9 Aula 09: 29/AGO/2019

9.1 Aulas passadas

- Pilhas: sequências bem formadas e notação polonesa.
- NumPy : `arrays`

9.2 Hoje

Problema: distâncias

Com isso, na aula de hoje, as alunas e alunos acabarão vendo:

- filas em *busca em largura*;
- classe `Queue` (= fila);
- representação de redes: matriz de adjacências;
- classe `Rede` fica para a próxima aula.

9.3 Distâncias

Considere `n` cidades numeradas de 0 a `n-1` interligadas por estradas de mão única.

O **comprimento** de um caminho é o número de estradas no caminho.

A **distância** de uma cidade `i` a uma cidade `j` é o menor comprimento de um caminho de `i` a `j`. Se não existe caminho de `i` a `j` a distância é *infinita*.

Problema: dada uma rede de estradas e uma cidade `c`, determinar a distância de `c` a cada uma das demais cidades.

9.4 Representação da rede

A rede de estradas será representada por um `array rede` tal que:

`rede[i,j] = True` se **existe** estrada da cidade `i` para a cidade `j`.

`rede[i,j] == False` se **não existe** estrada da cidade `i` para a cidade `j`.

9.4.1 Solução

```
import numpy as np
from queue import Queue

#-------------------------------------------------------------
def distancias(c, rede):
    '''(int, array) -> list

    Recebe o índice c de uma cidade e uma rede de estradas
    com n cidades.
```

```
    A função cria e retorna uma lista d[0:n] tal que para
    i = 0,...,n-1, d[i] é a distância da cidade c a cidade i.

    Se não existe caminho da cidade c a cidade i então d[i]=n.
    '''
    # pegue o número de cidades da rede
    n = len(rede)

    # crie o vetor de distancia com 'infinito' em cada posição
    d = np.full(n, n)

    # a distancia da origem a si mesma e zero
    d[c] = 0

    # crie a fila de cidades
    q = Queue()

    # coloque a cidade origem na fila
    q.enqueue(c)

    while not q.isEmpty():
        # i será o 1a. cidade na fila
        i = q.dequeue()

        # examine as cidades vizinhas da cidade i
        for j in range(n):
            if rede[i,j] and d[j] > d[i]+1:
                d[j] = d[i] + 1
                q.enqueue(j)
    return d
```

9.5 Queue

Uma **fila** (do inglês *queue*) é uma lista dinâmica em que todas as inserções são feitas em uma extremidade chamada de **fim** e todas as remoções são feitas na outra extremidade chamada de **início**.

9.5.1 Classe

```
class Queue:
    def __init__(self):
        self.itens = []

    def __str__(self):
        return str(self.itens)

    def isEmpty(self):
        return self.itens == []

    def enqueue(self, item):
        self.itens.append(item) # self.itens.insert(0, item)

    def dequeue(self):
        return self.itens.pop(0) # return self.itens.pop()

    def size(self):
        return len(self.itens)

    def __len__(self):
        return len(self.itens)
```

9.6 Main

```
from rede import Rede
from queue import Queue

def main():
    nome_arquivo = input("Digite o arquivo com a rede: ")

    rede = leia_rede(nome_arquivo)

    origem = int(input("Qual é a cidade origem: "))

    # calcule as distancias
    d = distancias(origem, rede)

    # imprima as distancias
    print("Distância da cidade %d a cidade:" %origem)
    for i in range(len(d)):
        print("   ", i, "=", d[i])

#-------------------------------------------------------
def leia_rede(nome_arquivo):
    '''(str) -> Rede

    Recebe um string nome_arquivo com o nome de um arquivo e
    lê desse arquivo a representação de uma rede de estradas.

    A primeira linha do arquivo contém o número n de cidades.

    A segunda linha do arquivo contém o número m de estradas.

    As demais m linhas contém pares de inteiros i e j entre 0..n-1
    indicando que existe uma estrada de i para j.

    Exemplo:

    6
    10
    0 2
    0 3
    0 4
    1 2
    1 4
    2 4
    3 4
    3 5
    4 5
    5 1 # esta linha é o fim do arquivo
```

```
    A rede tem 6 cidades 0,1,...,5 e 10 estradas.
    '''
    # abra o arquivo
    with open(nome_arquivo, 'r', encoding='utf-8') as arquivo:

        # leia do arquivo o número de cidades
        n = int(arquivo.readline())

        # leia do arquivo o número de estradas
        m = int(arquivo.readline())

        # crie uma rede com n cidades e sem estradas
        # rede = Rede(n)
        rede = np.full( (n, n), False)

        for k in range(m):
            linha  = arquivo.readline()
            cidade = linha.split()
            i = int(cidade[0])
            j = int(cidade[1])
            rede[i, j] = True

    # retorne a rede
    return rede

#-------------------------------------------------------------
if __name__ == "__main__":
    main()
```

9.7 Rede

Observação: Não foi feito na aula. Usamos a matriz `adj` diretamente.

```
import numpy as np
class Rede:
    #-------------------------------------------
    def __init__(self, n):
        '''(Rede, int) -> None

        Chamada pelo construtor.
        Recebe um inteiro positivo n e retorna uma Rede
        com n cidades e sem estradas.

        As cidades são números entre 0 e n-1.

        self.adj[i][j] == 1     existe estrada de i a j
        self.adj[i][j] == 0 não existe estrada de i a j
        '''
        self.adj = np.full((n,n), False)

    #-------------------------------------------
    def __str__(self):
        '''(Rede) -> str

        Recebe uma Rede referênciada por self e cria
        e retorna um string que representa a rede.
        '''
        s = "Matriz de adjacência:\n"
        adj = self.adj
        n   = len(adj)
        for i in range(n):
            for j in range(n):
                s += "1 " if adj[i,j] else "0 "
            s += "\n"
        return s

    #-------------------------------------------
    def insira_estrada(self, i, j):
        '''(Rede, int, int) -> None

        Recebe um rede referênciada por self e um par
        de inteiros i e j representando cidades e insere
        na rede a estrada de i a j.
        '''
        self.adj[i,j] = True

    #-------------------------------------------
    def existe_estrada(self, i, j):
        '''(Rede, int, int) -> bool
```

```
        Recebe uma rede referênciada por self e dois
        inteiros i e j representando duas cidades e retorna
        True se existe uma estrada de i a j e False em caso
        contrário.
        '''
        return self.adj[i, j]

    #--------------------------------------------
    def __len__(self):
        '''(Rede) -> int

        Recebe uma referência self e retorna o número de
        cidades na rede.
        '''
        return len(self.adj)
```