UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
DISCIPLINA INTRODUÇÃO À CIÊNCIA DE REDES

GERANDO GRAFOS DE ERDŐS RÉNYI

VIVIAN RIQUE GIL FERRARO

RIO DE JANEIRO
2022

Nesse trabalho, executei testes computacionais referentes a duas estratégias básicas que reproduzem instâncias do modelo Erdős-Rényi para grafos.
O tempo de processamento utilizado foi o necessário para produzir cada Grafo (tempo em nanossegundos).

Dados do PC utilizado:
Processador 11th Gen Intel(R) Core(TM) i5-11400
32GB de Memória Ram
Adaptador de vídeo Radeon RX550
SO: Windows 11 (22H2)

Estratégia 1: Testa individualmente se cada aresta deve ser adicionada ou não. Não permite auto link, nem vértices repetidos.

- P = 0,1

Código:

```
import networkx as nx
import numpy as np

def CriaGrafo(vertice):  #metodo para criar o grafo
    Grafo = nx.Graph()
    for no in range(vertice):  #cria os vértices
        Grafo.add_node(no)
    for x in range(vertice - 1):    #vertice
        for y in range(x + 1, vertice - 1):  #possibilidade de arestas com os demais vertices do grafo
            z = random.random()      # sorteio
            if z <= 0.1 and x != y:     #olinha o número sorteado com a probabilidade (p=0.1)   #não permite autolink
                if not Grafo.has_edge(y, x):   #não permite arestas duplicadas
                    Grafo.add_edge(x, y)    # adiciona as arestas
    return Grafo
n=4
tempo = 0
while(n<19):    #laço para gerar grafos com 2^4, 2^6, 2^8, 2^10, 2^12, 2^14, 2^16 e 2^18 vértices
    i = 0
    arestas = []
    densidades = []
    GrausMedios = []
    tempos = []
    while (i < 10):  # gera 10 grafos de cada (para calcular as médias)
        tempo1 = time.time_ns()   #tempo de referência (em nanosegundos)
        Grafo = CriaGrafo(2**n)   # gera o grafo segundo o numero de vertices
        tempo2 = time.time_ns()
        tempo = (tempo2 - tempo1)
        grau = []
        arestas.append(Grafo.number_of_edges())
        for no in Grafo.nodes():
            grau.append(nx.degree(Grafo, no))
        GrausMedios.append(np.mean(grau))
        densidades.append(nx.density(Grafo))
        tempos.append(tempo)
        i = i+1

    print(f"N° de nós: {Grafo.number_of_nodes()}")
    print(f"N° médio de Arestas: {np.mean(arestas)}")
    print(f"Média de Graus médios: {np.mean(GrausMedios)}")
    print(f"Média de Densidades: {np.mean(densidades)}")
    print(f"Média do tempo de processamento: {np.mean(tempos)}\n")
    n = n+2
```

Resultados:

C:\Users\vifer\PycharmProjects\EstudandoDL1\venv\Scripts\python.exe "C:\Users\vifer\OneDrive\Documents\BSI\7(5)Periodo\ICR\Codigos\ErdosRenyiAresta1.py"

Nº de nós: 16
Nº médio de Arestas: 8.9
Média de Graus médios: 1.1125
Média de Densidades: 0.07416666666666667
Média do tempo de processamento: 97270.0

Nº de nós: 64
Nº médio de Arestas: 199.1
Média de Graus médios: 6.221875
Média de Densidades: 0.09875992063492063
Média do tempo de processamento: 199840.0

Nº de nós: 256
Nº médio de Arestas: 3239.3
Média de Graus médios: 25.30703125
Média de Densidades: 0.09924325980392157
Média do tempo de processamento: 3904110.0

Nº de nós: 1024
Nº médio de Arestas: 52373.1
Média de Graus médios: 102.2912109375
Média de Densidades: 0.09999140854105572
Média do tempo de processamento: 63099720.0

Nº de nós: 4096
Nº médio de Arestas: 838265.9
Média de Graus médios: 409.309521484375
Média de Densidades: 0.09995348509996947
Média do tempo de processamento: 1039201480.0

Nº de nós: 16384
Nº médio de Arestas: 13421537.7
Média de Graus médios: 1638.3713012695312
Média de Densidades: 0.1000043521497608
Média do tempo de processamento: 23177208990.0
#Erro de memória

Estratégia 1: Testando se cada aresta deve ser adicionada ou não.. Não permite auto link, nem vértices repetidos.

- P = 10/n

Código:

```python
import networkx as nx
import numpy as np

def CriaGrafo(vertice):  #metodo para criar o grafo
    Grafo = nx.Graph()
    for no in range(vertice):  #cria os vértices
        Grafo.add_node(no)
    for x in range(vertice - 1):    #vertice
        for y in range(x + 1, vertice - 1):  #possibilidade de arestas com os demais vertices do grafo
            z = random.random()      # sorteio
            if z <= (10/vertice) and x != y:     #alinha o número sorteado com a probabilidade (10/n)   #não permite autolink
                if not Grafo.has_edge(y, x):   #não permite arestas duplicadas
                    Grafo.add_edge(x, y)    # adiciona as arestas
    return Grafo
n=4
tempo = 0
while(n<19):    #laço para gerar grafos com 2^4, 2^6, 2^8, 2^10, 2^12, 2^14, 2^16 e 2^18 vértices
    i = 0
    arestas = []
    densidades = []
    GrausMedios = []
    tempos = []
    while (i < 10):  # gera 10 grafos de cada (para calcular as médias)
        tempo1 = time.time_ns()   #tempo de referência (em nanosegundos)
        Grafo = CriaGrafo(2**n)   # gera o grafo segundo o numero de vertices
        tempo2 = time.time_ns()
        tempo = (tempo2 - tempo1)
        grau = []
        arestas.append(Grafo.number_of_edges())
        for no in Grafo.nodes():
            grau.append(nx.degree(Grafo, no))
        GrausMedios.append(np.mean(grau))
        densidades.append(nx.density(Grafo))
        tempos.append(tempo)
        i = i+1

    print(f"N° de nós: {Grafo.number_of_nodes()}")
    print(f"N° médio de Arestas: {np.mean(arestas)}")
    print(f"Média de Graus médios: {np.mean(GrausMedios)}")
    print(f"Média de Densidades: {np.mean(densidades)}")
    print(f"Média do tempo de processamento: {np.mean(tempos)}\n")
    n = n+2
```

Resultados:

C:\Users\vifer\PycharmProjects\EstudandoDL1\venv\Scripts\python.exe "C:\Users\vifer\OneDrive\Documents\BSI\7(5)Periodo\ICR\Codigos\ErdosRenyiAresta1-2.py"

Nº de nós: 16
Nº médio de Arestas: 64.2
Média de Graus médios: 8.025
Média de Densidades: 0.535
Média do tempo de processamento: 0.0

Nº de nós: 64
Nº médio de Arestas: 306.6
Média de Graus médios: 9.58125
Média de Densidades: 0.15208333333333332
Média do tempo de processamento: 396580.0

Nº de nós: 256
Nº médio de Arestas: 1261.0
Média de Graus médios: 9.8515625
Média de Densidades: 0.038633578431372544
Média do tempo de processamento: 2695680.0

Nº de nós: 1024
Nº médio de Arestas: 5093.2
Média de Graus médios: 9.94765625
Média de Densidades: 0.009724004154447703
Média do tempo de processamento: 37693920.0

Nº de nós: 4096
Nº médio de Arestas: 20518.0
Média de Graus médios: 10.0185546875
Média de Densidades: 0.0024465335012210016
Média do tempo de processamento: 515100010.0

Nº de nós: 16384
Nº médio de Arestas: 81817.1
Média de Graus médios: 9.98743896484375
Média de Densidades: 0.0006096221061370781
Média do tempo de processamento: 7877301170.0

Nº de nós: 65536
Nº médio de Arestas: 327571.7
Média de Graus médios: 9.996694946289063
Média de Densidades: 0.00015253978708001928
Média do tempo de processamento: 122599761740.0
#Erro de memória

Estratégia 2: Sorteia a quantidade de arestas do grafo, para cada aresta sorteia dois vértices aleatórios, não permite autolink, nem vértices repetidos.

- P=0,1

Código:

```
def CriaGrafo(vertice):   #metodo para criar o grafo
    Grafo = nx.Graph()
    nArestas = (vertice *(vertice-1))/2    # numero maximo de arestas
    for no in range(vertice):    #cria os vértices
        Grafo.add_node(no)
    m = np.random.binomial(nArestas, 0.1)   #numero aleatorio de arestas, segundo distribuição binomial com p = 0.1
    i = 1
    while (i < m):  #Enquanto (m) arestas
        x = random.randrange(vertice - 1)    #sorteia o vertice 1
        y = random.randrange(vertice - 1)    #sorteia o vertice 2
        if x != y:    ##não permite autolink
            if not Grafo.has_edge(y, x):     ##não permite arestas duplicadas
                Grafo.add_edge(x, y)      #cria a aresta
                i += 1
    return Grafo
n=4
tempo = 0
while(n<19):      #laço para gerar grafos com 2^4, 2^6, 2^8, 2^10, 2^12, 2^14, 2^16 e 2^18 vértices
    i = 0
    arestas = []
    densidades = []
    GrausMedios = []
    tempos = []
    while (i < 10):       # gera 10 grafos de cada (para calcular as médias)
        tempo1 = time.time_ns()       #tempo de referência (em nanosegundos)
        Grafo = CriaGrafo(2**n)         # gera o grafo segundo o numero de vertices e a probabilidade das arestas
        tempo2 = time.time_ns()
        tempo = (tempo2 - tempo1)
        grau = []
        arestas.append(Grafo.number_of_edges())
        for no in Grafo.nodes():
            grau.append(nx.degree(Grafo, no))
        GrausMedios.append(np.mean(grau))
        densidades.append(nx.density(Grafo))
        tempos.append(tempo)
        i = i+1
    print(f"N° de nós: {Grafo.number_of_nodes()}")
    print(f"N° médio de Arestas: {np.mean(arestas)}")
    print(f"Média de Graus médios: {np.mean(GrausMedios)}")
    print(f"Média de Densidades: {np.mean(densidades)}")
    print(f"Média do tempo de processamento: {np.mean(tempos)}\n")
    n = n+2
```

Resultados:

C:\Users\vifer\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\vifer\OneDrive\Documents\BSI\7(5)Periodo\ICR\Codigos\ErdosRenyAresta2.py"

Nº de nós: 16
Nº médio de Arestas: 10.2
Média de Graus médios: 1.275
Média de Densidades: 0.085
Média do tempo de processamento: 0.0

Nº de nós: 64
Nº médio de Arestas: 206.5
Média de Graus médios: 6.453125
Média de Densidades: 0.10243055555555555
Média do tempo de processamento: 199940.0

Nº de nós: 256
Nº médio de Arestas: 3268.0
Média de Graus médios: 25.53125
Média de Densidades: 0.10012254901960785
Média do tempo de processamento: 4010330.0

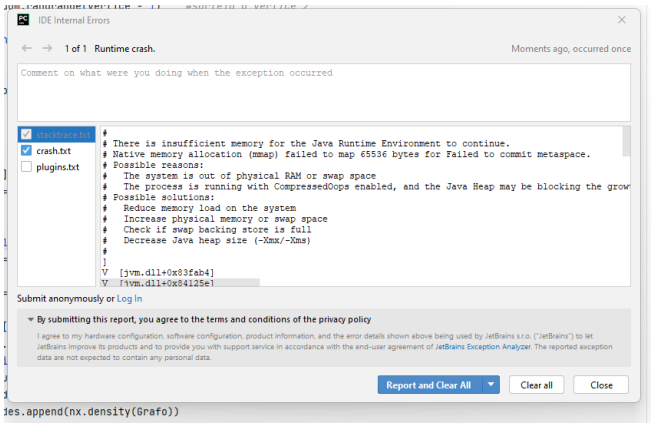
Nº de nós: 1024
Nº médio de Arestas: 52357.5
Média de Graus médios: 102.2607421875
Média de Densidades: 0.09996162481671556
Média do tempo de processamento: 66740420.0

Nº de nós: 4096
Nº médio de Arestas: 838474.5
Média de Graus médios: 409.411376953125
Média de Densidades: 0.09997835823031134
Média do tempo de processamento: 1264313530.0

Nº de nós: 16384
Nº médio de Arestas: 13422890.2
Média de Graus médios: 1638.5364013671874
Média de Densidades: 0.10001442967510148
Média do tempo de processamento: 33164224660.0

#Erro de memória

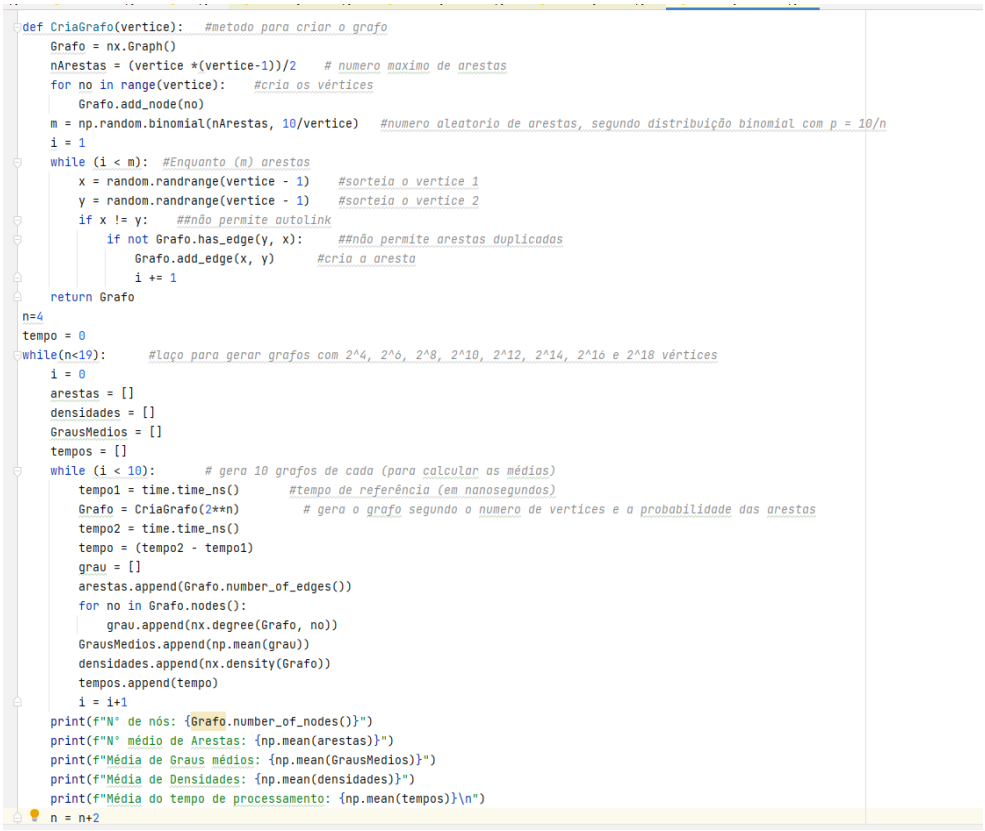
(Travou todo o computador, tive que reiniciar)



Estratégia 2: Sorteia a quantidade de arestas do grafo, para cada aresta sorteia dois vértices aleatórios, não permite auto link, nem vértices repetidos.

- P=10/n

Código:



```
def CriaGrafo(vertice):   #metodo para criar o grafo
    Grafo = nx.Graph()
    nArestas = (vertice *(vertice-1))/2    # numero maximo de arestas
    for no in range(vertice):    #cria os vértices
        Grafo.add_node(no)
    m = np.random.binomial(nArestas, 10/vertice)   #numero aleatorio de arestas, segundo distribuição binomial com p = 10/n
    i = 1
    while (i < m):  #Enquanto (m) arestas
        x = random.randrange(vertice - 1)    #sorteia o vertice 1
        y = random.randrange(vertice - 1)    #sorteia o vertice 2
        if x != y:    ##não permite autolink
            if not Grafo.has_edge(y, x):     ##não permite arestas duplicadas
                Grafo.add_edge(x, y)      #cria a aresta
                i += 1
    return Grafo
n=4
tempo = 0
while(n<19):      #laço para gerar grafos com 2^4, 2^6, 2^8, 2^10, 2^12, 2^14, 2^16 e 2^18 vértices
    i = 0
    arestas = []
    densidades = []
    GrausMedios = []
    tempos = []
    while (i < 10):       # gera 10 grafos de cada (para calcular as médias)
        tempo1 = time.time_ns()       #tempo de referência (em nanosegundos)
        Grafo = CriaGrafo(2**n)         # gera o grafo segundo o numero de vertices e a probabilidade das arestas
        tempo2 = time.time_ns()
        tempo = (tempo2 - tempo1)
        grau = []
        arestas.append(Grafo.number_of_edges())
        for no in Grafo.nodes():
            grau.append(nx.degree(Grafo, no))
        GrausMedios.append(np.mean(grau))
        densidades.append(nx.density(Grafo))
        tempos.append(tempo)
        i = i+1
    print(f"N° de nós: {Grafo.number_of_nodes()}")
    print(f"N° médio de Arestas: {np.mean(arestas)}")
    print(f"Média de Graus médios: {np.mean(GrausMedios)}")
    print(f"Média de Densidades: {np.mean(densidades)}")
    print(f"Média do tempo de processamento: {np.mean(tempos)}\n")
    n = n+2
```

Resultados:

C:\Users\vifer\PycharmProjects\EstudandoDL1\venv\Scripts\python.exe "C:\Users\vifer\OneDrive\Documents\BSI\7(5)Periodo\ICR\Codigos\ErdosRenyiAresta2-2.py"

Nº de nós: 16
Nº médio de Arestas: 74.3
Média de Graus médios: 9.2875
Média de Densidades: 0.6191666666666668
Média do tempo de processamento: 99900.0

Nº de nós: 64
Nº médio de Arestas: 304.4
Média de Graus médios: 9.5125
Média de Densidades: 0.15099206349206348
Média do tempo de processamento: 400080.0

Nº de nós: 256
Nº médio de Arestas: 1275.6
Média de Graus médios: 9.965625
Média de Densidades: 0.039080882352941174
Média do tempo de processamento: 1500060.0

Nº de nós: 1024
Nº médio de Arestas: 5083.3

Média de Graus médios: 9.9283203125
Média de Densidades: 0.009705102944770285
Média do tempo de processamento: 6900270.0

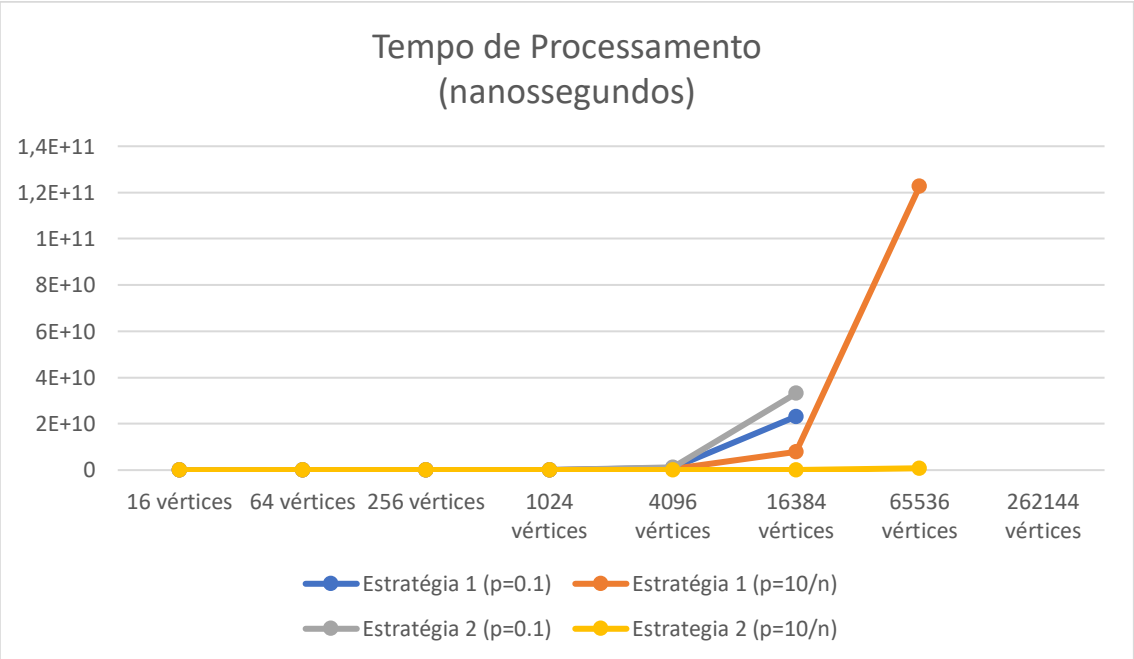
Nº de nós: 4096
Nº médio de Arestas: 20487.7
Média de Graus médios: 10.003759765625
Média de Densidades: 0.002442920577686203
Média do tempo de processamento: 31699800.0

Nº de nós: 16384
Nº médio de Arestas: 81718.7
Média de Graus médios: 9.97542724609375
Média de Densidades: 0.0006088889242564701
Média do tempo de processamento: 153902070.0

Nº de nós: 65536
Nº médio de Arestas: 327840.1
Média de Graus médios: 10.004885864257812
Média de Densidades: 0.0001526647724766584 6
Média do tempo de processamento: 799999880.0

#Erro de memória (OverflowError: Python int too large to convert to C long)

Comparação dos tempos de processamento



Podemos observar que quando o grafo tem poucos vértices / arestas (até 1024 vértices), a estratégia utilizada não faz muita diferença. A diferença está toda relacionada à densidade do grafo. Grafos mais densos (probabilidade da aresta de 0,1), que tem densidade de cerca de 0,1, tem mais arestas e o tempo de processamento é maior, esses estouraram a memória e não

consegui produzir grafos além de 16384 vértices. Já nos grafos com muitos vértices, e com densidade menor (p=10/n), que mantinham os graus dos vértices em média = 10, consegui produzir grafos de até 65536 vértices antes de estourar a memória.

Nesse caso (p=10/n), onde os grafos quanto maiores, menor a densidade (mais esparsos), a segunda estratégia se mostrou muito mais eficiente, acredito que o motivo é que precisamos gerar um número de arestas (m), que mesmo sendo grande, dependem de dois sorteios e de uma atribuição para cada, e isso consome muito menos recursos e tempo de processamento que a estratégia 1, onde precisamos continuar visitando cada vértice para decidir se a aresta será adicionada ou não.