Hashing (Tabela de Dispersão)

MOTIVAÇÃO

- Os métodos de pesquisa vistos até agora buscam informações armazenadas com base na comparação de suas chaves
- Para obtermos algoritmos eficientes, armazenamos os elementos ordenados e tiramos proveito dessa ordenação
- hashing (tabela de dispersão) ou método de cálculo de endereço
  - No caso médio é possível encontrar a chave em tempo constante

CONCEITOS BÁSICOS

- Índices em vetores ou listas sequenciais são utilizados para acessar informações
- No entanto, se quisermos acessar uma informação de um determinado conteúdo (e não posição)?
  - temos que procurá-lo

Família

1	2	3	4	5	6
Diogenes	Cristina	Gilson	Patricia	Marilia	Biu

```
Família[1] = "Diogenes"
Família[3] = "Gilson"
Família[2] = "Cristina"
```

Em qual posição está "Biu" ?

CONCEITOS BÁSICOS

- Ideal
  - Parte da informação poderia ser utilizada na recuperação ou busca(consulta)

DEFINIÇÃO DE HASH

- Hash é uma generalização da noção mais simples de um arranjo comum, sendo uma estrutura de dados do tipo dicionário

- Dicionários são estruturas especializadas em prover operações de inserir, pesquisar e remover.

DEFINIÇÃO DE HASH

- A idéia central do Hash é utilizar uma função, parte da informação (chave), para retornar o índice onde a informação deve ou deveria estar armazenada.

DEFINIÇÃO DE HASH

- Hash é uma generalização da noção mais simples de um arranjo comum, sendo uma estrutura de dados do tipo dicionário

- Dicionários são estruturas especializadas em prover as operações de inserir, pesquisar e remover.

DEFINIÇÃO DE HASH

- A idéia central do Hash é utilizar uma função, sobre parte da informação (chave), para retornar o índice onde a informação deve ou deveria estar armazenada.

DEFINIÇÃO DE HASH

- Esta função que mapeia a chave para um índice de *M* arranjo é chamada de Função de Hashing

- A estrutura de dados Hash é comumente chamada de Tabela Hash.

DEFINIÇÃO DE HASH

Tabela Hash

19	**123.456.781-00; Zezinho; Av. Canavial. Nº 45.**
20	
...	
37	**143.576.342-23;Izaque; Rua Oliva. Nº 27.**
...	
50	**345.365.768-93; Gugu ; Av. Atlântica. S/N.**
...	
85	**879.094.345-45 ; Ze de Camargo; Rua B. Nº 100.**
...	

TABELA HASH

- Tabela Hash
  - É uma estrutura de dados especial;

TABELA HASH

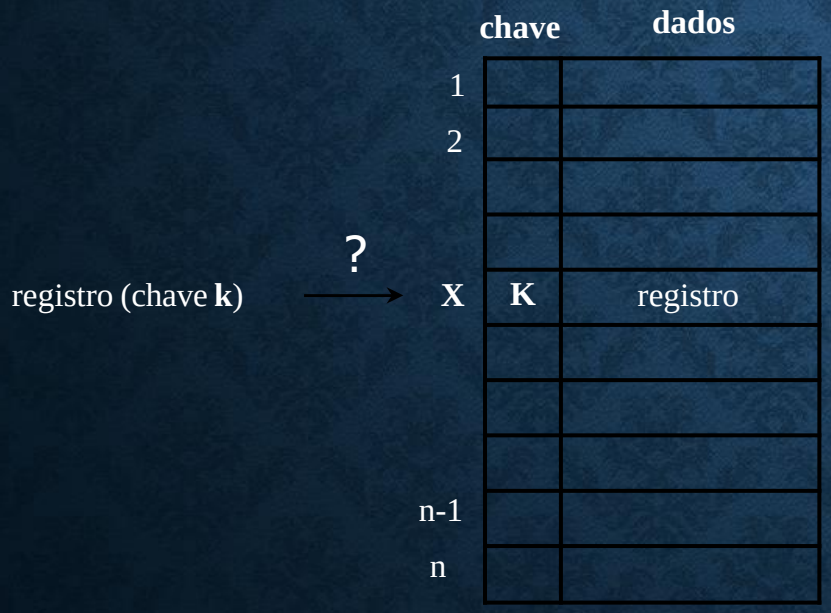
- Tabela Hash
  - É uma estrutura de dados especial;
  - Armazena as informações desejadas associando chaves;

TABELA HASH

- Tabela Hash
  - É uma estrutura de dados especial
  - Armazena as informações desejadas associando chaves

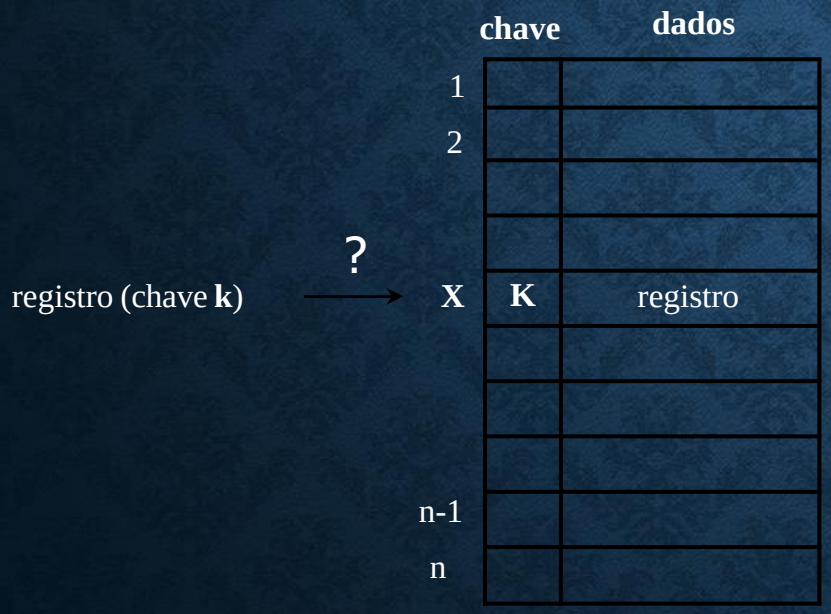
- Objetivo
  - a partir de uma chave, fazer uma busca rápida e obter o valor desejado.

ILUSTRAÇÃO DE UMA TABELA HASH



Como o registro (com chave **K**) foi armazenado na posição **X** na Tabela Hash ao lado?

ILUSTRAÇÃO DE UMA TABELA HASH



Como o registro (com chave **K**) foi armazenado na posição **X** na Tabela Hash ao lado?

Resp: Através de uma **Função de Hashing**.

COMO REPRESENTAR TABELAS HASH?

- Vetor ou lista sequencial
  - cada posição do vetor guarda uma informação.
  - Se a função de hashing aplicada a um conjunto de elementos determinar as informações $i_1, i_2, ..., i_n$
    - então o vetor $V[1... n]$ é usado para representar a tabela hash

- mas somente vetor?

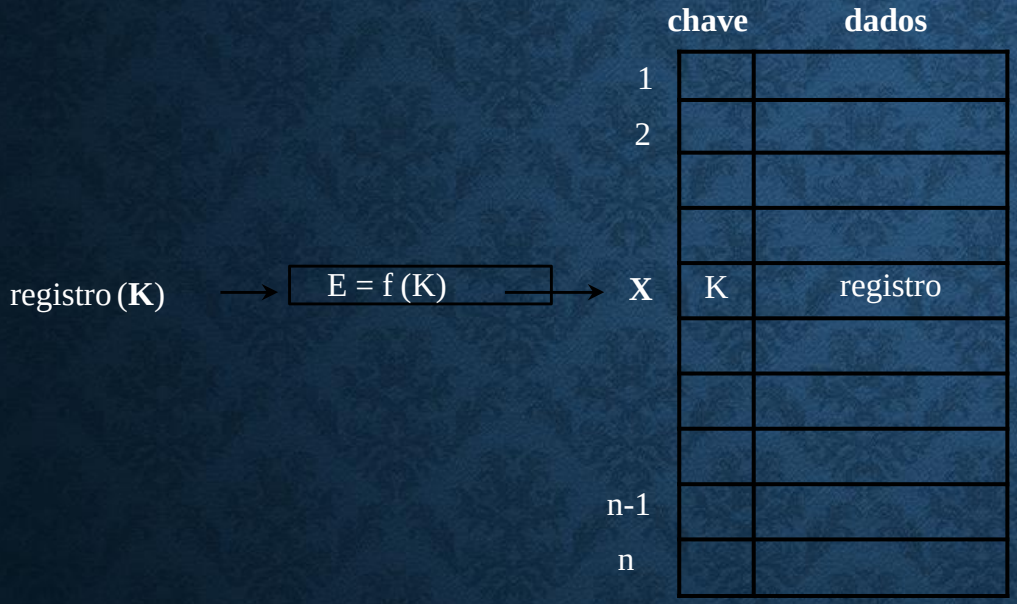
FUNÇÃO DE HASHING (OU DISPERSÃO)

- A Função de Hashing é a responsável por gerar uma tabela de uma determinada chave
  - O ideal é que a função forneça índices únicos para o conjunto das chaves de entrada possíveis
    - sem colisões
    - fácil de computar
    - uniforme (todos os locais da tabela sejam igualmente utilizados)

FUNÇÃO DE HASHING (OU DISPERSÃO)

- A Função de Hashing é a responsável por gerar uma tabela de uma determinada chave
  - O ideal é que a função forneça índices únicos para o conjunto das chaves de entrada possíveis
    - sem colisões
    - fácil de computar
    - uniforme (todos os locais da tabela sejam igualmente utilizados)
  - extremamente importante, pois ela é responsável por distribuir as informações pela Tabela Hash

ILUSTRAÇÃO DA FUNÇÃO DE HASHING



- Os valores da chave **numéricos**, **alfabéticos** ou **alfa-numéricos**.

- Executam a transformação do valor em um endereço, pela aplicação de operações aritméticas e/ou lógicas

f: C → E

f: função de cálculo de endereço

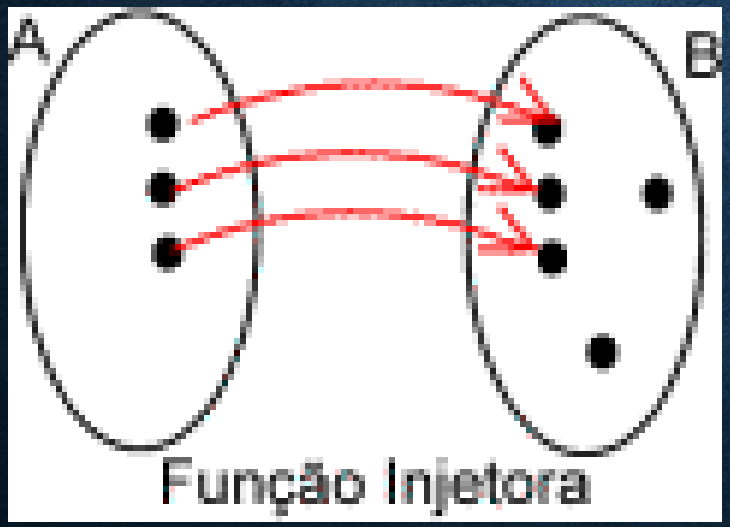
C: espaço de valores da chave (domínio de f)

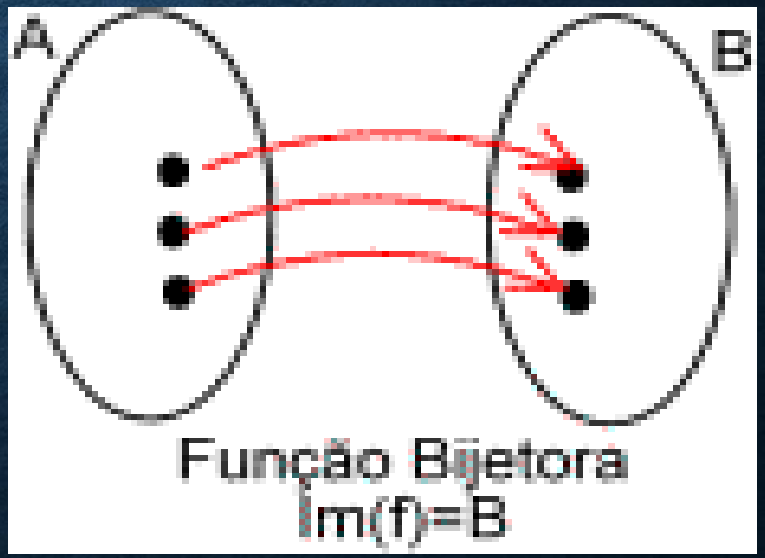
E: espaço de endereçamento (contradomínio de f)

HASHING PERFEITO

Característica:

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





EXEMPLO DE HASHING PERFEITO

- Armazenamento de alunos de uma determinada turma de um curso específico
- Cada aluno é identificado unicamente pela sua matrícula.

```
struct aluno {
        int mat;                // 4 bytes       =  4 bytes
        char nome[81];          // 1 byte * 81 = 81 bytes
        char email[41];         // 1 byte * 41 = 41 bytes
};                              //  Total  = 126 bytes

typedef struct aluno Aluno;
```

EXEMPLO DE HASHING PERFEITO

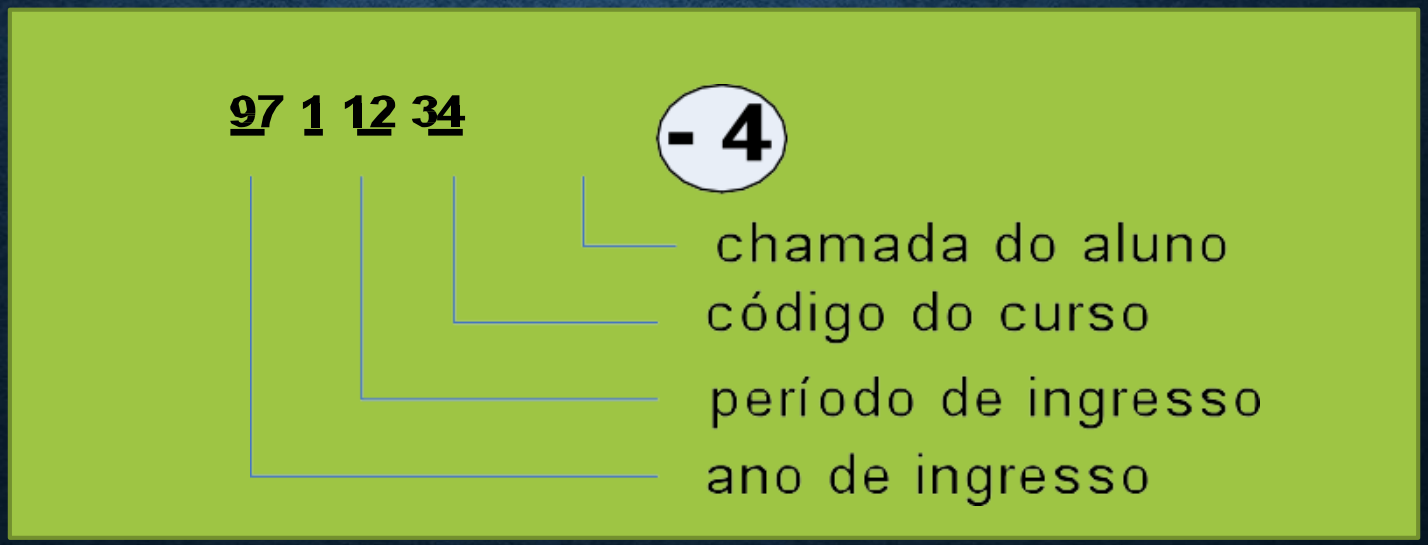
- O número de dígitos efetivos na matrícula são 7
- Para permitir um acesso a qualquer aluno em tempo constante, podemos usar o número de matrícula do aluno como índice de um vetor
- Um problema é que pagamos um preço alto para ter esse acesso rápido. Porque?

EXEMPLO DE HASHING PERFEITO

- O número de dígitos efetivos na matrícula são 7
- Para permitir um acesso a qualquer aluno em tempo constante, podemos usar o número de matrícula do aluno como índice de um vetor
- Um problema é que pagamos um preço alto para ter esse acesso rápido. Porque?
  - Visto que a matrícula é composta de 7 dígitos, então podemos esperar uma matrícula variando de 0000000 a 9999999. Portanto, precisaríamos dimensionar um vetor com DEZ Milhões de elementos

EXEMPLO DE HASHING PERFEITO

- Como economizar em espaço, mas ainda usando *hashing perfeito*
  - Identificando as partes significativas da chave:



EXEMPLO DE HASHING PERFEITO

- mostra a dimensão que a Tabela Hash deverá ter

- Por exemplo, dimensionando com apenas 100 elementos, ou seja,

```
aluno* tabAlunos[100];
```

- Função que aplicada sobre matrículas de alunos retorna os índices únicos da tabela

EXEMPLO DE HASHING PERFEITO (6/6)

- Supondo que a turma seja do 2º semestre de 20 (código 052) e do curso de Sistemas de Informação (código 35).

Qual seria a função de hashing **perfeito** !?

```
int funcao_hash(int matricula) {
    int valor = matricula – 0523500;
        if((valor >= 0) & (valor <=99)) then
            return valor;
        else
            return -1;
}
```

EXEMPLO DE HASHING PERFEITO (6/6)

- Supondo que a turma seja do 2º semestre de 20 (código 052) e do curso de Sistemas de Informação (código 35).

Qual seria a função de hashing **perfeito** !?

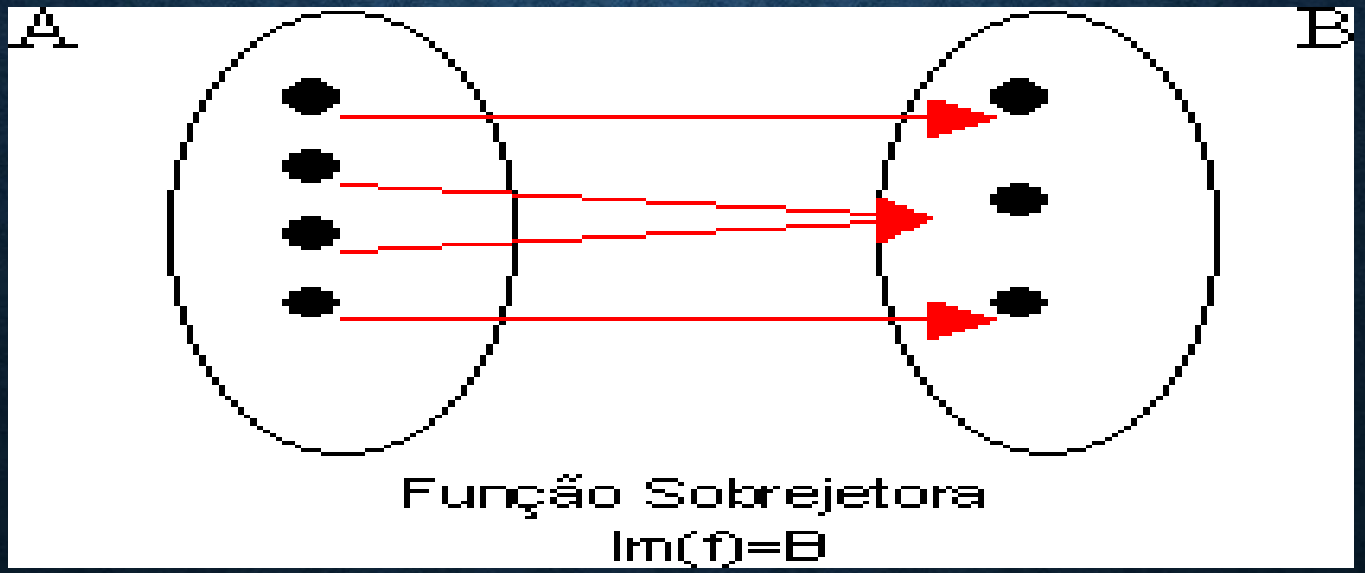
```
int funcao_hash(int matricula) {
    int valor = matricula – 0523500;
        if((valor >= 0) & (valor <=99)) then
            return valor;
        else
            return -1;
}
```

Acesso: dada **mat** tabAlunos[funcao_hash(mat)]

HASHING IMPERFEITO

Características:

- Existe chaves x e y diferentes e pertencentes a A, onde a função Hash utilizada fornece saídas iguais;



EXEMPLO DE HASHING IMPERFEITO

- Suponha que queiramos armazenar as seguintes chaves: C, H, A, V, E e S em um vetor de P = 7 posições (0..6) conforme a seguinte
  - função f(k) = k(código ASCII) % P.
- Exemplo

símbolo	ASCII
C	67
H	72
A	65
V	86
E	69
S	83

EXEMPLO DE HASHING IMPERFEITO

- Suponha que queiramos armazenar as seguintes chaves: C, H, A, V, E e S em um vetor de P = 7 posições (0..6) conforme a seguinte
  - função f(k) = k(código ASCII) % P.
- Exemplo

símbolo	ASCII	f(k)
C	67	4
H	72	2
A	65	2
V	86	2
E	69	6
S	83	6

COLISÕES

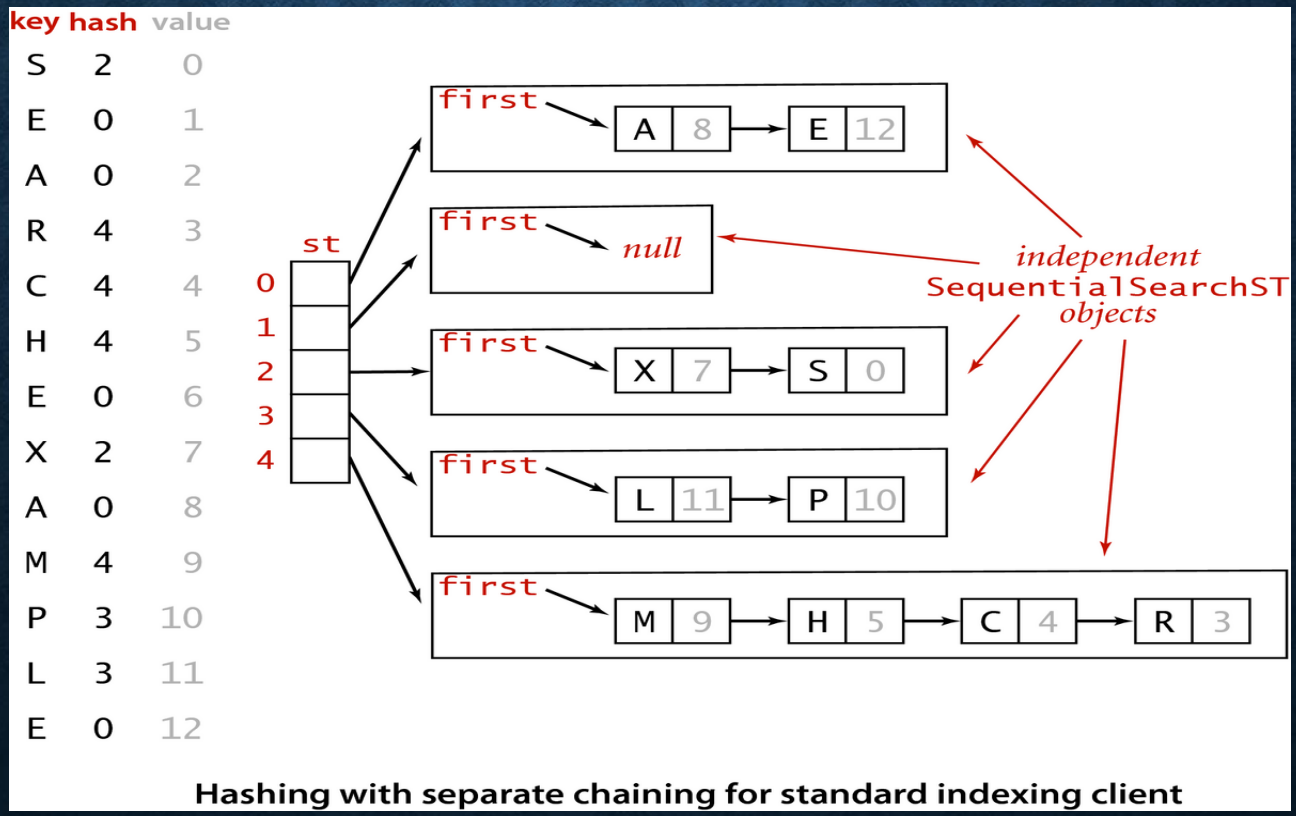
- Quando duas ou mais chaves geram o endereço da Tabela Hash
- É comum ocorrer colisões.
- Principais causas:
  - em geral o número N de chaves possíveis é muito maior que o número m disponíveis na tabela
  - não se pode garantir que as funções de hashing possuam um bom potencial de distribuição (espalhamento)

COLISÕES: ENCADEAMENTO

- Cada entrada na tabela aponta para uma lista encadeada
  - colisões geram uma nova entrada em uma lista
- A função utilizada deve ser uniforme para evitar uma grande lista encadeada em poucas posições da tabela
- Cada busca só será constante se o número de elementos em cada lista encadeada for pequeno

COLISÕES: ENCADEAMENTO

- A informação encadeadas é armazenada em estruturas

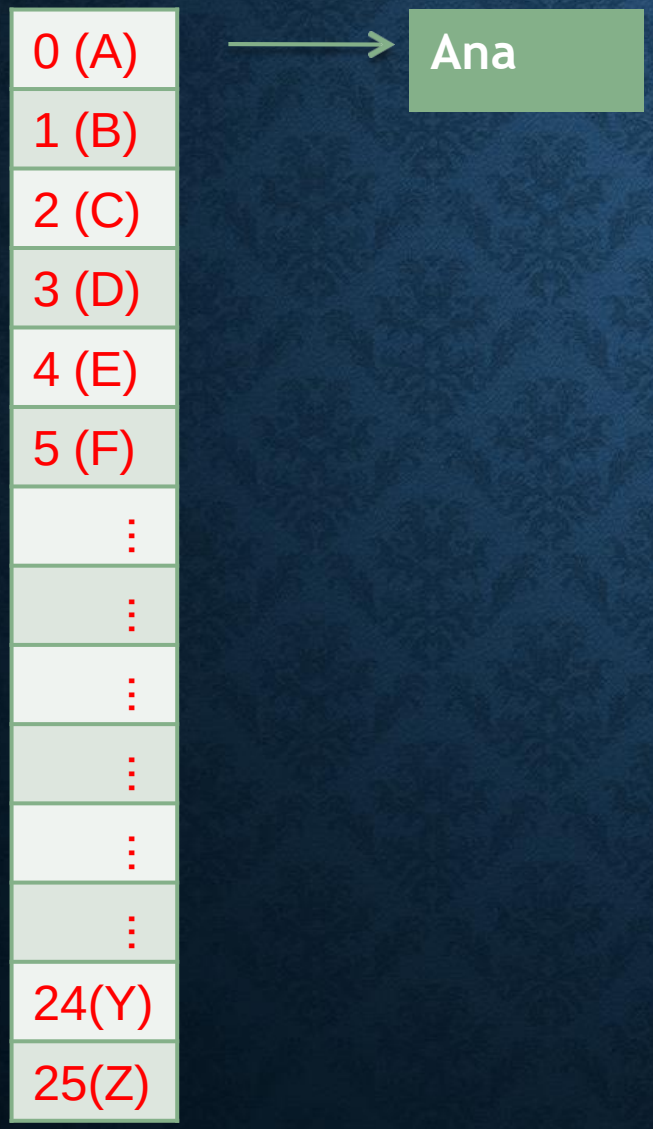


Hashing with separate chaining for standard indexing client

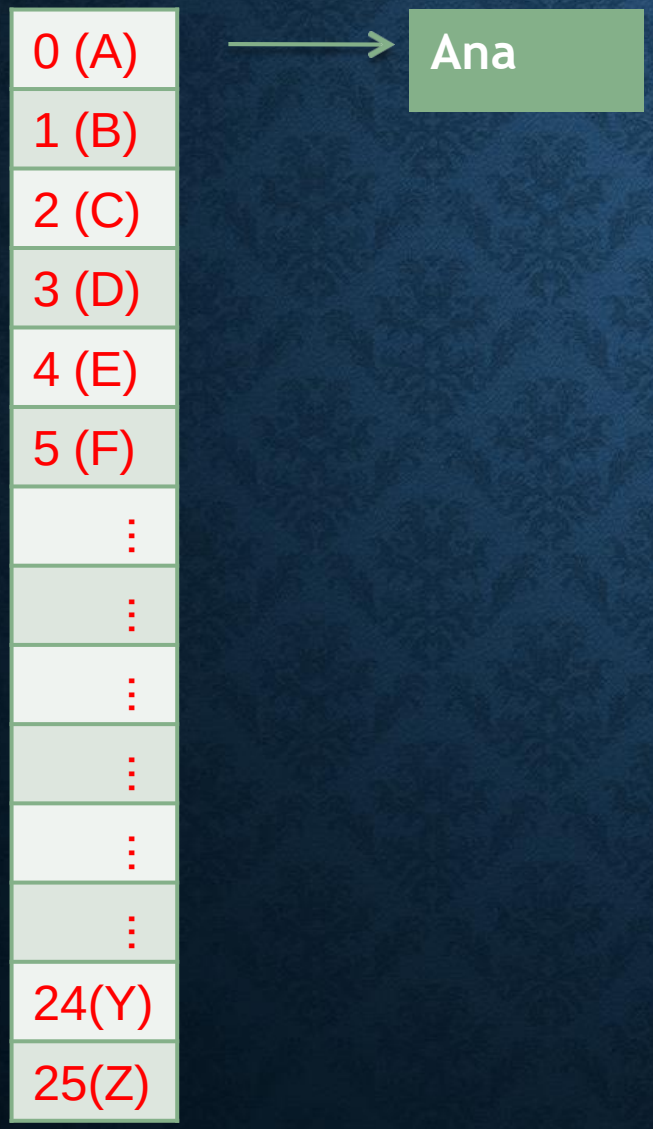
COLISÕES: ENCADEAMENTO

insere “Ana”

COLISÕES: ENCADEAMENTO

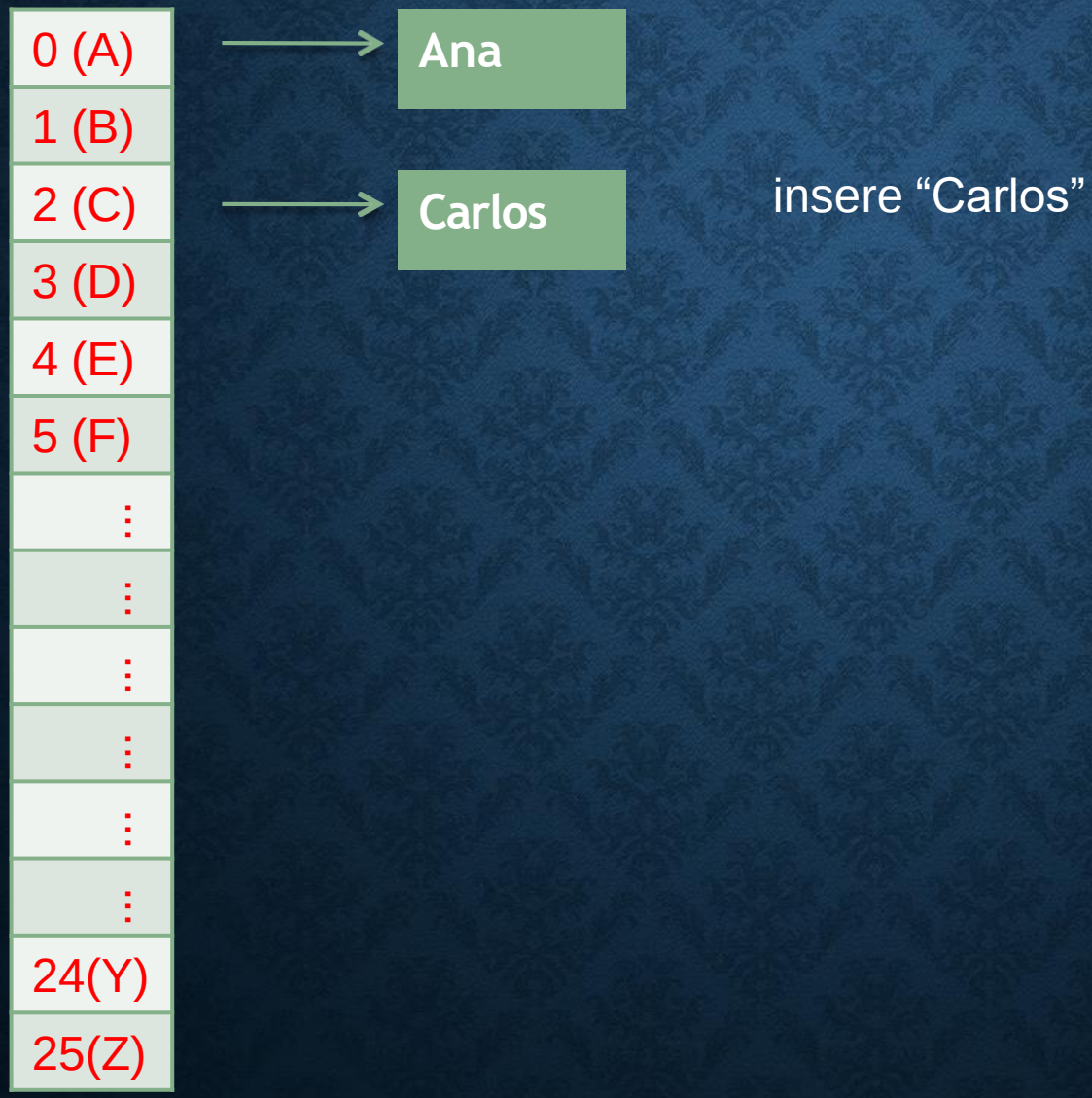


COLISÕES: ENCADEAMENTO



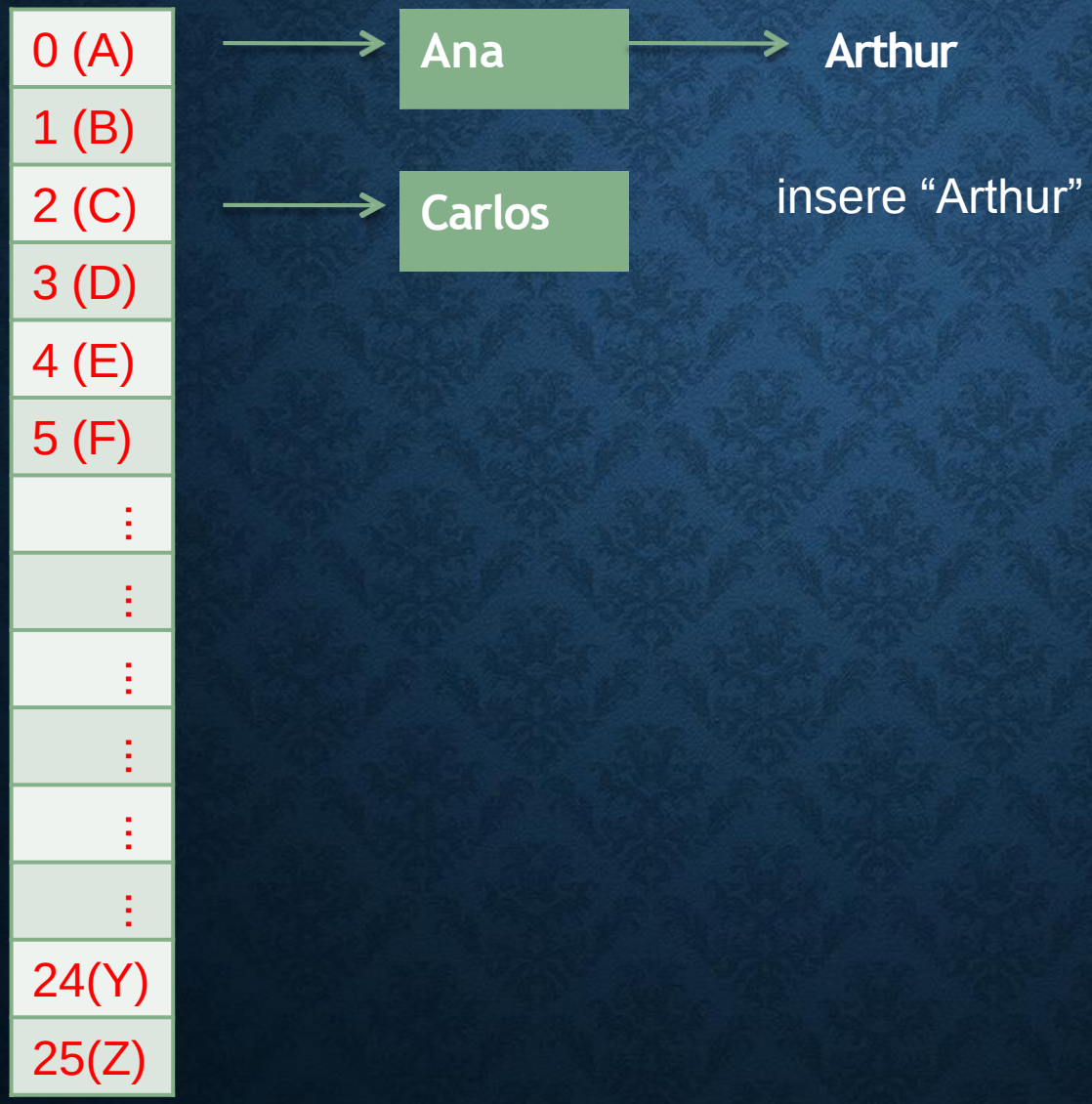
insere “Carlos”

COLISÕES: ENCADEAMENTO



COLISÕES: ENCADEAMENTO

COLISÕES: ENCADEAMENTO



COLISÕES: ENCADEAMENTO

COLISÕES: ENCADEAMENTO

COLISÕES: ENDEREÇAMENTO ABERTO

- sem listas encadeadas
- sem informação adicional
- quando houver colisões – através de um cálculo qual o próximo local a ser examinado
- sucesso: vai calculando até achar uma posição ou encontra a chave
- sem sucesso: a tabela está cheia ou não se encontra a chave

COLISÕES: ENDEREÇAMENTO ABERTO

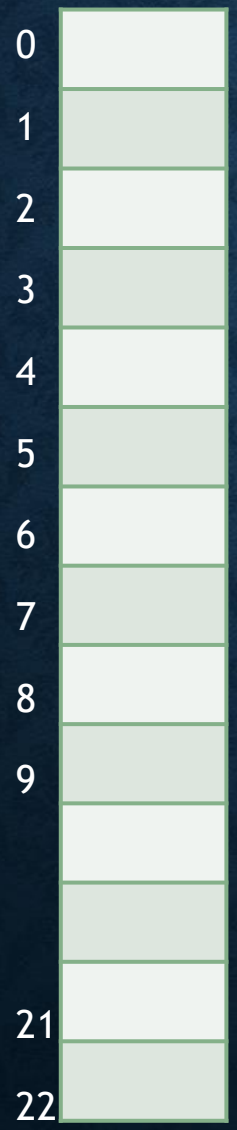
- supõe-se que exista uma função hash h() que um endereço base h(x,k) distinto para cada k de 0 a m
- inicialmente, veremos endereçamento aberto
  - tentativa/sondagem linear
  - tentativa/sondagem quadrática

COLISÕES: SONDAGEM LINEAR

- Ao verificar que uma posição h(k) da tabela está ocupada, tenta adicionar o elemento na primeira posição livre seguinte:
  - h(k) + 1, h(k) + 2, h(k) + 3, ... até uma posição vazia, considerando a tabela circular
  - função hash

$$h(x,k) = (h'(x) + k) \bmod m \qquad 0 <= k <= m-1$$

COLISÕES: SONDAGEM LINEAR

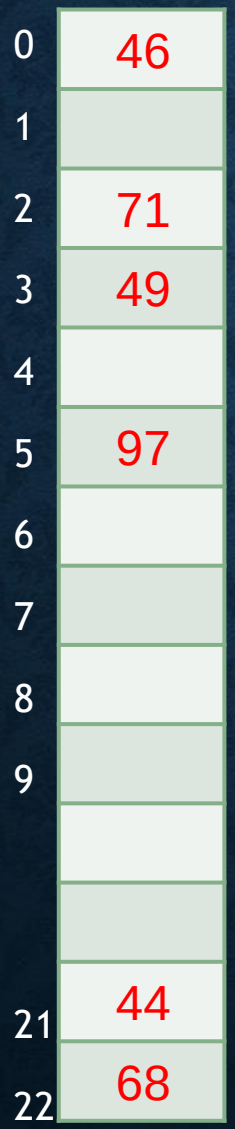


- m = 23

$h(x,k) = (h'(x) + k) \bmod m$

- Ao inserir
- 44 k = 0, 44 mod 23 = 21
- 46 k = 0, 46 mod 23 = 0
- 49 k = 0, 49 mod 23 = 3
- 68 k = 0, 68 mod 23 = 22
- 7 1 k = 0, 71 mod 23 = 3
- 97 k = 0, 44 mod 23 = 4

COLISÕES: SONDAGEM LINEAR



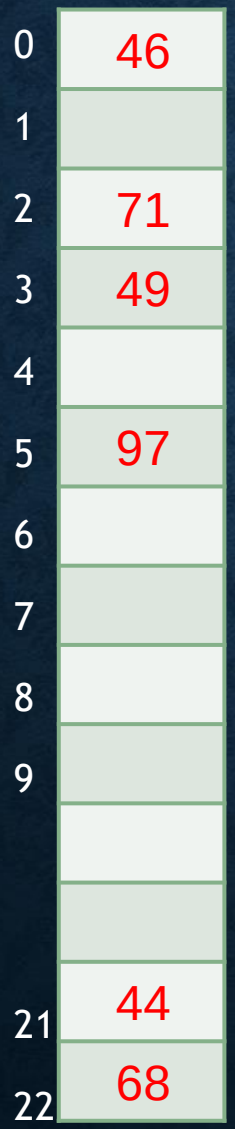
- m = 23

$h(x,k) = (h'(x) + k) \bmod m$

- Ao inserir

- 44 ✂ k = 0, 44 mod 23 = 21
- 46 ✂ k = 0, 46 mod 23 = 0
- 49 ✂ k = 0, 49 mod 23 = 3
- 68 ✂ k = 0, 68 mod 23 = ~~21~~
- 7 1 ✂ k = 0, 71 mod 23 = 3
- 97 ✂ k = 0, 44 mod 23 ~~4~~

COLISÕES: SONDAGEM LINEAR

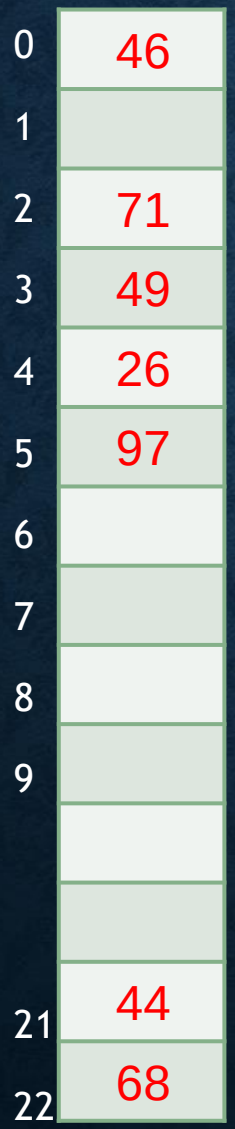


- m = 23

$$h(x,k) = (h'(x) + k) \bmod m$$

- inserir 26
  - k = 0, 26 mod 23 = 3
  - k = 1, 26+1 mod 23 = 4

COLISÕES: SONDAGEM LINEAR

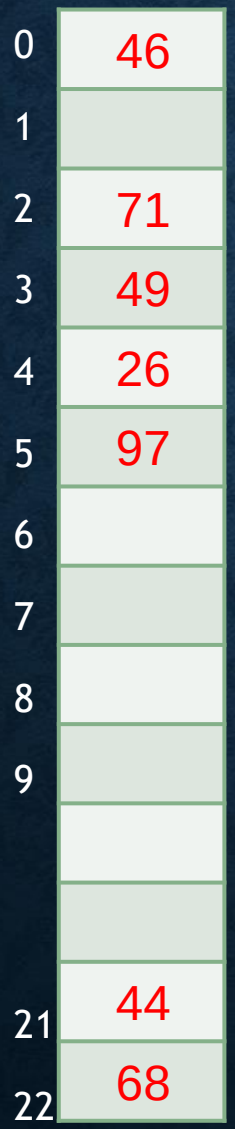


- m = 23

$$h(x,k) = (h'(x) + k) \bmod m$$

- inserir 26
  - k = 0, 26 mod 23 = 3
  - k = 1, 26+1 mod 23 = 4

COLISÕES: SONDAGEM LINEAR

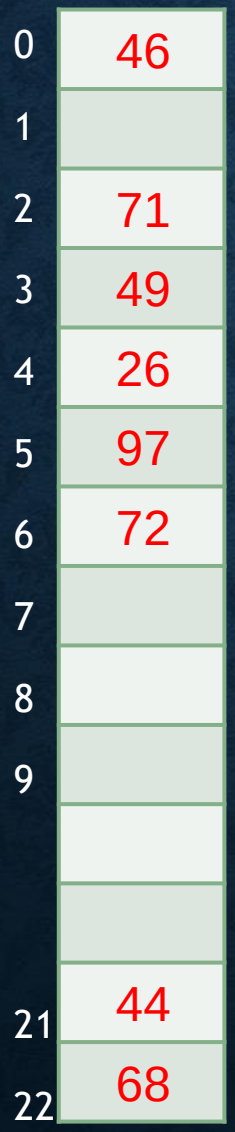


- m = 23

$$h(x,k) = (h'(x) + k) \bmod m$$

- inserir 72
  - k = 0, 72 mod 23 = 3
  - k = 1, 72+1 mod 23 = 4
  - k = 2, 72+2 mod 23 = 5
  - k = 3, 72+3 mod 23 = 6

COLISÕES: SONDAGEM LINEAR



- m = 23

$h(x,k) = (h'(x) + k) \bmod m$

- Ao inserir
- inserir 72
  - k = 0, 72 mod 23 = 3
  - k = 1, 72+1 mod 23 = 4
  - k = 2, 72+2 mod 23 = 5
  - k = 3, 72+3 mod 23 = 6

COLISÕES: SONDAGEM LINEAR

- Consequências:
  - Cria grandes blocos de dados numa mesma região da tabela
  - Dificulta a remoção de dados
  - Aumenta a complexidade para a busca nos casos de colisão
  - limitado pelo tamanho da tabela

COLISÕES: SONDAGEM QUADRÁTICA

- Tentativa de se espalhar mais os elementos
- função

$$h(x,k) = (h'(x) + c_1 k + c_2 k^2) \bmod m$$

- onde c_1 e c_2 são constantes, $c_2 \neq 0$ e $k = 0, \ldots, m-1$

COLISÕES: SONDAGEM QUADRÁTICA

- Tentativa de se espalhar mais os elementos
- função

$$h(x,k) = (h'(x) + c_1 k + c_2 k^2) \bmod m$$

- onde c_1 e c_2 são constantes, $c_2 \neq 0$ e k = 0, ..., m-1
- exemplo:

$$h(x,0) = h'(x)$$

$$h(x,k) = (h(x, k-1) + k) \bmod m \quad 0 < k < m-1$$

- tentar no exemplo anterior: as tentativas são as mesmas? O número de tentativas

COLISÕES: SONDAGEM QUADRÁTICA

- Como ficaria o algoritmo de
  - busca
  - inserção
  - remoção
- de uma chave v para endereçamento aberto em
  - sondagem linear
  - sondagem quadrática