Busca em Vetores

Busca sequencial, sequencial ordenada e busca binária

Maria Adriana Vidigal de Lima

FACOM - UFU

Sumário

- Problema da Busca
- Busca Sequencial
- Busca Sequencial Ordenada
- Busca Binária

Problema da Busca

Problema: Dado um vetor *vet* com *n* elementos, deseja-se saber se um determinado elemento *x* está ou não **presente** no vetor.

Questão: Que algoritmos existem para encontrar um elemento *x* num conjunto de dados (vetor) de forma eficiente?

Problema da Busca

Definições do Problema:

- Entradas:
  - vetor V com n elementos
  - elemento a ser procurado
- Saída:
  - m se o elemento procurado está em $V[m]$ ou -1 se o elemento não existe.

Algoritmos de Busca:

- Sequencial (Linear)
- Binária

Busca Sequencial

A **busca sequencial** é o método de pesquisa mais simples que existe.

Funcionamento: a partir do primeiro elemento, pesquisar sequencialmente até encontrar o valor procurado ou até chegar ao fim do vetor.

Exemplo: Buscar o valor 8 no vetor V = {4,5,2,8,3}:

i=0	4	5	2	8	3	não encontrado, avançar
i=1	4	5	2	8	3	não encontrado, avançar
i=2	4	5	2	8	3	não encontrado, avançar
i=3	4	5	2	8	3	encontrado, encerrar

Busca Sequencial: Implementação

Algoritmo: Percorrer o vetor, posição a posição, verificando se o elemento procurado é igual a um dos elementos do vetor.

```
1  int busca_sequencial(int v[], int n, int elemento){
2  int i;
3    for(i = 0; i < n; i++){
4       if(v[i] == elemento){
5           return i;
6       }
7   }
8    return -1;
9  }
```

Esse algoritmo de busca é extremamente simples, mas pode ser muito ineficiente quando o número de elementos no vetor for muito grande.

Busca Sequencial: Análise

Avaliação dos casos quanto ao tempo de execução:

- **Pior Caso**: o elemento não está no vetor. Neste caso são necessárias *n* comparações e, portanto: T(n) = n $\rightarrow$ **O(n)**
- **Melhor Caso**: o elemento é o primeiro, portanto T(n) = 1 $\rightarrow$ **O(1)**
- **Caso Médio**: na média, pode-se concluir que são necessárias n/2 comparações, portanto T(n) = n/2 $\rightarrow$ **O(n)**

Busca Sequencial: como ser mais eficiente?

Se os elementos estiverem armazenados em uma ordem aleatória no vetor, não é possível melhorar o algoritmo de busca, pois é necessário verificar todos os elementos.

No entanto, **se os elementos estiverem armazenados em ordem crescente**, pode-se concluir que um elemento não está presente no vetor ao ser encontrado um elemento maior: se o elemento que buscamos estivesse presente ele precederia um elemento maior na ordem do vetor.

Problema da busca em vetor ordenado: Dado um inteiro *x* e um vetor inteiro de tamanho *n* com seus elementos em ordem crescente V[1..n], encontrar i tal que $V[i-1] < x \leq V[i]$.

Busca Sequencial Ordenada

Funcionamento: a partir do primeiro elemento, pesquisar sequencialmente até encontrar o valor procurado ou um valor superior ao procurado.

Exemplo: Buscar o valor 15 no vetor V = {2,7,11,16,20}:

i=0	2	7	11	16	20	não encontrado, 2 < 15, avançar
i=1	2	7	11	16	20	não encontrado, 7 < 15, avançar
i=2	2	7	11	16	20	não encontrado, 11 < 15 avançar
i=3	2	7	11	16	20	não encontrado, 16 ≮ 15, encerrar

Busca Sequencial Ordenada: Implementação

Algoritmo: Percorrer o vetor, posição a posição, verificando se o elemento procurado é igual a um dos elementos do vetor. Caso seja encontrado um valor superior ao do elemento procurado, a busca se encerra.

```
1  int busca_ordenada (int v[], int n, int elemento){
2    int i;
3    for (i=0; i<n; i++) {
4      if (v[i] == elemento)
5          return i;
6      else
7          if (v[i] > elemento)
8              return -1; // encerramento da busca
9    }
10   return -1;
11 }
```

Busca Sequencial Ordenada: Análise

Qual a complexidade do algoritmo de busca linear em vetor ordenado?

- **Pior Caso**: O elemento não foi encontrado, é maior que o último do vetor: T(n) = n $\rightarrow$ **O(n)**
- **Melhor Caso**: O elemento está na primeira posição do vetor: T(n) = 1 $\rightarrow$ **O(1)**

Um algoritmo pode ter a mesma complexidade de outro, porém pode ser mais, ou menos, eficiente.

Busca Binária

A busca binária é um método de busca eficiente para um vetor ordenado. Este método tem um mecanismo semelhante à procurar uma palavra no dicionário. Deve-se abrir o dicionário ao meio e:

- comparar a palavra procurada com a que está na página do meio e então:
  - se a palavra foi encontrada, encerra-se a busca.
  - se a palavra procurada for menor (na ordem alfabética), repetir a busca na primeira metade do dicionário.
  - se a palavra procurada for maior (na ordem alfabética), repetir a busca na segunda metade do dicionário.
- este processo se repete enquanto a palavra não for encontrada e enquanto existirem trechos do dicionário a serem consultados.

Busca Binária

Funcionamento:

- Dado um vetor *V* e um elemento *e* a ser procurado
- Compare o elemento e com o elemento do meio de *V*

  Se o elemento *e* for menor, pesquise na primeira metade de *V*
  Se o elemento *e* for maior, pesquise a segunda metade de *V*
  Se o elemento *e* for igual, retorne a posição
- Continue o processo subdividindo a parte de interesse até encontrar o elemento *e* ou chegar ao fim

Exemplo: Buscar o valor 16 no vetor V = {2,7,11,16,20,25,32,46,55}:

meio=4	2	7	11	16	20	25	32	46	55	não encontrado
meio=1	2	7	11	16	~~20~~	~~25~~	~~32~~	~~46~~	~~55~~	não encontrado
meio=2	~~2~~	~~7~~	11	16	~~20~~	~~25~~	~~32~~	~~46~~	~~55~~	não encontrado
meio=3	~~2~~	~~7~~	~~11~~	16	~~20~~	~~25~~	~~32~~	~~46~~	~~55~~	encontrado

Busca Binária: Implementação

Algoritmo: Percorrer o vetor, posição a posição, verificando se o elemento procurado é igual a um dos elementos do vetor. Caso seja encontrado um valor superior ao do elemento procurado, a busca se encerra.

```
1  int busca_binaria(int v[], int n, int elemento){
2     int esq = 0, dir = n-1, meio;
3     while (esq <= dir){
4       meio = (esq+dir)/2;
5       if (v[meio] == elemento)
6           return meio;
7       else
8           if(v[meio] > elemento)
9               dir = meio-1;
10          else
11              esq = meio+1;
12    }
13    return -1;
14 }
```

Busca Binária: Análise

O desempenho desse algoritmo é muito superior ao de busca linear. Novamente, o pior caso é quando o elemento não está no vetor. Quantas vezes é necessário repetir o procedimento de subdivisão para concluir que o elemento não está presente no vetor?

Repetição	Tamanho do vetor
1	n
2	n/2
3	n/4
...	...
$\log_2 n$	1

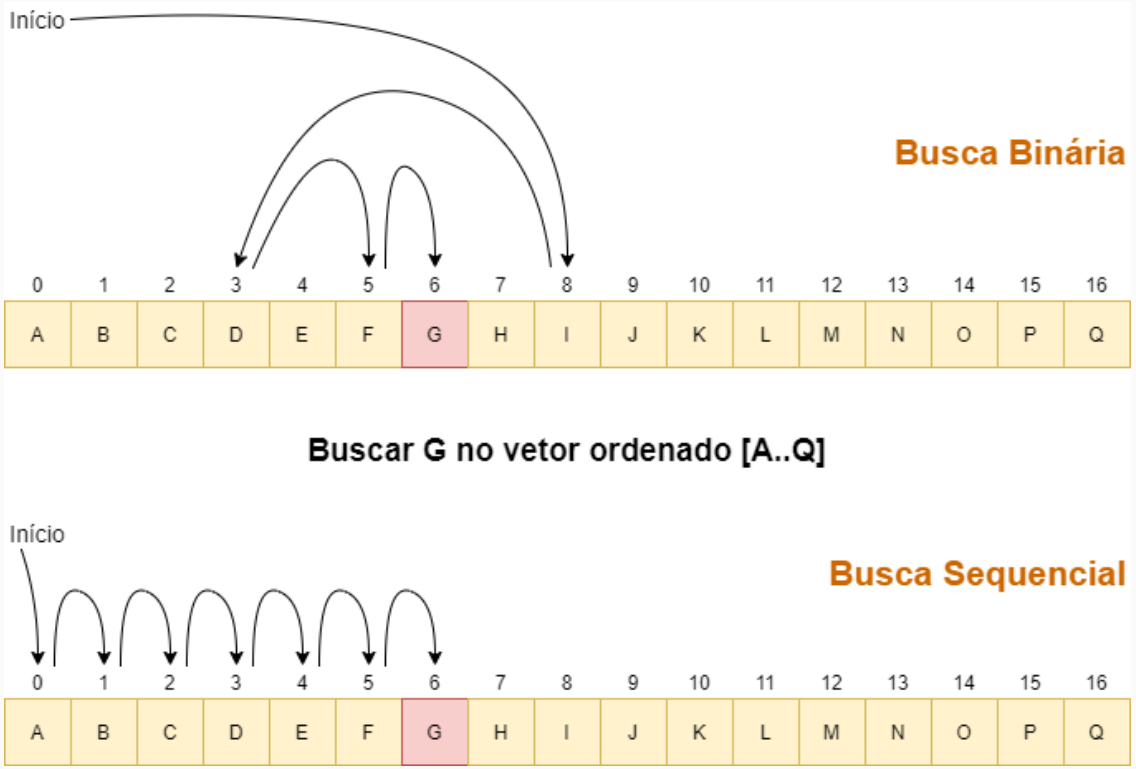
Considerando um vetor com N elementos, o tempo de execução é:

- **Pior Caso**: O elemento não foi encontrado: **$O(\log_2 n)$**
- **Caso Médio**: **$O(\log_2 n)$**
- **Melhor Caso**: O elemento está no meio do vetor: **$O(1)$**

Busca Binária: Análise e Visualização

Exemplo de execução, no pior caso, para a busca de um valor num vetor contendo 1000 elementos:

- Busca Sequencial: 1000 comparações
- Busca Binária: 10 comparações



Referência e Material Extra

- Celes, W.; Cerqueira, R.; Rangel, J.L. *Introdução a Estruturas de Dados com Técnicas de Programação em C*. 2a. ed. Elsevier, 2016.

* Backes, A. *Programação Descomplicada - Estruturas de Dados*.
    - Vídeo-aula 45: *http://youtu.be/ptvnLzqcJuA*
    - Vídeo-aula 46: *http://youtu.be/zxwCSxbntKA*