MC-102 — Aula 10
Ordenação – BubbleSort
e Algoritmos de busca

Instituto de Computação – Unicamp

17 de Março de 2014

Roteiro

Ordenação

- Vamos estudar alguns algoritmos para o seguinte problema:

Dada uma coleção de elementos com uma relação de ordem entre si, devemos gerar uma saída com os elementos ordenados.

- Nos nossos exemplos usaremos um vetor de inteiros como sendo essa coleção.
  - É claro que quaisquer inteiros possuem uma relação de ordem entre si.
- Apesar de usarmos inteiros, os algoritmos servem para ordenar qualquer coleção de elementos que possam ser comparados.

Ordenação

- O problema de ordenação é um dos mais básicos em computação.
  - Mas muito provavelmente é um dos problemas com o maior número de aplicações diretas ou indiretas (como parte da solução para um problema maior).
- Exemplos de aplicações diretas:
  - Criação de *rankings*, definir preferências em atendimentos por prioridade, criação de listas etc.
- Exemplos de aplicações indiretas:
  - Otimizar sistemas de busca, manutenção de estruturas de bancos de dados etc.

Bubble-Sort

- Seja **vet** um vetor contendo números inteiros.
- Devemos deixar **vet** em ordem crescente.
- O algoritmo faz algumas iterações repetindo o seguinte:
  - Compare $vet[0]$ com $vet[1]$ e troque-os se $vet[0] > vet[1]$.
  - Compare $vet[1]$ com $vet[2]$ e troque-os se $vet[1] > vet[2]$.
  - .....
  - Compare $vet[tam-2]$ com $vet[tam-1]$ e troque-os se $vet[tam-2] > vet[tam-1]$.

  Após uma iteração repetindo estes passos o que podemos garantir???
  - O maior elemento estará na posição correta!!!

Bubble-Sort

- Seja **vet** um vetor contendo números inteiros.
- Devemos deixar **vet** em ordem crescente.
- O algoritmo faz algumas iterações repetindo o seguinte:
  - Compare $vet[0]$ com $vet[1]$ e troque-os se $vet[0] > vet[1]$.
  - Compare $vet[1]$ com $vet[2]$ e troque-os se $vet[1] > vet[2]$.
  - .....
  - Compare $vet[tam-2]$ com $vet[tam-1]$ e troque-os se $vet[tam-2] > vet[tam-1]$.

  Após uma iteração repetindo estes passos o que podemos garantir???
  - O maior elemento estará na posição correta!!!

Bubble-Sort

- Após uma iteração de trocas, o maior elemento estará na última posição.
- Após outra iteração de trocas, o segundo maior elemento estará na posição correta.
- E assim sucessivamente.
- Quantas iterações destas trocas precisamos para deixar o vetor ordenado?

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:

$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:
$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:
$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:
$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:
$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:
$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:
$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:
$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

- O código abaixo realiza as trocas de uma iteração.
- São comparados e trocados, os elementos das posições: 0 e 1; 1 e 2; ...; $i-1$ e i.
- Assumimos que de $(i+1)$ até $(tam-1)$, o vetor já tem os maiores elementos ordenados.

```
for(j=0; j < i; j++)
   if( vet[j] > vet[j+1] ){
       aux = vet[j];
       vet[j] = vet[j+1];
       vet[j+1] = aux;
   }
```

Bubble-Sort

- O código abaixo realiza as trocas de uma iteração.
- São comparados e trocados, os elementos das posições: 0 e 1; 1 e 2; ...; $i-1$ e i.
- Assumimos que de $(i+1)$ até $(tam-1)$, o vetor já tem os maiores elementos ordenados.

```
for(j=0; j < i; j++)
   if( vet[j] > vet[j+1] ){
       aux = vet[j];
       vet[j] = vet[j+1];
       vet[j+1] = aux;
   }
```

Bubble-Sort

```
int vetor[10]={14,7,8,34,56,4,0,9,-8,100};
int i,j, aux, tam=10;

 for(i=tam-1; i>0; i--){
    for(j=0; j < i; j++)  //Faz trocas até posição i
       if( vet[j] > vet[j+1] ){
           aux = vet[j];
           vet[j] = vet[j+1];
           vet[j+1] = aux;
       }
 }
```

Bubble-Sort

- Notem que as trocas na primeira iteração ocorrem até a última posição.
- Na segunda iteração ocorrem até a penúltima posição.
- E assim sucessivamente.
- Por que?

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.

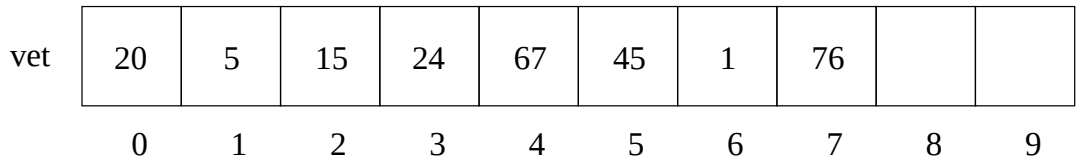
- Nos nossos exemplos usaremos um vetor de inteiros como a coleção.
  - O valor da chave será o próprio valor de cada número.
- Apesar de usarmos inteiros, os algoritmos servem para buscar elementos em qualquer coleção de elementos que possuam chaves que possam ser comparadas.

O Problema da Busca

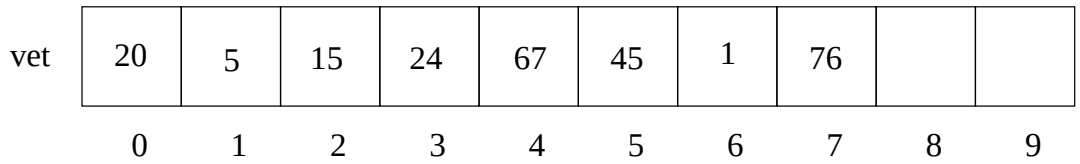
- O problema da busca é um dos mais básicos em Computação e também possui diversas aplicações.
  - Suponha que temos um cadastro com registros de motoristas.
  - Um vetor de registros é usado para armazenar as informações dos motoristas. Podemos usar como chave o número da carteira de motorista, ou o 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

chave = 45 tam = 8



chave = 100 tam = 8



No primeiro exemplo o algoritmo deve encontrar o elemento na posição 5, enquanto no segundo a deve indicar que não encontrou o elemento.

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 imprima esta posição.
  - Se o vetor todo foi percorrido então imprima que não encontrou o elemento.

Busca Sequencial

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

Busca Sequencial

```
#include <stdio.h>
int main(){
  int i,tam=10,chave, pos, vet[] = {20, 5, 15, 24, 67, 45, 1, 76, -1, -1};

  scanf("%d", &chave);
  pos=-1;
  for(i=0; i<tam; i++){
    if(vet[i] == chave)
       pos = i;
  }

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

}
```

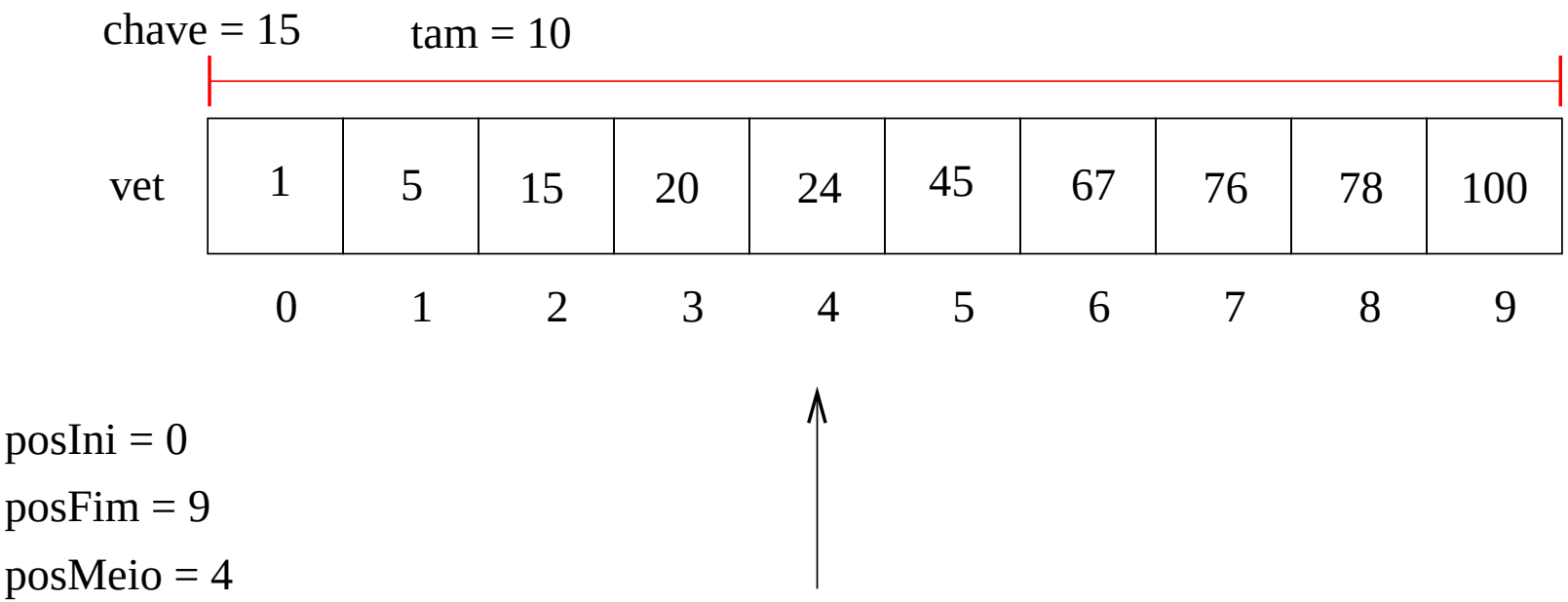
Busca Binária

- 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, imprima 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

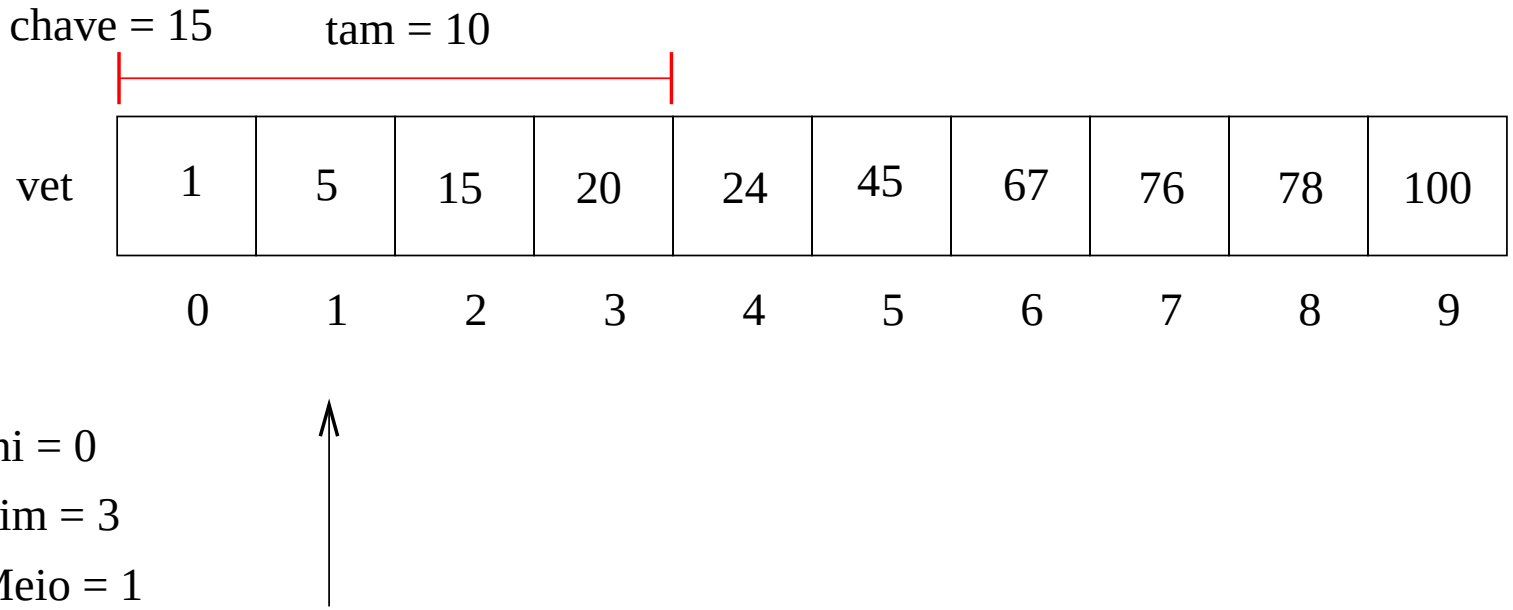
```
//vetor começa em posIni e termina em posFim
posIni = 0
posFim = tam-1

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

        Se vet[posMeio] == chave Então
                posicao=posMeio

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

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

Busca Binária



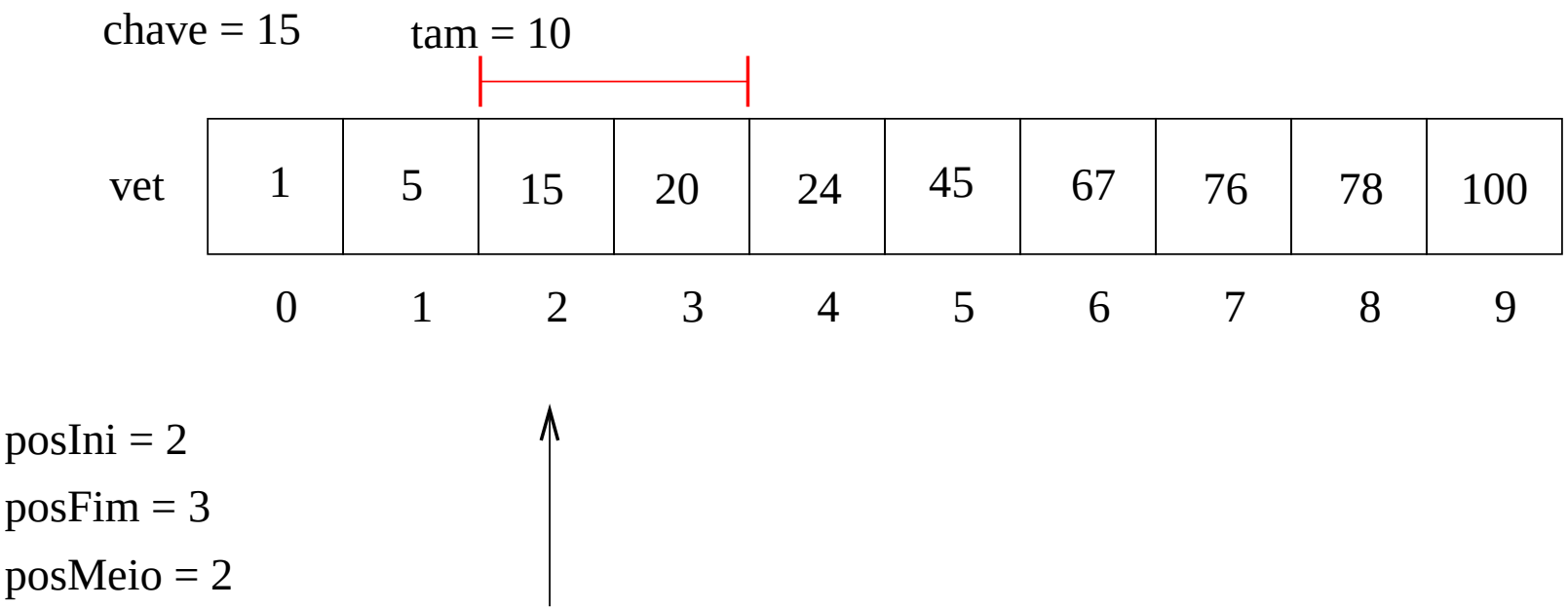
Como o valor da posição do meio é maior que a chave, atualizamos **posFim** do vetor considerado.

Busca Binária



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

Busca Binária



Finalmente encontramos a chave e podemos devolver sua posição 2.

Busca Binária

```
int main(){
  int tam=10,chave, pos, vet[] =  {1, 5, 15, 24, 45, 67, 76, 77, 78, 100};
  int posIni, posFim, posMeio;

  scanf("%d", &chave);
  posIni=0;
  posFim=tam-1;
  pos=-1;
  while(posIni <= posFim){ //enquanto o vetor tiver pelo menos 1 elemento
    posMeio = (posIni+posFim)/2;

    if(vet[posMeio] == chave) {
      pos=posMeio;
      break;
    }
    else if(vet[posMeio] > chave)
      posFim = posMeio - 1;
    else
      posIni = posMeio + 1;
  }
  if(pos != -1)
    printf("A posicao da chave %d no vetor é: %d\n", chave, pos);
  else
    printf("A chave %d não está no vetor! \n", chave);
}
```

Exercício

Altere o algoritmo de ordenação vistos nesta aula para que estes ordenem um vetor de inteiros em ordem decrescente ao invés de ordem crescente.

Exercício

- Refaça os algoritmos de busca sequencial e busca binária assumindo que o vetor possui chaves que podem aparecer repetidas. Neste caso, você deve armazenar em um outro vetor (vetor **posicoes**) todas as posições onde a chave foi encontrada.
- **OBS:** O vetor **posições** deve ter espaço suficiente (tam) para guardar todas as possíveis ocorrências.