



Algoritmos e Programação de Computadores

Busca Sequencial e Binária

Agenda

— — —

- O Problema da Busca
- Busca Sequencial
- Busca Binária
- Exercícios

Busca

A large, empty search input field with a small keyboard icon on the right side.

Pesquisa Google

Estou com sorte

Disponibilizado pelo Google em: [English](#)

Busca

Temos uma coleção de elementos, onde cada elemento possui um identificador/chave único, e recebemos uma chave para busca. Devemos **encontrar o elemento da coleção que possui a mesma chave** ou identificar que não existe nenhum elemento com a chave dada.

Busca

- O problema da busca é um dos mais básicos em Computação e também possui diversas aplicações.
 - Suponha que temos um cadastro com registros de alunos.
 - Uma lista de registros é usada para armazenar as informações dos alunos. Podemos usar como chave o número do RA ou o CPF.
- Veremos algoritmos simples para realizar a busca assumindo que dados estão em uma lista.

Busca

- Nos nossos exemplos vamos criar a função:
 - **busca(lista, chave)**, que recebe uma lista e uma chave para busca.
 - A função deve retornar o índice da lista que contém a chave ou -1 caso a chave não esteja na lista.

Busca

- Python já contém um método em listas que faz a busca `index()`:

```
lista = [20, 5, 15, 24, 67, 45, 1, 76]  
lista.index(24)
```

3

mas esse método não funciona da forma que queremos para chaves que **não** estão na lista.

```
lista.index(100)
```

```
Traceback (most recent call last):  
  File "<std in>", line 1, in <module>  
ValueError : 100 is not in list
```

Busca Sequencial

Busca Sequencial

- A busca sequencial é o algoritmo mais simples de busca:
 - Percorra toda a lista comparando a chave com o valor de cada posição.
 - Se for igual para alguma posição, então devolva esta posição.
 - Se a lista toda foi percorrida então devolva -1.

Busca Sequencial

```
def buscaSequencial(lista, chave):  
    for i in range(len(lista)):  
        if lista[i] == chave:  
            return i  
    return -1
```

```
lista = [20, 5, 15, 24, 67, 45, 1, 76]  
buscaSequencial(lista, 24)  
buscaSequencial(lista, 100)
```

3
-1

Busca Binária

Busca Binária

- A busca binária é um algoritmo um pouco mais sofisticado.
- É mais eficiente, mas requer que a **lista esteja ordenada** pelos valores da chave de busca.

Busca Binária

- A ideia do algoritmo é a seguinte (assuma que a lista está ordenada):
 - Verifique se a chave de busca é igual ao valor da posição do meio da lista.
 - Caso seja igual, devolva esta posição.
 - Caso o valor desta posição seja maior, então repita o processo mas considere que a lista tem metade do tamanho, indo até posição anterior a do meio.
 - Caso o valor desta posição seja menor, então repita o processo mas considere que a lista tem metade do tamanho e inicia na posição seguinte a do meio.

Busca Binária

Pseudo código

```
#vetor começa em inicio e termina em fim
inicio = 0
fim = tam-1
repita enquanto tamanho da lista considerado for >= 1
    meio = (inicio + fim)/2
    se lista[meio] == chave então
        devolva meio
    se lista[meio] > chave então
        fim = meio - 1
    se lista[meio] < chave então
        inicio = meio + 1
```

Busca Binária

chave = 15

lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9

inicio =

fim =

meio =

Busca Binária

chave = 15

lista

1	5	15	20	24	45	67	76	78	100
0	1	2	3	4	5	6	7	8	9



inicio = 0

fim = 9

meio = 4

Busca Binária

chave = 15

lista

1	5	15	20	24	45	67	76	78	100
0	1	2	3	4	5	6	7	8	9



se `lista[meio] > chave` então
`fim = meio - 1`

inicio = 0

fim = 9

meio = 4

Busca Binária

chave = 15

lista

1	5	15	20	24	45	67	76	78	100
0	1	2	3	4	5	6	7	8	9



se `lista[meio] > chave` então
`fim = meio - 1`

inicio = 0

fim = 3

meio = 4

Como o valor da posição do meio é maior que a chave, atualizamos o **fim** da lista considerada.

Busca Binária

chave = 15

lista

1	5	15	20	24	45	67	76	78	100
0	1	2	3	4	5	6	7	8	9



inicio = 0

fim = 3

meio = 1

Busca Binária

chave = 15

lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9



se `lista[meio] < chave` então
 `inicio = meio + 1`

`inicio = 0`

`fim = 3`

`meio = 1`

Busca Binária

chave = 15

lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9



se `lista[meio] < chave` então
 `inicio = meio + 1`

`inicio = 0`

`fim = 3`

`meio = 1`

Como o valor da posição do meio é menor que a chave, atualizamos **inicio** da lista considerada.

Busca Binária

chave = 15

lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9



se `lista[meio] < chave` então
 `inicio = meio + 1`

inicio = 2

fim = 3

meio = 1

Como o valor da posição do meio é menor que a chave, atualizamos **inicio** da lista considerada.

Busca Binária

chave = 15

lista

1	5	15	20	24	45	67	76	78	100
0	1	2	3	4	5	6	7	8	9



inicio = 2

fim = 3

meio = 2

Busca Binária

chave = 15

lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9



se `lista[meio] == chave` então
devolva `meio`

inicio = 2

fim = 3

meio = 2

Finalmente encontramos a chave e podemos
devolver sua posição 2.

Busca Binária

```
def buscaBinaria(lista, chave):  
    inicio = 0  
    fim = len(lista)-1  
    while inicio <= fim:  
        meio = (inicio + fim)//2  
        if lista[meio] == chave:  
            return meio  
        elif lista[meio] > chave:  
            fim = meio - 1  
        else:  
            inicio = meio + 1  
    return -1
```

Busca Binária

```
lista = [20, 5, 15, 24, 67, 45, 1, 76]  
insertionSort(lista)  
lista
```

```
[1, 5, 15, 20, 24, 45, 67, 76]
```

```
buscaBinaria(lista, 24)  
buscaBinaria(lista, 25)
```

```
4
```

```
-1
```

Exercício

Refaça as funções de busca sequencial e busca binária assumindo que a lista possui chaves que podem aparecer repetidas.

Neste caso, você deve retornar uma lista com todas as posições onde a chave foi encontrada.

Se a chave só aparece uma vez, a lista conterá apenas um índice. E se a chave não aparece, as funções devem retornar a lista vazia.

“O Futuro do seu Emprego”

<https://youtu.be/qVGxWi6XDAI> (Canal Nerdologia)



Referências

- Os slides dessa aula foram baseados no material de MC102 do Prof. Eduardo Xavier (IC/Unicamp).