



AED - Algoritmos e Estruturas de Dados

Aula 9 – Listas Sequenciais Ordenadas

Prof. Rodrigo Mafort

Voltando as Listas Sequenciais

- Ao definir uma lista (vetor) nas linguagens de programação temos duas opções:
 - **Alocação sequencial:**
 - Todo o espaço é alocado de uma única vez
 - Os elementos da lista são alocados de forma contígua (uma posição ao lado da outra).
 - Vantagem: Podemos acessar diretamente qualquer elemento da lista
 - Desvantagem: O espaço alocado não pode ser modificado (a lista tem 50 posições, não é possível estender para incluir 51 elementos)
 - Alocação dinâmica:
 - Alocação realizada de acordo com a necessidade
 - Cada elemento fica alocado em uma posição diferente da memória.
 - Não é possível acessar diretamente um elemento qualquer da lista. Apenas o primeiro.

Alocação Sequencial

- Listas de alocação sequencial podem ter comportamentos distintos de acordo com as regras utilizadas nas operações de inserção e remoção.
- Listas Sequenciais Não Ordenadas (Aula 8)
- **Listas Sequenciais Ordenadas**
 - Filas
 - Pilhas
 - Deques

Listas de Alocação Sequencial Ordenadas

- Assim como nas listas não ordenadas, os dados são armazenados como elementos de um vetor (estrutura subjacente).
- Precisamos identificar as operações que podem ser realizadas em listas ordenadas:
 1. Como definir a ordem entre dois elementos
 2. Como definir a lista (o vetor)
 3. Como verificar o número de elementos contidos na lista
 4. Como inserir um novo elemento na lista
 5. Como buscar por um determinado elemento na lista
 6. Como remover um elemento na lista
 7. Como alterar um elemento da lista

1 – Ordem entre dois elementos

- Diferente das listas não ordenadas, nas listas ordenadas precisamos definir como dois elementos serão comparados.
- Assim podemos dizer qual a ordem correta entre dois elementos.
- Essa ordem pode ser crescente (primeiro o menor, seguido do maior) ou decrescente (primeiro o maior, seguido do menor).
- Quando cada elemento da lista corresponde a uma estrutura (struct), precisamos definir um campo (em geral, numérico) como critério da ordenação.

1 – Definir uma chave as para comparações

```
typedef struct {  
    int ID;  
    char Nome[255];  
} elemento;
```

Definição da estrutura utilizada para cada célula da lista.

Podemos usar o campo ID para estabelecer a comparação entre dois elementos dessa estrutura

elemento e1;

elemento e2;

e1 < e2: Erro – Não sabemos comparar

E1.ID < e2.ID: Agora sabemos comparar

Listas de Alocação Sequencial Ordenadas

- Assim como nas listas não ordenadas, os dados são armazenados como elementos de um vetor.
- Precisamos identificar as operações que podem ser realizadas em listas ordenadas:
 1. Como definir a ordem entre dois elementos ✓
 2. Como definir a lista (o vetor)
 3. Como verificar o número de elementos contidos na lista
 4. Como inserir um novo elemento na lista
 5. Como buscar por um determinado elemento na lista
 6. Como remover um elemento na lista
 7. Como alterar um elemento da lista

2 – Como definir a lista

- Assim como nas listas não ordenadas, precisamos de um vetor (estrutura subjacente) para armazenar a lista.
- Podemos aplicar as mesmas regras já utilizadas na definição de listas não ordenadas (enquanto a lista estiver vazia, ela estará ordenada).
- Podemos definir um vetor estaticamente ou podemos usar o malloc. Entretanto, independentemente da forma escolhida é necessário lembrar que o tamanho da lista é finito e que as posições da lista ocupam posições contiguas na memória.

2 – Como definir a lista

```
#define TAM 50
```

```
typedef struct {  
    int ID;  
    char nome[255];  
} elemento;
```

```
elemento L[TAM];
```

Definição de uma **constante** que especifica o tamanho máximo da lista

Definição da estrutura utilizada para cada célula da lista.

Alocação estática da lista *L*

Observe que a lista terá no máximo TAM elementos.

2 - Definir a lista *L*: usando o malloc

```
#define TAM 50
```

```
typedef struct {  
    int ID;  
    char nome[255];  
} elemento;
```

```
elemento* L;  
L = (elemento*) malloc(TAM * sizeof(elemento));  
...  
free(L); //Depois de usar a lista, liberar a memória
```

Listas de Alocação Sequencial Ordenadas

- Assim como nas listas não ordenadas, os dados são armazenados como elementos de um vetor.
- Precisamos identificar as operações que podem ser realizadas em listas ordenadas:
 1. Como definir a ordem entre dois elementos ✓
 2. Como definir a lista (o vetor subjacente) ✓
 3. Como verificar o número de elementos contidos na lista
 4. Como inserir um novo elemento na lista
 5. Como buscar por um determinado elemento na lista
 6. Como remover um elemento na lista
 7. Como alterar um elemento da lista

3 - Verificar o número de elementos contidos na lista

- Vale a mesma regras das listas não ordenadas: após alocarmos uma lista, quantas posições dessa lista estão efetivamente ocupadas?
- Nenhuma! Não inserimos nenhum elemento na lista
- Como vamos controlar quantos elementos já inserimos na lista?
- Usando a mesma estratégia do exercício dos contêineres.
- Uma variável contadora: `nElementos`
- Na inicialização: `nElementos = 0`
- A cada inserção efetuada com sucesso: `nElementos = nElementos + 1`
- A cada remoção efetuada com sucesso: `nElementos = nElementos - 1`

Listas Sequenciais: Ordenadas ou não

- Ao usar listas precisamos de duas variáveis:
 - A própria lista (independente do tipo)
 - Uma variável para contabilizar quantas posições estão efetivamente ocupadas.
- Até agora para usar criar uma nova lista:
 - Inicializar/Declarar a lista
 - Inicializar a contadora de posições com 0
- Todas as operações (inserção, remoção, busca) vão necessitar dessas duas informações

Listas de Alocação Sequencial Ordenadas

- Assim como nas listas não ordenadas, os dados são armazenados como elementos de um vetor.
- Precisamos identificar as operações que podem ser realizadas em listas ordenadas:
 1. Como definir a ordem entre dois elementos ✓
 2. Como definir a lista (o vetor subjacente) ✓
 3. Como verificar o número de elementos contidos na lista ✓
 4. Como inserir um novo elemento na lista
 5. Como buscar por um determinado elemento na lista
 6. Como remover um elemento na lista
 7. Como alterar um elemento da lista

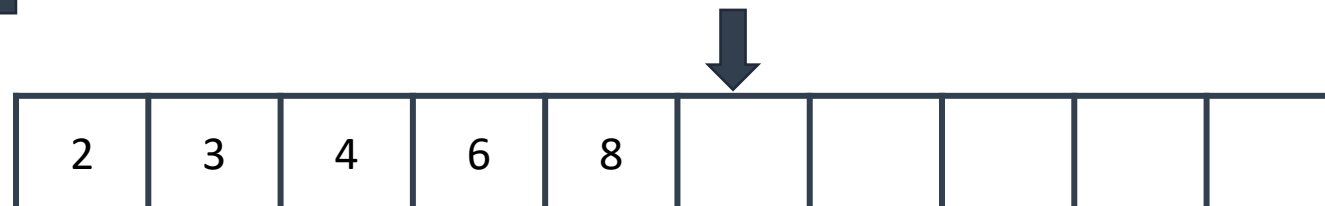
Inserção em Listas Sequenciais – Não Ord.

- A inserção é a operação mais simples em uma lista não ordenada
- Constituída de três passos:
 1. Verificar se a lista não está cheia. Se estiver, interromper a operação de inserção (não há espaço disponível).
 2. Ocupar a última posição da lista
 3. Incrementar a contadora de posições ocupadas.
- Isso vale para listas ordenadas?
- Inserir na última posição respeita a ordenação entre os elementos?
- Não... Precisamos ser mais criteriosos sobre onde vamos inserir na lista.

4- Inserção em Listas Sequenciais Ordenadas

- Temos três casos possíveis:
 - A lista está vazia: Podemos inserir na primeira posição
 - A lista está cheia: Não podemos inserir
 - A lista já possui elementos, mas não está cheia.
- Nesse último caso, precisamos identificar a posição correta da inserção:

Inserir 10:



4- Inserção em Listas Sequenciais Ordenadas

- Temos três casos possíveis:
 - A lista está vazia: Podemos inserir na primeira posição
 - A lista está cheia: Não podemos inserir
 - A lista já possui elementos, mas não está cheia.
- Nesse último caso, precisamos identificar a posição correta da inserção:

Inserir 5:



4- Inserção em Listas Sequenciais Ordenadas

- Temos três casos possíveis:
 - A lista está vazia: Podemos inserir na primeira posição
 - A lista está cheia: Não podemos inserir
 - A lista já possui elementos, mas não está cheia.
- Nesse último caso, precisamos identificar a posição correta da inserção:



4 - Inserção em Listas Sequenciais Ordenadas

- Como identificar a posição correta para inserir um novo elemento?
- Ideia:
 - Olhar a posição i da lista
 - O elemento que ocupa essa posição é maior ou menor do que o elemento que eu desejo inserir?
 - Menor: O novo elemento deve ficar depois desse elemento na lista. Vamos olhar a próxima posição ($i++$).
 - Maior: O novo elemento deve ficar antes desse elemento na lista.
 - Igual: O novo elemento é igual ao elemento que ocupa essa posição da lista.
- Essa ideia não é uma operação de busca?
- E se usássemos a busca na inserção:
 - A busca vai retornar o elemento encontrado ou a posição que esse novo elemento deve ocupar
 - Observe que essa posição é ocupada justamente pelo primeiro elemento maior do que o elemento que desejamos inserir.

Listas de Alocação Sequencial Ordenadas

- Assim como nas listas não ordenadas, os dados são armazenados como elementos de um vetor.
- Precisamos identificar as operações que podem ser realizadas em listas ordenadas:
 1. Como definir a ordem entre dois elementos ✓
 2. Como definir a lista (o vetor subjacente) ✓
 3. Como verificar o número de elementos contidos na lista ✓
 4. Como inserir um novo elemento na lista
 5. Como buscar por um determinado elemento na lista
 6. Como remover um elemento na lista
 7. Como alterar um elemento da lista

Listas de Alocação Sequencial Ordenadas

- Assim como nas listas não ordenadas, os dados são armazenados como elementos de um vetor.
- Precisamos identificar as operações que podem ser realizadas em listas ordenadas:
 1. Como definir a ordem entre dois elementos ✓
 2. Como definir a lista (o vetor subjacente) ✓
 3. Como verificar o número de elementos contidos na lista ✓
 4. Como buscar por um determinado elemento na lista
 5. Como inserir um novo elemento na lista
 6. Como remover um elemento na lista
 7. Como alterar um elemento da lista

4 - Busca em listas sequenciais ordenadas

- Como ainda não observamos o algoritmo de inserção, vamos assumir que as listas estão ordenadas em ordem crescente.
- Houve algum critério na ordenação? **Sim**.
- Cada novo elemento foi inserido no final da lista (ocupando a última posição da lista).
- Se a inserção seguiu a ordem 1, 2, 3, 4, 5, a lista é [1,2,3,4,5]
- Se a inserção seguiu a ordem 5, 4, 3, 2, 1, a lista é [1,2,3,4,5]
- Independente das inserções e remoções, a lista segue ordenada.

4 - Busca em listas sequenciais ordenadas

- Existem algumas estratégias possíveis para realizar a busca em listas ordenadas.
 - **Busca exaustiva** (tal como em listas não ordenadas)
 - Busca ordenada
 - Busca binária
- **Busca exaustiva – $O(n)$:**
 - Percorrer toda a lista.
 - Se encontrar retornar o elemento.
 - Se terminar a lista sem encontrar, informar que não está na lista.
- Em uma lista com 100 elementos, quantas comparações precisamos?
 - Estou com sorte: 1 comparação (o elemento é o primeiro da lista)
 - Estou com azar: 100 comparações (o elemento não está na lista)
- Podemos afirmar que um elemento não está na lista sem percorrer toda a lista???
 - Sim! Basta identificar um elemento maior do que o procurado.

4 - Busca em listas sequenciais ordenadas

- Existem algumas estratégias possíveis para realizar a busca em listas ordenadas.
 - Busca ordenada
 - Busca binária
- **Busca ordenada:**
 - Percorrer toda a lista.
 - Encontrou: retorne a posição do elemento
 - Não encontrou - O elemento atual é maior do que o buscado?
 - Sim: interromper a busca (se o elemento estivesse na lista, estaria antes do atual)
 - Não: continuar a busca até atingir o final da lista
- Em uma lista com 100 elementos, quantas comparações precisamos?
 - Estou com sorte: 1 comparação (o elemento é o primeiro da lista)
 - Estou com azar: 100 comparações (o elemento é maior do que o último da lista)

4 - Busca em listas sequenciais ordenadas

```
int Buscar(Elemento L[], int nElementos, int id)
{
    int i = 0;
    while (i < nElementos && L[i].ID < id)
        i++;
    return i;
}
```

4 - Busca em listas sequenciais ordenadas

- Função Buscar:
 - Se a lista está vazia:
 - Retorna 0
 - Se o elemento não está na lista e não existe ninguém menor do que ele:
 - Retorna 0
 - Se o elemento não está na lista, mas existe alguém maior do que ele:
 - Retorna a posição do primeiro elemento maior do que ele
 - Se o elemento não está na lista e não existe ninguém maior do que ele:
 - Retorna nElementos
 - Se o elemento está na lista:
 - Retorna a posição do elemento buscado
- O algoritmo que chamou a busca precisa identificar e tratar cada caso.

Complexidade da Busca:

1. Qual é a operação dominante?
 - Comparações.
 2. Qual é o pior caso possível para a busca?
 - O elemento é maior do que o último elemento
 3. No pior caso, quantas comparações precisamos realizar para identificar a resposta correta em uma lista com n elementos?
 - n comparações.
- Logo:
 - Complexidade da Busca: $O(n)$

4 - Busca em listas sequenciais ordenadas

- Existem algumas estratégias possíveis para realizar a busca em listas ordenadas.
 - Busca ordenada ✓
 - Busca binária

Busca Binária

- O algoritmo de busca anterior, no pior dos casos, ainda precisamos percorrer todo o vetor: $O(n)$
- Como podemos melhorar?



Como buscamos uma palavra p no dicionário?

Busca no dicionário

- O que fazemos quando queremos encontrar uma palavra p no dicionário?
 - Dividimos o dicionário em duas partes e olhamos a primeira palavra da página do meio. Seja p_{meio} essa palavra
 - Se a palavra que buscamos p vem antes de $p_{meio} : p < p_{meio}$
 - p está na primeira metade do dicionário
 - E podemos ignorar o segunda parte
 - Se a palavra que buscamos p vem depois de $p_{meio} : p > p_{meio}$
 - p está na segunda metade do dicionário
 - E podemos ignorar a primeira parte

Busca no dicionário

- Agora que descobrimos em qual metade do dicionário está a palavra que procuramos, como vamos achar a palavra?
- Procurar uma por uma? Não!
- Vamos usar a mesma estratégia:
 - Vamos olhar a página situada no meio da parte do dicionário onde nossa palavra está
 - Se a palavra que buscamos vem antes:
 - Ela está na primeira metade dessa parte do dicionário
 - E podemos ignorar o restante do dicionário
 - Se a palavra que buscamos vem depois:
 - Ela está na segunda metade dessa parte do dicionário
 - E podemos ignorar o restante do dicionário

Busca no dicionário

D

1	autêntico
2	bagunceiro
3	crença
4	desembarque
5	empreitada
6	fatura
7	gaiola
8	habilidade
9	iceberg
10	jangada
11	kart
12	limitado
13	maestro
14	navegante
15	oxigênio
16	palavra

p

maestro

Busca no dicionário

D

1	autêntico
2	bagunceiro
3	crença
4	desembarque
5	empreitada
6	fatura
7	gaiola
8	habilidade
9	iceberg
10	jangada
11	kart
12	limitado
13	maestro
14	navegante
15	oxigênio
16	palavra

← *ini*

← *meio = habilidade*

← *fim*

p

maestro

Busca no dicionário

D

1	autêntico
2	bagunceiro
3	crença
4	desembarque
5	empreitada
6	fatura
7	gaiola
8	habilidade
9	iceberg
10	jangada
11	kart
12	limitado
13	maestro
14	navegante
15	oxigênio
16	palavra

ini

meio = habilidade

p

maestro

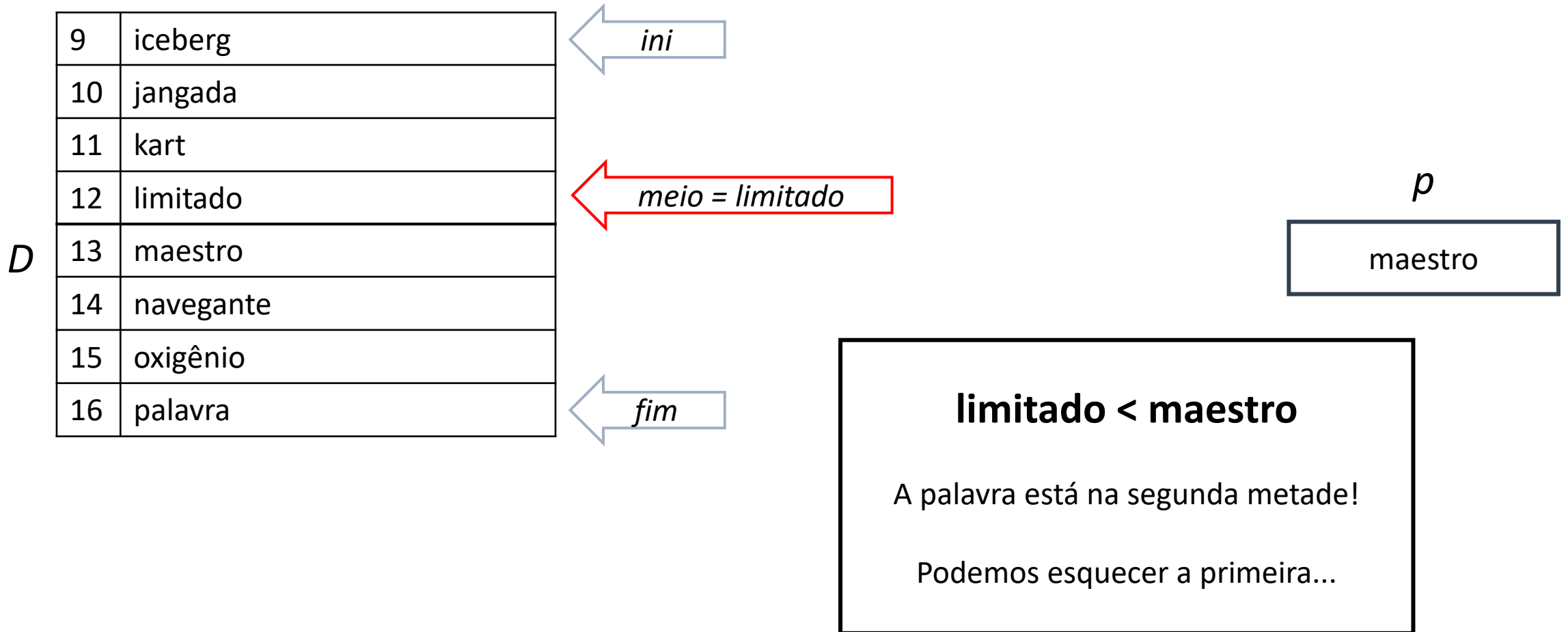
habilidade < maestro

A palavra está na segunda metade!

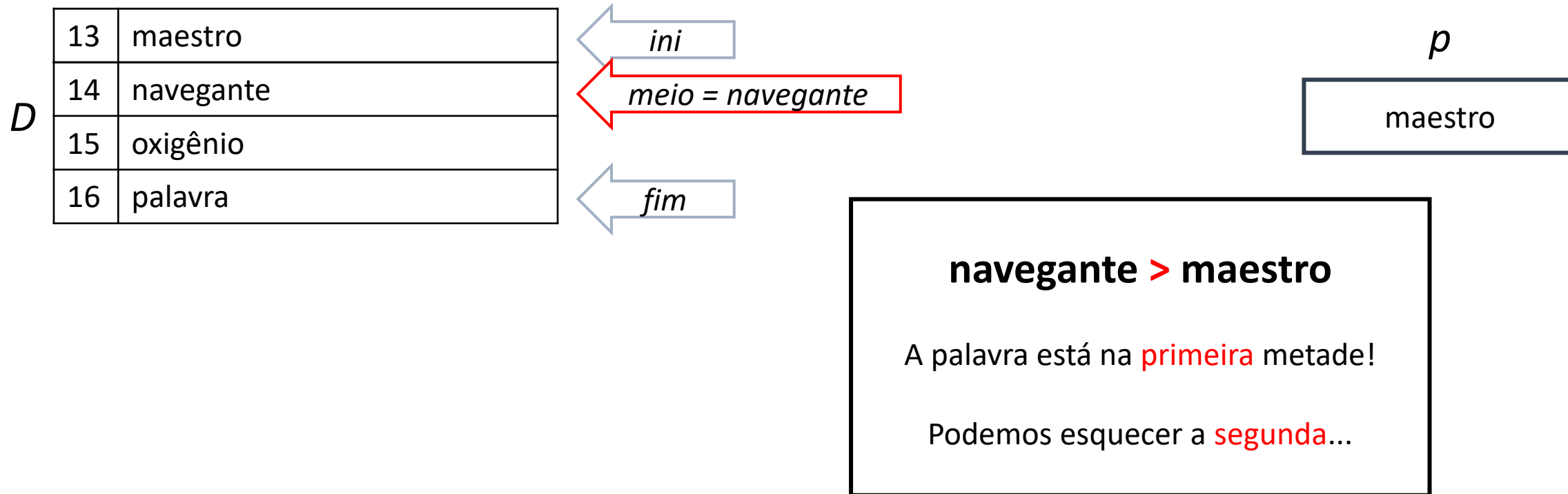
Podemos esquecer a primeira...

fim

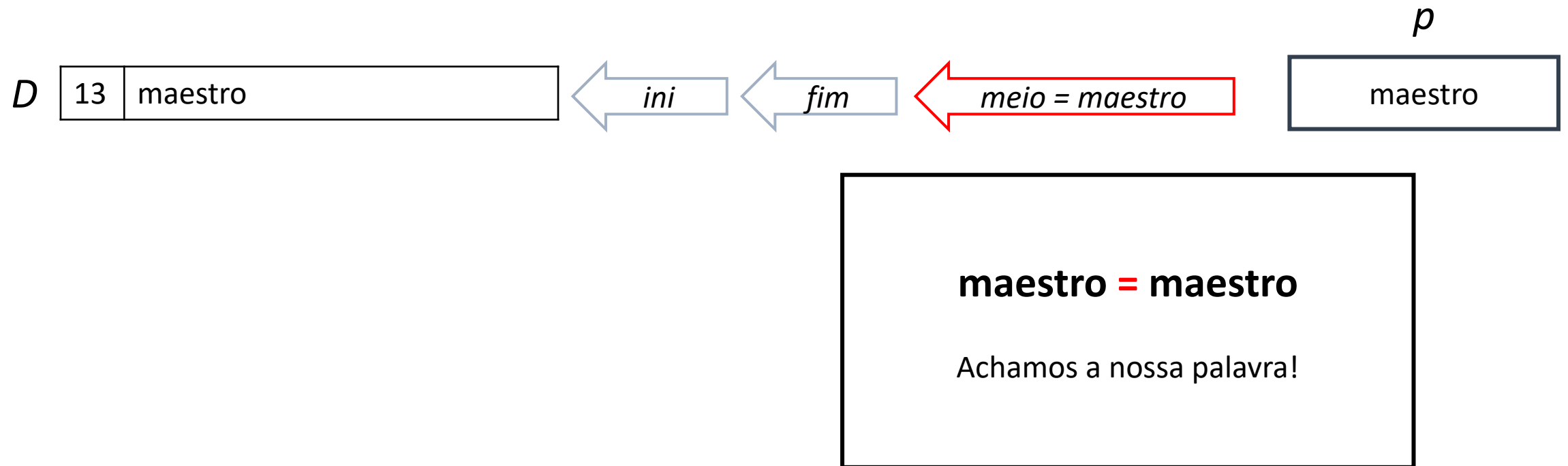
Busca no dicionário



Busca no dicionário



Busca no dicionário



Número de comparações

1. Habilidade e Maestro
 2. Limitado e Maestro
 3. Navegante e Maestro
 4. Maestro e Maestro
- Para localizar nossa palavra em um dicionário com 16 palavras, fizemos somente 4 comparações!
 - Na busca sequencial, precisaríamos de 13 comparações

Busca no dicionário – Um algoritmo

Algoritmo: Busca por uma palavra no dicionário

Entrada: Dicionário D e Palavra desejado p

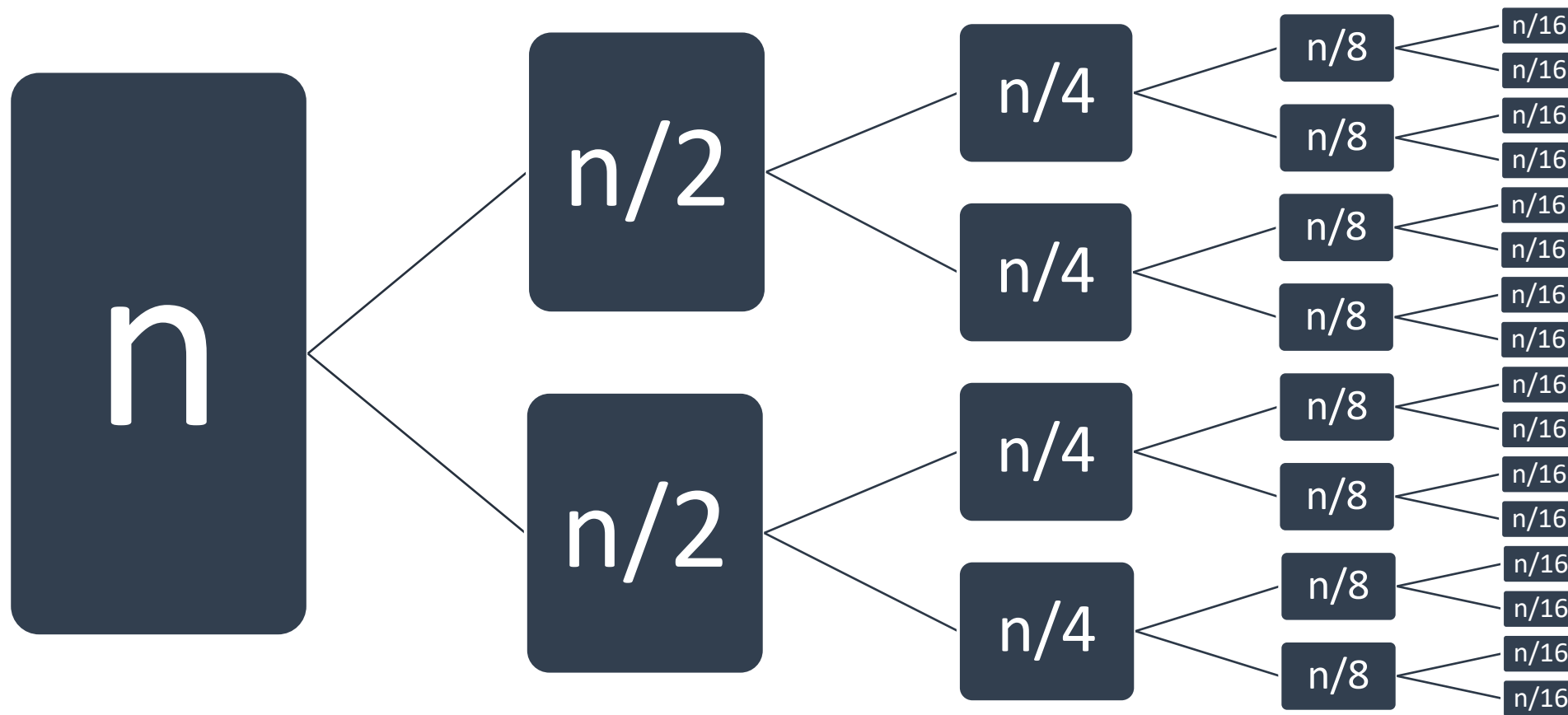
Saída: Elemento localizado ou não?

- 1 Seja ini a primeira palavra do dicionário D
- 2 Seja fim a última palavra do dicionário D
- 3 **retorne** BuscarDicionário (D, ini, fim, p)

```
1 Função BuscarDicionário( $D, ini, fim, p$ )
2   Seja  $meio$  a palavra situada na metade de  $D$ 
3   se  $ini \leq fim$  então
4     se  $meio = p$  então
5       retorne Achei a palavra!
6     senão
7       se  $meio > p$  então
8         A palavra deve estar na primeira metade de  $D$ 
9         retorne BuscarDicionário ( $D, ini, meio - 1, p$ )
10      senão
11        A palavra deve estar na segunda metade de  $D$ 
12        retorne BuscarDicionário ( $D, meio + 1, fim, p$ )
13    senão
14      retorne A palavra não está no dicionário...
```

O "pulo do gato"

- A ideia de dividir o universo de busca em duas partes reduz significativamente a complexidade da busca.
- A cada decisão, reduzimos a nossa área de busca em 50%



Busca binária

- Essa estratégia de dividir a lista em duas partes até localizar o elemento (ou descobrir que ele não se encontra) é chamada de busca binária.
- binária → dois elementos
- Busca binária → uma busca baseada na divisão da lista em duas partes. A cada divisão, uma das partes é descartada.
- A busca binária é um exemplo de algoritmo baseado em divisão e conquista (técnica de projeto de algoritmos):
 - Dividir o universo de busca até resolver o problema

4 - Busca Binária em Lista Ordenada

- Vamos retornar ao problema da busca em uma lista ordenada
- Até agora percebemos que somente a ordenação da lista não melhora a complexidade do nosso algoritmo de busca.
- Mas pensamos em uma nova estratégia: Busca Binária
 - Dividir a lista em duas partes
 - Repetir essa divisão até localizar o elemento
 - (ou descobrir que ele não está na lista)

4 - Busca Binária em Listas Ordenadas

Algoritmo: Busca por um elemento em uma lista ordenada

Entrada: Lista ordenada L e Elemento procurando e

Saída: Elemento localizado ou não?

```
1  Função Buscar( $D, ini, fim, p$ )
2      se  $ini \leq fim$  então
3           $meio = \lfloor (fim + ini)/2 \rfloor$ 
4          se  $L[meio] = p$  então
5              retorne Achei o elemento na posição  $meio$ 
6          senão
7              se  $L[meio] > e$  então
8                  O elemento deve estar na primeira metade de  $L$ 
9                  retorne Buscar ( $L, ini, meio - 1, e$ )
10             senão
11                 O elemento deve estar na segunda metade de  $L$ 
12                 retorne Buscar ( $L, meio + 1, fim, e$ )
13         senão
14             retorne O elemento não está na lista...
```

4 - Busca Binária em Listas Ordenadas

- Pense em como adaptar o algoritmo anterior para retornar também a posição onde um elemento com o ID buscado deve ser inserido na lista

4 - Busca Binária em Listas Ordenadas

```
int BuscaBinaria(elemento L[], int ini, int fim, int id) {  
    if (ini >= fim) return ini;  
    else  
    {  
        int meio = (fim + ini) / 2;  
        if (L[meio].ID == id) return meio;  
        else  
        {  
            if (L[meio].ID > id)  
                return BuscaBinaria(L, ini, meio, id); //está antes do meio  
            else  
                return BuscaBinaria(L, meio+1, fim, id); //está depois do meio  
        }  
    }  
}
```

4 - Busca Binária em Listas Ordenadas

```
int BuscaBinaria(elemento L[], int ini, int fim, int id)
{
    if (ini >= fim)
        return ini;
    int meio = (fim + ini) / 2;
    if (L[meio].ID == id)
        return meio;
    if (L[meio].ID > id)
        return BuscaBinaria(L, ini, meio, id); //está antes do meio
    else
        return BuscaBinaria(L, meio+1, fim, id); //está depois do meio
}
```

4 - Busca Binária em Listas Ordenadas

- E a complexidade dessa busca?
- Qual é o pior caso? Não encontrar o elemento que procuramos
- A chave para descobrir a complexidade desse algoritmo é observar que a lista em uma recursão tem **metade do tamanho da anterior**
- $$T(n) = \begin{cases} 1 + T(n/2), & \text{se } n > 1 \\ 1, & \text{se } n = 1 \end{cases}$$

4 - Busca Binária em Listas Ordenadas

- $T(n) = \begin{cases} 1 + T(n/2), & \text{se } n > 1 \\ 1, & \text{se } n = 1 \end{cases}$
- Quantas vezes podemos dividir uma lista de tamanho n ao meio até atingir uma lista de tamanho 1 ($n = 1$)?

Iteração do Algoritmo	Dimensão da Lista	
Primeira	n	$n/2^0$
Segunda	$\lfloor n / 2 \rfloor$	$n/2^1$
Terceira	$\lfloor \lfloor n / 2 \rfloor / 2 \rfloor$	$n/2^2$
Quarta	$\lfloor \lfloor \lfloor n / 2 \rfloor / 2 \rfloor / 2 \rfloor$	$n/2^3$
...		
Última iteração	1	$n/2^d = 1$

4 - Busca Binária em Listas Ordenadas

Iteração do Algoritmo	Dimensão da Lista	
Primeira	n	$n/2^0$
Segunda	$\lfloor n / 2 \rfloor$	$n/2^1$
Terceira	$\lfloor \lfloor n / 2 \rfloor / 2 \rfloor$	$n/2^2$
Quarta	$\lfloor \lfloor \lfloor n / 2 \rfloor / 2 \rfloor / 2 \rfloor$	$n/2^3$
...		
Última iteração	1	$n/2^d = 1$

$$\frac{n}{2^d} = 1$$

$$2^d = n$$

$$d = \log_2 n$$

- No pior caso, vamos precisar de d comparações para encontrar o elemento.
- Logo, a complexidade da busca binária é $O(\log_2 n)$

4 - Busca Binária em Listas Ordenadas

- Vamos comparar o número de comparações realizadas pela busca sequencial - $O(n)$ com a busca binária - $O(\log_2 n)$

Tamanho da Lista	Busca Sequencial	Busca Binária
16	16	4
128	128	7
1.024	1.024	10
1.000.000	1.000.000	20
1.000.000.000	1.000.000.000	30
1.000.000.000.000	1.000.000.000.000	40

- A busca binária é muito mais eficiente do que a busca sequencial.

4 - Busca em listas sequenciais ordenadas

- Existem algumas estratégias possíveis para realizar a busca em listas ordenadas.
 - Busca ordenada ✓
 - Busca binária ✓

Listas de Alocação Sequencial Ordenadas

- Assim como nas listas não ordenadas, os dados são armazenados como elementos de um vetor.
- Precisamos identificar as operações que podem ser realizadas em listas ordenadas:
 1. Como definir a ordem entre dois elementos ✓
 2. Como definir a lista (o vetor subjacente) ✓
 3. Como verificar o número de elementos contidos na lista ✓
 4. Como buscar por um determinado elemento na lista ✓
 5. Como inserir um novo elemento na lista
 6. Como remover um elemento na lista
 7. Como alterar um elemento da lista

5- Inserção em Listas Sequenciais Ordenadas

- Agora que temos um algoritmo de busca capaz de identificar se o elemento está na lista e, caso ele não esteja, onde ele deve ser inserido.
- Podemos continuar no algoritmo de inserção.
- Mas precisamos avaliar como tratar elementos duplicados:
 - Vamos inserir na posição correta, ao lado dos elementos de mesma chave?
 - **Ou vamos abortar essa inserção?**

5- Inserção em Listas Sequenciais Ordenadas

- Nosso algoritmo de inserção precisa de três etapas:
 1. Verificar e tratar os caso de lista cheia e lista vazia
 2. Buscar pelo elemento que desejamos inserir e tratar a resposta da busca
 3. Caso o elemento não esteja na lista, inserir na posição indicada
- Vamos começar pelas duas primeiras partes.

5 - Inserção em Listas Sequenciais Ordenadas

```
int Inserir(elemento L[], int *nElementos)
{
    int id;
    scanf("%d",&id);
    if (*nElementos >= TAM) //Lista cheia
    {
        printf("Lista cheia\n");
        return 0;
    }
    if (*nElementos == 0) //Lista vazia
    {
        L[0].ID = id;
        //Inserir demais campos
        *nElementos = *nElementos + 1;
        return 1;
    }
}
```


5 - Inserção em Listas Sequenciais Ordenadas

```
int pos = BuscaBinaria(L,0,*nElementos-1,id);
if (pos >= *nElementos)
{
    //0 elemento não está na lista e devemos inserir na última posição
    L[*nElementos].ID = id;
    //Inserir demais campos
    *nElementos = *nElementos + 1;
    return 1;
}
else
```

5 - Inserção em Listas Sequenciais Ordenadas

```
else
{
    if (L[pos].ID == id)
    {
        printf("O elemento já está na lista\n");
        return 0;
    }
    else
    {
        //Vamos inserir na posição pos
        //Como efetuar a inserção???
    }
}
```

Como efetuar a inserção nesse caso?

- Considere a seguinte lista:

1	3	5	7	19	23	42	50		
---	---	---	---	----	----	----	----	--	--

- Inserir elemento com chave 4
 - `BuscaBinaria(L,0,8,4) //(Lista,Ini,Fim,Chave)`

Como efetuar a inserção nesse caso?

- Considere a seguinte lista:



- Inserir elemento com chave 4
 - `BuscaBinaria(L, 0, 8, 4)` //(Lista, Ini, Fim, Chave) => 2
- Não podemos somente atribuir o novo elemento nessa posição...
- Precisamos deslocar todos os elementos dessa posição até o final da lista uma posição na lista: `AbrirEspaco`

Inserção em listas sequenciais ordenadas

```
void AbrirEspaco(elemento L[], int nElementos,  
                                     int pos)  
{  
    for (int i = nElementos; i > pos; i--)  
    {  
        L[i].ID = L[i-1].ID;  
        //Copiar demais campos da estrutura  
    }  
}
```

Inserção em listas sequenciais ordenadas

- Algoritmo de AbrirEspaco:

1	3	5	7	19	23	42	50		
---	---	---	---	----	----	----	----	--	--

- AbrirEspaco(L, 8, 2):

Inserção em listas sequenciais ordenadas

- Algoritmo de AbrirEspaco:



- AbrirEspaco(L, 8, 2):
 - Lista[8] = Lista[7]

Inserção em listas sequenciais ordenadas

- Algoritmo de AbrirEspaco:



- AbrirEspaco(L, 8, 2):
 - Lista[8] = Lista[7]
 - Lista[7] = Lista[6]

Inserção em listas sequenciais ordenadas

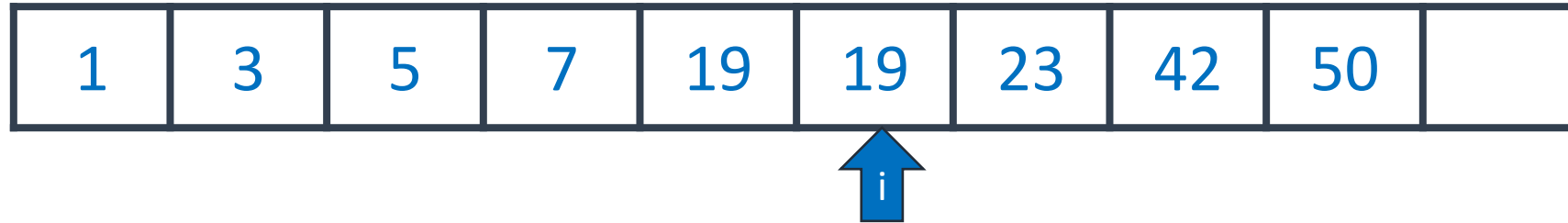
- Algoritmo de AbrirEspaco:



- AbrirEspaco(L, 8, 2):
 - Lista[8] = Lista[7]
 - Lista[7] = Lista[6]
 - Lista[6] = Lista[5]

Inserção em listas sequenciais ordenadas

- Algoritmo de AbrirEspaco:

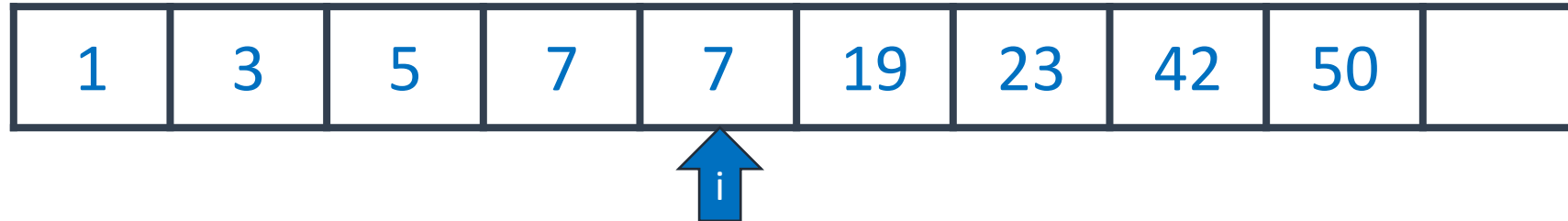


- AbrirEspaco(L, 8, 2):

- Lista[8] = Lista[7]
- Lista[7] = Lista[6]
- Lista[6] = Lista[5]
- Lista[5] = Lista[4]

Inserção em listas sequenciais ordenadas

- Algoritmo de AbrirEspaco:



- AbrirEspaco(L, 8, 2):
 - Lista[8] = Lista[7]
 - Lista[7] = Lista[6]
 - Lista[6] = Lista[5]
 - Lista[5] = Lista[4]
 - Lista[4] = Lista[3]

Inserção em listas sequenciais ordenadas

- Algoritmo de AbrirEspaco:



- AbrirEspaco(L, 8, 2):

- Lista[8] = Lista[7]
- Lista[7] = Lista[6]
- Lista[6] = Lista[5]
- Lista[5] = Lista[4]
- Lista[4] = Lista[3]
- Lista[3] = Lista[2]

5 - Inserção em Listas Sequenciais Ordenadas

- Algoritmo de AbrirEspaco:

1	3	5	5	7	19	23	42	50	
---	---	---	---	---	----	----	----	----	--

- AbrirEspaco(L, 8, 2):

- Lista[8] = Lista[7]
- Lista[7] = Lista[6]
- Lista[6] = Lista[5]
- Lista[5] = Lista[4]
- Lista[4] = Lista[3]
- Lista[3] = Lista[2]

5 - Inserção em Listas Sequenciais Ordenadas

- Algoritmo de AbrirEspaco:

1	3	5	5	7	19	23	42	50	
---	---	---	---	---	----	----	----	----	--

- Observe que a posição que queremos ocupar com o novo elemento agora pode receber o valor sem perder o elemento que estava armazenado anteriormente.

5- Inserção em Listas Sequenciais Ordenadas

```
else
{
    if (L[pos].ID == id)
    {
        printf("O elemento já está na lista\n");
        return 0;
    }
    else
    {
        //Vamos inserir na posição pos
        AbrirEspaco(L, *nElementos, pos);
        L[pos].ID = id;
        //Inserir demais campos
        *nElementos = *nElementos + 1;
        return 1;
    }
}
```

Saídas da inserção

- A lista atualizada (passagem por referência)
- O número de posições ocupadas (passagem por referência)
- Um retorno para indicar se a inserção foi realizada?
 - Se 1, inserção OK
 - Se 0, erro na inserção (p.ex. lista cheia, elemento duplicado).

Listas de Alocação Sequencial Ordenadas

- Assim como nas listas não ordenadas, os dados são armazenados como elementos de um vetor.
- Precisamos identificar as operações que podem ser realizadas em listas ordenadas:
 1. Como definir a ordem entre dois elementos ✓
 2. Como definir a lista (o vetor subjacente) ✓
 3. Como verificar o número de elementos contidos na lista ✓
 4. Como buscar por um determinado elemento na lista ✓
 5. Como inserir um novo elemento na lista ✓
 6. Como remover um elemento na lista
 7. Como alterar um elemento da lista

6 - Remoção em listas sequenciais ordenadas

- A remoção de elementos em listas sequencias pode ser realizada:
 - Para remover sempre um elemento de uma posição específica:
 - Se remover sempre o último elemento: comportamento de pilha
 - Pilha: Último a entrar – Primeiro a sair
 - Se remover sempre o primeiro elemento: comportamento de fila
 - Fila: Primeiro a entrar – Primeiro a sair
 - Remover um determinado elemento (por exemplo: remover o elemento de chave 123).
 - Como pilhas e filas serão tema de aulas futuras, vamos considerar apenas o último caso.

6 - Remoção em listas sequenciais ordenadas

- A remoção de uma lista requer primeiro identificar o elemento que queremos remover.
- Para isso, precisamos usar a busca.
 - Se a busca retornar um índice: remover o elemento nessa posição
 - Se a busca retornar -1: o elemento não pode ser removido da lista (ele não está contido na lista).
- Desta forma:
 - Realizar busca
 - Remover o elemento que a busca apontou

6 - Remoção em listas sequenciais ordenadas

- Problemas na remoção:
 - Considere a seguinte lista:

1	3	3	7	19	23	42	47	53	68
---	---	---	---	----	----	----	----	----	----

- Remover o elemento 3.
 - Mas qual 3? Existem dois!
- Como lidar com elementos duplicados?
 1. Remover o primeiro elemento encontrado
 2. **Não permitir inserções de elementos duplicados** (mesma chave)

6 - Remoção em listas sequenciais ordenadas

- Problemas na remoção:
 - Considere a seguinte lista:

1	3	5	7	19	23	42	47	53	68
---	---	---	---	----	----	----	----	----	----

- Remover o elemento 1.
 - OK! Só tem um.

	3	5	7	19	23	42	47	53	68
--	---	---	---	----	----	----	----	----	----

Ficou um “buraco” na lista.
Como lidar com esses buracos?

6 - Remoção em listas sequenciais ordenadas

- Problemas na remoção:

	3	3	7	19	23	42	47	53	68
--	---	---	---	----	----	----	----	----	----

- Como fechar o “buraco” que ficou na lista?
- Duas estratégias:
 1. Mover o último elemento para fechar o buraco. Vale observar que as listas não ficam com buracos na última posição. Basta decrementar o contador de elementos na lista.
 2. Todos os demais elementos dão um passo a frente.

6 - Remoção em listas sequenciais ordenadas

- Nas listas não ordenadas usamos a primeira estratégia (mover o último elemento) para lidar com os buracos decorrentes das remoções.
- Mas essa estratégia funciona em listas ordenadas?
 - Não... Mover um elemento do final da lista para uma outra posição do vetor viola a ordenação.

68	3	3	7	19	23	42	47	53	68
----	---	---	---	----	----	----	----	----	----

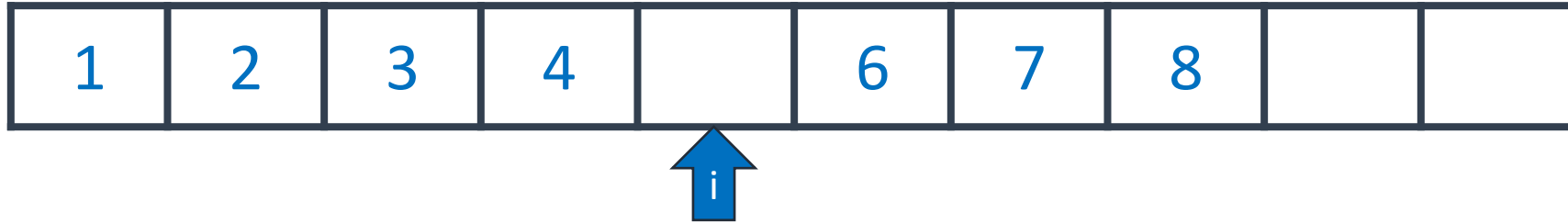
- Vale a pena reordenar o vetor? $O(n \log_2 n)$
- Ou é melhor usar a segunda estratégia? Qual é a complexidade?

6 - Remoção em listas sequenciais ordenadas

```
void FecharEspaco(elemento L[], int nElementos,  
                  int pos)  
{  
    for (int i = pos; i < nElementos - 1; i++)  
    {  
        L[i].ID = L[i+1].ID;  
        //Copiar demais campos da estrutura  
    }  
}
```


6 - Remoção em listas sequenciais ordenadas

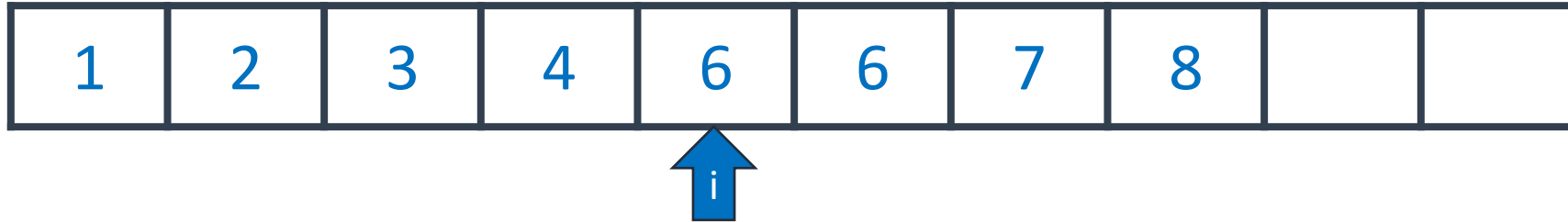
- Algoritmo de FecharEspaco:



- FecharEspaco(L, 8, 4):

6 - Remoção em listas sequenciais ordenadas

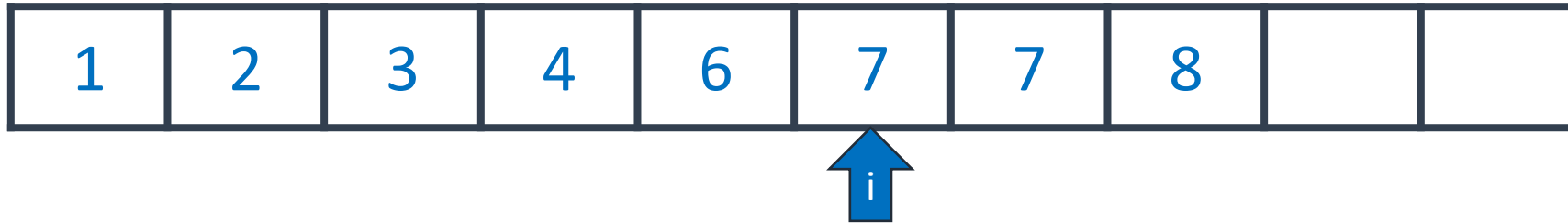
- Algoritmo de FecharEspaco:



- FecharEspaco(L, 8, 4):
 - Lista[4] = Lista[5]

6 - Remoção em listas sequenciais ordenadas

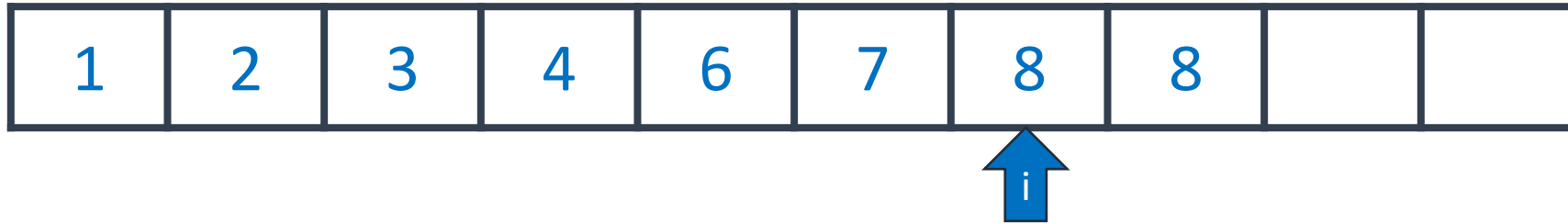
- Algoritmo de FecharEspaco:



- FecharEspaco(L, 8, 4):
 - Lista[4] = Lista[5]
 - Lista[5] = Lista[6]

6 - Remoção em listas sequenciais ordenadas

- Algoritmo de FecharEspaco:



- FecharEspaco(L, 8, 4):
 - Lista[4] = Lista[5]
 - Lista[5] = Lista[6]
 - Lista[6] = Lista[7]

6 - Remoção em listas sequenciais ordenadas

- Algoritmo de FecharEspaco:

1	2	3	4	6	7	8	8		
---	---	---	---	---	---	---	---	--	--

- O “buraco” da inserção foi fechado.
- Falta apenas remover o último elemento
- Assim como nas listas não ordenadas, não precisamos apagar esse elemento fisicamente, basta considerar que a lista tem um elemento a menos (as demais posições contém lixo).

6 - Remoção em listas sequenciais ordenadas

- Qual é a complexidade desse método?
 - Qual é o pior caso?
 - Fechar um buraco na primeira posição.
 - Qual é a operação dominante?
 - Atribuição
 - Quantas atribuições vamos fazer nesse caso?
 - $n - 1$
 - Qual é a complexidade?
 - $O(n)$
- O que é melhor?
 - Usar algoritmo de ordenação: $O(n \log_2 n)$
 - Fechar o buraco com passo a frente: $O(n)$

6 - Remoção em listas sequenciais ordenadas

- Podemos então resumir o processo de remoção em:
 1. Buscar o elemento na lista
 2. Sabendo a posição, guardar/apresentar as informações do elemento
 3. Todos os elementos situados a frente do elemento removido devem ser movidos uma posição a frente para fechar o buraco da remoção
 4. Decrementar o contador de elementos na lista

6 - Remoção em listas sequenciais ordenadas

```
int Remover(elemento L[], int *nElementos, int id)
{
    int pos = BuscaBinaria(lista, 0, *nElementos, chave);
    if (pos < *nElementos && lista[pos].ID == id)
    {
        FecharEspaco(L, *nElementos, pos);
        *nElementos = *nElementos - 1;
        printf("Remoção efetuada com sucesso");
        return 1;
    }
    return 0;
}
```


6 - Remoção em listas sequenciais ordenadas

- Lista L

1	3	7	9	13	24	39	42	53	88
---	---	---	---	----	----	----	----	----	----

- $nElementos = 10$
- $chave = 7$
- 1) Buscar pela chave na lista
 - A busca retorna 2 como resposta (a chave 7 está na posição 2)

6 - Remoção em listas sequenciais ordenadas

- Lista L

1	3	7	9	13	24	39	42	53	88
---	---	---	---	----	----	----	----	----	----



- $nElementos = 10$
- $chave = 7$
- $pos = 2$
- 2) Fechar o buraco:
 - PassoAFrente(L , $nElementos$, pos)

6 - Remoção em listas sequenciais ordenadas

- Lista L

1	3	7	9	13	24	39	42	53	88
---	---	---	---	----	----	----	----	----	----




- $nElementos = 10$
- $chave = 7$
- $pos = 2$
- 2) Fechar o buraco:
 - $PassoAFrente(L, 10, 2)$

6 - Remoção em listas sequenciais ordenadas

- Lista L

1	3	9	13	24	39	42	53	88	88
---	---	---	----	----	----	----	----	----	----



- $nElementos = 10$
- $chave = 7$
- $pos = 2$
- 2) Fechar o buraco:
 - $PassoAFrente(L, 10, 2)$

6 - Remoção em listas sequenciais ordenadas

- Lista L

1	3	9	13	24	39	42	53	88	88
---	---	---	----	----	----	----	----	----	----

- $nElementos = 9$
- $chave = 1$
- $pos = 2$
- 3) Decrementar o contador de elementos na lista
 $*nElementos = *nElementos - 1;$

Listas de Alocação Sequencial Ordenadas

- Assim como nas listas não ordenadas, os dados são armazenados como elementos de um vetor.
- Precisamos identificar as operações que podem ser realizadas em listas ordenadas:
 1. Como definir a ordem entre dois elementos ✓
 2. Como definir a lista (o vetor subjacente) ✓
 3. Como verificar o número de elementos contidos na lista ✓
 4. Como buscar por um determinado elemento na lista ✓
 5. Como inserir um novo elemento na lista ✓
 6. Como remover um elemento na lista ✓
 7. Como alterar um elemento da lista

7- Alteração em listas sequenciais ordenadas

- Assim como a remoção, a alteração requer que o elemento seja localizado na lista.
- Todos os problemas relacionados a chaves duplicadas se aplicam na alteração também.
- Após a busca:
 - Se o elemento não foi localizado: interromper a alteração
 - Alterar a informação desejada do elemento.
- E se a chave for alterada?

7- Alteração em listas sequenciais ordenadas

- A alteração de elementos pode ocasionar alguns problemas:
 - Como tratar chaves repetidas?
 - Impedir a alteração?
 - Permitir chaves duplicadas?
 - Como lidar com a alteração da chave em listas ordenadas?
 - Se a chave for alterada, a ordenação pode ser comprometida.
 - Por exemplo: [1,2,3,4,5]. Suponha que a chave do 2 é alterada para 6. [1,6,3,4,5]
 - Nesses casos, um tratamento comum consiste em:
 1. Remover o elemento: $O(n)$
 2. Inserir novamente, com a chave correta: $O(n)$

Listas de Alocação Sequencial Ordenadas

- Assim como nas listas não ordenadas, os dados são armazenados como elementos de um vetor.
- Precisamos identificar as operações que podem ser realizadas em listas ordenadas:
 1. Como definir a ordem entre dois elementos ✓
 2. Como definir a lista (o vetor subjacente) ✓
 3. Como verificar o número de elementos contidos na lista ✓
 4. Como buscar por um determinado elemento na lista ✓
 5. Como inserir um novo elemento na lista ✓
 6. Como remover um elemento na lista ✓
 7. Como alterar um elemento da lista — Tarefa para a aula prática

Listas Sequenciais

Operação	Listas Não Ordenadas	Listas Ordenadas
Busca	$O(n)$: Busca sequencial	$O(\log_2 n)$: Busca binária
Inserção – Sem chaves repetidas	$O(n)$ – Busca para identificar se o elemento já está na lista $O(1)$ – Para inserir Total: $O(n)$	$O(\log_2 n)$ – Busca para identificar se o elemento já está na lista $O(n)$ – Para inserir Total: $O(n)$
Inserção – Com chaves repetidas	$O(1)$ – Inserção na primeira posição livre no final da lista	$O(\log_2 n)$ – Busca para a posição correta da inserção $O(n)$ – Para inserir Total: $O(n)$
Remoção	$O(n)$ – Busca para identificar a posição do elemento (e se ele está na lista) $O(1)$ – Para remover Total: $O(n)$	$O(\log_2 n)$ – Busca para identificar a posição do elemento (e se ele está na lista) $O(n)$ – Para remover Total: $O(n)$

- Como optar pela lista mais adequada?

Como optar pela lista mais adequada?

- Se a lista sofre mais alterações (inserções e remoções) do que buscas:
 - Inserção em lista não ordenada: $O(1)$
 - Inserção em lista ordenada: $O(n)$
 - Remoção não apresenta diferenças nos dois casos: $O(n)$
 - Melhor usar listas não ordenadas nesse caso.
- Se a lista passa por mais operações de busca do que alterações:
 - Busca em lista não ordenada: $O(n)$
 - Busca em lista ordenada: $O(\log_2 n)$
 - Melhor usar listas ordenadas nesse caso.

Exercícios

- Repetir o mesmo exercício da aula anterior, usando uma lista ordenada pelo ID.
 - Implemente um sistema de controle de funcionários de uma empresa.
 - A empresa deseja armazenar os seguintes dados de cada funcionário:
 - ID
 - Nome
 - DataNascimento
 - Salario
 - CargaHoraria
- Implemente um sistema para cadastrar funcionários, remover funcionários do cadastro, buscar um funcionário no cadastro e oferecer um aumento para um funcionário em específico (pelo ID do mesmo).
- Implemente um Menu questionando qual operação deve ser realizada. Programe também uma opção SAIR.