Estrutura de Dados

Rafael D. Ribeiro, M.Sc.
rafaeldiasribeiro@gmail.com
http://www.rafaeldiasribeiro.com.br

Estrutura de Dados

Alocação Encadeada

- Os elementos podem ocupar quaisquer células de memória (não necessariamente consecutivas) e, para manter a relação de ordem linear, juntamente com cada elemento é armazenado o endereço do próximo elemento da lista.

- Os elementos são armazenados em blocos de memória denominados **nodos**, sendo que cada nodo é composto por dois campos: um para armazenar dados e outro para armazenar endereço.

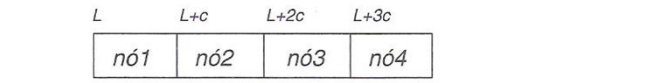
Estrutura de Dados

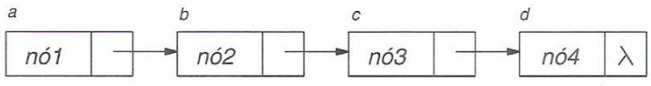
Alocação Encadeada

Endereço	Conteúdo		
L=3FFA	a_1	1C34	←Primeiro elemento, acessível a partir de L
1C34	a_2	BD2F	←Note que o segundo não ocupa o endereço consecutivo à a_1
BD2F	a_3	AC12	
	...		
1000	a_{n-2}	3A7B	← Cada nodo armazena um elemento e um endereço do próximo elemento da lista
	...		
14F6	a_{n-1}	5D4A	
5D4A	a_n	null	← Último elemento da cadeia, o endereço nulo indica que o elemento não tem um sucessor.

Estrutura de Dados

Alocação Sequencial X Alocação Encadeada





Estrutura de Dados

Alocação Encadeada

- Os nós de uma lista se encontram em posição não obrigatoriamente contíguas de memória. Se nas listas são feitas inserções e remoções, há necessidade de gerenciar (liberar e encontrar) as posições de memória. Para isso é criada uma **L**ista de **E**spaço **D**isponível (LED), que contem posições de memória ainda não utilizadas ou dispensadas após sua utilização.

Estrutura de Dados

Alocação Encadeada

- Implementação da LED
  - No princípio, como a memória se acha totalmente disponível, todos os seus nós são encadeados na LED. A variável ponteiro **vago** se refere ao topo da estrutura. Para inicializar o LED são necessários:
  - PASSO 1: Os campos dos ponteiros dos nós são encadeados sequencialmente
  - PASSO 2: O ponteiro vago é inicializado com o endereço do primeiro nó da lista que foi encadeada no primeiro passo
  - PASSO 3: O campo ponteiro do último nó recebe o valor λ , indicando fim da lista.

Estrutura de Dados

Alocação Encadeada

- Quando se precisa de um novo nó para executar o algoritmo de inserção, deve-se buscá-lo na LED.

algoritmo 2.12: Busca de um elemento na LED

```
procedimento ocupar(pt)
    se vago ≠ λ então
            pt := vago
            vago := M[vago].prox
    senão overflow
```

Estrutura de Dados

Alocação Encadeada

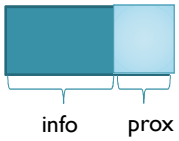
- Quando um nó é dispensado pela remoção, deve-se devolver o nó dispensado para a LED

algoritmo 2.13: Devolução de um nó à LED

```
procedimento desocupar(pt)
    M[pt].prox := vago
    vago := pt
```

Estrutura de Dados

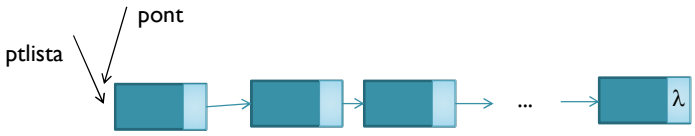
Lista Simplesmente Encadeada



- Impressão da Lista

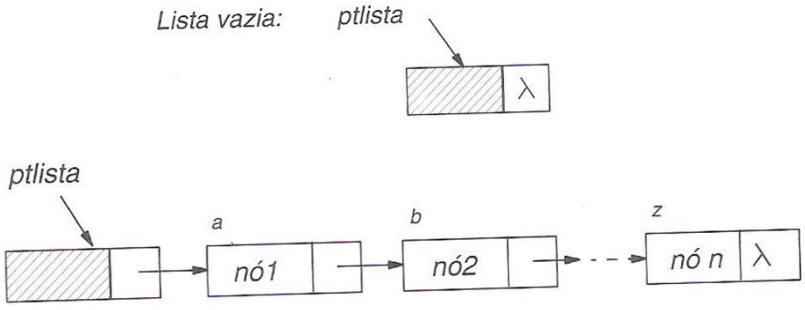
algoritmo 2.14: Impressão da lista apontada por *ptlista*

pont := *ptlista*
enquanto *pont* ≠ λ **faça**
    imprimir(*pont* ↑ .*info*)
    pont := *pont* ↑ .*prox*



Estrutura de Dados

Lista Simplesmente Encadeada



Estrutura de Dados

Lista Simplesmente Encadeada

- Busca em uma Lista Ordenada

```
procedimento busca-enc(x, ant, pont)
    ant := ptlista;   pont := λ
    ptr := ptlista ↑ .prox                    % ptr: ponteiro de percurso
    enquanto ptr ≠ λ faça
          se ptr ↑ .chave < x então
                ant := ptr                    % atualiza ant e ptr
                ptr := ptr ↑ .prox
          senão se ptr ↑ .chave = x então
                      pont := ptr             % chave encontrada
                ptr := λ
```

Estrutura de Dados

Lista Simplesmente Encadeada

- Inserção

```
algoritmo 2.16: Inserção de um nó após o nó apontado por ant
busca-enc(x, ant, pont)
se pont = λ então
      ocupar(pt)                              % solicitar nó
      pt ↑ .info := novo-valor                % inicializar nó
      pt ↑ .chave := x;   pt ↑ .prox := ant ↑ .prox
      ant ↑ .prox := pt                       % acertar lista
senão "elemento já está na tabela"
```

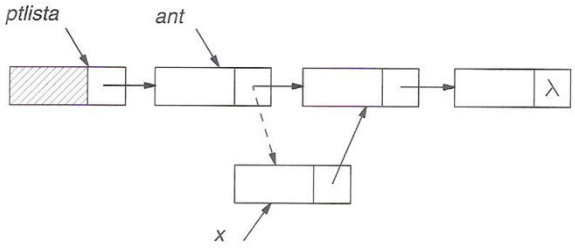
Estrutura de Dados

Lista Simplesmente Encadeada

- Inserção

algoritmo 2.16: Inserção de um nó após o nó apontado por *ant*

```
busca-enc(x, ant, pont)
se pont = λ então
        ocupar(pt)                                  % solicitar nó
        pt↑.info := novo-valor                      % inicializar nó
        pt↑.chave := x;   pt↑.prox := ant↑.prox
        ant↑.prox := pt                             % acertar lista
senão "elemento já está na tabela"
```



Estrutura de Dados

Lista Simplesmente Encadeada

- Remoção

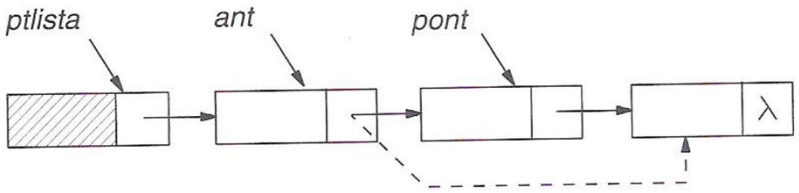
algoritmo 2.17: Remoção do nó apontado por *pont* na lista

```
busca-enc(x, ant, pont)
se pont ≠ λ então
        ant↑.prox := pont↑.prox                     % acertar lista
        valor-recuperado := pont↑.info              % utilizar nó
        desocupar(pont)                             % devolver nó
senão "nó não se encontra na tabela"
```

Estrutura de Dados
Lista Simplesmente Encadeada

- Remoção

algoritmo 2.17: Remoção do nó apontado por *pont* na lista
busca-enc$(x, ant, pont)$
se $pont \neq \lambda$ então
    $ant\uparrow.prox := pont\uparrow.prox$ % acertar lista
    $valor\text{-}recuperado := pont\uparrow.info$ % utilizar nó
    $desocupar(pont)$ % devolver nó
senão "nó não se encontra na tabela"



Estrutura de Dados
Lista Simplesmente Encadeada

- Pilhas

algoritmo 2.18: Inserção na pilha
$ocupar(pt)$ % solicitar nó
$pt\uparrow.info := novo\text{-}valor$ % inicializar nó
$pt\uparrow.prox := topo$
$topo := pt$ % acertar pilha

Estrutura de Dados
Lista Simplesmente Encadeada

- Pilhas

```
algoritmo 2.19: Remoção da pilha
se topo ≠ λ então
        pt := topo                                  % acertar pilha
        topo := topo ↑ .prox
        valor-recuperado := pt ↑ .info              % utilizar nó
        desocupar(pt)                               % devolver nó
senão underflow
```

Estrutura de Dados
Lista Simplesmente Encadeada

- Fila

```
algoritmo 2.20: Inserção na fila
ocupar(pt)                                          % solicitar nó
pt ↑ .info := novo-valor                            % inicializar nó
pt ↑ .prox := λ
se fim ≠ λ então                                    % acertar fila
        fim ↑ .prox := pt
senão inicio := pt
fim := pt
```

Estrutura de Dados

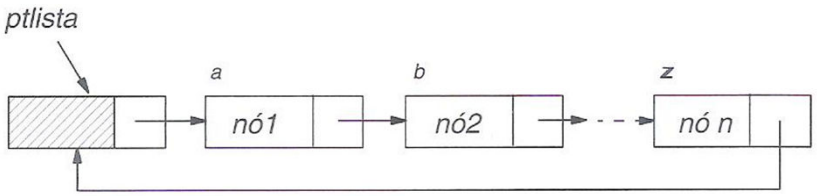
Lista Simplesmente Encadeada

- Fila

```
algoritmo 2.21: Remoção da fila
se inicio ≠ λ então
        pt := inicio
        inicio := inicio ↑ .prox                    % acertar fila
        se inicio = λ então fim := λ
        valor-recuperado := pt ↑ .info              % utilizar nó
        desocupar(pt)                               % devolver nó
senão underflow
```

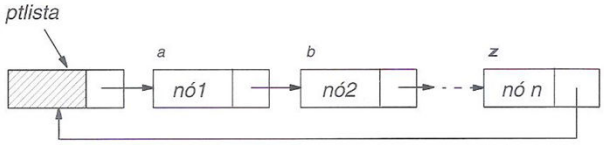
Estrutura de Dados

Lista Circular



Estrutura de Dados

Lista Circular

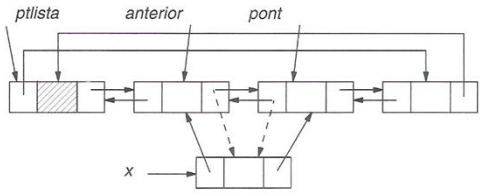


algoritmo 2.23: Busca numa lista circular encadeada ordenada

```
procedimento busca-cir(x, ant, pont)
    ant := ptlista
    ptlista ↑ .chave := x
    pont := ptlista ↑ .prox
    enquanto pont ↑ .chave < x faça
        ant := pont
        pont := pont ↑ .prox
    se pont ≠ ptlista e pont ↑ .chave = x então
        "chave localizada"
    senão "chave não localizada"
```

Estrutura de Dados

Lista Duplamente Encadeada

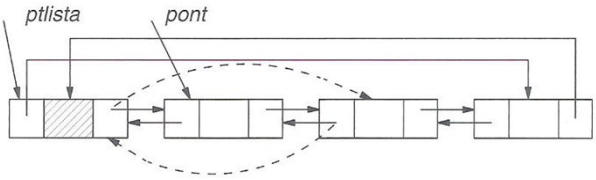


algoritmo 2.25: Inserção de um nó em uma lista duplamente encadeada

```
pont := busca-dup(x)
se pont = ptlista ou pont ↑ .chave ≠ x então
        anterior := pont ↑ .ant
        ocupar(pt)                          % solicitar nó
        pt ↑ .info := novo-valor            % inicializar nó
        pt ↑ .chave := x
        pt ↑ .ant := anterior
        pt ↑ .post := pont
        anterior ↑ .post := pt              % acertar lista
        pont ↑ .ant := pt
senão "elemento já se encontra na lista"
```

Estrutura de Dados

Lista Duplamente Encadeada



algoritmo 2.26: Remoção de um nó em uma lista duplamente encadeada

```
pont := busca-dup(x)
se pont ≠ ptlista e pont ↑ .chave = x então
        anterior := pont ↑ .ant
        posterior := pont ↑ .post
        anterior ↑ .post := posterior          % acertar lista
        posterior ↑ .ant := anterior
        valor-recuperado := pont ↑ .info       % utilizar nó
        desocupar(pont)                        % devolver nó
senão "elemento não se encontra na lista"
```

Estrutura de Dados

Bibliografia das Notas de Aula

Estruturas de Dados e Seus Algoritmos - Jayme Luiz Szwarcfiter, Lilian Markenzon LTC Editora - 2 ° Ed.