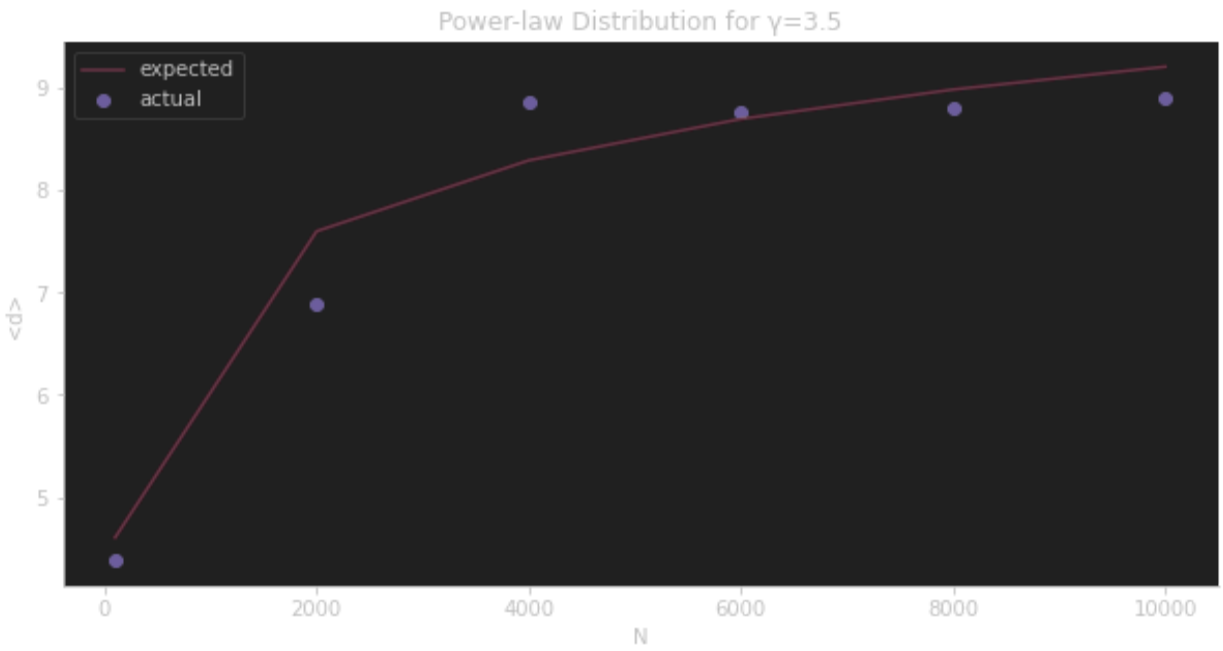
—

با رسم نمودار ها نیز میتوان تا حدودی از درست بودن فرمول ها اطمینان یافت.




Power-law Distribution for γ=2.5
expected
actual
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
<d>
0
2000
4000
6000
8000
10000
N

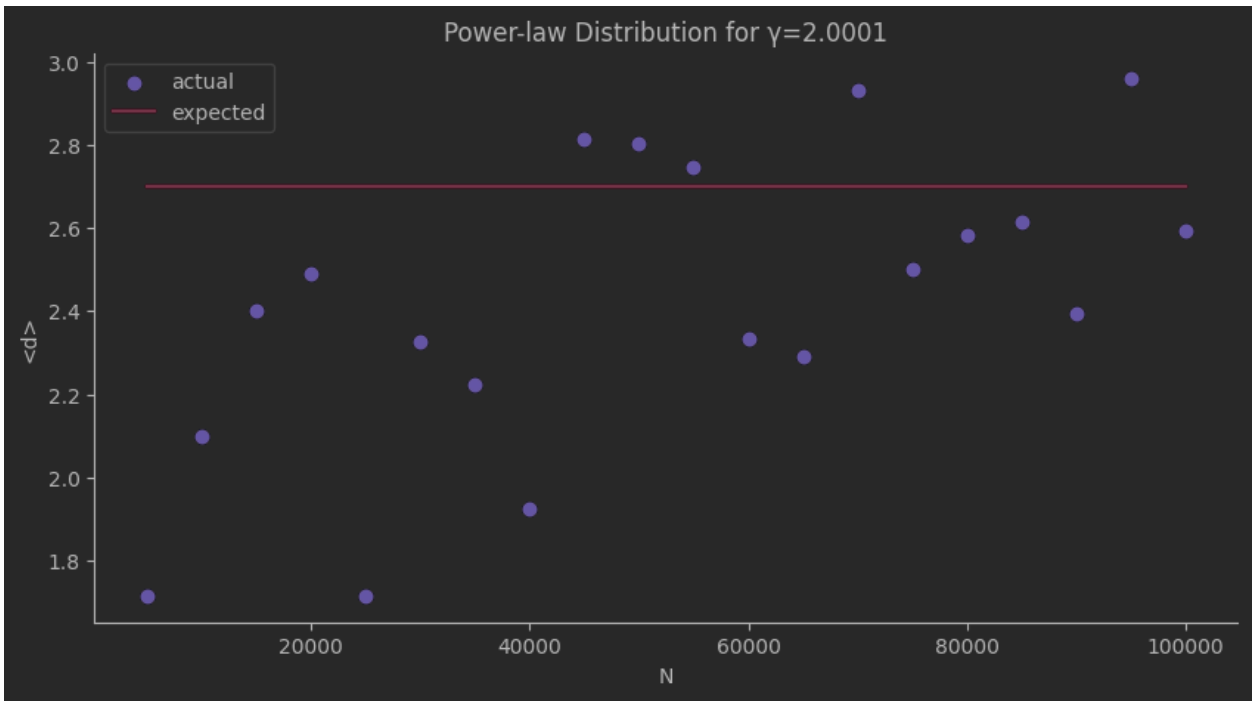

Power-law Distribution for γ=3
expected
actual
7.0
6.5
6.0
5.5
5.0
4.5
4.0
<d>
0
2000
4000
6000
8000
10000
N

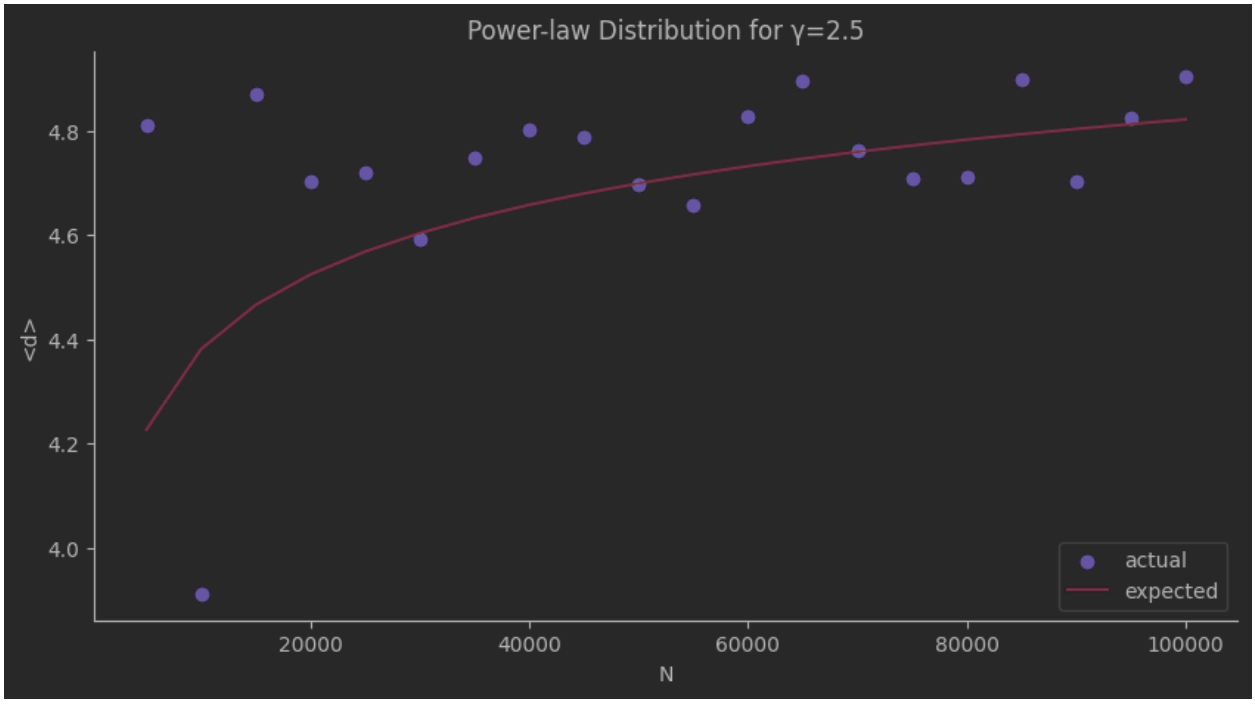


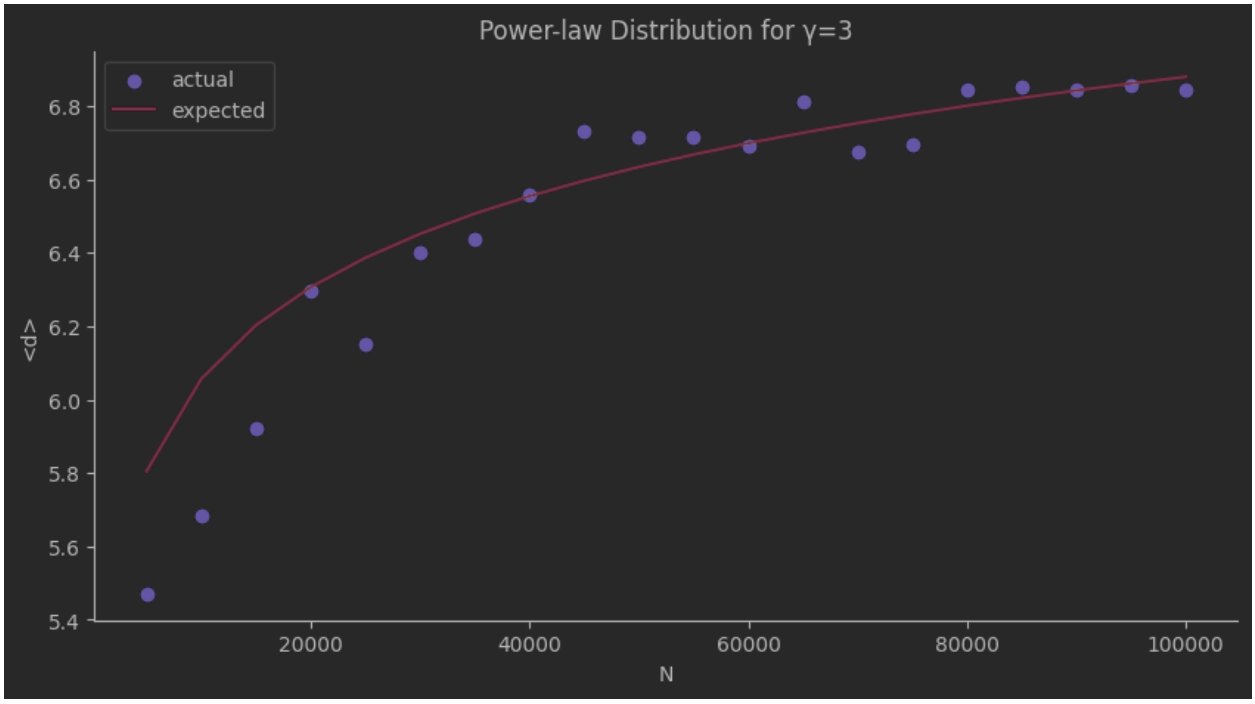
برای اینکه بتوانیم ابعاد بالاتر را بررسی کنیم تابع زیر پیاده سازی شده است که به صورت تصادفی ln(N) جفت گره از گراف را انتخاب کرده و میانگین فاصله را برای آن ها حساب میکند.

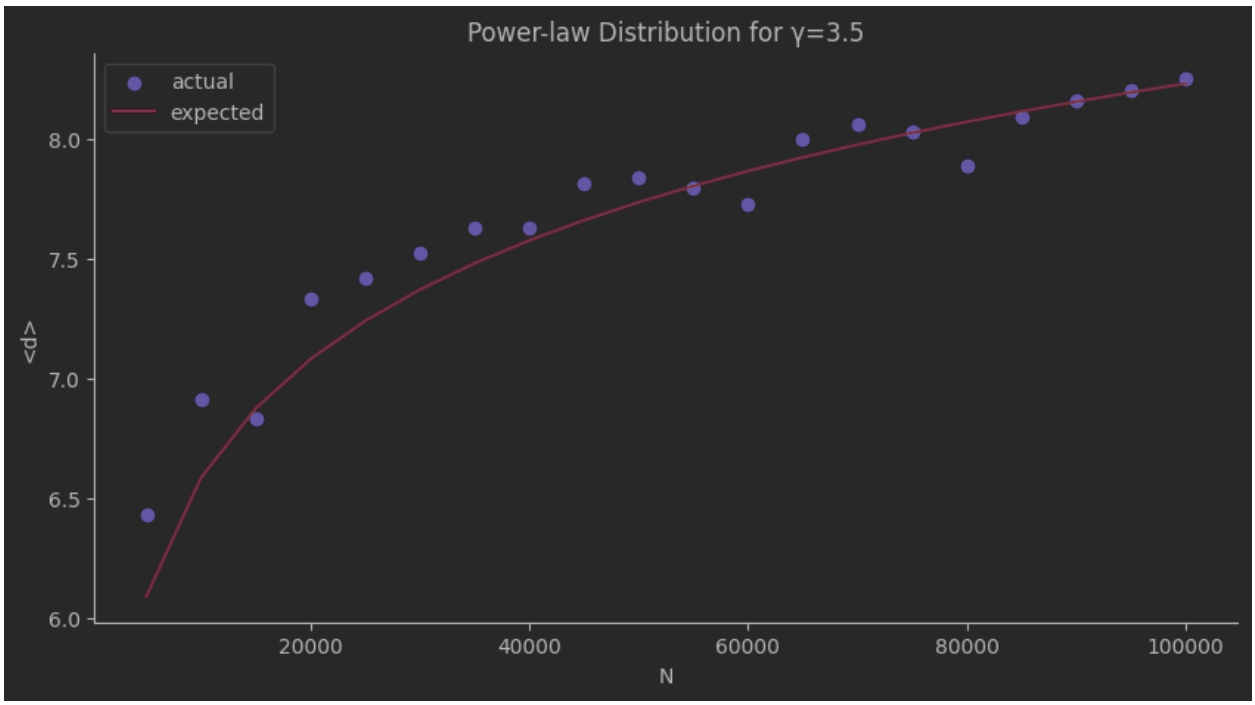
```
def caculate_distance_sampling(graph):
    sampling_size = int(len(graph.nodes) + 1) + 1
    nodes = np.array(graph.nodes)
    d = 0
    for i in range(sampling_size):
        selected_node = np.random.choice(nodes, size=2, replace=True)
        while selected_node[0] == selected_node[1]:
            selected_node = np.random.choice(nodes, size=2, replace=True)
        d += nx.shortest_path_length(graph,
                                     source=selected_node[0],
                                     target=selected_node[1])
    return d / sampling_size
```

در نتایج زیر گراف با ابعداد تا 100000 گره در نظر گرفته شده است.









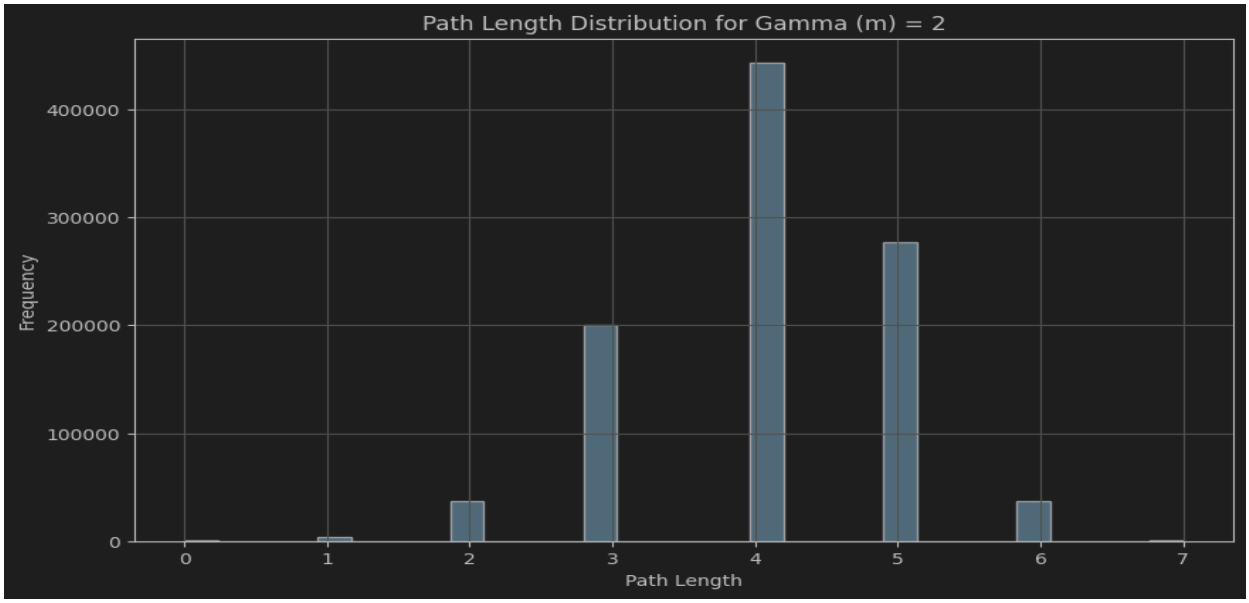
همانطور که مشاهده میکنید در بینهایت نتایج به دست آمده به فرمول های گفته شده میل میکنند.

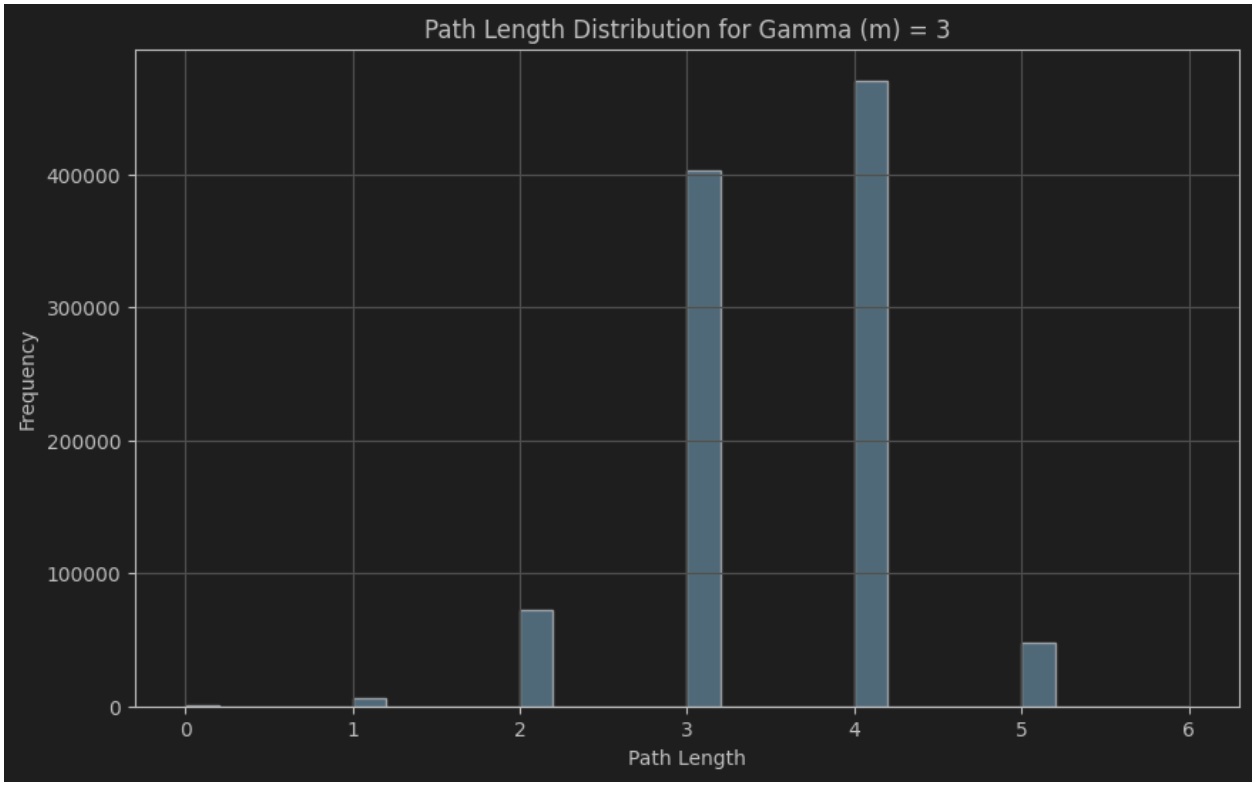
همچنین با استفاده از کد زیر نیز میتوان نشان داد

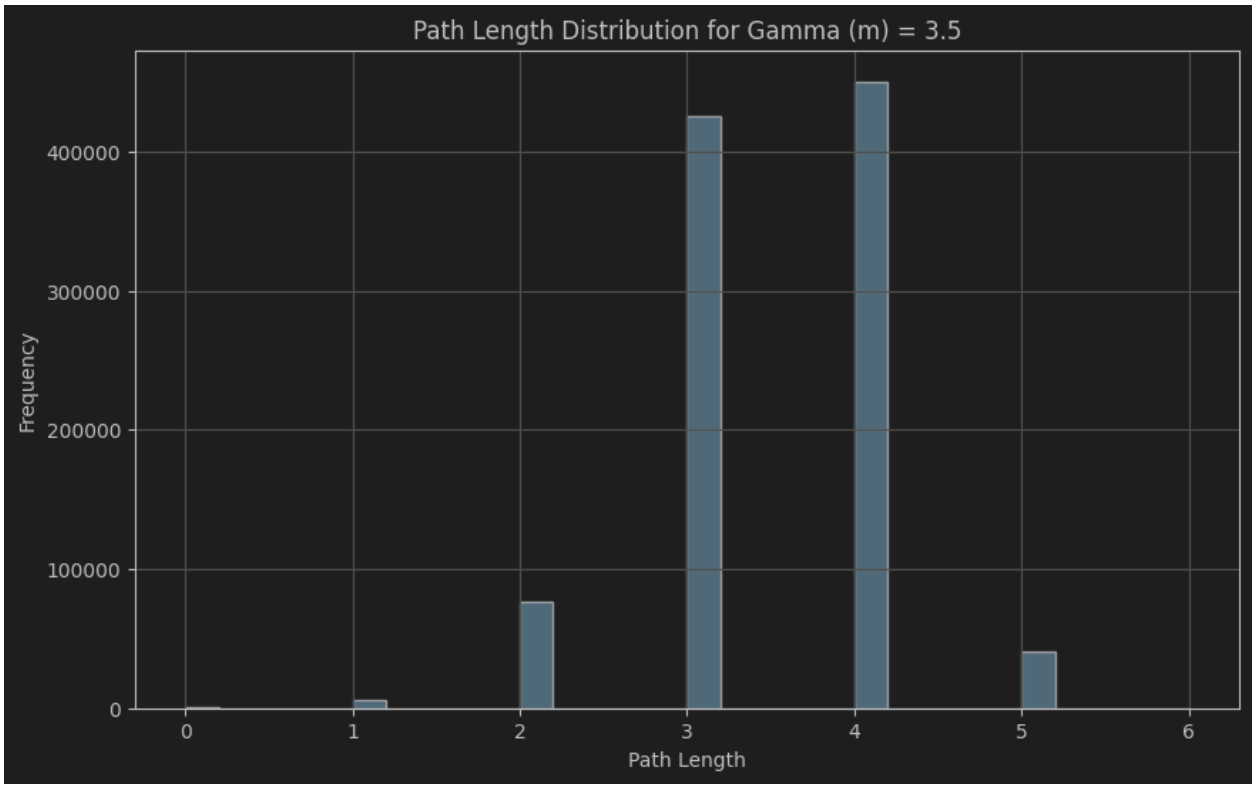
```
def generate_ba_graph(n, m):
      return nx.barabasi_albert_graph(n, m)



def average_shortest_path_length(graph):
   return nx.average_shortest_path_length(graph)



def plot_path_length_distribution(graph, gamma):
   path_lengths = dict(nx.shortest_path_length(graph))
   all_lengths = []

   for node, lengths in path_lengths.items():
       all_lengths.extend(lengths.values())

   plt.figure(figsize=(10, 6))
   plt.hist(all_lengths, bins=30, alpha=0.75, edgecolor='black')
   plt.xlabel('Path Length')
   plt.ylabel('Frequency')
   plt.title(f'Path Length Distribution for Gamma (m) = {gamma}')
   plt.grid(True)
   plt.show()



n = 1000  # Number of nodes
gamma_values = [2, 2.5, 3, 3.5]

for gamma in gamma_values:
   m = int(gamma)
   graph = generate_ba_graph(n, m)
```

```
plot_path_length_distribution(graph, gamma)

avg_path_length = average_shortest_path_length(graph)

print(f'Gamma (m): {gamma}, Average Shortest Path Length: {avg_path_length}')
```

برای هر یک از مقادیر گاما (2، 2.5، 3 و 3.5)، یک گراف با استفاده از مدل Barabási–Albert تولید شده و توزیع فاصله‌های کوتاهترین مسیرها رسم می‌شود. همچنین متوسط فاصله کوتاهترین مسیر برای هر گاما محاسبه و چاپ می‌شود.
نتایج بدست امده از کد بالا به صورت زیر است:







این نمودار ها نشان می‌دهند که با افزایش گاما، توزیع فاصله‌های کوتاهترین مسیر ها چگونه تغییر می‌کند. همچنین متوسط فاصله کوتاهترین مسیر برای هر گاما محاسبه شده .