Teoria dos Grafos Aplicados
Exercício de programação

Prof. Miguel Aroztegui, DCC - CI - UFPB

O exercício pretende estimular a criatividade na programação python com com networkx (https://networkx.org/) visando a solução computacional de problemas da teoria dos grafos.

Regras:

R1 A equipe de trabalho pode ter até dois integrantes.

R2 A realização das **Tarefas** devem estar em um único arquivo com nome `ep_2_<mat>.py` (onde `<mat>` é o número de matrícula de um dos integrantes da equipe). Deve ser entregue por e-mail até o prazo estabelecido neste documento.

R3 O e-mail deve ter as seguintes características: *to*: `jose.miguel@ci.ufpb.br`. *Subject*: TGA: exercício de programação 2. *Body*: 1) Nomes dos intergrantes da equipe. 2) Versão do python empregado para executar o código. *Attachment*: `ep_2_<mat>.py`

R4 Prazo: enviar o e-mail até segunda-feira 7 de outubro de 2024, meia noite.

R5 Escrever um código focado na solução da tarefa. Não adicionar funcionalidades não solicitadas. Tentar, dentro do possível, usar o menor número de linhas sem que isso prejudique a legibilidade e comprensão do código.

R6 Quando se pede definir `name` com entradas `i1,...,iK,d1=D1,...,dN=DN` e saídas `o1,...,oM`, deve ser programado da seguinte forma:

```
def name(i1, ..., iK, d1 = D1, ..., dN = DN):
  ...
  return o1, ..., oN
```

R7 Em **Procedimentos sugeridos** se enumeram alguns procedimentos do networkx que podem ajudar a resolução das tarefas. Caso a equipe considere oportuno empregar uma outra função do networkx, esta deve estar sinalizada com um comentário no código.

Procedimentos sugeridos:

```
import networkx as nx
nx.Graph
nx.get_node_attributes
nx.get_edge_attributes
nx.draw_networkx
nx.draw_networkx_edge_labels
nx.algorithms.shortest_paths.weighted.single_source_dijkstra
nx.dfs_edges
```

Tarefas: Para melhorar o entendimento das tarefas, faça a leitura destas acompanhada com a leitura do **Exemplo** na página seguinte.

T1: Definir `ugraph` com entradas `V,C,E,W=None` e saída `G`. Esta cria um grafo não dirigido `G`, com nós etiquetados na lista `V`, arestas na lista `E` e cujos pesos estão na lista `W`. O nó `V[i]` tem coordenadas `C[i]`. A aresta `E[i]` tem peso `W[i]`.

T2: Definir `plot_ugraph` com entradas `G,weighted=False`. Ela desenha o grafo `G` criado com `ugraph`.

T3: Definir `dijkstra` com entradas `G,s,f=None` e saída `edges`. Esta função retorna as arestas `edges` do caminho mais curto do nó `s` para o nó `f` de um grafo ponderado `G`. Se `f=None`, então retorna em `edges` as arestas de todos os caminhos mais curtos de `s` para todos os demais nós de `G`.

T4: Definir `connected` com entrada `G` e saída b. Retorna `b=True` se o grafo `G` é conectado e `b=False` caso contrário.

T5: Definir `bridge` com entrada `a,G` e saída b. Retorna `b=True` se a aresta `a` no grafo `G` é uma ponte e `b=False` caso contrário.

T6: Definir `cycle` com entrada `G` e saída *edges*. Retorna em `edges` as arestas de um ciclo no grafo `G` e `edge=None` caso o grafo `G` não contenha ciclos.

Exemplo:

```
import matplotlib.pyplot as plt—
V = [ 'a',  'b',  'c',  'd' ]
C = [(0,0),(1,0),(0,1),(1,1)]
E = [('b','a'),('b','c'),('a','c')]
W = [    3,        2,        7    ]
############ T1: ugraph ############
G1 = ugraph(V, C, E)    # cria G1 sem pesos W
G2 = ugraph(V, C, E, W) # cria G2 com pesos W
############ T2: plot_ugraph ############
plt.figure()
plot_ugraph(G1) # desenha G1 (não ponderado)
plt.axis('equal')
...
plot_ugraph(G2) # desenha G2 sem os pesos W
...
plot_ugraph(G2,weight=True) # desenha G2 com pesos W
...
# Recomenda-se usar plt.figure() para desenhar o grafo
# em uma janela e plt.axis('equal') para desenhar os nós
# com coordenadas C com a mesma escala na horizontal
# e na vertical.
############ T3: dijstra ############
edges=dijkstra(G2,'b','c') # edges=dijkstra(G2,'b')
G3 = ugraph(V, C, edges, W)
plt.figure()
plot_ugraph(G3,weighted=True) # plot short path from 'b' to 'c'
plt.axis('equal')
############ T4: connected ############
b1 = connected(G1) # b1 é False
G1.add_adge('c','d')
b2 = connected(G1) # b2 é True
############ T5: bridge ############
b1 = bridge(G1,('a','b')) # b1 é False
b2 = bridge(G1,('c','d')) # b2 é True
############ T6: cycle ############
edges = cycle(G1)
G4 = ugraph(V, C, edges)
plt.figure()
plot_ugraph(G4) # desenha o ciclo encontrado
plt.axis('equal')
```