Relatório A2 Capítulo 3

José Guilherme De Oliveira Antunes

June 2024

Introdução

O intuito deste trabalho é analisar as redes e os métodos descritivos de redes abordados durante a disciplina de introdução à modelagem matemática. O desenvolvimento do documento será dividido em 2 partes, consistindo na Parte 1, subdividida em duas subseções distintas A e B abordadando geração e crescimento de redes, respectivamente, e na Parte 2, focada na análise de redes com métricas específicas.Todos os códigos deste trabalho foram feitos em python com a biblioteca Networkx, embora não tenham sido utilizadas as funções de geração e crescimento nativas dessa aplicação. Já na parte 2, foram utilizadas as funções típicas do módulo para se obter as medidas desejadas.

Parte 1: Geração e crescimento de redes

Parte A: Geração de redes

O primeiro modelo abordado é o de Erdös-Rényi, um método de geração de redes estocásticas simples, onde, para cada subconjunto de 2 nós distintos, há uma probalilidade p de surgir uma aresta entre eles. A implementação de tal modelo é dada abaixo:

```
def modelo_erdos_renyi(qtd_nos, p_aresta):
# Criando o grafo
G = nx.Graph()

# Adicionando os nós ao grafo
G.add_nodes_from(range(qtd_nos))

# Varrendo todos os nós e estabelecendo
# conexões entre eles usando a probabilidade
for i in range(qtd_nos):
    for j in range(i+1, qtd_nos):
        # Geramos um valor aleatório entre 0 e 1.
        Se ele for menor que a probabilidade
        # passada, então cria-se a aresta, caso contrário,
        # não ocorre nada
        if np.random.rand() < p_aresta:
            G.add_edge(i, j)
return G
```

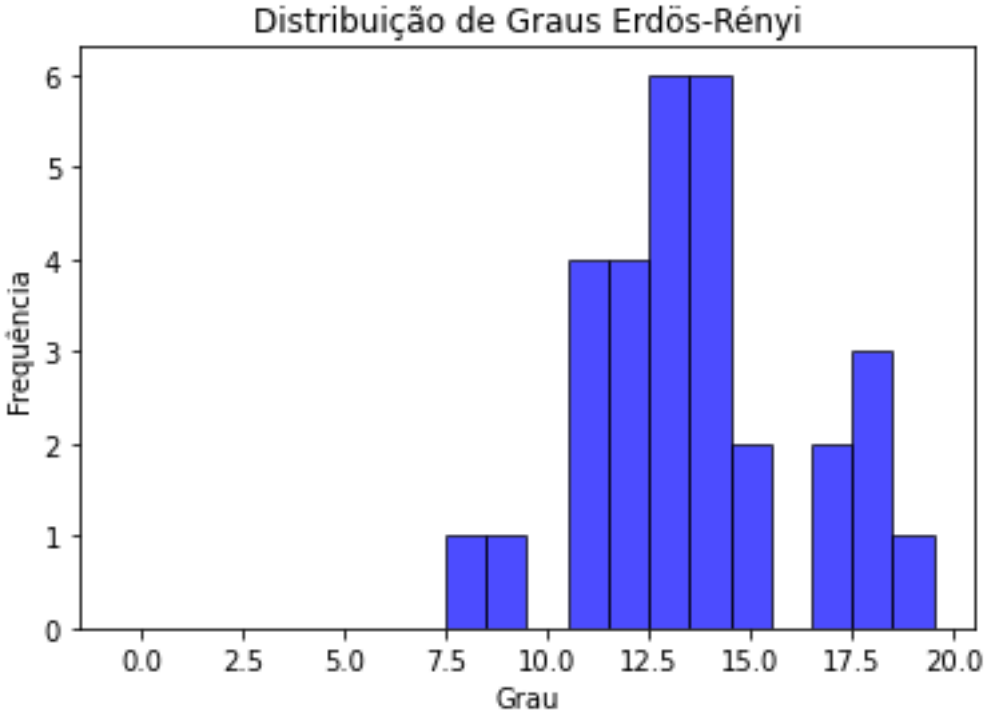
Já o segundo modelo, de Watts-Strogatz, segue uma abordadem diferente. Embora também seja estocástico, neste temos um grafo anelado previamente formado e aleatóriamente as arestas são reconectadas com outros nós da rede. Seu código é:

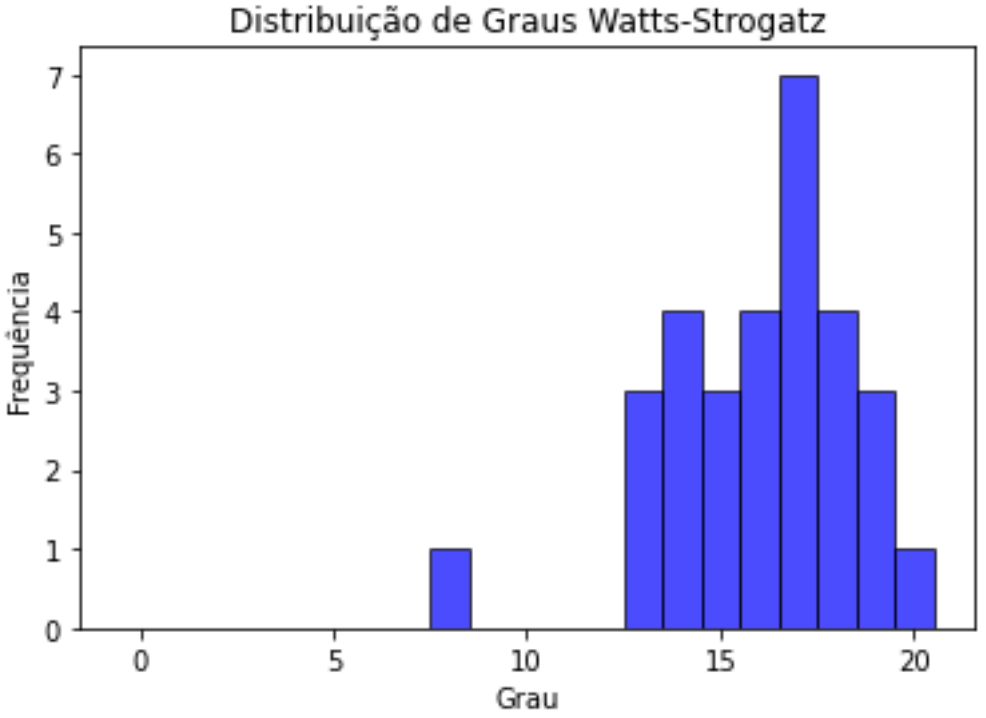
```
def modelo_watts_strogatz(qtd_nos, qtd_vizinhos, p_redirecionamento):
# Criando o grafo
G = nx.Graph()

# Adicionando os nós ao grafo
G.add_nodes_from(range(qtd_nos))

# Conectando com os vizinhos mais próximos
for node in (range(qtd_nos)):
    for i in range(1, qtd_vizinhos//2 + 1):
        vizinho = (node + i) % qtd_nos
        G.add_edge(node, vizinho)
        vizinho = (node - i) % qtd_nos
        G.add_edge(node, vizinho)

# Reordena arestas com a probabilidade
arestas = list(G.edges())
for aresta in arestas:
    if np.random.rand() < p_redirecionamento:
       no_1 = aresta[0]
       no_2 = aresta[1]
       G.remove_edge(no_1, no_2)
       # ligando ao novo no
       novo_no = random.choice(list(set(range(qtd_nos)) -{no_1} -set(G.neighbors(no_1))))
       G.add_edge(no_1, novo_no)
return G
```

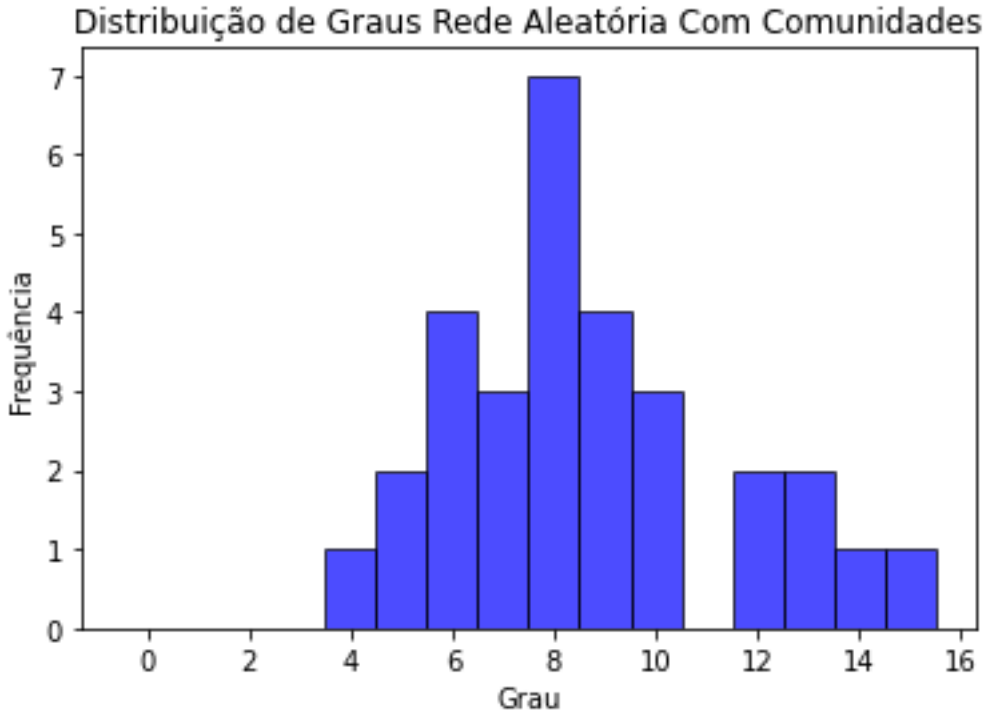
E, por fim, o último modelo abordado é o de redes aleatórias com comunidades. Semelhante ao primeiro, nele arestas são estabelecidas aleatóriamente entre os nós, mas, diferente do outro, existem diferentes categorias de nós e cada uma possui uma probabilidade diferente de estabelecer conexões uma com a outra, além de que cada uma possui uma probabilidade diferente de estabelecer conexões entre nós de uma mesma categoria. Implementando-a, tem-se:

```
def rede_aleatoria_comunidade(nos_por_comunidade, matriz_de_probabilidade):
# Criando o grafo
G = nx.Graph()

# Adicionando nós e os atribuindo a uma comunidade
comunidade = {}
contagem_nos = 0
num_comunidades = len(nos_por_comunidade)
for indice_comunidade in range(num_comunidades):
    for _ in range(nos_por_comunidade[indice_comunidade]):
        G.add_node(contagem_nos)
        comunidade[contagem_nos] = indice_comunidade
        contagem_nos = contagem_nos + 1

# Conectando nós entre comunidades e dentro das comunidades
nos = list(G.nodes())
for i in range(len(nos)):
    for j in range(i + 1, len(nos)):
        comunidade_i = comunidade[nos[i]]
        comunidade_j = comunidade[nos[j]]
        if random.random() < matriz_de_probabilidade[comunidade_i, comunidade_j]:
            G.add_edge(nos[i], nos[j])
return G, comunidade
```

Comparando os três modelos, pode-se observar algumas caracteristicas interessantes, como por exemplo que o modelo de Watts-Strogatz tende a ter uma distribuição de graus mais afunilada que os outros modelos, que aparentemente são mais distribuidos. Outro ponto que chama a antenção é que embora os modelos de Erdös-Rényi e de rede aleatória com comunidade sejam mais esparçamente distribuidos que o de Watts-Strogatz, o de comunidades é mais deslocado a esquerda na média, isto é, possui graus
menores que a rede de Erdös como pode ser visto nos gráficos abaixo:


Distribuição de Graus Erdös-Rényi
Frequência
Grau
0
1
2
3
4
5
6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Distribuição de Graus Watts-Strogatz
Frequência
Grau
0
1
2
3
4
5
6
7
0
5
10
15
20



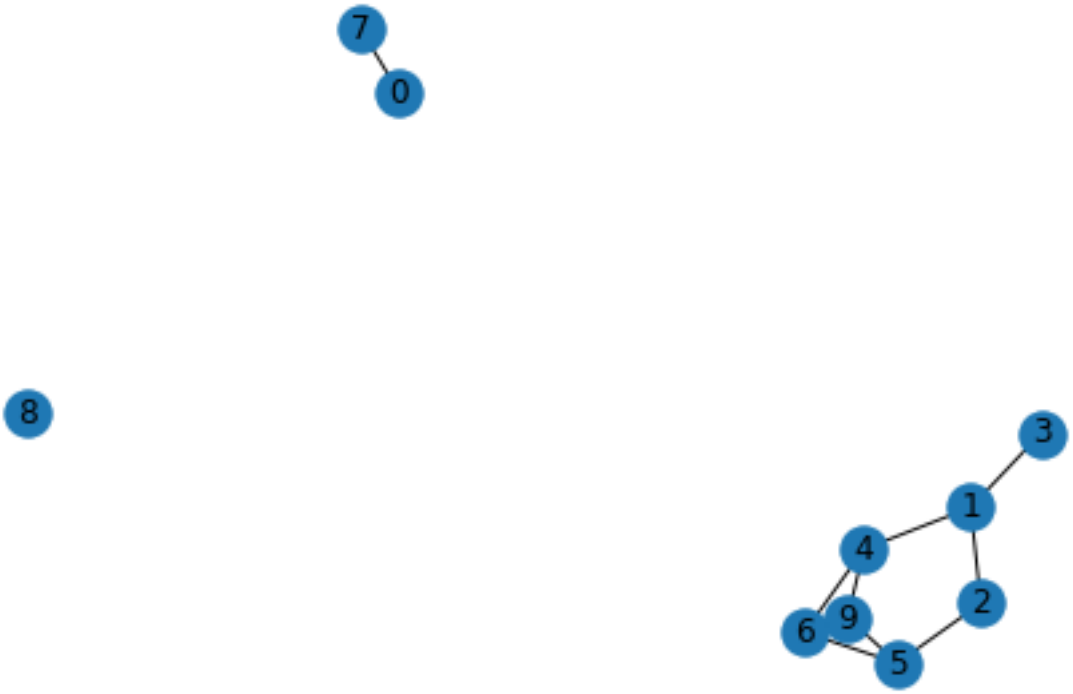
Na questão de medidas tiradas em cima das redes, a clusterização do modelo de Strogatz é a maior, o que já era esperado pelo modo que esse método é executado(triângulos são formados devido a geração da cíclica previa). Quanto ao grau médio e a média dos shortest paths, a análise parece indicar que para probabilidades parecidas, os 3 modelos tem medidas semelhantes, embora o de redes com comunidade seja sensível a grandes mudanças devido as probabilidades utilizadas em sua matriz.

```
 Grau de conectividade médio Erdös-Rényi:
14.361367109651423
Clusterização média Erdös-Rényi:
0.47123610594198845
Média dos shortest path Erdös-Rényi:
1.5218390804597701
Grau de conectividade médio Watts-Strogatz:
16.38824123935347
Clusterização média Watts-Strogatz:
0.5559877709722909
Média dos shortest path Watts-Strogatz:
1.4482758620689655
Grau de conectividade médio rede aleatória com comunidade:
8.154940702301813
Clusterização média rede aleatória com comunidade:
0.26274521774521775
Média dos shortest path rede aleatória com comunidade:
1.8666666666666667
```

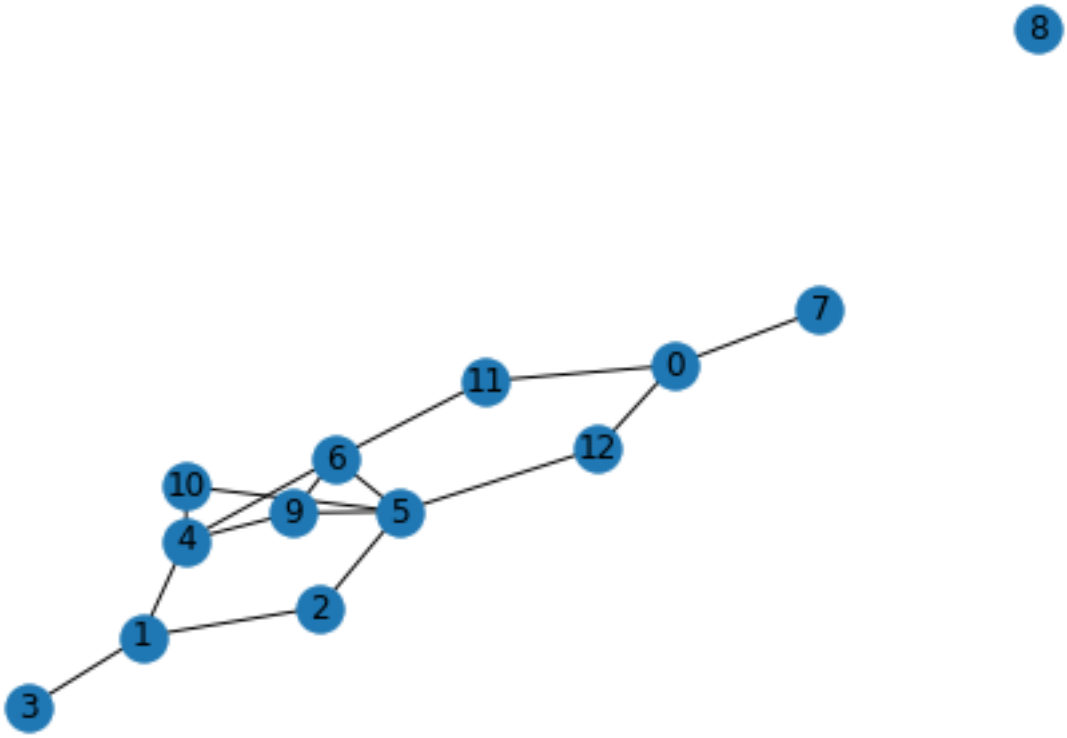
Parte B: Crescimento de redes

Quanto ao crescimento da rede, 3 modelos foram testados. O primeiro é o modelo de anexação uniforme. Neste método, cada novo nó é conectado a outros nós previamente existentes da rede aleatóriamente de modo equiprovável para cada nó. No segundo modelo abordado, de anexação preferencial, nós com grau maior possuem uma probabilidade maior de serem conectados, o que favorece o fechamento de redes em comunidades grandes, como pode ser visto abaixo:

Comparação do crescimento com anexação uniforme



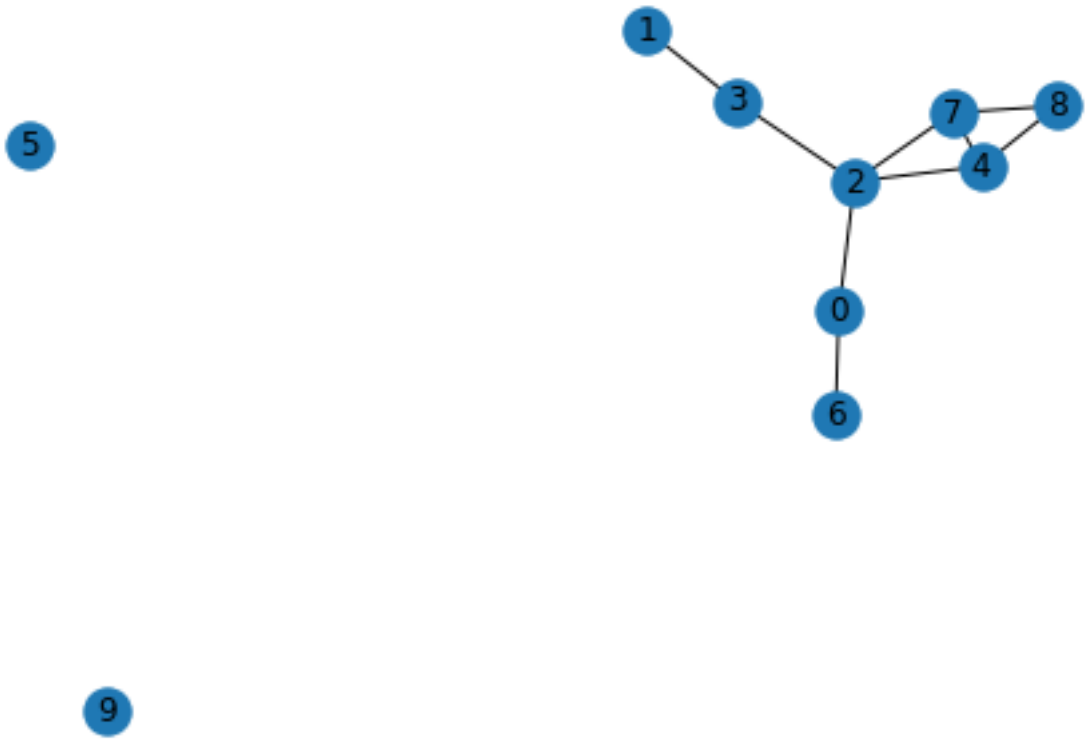
Acima a rede antes



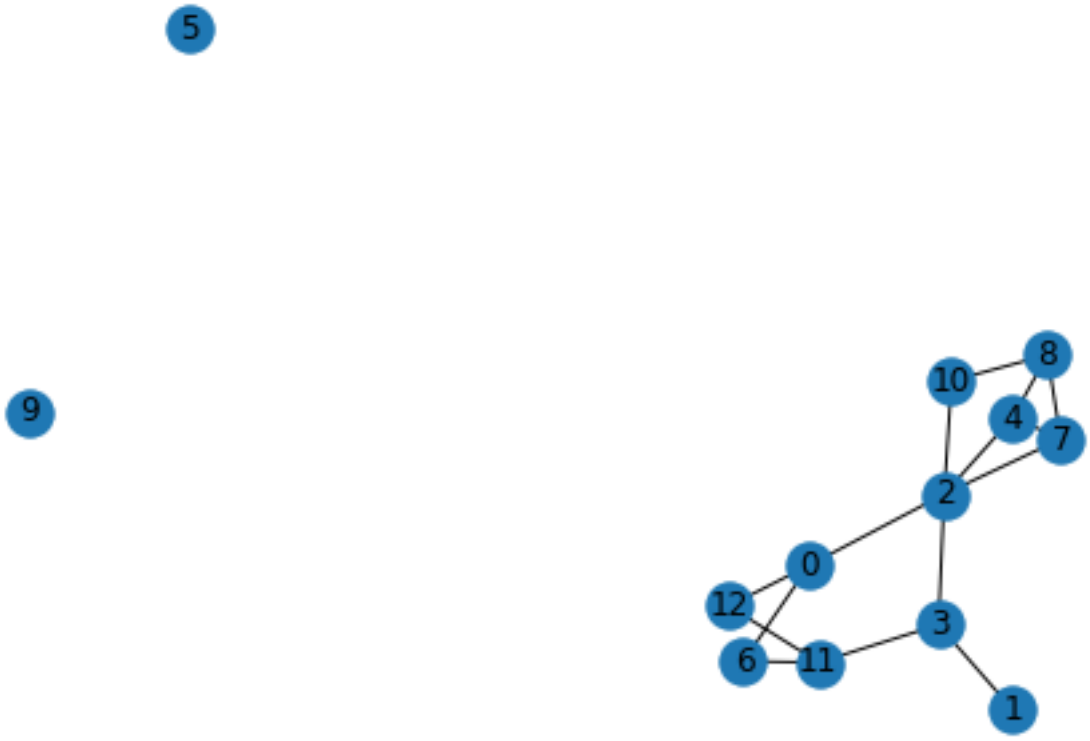
Acima a rede depois

Pode-se observar que nessa rede haviam nós desconectados que se conectaram.

Comparação do crescimento com anexação preferencial



Acima a rede antes



Acima a rede depois

Como pode ser visto, o crescimento com o método de anexação preferencial é fechado dentro daquele ciclo de nós já feito, pois eles são mais prováveis de receberem arestas com novos nós. Segue abaixo o código de implementação dos dois:

```
def anexacao_uniforme(G, num_novos_nos, num_ligacoes):
num_nos_existentes = len(G.nodes)

for i in range(num_novos_nos):
    novo_no = num_nos_existentes + i
    G.add_node(novo_no)

    # Selecionar nós existentes para se conectar com o novo nó
    nos_existentes = list(G.nodes)
    nos_existentes.remove(novo_no)

    # Selecionar ‘num_ligacoes‘ nós de forma uniforme
    nos_para_conectar = random.sample(nos_existentes, num_ligacoes)
```

```
    # Adicionar arestas entre o novo nó e os nós selecionados
    for no in nos_para_conectar:
        G.add_edge(novo_no, no)


def anexacao_preferencial(G, num_novos_nos, num_ligacoes):
num_nos_existentes = len(G.nodes)

for i in range(num_novos_nos):
    novo_no = num_nos_existentes + i
    G.add_node(novo_no)

    # Obter a lista de nós existentes e suas respectivas quantidades de ligações
    nos_existentes = list(G.nodes)
    nos_existentes.remove(novo_no)
    graus = [G.degree(no) for no in nos_existentes]

    # Selecionar nós existentes com probabilidade proporcional ao grau
    nos_para_conectar = random.choices(nos_existentes, weights=graus, k=num_ligacoes)

    # Adicionar arestas entre o novo nó e os nós selecionados
    for no in nos_para_conectar:
        G.add_edge(novo_no, no)
```

Já o modelo de price é uma espécie de mistura dos dois modelos anteriores, ou seja, uma parcela da ligação entre novos nos segue uma anexação uniforme e outra parcela segue um modelo preferencial. Abaixo o código da implementação desse modelo:

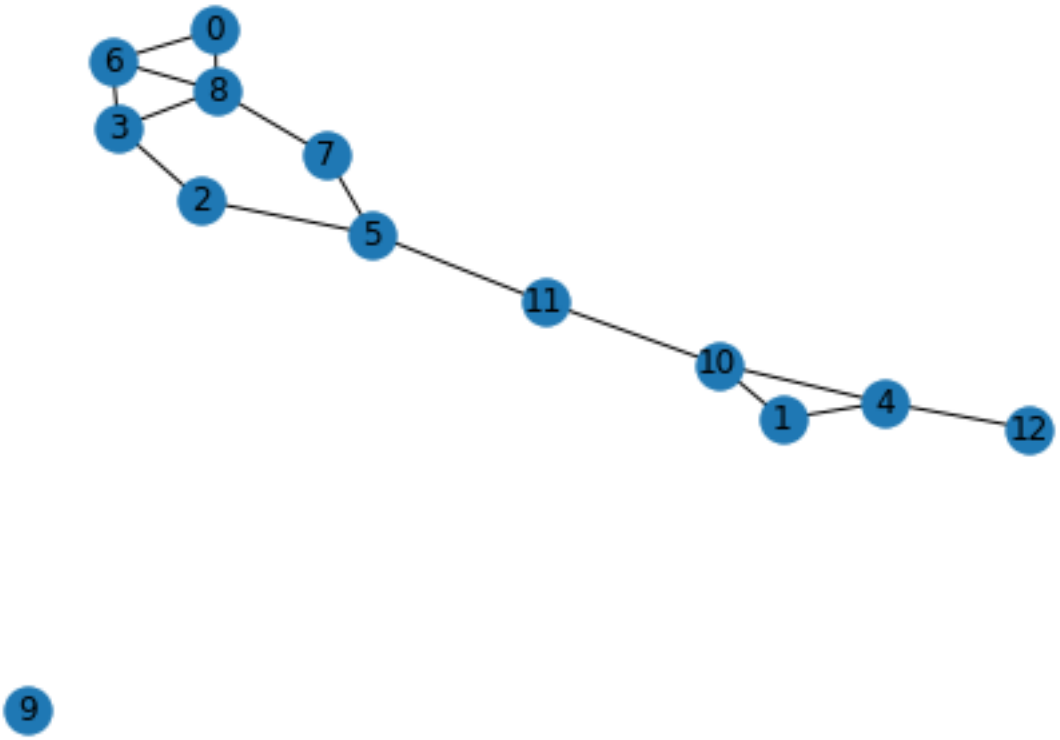
```
def modelo_price(G, num_novos_nos, num_ligacoes, proporcao_preferencial):
num_nos_existentes = len(G.nodes)

for i in range(num_novos_nos):
    novo_no = num_nos_existentes + i
    G.add_node(novo_no)

    # Calcular o número de ligações preferenciais e uniformes
    num_ligacoes_preferenciais = int(num_ligacoes * proporcao_preferencial)
    num_ligacoes_uniformes = num_ligacoes - num_ligacoes_preferenciais

    # Obter a lista de nós existentes e suas respectivas quantidades de ligações
    nos_existentes = list(G.nodes)
    nos_existentes.remove(novo_no)

    # Selecionar nós existentes com probabilidade proporcional
```

```
# ao grau (anexação preferencial)
graus = [G.degree(no) for no in nos_existentes]
nos_para_conectar_preferencial = random.choices(nos_existentes,
                                                weights=graus,
                                                k=num_ligacoes_preferenciais)

# Selecionar nós existentes de forma uniforme (anexação uniforme)
nos_para_conectar_uniforme = random.sample(nos_existentes, num_ligacoes_uniformes)

# Combinar os nós selecionados
nos_para_conectar = nos_para_conectar_preferencial + nos_para_conectar_uniforme

# Adicionar arestas entre o novo nó e os nós selecionados
for no in nos_para_conectar:
    G.add_edge(novo_no, no)
```

Visualizando o crescimento das redes com esse modelo:

Comparação do crescimento com o modelo de price

Acima a rede antes



Acima a rede depois

Parte 2: Análise das redes

Nessa segunda parte, o objetivo é a análise de 4 medidas distintas em redes distintas. A primeira medida é o grau, a quantidade de arestas incidentes no nó. A segunda é a closeness centrality, que mede quão próximo um nó está dos outros nós da rede. Outra medida que aparece é a betwenness centrality, que mede a importância do nó como ponte para a interação com outros nós. E, por fim, a última medida que aparece é o PageRank, que mede a importância geral de um nó para uma rede considerando todas as suas interações. A biblioteca contém naturalmente ferramentas para o cálculo de todas essas medidas. Após fazer o cáclulo de todas essas métricas, elas foram postas no arquivo excel redes:

#Nó	Grau			
	Rede-A	Rede-B	Rede-C	Rede-D
1	2	2	2	2
2	3	3	3	3
5	3	4	4	3
9	3	3	4	4
17	1	1	1	1

#Nó	Closeness Centrality			
	Rede-A	Rede-B	Rede-C	Rede-D
1	0.2411	0.2411	0.2411	0.2411
2	0.2411	0.2479	0.2479	0.244
5	0.1831	0.2337	0.2337	0.2040
9	0.1568	0.1924	0.1961	0.1961
17	0.1358	0.1618	0.1644	0.1644

#Nó	Betweenness Centrality			
	Rede-A	Rede-B	Rede-C	Rede-D
1	0.50	0.27	0.27	0.37
2	0.63	0.45	0.45	0.52
5	0.18	0.30	0.28	0.19
9	0.06	0.06	0.08	0.15
17	0.00	0.00	0.00	0.00

#Nó	PageRank			
	Rede-A	Rede-B	Rede-C	Rede-D
1	0.013	0.013	0.013	0.013
2	0.020	0.019	0.019	0.019
5	0.022	0.027	0.026	0.021
9	0.024	0.023	0.028	0.029
17	0.009	0.009	0.008	0.008