Listas Lineares

Notas de aula da disciplina IME 04-10820
ESTRUTURAS DE DADOS I

Paulo Eustáquio Duarte Pinto
(pauloedp arroba ime.uerj.br)

junho/2018

Alocação Encadeada

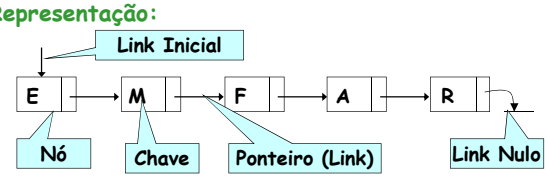
Listas encadeadas:

- Conceitos
- Buscas/Inserções/deleções de elementos
- Pilhas e Filas
- Listas Circulares e Listas Duplamente Encadeadas

Alocação Encadeada

Conceito:
Listas encadeadas são listas onde os elementos consecutivos podem não ocupar posições consecutivas na memória. Então há um ponteiro para o próximo elemento da lista.

Representação:



Alocação Encadeada

Declaração em Tupy:

```
tipo lista:
    caracter c
    lista    prox
```

Declaração em C/C++:

```
typedef struct node *lista;
struct node { char c;  lista prox;}
```

Link Nulo:
nulo (Tupy), NULL (C/C++).

Alocação Encadeada

Alocação de um nó:

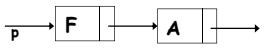
```
Nó p ← Nó()      (Tupy).
lista p = malloc(sizeof *p); (C).
p = new(no);  (C++).
```

Efeito de um comando Alocar:
o Sistema Operacional retira espaço da Pilha de Espaço Disponível (PED) e retorna o endereço.

Desalocação de um nó:
desalocar (p); free(p); (C/C++).

Efeito de um comando Desalocar:
o espaço é retornado para a PED.

Alocação Encadeada



Referência a um elemento de um nó:

```
p.c
p.prox
```

F → A

p->c (C/C++).

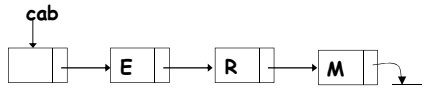
Outras referências:

```
p.prox.c
p.prox.prox
```

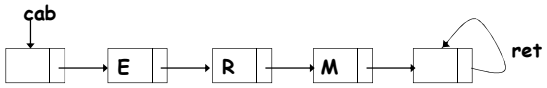
A → A

Alocação Encadeada

Nó cabeça:
É comum utilizar o primeiro nó como nó especial, ou nó cabeça.

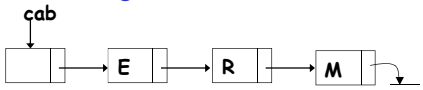


Nó retaguarda:
Às vezes usa-se também um nó especial no final.



Alocação Encadeada

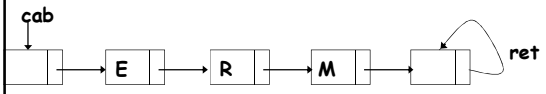
Impressão de uma lista Encadeada com cabeça e sem retaguarda:




```
Imprime():
#Dados: lista cab com nó cabeça
        p ← cab.prox
        enquanto (p ≠ nulo):
                escrever (p.c)
                p ← p.prox
```

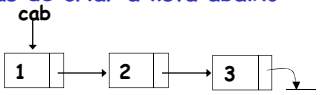

Alocação Encadeada

Exercício: reescrever a impressão de uma lista Encadeada com cabeça e retaguarda:



Alocação Encadeada

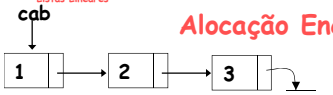
Três maneiras de criar a lista abaixo:




```
Alg1():
    cab ← Nó();    cab.c ← 1;    p ← Nó();  p.c ← 2;  q ← Nó(); q.c ← 3;
    cab.prox ← p;  p.prox ← q;   q.prox ← nulo;
```

```
Alg2():
    cab ← Nó();    cab.c ← 1;    p ← Nó();  p.c ← 2;  cab.prox ← p;
    p ← Nó();      p.c ← 3;      cab.prox.prox ← p;   p.prox ← nulo;
```

```
Alg3():
    cab ← Nó();    p ← cab;   p.c ← 1;    p.prox ← Nó();    p ← p.prox;
    p.c ← 2;  p.prox ← Nó();  p ← p.prox;  p.c ← 3;          p.prox ← nulo;
```

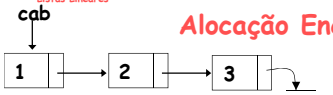



Alocação Encadeada

```
Alg1():
    cab ← Nó();    cab.c ← 1;    p ← Nó();  p.c ← 2;  q ← Nó(); q.c ← 3;
    cab.prox ← p;  p.prox ← q;   q.prox ← nulo;
```

cab p q

1-100	?	?	?			....			
101-200						....			
201-300						....			
301-400						....			
401-500						....			



Alocação Encadeada

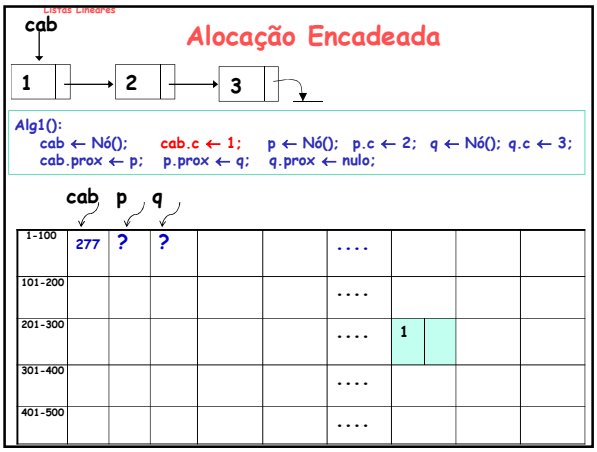
```
Alg1():
    cab ← Nó();    cab.c ← 1;    p ← Nó();  p.c ← 2;  q ← Nó(); q.c ← 3;
    cab.prox ← p;  p.prox ← q;   q.prox ← nulo;
```

cab p q

1-100	277	?	?			....			
101-200						....			
201-300						....			
301-400						....			
401-500						....			

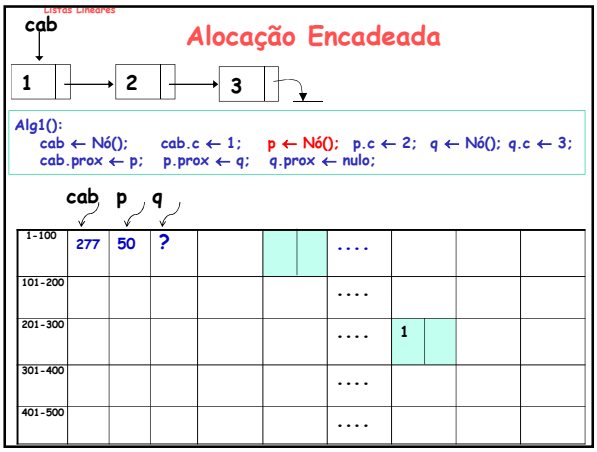
Listas Lineares

Alocação Encadeada



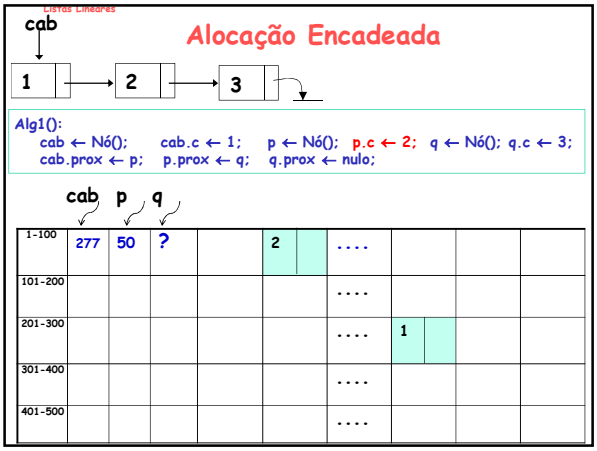
Listas Lineares

Alocação Encadeada



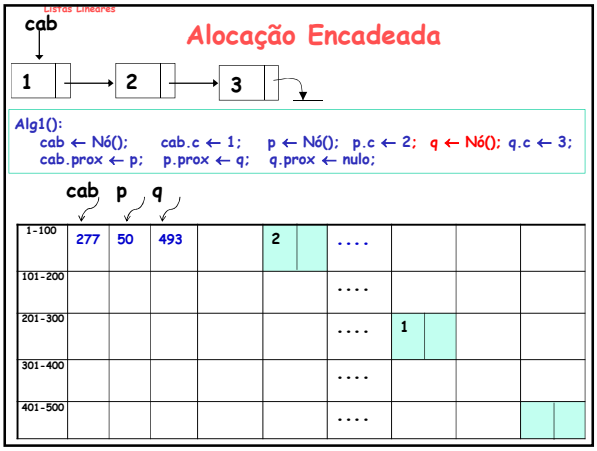
Listas Lineares

Alocação Encadeada



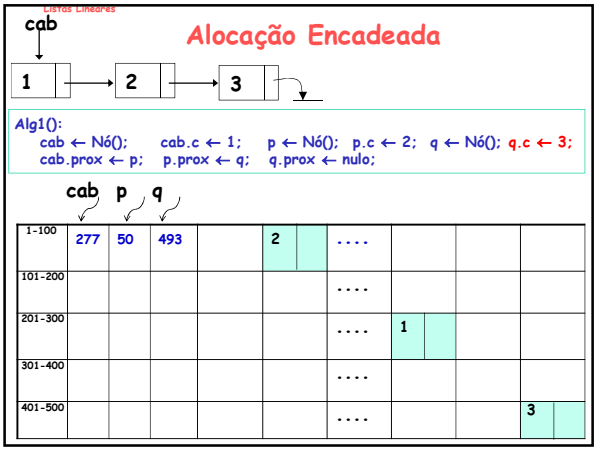
Listas Lineares

Alocação Encadeada



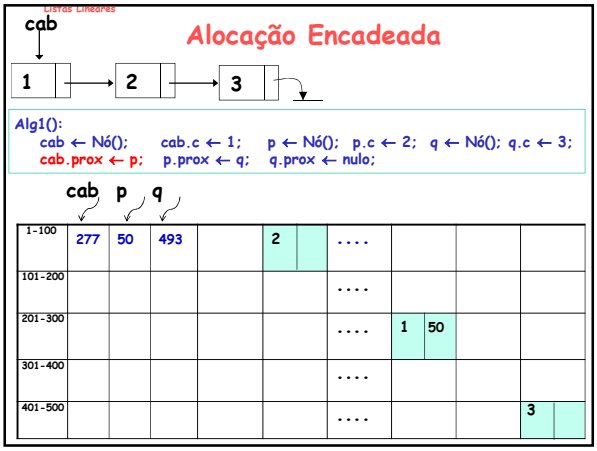
Listas Lineares

Alocação Encadeada



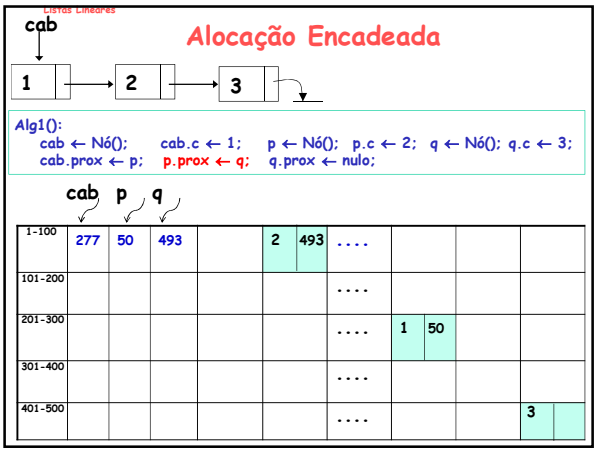
Listas Lineares

Alocação Encadeada



Listas Lineares

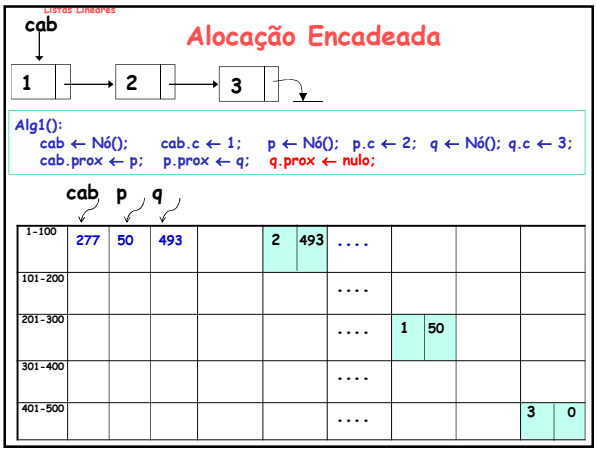
Alocação Encadeada




```
Alg1():
    cab ← Nó();     cab.c ← 1;    p ← Nó();  p.c ← 2;  q ← Nó(); q.c ← 3;
    cab.prox ← p;   p.prox ← q;   q.prox ← nulo;
```


Listas Lineares

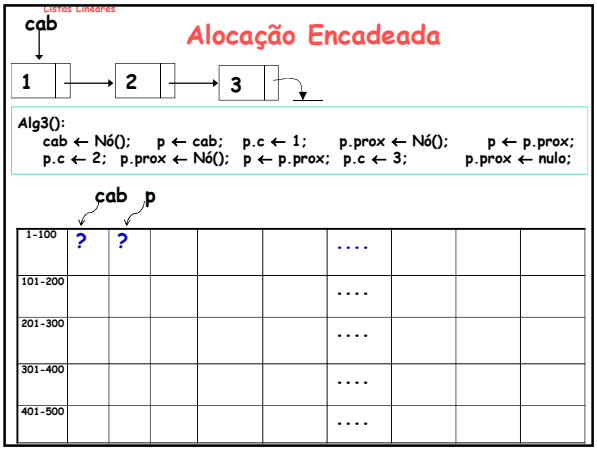
Alocação Encadeada




```
Alg1():
    cab ← Nó();     cab.c ← 1;    p ← Nó();  p.c ← 2;  q ← Nó(); q.c ← 3;
    cab.prox ← p;   p.prox ← q;   q.prox ← nulo;
```


Listas Lineares

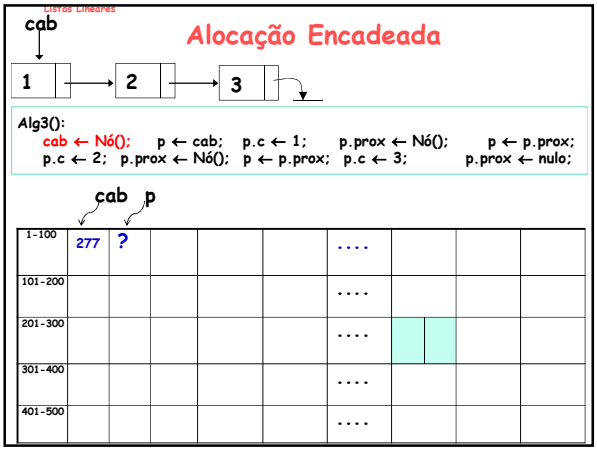
Alocação Encadeada




```
Alg3():
    cab ← Nó();     p ← cab;   p.c ← 1;     p.prox ← Nó();     p ← p.prox;
    p.c ← 2;  p.prox ← Nó();   p ← p.prox;  p.c ← 3;           p.prox ← nulo;
```


Listas Lineares

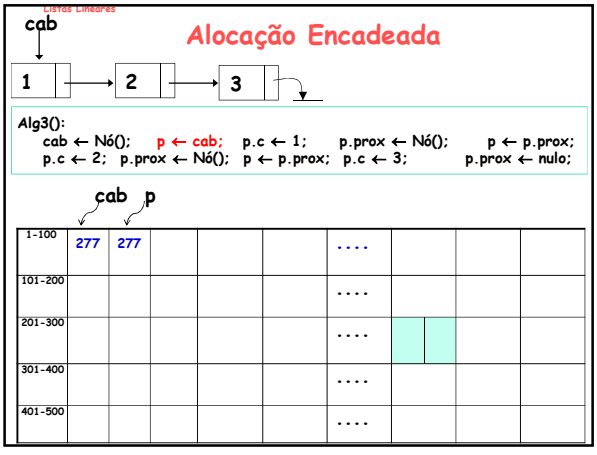
Alocação Encadeada




```
Alg3():
    cab ← Nó();     p ← cab;   p.c ← 1;     p.prox ← Nó();     p ← p.prox;
    p.c ← 2;  p.prox ← Nó();   p ← p.prox;  p.c ← 3;           p.prox ← nulo;
```


Listas Lineares

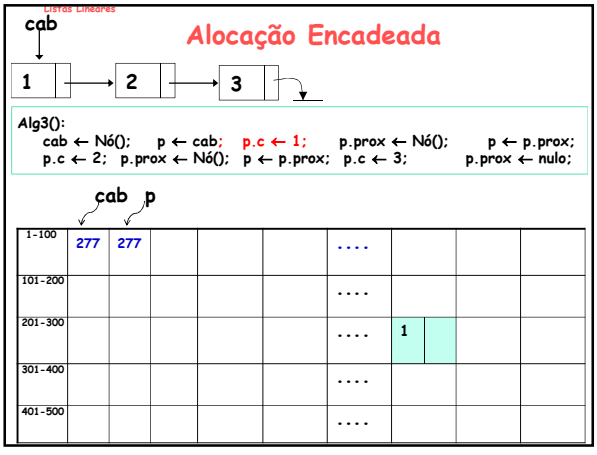
Alocação Encadeada




```
Alg3():
    cab ← Nó();     p ← cab;   p.c ← 1;     p.prox ← Nó();     p ← p.prox;
    p.c ← 2;  p.prox ← Nó();   p ← p.prox;  p.c ← 3;           p.prox ← nulo;
```


Listas Lineares

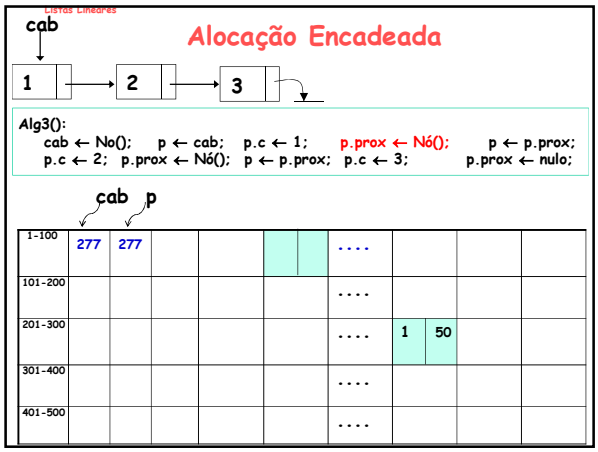
Alocação Encadeada




```
Alg3():
    cab ← Nó();     p ← cab;   p.c ← 1;     p.prox ← Nó();     p ← p.prox;
    p.c ← 2;  p.prox ← Nó();   p ← p.prox;  p.c ← 3;           p.prox ← nulo;
```

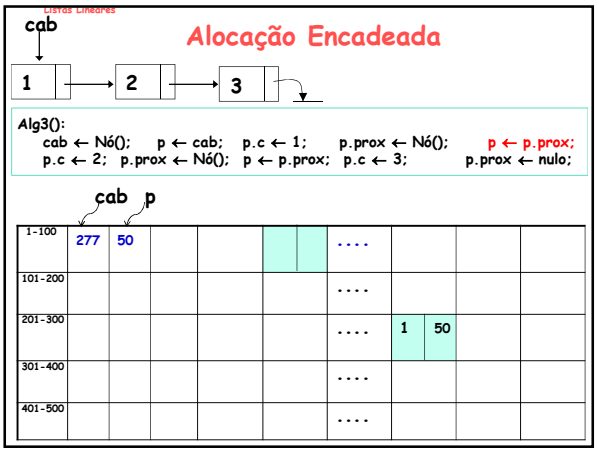
Listas Lineares

Alocação Encadeada



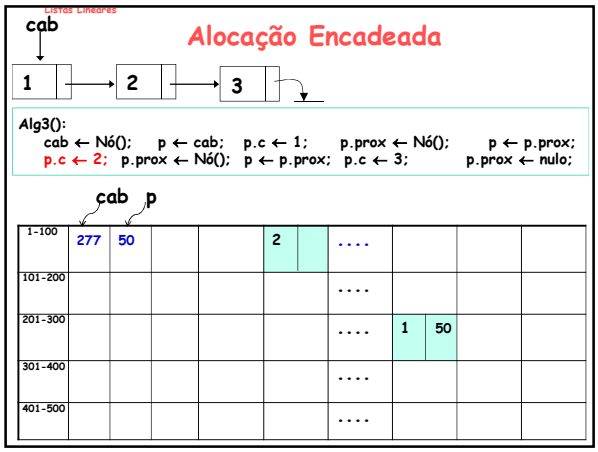
Listas Lineares

Alocação Encadeada



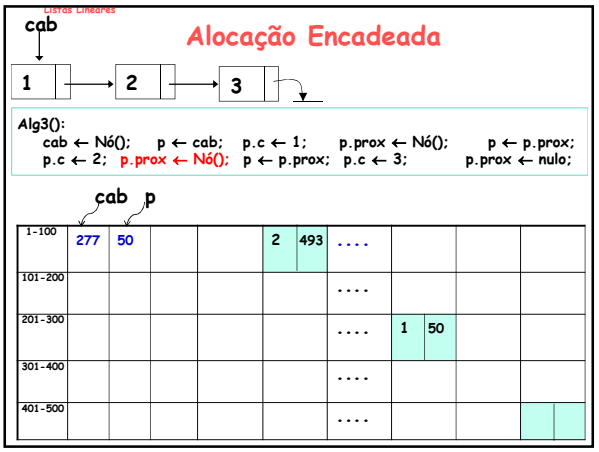
Listas Lineares

Alocação Encadeada



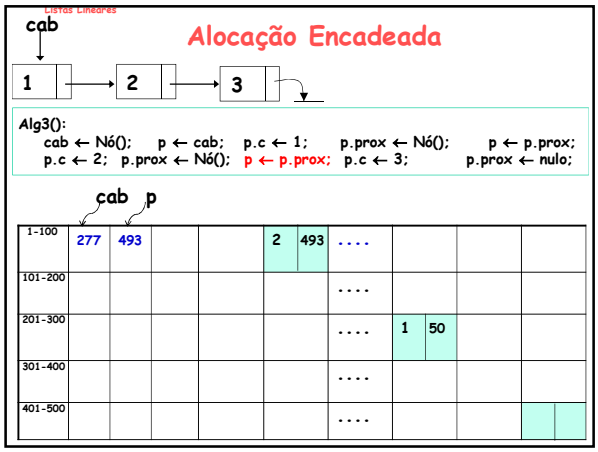
Listas Lineares

Alocação Encadeada



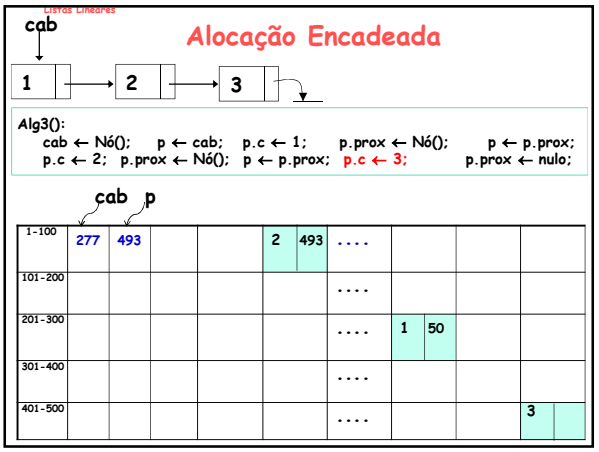
Listas Lineares

Alocação Encadeada

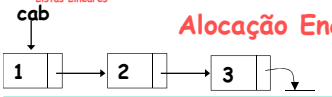


Listas Lineares

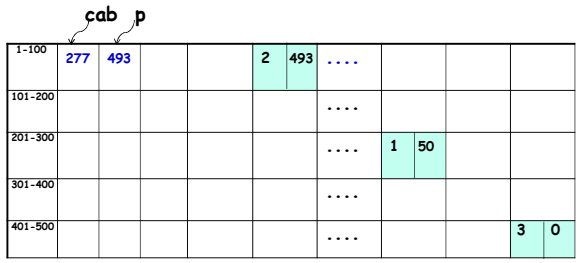
Alocação Encadeada



Alocação Encadeada

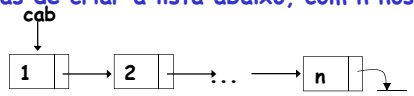



```
Alg3():
    cab ← Nó();   p ← cab;   p.c ← 1;   p.prox ← Nó();   p ← p.prox;
    p.c ← 2;  p.prox ← Nó();  p ← p.prox;  p.c ← 3;   p.prox ← nulo;
```



Alocação Encadeada

Duas maneiras de criar a lista abaixo, com n nós:

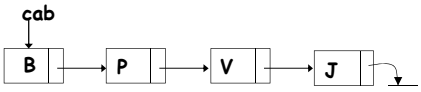



```
Cria1():
#Dados: inteiro n
    para i ← 1 até n incl.:
        p ← Nó()
        p.c ← i
        se (i = 1):
            cab ← p
        senão:
            q.prox ← p
        q ← p

    p.prox ← nulo
```

```
Cria2():
#Dados: inteiro n
    cab ← Nó()
    p ← cab
    p.c ← 1
    para i ← 2 até n incl.:
        p.prox ← Nó()
        p ← p.prox
        p.c ← i

    p.prox ← nulo
```


Alocação Encadeada

Exercício: explicar o que faz o seguinte algoritmo, em relação a uma lista encadeada sem nó cabeça:




```
Algoritmo misterioso()
#Dados: lista cab
    p ← cab;   r ← nulo;
    enquanto (p ≠ nulo):
        t ← p.prox;     p.prox ← r;
        r ← p;          p ← t;

    cab ← r
```


Alocação Encadeada

Três maneiras de inverter uma lista encadeada

Primeira: usar o algoritmo do exercício anterior, que só faz alteração em links.

Segunda: usar o algoritmo abaixo, que empilha os valores dos nós e depois varre novamente a lista, alterando os conteúdos dos nós.

```
Inverte2():
#Dados: lista cab
    p ← cab
    enquanto (p ≠ nulo):
        PUSH (p.c)
        p ← p.prox

    p ← cab
    enquanto (p ≠ nulo):
        v ← POP ();  p.c ← v;
        p ← p.prox
```


Alocação Encadeada

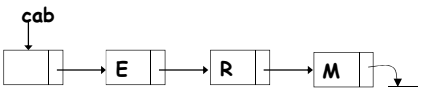
Três maneiras de inverter uma lista encadeada

Terceira: usar o algoritmo abaixo, que empilha os links na pilha S e depois desempilha, alterando as ligações entre os nós.

```
Inverte3():
#Dados: lista cab
    p ← cab
    enquanto (p ≠ nulo):
        PUSH (p)
        p ← p.prox

    POP(p);  cab ← p;
    enquanto (topo ≠ 0):
        p.prox ← S[topo]
        p ← POP()

    p.prox ← nulo
```


Alocação Encadeada

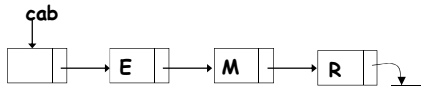
Busca em uma lista encadeada:




```
Busca(k):
#Dados: lista cab com nó cabeça, chave k
        p ← cab.prox;     pont ← nulo;
        enquanto (p ≠ nulo):
                se (p.c ≠ k):
                        p ← p.prox
                senão:
                        pont ← p;  p ← nulo;

        retornar pont
```

Alocação Encadeada

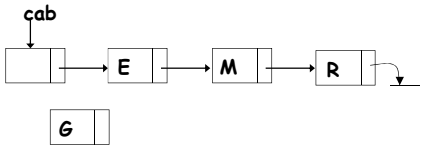
Busca em uma lista encadeada ordenada:




```
(ant, pont) Busca(k):
#Dados: lista cab com nó cabeça, chave k
        ant ← cab;  p ← cab.prox;    pont ← nulo;
        enquanto (p ≠ nulo):
                se (p.c < k):
                        ant ← p;   p ← p.prox;
                senão:
                        se (p.c = k):
                                pont ← p
                        p ← nulo

        retornar (ant, pont)
```

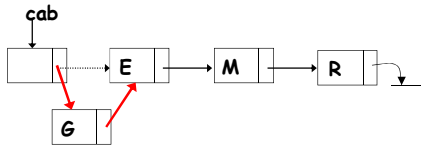

Alocação Encadeada

Inserção em uma lista encadeada não ordenada:



Alocação Encadeada

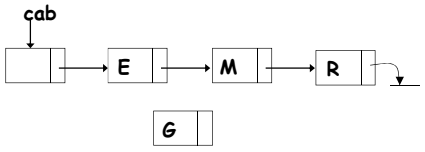
Inserção em uma lista encadeada não ordenada:




```
Nó Inserção(k):
#Dados: lista cab com nó cabeça, chave k
        pont ← Busca(k)
        se (pont = nulo):
                p ← Nó();   p.c  ← k;
                p.prox ←  cab.prox;    cab.prox ← p;
        senão:
                p ← nulo
        retornar p
```

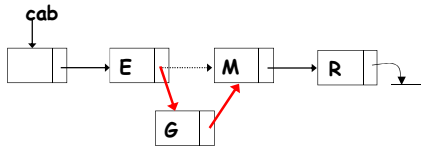

Alocação Encadeada

Inserção em uma lista encadeada ordenada:



Alocação Encadeada

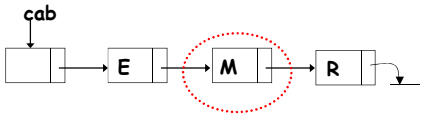
Inserção em uma lista encadeada ordenada:




```
Nó Inserção(k):
#Dados: lista cab com nó cabeça, chave k
        (ant, pont) ← Busca(k)
        se (pont = nulo):
                p  ← Nó();              p.c  ← k;
                p.prox ← ant.prox;     ant.prox ← p;
        senão:
                p ← nulo
        retornar p
```

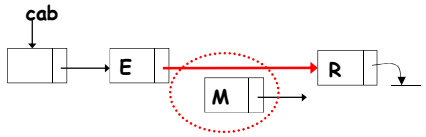

Alocação Encadeada

Remoção em uma lista encadeada ordenada:



Alocação Encadeada

Remoção em uma lista encadeada ordenada:

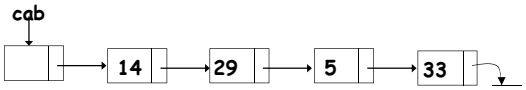



```
Nó Remoção(k):
#Dados: lista cab com nó cabeça, chave k
        (ant, pont) ← Busca(k)
        se (pont ≠ nulo):
                ant.prox ← pont.prox;    Desalocar(pont);
        senão:
                ant ← nulo
        retornar ant
```


Alocação Encadeada

Exercício: escrever um algoritmo para somar os elementos de posição par em uma lista encadeada contendo inteiros.

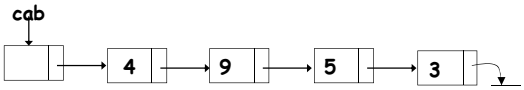
Ex: Na lista abaixo a soma é 62 (29+33):



Alocação Encadeada

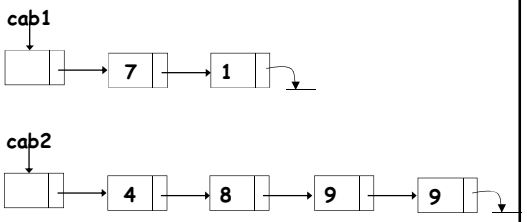
Exercício: escrever um algoritmo para transformar um número inteiro pequeno para a representação de inteiro grande, em uma lista encadeada.

Ex: o número 3594 seria representado como:



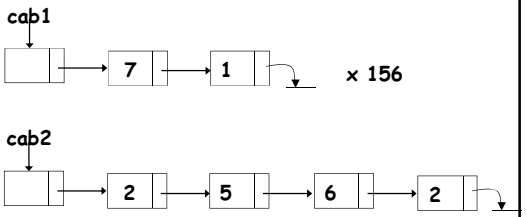
Alocação Encadeada

Exercício: escrever um algoritmo somar 2 grandes números representados em listas encadeadas.



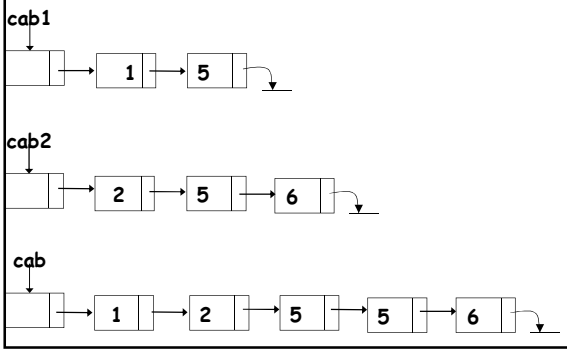
Alocação Encadeada

Exercício: escrever um algoritmo multiplicar um número pequeno por um número grande representado em listas encadeadas.



Alocação Encadeada

Merge de Listas encadeadas (ordenadas):



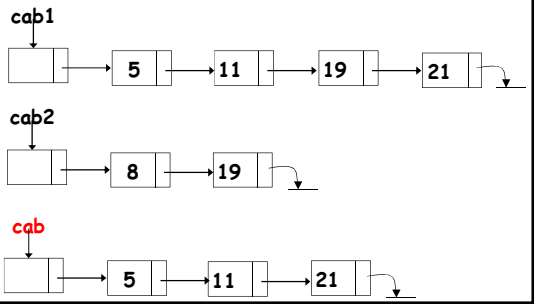
Alocação Encadeada

```
Nó Merge(Nó cab1, Nó cab2):
#Dados: listas encadeadas com nós cabeça cab1 e cab2
        p ← cab1.prox;      q ← cab2.prox;
        cab ← Nó();         r ← cab;

        enquanto (p ≠ nulo   e  q ≠ nulo):
                se (p.c < q.c):
                        r.prox ← p;   p ← p.prox;
                senão:
                        r.prox ← q;   q ← q.prox;
                r ← r.prox

        se (p ≠ nulo):
                r.prox ← p
        senão:
                r.prox ← q

        cab1.prox ← nulo;     cab2.prox ← nulo;

        retornar cab
```

Nesta solução as duas listas iniciais são esvaziadas.

Alocação Encadeada

Diferença de conjuntos: dados o conjunto A representado na lista cab1 (ordenada) e o conjunto B, na lista cab2 (ordenada), escrever um algoritmo para encontrar C = A-B.



Alocação Encadeada

```
Nó Diferança(Nó cab1, Nó cab2):
#Dados: listas encadeadas com nós cabeça cab1 e cab2
        p ← cab1.prox;      q ← cab2.prox;
        cab ← Nó();         r ← cab;

        enquanto (p ≠ nulo   e  q ≠ nulo):
              se (p.c < q.c):
                    r.prox ← Nó();  r ← r.prox;   r.c ← p.c;     p ← p.prox;
              senão se (p.c = q.c):
                    q ← q.prox;     p ← p.prox;
              senão:
                    q ← q.prox;

        enquanto (p ≠ nulo):
              r.prox ← Nó();    r ← r.prox;    r.c ← p.c;    p ← p.prox;

        r.prox ←  nulo;

        retornar cab
```

Nesta solução as duas listas iniciais são mantidas.

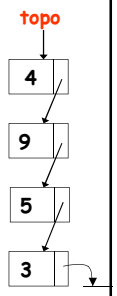
Alocação Encadeada

Pilhas como listas encadeadas:

```
Esvazia:
        topo ← nulo;
```

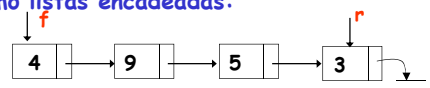
```
Nó PUSH(k):
        p ← Nó()
        p.c ← k;   p.prox ← topo;   topo ← p;
        retornar topo
```

```
inteiro POP():
        se (topo ≠ nulo):
              p ← topo;   k ← p.c;
              topo ← topo.prox;    desalocar(p);
        senão:
              k ← nulo
        retornar k
```



Alocação Encadeada

Filas como listas encadeadas:




```
Enfila(k):
        p ← Nó();  p.c ← k;    p.prox ← nulo;
        se (r ≠ nulo):
                r.prox ← p
        senão:
                f ← p
        r ← p
```

```
Esvazia:
        f ← nulo;   r ← nulo;
```

```
Nó Desenfila():
        se (f ≠ nulo):;
              p ← f;    k ← p.c;    f ← f.prox;    desalocar(p);
              se (f = nulo):
                    r ← nulo
        senão:
              k ← nulo
        retornar k
```


Alocação Encadeada

Ordenação por distribuição:

Este método de ordenação trabalha com a representação decimal dos números e executa tantos "passos" sobre o conjunto de números quantos sejam os dígitos do número. Em cada passo uma posição é examinada.

Ele tem uma máquina correspondente: a classificadora de cartões.

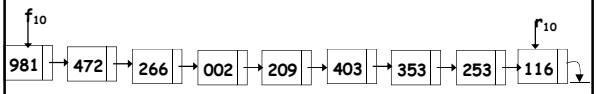
Exemplo:
{981, 472, 266, 002, 209, 403, 353, 253, 116}

Alocação Encadeada

Ordenação por distribuição - Exemplo

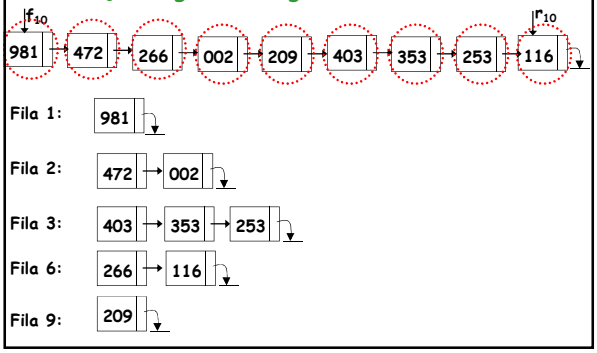
{981, 472, 266, 002, 209, 403, 353, 253, 116}

Passo 0 - criação da fila 10.



Alocação Encadeada

Ordenação por distribuição - Exemplo Passo 1 - Distribuição segundo o dígito 1.

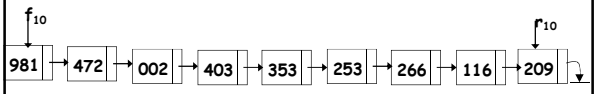


Alocação Encadeada

Ordenação por distribuição - Exemplo

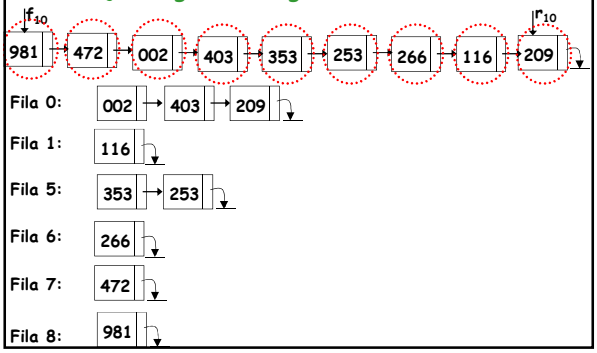
{981, 472, 266, 002, 209, 403, 353, 253, 116}

Fim do passo 1 - Aglomeração das filas.



Alocação Encadeada

Ordenação por distribuição - Exemplo Passo 2 - Distribuição segundo o dígito 2.

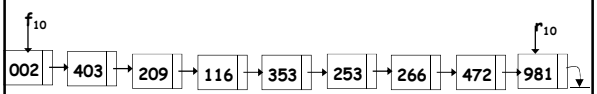


Alocação Encadeada

Ordenação por distribuição - Exemplo

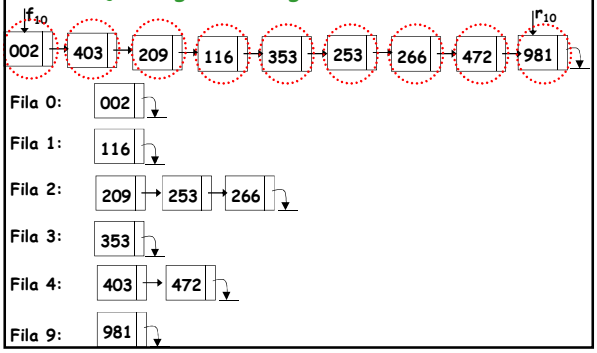
{981, 472, 266, 002, 209, 403, 353, 253, 116}

Fim do passo 2 - Aglomeração das filas.



Alocação Encadeada

Ordenação por distribuição - Exemplo Passo 3 - Distribuição segundo o dígito 3.

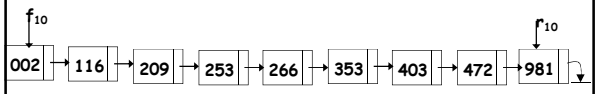


Alocação Encadeada

Ordenação por distribuição - Exemplo

{981, 472, 266, 002, 209, 403, 353, 253, 116}

Fim do passo 3 - Aglomeração das filas.
(fim da ordenação)



Alocação Encadeada

Ordenação por distribuição:

```
Distribuição():
#Dados: inteiro V[*], nd
    CriaFila10();
    para d ← 1 até nd incl.:
        para i ← 1 até n incl.:
            p ← Desenfila (10);   m ← Digito (d, p.c);   Enfila (m, p);

        para j ← 0 até 9 incl.:
                enquanto (Filas[j].f ≠ nulo):
                        p ← Desenfila (j);   Enfila (10, p);
```

Complexidade: Θ(n.nd)

Alocação Encadeada

Análise da Ordenação por Distribuição:

Complexidade:
Pior caso = Melhor caso: Θ(n.nd)

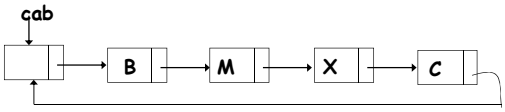
Estabilidade (manutenção da ordem relativa de chaves iguais):
Algoritmo estável

Memória adicional:
Praticamente nenhuma memória adicional

Usos especiais:
Chaves "pequenas"

Alocação Encadeada

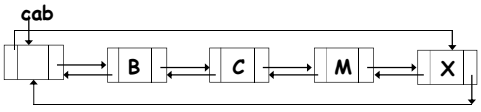
Listas circulares:




```
Nó Busca_circular(k):
#Dados: lista cab, chave k
        cab.c ← k;   pont ← cab.prox;
        enquanto (pont.c ≠ k):
                pont ← pont.prox

        se (pont = cab):
                pont ← nulo

        retornar pont
```


Alocação Encadeada

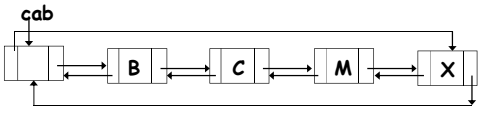
Listas Duplamente Encadeadas Ordenadas:




```
Nó Busca_dupla(k):
#Dados: lista cab, chave k
        cab.c ← k;   pont ← cab.prox;
        enquanto (pont.c < k):
            pont ← pont.prox

        retornar pont
```


Alocação Encadeada

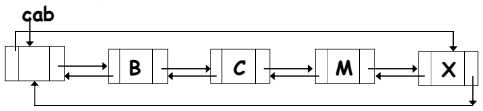
Listas Duplamente Encadeadas Ordenadas- Inserção:




```
Nó Inserção_dupla(k):
#Dados: lista cab, chave k
        pont ← Busca_dupla(k)
        se (pont = cab) ou (pont.c ≠ k):
                q ← pont.ante
                p ← Nó();          p.c ← k;
                p.prox ← pont;     p.ante ← pont.ante;
                q.prox ← p;        pont.ante ← p;
        senão:
                p ← nulo
        retornar p
```

Alocação Encadeada

Listas Duplamente Encadeadas - Remoção:



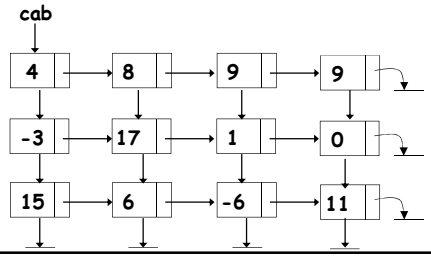

```
Nó Remoção_dupla(k):
#Dados: lista cab, chave k
        pont ← Busca_dupla(k)
        se (pont ≠ cab) e (pont.c = k):
             q ← pont.ante;    p ← pont.prox;
             q.prox ← p;        p.ante ← q;
             Desalocar (pont)
        senão:
             p ← nulo
        retornar p
```


Alocação Encadeada

Exercício: dado o grid encadeado:

a) escrever um algoritmo que soma todos os números do grid

b) escrever um algoritmo que encontra o maior elemento do grid.



Fim