

# MC-102 — Aula 18

## Busca Sequencial e Binária

Instituto de Computação – Unicamp

22 de Maio de 2012

# Roteiro

- 1 O Problema da Busca
- 2 Busca Sequencial
- 3 Busca Binária
- 4 Questões sobre eficiência
- 5 Exercícios

# O Problema da Busca

- Vamos estudar alguns algoritmos para o seguinte problema:

Temos uma coleção de elementos, onde cada elemento possui um identificador/chave único, e recebemos uma chave para busca. Devemos encontrar o elemento da coleção que possui a mesma chave ou identificar que não existe nenhum elemento com a chave dada.

- Nos nossos exemplos usaremos um vetor de inteiros como a coleção.
  - ▶ O valor da chave será o próprio valor de cada número.
- Apesar de usarmos inteiros, os algoritmos servem para buscar elementos em qualquer coleção de elementos que possuam chaves que possam ser comparadas.

# O Problema da Busca

- O problema da busca é um dos mais básicos em Computação e também possui diversas aplicações.
  - ▶ Suponha que temos um cadastro com registros de motoristas.
  - ▶ Um vetor de registros é usado para armazenar as informações dos motoristas. Podemos usar como chave o número da carteira de motorista, ou o RG, ou o CPF.
- Veremos algoritmos simples para realizar a busca assumindo que dados estão em um vetor.
- Em cursos mais avançados são estudados outros algoritmos e estruturas (que não um vetor) para armazenar e buscar elementos.

# O Problema da Busca

- Nos nossos exemplos vamos criar a função:
  - ▶ **int busca(int vet[], int tam, int chave)**, que recebe um vetor com um determinado tamanho, e uma chave para busca.
  - ▶ A função deve retornar o índice do vetor que contém a chave ou -1 caso a chave não esteja no vetor.

# O Problema da Busca

chave = 45      tam = 8

vet	20	5	15	24	67	45	1	76		
	0	1	2	3	4	5	6	7	8	9

chave = 100      tam = 8

vet	20	5	15	24	67	45	1	76		
	0	1	2	3	4	5	6	7	8	9

No exemplo mais acima, a função deve retornar 5, enquanto no exemplo mais abaixo a função deve retornar -1.

# Busca Sequencial

- A busca sequencial é o algoritmo mais simples de busca:
  - ▶ Percorra todo o vetor comparando a chave com o valor de cada posição.
  - ▶ Se for igual para alguma posição, então devolva esta posição.
  - ▶ Se o vetor todo foi percorrido então devolva -1.

# Busca Sequencial

```
int buscaSequencial(int vet[], int tam, int chave){
    int i;
    for(i=0; i<tam; i++){
        if(vet[i] == chave)
            return i;
    }
    return -1;
}
```



# Busca Sequencial

```
#include <stdio.h>

int buscaSequencial(int vet[], int tam, int chave);

int main(){
    int pos, vet[] = {20, 5, 15, 24, 67, 45, 1, 76, -1, -1}; //-1 indica
                                                    //posição não usada

    pos = buscaSequencial(vet, 8, 45);
    if(pos != -1)
        printf("A posicao da chave 45 no vetor é: %d\n", pos);
    else
        printf("A chave 45 não está no vetor! \n");

    pos = buscaSequencial(vet, 8, 100);
    if(pos != -1)
        printf("A posicao da chave 100 no vetor é: %d\n", pos);
    else
        printf("A chave 100 não está no vetor! \n");
}

int buscaSequencial(int vet[], int tam, int chave){
    int i;
    for(i=0; i<tam; i++){
        if(vet[i] == chave)
            return i;
    }
    return -1;
}
```

# Busca Binária

- A busca binária é um algoritmo um pouco mais sofisticado.
- É mais eficiente, mas requer que o vetor esteja ordenado pelos valores da chave de busca.
- A idéia do algoritmo é a seguinte (assuma que o vetor está ordenado):
  - ▶ Verifique se a chave de busca é igual ao valor da posição do meio do vetor.
  - ▶ Caso seja igual, devolva esta posição.
  - ▶ Caso o valor desta posição seja maior, então repita o processo mas considere que o vetor tem metade do tamanho, indo até posição anterior a do meio.
  - ▶ Caso o valor desta posição seja menor, então repita o processo mas considere que o vetor tem metade do tamanho e inicia na posição seguinte a do meio.

# Busca Binária

```
//vetor começa em posIni e termina em posFim  
posIni = 0  
posFim = tam-1
```

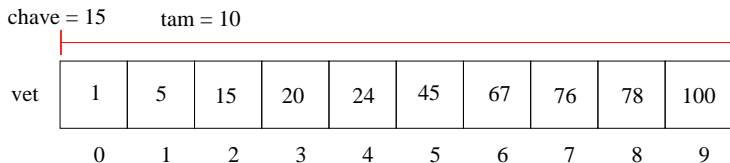
```
Repita enquanto tamanho do vetor considerado for >= 1  
    posMeio = (posIni + posFim)/2
```

```
    Se vet[posMeio] == chave Então  
        devolva posMeio
```

```
    Se vet[posMeio] > chave Então  
        posFim = posMeio - 1
```

```
    Se vet[posMeio] < chave Então  
        posIni = posMeio + 1
```

# Busca Binária



posIni = 0

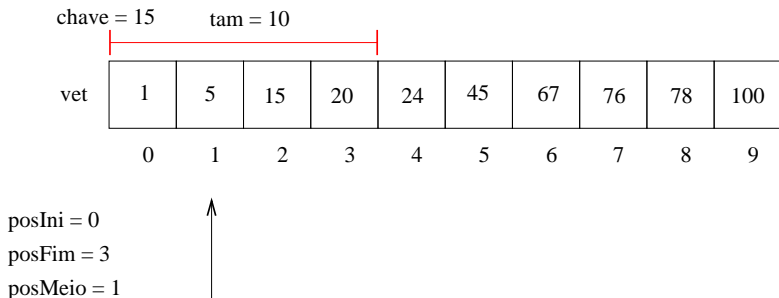
posFim = 9

posMeio = 4



Como o valor da posição do meio é maior que a chave, atualizamos **posFim** do vetor considerado.

# Busca Binária



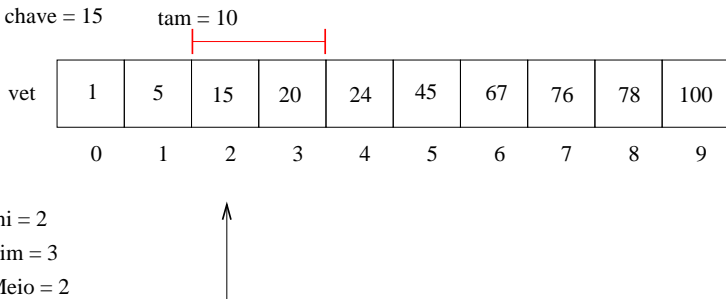
Como o valor da posição do meio é menor que a chave, atualizamos **posIni** do vetor considerado.

# Busca Binária

chave = 15

tam = 10

vet	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9



posIni = 2

posFim = 3

posMeio = 2

Finalmente encontramos a chave e podemos devolver sua posição 2.

# Busca Binária

```
int buscaBinaria(int vet[], int tam, int chave){
    int posIni=0, posFim=tam-1, posMeio;

    while(posIni <= posFim){ //enquanto o vetor tiver pelo menos 1 elemento
        posMeio = (posIni+posFim)/2;

        if(vet[posMeio] == chave)
            return posMeio;
        else if(vet[posMeio] > chave)
            posFim = posMeio - 1;
        else
            posIni = posMeio + 1;
    }

    return -1;
}
```

# Busca Binária

```
int main(){
    int vet[] = {20, 5, 15, 24, 67, 45, 1, 76, 78, 100};
    int pos, i;

    //antes de usar a busca devemos ordenar o vetor
    insertionSort(vet,10);
    printf("Vetor Ordenado:");
    for(i =0; i<10; i++){
        printf("%d, ", vet[i]);
    }
    printf("\n");
    pos = buscaBinaria(vet, 10, 15);
    if(pos != -1)
        printf("A posicao da chave 15 no vetor é: %d\n", pos);
    else
        printf("A chave 15 não está no vetor! \n");

}
```



# Eficiência dos Algoritmos

Podemos medir a eficiência de qualquer algoritmo analisando a quantidade de recursos (tempo, memória, banda de rede, etc.) que o algoritmo usa para resolver o problema para o qual foi proposto.

- A forma mais simples é medir a eficiência em relação ao tempo. Para isso, analisamos quantas instruções um algoritmo usa para resolver o problema.
- Podemos fazer uma análise simplificada dos algoritmos de busca analisando a quantidade de vezes que os algoritmos **acessam** uma posição do vetor.

# Eficiência dos Algoritmos

No caso da busca sequencial existem três possibilidades:

- Na melhor das hipóteses a chave de busca estará na posição 0. Portanto teremos um único acesso em **vet[0]**.
- Na pior das hipóteses, a chave é o último elemento ou não pertence ao vetor, e portanto acessaremos todas as *tam* posições do vetor.
- Assim se uma chave qualquer pode ser requisitada com a mesma probabilidade, então o número de acessos será

$$(tam + 1)/2$$

na média.

# Eficiência dos Algoritmos

No caso da busca binária temos as três possibilidades:

- Na melhor das hipóteses a chave de busca estará na posição do meio. Portanto teremos um único acesso.
- Na pior das hipóteses, teremos  $(\log \text{tam})$  acessos.
  - ▶ Para ver isso note que a cada verificação de uma posição do vetor, o tamanho do vetor considerado é dividido pela metade. No pior caso repetiremos a busca até o vetor considerado ter tamanho 1.
- Assim se uma chave qualquer pode ser requisitada com a mesma probabilidade, então o número de acessos será

$$(\log \text{tam}) - 1$$

na média.

# Eficiência dos Algoritmos: exemplo

Para se ter uma idéia da diferença de eficiência dos dois, considere que temos um cadastro com  $10^6$  (um milhão) de itens.

- Com a busca sequencial, a procura de um item qualquer gastará na média

$$(10^6 + 1)/2 \approx 500000 \text{ acessos.}$$

- Com a busca binária teremos

$$(\log 10^6) - 1 \approx 20 \text{ acessos.}$$

# Eficiência dos Algoritmos

Mas uma ressalva deve ser feita: Para utilizar a busca binária, o vetor precisa estar ordenado!

- Se você tiver um cadastro onde vários itens são removidos e inseridos com frequência, e a busca de ser feita intercalado com estas operações, então a busca binária pode não ser a melhor opção, já que você precisará ficar mantendo o vetor ordenado.
- Caso o número de buscas feitos seja muito maior quando comparado com outras operações, então a busca binária é uma boa opção.

# Exercícios

- Refaça as funções de busca sequencial e busca binária assumindo que o vetor possui chaves que podem aparecer repetidas. Neste caso, você deve retornar em um outro vetor todas as posições onde a chave foi encontrada.

Protótipo: **void busca(int vet[], int tam, int chave, int posicoes[], int \*n)**

- Você deve devolver em **posicoes[]** as posições de **vet** que possuem a **chave**, e devolver em **\*n** o número de ocorrências da chave.
  - ▶ **OBS:** Na chamada desta função, o vetor **posições** deve ter espaço suficiente (**tam**) para guardar todas as possíveis ocorrências.

# Exercícios

- Implemente uma pequena aplicação para cadastro de motoristas, com funções de incluir/excluir motorista, e uma opção para fazer busca pelo número da carteira de motorista.