Programação de Computadores

Instituto de Computação UFF
Departamento de Ciência da Computação

Otton Teixeira da Silveira Filho

Conteúdo

- Estrutura de dados: listas
- Manipulando listas
- Vetores como listas
- Matrizes como lista de listas
- Determinação do maior elemento de uma lista
- Ordenando uma lista:

  Ordenação por Seleção

  Ordenação Bolha

Estrutura de dados

Uma estrutura de dados é uma determinada configuração de nossa informação de modo a podemos acessar e modificar esta informação da forma mais conveniente para um determinado fim ou determinada aplicação

Estrutura de dados

Em Python existem uma boa variedade de estruturas de dados pré-definidas

Inicialmente trataremos de listas que serão apresentadas como se fossem de conteúdo homogêneo, ou seja, todos de um mesmo tipo.

Mostraremos mais possibilidades a medida que evoluirmos

Lista

- Uma lista (list) em Python é uma sequência ou coleção ordenada de valores. Cada valor na lista é identificado por um índice
- O valores que formam uma lista são chamados elementos
- O primeiro elemento da lista tem índice 0
- Podemos consultar a lista de trás para frente. Neste caso o índice do último elemento da lista é -1

Lista

- A lista pode ser de qualquer tipo, incluindo uma lista
- Podemos ter uma lista especial denominada lista vazia
- A lista é mutável, ou seja, pode ter seus valores alterados

Lista

- Uma lista é identificada pelos elementos colocados entre os caracteres [e], separados por vírgula
- Podemos atribuir a uma lista um identificador

  Exemplo:

  lista1 = [1, 2, 3, 4]

  lista2 = [‘a‘, ‘b‘, ‘c‘]

  lista3 = [“Chico“, “Toninho“, “Zeca“, “Zico”, “Tom”]

Exemplo de listas

Criando e acessando listas e elementos de listas

```
# Exemplo de lista
# Criacao e impressao

def main():

    lista1 = [1, 2, 3, 4]
    lista2 = ['a', 'b', 'c', 'd']
    lista3 = ["Chico", "Toninho", "Zeca", "Zico", "Tom"]

    print("Conteudo da lista 1", lista1)
    print("Conteudo da lista 2", lista2)
    print("Conteudo da lista 3", lista3)

    print("Elemento 0 da lista 1: ", lista1[0])
    print("Elemento 1 da lista 2: ", lista2[1])
    print("Elemento 4 da lista 3: ", lista3[4])

    print("Ultimo elemento da lista 3: ", lista3[-1])
    print("Penultimo elemento da lista 3: ", lista3[-2])

main()
```

Lista

Podemos criar uma lista dinamicamente, ou seja, criamos uma lista e acrescentarmos elementos a mesma.

Exemplo:

lista = [1, 2, 3, 4]

lista.append(4)

Desta maneira incluiremos o valor colocado entre parênteses ao final da lista

* append é denominado **método**, conceito de programação orientada à objetos o qual discutiremos superficialmente neste curso

Exemplo de listas

Criando e acessando listas e elementos de listas e adicionando novos elementos

```
# Exemplo de lista
# Criacao e impressao
# Uso do append

def main():

    lista1 = [1, 2, 3, 4]
    lista2 = ['a', 'b', 'c', 'd']
    lista3 = ["Chico", "Toninho", "Zeca", "Zico", "Tom"]

    print("Conteudo da lista 1", lista1)
    print("Conteudo da lista 2", lista2)
    print("Conteudo da lista 3", lista3)

    lista1.append(4)
    lista2.append("z")
    lista3.append("Zica")

    print("Conteudo da lista 1 apos acrescimo ", lista1)
    print("Conteudo da lista 2 apos acrescimo ", lista2)
    print("Conteudo da lista 3 apos acrescimo ", lista3)

main()
```

Lista

Existem outras maneiras de manipular elementos de uma lista como

- saber a localização de um elemento
- eliminar um determinado elemento,
- Etc.

Veremos isto mais a frente

Lista

Outra maneira de criar uma lista é especificando quantos elementos ela terá inicialmente e qual será o conteúdo inicial da mesma.

Exemplo:

lista = [‘a‘] * 10

Criará uma lista de inicialmente dez elementos, cada um deles tendo como conteúdo o caracter a

Lista

Repare em mais uma sobrecarga do operador *

Vimos isto brevemente no programa tipos.py

Lista

Podemos usar as listas para criar o que chamamos na matemática de vetores, lembrando que a lista é uma forma de armazenamento da informação e não o vetor matemático

Listas e vetores

Um exemplo

Crie um programa Python que gere um vetor como uma lista de int de até 10 elementos com suas componentes iguais ao quadrado dos índices. Ao final do processo, imprima o vetor calculado.

Listas e vetores

```
# Exemplo de lista
# Usando como vetor

def main():

    lista = []

    n = int(input("Entre com um numero de valor máximo 10 "))

    if n > 10 :
        print(" O valor dado e' maior que 10")

    else :
        for i in range(0, n + 1):
            lista.append(i * i)

        print("Os valores sao :", lista)

main()
```

Listas e vetores

Podemos escrever este programa de outra forma

Listas e vetores

```
# Exemplo de lista
# Usando como vetor

def main():

    n = int(input("Entre com um numero de valor máximo 10 "))

    if n > 10 :
        print(" O valor dado e' maior que 10")

    else :
        lista = [0] * (n + 1)

        for i in range(0, n + 1):
            lista[i] = i * i

        print("Os valores sao :", lista)

main()
```

Matrizes: listas de listas

Apresentemos o uso de listas de listas para representarmos matrizes matemáticas.

Matrizes: listas de listas

Apresentemos o uso de listas de listas para representarmos matrizes

Façamos a apresentação por um exemplo. Geremos uma matriz 3 x 3 preenchida com o valor 1:

```
matriz_de_uns = []

for i in range(3) :
    matriz_de_uns.append([1] * 3)
```

Matrizes

```
# Exemplo de lista
# Usando como vetor

def main():

    matriz_de_uns = []

    for i in range(3):
        matriz_de_uns.append([1] * 3)

    print(matriz_de_uns)

main()
```

Matrizes: listas de listas

Observe que a impressão é bem deselegante mas deixa claro que trabalhamos com uma lista de listas

Podemos imprimir a matriz acessando elemento por elemento

Matrizes: listas de listas

```
# Exemplo de lista
# Usando como vetor

def main():

    matriz_de_uns = []

    for i in range(3):
        matriz_de_uns.append([1] * 3)

    for i in range(3):
        for j in range(3):
            print(matriz_de_uns[i][j])

main()
```

Matrizes: listas de listas

Continua feio...

Veremos outras possibilidades no próximo exemplo

Matrizes: listas de listas

Outro exemplo

Crie um programa em Python que gere uma matriz **mat** de int 3 x 3 de forma que ao final teremos o equivalente à matriz matemática abaixo

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Ao final do processo imprima a matriz.

Matrizes

```
# Exemplo de lista
# Usando como vetor

def main():

    mat = []
    cont = 1

    for i in range(3):
        mat.append([0] * 3)

    for i in range(3):
        for j in range(3):
            mat[i][j] = cont
            cont = cont + 1

    for i in range(3):
        for j in range(3):
            print(mat[i][j])

main()
```

Estrutura de dados

Observe que a saída não é o que esperaríamos da apresentação de uma matriz.

Para o computador isto não tem a menor importância (de fato, computador não se importa com nada. Ele só segue o que o programador especificou)

Nós é que necessitamos de ordem para compreender o que ocorre a nossa volta.

Estrutura de dados

Outro exemplo – outra versão

Crie um programa FORTRAN que gere uma matriz **mat** de dois índices de INTEGER de até 3 em cada índice de forma que ao final teremos o equivalente à matriz matemática abaixo

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Ao final do processo imprima a matriz **no formato acima**.

Matrizes

```
# Exemplo de lista
# Usando como vetor

def main():

    mat = []
    cont = 1

    for i in range(3):
        mat.append([0] * 3)

    for i in range(3):
        for j in range(3):
            mat[i][j] = cont
            cont = cont + 1

    for i in range(3):
        print(mat[i])

main()
```

Matrizes: listas de listas

Embora isto funcione, é mais interessante usar os módulos numpy e scipy para quando queremos trabalhar com matrizes, vetores e desejamos fazer operações típicas de álgebra linear

Continuaremos desta forma neste curso

Determinar o maior elemento de um vetor

É fato que as informações estruturadas, ou seja, convenientemente organizadas, simplificam vários procedimentos.

Lembrem-se do código de exercício no qual era pedido achar o maior de três números int?

Determinar o maior elemento de três valores

Maior de três números int

```
# Determina o maior valor de tres valores int dados

def main():

    a = int(input("Entre com um valor : "))
    b = int(input("Entre com outro um valor : "))
    c = int(input("Entre com mais um valor : "))

    if a > b :
        if a > c :
            print("O maior valor e' :", a)
        else :
            print("O maior valor e' :", c)
    elif b > c :
        print("O maior valor e' :", b)
    else :
        print("O maior valor e' :", c)

main()
```

Determinar o maior elemento de um vetor

Façamos a versão vetorial para n elementos com o seguinte algoritmo:

- Dado um vetor **vet** com n valores int
- Faça que o primeiro elemento do vetor seja atribuído à variável **maior**
- Teste este valor com a segunda componente do vetor. Caso ela seja maior que a da variável **maior**, atribua este valor à variável **maior**, caso não, teste a próxima componente do vetor até a última componente.

Exemplo

Façamos um exemplo para esclarecer a questão de acharmos o maior elemento de um vetor.

Dado o vetor

$$\vec{v} = \begin{pmatrix} -3 \\ 1 \\ 4 \\ 2 \\ 4 \end{pmatrix}$$

e com o índice i começando de zero.

Exemplo

$$\vec{v} = \begin{pmatrix} -3 \\ 1 \\ 4 \\ 2 \\ 4 \end{pmatrix}$$

1) maior ← -3

2) i = 1, maior < 1, portanto, maior ← 1

3) i = 2, maior < 4, portanto, maior ← 4

4) i = 3, maior > 2, portanto, nada é feito

5) i = 4, maior = 4, portanto, nada é feito

Determinar o maior elemento de um vetor

```
# Programa que determina o maior valor de um vetor dado

def main():

    vet = [0] * 5

    vet[0] = -3
    vet[1] = 1
    vet[2] = 4
    vet[3] = 2
    vet[4] = 4

    maior = vet[0]

    for i in range(1,5):
        if vet[i] > maior : maior = vet[i]

    print("O maior elemento do vetor = ", maior)

main()
```

Custo computacional

Esta versão é bem simples e funcional.

Custo computacional

Esta versão é bem simples e funcional.

Ela também nos faz refletir sobre o Custo Computacional.

Quantas operações são feitas para obtermos o resultado que desejamos?

Custo computacional

Esta versão é bem simples e funcional.

Ela também nos faz refletir sobre o Custo Computacional.

Quantas operações são feitas para obtermos o resultado que desejamos?

Observe que aqui com um vetor com 5 elementos fizemos 4 comparações.

Se fosse **n** elementos seriam **n-1** comparações.

Determinar o menor elemento de um vetor

Fazer uma versão para determinar o menor elemento basta mudar o teste feito dentro do laço de repetição

Exercício

Aprimoremos este programa:

Criemos um outro código no qual possamos entrar com os valores do vetor via teclado, imprimir o vetor dado e ao final imprimamos o maior valor do vetor.

Exercício

```
# Programa que determina o maior valor de um vetor dado

def main():

    vet = []

    n = int(input("Entre com o numero de elementos: "))

    for i in range(n):
        vet.append(int(input("Entre com um elemento: ")))

    maior = vet[0]

    for i in range(1,n):
        if vet[i] > maior : maior = vet[i]

    print("O maior elemento do vetor = ", maior)

main()
```

Ordenação

Vamos a algo mais complexo:

Faça um programa que ordene de forma crescente um vetor de int.

Ordenação

Vamos a algo mais complexo:

Faça um programa que ordene de forma crescente um vetor de int.

Imprimir três valores em ordem deu um certo trabalho.

Ordenação

Imprimir três valores dados em ordem crescente

```
# Imprime tres numeros dados em ordem crescente

def main():

    a = int(input("Entre com um numero : "))
    b = int(input("Entre com um numero : "))
    c = int(input("Entre com um numero : "))

    if a < b :
        if b < c :
            print(a,b,c)
        else :
            if a < c :
                print(a,c,b)
            else :
                print(c,a,b)
    else :
        if b < c :
            if a < c :
                print(b,a,c)
            else :
                print(b,c,a)
        else :
            print(c,b,a)
main()
```

Mais sobre print()

Mas antes de seguir em frente, vejamos uma maneira de ler várias variáveis com um único input

Mais sobre print()

...

```
# Imprime tres numeros dados em ordem crescente

def main():

    a, b, c = input("Entre com tres valores int separados por espaço em branco: ").split(' ')

    a, b, c = int(a), int(b), int(c)

    if a < b :
        if b < c :
            print(a,b,c)
        else :
            if a < c :
                print(a,c,b)
            else :
                print(c,a,b)
    else :
        if b < c :
            if a < c :
                print(b,a,c)
            else :
                print(b,c,a)
        else :
            print(c,b,a)
main()
```

Mais sobre print()

O .split(‘ ‘) usará o espaço em branco como separador entre as entradas de forma a identificar uma variável da outra

Você pode usar qualquer caracter como separador

- Observe ainda que a linha do input() está desagradavelmente grande o que dificulta a legibilidade

Ordenação

Retornemos...

Vamos a algo mais complexo:

Faça um programa que ordene de forma crescente um vetor de int.

Fazer com três foi um tanto complicado. E **n** valores?

Ordenação

A dificuldade que tivemos está em que uma abordagem aparentemente direta nem sempre é boa

Formas mais organizadas de resolver um problema estuda mais profundamente o problema e tenta criar um ou mais algoritmos mais convenientes para resolução do mesmo

Algoritmo de Seleção

Apresentemos um algoritmo baseado no que já vimos

O Algoritmo de Seleção

Esquematizemos...

Ordenação por Seleção

Vetor inicial (5, 1, 3, 8, 4)

- Procure o menor valor do vetor (já sabemos fazer isto) e troque este valor com o primeiro valor

  (1, 5, 3, 8, 4)

- Procure o menor valor do vetor **a partir da segunda posição** e troque este valor com o valor que se encontra na segunda posição

  (1, 3, 5, 8, 4)

- Procure o menor valor **a partir da terceira posição** e troque este valor com o valor que se encontra na terceira posição

  (1, 3, 5, 4, 8)

- Procure o menor valor **a partir da quarta posição** e troque este valor com o valor que se encontra na quarta posição

  (1, 3, 4, 5, 8)

Ordenação por Seleção

- Observe que podemos aproveitar o código que foi escrito para determinar o maior elemento (ou o menor elemento)

Ordenação por Seleção

- Observe que podemos aproveitar o código que foi escrito para determinar o maior elemento (ou o menor elemento)
- Observe ainda que não interessa o menor valor mas **onde** ele se encontra

Ordenação por Seleção

...

```
# Ordenacao por selecao

def main():

    vet = [0] * 5


    vet[0] = -3
    vet[1] = 1
    vet[2] = 4
    vet[3] = 2
    vet[4] = 4

    for i in range(0,5):
        i_menor = i

        for j in range(i + 1, 5):
            if vet[i_menor] > vet[j] :
                aux          = vet[i_menor]
                vet[i_menor] = vet[j]
                vet[j]       = aux

    print("Vetor ordenado", vet)

main()
```

Custo computacional

Qual o custo computacional deste algoritmo?

Custo computacional

Qual o custo computacional deste algoritmo?

Observe que para 5 elementos procuramos fazemos 4 comparações para achar o primeiro valor, 3 comparações para achar o segundo valor, 2 para o terceiro e 1 para o quarto valor.

Custo computacional

Qual o custo computacional deste algoritmo?

Observe que para 5 elementos procuramos fazemos 4 comparações para achar o primeiro valor, 3 comparações para achar o segundo valor, 2 para o terceiro e 1 para o quarto valor.

- Número total de comparações: 4 + 3 + 2 + 1 = 10

Custo computacional

Qual o custo computacional deste algoritmo?

Observe que para 5 elementos procuramos fazemos 4 comparações para achar o primeiro valor, 3 comparações para achar o segundo valor, 2 para o terceiro e 1 para o quarto valor.

- Número total de comparações: 4 + 3 + 2 + 1 = 10

  Se fosse n valores teríamos n -1 + n -2 + n – 3 + … + 2 + 1

  Sabemos que 1 + 2 + 3 + 4 + … + n-2+n-1+n = n(n+1)/2 logo

- Se fossem n valores teríamos n(n-1)/2 comparações

Custo computacional

- É um algoritmo de fácil compreensão
- É um algoritmo bem ruim (está entre os piores!) pois se o vetor já estiver ordenado ele fará o mesmo número de comparações
- Quando o número de operações é proporcional ao número de elementos ao quadrado, que é como este caso, diremos que o **custo computacional é quadrático** ou **$O(n^2)$.**

Algoritmo da Bolha

Vamos a algo mais complexo:

Faça um programa que ordene um vetor dado usando o algoritmo da Bolha.

Algoritmo da Bolha

Melhor deixar claro o que é o Algoritmo da Bolha com um exemplo

Ordenação por Bolha

Vetor inicial (5, 1, 3, 8, 4)

- Compare os dois termos contíguos. Se estiverem fora de ordem, troque-os

  (1, 5, 3, 8, 4)
- Compare o segundo com o terceiro. Se estiverem fora de ordem, troque-os

  (1, 3, 5,8 ,4)
- Compare o terceiro com o quarto. Se estiverem fora de ordem, troque-os

  (1, 3, 5, 8, 4)
- Compare co terceiro com o quinto. Se estiverem fora de ordem, troque-os

  (1, 3, 5, 4, 8)
- Se tiver acontecido alguma troca, repita o processo

Ordenação por Bolha

Vetor parcialmente ordenado (1, 3, 5, 4, 8)

- Compare os dois termos contíguos. Se estiverem fora de ordem, troque-os

  (1, 3, 5, 4, 8)
- Compare o segundo com o terceiro. Se estiverem fora de ordem, troque-os

  (1, 3, 5, 4 ,8)
- Compare o terceiro com o quarto. Se estiverem fora de ordem, troque-os

  (1, 3, 4, 5, 8)
- Compare co terceiro com o quinto. Se estiverem fora de ordem, troque-os

  (1, 3, 4, 5, 8)
- Se tiver acontecido alguma troca, repita o processo

Ordenação por Bolha

Vetor parcialmente ordenado (1, 3, 4, 5, 8)

- Compare os dois termos contíguos. Se estiverem fora de ordem, troque-os

  (1, 3, 4, 5, 8)
- Compare o segundo com o terceiro. Se estiverem fora de ordem, troque-os

  (1, 3, 4, 5 ,8)
- Compare o terceiro com o quarto. Se estiverem fora de ordem, troque-os

  (1, 3, 4, 5, 8)
- Compare co terceiro com o quinto. Se estiverem fora de ordem, troque-os

  (1, 3, 4, 5, 8)
- Não havendo trocas, pare.

Custo computacional

- Este algoritmo é de compreensão um pouco mais difícil que o de Seleção
- Não é difícil mostrar que no pior caso (quando o vetor está em ordem decrescente) o custo é n(n-1)/2
- É fácil de ver que se o vetor estiver ordenado, o número de comparações é n - 1

Custo computacional

- Este algoritmo é de compreensão um pouco mais difícil que o de Seleção
- Não é difícil mostrar que no pior caso (quando o vetor está em ordem decrescente) o custo é n(n-1)/2
- É fácil de ver que se o vetor estiver ordenado, o número de comparações é n - 1
- Este é um algoritmo sensível à ordenação do vetor
- Mas também não é um algoritmo eficiente

Ordenação por Bolha

```
# Ordenacao Bolha

def main():

    vet = [0] * 5

    vet[0] = -3
    vet[1] = 1
    vet[2] = 4
    vet[3] = 2
    vet[4] = 4

    trocou = True

    while trocou :
        trocou = False
        for i in range(4):
            if vet[i + 1] < vet[i] :
                aux = vet[i]
                vet[i] = vet[i + 1]
                vet[i + 1] = aux
                trocou = True

    print("Vetor ordenado", vet)

main()
```

Ordenação por Bolha

Repare o uso do while com a condição de “detectar“ quando houve troca e o uso do algoritmo de troca de duas variáveis que já usamos antes