Algoritmos e Estruturas de Dados: Pesquisa Binária

Rômulo Silva de Oliveira
Departamento de Automação e Sistemas – DAS – UFSC

romulo@das.ufsc.br
http://www.das.ufsc.br/~romulo
Maio/2011

Referências

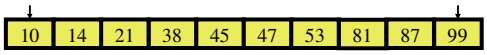
- Mastering Algorithms with C
  Kyle Loudon
  O´Reilly, 1999
- Livros de algoritmos e estruturas de dados em geral

Pesquisa Binária – Introdução 1/9

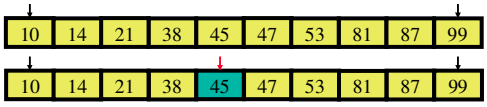
- Pesquisa binária inicia com um conjunto de dados que está ordenado
- Para iniciar a pesquisa, inspeciona-se o elemento do meio
- Se o elemento do meio é maior do que o buscado
  - Repete a pesquisa na metade inferior
- Se o elemento do meio é menor do que o buscado
  - Repete a pesquisa na metade superior
- Repete até que
  - Localizar o elemento buscado
  - Não ser mais possível dividir o conjunto

- Funciona com qualquer tipo de dado ordenado
- Simples
- Ineficiente se muitas inserções e remoções
  - Precisa manter ordenado
  - Neste caso melhor usar uma árvore ou hash
- Também todos os elementos precisam estar contíguos na memória
- Muito bom quando os dados são estáticos (read-only)

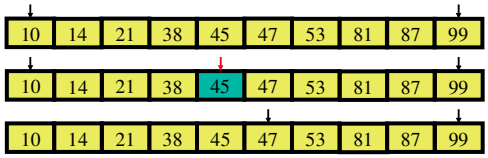
- Repetidamente divide o conjunto e inspeciona o elemento do meio
  - Busca 47



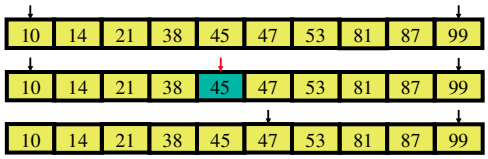
- Repetidamente divide o conjunto e inspeciona o elemento do meio
  - Busca 47



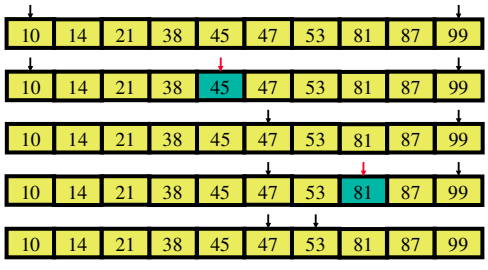
- Repetidamente divide o conjunto e inspeciona o elemento do meio
  - Busca 47



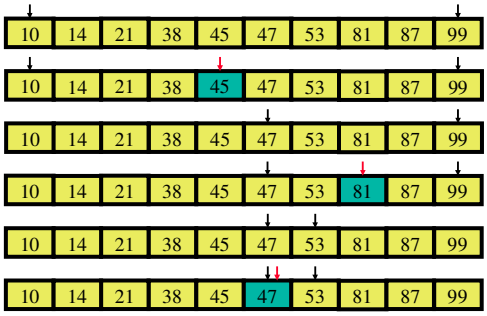
- Repetidamente divide o conjunto e inspeciona o elemento do meio
  - Busca 47



- Repetidamente divide o conjunto e inspeciona o elemento do meio
  - Busca 47



- Repetidamente divide o conjunto e inspeciona o elemento do meio
  - Busca 47



Pesquisa Binária – Interface 1/1

- ***bisearch***
  - int bisearch(void **sorted*, void **target*, int *size*, int *esize*,
    int (**compare*)(const void **key1*, const void **key2*);
- **Retorno**
  - Indice do elemento alvo quando encontrado, ou -1 caso contr ário.
- **Descrição**
  - Usa pesquisa binária para localizar *target* em *sorted*, um array ordenado de elementos. O número de elementos em *sorted* é especificado por *size*. O tamanho de cada elemento é especificado por *esize*. O pointer para função *compare* especifica uma função definida por usuário para comparar elementos. Esta função deve retornar 1 se *key1* > *key2*, 0 se *key1* = *key2*, e -1 se *key1* < *key2*.

Pesquisa Binária – Implementação 1/2

```
#include <stdlib.h>
#include <string.h>

#include "search.h"

int bisearch(void *sorted, const void *target, int size, int esize,
             int (*compare)(const void *key1, const void *key2)) {

    int  left,  middle,  right;

    //  Continue searching until the left and right indices cross.

    left = 0;
    right = size - 1;
```

Pesquisa Binária – Implementação 2/2

```
    while (left <= right) {

        middle = (left + right) / 2;

        switch (compare(((char *)sorted + (esize * middle)), target)) {

                case -1:    // Prepare to search to the right of the middle index
                            left = middle + 1;
                            break;

                case 1:     // Prepare to search to the left of the middle index
                            right = middle - 1;
                            break;

                case 0:     // Return the exact index where the data has been found
                            return middle;
        }
    }
    return -1;
}
```


Pesquisa Binária – Sumário

- Simples
  - Baixo overhead
- Eficiente
- Precisa manter os dados ordenados
- Excelente para tabelas read-only