Atividade 5 - Fluxos (parte 1)

Filipe Moreira Mateus

Como executar:

- Em uma máquina com Python instalado basta executar o comando `python main.py` para iniciar o programa.
- O programa lê grafos de arquivos em que cada linha representa uma aresta, e possui três valores, representando os vértices, separados por espaços: origem destino valor. Um exemplo de arquivo é o arquivo `g` que contém o grafo que aparece ná pagina 8 dos slides da aula:

```
1 2 50
1 4 40
2 3 60
4 3 70
4 5 60
3 6 30
5 6 50
```

- No início da execução o programa pede que o usuário entre com o nome do arquivo que contém o grafo. Para entrar com o grafo acima basta digitar `g` e teclar enter.
- O grafo é carregado na estrutura de dados implementada na Atividade 1.
- Em seguida o programa exibe o vértice fonte e o alvo além de uma listagem com suas arestas associadas a um índice:

```
Insira o nome do arquivo que contém o Grafo: g

Vértice fonte: 1, Vértice alvo 6

0 - Origem: 1 Fim: 2 Peso: 50
1 - Origem: 1 Fim: 4 Peso: 40
2 - Origem: 2 Fim: 3 Peso: 60
3 - Origem: 4 Fim: 3 Peso: 70
4 - Origem: 4 Fim: 5 Peso: 60
5 - Origem: 3 Fim: 6 Peso: 30
6 - Origem: 5 Fim: 6 Peso: 50
```

- Em seguida o usuário deve indicar quantas arestas compõem o conjunto de arestas que deseja cortar, e também o índice de cada uma dessas arestas.
- Após isso o programa calcula se aquele conjunto realiza um corte, e a capacidade desse corte caso ele exista.

- Aqui vão alguns exemplos de entradas e saídas para o grafo da página 8:

```
Insira o número de arestas que compõem o corte: 2
Qual aresta deseja cortar: 0
Qual aresta deseja cortar: 1

Arestas após cortes:
2 - Origem: 2 Fim: 3 Peso: 60
3 - Origem: 4 Fim: 3 Peso: 70
4 - Origem: 4 Fim: 5 Peso: 60
5 - Origem: 3 Fim: 6 Peso: 30
6 - Origem: 5 Fim: 6 Peso: 50

O conjunto representa um corte de capacidade 90
```

```
Insira o número de arestas que compõem o corte: 2
Qual aresta deseja cortar: 5
Qual aresta deseja cortar: 6

Arestas após cortes:
0 - Origem: 1 Fim: 2 Peso: 50
1 - Origem: 1 Fim: 4 Peso: 40
2 - Origem: 2 Fim: 3 Peso: 60
3 - Origem: 4 Fim: 3 Peso: 70
4 - Origem: 4 Fim: 5 Peso: 60

O conjunto representa um corte de capacidade 80
```

```
Insira o número de arestas que compõem o corte: 2
Qual aresta deseja cortar: 1
Qual aresta deseja cortar: 5

Arestas após cortes:
0 - Origem: 1 Fim: 2 Peso: 50
2 - Origem: 2 Fim: 3 Peso: 60
3 - Origem: 4 Fim: 3 Peso: 70
4 - Origem: 4 Fim: 5 Peso: 60
6 - Origem: 5 Fim: 6 Peso: 50

O conjunto representa um corte de capacidade 70
```

```
Insira o número de arestas que compõem o corte: 2
Qual aresta deseja cortar: 2
Qual aresta deseja cortar: 4

Arestas após cortes:
```

```
0 - Origem: 1 Fim: 2 Peso: 50
1 - Origem: 1 Fim: 4 Peso: 40
3 - Origem: 4 Fim: 3 Peso: 70
5 - Origem: 3 Fim: 6 Peso: 30
6 - Origem: 5 Fim: 6 Peso: 50

O conjunto não representa um corte
```

```
Insira o número de arestas que compõem o corte: 3
Qual aresta deseja cortar: 0
Qual aresta deseja cortar: 3
Qual aresta deseja cortar: 6

Arestas após cortes:
1 - Origem: 1 Fim: 4 Peso: 40
2 - Origem: 2 Fim: 3 Peso: 60
4 - Origem: 4 Fim: 5 Peso: 60
5 - Origem: 3 Fim: 6 Peso: 30

O conjunto representa um corte de capacidade 170
```

Funções:

- O primeiro passo para implementação foi encontrar os vértices fonte e alvo. Isso pode ser feito buscando aqueles vértices que não recebem nenhuma aresta como entrada, ou como saída respectivamente:

```
def get_source_target(self):
    for aresta1 in self.arestas:
        is_source = True
        is_target = True
        for aresta2 in self.arestas:
            if aresta1.de == aresta2.para:
                is_source = False
            if aresta1.para == aresta2.de:
                is_target = False
        if is_source:
            source = aresta1.de
        if is_target:
            target = aresta1.para
    return source, target
```

- Em seguida é necessário "cortar" as arestas desejadas, para isso minha abordagem foi inserir um peso muito grande (um bilhão) aos vértices. Dessa forma se torna fácil identicicar se um caminho passa por ele ou não.

```
def corta_aresta(self, aresta):
    if aresta in range(0,len(self.arestas)):
        self.arestas[aresta].peso = 10**9
    else:
        print("índice inválido para aresta!")
```

- Em seguida é necessário verificar se após os "cortes" ainda é possível atingir o vértice alvo a partir do vértice fonte sem passar pelos vértices cortados. Para isso aplicamos o algoritmo Floyd-Warshall e verificamos se a distância entre fonte e alvo é menor que um bilhão (isso indica que não passou por algum vértice "cortado").

```
def dist_source_target(self):
    source, target = self.get_source_target()
    source, target = int(source)-1, int(target)-1
    dist = self.floyd_warshall()
    return dist[source][target]
```