Listas Estáticas

SCC0202 - Algoritmos e Estruturas de Dados I

Prof. Fernando V. Paulovich
**Baseado no material do Prof. Gustavo Batista*
http://www.icmc.usp.br/~paulovic
paulovic@icmc.usp.br

Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)

4 de outubro de 2017

Sumário

Sumário

Listas

Definição

- Listas são estruturas flexíveis que admitem as operações de **inserção**, **remoção** e **recuperação** de itens

Listas

Definição

- Listas são estruturas flexíveis que admitem as operações de **inserção**, **remoção** e **recuperação** de itens
- É uma sequencia de zero ou mais itens $x_1, x_2, \ldots, x_n$ na qual x_i é de um determinado tipo e n representa o tamanho da lista

Listas

Definição

- Listas são estruturas flexíveis que admitem as operações de **inserção**, **remoção** e **recuperação** de itens
- É uma sequencia de zero ou mais itens $x_1, x_2, \ldots, x_n$ na qual x_i é de um determinado tipo e n representa o tamanho da lista
- De forma geral as **posições relativas** dos itens na lista não tem importância ou significado para os dados

Listas

Definição

- Listas são estruturas flexíveis que admitem as operações de **inserção**, **remoção** e **recuperação** de itens
- É uma sequencia de zero ou mais itens $x_1, x_2, \ldots, x_n$ na qual x_i é de um determinado tipo e n representa o tamanho da lista
- De forma geral as **posições relativas** dos itens na lista não tem importância ou significado para os dados
  - Ordem não importa

Listas

Aplicação

- Diversos tipos de aplicações requerem uma lista
  - Lista telefônica
  - Lista de tarefas
  - Gerência de memória
  - Simulação
  - Compiladores, etc.

TAD Listas

Tipo Abstrato de Dados

- Vamos definir um TAD com as principais operações sobre uma lista
  - Para simplificar vamos definir apenas as operações principais, posteriormente, outras operações podem ser definidas

TAD Listas

Principais operações

- Criar lista
- Apagar lista
- Inserir item
- Remover item (dado uma chave)
- Recuperar item (dado uma chave)
- Contar número de itens
- Verificar se a lista está vazia
- Verificar se a lista está cheia
- Imprimir lista

TAD Listas I

Criar lista

- Pré-condição: existir espaço na memória
- Pós-condição: inicia a estrutura de dados

Limpar lista

- Pré-condição: nenhuma
- Pós-condição: remove a estrutura de dados da memória

TAD Listas II

Inserir item

- Pré-condição: a lista não está cheia
- Pós-condição: insere um item na lista, retorna **true** se a operação foi executada com sucesso, **false** caso contrário

Remover item (dado uma chave)

- Pré-condição: nenhuma
- Pós-condição: remove um determinado item da lista dado uma chave, retorna **true** se a operação foi executada com sucesso, **false** caso contrário

TAD Listas III

Recuperar item (dado uma chave)

- Pré-condição: nenhuma
- Pós-condição: recupera o item dado uma chave, retorna o item se a operação foi executada com sucesso, **null** caso contrário

Contar número de itens

- Pré-condição: nenhuma
- Pós-condição: retorna o número de itens na lista

TAD Listas IV

Verificar se a lista está vazia

- Pré-condição: nenhuma
- Pós-condição: retorna **true** se a lista estiver vazia e **false** caso-contrário

Verificar se a lista está cheia

- Pré-condição: nenhuma
- Pós-condição: retorna **true** se a lista estiver cheia e **false** caso-contrário

TAD Listas V

Imprimir lista

- Pré-condição: nenhuma
- Pós-condição: imprime na tela os itens da lista

TAD Listas

Como implementar o TAD Listas?

Basicamente existem duas formas

- Estática (utilizando vetores)
- Dinâmica (utilizando listas ligadas)

Sumário

Listas - Implementação Estática

Características

- Os itens da lista são armazenados em posições contíguas de memória

Listas - Implementação Estática

Características

- Os itens da lista são armazenados em posições contíguas de memória
- A lista pode ser percorrida em qualquer direção

Listas - Implementação Estática

Características

- Os itens da lista são armazenados em posições contíguas de memória
- A lista pode ser percorrida em qualquer direção
- A inserção de um novo item pode ser realizada com custo constante

Listas - Implementação Estática

```
1  #ifndef ITEM_H
2  #define ITEM_H
3
4  //representa o item armazenado
5  typedef struct {
6    int chave;
7    int valor;
8  } ITEM;
9
10 ITEM *criar_item(int chave, int valor);
11 void apagar_item(ITEM **item);
12 void imprimir_item(ITEM *item);
13
14 #endif
```

Listas - Implementação Estática

```
1  ITEM *criar_item(int valor, int chave) {
2    ITEM *item = (ITEM *)malloc(sizeof(ITEM));
3
4    if(item != NULL) {
5      item->valor = valor;
6      item->chave = chave;
7    }
8
9    return item;
10 }
11
12 void apagar_item(ITEM **item) {
13   if(item != NULL && *item != NULL) {
14     free(*item);
15     *item = NULL;
16   }
17 }
18
19 void imprimir_item(ITEM *item) {
20   if(item != NULL) {
21     printf("%d - %d\n", item->chave, item->valor);
22   }
23 }
```

Listas - Implementação Estática

```
1  #ifndef LISTAESTATICA_H
2  #define LISTAESTATICA_H
3
4  #include "Item.h"
5
6  typedef struct lista_estatica LISTA_ESTATICA;
7
8  LISTA_ESTATICA *criar_lista();
9  void apagar_lista(LISTA_ESTATICA **lista);
10
11 int inserir_item(LISTA_ESTATICA *lista, ITEM *item);
12 int remover_item(LISTA_ESTATICA *lista, int chave);
13 ITEM *recuperar_item(LISTA_ESTATICA *lista, int chave);
14
15 int vazia(LISTA_ESTATICA *lista);
16 int cheia(LISTA_ESTATICA *lista);
17 int tamanho(LISTA_ESTATICA *lista);
18 void imprimir(LISTA_ESTATICA *lista);
19
20 #endif
```

Listas - Implementação Estática

```
1  #define TAM 100
2
3  struct lista_estatica {
4    ITEM *vetor[TAM];
5    int fim;
6  };
7
8  LISTA_ESTATICA *criar_lista() {
9    LISTA_ESTATICA *lista = (LISTA_ESTATICA *) malloc(sizeof (LISTA_ESTATICA));
10
11   if(lista != NULL) {
12     lista->fim = -1;
13   }
14
15   return lista;
16 }
17
18 void apagar_lista(LISTA_ESTATICA **lista) {...}
19
20 int inserir_item(LISTA_ESTATICA *lista, ITEM *item) {...}
21 int remover_item(LISTA_ESTATICA *lista, int chave) {...}
22 ITEM *recuperar_item(LISTA_ESTATICA *lista, int chave) {...}
23
24 int vazia(LISTA_ESTATICA *lista) {...}
25 int cheia(LISTA_ESTATICA *lista) {...}
26 int tamanho(LISTA_ESTATICA *lista) {...}
27 void imprimir(LISTA_ESTATICA *lista) {...}
```

Implementação da Inserção e Remoção

```
1  int inserir_item(LISTA_ESTATICA *lista, ITEM *item) {
2    if(!cheia(lista)) {
3      lista->vetor[++lista->fim] = item;
4      return 1;
5    }
6    return 0;
7  }
8
9  int remover_item(LISTA_ESTATICA *lista, int chave) {
10   int i;
11   if(!vazia(lista)) {
12     for(i=0; i <= lista->fim; i++) {
13       if((lista->vetor[i])->chave == chave) {
14         apagar_item(&(lista->vetor[i]));
15         lista->vetor[i] = lista->vetor[lista->fim];
16         lista->vetor[lista->fim] = NULL;
17         lista->fim--;
18       }
19     }
20     return 1;
21   }
22   return 0;
23 }
```

Listas - Implementação Estática

Exercícios

Crie funções que implementem as seguintes operações:

- Verificar se a lista **L** está ordenada (crescente ou decrescente)
- Fazer uma cópia da Lista **L1** em outra **L2**
- Fazer uma cópia da Lista **L1** em **L2**, eliminando repetidos
- Inverter **L1**, colocando o resultado em **L2**
- Inverter a própria **L1**
- Intercalar **L1** com **L2**, gerando **L3** ordenada (considere **L1** e **L2** ordenadas)
- Eliminar de **L1** todas as ocorrências de um dado item (**L1** está ordenada)

Sumário

Listas Ordenadas

Características

- Uma lista pode ser mantida em **ordem crescente/decrescente** segundo o valor de alguma **chave**

Listas Ordenadas

Características

- Uma lista pode ser mantida em **ordem crescente/decrescente** segundo o valor de alguma **chave**
- Essa ordem **facilita a pesquisa** de itens

Listas Ordenadas

Características

- Uma lista pode ser mantida em **ordem crescente/decrescente** segundo o valor de alguma **chave**
- Essa ordem **facilita a pesquisa** de itens
- Por outro lado, a **inserção e remoção são mais complexas** pois deve manter os itens ordenados

TAD Listas Ordenadas

Tipo Abstrato de Dados

- O TAD listas ordenadas é o mesmo do TAD listas, apenas difere na implementação
- As operações diferentes serão a inserção e remoção de itens

TAD Listas Ordenadas

Tipo Abstrato de Dados

- O TAD listas ordenadas é o mesmo do TAD listas, apenas difere na implementação
- As operações diferentes serão a inserção e remoção de itens

Inserir item

- Pré-condição: a lista não está cheia
- Pós-condição: insere um item em uma posição tal que a lista é mantida ordenada

TAD Listas Ordenadas

Tipo Abstrato de Dados

- O TAD listas ordenadas é o mesmo do TAD listas, apenas difere na implementação
- As operações diferentes serão a inserção e remoção de itens

Inserir item

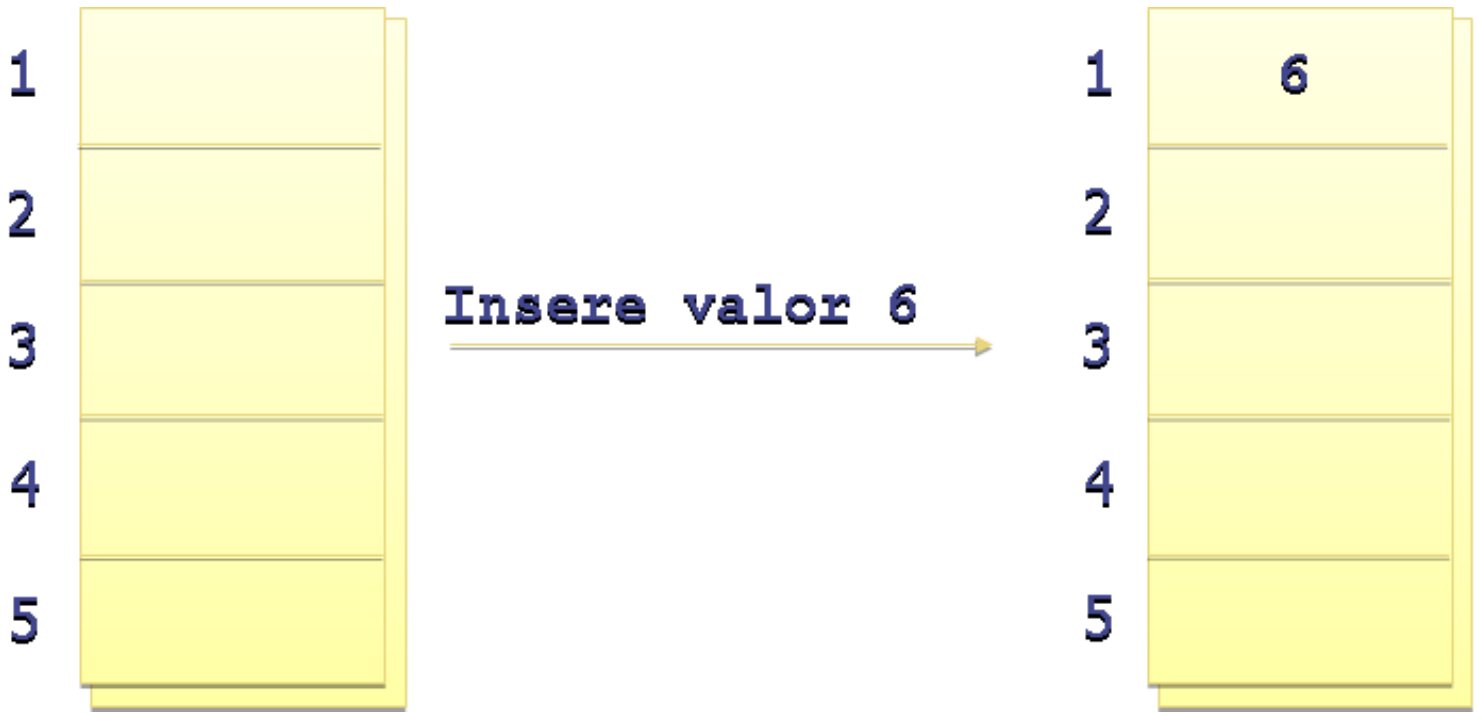
- Pré-condição: a lista não está cheia
- Pós-condição: insere um item em uma posição tal que a lista é mantida ordenada

Remover item (dado uma chave)

- Pré-condição: uma posição válida da lista é informada
- Pós-condição: o item com a chave fornecida é removido da lista, a lista é mantida ordenada

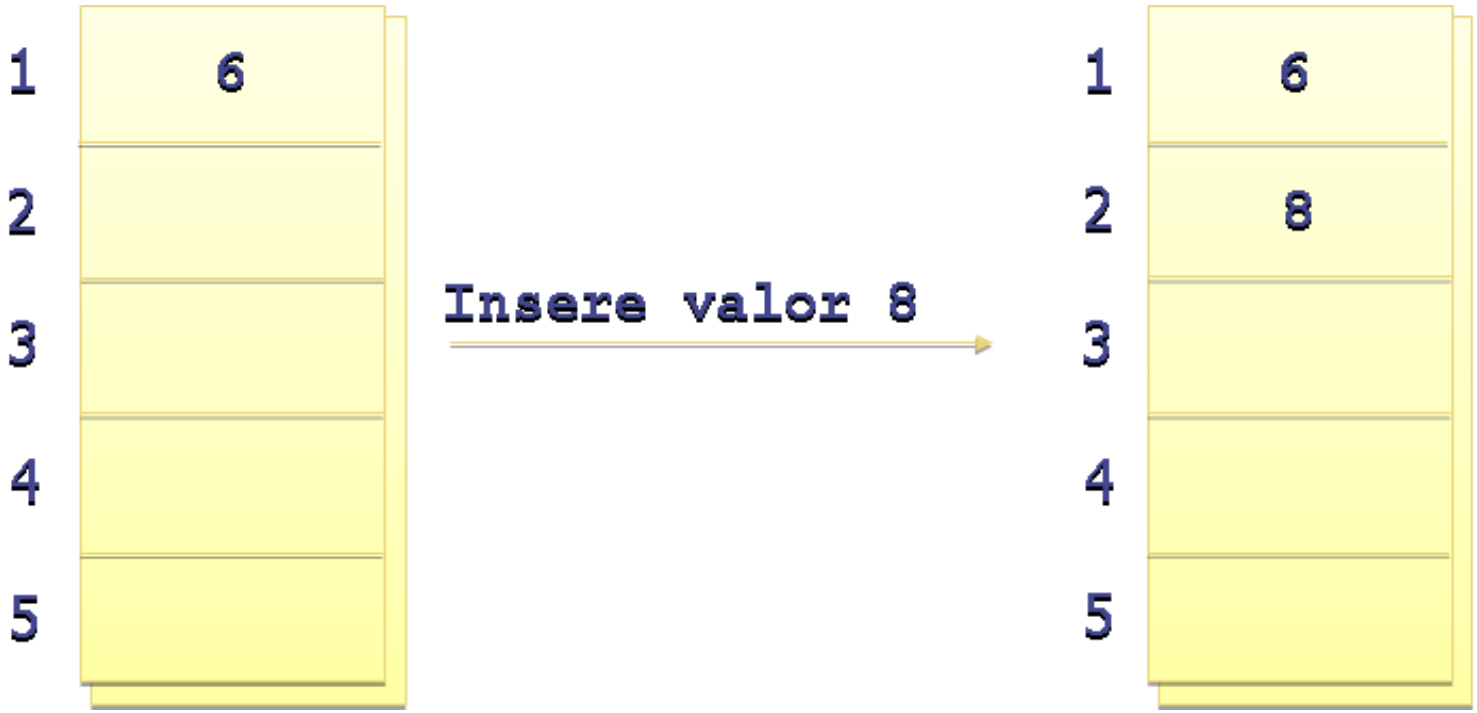
Inserção Ordenada

- Exemplo de inserção ordenada



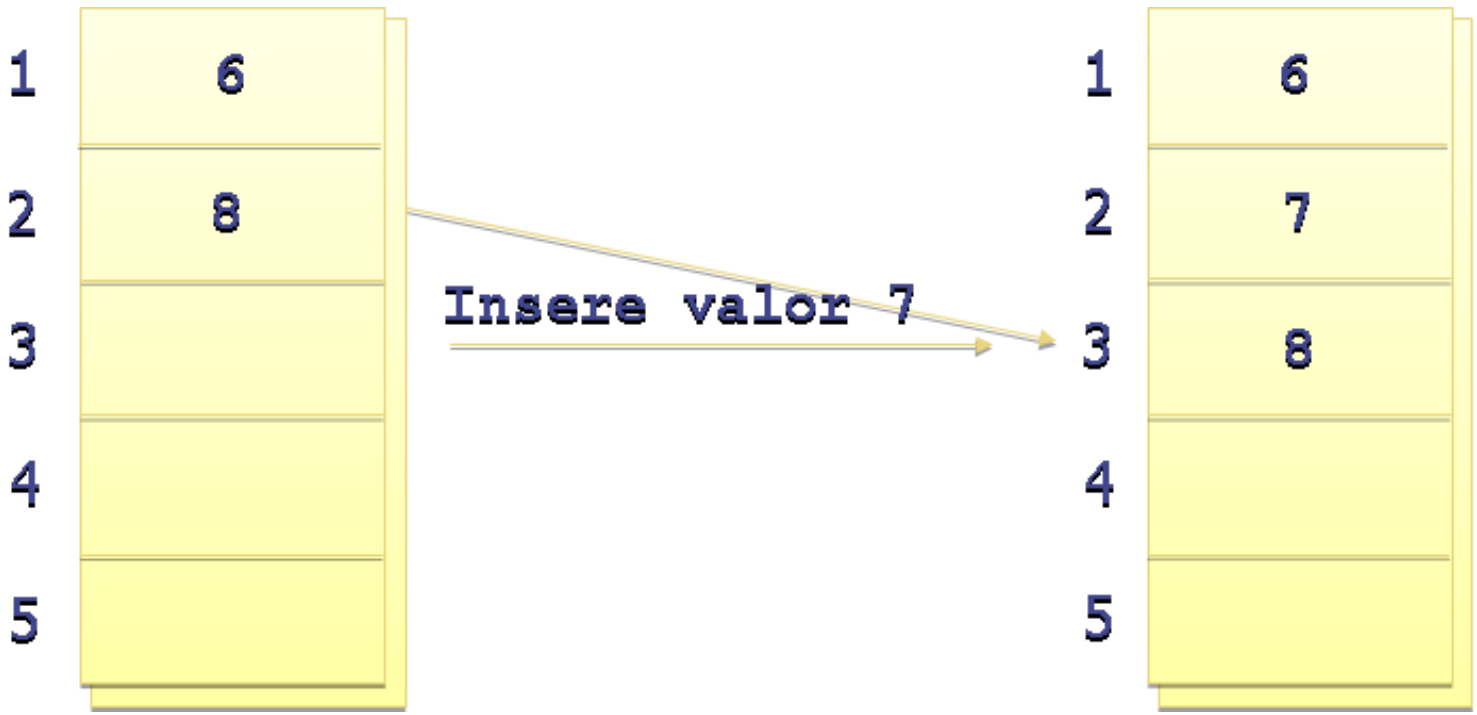
Inserção Ordenada

- Exemplo de inserção ordenada



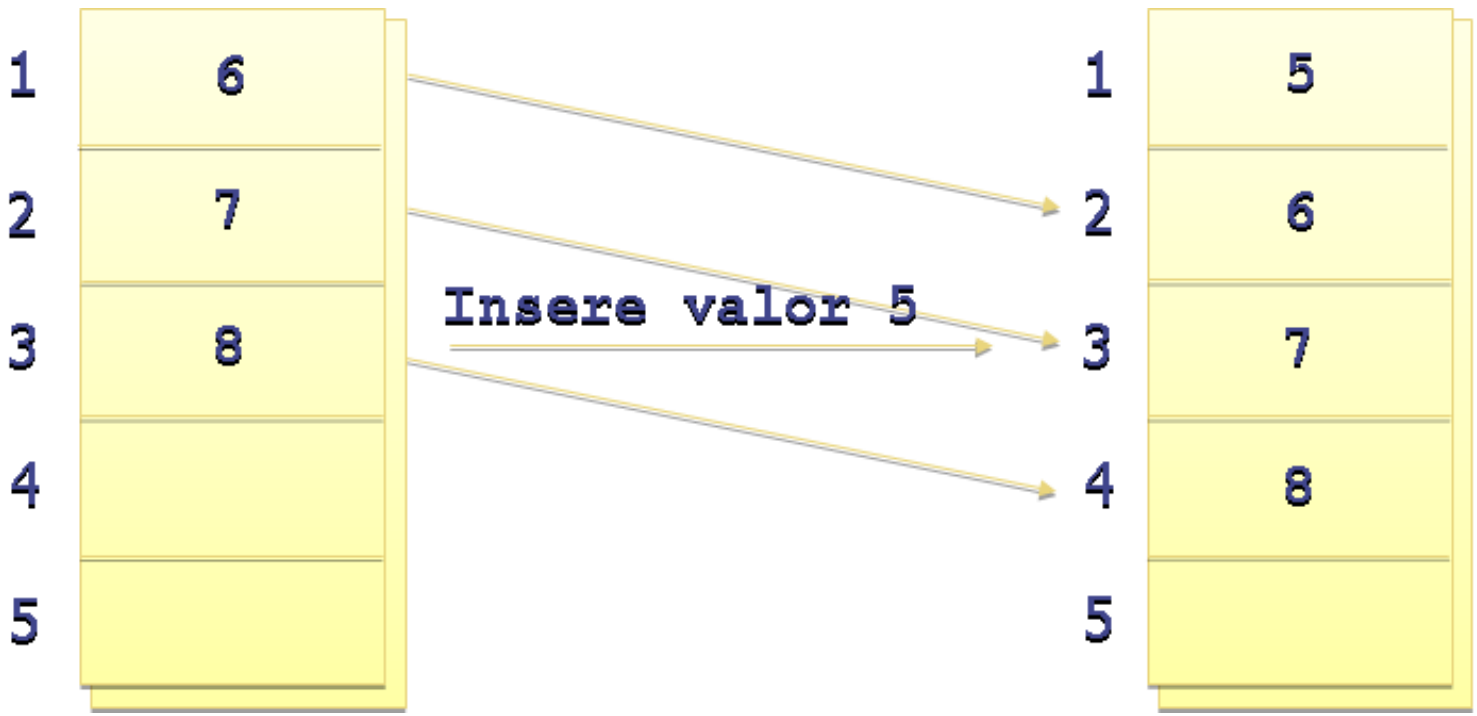
Inserção Ordenada

- Exemplo de inserção ordenada



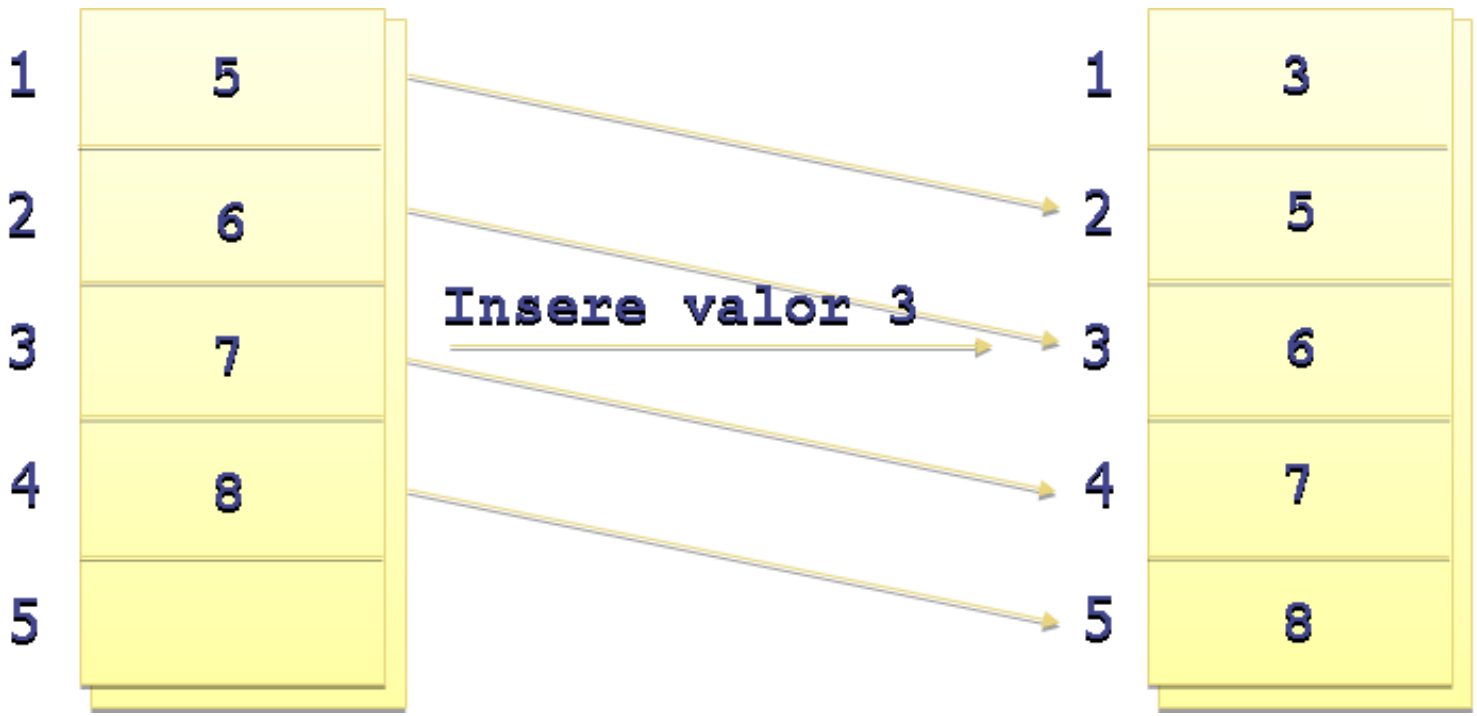
Inserção Ordenada

- Exemplo de inserção ordenada



Inserção Ordenada

- Exemplo de inserção ordenada



Implementação Inserção Ordenada

```
1  int inserir_ordenado(LISTA_ESTATICA *lista, ITEM *item) {
2    if (!cheia(lista)) { //verifica se existe espaço
3      int pos = lista->fim+1;
4
5      //move os itens até encontrar a posição de inserção
6      while (pos > 0 && (lista->vetor[pos-1])->chave > item->chave) {
7        lista->vetor[pos] = lista->vetor[pos-1];
8        pos--;
9      }
10
11     lista->vetor[pos] = *item; //insire novo item
12     lista->fim++; //incrementa tamanho da lista
13     return 1;
14   }
15
16   return 0;
17 }
```

Implementação Remoção Ordenada

Exercícios

Implemente a remoção de itens de uma lista ordenada - mantendo a ordenação após a eliminação.

```
1 int remover_item(LISTA_ESTATICA *lista, int chave) {
2 ...
3 }
```

Sumário

Busca em Listas Estáticas

Introdução

- Uma tarefa comum a ser executada sobre listas é a **busca de itens** dado uma chave

Busca em Listas Estáticas

Introdução

- Uma tarefa comum a ser executada sobre listas é a **busca de itens** dado uma chave
- No caso da lista **não ordenada**, a busca será **sequencial** (consultar todos os elementos)

Busca em Listas Estáticas

Introdução

- Uma tarefa comum a ser executada sobre listas é a **busca de itens** dado uma chave
- No caso da lista **não ordenada**, a busca será **sequencial** (consultar todos os elementos)
- Porém, com uma **lista ordenada**, diferentes estratégias podem ser aplicadas que aceleram essa busca

Busca em Listas Estáticas

Introdução

- Uma tarefa comum a ser executada sobre listas é a **busca de itens** dado uma chave
- No caso da lista **não ordenada**, a busca será **sequencial** (consultar todos os elementos)
- Porém, com uma **lista ordenada**, diferentes estratégias podem ser aplicadas que aceleram essa busca
  - Busca sequencial "otimizada"

Busca em Listas Estáticas

Introdução

- Uma tarefa comum a ser executada sobre listas é a **busca de itens** dado uma chave
- No caso da lista **não ordenada**, a busca será **sequencial** (consultar todos os elementos)
- Porém, com uma **lista ordenada**, diferentes estratégias podem ser aplicadas que aceleram essa busca
  - Busca sequencial "otimizada"
  - Busca binária

Busca Sequencial

Busca sequencial

- Na busca sequencial, a ideia é procurar um elemento que tenha uma determinada chave, começando do início da lista, e parar quando a lista terminar ou quando o elemento for encontrado

Implementação Busca Sequencial

```
1  ITEM *busca_sequencial(LISTA_ESTATICA *lista, int chave) {
2    int i;
3
4    for (i = 0; i <= lista->fim; i++) {
5      if (lista->vetor[i]->chave == chave) {
6        return lista->vetor[i];
7      }
8    }
9
10   return NULL;
11 }
```

Busca Sequencial (Ordenada)

- Quando os elementos estão ordenados, a busca sequencial pode ser "otimizada" (acelerada)

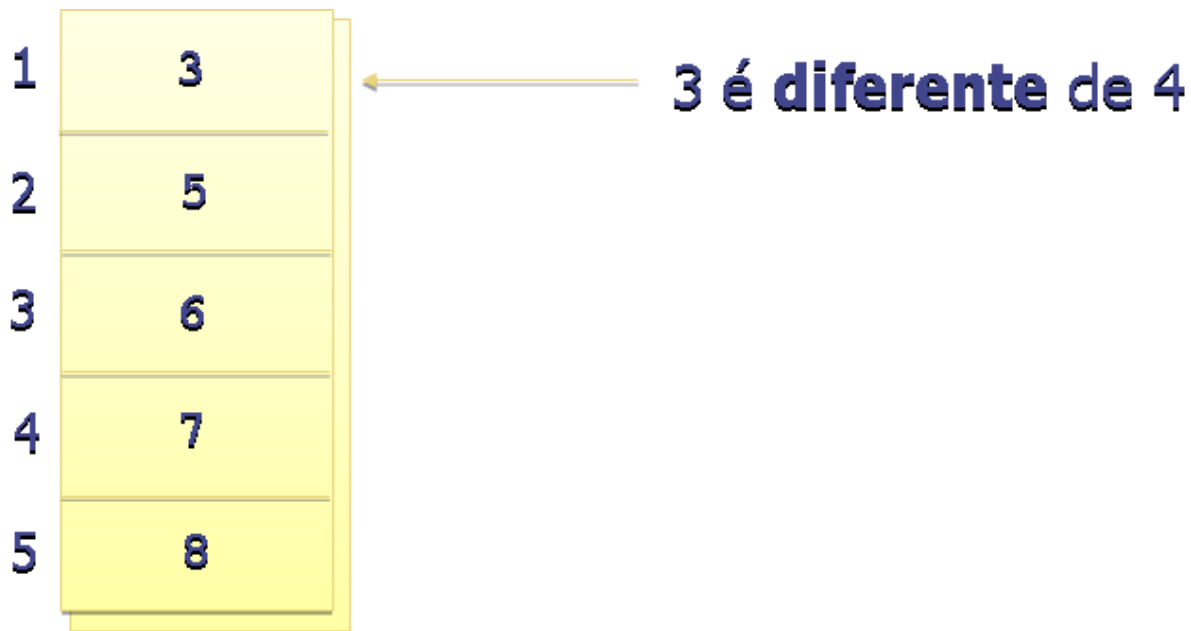
Busca Sequencial (Ordenada)

- Quando os elementos estão ordenados, a busca sequencial pode ser "otimizada" (acelerada)
- Vejamos o seguinte exemplo

Busca Sequencial (Ordenada)

Exemplo 1

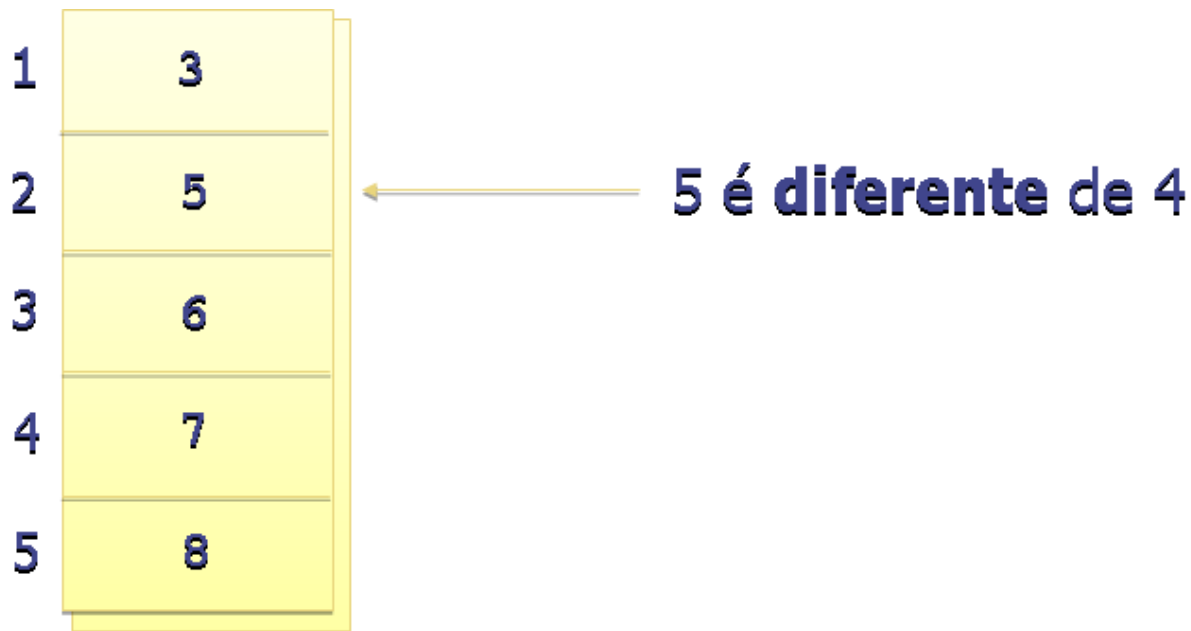
- procurar pelo valor 4



Busca Sequencial (Ordenada)

Exemplo 1

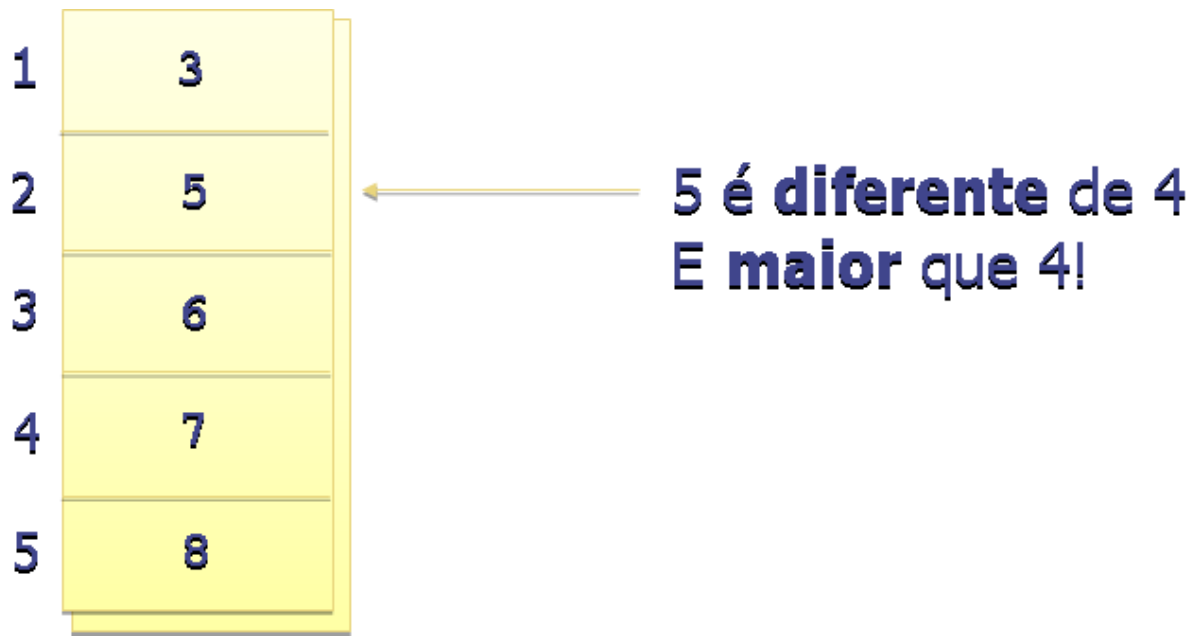
- procurar pelo valor 4



Busca Sequencial (Ordenada)

Exemplo 1

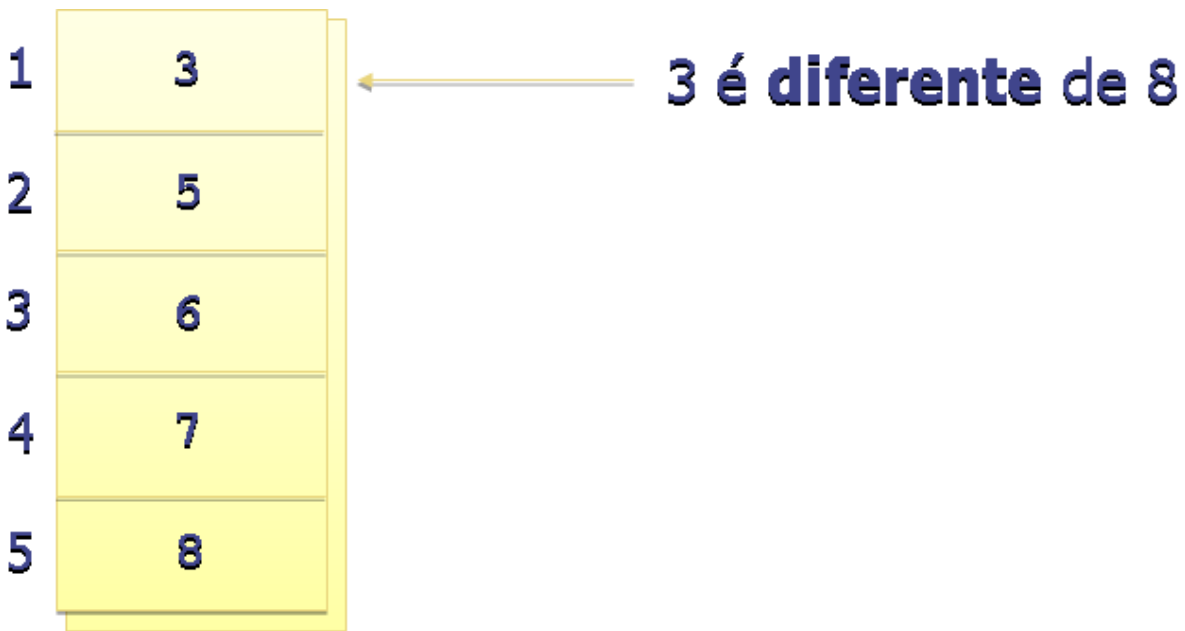
- procurar pelo valor 4



Busca Sequencial (Ordenada)

Exemplo 2

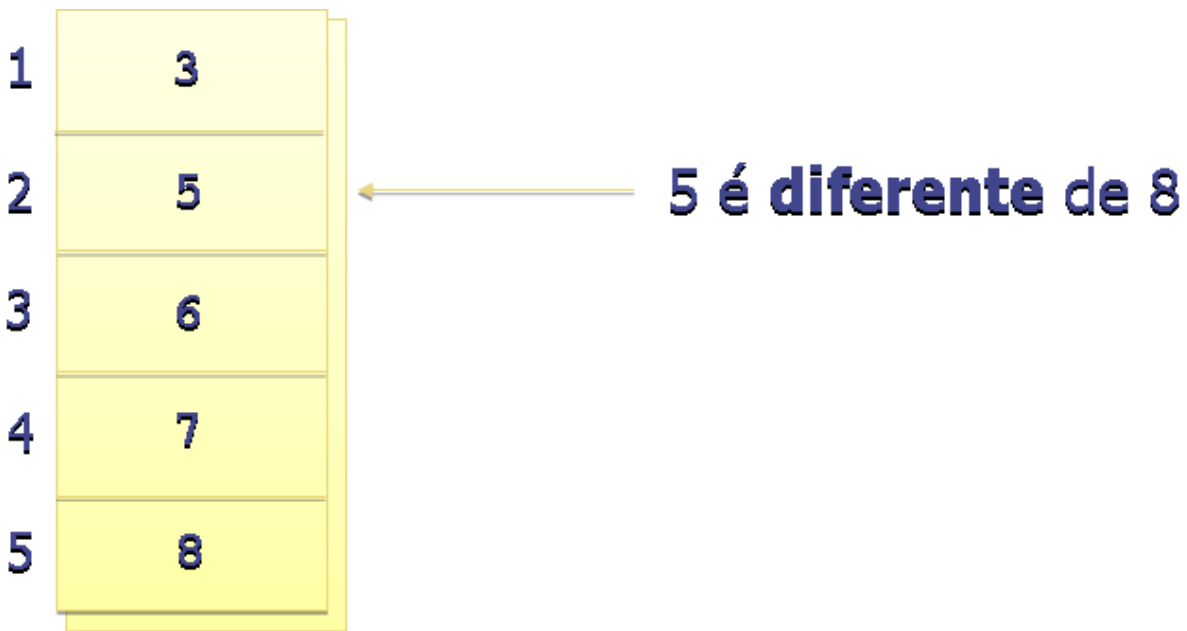
- procurar pelo valor 8



Busca Sequencial (Ordenada)

Exemplo 2

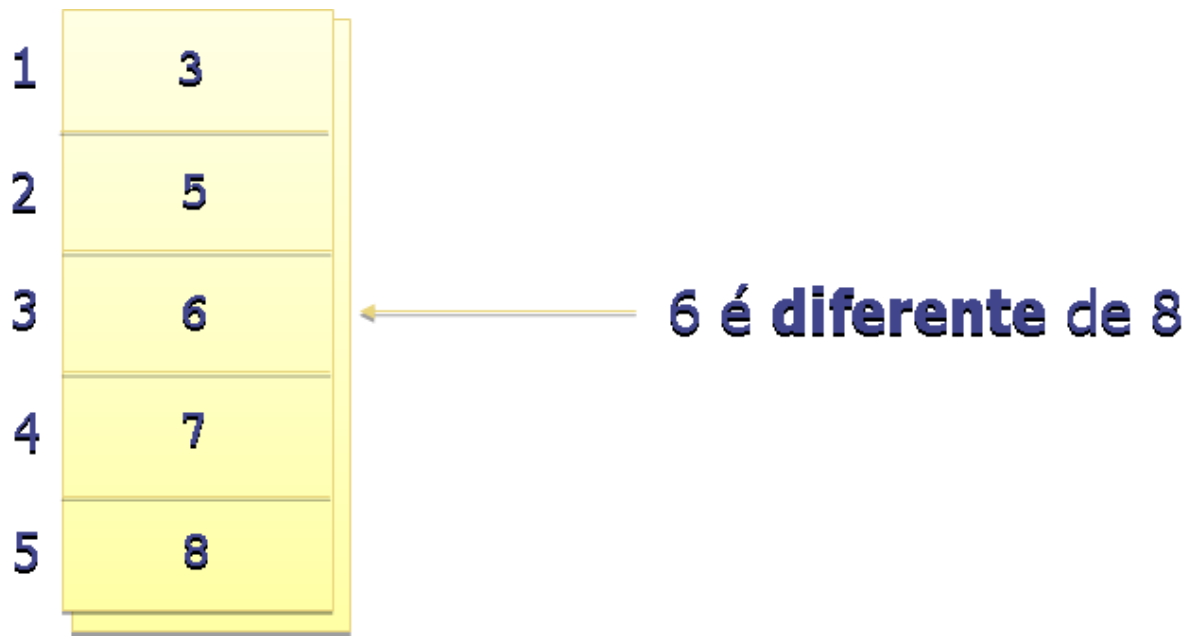
- procurar pelo valor 8



Busca Sequencial (Ordenada)

Exemplo 2

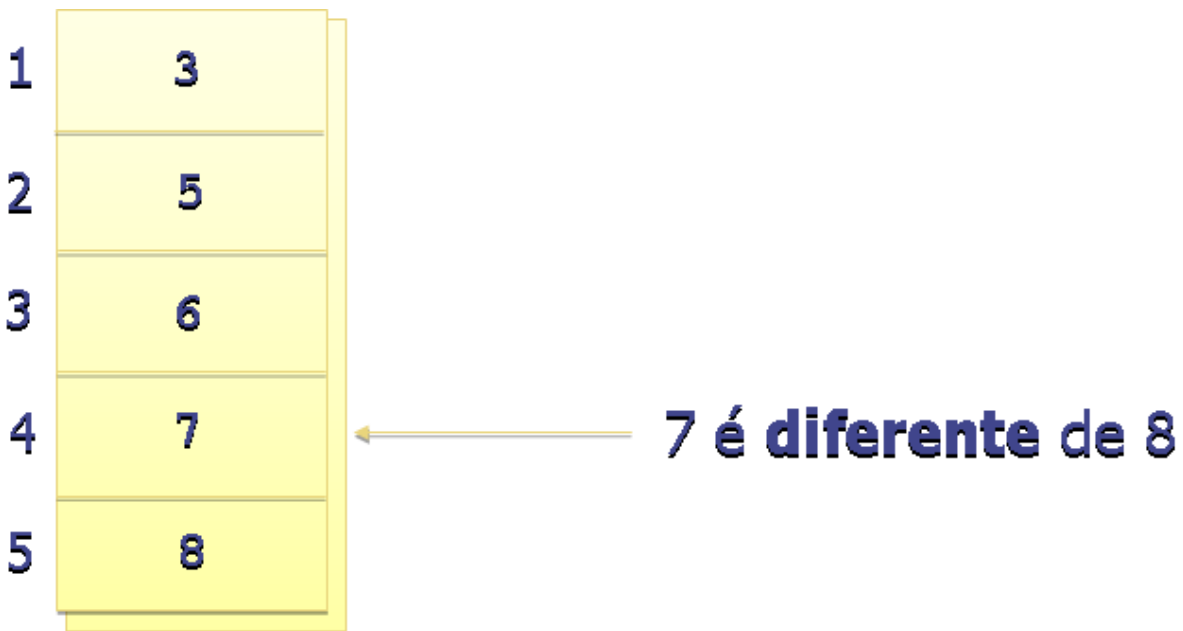
- procurar pelo valor 8



Busca Sequencial (Ordenada)

Exemplo 2

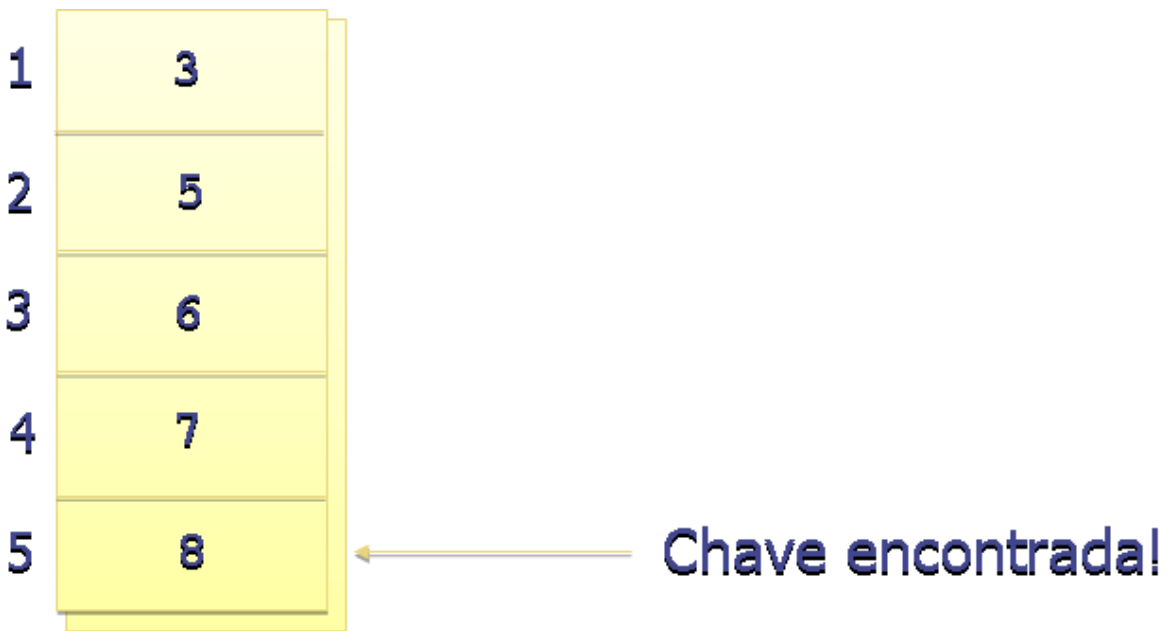
- procurar pelo valor 8



Busca Sequencial (Ordenada)

Exemplo 2

- procurar pelo valor 8



Busca Sequencial (Ordenada)

- A **lista ordenada** permite **realizar buscas sequenciais mais rápidas** uma vez que, caso a chave procurada não exista, pode-se parar a busca tão logo se encontre um elemento com chave maior que a procurada

Implementação Busca Sequencial

```
1  ITEM *busca_sequencial(LISTA_ESTATICA *lista, int chave) {
2    int i;
3
4    for (i = 0; i <= lista->fim; i++) {
5      if (lista->vetor[i]->chave == chave) {
6        return lista->vetor[i];
7      } else if (lista->vetor[i]->chave > chave) {
8        return NULL;
9      }
10   }
11
12   return NULL;
13 }
```

Busca Sequencial (Ordenada)

- Entretanto, essa melhoria **não altera a complexidade da busca sequencial**, que ainda é $O(n)$. Porque?

Busca Binária

- A **busca binária** é um algoritmo de busca mais sofisticado e **bem mais eficiente** que a busca sequencial

Busca Binária

- A **busca binária** é um algoritmo de busca mais sofisticado e **bem mais eficiente** que a busca sequencial
- Entretanto, a busca binária **somente pode ser aplicada em estruturas que permitem acessar cada elemento em tempo constante**, tais como os vetores

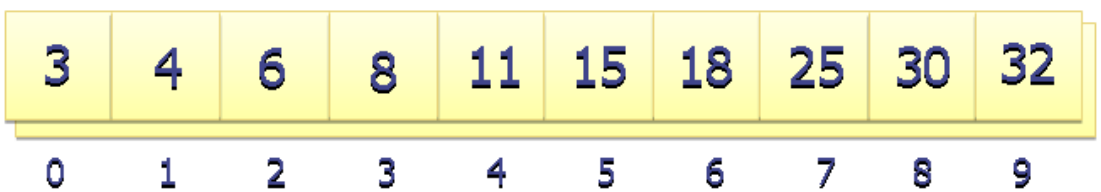
Busca Binária

- A **busca binária** é um algoritmo de busca mais sofisticado e **bem mais eficiente** que a busca sequencial
- Entretanto, a busca binária **somente pode ser aplicada em estruturas que permitem acessar cada elemento em tempo constante**, tais como os vetores
- A ideia é, a cada iteração, dividir o vetor ao meio e descartar metade do vetor

Busca Binária

Exemplo 1

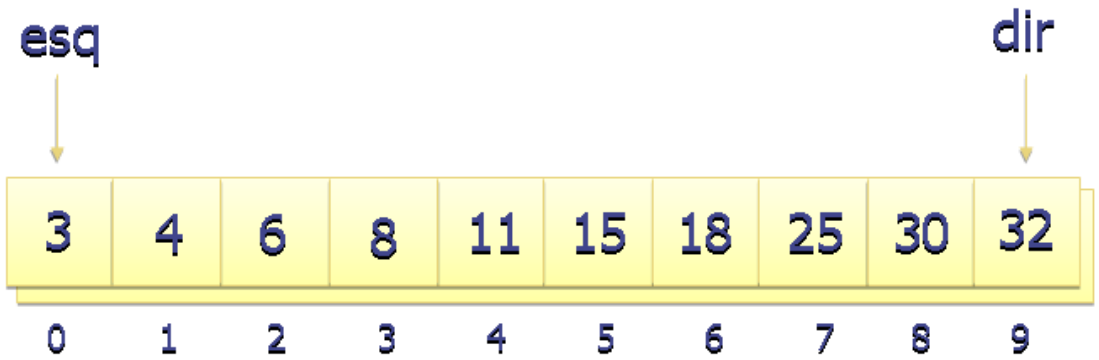
- Para procurar pelo valor 30



Busca Binária

Exemplo 1

- Para procurar pelo valor 30

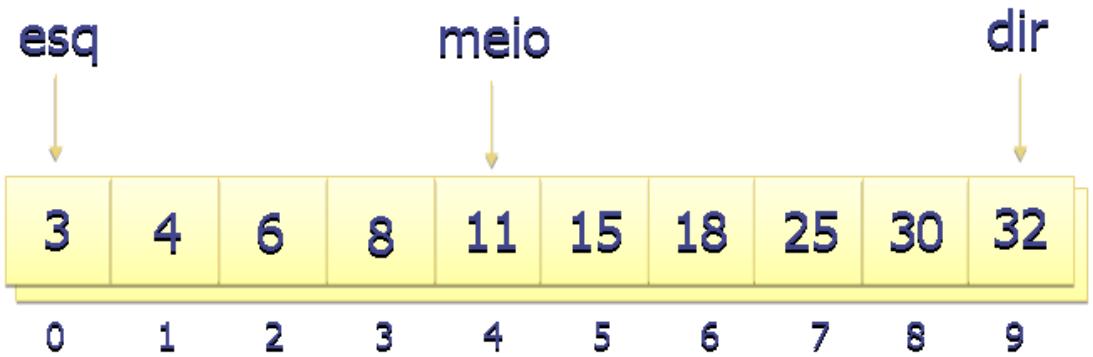


meio = (esq + dir) / 2
9 / 2 = 4

Busca Binária

Exemplo 1

- Para procurar pelo valor 30

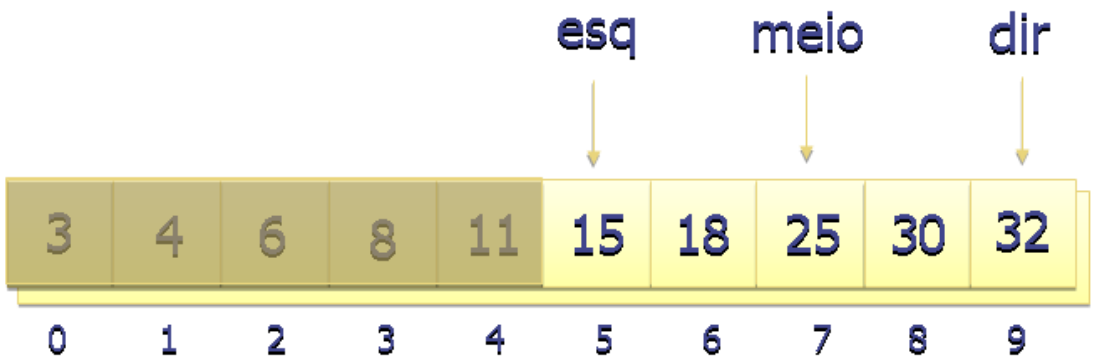


itens[meio].chave **menor** do que 30
esq = meio + 1

Busca Binária

Exemplo 1

- Para procurar pelo valor 30

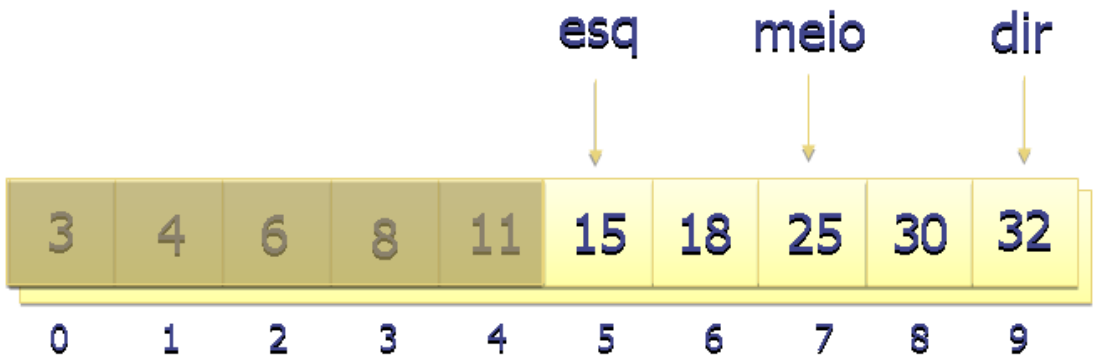


meio = (esq + dir) / 2
14 / 2 = 7

Busca Binária

Exemplo 1

- Para procurar pelo valor 30

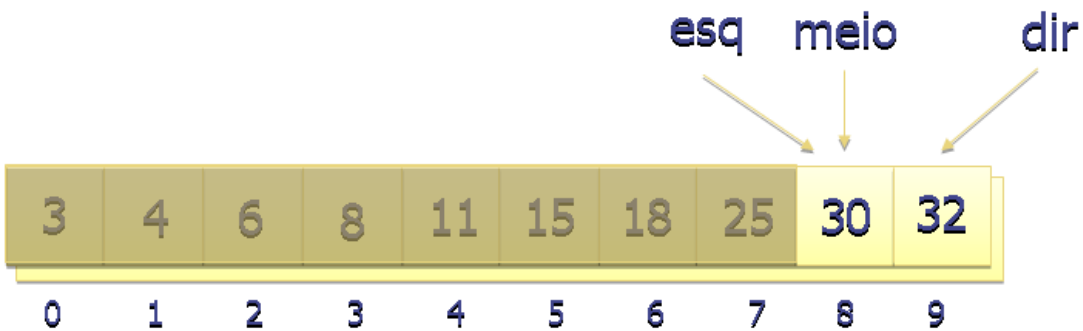


itens[meio].chave **menor** do que 30
esq = meio + 1

Busca Binária

Exemplo 1

- Para procurar pelo valor 30

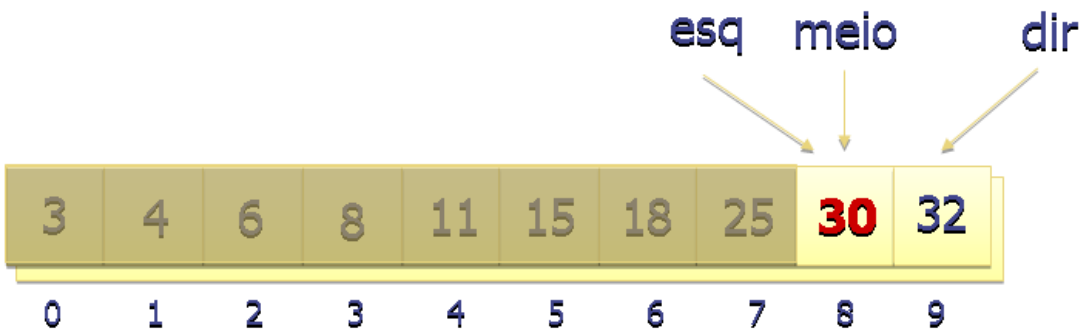


itens[meio].chave **igual** a 30

Busca Binária

Exemplo 1

- Para procurar pelo valor 30

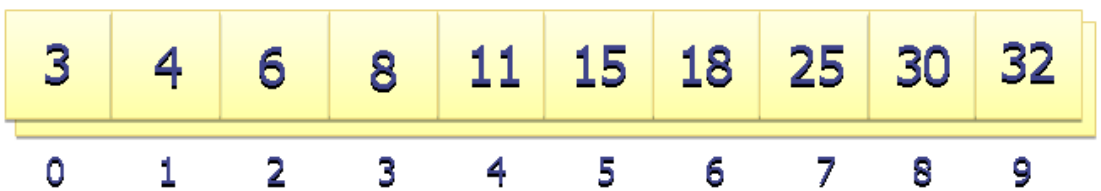


Chave Encontrada.

Busca Binária

Exemplo 2

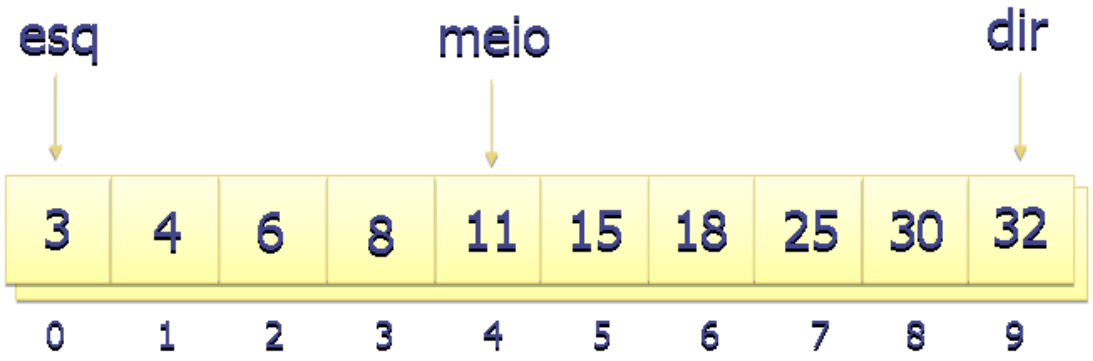
- Para procurar pelo valor 7 (inexistente!)



Busca Binária

Exemplo 2

- Para procurar pelo valor 7 (inexistente!)

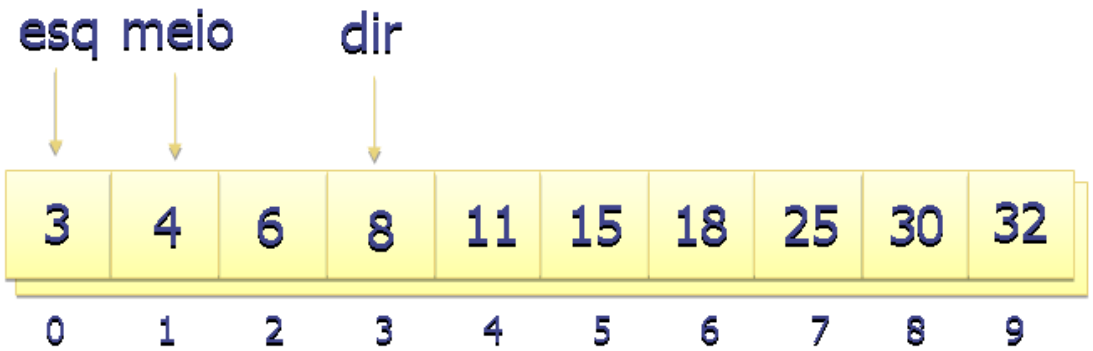


itens[meio].chave **maior** do que 7
dir = meio - 1

Busca Binária

Exemplo 2

- Para procurar pelo valor 7 (inexistente!)

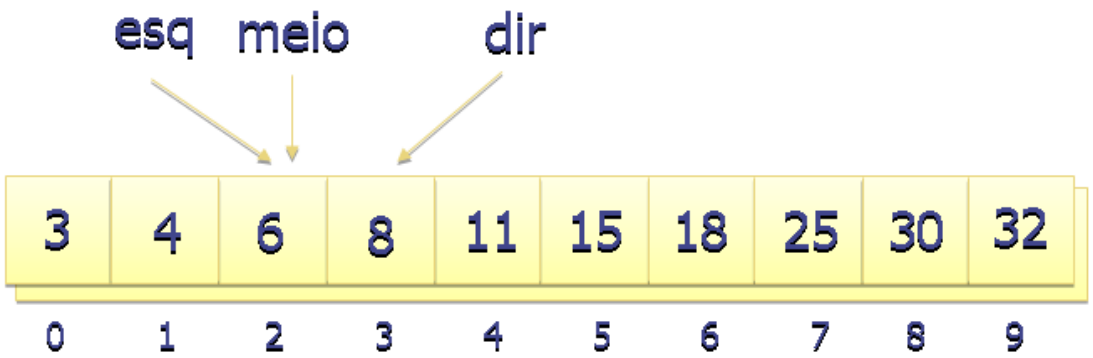


itens[meio].chave **menor** do que 7
esq = meio + 1

Busca Binária

Exemplo 2

- Para procurar pelo valor 7 (inexistente!)

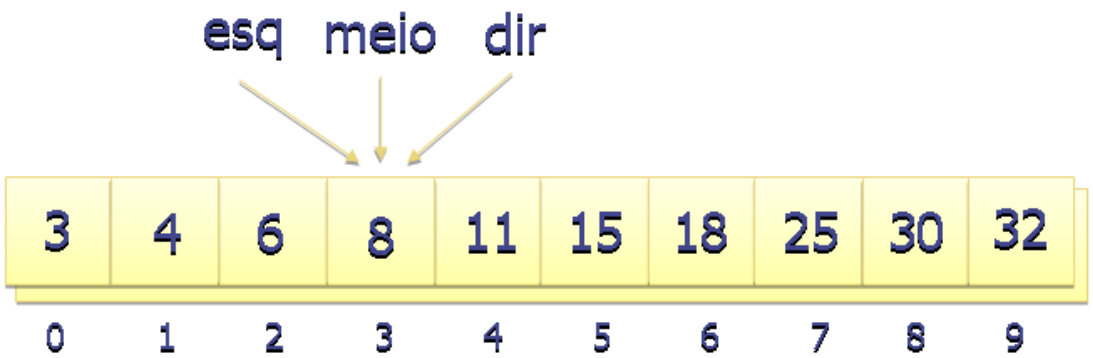


itens[meio].chave **menor** do que 7
esq = meio + 1

Busca Binária

Exemplo 2

- Para procurar pelo valor 7 (inexistente!)

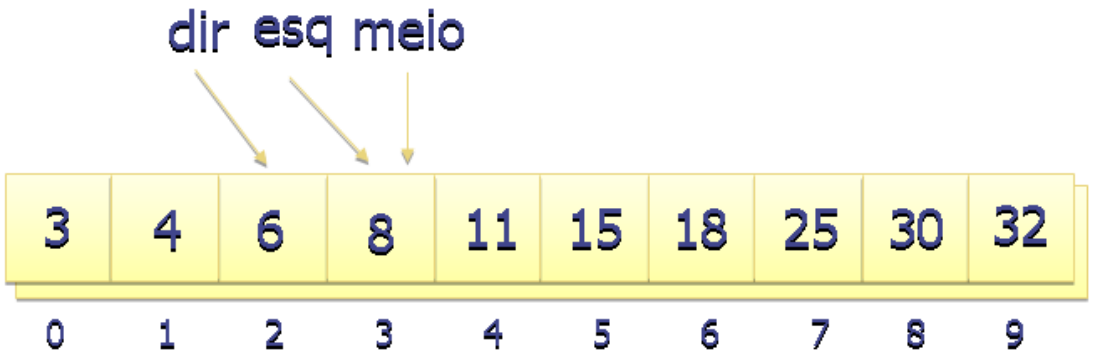


itens[meio].Chave **maior** do que 7
dir = meio - 1

Busca Binária

Exemplo 2

- Para procurar pelo valor 7 (inexistente!)



dir < esq => chave não encontrada!

Busca Binária

- É importante lembrar que

Busca Binária

- É importante lembrar que
  - Busca binária somente funciona em vetores ordenados

Busca Binária

- É importante lembrar que
  - Busca binária somente funciona em vetores ordenados
  - Busca sequencial funciona com vetores ordenados ou não

Busca Binária

- É importante lembrar que
  - Busca binária somente funciona em vetores ordenados
  - Busca sequencial funciona com vetores ordenados ou não
- A busca binária é muito eficiente. Ela é $O(log_2\ n)$. Para $n = 1.000.000$, aproximadamente 20 comparações são necessárias

Implementação Busca Binária

```
1  ITEM *busca_binaria(LISTA_ESTATICA_ORDENADA *lista, int chave) {
2    int esq = 0;
3    int dir = lista->fim;
4
5    while (esq <= dir) {
6      int meio = (esq + dir) / 2;
7
8      if (lista->vetor[meio]->chave == chave) {
9        return lista->vetor[meio];
10     } else if (lista->vetor[meio]->chave > chave) {
11       dir = meio - 1;
12     } else {
13       esq = meio + 1;
14     }
15   }
16
17   return NULL;
18 }
```

Exercício

Exercício

- Implemente a busca binária utilizando um procedimento recursivo