BCC702 - Programação de Computadores II

Algoritmos de Busca
Comparação de Algoritmos

Profª Valéria de Carvalho Santos

UFOP

decom
departamento
de computação

O problema de busca

- Imagine um vetor de alunos, sendo que aluno é um registro com os campos matrícula, nome e curso.
- Como você faria para localizar um determinado aluno?

O problema de busca

- O objetivo da busca (ou pesquisa) é encontrar uma ou mais ocorrências de registros com valores iguais ao valor procurado.
- Existem vários métodos de pesquisa. A escolha do método mais adequado depende, principalmente:
  - da quantidade de dados envolvidos;
  - da possibilidade de o arquivo sofrer inserções e/ou retiradas.

Algoritmos de busca

- Busca sequencial
- Busca binária

Busca sequencial

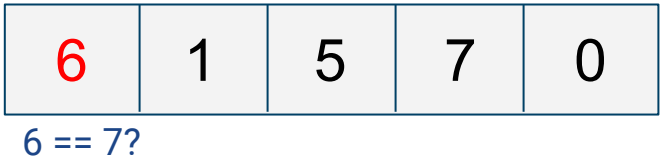
- É o método de pesquisa mais simples que existe.
- Funcionamento:
  - a partir do primeiro registro, pesquise sequencialmente até encontrar o valor procurado ou até chegar ao fim do vetor;
  - então pare.

Busca sequencial - Exemplo

- Valor procurado: 7

Busca sequencial - Exemplo

- Valor procurado: 7

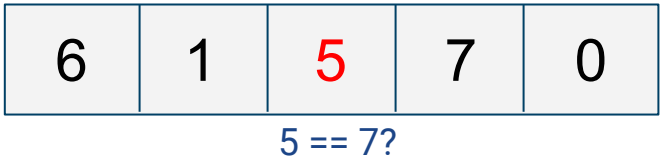


Busca sequencial - Exemplo

- Valor procurado: 7

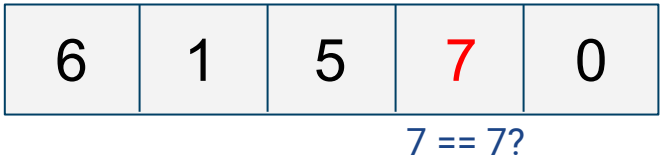
Busca sequencial - Exemplo

- Valor procurado: 7



Busca sequencial - Exemplo

- Valor procurado: 7



Algoritmo de busca sequencial

```
bool busca_sequencial(int v[], int n, int procurado){
    for(int i = 0; i < n; i++){
        if(v[i] == procurado){
            return true;
        }
    }
    return false;
}
```

Busca binária

- Método de busca eficiente para um vetor ordenado
- Esse método é semelhantes ao que *usávamos* para procurar uma palavra no dicionário, por exemplo.

Busca binária

- Compare o elemento procurado com o elemento que está na posição do meio do vetor:
  - Se o elemento foi **encontrado**, termine a busca.
  - Se o elemento procurado é **menor** do que o elemento do meio, repita a busca para a primeira metade do vetor.
  - Se o elemento procurado é **maior** do que o elemento do meio, repita a busca para a segunda metade do vetor.
- Esse processo se repete até que o elemento seja encontrado ou até que todo vetor tenha sido consultado sem sucesso

Busca binária - Exemplo

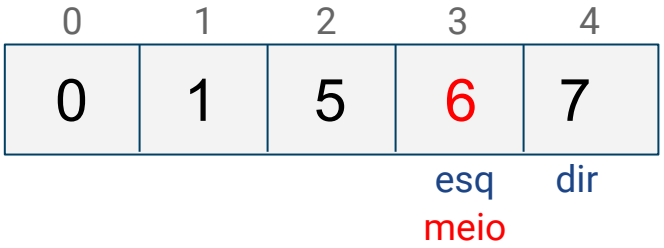
- Valor procurado: 7

Busca binária - Exemplo

- Valor procurado: 7

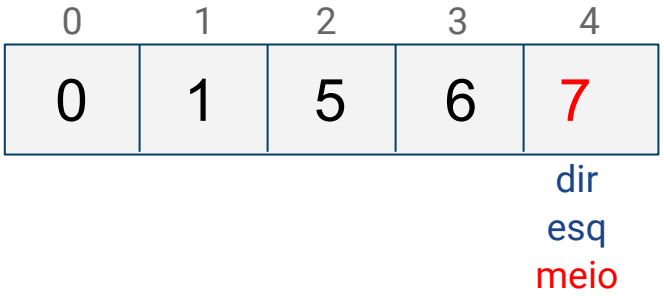
Busca binária - Exemplo

- Valor procurado: 7



Busca binária - Exemplo

- Valor procurado: 7



Algoritmo de busca binária

```
bool busca_binária(int v[], int n, int procurado){
    int esq = 0, dir = n-1, meio;
    while(esq <= dir){
        meio = (esq+dir)/2;
        if(v[meio] == procurado)
            return true;
        else if(v[meio] > procurado)
            dir = meio-1;
        else
            esq = meio+1;
    }
    return false;
}
```

Busca binária

- Se o elemento procurado estiver ou não no vetor, a busca binária apresenta resultado melhores do que a busca sequencial
- Porém, é preciso manter o vetor ordenado (alto custo computacional)
- Adequada quando a quantidade de consultas realizadas é bem maior do que a quantidade de inserções feitas no conjunto de elementos

Comparação de Algoritmos

Comparação de algoritmos

- Dado um determinado problema, pode haver diferentes soluções algorítmicas para ele
- Como determinar qual solução é mais eficiente?
  - O que é um algoritmo eficiente?
  - Como “medir” a eficiência de um algoritmo?

Eficiência de algoritmos

- Eficiência de espaço
  - Refere-se à quantidade de memória requerida pelo algoritmo, além do espaço necessário para entradas e saídas
- Eficiência de tempo
  - Indica quão rápido um algoritmo é executado

Eficiência de algoritmos

- Eficiência de espaço
  - Refere-se à quantidade de memória requerida pelo algoritmo, além do espaço necessário para entradas e saídas
- **Eficiência de tempo**
  - Indica quão rápido um algoritmo é executado
  - A maioria das pesquisas focam em melhorar a eficiência de tempo

Tamanho da entrada

- Um algoritmo vai demorar mais para executar entradas maiores
  - Exemplo: percorrer um vetor de 10 ou 1000 elementos
- Em geral, a eficiência do algoritmo é analisada em função do tamanho da sua entrada. Exemplo:
  - tamanho do vetor para problemas que trabalham com vetores: ordenação, buscar, encontrar o menor elemento, etc.
  - polinômio: grau do polinômio

Como medir o tempo?

- Unidades de tempo (minutos, segundos...)
  - depende do computador
  - depende do compilador
- Contar quantas vezes cada operação é executada
  - muito difícil
  - desnecessário
- Identificar a operação mais importante: **operação básica**

Como medir o tempo?

- Operação básica
  - operação que impacta mais no tempo de execução
  - em geral, é a operação no laço mais interno do algoritmo
  - contar quantas vezes ela é executada
- Qual é a operação básica da busca sequencial?

Pior caso, melhor caso e caso médio

- A eficiência de alguns algoritmos dependem não apenas do tamanho da entrada, mas também de uma entrada específica
- Exemplo: busca sequencial

Pior caso

- É uma entrada de tamanho *n* para a qual o algoritmo gastará o **maior** tempo possível entre todas as entradas do mesmo tamanho
- Como calcular a eficiência no pior caso:
  - verificar o tipo de entrada(s) que resulta no maior valor de vezes que a operação básica vai ser executada
- Qual é o pior caso da busca sequencial?

Melhor caso

- É uma entrada de tamanho *n* para a qual o algoritmo gastará o **menor** tempo possível entre todas as entradas do mesmo tamanho
- Como calcular a eficiência no melhor caso:
  - verificar o tipo de entrada(s) que resulta no menor valor de vezes que a operação básica vai ser executada
- Qual é o melhor caso da busca sequencial?

Caso médio

- Representa uma entrada comum ou aleatória
- Mais complexa de calcular
- Divide todas as instâncias de tamanho *n* em várias classes, sendo que para cada instância da classe, o número de vezes que a operação básica do algoritmo é executada é a mesma

Ordem de crescimento

- Avalia o quanto o tempo de execução de um algoritmo aumenta, de acordo com o aumento do tamanho da entrada *n*
- Considera valores grandes de *n*
- Para comparar a ordem de crescimento dos algoritmos, há três notações: O, Ω e θ
- Vamos ver a definição da notação O

Notação *O*

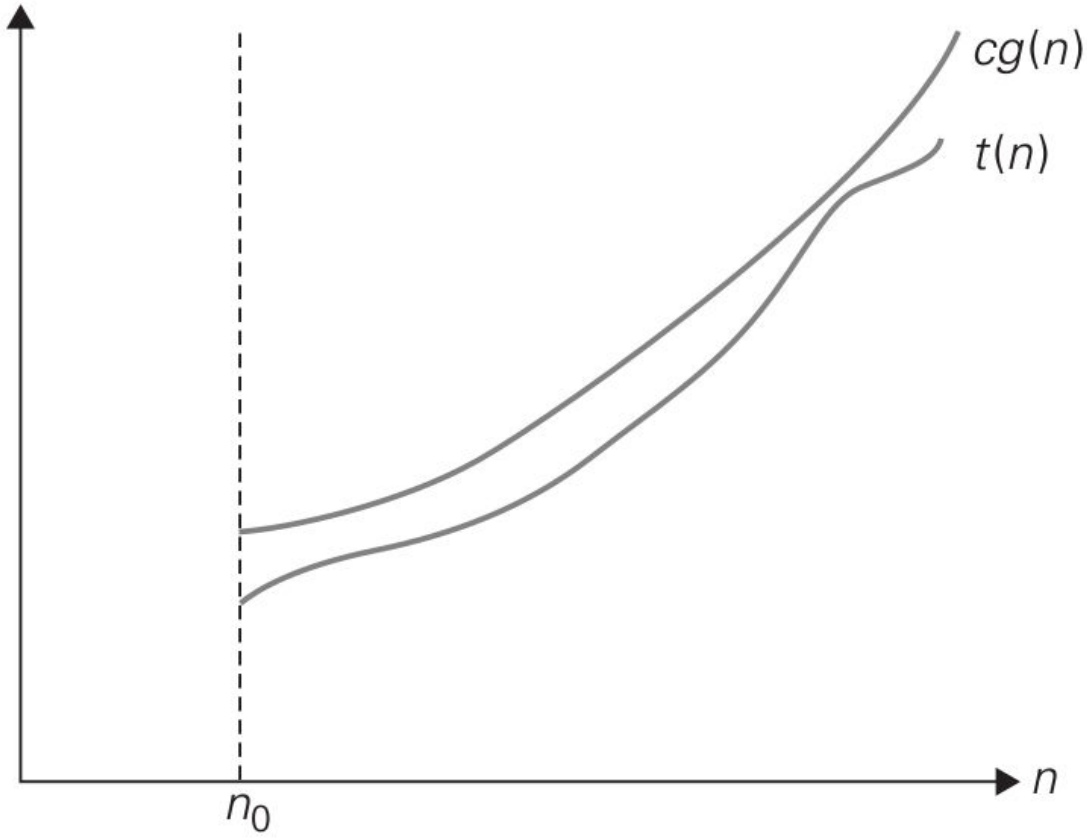
- Considere uma função *g(n)* que representa um algoritmo
- Informalmente, *O(g(n))* é o conjunto de todas as funções com ordem de crescimento menor ou igual a *g(n)*.
- Exemplo:
  - $n \in O(n^2)$
  - $100n+5 \in O(n^2)$
  - $n(n-1) \in O(n^2)$
  - $n^3 \notin O(n^2)$

Notação *O*

- Definição: Uma *t(n)* pertence a *O(g(n))*, *t(n)* ∈ *O(g(n))*, se t(n) é limitada acima por alguma constante múltipla de g(n), se existe alguma constante positiva c e algum inteiro não-negativo n_0, tal que:

$$t(n) \leq c(g(n)), \text{ para todo } n \geq n_0$$

Notação *O*



Classes de algoritmo

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Exercício 1

- Considere a função que calcula o maior elemento de um vetor.

```
bool maior(int v[], int n){
      int maior = v[0];
      for(int i=1; i<n; i++){
            if(v[i]>maior)
                  maior = v[i];
      }
      return maior;
}
```

- Qual é a operação básica?
- Quantas vezes a operação básica é executada?
- Qual é a classe desse algoritmo?

Exercício 2

- Considere a operação de atribuição destacada.

```
bool maior(int v[], int n){
      int maior = v[0];
      for(int i=1; i<n; i++){
            if(v[i]>maior)
                  maior = v[i];
      }
      return maior;
}
```

- Em que situação ele será executada o menor número de vezes?
- Em que situação ele será executada o maior número de vezes?