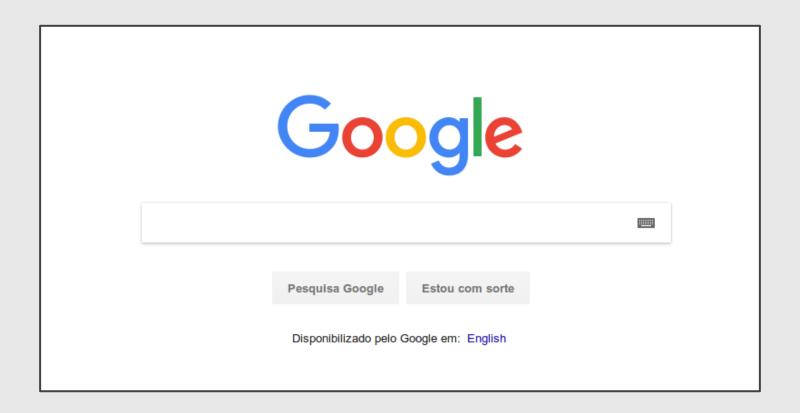
Algoritmos e Programação de Computadores

Busca Sequencial e Binária

Agenda

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



Definição

Trata-se de um *processo*, onde procura-se selecionar, utilizando um critério específico, alguma *informação particular*, a partir de uma *coleção de dados*.

Desta forma, no contexto desta matéria (secção/capítulo deste curso MC102), o termo *busca* será restrito ao processo de encontrar um *ítem específico* em uma coleção de *ítens de dados*. Em particular, dentro uma estrutura de dados de tipo *sequencial*.

Ref.: Data Structures and Algorithms using Python; R. Necaise; J. Wiley & Sons Pub., 2011

Exemplo:

Considere-se uma coleção de elementos (estrutura sequencial de dados), onde cada elemento possui um identificador/chave único, e recebe-se uma chave para busca.

Deve-se encontrar o elemento da coleção que possua a mesma chave ou identificar (como conclusão do processo de busca), que não existe nenhum elemento com a chave dada.

- O problema da **busca**, junto com a **ordenação**, são os mais comuns em Computação, envolvendo diversas aplicações.
- Ex.
 - Suponha que temos um cadastro com registros de motoristas.
 - Uma lista de registros é usado para armazenar as informações dos motoristas.
 Podemos usar como chave o número da carteira de motorista, ou RG, ou o CPF.
- Serão tratados algoritmos básicos para realizar a busca assumindo que os dados estão organizados em uma estrutura sequencial (ex. uma lista).

- Implementação de um processo de busca utilizando Python
- Nos exemplos que serão tratados a seguir será utilizada uma função:
 - o busca(lista,chave),
 - que recebe uma lista e uma chave para busca como argumentos (ou parâmetros).
 - Esta função busca deve retornar:
 - o índice da lista que contém a chave, ou
 - -1, caso a chave não esteja na lista.

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

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

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 (sequential or linear search) é o algoritmo mais simples de busca:
 - Percorre-se uma lista (ou linear array, ou vetor) 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
```

- 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.

- 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.

Pseudo código

```
#vetor começa em inicio e termina em fim
inicio = 0
fim = t.am-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
```

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 =

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

$$inicio = 0$$

$$fim = 9$$

$$meio = 4$$

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.

lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9
					1			io] > (então

$$inicio = 0$$

$$fim = 9$$

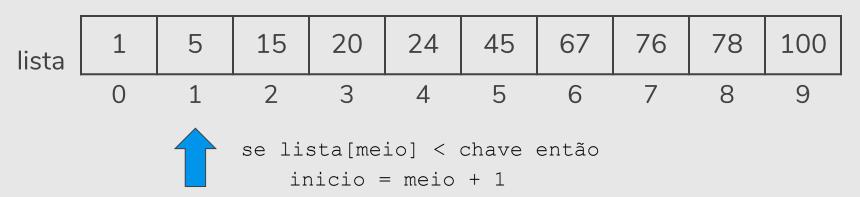
$$meio = 4$$

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

$$inicio = 0$$

$$fim = 3$$

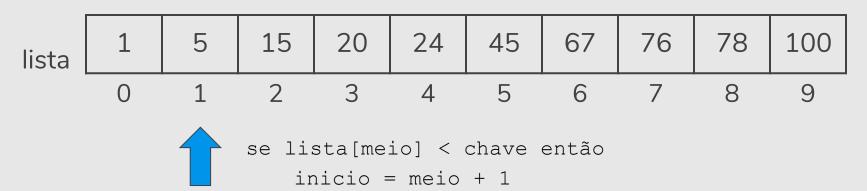
$$meio = 1$$



$$fim = 3$$

$$meio = 1$$

chave = 15



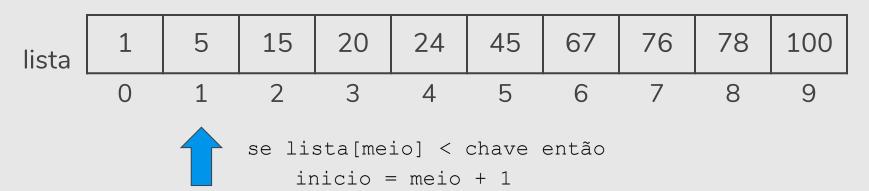
$$inicio = 0$$

$$fim = 3$$

$$meio = 1$$

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

chave = 15



$$inicio = 2$$

$$fim = 3$$

 $meio = 1$

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

chave = 15

lista	1	5	15	20	24	45	67	76	78	100
		1								



inicio = 2

fim = 3

meio = 2

chave = 15

lista	1	5	15	20	24	45	67	76	78	100
'	0	1	2	3	4	5	6	7	8	9
			1		sta[me evolva	io] == meio	chave	então)	

$$inicio = 2$$

$$fim = 3$$

$$meio = 2$$

Finalmente encontramos a chave e podemos devolver sua posição 2.

```
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
```

```
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 contera apenas um índice. E se a chave não aparece, as funções devem retornar a lista vazia.

Exercícios

Busca Sequencial e Binária Soluções para múltiplos itens

Busca Sequencial múltiplos itens

```
def buscaSequencial(lista, chave):
    lista_indices = []
    for i in range(len(lista)):
        if lista[i] == chave:
            lista_indices.append(i)
        return lista_indices
```

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

[1, 8]
[]
```

- Solução proposta de busca binária para múltiplos itens.
 - Da mesma forma que na solução original, requer que a lista esteja ordenada pelos valores da chave de busca.
 - O procedimento é iniciado a través da busca binária convencional, porém, incorporando um procedimento de busca linear, quando ocorre a primeira coincidência (matching) com a chave de busca.
 - Caso um primeiro elemento **m** de lista seja encontrado (na busca binária), é ativado um 20 procedimento de busca, iniciado na vizinhança do elemento **m**, utilizando um algoritmo de busca linear/sequencial.
 - Inicialmente são procurados os elementos adjacentes, na vizinhança iniciada com o índice m-1.
 - Na sequência o procedimento prossegue, com a busca iniciada nos elementos adjacentes, a partir do elemento m+1.
 - Para caso caso de matching dos procedimentos anteriores, o valor índice é adicionado a uma lista.
 - A busca linear para cada direção finaliza quando não há mais coincidências (do tipo elemento=chave)

```
# 1. Ordenação por inserção
def insertionSort(vetor in):
    for i in range(1,len(vetor in)):
        aux = vetor in[i]
        i= i - 1
        while inicio >= 0 and vetor in[j] > aux:
            vetor in[j+1] = vetor[j]
            i = i - 1
        vetor in[j+1] = vetor[j]
# 2. Busca linear/sequencial, para elementos iquais na vizinhança ao valor do meio
# Parametros para Busca linear/sequencial: lista + elemento do meio, m
def look at neighbourhood(lista, m):
    global lista indices
    aux = True
    i left= m - 1
    # iniciar busca na direção à esquerda e à direita do indice m
```

```
# 2a. Iniciar Busca à Esquerda de m
    while i left >= 0 and aux:
        if lista[i left] == lista[m]:
            lista indices.append(i left)
        else:
            aux = False
        i left = i left - 1
# 2b. Iniciar Busca à Direita de m
    i right = m+1
    aux = True
    while i right <= len(lista) and aux:</pre>
        if lista[i right] == lista[m]:
            lista indices.append(i right)
        else inicio >= 0 and vetor in[j] > aux:
            aux = False
        i right = i right + 1
# fim de função de busca linear na vizinhança (look at neighbourhood)
```

```
# 3. Busca Binaria, incluindo busca linear/sequencial, caso matching
def buscaBinaria(lista, chave):
    global lista indices
    inicio = 0
    fim = len(lista) - 1
    lista indices = []
    while inicio <= fim:
        meio = (inicio + fim)//2
        if lista[meio] == chave:
            lista indices.append(meio)
            look at neighbourhood(lista, meio)
            return lista indices
        elif lista[meio] > chave:
            fim = meio - 1
        else:
            inicio = meio + 1
    return lista indices
```

```
# Programa principal
lista = [20,5,15,24,67,45,1,76,5,101,24,0,-1,76,5,0,5]
insertionSort = (lista)
# ingressa chave para iniciar buscas
key = int(input("Ingresse chave de busca: "))
lista_indx = buscaBinaria(lista, key)
# imprime lista ordenada e lista de indices (com todas as posições das ocorrencias; ou vazia, caso nao haja ocorrencia)
print(lista, lista_indx)
```

```
lista = [-1,0,0,1,5,5,5,5,15,20,24,24,45,67,76,76,101]
buscaSequencial(lista, 5)
buscaSequencial(lista, 33)

[5,4,6,7]
[]
```

Referências

Os slides dessa aula foram baseados no material de MC102 dos Profs.
 Marcelo Jara, Sandra Avila e Eduardo Xavier (IC/Unicamp).

Livros:

- Data Structures and Algorithms using Python; R. Necaise; J. Wiley & Sons Pub., 2011
- Introduction to Computing, Explorations in Language, Logic and Machines using Python; D. Evans (Univ. Virginia); Creative Commons, 2010

Exercícios: Matrizes

Exercícios

 Escreva um programa que leia todas as posições de uma matriz 10x10. O programa deve então exibir o número de posições não nulas na matriz.

```
# Adiciona cada número na matriz 10x10
matriz = []
for i in range (10):
         lista = []
         for j in range (10):
                   numero = int(input())
                   lista.append(numero)
         matriz.append(lista)
# Conta quantos número são não nulos
nao nulo = 0
for i in matriz:
         for j in i:
                   if j != 0:
                   nao nulo += 1
print("O número de posições não nulas é", nao nulo)
```

```
# Adiciona cada número na matriz 10x10
matriz = []
nao nulo = 0
for i in range(10):
         lista = []
         for j in range(10):
                   numero = int(input())
                   lista.append(numero)
                   if numero != 0: # Conta quantos número são não nulos
                            nao nulo += 1
         matriz.append(lista)
print("O número de posições não nulas é", nao nulo)
```