TP1 – Exercício 2

Grupo 1

Diogo Coelho da Silva A100092

Pedro Miguel Ramôa Oliveira A97686

Problema proposto:

2. Um sistema de tráfego é representado por um grafo orientado ligado. Os nodos denotam pontos de acesso e os arcos denotam vias de comunicação só com um sentido. O grafo tem de ser ligado: *entre cada par de nodos* [n1, n2] *tem de existir um caminho* n1 -> n2 *e um caminho* n2 -> n1.
    1. Gerar aleatoriamente o grafo com N ∈ {6...10} nodos e com ramos verificando:
        i. Cada nodo tem um número aleatório de descendentes d ∈ {1... 3}, cujos destinos são também gerados aleatoriamente.
        ii. Se existirem "loops" ou destinos repetidos, deve-se gerar outro grafo.
    2. Pretende-se fazer manutenção interrompendo determinadas vias. Determinar o maior número de vias que é possível remover mantendo o grafo ligado.

Objetivo

O objetivo do problema é:

1. **Gerar um grafo fortemente conectado** com um número aleatório de nós, onde cada nó será conectado a 1 a 3 descendentes de forma aleatória.

2. **Encontrar o conjunto máximo de arestas removíveis**, ou seja, aquelas que podem ser removidas sem que o grafo perca sua conectividade.

3. **Visualizar o grafo** original e o grafo resultante após a remoção dessas arestas.

1.Importação das Bibliotecas

Para resolver este problema, utilizaremos bibliotecas especializadas para manipulação de grafos e visualização. O código começa com a importação das seguintes bibliotecas:

```
from IPython import get_ipython
from IPython.display import display
import networkx as nx
import random
import matplotlib.pyplot as plt
from itertools import combinations
```

- networkx: Biblioteca utilizada para criar e manipular grafos.
- random: Usada para a geração de números aleatórios, necessária na criação dos descendentes dos nós.
- matplotlib: Utilizada para a visualização do grafo.
- itertools: Contém funções que auxiliam na manipulação de estruturas como combinações, usada para testar subconjuntos de arestas removíveis.

2.Função para Gerar um Grafo Fortemente Conectado

A função generate_strongly_connected_graph(N) cria um grafo não direcionado com N nós, garantindo que ele seja conectado. Cada nó é conectado a um número aleatório de 1 a 3 descendentes, escolhidos entre os nós disponíveis.

```
def generate_strongly_connected_graph(N):
    while True:
        G = nx.Graph()  # Inicializa um novo grafo

        # Adicionar os nós ao grafo
        G.add_nodes_from(range(N))

        # Conectar cada nó com 1 a 3 descendentes
        for node in G.nodes():
            num_descendants = random.randint(1, 3)  # Número de descendentes para o nó atual
            potential_descendants = []  # Lista para armazenar descendentes potenciais

            # Encontrar descendentes potenciais
            for n in G.nodes():
```

```
            if n != node and not G.has_edge(node, n):  # Verifica se o nó não é o mesmo e se não existe aresta
                potential_descendants.append(n)

        # Selecionar descendentes aleatórios
        descendants = []
        number_of_choices = min(num_descendants, len(potential_descendants))  # Garantir que não exceda disponíveis
        if number_of_choices > 0:
            selected_indices = random.sample(range(len(potential_descendants)), number_of_choices)
            for index in selected_indices:
                descendants.append(potential_descendants[index])  # Adiciona os descendentes selecionados

        # Adicionar as arestas entre o nó e seus descendentes
        for descendant in descendants:
            G.add_edge(node, descendant)

    # Verificar se o grafo é conectado
    if nx.is_connected(G):  # Retorna o grafo se for conectado
        return G
```

Esta função executa um ciclo que só é interrompido quando o grafo gerado é conectado (verificado pela função nx.is_connected(G)).

Função para Encontrar Arestas Removíveis

A função find_max_removable_edges(G) identifica o conjunto máximo de arestas que podem ser removidas do grafo sem que ele perca a sua conectividade. Ela testa combinações de arestas e retorna aquelas que, se removidas, não afetam a conectividade global do grafo.

```
def find_max_removable_edges(G):
    max_removable_edges = set()  # Conjunto para armazenar as arestas removíveis
    # Verifica todas as combinações de arestas do grafo
    for i in range(len(G.edges()), 0, -1):
        for edge_set in combinations(G.edges(), i):
            G_temp = G.copy()  # Cria uma cópia do grafo
            G_temp.remove_edges_from(edge_set)  # Remove um conjunto de arestas
            if nx.is_connected(G_temp):  # Verifica se a cópia ainda está conectada
                return set(edge_set)  # Retorna as arestas removíveis
    return max_removable_edges  # Retorna conjunto vazio se nenhuma aresta puder ser removida
```

Aqui, são verificadas todas as combinações de arestas do grafo, começando pelas maiores, e testado se sua remoção mantém o grafo conectado.

Função para Visualizar o Grafo

A função plot_graph(G, title) é responsável por visualizar o grafo utilizando matplotlib.

```
def plot_graph(G, title):
    pos = nx.spring_layout(G)  # Define a posição dos nós
    plt.figure(figsize=(8, 6))  # Define o tamanho da figura

    # Desenha os nós
    nx.draw_networkx_nodes(G, pos, node_color='lightblue', node_size=2000)
    # Desenha as arestas
    nx.draw_networkx_edges(G, pos, edge_color='gray', arrows=True, arrowstyle='->', arrowsize=20)
    # Desenha os rótulos dos nós
    nx.draw_networkx_labels(G, pos, font_size=15, font_color='black', font_weight='bold')

    plt.title(title)  # Define o título do gráfico
    plt.show()  # Mostra o gráfico
```

Execução do Código

O código a seguir executa todas as etapas: gera um grafo fortemente conectado, encontra as arestas removíveis, remove-as e exibe tanto o grafo original quanto o grafo resultante.

```
# Geração do grafo
N = random.randint(6, 10)  # Número aleatório de nós
G = generate_strongly_connected_graph(N)  # Gera o grafo

print(f"Grafo original com {N} nós:")
print(f"Arestas do grafo original: {list(G.edges())}")
plot_graph(G, "Grafo Original")  # Plota o grafo original

# Encontrar arestas removíveis
removable_edges = find_max_removable_edges(G)
```

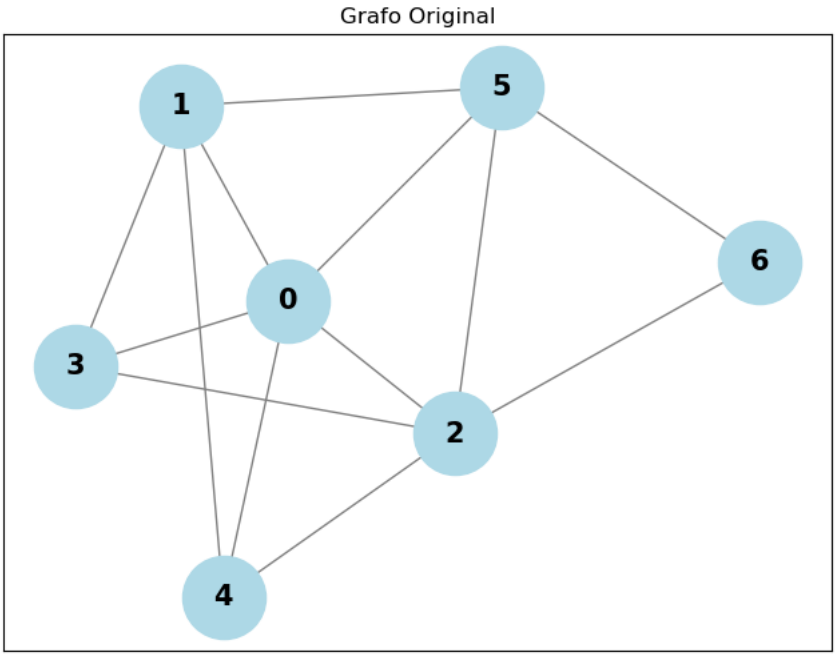
```
# Criar um novo grafo após a remoção das arestas
G_result = G.copy()
G_result.remove_edges_from(removable_edges)

print(f"Número de arestas removíveis mantendo o grafo ligado: {len(removable_edges)}")
print(f"Arestas removíveis: {removable_edges}")

# Verificar se as arestas removíveis existem no grafo original
for edge in removable_edges:
    if not G.has_edge(*edge):
        print(f"Erro: a aresta {edge} não existe no grafo original!")

print("Grafo resultante após remoção das arestas:")
plot_graph(G_result, "Grafo Após Remoção das Arestas")  # Plota o grafo após remoção
```

Conclusão

Este código resolve o problema de geração e manipulação de um grafo fortemente conectado. Através dele, é possível identificar as arestas que podem ser removidas mantendo a conectividade do grafo e visualizar o grafo original e o grafo resultante após a remoção das arestas.

Grafo original com 7 nós:

Arestas do grafo original: [(0, 5), (0, 1), (0, 2), (0, 3), (0, 4), (1, 3), (1, 4), (1, 5), (2, 5), (2, 3), (2, 4), (2, 6), (5, 6)]



Número de arestas removíveis mantendo o grafo ligado: 7

Arestas removíveis: {(0, 1), (0, 3), (1, 4), (0, 2), (0, 5), (2, 5), (1, 3)}

Grafo resultante após remoção das arestas: