

## Programação 2

### **6 – Busca em Vetores**

# Tópicos Principais

- Busca em vetor
  - Busca linear
  - Busca binária

# Busca em Vetor

- Busca em vetor:
  - entrada: vetor `vet` com `n` elementos  
elemento `elem`
  - saída: `n` se o elemento `elem` ocorre em `vet[n]`  
`-1` se o elemento não se encontra no vetor

# Busca Linear em Vetor

- percorra o vetor **vet**, elemento a elemento, verificando se **elem** é igual a um dos elementos de **vet**

```
int busca (int n, int* vet, int elem) {  
    int i;  
  
    for (i=0; i<n; i++) {  
        if (elem == vet[i]) {  
            return i; /* encontrou */  
        }  
  
        /* não encontrou */  
        return -1;  
    }  
}
```

# Análise da Busca Linear em Vetor

- pior caso:
  - n comparações, onde n representa o número de elementos do vetor
    - desempenho computacional varia linearmente em relação ao tamanho do problema (algoritmo de busca *linear*)
  - complexidade:  $O(n)$
- caso médio:
  - $n/2$  comparações
    - desempenho computacional continua variando linearmente em relação ao tamanho do problema
  - complexidade:  $O(n)$

# Busca Linear em Vetor Ordenado

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|



6

```
int busca_ord (int n, int* vet, int elem) {  
    int i;  
    for (i=0; i<n; i++) {  
        if (elem == vet[i])  
            return i; /* encontrou */  
        else if (elem < vet[i])  
            return -1; /* interrompe busca */  
    }  
  
    /* não encontrou */  
    return -1;  
}
```

# Análise da Busca Linear em Vetor Ordenado

- caso o elemento procurado não pertença ao vetor, a busca linear com vetor ordenado apresenta um desempenho ligeiramente superior à busca linear
- pior caso:
  - algoritmo continua sendo linear
  - complexidade:  $O(n)$

# Busca Binária em Vetor Ordenado

- entrada:       vetor *vet* com *n* elementos, ordenado  
                  elemento *elem*
- saída:            n            se o elemento *elem* ocorre em *vet[n]*  
                  -1           se o elemento não se encontra no vetor
- procedimento:
  - compare *elem* com o elemento do meio de *vet*
  - se *elem* for menor, pesquise na primeira metade do vetor
  - se *elem* for maior, pesquise na segunda parte do vetor
  - se for igual, retorne a posição
  - continue o procedimento, subdividindo a parte de interesse, até encontrar o elemento ou chegar a uma parte do vetor com tamanho 0



```

int busca_bin (int n, int* vet, int elem)
{
    /* no início consideramos todo o vetor */
    int ini = 0;
    int fim = n-1;

    /* enquanto a parte restante for maior que zero */
    while (ini <= fim) {
        int meio = (ini + fim) / 2;
        if (elem < vet[meio]) {
            fim = meio - 1; /* ajusta posição final */
        }
        else if (elem > vet[meio]) {
            ini = meio + 1; /* ajusta posição inicial */
        }
        else
            return meio; /* elemento encontrado */
    }

    /* não encontrou: restou parte de tamanho zero */
    return -1;
}

```

# Análise da Busca em Vetor Ordenado

pior caso:  $O(\log n)$

- elemento não ocorre no vetor
- 2 comparações são realizadas a cada ciclo
- a cada repetição, a parte considerada na busca é dividida à metade
- logo, no pior caso, são necessárias  $\log n$  repetições

| <b>Repetição</b> | <b>Tamanho do problema</b> |
|------------------|----------------------------|
| <b>1</b>         | <b>n</b>                   |
| <b>2</b>         | <b>n/2</b>                 |
| <b>3</b>         | <b>n/4</b>                 |
| <b>...</b>       | <b>...</b>                 |
| <b>log n</b>     | <b>1</b>                   |

# Diferença entre $n$ e $\log(n)$

| tamanho        | $O(n)$   | $O(\log(n))$ |
|----------------|----------|--------------|
| 10             | 10 seg   | 3            |
| 60             | 1 min    | 6            |
| 600            | 10 min   | 9            |
| 3 600          | 1 h      | 12           |
| 86 400         | 1 dia    | 16           |
| 2 592 000      | 1 mês    | 21           |
| 946 080 000    | 1 ano    | 30           |
| 94 608 000 000 | 100 anos | 36           |

# Busca Binária em Vetor Recursiva

- dois casos a tratar:
  - busca deve continuar na primeira metade do vetor:
    - chamada recursiva com parâmetros:
      - o número de elementos da primeira parte restante
      - o mesmo ponteiro para o primeiro elemento (pois a primeira parte tem o mesmo primeiro elemento do que o vetor como um todo)
  - busca deve continuar apenas na segunda parte do vetor:
    - chamada recursiva com parâmetros:
      - número de elementos restantes
      - ponteiro para o primeiro elemento dessa segunda parte
    - valor retornado deve ser corrigido

```

int busca_bin_rec (int n, int* vet, int elem)
{
    /* testa condição de contorno: parte com tamanho zero */
    if (n <= 0) {
        return -1;
    }
    else {
        int meio = n/2;
        if (elem < vet[meio]) {
            return busca_bin_rec(meio,vet,elem);
        }
        else if (elem > vet[meio]) {
            int r = busca_bin_rec(n-1-meio, &vet[meio+1],elem);
            if (r==-1)
                return -1;
            else
                return (meio+1+r); /* correção da origem */
        }
        else /* elem==vet[meio] */
            return meio; /* elemento encontrado */
    }
}

```

# Busca binária em Vetor

- prós:
  - dados armazenados em vetor, de forma ordenada
  - bom desempenho computacional para pesquisa
- contra:
  - inadequado quando inserções e remoções são freqüentes
    - exige re-arrumar o vetor para abrir espaço uma inserção
    - exige re-arrumar o vetor após uma remoção

# Atenção

- Busca binária – O que pode variar?
  - Critério de ordenação (primário e desempates)
  - A informação retornada:
    - O índice do elemento encontrado ou -1;
    - O valor de um campo específico;
    - O ponteiro para o elemento encontrado;
    - 1 se encontrou, ou 0, caso contrário;
    - Outras...
  - Repetição ou não de valores (chaves)

# Exercício 1

- Considere um tipo que representa um funcionário de uma empresa, definido pela estrutura a seguir:

```
struct funcionario {  
    char nome[81];      /* nome do funcionario */  
    float valor_hora;    /* valor da hora de trabalho em Reais */  
    int horas_mes;      /* horas trabalhadas em um mes */  
};  
  
typedef struct funcionario Funcionario;
```

- Escreva uma função que faça uma *busca binária* em um vetor de ponteiros para o tipo Funcionario, cujos elementos estão em ordem alfabética dos nomes dos funcionários. Essa função deve receber como parâmetros o número de funcionários, o vetor e o nome do funcionário que se deseja buscar, e deve ter como valor de retorno um ponteiro para o registro do funcionário procurado. Se não houver um funcionário com o nome procurado, a função deve retornar NULL. Sua função deve ter o seguinte cabeçalho:

```
Funcionario* busca (int n, Funcionario** v, char* nome);
```



```

Funcionario* busca (int n, Funcionario** v, char* nome) {
    /* no início consideramos todo o vetor */
    int ini = 0;
    int fim = n-1;

    /* enquanto a parte restante for maior que zero */
    while (ini <= fim) {
        int meio = (ini + fim) / 2;
        switch (strcmp(nome, v[meio]->nome)) {
            case -1:
                fim = meio - 1; /* ajusta posição final */
                break;
            case 1:
                ini = meio + 1; /* ajusta posição inicial */
                break;
            case 0:
                return v[meio]; /* elemento encontrado */
        }
    }

    /* não encontrou: restou parte de tamanho zero */
    return NULL;
}

```

# Exercício 2

- Considere um tipo que representa as licenças dos funcionários de uma empresa, definido pela estrutura a seguir:

```
struct licenca {  
    char nome[51];    /* nome do funcionario    */  
    Data inicio;      /* data de inicio da licenca */  
    Data final;       /* data de final da licenca  */  
};  
typedef struct licenca Licenca;
```

- Os campos *inicio* e *final* são do tipo *Data*, descrito a seguir:

```
struct data {  
    int dia, mes, ano;  
};  
typedef struct data Data;
```

- Escreva uma função que faça uma *busca binária* em um vetor de ponteiros para o tipo *Licenca*, cujos elementos estão em ordem cronológica, de acordo com a data de início das licenças, com desempate pela ordem alfabética de acordo com o nome dos funcionários. Se existir mais de uma licença com início na data procurada, a função deve retornar o índice da primeira delas. Se não houver uma licença com a data procurada, a função deve retornar -1. Sua função deve ter o seguinte cabeçalho:

```
int busca (Licenca** v, int n, Data d);
```

## Exercício 2

```
Int dtacmp(Data d1, Data d2)
{
    if(d1.ano<d2.ano) return -1;
    if(d1.ano>d2.ano) return 1;
    if(d1.mes<d2.mes) return -1;
    if(d1.mes>d2.mes) return 1;
    if(d1.dia<d2.dia) return -1;
    if(d1.dia>d2.dia) return 1;
    return 0;
}
```

```
int busca (Licenca** v, int n, Data d) {
    int i=0, f=n-1, m;
    while (i<=f) {
        m = (i+f)/2;
        if (dtacmp(d, v[m]->inicio)==-1)
            f = m-1; /* ajusta posição final*/
        else if (dtacmp(d, v[m]->inicio)==1)
            i = m+1; /* ajusta posição inicial */
        else {
            while((m>0)&&dtacmp(d, v[m-1]->inicio)==0)
                m--; /*elemento encontrado, procura primeira ocorrencia */
            return m;
        }
    }
    return -1; /* não encontrou */
}
```

# Resumo

- Busca linear em vetor:
  - percorra o vetor, elemento a elemento, verificando se o elemento de interesse é igual a um dos elementos do vetor
- Busca binária:
  - compare *elem* com o elemento do meio de vet
  - se *elem* for menor, pesquise na primeira metade do vetor
  - se *elem* for maior, pesquise na segunda parte do vetor
  - se for igual, retorne a posição
  - continue o procedimento, subdividindo a parte de interesse, até encontrar o elemento ou chegar a uma parte do vetor com tamanho 0

# Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel,  
*Introdução a Estruturas de Dados*, Editora Campus  
(2004)

Capítulo 17 – Busca