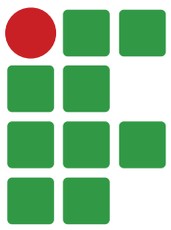
07-Hash Tables

BCC - Estruturas de Dados

Prof. Dr. Paulo César Rodacki Gomes
paulo.gomes@ifc.edu.br

Blumenau, 2022



INSTITUTO FEDERAL
Catarinense | Campus Blumenau

Tópicos

- Tabelas de dispersão
- Tratamento de colisão
- Implementação
- Referências:
  - Waldemar Celes, Renato Cerqueira, José Lucas Rangel, Introdução a Estruturas de Dados, Editora Campus (2004)
  - Michael T. Goodrich, Roberto Tamassia. Estruturas de Dados e Algoritmos em Java. 5a Ed. Bookman (2013)

INSTITUTO FEDERAL
Catarinense
Campus Blumenau

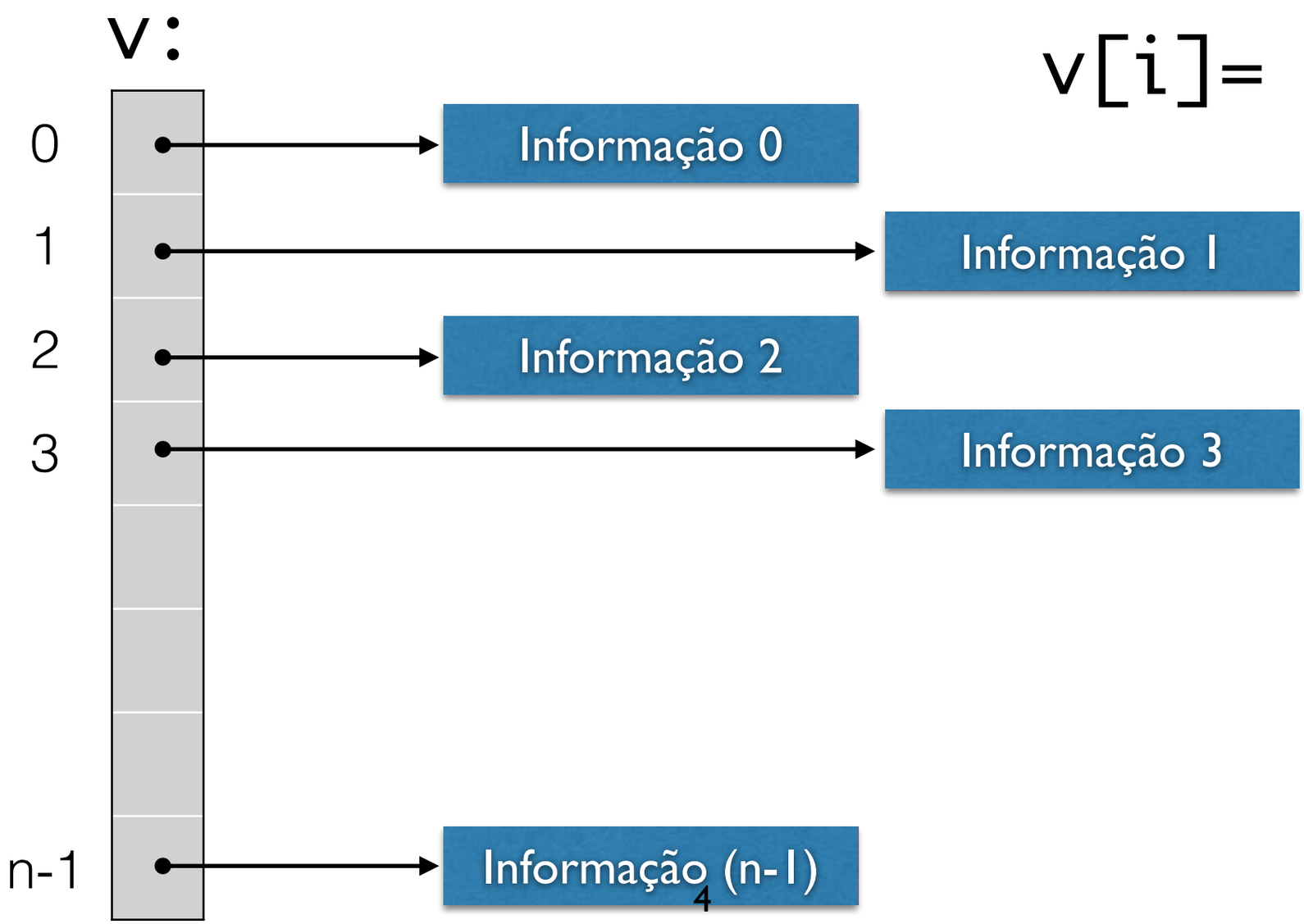
Motivação

- acesso a elementos em um array é O(1) se tivermos conhecimento de sua posição
- se o array estiver ordenado, podemos usar busca binária com custo O(log n)
- por outro lado… a indexação é restrita a números inteiros, limitados pelo tamanho do array

Motivação

- Arranjos (Arrays)

```
Info v[N];
...
v[i]= ...;
```



Motivação

- nem sempre temos chaves numéricas pequenas…
- Como fazer para indexar pelo nome ou pela matrícula?

Aluno
- nome : String - matricula : int - mediageral : float - turma : String - email : String
+set... +get...

INSTITUTO FEDERAL Catarinense Campus Blumenau

Motivação

- Escaninhos: a primeira letra do nome define onde a informação é depositada

Tabela de dispersão

- Uma tabela de dispersão (ou tabela hash) é composta por um arranjo de "buckets" e uma função de dispersão (ou função hash)
- O arranjo de buckets é um array A de tamanho N. Cada célula de A é considerada como um bucket (ou seja, um container) para pares chave-elemento

Tabela de dispersão

- Exemplo de um arranjo de buckets de tamanho 11 para as entradas (1,D), (25,C), (3,F), (14,Z), (6,A), (39,C) e (7,Q).

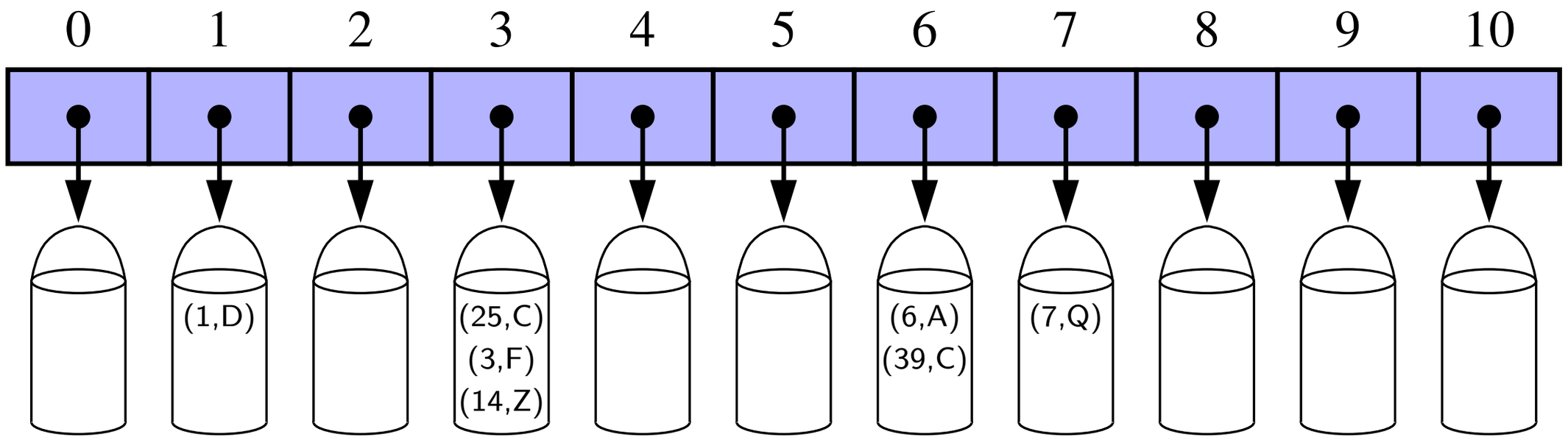


Tabela de dispersão

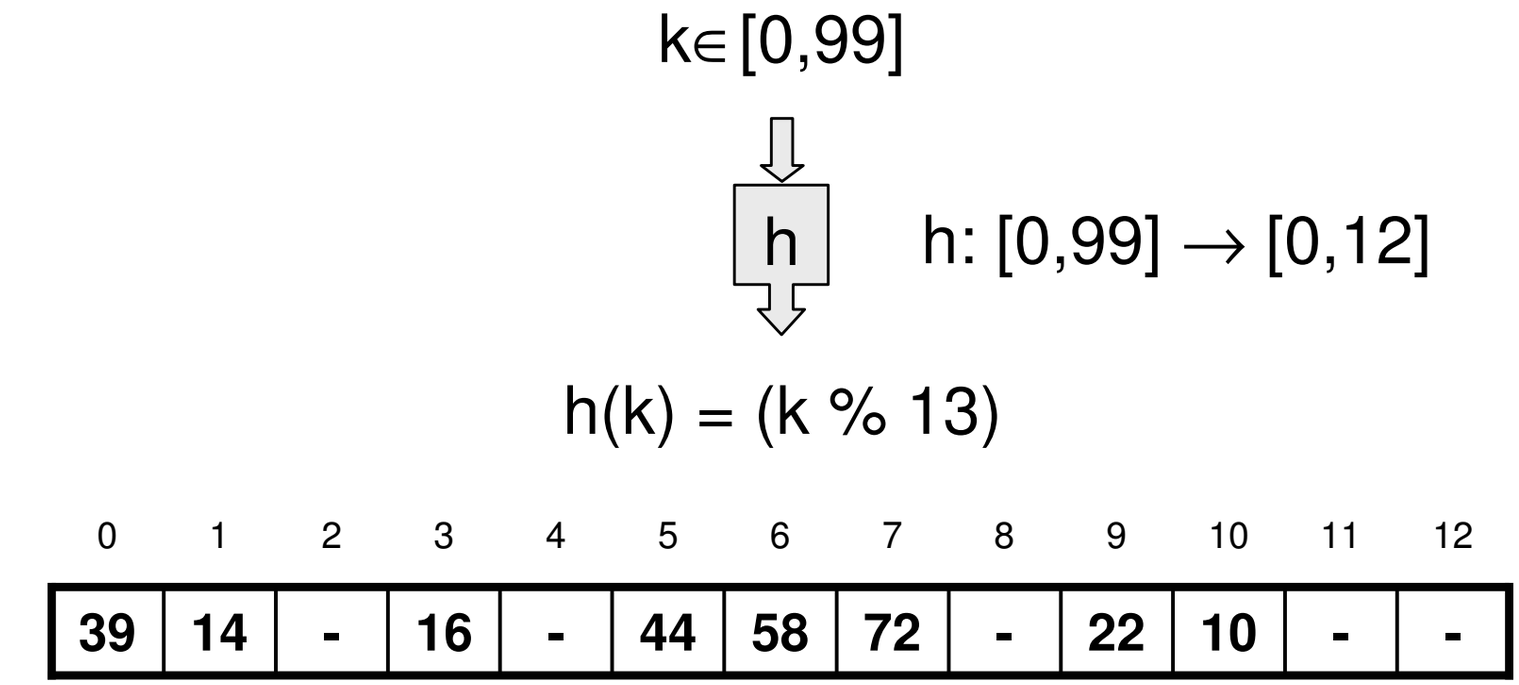
- A função hash mapeia uma chave k para um inteiro no intervalo [0, N-1], onde N é o tamanho do arranjo de buckets
- A idéia é usar o valor da função hash h(k) como índice do arranjo de busckets (ao invés de usar diretamente a chave k)
- O ítem (k, e) é armazenado na posição A[h(k)]

Tabela de dispersão

- se duas ou mais chaves resultarem no mesmo valor de hash, são mapeadas para o mesmo bucket no arranjo, neste caso dizemos que houve colisão, e esta precisa ser resolvida
- uma função hash é considerada boa se o mapeamento de chaves para posições no arranjo causa a menor quantidade de colisões possível

Tabelas de Dispersão

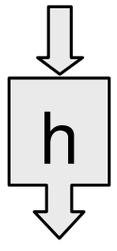
- Tabelas de dispersão (hash tables):
  - utilizadas para buscar um elemento em ordem constante O(1)
  - necessitam de mais memória, proporcional ao número de elementos armazenado



Tabelas de Dispersão

- Função de dispersão (função de *hash*):
  - mapeia uma chave de busca em um índice da tabela
  - colisão = duas ou mais chaves de busca são mapeadas para um mesmo índice da tabela de *hash*
  - deve apresentar as seguintes propriedades:
    - ser eficientemente avaliada (para acesso rápido)
    - espalhar bem as chaves de busca (para minimizarmos colisões)

$$k \in [0,99]$$



$$h: [0,99] \rightarrow [0,12]$$

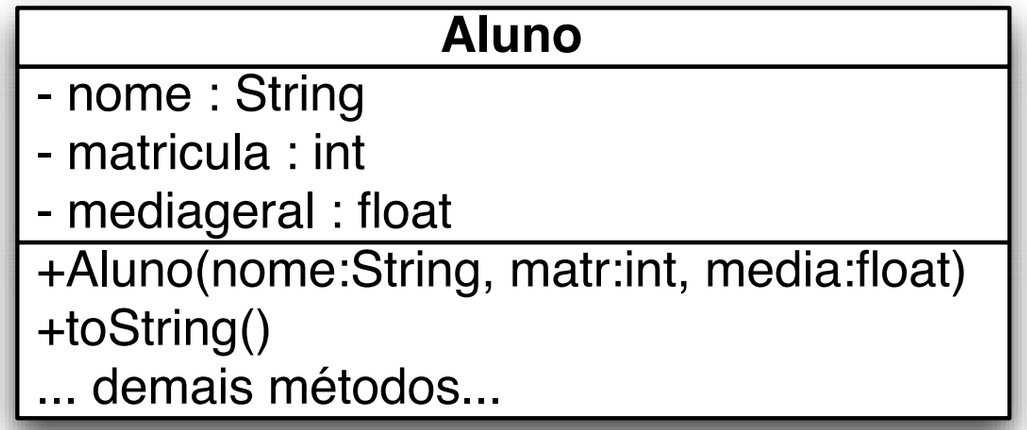
$$h(k) = (k \% 13)$$

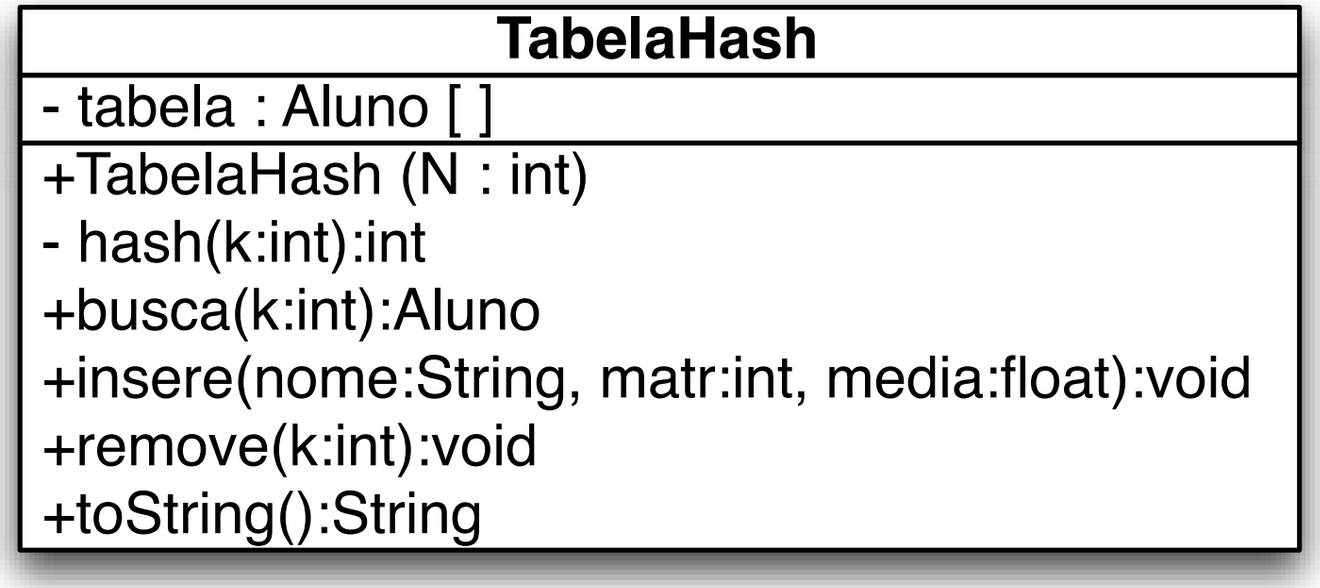
Tabelas de Dispersão

- Dimensão da tabela:
  - deve ser escolhida para diminuir o número de colisões
  - costuma ser um valor primo
  - a *taxa de ocupação* não deve ser muito alta:
    - a taxa de ocupação não deve ser superior a 75%
    - uma taxa de 50% em geral traz bons resultados
    - uma taxa menor que 25% pode representar um gasto excessivo de memória

Tabelas de Dispersão

- Exemplo





Tratamento de Colisão

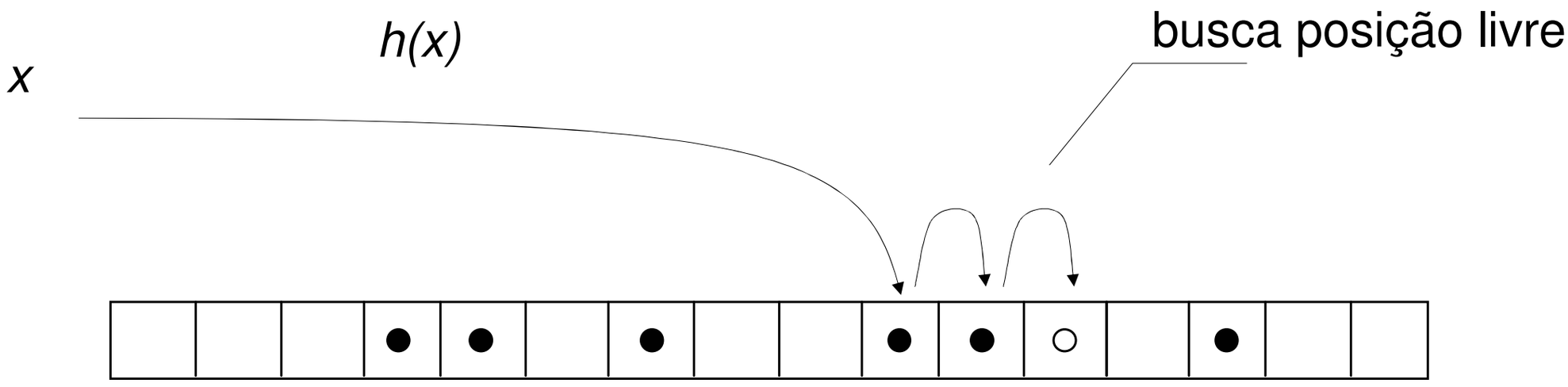
- Estratégias para tratamento de colisão:
  - uso da primeira posição consecutiva livre:
    - simples de implementar
    - tende a concentrar os lugares ocupados na tabela
  - uso de uma segunda função de dispersão:
    - evita a concentração de posições ocupadas na tabela
    - usa uma segunda função de dispersão para re-posicionar o elemento
  - uso de listas encadeadas:
    - simples de implementar
    - cada elemento da tabela hash representa um ponteiro para uma lista encadeada

Tratamento de Colisão

- Uso da posição consecutiva livre:
  - estratégia geral:
    - armazene os elementos que colidem em outros índices, ainda não ocupados, da própria tabela
  - estratégias particulares:
    - diferem na escolha da posição ainda não ocupada para armazenar um elemento que colide

Tratamento de Colisão

- Uso da posição consecutiva livre:
  - Estratégia 1:
    - se a função de dispersão mapeia a chave de busca
      para um índice já ocupado, procure o próximo índice livre da tabela
      (usando incremento circular) para armazenar o novo elemento



Tratamento de Colisão

- Operação de busca
  - suponha que uma chave x for mapeada pela função de hash h para um determinado índice h(x)
  - procure a ocorrência do elemento a partir de h(x), até que o elemento seja encontrado ou que uma posição vazia seja encontrada
  - entrada: a tabela e a chave de busca
  - saída: o ponteiro do elemento, se encontrado
    NULL, se o elemento não for encontrado

```
public Aluno busca(int mat)

int h ← hash(mat);
enquanto (tabela[h] ≠ null) faça
    se (tabela[h].matricula == mat) então
        retorna tabela[h];
    h ← (h + 1)%tabela.length;
retorna null;
```

Tratamento de Colisão

- Operação de inserção e modificação:
  - suponha que uma chave x for mapeada pela função de hash h para um determinado índice h(x)
  - procure a ocorrência do elemento a partir de h(x), até que o elemento seja encontrado ou que uma posição vazia seja encontrada
  - se o elemento existir, modifique o seu conteúdo
  - se não existir, insira um novo na primeira posição livre que encontrar na tabela, a partir do índice mapeado

```
public void insere(int mat, String nome, int media)

int h ← hash(mat);

// procura aluno
enquanto (tabela[h] ≠ null) faça
    se (tabela[h].matricula == mat) então
        break;
    h ← (h + 1)%tabela.length;

// não encontrou aluno, então cria novo
se (tabela[h] == null) então
    tabela[h] ← newAluno();
    tabela[h].matricula ← mat;

// atribui/modifica informação
tabela[h].nome ← nome;
tabela[h].mediageral ← media;
```

Tratamento de Colisão

- Uso de uma segunda função de dispersão:
  - exemplo:
    - primeira função de hash: $h(x) = x\%N$
    - segunda função de hash: $h'(x) = N - 2 - x\%(N-2)$

    onde *x* representa a chave de busca e *N* a dimensão da tabela

  - se houver colisão, procure uma posição livre na tabela com incrementos dados por *h'(x)*
    - em lugar de tentar *(h(x)+1)%N*, tente *(h(x)+h'(x))%N*

Tratamento de Colisão

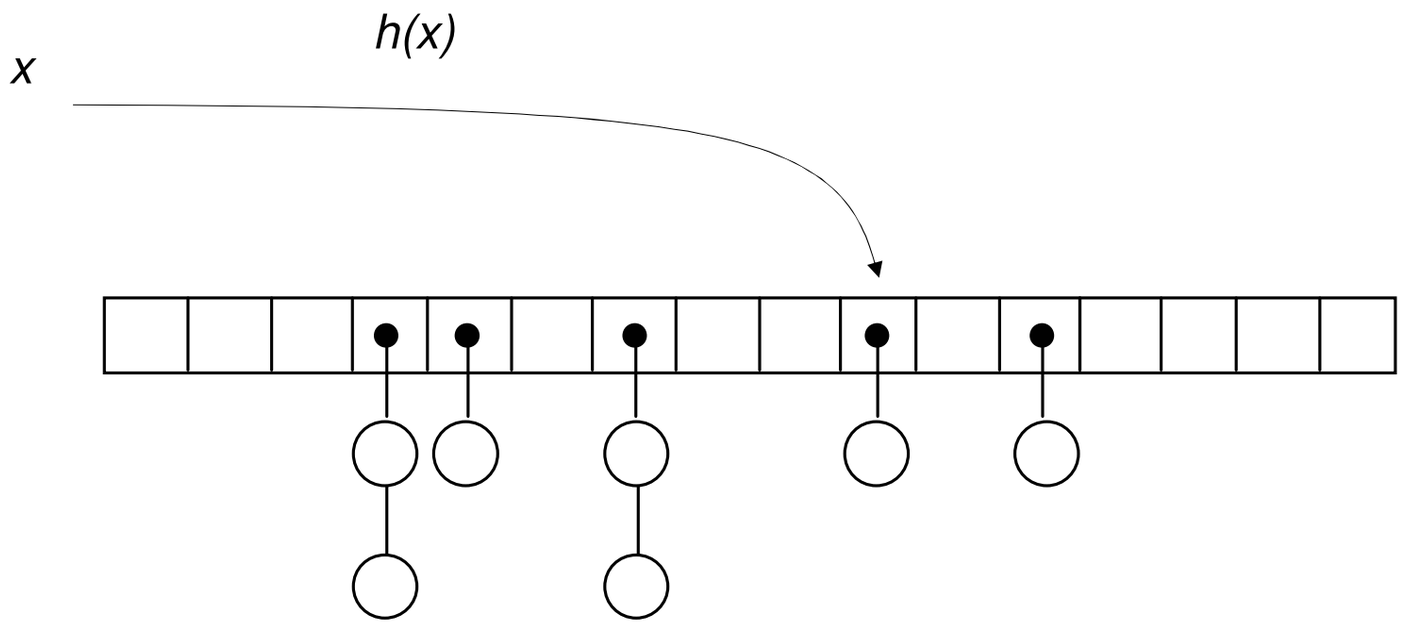
- Uso de uma segunda função de dispersão (cont):
  - cuidados na escolha da segunda função de dispersão:
    - nunca pode retornar zero
      - pois isso não faria com que o índice fosse incrementado
    - não deve retornar um número divisor da dimensão da tabela
      - pois isso limitaria a procura de uma posição livre a um sub-conjunto restrito dos índices da tabela
      - se a dimensão da tabela for um número primo, garante-se automaticamente que o resultado da função não será um divisor

```
private int hash2(int mat)

int N ← tabela.length;
retorna N − 2 − mat%(N − 2);
```

```
public Aluno busca(int mat)

int h ← hash(mat);
int h2 ← hash2(mat);
enquanto (tabela[h] ≠ null) faça
    se (tabela[h].matricula == mat) então
        retorna tabela[h];
    h ← (h + h2)%tabela.length;
retorna null;
```

Tratamento de Colisão

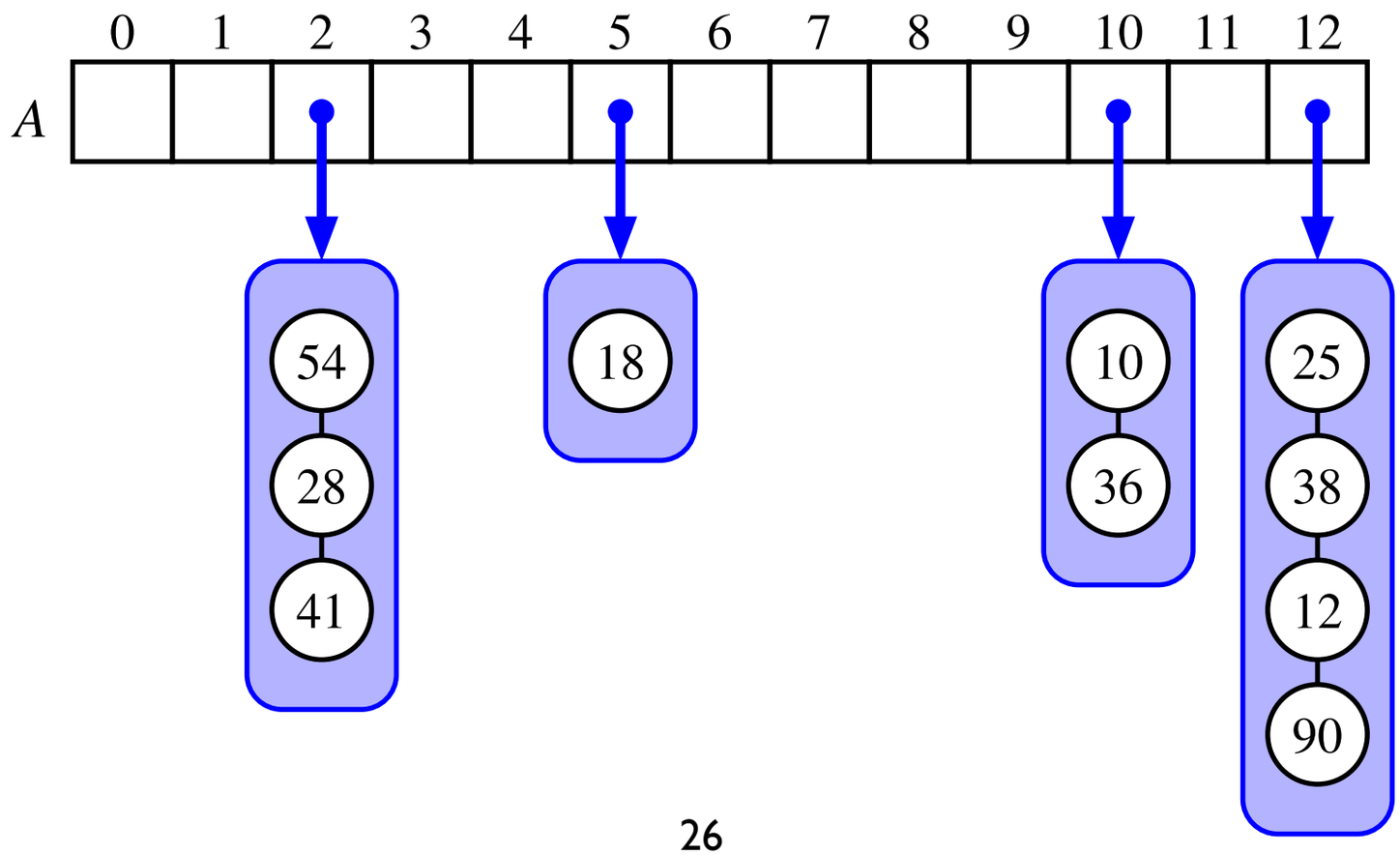
- Uso de listas encadeadas
  - cada elemento da tabela hash representa um ponteiro para uma lista encadeada
    - todos os elementos mapeados para um mesmo índice são armazenados na lista encadeada
    - os índices da tabela que não têm elementos associados representam listas vazias



INSTITUTO FEDERAL Catarinense Campus Blumenau

Tratamento de colisão

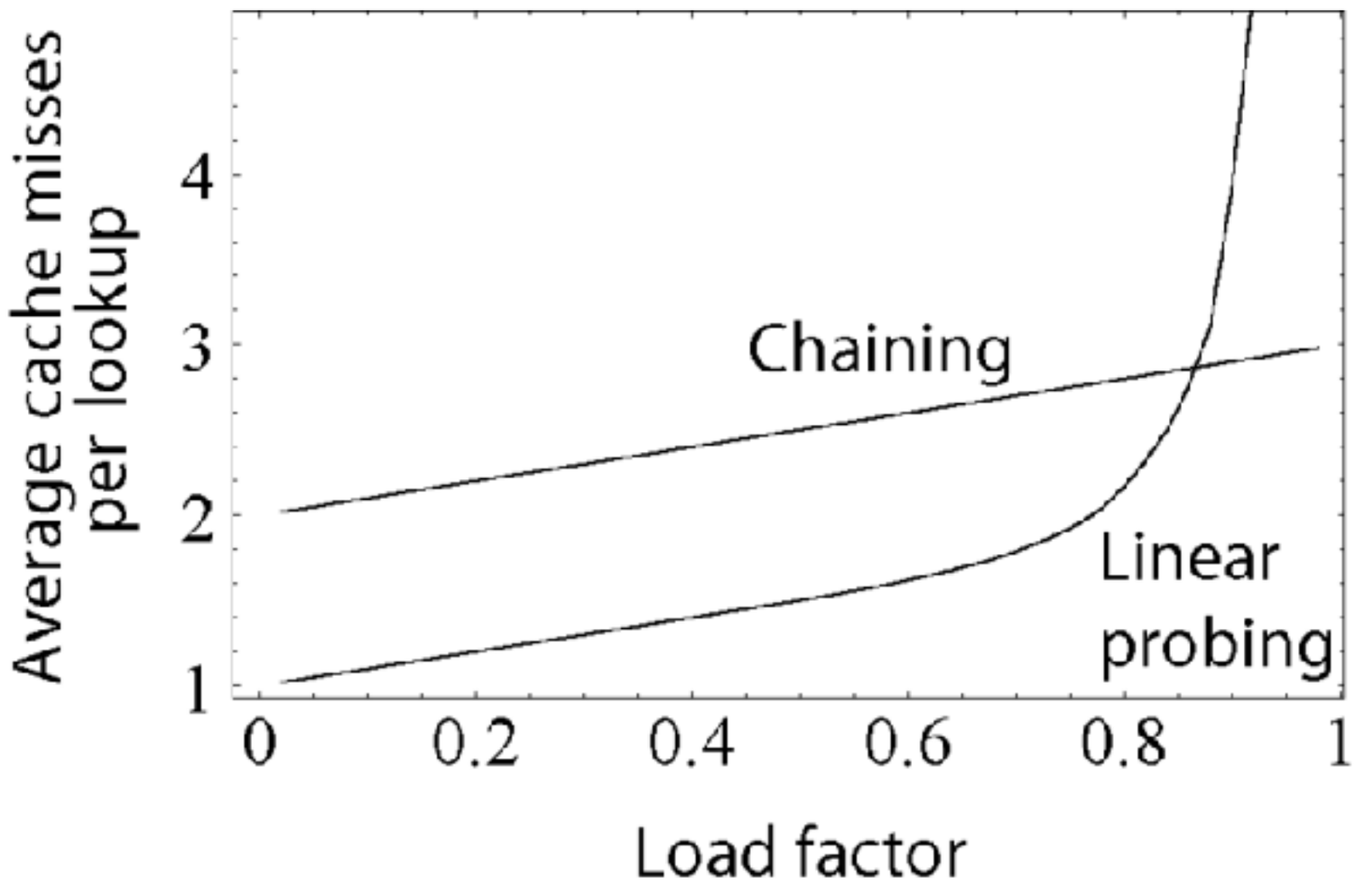
- Uso de listas encadeadas - exemplo: tabela de tamanho 13, com *h(k) = k % 13*
- para simplificar, não estão sendo mostrados os objetos associados às chaves



INSTITUTO FEDERAL Catarinense Campus Blumenau

Tratamento de Colisão

- Comparação entre os métodos de posição consecutiva livre (“Linear probing") e lista encadeada (“Chaining")



Tratamento de Colisão

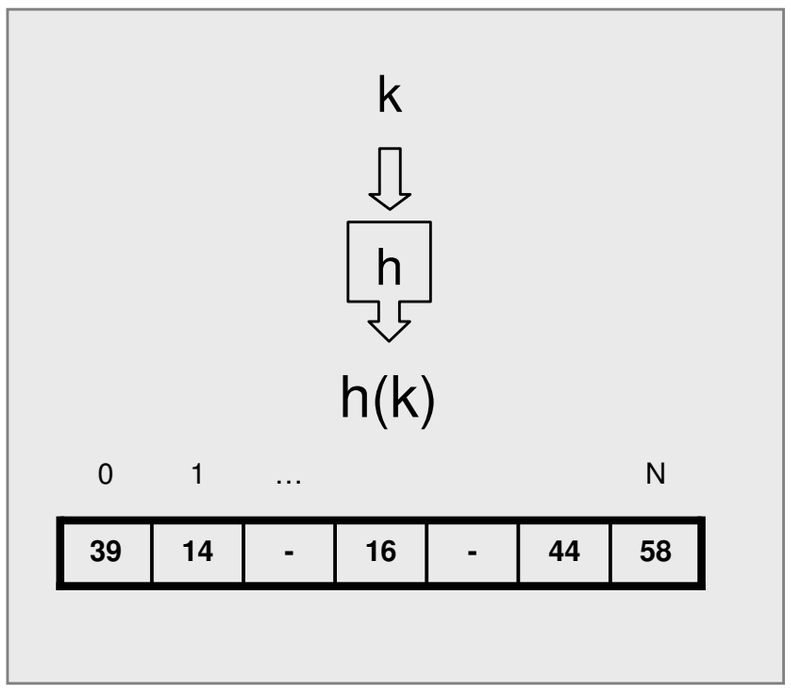
- Exemplo:
  - cada elemento armazenado na tabela será um elemento de uma lista encadeada
  - a estrutura da informação deve prever um ponteiro adicional para o próximo elemento da lista

Aluno
- nome : String - matricula : int - mediageral : float **- prox : Aluno**
+Aluno(nome:String, matr:int, media:float) +toString() ... demais métodos...

```
public void insere(int mat, String nome, int media)

int h ← hash(mat);
Aluno p ← tabela[h];

// procura aluno
enquanto p ≠ null) faça
    se (p.matricula == mat) então
        break;
    p ← p.prox;

// não encontrou aluno, então cria novo
se (p == null) então
    p ← newAluno();
    p.matricula ← mat;
    p.prox ← tabela[h];
    tabela[h] ← p;

// atribui/modifica informação
p.nome ← nome;
p.mediageral ← media;
```

Resumo

- Tabelas de dispersão (hash tables):
  - utilizadas para buscar um elemento em ordem constante O(1)
- Função de dispersão (função de hash):
  - mapeia uma chave de busca em um índice da tabela
- Estratégias para tratamento de colisão:
  - uso da primeira posição consecutiva livre
  - uso de uma segunda função de dispersão
  - uso de listas encadeadas



INSTITUTO FEDERAL Catarinense Campus Blumenau