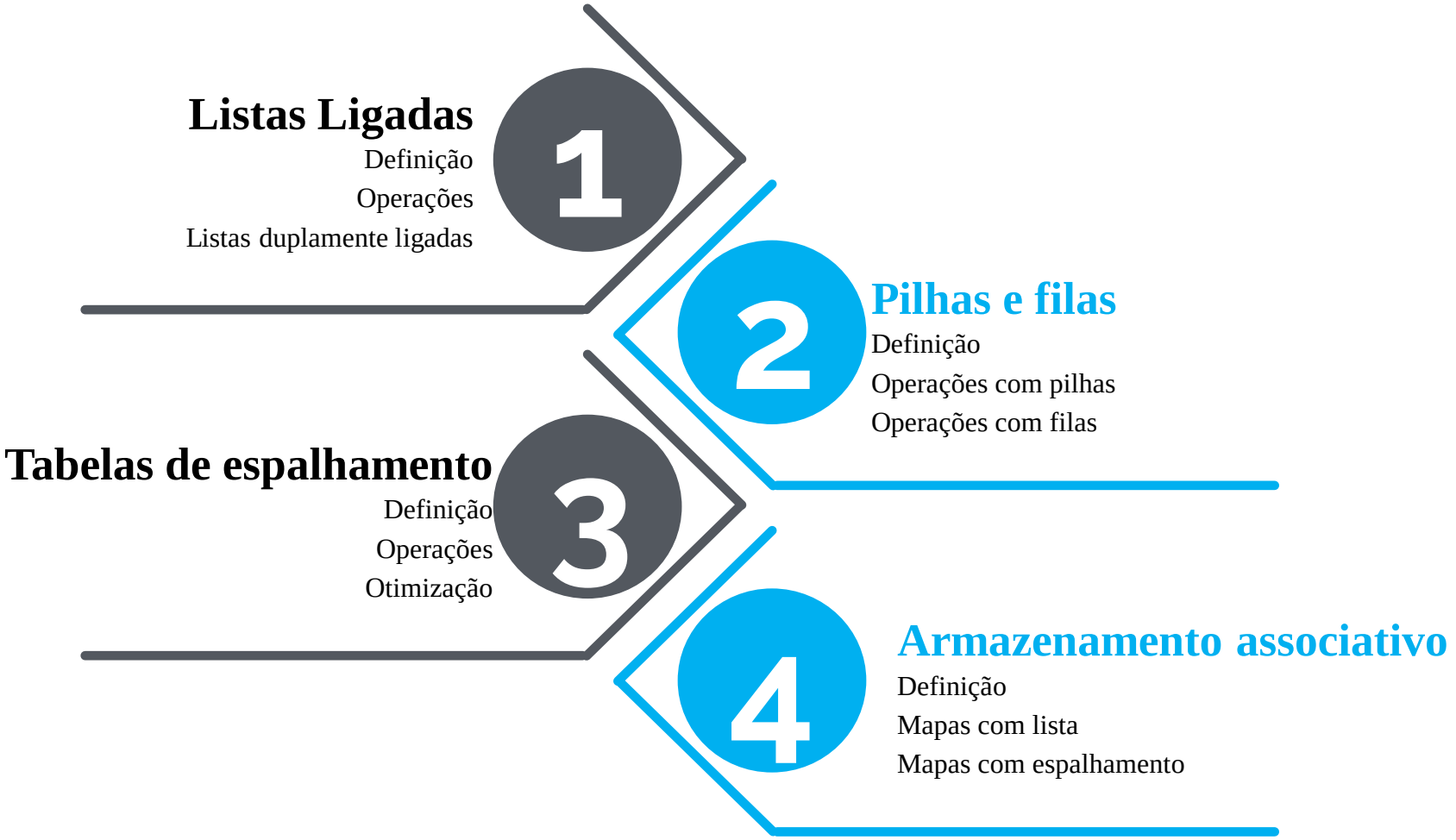
Algoritmos
e
Estrutura de Dados

Profª. Angela Abreu Rosa de Sá, Drª.

Contato: angelaabreu@gmail.com

Ementa


1
Listas Ligadas
Definição
Operações
Listas duplamente ligadas
2
Pilhas e filas
Definição
Operações com pilhas
Operações com filas
3
Tabelas de espalhamento
Definição
Operações
Otimização
4
Armazenamento associativo
Definição
Mapas com lista
Mapas com espalhamento

Algoritmos e Estrutura de Dados

Sumário

Tabelas de Espalhamento

- **Tabelas de espalhamento**
- **Tabelas Hash**
- **Tabelas de dispersão**
- **Tabelas de indexação**

Utilizam **técnicas de endereçamento** para **otimizar** o processo de **consulta de informações**.

Tabelas de Espalhamento

Chave : Otávio

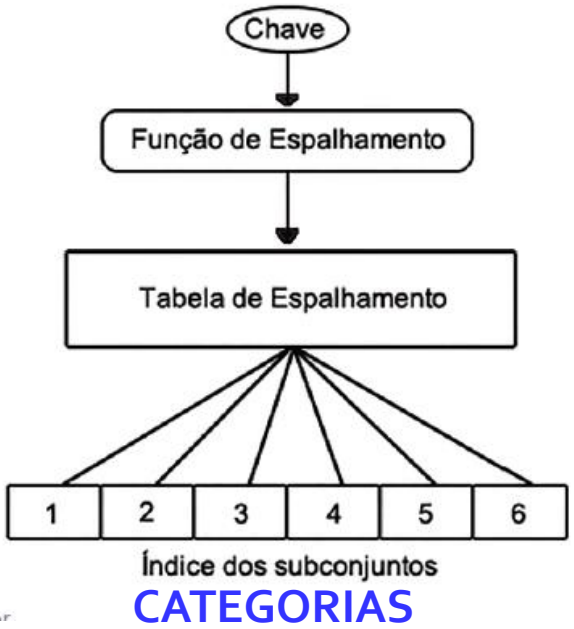
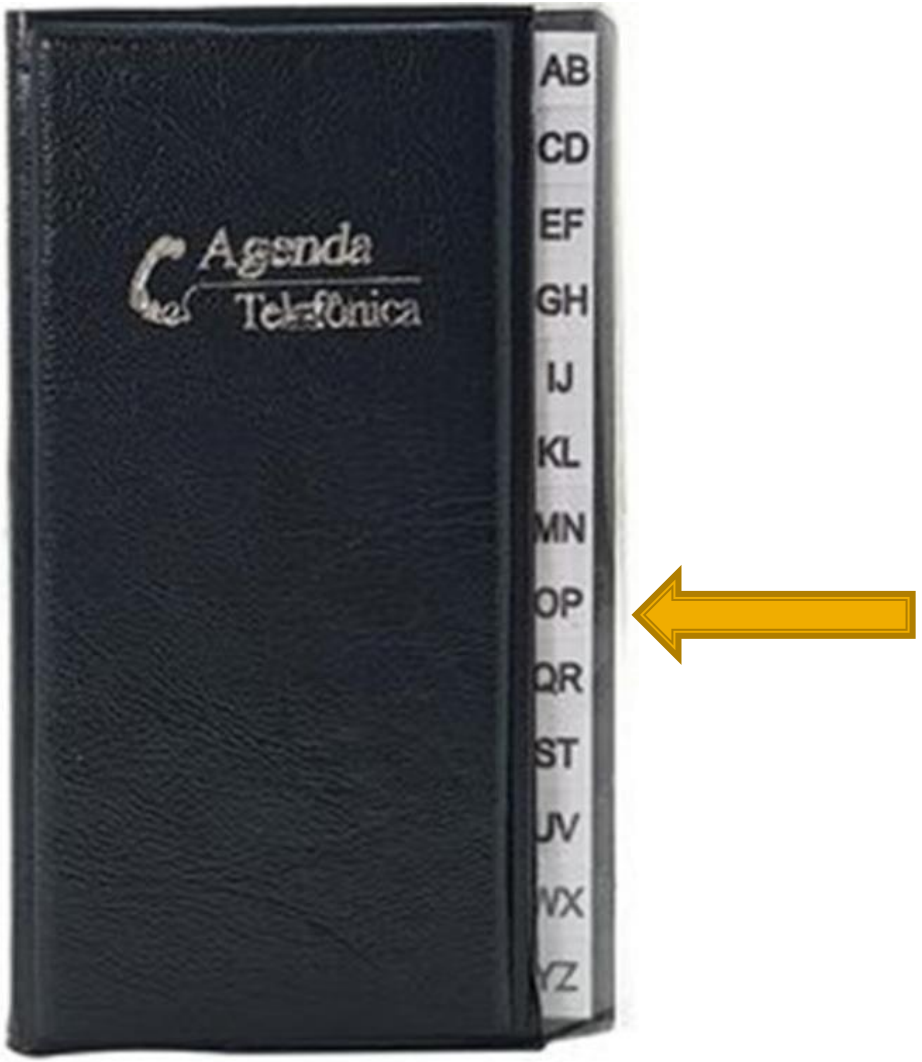
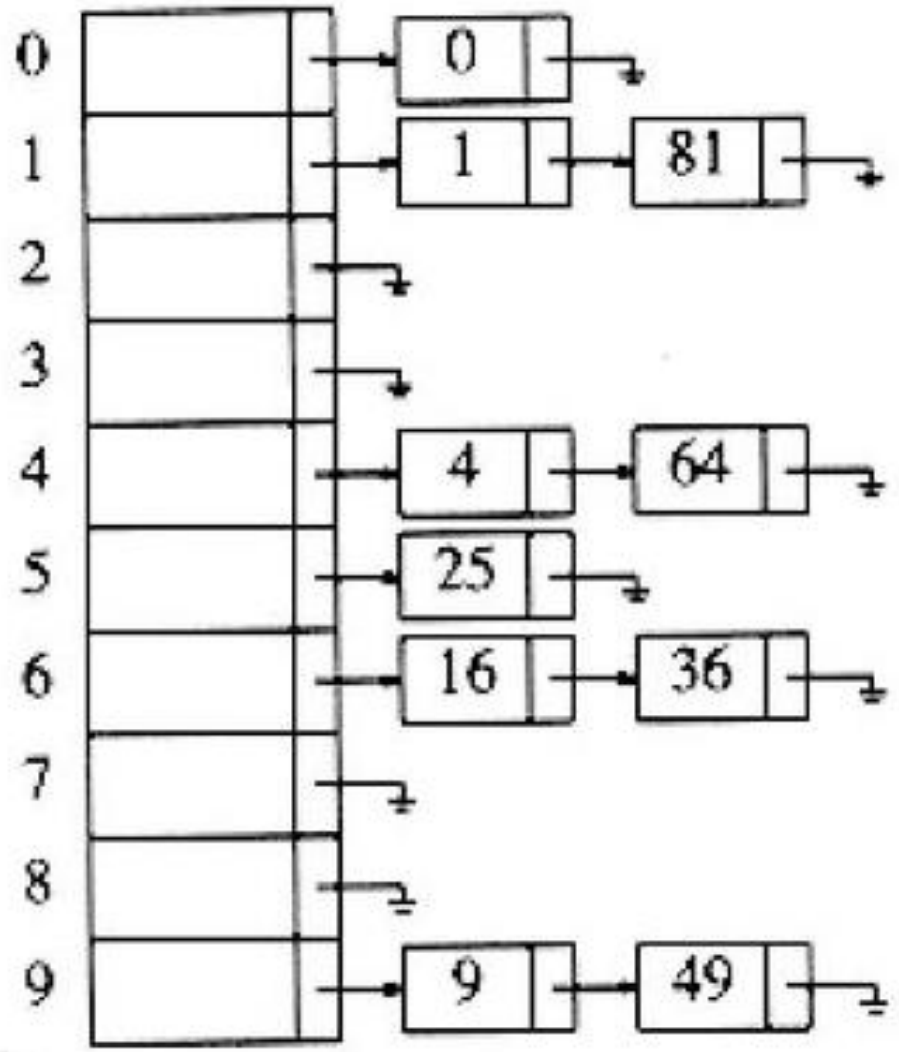


Tabela Hash com 13 entradas

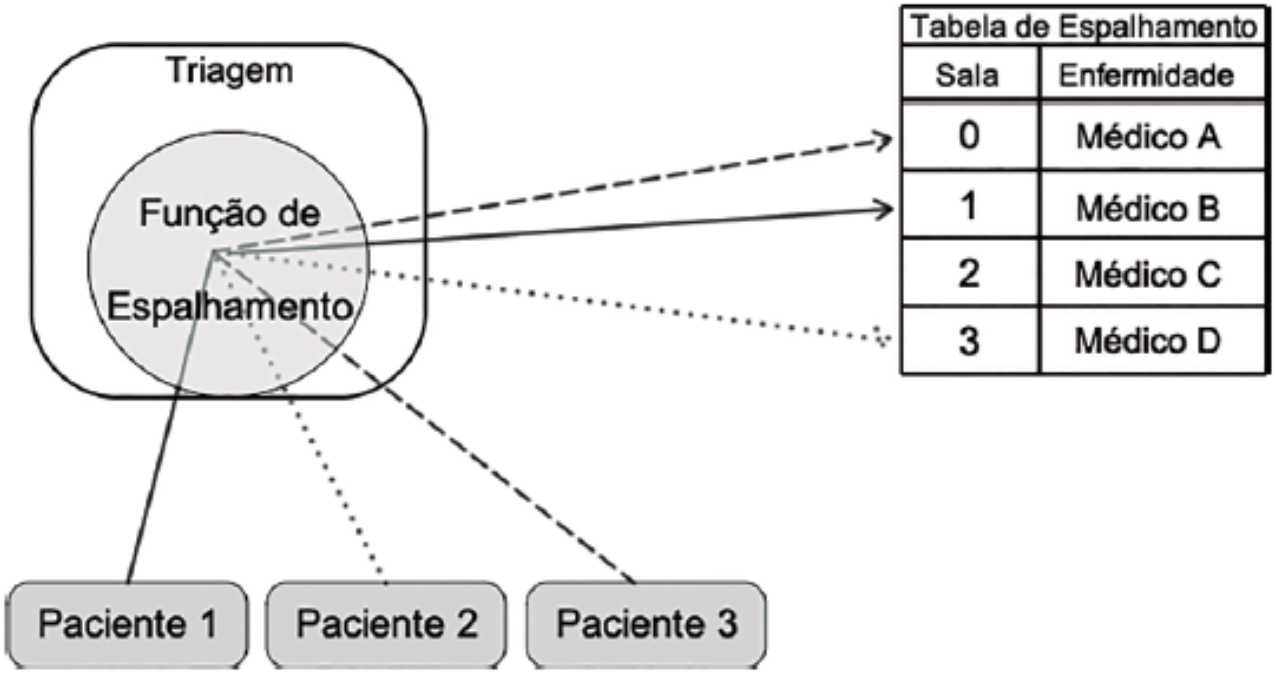


Tabelas de Espalhamento

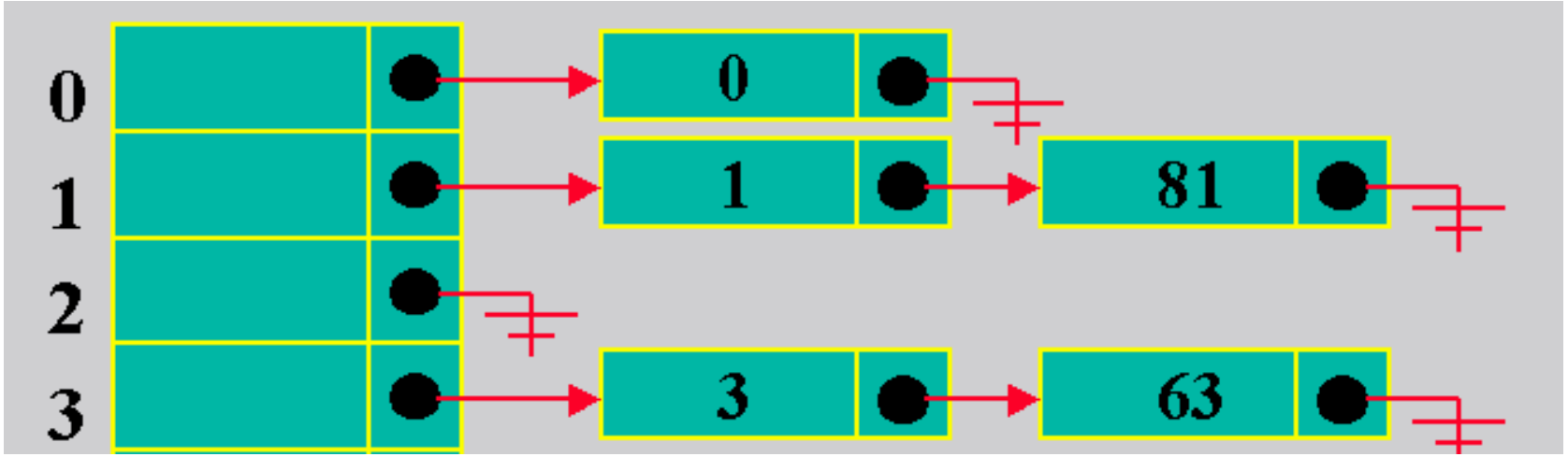


Exemplo: fila de atendimento de pacientes

Figura 3.7 | Esquema de atendimento



Exemplo: fila de atendimento de pacientes



Fila de espera para cada especialidade médica.

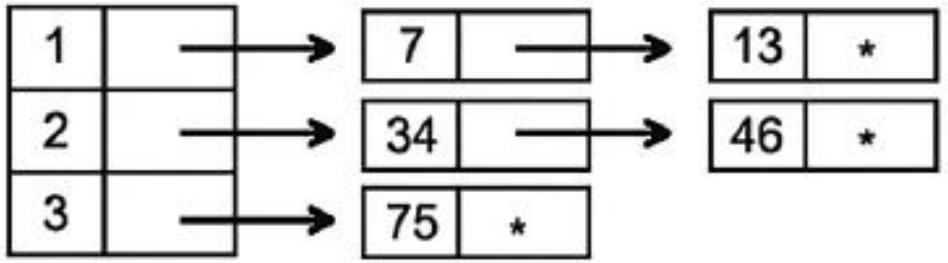
Exemplo: lista de espera por idades

Implementar uma **Tabela Hash** para armazenar a lista de espera para atendimento médico, de acordo com a idade, utilizando a seguinte **Função Hash**:

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

- Inserir as seguintes idades: **7 34 13 46 75**

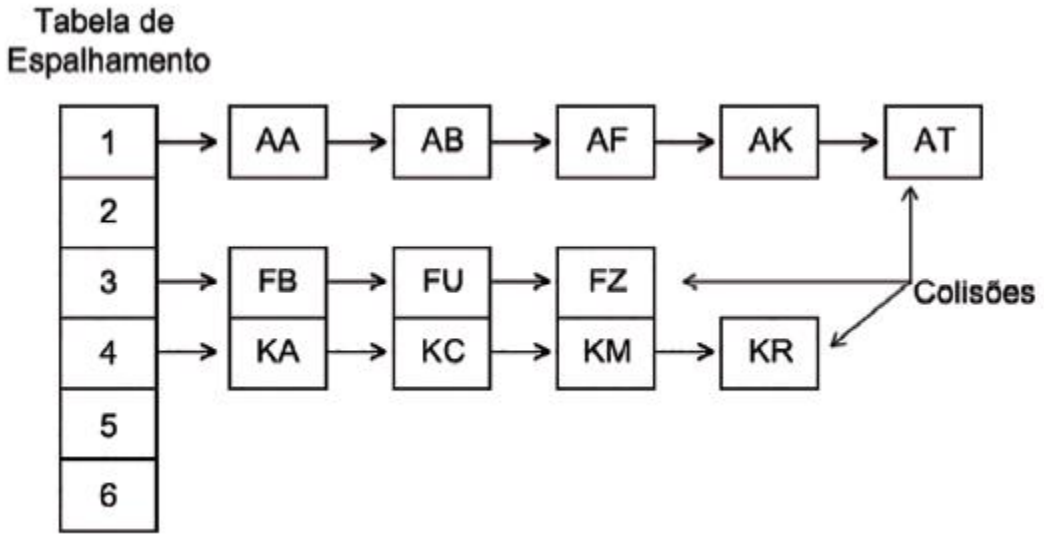
Figura 3.8 | Resultado da aplicação da Função de Espalhamento



Função de Espalhamento Perfeita

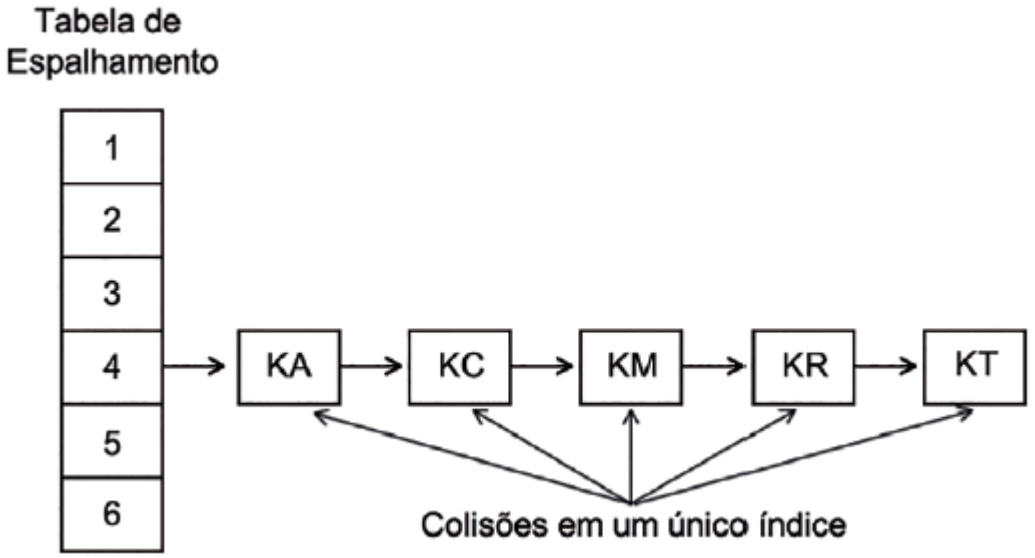
Funcao de Espalhamento Perfeita **apresenta todos os elementos de um conjunto de chaves** em apenas uma entrada na Tabela de Espalhamento

Função de Espalhamento com COLISÃO



As Tabelas de Espalhamento podem sofrer com a deterioração do seu desempenho, causado principalmente pelo **acúmulo de elementos com a mesma chave calculada pela Função de Espalhamento,** o que chamamos de **colisão**

Função de Espalhamento – PIOR CASO

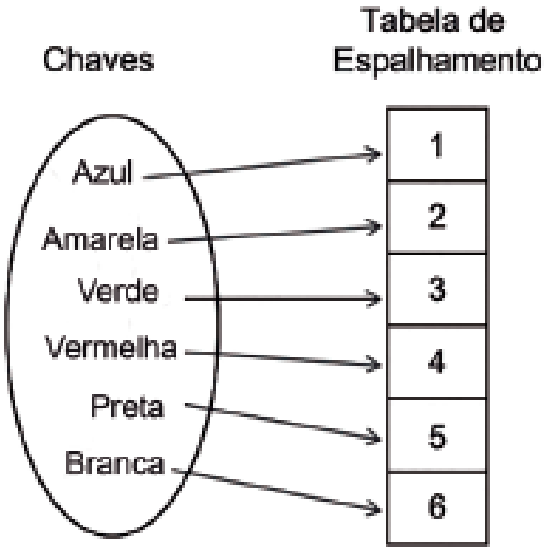


Função de Espalhamento **não adequada** para as chaves tende a **deteriorar o desempenho da tabela**, como utilizar uma função **sem tratamento de colisões** fará a função ter um desempenho prejudicado.

Função de Espalhamento PERFEITA

Para garantir um **bom desempenho** da função de espalhamento, como requisito básico esta deve prover uma **distribuição uniforme nos índices da tabela**.

A **não uniformidade** na distribuição tende a **aumentar o número de colisões**, assim como o **esforço para buscar ou armazenar os elementos na tabela**.

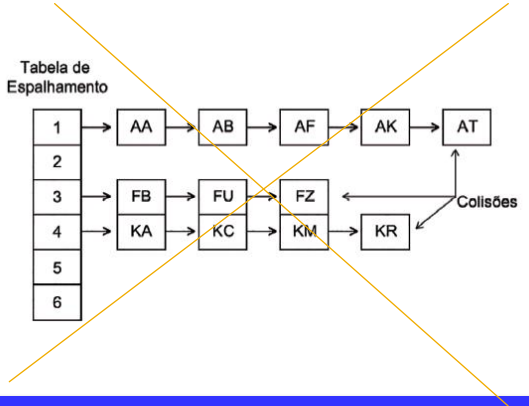


Função de Espalhamento PERFEITA

OTIMIZAÇÃO

Assimile

A principal vantagem de ter uma Tabela de Espalhamento perfeita é o fato de ser possível realizar pesquisas com tempo constante. Assim, sendo de conhecimento todas as chaves de início, uma Tabela de Espalhamento perfeita pode ser criada por uma Função de Espalhamento Perfeita, sem ocasionar nenhuma colisão.



Otimização em Tabelas de Espalhamento

- **Função de Espalhamento Linear:**

Procura a **próxima posição vazia** após o endereço-base da chave.

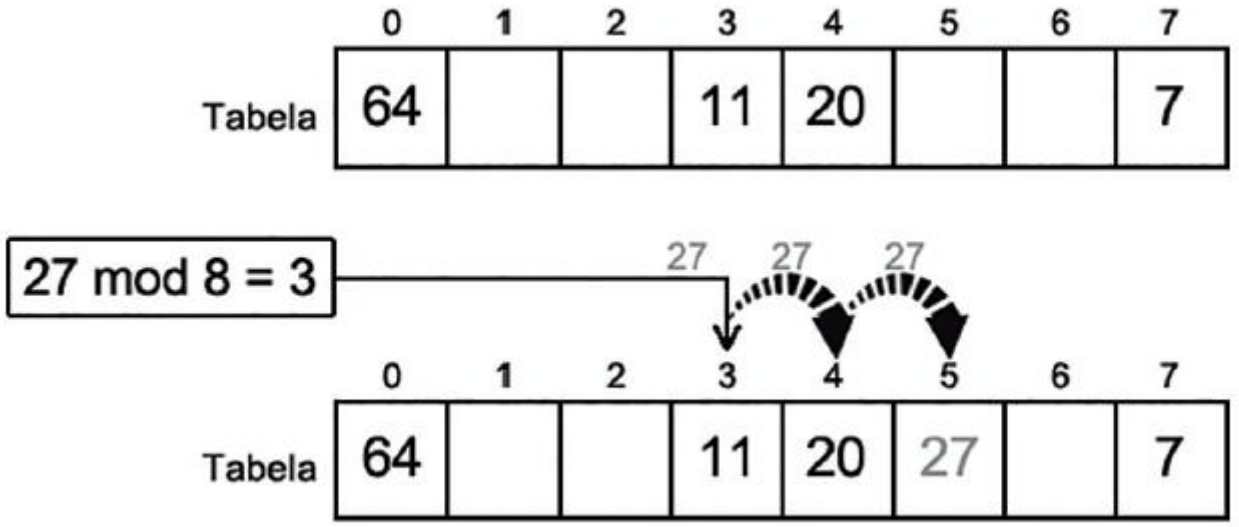
- **Função de Espalhamento Duplo:**

Uma Função de Espalhamento secundária **calcula o incremento,** substituindo o incremento da posição em 1 (um).

Função de Espalhamento Linear

Consiste em **procurar a próxima posição vazia** depois do índice-base da chave

h(k) = k mod N onde k é a chave e N é o número de posições da tabela



Otimização em Tabelas de Espalhamento

- **Função de Espalhamento Linear:**

Procura a **próxima posição vazia** após o endereço-base da chave.

- **Função de Espalhamento Duplo:**

Uma Função de Espalhamento secundária **calcula o incremento,** substituindo o incremento da posição em 1 (um).

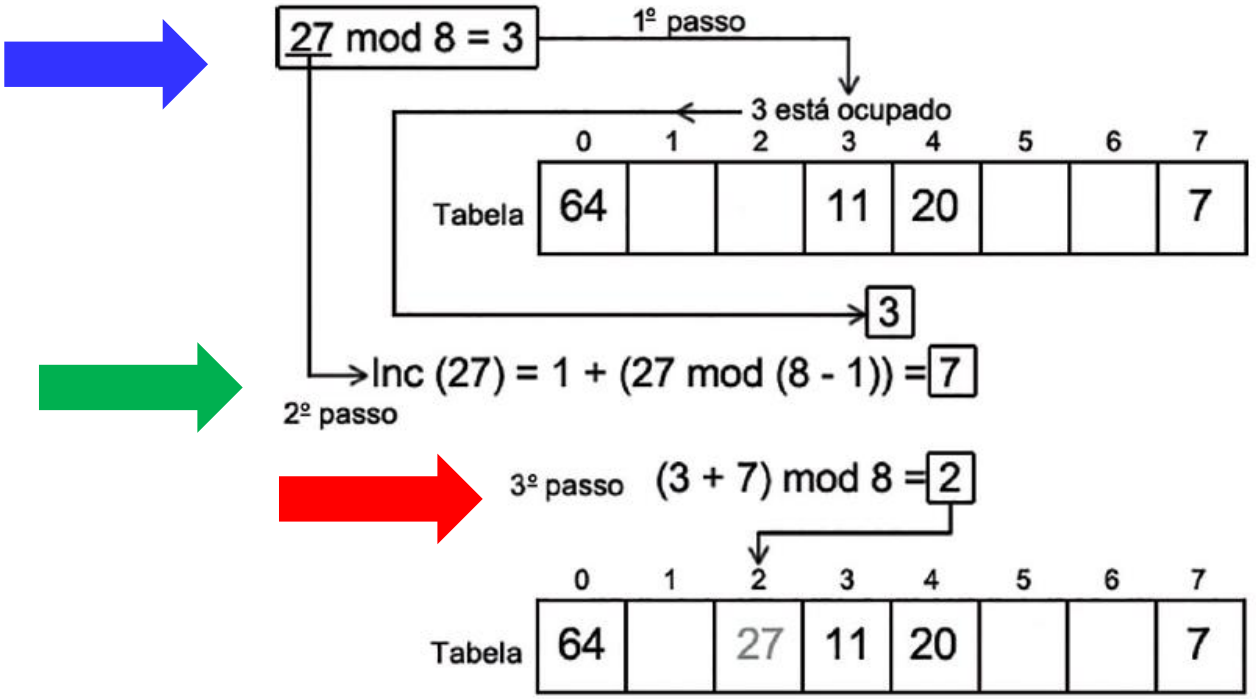
Função de Espalhamento Duplo

Conhecida também como **re-hash**, a **Função de Espalhamento Duplo**, em vez de incrementar a posição do elemento até a próxima posição vazia, **utiliza uma Função de Espalhamento auxiliar** para calcular qual o incremento que será dado à posição, levando em consideração o valor da chave.

A função dupla tende a **espalhar melhor ao longo dos índices da tabela**, o que é uma vantagem sobre a função linear. No entanto, como desvantagem, os **índices podem estar muito distantes um do outro**, provocando buscas adicionais na tabela.

Função de Espalhamento Duplo

- **1º passo:** h(k) = k mod N
- **2º passo:** f(k) = 1 + (k mod (N-1))
- **3º passo:** re-hask(k) = (h(k) + f(k)) mod N



Função de Espalhamento Linear

```c
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;

int FuncaoHash(int chave)
{
    int indice = chave % 10; //Tabela Hash com 10 posições
   return indice;
}
```

```c
int InserirNaTabelaHash(int TabelaHash[], int chave, int TamanhoTabelaHash)
{
    int indice;
    indice = FuncaoHash(chave);

    // Busca a próxima posição livre
    while (TabelaHash[indice] != -1)
    {
        if(indice == TamanhoTabelaHash)
        {  indice = -2; // Tabela cheia
           break;
       }

        if (TabelaHash[indice] == chave)
        { indice = -1; // elemento já existente
          break;
        }

        indice++;
    }

    // Insere na posicao livre encontrada (exceto indice = -1 ou -2)
    if(indice >= 0)
       TabelaHash[indice] = chave;

    return indice;
}
```

```c
int ProcuraNaTabelaHash(int TabelaHash[], int chave, int TamanhoTabelaHash)
{
    int indice;
    int contador;

    indice = FuncaoHash(chave);

    // Procura a chave a partir da posição i
    while (TabelaHash[indice] != chave)
    {
        if (TabelaHash[indice] == -1)
        { indice = -1;//nao achou a chave pois existe uma posiçao vazia
          break;
        }

        if (indice == TamanhoTabelaHash)
         { indice = -2; //retorna pois chegou ao fim da tabela e não encontrou a chave
           break;
         }

        indice++;
    }
    /* Retorna a posição que encontrou */
    return indice;

}
```

```c
void ImprimirTabelaHash(int TabelaHash[], int TamanhoTabelaHash)
{
    for(int i = 0; i < TamanhoTabelaHash; i++)
    {
        printf("[%d] %d ",i, TabelaHash[i]);
    }

    printf("FIM");

}
```

```c
int main(){

    int TamanhoTabela = 10;
    // alocação estática:  int TabelaHash[TamanhoTabela];
    //alocação dinâmica:
    int *TabelaHash  = (int*)malloc(sizeof(int) * TamanhoTabela);


    //iniciar a Tabela Hash com -1 (indicar posições vazias)
    for(int i = 0; i < 10; i++)
    {
        TabelaHash[i] = -1;
    }
```

```
int chave = 0;
int resultado;
while(chave != -1)
{
    printf("\n **** Digite o numero a ser inserido:  ");
    cin>>chave;

    resultado = InserirNaTabelaHash(TabelaHash,chave,TamanhoTabela);
    if(resultado >= 0)
    {
        printf("\n -- Chave inserida com sucesso! \n");
        ImprimirTabelaHash(TabelaHash,TamanhoTabela);
    }
    else if(resultado == -2)
            printf("\n -- Não existe espaço livre na tabela hash");
         else printf("\n -- Chave já existente!");

}
```

```
    printf("\n **** BUSCA NA TABELA HASH **** : \n ");
    chave = 0;
    while(chave != -1)
    {

        printf("\n -- Digite a chave para ser localizada:  ");
        cin>>chave;

        resultado = ProcuraNaTabelaHash(TabelaHash,chave,TamanhoTabela);
        if(resultado >= 0)
        {
            printf("\n A chave procurada está no indice %d \n", resultado);
            ImprimirTabelaHash(TabelaHash,TamanhoTabela);
        }
        else  printf("\n Chave não localizada!");


    }

    return 1;
}
```

```
 -- Chave inserida com sucesso!
[0] -1 [1] 1 [2] 2 [3] 3 [4] 4 [5] -1 [6] -1 [7] -1 [8] -1 [9] -1 FIM
 **** Digite o numero a ser inserido:  5

 -- Chave inserida com sucesso!
[0] -1 [1] 1 [2] 2 [3] 3 [4] 4 [5] 5 [6] -1 [7] -1 [8] -1 [9] -1 FIM
 **** Digite o numero a ser inserido:  6

 -- Chave inserida com sucesso!
[0] -1 [1] 1 [2] 2 [3] 3 [4] 4 [5] 5 [6] 6 [7] -1 [8] -1 [9] -1 FIM
 **** Digite o numero a ser inserido:  7

 -- Chave inserida com sucesso!
[0] -1 [1] 1 [2] 2 [3] 3 [4] 4 [5] 5 [6] 6 [7] 7 [8] -1 [9] -1 FIM
 **** Digite o numero a ser inserido:  8

 -- Chave inserida com sucesso!
[0] -1 [1] 1 [2] 2 [3] 3 [4] 4 [5] 5 [6] 6 [7] 7 [8] 8 [9] -1 FIM
 **** Digite o numero a ser inserido:  9

 -- Chave inserida com sucesso!
[0] -1 [1] 1 [2] 2 [3] 3 [4] 4 [5] 5 [6] 6 [7] 7 [8] 8 [9] 9 FIM
 **** Digite o numero a ser inserido:  10

 -- Chave inserida com sucesso!
[0] 10 [1] 1 [2] 2 [3] 3 [4] 4 [5] 5 [6] 6 [7] 7 [8] 8 [9] 9 FIM
 **** Digite o numero a ser inserido:
```

Função de Espalhamento Duplo

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;

int FuncaoHash(int chave,int TamanhoTabelaHash)
{
    int indice = chave % TamanhoTabelaHash;
   return indice;
}

int FuncaoHashSegundoPasso(int chave,int TamanhoTabelaHash)
{
    int indice = (1+(chave % (TamanhoTabelaHash-1)));
   return indice;
}

int FuncaoReHash(int i, int k, int TamanhoTabelaHash)
{
    int indice = (i+k) % TamanhoTabelaHash;
    return indice;
}
```

```c
int InserirNaTabelaHash(int TabelaHash[], int chave, int TamanhoTabelaHash)
{
    int indice,i,k;
    i = FuncaoHash(chave,TamanhoTabelaHash);
    k = FuncaoHashSegundoPasso(chave,TamanhoTabelaHash);
    indice =  FuncaoReHash(i,k, TamanhoTabelaHash);

    // Busca a próxima posição livre
    while (TabelaHash[indice] != -1)
    {
        if(indice == TamanhoTabelaHash)
        {  indice = -2; // Tabela cheia
          break;
       }

        if (TabelaHash[indice] == chave)
        { indice = -1; // elemento já existente
          break;
        }
        indice++;
    }

    // Insere na posicao livre encontrada (exceto indice = -1 ou -2)
    if(indice >= 0)
       TabelaHash[indice] = chave;

    return indice;
}
```

```c
int ProcuraNaTabelaHash(int TabelaHash[], int chave, int TamanhoTabelaHash)
{
    int indice;
    int i;
    int k;
    i = FuncaoHash(chave,TamanhoTabelaHash);
    k = FuncaoHashSegundoPasso(chave,TamanhoTabelaHash);
    indice =  FuncaoReHash(i,k, TamanhoTabelaHash);

    // Procura a chave a partir da posição i
    while (TabelaHash[indice] != chave)
    {
        if (TabelaHash[indice] == -1)
        { indice = -1;//nao achou a chave pois existe uma posiçao vazia
          break;
        }

        if (indice == TamanhoTabelaHash)
         { indice = -2; //retorna pois chegou ao fim da tabela e não encontrou a chave
           break;
         }

        indice++;
    }
    /* Retorna a posição que encontrou */
    return indice;
}
```

```c
void ImprimirTabelaHash(int TabelaHash[], int TamanhoTabelaHash)
{
    for(int i = 0; i < TamanhoTabelaHash; i++)
    {
        printf("[%d] %d ",i, TabelaHash[i]);
    }

    printf("FIM");

}
```

```c
int main(){

    int TamanhoTabela = 10;
    // alocação estática:  int TabelaHash[TamanhoTabela];
    //alocação dinâmica:
    int *TabelaHash  = (int*)malloc(sizeof(int) * TamanhoTabela);


    //iniciar a Tabela Hash com -1 (indicar posições vazias)
    for(int i = 0; i < 10; i++)
    {
        TabelaHash[i] = -1;
    }
```

```
int chave = 0;
int resultado;
while(chave != -1)
{
    printf("\n **** Digite o numero a ser inserido:  ");
    cin>>chave;

    resultado = InserirNaTabelaHash(TabelaHash,chave,TamanhoTabela);
    if(resultado >= 0)
    {
        printf("\n -- Chave inserida com sucesso! \n");
        ImprimirTabelaHash(TabelaHash,TamanhoTabela);
    }
    else if(resultado == -2)
            printf("\n -- Não existe espaço livre na tabela hash");
        else printf("\n -- Chave já existente!");

}
```

```
    printf("\n **** BUSCA NA TABELA HASH **** : \n ");
    chave = 0;
    while(chave != -1)
    {

        printf("\n -- Digite a chave para ser localizada:  ");
        cin>>chave;

        resultado = ProcuraNaTabelaHash(TabelaHash,chave,TamanhoTabela);
        if(resultado >= 0)
        {
            printf("\n A chave procurada está no indice %d \n", resultado);
            ImprimirTabelaHash(TabelaHash,TamanhoTabela);
        }
        else  printf("\n Chave não localizada!");


    }

    return 1;
}
```

```
-- Chave inserida com sucesso!
0] -1 [1] -1 [2] -1 [3] 1 [4] -1 [5] 2 [6] -1 [7] 3 [8] -1 [9] 4 FIM
**** Digite o numero a ser inserido:  5

-- Chave inserida com sucesso!
0] -1 [1] 5 [2] -1 [3] 1 [4] -1 [5] 2 [6] -1 [7] 3 [8] -1 [9] 4 FIM
**** Digite o numero a ser inserido:  6

-- Chave inserida com sucesso!
0] -1 [1] 5 [2] -1 [3] 1 [4] 6 [5] 2 [6] -1 [7] 3 [8] -1 [9] 4 FIM
**** Digite o numero a ser inserido:  7

-- Chave inserida com sucesso!
0] -1 [1] 5 [2] -1 [3] 1 [4] 6 [5] 2 [6] 7 [7] 3 [8] -1 [9] 4 FIM
**** Digite o numero a ser inserido:  8

-- Chave inserida com sucesso!
0] -1 [1] 5 [2] -1 [3] 1 [4] 6 [5] 2 [6] 7 [7] 3 [8] 8 [9] 4 FIM
**** Digite o numero a ser inserido:  9

-- Chave inserida com sucesso!
0] 9 [1] 5 [2] -1 [3] 1 [4] 6 [5] 2 [6] 7 [7] 3 [8] 8 [9] 4 FIM
**** Digite o numero a ser inserido:  10

-- Chave inserida com sucesso!
0] 9 [1] 5 [2] 10 [3] 1 [4] 6 [5] 2 [6] 7 [7] 3 [8] 8 [9] 4 FIM
**** Digite o numero a ser inserido:
```

Exercícios 😀

Muito Obrigada!

Profª. Angela Abreu Rosa de Sá, Drª.

Contato: angelaabreu@gmail.com