Trabalho Prático 1

Grupo 22

Alexis Correia - A102495 João Fonseca - A102512

Exercício 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 (n_1, n_2) tem de existir um caminho $n_1 -> n_2$ e um caminho $n_2 -> n_1$.

1. Gerar aleatoriamente o grafo com $N \in \{6..10\}$ nodos e com ramos verificando:
   a. Cada nodo tem um número aleatório de descendentes $d \in \{0..3\}$, cujos destinos são também gerados aleatoriamente.
   b. 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.

Resolução

O primeiro passo na resolução deste problema é criar um grafo aleatório de acordo com as intruções acima. E, com auxílio das bibliotecas importadas (nomeadamente `networkx`), podemos desenhar o grafo de forma a facilitar a visualização do mesmo.

```
#Inicialização

import networkx as nx
import random
from ortools.linear_solver import pywraplp
#Criação Grafo

def ligado(adj, N): # A função confirma que o grafo criado é ligado
    res = True
    n = 0
    while res and  n < N:
        visited = set()
        queue = [n]
        visited.add(n)
        while queue:
            nodo = queue.pop(0)
            for d in adj[nodo]:
                if d not in visited:
                    visited.add(d)
                    queue.append(d)
```

```
        if len(visited) < N:
            res = False
        n += 1
    return res

random.seed(None)

def cria_adj(N): # Função que cria o grafo aleatóriamente de acordo
com as intruções do enunciado
    adj = {n: [] for n in range(N)}

    for n in range(N): # Adiciona, no máximo, três arestas com
destinos aleatórios (para cada nodo)
        num_arestas = random.randint(1, 3) # Seleciona um número
aleatório entre 1 e 3 (inclusos)
        i = 0
        while i < num_arestas: #o ciclo while permite selecionar outro
destino para a aresta
            d = random.randint(0, N - 1)    # se o d não obedecer aos
requisitos
            if d != n and d not in adj[n]: # Dessa forma, garantimos
que não haja "loops" e repetições
                adj[n].append(d)
                i += 1
    if not ligado(adj,N):
        adj = cria_adj(N)
    return adj

N = 8
adj = cria_adj(N) #Criamos um dicionário de adjacências: adj
print(adj)

G = nx.DiGraph(adj) # Criamos o grafo a partir do dicionário "adj"
nx.draw(G, with_labels=True)
```

Com o grafo criado de forma aleatória, podemos começar a utilizar a programação inteira para determinar quantas arestas do grafo podemos retirar de forma que o grafo permaneça ligado.

```
solver = pywraplp.Solver.CreateSolver('SCIP')
O=D=N # Substituimos N por O ou D para facilitar a compreensão do
código
X={}
for o in range(O):
    for d in range(D):
        X[o,d] = solver.BoolVar(f"X[{o},{d}]")
```

Agora podemos adicionar as restrições necessárias:

1. Primeiramente, não podemos adicionar novas arestas ao grafo. Logo, se uma aresta não estiver no grafo (ou seja, no dicionário de adjacências), então o valor será zero.

$$\forall o \in O, d \in D . d \notin adj[o] \rightarrow X[o,d] = 0$$

Para que o grafo possa ser ligado, como desejado, é preciso que cada nodo n tenha pelo menos uma aresta cujo n é origem e uma areste no qual n é destino. Isso pode ser traduzido nas duas restrições a seguir:

$$\forall o \in N, \sum_{d \in N} X_{o,d} \geq 1$$

$$\forall d \in N, \sum_{o \in N} X_{o,d} \geq 1$$

Por fim, garantimos que haja um limite superior quanto ao número de arestas que chegam e partem de cada nodo.

$$\forall n \in N, \sum_{d \in N} X_{n,d} + \sum_{o \in N} X_{o,n} \leq 3$$

```
#R1
for o in range(O):
    for d in range(D):
        if d not in adj[o]:
            solver.Add(X[o,d] == 0)

#R2
for o in range(O):
    solver.Add(sum([X[o,d] for d in range(D)])>=1)

#R3
for d in range(D):
    solver.Add(sum([X[o,d] for o in range(O)])>=1)

#R4
for n in range(N):
    solver.Add(sum([X[n,d] for d in range(D)])+sum([X[o,n] for o in
range(O)])<=3) #2?
```

Por fim, resolvemos o problema e podemos determinar quantas arestas podem ser removidas do grafo sem que o mesmo deixe de ser ligado.

```
status = solver.Solve()
if status == pywraplp.Solver.OPTIMAL:
    print("X", end=" | ")
    for d in range(N):
        print(d, end = " | ")
    print()
    aux = 0
    ed = []
    for o in range(N):
```

```
        print(o, end = " | ")
        for d in range(N):
            if X[o,d].solution_value() == 1:
                print("1", end = " | ")
                aux += 1
                ed.append((o,d))
            else:
                print("0", end = " | ")
        print()

    print(f"Início: {G.number_of_edges()} arestas")
    print(f"Fim: {aux} arestas")
    print(f"Resposta: Podemos fechar {G.number_of_edges()-aux} ruas")

    #print(adj)

    R = nx.DiGraph(ed)
    nx.draw(R, with_labels = True)
else:
    print("ERROR")
```