

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Computational Thinking PROF. EDUARDO GONDO



Problema: dada uma lista vet de números reais e um número real x qualquer, devolva uma posição de x em vet se $x \in vet$ ou -1 se $x \notin vet$.

0	1	2	3	4	5	6	7	8	9



Problema: dada uma lista vet de números reais e um número real x qualquer, devolva uma posição de x em vet se $x \in vet$ ou -1 se $x \notin vet$.

0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
6									

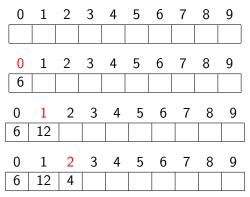


Problema: dada uma lista vet de números reais e um número real x qualquer, devolva uma posição de x em vet se $x \in vet$ ou -1 se $x \notin vet$.

(0	1	2	3	4	5	6	7	8	9
(0	1	2	3	4	5	6	7	8	9
(6									
0)	1	2	3	4	5	6	7	8	9
6	5	12								



Problema: dada uma lista vet de números reais e um número real x qualquer, devolva uma posição de x em vet se $x \in vet$ ou -1 se $x \notin vet$.





0	1	2	3	4	5	6	7	8	9
6	12	4	-7						



Continuando a busca do 14:

Neste ponto a repetição pára e retorna a posição 4 do vetor que é a posição que contém a primeira ocorrência do 14 no vetor.

Vejamos duas implementações do algoritmo de busca usando while e depois usando o for.



Busca simples - algoritmo

Veja abaixo um função que faz a busca dos dados, note que ele recebe a lista de números e o número x a ser procurado. A função retorna a posição da primeira ocorrência de x em vet.

```
1  def busca(vet, x):
2    #(list, double) -> int
3    i = 0
4    while i < len(vet) and vet[i] != x:
5         i = i + 1
6    if i == len(vet):
7         return -1
8    else:
9         return i</pre>
```



Busca simples - Versão com for

```
1  def busca(vet, x):
2    for i in range(len(vet)):
3        if vet[i] == x:
4            return i
5    return -1
```

Ambas as soluções são equivalentes em termos de performance, pois no for quando encontramos o elemento x a função termina.



Busca - considerações

- note que estamos interessados na posição de x em vet
- o valor x pode ocorrer mais de uma vez no vetor vet, mas nossa função retorna apenas uma posição
- o pior caso, ou seja, a situação que o algoritmo vai levar mais tempo de processamento é quando o valor x não pertence à vet
- nesse caso procuramos em todo vetor a procura da informação e não encontramos
- o tempo de execução de pior caso é proporcional ao tamanho do vetor
- cada comparação de um elemento do vetor com x elimina apenas um elemento
- será que pode ser melhor, ou seja, cada comparação eliminar vários elementos do vetor?



- Busca busca binária
 - ▶ a resposta é sim! MAS...



Busca - busca binária

- ▶ a resposta é sim! MAS...
- apenas se os dados do vetor estiverem ordenados, daí podemos consultar como uma lista telefônica
- a idéia principal do algoritmo é com uma comparação eliminar grande parte dos dados do vetor

chamando o elemento procurado de x e o vetor de vet, podemos descrever o algoritmo de busca binária do seguinte modo:

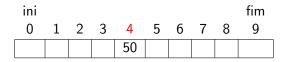


Busca binária - simulação

ini									fim
0	1	2	3	4	5	6	7	8	9
				50					



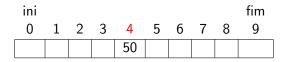
Busca binária - simulação



ini			fim						
0	1	2	3	4	5	6	7	8	9
	4			/	/	/	/	/	/



Busca binária - simulação



ini			fim						
0	1	2	3	4	5	6	7	8	9
	4			/	/	/	/	/	/

		ini	fim						
0	1	2	3	4	5	6	7	8	9
/	/	14		/	/	/	/	/	/



Busca - busca binária

Segue abaixo a implementação do busca binária em Python

```
1
   def buscaBinaria(vet, x):
       inicio = 0;
       fim = len(vet) - 1:
5
       while inicio <= fim:
6
          meio = (inicio + fim) // 2
          if vet[meio] > x:
8
               fim = meio - 1
          elif vet[meio] < x:</pre>
10
                inicio = meio + 1
11
          else:
12
               return meio
13
       return -1
```



Busca binária - observações

vejamos algumas considerações sobre a implementação

- crie apontadores para marcar as extremidades dos vetor que vocês estão considerando
- como sugestão indico os inteiros inicio e fim
- no começo do algoritmo inicio = 0 e fim = len(vet) 1
- para obter o elemento central faça a média entre inicio e fim
- se inicio <= fim então ainda tenho elementos para examinar</p>
- será que inicio > fim em algum momento do meu algoritmo?



Instruções prontas em Python

Importante!

Em Python, temos a instrução in e o método index que são implementações do algoritmo de busca sobre listas, tuplas e String. Porém, de qualquer maneira, é muito importante conhecer o algoritmo de busca pois existem linguagens que não possuem os recursos que Python oferece.



Referência Bibliográfica

- Puga e Rissetti Lógica de Programação e Estrutura de Dados
- Ascêncio e Campos Fundamentos da Programação de Computadores
- Forbelone e Eberspacher Lógica de programação: a construção de algoritmos e estruturas de dados
- ▶ Documentação do Python https://docs.python.org/3.8/
- Python Programming For Beginners: Learn The Basics Of Python Programming (Python Crash Course, Programming for Dummies) (English Edition). Kindle
- Python: 3 Manuscripts in 1 book: Python Programming For Beginners - Python Programming For Intermediates - Python Programming for Advanced (English Edition). Kindle



Copyleft

Copyleft © 2022 Prof. Eduardo Gondo Todos direitos liberados. Reprodução ou divulgação total ou parcial deste documento é liberada.