



Estruturas de Dados Lineares

Pesquisa Sequencial e Binária

Prof. Vinicius Bischoff

Pesquisa de Dados em Tabelas

- Métodos para localizar entradas em tabelas, dado o valor de uma chave primária como argumento da pesquisa.

	Chave Primária	Info
1	7	...
2	9	...
3	14	...
4	35	...
5	78	...

Localizar: informações relativas às chaves: 9, 35


Técnica para Pesquisa Sequencial

- Fazer uma varredura serial de um array, comparando o argumento de pesquisa com o elemento de cada posição, até ser encontrado um que seja igual (sucesso) ou até que seja atingido o final do array (não foi encontrado).

Pesquisa Sequencial em Array Não Ordenado

```
def linear_search(a: np.array, key: int) -> int:
    for i, elm in enumerate(a):
        if elm == key:
            return i
    return -1
```

Exemplo: procurando o elemento 4



0	1	2	3	4	5	6	7	8	9
5	20	10	9	2	15	7	4	20	6

Pesquisa Sequencial em Array Ordenado

```
def sorted_linear_search(a: np.array, key: int) -> int:  
    i: int = 0  
    while i < len(a) and a[i] <= key:  
        if a[i] == key:  
            return i  
        i += 1  
    return -1
```

Exemplo: procurando o elemento **11**

0	1	2	3	4	5	6	7	8	9
2	4	5	7	9	10	12	13	15	16

Pesquisa Binária

- Método para ser aplicado em **arrays ordenados**;
- Reduz pelo metade o número de elementos a serem considerados;
- Exemplo: pesquisa do elemento com valor 17.

0	1	2	3	4	5	6	7	8	9	10
1	3	5	7	9	11	13	15	17	19	21

1	2
---	---

Pesquisa Binária

- Algoritmo:
 - Consiste na comparação do argumento de pesquisa com o elemento localizado no endereço médio da tabela;
 - Se o *argumento* for igual, a busca encerra com sucesso;
 - Se o argumento for menor do que o elemento contido naquele endereço, o processo é repetido para a metade inferior da tabela;
 - Se o argumento for maior do que o elemento contido naquele endereço, o processo é repetido para a metade superior da tabela.
- A área de pesquisa é reduzida à metade do número de elementos a cada vez.

Uma Pesquisa Binária

- Argumento de pesquisa: **32**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	15	17	30	32	34	40	50	80	90	95	97	99	101	105
			↑	↑	↑		↑							
			2 ^a	4 ^a	3 ^a		1 ^a							

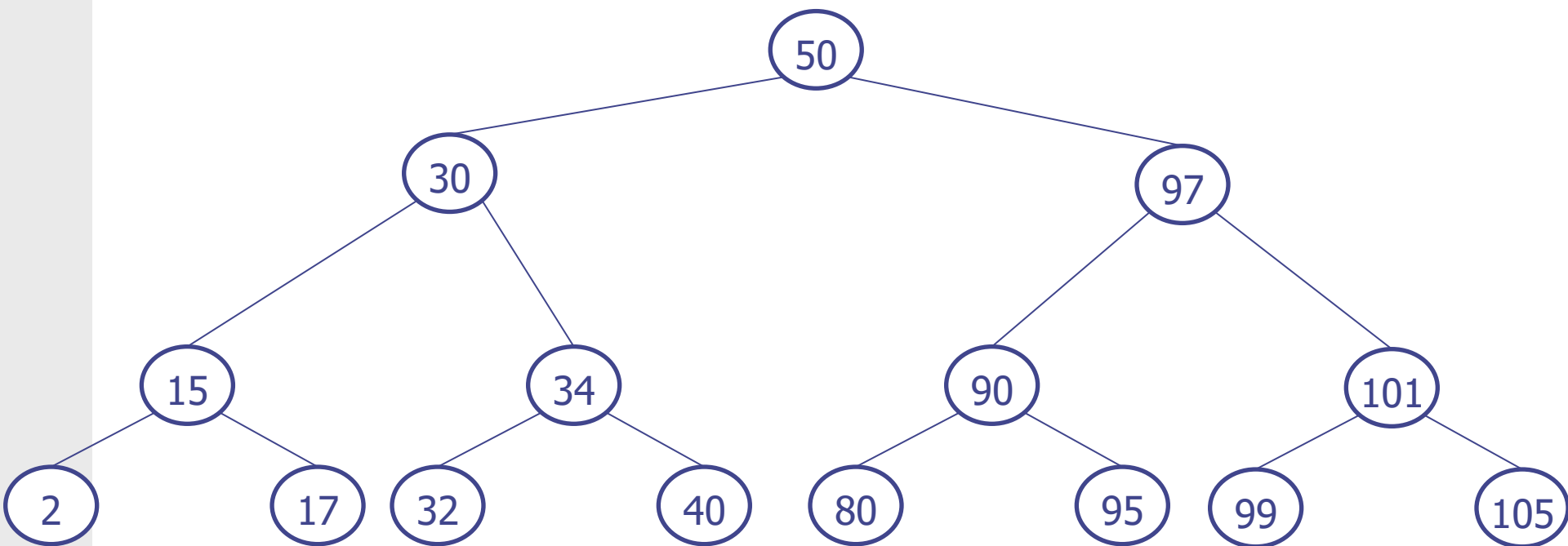
Iteração	inferior	superior	Metade = (inferior + superior) / 2	Elemento
1 ^a .				
2 ^a .				
3 ^a .				
4 ^a .				

Uma Pesquisa Binária

- Argumento de pesquisa: **32**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	15	17	30	32	34	40	50	80	90	95	97	99	101	105

↑ ↑ ↑ ↑
2^a 4^a 3^a 1^a



Outra Pesquisa

- Argumento de pesquisa: **54**

0	1	2	3	4	5	6	7	8	9
21	32	43	54	65	76	87	98	109	200
	↑	↑	↑	↑					
	2 ^a	3 ^a	4 ^a	1 ^a					

Mais uma Pesquisa

- Argumento de pesquisa: **100**

0	1	2	3	4	5	6	7	8	9
21	32	43	54	65	76	87	98	109	200

1^a

2^a

3^a

Algoritmo de Pesquisa Binária

- Localizar, por busca binária, a posição ocupada pelo elemento de valor x em um array a , ordenado.
- Parâmetros:
 - a : array onde será feita a pesquisa;
 - x : argumento de pesquisa.
- Retorno:
 - -1 : não encontrou;
 - $\neq -1$: elemento está na posição.

Algoritmo de Pesquisa Binária

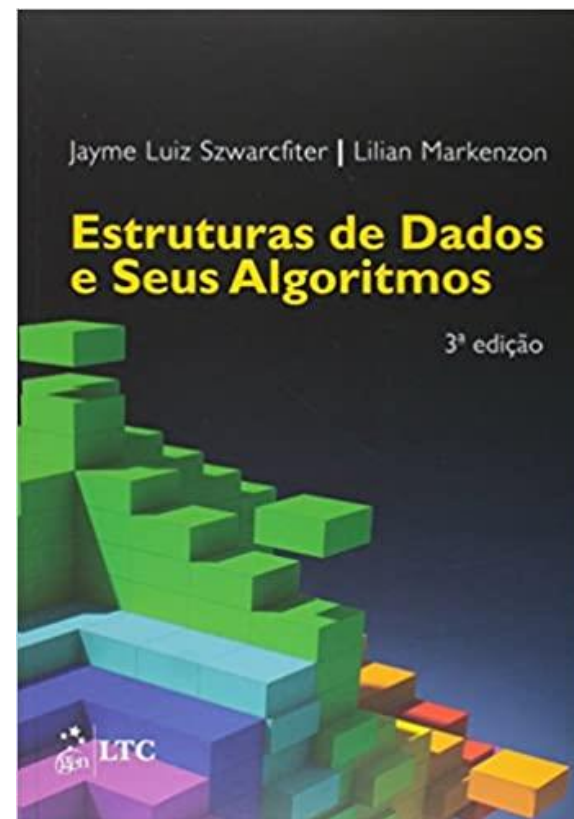
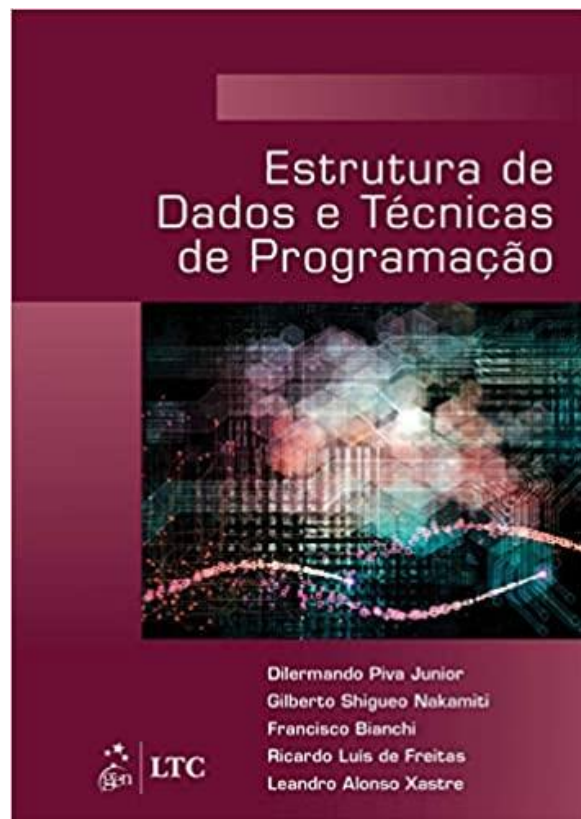
```
def binary_search(a: np.array, key: int) -> int:
    inf: int = 0
    sup: int = len(a) - 1
    while inf <= sup:
        middle: int = (inf + sup) // 2 # divisão inteira
        if key == a[middle]:
            return middle;
        elif key < a[middle]:
            sup = middle - 1 # procura na 1a. metade
        else:
            inf = middle + 1 # procura na 2a. metade
    return -1
```

Exemplo: busca
do elemento 43

0	1	2	3	4	5	6	7	8	9
21	32	43	54	65	76	87	98	109	200

↑ ↑ ↑
2º 3º 1º

Referências Bibliográficas



Material de Programação II. Professores de Programação II e Laboratório II. Acessado em 01/03/2022.