

# **Busca linear, busca binária e análise de algoritmos**

---

Fabio Lubacheski  
fabio.aglubacheski@sp.senac.br

# Busca de um elemento

Considere o problema em se determinar se um elemento está presente em uma lista de elementos, ou seja:

**Dado um inteiro  $x$  e um vetor  $v[0 \dots n-1]$  de inteiros, o problema da busca consiste em encontrar  $x$  em  $v$ , ou seja, encontrar um índice  $k$  tal que  $v[k] == x$ .**

# Busca de um elemento

Poderíamos realizar a busca comparando o valor do elemento a ser encontrado com os elementos do vetor, um a um, da esquerda para a direita, sequencialmente (=linearmente).

Esse tipo de busca é denominada busca linear, e para implementar a busca linear é preciso tomar algumas decisões importante antes de começar, vamos considerar as configurações de entrada abaixo:

- O que fazer quando  $x$  está no vetor ?
- O que fazer se  $x$  não está no vetor?
- O que fazer se tivermos elementos repetidos, ou seja, aparece duas vezes no vetor ?

# Busca linear

A função implementada abaixo devolve a posição em que o elemento  $x$  se encontra no vetor, caso ele esteja lá, ou um valor inválido ( $-1$ ) que represente o insucesso da busca caso o elemento não esteja no vetor. Caso tenhamos elementos repetidos é retornado o índice da primeira ocorrência.

```
public static int buscaLinear(int v[], int x) {  
    for( int k=0; k < v.length; k++ ){  
        if( v[k] == x )  
            return k;  
    }  
    return -1;  
}
```

# Busca linear

Existe alguma outra forma de fazer a busca de um elemento dentro do vetor ?

Será que é melhor ? Como comparar as duas formas ?

Vejam o vídeo abaixo para se inspirarem da computação desplugada.

<https://www.youtube.com/watch?v=iDVH3oCTc2c>

# Busca binária

Quando o vetor está **ordenado** podemos utilizar uma **técnica de programação** chamada de **divisão-e-conquista** para realizarmos uma busca binária:

- Dividimos o vetor ao meio
- Se o elemento procurado for o elemento do meio, pare.
- Senão, se o elemento procurado for menor que o elemento do meio, repetimos o processo para os elementos à esquerda do elemento do meio; caso contrário, repetimos o processo nos elementos à direita do elemento do meio.

# Busca Binária

A função recebe um número  $x$  e um vetor em ordem crescente  $v[0..n-1]$  com  $n$  elementos. A função devolve um índice  $m$  tal que  $v[m] == x$ , ou seja achou  $x$  em  $v[]$ , ou devolve  $-1$  se tal  $m$  não existe.

Vocês conseguem fazer o método?

# Busca Binária

A função recebe um número  $x$  e um vetor em ordem crescente  $v[0..n-1]$  com  $n$  elementos. A função devolve um índice  $m$  tal que  $v[m] == x$  ou devolve  $-1$  se tal  $m$  não existe.

```
public static int buscaBinaria( int v[], int x ){
    int i, m, f;
    i = 0; f = v.length-1;
    while (i <= f){
        m = (i + f)/2;
        if (v[m] == x)
            return m;
        if (x < v[m]) // esquerda
            f = m - 1;
        else // x > v[m] - direita
            i = m + 1;
    }
    return -1;
}
```



# Comparação de algoritmos

Como podemos comparar **dois algoritmos diferentes** que resolvem o **mesmo problema**, ou seja, qual algoritmo é mais **eficiente (mais rápido)**.

Isso nos leva à algumas questões:

- Como comparar os algoritmos **independente** do **computador** usado para execução?
- A **linguagem** de programação influência ?
- Qual a importância do **sistema operacional** na comparação ?

# Análise de algoritmo – tempo de execução

A análise de um algoritmo determina o ***número de passos*** (aproximados) que o algoritmo executa dado o **tamanho de sua entrada**, no caso da busca linear e binária o tamanho da entrada que importa é o **tamanho do vetor**.

Considere que **um passo** são todas as **instruções** que um algoritmo executa a **cada iteração** (repetição) do algoritmo.

Podemos dizer então que **número de passos** executados pelo algoritmo é o **tempo de execução** do algoritmo.

# Análise da busca linear

Qual seria o número de passos da busca linear ? Os valores dentro do vetor e o valor do  $x$  influenciam no cálculo dos passos do algoritmo ?

Um algoritmo pode ter várias entradas possíveis, para calcular o **número de passos** é considerado a configuração da entrada mais desfavorável para uma entrada de tamanho  $n$ .

Qual seria a **configuração de entrada** para função **busca linear** que teríamos o **maior** número de passos ? Ou seja qual é o **pior caso** da busca linear ?

# Análise da busca linear - pior caso

O **pior caso da busca linear** é quando o valor procurado **não se encontra no vetor**, pois o algoritmo realizará a comparação do valor procurado com todos os elementos do vetor.

Podemos dizer então que **busca linear** no **pior caso** executa aproximadamente  $n$  passos para verificar se um determinado elemento está presente no vetor de tamanho  $n$ , onde  $n$  é o tamanho da entrada. (complexidade do algoritmo)

O **pior caso** do algoritmo é importante pois assim teremos um **limite superior**, e considerar que o **algoritmo nunca ultrapassará o número de passos calculado**.

# Análise da busca binária

Para encontrar a função de complexidade da **busca binária** é necessário identificar o pior caso do algoritmo.

**O pior caso da busca binária é quando o elemento procurado não pertence a lista.**

Agora, vamos realizar vários testes considerando o pior caso da Busca Binária, variando o tamanho do vetor  $n$  de **1** até **128**.

# Análise da busca binária

$n$	Passos
1	1
2	2
4	3
8	4
16	5
32	6
64	7
128	8

Note que os valores calculados para os passos são modificados somente para  $n=1$ ,  $n=2$ ,  $n=4$ ,  $n=8$ ,  $n=16$ , ...,  $n=128$ .

A partir dessa análise poderíamos deduzir que se tivermos  $n=256$  a Busca Binária executaria 9 passos, conseqüentemente se  $n=1024$  teríamos 11 passos.

# Análise da busca binária

$n$	Passos
1	1
2	2
4	3
8	4
16	5
32	6
64	7
128	8

Podemos que se  $n$  igual 1 poderíamos escrever  $n$  como  $2^0$ , para  $n=2=2^1$ , para  $n=4=2^2$ , para  $n=8=2^3$ ,  $n=16=2^4$ , e assim sucessivamente.

E o valor do expoente somado a 1 seria o número de passos.

Como usar uma expressão matemática para descrever isso ?

# Análise da busca binária

$n$	Passos
1	1
2	2
4	3
8	4
16	5
32	6
64	7
128	8

Concluimos que assim que para uma entrada de tamanho  $n$  a **busca binária** executa  $\log_2 n + 1$  passos, considerando o pior caso.



# Comparação entre busca binária e linear

Para uma entrada de tamanho  $n$  a busca **binária** executa  $\log_2 n + 1$  passos no seu pior caso.

Concluimos que a **busca binária** é mais **eficiente** que a **busca linear**, pois dado o valor de  $n$  (**para  $n$  bem grande???**) a **busca binária** executa **muito menos passos** que a **busca linear**.

# Exercícios

- 1) Considere o algoritmo que faz a **busca linear** no vetor, ele funciona ?  
Critique o algoritmo, ou seja, de um exemplo de entrada que o algoritmo não irá funcionar:

```
public static int buscaCriticar( int x, int v[]) {  
    int m = 0;  
  
    while (v[m] < x && m < v.length)  
        m++;  
  
    if (v[m] == x)  
        return m;  
    else  
        return -1;  
}
```

- 2) Reescreva a função de busca binária para que ela funcione em um vetor decrescente.

# Exercícios

3) Ao final da execução do trecho de algoritmo abaixo, qual o **valor de S em função de  $n$**  ? Dê uma **expressão matemática para S**, ou seja, o **número de passos** do trechos de algoritmos

a)

$S \leftarrow 0$

**para i de 1 até n faça**

**para j de 1 até n faça**

$S \leftarrow S + 1$

Lembrando que o trecho de código acima em Java ficaria assim:

```
int S=0;
```

```
for( int i=0; i <= n; i++ )
```

```
    for(int j=0; j<=n;j++)
```

```
        S=S+1;
```

# Exercícios

3)

b)

$S \leftarrow 0$

$i \leftarrow 1$

$j \leftarrow 1$

**enquanto**  $i \leq n$  **faça**

$S \leftarrow S + 1$

$j \leftarrow j + 1$

**se**  $j > n$  **então**

$i \leftarrow i + 1$

$j \leftarrow 1$

**fim-se**

**fim-enquanto**

# Exercícios

c)

$S \leftarrow 0$

**para i de 1 até n faça**

**para j de 1 até i faça**

$S \leftarrow S + 1$

d)

$S \leftarrow 0$

**para i de 1 até n faça**

**para j de 1 até n faça**

**para k de 1 até j faça**

$S \leftarrow S + 1$

e)

$S \leftarrow 0$

**enquanto n > 0 faça**

$S \leftarrow S + 1$

$n \leftarrow n / 2$

# Exercícios

4) Escreva o algoritmo que recebe um vetor  $A$  de tamanho  $n$  contendo inteiros e encontra um par de elementos distintos  $a$  e  $b$  do vetor que fazem com que a diferença absoluta  $a-b$  seja a maior possível. A função deve resolver o problema em no máximo  **$n$  passos** (tamanho do vetor) e ao final retornar a maior diferença entre os números  $a$  e  $b$ .

5) Dado um vetor de  $n$  números inteiros, faça uma função para determinar o comprimento de um segmento crescente de comprimento máximo.

Exemplos: na sequência

$\{5, 10, 3, 2, 4, 7, 9, 8, 5\}$  o comprimento do segmento crescente máximo é 4  $\{2, 4, 7, 9\}$ .

na sequência

$\{10, 8, 7, 5, 2\}$  o comprimento de um segmento crescente máximo é 1.

A função deve ter no máximo  **$n$  passos**, ou seja, o tamanho do vetor.

# Exercícios de complexidade de algoritmos

- 6) Dado um vetor com números pares e ímpares, escreva uma função para colocar todos os números pares à frente no vetor e os ímpares ao final. Você não pode usar outro vetor como área auxiliar. Resolva esse exercício sem um vetor um auxiliar e a função deve rearranjar o vetor com no máximo  **$n$  passos**.
- 7) A **intercalação** é o processo utilizado para construir uma vetor ordenado, de tamanho  $n+m$ , a partir de dois vetores já ordenados de tamanhos  $n$  e  $m$ . Escreva uma função que receba dois vetores ( $A[]$  e  $B[]$ ), com  $n$  e  $m$  elementos, respectivamente. Os vetores estão ordenados em ordem crescente, a função aloca um vetor  $C[]$ , exatamente com soma dos tamanhos de  $A$  e  $B$ , e intercala os elementos de  $A[]$  e  $B[]$  em  $C[]$ , de forma que o vetor  $C[]$  fique em ordem crescente. A função deve ter no máximo  **$n+m$  passos**, ou seja, a soma dos tamanho dos vetores.

Exemplo:

$A[] = \{ 1, 3, 6, 7 \}$  e  $B[] = \{ 2, 4, 5 \}$ ,

o novo vetor  $C$  é preenchido a partir de  $A[]$  e  $B[]$ :

$C[] = \{ 1, 2, 3, 4, 5, 6, 7 \}$

Fim