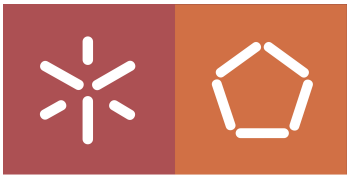

Universidade do Minho
Escola de Engenharia

INTELIGÊNCIA AMBIENTE:
TECNOLOGIAS E APLICAÇÕES

QUESTÃO DE AULA 2

CONHECIMENTO EM REDES SOCIAIS

JOANA AFONSO GOMES **A84912**

15 de janeiro de 2021

Conteúdo

1 INTRODUÇÃO / OBJETIVOS DO TRABALHO

Foi-me proposto, no âmbito da Unidade Curricular de **Inteligência Artificial: Tecnologias e Aplicações**, a realização desta segunda Questão de Aula, focada na manipulação e aplicação de conhecimento em Redes Sociais, utilizando a biblioteca NetworkX de Python.

NOTA: Ao correr o *script* da Questão de Aula, as várias funcionalidades descritas no relatório executam sequencialmente (com a mesma ordem que são apresentadas no presente relatório). Às funcionalidades que têm *output* em terminal segue-se um ***delay* de 10 segundos**, antes da inicialização da sequência seguinte (atribuí ainda outros *delays* mais pequenos para facilitar a legibilidade do progama). As que têm *output* numa janela da aplicação devem ser fechadas para prosseguir a execução do programa pela ordem definida.

2 INFORMAÇÃO DO FICHEIRO .TXT

Perante uma análise crítica, posso concluir que as linhas do ficheiro lista_arestas.txt se dividem em 4 colunas distintas:

- **1ª coluna:** A primeira pessoa da relação;
- **2ª coluna:** A segunda pessoa da relação;
- **3ª coluna:** O peso da relação (quanto maior, mais forte);
- **4ª coluna:** A existência (ou não) de uma relação (1 corresponde à existência de uma relação confirmada e 3 a uma relação possível mas não confirmada.

Assim, exemplificando com a linha seguinte, que é uma das entradas do ficheiro,

```
1 Nawaf Alhazmi,Osama Awadallah,5,1
```

Nawaf Alhazmi tem uma relação confirmada com Osama Awadallah, e esta relação é considerada forte.

3 CRIAÇÃO DO GRAFO

Seguindo o primeiro ponto do enunciado, comecei por criar um grafo com as informações contidas no ficheiro .txt fornecido, utilizando os parâmetros que considerei adequados para uma tentativa de maior legibilidade das *labels*.

```
##### Abrir e processar o ficheiro (codigo fornecido) #####

in_file=csv.reader(open('lista_arestas.txt','r'))

g=net.Graph()
for line in in_file:
    g.add_edge(line[0],line[1],weight=line[2],conf=line[3])

##### Criar grafo com as informacoes do ficheiro .txt #####

# Posicoes para todos os nodos
pos = net.circular_layout(g)

# Ajustar tamanho da figura
plt.figure(figsize=(50,50))

# Nodos
net.draw_networkx_nodes(g, pos, node_size=50)

# Arestas
net.draw_networkx_edges(g, pos, width=0.5, alpha=0.5, edge_color='b')

# Labels
net.draw_networkx_labels(g, pos, font_size=7.5, font_family='FreeMono',
    font_weight='bold')

plt.axis('off')
plt.show();
```

Listing 1: Criação do grafo a partir do ficheiro `lista_arestas.txt`

Como resultado, obtive o grafo exibido no Anexo A, que intentei colocar no relatório de forma a atingir a maior percepção dos dados possível.

4 ANÁLISE DA REDE

NOTA: Defini a seguinte variável no início do programa, que é usada na elaboração das diferentes funcionalidades, e que lista todos os nodos do grafo.

```
all_nodos = list(g.nodes)
```

4.1 DISTRIBUIÇÃO DE GRAU

Sendo o **grau** de um nó de uma rede o número de conexões que ele possui com outros nodos, a **distribuição de grau** é a distribuição de probabilidade desses graus ao longo de toda a rede.

Deste modo, é possível criar um histograma que apresente qual a distribuição de grau no grafo g dado.

Primeiramente usei a função `sorted` para fazer uma sequência númerica dos graus dos nodos. De seguida, importei o módulo `collections` para recorer à classe `Counter` e poder associar a cada valor de grau o número de nodos em que este se verifica. Por fim uso a função `zip` para extrair esses dois resultados e poder desta forma criar o histograma com os valores corretos no eixo dos `xx` e no eixo dos `yy`, sendo que a abcissa corresponde ao valor do grau e a ordenada ao número de vezes que o mesmo se verifica num nó de uma rede.

```
grau_seq = sorted([gr for n, gr in g.degree()], reverse=True)

grau_count = collections.Counter(grau_seq)
grau, count = zip(*grau_count.items())

fig, ax = plt.subplots()
plt.bar(grau, count, width=0.80, color="b")

# Labels
plt.title("Distribuicao de grau")
plt.ylabel("Contador")
plt.xlabel("Grau")
ax.set_xticks([gr + 0.4 for gr in grau])
ax.set_xticklabels(grau)

plt.show()
```

Listing 2: Criar histograma da distribuição de grau

O resultado obtido foi o seguinte histograma:

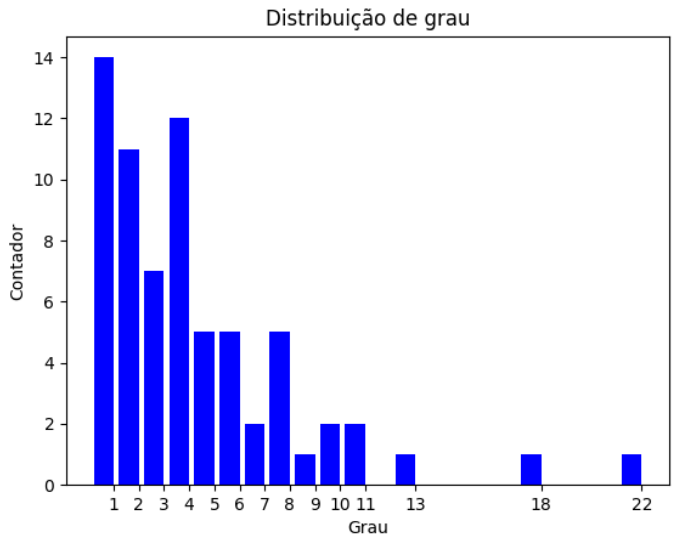

Figura 1: Histograma da distribuição de grau

4.2 DIÂMETRO

O diâmetro de um grafo é a excentricidade máxima, sendo que a excentricidade é a maior distância entre um nodo e todos os outros. Dito de outra forma, é a máxima distância entre um par de vértices, ou seja, o maior *subpath*.

Existe um método `diameter` no módulo `Networkx` para calcular o diâmetro de uma rede. Porém, no grafo dado, tentar calcular o diâmetro por este método daria um erro "*graph is not connected*". Neste caso, há que dividir o grafo nos seus componentes conectados, ou seja, nos seus subgrafos, cujos diâmetro apresento no *output* do programa.

```
1 S = [g.subgraph(c).copy() for c in net.connected_components(g)]
2
3 for s in S:
4     print(net.diameter(s))
```

Listing 3: Dividir o grafo nos seus subgrafos

Iterando sobre estes subgrafos escolho o que tem o maior diâmetro, e esse é portanto o diâmetro do grafo g.

```
sub = 0
for s in S:
    if (net.diameter(s)) > sub:
        sub = net.diameter(s)
```

Listing 4: Cálculo do diâmetro do grafo

O *output* neste caso é então:

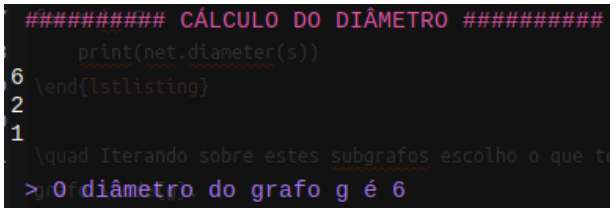

Figura 2: Output do diâmetro.

4.3 DISTÂNCIA

No que toca à distância, podemos considerar várias vertentes da rede em questão, desta vez tendo em conta nodos específicos.

4.3.1 GRAU DE UM NODO

Por um lado, podemos analisar o grau de um nodo, ou seja, o número de nodos que são adjacentes a esse nodo.

Para calcular o grau existe existe a função `degree` da *NetworkX*. O programa lista o grau de cada um dos nodos do grafo.

```
for n in all_nodos:
    degree = net.degree(g,n)
    print(n,":",degree)
    time.sleep(0.05)
```

Listing 5: Calcular grau de cada um dos nodos

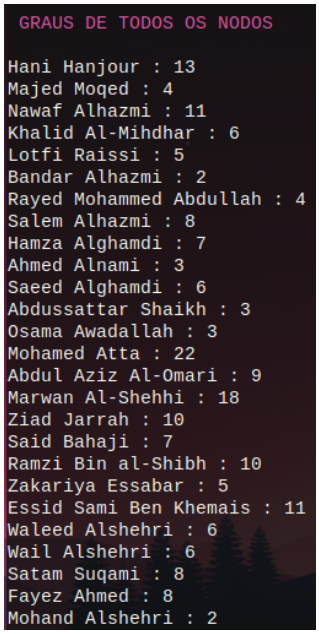

Figura 3: Parte do output dos graus dos nodos

4.3.2 SHORTEST PATH ENTRE DOIS NODOS

É possível também calcular o menor caminho (shortest path) entre dois nodos e o seu comprimento, através dos métodos `shortest_path` e `shortest_path_length`.

No programa é apresentada uma lista de todos os nodos numerados e é pedido para inserir como *input* os números correspondentes a dois nodos, para ser calculado qual o *shortest path* entre eles e qual o seu comprimento.

```
# Listagem dos nodos para escolha do user
ind = 0
for n in all_nodos:
    print(ind,":",n)
    ind += 1
val1 = input("Escolha o primeiro nodo usando a numeracao acima: ")
val2 = input("Escolha o segundo nodo: ")

if (int(val1) < len(all_nodos)) and (int(val2) < len(all_nodos)):
    nodo1 = all_nodos[int(val1)]
    nodo2 = all_nodos[int(val2)]
    try:
        net.shortest_path(g, nodo1, nodo2)
        sp = net.shortest_path(g, nodo1, nodo2)
        spl = net.shortest_path_length(g, nodo1, nodo2)
```

```
16          print("\n")
17          print(colored('> Caminho mais pequeno entre os nodos escolhidos:', '
        blue'),sp)
18          print(colored('> Comprimento:', 'blue'),spl)
19      except:
20          print(colored('\n > Nao existe path entre os dois nodos escolhidos!'
        , 'blue'))
21  else :
22      print("\n > Os nodos inseridos sao invalidos")
```

Listing 6: Cálculo do menor caminho entre dois nodos e o seu comprimento.

```
 SHORTEST PATH ENTRE DOIS NODOS

0 : Hani Hanjour
1 : Majed Moqed
2 : Nawaf Alhazmi
3 : Khalid Al-Mihdhar
4 : Lotfi Raissi
5 : Bandar Alhazmi
6 : Rayed Mohammed Abdullah
7 : Salem Alhazmi
8 : Hamza Alghamdi
9 : Ahmed Alnami
10 : Saeed Alghamdi
11 : Abdussattar Shaikh
12 : Osama Awadallah
13 : Mohamed Atta
14 : Abdul Aziz Al-Omari
15 : Marwan Al-Shehhi
```

Figura 4: Parcial do output para a escolha de dois nodos para calcular o shortest path

```
Escolha o primeiro nodo usando a numeração acima: 10
Escolha o segundo nodo: 15


> Caminho mais pequeno entre os nodos escolhidos: ['Saeed Alghamdi', 'Hamza Alghamdi', 'Marwan Al-Shehhi']
> Comprimento: 2
```

Figura 5: Exemplo de input e output para o shortest path

4.3.3 ÁRVORE DOS SHORTEST PATH (BFS & DFS)

Outra opção é retornar a **ÁRVORE DOS SHORTEST PATHS** entre um nodo e todos os outros nodos, recorrendo a algoritmos de procura que a *NetworkX* disponibiliza.

No meu programa utilizei primeiramente a travessia em largura (**BREADTH FIRST SEARCH** ou **BFS**). Novamente, peço o *input* de um nodo pelo *user*, e de

seguida calculo-a, através do método `bfs_tree`, dando como argumentos o grafo inicial e o nodo recebido como *input* do *user*. Depois de desenhar a rede, recorro ao módulo `matplotlib` para a apresentar.

```
# ... codigo para a selecao um nodo pelo user ... #

plt.figure(figsize=(100,50))

if int(val) < len(list(g.nodes)):
    nodo = all_nodos[int(val)]
    S = net.bfs_tree(g,nodo)
    net.draw_networkx(S)
    print(colored('\n A apresentar a arvore BFS para o nodo', 'blue'),
    colored(nodo, 'blue'),colored("...", 'blue'))
    time.sleep(4)
    plt.title("BFS")
    plt.axis('on')
    plt.show()
else :
    print("\n > Nodo invalido.")
```

Listing 7: Breadth First Search da árvore dos caminhos mais curtos

```
66 : Abu Zubeida
67 : Mohammad Pervez
68 : Madjid Sahoune


Insira o nodo usando a numeração acima: 44

 A apresentar o grafo BFS para o nodo Nabil al-Marabh ...
```

Figura 6: Exemplo de um parcial do output do terminal para a BFS

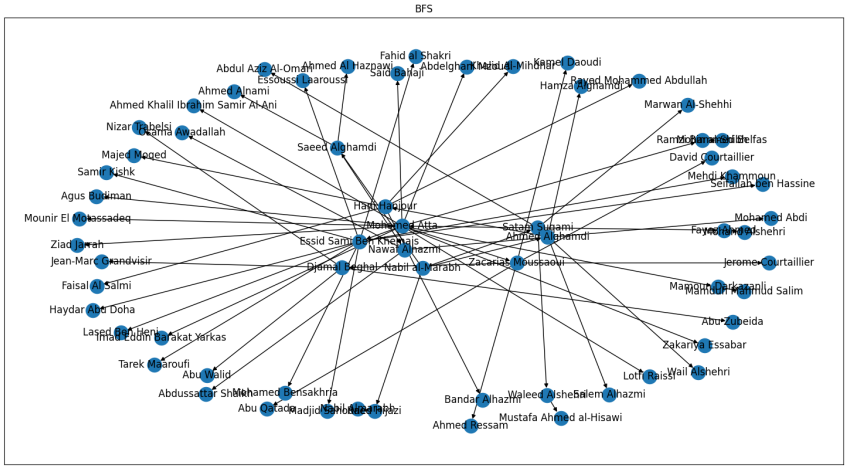

Figura 7: Exemplo de uma *tree* de *shortest paths* com o algoritmo BFS

Com a mesma lógica, prossigo ao cálculo da árvore dos *shortest paths* de um nodo a todos os outros, mas desta vez com a procura **DEPTH FIRST** ou **DFS**.

```
# ... codigo para a selecao um nodo pelo user... #

plt.figure(figsize=(100,50))

if int(val) < len(list(g.nodes)):
    nodo = all_nodos[int(val)]
    S = net.dfs_tree(g,nodo)
    net.draw_networkx(S)
    print(colored('\n A apresentar a arvore DFS para o nodo', 'blue'),
    colored(nodo, 'blue'),colored("...", 'blue'))
    time.sleep(4)
    plt.title("DFS")
    plt.axis('on')
    plt.show()
else :
    print("\n > Nodo invalido.")
```

Listing 8: Depth First Search da árvore dos caminhos mais curtos

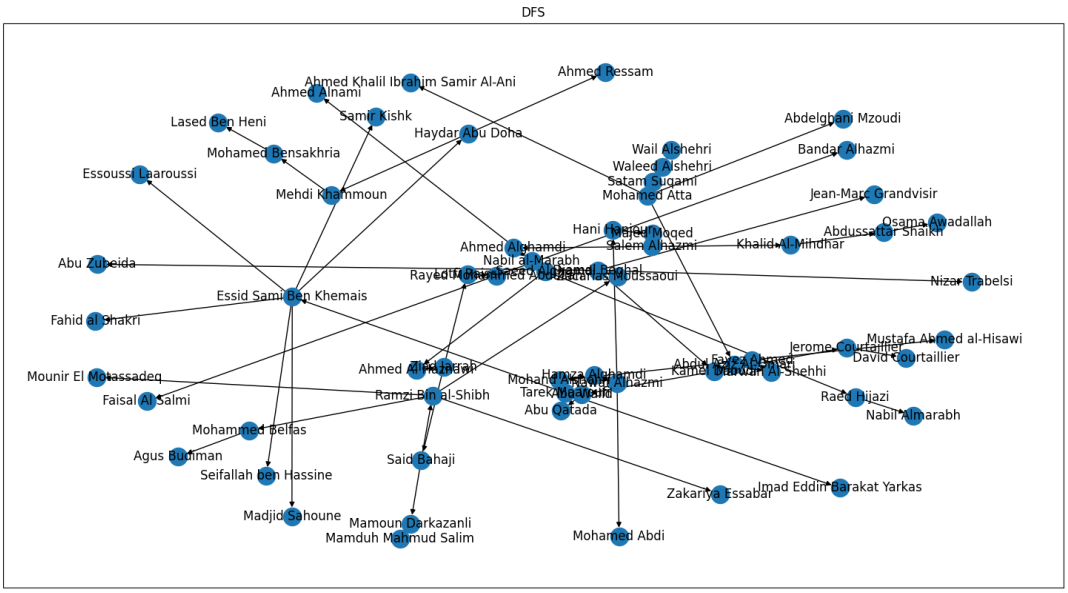

Figura 8: Exemplo de uma *tree* de *shortest paths* com o algoritmo DFS

4.3.4 ECCENTRICY

→ Para calcular unicamente a maior distância entre um nodo e todos os outros, ou seja, a eccentricy, utiliza-se o método `eccentricity` da *NetworkX*, dando como parâmetros o grafo e o nodo em análise. Novamente, é necessáro dividir o grafo em subgrafos. É pedido como input o número que corresponde a um nodo, mostrando inicialmente o mesmo *ouput* apresentado no *shortest path* que associa a cada nodo um número.

```
# ... codigo para a selecao um nodo pelo user... #

S = [g.subgraph(c).copy() for c in net.connected_components(g)]

if int(val) < len(list(g.nodes)):
    nodo = all_nodos[int(val)]
    for s in S:
        try:
            dist = net.eccentricity(s,nodo)
            if(dist) :
                print(colored('\n > A maior distancia entre o nodo','blue'),
     colored(nodo,'blue'), colored('e todos os outros e','blue'), colored(
    dist,'blue'))
        except:
            print("erro")
else :
    print("\n > Nodo invalido.")
```

Listing 9: Eccentricy de um nodo

```
63 : Ahmen Hannan
64 : Nizar Trabelsi
65 : Jean-Marc Grandvisir
66 : Abu Zubeida
67 : Mohammad Pervez
68 : Madjid Sahoune


Insira o nodo usando a numeração acima: 65

 > A maior distância entre o nodo Jean-Marc Grandvisir e todos os outros é 6
```

Figura 9: Parcial de um exemplo de output (e input) para a eccentricy

5 MEDIDAS DE CENTRALIDADE

Dentro das medidas de centralidade temos

- **DEGREE CENTRALITY**: calcula o número de conecções de um nodo para todos os outros;
- **EIGENVECTOR CENTRALITY**: o quão importante é um nodo em função de quão bem conectado está
- **CLOSENESS CENTRALITY**: importância de um nodo em função da sua proximidade com os outros da rede
- **BETWEENESS CENTRALITY**: quantifica quantas vezes um nodo aparece nos caminhos mais curtos entre dois nodos

```
print("Centralidade do grau:",net.degree_centrality(g))

print("Eigenvector centrality:",net.eigenvector_centrality(g))

print("Closeness centrality:", net.closeness_centrality(g))

print("Betweeness centrality:",net.betweenness_centrality(g))
```

Listing 10: Cálculo de medidas de centralidade

6 WEBGRAFIA

https://networkx.org/documentation
https://www.programiz.com/python-programming
https://realpython.com/
https://www.datacamp.com/

7 ANEXO A

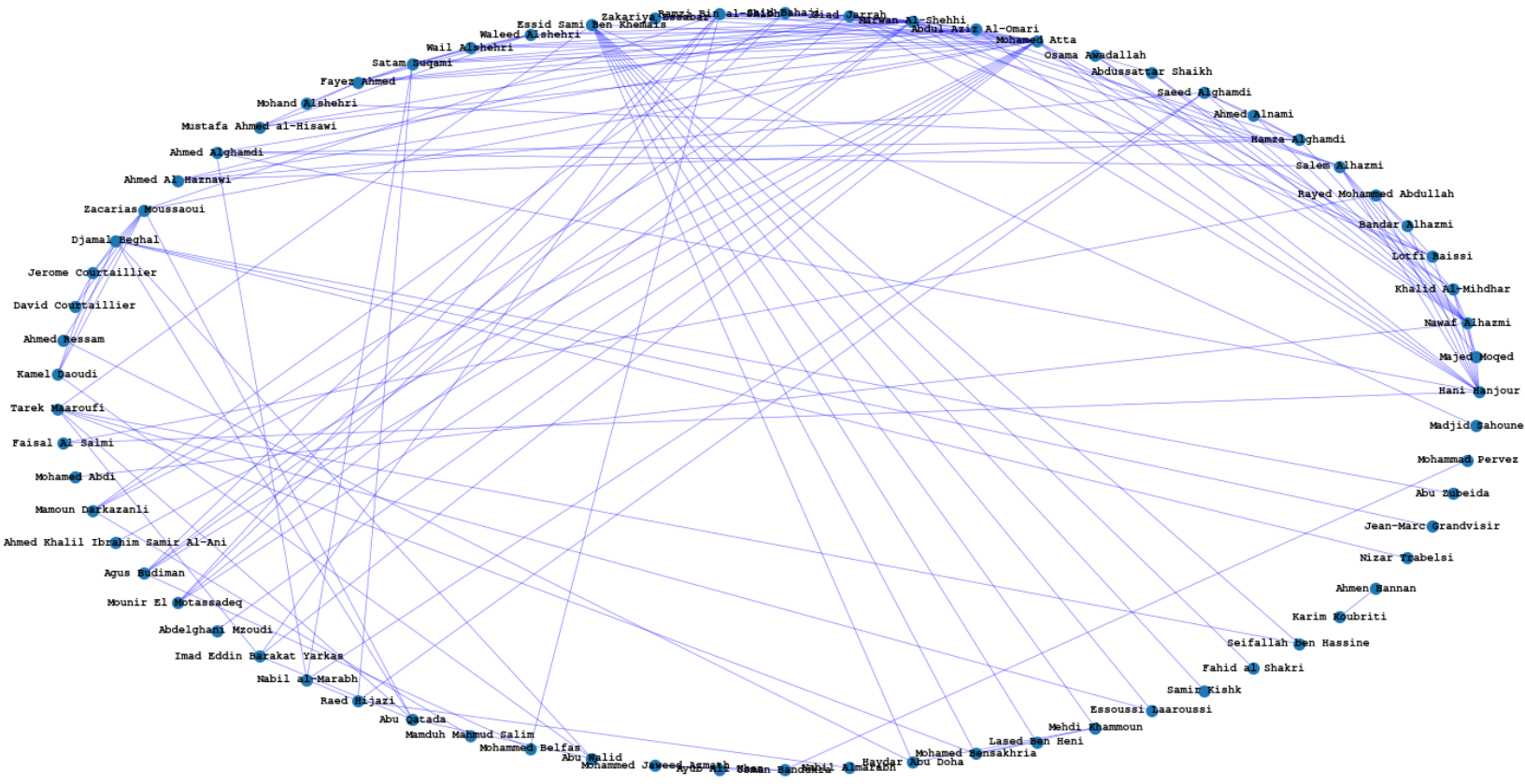

Figura 10: Grafo obtido.