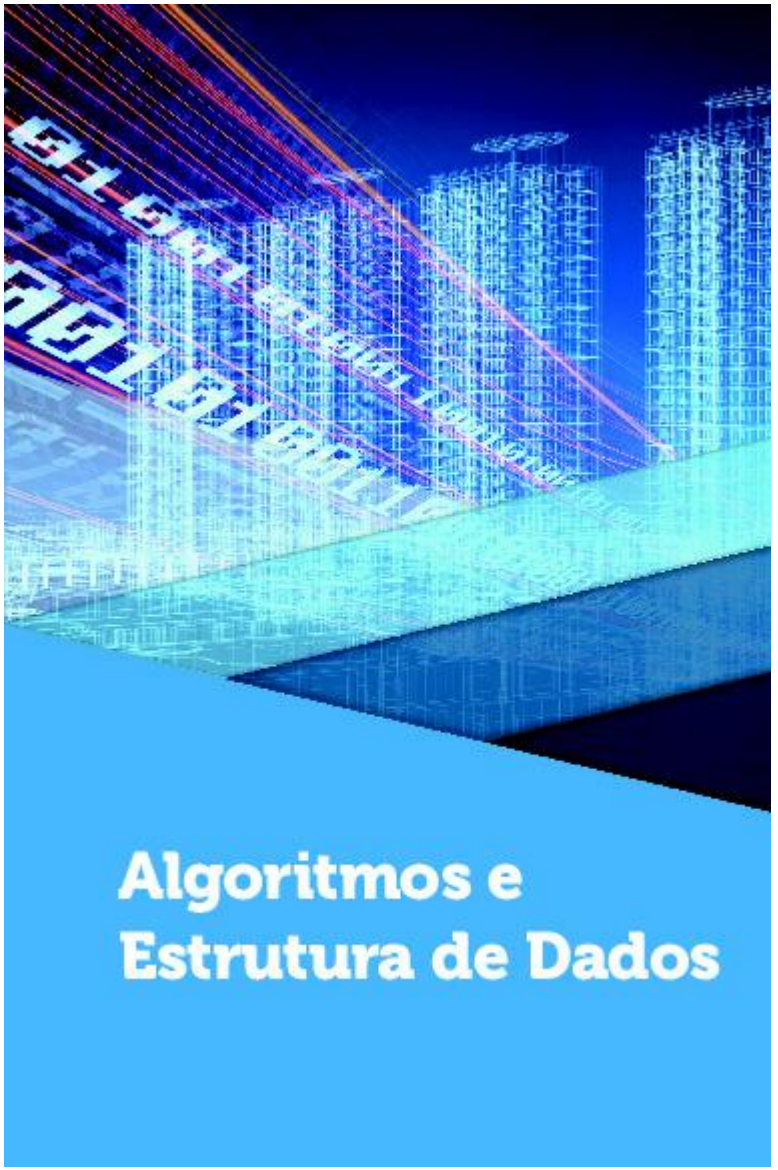
Estruturas de Dados

Eduardo Furlan Miranda

2024-02-01



ROVAI, K. R. **Algoritmos e estrutura de dados**. EDE, 2018.
ISBN 978-85-522-0660-6

- Listas Ligadas
  - Definição e Elementos de Listas Ligadas
  - Operações com Listas Ligadas
  - Listas Duplamente Ligadas

- Pilhas e filas
  - Definição, elementos e regras de pilhas e filas
  - Operações e problemas com pilhas
  - Operações e problemas com filas

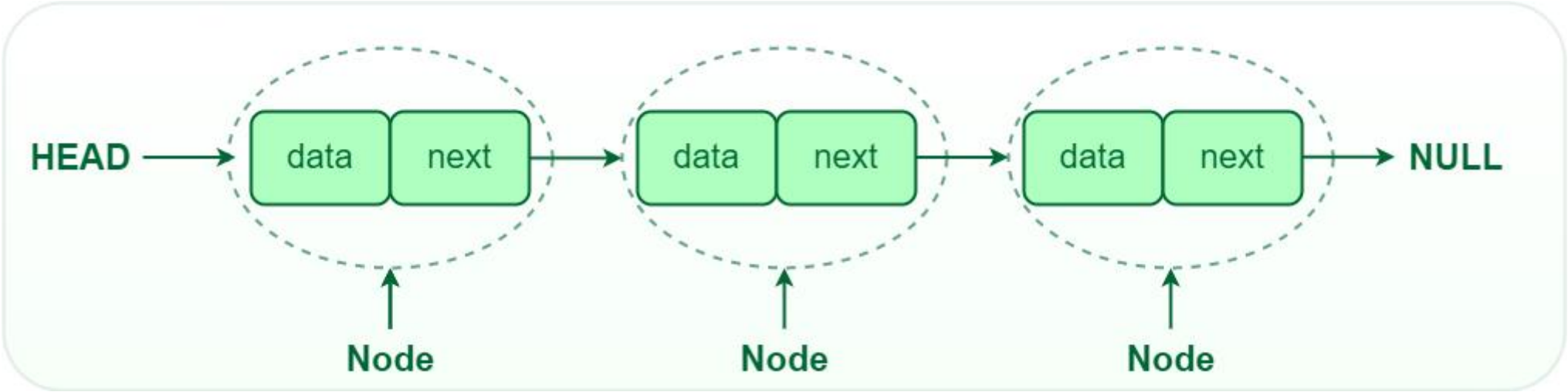
- Tabelas de Espalhamento
  - Definição e Usos de Tabela de Espalhamento
  - Operações em Tabelas de Espalhamento
  - Otimização de Tabelas de Espalhamento

- Armazenamento associativo
  - Definição e usos de Mapas de Armazenamento
  - Mapas com Lista
  - Mapas com Espalhamento

- “&” = “endereço”
- “*” = “ponteiro”
- “->” = “ponteiro” (usado com *struct* ou *union*)
  - (pointer variable) -> (variable) = value;

```
union Movie_info {
    int id;
    float net_val;
};
int main() {
    union Movie_info * M;
    M = ... malloc ...
    M -> net_val = 125.45;
    printf("\n NET VALUE: %.1f", M -> net_val);
}
```

Lista ligada



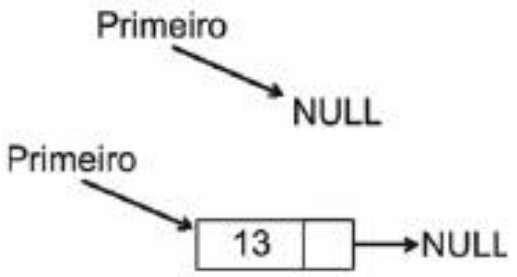

```
struct node {
    int val;
    struct node * next;
};

typedef struct node prt;
```

```
...
while (prt != NULL) {
    if (ptr -> val == 7) {
        found = true;
        break;
    } else {
        ptr = ptr -> next;
    }
}
...
```

Operações com listas ligadas

- Inserir um novo elemento no início da lista

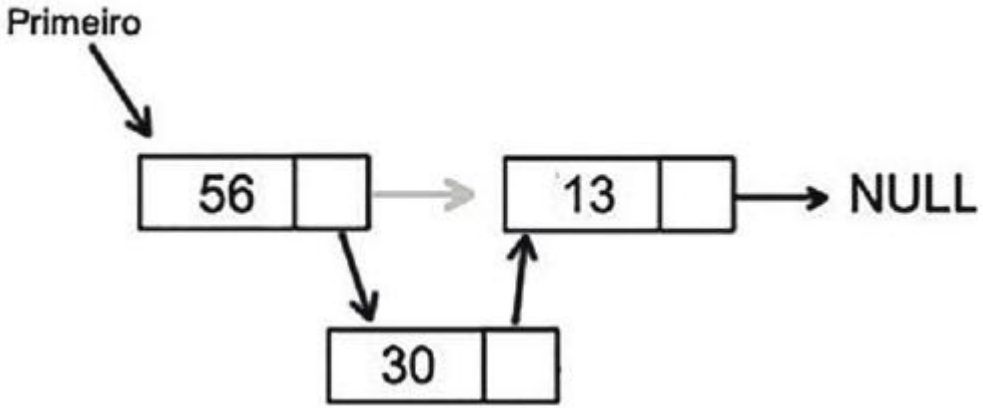
```
Lista * inserir(Lista * l, int i) {
    Lista * novo = (Lista * ) malloc(sizeof(Lista));
    novo -> info = i;
    novo -> prox = l;
    return novo;
}
```




```
int main() {
    Lista * listaFinal;
    listaFinal = inicializar();
    listaFinal = inserir(listaFinal, 13);
    listaFinal = inserir(listaFinal, 56);
}
```

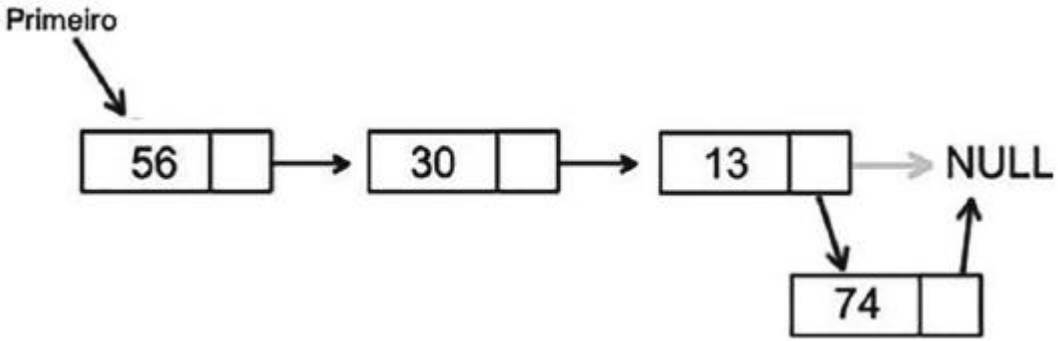
- Inserir um novo elemento no meio da lista

```
Lista * inserirPosicao(Lista * l, int pos, int v) {
    int cont = 1;
    Lista * p = l;
    Lista * novo = (Lista * ) malloc(sizeof(Lista));
    while (cont != pos) {
        p = p -> prox;
        cont++;
    }
    novo -> info = v;
    novo -> prox = p -> prox;
    p -> prox = novo;
    return l;
}
```

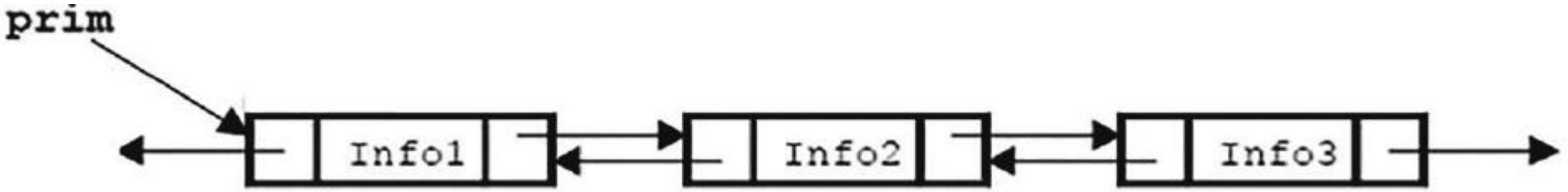


- Inserir um novo elemento no fim da lista

```
Lista * inserirFim(Lista * l, int v) {
    Lista * p = l;
    Lista * novo = (Lista * ) malloc(sizeof(Lista));
    while (p -> prox != NULL) {
        p = p -> prox;
        cont++;
    }
    novo -> info = v;
    novo -> prox = p -> prox;
    p -> prox = novo;
    return l;
}
```



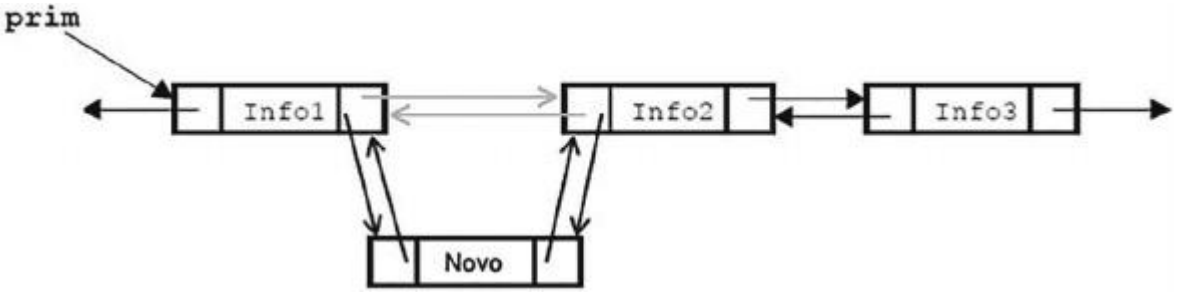
Listas Duplamente Ligadas



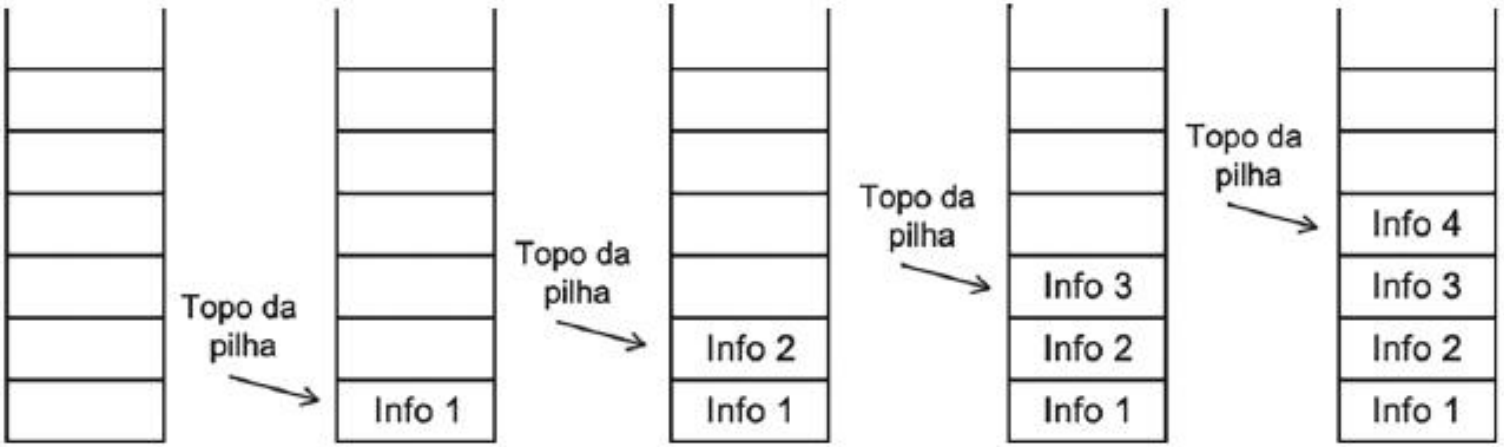

```
struct lista {
        int info;
        struct lista* ant;
        struct lista* prox;
};
typedef struct lista Lista;
```

```
Lista* inserirPosicao(Lista* l, int pos, int v){
    int i, cont = 1;
    Lista *p = l;
    Lista* novo = (Lista*)malloc(sizeof(Lista));
    Lista* temp = (Lista*)malloc(sizeof(Lista));

    while (cont != pos){
            p = p -> prox;
            cont++;
    }

    novo -> info = v;
    temp = p -> prox;
    p -> prox = novo;
    novo -> ant = p;
    novo -> prox = temp;
    temp -> ant = novo;
    return l;
}
```

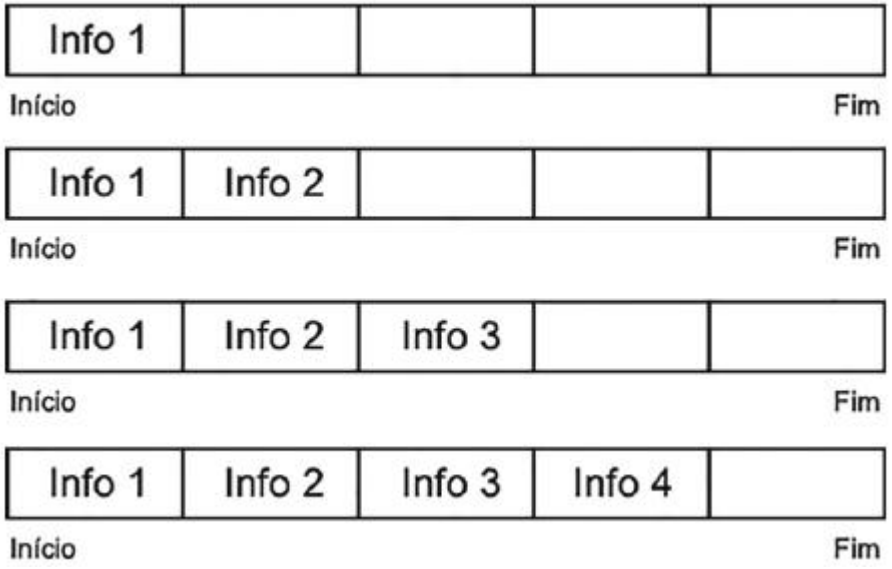
- Inserindo no meio da lista



Pilhas e filas

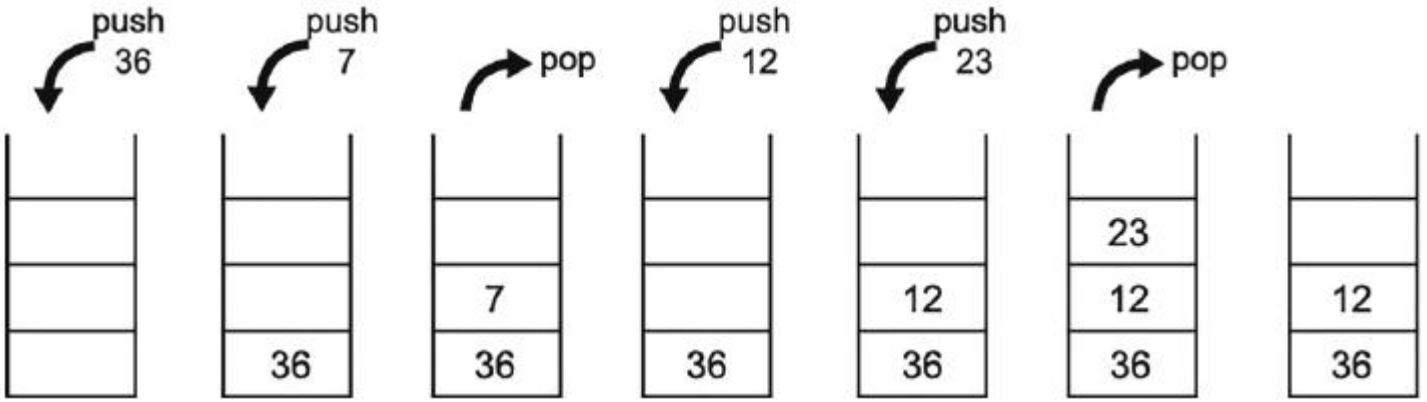


Insere e retira no topo



Insere em uma ponta e retira na outra

Operações e problemas com pilhas

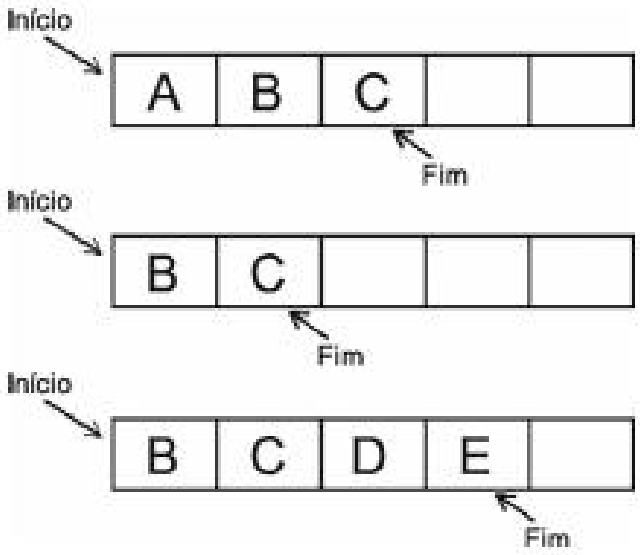


Inserir

```
void push_pilha(struct Pilha *p, float v){
    p -> topo++;
    p -> proxElem [p -> topo] = v;
}
```

Problemas que podem usar pilhas:
- Jogos que precisam retornar a um ponto depois de realizar uma tarefa
- Ordenação
- Algoritmos que devem armazenar um ponto de retorno
- Chamadas de funções em C

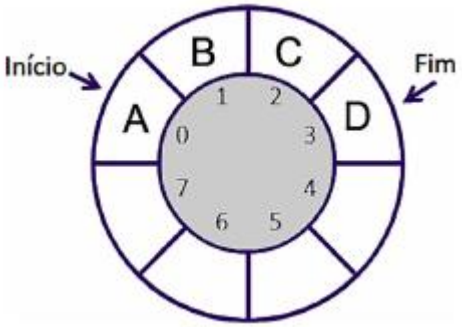
Operações e problemas com filas



Remover elementos

```
float remove_fila (Fila* f)(
        char elem;
        if (fila_vazia(f)){
                printf("A Fila esta vazia\n");
                exit(1);
        }
        elem = f -> vet[f -> ini];
        f -> ini = (f -> ini + 1) % N;
        f -> n--;
        return elem;
}
```

Filas circulares




```
/* Vamos definir a constante N com valor de 10 */
#define N 10
struct filacirc {
    /* Criação da estrutura da Fila Circular */
    int tam, ini, fim;
    char vet[N];
};
typedef struct filacirc FilaCirc;
/* Função para inicializar a Fila */
void inicia_fila(FilaCirc * f) {
    f -> tam = 0;
    f -> ini = 1;
    f -> fim = 0;
}
```

```
/* Função para inserir na Fila */
void insere_fila(FilaCirc * f, char elem) {
    if (f -> tam == N - 1) {
        /* Verifica se a Fila está completa */
        printf("A fila esta cheia\ n");
    } else {
        /* Caso a Fila não esteja completa, inserimos o elemento */
        f -> fim = (f -> fim % (N - 1)) + 1;
        f -> vet[f -> fim] = elem;
        f -> tam++;
    }
}
int fila_vazia(FilaCirc * f) {
    return (f -> tam == 0); /* Retorna verdadeiro se a Fila estiver vazia */
}
char remove_fila(FilaCirc * f) {
    if (fila_vazia(f)) {
        /* Verifica se a Fila está vazia*/
        printf("Fila vazia\ n");
    } else {
        /* Caso a Fila contenha elemento, é removido o primeiro */
        f -> ini = (f -> ini % (N - 1)) + 1;
        f -> tam--;
    }
}
```

Tabela Hash (Dispersão/Espalhamento)

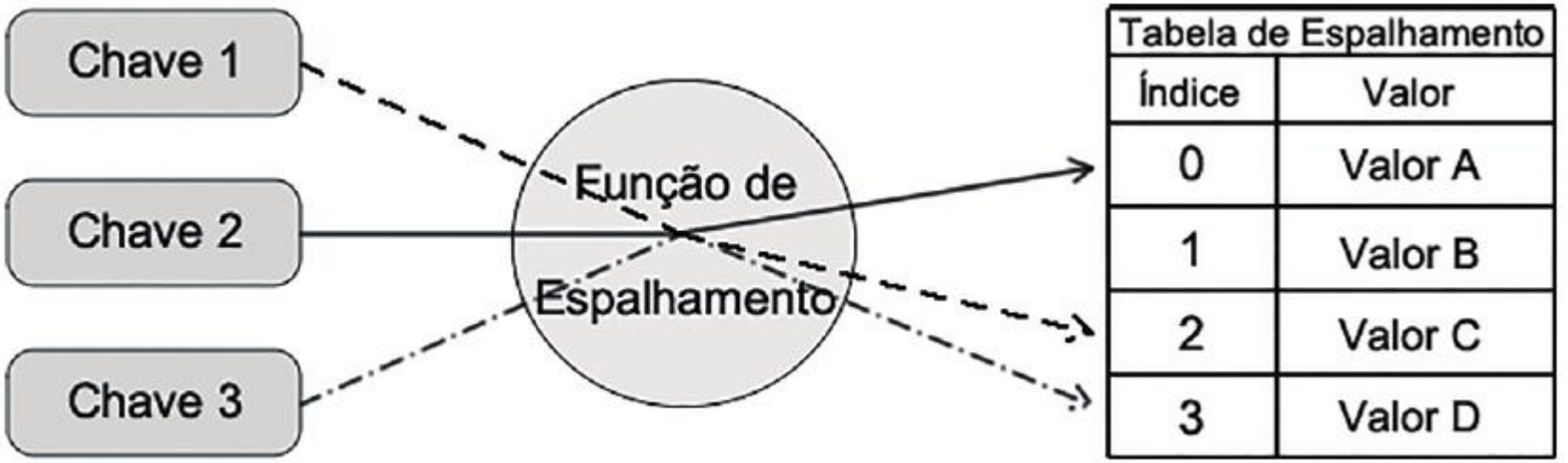
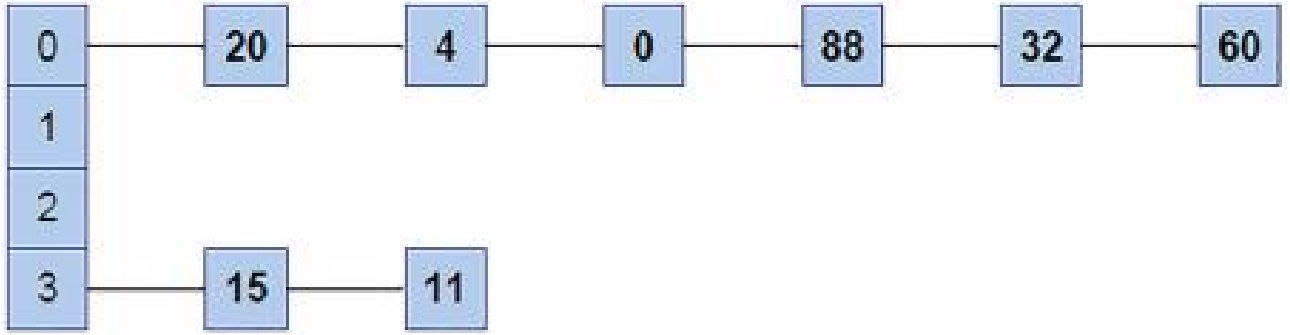


Figura 3.6 | Exemplo de uma Tabela de Espalhamento com Lista Ligada



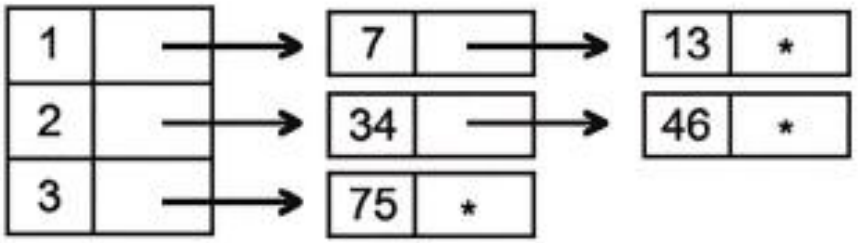
	Tabelas de Espalhamento	Listas Ligadas
Acesso aos dados	Acesso direto	Acesso sequencial
Inserção de dados	Insere com base na Função de Espalhamento por subconjuntos	Pode-se inserir dados no início, meio ou final da Lista
Remoção de dados	Remove com base na Função de Espalhamento por subconjuntos	A remoção pode ser realizada no inicio, meio ou final da Lista
Duplicidade de informações	Utiliza métodos para controle de colisão, podendo usar uma Lista Ligada como auxiliar para armazenamento	Não há controle de duplicidade de informação
Pesquisa de dados	Acesso direto aos dados pesquisado devido à Função de Espalhamento	Pesquisa sequencial, sendo a pesquisa realizada chave a chave

Operações em Tabelas *Hash*

Tabela 3.1 | Função de Espalhamento para cálculo de endereço

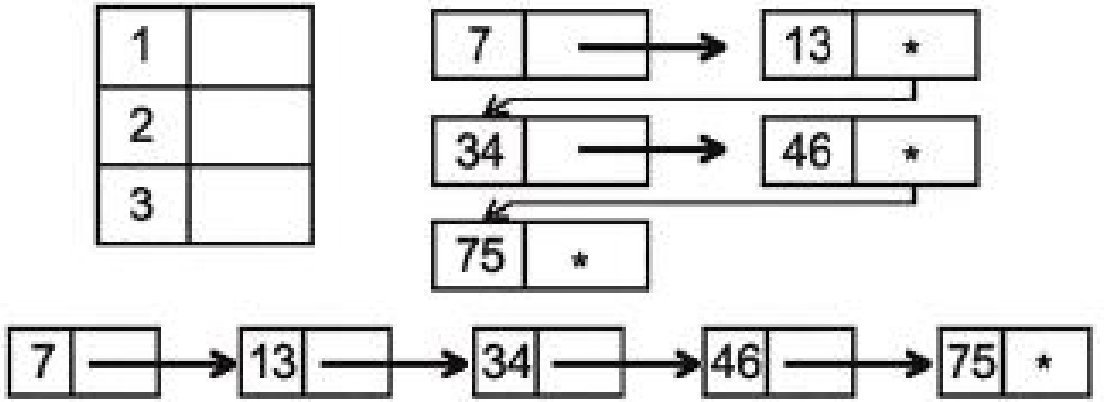
	F (valor) =
1	Se valor < 30
2	Se 30 ≤ valor < 50
3	Se valor ≥ 50

Figura 3.8 | Resultado da aplicação da Função de Espalhamento



Lista ordenada

Figura 3.9 | Lista ligada ordenada com a Função de Espalhamento

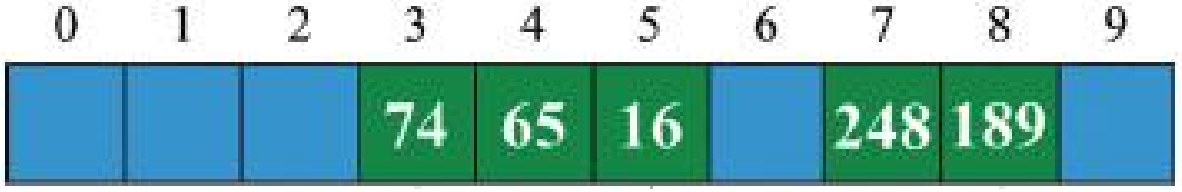


Função *Hash* - Divisão (Resto)

- h(k) = mod(k, n)

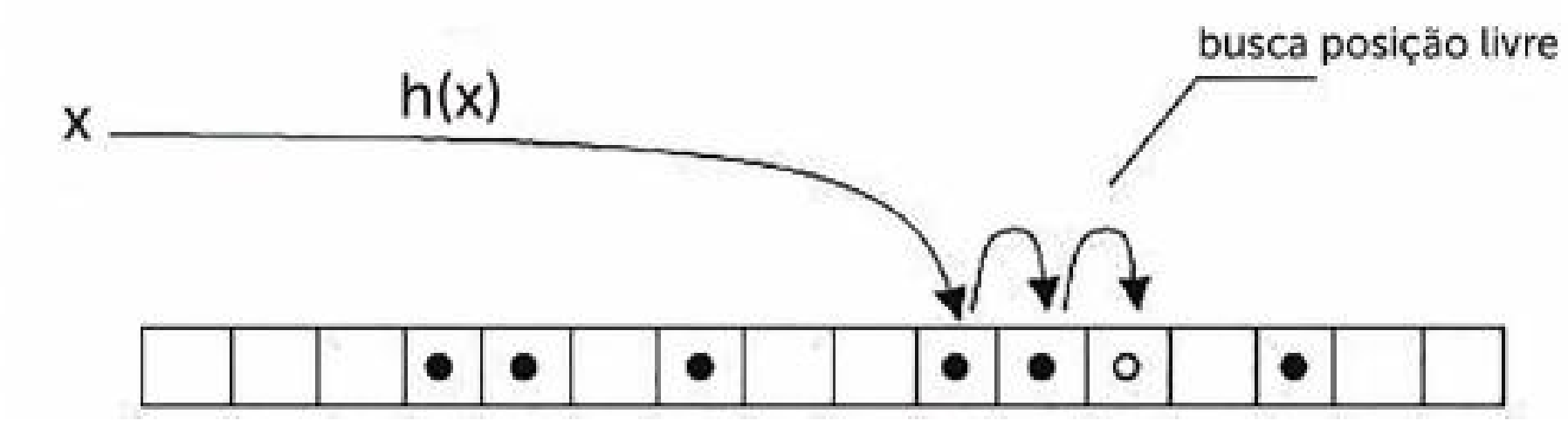
Tabela 3.3| Distribuição na Tabela de Espalhamento

Chave	Cálculo da função	Endereço
16	(16 mod 10) = 6	6
65	(65 mod 10) = 5	5
248	(248 mod 10) = 8	8
189	(189 mod 10) = 9	9
74	(74 mod 10) = 4	4



Colisão

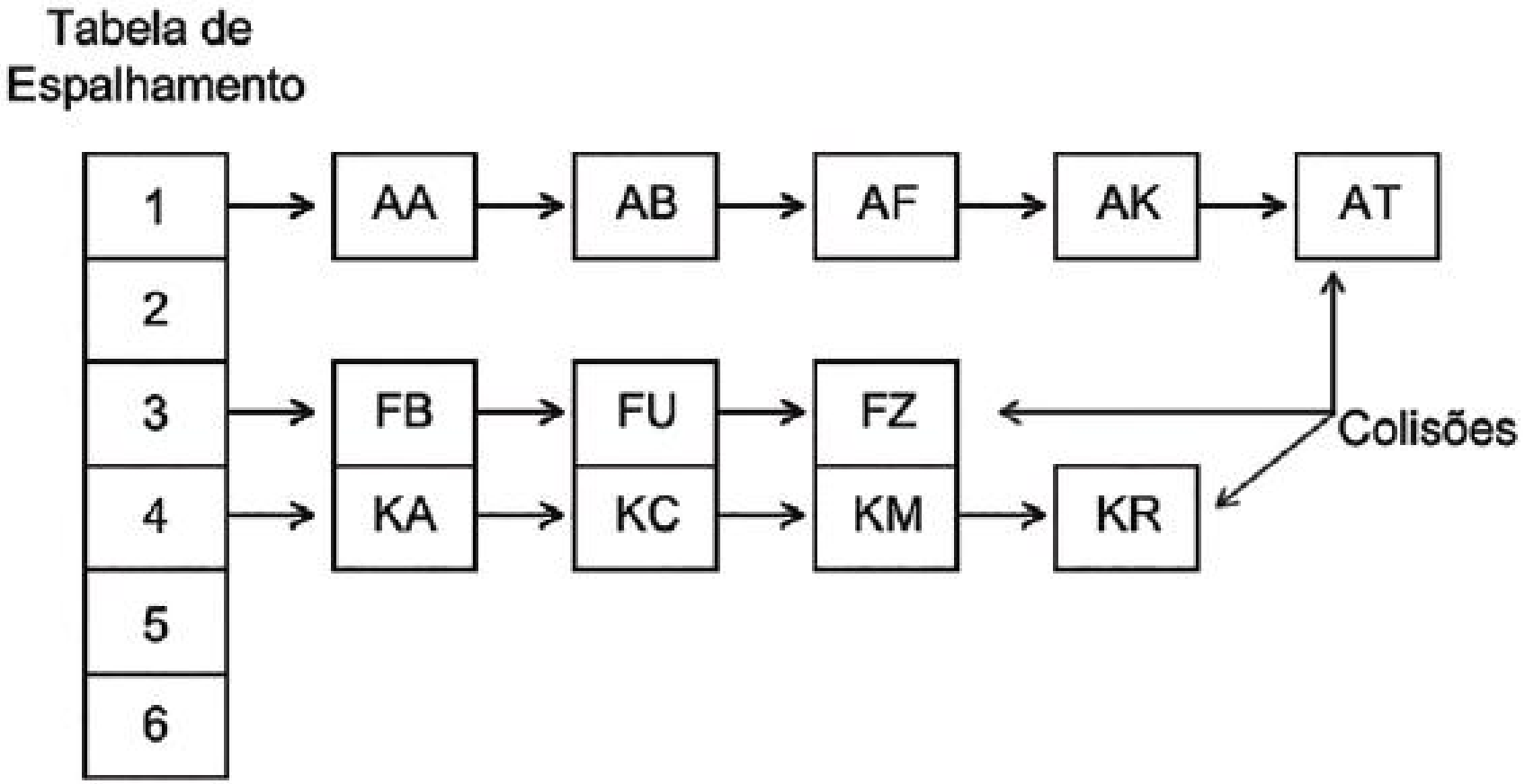
Figura 3.11 | Inserindo elemento na tabela com colisão



Inserção

```
MatAluno * insere_Esp(Hash tab, int RA, char * name, char * mail, char turma) {
    int h = funcao_Esp(RA);

    while (tab[h] != NULL) {
        if (tab[h] -> RA == RA)
            break;
        h = (h + 1) % tam;
    }

    /* Caso não encontre elemento */
    if (tab[h] == NULL) {
        tab[h] = (MatAluno) * malloc(sizeof(MatAluno));
        tab[h] -> RA = RA;
    }

    /* Adiciona ou altera as informações na tabela */
    strcpy(tab[h] -> nome, name);
    strcpy(tab[h] -> email, mail);
    tab[h] -> turma, turma;
    return tab[h];
}
```

Otimização de Tabelas *Hash*

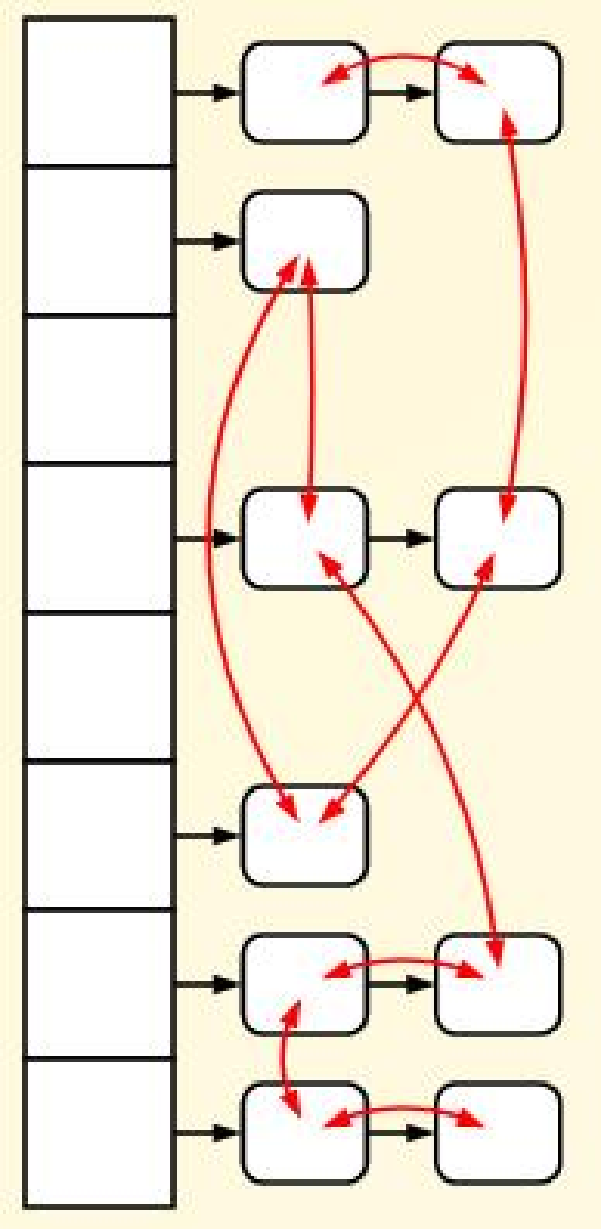


índices 2, 5, e 6 estão vazios

o ideal é que a função *hash* não provoque colisões

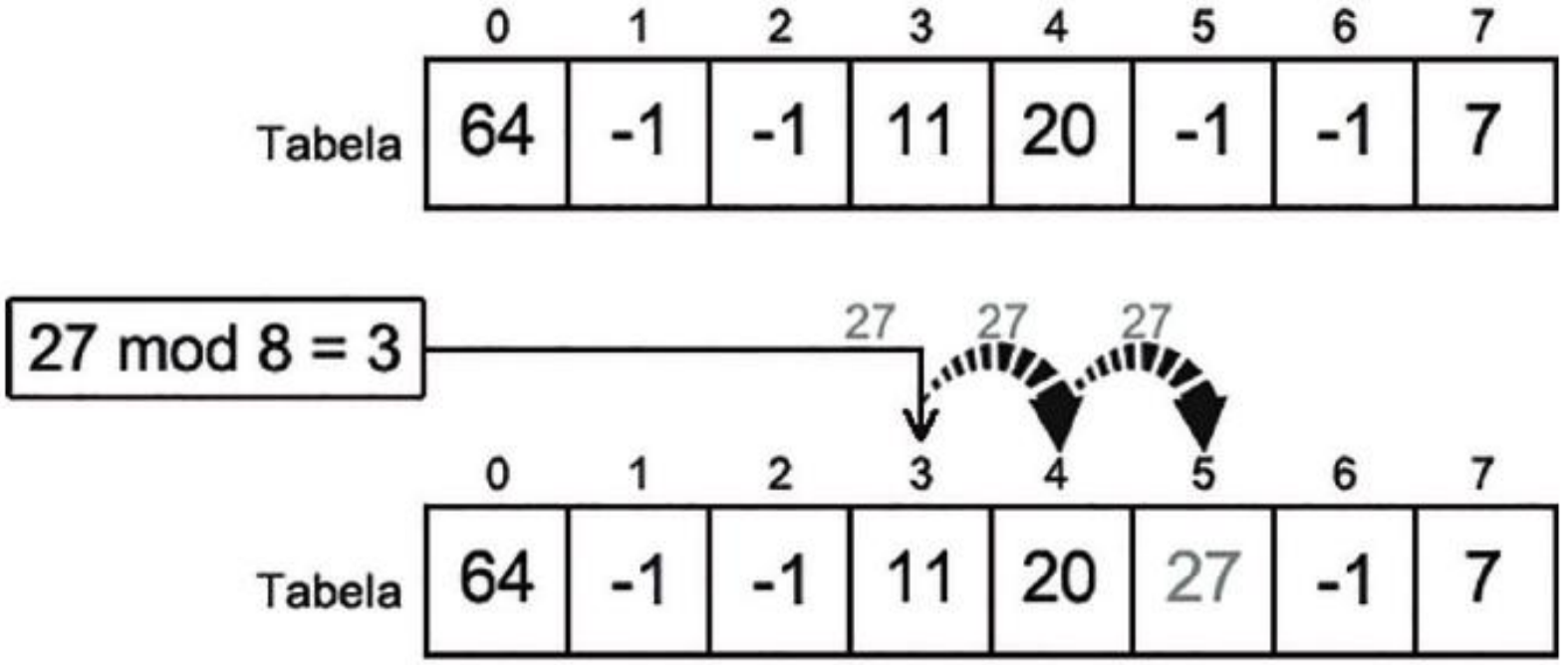
- Endereçamento Fechado
  - Colisão usa lista ligada
- Endereçamento Aberto
  - Colisão coloca na próxima posição sem usar lista ligada
    - Função h*ash* linear: próxima posição vazia
    - Função h*ash* dupla: função *hash* secundária

Encadeamento separado

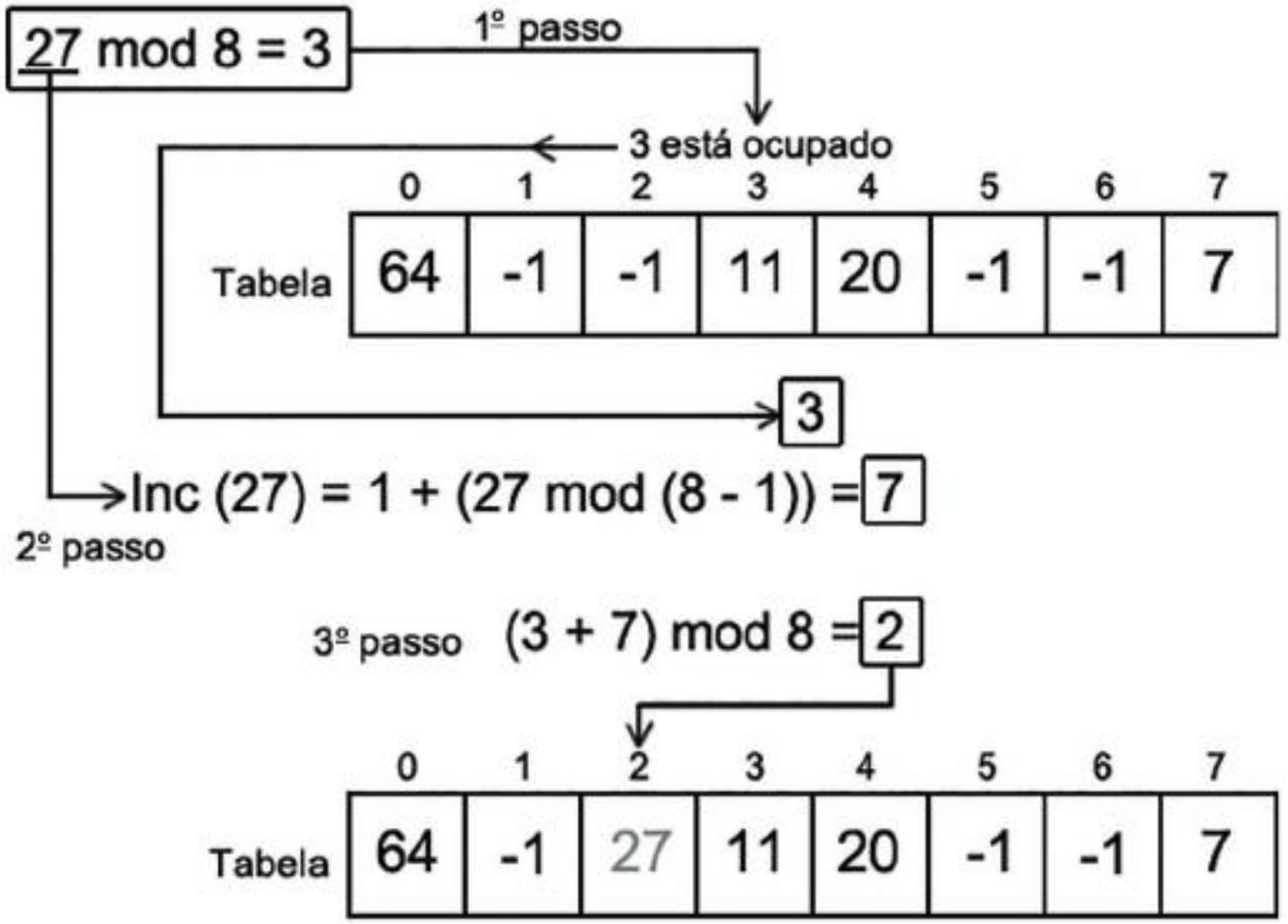


- Combinação de tabela hash com lista ligada

Função *hash* linear



Função de *hash* duplo



Armazenamento associativo

- Estrutura que permite o acesso aos seus elementos com base apenas no seu valor
  - Independentemente de sua posição na estrutura
- Tem como base o tipo de estrutura de tabelas na construção
- Aplicações: tabelas de símbolos (compiladores), tabelas de arquivos (sistemas operacionais)

A	·−	J	·−−−	S	···	2	··−−−
B	−···	K	−·−	T	−	3	···−−
C	−·−·	L	·−··	U	··−	4	····−
D	−··	M	−−	V	···−	5	·····
E	·	N	−·	W	·−−	6	−····
F	··−·	O	−−−	X	−··−	7	−−···
G	−−·	P	·−−·	Y	−·−−	8	−−−··
H	····	Q	−−·−	Z	−−··	9	−−−−·
I	··	R	·−·	1	·−−−−	0	−−−−−

Mapas com lista