Tabela de Dispersão - Hash

Espalhamento de dados e busca

Maria Adriana Vidigal de Lima

FACOM - UFU

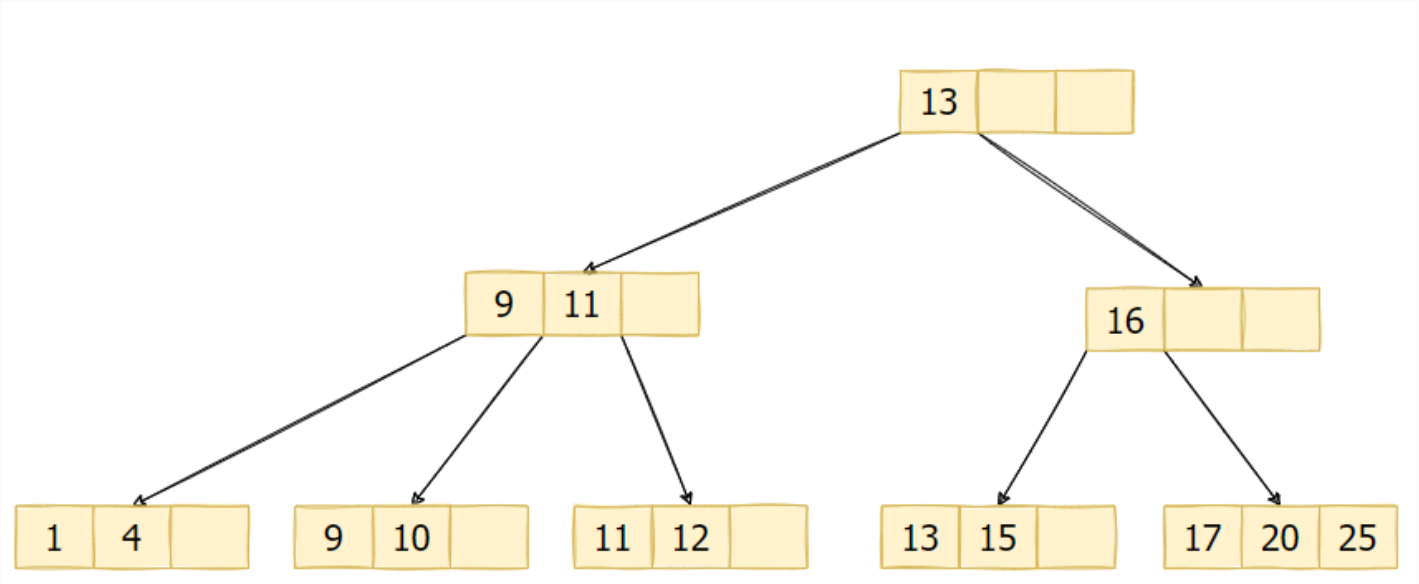
Sumário

- Motivação
- Definição de Hash e Conceitos
- Funções de Dispersão: Divisão, Dobra e Multiplicação
- Colisão e Tratamento: Encadeamento Separado e Endereçamento Aberto
- Implementação

Motivação

Alternativas para tornar buscas em grandes volumes de dados mais eficientes:

- Usar armazenamento em árvore de busca auto-balanceada (ex. Árvore B). Uma busca tem desempenho $O(log_M N)$ sendo N o número de elementos e M a ordem (aridade da árvore):



- Usar cálculo de endereço para acessar diretamente o registro procurado. Desta forma tem-se a busca em $O(1)$: **Tabelas Hash**.

Motivação

Uma tabela de dispersão (ou *hash*) segue a ideia de um armário com **escaninhos** para correspondências num condomínio residencial:

- Existe um escaninho para cada unidade residencial.
- Todos os moradores de uma mesma unidade residencial procuram sua correspondência dentro do mesmo escaninho.
- A busca é simples e direta pelo número da unidade residencial: muitos escaninhos ficam vazios e alguns podem ter várias correspondências.
- O armário tem que possuir no mínimo M escaninhos, sendo M o número de unidades residenciais.

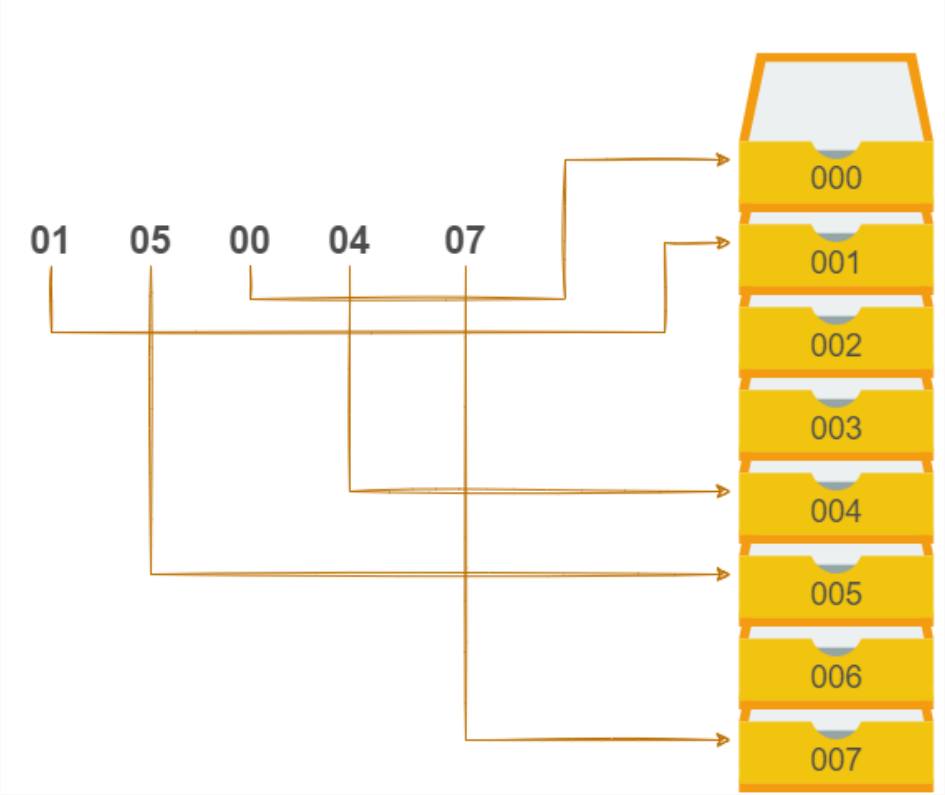
Funcionamento da Tabela de Dispersão: Simplificação

Como distribuir M elementos (chaves) em uma estrutura de forma que o acesso possa ser realizado de forma direta?

- A estrutura tem [0, M-1] compartimentos.
- Cada compartimento armazena um elemento.
- O próprio elemento (chave), ou uma parte, é utilizada para calcular o local do seu armazenamento.

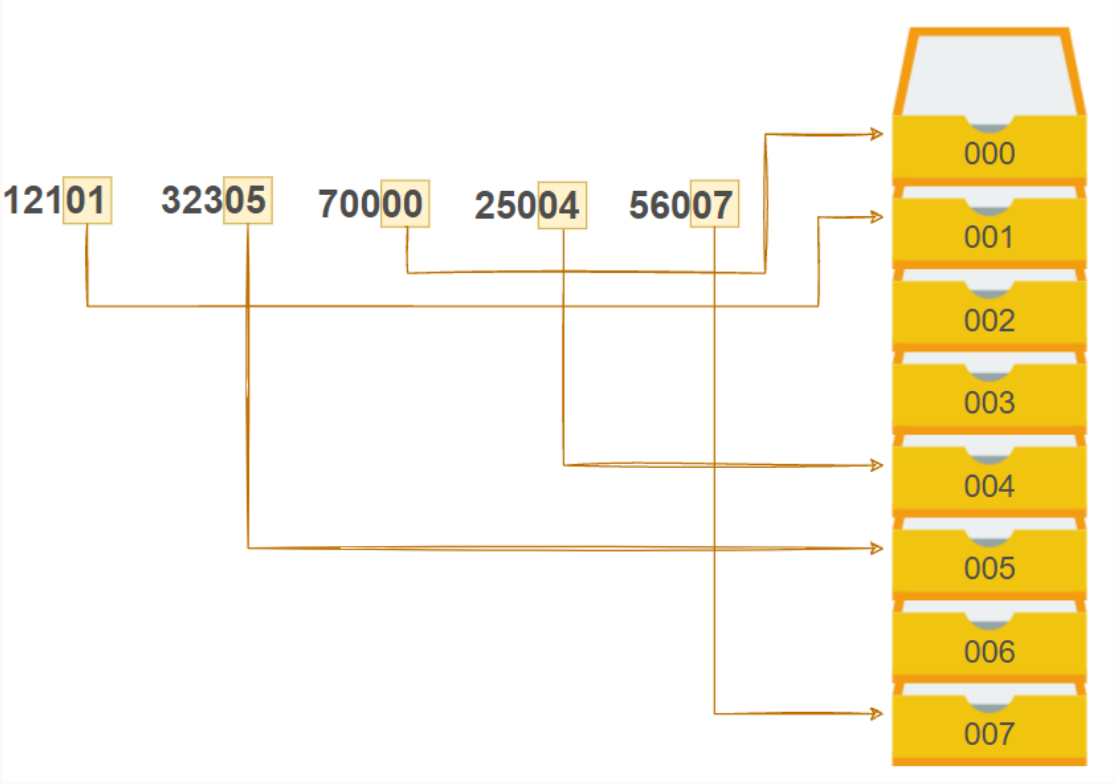
Simplificação da Tabela de Dispersão: Exemplo 1

- O valor da chave é utilizado como seu índice na estrutura.
- Cada chave x é adicionada no compartimento x.
- Acesso direto.



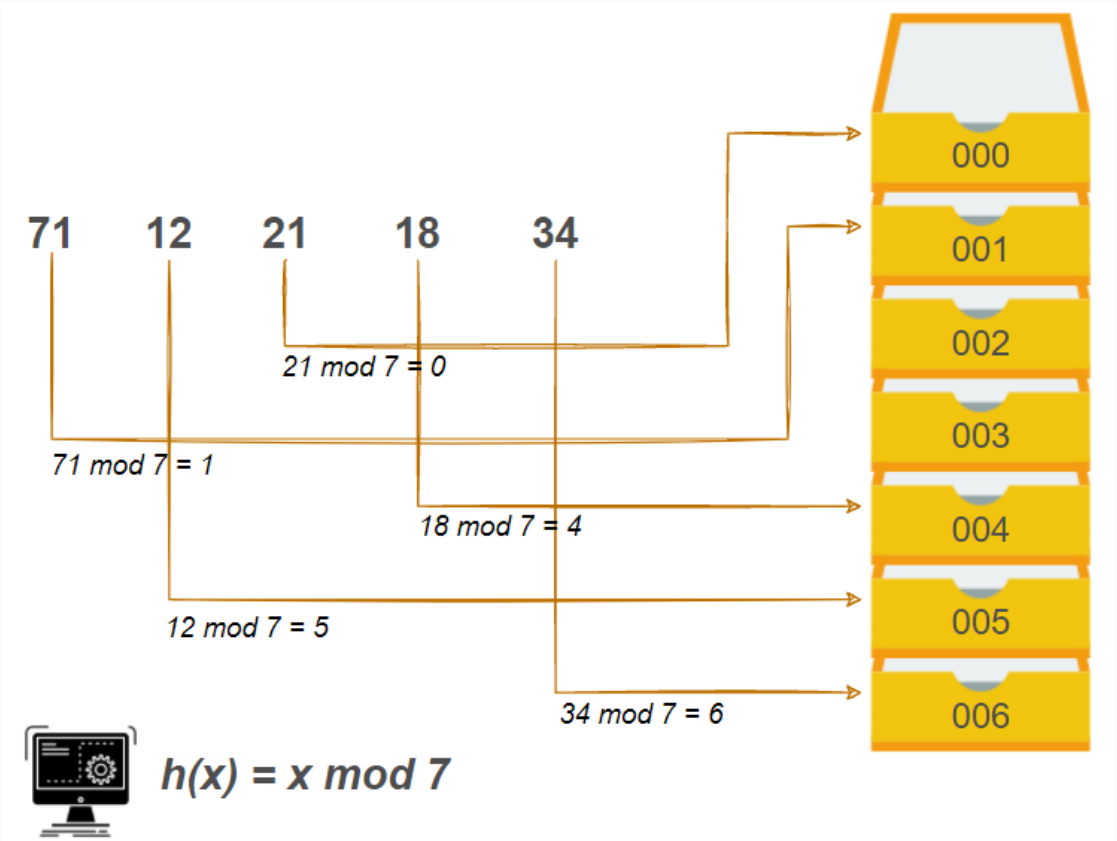
Simplificação da Tabela de Dispersão: Exemplo 2

- Uma parte do valor da chave é utilizada como seu índice na estrutura.
- Cada chave x é adicionada no compartimento x.
- Acesso direto: parte da chave é utilizada na sua recuperação.



Simplificação da Tabela de Dispersão: Exemplo 3

- Uma função *h* será utilizada para mapear um valor de chave *x* para um endereço da estrutura.
- A chave é armazenada no compartimento apontado por $h(x)$.
- Acesso direto: $h(x)$ produz um endereço-base para *x*.

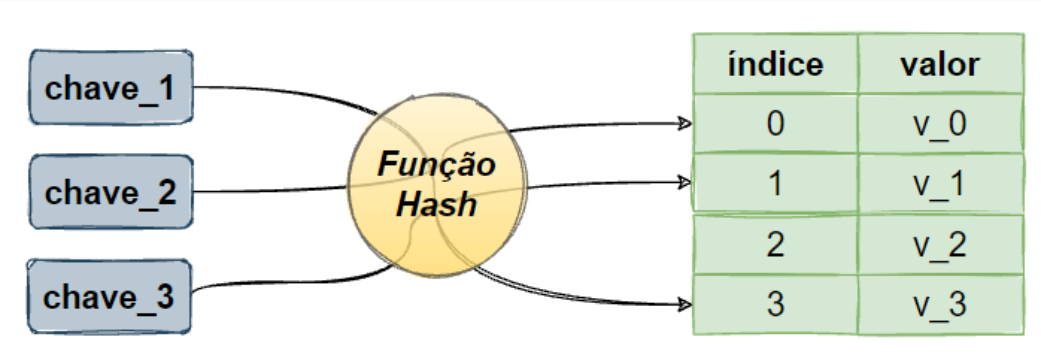


Definição de Hash: Conceitos

- Hash é uma generalização de um arranjo comum, sendo uma estrutura de dados baseada em **dicionário**.
- Dicionários são estruturas especializadas em prover as operações de inserir, pesquisar e remover entradas. Não há repetição de chaves.
- O propósito do Hash é utilizar uma função, aplicada sobre parte da informação (chave), para obter o índice onde a informação deve ou deveria estar armazenada.

Definição de Hash: Conceitos

- A função que mapeia a chave para um índice de um arranjo é chamada de Função de Dispersão (ou Hashing).
- A estrutura de dados é comumente chamada de Tabela de Dispersão ou Hash.



Objetivo: a partir de uma chave, fazer uma busca rápida e obter o valor desejado.

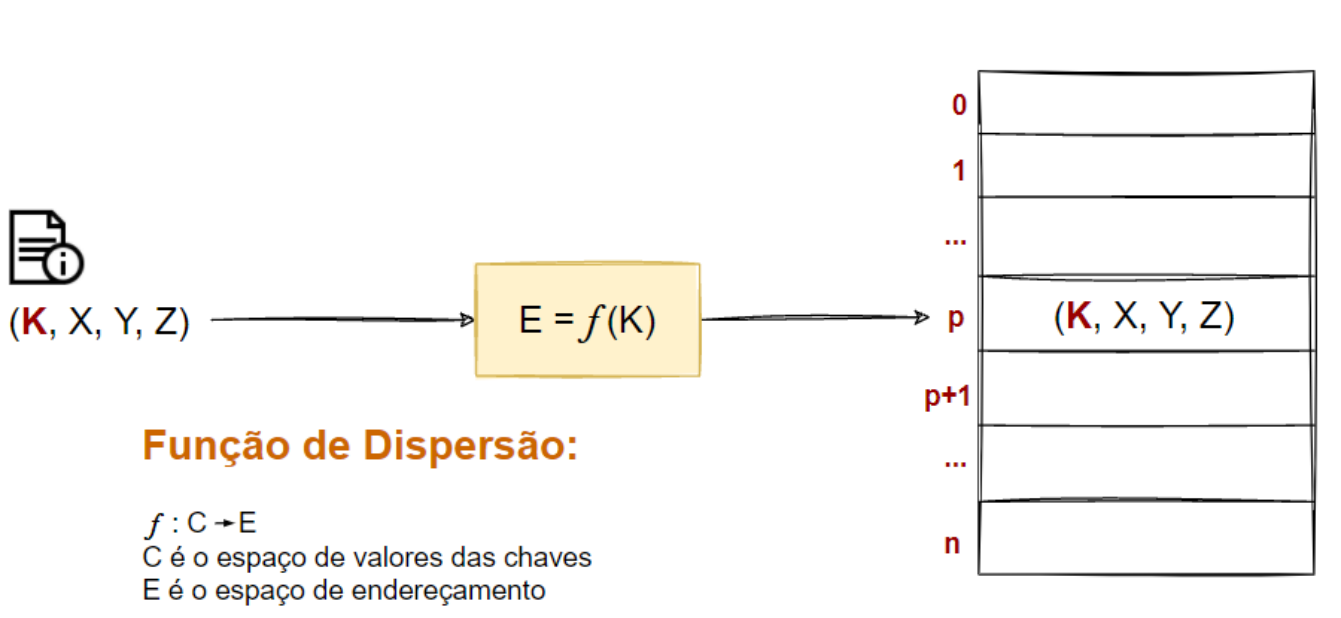
Definição de Hash: Representação

- **Vetor**: cada posição do vetor armazena um item. A função de dispersão é aplicada a um conjunto de items $I_1, I_2, \ldots, I_n$, para que sejam espalhados no vetor $V[1..n]$ que representa a tabela hash.

- **Vetor + Lista Encadeada**: cada posição do vetor contém ponteiro para uma lista, responsável por guardar os itens.

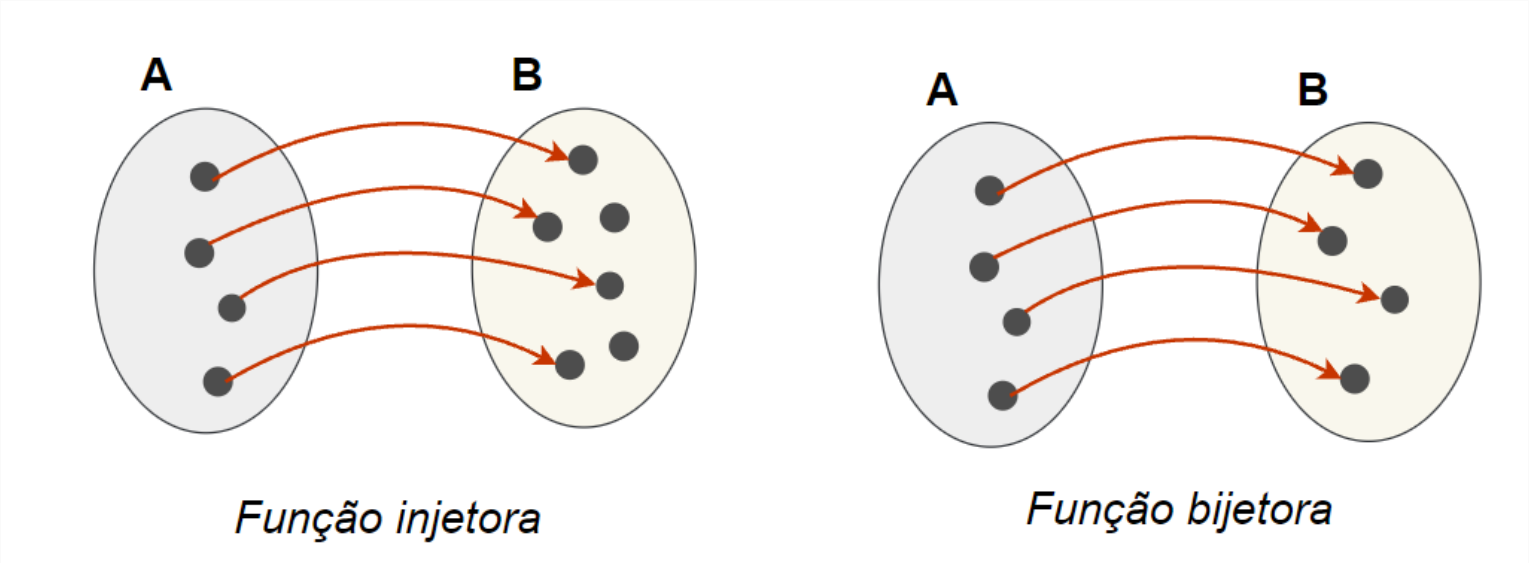
Função de Dispersão

- A **função de dispersão** é responsável por gerar um índice a partir de uma determinada chave (parte do item).
- O ideal é que a função forneça índices únicos para o conjunto das chaves de entrada possíveis.
- Os valores da chave (K) podem ser numéricos, alfabéticos ou alfa-numéricos.



Função de Dispersão: dispersão perfeita

- Para quaisquer chaves *x* e *y* diferentes e pertencentes à A, a função utilizada fornece saídas diferentes.



Função injetora *Função bijetora*

Dispersão Perfeita: Exemplo

- Armazenamento de 40 alunos de uma turma:

```
struct aluno {
   int matric;
   char nome[50];
   char telefone[20];  };
```

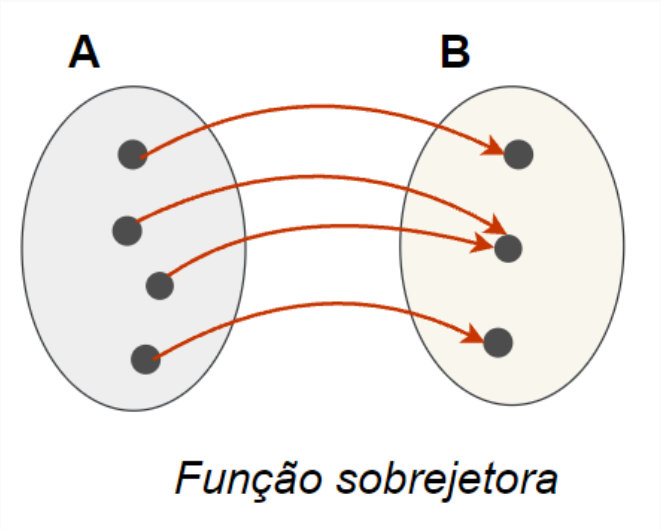
- O acesso a qualquer aluno, em ordem constante, pode ser feito se usarmos o número de matrícula do aluno como índice de um vetor.

Problema: Se a matrícula for numérica e composta de 7 dígitos, então tem-se um valor variando de 0000000 a 9999999. Portanto, precisaríamos dimensionar um vetor com **dez milhões** de posições.

Solução: Obter na matrícula uma parte que seja significativa e que identifique o aluno na turma. Se os dois últimos dígitos puderem ser utilizados, então o espaço varia de 00 a 99 (100 posições).

Função de Dispersão: dispersão imperfeita

- Existem chaves *x* e *y* diferentes e pertencentes a *A*, em que a função de dispersão fornece saídas iguais.



Função sobrejetora

Função de Dispersão: dispersão imperfeita

Exemplo: Construir uma tabela com os elementos 34, 45, 67, 78, 89, 94.

Utiliza-se um vetor com 10 elementos e a função de dispersão *x mod* 10 (resto da divisão por 10). Após a inserção dos elementos as posições da tabela contêm:

i	0	1	2	3	4	5	6	7	8	9
v[i]	-1	-1	-1	-1	**34 94**	45	-1	67	78	89

Observações:

- -1 indica que não existe elemento naquela posição.
- Os elementos 34 e 94 caíram na mesma posição: ocorreu uma **colisão**.

```
int hash(int x) {
 return x % 10;
}
void insere(int a[], int x) {
 a[hash(x)] = x;
}
```

Função de Dispersão: Características Desejáveis

Uma boa função de dispersão é essencial para garantir boa performance em tabelas hash. Uma função não adequada tende a degradar o desempenho geral da tabela.

As seguintes características são desejadas:

- Produzir um número baixo de colisões.
- Ser facilmente computável.
- Ser uniforme: idealmente, a função de dispersão deve ser tal que todos os compartimentos possuam a mesma probabilidade de serem escolhidos.

Função de Dispersão: Exemplos

Algumas funções de dispersão são bastante empregadas na prática por possuírem algumas das características desejáveis:

- Método da Divisão
- Método da Dobra
- Método da Multiplicação

Função de Dispersão: Divisão

Método da Divisão:

- Fácil, eficiente e bastante utilizado.
- A chave x é dividida pela dimensão m da tabela, e o resto da divisão é usado com endereço chave:

  $h(x) = x \ mod \ m$

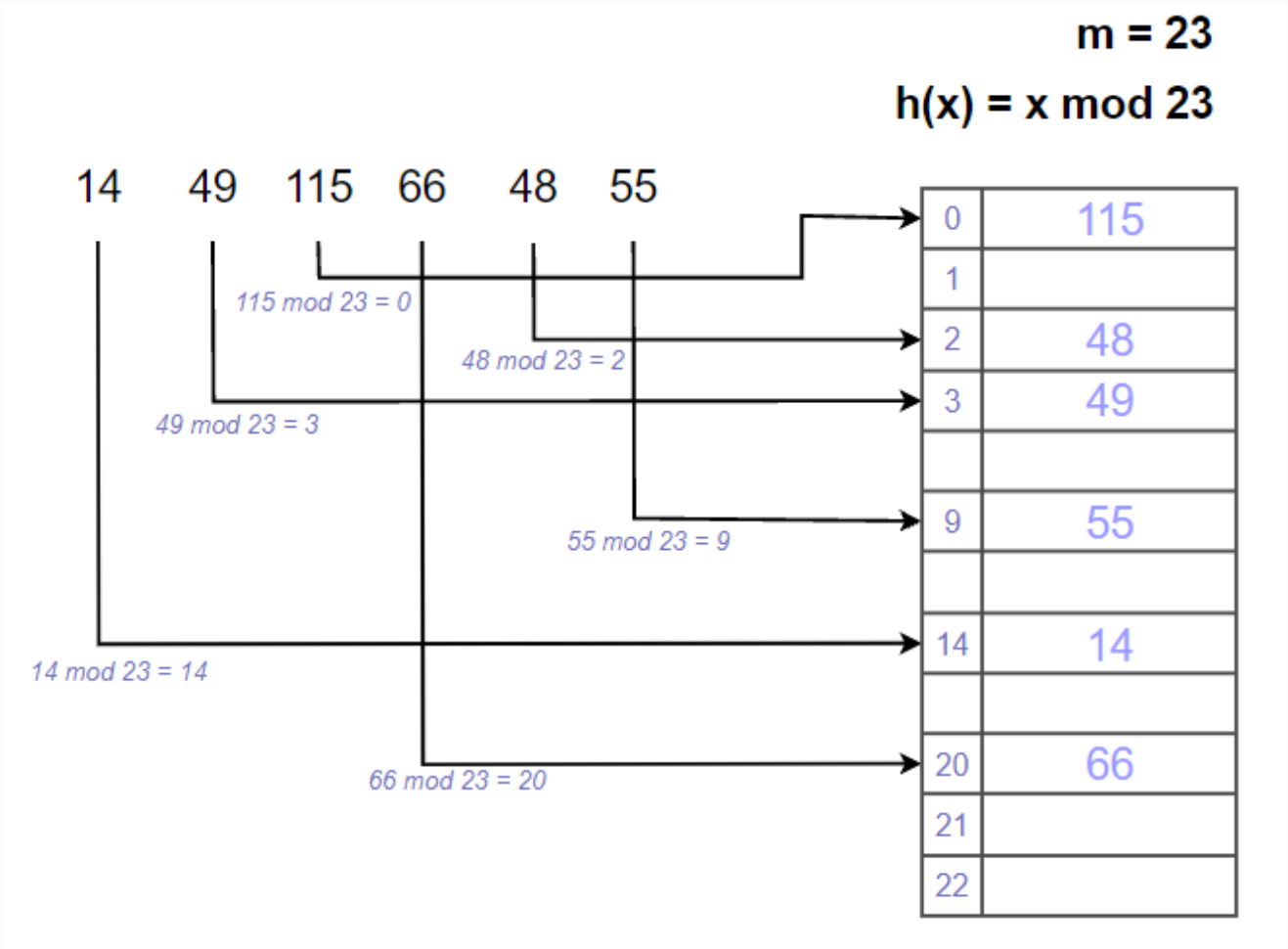
  resulta em endereços no intervalo [0, m-1]

- Estudos mostram que alguns valores de m são melhores que outros:
  - Valores a serem evitados:
    - Número par: $h(x)$ será par quando x for par e ímpar quando x for ímpar.
    - Potência de 2: $h(x)$ dependerá apenas de alguns dígitos de x.
  - Valores com boas possibilidades:
    - Número primo não próximo à uma potência de 2.
    - Número sem divisores primos menores que 20.

Função de Dispersão: Divisão

Método da Divisão:

- Exemplo com m=23



Função de Dispersão: Dobra

Método da Dobra:

- Seja a chave uma sequência de dígitos escritos num pedaço de papel. O método consiste em:
  - *dobrar* o papel, de maneira que os dígitos se sobreponham.
  - somar os dígitos sobrepostos sem considerar o *vai um*.

Exemplo: Efetuar 2 dobras de tamanho 2 na chave 359423.

1a. Dobra de tamanho 2:

- Dobra 35 em 94 e soma, descartando o ”vai um”.
- Resultado intermediário: 4723

2a. Dobra de tamanho 2:

- Dobra 47 em 23 e soma.
- Resultado final: 79

Função de Dispersão: Dobra usando bits

- Pode-se obter um endereço-base de k bits para uma chave qualquer separando-se a chave em diversas partes de k bits e utilizando a operação XOR (ou exclusivo).

  Exemplo usando bits: Realizar uma dobra de tamanho 5 na chave 68.

  68 = 00010 00100

$$\begin{array}{r} 00010 \\ \oplus\ 00100 \\ \hline 00110 \end{array}$$

  Resultado: $00110_2 = 6_{10}$

- A dobra de tamanho 5 resulta em endereços de 5 bits: valores entre 0 e 31 (base 10).

Função de Dispersão: Multiplicação

- Na utilização da multiplicação para criar uma dispersão, uma das estratégias é o **meio do quadrado**:
  - Multiplicar a chave por ela mesma
  - Armazenar o resultado em bits
  - Descartar alternadamente os bits das extremidades esquerda e direita, até obter o tamanho da sequência de bits desejada

  Exemplo: Obter o meio do quadrado para a chave 25, armazenar em 12 bits e reduzir para 6 bits:

  $25 * 25 = 625$

  $625_{10} = 0010\ 0111\ 0001_2$

0	0	1	0	0	1	1	1	0	0	0	1
~~0~~	~~0~~	~~1~~	0	0	1	1	1	0	~~0~~	~~0~~	~~1~~
			0	0	1	1	1	0			

  $1110_2 = 14_{10}$ (6 bits $\rightarrow$ valores entre 0 e 63)

Função de Dispersão: Multiplicação

- Outra alternativa é o método **congruente linear multiplicativo** (CLM), usado para gerar números aleatórios:
  - Definir uma constante fracionária C tal que 0 < C < 1.
  - Multiplicar o valor da chave por C e armazenar em V.
  - Obter a parte inteira da multiplicação da parte fracionária de V com o tamanho da tabela.

  Exemplo: Para uma tabela de 500 posições, encontrar a posição da chave 151617 usando o método CLM:

  C = 0.324516
  151617 * 0.324516 = 49202.142372
  0.142372 * 500 = 71.186
  71

  A chave 151617 deverá ser armazenada na posição 71 da tabela.

Tabela de Dispersão: Colisão

Qualquer que seja a função de espalhamento, algumas colisões irão ocorrer fatalmente, e tais colisões têm de ser resolvidas.

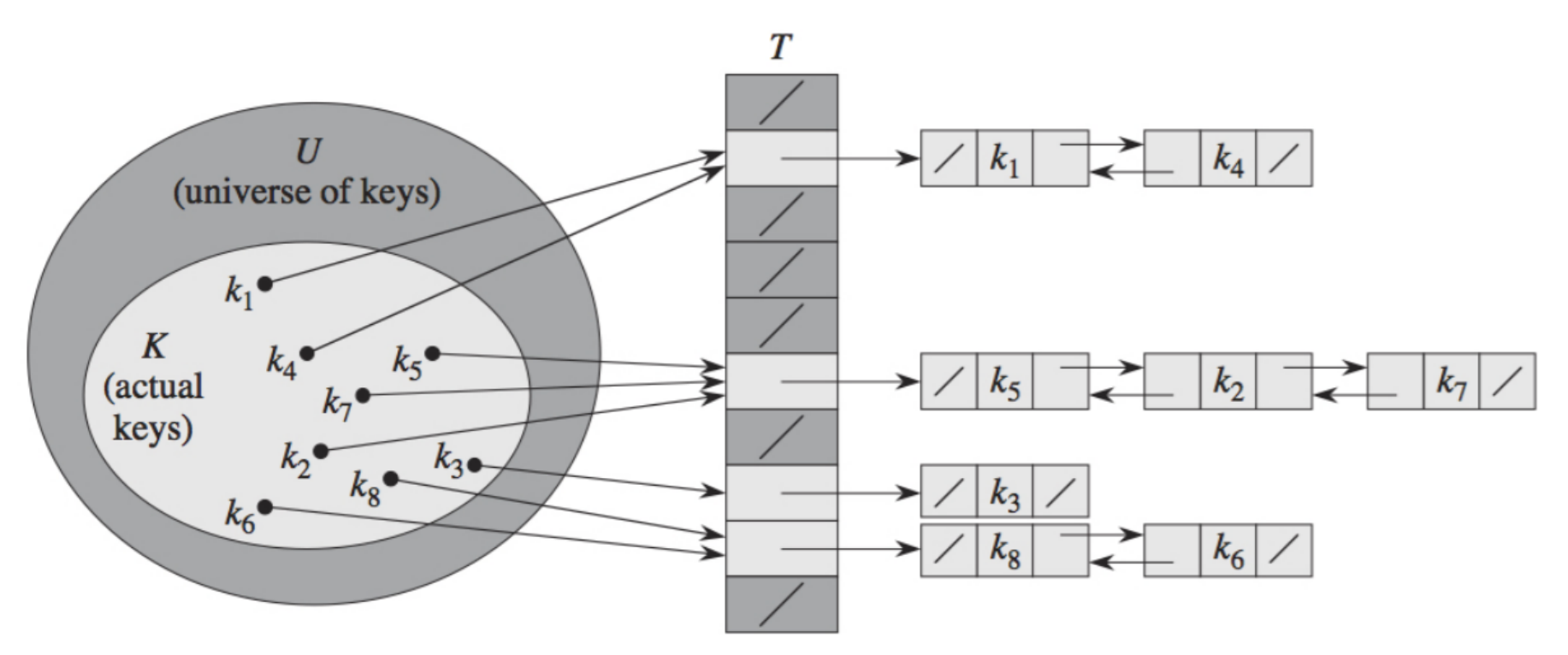
A construção de uma tabela de dispersão eficiente consiste de:

- uma função de espalhamento (hashing).
- abordagem para o tratamento de **colisões**:
  - **encadeamento separado**: cada posição da tabela aponta para o início de uma lista dinâmica encadeada, que armazena as colisões (itens com chaves iguais) desta posição.
  - **endereçamento aberto**: os elementos são armazenados na própria tabela de dispersão, e no caso de colisão, busca-se uma posição ainda não ocupada na tabela.

Tabela de Dispersão: Encadeamento Separado

- Uma das formas de resolver as colisões é simplesmente construir uma **lista linear encadeada** para cada endereço da tabela.
- Desta forma, todas as chaves com mesmo endereço são encadeadas em uma lista linear.
- Cada posição da tabela de dispersão **T[j]** contém uma lista encadeada com todas as chaves cujo hash seja **j**.

Tabela de Dispersão: Encadeamento Separado



- Na figura, h(k1) = h(k4) e h(k5) = h(k7) = h(k2).
- Se listas **duplamente encadeadas** forem utilizadas, a remoção de elementos se torna mais rápida.

Tabela de Dispersão: Encadeamento Separado

Exemplo de Implementação - Encadeamento Separado

Considere uma coleção de termos em português e seus sinônimos mais comuns na língua inglesa.

```
grande large
soletrar spell
adicionar add
mesmo even
terra land
aqui here
necessário must
grande big
....
```

Estratégia inicial de inclusão na tabela: Palavras idênticas terão seu significado substituído, serão armazenadas uma única vez. Não há armazenamento de palavras repetidas.

Tabela de Dispersão: Encadeamento Separado

Estruturas de dados para a manipulação dos pares (*termo português - termo inglês*) em uma Tabela Hash, com encadeamento separado em listas ligadas:

```
1  struct termo{
2     char port[30];
3     char engl[30];
4     struct termo *prox;
5  };
6  typedef struct termo Termo;
7
8  struct tabelaHs {
9     int tamanho;
10    Termo **tab;
11 };
12 typedef struct tabelaHs TabelaHs;
```

Tabela de Dispersão: Encadeamento Separado

Funções básicas para a criação, inclusão, busca e percurso na Tabela Hash:

```
1 TabelaHs *criar (int tam);
2
3 void inserirTermoTh(TabelaHs *tabTermos, char *ptChave, char *enValor);
4
5 char *buscarTermoTh(TabelaHs *tabTermos, char *ptChave);
6
7 void percursoTh(TabelaHs *tabTermos);
```

Tabela de Dispersão: Encadeamento Separado

```
1  /* Criação de uma nova tabela hash. */
2  TabelaHs *criar (int tam) {
3
4    TabelaHs *tabelaHash = NULL;
5    int i;
6
7    if(tam < 1) return NULL;
8
9      /* Alocação de memória para a tabela (estrutura). */
10     if((tabelaHash = malloc(sizeof(TabelaHs))) == NULL ) {
11         return NULL;
12     }
13     /* Alocação de memória para o vetor de ponteiros associado à tabela. */
14     if((tabelaHash->tab = malloc(sizeof(Termo *) * tam)) == NULL) {
15         return NULL;
16     }
17     for(i = 0; i < tam; i++ ) {
18         tabelaHash->tab[i] = NULL;
19     }
20     tabelaHash->tamanho = tam;
21     return tabelaHash;
22 }
```

Tabela de Dispersão: Encadeamento Separado

```
1  /* Inserir par chave/valor (pt/en) numa tabela hash. */
2  void inserirTermoTh(TabelaHs *tabTermos, char *ptChave, char *enValor) {
3      int bin = 0;  Termo *novoT = NULL, *pos = NULL, *ult = NULL;
4
5      bin = ht_hash(tabTermos,ptChave); /* Gera posição de espalhamento */
6      pos = tabTermos->tab[bin];
7      while(pos != NULL  && strcmp(ptChave, pos->port ) > 0 ) {
8          ult = pos;  pos = pos->prox;  }
9
10     /* Se a chave já existe, substituir o valor. */
11     if(pos != NULL && strcmp(ptChave, pos->port ) == 0 )
12         strcpy(pos->engl,enValor);
13     else {              /* Se chave não existe, inserir novo par (termo) */
14         novoT = criarNovoTermo(ptChave, enValor);
15         if(pos == tabTermos->tab[bin]) {   /* Inserção no início da lista. */
16             novoT->prox = pos;
17             tabTermos->tab[bin] = novoT; }
18         else
19             if (pos == NULL)                    /* Inserção ao final da lista. */
20                 ult->prox = novoT;
21             else  {                             /* Inserção ordenada */
22                 novoT->prox = pos;
23                 ult->prox = novoT;
24             }
25     } }
```

Tabela de Dispersão: Encadeamento Separado

```
1  /* Espalhamento das chaves (termos em portugues) */
2  int ht_hash(TabelaHs *tabTermos, char *pt) {
3      unsigned long int hashval = 0;
4      int i = 0;
5
6      while(hashval < ULONG_MAX && i < strlen(pt) ) {
7          hashval = hashval << 8;  // deslocamento de oito bits à direita
8          hashval += pt[i];
9          i++;
10     }
11     return hashval % tabTermos->tamanho;
12 }
13 // ULONG_MAX -> valor máximo possível para unsigned long
14 // usar #include <limits.h>
```

Tabela de Dispersão: Encadeamento Separado

Exemplo de execução da função **ht_hash** para uma tabela hash de tamanho 1029, tendo como entrada a chave **casa**:

```
hashval = 0
     -> desloca 8 bits e inclui a letra 'c' (99 - 01100011)
     -> 00000000 01100011
hashval = 99
     -> desloca 8 bits e inclui a letra 'a' (97 - 01100001)
     -> 00000000 01100011 01100001
hashval = 25441
     -> desloca 8 bits e inclui a letra 's' (115 - 01110011)
     -> 00000000 01100011 01100001 01110011
hashval = 6513011
     -> desloca 8 bits e inclui a letra 'a' (97 - 01100001)
     -> 00000000 01100011 01100001 01110011 01100001
hashval = 1667330913
hashval = 1667330913 mod 1029
hashval = 24
```

Tabela de Dispersão: Encadeamento Separado

```
1  /* Buscar um par chave/valor (termo pt/en) em uma tabela hash. */
2  char *buscarTermoTh(TabelaHs *tabTermos, char *chv ) {
3      int bin = 0;
4      Termo *par;
5      bin = ht_hash(tabTermos, chv);
6
7      /* Alcança a posição bin para buscar o valor da chave. */
8      par = tabTermos->tab[bin];
9      while( par != NULL && strcmp( chv, par->port ) > 0 ) {
10         par = par->prox;
11     }
12
13     if( par == NULL || strcmp( chv, par->port ) != 0 ) {
14         return NULL;
15     }
16     else {
17         return par->engl;
18     }
19 }
```

Tabela de Dispersão: Endereçamento Aberto

Endereçamento Aberto

- Quando uma chave k é endereçada à uma posição já ocupada na tabela, então é realizado o cálculo de posições alternativas $h_1(k)$, $h_2(k)$, ... dentro da tabela que possam ser ocupadas.
- A nova posição de inserção do novo elemento pode ser calculada a partir das estratégias:
  - Sondagem Linear
  - Sondagem Quadrática
  - Duplo Hash

Endereçamento Aberto: Sondagem Linear

- **Sondagem Linear**: Após a colisão, as novas posições vão sendo calculadas de forma sequencial, com início na posição gerada (e já ocupada) para o elemento em questão.
- **Funcionamento**:
  - Suponha que o endereço-base de uma chave k é $h(k)$.
  - Já existe uma chave ocupando o endereço $h(k)$.
  - Método: tentar armazenar k no endereço consecutivo a $h(k)$. Se este também já estiver ocupado, tenta-se o próximo e assim sucessivamente.
- Considera-se que a tabela é uma estrutura **circular**.

Endereçamento Aberto: Sondagem Quadrática

- **Sondagem Quadrática**: Tenta espalhar os elementos utilizando como incremento, uma função quadrática (equação do segundo grau).
- **Exemplo de função**:

  $pos + (c_1 * i) + (c_2 * i^2)$

  pos é a posição obtida pela função de espalhamento

  c_1 e c_2 são os coeficientes da equação

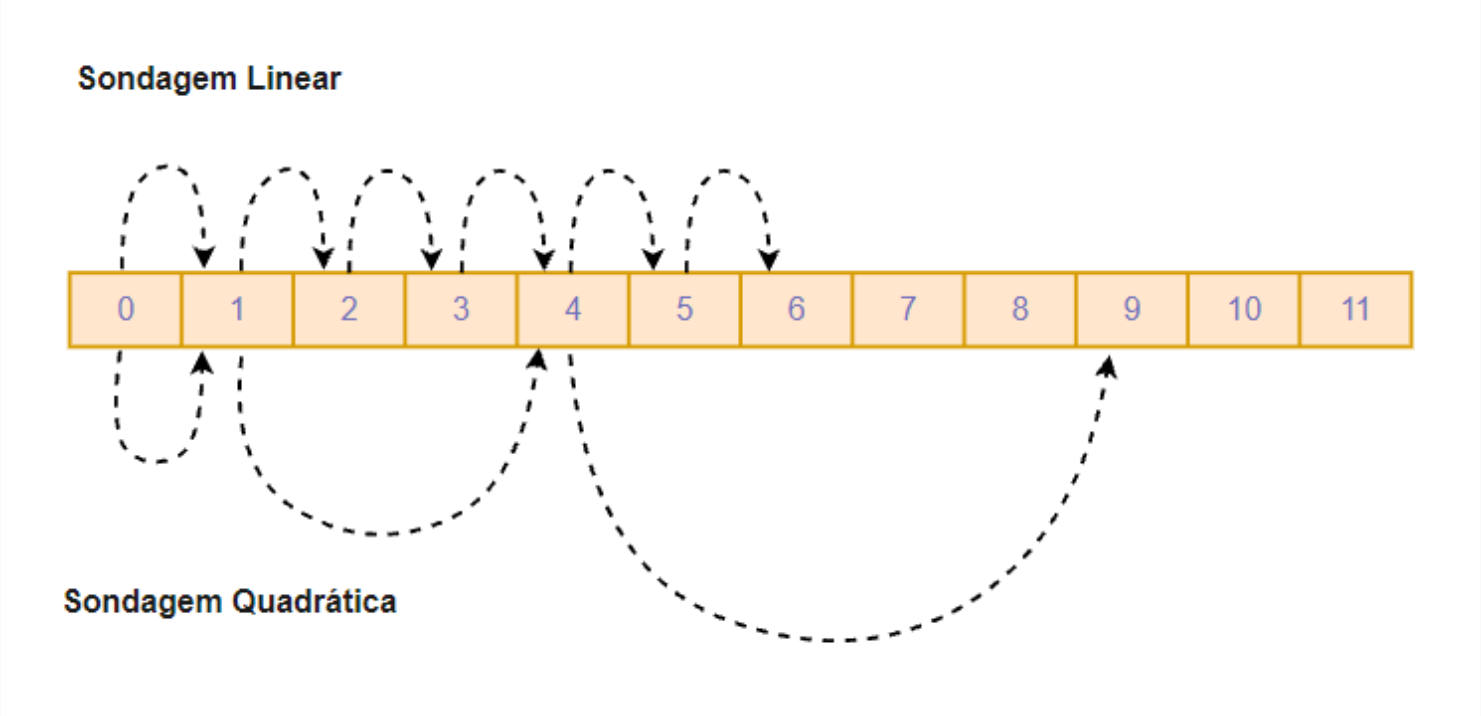
- **Passos**:

  i=0 o primeiro elemento é colocado na posição *pos* retornada pela função de espalhamento (hash).

  i=1 segundo elemento: $pos + (c_1 * 1) + (c_2 * 1^2)$

  i=2 terceiro elemento: $pos + (c_1 * 2) + (c_2 * 2^2)$

Endereçamento Aberto: Sondagem Linear X Quadrática



Endereçamento Aberto: Duplo Hash

- **Duplo hash**: Tenta espalhar os elementos utilizando duas funções de hashing:
  - a primeira função de hashing, H1, é utilizada para calcular a posição inicial do elemento;
  - a segunda função de hashing, H2, é utilizada para calcular os deslocamentos em relação a posição inicial (no caso de uma colisão).
- A posição de um novo elemento é calculada por: **H1 + i * H2** sendo i é tentativa atual de inserção.
- **Passos:**
  - Primeiro elemento (i = 0) é colocado na posição obtida por H1
  - Segundo elemento (colisão) é colocado na posição H1 + 1 * H2
  - Terceiro elemento (nova colisão) é colocado na posição H1 + 2 * H2

Tabela de Dispersão: Conclusão

Resumo:

- A tabela de dispersão é uma estrutura do tipo dicionário, e por isso:
  - não são armazenadas chaves repetidas
  - não é possível recuperar os elementos de forma ordenada
- A função de dispersão (hash) pode ser otimizada através do estudo da natureza e domínio da chave a ser espalhada.
- No pior caso a complexidade das operações pode ser O(N) se todos as chaves inseridas sofrerem colisão.
- Tabelas de dispersão com endereçamento aberto podem necessitar de redimensionamento.
- Para a resolução de colisões, o encadeamento é o método mais simples, mas gasta mais espaço.

Referência e Material Extra

- Celes, W.; Cerqueira, R.; Rangel, J.L. *Introdução a Estruturas de Dados com Técnicas de Programação em* C. 2a. ed. Elsevier, 2016.
- Tabelas Hash - Vanessa Braganholo (UFF) *http://www2.ic.uff.br/ vanessa/material/ed/13-TabelasHash.pdf*
* Backes, A. *Programação Descomplicada - Estruturas de Dados*.
    - Vídeo-aulas 79 a 84: *https://programacaodescomplicada.wordpress.com/indice/estrutura-de-dados/*