Aula 08: 28/AGO/2018

Aula passada

Problema das distâncias

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

- filas em busca em largura: usamos o nome fila, mas não falamos em *busca em largura*;
- ~~classe `Fila`~~;
- representação de grafos/redes: matriz de adjacência ou ~~lista de adjacências~~;
- ~~classe `Rede`~~.

Nessa aula não construímos classe alguma. A `Rede` foi implementada diretamente através de uma matriz de adjacência. Também não fizemos um função para construir a matriz de adjacência a partir dos pares que foram os arcos.

Hoje

Rudimentos de análise de experimental e analítica de algoritmos. O pano de fundo é o problema de contar o número de ocorrência de cada palavra em uma lista de palavras. Concretamente essa lista será dada através de um arquivo:

- implementação de três funções: `indice()`, `get()`, `put()` que implementam um dicionário.
- implementação de um `main()` que se apópia em `get()` e `put()` e conta o número de ocorrência de palavras em um arquivo que contém uma lista de palavras.

Tudo isso está no arquivo `conta_funcoes.py`.

Mudar a implementação para objetos:

- implementamos uma classe `Dicio` apenas implementando o método `__init__()` e transformando as funções `indice()`, `get()` e `put()` em métodos (arquivo `dicio.py`);
- implementar `main()` utilizando a classe `Dicio` (arquivo `conta_Dicio.py`).

Observar que o consumo de tempo torna difícil usar essa implementação para arquivos grades.

Os arquivos com listas de palavras estão no diretório `pal`.

Os arquivos com os textos originais estão no diretóruo `txt`.

Problema: `indice()`

Escreva uma função `indice()` que respeite a especificação a seguir.

```
#---------------------------------------------
def indice(chaves, chave):
    '''(list, Chave) -> int ou None

    Recebe uma lista chaves e uma chave.
    Retorna a posição da chave na lista chaves.
    Se a chave não está na lista, a função retorna None.
```

```
    Chave pode ser int, float, string ou tuple.
    '''
    # apelidos
    n = len(chaves)

    # percorra o dicionário
    for i in range(n):
        if chaves[i] == chave:
            return i
    return None
```

Problema: `get()`

Escreva uma função `get()` que respeite a especificação a seguir.

```
#-------------------------------------------
def get(chaves, valores, chave):
    '''(list, list, Chave) -> Valor

    Recebe listas chaves e valores e uma chave.
    Retorna o valor correspondente à chave.
    Se a chave não está na lista, a função retorna None.

    Chave pode ser int, float, string ou tuple.
    Valor pode ser qualquer classe, menos None ;-).
    '''
    i = indice(chaves,chave)
    if i != None: return valores[i]
    return None
```

Problema: `put()`

Escreva uma função `put()` que respeite a especificação a seguir.

```
#-------------------------------------------
def put(chaves, valores, chave, valor):
    '''(list, list, Chave, Valor) -> None

    Recebe listas chaves e valores e um par chave-valor.
    Se a chave não está na lista chaves o par chave-valor é inserido nas listas.
    na posição correspondente da lista valores.
    Se a chave está na lista, o seu valor é atualizado.
    '''
    i = indice(chaves, chave)
    if i == None:
        chaves.append(chave)
        valores.append(valor)
    else:
        valores[i] = valor
```

Problema: conta palavras

Dado um arquivo com uma lista de palavras, conta o número de ocorrências de cada palavra na lista.

```
MOSTRE  = 'show'
CHAVES  = 'chaves'
VALORES = 'valores'
ITENS   = 'itens'
SALVE   = 'salve'
LEN     = 'len'
MAX     = 'max'
PROMPT  = 'consulta >>> '

#-------------------------------------------------------------
def main():
    '''
    Programa que lê uma lista de palavras de um arquivo, cria um dicionário
    em que as chaves são cada uma das palavras no texto e em
    que os respectivos valores são o número de ocorrências de
    cada palavra no texto.


    Em seguida, o programa abre um prompt que permite consultas
    interativas a um dicionário.
    '''

    # leia o nome do arquivo
    nome = input("Digite o nome do arquivo com palavras: ")

    # abra o arquivo
    arq_entrada = open(nome, 'r', encoding='utf-8')

    print("lendo as palavras")
    # leia o texto no aqruivo
    texto = arq_entrada.read()

    # feche o arquivo
    arq_entrada.close()
    print("leitura encerrada")

    # crie lista de palavras
    print("criando dicionario de palavras")
    lista_pals = texto.split()

    palavras  = []
    ocorrs    = []
    for palavra in lista_pals:
        valor = get(palavras, ocorrs, palavra)
        if valor == None:
            put(palavras, ocorrs, palavra, 1)
```

```
    else:
        put(palavras, ocorrs, palavra, valor+1)
print("dicionário criado")

# consulte o dicionário interativamente
print("inciando consulta: tecle ENTER para encerrar")
palavra = input(PROMPT).strip()
while palavra != "":
    if palavra == MOSTRE:
        for i in range(palavras):
            print(palavras[i], ":", ocorrs[i])
    elif palavra == CHAVES:
        print(palavras)
    elif palavra == VALORES:
        print(ocorrs)
    elif palavra == LEN:
        print(len(palavras))
    else:
        valor = get(palavras, ocorrs, palavra)
        print('%s: %s' %(palavra,str(valor)))
    palavra = input(PROMPT).strip()
```

Dicionários

Um **dicionário** é um conjunto de objetos ou itens cada um dotado de uma `chave` e de um `valor`.

As chaves podem ser números inteiros ou strings ou outros tipos de dados.

Um dicionário está sujeito a dois tipos de operações:

- *inserção*: consiste em introduzir um objeto na tabela
- *busca*: consiste em encontrar um elemento que tenha uma dada chave.

Solução: classe `Dicio`

```
# importe classe Dicio que implementa um dicionário
from dicio import Dicio # astro do show

MOSTRE  = 'show'
CHAVES  = 'chaves'
VALORES = 'valores'
ITENS   = 'itens'
SALVE   = 'salve'
LEN     = 'len'
MAX     = 'max'
PROMPT  = 'consulta >>> '

#-------------------------------------------------------------
def main():
    '''
    Programa que lê um texto de um arquivo, cria um dicionário
    em que as chaves são cada uma das palavras no texto e em
    que os respectivos valores são o número de ocorrências de
    cada palavra no texto.


    Em seguida, o programa abre um prompt que permite consultas
    interativas a um dicionário.
    '''

    # leia o nome do arquivo
    nome = input("Digite o nome do arquivo com palavras: ")

    # abra o arquivo
    arq_entrada = open(nome, 'r', encoding='utf-8')

    print("lendo as palavras")
    # leia o texto no aqruivo
    texto = arq_entrada.read()

    # feche o arquivo
    arq_entrada.close()
```

```
    print("leitura encerrada")

    # crie lista de palavras
    print("criando dicionario de palavras")
    lista_pals = texto.split()

    dicio = Dicio()
    for palavra in lista_pals:
        valor = dicio.get(palavra)
        if valor == None:
            dicio.put(palavra, 1)
        else:
            dicio.put(palavra, valor+1)
    print("dicionário criado")

    # consulte o dicionário interativamente
    print("inciando consulta: tecle ENTER para encerrar")
    palavra = input(PROMPT).strip()
    while palavra != "":
        if palavra == MOSTRE:
            print(dicio)
        elif palavra == CHAVES:
            print(dicio.keys())
        elif palavra == VALORES:
            print(dicio.values())
        elif palavra == LEN:
            print(len(dicio))
        else:
            valor = dicio.get(palavra)
            print('%s: %s' %(palavra,str(valor)))
        palavra = input(PROMPT).strip()
```

Classe `Dicio`

Implementação usando uma lista `chaves` e uma lista `valores`.

```
#-------------------------------------------------------------
class Dicio:
    #---------------------------------------------
    def __init__(self):
        ''' (Dicio) -> None

        Chamado pelo construtor.

        Recebe uma referência `self` para um objeto Dicio e "monta"
        esse objeto.

        Inicialmente o dicionário está vazio
```

```
    Método especial: usado pelo Python quando fazemos Dicio().
    '''
    self.chaves  = []
    self.valores = []

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

    Método mágico/especial: usado pelo Python quando
        usamos print() ou str()
    '''
    s = ''

    # apelidos
    n = len(self)
    chaves  = self.chaves
    valores = self.valores

    # percorra o dicionário
    i = 0
    while i < n:
        s += "'%s': %d\n" %(chaves[i],valores[i])
        i += 1

    return s

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

    Recebe um dicionário e retorna o número de chaves no
    dicionario.

    Método mágico/especial: usado pelo Python quando
        usamos len().
    '''
    return len(self.chaves)

#--------------------------------------------
def get(self, chave):
    '''(Dicio, str) -> int ou None

    Recebe um dicionário referenciado por `self` e uma chave.
    Retorna o valor correspondente à chave.
    Se a chave não está no dicionário, o método retorna None.
    '''
    i = self.indice(chave)
    if i != None: return self.valores[i]
    return None
```

```
#-------------------------------------------
def indice(self, chave):
    '''(Dicio, str) -> int ou None

    Recebe um dicionário referenciado por `self` e uma chave (string).
    Retorna a posição da chave na lista chaves.
    Se a chave não está no dicionário, o método retorna None.
    '''
    # apelidos
    n = len(self)
    chaves  = self.chaves

    # percorra o dicionário
    i = 0
    while i < n:
        if chaves[i] == chave:
            return i
        i += 1
    return None

#-------------------------------------------
def put(self, chave, valor):
    '''(Dicio, str) -> None

    Recebe um dicionário referenciado por `self` e uma chave (string).
    Se a chave não está no dicionário ela é inserida com seu valor.
    Se a chave está no dicionário, o seu valor é atualizado.
    '''
    i = self.indice(chave)
    if i == None:
        self.chaves.append(chave)
        self.valores.append(valor)
    else:
        self.valores[i] += 1



#-------------------------------------------
def keys(self):
    '''(Dicio) -> list

    Recebe um dicionário referenciado por `self` e cria e retorna
    uma lista com as chaves no dicionario.

    Observação: imitação do método de mesmo nome  da classe nativa dict.
    '''
    # return self.chaves.copy() # retorna um clone
    lista = []
    n = len(self)
```

```
    i = 0
    while i < n:
        chave = self.chaves[i]
        lista.append(chave)
        i += 1
    return lista

#-------------------------------------------
def values(self):
    '''(Dicio) -> list

    Recebe um dicionário referenciado por `self` e cria e retorna
    uma lista com os valores das chaves no dicionário.

    Observação: imitação do método de mesmo nome  da classe nativa dict.
    '''
    # return self.valores.copy() # retorna um clone
    lista = []
    n = len(self)

    # percorra o dicionário
    i = 0
    while i < n:
        valor = self.valores[i]
        lista.append(valor)
        i += 1

    return lista


#-------------------------------------------
def items(self):
    '''(Dicio) -> list

    Recebe um dicionário referenciado por `self` e cria e retorna
    uma lista com pares da forma (chave, valor) dos itens no
    dicionário.

    Observação: imitação do método de mesmo nome da classe nativa dict.
    '''
    lista = []
    # apelidos
    n = len(self)
    chaves  = self.chaves
    valores = self.valores

    # percorra o dicionário
    i = 0
    while i < n:
        lista.append((chaves[i],valores[i]))
```

```
        i += 1
    return lista
```

Apêndice

Percorrer intervalos, listas, strings, e dicionários

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

i = inicio
while i < fim:
    |
    | Comandos: print(i)
    | i += passo

#------------------------------
for item in lista:
    |
    | Comandos: print(item)
    |

i = 0
n = len(lista)
while i < n:
    | item = lista[i]
    |
    | Comandos: print(item)
    |
    | i += 1

#-------------------------------------
for car in texto:
    |
    | Comandos: print(car)
    |

i = 0
n = len(texto)
while i < n:
    | car = texto[i]
    |
    | Comandos: print(car)
    |
    | i += 1

for chave in dicionario:
    print(chave)
```

Sobrecarga de operadores aritméticos

```
Operador         Método
-----------------------
+          object.__add__(self, other)
-          object.__sub__(self, other)
*          object.__mul__(self, other)
//         object.__floordiv__(self, other)
/          object.__div__(self, other)
%          object.__mod__(self, other)
**         object.__pow__(self, other[, modulo])
<<         object.__lshift__(self, other)
>>         object.__rshift__(self, other)
&          object.__and__(self, other)
^          object.__xor__(self, other)
|          object.__or__(self, other)
```

Sobrecarga de operadores relacionais

```
Operador         Método
-----------------------
<          object.__lt__(self, other)
<=         object.__le__(self, other)
==         object.__eq__(self, other)
!=         object.__ne__(self, other)
>          object.__gt__(self, other)
>=         object.__ge__(self, other)
```

Método para manipular `list`

```
Método   Parâmetros    Resultado   Descrição
-----------------------------------------
append   item          mutador     Acrescenta um novo item no final da lista
insert   posição,item  mutador     Insere um novo item na posição dada
pop      nenhum        híbrido     Remove e retorna o último item
pop      posição       híbrido     Remove e retorna o item da posição.
sort     nenhum        mutador     Ordena a lista
reverse  nenhum        mutador     Ordena a lista em ordem reversa
index    item          retorna idx Retorna a posição da primeira ocorrência do item
count    item          retorna ct  Retorna o número de ocorrências do item
remove   item          mutador     Remove a primeira ocorrência do item
```