Aula 07: 23/AGO/2018

Aulas passadas

Pilhas: sequências bem formadas e notação polonesa.

Hoje

Problema: distâncias

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

- filas em busca em largura;
- classe `Fila`;
- representação de grafos/redes: matriz de adjacências;
- classe `Rede`.

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.

TEMPO PARA PENSAR/SIMULAÇÃO

Representação da rede

A rede de estradas será representada por um objeto rede de uma classe `Rede` tal que:

`rede.existe_estrada(i,j) == True` se **existe** estrada da cidade `i` para a cidade `j`.

`rede.existe_estrada(i,j) == False` se **não existe** estrada da cidade `i` para a cidade `j`.

Solução

```
def distancias(c, rede):
    '''(int, Rede) - 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 à cidade i.
```

```
    Se não existe caminho da cidade c à 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 = n*[n]

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

    # crie a fila de cidades
    q = Fila()  # fila = []

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

    while not q.vazia(): # while fila != []:
        # i será o 1a. cidade na fila
        i = q.remova() # i = fila.pop(0)

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

Fila

Observação: Não foi feito na aula. Usamos os métodos `list.pop()` e `list.append()` diretamente.

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**.

Classe

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

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

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

    def insira(self, item):
        self.itens.append(item)

    def remova(self):
        return self.itens.pop(0)

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

Rede

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

```
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 = []
        for i in range(n):
            linha = n * [0]
            self.adj.append(linha)

    #-------------------------------------------
    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 += str(adj[i][j]) + " "
            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] = 1

    #-------------------------------------------
```

```
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] == 1

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

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

Main

```
from rede import Rede
from fila import Fila

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

    rede = crie_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 crie_rede(nome_arquivo):
    '''(str) -> Rede

    Recebe um string nome_arquivo 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.

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

    Exemplo:

    6
    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())

        # crie uma rede com n cidades e nenhuma estrada
        rede = Rede(n)

        for linha in arquivo:
            cidade = linha.split()
            i = int(cidade[0])
            j = int(cidade[1])
            rede.insira_estrada(i,j)

    # retorne a rede
    return rede

#------------------------------------------------------------
def pause():
    input("Tecle ENTER para continuar.")

#------------------------------------------------------------
main()
```

Apêndice

Comprimentos

```
len(range(10))

len(lista)

len(dicionario)

len(texto)
```

Pertinência

```
if i in range(início,fim,passo):

if chave in dicionario:
    print(chave)

if car in texto:
    print(car)

if item in lista:
    print(item)
```

Percorrer

```
for i in range(início,fim,passo):
    print(i)

for chave in dicionario:
    print(chave)

for car in texto:
    print(car)

for item in lista:
    print(item)
```