Estruturas de Dados - Lista Sequencial

Estruturas de dados

- Programas operam sobre dados
- Dados, em sua maioria, são relacionados e possuem uma estrutura
- Estruturas de dados: apresentar novas maneiras de manipular dados em um computador
- O uso de uma estrutura de dados permite fazer diferentes operações sobre um conjunto de dados

Estruturas de dados

- Exemplo: baralho
- Cada carta tem duas informações: um naipe e um valor
- Exemplos de operações sobre um baralho:
  - ordenar o baralho
  - buscar uma carta no baralho
- Também podemos executar operações sobre o conjunto de cartas
  - comprar uma carta do topo do baralho
  - colocar uma carta no fundo do baralho
  - pegar uma carta aleatória do baralho
  - embaralhar

Estruturas de dados

- Estruturas de dados tem vantagens e desvantagens
- A "melhor" estrutura depende da situação
  - precisamos analisar as operações que vamos realizar sobre os dados, para escolhermos a estrutura a ser utilizada
- Estruturas básicas:
  - lista linear
  - pilha
  - fila
  - árvore
- Obs: nesta disciplina, trabalharemos com estruturas contendo números!

Lista

- Tipo básico de estrutura de dados
- Ordem de inserção dos elementos não é importante
  - não importa se um elemento inserido vem antes ou depois de outro
  - o importante é garantir que um elemento inserido continue na lista até ser removido (ou seja, não haja perca de dados)
- Operações básicas: criar, buscar, incluir, excluir
- Há duas formas básicas de armazenamento dos elementos na memória
  - alocação sequencial (vetor)
  - alocação encadeada (nós)

Lista

Operações a serem analisadas em uma lista:

- Criar:
  - criar uma lista inicialmente vazia
- Buscar:
  - buscar um elemento em uma lista
  - obs: deve retornar algo que indique se o elemento está presente ou não na lista
- Incluir:
  - inserir um elemento em uma lista
- Excluir:
  - excluir um elemento de uma lista (caso ele esteja presente na lista)

Tipos de alocação

Alocação sequencial:

- Utiliza vetor
- Elementos estão dispostos em posições consecutivas de memória
- Elementos estão dispostos na memória respeitando uma ordem entre eles

Alocação encadeada:

- Utiliza uma sequência de nós ligados (encadeados)
- Elementos estão dispostos em posições arbitrárias de memória
- A posição ocupada por um elemento na memória não guarda relação com o seu índice na estrutura

Lista sequencial

- Elementos dispostos em posições consecutivas de memória
- Para todo índice i, a posição de memória ocupada pelo i-ésimo elemento pode ser obtida em tempo constante
  - leva em conta a posição do primeiro elemento, o tamanho de cada elemento, e o próprio índice i
- A alocação sequencial exige a reserva de um espaço de memória para o armazenamento dos seus elementos
- Esse espaço define a quantidade máxima de elementos que o vetor pode conter

Lista sequencial

- Composição de um vetor:
  - *capacidade*: número máximo de elementos do vetor
  - *tamanho*: número de elementos presentes no vetor
- Obs: consideramos que nosso vetor tem campos *capacidade* e *tamanho*, como explicados acima
- Operações básicas:
  - criar lista
  - buscar em uma lista
  - incluir um elemento em uma lista
  - excluir um elemento de uma lista

Lista sequencial: criar

Algoritmo: CriarListaSequencial(n)

Entrada: capacidade n da lista a ser criada

Saída: lista sequencial criada

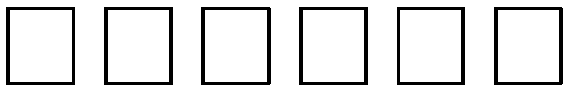
1 criar nova lista sequencial L com um vetor de n posições $(0..n-1)$

2 $L.capacidade = n$

3 $L.tamanho = 0$

4 **retorne** L

Complexidade: $O(1)$



Lista sequencial: buscar

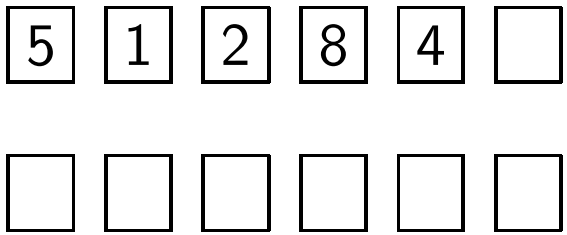
Algoritmo: BuscarEmListaSequencial(L, x)

Entrada: lista sequencial L, valor x

Saída: índice de x na lista, ou -1 se x não pertence à lista

```
1 i = 0
2 enquanto i < L.tamanho faça
3 |   se L[i] == x então
4 |   |   retorne i
5 |   i = i + 1
6 retorne −1
```

Complexidade: $O(n)$ (onde n é a quantidade de elementos na lista)



Lista sequencial: incluir

Algoritmo: IncluirEmListaSequencial(L, x)

Entrada: lista sequencial L, valor x

Saída: tamanho da lista após a inserção

1 $L[L.tamanho] = x$

2 $L.tamanho = L.tamanho + 1$

3 **retorne** $L.tamanho$

Complexidade: $O(1)$

Lista sequencial: incluir

Algoritmo: IncluirEmListaSequencial(L, x)

Entrada: lista sequencial L, valor x

Saída: tamanho da lista após a inserção

1 $L[L.tamanho] = x$

2 $L.tamanho = L.tamanho + 1$

3 **retorne** $L.tamanho$

Problema: e se a lista estiver cheia?

Em outras palavras: e se o vetor não tiver mais posições livres?

Lista sequencial: incluir (corrigido)

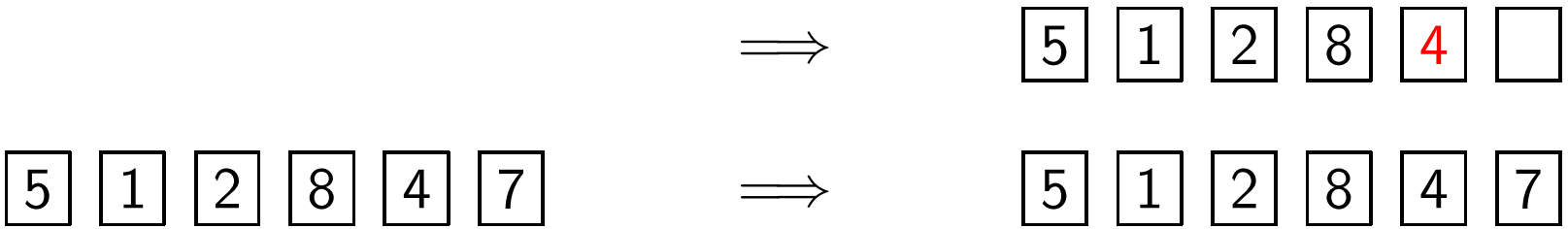
Algoritmo: IncluirEmListaSequencial(L, x)

Entrada: lista sequencial L, valor x

Saída: tamanho da lista após a inserção, ou -1 se a lista estiver cheia

```
1 se L.tamanho < L.capacidade então
2 |   L[L.tamanho] = x
3 |   L.tamanho = L.tamanho + 1
4 |   retorne L.tamanho
5 retorne −1
```

Complexidade: $O(1)$



Lista sequencial: excluir

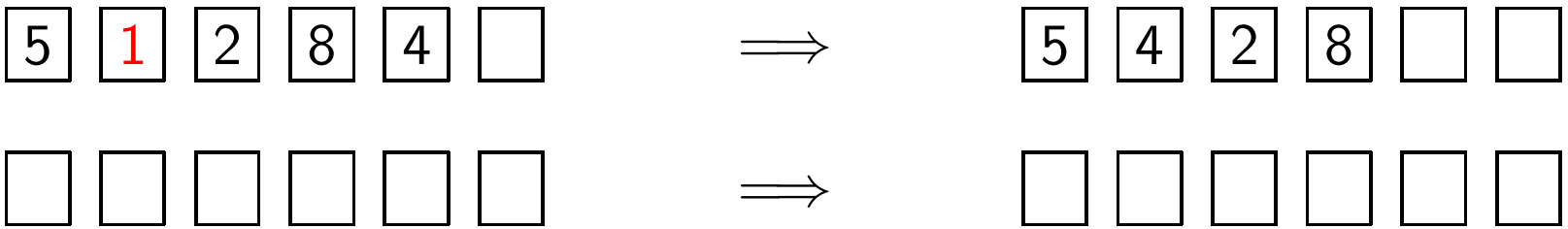
Algoritmo: ExcluirEmListaSequencial(L, x)

Entrada: lista sequencial L, valor x

Saída: tamanho da lista após a remoção, ou -1 se x não pertence à lista

```
1 i = BuscarEmListaSequencial(L, x)
2 se i ≥ 0 então
3 |   L.tamanho = L.tamanho − 1
4 |   L[i] = L[L.tamanho]
5 retorne i
```

Complexidade: $O(n)$ (onde n é a quantidade de elementos na lista)



Lista encadeada

- Elementos dispostos em posições arbitrárias de memória
- A posição ocupada por um elemento na memória não guarda relação com o seu índice na lista
- A ordem dos elementos na lista é estabelecida pelo uso de nós para o armazenamento dos elementos
- Um nó v é uma estrutura com as seguintes informações:
  - $v.chave$: guarda o elemento
  - $v.prox$: indica a localização do elemento que o sucede na lista

Lista encadeada

- Vamos considerar a existencia de um nó especial chamado **sentinela** e denotado por λ, indicando a ausência do elemento
  - se $v.prox = \lambda$, então v é o último nó da lista
- Para passarmos uma lista encadeada como parâmetro, basta passarmos o primeiro nó da lista (ou nó cabeça)
- Além disso, trabalharemos com a ideia de que o nó cabeça de uma lista é um nó vazio
  - ou seja, uma lista começa a partir do nó cabeça, mas excluindo-o
  - isto facilita a escrita de alguns métodos em estruturas encadeadas

Lista encadeada: exemplo

Algoritmo: MaiorListaEncadeada(v)

Entrada: nó inicial v

Saída: nó de maior chave na lista encadeada, ou λ se a lista estiver vazia

```
1 se v.prox == λ então
2 |   retorne λ
3 m = v.prox
4 enquanto v.prox ≠ λ faça
5 |   v = v.prox
6 |   se v.chave > m.chave então
7 |   |   m = v
8 retorne m
```

Complexidade: $O(n)$ (onde n é a quantidade de elementos na lista)

Exemplo: buscar o nó de maior chave em uma lista encadeada