Algoritmos e Programação de Computadores

Aula 07 - Estruturas de Dados

Prof. Dr. Rodolfo Carneiro Cavalcante
rodolfo.cavalcante@arapiraca.ufal.br
Universidade Federal de Alagoas
Campus de Arapiraca

Introdução

- Estruturas de dados
  - Conjuntos de dados relacionados entre si
- Conjuntos são fundamentais para a ciência da computação
- Algoritmos exigem vários tipos diferentes de operações sobre conjuntos
- Estudo de estruturas de dados é parte fundamental para o desenvolvimento de algoritmos
- Programas são compostos por algoritmos e estruturas de dados

Introdução

- A linguagem Python fornece várias estruturas de dados já implementadas
  - E as operações sobre estas estruturas de dados
- Já temos utilizado as listas até agora

Listas

```
#criar lista vazia
lista = [ ]
#criar lista com valores pré-definidos
lista = [1,2,3,4,5]
#criar lista com valores específicos
Lista = [0]*10
#criar lista de abrangência
lista = [ ]
for i in range(10):
        lista.append(i)
#ou de outra forma
lista = [i for i in range(10)]
```

Listas

```
#adicionar elemento a lista
lista.append(1)

#juntar duas listas
lista1 = [1,2,3,4,5]
lista2 = [6,7,8,9,10]
lista1[len(lista1):] = lista2

#ou de outra forma
lista1.extend(lista2)
```

Listas

```
#remover o primeiro elemento com valor específico
lista.remove(5)
#remove o ultimo elemento da lista
lista.pop()
#remove o elemento na posicao especificada
lista.pop(1)
#ou de outra forma
del lista[1]
#conta quantos elementos com valor específico existem na lista
lista.count(1)
#ordena a lista
sorted(lista)
```

Tuplas

- Sequência de valores
- Conjunto de dados imutável

```
#definindo uma tupla
tupla = ('José','Rua 15',152)
print(tupla)

#acessando seus valores
print(tupla[1])

#acessndo tamanho da tupla
print(len(tupla))
```

Tuplas

- Muito utilizada para:
  - definição de pares ordenados, registros extraídos de um banco de dados, etc
  - retorno de função

```
#desempacotando uma tupla
tupla = ('José','Rua 15',152)
nome,rua, numero = tupla
print(nome)
print(rua)
print(numero)
```

Sets

- Tipo de dados para definição de conjuntos
- Coleção desordenada de elementos
- Sem elementos repetidos
- Muito útil para verificação eficiente de existência de objetos e eliminação de itens duplicados
- Não suporta indexação
- Oferece suporte para operações matemáticas de conjuntos
  - união, intersecção, diferença, etc

Sets

```
numeros = [1,3,5,4,3,8,9,12,3]

#transformando em conjunto
conjunto = set(numeros)
print(conjunto)

#verificar se existe no conjunto
print(3 in conjunto)
print(2 in conjunto)
```

Sets

```
conjunto = set([1,3,5,4,3,8,9,12,3])
conjunto2 = set([-1,-5,-3,-2,1,3,5])

#uniao
print(conjunto | conjunto2)
#interseccao
print(conjunto & conjunto2)
#diferenca
print(conjunto – conjunto2)
#elementos em a ou b, menos intersecção (ou exclusivo)
print(conjunto ^ conjunto2)
```

Dicionários

- Também chamado de vetor associativo
- Conjuntos de pares chave-valor
  - Chaves são únicas em um dicionário
- Dicionários são indexados por chaves (keys)
  - Podem ser de qualquer tipo
- Delimitados por chaves
- Pares chave-valor separados por vírgula
- Tipo muito comum de dados
  - arquivo JSON

Dicionários

```
#definindo dicionarios
telefones = {'jose': 2545, 'pedro': 5214, 'ana': 5522}
print(telefones)
print(telefones['ana'])
print(sorted(telefones))
print(telefones.keys())
print('ana' in telefones.keys())
```

Dicionários

```
#definindo dicionario vazio
dicionario = {}
#adicionando chaves
dicionario[‘jose’] = 5522
dicionario[‘carlos’] = 2354
#excluindo chaves
del dicionario[‘jose’]
```

Dicionários

```
#iterando sobre dicionarios
telefones = {'jose': 2545, 'pedro': 5214, 'ana': 5522}
for key in telefones:
        print(key)
for key in telefones:
        print(key, telefones[key])
```

Exercícios

1. Escreva uma função que conta a quantidade de vogais em um texto e armazena tal quantidade em um dicionário, onde a chave é a vogal considerada.

2. Escreva um programa que lê duas notas de vários alunos e armazena tais notas em um dicionário, onde a chave é o nome do aluno. A entrada de dados deve terminar quando for lida uma string vazia como nome. Escreva uma função que retorna a média do aluno, dado seu nome.

Exercícios

3. Escreva um programa para armazenar uma agenda de telefones em um dicionário. Cada pessoa pode ter um ou mais telefones e a chave do dicionário é o nome da pessoa. Seu programa deve ter as seguintes funções:

incluirNovoNome – essa função acrescenta um novo nome na agenda, com um ou mais telefones. Ela deve receber como argumentos o nome e os telefones.

Exercícios

incluirTelefone – essa função acrescenta um telefone em um nome existente na agenda. Caso o nome não exista na agenda, você deve perguntar se a pessoa deseja incluí-lo. Caso a resposta seja afirmativa, use a função anterior para incluir o novo nome.
excluirTelefone – essa função exclui um telefone de uma pessoa que já está na agenda. Se a pessoa tiver apenas um telefone, ela deve ser excluída da agenda.

excluirNome – essa função exclui uma pessoa da agenda.

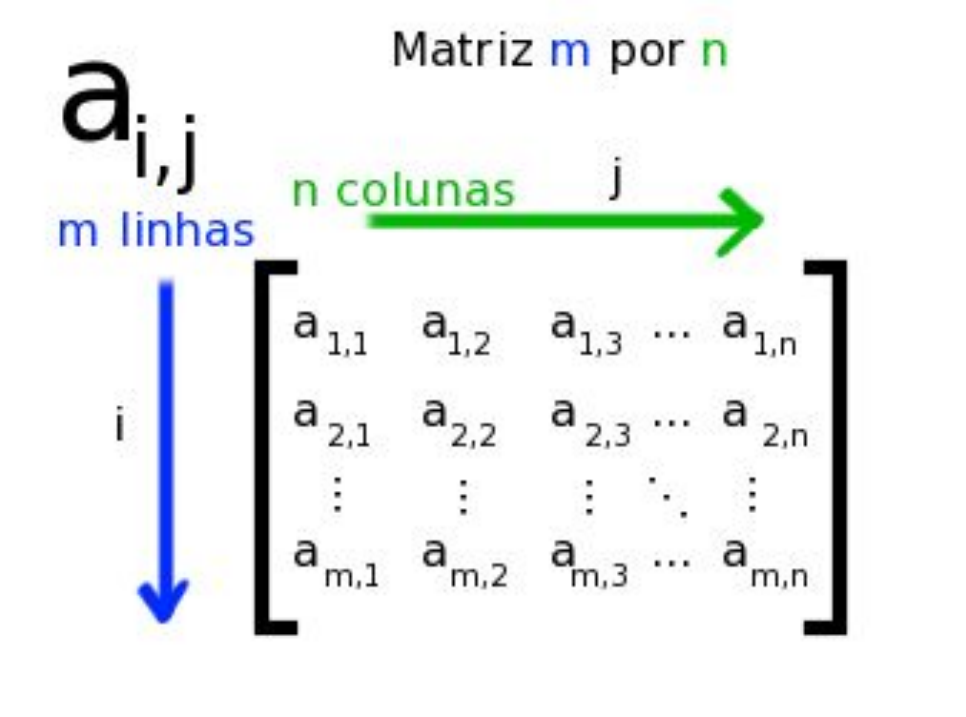
Exercícios

consultarTelefone – essa função retorna os telefones de uma pessoa na agenda.

Matrizes

- Matrizes são uma importante ferramenta matemática para modelagem de problemas
- Na programação, matrizes são importantes estruturas de dados
- Estrutura de alta dimensão
  - Lista de listas
- Ex:
  - matriz de pixels para formar uma imagem
  - Tabela de um banco de dados

Matrizes


Matriz m por n
$a_{i,j}$
m linhas
n colunas
j
i
$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{bmatrix}$

Matrizes

```
#criar matriz de zeros
m = 4 #linhas
n = 4  #colunas
A = [0]*m
for i in range(m):
        A[i] = [0]*n

#criar matriz de strings vazias
m = 4 #linhas
n = 3  #colunas
A = [' ']*m
A = [[' ']*n for i in range(m)]
```

Matrizes

```
#função para criar matriz vazia
def matriz(m,n,x):
        A = [x]*m
        for i in range(m):
                A[i] = [x]*n
        return A

A = matriz(4,4,0)
print(A)
```

Matrizes

```
#Acessando posições aleatórias de uma matriz
A[0][0] = 5
A[1][1] = 5
A[2][2] = 5
A[3][3] = 5
print(A)
```

Matrizes

```
#acessando todas as posições da matriz
for i in range(m):
        for j in range(n):
                A[i][j] = i+j

#imprimindo matriz
def imprimir(A):
        for i in range(len(A)):
                for j in range(len(A[i])):
                        print(A[i][j],end=' ')
                print()
```

Matrizes

```
#somar duas matrizes
def somar(A,B):
        m = len(A)
        n = len(A[0])
        C = matriz(m,n,0)
        for i in range(m):
                for j in range(n):
                        C[i][j] = A[i][j] + B[i][j]
        return C

A = [[2,3,1], [4,0,1]]
B = [[1,5,2], [0,1,1]]

C = somar(A,B)
```

Matrizes

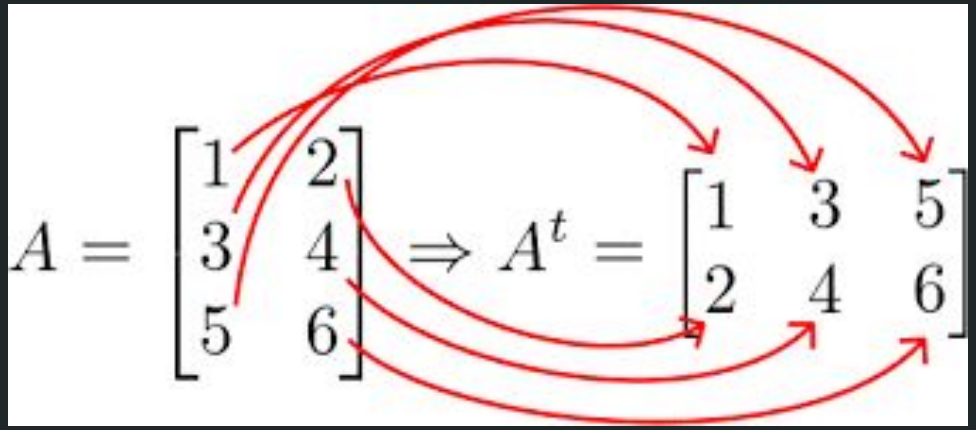
```
#somar duas matrizes
def somar(A,B):
        m = len(A)
        n = len(A[0])
        C = matriz(m,n,0)
        for i in range(m):
                for j in range(n):
                        C[i][j] = A[i][j] + B[i][j]
        return C

A = [[2,3,1], [4,0,1]]
B = [[1,5,2], [0,1,1]]

C = somar(A,B)
```

Matrizes

Matriz transposta



Matrizes

```
#matriz transposta
def transposta(A):
  m = len(A)
  n = len(A[0])
  B = matriz(n,m,0)
  for i in range(m):
    for j in range(n):
      B[j][i] = A[i][j]
  return B
```

Exercícios

1. Implemente uma função que recebe uma matriz e verifica se ela é simétrica

2. Implemente uma função que recebe uma matriz e um inteiro e faz a multiplicação da matriz pelo inteiro

3. Implemente uma função que recebe uma matriz e um inteiro e verifica se o inteiro existe na matriz

Exercícios

4. Um quadrado mágico é aquele dividido em linhas e colunas no qual a soma das linhas, das colunas e diagonais é a mesma. Ex:

8 3 4

1 5 9

6 7 2

Implemente uma função que recebe uma matriz e verifica se é um quadrado mágico

Algoritmos e Programação de Computadores

Aula 07 - Estruturas de Dados

Prof. Dr. Rodolfo Carneiro Cavalcante
rodolfo.cavalcante@arapiraca.ufal.br