Alfredo's MAC0110 Journal

Alfredo Goldman

April 26, 2020

0.1 Aula 12 - *<2020-04-27 seg>*

0.1.1 Conhecendo melhor os vetores

1. Passar vetores como parâmetro é diferente Assim como já passamos variáveis normais, ou escalares, como parâmetro de funções, também podemos passar vetores. Mas, é importante ressaltar que isso ocorre de forma distinta, isso é, os vetores são passados por referência.

   Vamos a uma analogia, quando se passa uma variável escalar, como parâmetro a função recebe uma cópia dela. Já para um vetor, o que se recebe é uma cópia do endereço dele. Em algumas linguages como C, isso é completamente explícito, e essa referência é denominada ponteiro.

```
function mudavalores(x, v)
   println("x = ", x, " e v = ", v)
   x = 1
   v[1] = 1
   println("x = ", x, " e v = ", v)
end
function vesemuda()
   x = 0
   v = [2, 3, 4]
   println("x = ", x, " e v = ", v)
   println("Antes da mudavalores")
   mudavalores(x, v)
   println("Depois da mudavalores")
   println("x = ", x, " e v = ", v)
end
```

   Ao executar a função vesemuda(), podemos ver que como esperado para a variável escalar, no escopo da função a cópia foi alterada, sem mudança na variável x original.

   Por outro lado, para o vetor, como tínhamos uma cópia do endereço, mudar sua primeira posição, fez com que o vetor original fosse alterado. Mas, alterar a cópia do endereço, não muda o endereço final, como pode se ver abaixo:

```
function novovetor(v)
   println("O vetor era = ", v)
   v = ["a", "b", "c"]
   println("O vetor ficou sendo = ", v)
end
function vesemuda2()
   v = [2, 3, 4]
   println("v = ", v)
   println("Antes de novovetor")
   novovetor(v)
   println("Depois de novovetor")
   println("v = ", v)
end
```

2. Vamos continuar vendo como manipular vetores

   Para isso, vamos precisar conhecer algo de Julia, que vai nos ajudar. Por enquanto já sabemos como acessar posições específicas de um vetor.

   Mas, vamos precisar de alguns comandos adicionais para fazer os próximos exercícios. São eles, criar um vetor vazio, adicionar um ou mais elementos ao final do vetor, e criar um vetor de um tamanho definido.

```
v = [] # define um vetor vazio
push!(v, 1)  # adiciona um primeiro elemento 1 ao vetor
push!(v, 2, 3) # adiciona os elementos 2 e 3 ao vetor
zeros(Int, 3) # cria um vetor para guardar inteiros com 3 posicoes
zeros(Float, 10) # cria um vetor para guardar floats com 10 posicoes
```

   Nós já vimos que usualmente vetores em Julia podem guardar qualquer tipo de variável, mas já é bom saber que ao usarmos um vetor com tipo pré-identificado, isso é, por exemplo, só de inteiros. Seu uso fica mais eficiente.

Vamos agora exercitar um pouco o uso de vetores:

Faça uma função inverte que dado um vetor, devolve esse vetor com os valores invertidos (isso é, quem estava na primeira posição vai para a última e assim por diante).

```
# Aqui vai a function
```

Faça uma função que recebe um vetor de inteiros e devolve um vetor apenas com os números ímpares do vetor original

```
# Aqui vai a function
```

Dado um vetor de números inteiros, faça uma função que devolve um vetor que corresponde a uma leitura desse vetor, conforme o número de elementos. Ou seja, dado o vetor [1, 1, 1, 4, 10, 10, 1] o vetor de saída dever ser [3, 1, 1, 4, 2, 10, 1, 4] ou seja, três "1", um "4", dois "10" e um "quatro".

```
# Aqui vai a function
```

Sabendo que o comando rand(1:n), vai devolver um número entre 1 e n. Escreva uma função que dado um inteiro n, devolve um vetor com uma permutação de 1 a n.

```
# Aqui vai a function
```

Voltando ao exercício da aula passada vamos ver como fazer com que a verificação se um vetor corresponde a uma permutação de forma mais eficiente. Como você faria isso