

BCM0505-22 – Processamento da Informação

Vetores - Parte 1

Carla Negri Lintzmayer
carla.negri@ufabc.edu.br

<http://professor.ufabc.edu.br/~carla.negri/>

Outline

Introdução

Exemplos

Exercícios

Introdução

Motivação

Crie programas para resolver os seguintes problemas:

- Leia n números e mostre quais deles são primos.

Motivação

Crie programas para resolver os seguintes problemas:

- Leia n números e mostre quais deles são primos.
- Leia n números e mostre qual deles é o maior.

Motivação

Crie programas para resolver os seguintes problemas:

- Leia n números e mostre quais deles são primos.
- Leia n números e mostre qual deles é o maior.
- Leia n números e imprima-os na ordem inversa da leitura.

Motivação

Crie programas para resolver os seguintes problemas:

- Leia n números e mostre quais deles são primos.
- Leia n números e mostre qual deles é o maior.
- Leia n números e imprima-os na ordem inversa da leitura.
- Leia n números e calcule o desvio padrão

$$\sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}}$$

Motivação

Crie programas para resolver os seguintes problemas:

- Leia n números e mostre quais deles são primos.
- Leia n números e mostre qual deles é o maior.
- Leia n números e imprima-os na ordem inversa da leitura.
- Leia n números e calcule o desvio padrão

$$\sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}}$$

- Leia n números e um valor k e mostre qual é o k -ésimo maior número lido.

Vetores

- Uma *estrutura de dados* é uma forma de organizar os dados que desejamos processar.
- Existem diversas estruturas de dados, com diversos propósitos, cada uma delas tendo seus pontos positivos e negativos.
- Agora iremos apresentar uma estrutura de dados extremamente simples, mas também extremamente útil: *vetores* (*arrays*).

Vetores

Um *vetor* (*array*) é uma estrutura de dados *sequencial*.

- Os elementos são armazenados de forma contígua (sequencial) na memória do computador
- Podemos acessar os elementos de forma eficiente por meio de *índices*
- Na maioria das linguagens de programação a indexação começa a partir de 0 (zero)

Criando um Vetor

Para criar um vetor, basta listar o seus elementos separados por , e entre os caracteres [e]

```
1 conc = [ "A", "B", "C", "D", "F", "O" ]
```

	0	1	2	3	4	5
conc	A	B	C	D	F	O

Outros Exemplos de Criação de Vetores

Código	Resultado
<code>[3, 7, 8]</code>	???
<code>["João", "M", 8.5]</code>	???
<code>list(range(5))</code>	???
<code>[0] * 4</code>	???
<code>[]</code>	???

- Em algumas linguagens, você não pode aumentar o tamanho do vetor, você precisa criá-lo com a dimensão certa.

Outros Exemplos de Criação de Vetores

Código	Resultado
<code>[3, 7, 8]</code>	???
<code>["João", "M", 8.5]</code>	???
<code>list(range(5))</code>	???
<code>[0] * 4</code>	???
<code>[]</code>	???

- Em algumas linguagens, você não pode aumentar o tamanho do vetor, você precisa criá-lo com a dimensão certa.
 - Não é o caso de Python

Outros Exemplos de Criação de Vetores

Código	Resultado
<code>[3, 7, 8]</code>	???
<code>["João", "M", 8.5]</code>	???
<code>list(range(5))</code>	???
<code>[0] * 4</code>	???
<code>[]</code>	???

- Em algumas linguagens, você não pode aumentar o tamanho do vetor, você precisa criá-lo com a dimensão certa.
 - Não é o caso de Python
- Na maioria das linguagens compiladas, todos os elementos do vetor precisam ter o mesmo tipo

Outros Exemplos de Criação de Vetores

Código	Resultado
<code>[3, 7, 8]</code>	???
<code>["João", "M", 8.5]</code>	???
<code>list(range(5))</code>	???
<code>[0] * 4</code>	???
<code>[]</code>	???

- Em algumas linguagens, você não pode aumentar o tamanho do vetor, você precisa criá-lo com a dimensão certa.
 - Não é o caso de Python
- Na maioria das linguagens compiladas, todos os elementos do vetor precisam ter o mesmo tipo
 - Não é o caso de Python

Acessando um elemento de um vetor

Considere uma variável `x` que é do tipo vetor (`list`, em Python).

Acessamos a posição i do vetor `x` escrevendo `x[i]`.

- Vetores são indexados a partir do 0, então `x[i]` armazena o $(i + 1)$ -ésimo elemento do vetor.

	0	1	2	3	4	5
x	x_1	x_2	x_3	x_4	$\dots$	x_{n-1}

Acessando um elemento de um vetor

Considere uma variável `x` que é do tipo vetor (`list`, em Python).

Acessamos a posição i do vetor `x` escrevendo `x[i]`.

- Vetores são indexados a partir do 0, então `x[i]` armazena o $(i + 1)$ -ésimo elemento do vetor.

	0	1	2	3	4	5
x	x_1	x_2	x_3	x_4	$\dots$	x_{n-1}

Se o vetor `x` armazena n elementos, então as entradas válidas são: `x[0]`, `x[1]`, `x[2]`, ..., `x[n-1]`

- É **sua** responsabilidade usar somente índices válidos
- Acessar uma entrada inválida resulta no erro `IndexError`
 - Um erro **muito** comum é tentar acessar a entrada `x[n]`
 - Algumas linguagens, como C, não acusam o erro

Exemplo

```
1  naipe = ["ouro", "espada", "copas", "paus"]
2  i = 0
3  while i < 4:
4      print(naipe[i])
5      i += 1
```

Atualizando uma entrada do vetor

Para atualizar uma entrada do vetor, basta fazer uma atribuição àquela entrada

```
1 x = [10, 15, 20]
2 for i in range(3):
3     x[i] = x[i] + 1
4
5 for i in range(3):
6     print(x[i])
```

Atualizando uma entrada do vetor

Para atualizar uma entrada do vetor, basta fazer uma atribuição àquela entrada

```
1 x = [10, 15, 20]
2 for i in range(3):
3     x[i] = x[i] + 1
4
5 for i in range(3):
6     print(x[i])
```

- A posição **i** do vetor precisa ser uma posição válida, caso contrário, receberemos um erro. O código abaixo resulta no erro **IndexError**

```
1 x = [10, 15, 20]
2 x[6] = 30
```

Recuperando o tamanho do vetor

Podemos usar a função `len(v)` para recuperar o tamanho (número de elementos) do vetor `v`

```
1 x = [10, 15, 20]
2 n = len(x)
3 i = 0
4 while i < n:
5     print(x[i])
6     i += 1
```

Adicionando elementos ao vetor

- Em Python, podemos adicionar novos elementos ao final de um vetor com o método `append()`
- Se `v` é um vetor, escrevemos `v.append(x)` para adicionar o elemento `x` ao final do vetor

```
1  frutas = ["banana", "maca", "uva", "abacaxi"]
2  frutas.append("pera")
3  frutas.append("melancia")
4
5  i = 0
6  while i < len(frutas):
7      print(frutas[i])
8      i += 1
```

Adicionando elementos ao vetor (cont.)

`append()` é uma “função”!!

A sintaxe é diferente daquela que usamos até o momento porque essa é uma função exclusiva para o tipo *list*.

- Em *linguagens orientadas a objetos*, que é o caso de Python, podemos criar funções que estão atreladas a um tipo particular de dado.
- Uma função atrelada a um tipo de dado é chamada de *método*
- Métodos e orientação a objetos estão fora do escopo desse curso
 - Nós apenas **memorizaremos** alguns métodos, sem entrar na discussão de orientação a objetos e métodos.

Exemplos

Imprimindo uma sequência de trás para frente

Leia uma sequência de n inteiros e os imprima-os segundo a ordem inversa na qual foram fornecidos

Imprimindo uma sequência de trás para frente

Leia uma sequência de n inteiros e os imprima-os segundo a ordem inversa na qual foram fornecidos

```
1  n = int(input())
2
3  numeros = []
4  i = 0
5  while i < n:
6      val = int(input())
7      numeros.append(val)
8      i += 1
9
10 i -= 1
11 while i >= 0:
12     print(numeros[i])
13     i -= 1
```

Produto interno de dois vetores

Dado dois vetores $\vec{u}$ e $\vec{v}$ do $\mathbb{R}^n$, o *produto interno* (ou *produto escalar*) de $\vec{u}$ e $\vec{v}$ é

$$\vec{u} \cdot \vec{v} = \sum_{i=1}^n u_i v_i$$

Produto interno de dois vetores

Dado dois vetores $\vec{u}$ e $\vec{v}$ do $\mathbb{R}^n$, o *produto interno* (ou *produto escalar*) de $\vec{u}$ e $\vec{v}$ é

$$\vec{u} \cdot \vec{v} = \sum_{i=1}^n u_i v_i$$

```
1  n = int(input())
2
3  print("Lendo v1")
4  v1 = read_1Darray(n)
5
6  print("Lendo v2")
7  v2 = read_1Darray(n)
8
9  resp = inner_product(v1, v2)
10
11 print("Produto Interno (Escalar):", resp)
```

Produto interno de dois vetores (cont.)

```
1 def read_1Darray(n):
2     vec = []
3     for i in range(n):
4         val = float(input())
5         vec.append(val)
6     return vec
7
8 def inner_product(v1, v2):
9     n = len(v1)
10    total = 0.0
11    for i in range(n):
12        total += v1[i] * v2[i]
13    return total
```

Frequência

Considere uma sequência de números que representam os votos de n pessoas. Cada pessoa votou em um candidato cujo número é entre 0 e 10. Conte quantos votos cada candidato recebeu e imprima o boletim de urna.

Frequência

Considere uma sequência de números que representam os votos de n pessoas. Cada pessoa votou em um candidato cujo número é entre 0 e 10. Conte quantos votos cada candidato recebeu e imprima o boletim de urna.

```
1  n = int(input())
2
3  votos = [0] * 11
4  for i in range(n):
5      voto = int(input())
6      votos[voto] += 1
7
8  print("Resultado das eleições:")
9  for i in range(11):
10     print(f"Candidato {i} recebeu {votos[i]} votos")
```

Exercícios

Pares antes de ímpares

Implemente um programa que leia uma sequência de n números e exiba primeiro todos os números pares, seguidos por todos os números ímpares, preservando a ordem relativa dentro de cada grupo.

Entrada

(O primeiro número dá o tamanho da sequência)

```
8
8
9
6
10
1
3
12
7
```

Saída

```
8
6
10
12
9
1
3
7
```

Elemento máximo de um vetor

Implemente uma função chamada `max(v)` que recebe um vetor `v` como único argumento e devolve o maior elemento presente nesse vetor.

Desvio padrão

Faça um programa que lê n números e calcula o desvio padrão deles. Se $x_1, x_2, \dots, x_n$ são os números lidos, então o desvio padrão é

$$\sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}}$$

onde $\bar{x}$ é a média dos números lidos.

Busca em um vetor

Escreva uma função chamada `search(v, x)` que recebe um vetor `v` e um valor `x` e devolve o menor índice `i` para o qual `v[i] = x`, ou devolve `-1`, caso `x` não esteja presente em `v`.

Frequência++

Escreva um programa que recebe uma sequência de n números quaisquer que representam os votos aos candidatos de uma eleição. Você não sabe quantos eram os candidatos e nem quais eram os números dos candidatos.

Conte quantos votos cada candidato recebeu e imprima o boletim de urna.