2.3.1

1. For a Caylay tree with generic parameter k, you find at distance d from the center:

$$k(k-1)^{(d-1)} \text{ nodes.}$$

2. The diameter is the longest shortest path in the network.

In the Caylay tree, the diameter does not depend on k.

$D = 2d$

Therefore, when $d = 8$: $\underline{\underline{D = 16}}$

The total number of nodes is the sum over the number of nodes at a certain distance plus the center node:

$$N = k \sum_{i=0}^{d-1} (k-1)^i + 1$$

$$N = 3 \sum_{i=0}^{7} 2^i + 1 \Rightarrow \underline{\underline{N = 766}}$$

3. For $k = 3$ we have: $N = 3 \sum_{i=0}^{d-1} 2^i + 1$

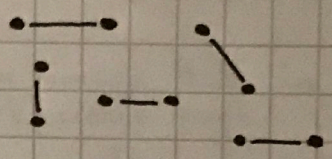
This simplyfies to: $N = 3 \cdot 2^d - 2 \Rightarrow$

$$\frac{N+2}{3} = 2^d \Rightarrow \ln\left(\frac{N+2}{3}\right) = d \cdot \ln(2) \Rightarrow$$

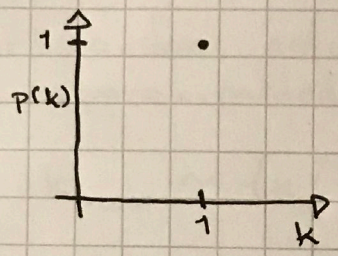
$$d = \frac{\ln\left(\frac{N+2}{3}\right)}{\ln(2)} \Rightarrow \underline{\underline{D = 2 \cdot \frac{\ln\left(\frac{N+2}{3}\right)}{\ln(2)}}}$$

2.4.1

1. N must be an even number, i.e. it is a multiple of 2.

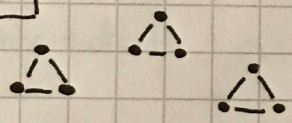


2. $p(k=1) = 1$



3. The network has $\frac{N}{2}$ components.

2.4.2

1. 

2. N is a multiple of 3.

3.2.1

For scale-free networks the maximum degree can be approximated to:

$$k_{max} \sim N^{\frac{1}{(\gamma-1)}}$$

(formula from image 4.5 in the network science book) — this is derived from equation 4.17 and 4.18

For random networks the maximum degree can be approximated to:

$$k_{max} \sim \ln(N)$$

(formula from image 4.5 in the network science book)

Two networks from table 4.1 are [redacted] considered scale-free.

	N	γ	k_{max}
Actor network	702,388	2.12	166.019
Protein interact.	2,018	2.89	56

To the extend that the internet and science collaboration network can be considered scale-free, we approximate their k_{max} to:

	N	γ	k_{max}
Internet	192,244	3.42	153
Science colab.	23,133	3.35	72

The last network is not scale-free. We estimate this to a random network

	N	k_{max}
Power Grid	4,941	$8.5 \approx 9$

3.4.1

From equation 3.3 in the network science book we find that:

$$\langle k \rangle = \frac{2\langle L \rangle}{N} = p(N-1)$$

(i) $\langle k \rangle = 0.01(1000-1) = 9.99$

(ii) $\langle k \rangle = 0.1(1000-1) = 99.9$

(iii) $\langle k \rangle = 1000 - 1 = 999$

In the last case the network is called a complete network / graph

Network Analysis | Exercises | Week 3

3.4.2

$N = 10,000$ and $\langle k \rangle = 2$

To find the number of links we expect to find we isolate L in the formula for average degree.

$$\langle k \rangle = \frac{2L}{N} \Rightarrow \frac{\langle k \rangle \cdot N}{2} = L$$

$$L = \frac{2 \cdot 10,000}{2} \Rightarrow L = 10,000$$

We expect to find the same number of links as the number of nodes in this case.

To find the p used to generate the network, we isolate p in the other formula for average degree. seen in exercise 3.4.1:

$$\langle k \rangle = p(N-1) \Rightarrow p = \frac{\langle k \rangle}{(N-1)}$$

$$p = \frac{2}{(10,000-1)} \Rightarrow p = 0.0002$$

Network Analysis | Exercises | Week 3

3.5

A poisson distribution is as follows:

$$P_k = e^{-\langle k \rangle} \cdot \frac{\langle k \rangle^k}{k!}$$

A network with degree distribution:

$$P_k = e^{-3} \cdot \frac{3^k}{k!}$$

has an average degree of: $\underline{\underline{\langle k \rangle = 3}}$

4.3: Mastering power law distributions

Write code to generate three synthetic networks having 10,000 nodes and degree distribution following a power-law with exponent γ = 2.2, γ = 2.5, and γ = 3. After generating the networks, fit the three obtained distributions using the tools covered in class. Is there an agreement between the result of the fitting and what you would expect? Why or why not?

In [1]:

```
# imports

import networkx as nx
from matplotlib import pyplot as plt
import powerlaw
from collections import Counter
from tqdm import tqdm
import random
import warnings
warnings.filterwarnings("ignore")
```

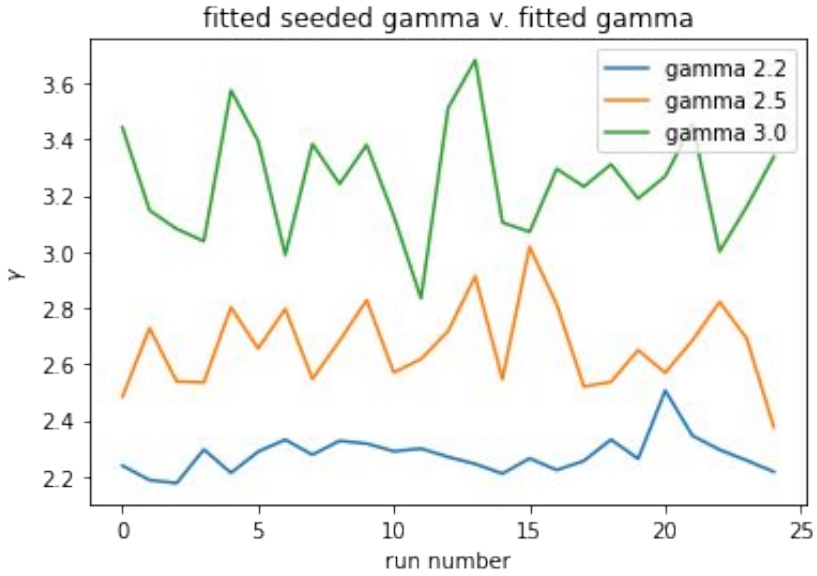
In [29]:

```
n = 10 ** 4                                 # 10,000 nodes
alphas = [2.2, 2.5, 3.0]                    # the three given gammas
_2_2, _2_5, _3_0 = [], [], []               # arrays to hold fitted gammas ("alpha"
in powerlaw library)

def run():
    """
    Function generates a graph with each of the provided gammas.
    Then it determines gamma through fitting and appends to the
    corresponding gamma array.
    """
    for i in range(len(alphas)):
        degrees = nx.utils.powerlaw_sequence(n, alphas[i])
        degrees = [int(d) for d in degrees]
        if sum(degrees) % 2 != 0:
            degrees[-1] += random.choice([1, -1])
        G = nx.configuration_model(degrees)
        #G = nx.Graph(G)
        fit = powerlaw.Fit(list(dict(G.degree()).values()), verbose=False)
        if i % 3 == 0:
            _2_2.append(fit.alpha)
        elif i % 3 == 1:
            _2_5.append(fit.alpha)
        else:
            _3_0.append(fit.alpha)
```

In [30]:

```
"""
We run the experiement above 10 times, and plot the outcome.
"""
random.seed(42)
for i in tqdm(range(25)):
    run()


plt.plot(_2_2, label='gamma 2.2')
plt.plot(_2_5, label='gamma 2.5')
plt.plot(_3_0, label='gamma 3.0')
plt.legend()
plt.title('fitted seeded gamma v. fitted gamma')
plt.ylabel(r'$\gamma$')
plt.xlabel('run number')
plt.show()
```

```
100%|██████████| 25/25 [00:14<00:00,  1.72it/s]
```



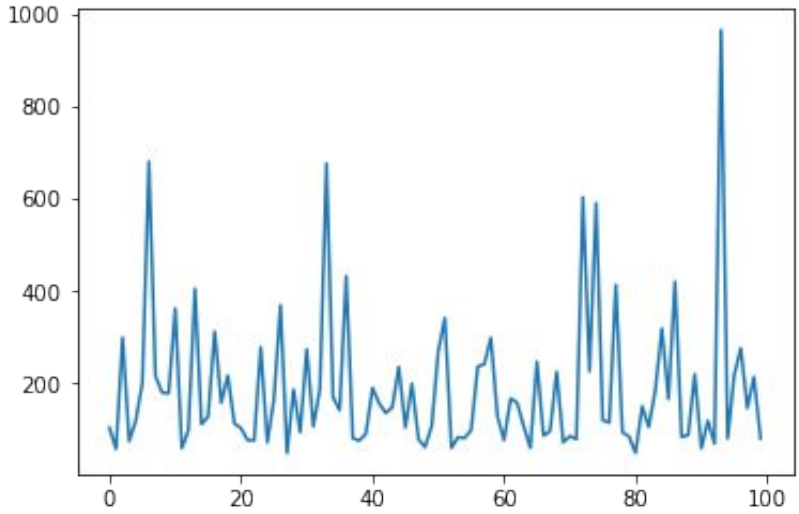
Though the graphs above are clearly oscilating in the vicinity of their corresponding originial γ's, they do seem to reliably overshoot.

One explanation could be that γ is largely determined by how big the largest values are relative to the length of the array. This possibility is echoed by variance of the largest degree in the genereated sequences below.

What is intersting here is that the most common largest degree is 300-ish, though when it does deviate significantly, it deviates upwards.

In [31]:

```
random.seed(42)
degrees = []
for i in range(100):
    degrees.append(max(nx.utils.powerlaw_sequence(10 ** 4, 3)))
plt.plot(degrees);
```



5.1 Generating Barabási-Albert (BA) networks

Write code to generate a network with $N = 10^4$ nodes using the BA model with $m = 4$. Use as initial condition a fully connected network with $m = 4$ nodes.

In [1]:

```
# imports

import networkx as nx
import random
import numpy as np
import pandas as pd
from tqdm import tqdm
import powerlaw
from matplotlib import pyplot as plt
import matplotlib; matplotlib.rcParams["figure.dpi"] = 100
import warnings
```

We perform the setup described above

In [2]:

```
alphas = []
n = 10 ** 4
N = {i : [j for j in range(1, 5)] for i in range(1, 5)}
D = []
random.seed(42)

for node in tqdm(range(5, n + 1)):
    samples = [k for k in N.keys()]
    weights = [len(v) for v in N.values()]
    choices = random.choices(samples, weights, k=4)
    N[node] = choices
    for choice in choices:
        N[choice].append(node)
    if node in (10 ** 2, 10 ** 3, 10 ** 4):
        D.append([len(v) for v in N.values()])
```

```
100%|██████████| 9996/9996 [00:06<00:00, 1440.49it/s]
```

5.1.1

Measure the degree distribution at intermediate steps, namely when the network has 10^2, 10^3 and 10^4 nodes.

In [10]:

```
for i in range(len(D)):
    df = pd.DataFrame(D[i])
    print(df.describe())
```

```
                0
count  100.000000
mean     7.840000
std      6.536177
min      4.000000
25%      4.000000
50%      5.500000
75%      8.000000
max     40.000000
                0
count  1000.00000
mean      7.98400
std       8.98713
min       4.00000
25%       4.00000
50%       5.00000
75%       8.00000
max     115.00000
                  0
count  10000.000000
mean       7.998400
std       11.113575
min        4.000000
25%        4.000000
50%        5.000000
75%        8.000000
max      362.000000
```

5.1.{2, 3}

5.1.2:

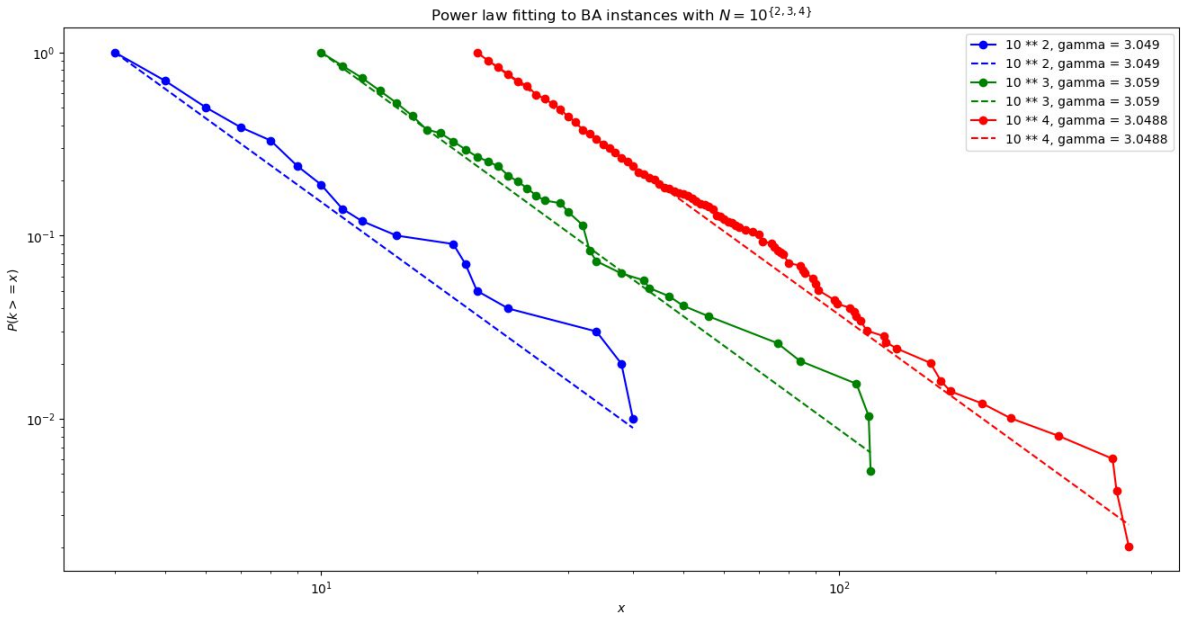
Compare the distributions at these intermediate steps by plotting them together and fitting each to a power-law with degree exponent γ. Do the distributions “converge”? Do you find the exponent you would expect?

5.1.3:

Measure the average clustering coefficient in function of N.

In [16]:

```
# 5.1.2

fig, axis = plt.subplots(1, 1, figsize=(16, 8))
warnings.filterwarnings("ignore")
colors = ['r', 'g', 'b'][::-1]
labels = ['10 ** 2', '10 ** 3', '10 ** 4']
for i in range(len(D)):
    # frq = [d / max(D[i]) for d in D[i]]
    frq = D[i]
    fit = powerlaw.Fit(frq, ax=axis, verbose=False)
    fit.plot_ccdf(ax=axis, marker='o', color=colors[i], label=f'{labels[i]}, g
amma = {round(fit.alpha, 4)}')
    fit.power_law.plot_ccdf(ax=axis, linestyle='--', color=colors[i], label=f'
{labels[i]}, gamma = {round(fit.alpha, 4)}')
    alphas.append(fit.alpha)
plt.title('Power law fitting to BA instances with $N = 10^{\{2, 3, 4\}}$')
plt.xlabel(r'$x$'); plt.ylabel(r'$P(k >= x)$'); plt.legend()
plt.show()
```



The above plot is as we would expect, with each line being offset by one order of magnitude, and paralel to one another. As for problem 4.3, `powerlaw` slightly overestimates γ.

In [ ]:

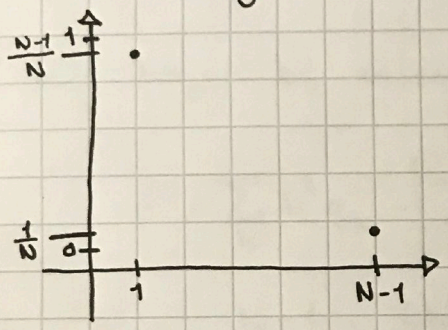
```

```

Network Analysis | Exercises | Week 6

6.2.1

We assume that the exercise description means that one node is connected to all other nodes, and that all other nodes are only connected to this hub.

The degree distribution is:



6.2.2

Generally, the probability of finding a node with degree k is:

$$q_k = \frac{N_k}{N}$$

For $k = 1$: $q_1 = \frac{N-1}{N}$

For $k = N-1$: $q_{N-1} = \frac{1}{N}$

Network Analysis | Exercises | Week 6

6.2.3

The degree correlation coefficient is:

$$r = \sum_{jk} \frac{jk(e_{jk} - q_j q_k)}{\sigma^2}$$

We begin by calculating σ^2:

$$\sigma^2 = \sum_k k^2 q_k - \left[\sum_k k q_k\right]^2$$

$$\sigma^2 = \frac{N-1}{N} + \frac{(N-1)^2}{N} - \left[\frac{N-1}{N} + \frac{N-1}{N}\right]^2 \Rightarrow$$

$$\sigma^2 = \frac{N-1}{N} + \frac{(N-1)^2}{N} - \frac{4(N-1)^2}{N^2} \Rightarrow$$

$$\sigma^2 = \frac{(N-1)+(N-1)^2}{N} - \frac{4(N-1)^2}{N^2} \Rightarrow$$

$$\sigma^2 = \frac{N(N-1)+N(N-1)^2}{N^2} - \frac{4(N-1)^2}{N^2} \Rightarrow$$

$$\sigma^2 = \frac{N(N-1)(1+N-1)}{N^2} - \frac{4(N-1)^2}{N^2} \Rightarrow$$

$$\sigma^2 = \frac{N^2(N-1)}{N^2} - \frac{4(N-1)^2}{N^2} \Rightarrow \sigma^2 = N - 1 - 4\underbrace{\frac{(N-1)^2}{N^2}}$$

This term goes to 1 as N grows large.

For $N \gg 1$ $\sigma^2 = N - 5$

Network Analysis | Exercises | Week 6

6.2.3 continued...

Next we calculate the degree correlation coefficient:

$$r = \frac{1^2\left(0 - \left(\frac{N-1}{N}\right)^2\right) + 2\cdot(N-1)\left(\frac{1}{2} - \frac{N-1}{N^2}\right) + (N-1)^2\left(0 - \frac{1}{N^2}\right)}{N-5} \Rightarrow$$

$$r = \frac{-\left(\frac{N-1}{N}\right)^2 + N - 1 - 2\left(\frac{N-1}{N}\right)^2 - \left(\frac{N-1}{N}\right)^2}{N-5} \Rightarrow$$

$$r = \frac{N - 1 - 4\left(\frac{N-1}{N}\right)^2}{N-5}$$

} The last term again goes to 1 as N grows large.

For $N \gg 1$:

$$r \approx \frac{N-5}{N-5} \Rightarrow \underline{\underline{r \approx 1}}$$

6.2.4

For $-1 \leq r < 0$ a network is assortative, for $r = 0$ a network is neutral and for $0 < r \leq 1$ a network is disassortative.

Since this network's degree correlation coefficient goes to $r = 1$ the network is disassortative.

8.1

	N	Random (real netw.)	Random (randomized netw.)
Internet	192,244	0.92	0.84
Power Grid	4,941	0.61	0.63
Science Cdlab.	23,133	0.92	0.88
Actor network.	702,388	0.98	0.99
Protein interact.	2,018	0.88	0.66

These numbers are a combination of table 4.1 and table 8.1

The numbers in column 3 is the critical threshold, f_c, for the real network. The numbers in column 4 is the critical threshold for the randomized network (same number of links and nodes). $f_c^{ER} = 1 - \frac{1}{\langle k \rangle}$

The number of nodes that need to randomly fail to break the networks are:

	Number of nodes to randomly fail in real network	Number of nodes to randomly fail in randomized network
Internet	176,864	161,485
PowerGrid	3,014	3,113
Science Collab.	21,282	20,357
Actor Network	688,340	695,364
Protein interact.	1,776	1,332

9.1: Projection

Calculate the unipartite projections using simple weights (function jaccard) and random walks (function ycn edges). Convert the outputs to two networkx Graphs: G_sw and G_rw – make sure to also transfer the edge weights. Calculate their densities.

In [2]:

```
# imports
import sys; sys.path.append('bin/')
import pandas as pd
import networkx as nx
import network_map2 as nm2
from matplotlib import pyplot as plt
```

In [27]:

```
G = nx.Graph()

with open('../data/other/Davis_southern_club_women-two_mode.txt', 'r') as data
file:
    data = datafile.readlines()
    for edge in data:
        source, target = edge.strip().split(" ")
        G.add_edge(source, f"E{target}")

ed = len(data) / ((1/2) * (18 * 17))
print("Edge density : ", round(ed, 3))
```

```
Edge density :  0.582
```

In [4]:

```
nodes = nx.algorithms.bipartite.basic.sets(G)
rows = sorted(list(nodes[1]))
cols = sorted(list(nodes[0]))
```

In [5]:

```
G_sw = nm2.jaccard(G, cols)
#print(nx.info(G_sw))
edges = [(s, t, w) for (s, t, w) in G_sw.edges(data=True)]
G_sw = nx.Graph()
G_sw.add_weighted_edges_from(edges)
```

```
/Users/syrkis/opt/anaconda3/lib/python3.8/site-packages/sklearn/me
trics/pairwise.py:1765: DataConversionWarning: Data was converted
to boolean for metric jaccard
  warnings.warn(msg, DataConversionWarning)
```

In [6]:

```
G_rw = nm2.ycn(G, cols)
#print(nx.info(G_sw))
edges = [(s, t, w) for (s, t, w) in G_rw.edges(data=True)]
G_rw = nx.Graph()
G_rw.add_weighted_edges_from(edges)
```

We now compute the proporion of edges compard to all possible edges (their densities)

In [26]:

```
density = lambda x : len(x.edges) / ((len(rows)) * (len(cols) - 1)) #

d1 = density(G_rw)
d2 = density(G_sw)

print("Edge desnity Jaccard : \t", round(d1, 2))
print("Edge desnity YCN :\t", round(d2, 2))
```

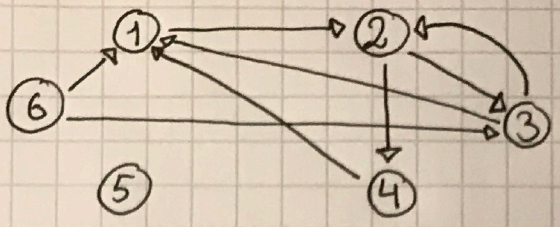
```
Edge desnity Jaccard :   0.58
Edge desnity YCN :       0.58
```

Thus we see that the edge densities are the same as before to within two decimals, regardless of the form projection.

In [ ]:

```

```

13.1



The adjacency matrix is:

$$A = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The transition matrix, M, is:

$$M = \begin{bmatrix} 0 & 0 & \frac{1}{2} & 1 & 0 & \frac{1}{2} \\ 1 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The iterative formula for calculating pagerank is:

$$v' = \beta M v + (1-\beta)e/n$$

see chapter 5 page 186 from lecture 13 reading material.

Case $\beta = 0$

The random surfer will always pick a new page with equal probability, because only the random surfer term $(1-\beta)e/n$ remains.
Thus, pagerank of node 5 is $\frac{1}{6}$

Case $\beta = 0.15$

Since row 5 in the transition matrix only consists of 0's $\beta M v$ doesn't contribute to the pagerank of node 5.

Therefore, after the first iteration the pagerank of node 5 stabilises at:

$$\frac{1-0.15}{6} = 0.142$$

Conclusion: In both cases the pagerank is greater than 0