TAG 2023.2 - Projeto 1
Vitor Guedes Frade - 221017130

Este projeto visa gerar soluções para os seguintes exercícios propostos em https://ona-book.org/index.html Após estudar e analisar o capítulo 7 (Components, Communities and Cliques), segue os códigos e respostas aos exercícios, do item 7.5.2 (Data exercises): 4, 5, 6, 7, 8, 9, e 10.

Para resolução dos itens foram utilizadas as seguintes bibliotecas que devem ser instaladas na máquina para execução programa:

```
import networkx as nx
import pandas as pd
import networkx as nx
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from networkx.algorithms import community
from matplotlib import cm
import numpy as np
from community import community_louvain
```

Construção do objeto do grafo para uso em todos os itens:

```
verticesRead = pd.read_csv("email_vertices.csv")
arestas = pd.read_csv("email_edgelist.csv")

#Cria dicionário de vertices como ID, e dept como atributo de cada vértice
vertices = {}
for _,vertice in verticesRead.iterrows():
    vertices[vertice["id"]] = {"dept": vertice["dept"]}

#Cria o grafo e passa os vértices
grafo = nx.Graph()
grafo.add_nodes_from(vertices.items())

#Passa as arestas do grafo
for _,aresta in arestas.iterrows():
     grafo.add_edge(aresta["from"],aresta["to"])

questao4(grafo)
#questao5(grafo)
#questao6(grafo)
#questao7(grafo)
#questao8(grafo)
#questao9(grafo)
#questao10(grafo)
```

OBS: Para cada item foi feita uma função que recebe o objeto do grafo como parâmetro e executa um algoritmo para encontrar a resposta como solicita o item.

4.Determine the connected components of this network and reduce the network to its largest connected component.

RESPOSTA: Para encontrar o maior componente conectado foi criar uma função separada para retornar o maior componente conectado e a partir dele fazer as análises.

```
def maiorComponente(grafo):
    # Cria lista de componentes do grafo
    componentes = nx.connected_components(grafo)

    #Cria dicionário de subGrafos contidos no grafo com tamanhos como chave
    subGrafos = {}
    for componente in componentes:
        subGrafo = grafo.subgraph(componente)
        subGrafos[subGrafo.size()] = subGrafo  #{numero de vertices: componente}

    #Obtem o maior subGrafo do conjunto
    maiorComponente = subGrafos[max(subGrafos.keys())]

    return maiorComponente
```

Para visualização do maior componente conectado ou qualquer outro componente do grafo foi criada uma função que recebe o objeto (grafo) a ser visualizado e gera um arquivo png com o nome passado como parâmetro e guarda o arquivo no diretório do projeto.

```
def gerarPng(grafo, nome):
    # Configura o layout do grafo
    pos = nx.spring_layout(grafo, seed=42)  # Experimente diferentes layouts conforme necessário

    # Desenha o grafo
    plt.figure(figsize=(12, 8))
    nx.draw_networkx_nodes(grafo, pos, node_size=12, node_color='skyblue')
    nx.draw_networkx_edges(grafo, pos, edge_color='gray', alpha=0.5)

    #Adiciona rótulos aos vertices
    node_labels = {node: str(node) for node in grafo.nodes()}
    nx.draw_networkx_labels(grafo, pos, labels=node_labels, font_size=8)

    #Remove as limitações de dimensão na visualização
    plt.axis('off')

    #Cria um png e salva no diretorio do projeto
    plt.savefig(nome + ".png")
```

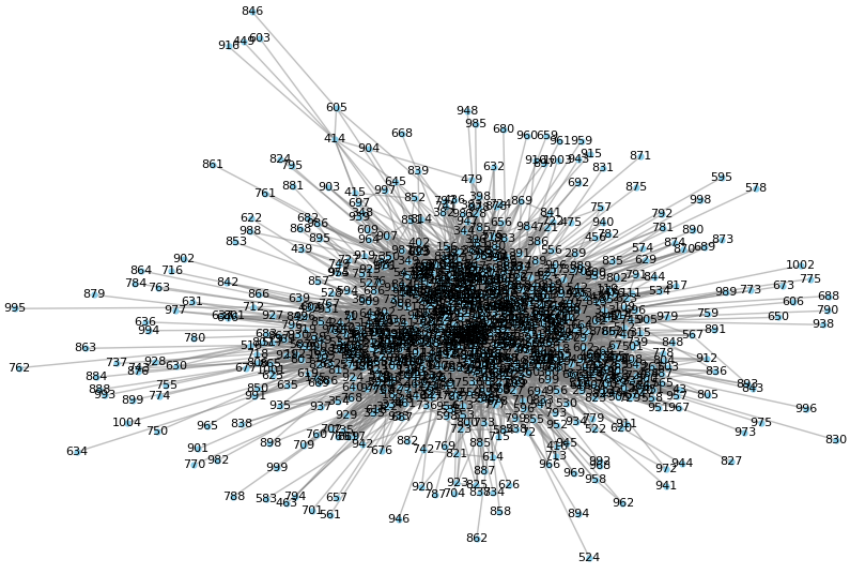
A função abaixo executa o algoritmo de solução do item 4 achando o maior componente conectado a imprimindo valores referentes a ele, assim como gerando o arquivo png contendo a visualização do respectivo componente.

```
def questao4(grafo):
    maior = maiorComponente(grafo)

    print(f'Grafo com {grafo.number_of_nodes()} vertices, e {grafo.number_of_edges()} arestas')
    print(f'Maior componente conectado: {maior.number_of_nodes()} vertices, e {maior.number_of_edges()} arestas')
```

```
gerarPng(maior,"MaiorComponenteConectado")
```

Resultados obtidos ao executar a função para o item 4:

```
Grafo com 1005 vertices, e 16064 arestas
Maior componente conectado: 986 vertices, e 16064 arestas
```



5.Use the Louvain algorithm to determine a vertex partition/community structure with optimal modularity in this network.

RESPOSTA: A modularidade indica a densidade das conexões entre os vértices dentro de uma partição de comunidades, sendo assim a modularidade ideal para uma

partição é subjetiva e depende do contexto de análise para ser definida.No contexto desse grafo considerei que a modularidade ideal seria o maior valor possível,assim quanto mais próximo de 1 a modularidade melhor por indicar por indicar uma forte interação entre diferentes ID na partição.A modularidade também pode indicar a probabilidade das comunidades dentro da partição seres reais, a análise é feita avaliando e comparando com resultados esperados de um grafo não direcionado gerado aleatoriamente,se o valor da modularidade pertence a um intervalo entre [-0.5,1], então valores acima de zero indicam veracidade por serem maiores que o esperado de um grafo aleatório, e valores próximos de 0 indicadores de densidade baixa densidade.

Para fazer a avaliação foram utilizados os métodos “louvain_communities()” e “modularity” da biblioteca Networkx.O algoritmo de louvain faz buscas por uma partição de comunidades com modularidade maximizada e retorna o conjunto de comunidades da partição encontrada, então foi utilizado o algoritmo de Louvain com semente para sempre retornar a mesma partição para um mesmo grafo, a após a obtenção da partição a modularidade foi calculada para fazer avaliação.

```
def questao5(grafo):
    #Cria uma partição de comunidades usando o algoritmo de Louvain e padroniza a geração usando a semente "123"
    particao = nx.community.louvain_communities(grafo,seed = 123)

    #Calcula a modularidade da partição em relação ao grafo
    modularidade = nx.community.modularity(grafo,particao)

    print(f'Partição com {len(particao)} comunidades')
    print(f'Maior comunidade com {len(max(particao, key=len))} vertices')

    print("Modularidade esperada em um grafo aleatorio NÃO ponderado e NÃO direcionado:[-0.5,1]")
    print(f"Modularidade encontrada na partição gerada pelo algoritmo de Louvain: {modularidade}")

    if modularidade > 0:
        print("Os vertices estão mais densamente conectados do que o esperado pelo acaso")
    elif modularidade == 0:
        print("Os vertices estão conectados com a densidade esperada pelo acaso")
    else:
        print("Os vertices estão menos densamente conectados do que o esperado pelo acaso")
```

RESULTADOS obtidos na execução da função da questão 5:

```
Parti��o com 27 comunidades
Maior comunidade com 252 vertices
Modularidade esperada em um grafo aleatorio N�O ponderado e N�O
direcionado:[-0.5,1]
Modularidade encontrada na parti��o gerada pelo algoritmo de
Louvain: 0.4165825628480848
Os vertices est�o mais densamente conectados do que o esperado
pelo acaso
```

Com base nos resultados de modularidade pode-se concluir que as comunidades da partição geradas pelo algoritmo de Louvain são verdadeiras, e possuem uma boa

densidade de conexões indicando forte troca de mensagens entre diferentes usuários da rede.

6.Compare the modularity of the Louvain community structure with that of the ground truth department structure.

RESPOSTA: O ground truth é a verdadeira informação ou dados que servem como referência para validar outras informações ou dados. Assim como a modularidade ideal, a ground truth também é subjetiva e depende de analisar o contexto para ser determinada, um grafo nesse contexto tende a possuir alta densidade de conexões entre ID de mesmo departamento, então a ground truth foi considerada como uma partição onde todos os vértices das comunidades pertenceriam ao mesmo departamento, e a partir desses preceitos a partição foi gerada por um algoritmo específico não presente na biblioteca Networkx.Após a obtenção da partição de Louvain e a de ground truth foram calculadas suas respectivas modalidades e comparadas para verificar suas densidades de conexões.

Função para gerar partição de ground truth:

```
def ground_truth(grafo):
    comunidades = dict()

    for vertice,dictDept in grafo.nodes(data = True):
        if not dictDept["dept"] in comunidades:
            comunidades[dictDept["dept"]] = {vertice}
        else:
            comunidades[dictDept["dept"]].add(vertice)

    return comunidades
```

Função com algoritmo de solução da questão 6:

```
def questao6(grafo):
    #Cria uma partição de comunidades usando o algoritimo de Louvain e padroniza a geração usando a semente "123"
    particao = list(nx.community.louvain_communities(grafo,seed = 123))

    #Obtem a modularidade da partição de ground truth
    groundTruth = list(ground_truth(grafo).values())

    #Caulcula as mudularidades das partições
    modularidadeLouvain = nx.community.modularity(grafo,particao)
    modularidadeGroundTruth = nx.community.modularity(grafo,groundTruth)

    print(f'Modularidade do grafo usando Louvain: {modularidadeLouvain}')
    print(f'Modularidade obtida pela ground truth: {modularidadeGroundTruth}')

    if modularidadeLouvain > modularidadeGroundTruth:
        print("Partição de louvain apresenta maior densidade de conexões que a partição de ground truth")
    elif modularidadeLouvain < modularidadeGroundTruth:
        print("Partição de ground truth apresenta maior densidade de conexões que a partição de Louvain")
```

RESULTADOS obtidos na execução da função da questão 6:

```
Modularidade do grafo usando Louvain: 0.4165825628480848
Modularidade obtida pela ground truth: 0.28801318862374214
Parti◆◆o de louvain apresenta maior densidade de conex◆es que a parti◆◆o de ground truth
```

Portanto, as conexões na partição de Louvain são mais densas que as da partição de ground truth determinada.Logo, a partição de Louvain possui conexões mais densas que extrapolam os limites de interações de ID restritas apenas ao mesmo departamento, aproximando-se mais ao determinado como ideal.

7.Visualize the graph color-coded by the Louvain community, and then visualize the graph separately color-coded by the ground truth department. Compare the visualizations. Can you describe any of the Louvain communities in terms of departments?

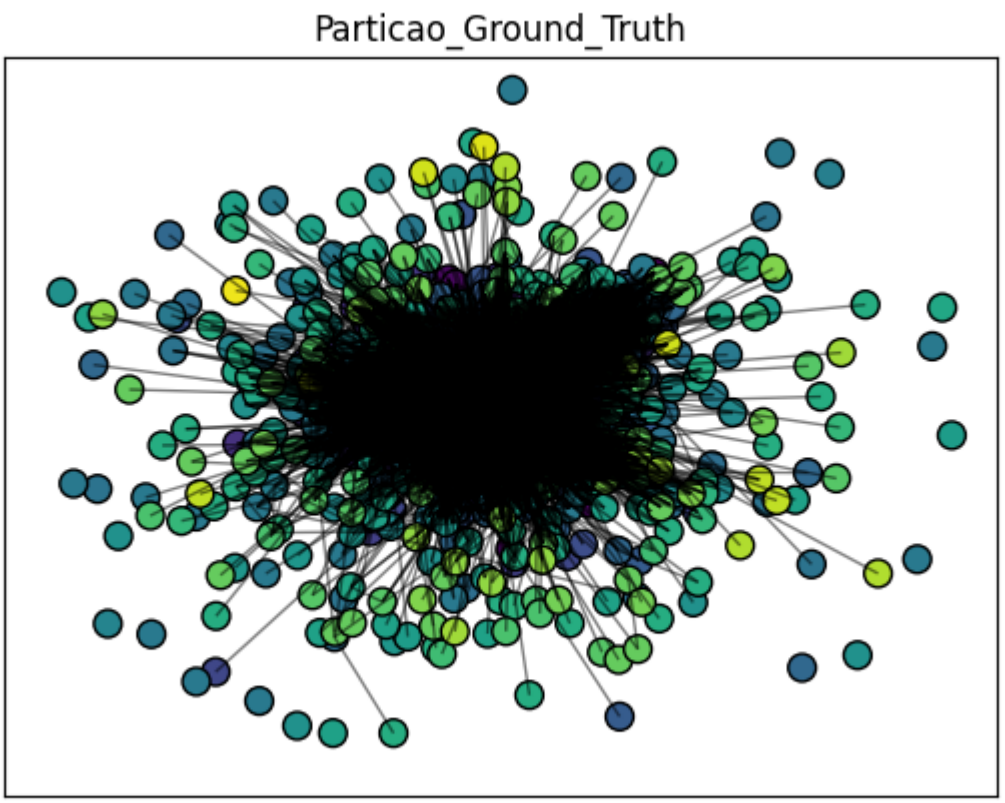
RESPOSTA: Para visualização primeiro foram obtidas ambas partições através de métodos utilizados em questões anteriores, e depois foram visualizadas através do uso de um função criada que divide as comunidades da partição em cores diferentes e gera um png do desenho e salva no diretório.

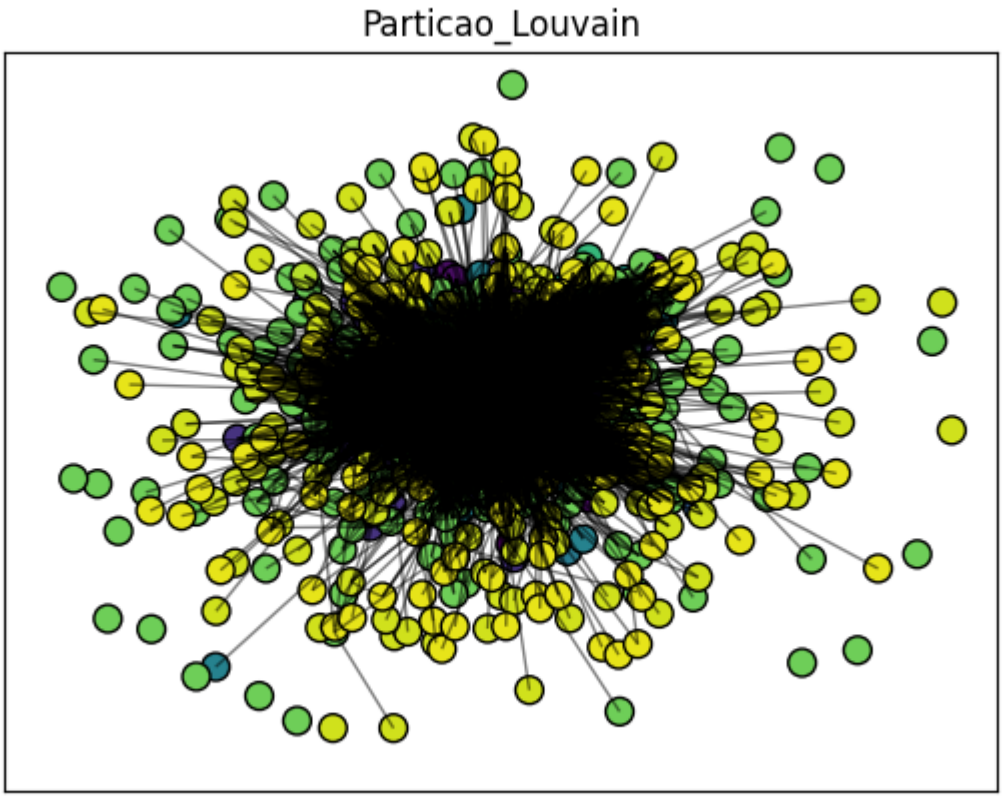
```python
def visualizarParticao(grafo, partitions, nome):
    #Configura o layout
    pos = nx.spring_layout(grafo,seed=42,k=0.2)

    #Obtem o número de comunidades
    num_partitions = len(partitions)

    #Obtem uma mapa de cores
    cmap = cm.get_cmap('viridis', num_partitions + 1)

    comunidades = {}
    #Para cada vertice é atribuido uma comunidade
    for community_id,nodes in enumerate(partitions):
        for node in nodes:
            comunidades[node] = community_id

    #Cria uma lista de codigos de cores
    cores =[]
    for community_id in comunidades.values():
        #Cada comunidade recebera um codigo de cor entre [0,1]
        cores.append(cmap(community_id / num_partitions))

    #Cria uma janela interna no layout para conter a imagem do grafo
    fig, ax = plt.subplots()
    ax.scatter(*zip(*pos.values()), s=100, c=cores, edgecolors='k', cmap='viridis')

    #Desenha arestas
    nx.draw_networkx_edges(grafo, pos,ax=ax, alpha=0.5)
```

```
    #Adicionar titulo
    plt.title(nome)

    #Salva png do desenho no diretorio
    plt.savefig(nome + ".png")
```

```
def questao7(grafo):
    #descobrindo a melhor partição com a maior modularidade
    particaoLouvain = list(nx.community.louvain_communities(grafo,seed = 123))

    #Obtendo a partição ground truth sendo comunidades de vertices do mesmo departamento
    particaoGroundTruth = list(ground_truth(grafo).values())

    #Gera um png para cada visualização das diferentes partições
    visualizarParticao(grafo,particaoLouvain,"Particao_Louvain")
    visualizarParticao(grafo,particaoGroundTruth,"Particao_Ground_Truth")
```

RESULTADOS obtidos na execução da função da questão 7:





Pela distribuição de cores em ambas visualizações é possível perceber a dominância de comunidades específicas em números de membros por suas altas frequência de aparecimento na visualização, mas nas comunidades de Louvain não necessariamente todos os membros de uma comunidade pertencem ao mesmo departamento, apesar de sua alta probabilidade de ocorrência como foi verificado na análise de modalidades.

8.Create a dataframe containing the community and department for each vertex. Manipulate this dataframe to show the percentage of individuals from each department in each community. Try to visualize this using a heatmap or other style of visualization and try to use this to describe the communities in terms of departments.

RESPOSTA: Para visualizar a distribuição de departamentos nas comunidades da partição gerada pelo algoritmo de louvain, primeiro foram criados dicionários contendo as informações de porcentagem de departamentos por comunidade através de funções não inerentes nas bibliotecas utilizadas.Após o levantamento de dados foi gerado o mapa de calor usando objetos e métodos das bibliotecas pandas e matplotlib.Pela visualização do mapa pode-se concluir que os departamentos possuem uma distribuição com valores

uniformes em diferentes comunidades, uma vez que as variações de cores de um mesmo departamento em diferentes comunidades é quase imperceptível, assim manifestando um padrão de variação muito baixa.Portanto, a análise foi concentrada em comparar a relevância dos departamentos na comunidade, sendo perceptível uma baixa frequência da maioria dos departamentos em toda partição, mas com destaque para a frequência de alguns departamentos específicos que possuem frequências superiores a 5%.

Funções auxiliares:

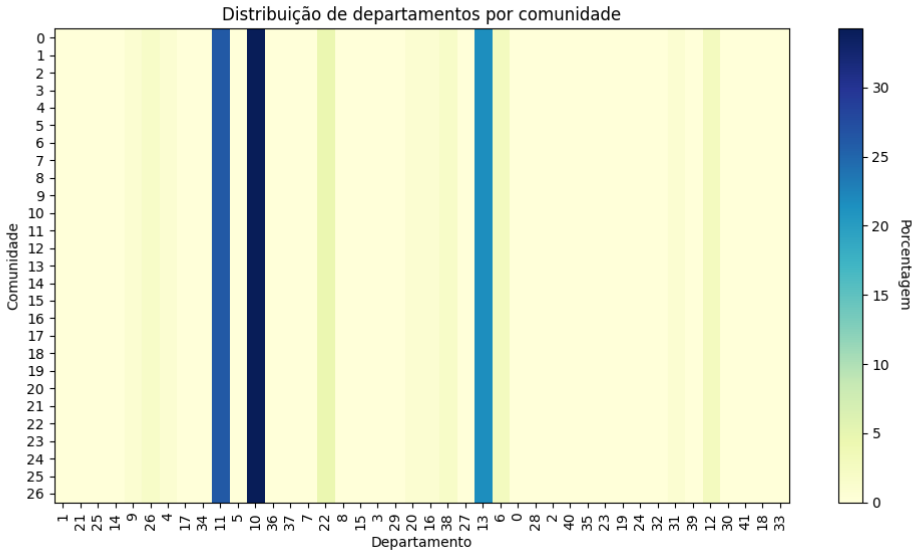
```
def deptComunidade(grafo,particao):
    #Atribui a cada departamento uma porcentagem zero
    departamentos = {}
    for vertice,atributos in grafo.nodes(data = True):
        if not atributos["dept"] in departamentos.keys():
            departamentos[atributos["dept"]] = 0

    #Cada comunidade recebe os departamentos com porcentagem inicial zero
    comunidades = {comunidade:departamentos for comunidade in range(len(particao))}

    #Passa a porcentagem de cada departamento em cada comunidade
    for comunidade in comunidades:
        for dept in comunidades[comunidade]:
            frequencia = porcentagem(grafo,particao[comunidade],dept)
            comunidades[comunidade][dept] = frequencia

    return comunidades

def porcentagem(grafo,comunidade,dept):
    #Define um valor base para o numerador
    numerador = 0

    #Busca frequência do dept no na comunidade
    for vertice in comunidade:
        if grafo.nodes[vertice]["dept"] == dept:
            numerador += 1

    #Retorna o valor da porcentagem
    return (numerador/len(comunidade)) * 100
```

Função principal para resolução da questão 8:

```
def questao8(grafo):
    #Cria uma partição de comunidades usando o algoritmo de Louvain e padroniza a geração usando a semente "123"
    particao = nx.community.louvain_communities(grafo,seed = 123)

    #Obtém dict de porcentagens
    #chave -> comunidade
    #valores -> dict de dept e porcentagem
    parametros = deptComunidade(grafo,particao)

    #Obtém lista de comunidades
    comunidades = parametros.keys()

    #Acessa os valores da primeira comunidade
    dept = parametros[list(parametros.keys())[0]]

    #Obtém as listas de departamentos
    departamentos = list(dept.keys())
```

```
#Obtém as porcentagens de dept/comunidade
porcentagens = []
for comunidade in parametros.values():
    porcentagens.append(list(comunidade.values()))

#Cria o data frame com os 3 parâmetros necessários para o mapa de calor
dataFrame = pd.DataFrame(porcentagens,columns = departamentos, index = comunidades)

#Colore o mapa com paleta de amarelo a azul
plt.imshow(dataFrame, cmap ="YlGnBu")

#Insere barra descritiva de cores em porcentagem
barra_porcentagem = plt.colorbar()

#Posiciona e descreve barra de porcentagem
barra_porcentagem.set_label('Porcentagem', rotation=270, labelpad=15)

#Rotula as linhas e colunas dentro dos intervalos existentes no mapa
plt.xticks(range(len(dataFrame.columns)), dataFrame.columns, rotation='vertical')
plt.yticks(range(len(dataFrame)), dataFrame.index)

#Descreve dados dos eixos
plt.xlabel("Departamento")
plt.ylabel("Comunidade")

#Adiciona titulo ao frame
plt.title("Distribuição de departamentos por comunidade")

#Visualiza o frame
plt.show()

return
```

RESULTADO do mapa gerado:



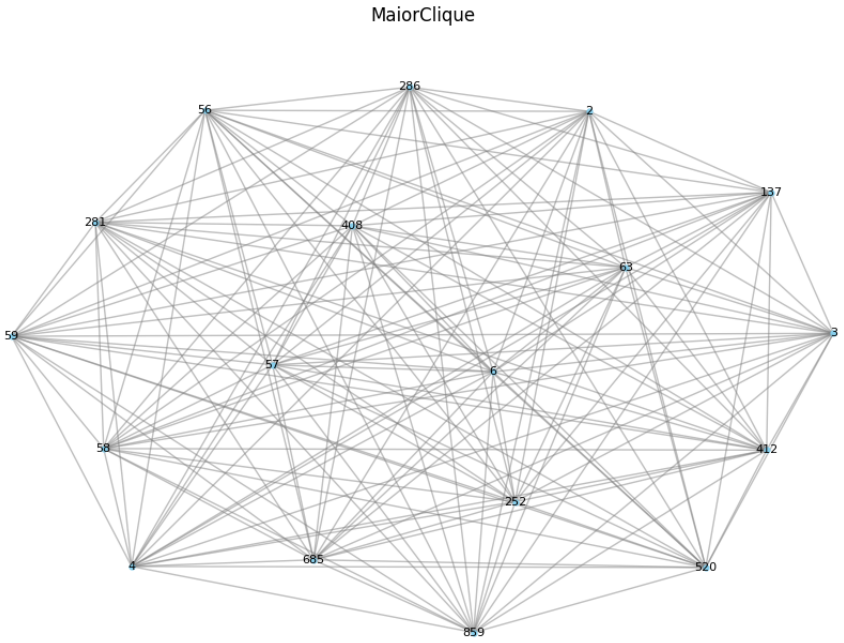
9.Find the largest clique size in the graph. How many such largest cliques are there? What do you think a clique represents in this context?

RESPOSTA: Para encontrar os maiores cliques contidos no grafo foi utilizado o método “find_cliques()” da biblioteca networkx para obter uma lista de cliques maximais contidos no grafo independente da ordem ou grau de vértices. Após obter todos os cliques contidos no grafo, foi feita uma busca pela ordem dos maiores cliques máximos usando o método max() do python e criada uma lista com os maiores cliques maximais para realizar a análise .A existência de um clique nesse contexto manifesta uma relação de interação muito forte entre o ID dos vértices envolvidos, podendo ser assim mensagens trocadas entre um grupo de amigos ou um grupo de membros que estejam envolvidos em uma mesma tarefa.

```python
def questao9(grafo):
    #Cria uma lista com todos os cliques maximais do grafo
    cliques = nx.find_cliques(grafo)

    #Conta a quantidade de cliques maximais no grafo
    quantCliques = 0
    for clique in cliques:
        quantCliques += 1

    #Encontra o clique com maior numero de vertices na lista de cliques
    tamanhoMaioresCliques = len(max(nx.find_cliques(grafo), key=len))

    #Lista maiores cliques maximais do grafo
    maioresCliques = []
    for clique in nx.find_cliques(grafo):
        if len(clique) == tamanhoMaioresCliques:
            maioresCliques.append(clique)

    print(f'Quantidade de cliques no grafo: {quantCliques}')
    print(f'Ordem dos maiores clique maximais: {tamanhoMaioresCliques}')
    print(f'Quantidade de cliques maximais com ordem {tamanhoMaioresCliques}:{len(maioresCliques)}')

    #Gera um subgrafo a partir de um dos maiores cliques maximais
    grafoClique = grafo.subgraph(maioresCliques[0])

    #Gera um png com o grafo do clique maximal marcado
    gerarPng(grafoClique,"MaiorClique")
```

Resultados obtidos ao executar a função para o item 9:

```
Quantidade de cliques no grafo: 42728
Ordem dos maiores clique maximais: 18
Quantidade de cliques maximais com a maior ordem:56
```



10.Try to visualize the members of these cliques in the context of the entire graph. What can you conclude?

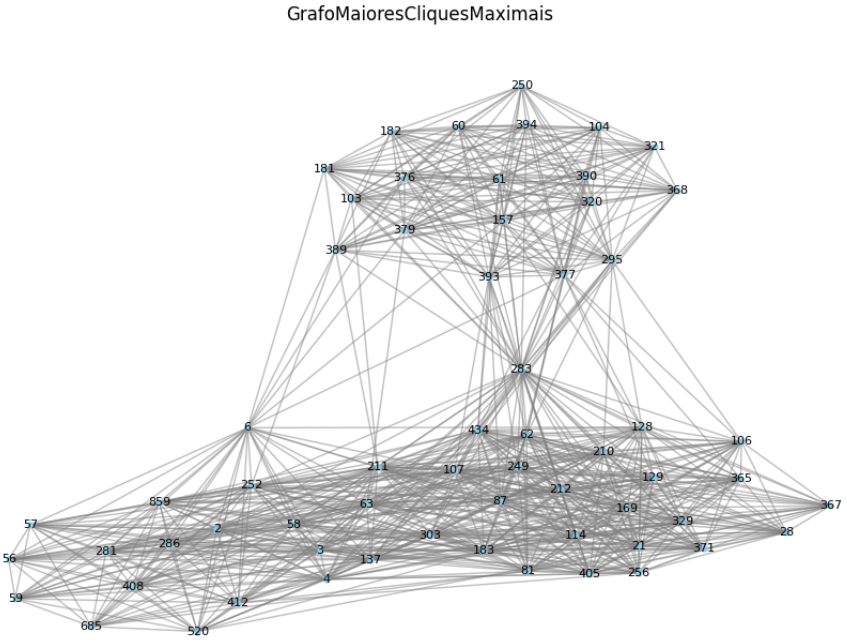
RESPOSTA: Como demonstrado na questão 9, o grafo possui 42728 cliques maximais, sendo um número muito alto não seria trivial a visualização de todos.A visualização foi feita reduzindo o grafo aos vértices dos maiores cliques maximais, assim sendo perceptivel a intercecção de vertices entre os cliques maximais, mas permitindo que um mesmo vértice possa pertencer a mais de um clique maximal no grafo.

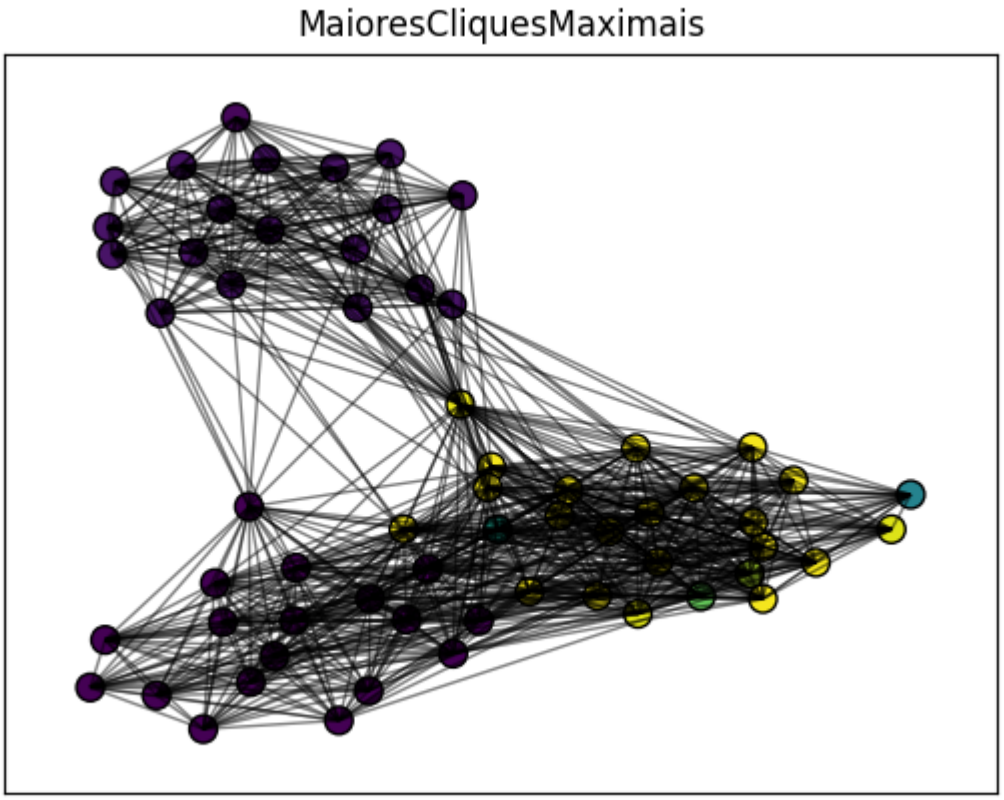
```
def questao10(grafo):
    #Cria uma lista com todos os cliques do grafo
    cliques = nx.find_cliques(grafo)

    #Obtem ordem dos maiores cliques maximais
    tamanhoMaioresCliques = len(max(nx.find_cliques(grafo), key=len))

    #Gera lista com os maiores cliques maximais
    maioresCliques =[]
    for clique in cliques:
        if len(clique) == tamanhoMaioresCliques:
            maioresCliques.append(set(clique))

    #Gera grafo com todos vertices presentes nos maiores cliques maximais
    grafoCliques = nx.Graph()
    for clique in maioresCliques:
        if len(clique) == tamanhoMaioresCliques:
            grafoCliques.add_nodes_from(clique)

    #Adiciona as arestas ao grafo de vertices presentes nos maiores cliques maximais
    verticesGrafoClique = grafoCliques.nodes()
```

```
    for edge in grafo.edges():
        if (edge[0] in verticesGrafoClique) and (edge[1] in verticesGrafoClique):
            grafoCliques.add_edge(edge[0],edge[1])

    print(f'Ordem dos maiores cliques maximais:{tamanhoMaioresCliques}')
    print(f'Grafo reduzido aos vertices dos maiores cliques maximais:')
    print(grafoCliques)

    #Visualiza grafo por vertices
    gerarPng(grafoCliques,"GrafoMaioresCliquesMaximais")

    #Visualiza grafo colorido por cliques
    visualizarParticao(grafoCliques,list(maioresCliques),"MaioresCliquesMaximais")
```

RESULTADOS:

```
Ordem dos maiores cliques maximais:18
Grafo reduzido aos vertices dos maiores cliques maximais:
Graph with 62 nodes and 809 edges
```





Fontes de pesquisa disponíveis em:

https://ona-book.org/community.html

https://diegomariano.com/networkx/

https://networkx.org/documentation/stable/reference/algorithms/community.html

https://ww2.uft.edu.br/index.php/nit/vitrine-tecnologica/portfolio-de-softwares/27537-editor-de-ground-truth-data-para-aplicacoes-de-reconhecimento-de-padrao-usando-imagens

https://dicionariodopetroleo.com.br/verdade-terra-ou-terra-verdade/

https://pandas.pydata.org/docs/reference/index.html#api

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.colorbar.html

https://seaborn.pydata.org/tutorial/objects_interface.html

https://numpy.org/doc/

https://lume.ufrgs.br/bitstream/handle/10183/86094/000909891.pdf

https://www.ic.unicamp.br/~wainer/cursos/2s2021/433/aula8.html