

EXEMPLOS DE APLICAÇÃO E ALGORITMOS GERAIS EM LISTAS LIGADAS

Prof.: Julio Cesar dos Reis

Roteiro

2

- Exemplo de aplicação de listas
 - ▣ Problema de Josephus
 - ▣ Polinômios
 - ▣ Matrizes esparsas

Roteiro

3

- Exemplo de aplicação de listas
 - ▣ Problema de Josephus
 - ▣ Polinômios
 - ▣ Matrizes esparsas
- Algoritmos gerais em listas
 - ▣ Cópia
 - ▣ Inversão
 - ▣ Concatenação

Problema de Josephus

4

- Um grupo de N rebeldes serão mortos exceto 1 deles
- Escolhe-se um valor $x < N$
- Conta-se x rebeldes, e aquele selecionado é morto

Problema de Josephus

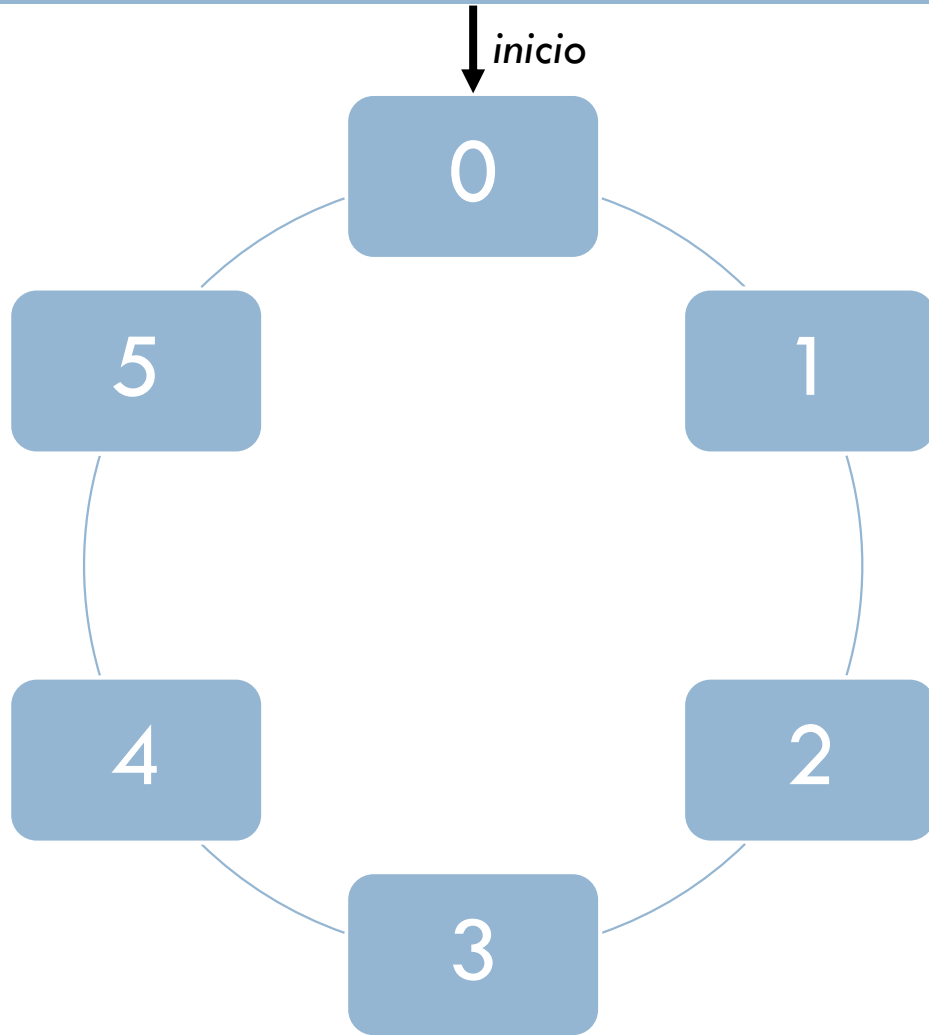
5

- Um grupo de N rebeldes serão mortos exceto 1 deles
- Escolhe-se um valor $x < N$
- Conta-se x rebeldes, e aquele selecionado é morto
- Continua-se do próximo rebelde em sentido horário
 - ▣ Conta-se mais x rebeldes, e aquele é morto
- Ciclicamente os rebeldes são mortos e apenas o último restante é salvo

Exemplo do Problema de Josephus

6

$N=6$
 $x=2$

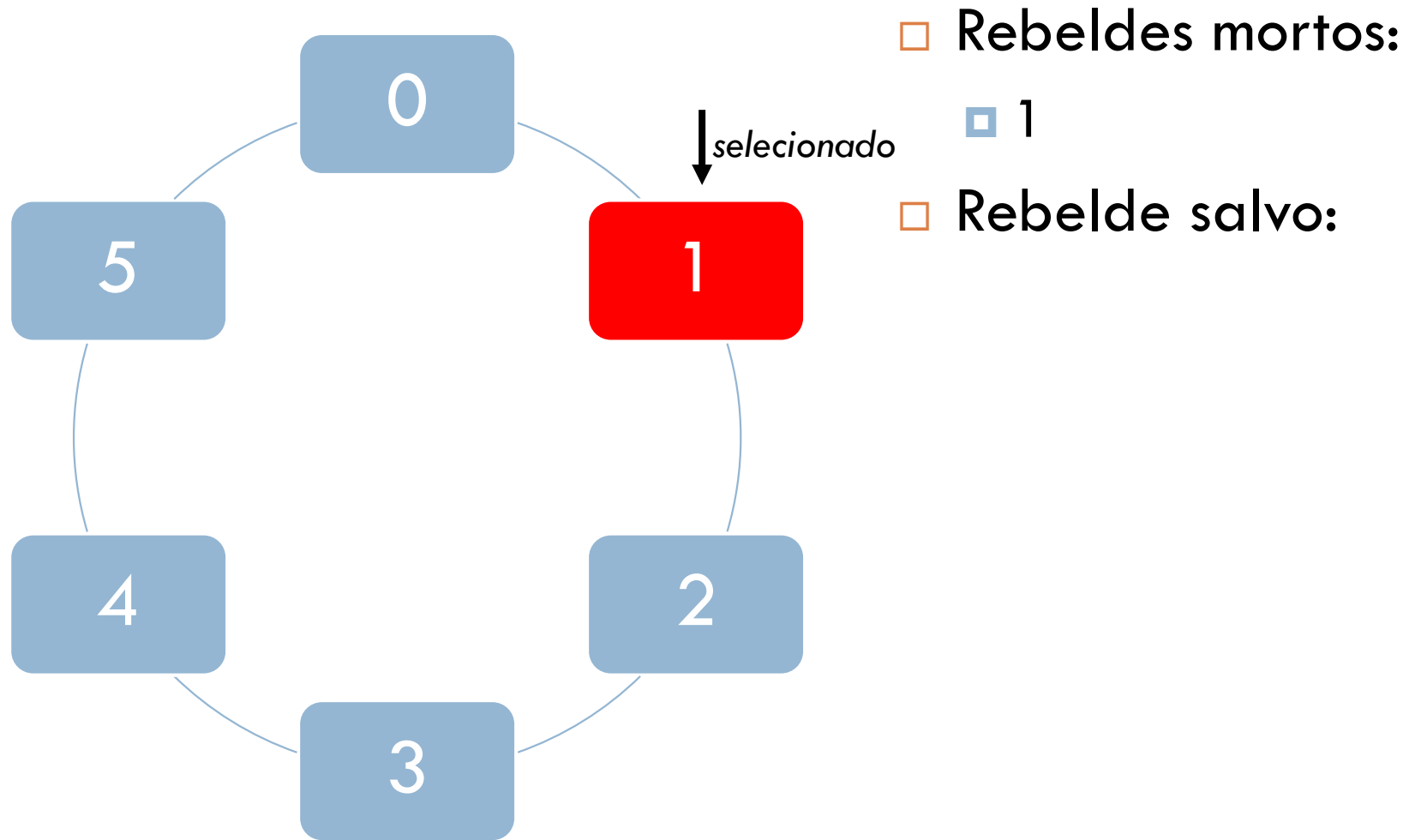


□ Rebeldes mortos:

□ Rebelde salvo:

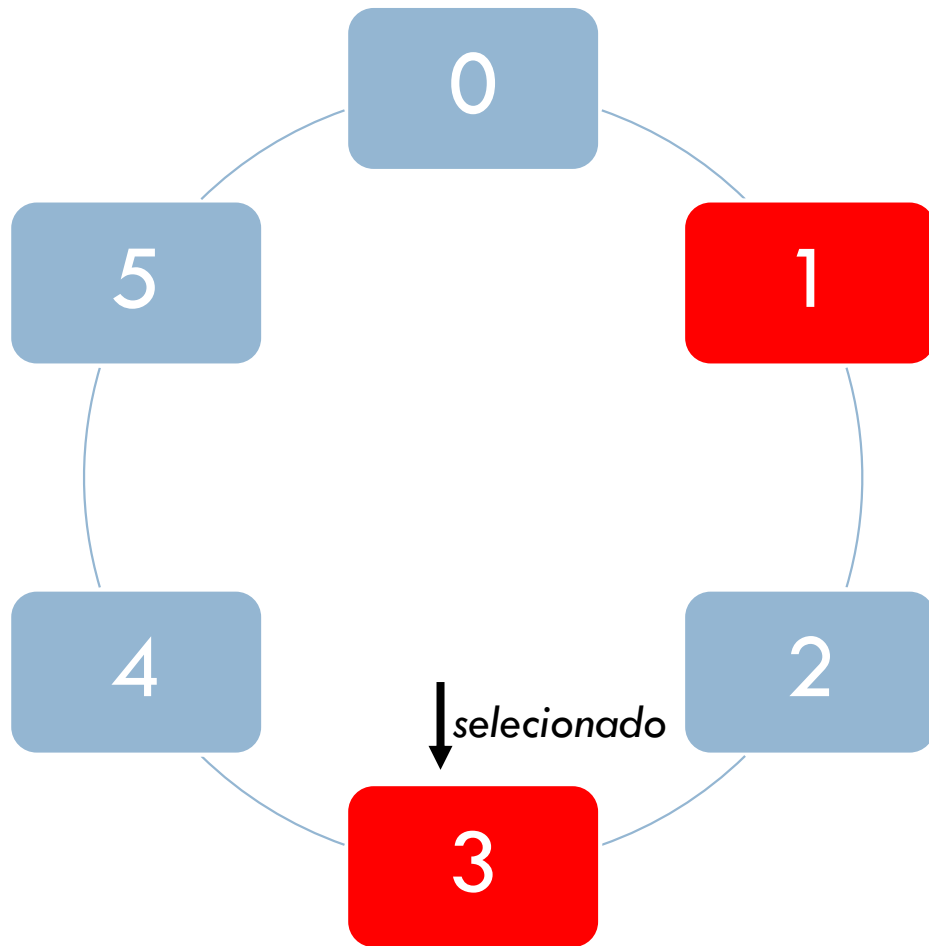
Exemplo do Problema de Josephus

7



Exemplo do Problema de Josephus

8



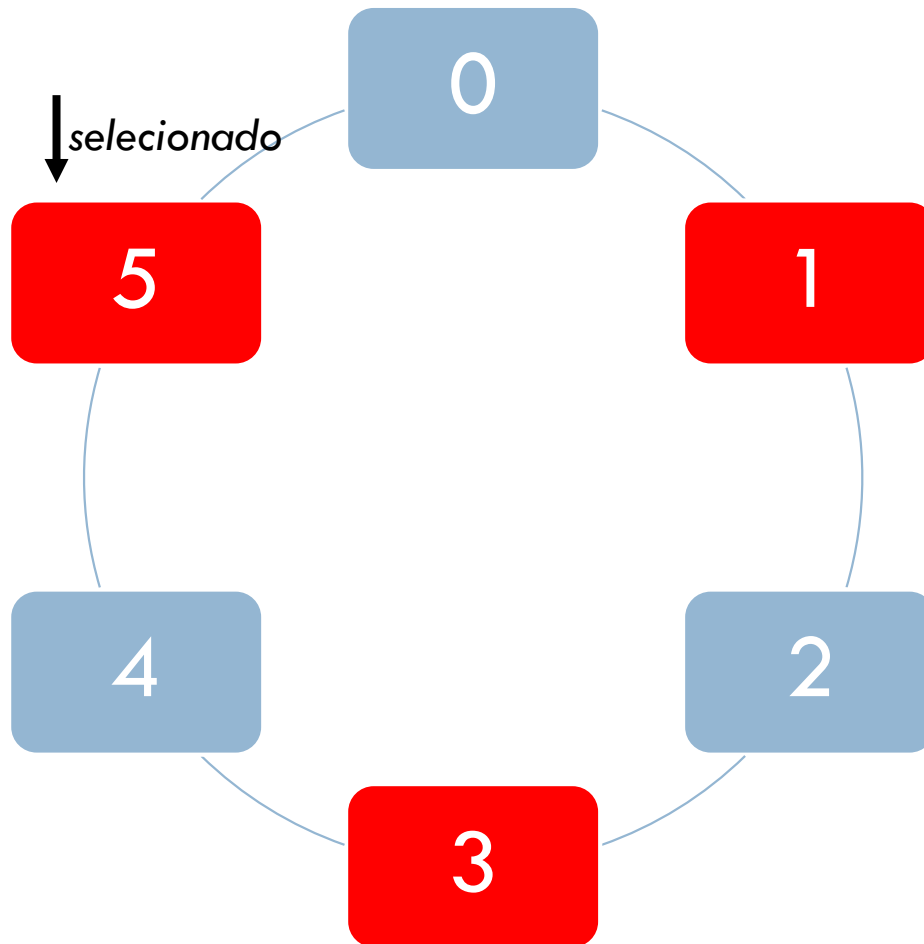
□ Rebeldes mortos:

▣ 1, 3

□ Rebelde salvo:

Exemplo do Problema de Josephus

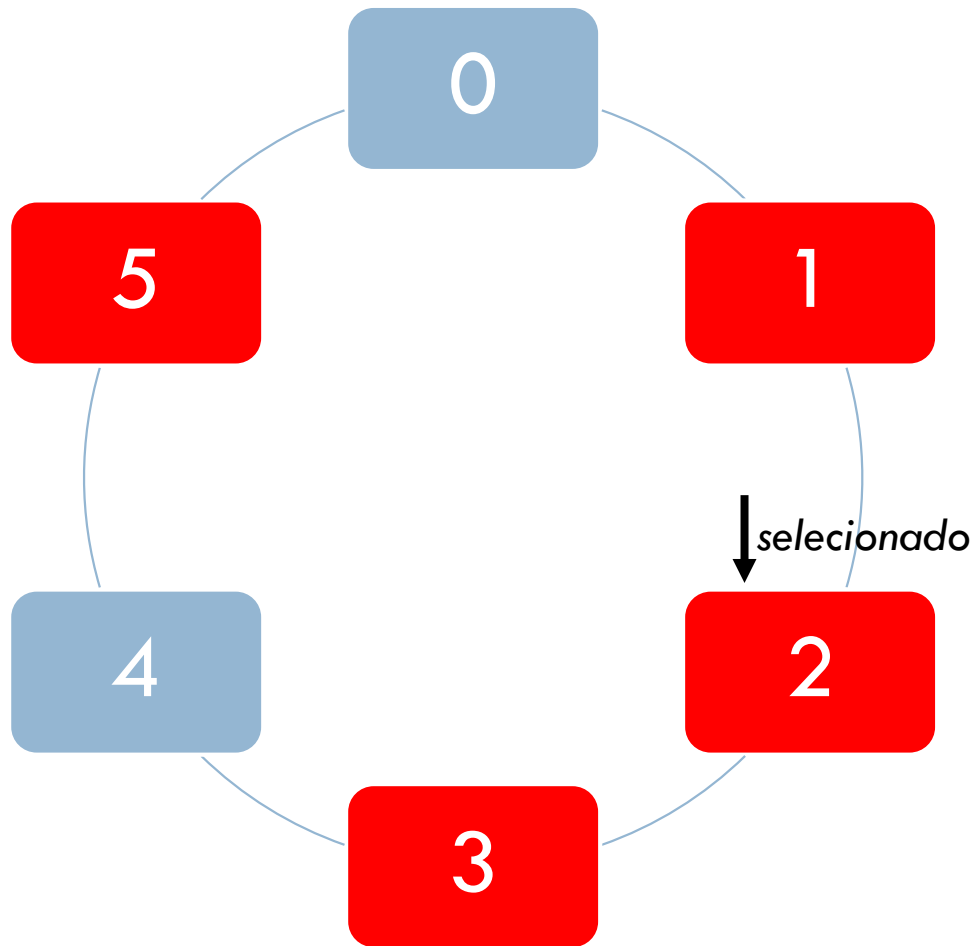
9



- Rebeldes mortos:
 - ▣ 1, 3, 5
- Rebelde salvo:

Exemplo do Problema de Josephus

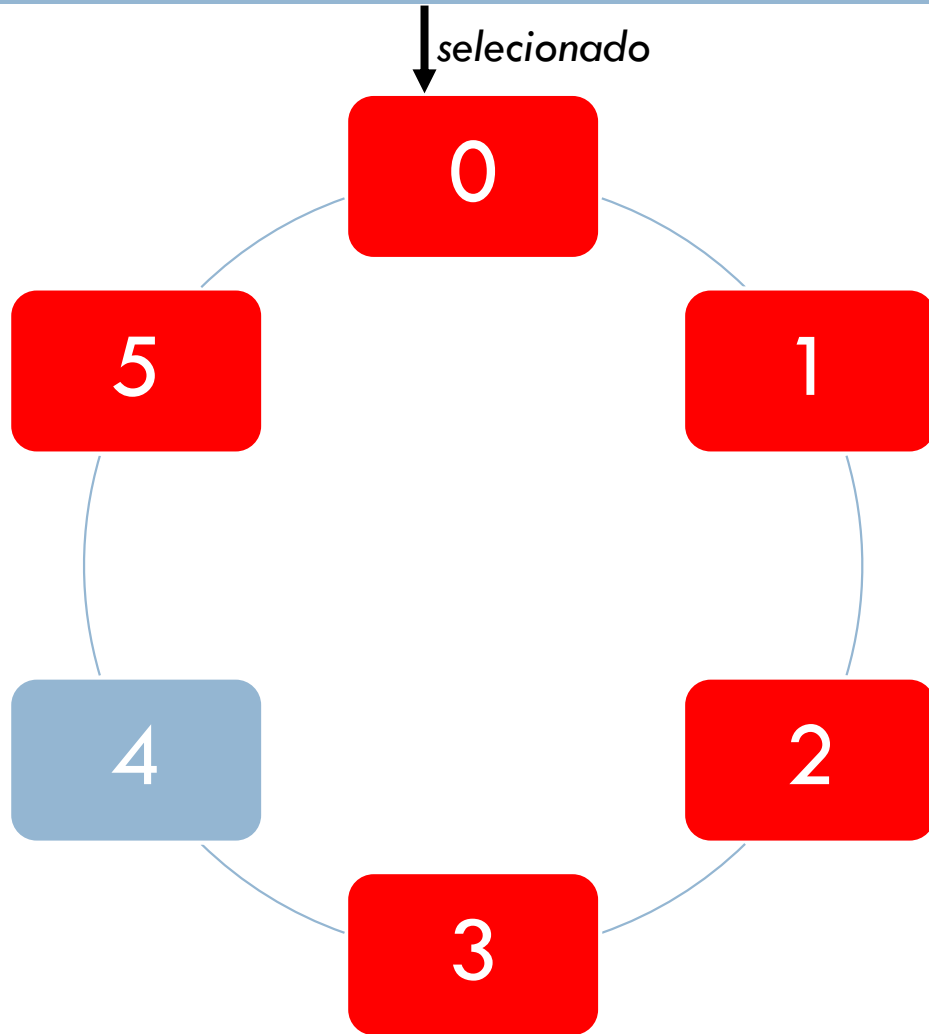
10



- Rebeldes mortos:
 - ▣ 1, 3, 5, 2
- Rebelde salvo:

Exemplo do Problema de Josephus

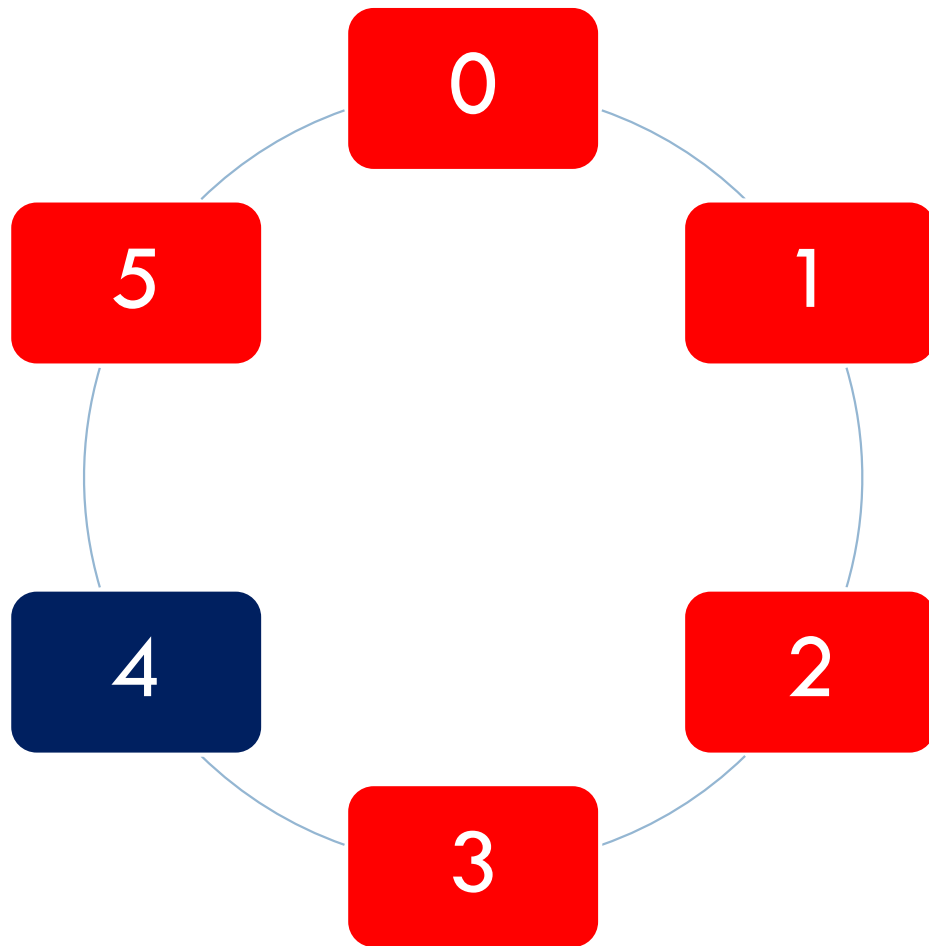
11



- Rebeldes mortos:
 - ▣ 1, 3, 5, 2, 0
- Rebelde salvo:

Exemplo do Problema de Josephus

12



□ Rebeldes mortos:

▣ 1, 3, 5, 2, 0

□ Rebelde salvo:

▣ 4

Qual estrutura ligada
usar para implementar o
problema?

Resolvendo o Problema de Josephus

13

- Use uma lista circular
 - ▣ Cada elemento da lista representa um rebelde
 - Incluir os elementos na lista

Resolvendo o Problema de Josephus

14

- Use uma lista circular
 - ▣ Cada elemento da lista representa um rebelde
 - Incluir os elementos na lista
 - ▣ Percorra a lista circularmente enquanto não sobrar apenas um elemento
 - Efetue as seleções
 - Remova o elemento selecionado
 - Atualize os ponteiros

Uso de lista no Problema de Josephus

15

```
int main() {  
    int i, N = 6, x = 2;  
    //inicializando a lista circular  
    Lista l, *pl; pl=&l;  
    No* novo, atual, ant, temp;
```

Uso de lista no Problema de Josephus

16

```
int main() {  
    int i, N = 6, x = 2;  
    //inicializando a lista circular  
    Lista l, *pl; pl=&l;  
    No* novo, atual, ant, temp;  
    //inserindo no final da lista circular  
    for (i=0; i<N;i++){  
        novo = malloc(sizeof(No)); novo->chave=i;  
        if (i==0) {pl->inicio=novo; atual=pl->inicio;}  
        novo->prox=pl->inicio; //último aponta para o 1º  
        atual->prox=novo;  
        atual=novo;  
    }  
    (...)
```


Uso de lista no Problema de Josephus

18

```
atual = pl->inicio; //inicio da lista
//percorrer até sobrar apenas um elemento
while (atual != atual->prox) {
    for (i = 1; i < x; i++){    ant=atual;
                                atual = atual->prox; }

    //remoção do nó atual
    temp = atual;
    ant->prox = atual->prox;
    atual = atual->prox;
    free(temp);
}
printf("Rebelde salvo: %d\n", atual->chave);
return 0;
}
```

Desafio

19

- Quais modificações são necessárias no procedimento para que após a seleção, o rebelde $x+1$ seja o eliminado?

Polinômios

20

- Polinômio de grau n é uma expressão da seguinte forma:

$$P(x) = \underbrace{a_n}_{\text{coeficiente}} x^{\overbrace{n}^{\text{exponente}}} + \underbrace{a_{n-1} x^{n-1}}_{\text{termo}} + \dots + a_1 x^1 + a_0 x^0$$

Polinômios

21

- Polinômio de grau n é uma expressão da seguinte forma:

$$P(x) = \underbrace{a_n}_{\text{coeficiente}} x^{\overbrace{n}^{\text{exponente}}} + \underbrace{a_{n-1} x^{n-1}}_{\text{termo}} + \dots + a_1 x^1 + a_0 x^0$$

- Exemplos:

$$P_1(x) = 5x^{20} - 3x^5 + 7 \quad \text{e} \quad P_2(x) = 0$$

Representando polinômios

22

Elemento nó

```
typedef struct Termo{  
    double coeficiente;  
    int expoente;  
    struct Termo *prox;  
} Termo;
```

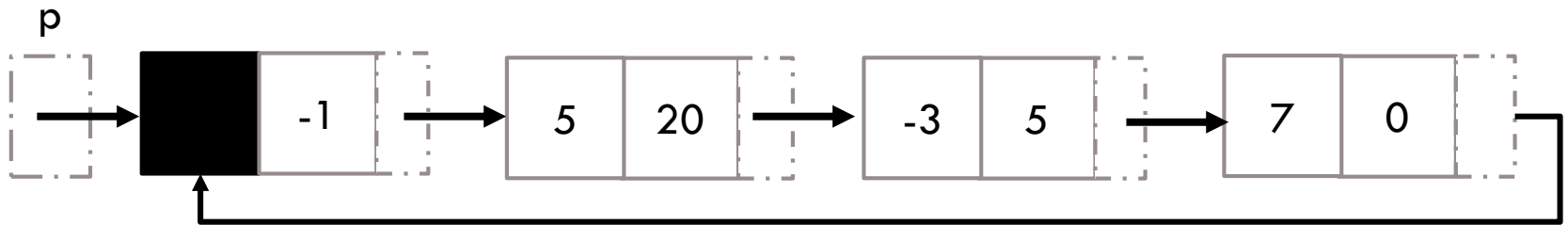
Lista ligada

```
typedef struct Polinomio{  
    Termo *inicio;  
} Polinomio;
```

Uso de lista ligada circular com cabeça

23

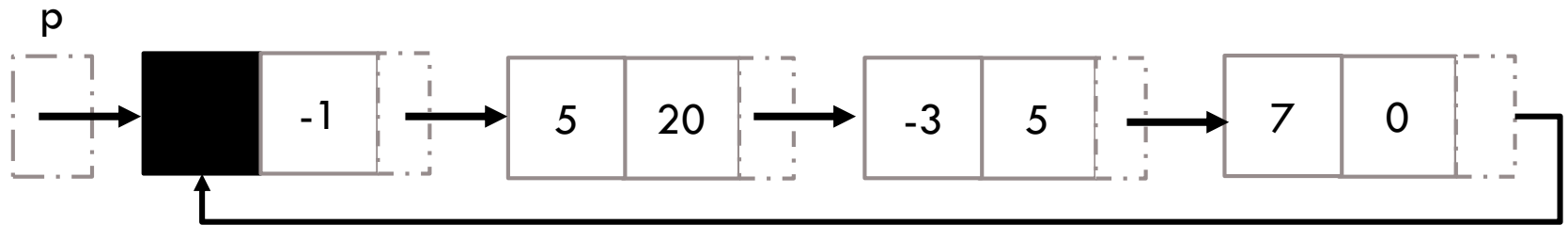
□ $P_1(x) = 5x^{20} - 3x^5 + 7$



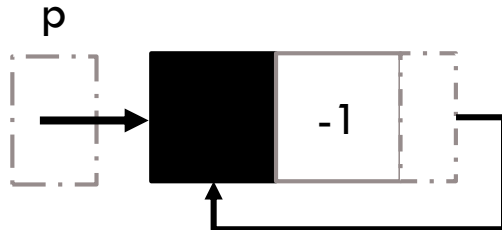
Uso de lista ligada circular com cabeça

24

□ $P_1(x) = 5x^{20} - 3x^5 + 7$



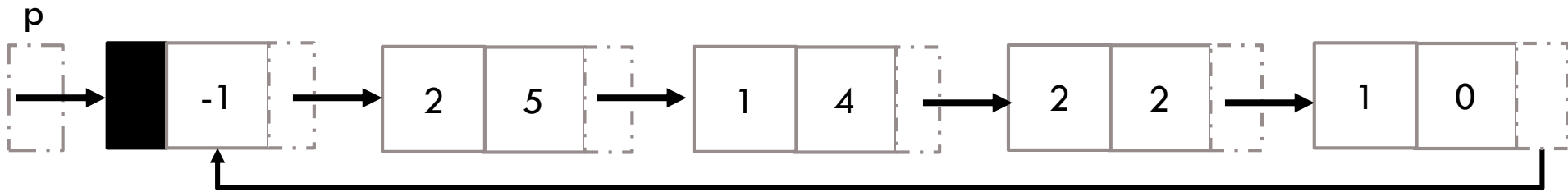
□ $P_2(x) = 0$



Uso de lista ligada circular com cabeça

25

□ $P_3(x) = 2x^5 + x^4 + 2x^2 + 1$



Imprimindo um polinômio

26

```
void imprimir_polynomial(Polynomial *pl){  
    if (pl->inicio->prox==pl->inicio){  
        printf("Polinômio nulo"); return;  
    }  
}
```

Imprimindo um polinômio

27

```
void imprimir_polynomial(Polynomial *pl){
    if (pl->inicio->prox==pl->inicio){
        printf("Polinômio nulo"); return;
    }
    Termo* aux = pl->inicio->prox;
    while (aux->expoente!=-1){
        printf("(coef =%5.1f, expo=%2d) ",
            aux->coeficiente, aux->expoente);
        aux=aux->prox;
    }
    printf("\n");
}
```

Somando polinômios

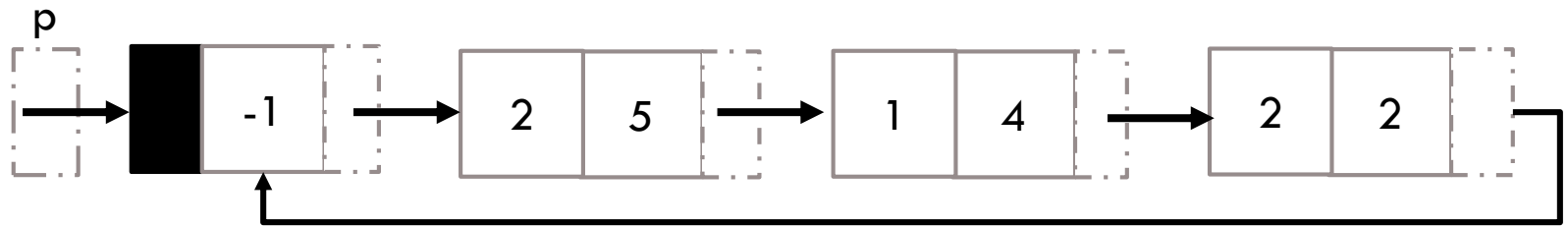
28

- Objetivo: obter a soma de dois polinômios dados, sendo:
 - P_1 e P_2
 - Gerar um polinômio P_r correspondente

Somando polinômios

29

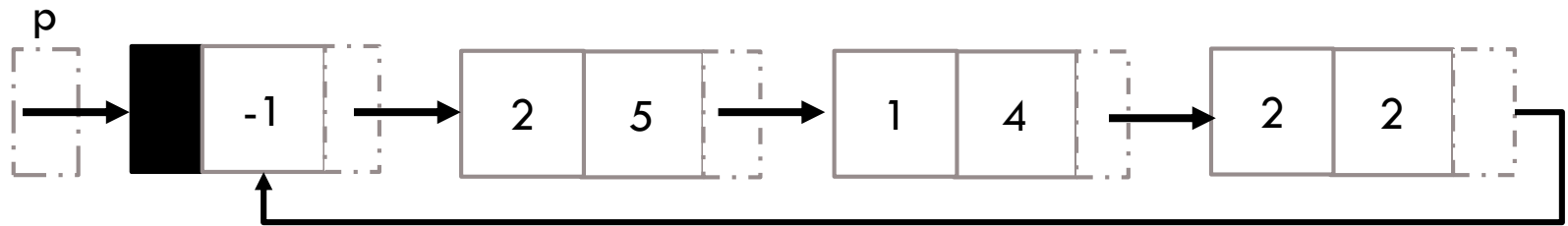
□ $P_1(x) = 2x^5 + x^4 + 2x^2$



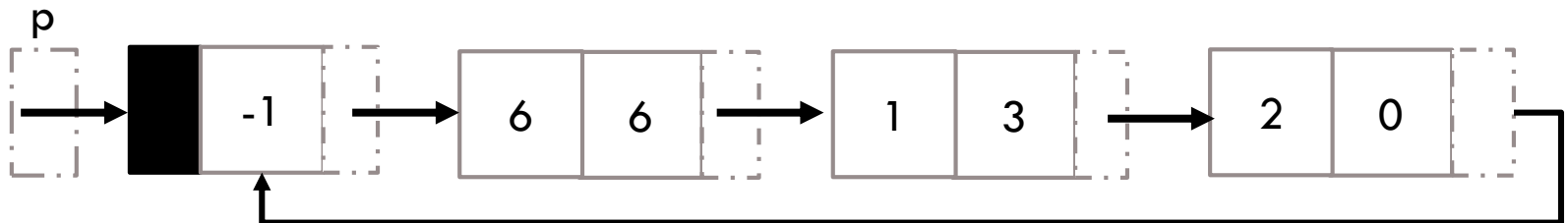
Somando polinômios

30

□ $P_1(x) = 2x^5 + x^4 + 2x^2$



□ $P_2(x) = 6x^6 + x^3 + 2x$



Procedimento para ler um polinômio

31

```
Polinomio* ler_polinomio(){  
    Polinomio p; Polinomio* pl=&p;  
    Termo* atual = iniciar lista cab(pl);  
    int tam, exp;  
    double coef;  
    scanf("%d", &tam);
```

Procedimento para ler um polinômio

32

```
Polinomio* ler_polinomio(){
    Polinomio p; Polinomio* pl=&p;
    Termo* atual = iniciar lista cab(pl);
    int tam, exp;
    double coef;
    scanf("%d", &tam);

    for (i = 0; i < tam; i++) {
        printf("Digite o coeficiente e o expoente: ")
        scanf("%f %d", &coef , &exp);
        atual = inserirTermoPol(atual, coef , exp);
    }
    return pl;
}
```


Procedimento para ler um polinômio

33

```
Polinomio* ler_polinomio(){
    Polinomio p; Polinomio* pl=&p;
    Termo* atual = iniciar lista cab(pl);
    int tam, exp;
    double coef;
    scanf("%d", &tam);

    for (i = 0; i < tam; i++) {
        printf("Digite o coeficiente e o expoente: ")
        scanf("%f %d", &coef , &exp);
        atual = inserirTermoPol(atual, coef , exp);
    }
    return pl;
}
```

Analise como o procedimento de inserção em lista circular deve ser adaptado para este procedimento

Algoritmo para somar polinômios

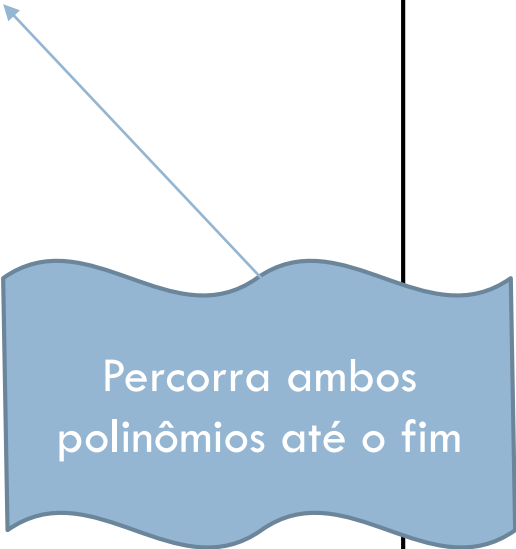
34

```
Polinomio* somar_polinomios(Polinomio *p1, Polinomio *p2){  
    Polinomio pr; Polinomio* plr=&pr;  
    Termo* at_r = iniciar_lista_cab(plr);  
    Termo* at_p1, at_p2;  
    at_p1=p1->inicio->prox;  
    at_p2=p2->inicio->prox;  
  
    //percorrer ambos polinômios até o final  
    (...)
```

Algoritmo para somar polinômios

35

```
while (at_p1->prox!=p1->inicio && at_p2->prox!=p2->inicio){  
    Termo* novo = malloc(sizeof(Termo));
```

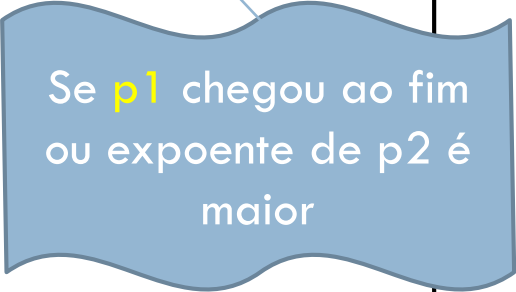


Percorra ambos
polinômios até o fim

Algoritmo para somar polinômios

36

```
while (at_p1->prox!=p1->inicio && at_p2->prox!=p2->inicio){  
    Termo* novo = malloc(sizeof(Termo));  
    if (at_p1->prox==p1->inicio || at_p2->exp > at_p1->exp) {  
        novo->coef = at_p2->coef;  
        novo->exp = at_p2->exp;  
        at_p2=at_p2->prox;  
    }
```

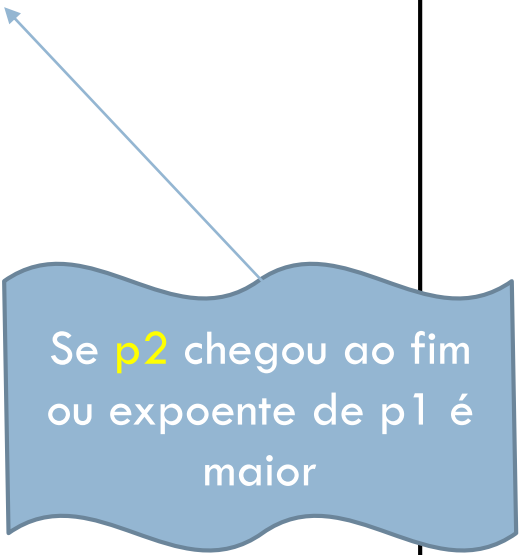


Se **p1** chegou ao fim
ou expoente de p2 é
maior

Algoritmo para somar polinômios

37

```
while (at_p1->prox!=p1->inicio && at_p2->prox!=p2->inicio){  
    Termo* novo = malloc(sizeof(Termo));  
    if (at_p1->prox==p1->inicio || at_p2->exp > at_p1->exp) {  
        novo->coef = at_p2->coef;  
        novo->exp = at_p2->exp;  
        at_p2=at_p2->prox;  
    } else if (at_p2->prox==p2->inicio || at_p1->exp > at_p2->exp) {  
        novo->coef = at_p1->coef;  
        novo->exp = at_p1->exp;  
        at_p1=at_p1->prox;
```



Se **p2** chegou ao fim
ou expoente de p1 é
maior

Algoritmo para somar polinômios

38

```
while (at_p1->prox!=p1->inicio && at_p2->prox!=p2->inicio){
    Termo* novo = malloc(sizeof(Termo));
    if (at_p1->prox==p1->inicio || at_p2->exp > at_p1->exp) {
        novo->coef = at_p2->coef;
        novo->exp = at_p2->exp;
        at_p2=at_p2->prox;
    } else if (at_p2->prox==p2->inicio || at_p1->exp > at_p2->exp) {
        novo->coef = at_p1->coef;
        novo->exp = at_p1->exp;
        at_p1=at_p1->prox;
    } else {
        novo->coef = at_p1->coef+at_p2->coef;
        novo->exp = at_p1->exp;
        at_p1=at_p1->prox; at_p2=at_p2->prox;
    }
}
```

Soma os coeficientes e
mantém o expoente

Algoritmo para somar polinômios

39

```
while (at_p1->prox!=p1->inicio && at_p2->prox!=p2->inicio){
    Termo* novo = malloc(sizeof(Termo));
    if (at_p1->prox==p1->inicio || at_p2->exp > at_p1->exp) {
        novo->coef = at_p2->coef;
        novo->exp = at_p2->exp;
        at_p2=at_p2->prox;
    } else if (at_p2->prox==p2->inicio || at_p1->exp > at_p2->exp) {
        novo->coef = at_p1->coef;
        novo->exp = at_p1->exp;
        at_p1=at_p1->prox;
    } else {
        novo->coef = at_p1->coef+at_p2->coef;
        novo->exp = at_p1->exp;
        at_p1=at_p1->prox; at_p2=at_p2->prox;
    } //inserção no final da lista circular
    at_r->prox=novo;          novo->prox=plr->inicio; at_r=novo;
} return plr;
}
```

Código cliente para soma de polinômios

40

```
int main(){
    Polinomio *p1, *p2, *pr;
    p1 = ler_polinomio();
    imprimir_polinomio(p1);
    p2 = ler_polinomio();
    imprimir_polinomio(p2);
    pr = somar_polinomios(p1, p2);
    imprimir_polinomio(pr);
    return 0;
}
```


Matrizes Esparsas

41

- Estrutura bidimensional composta por m linhas e n colunas

Matrizes Esparsas

42

- Estrutura bidimensional composta por m linhas e n colunas
- Uma matriz é dita esparsa quando o número de elementos “não-nulos” é considerado muito menor que o número total de elementos
 - ▣ Grande parte dos elementos possuem um elemento padrão (e.g., zero)

Lidando com matrizes esparsas

43

$$\begin{vmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{vmatrix}$$

- Como evitar gastar $m \times n$ posições de memória sendo que apenas um pequeno número de elementos tem valor diferente de zero (ou nulo)?

Lidando com matrizes esparsas

44

$$\begin{vmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{vmatrix}$$

- Como evitar gastar $m \times n$ posições de memória sendo que apenas um pequeno número de elementos tem valor diferente de zero (ou nulo)?
- É possível criar uma estrutura para gerenciar matrizes esparsas na qual apenas elementos diferente de zero sejam alocados?

Solução para matrizes esparsas

45

- Uso de listas ligadas para representar a matriz
 - ▣ Algoritmos podem supor que todos elementos não lidos são nulos

Solução para matrizes esparsas

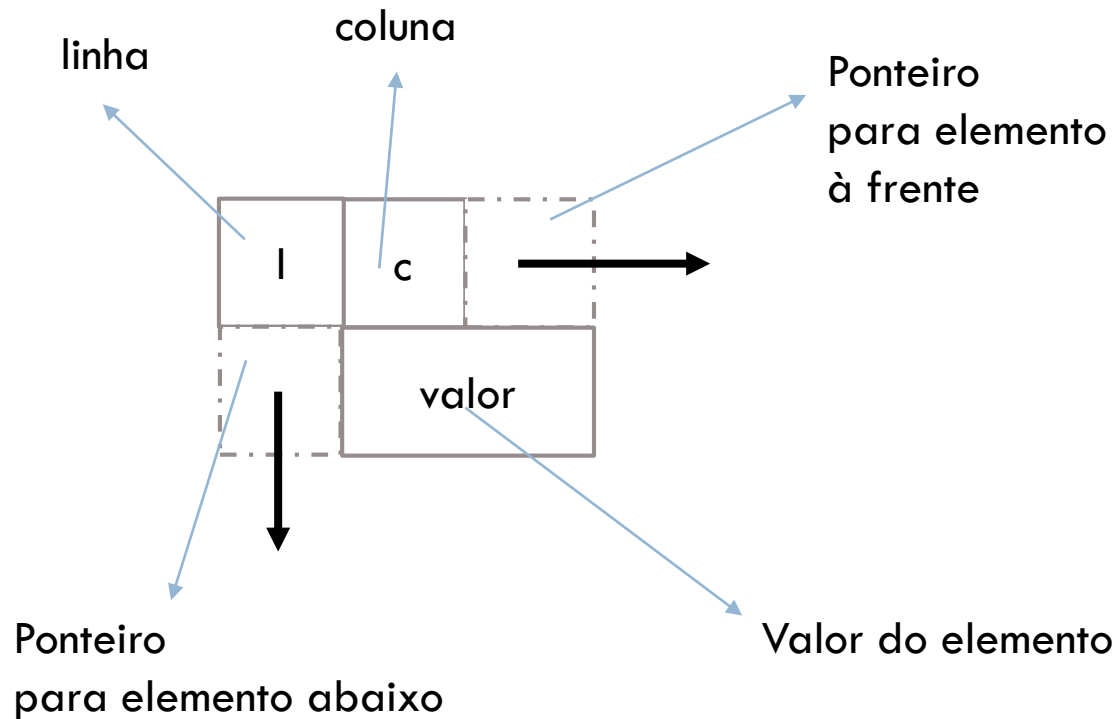
46

- Uso de listas ligadas para representar a matriz
 - ▣ Algoritmos podem supor que todos elementos não lidos são nulos

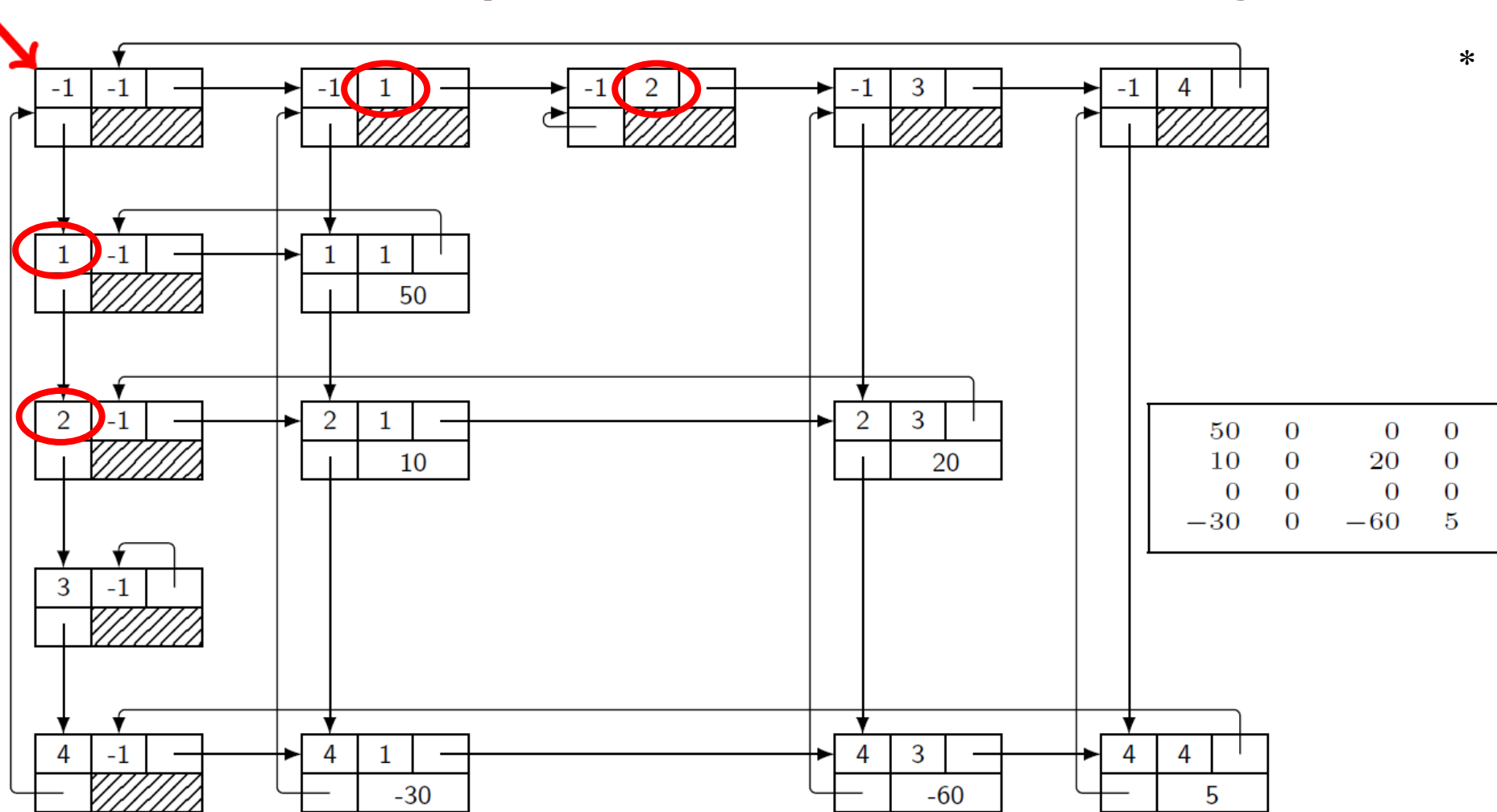
- Cada linha da matriz representada por uma lista ligada
 - ▣ Contém apenas os elementos não-nulos

Solução para matrizes esparsas

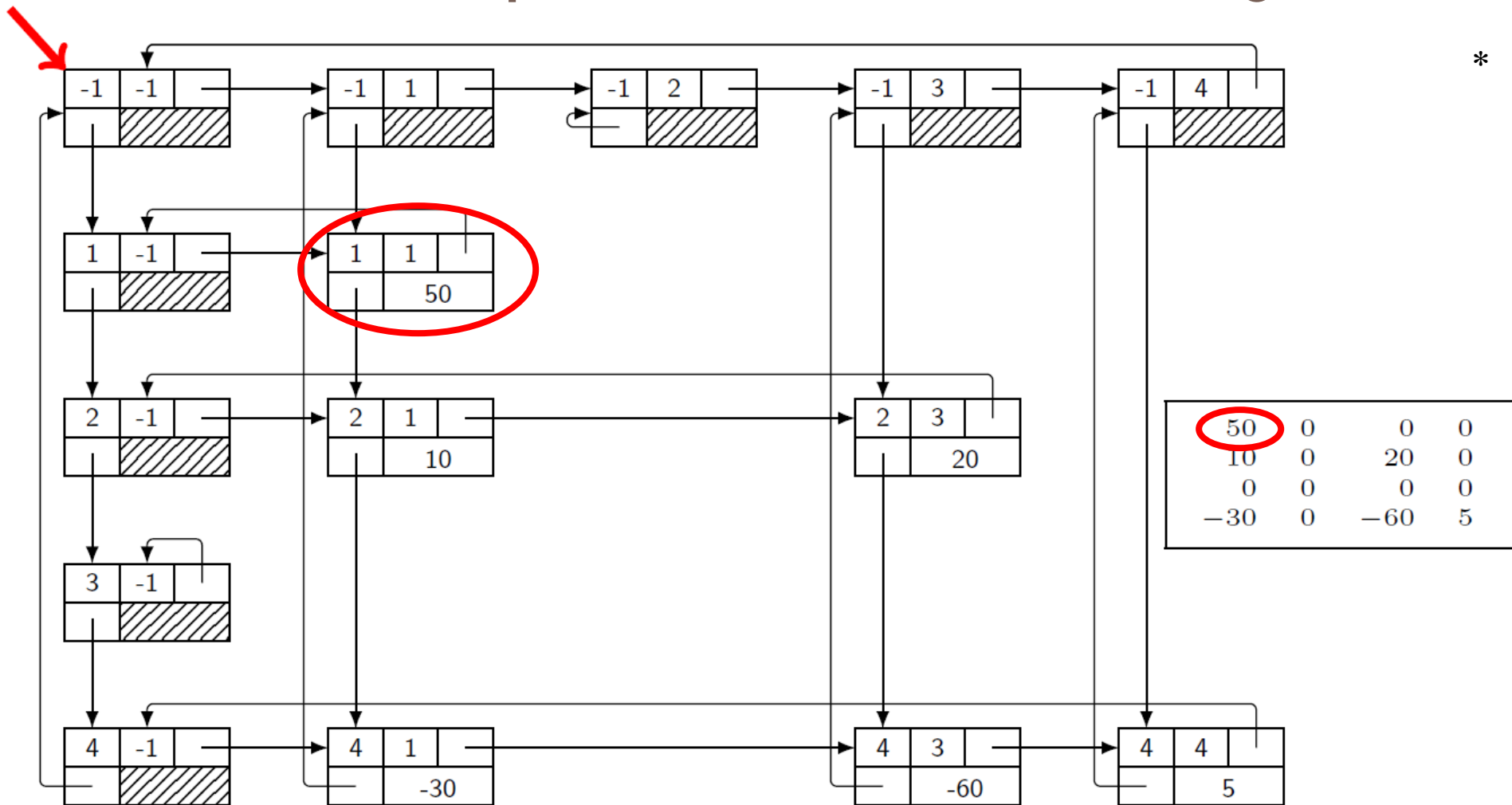
47



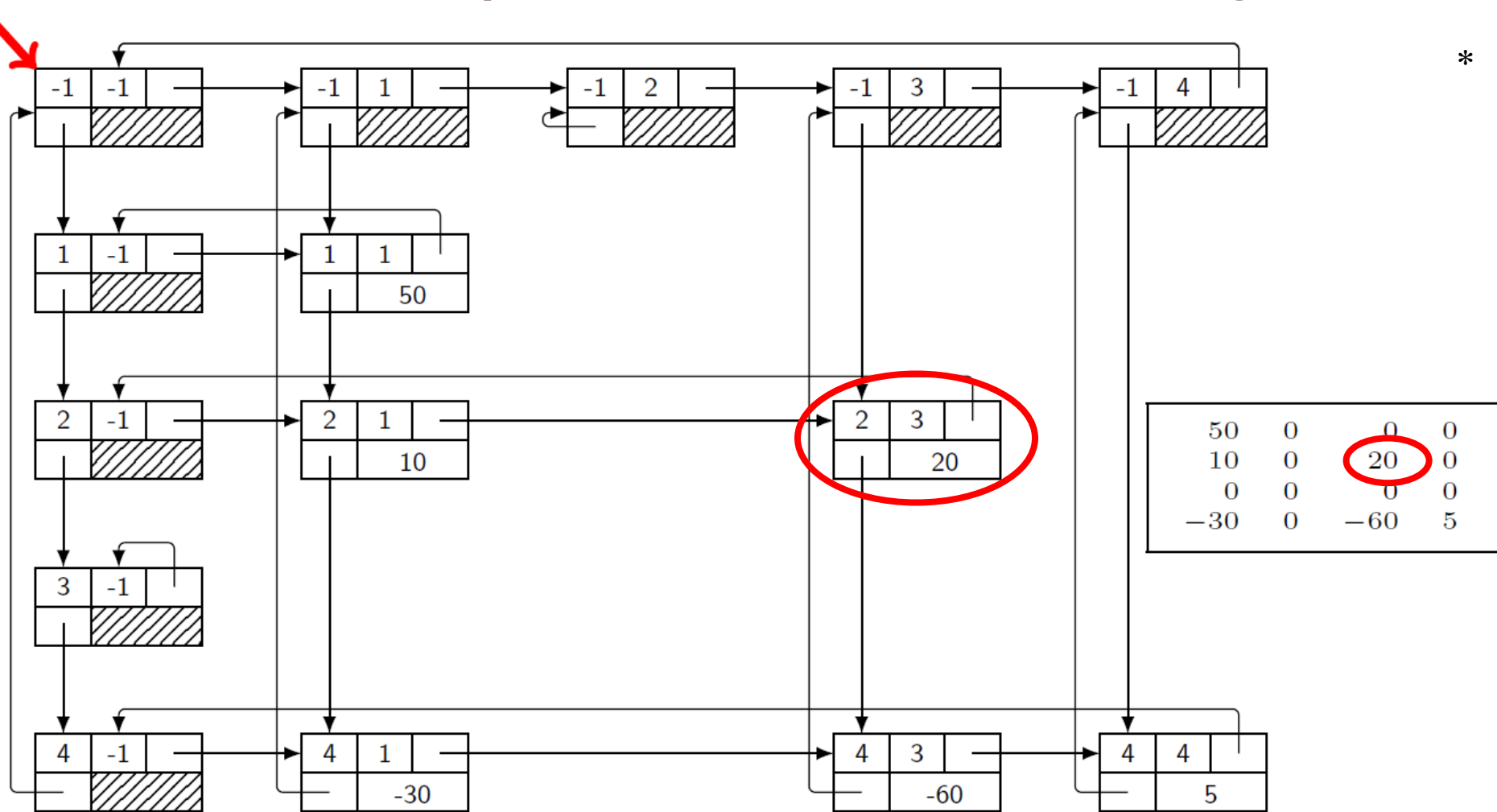
Matrizes esparsas com listas ortogonais



Matrizes esparsas com listas ortogonais



Matrizes esparsas com listas ortogonais



Estrutura da Matriz Esparsa

52

Elemento nó da Matriz

```
typedef struct MatNo{  
    int linha, coluna;  
    double valor;  
    struct MatNo *direita, *abaixo;  
} MatNo;
```

Lista ligada para a Matriz

```
typedef struct Matriz{  
    int linhas, colunas;  
    MatNo *superCabeca;  
} Matriz;
```

Operações sobre matriz esparsa

53

- `Matriz* inicializaMatriz(int linhas, int colunas);`
- `void finalizaMatriz(Matriz *m);`
- `void atribuiValorMatriz(Matriz *m, int i, int j, double valor);`

Operações sobre matriz esparsa

54

- ❑ `Matriz* inicializaMatriz(int linhas, int colunas);`
- ❑ `void finalizaMatriz(Matriz *m);`
- ❑ `void atribuiValorMatriz(Matriz *m, int l, int c, double valor);`
- ❑ `double acessaValorMatriz(Matriz *m, int l, int c);`
- ❑ `Matriz* somaMatrizes(Matriz *a, Matriz *b);`
- ❑ `Matriz* multiplicaMatrizes(Matriz *a, Matriz *b);`

Operação de soma de matrizes

55

```
Matriz* somaMatrizes(Matriz *a, Matriz *b) {  
    Matriz *resultado;  
    int linha, coluna;    double valor_a, valor_b;  
    // Se as matrizes "a" e "b" são compatíveis para soma
```

Operação de soma de matrizes

56

```
Matriz* somaMatrizes(Matriz *a, Matriz *b) {  
    Matriz *resultado;  
    int linha, coluna;    double valor_a, valor_b;  
    // Se as matrizes "a" e "b" são compatíveis para soma  
    if ( a->linhas == b->linhas && a->colunas == b->colunas ) {  
        // Inicializa a matriz de resultado com as dimensões de ambas  
        resultado = inicializaMatriz(a->linhas, a->colunas);  
    }
```


Operação de soma de matrizes

57

```
Matriz* somaMatrizes(Matriz *a, Matriz *b) {  
    Matriz *resultado;  
    int linha, coluna;    double valor_a, valor_b;  
    // Se as matrizes "a" e "b" são compatíveis para soma  
    if ( a->linhas == b->linhas && a->colunas == b->colunas ) {  
        // Inicializa a matriz de resultado com as dimensões de ambas  
        resultado = inicializaMatriz(a->linhas, a->colunas);  
        for(linha=0 ; linha < a->linhas ; linha++) {  
            for(coluna=0 ; coluna < a->colunas ; coluna++) {  
                // Recupera o valor [linha,coluna] de A e B  
                valor_a = acessaValorMatriz(a, linha, coluna);  
                valor_b = acessaValorMatriz(b, linha, coluna);  
                // Insere esse valor na matriz resultado  
                atribuiValorMatriz(resultado, linha, coluna, valor_a + valor_b);  
            }  
        }  
        return resultado; // Retorna a matriz resultado  
    } return NULL;  
}
```

Operação de soma de matrizes

58

```
Matriz* somaMatrizes(Matriz *a, Matriz *b) {  
    Matriz *resultado;  
    int linha, coluna;    double valor_a, valor_b;  
    // Se as matrizes "a" e "b" são compatíveis para soma  
    if ( a->linhas == b->linhas && a->colunas == b->colunas ) {  
        // Inicializa a matriz de resultado com as dimensões de ambas  
        resultado = inicializaMatriz(a->linhas, a->colunas);  
        for(linha=0 ; linha < a->linhas ; linha++) {  
            for(coluna=0 ; coluna < a->colunas ; coluna++) {  
                // Recupera o valor [linha,coluna] de A e B  
                valor_a = acessaValorMatriz(a, linha, coluna);  
                valor_b = acessaValorMatriz(b, linha, coluna);  
                // Insere esse valor na matriz resultado  
                atribuiValorMatriz(resultado, linha, coluna, valor_a + valor_b);  
            }  
        }  
        return resultado; // Retorna a matriz resultado  
    }  
    return NULL;  
}
```

Esta operação
poderia ser feita
diferente?

59

Algoritmos gerais em listas

Cópia de listas

60

- Dado uma lista, gerar uma cópia da mesma

Elemento nó

```
typedef struct No{  
    int chave;  
    struct No *prox;  
} No;
```

Lista ligada

```
typedef struct Lista{  
    No *inicio;  
} Lista;
```

- `Lista* copiarLista(Lista *lp);`

Copiar lista ligada

61

```
Lista* copiarLista(Lista *lp){  
    Lista l_copia, *p_copia;  p_copia=&l_copia;  
    No *aux, *ultimo;  aux = lp->inicio;  ultimo=NULL;
```

Copiar lista ligada

62

```
Lista* copiarLista(Lista *lp){
    Lista l_copia, *p_copia;  p_copia=&l_copia;
    No *aux, *ultimo;  aux = lp->inicio;  ultimo=NULL;
    while(aux!=NULL){
        No *novo = malloc(sizeof(No));
        if (novo!=NULL){ //tem memória
            novo->chave=aux->chave; novo->prox=NULL;
```

Copiar lista ligada

63

```
Lista* copiarLista(Lista *lp){
    Lista l_copia, *p_copia;  p_copia=&l_copia;
    No *aux, *ultimo;  aux = lp->inicio;  ultimo=NULL;
    while(aux!=NULL){
        No *novo = malloc(sizeof(No));
        if (novo!=NULL){ //tem memória
            novo->chave=aux->chave; novo->prox=NULL;
            //inserir novo Nó no final da lista cópia
            if (ultimo!=NULL){
                ultimo->prox=novo;
            }else{ //primeiro Nó
                p_copia->inicio=novo;
            } ultimo=novo; aux=aux->prox; //novo é o ultimo
        }
    }
    return p_copia;
}
```

Inversão de Listas

64

- Dado uma lista a , gere uma nova lista b no qual o primeiro elemento da lista a corresponde ao último da lista b , e vice-versa
- `Lista* inverterLista(Lista *lp);`

Inverter lista ligada

65

```
Lista* inverterLista(Lista *lp){  
    No *aux, *ant, *inver;  
    aux = lp->inicio; ant=NULL; inver=NULL;
```

Inverter lista ligada

66

```
Lista* inverterLista(Lista *lp){
    No *aux, *ant, *inver;
    aux = lp->inicio; ant=NULL; inver=NULL;
    while(aux!=NULL){
        ant=aux;
        aux=ant->prox;
        ant->prox=inver;
        inver=ant;
    }
    lp->inicio=inver; return lp;
}
```

Concatenação de Listas

67

- Considere uma lista a , e uma lista b
- O último elemento da lista a aponta para o primeiro elemento da lista b
- `Lista* concatenarListas(Lista *lp_a, Lista *lp_b);`

Concatenar lista ligada

68

```
Lista* concatenarListas(Lista *lp_a, Lista *lp_b){  
    if (lp_a->inicio==NULL){ //lista a vazia  
        return lp_b;  
    }  
}
```

Concatenar lista ligada

69

```
Lista* concatenarListas(Lista *lp_a, Lista *lp_b){  
    if (lp_a->inicio==NULL){//lista a vazia  
        return lp_b;  
    }else{  
        No* aux=lp_a->inicio;  
        //fazer aux apontar para o último elemento da lista a  
        while (aux->prox!=NULL)  
            aux=aux->prox;  
    }
```

Concatenar lista ligada

70

```
Lista* concatenarListas(Lista *lp_a, Lista *lp_b){
    if (lp_a->inicio==NULL){
        return lp_b;
    }else{
        No* aux=lp_a->inicio;
        //fazer aux apontar para o último elemento da lista a
        while (aux->prox!=NULL)
            aux=aux->prox;
        //último da lista a aponta para o primeiro da lista b
        aux->prox=lp_b->inicio;
    }
    lp_b->inicio=NULL; //invalida lista b
    return lp_a; //resultado lista a concatenada com a lista b
}
```

Síntese

71

- Estruturas ligadas são úteis para resolver diversos problemas
 - ▣ Estudamos o problema de Josephus, Polinômios e Matrizes Esparsas

Síntese

72

- Estruturas ligadas são úteis para resolver diversos problemas
 - ▣ Estudamos o problema de Josephus, Polinômios e Matrizes Esparsas

- Para além de operações tradicionais sobre listas, há problemas que requerem algoritmos que manipulem listas
 - ▣ Estudamos copia, inversão e concatenação de listas