Listas Lineares

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

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

maio/2018

Listas lineares são um conjunto de n nós referentes a n objetos, onde existe uma hierarquia entre os mesmos, dependente unicamente da posição relativa dos mesmos.

Listas lineares podem ter alocação sequencial ou encadeada.

Na Alocação sequencial (vetores), os nós são guardados em posições consecutivas da memória.

Na Alocação Encadeada, os nós são guardados em posições de memória não necessariamente consecutivas.

Vetores

Operações em vetores:

- Ordenação
- Buscas/atualizações
- Merge de Vetores
- Pesquisa Binária
- Representação de inteiros grandes

Ordenação de vetores:

- Seleção/Inserção
- Contagem
- Distribuição
- Mergesort
- Heapsort
- Outros métodos...

Vetores

Busca em Vetores (chave primária):

Problema:
dada uma lista com n registros, cada um identificado por uma chave única, determinar se o elemento de chave k faz parte do conjunto.

Operações correlatas:

- Busca de um elemento
- Busca por faixa
- Busca à proxima chave
- Busca por prefixo
- Atualizações

Vetores

Busca Sequencial I em Vetores (chave primária):

Percorrer sequencialmente o vetor até encontrar a chave procurada ou chegar ao final do mesmo.

```
inteiro Busca(k);
#Dados: Vetor V, inteiro n = |V|
    i ← 1;

    enquanto (i ≤ n) e (V[i] ≠ k):
        i ← i+1;

    se (i ≤ n):
        retornar i
    senão:
        retornar Nulo;
```


Vetores

Análise da Busca Sequencial (I):

Busca de um elemento:

- -BBS (pior caso): n comparações $\Theta(n)$
- -BBS (caso médio): n/2 comparações $\Theta(n)$
- -BMS: n comparações $\Theta(n)$

Busca por faixa:

- -varrer todo o vetor: n comparações $\Theta(n)$

Busca à proxima chave:

- -varrer todo o vetor: n comparações $\Theta(n)$

Busca por prefixo:

- -varrer todo o vetor: n comparações $\Theta(n)$

Vetores

Análise da Busca Sequencial (I):

Inserção de um elemento:
- no final do vetor $\Theta(1)$

Exclusão de um elemento:
- substituição pelo último: $\Theta(1)$

Vetores

Busca Sequencial II em Vetores (chave primária):

Percorrer sequencialmente o vetor até encontrar a chave procurada.

```
inteiro Busca(k);
#Dados: Vetor V, inteiro n = |V|
      i ← 1;   V[n+1]  ← k;

      enquanto (V[i] ≠ k):
            i ← i+1;

      se (i ≤ n):
            retornar i
      senão:
            retornar Nulo;
```


Vetores

Análise da Busca Sequencial (II):

= Busca Sequencial (I) só que a busca é mais rápida.

Vetores

Busca Sequencial III em Vetores (chave primária):

Percorrer sequencialmente o vetor ordenado até encontrar a chave procurada.

```
inteiro Busca(k);
#Dados: Vetor V, inteiro n = |V|
      i ← 1;   V[n+1]  ← ∞;
      enquanto (V[i] < k):
            i ← i+1;

      se (V[i] = k):
            retornar i
      senão:
            retornar Nulo;
```


Vetores

Análise da Busca Sequencial (III):

Busca de um elemento:
- BBS (pior caso): n comparações $\Theta(n)$
- BBS (caso médio): n/2 comparações $\Theta(n)$
- BMS: n/2 comparações $\Theta(n)$

Busca por faixa:
- varrer todo o vetor: n comparações $\Theta(n)$

Busca à proxima chave:
- teste da próxima posição: $\Theta(1)$

Busca por prefixo:
- varrer todo o vetor: n comparações $\Theta(n)$

Vetores

Análise da Busca Sequencial (I):

Inserção de um elemento:
- usando o algoritmo de Inserção: $\Theta(n)$

Exclusão de um elemento:
- rearrumação do vetor: $\Theta(n)$

Vetores

Exercício:

escrever os algoritmos para inserção e deleção de um elemento em um vetor ordenado.

Vetores

Merge de Vetores:

A operação de Merge (União,Fusão, Intercalação) é feita com vetores ordenados, gerando um novo vetor ordenado, resultado da união dos 2 vetores iniciais.

Exemplo: Merge de V1 com V2, gerando V3

	1	2	3	4	5	6	7	8	9	10	11
V1	5	10	19	20	31						
V2	2	5	7	20	21	29					
V3	2	5	5	7	10	19	20	20	21	29	31

Vetores

Merge de Vetores (I):

```
Merge(n, m);
#Dados: Vetores V1, V2, n =|V1|, m=|V2|
      V1[n+1] ← ∞;   V2[m+1]  ← ∞;
      i ← 1;     j  ← 1;

      para k ← 1 até (n+m) incl;
            se (V1[i] ≤ V2[j]):
                  V3[k] ← V1[i];   i  ← i+1;
            senão:
                  V3[k] ← V2[j];   j  ← j+1;
```

Complexidade: $\Theta(n+m)$

Vetores

Merge de Vetores:

Exemplo: Merge de V1 com V2, gerando V3

	1	2	3	4	5	6	7	8	9	10	11
V1	5	10	19	20	31						
V2	2	5	7	20	21	29					
V3	2	5	5	7	10	19	20	20	21	29	31

Vetores

Merge de Vetores (II):

Fazer merge de parte dos vetores, V1[1..n], V2[1..m],obtendo V3[1..n+m].

Usado quando não é possível colocar "sentinelas". Pode ser que o Merge tenha que ser feito entre pedaços de um mesmo vetor, como será mostrado no MERGESORT.

Vetores

Merge de Vetores (II):

```
Merge(n, m):
#Dados: Vetores V1, V2, inteiro n =|V1|, m = |V2|
      i ← 1;    j  ← 1;   k  ← 0;

      enquanto (i ≤ n) e (j ≤ m):
            k  ← k+1
            se (V1[i] ≤ V2[j]):
                  V3[k] ← V1[i];    i  ← i+1;
            senão:
                  V3[k] ← V2[j];    j  ← j+1;

      enquanto (i ≤ n):
            k  ← k+1;      V3[k] ← V1[i];    i  ← i+1;

      enquanto (j ≤ m):
            k  ← k+1;      V3[k] ← V2[j];    j  ← j+1;
```

Vetores

Exercício: dados dois conjuntos C1 e C2, representados em dois vetores ordenados, V1 e V2, escrever um algoritmo para obter o conjunto C3 = C1 - C2, (conjunto diferença), representado no vetor V3, também ordenado.

V1

1	2	3	4	5	6	7	8	9	10
B	D	F	H	J	M	O	Q	S	V

V2

1	2	3	4	5	6	7
A	D	E	G	O	Q	X

V3

1	2	3	4	5	6	7
B	F	H	J	M	S	V

Vetores

Pesquisa Binária:

A Pesquisa Binária é feita sobre um vetor ordenado e a idéia é, a cada passo, descartar metade das opções de busca do vetor.

Exemplo: Busca do O

1	2	3	4	5	6	7	8	9	10	11
B	D	F	H	J	M	O	Q	S	V	Y

O

O > M

(1+11)/2=6

c	f	i
1	11	6
7	11	

Vetores

Pesquisa Binária:

A Pesquisa Binária é feita sobre um vetor ordenado e a idéia é, a cada passo, descartar metade das opções de busca do vetor.

Exemplo: Busca do O

1	2	3	4	5	6	7	8	9	10	11
B	D	F	H	J	M	O	Q	S	V	Y

O

O < S

(7+11)/2=9

c	f	i
1	11	6
7	11	9

Vetores

Pesquisa Binária:

A Pesquisa Binária é feita sobre um vetor ordenado e a idéia é, a cada passo, descartar metade das opções de busca do vetor.

Exemplo: Busca do O

1	2	3	4	5	6	7	8	9	10	11
B	D	F	H	J	M	O	Q	S	V	Y

O = O

(7+8)/2=7

O

Busca bem sucedida!!

c	f	i
1	11	6
7	11	9
7	8	7

Vetores

Pesquisa Binária:

A Pesquisa Binária é feita sobre um vetor ordenado e a idéia é, a cada passo, descartar metade das opções de busca do vetor.

Exemplo: Busca do R

1	2	3	4	5	6	7	8	9	10	11
B	D	F	H	J	M	O	Q	S	V	Y

R

R > M

(1+11)/2=6

c	f	i
1	11	6
7	11	

Vetores

Pesquisa Binária:

A Pesquisa Binária é feita sobre um vetor ordenado e a idéia é, a cada passo, descartar metade das opções de busca do vetor.

Exemplo: Busca do R

1	2	3	4	5	6	7	8	9	10	11
B	D	F	H	J	M	O	Q	S	V	Y

R

R < S

(7+11)/2=9

c	f	i
1	11	6
7	11	9
7	8	7

Vetores

Pesquisa Binária:

A Pesquisa Binária é feita sobre um vetor ordenado e a idéia é, a cada passo, descartar metade das opções de busca do vetor.

Exemplo: Busca do R

1	2	3	4	5	6	7	8	9	10	11
B	D	F	H	J	M	O	Q	S	V	Y

R

R > O

(7+8)/2=7

c	f	i
1	11	6
7	11	9
7	8	7
8	8	8

Vetores

Pesquisa Binária:

A Pesquisa Binária é feita sobre um vetor ordenado e a idéia é, a cada passo, descartar metade das opções de busca do vetor.

Exemplo: Busca do R

1	2	3	4	5	6	7	8	9	10	11
B	D	F	H	J	M	O	Q	S	V	Y

R

R > Q

(8+8)/2=8

c	f	i
1	11	6
7	11	9
7	8	7
8	8	8
9	8	

Vetores

Pesquisa Binária:

A Pesquisa Binária é feita sobre um vetor ordenado e a idéia é, a cada passo, descartar metade das opções de busca do vetor.

Exemplo: Busca do R

1	2	3	4	5	6	7	8	9	10	11
B	D	F	H	J	M	O	Q	S	V	Y

R

Busca mal sucedida!!

Vetores

Pesquisa Binária:

```
inteiro PB(k):
#Dados: Vetor V ordenado, inteiro n=|V1|,  k
        c ← 1;   f  ← n;

        enquanto (c ≤ f):
                i ← ⌊(c+f)/2⌋;
                se (V[i] > k):
                        f  ← i - 1
                senão se (V[i] < k):
                        c  ← i + 1
                senão:
                        parar

        se (V[i] ≠ k):
                i ← Nulo

        retornar i
```


Vetores

Pesquisa Binária - Versão 2:

```
inteiro PB(k):
#Dados: Vetor V ordenado, inteiro n=|V1|,  k
        c ← 1;   f  ← n;

        enquanto (c ≤ f):
                i ← ⌊(c+f)/2⌋;
                se (V[i] > k):
                        f  ← i - 1
                senão se (V[i] < k):
                        c  ← i + 1
                senão:
                        parar

        retornar i
```


Vetores

Exercício: mostrar os valores que assumem as variáveis c, f, i, na execução do algoritmo da Pesquisa Binária, Versão 2, para as seguintes chaves:
A, C, E, K, X, no vetor abaixo.

1	2	3	4	5	6	7	8	9	10
B	D	F	H	J	M	O	Q	S	V

Vetores

Exercício: mostrar os valores que assume a variável de retorno, **i**, na execução do algoritmo da Pesquisa Binária, Versão 2, para as seguintes chaves: **A, C, E, K, X,** no vetor abaixo.

1	2	3	4	5	6	7	8	9	10
B	D	F	H	J	M	O	Q	S	V

Conclusão: nas buscas mal sucedidas da PB, a versão 2 do algoritmo sempre retorna ou o valor imediatamente superior ou o imediatamente inferior à chave buscada.

Vetores

Teorema: *O número máximo de comparações de chaves na Pesquisa Binária é* $\lfloor \log_2 n \rfloor + 1$.

Prova:

a) Para n = 1 e 2, o número de comparações é, no máximo, 1 e 2, respect., e, portanto, de acordo com o teorema.

b) Suponhamos o teorema válido para valores inferiores a $n > 2$. Quando consideramos o pior caso para n elementos, a primeira comparação, que falha, ocorre na posição $\lceil n/2 \rceil$, gerando dois sub-vetores de tamanhos $\lceil n/2 \rceil - 1$, à esquerda e $\lfloor n/2 \rfloor$ à direita, sendo este o maior. Portanto, o número de comparações máximo total, pela hipótese de indução é
$1 + \lfloor \log_2 \lfloor n/2 \rfloor \rfloor + 1 = 1 + \lfloor \log_2 n/2 \rfloor + 1 = \lfloor \log_2 n \rfloor + 1$.
i.e. o teorema também é válido para n.

Vetores

Análise da Pesquisa Binária:

Busca de um elemento:
- -BBS (pior caso): log n comparações Θ(log n)
- -BBS (caso médio): Θ(log n)
- -BMS: log n comparações Θ(log n)

Busca por faixa (k_1, k_2):
- -PB(k_1) + Busca sequencial: Θ(log n + c)

Busca à proxima chave:
- -teste da próxima posição: Θ(1)

Busca por prefixo:
- -PB(k_1) + Busca sequencial: Θ(log n + c)

Vetores

Análise da Pesquisa Binária:

Inserção de um elemento:
- -usando o algoritmo de Inserção: Θ(n)

Exclusão de um elemento:
- -rearrumação do vetor: Θ(n)

Vetores

Ordenação por Contagem:

Aplicável quando todos os valores das chaves são pequenos.

Exemplo: Ordenar

9	1	6	5	5	3	1	2	5	8	7	7	3	7	6

A solução é baseada na contagem (simples e acumulada) das chaves de cada valor:

K	1	2	3	4	5	6	7	8	9
VS	2	1	2	0	3	2	3	1	1
VA	2	3	5	5	8	10	13	14	15

Vetores

Ordenação por Contagem:

Ex:

K	1	2	3	4	5	6	7	8	9
VA	2	3	5	5	8	~~10~~ 9	13	14	15

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
9	1	6	5	5	3	1	2	5	8	7	7	3	7	6
									6					

K	1	2	3	4	5	6	7	8	9
VA	2	3	5	5	8	9	~~13~~ 12	14	15

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
9	1	6	5	5	3	1	2	5	8	7	7	3	7	6
									6			7		

Vetores

Ordenação por Contagem:

Ex:

VA	2	3	5	5	8	9	12	14	15

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
9	1	6	5	5	3	1	2	5	8	7	7	3	7	6
				3					6			7		

Situação Final:

k	1	2	3	4	5	6	7	8	9
VA	0	2	3	5	5	8	10	13	14

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	3	5	5	5	6	6	7	7	7	8	9

Vetores

Ordenação por Contagem:

```
Contagem (n, p):
#Dados: Vetor V, inteiro n=|V|, p  #valor máximo
    para j ← 1 até p incl:
        V_A[j] ← 0

    para i ← 1 até n incl:
        V_A[V_1[i]] ← V_A[V_1[i]]+1

    para j ← 2 até p incl:
        V_A[j] ← V_A[j] + V_A[j-1]

    para i ← n..1 incl:
        V_2[V_A[V_1[i]]] ← V_1[i]
        V_A[V_1[i]] ← V_A[V_1[i]] -1
```


Vetores

Análise da Ordenação por Contagem:

Complexidade:

Pior caso = Melhor caso: $\Theta(n+p)$

Estabilidade (manutenção da ordem relativa de chaves iguais):

Algoritmo estável

Memória adicional:

Vetores para contagem e para cópia

Usos especiais:

Chaves "pequenas"

Vetores

Representação de inteiros grandes

Convenções usadas, usando vetor:

a) A posição 0 indica o sinal (1, positivo,-1, negativo)
b) Número representado da direita para a esquerda, com um dígito em cada posição. Vetor completado com zeros à esquerda.
c) variável associada, tb, que indica o tamanho dos vetores, excluindo a posição 0.

Ex. representação do número 1234, c/ tb = 8

0	1	2	3	4	5	6	7	8
1	0	0	0	0	1	2	3	4

Obs: há muitas outras convenções para a representação.

Vetores

Representação de inteiros grandes - Métodos

a) Conversão de um número pequeno para grande

```
#ConvPNBN(k)
#Dados: inteiro k
    se (k > 0):
        B[0] ← 1
    senão:
        B[0] ← -1;  k ← -k;

    para i ← tb..1:
        B[i] ← K mod 10;  k ← ⌊k/10⌋;

    retornar B
```

Ex:
k = 1234

	0	1	2	3	4	5	6	7	8
B	1	0	0	0	0	1	2	3	4
k		0	0	0	0	0	1	12	123

Vetores

Representação de inteiros grandes - Métodos

b) Produto de um número grande por um pequeno.

```
# ProdPNBN(B, k);
#Dados: inteiro k, vetor B  #número grande
    se (k < 0):
        B[0] ← -B[0];  k ← -k;

    v1 ← 0;
    para i ← tb..1:
        x ← B[i]*k + v1
        B[i] ← x mod 10
        v1 ← ⌊x/10⌋

    retornar B;
```

Vetores

Representação de inteiros grandes - Métodos

b) Produto de um número grande por um pequeno
Ex: para k = 254.

	0	1	2	3	4	5	6	7	8
B(antes)	1	0	0	0	0	1	2	3	4
B(depois)	1	0	0	3	1	3	4	3	6
x		0	0	3	31	313	594	863	1016
v1		0	0	0	3	31	59	86	101

Vetores

Representação de inteiros grandes - Métodos

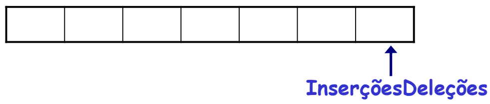
Exercício:
escrever um algoritmo para, dados os números grandes positivos B1, B2 e B3, obter:
B3 = B1 + B2.

Pilhas e Filas

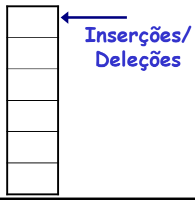
Em muitas situações temos procedimentos repetidos, quando se consideram inserções e deleções no vetor. Algumas dessas situações recebem nomes especiais:

Pilhas, Filas, Deques.

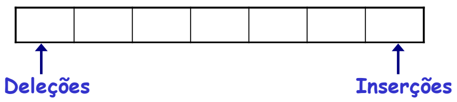
Pilhas: inserções e deleções no fim do vetor
LIFO (Last In, First Out)



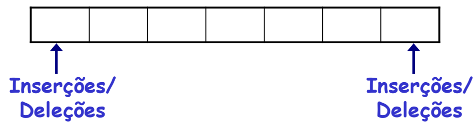
A representação usual de uma pilha é um vetor "em pé:



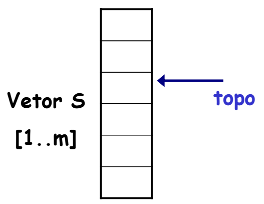
Filas: inserções no final,deleções no início do vetor
FIFO (First In, First Out)



Deques (Double Entry Queues):
inserções e deleções nas extremidades do vetor



Pilhas: representação computacional.



Pilhas: Esvaziamento

```
Esvazia():
      topo ← 0
```

Pilhas: Inserções

```
inteiro PUSH(k);
#Dados: pilha S, k
      se (topo ≠ m):
            topo ← topo + 1; S[topo] ← k;
      senão:
            topo ← Nulo
      retornar topo
```


Pilhas: Deleções

```
inteiro POP():
#Dados: pilha S
      se (topo > 0):
            k ← S[topo];  topo ← topo - 1;
      senão
            k ← Nulo
      retornar k
```


Versão simplificada dos algoritmos de pilha

Pilhas: Inserções

```
inteiro PUSH(k):
#Dados: pilha S, inteiro k
      topo ← topo + 1; S[topo] ← k;
```

Pilhas: Deleções

```
inteiro POP():
#Dados: pilha S
      k ← S[topo];  topo ← topo - 1;
      retornar k;
```


Pilhas: Exemplo 1 - Inversão de vetor

```
Inverte1():
#Dados: inteiro P[*]
      para i ← 1 até n incl:
            PUSH(P[i])

      para i ← 1 até n incl:
            P[i] ← POP()
```

```
Inverte2():
#Dados: inteiro P[*]
      para i ← 1 até n incl:
            PUSH(k)

      i ← 0
      enquanto (topo > 0):
            i ← i+1
            P[i] ← POP()
```


Pilhas: Exemplo 2 - Cálculo de Expressões em Notação Polonesa Reversa

Notação de Expressões Aritméticas:

a) Informal: (4 + 5*6) - (2 + 7)

Notação ambígua. Ambiguidade resolvida informal-mente, considerando a prioridade de operadores.

b) Parentizada: ((4 + (5*6)) - (2 + 7))

Notação não ambígua. Utiliza um par de parênte-ses para cada operador.

Pilhas: Exemplo 2 - Cálculo de Expressões em Notação Polonesa Reversa

Notação de Expressões Aritméticas:

c) Polonesa direta: - + 4 * 5 6 + 2 7

Notação não ambígua. Resolvida operando, sucessivamente, dois operandos contíguos com o operador precedente.

c) Polonesa reversa: 4 5 6 * + 2 7 + -

Notação não ambígua. Resolvida operando, sucessivamente, cada operador com dois operandos precedentes. Muito utilizada em calculadoras.

Pilhas: Exemplo 2 - Cálculo de Expressões em Notação Polonesa Reversa

Idéia: varrer a expressão, empilhando operandos e, para cada operador encontrado, substituir os 2 elementos do topo da pilha pelo resultado de sua operação.

1	2	3	4	5	6	7	8	9
4	5	6	*	+	2	7	+	-

		6				7		
	5	5	30		2	2	9	
4	4	4	4	34	34	34	34	25

Pilhas: Exemplo 2 - Cálculo de Expressões em Notação Polonesa Reversa

```
Calculo():
#Dados: cadeia V[*], n = |V|, pilha S
     Esvazia(S)
     para i ← 1 até n incl;
          se (V[i] é operando):
               PUSH (S, V[i])
          senão:
               op2 ← POP ();   op1 ← POP ();
               PUSH (S,Resultado(op1, op2, V[i]))

     retornar S[1];
```


Exercício:

converter (intuitivamente) a seguinte expressão para notação polonesa reversa e mostrar a situação da pilha no seu cálculo.

25/(2-3 - 2*(4-6/3))

Pilhas: Exemplo 3 - Conversão de Notação Parentisada para Notação Polonesa Reversa

Idéia: varrer a expressão empilhando operadores, transcrevendo operandos e desempilhando operadores quando encontrar ")". "(" são ignorados.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
(	(	4	+	(	5	*	6	)	)	-	(	2	+	7	)	)

1	2	3	4	5	6	7	8	9
4	5	6						

*
+

Pilhas: Exemplo 3 - Conversão de Notação Parentisada para Notação Polonesa Reversa

Idéia: varrer a expressão empilhando operandores, transcrevendo operandos e desempilhando operadores quando encontrar). (são ignorados.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
(	(	4	+	(	5	*	6	)	)	-	(	2	+	7	)	)

1	2	3	4	5	6	7	8	9
4	5	6	*					

+

Pilhas: Exemplo 3 - Conversão de Notação Parentisada para Notação Polonesa Reversa

Idéia: varrer a expressão empilhando operandores, transcrevendo operandos e desempilhando operadores quando encontrar). (são ignorados.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
(	(	4	+	(	5	*	6	)	)	-	(	2	+	7	)	)

1	2	3	4	5	6	7	8	9
4	5	6	*	+	2	7		

+
-

Pilhas: Exemplo 3 - Conversão de Notação Parentisada para Notação Polonesa Reversa

Idéia: varrer a expressão empilhando operandores, transcrevendo operandos e desempilhando operadores quando encontrar). (são ignorados.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
(	(	4	+	(	5	*	6	)	)	-	(	2	+	7	)	)

1	2	3	4	5	6	7	8	9
4	5	6	*	+	2	7	+	

-

Pilhas: Exemplo 3 - Conversão de Notação Parentisada para Notação Polonesa Reversa

Idéia: varrer a expressão empilhando operandores, transcrevendo operandos e desempilhando operadores quando encontrar). (são ignorados.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
(	(	4	+	(	5	*	6	)	)	-	(	2	+	7	)	)

1	2	3	4	5	6	7	8	9
4	5	6	*	+	2	7	+	-

Pilhas: Exemplo 3 - Conversão de Notação Parentisada para Notação Polonesa Reversa

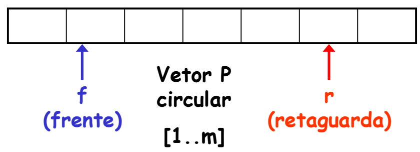
```
Conversão():
#Dados: cadeia E
     Esvazia(S);   p ← 0;
     para i ← 1 até n incl;
          se (E[i] é operando):
               p ← p+1;    R[p] ← E[i];
          senão se (E[i] é operador):
               PUSH (E[i])
          senão se (E[i] = ')'):
               op  ← POP ()
               p ← p+1;    R[p] ← op;
```


Exercício:

mostrar a situação da pilha na conversão da seguinte expressão parentizada para notação polonesa reversa.

(25 / (((6 / 3) - 3) - (2 * (4 - 2))))

Filas: representação computacional.



Filas: Esvaziamento

```
Esvazia():
    f ← 0;   r ← 0;
```

Filas: Inserções

```
inteiro Enfila(k):
#Dados: fila Q, inteiro k
    prov ← (r mod m) + 1
    se (prov ≠ f):
        r ← prov;   Q[r] ← k;
        se (f = 0):
            f ← 1
        retornar r
    senão:
        retornar Nulo
```


Filas: Deleções

```
inteiro Desenfila ():
#Dados: fila Q
    se (f ≠ 0):
        k ← Q[f]
        se (f = r):
            Esvazia
        senão:
            f ← (f mod m) + 1
    senão:
        k ← Nulo
    retornar k
```


Versão simplificada dos algoritmos de filas

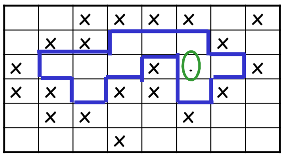
Filas: Inserções

```
Enfila(k):
#Dados: fila Q, chave k
    r ← (r mod m) + 1;    Q[r] ← k;
    se (f = 0):
        f ← 1
```

Filas: Deleções

```
inteiro Desenfila ():
#Dados: fila Q
    k ← Q[f]
    se (f = r):
        Esvazia
    senão:
        f ← (f mod m) + 1
    retornar k
```

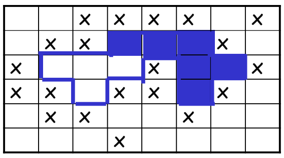

Filas: Exemplo - Coloração de área



Colorir a área livre que contém (3,6)

Idéia: dada uma matriz nl x nc, onde as áreas livres são delimitadas por 'x' e a vizinhança de um ponto corresponde aos 4 pontos cardeais, a coloração de uma área que contém dado ponto pode ser feita usando uma fila.

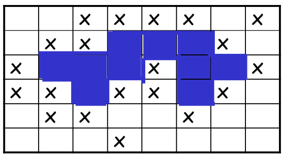
Filas: Exemplo - Coloração de área



Colorir a área livre que contém (3,6)

(3,6)	(2,6	(4,6)	(3,7						
	(2)6)	(4,6)	(3)7)	(2,5)					
		(4,6)	(3,7)	(2,5)					
			(3,7)	(2,5)					
				(2,5)	(2,4)				

Filas: Exemplo - Coloração de área



Colorir a área livre que contém (3,6)

					(2,4)	(3,4)			
						(3,4)	(3,3		
							(3)3)	(3,2)	(4,3)
								(3,2)	(4,3)
									(4,3)

Filas: Exemplo - Coloração de área (outra visão)

		x	x	x	x		x
	x	x				x	
x				x			x
x	x		x	x		x	
	x	x			x		
			x				

(3,6)	(2,6)	(4,6	(3,7)
(2,5)	(2,6)	(4)6)	(3,7)
(2,5)		(4,6)	(3,7)
(2,5)			(3,7)
(2,5)	(2,4		
	(2)4)	(3,4)	
		(3,4)	(3,3)
(3,2)	(4,3		(3,3)
(3,2)	(4)3)		
	(4,3)		

Colorir a área livre que contém (3,6)

Filas: Coloração

```
Coloraçao (p, q):
#Dados: matriz M, ponto (p, q)
        EsvaziaFila();   Enfila(p, q);
        enquanto (f ≠ 0):
                (a, b) ← primeiro elemento da fila;
                Enfila(a-1, b);   Enfila(a, b-1);
                Enfila(a+1, b);   Enfila(a, b+1);
                (a, b) ← Desenfila();
```

```
Enfila (x, y):
        se (1 ≤ x ≤ nl) e (1 ≤ y ≤ nc) e (Mat[x, y] = ' '):
                Mat[x, y] ← cor
                r ← r mod m + 1;   Q[r].x ← x;   Q[r].y ← y;
                se (f = 0):
                        f ← 1
```

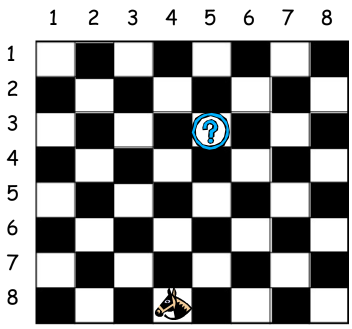

Filas: Exercício - Coloração de área

Mostrar a situação da fila na coloração de área para o ponto (6,5).

		x	x	x	x		x
	x	x				x	
x				x			x
x	x		x	x		x	
	x	x			x		
			x	.			

Colorir a área livre que contém (6,5)

Filas: Exemplo 2 - Cavalo no Xadrez



Determinar o número mínimode saltos para um cavalo chegar a determinada posição do tabuleiro

Filas: Exemplo 2 - Cavalo no Xadrez

	1	2	3	4	5	6	7	8
1	4		4		4		4	
2	3	4	3	4	3	4	3	4
3	4	3	4	3	4	3	4	3
4	3	2	3	2	3	2	3	4
5	2	3	2	3	2	3	2	3
6	3	4	1	2	1	4	3	2
7	2	1	2	3	2	1	2	3
8	3	2	3		3	2	3	2

Determinar o número mínimo de saltos para um cavalo chegar a determinada posição do tabuleiro

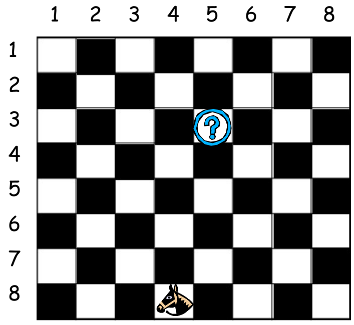
R: 4

Filas: Cavalo no Xadrez

```
Cavalo (p, q, x, y):
#Dados: matriz T, ponto (p, q) , (x, y)
        T[*,*] ← -1;  EsvaziaFila();   Enfila(p, q, 0);
        Enquanto (f ≠ 0):
                (a, b, c) ← primeiro elemento da fila;
                Enfila(a-2, b-1, c+1);      Enfila(a-2, b+1, c+1);
                Enfila(a-1, b-2, c+1);      Enfila(a-1, b+2, c+1);
                Enfila(a+1, b-2, c+1);      Enfila(a+1, b+2, c+1);
                Enfila(a+2, b-1, c+1);      Enfila(a+2, b+2, c+1);
                (a, b, c) ←  Desenfila();

        Imprimir (T[x,y]);
```

```
Enfila (x, y, z):
        se (x ≥ 1) e (x ≤ 8) e (y ≥ 1) e (y ≤ 8) e (T[x, y] = -1):
                T[x, y] ← z
                r ← r mod m+1;  Q[r].x ← x;  Q[r].y ← y;  Q[r].z ← z;
                se (f = 0):
                        f ← 1
```

Filas: Exemplo 2 - Cavalo no Xadrez



Qual o tamanho mínimo da fila para resolver este problema?

Filas: Exemplo 2 - Cavalo no Xadrez

	1	2	3	4	5	6	7	8
1	4		4		4		4	
2	3	4	3	4	3	4	3	4
3	4	3	4	3	?	3	4	3
4	3	2	3	2	3	2	3	4
5	2	3	2	3	2	3	2	3
6	3	4	1	2	1	4	3	2
7	2	1	2	3	2	1	2	3
8	3	2	3		3	2	3	2

Como determinar um caminho mínimo entre (8,4) e (3,5)

R: Guardando toda a fila e, em cada posição, a posição anterior que colocou a atual na fila.

Filas: Exemplo 2 - Caminho Mínimo

1	2	3	4	5	6	7	8
(8,4,0,0)	(7,6,1,1)	(6,5,1,1)	(6,3,1,1)	(7,2,1,1)	(6,8,2,2)	....	....

21	22	...	....	45	46	....
(5,1,2,5)	(4,7,3,6)	.....	.....	(2,1,3,20)	(3,5,4,22)	....

Caminho Mínimo: (8,4)→(7,6)→(6,8)→(4,7)→(3,5)

Filas: Exemplo 3 – "Food Fill" em matriz

	1	2	3	4	5
1					
2					
3					
4				.	
5					

Idéia: mostrar, de forma simplificada, a fila, no processo de "enchente" da matriz 5 x 5, começando no ponto (2, 5). Será mostrada a ordem de preenchimento da matriz e, para cada ponto, quais e quanto pontos estão na fila.

Filas: Exemplo 3 – "Food Fill" em matriz

Matriz

	1	2	3	4	5
1	25	23	18	12	19
2	24	20	13	6	14
3	21	15	7	2	8
4	16	9	3	1	5
5	22	17	10	4	11

Na matriz é representada a ordem de preenchimento de cada célula. Na fila é representada a faixa de células que estão na fila no momento da saída da célula do início da faixa, bem como o tamanho dessa fila.

Fila simplificada

1	5	5
2	8	7
3	10	8
4	11	8
5	11	7
6	14	9
7	15	9
8	15	8
9	17	9
10	17	8
11	17	7
12	19	8
13	20	8

14	20	7
15	21	6
16	22	7
17	22	6
18	23	6
19	23	5
20	24	5
21	24	4
22	24	3
23	25	3
24	25	2
25	25	1

Filas: Exercício – "Food Fill" em matriz

	1	2	3	4
1				
2				
3				
4				

Mstrar, de forma simplificada, a matriz e a fila, no processo de "enchente" da matriz 4 x 4, começando em um ponto a ser escolhido.

Alocação Encadeada

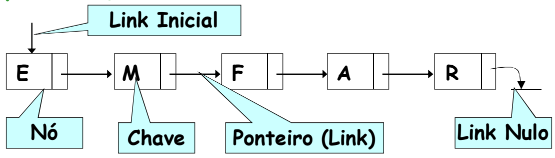
Listas encadeadas:

- 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

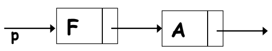
Alocação de um nó:
Nó p ← No()
lista p = malloc(sizeof *p); (C).
p = new(no); (C++).

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++).

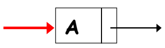
Efeito de um comando Desalocar:
o espaço é retornado para a PED.

Alocação Encadeada



Referência a um elemento de um nó:

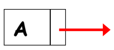
p.c F
p.prox



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

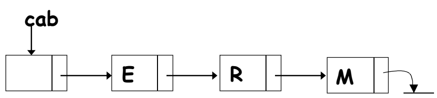
Outras referências:

p.prox.c A
p.prox.prox

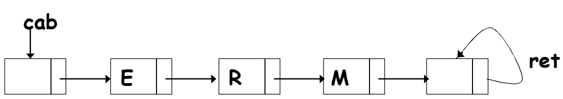


Alocação Encadeada

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

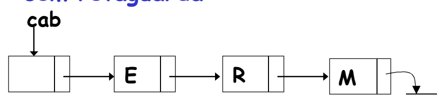


Nó retaguarda:
Às vezes usa-se também um nó especial no final.



Alocação Encadeada

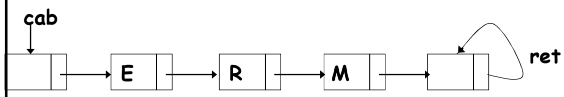
Impressão de uma lista Encadeada com cabeça e sem retaguarda:




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

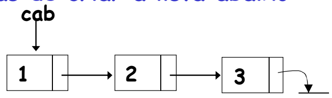

Alocação Encadeada

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



Alocação Encadeada

Três maneiras de criar a lista abaixo:



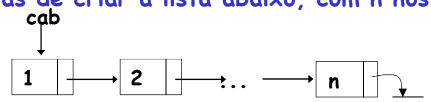

```
Alg1():
    cab ← No();     cab.c ← 1;    p ← No();  p.c ← 2;  q ← No(); q.c ← 3;
    cab.prox ← p;   p.prox ← q;   q.prox ← nulo;
```

```
Alg2():
    cab ← No();     cab.c ← 1;    p ← No();   p.c ← 2;    cab.prox ← p;
    p ← No();       p.c ← 3;      cab.prox.prox ← p;      p.prox ← nulo;
```

```
Alg3():
    cab ← No();    p ← cab;   p.c ← 1;    p.prox ← No();      p ← p.prox;
    p.c ← 2;   p.prox ← No();  p ← p.prox;  p.c ← 3;          p.prox ← nulo;
```


Alocação Encadeada

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

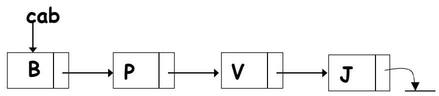



```
Cria1():
#Dados: inteiro n
    para i ← 1 até n incl.:
        p ← Nó()
        p.c ← i
        se (i = 1):
            cab ← p
        senão:
            q.prox ← p;
        q ← p;

    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

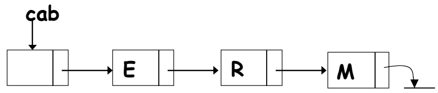
Três maneiras de inverter uma lista encadeada

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

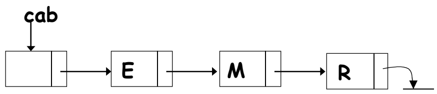
Busca em uma lista encadeada:




```
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 ← nulol;

        retornar pont;
```


Alocação Encadeada

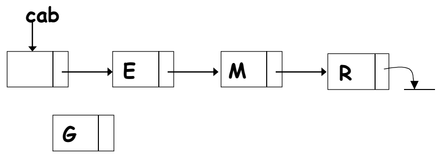
Busca em uma lista encadeada ordenada:




```
(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);
```

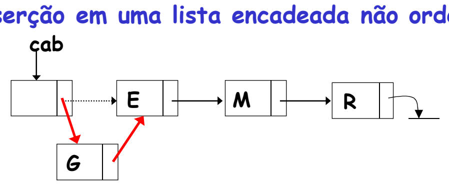

Alocação Encadeada

Inserção em uma lista encadeada não ordenada:



Alocação Encadeada

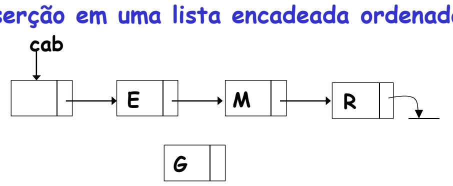
Inserção em uma lista encadeada não ordenada:




```
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;
```

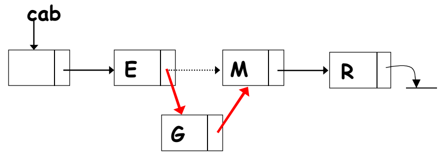

Alocação Encadeada

Inserção em uma lista encadeada ordenada:



Alocação Encadeada

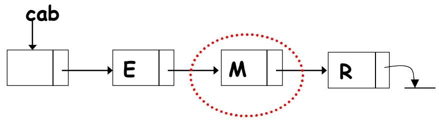
Inserção em uma lista encadeada ordenada:




```
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;
```

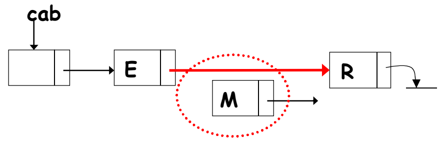

Alocação Encadeada

Remoção em uma lista encadeada ordenada:



Alocação Encadeada

Remoção em uma lista encadeada ordenada:

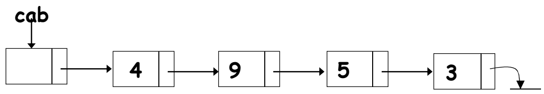



```
Nó Remoção(k):
#Dados: lista cab com nó cabeça, chave k
        (ant, pont) ← Busca(k);
        se (pont ≠ nulo):
                ant.prox ← pont.prox;    Desalocar(pont);
        senão:
                ant ← nulo;
        retornar ant;
```


Alocação Encadeada

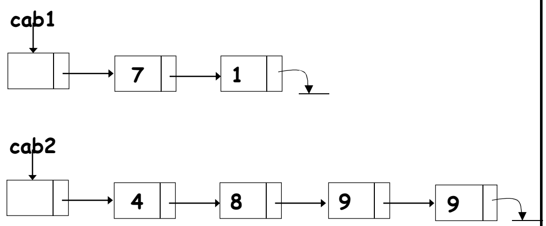
Exercício: escrever um algoritmo para transformar um número inteiro pequeno para a representação de inteiro grande, em uma lista encadeada.

Ex: o número 3594 seria representado como:



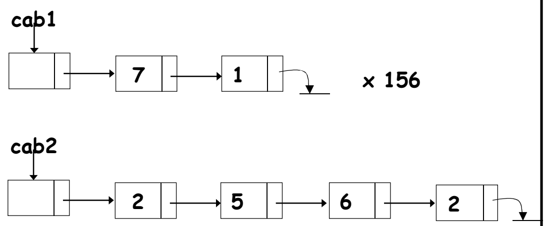
Alocação Encadeada

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



Alocação Encadeada

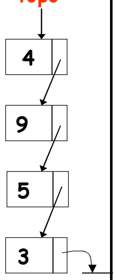
Exercício: escrever um algoritmo multiplicar um número pequeno por um número grande representado em listas encadeadas.



Alocação Encadeada

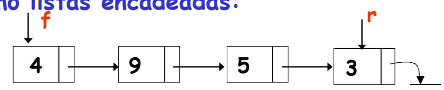
Pilhas como listas encadeadas:

```
#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.

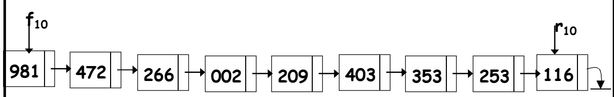
Exemplo:
{981, 472, 266, 002, 209, 403, 353, 253, 116}

Alocação Encadeada

Ordenação por distribuição - Exemplo

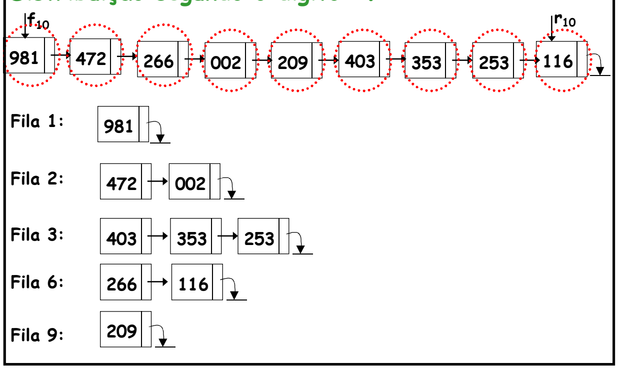
{981, 472, 266, 002, 209, 403, 353, 253, 116}

Passo 0 - criação da fila 10.



Alocação Encadeada

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

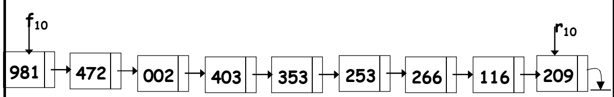


Alocação Encadeada

Ordenação por distribuição - Exemplo

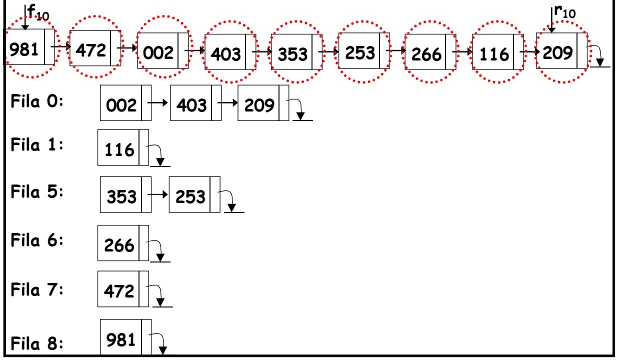
{981, 472, 266, 002, 209, 403, 353, 253, 116}

Fim do passo 1 - Aglomeração das filas.



Alocação Encadeada

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

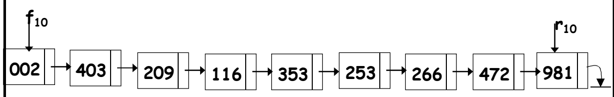


Alocação Encadeada

Ordenação por distribuição - Exemplo

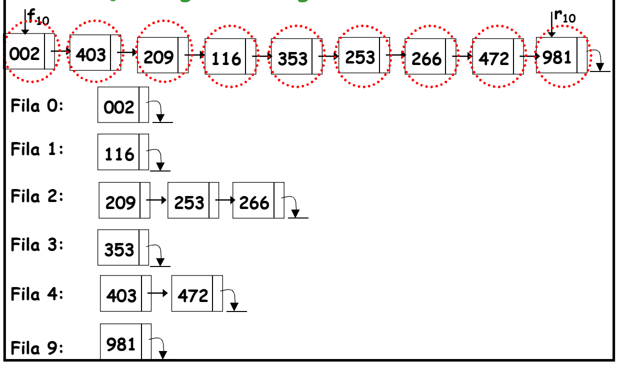
{981, 472, 266, 002, 209, 403, 353, 253, 116}

Fim do passo 2 - Aglomeração das filas.



Alocação Encadeada

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

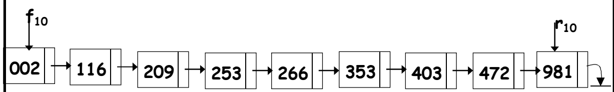


Alocação Encadeada

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: $\Theta(n.nd)$

Alocação Encadeada

Análise da Ordenação por Distribuição:

Complexidade:
Pior caso = Melhor caso: $\Theta(n.nd)$

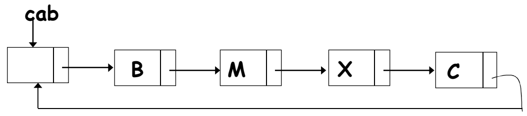
Estabilidade (manutenção da ordem relativa de chaves iguais):
Algoritmo estável

Memória adicional:
Praticamente nenhuma memória adicional

Usos especiais:
Chaves "pequenas"

Alocação Encadeada

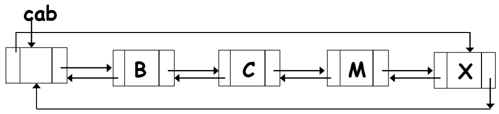
Listas circulares:




```
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

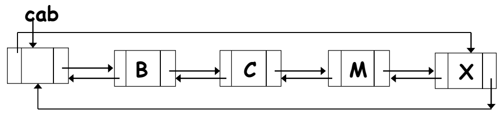
Listas Duplamente Encadeadas Ordenadas:




```
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

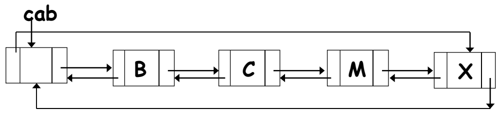
Listas Duplamente Encadeadas Ordenadas-Inserção:




```
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:



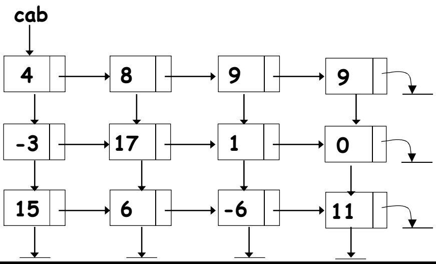

```
Nó Remoção_dupla(k):
#Dados: lista cab, chave k
        pont ← Busca_dupla(k);
        se (pont ≠ cab) e (pont.c = k):
             q ← pont.ante;    p ← pont.prox;
             q.prox ← p;       p.ante ← q;
             Desalocar (pont);
        senão:
             p ← nulo;
        retornar p;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


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