

Algoritmos e Estruturas de Dados

Vetores: Pesquisa

Prof. Maiquel de Brito

Pesquisa em vetores

Comumente grandes quantidades de dados são armazenados em vetores.

- $V[0] = 44$
- $V[2] = 230$
- $V[3] = 54$
- ...
- $V[998] = 6000$
- $V[999] = 37$

Como verificar se este vetor contém um determinado valor?

Pesquisa em vetores

Problema:

- Verificar se um valor existe no vetor
 - A. no caso de existir, indicar sua posição.
- No caso de vetores com valores repetidos:
 - A. indicar a posição da primeira ocorrência
 - B. indicar a posição da última ocorrência
 - C. indicar a posição de uma ocorrência qualquer

Pesquisa

Duas técnicas para pesquisa em vetores

- Pesquisa Linear/Sequencial
- Pesquisa Binária

Pesquisa Sequencial

Uma solução possível consiste em percorrer sequencialmente todas as posições do vetor.

Para cada posição i , compara-se $\text{vetor}[i]$ com o *valor* desejado.

- Se forem iguais diz-se que o *valor* existe
- Se chegarmos ao fim do vetor sem sucesso diz-se que o *valor* não existe

| | | | | | | | |
|---|---|---|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 7 | 9 | 5 | 11 | 19 | 15 | 13 | 17 |

Pesquisa Sequencial

1º Passo: Inicialização

```
int i = 0;  
int encontrado = 0; /* falso */
```

Pesquisa Sequencial

2º Passo: Pesquisa

```
while (i < tamanho && encontrado!=1){  
    if(vetor[i]== valor){  
        encontrado = 1; /* Verdadeiro */  
    }  
    i++;  
}
```

Pesquisa Sequencial

3º Passo: tratamento dos resultados

```
if (encontrado==1){  
    printf("Valor %d está na posição %d\n",  
           vetor[i],i);  
}else{  
    printf("Valor %d nao encontrado\n", valor);  
}
```


Pesquisa Sequencial

Quanto tempo a busca sequencial demora para executar? Em outras palavras, quantas vezes a comparação `valor == vetor[i]` é executada?

- Caso valor não esteja presente no vetor:
 - n vezes.
- Caso valor esteja presente no vetor:
 - Melhor caso: 1 vez (valor está na primeira posição).
 - Pior caso: n vezes (valor está na última posição).
 - Caso médio: $n/2$ vezes

Implementação da Pesquisa Sequencial em C

```
/* Procura um valor inteiro (x) num vetor (v). Retorna o índice
da sua primeira ocorrência, se encontrar; senão, retorna -1. */

int pesquisaSequencial(int *v, int tamanho, int x){
    int i;
    for (i = 0; i < tamanho; i++)
        if (v[i] == x)
            return i; // encontrou

    return -1; // não encontrou
}
```

Pesquisa em Vetores Ordenados

Supondo que o vetor inicial está ordenado em ordem crescente, é possível resolver o problema de modo mais eficiente?

- Caso o vetor esteja ordenado:
dados i e j , se $i < j$, então: $A[i] \leq A[j]$
- Portanto, comparando um determinado elemento com o elemento procurado, saberemos:
 - se o elemento procurado é o elemento comparado;
 - se ele está antes do elemento comparado ou;
 - se está depois.

| | | | | | | | |
|---|---|---|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 |

Pesquisa em vetores ordenados

- Se fizermos isso sempre com o elemento do meio da lista, a cada comparação dividiremos o vetor em dois, reduzindo nosso tempo de busca.
- Se em um determinado momento o vetor, após sucessivas divisões, tiver tamanho zero, então o elemento não está no vetor.

Pesquisa Binária

Algoritmo:

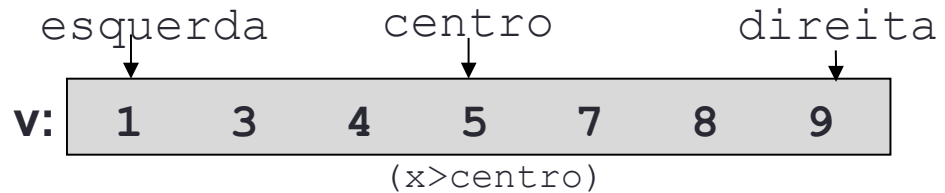
- Comparar o valor que se encontra a meio do vetor com o valor procurado:
 - A. = valor procurado - está encontrado
 - B. >valor procurado - continuar a procurar (do mesmo modo) no sub-vetor à esquerda da posição inspecionada
 - C. <valor procurado - continuar a procurar (do mesmo modo) no sub-vetor à direita da posição inspecionada.
- Se o vetor a inspecionar se reduzir a um vetor vazio, conclui-se que o valor procurado não existe.



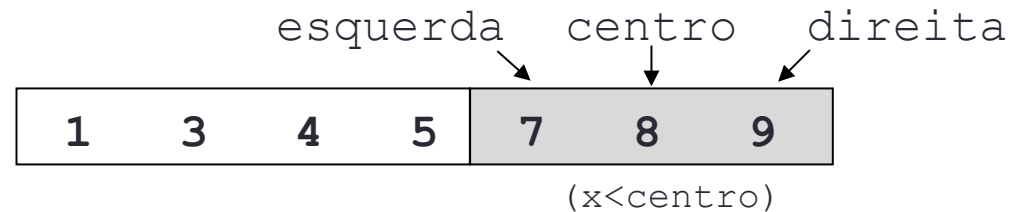
Exemplo de Pesquisa Binária

x: 7

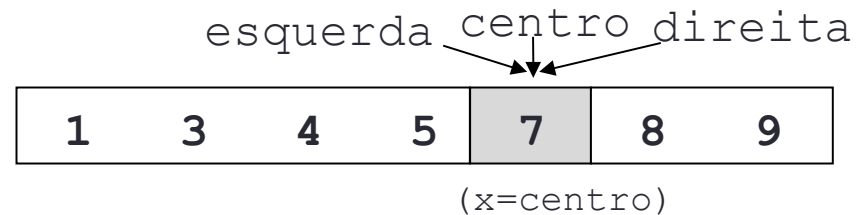
1ª iteração



2ª iteração

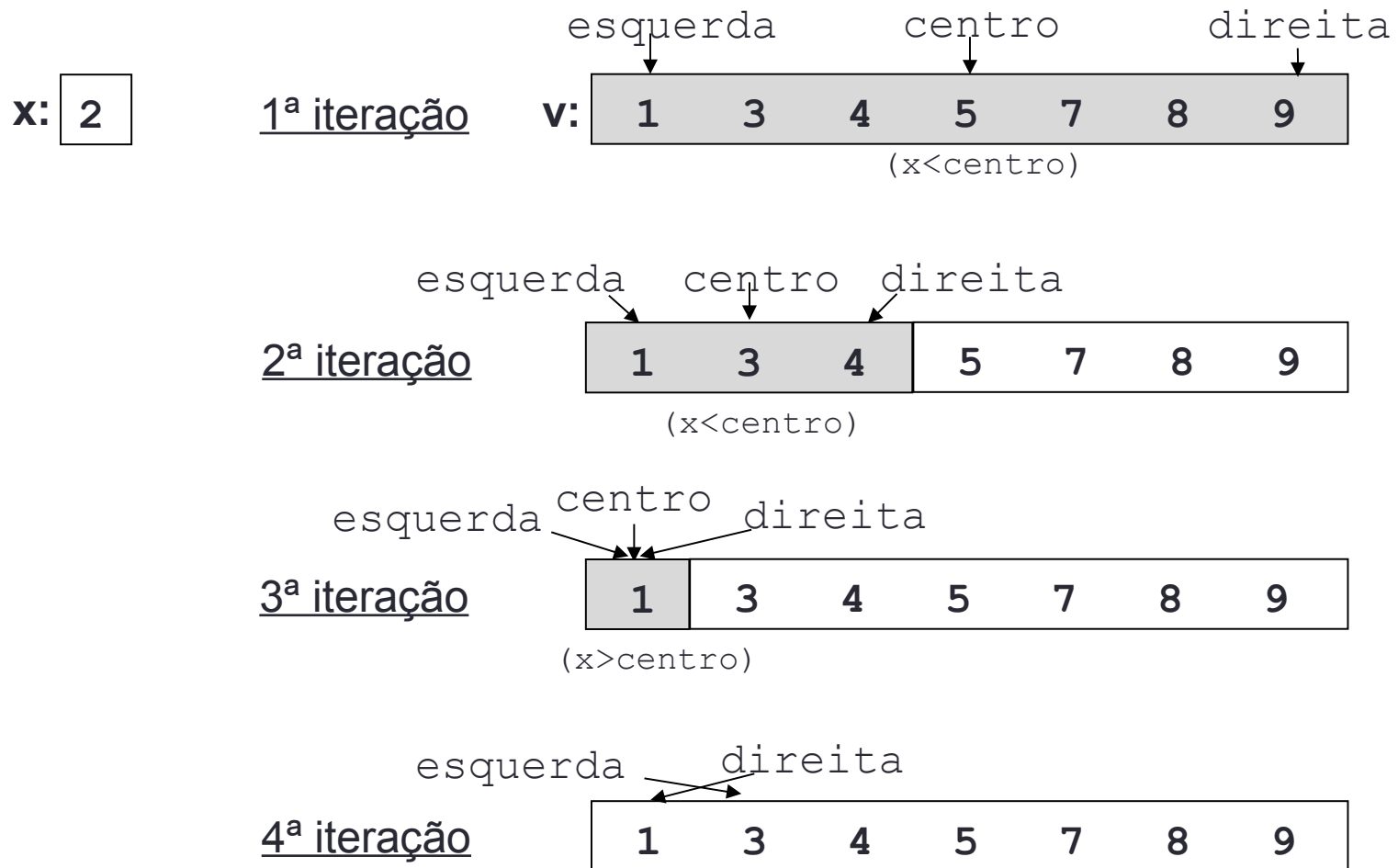


3ª iteração



O valor procurado está no centro do vetor → valor encontrado

Exemplo de Pesquisa Binária



vetor a inspecionar vazio \Rightarrow o valor 2 não existe no vetor inicial !

Encontrando o elemento central

Caso 1: vetores com número n **ímpar** de elementos

| | | | | | | | |
|-----------|--------|---|---|---|---|---|---|
| | centro | | | | | | |
| | ↓ | | | | | | |
| v: | 1 | 3 | 4 | 5 | 7 | 8 | 9 |
| Índice: | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

$$\text{meio} = (\text{inicio} + \text{fim}) / 2$$

Caso 2: vetores com número n **par** de elementos

| | | | | | | | | |
|-----------|--------|---|---|---|---|---|---|----|
| | centro | | | | | | | |
| | ↓ ↓ | | | | | | | |
| v: | 1 | 3 | 4 | 5 | 7 | 8 | 9 | 10 |
| Índice: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

$$\text{centro} = \left\lfloor \frac{\text{inicio} + \text{fim}}{2} \right\rfloor$$

ou

$$\text{centro} = \left\lceil \frac{\text{inicio} + \text{fim}}{2} \right\rceil$$

Complexidade

Avaliar a complexidade da busca binária consiste em avaliar quantas comparações são feitas na busca por um elemento em um vetor.

Seja um vetor com n elementos e uma função $W(n)$ que retorna o número de comparações neste vetor. Pode-se dizer que:

$$W(n) = 1 + W\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$$

Considerando as sucessivas divisões de um vetor, tem-se

Complexidade

Considerando as sucessivas divisões de um vetor, tem-se

| Iteração (i) | Tamanho do vetor | Comparações |
|--------------|------------------|--|
| 1 | n | $W(n) = 1 + W\left(\left\lceil \frac{n}{2} \right\rceil\right)$ |
| 2 | n/2 | $W(n) = 1 + 1 + W\left(\left\lceil \frac{n}{2} \right\rceil\right) = 1 + W\left(\left\lceil \frac{n}{2^2} \right\rceil\right)$ |
| 3 | n/4 | $W(n) = 1 + 1 + 1 + W\left(\left\lceil \frac{n}{4} \right\rceil\right) = 1 + W\left(\left\lceil \frac{n}{2^3} \right\rceil\right)$ |
| ... | ... | |
| k | 1 | $W(n) = k + W\left(\left\lceil \frac{n}{2^k} \right\rceil\right)$ |

$$W(n) = k + W\left(\left\lceil \frac{n}{2^k} \right\rceil\right)$$

$$W\left(\left\lceil \frac{n}{2^k} \right\rceil\right) = W(1)$$

$$\frac{n}{2^k} = 1$$

$$n = 2^k$$

$$k = \log(n)$$

$$W(n) = k + W\left(\left\lceil \frac{n}{2^k} \right\rceil\right) = \log(n) + 1$$

Complexidade: $O(\log_2^n)$