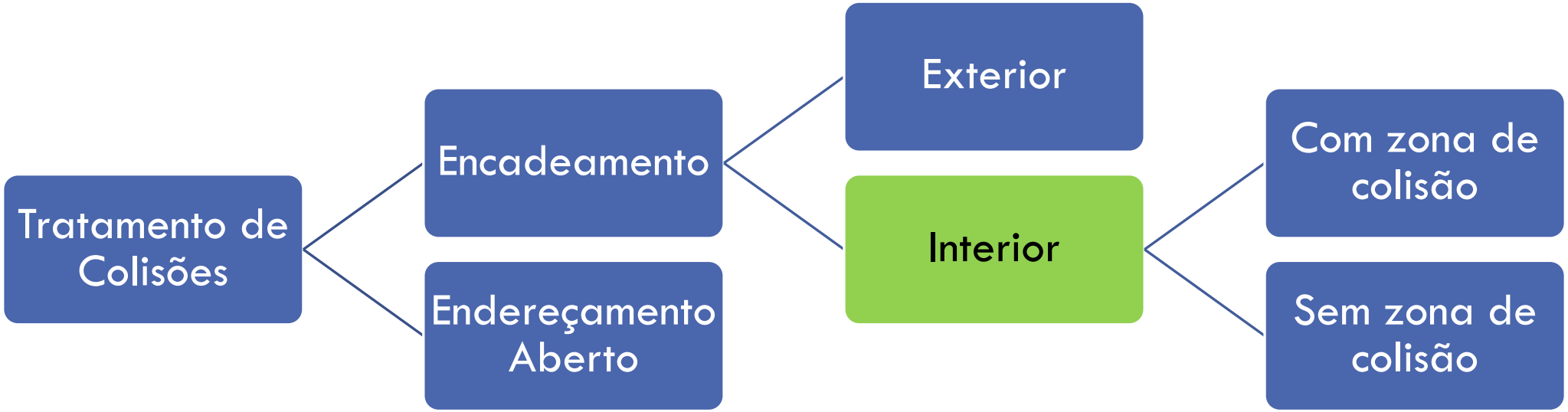
TABELAS HASH
TRATAMENTO DE COLISÕES POR ENCADEAMENTO INTERIOR

Vanessa Braganholo

Estruturas de Dados e Seus Algoritmos


Tratamento de Colisões
Encadeamento
Endereçamento Aberto
Exterior
Interior
Com zona de colisão
Sem zona de colisão

ENCADEAMENTO INTERIOR

Em algumas aplicações não é desejável manter uma estrutura externa à tabela hash, ou seja, não se pode permitir que o espaço de registros cresça indefinidamente

Nesse caso, ainda assim pode-se fazer tratamento de colisões

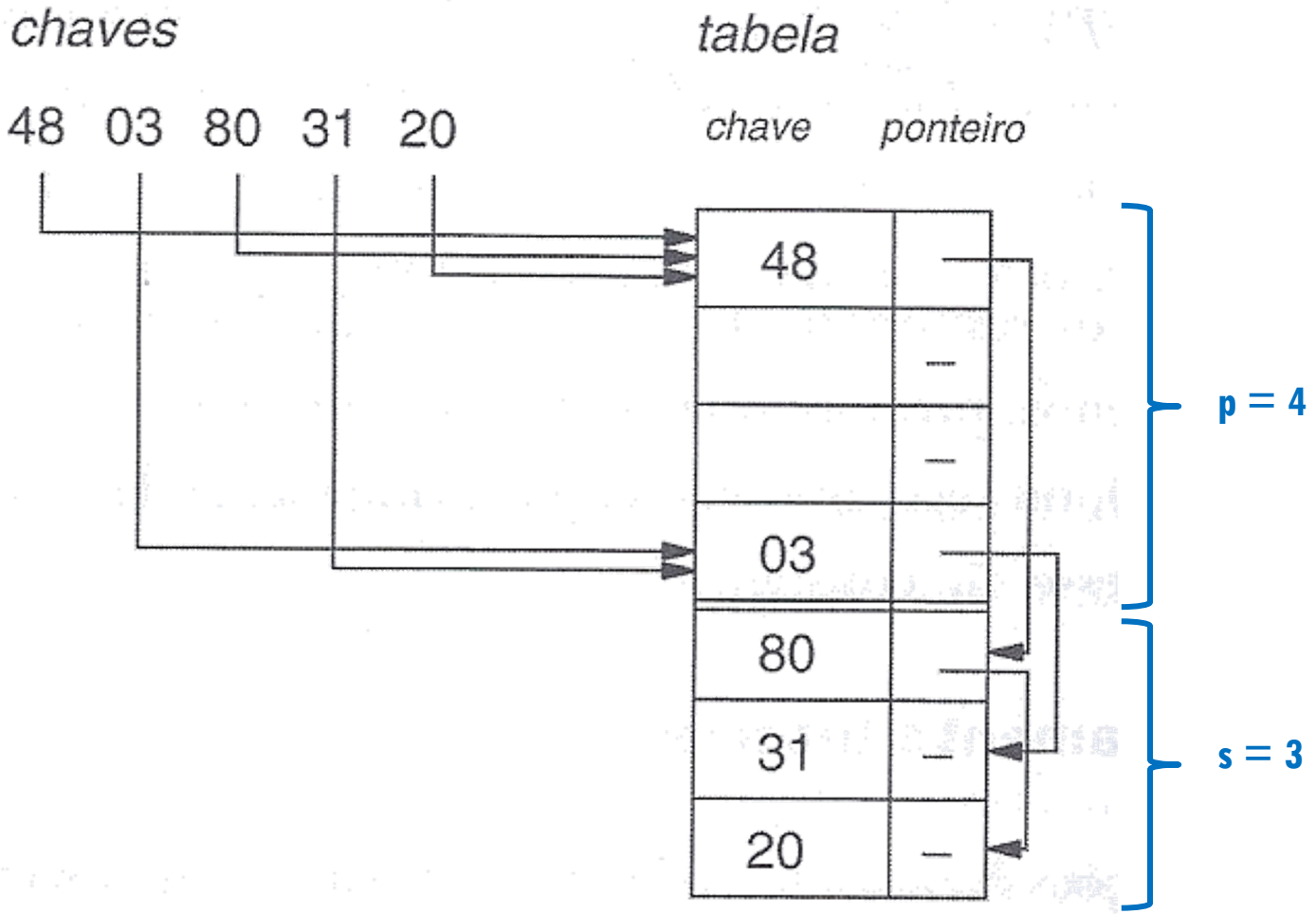
ENCADEAMENTO INTERIOR COM ZONA DE COLISÕES

Dividir a tabela em duas zonas

- Uma de endereços-base, de tamanho **p**
- Uma de colisão, de tamanho **s**
- **p + s = m**

- Função de hash deve gerar endereços no intervalo **[0, p-1]**
- Cada nó tem a mesma estrutura utilizada no Encadeamento Exterior (tabela de dados)

EXEMPLO: ENCADEAMENTO INTERIOR COM ZONA DE COLISÕES



h(x) = x mod 4

Fonte: Fig. 10.6, pag 242

OVERFLOW

Em um dado momento, pode acontecer de não haver mais espaço para inserir um novo registro

REFLEXÕES

Qual deve ser a relação entre o tamanho de **p** e **s**?

- O que acontece quando **p** é muito grande, e **s** muito pequeno?
- O que acontece quando **p** é muito pequeno, e **s** muito grande?

- Pensem nos casos extremos:
  - p = m-1; s = 1
  - p = 1; s = m – 1

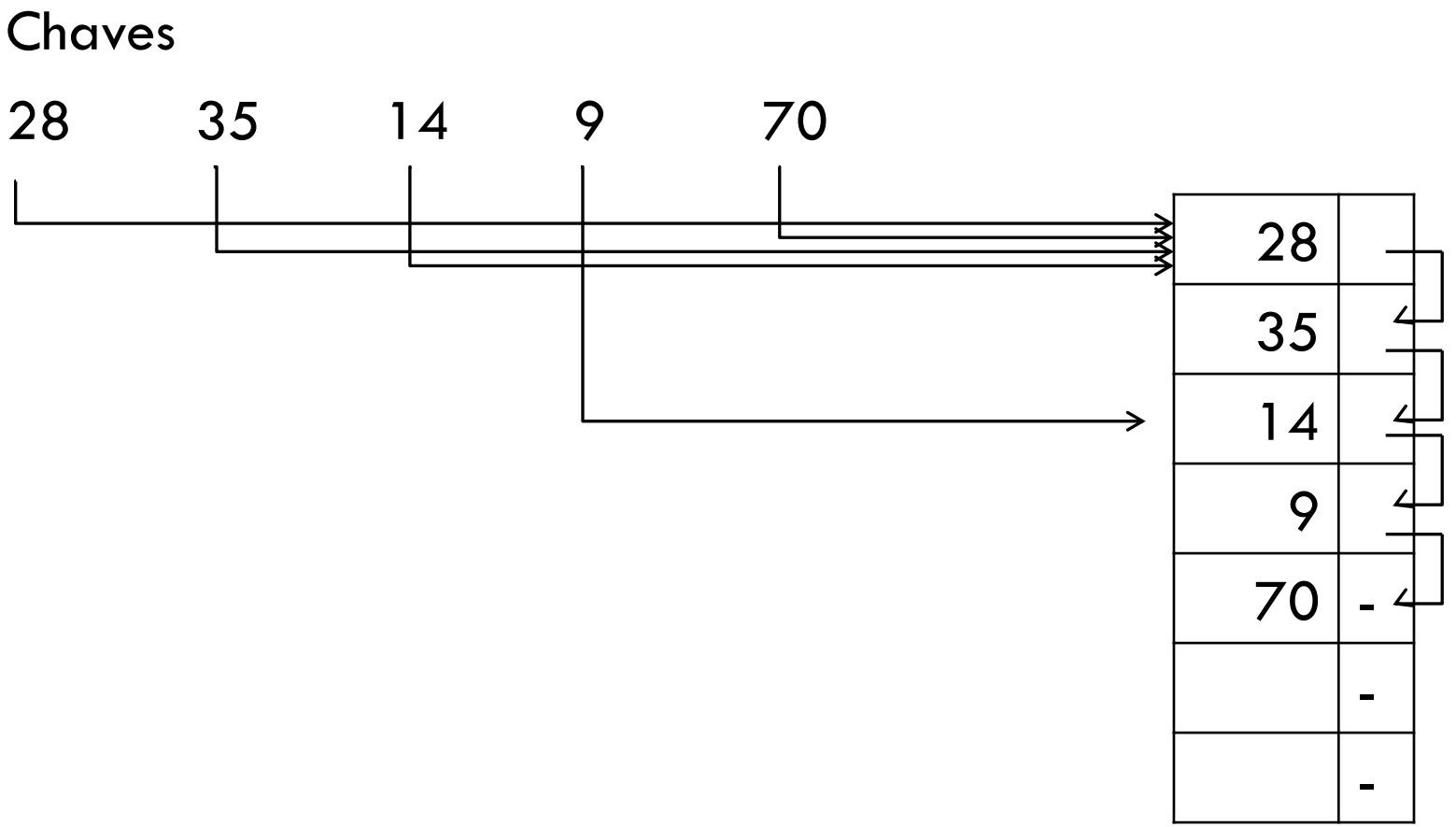
ENCADEAMENTO INTERIOR **SEM** ZONA DE COLISÕES

Outra opção de solução é não separar uma zona específica para colisões

- Qualquer endereço da tabela pode ser de base ou de colisão
- Quando ocorre colisão a chave é inserida no **primeiro compartimento vazio** a partir do compartimento em que ocorreu a colisão
- Efeito indesejado: **colisões secundárias**
  - Colisões secundárias são provenientes da coincidência de endereços para chaves que não são sinônimas

EXEMPLO: ENCADEAMENTO INTERIOR **SEM** ZONA DE COLISÕES

$h(x) = x \bmod 7$



Note que a Fig. 10.7, pag 243 do livro busca compartimentos livres de baixo para cima

IMPLEMENTAÇÃO EM MEMÓRIA PRINCIPAL

```
#define LIBERADO 0
#define OCUPADO 1


typedef struct aluno {
    int matricula;
    float cr;
    int prox;
    int ocupado;
} TAluno;


//Hash é um vetor que será alocado dinamicamente
typedef TAluno *Hash;
```

INICIALIZAÇÃO

```
TAluno *aloca(int mat, float cr, int status, int prox) {
    TAluno *novo = (TAluno *) malloc(sizeof(TAluno));
    novo->matricula = mat;
    novo->cr = cr;
    novo->ocupado = status;
    novo->prox = prox;
    return novo;
}

void inicializa(Hash *tab, int m) {
    int i;
    for (i = 0; i < m; i++) {
        tab[i] = aloca(-1, -1, LIBERADO, -1);
    }
}
```

BUSCA EM ENCADEAMENTO INTERIOR

```
/*
  Função busca assume que a tabela tenha sido inicializada
  da seguinte maneira:
    T[i].ocupado = LIBERADO, e
    T[i].pont = -1, para 0 < i < m-1

  RETORNO:
     Se chave x for encontrada, achou = 1,
     função retorna endereço onde x foi encontrada

     Se chave x não for encontrada, achou = 0, e há duas
     possibilidades para valor retornado pela função:
         endereço de algum compartimento livre, encontrado
                na lista encadeada associada a h(mat)
         -1 se não for encontrado endereço livre
*/
```

Fonte: Implementação baseada no algoritmo 10.1, pag 244 (algoritmo no livro contém pequeno erro)

```
int busca(Hash *tab, int m, int mat, int *achou) {
    *achou = -1;
    int temp = -1;
    int end = hash(mat, m);
    while (*achou == -1) {
        TAluno *aluno = tab[end];
        if (!aluno->ocupado) {//achou compartimento livre -- guarda para
 retorná-lo caso chave não seja encontrada
            temp = end;
        }
        if (aluno->matricula == mat && aluno->ocupado) {
            //achou chave procurada
            *achou = 1;
        } else {
            if (aluno->prox == -1) {
                //chegou no final da lista encadeada
                *achou = 0;
                end = temp;
            } else {
                //avança para o próximo
                end = aluno->prox;
            }
        }
    }
    return end;
}
```

INSERÇÃO EM ENCADEAMENTO INTERIOR

```
/* Função assume que pos é o endereço onde
   será efetuada a inserção. Para efeitos de
   escolha de pos, a tabela foi considerada
   como circular, isto é, o compartimento 0 é
   o seguinte ao m-1
*/
```

Ver implementação no site da disciplina

Fonte: Implementação baseada no algoritmo 10.2, pag 244

EXCLUSÃO EM ENCADEAMENTO INTERIOR

```
void exclui(Hash *tab, int m, int mat) {
     int achou;
     int end = busca(tab, m, mat, &achou);
     if (achou) {
          //remove marcando flag para liberado
          tab[end]->ocupado = LIBERADO;
     } else {
          printf("Matrícula não encontrada. Remoção não realizada!");
     }
 }
```

Fonte: Implementação baseada no algoritmo 10.3, pag 245

EXERCÍCIOS

1. Desenhe a tabela hash (em disco) resultante das seguintes operações (cumulativas) usando o algoritmo de inserção em **Tabela Hash com Encadeamento Interior SEM zona de colisão**. Considere que a tabela tem tamanho 7 e a função de hash usa o método da divisão.

(a) Inserir as chaves 10, 3, 5, 7, 12, 6, 14

(b) Inserir as chaves 4, 8

2. Repita o exercício anterior usando **Tabela Hash com Encadeamento Interior COM zona de colisão**. Considere que a zona de colisão tem tamanho 3.

REFERÊNCIA

Szwarcfiter, J.; Markezon, L. Estruturas de Dados e seus Algoritmos, 3a. ed. LTC. Cap. 10