10 Aula 10: 10/SET/2019

10.1 Aulas passadas

Problema: distâncias

Com isso, na aula de hoje, veremos:

- filas em *busca em largura*;
- uso da classe `Queue`

Omitimos a contrução da rede

10.2 Hoje

Representação de redes (*grafos*) através de matriz de adjacência e listas de adjacências.

Matrizes de adjacências se dão melhor com redes *densas*:

- `insira(i,j)`: tempo constante
- `existe(i,j)`: tempo constante
- `vizinhas(i)`: tempo linear no número de nós da rede.
- espaço: proporcional a `n x n` onde `n` é o número de nós

Listas de adjacências são melhor com redes *esparsas*:

- `insira(i,j)`: tempo constante; se não verificamos se o vertex já está na lista
- `existe(i,j)`: tempo proporcional ao número de nós na lista, no pior caso
- `vizinhas(i)`: tempo proporcional ao tamanho da lista
- espaço: proporcional a `n + m` onde `n` é o número de nós e `m` é o número de arcos.

Para o digrafo `largeG.txt` as implementações `rede_matriz.py` e `rede_numpy.py` explodem por falta de memória.

As demais implementações, com lista de listas ou dicionário de listas se dão bem, demoram (12s), mas se dão bem.

Observação 1: em MAC0110 usamos redes no EP *Escrevendo como Machado de Assis*: basicamente, os nós eram as palavras e os arcos/flechas correspondiam a palavras seguidas no texto.

Observação 2: podem usar a *estrutura da função* `distância()` no próximo EP07.

10.3 Testes

Os arquivos com redes para testes estão na pasta `redes_txt`

A rede do arquivo `redes_txt/tinyG.txt` é o da imagem `imgs/dag.png`.

O arquivo `redes/mediumG.txt` contém uma rede com 250 nós.

O arquivo `redes_txt/largeG.txt` contém uma rede com 1M nós e 7M> arcos. Com esse arquivo redes representadas com matrizes explodem. Só redes com listas e dicionário de listas aguentam.

O arquivo `redes_txt/largeGG.txt` contém uma rede com 1M nós e 8M> arcos. Com esse arquivo redes representadas com matrizes explodem. Só redes com listas e dicionário de listas aguentam. Nesse arquivo o programa de distâncias trabalha bastante.

10.4 Programas

Os arquivos com os programas estão no diretório `py`.

Os arquivos com as implementações das redes estão em `py/redes/`

10.4.1 `distancias.py`

```
arquivo: redes_txt/tinyG.txt
criando rede
rede criada
Distância da cidade 8 a cidade (< 13):
    0 : 13
    1 : 13
    2 : 13
    3 : 13
    4 : 3
    5 : 13
    6 : 2
    7 : 1
    8 : 0
    9 : 3
    10 : 4
    11 : 4
    12 : 4
```

Com array:

```
arquivo redes_txt/largeG.txt
criando rede
Killed
```

Com listas:

```
arquivo: redes_txt/largeG.txt
criando rede
rede criada:    12.24s
```

Distâncias

```
arquivo: redes_txt/tinyG.txt
criando a rede
rede criada
calculando distâncias
distancias calculadas:     0.00s
Distância da cidade 8 a cidade (< 13):
    0 : 13
    1 : 13
    2 : 13
    3 : 13
    4 : 3
    5 : 13
    6 : 2
    7 : 1
    8 : 0
    9 : 3
    10 : 4
    11 : 4
    12 : 4
```

10.4.2 $G_{n,p}$

Redes aleatória com n nós. Um dado arco `ij` está na rede com probabilidade `p`.

```
> python g_np.py 5 0.3
5
4
0 2
1 2
2 1
3 0
```

10.5 Análise experimental

10.5.1 criação da rede

rede	matriz	dicionário	listas
g_512_0.5.txt	0.12	0.13	0.12
g_1024_0.5.txt	0.48	0.50	0.50
g_2048_0.5.txt	1.87	2.01	1.97
g_4096_0.5.txt	7.63	8.13	7.58
g_8192_0.5.txt	30.06	32.91	32.29

10.5.2 Cálculo das distâncias

rede	matriz	dicionario	listas
g_512_0.5.txt	0.07	0.05	0.05
g_1024_0.5.txt	0.30	0.18	0.19
g_2048_0.5.txt	1.24	0.73	0.71
g_4096_0.5.txt	4.69	2.97	2.82
g_8192_0.5.txt	18.45	11.41	11.71

10.6 Distâncias

A função a seguir foi vista na aula passada e foi a motivação para algoritmos em redes (grafos). Trocamos a matriz/`array` `rede` da aula passada por um objeto `Rede`.

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

    Recebe o índice c de uma cidade e uma rede de estradas
    com n cidades através de um array adj.

    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 = n*[n] # pode ser array 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()

        # NOVA VERSÃO: examine as cidades vizinhas da cidade i
        for j rede.vizinhas(i):
            if d[j] > d[i]+1: # ou d[j] == n
                d[j] = d[i] + 1
                q.enqueue(j)

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

10.7 Leia redes

Esse será a função que criará a rede. É cliente da classe `Rede`.

```
#-----------------------------------------------------------
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) # __init__

        for k in range(m):
            linha  = arquivo.readline()
            cidade = linha.split()
```

```
        i = int(cidade[0])
        j = int(cidade[1])
        rede.insira(i, j) # insira()

# retorne a rede
return rede
```

10.8 Redes em matrizes

Arquivo: `redes/rede_matriz.py`

```
class Rede:
    #-------------------------------------------
    def __init__(self, n):
        '''(Rede, str) -> 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.n   = n
        # matriz de adjacência
        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(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 vizinhas(self, i):
    '''(Rede, int, int) -> None

    Recebe um rede referênciada por self e um inteiro i
    e j retorna as cidades vizinhas de i.
    '''
    viz = []
    for j in range(self.n):
        if self.adj[i][j]:
           viz.append(j)
    return viz

#-------------------------------------------
def existe(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 self.n
```

10.9 Redes em arrays

Arquivo: `rede_numpy.py`

```
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.n = n
        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"
        s += str(self.adj) + "\n"
        return s

    #-------------------------------------------
    def insira(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 vizinhas(self, i):
        '''(Rede, int) -> list

        Recebe um rede referênciada por self e um
        inteiros i e retorna as cidades vizinhas de i
```

```
        '''
        viz = []
        for j in range(self.n):
            if self.adj[i,j]:
                viz.append(j)
        return viz

    #-----------------------------------------------
    def existe(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 self.n
```

10.10 Redes em dicionários

Arquivo: `rede_dict.py`

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

        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] é a lista das cidades adjacentes a i.
        '''
        self.n = n
        adj = { i:[] for i in range(n)}
        #for i in range(n):
        #    adj[i] = []
        self.adj = adj

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

        Recebe uma Rede referênciada por self e cria
        e retorna um string que representa a rede.
        '''
        s   = "Listas de adjacências:\n"
        adj = self.adj
        n   = len(adj)
        for i in range(n):
            s += "%d:" %i + str(adj[i]) + "\n"
        return s

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

        Recebe um rede referênciada por self e um par
        de inteiros representando cidades e insere
        na rede a estrada de i a j.
        '''
        self.adj[i].append(j)

    #-------------------------------------------
    def vizinhas(self, i):
        '''(Rede, int) -> list
```

```
    Recebe um rede referênciada por self e um
    inteiro representando cidades e retorna a lista dos
    vizinhos de i.
    '''
    return self.adj[i][:]


#-----------------------------------------------
def existe(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 j in self.adj[i]

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

    Recebe uma referência self e retorna o número de
    cidades na rede.
    '''
    return self.n
```

10.11 Redes em listas

10.12 Apêndice

10.12.1 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**.

A implementação abaixo usa uma lista em que as inserções são no início `insert(0, item)`e as remoções são do final (`pop()`).

10.12.2 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 itens.pop(0) # return self.itens.pop()

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

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

10.13 Main

```
# Temos duas implementações de uma rede
# from rede_matriz_adjacencia import Rede
# from rede_listas_adjacencia import Rede

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 crie_rede(nome_arquivo):
    '''(str) -> array

    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 = 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
```