ACH2024

Aula 21

HASHING

Hashing estático – endereçamento fechado com encadeamento interno

Profa. Ariane Machado Lima

Aulas anteriores

- Organização interna de arquivos
- Acesso à memória secundária (por blocos - seeks)
- Tipos de alocação de arquivos na memória secundária:
  - Sequencial (ordenado e não ordenado)
  - Ligada
  - Indexada
  - Árvores-B
  - Hashing (veremos também hashing em memória principal)
- Algoritmos de processamento cossequencial e ordenação em disco

Motivação e Conceitos Básicos

Agora você quer armazenar 6 chaves contendo valores {0, 7, 15, 367, 4067, 50876}

- Que estrutura de dados usaria? Onde armazenaria cada chave

IDEIA:

1) utilizar um vetor (tabela) de tamanho m
2) aplicar uma função que mapeie cada chave a um número de 0 a m-1

Ex: m = 10 e pegar o primeiro dígito (ou letra)

Hashing: picar/dividir o conjunto em **slots**

Tabela de armazenamento: **tabela de hash**

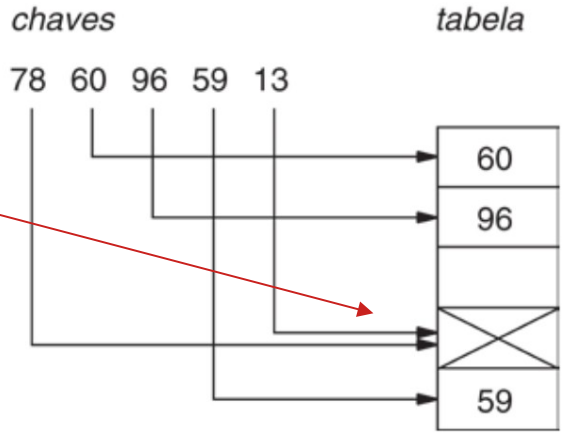
Função de mapeamento: **função de hash**

Endereço calculado: **endereço-base**

Motivação e Conceitos Básicos

Questões que podem surgir:

- O que fazer quando duas chaves caem na mesma posição? (colisão)
  - Tratamento de colisões
- Qual função de hash utilizar? Como ela impacta na ocorrência de colisões?



Vamos estudar essas questões

Funções de hash

Definição: considerando:

uma tabela de tamanho m (m slots)

Um domínio C de valores de chaves (strings, $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, ...)

um função de hash é uma função h: C $\rightarrow$ {0, 1, …, m-1}

Ou seja, se x $\in$ C é uma chave, h(x) retorna o endereço-base de x (ou seja, seu índice na tabela de hash)

Ex: h(x) = x (primeiro exemplo)

h(x) = dígito mais significativo (segundo exemplo)

Funções de hash

Propriedades desejáveis:

1) Poucas colisões

2) Ser rapidamente calculada (O(1), senão estraga vantagem do hashing)

3) Distribuição uniforme:

- idealmente se há m slots, P(h(x)) = 1/m ∀x

  (a probabilidade de qualquer endereço-base deve ser 1/m)

- importante para minimizar colisões (de pior caso)

- difícil de ser testada, mas bom senso pode ajudar. Ex: dígito mais significativo seria uma boa? – péssima idea na maioria dos casos

Funções de hash

Principais métodos de funções de hash:

1) Método da divisão
2) Método da dobra
   - baseado em soma
   - baseado em ou-exclusivo
3) Método da multiplicação
4) Método da análise de dígitos

Tratamento de colisões

Tratamento de colisões

Colisão: quando $x \neq y$ mas $h(x) = h(y)$

Fator de carga: $\alpha = n/m$ (m = nr de slots da tabela de hash, n = nr de chaves a serem inseridas)

Maior $\alpha \rightarrow$ maior o nr de colisões

Mas $\alpha < 1$ não garante ausência de colisões… $\rightarrow$ tem que tratar

Tratamento de colisões

Estratégias:

A) Hashing estático (tamanho da tabela é constante)

1) Encadeamento ou endereçamento fechado – colisões vão para uma lista ligada

1.1) Encadeamento exterior (fora da tabela)

1.2) Encadeamento interior (dentro da tabela)

2) Endereçamento aberto (chaves dentro da tabela, sem ponteiros)

2.1) Tentativa/Sondagem linear

2.2) Tentativa/Sondagem quadrática

2.3) Dispersão dupla / Hash duplo

B) Hashing dinâmico (tabela pode expandir/encolher)

3) Hashing extensível (estrutura de dados adicional)

4) Hashing linear

Tudo isso para hashing interno (em memória) quanto para externo (em disco).

Primeiro assumiremos hashing interno e depois discutiremos mudanças para hashing externo.

Tratamento de colisões - 1) Endereçamento fechado
1.1) Encadeamento exterior (fora da tabela de hash)

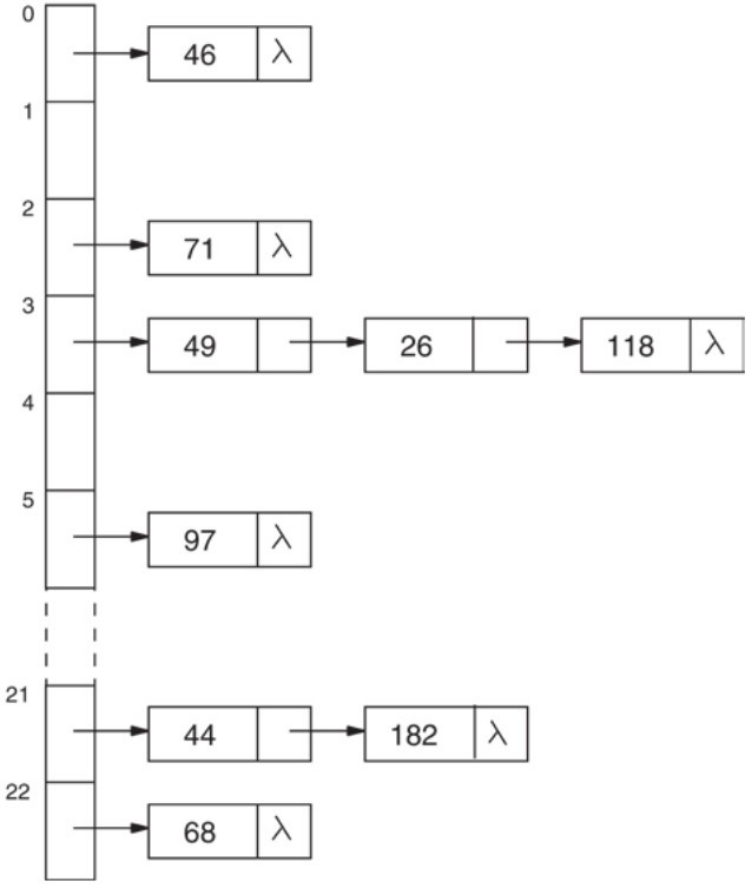
Complexidade de busca/inserção/remoção:

- pior caso: O(n)
- caso médio (assumindo hash uniforme):
  - sem sucesso: α
  - com sucesso: 1 + α/2 - 1/2m

Isto é, mais rápido conforme:

- α menor e m maior - linearmente

Note que a “tabela” pode crescer indefinidamente



Aula de hoje
Tratamento de colisões

Estratégias:

A) Hashing estático (tamanho da tabela é constante)

1) Encadeamento ou endereçamento fechado – colisões vão para uma lista ligada

1.1) Encadeamento exterior (fora da tabela)

1.2) Encadeamento interior (dentro da tabela)

2) Endereçamento aberto (chaves dentro da tabela, sem ponteiros)

2.1) Tentativa/Sondagem linear

2.2) Tentativa/Sondagem quadrática

2.3) Dispersão dupla / Hash duplo

B) Hashing dinâmico (tabela pode expandir/encolher)

3) Hashing extensível (estrutura de dados adicional)

4) Hashing linear

Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior (dentro da tabela de hash)

Tabela com tamanho máximo
(para aplicações em que o tamanho não pode variar)

Listas ligadas ficam dentro da tabela:

T[i] = (chave, slot do próximo)

Exige $\alpha = m/m \leq 1$

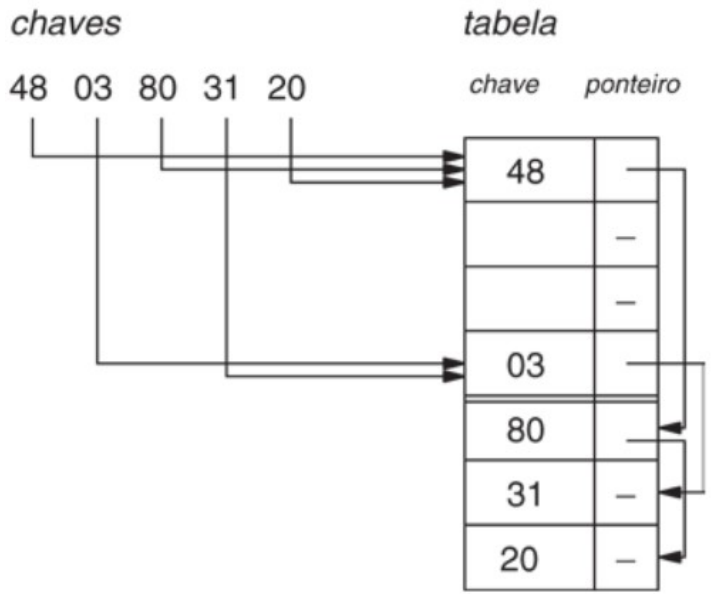
Opção 1 (zona de colisões) Tabela T dividida em duas áreas (m=p+s)

- p slots para endereços-base
- s slots para **sinônimos (zona de colisões)**
- h(x) → [0, p-1]
- ponteiros sempre apontam para um valor em [p, m-1]

Problema: zona de colisão pode ficar lotada mesmo havendo espaço na área de endereços-base (overflow)

Possibilidade: aumentar s e diminuir p (mas diminui poder de espalhamento

- se p = m-1 e s = 1 → overflow muito cedo
- no limite p = 1 e s = m-1 → O(n) para buscas



Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior (dentro da tabela de hash)

Listas ligadas ficam dentro da tabela:

T[i] = (chave, slot do próximo)

Exige $\alpha = n/m \leq 1$

Opção 2: Tabela T mistura endereços-base e sinônimos

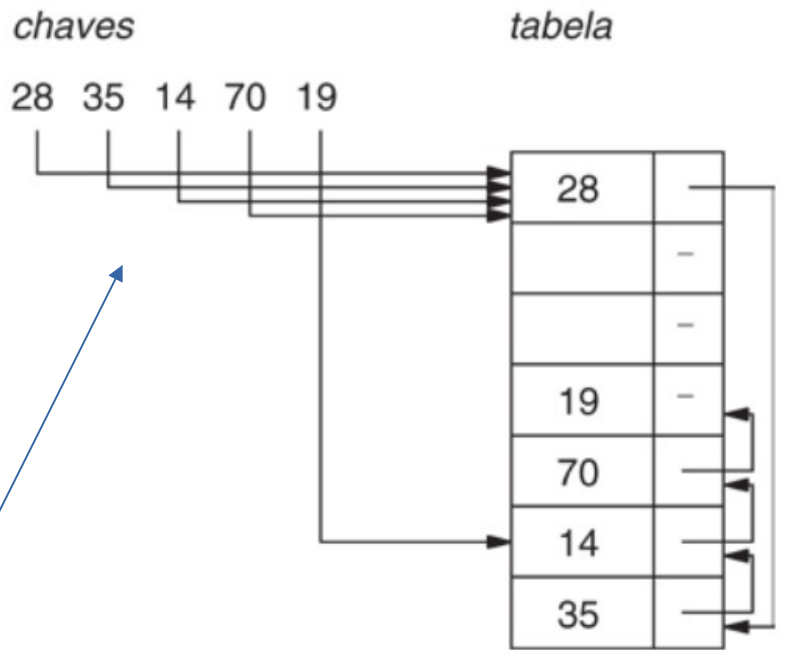
- h(x) → [0, m-1]
- se h(x) estiver vago armazena chave x lá

  Senão armazena x na próxima posição livre (a partir de h(x) ou a partir do fim da tabela)

Problema: colisões secundárias: h(y) já está ocupada por uma chave x em que h(x) ≠ h(y) (Ex: y = 19, x = 14)

→ fusão das listas encadeadas de h(x) e h(y)

→ diminuição da eficiência

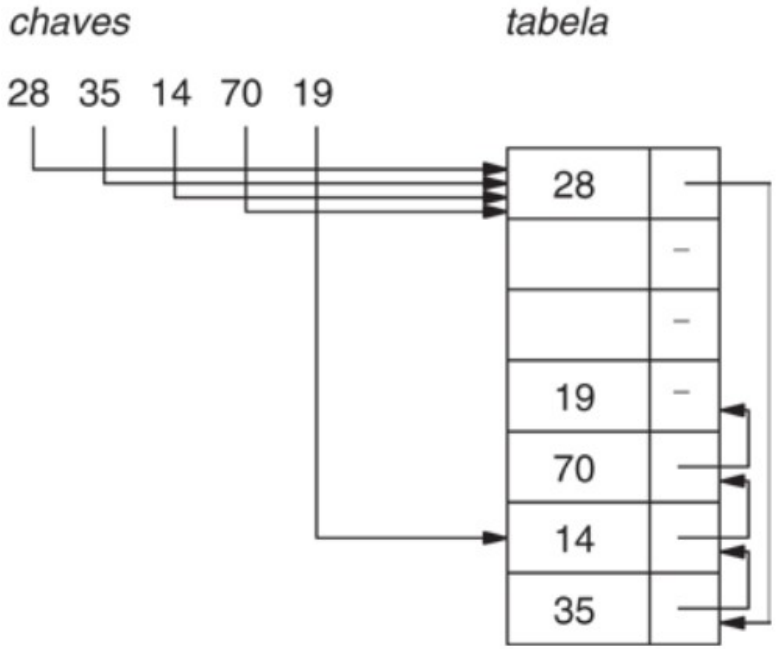


Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Operações:

Inserção e busca seriam tranquilas, mas e a remoção? (ex: remove 14)

- Não podemos abrir buracos no encadeamento…
- Reorganizar a tabela a cada remoção é normalmente inviável...



Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

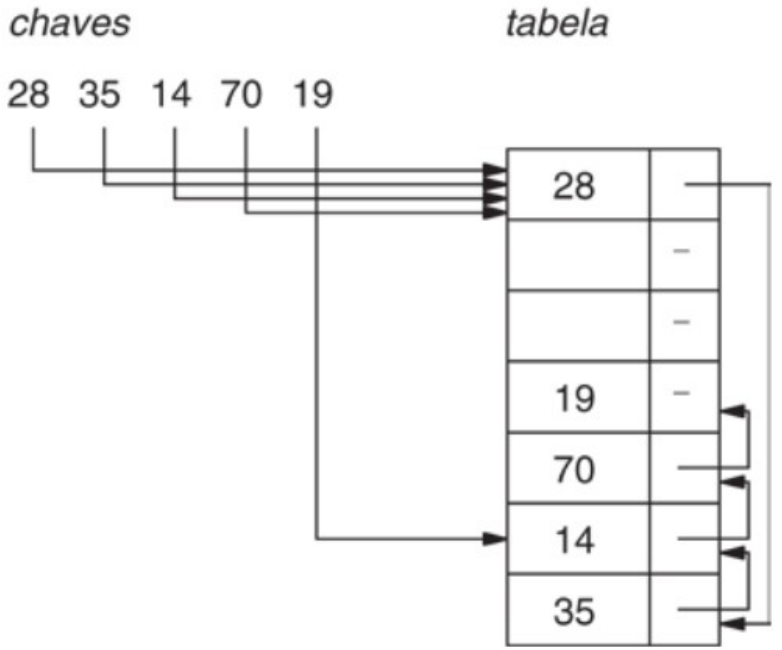
Operações:

Inserção e busca seriam tranquilas, mas e a remoção? (ex: remove 14)

- Não podemos abrir buracos no encadeamento…
- Reorganizar a tabela a cada remoção é normalmente inviável...

Solução:

- cada slot pode ter 1 campo adicional:
  - ocupado: tem chave
  - não ocupado: vazio (sem chave) ou liberado (tinha chave mas não tem mais - pode estar no meio de um encadeamento)



Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Operações:

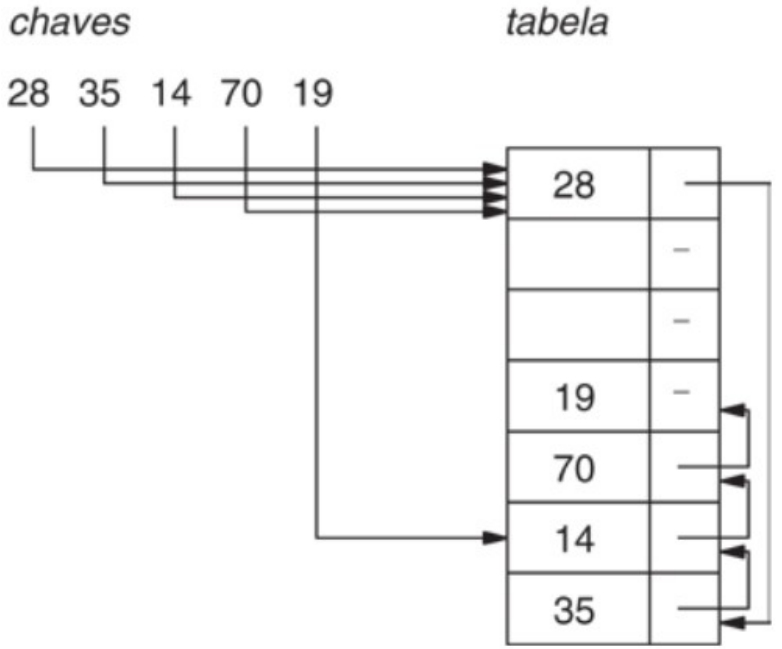
Inicialização: para i = 0 a m-1

T[i].estado = não ocupado

T[i].prox = -1

Remoção em T[i] deverá fazer

T[i].estado = não ocupado

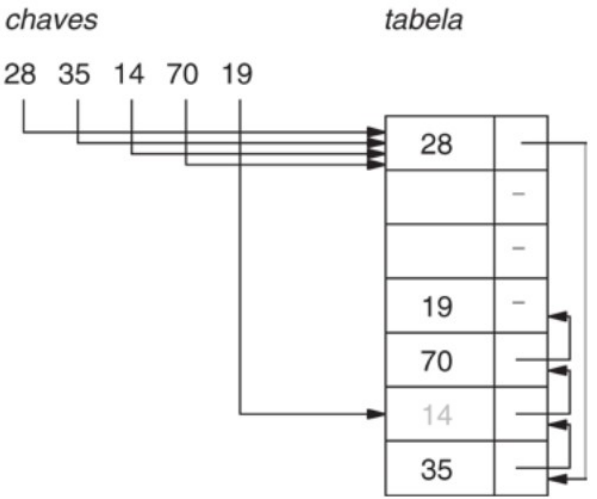


Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Busca (T, x, end, a): **a** indica presença/ausência da chave; **end** auxiliará a inserção

/* se a = 1 então chave encontrada no slot end ; se a = 2 a chave não está na tabela e end tem 2 possibilidades:
1) end = -1 => lista h(x) está vazia ou s/ buracos; 2) end = j ≥ 0 => slot j é da lista h(x) e está desocupado */

```
a ← 0; end ← h(x); j ← -1;   /* a = 0 significa que ainda não sei */
enquanto a = 0
   se T[end].estado = não ocupado
        j ←end
   se T[end].estado = ocupado   e   T[end].chave = x
        a ← 1   /* chave encontrada */
   senão
        end ← T[end].pont
        se end = -1     /* acabou a lista */
              a←2; end ← j      /* chave não encontrada */
```



Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Busca (T, x, end, a): **a** indica presença/ausência da chave; **end** auxiliará a inserção

/* se a = 1 então chave encontrada no slot end ; se a = 2 a chave não está na tabela e end tem 2 possibilidades:

1) end = -1 => lista h(x) está vazia ou s/ buracos; 2) end = j ≥ 0 => slot j é da lista h(x) e está desocupado */

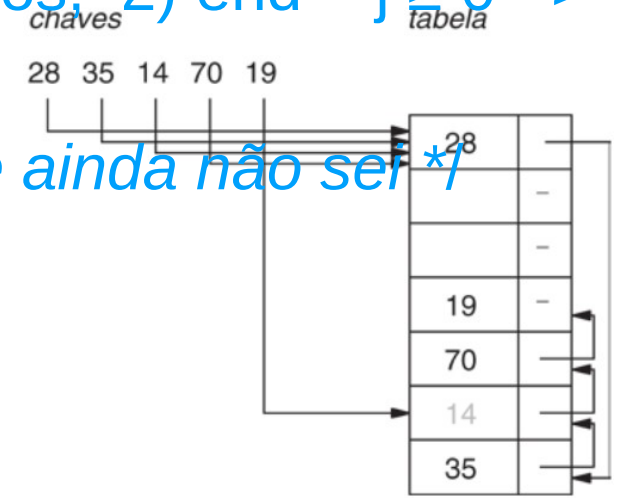
Ex: busca(T, 20, end, a):
a = 0
end = 0 (h(x) = 0)
j = -1
T[end].estado = ocupado
T[end].chave = 28

a ← 0; end ← h(x); j ← -1; /* a = 0 significa que ainda não sei */

enquanto a = 0

se T[end].estado = não ocupado

j ← end



Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Busca (T, x, end, a): **a** indica presença/ausência da chave; **end** auxiliará a inserção

/* se a = 1 então chave encontrada no slot end ; se a = 2 a chave não está na tabela e end tem 2 possibilidades:

1) end = -1 => lista h(x) está vazia ou s/ buracos; 2) end = j ≥ 0 => slot j é da lista h(x) e está desocupado */

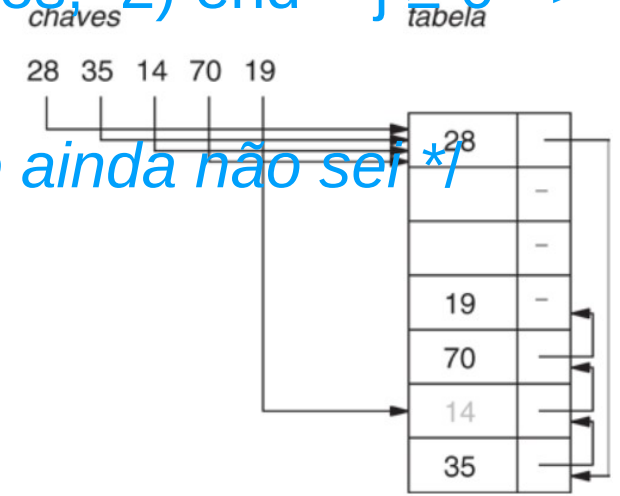
Ex: busca(T, 20, end, a):
a = 0
end = 6 (h(x) = 0)
j = -1
T[end].estado = ocupado
T[end].chave = 35

a ← 0; end ← h(x); j ← -1; /* *a = 0 significa que ainda não sei* */

enquanto a = 0

se T[end].estado = não ocupado

j ← end



Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Busca (T, x, end, a): **a** indica presença/ausência da chave; **end** auxiliará a inserção

/* se a = 1 então chave encontrada no slot end ; se a = 2 a chave não está na tabela e end tem 2 possibilidades:

1) end = -1 => lista h(x) está vazia ou s/ buracos; 2) end = j ≥ 0 => slot j é da lista h(x) e está desocupado */

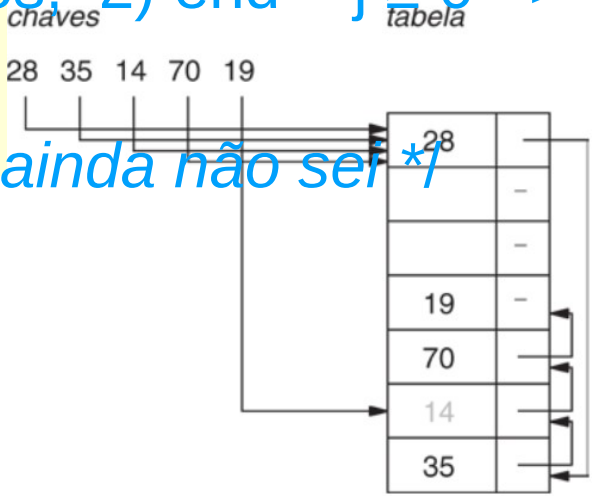
Ex: busca(T, 20, end, a):
a = 0
end = 5 (h(x) = 0)
j = 5
T[end].estado = não ocupado
T[end].chave = 14

a ← 0; end ← h(x); j ← -1; /* *a = 0 significa que ainda não sei* */

enquanto a = 0

se T[end].estado = não ocupado

j ← end



Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Busca (T, x, end, a): **a** indica presença/ausência da chave; **end** auxiliará a inserção

/* se a = 1 então chave encontrada no slot end ; se a = 2 a chave não está na tabela e end tem 2 possibilidades:

1) end = -1 => lista h(x) está vazia ou s/ buracos; 2) end = j ≥ 0 => slot j é da lista h(x) e está desocupado */

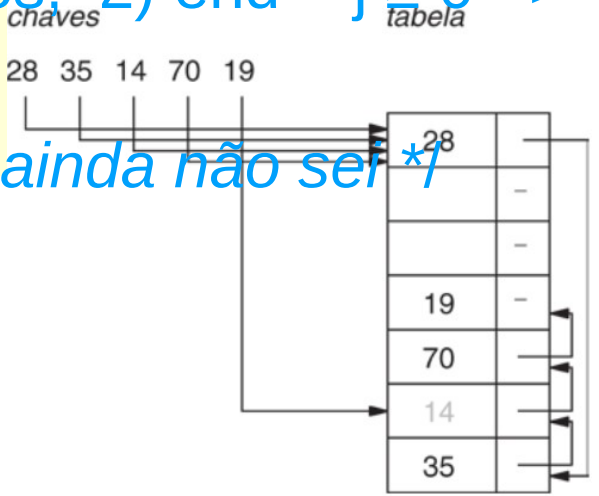
Ex: busca(T, 20, end, a):
a = 0
end = 4 (h(x) = 0)
j = 5
T[end].estado = ocupado
T[end].chave = 70

a ← 0; end ← h(x); j ← -1; /* *a = 0 significa que ainda não sei* */

enquanto a = 0

se T[end].estado = não ocupado

j ← end



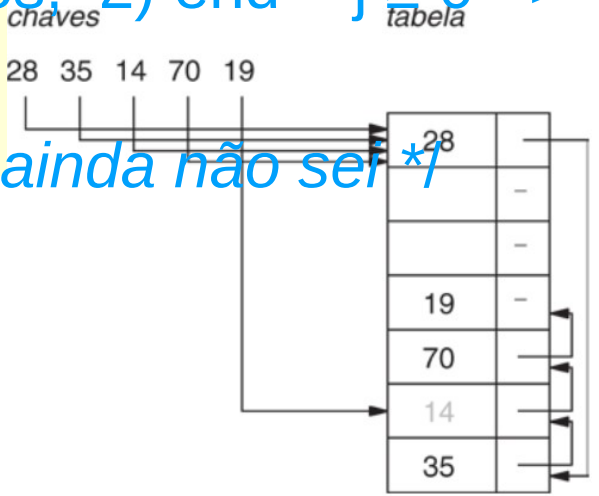
Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Busca (T, x, end, a): **a** indica presença/ausência da chave; **end** auxiliará a inserção

/* se a = 1 então chave encontrada no slot end ; se a = 2 a chave não está na tabela e end tem 2 possibilidades:

1) end = -1 => lista h(x) está vazia ou s/ buracos; 2) end = j ≥ 0 => slot j é da lista h(x) e está desocupado */

Ex: busca(T, 20, end, a):
a = 0
end = 3 (h(x) = 0)
j = 5
T[end].estado = ocupado
T[end].chave = 19



a ← 0; end ← h(x); j ← -1; /* *a = 0 significa que ainda não sei* */

enquanto a = 0

se T[end].estado = não ocupado

j ← end

Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Busca (T, x, end, a): **a** indica presença/ausência da chave; **end** auxiliará a inserção

/* se a = 1 então chave encontrada no slot end ; se a = 2 a chave não está na tabela e end tem 2 possibilidades:

1) end = -1 => lista h(x) está vazia ou s/ buracos; 2) end = j ≥ 0 => slot j é da lista h(x) e está desocupado */

Ex: busca(T, 20, end, a):
a = 0
end = -1 (h(x) = 0)
j = 5
T[end].estado =
T[end].chave =

a ← 0; end ← h(x); j ← -1; */* a = 0 significa que ainda não sei */*

enquanto a = 0

se T[end].estado = não ocupado

j ← end

Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Busca (T, x, end, a): **a** indica presença/ausência da chave; **end** auxiliará a inserção

/* se a = 1 então chave encontrada no slot end ; se a = 2 a chave não está na tabela e end tem 2 possibilidades:

1) end = -1 => lista h(x) está vazia ou s/ buracos; 2) end = j ≥ 0 => slot j é da lista h(x) e está desocupado */

Ex: busca(T, 20, end, a):
a = 2
end = 5 (h(x) = 0)
j = 5
T[end].estado = não ocupado
T[end].chave =

a ← 0; end ← h(x); j ← -1; /* a = 0 significa que ainda não sei */

enquanto a = 0

se T[end].estado = não ocupado

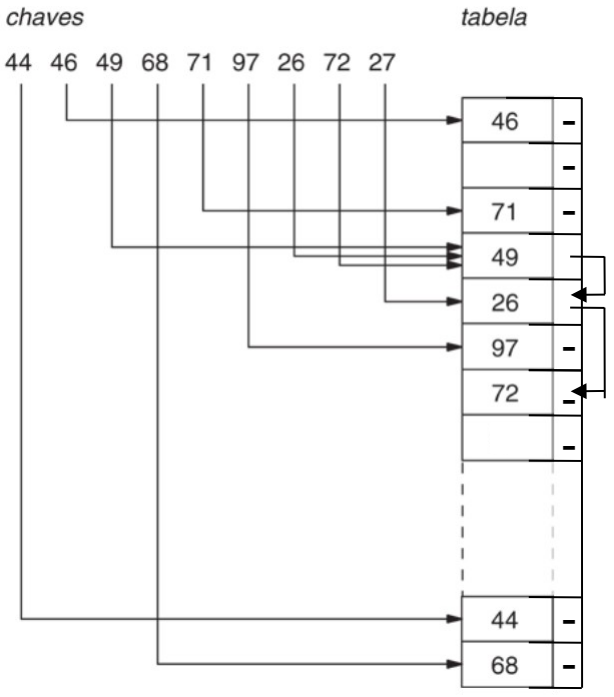
j ← end

Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Insere(T, x): **a** indica presença/ausência da chave; **end** auxiliará a inserção. Vou achar posição **j** para inserir

```
Busca(T, x, end, a)
Se a ≠ 1  /* ou seja, x não está na tabela */
  Se end ≠ -1  /* já sei onde vou inserir, no meio de uma lista */
    j ← end
  Senão  /* se end = -1 não sei ainda se a lista está vazia ou completa (sem slots desocupados) */
    i ← 1; j ← h(x); /* i vai controlar até quando procurar um slot  (até no máximo olhar todos os m slots) */
    Enquanto i ≤ m
      se T[j].estado = ocupado
        j ← (j + 1) mod m  /* procuro na próxima posição; quando chegar no fim da tabela volto para o início */
        i ← i + 1
      Senão i ← m + 2 /* para sair do loop, equivalente a um break ou last */
    Se i = m + 1
      PARE / OVERFLOW – TABELA CHEIA
    temp ← T[h(x)].pont   /* aqui j = h(x) (lista esatava vazia) ou j = uma posição vazia qualquer (de qualquer lista
    T[h(x)].pont ← j   /* vou inserir depois do primeiro elemento da lista */
    T[j].pont ← temp
  T[j].chave ← x
  T[j].estado ← ocupado
Senão PARE / CHAVE JÁ EXISTENTE
```

Ex: insere 27
end = -1
T[end] nunca foi ocupado



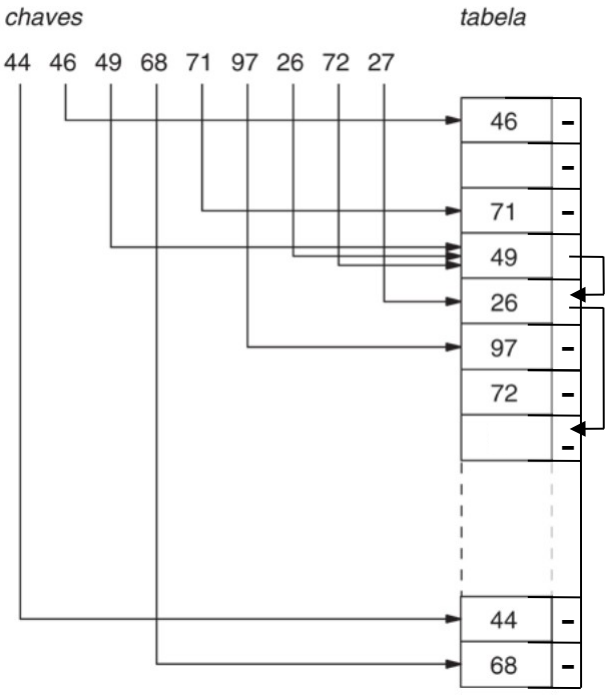
temp = 6

Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Insere(T, x): **a** indica presença/ausência da chave; **end** auxiliará a inserção. Vou achar posição **j** para inserir

```
Busca(T, x, end, a)
Se a ≠ 1  /* ou seja, x não está na tabela */
  Se end ≠ -1  /* já sei onde vou inserir, no meio de uma lista */
    j ← end
  Senão  /* se end = -1 não sei ainda se a lista está vazia ou completa (sem slots desocupados) */
    i ← 1; j ← h(x); /* i vai controlar até quando procurar um slot  (até no máximo olhar todos os m slots) */
    Enquanto i ≤ m
      se T[j].estado = ocupado
        j ← (j + 1) mod m  /* procuro na próxima posição; quando chegar no fim da tabela volto para o início */
        i ← i + 1
      Senão i ← m + 2 /* para sair do loop, equivalente a um break ou last */
    Se i = m + 1
      PARE / OVERFLOW – TABELA CHEIA
    temp ← T[h(x)].pont   /* aqui j = h(x) (lista esatava vazia) ou j = uma posição vazia qualquer (de qualquer lista
    T[h(x)].pont ← j   /* vou inserir depois do primeiro elemento da lista */
    T[j].pont ← temp
  T[j].chave ← x
  T[j].estado ← ocupado
Senão PARE / CHAVE JÁ EXISTENTE
```

Ex: insere 27
end = -1
T[end] nunca foi ocupado



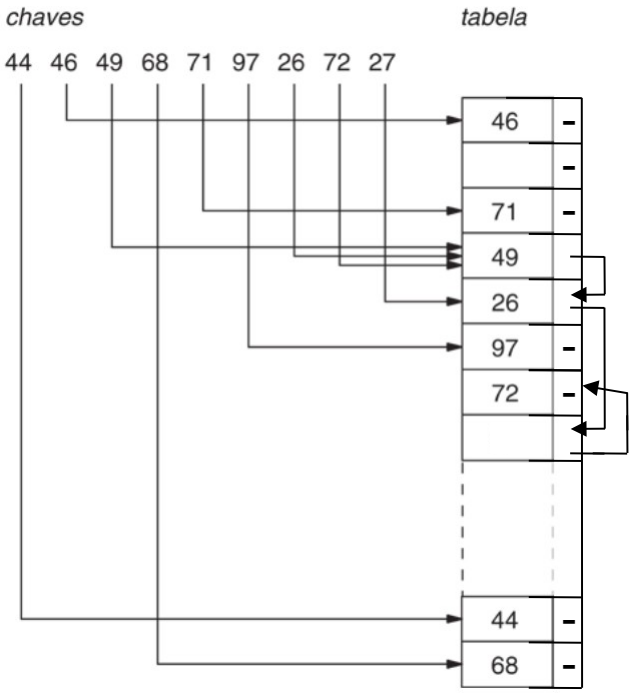
temp = 6
T[4].point = j = 7

Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Insere(T, x): **a** indica presença/ausência da chave; **end** auxiliará a inserção. Vou achar posição **j** para inserir

```
Busca(T, x, end, a)
Se a ≠ 1  /* ou seja, x não está na tabela */
  Se end ≠ -1  /* já sei onde vou inserir, no meio de uma lista */
    j ← end
  Senão  /* se end = -1 não sei ainda se a lista está vazia ou completa (sem slots desocupados) */
    i ← 1; j ← h(x); /* i vai controlar até quando procurar um slot  (até no máximo olhar todos os m slots) */
    Enquanto i ≤ m
      se T[j].estado = ocupado
        j ← (j + 1) mod m  /* procuro na próxima posição; quando chegar no fim da tabela volto para o início */
        i ← i + 1
      Senão i ← m + 2 /* para sair do loop, equivalente a um break ou last */
    Se i = m + 1
      PARE / OVERFLOW – TABELA CHEIA
    temp ← T[h(x)].pont   /* aqui j = h(x) (lista esatava vazia) ou j = uma posição vazia qualquer (de qualquer lista
    T[h(x)].pont ← j   /* vou inserir depois do primeiro elemento da lista */
    T[j].pont ← temp
  T[j].chave ← x
  T[j].estado ← ocupado
Senão PARE / CHAVE JÁ EXISTENTE
```

Ex: insere 27
end = -1
T[end] nunca foi ocupado



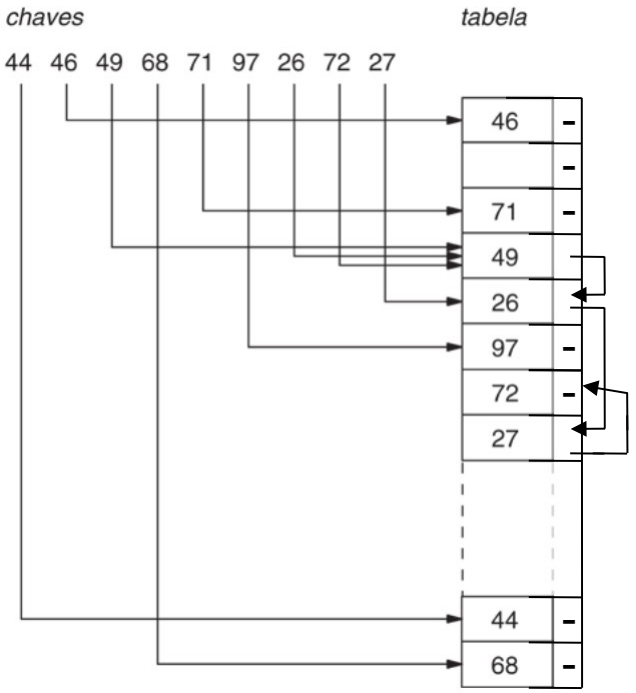
temp = 6
T[4].point = j = 7
T[4].point = 6
T[7].chave = 27
T[7].estado = ocupado

Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Insere(T, x): **a** indica presença/ausência da chave; **end** auxiliará a inserção. Vou achar posição **j** para inserir

```
Busca(T, x, end, a)
Se a ≠ 1  /* ou seja, x não está na tabela */
  Se end ≠ -1  /* já sei onde vou inserir, no meio de uma lista */
    j ← end
  Senão  /* se end = -1 não sei ainda se a lista está vazia ou completa (sem slots desocupados) */
    i ← 1; j ← h(x); /* i vai controlar até quando procurar um slot  (até no máximo olhar todos os m slots) */
    Enquanto i ≤ m
      se T[j].estado = ocupado
        j ← (j + 1) mod m  /* procuro na próxima posição; quando chegar no fim da tabela volto para o início */
        i ← i + 1
      Senão i ← m + 2 /* para sair do loop, equivalente a um break ou last */
    Se i = m + 1
      PARE / OVERFLOW – TABELA CHEIA
    temp ← T[h(x)].pont   /* aqui j = h(x) (lista esatava vazia) ou j = uma posição vazia qualquer (de qualquer lista
    T[h(x)].pont ← j   /* vou inserir depois do primeiro elemento da lista */
    T[j].pont ← temp
  T[j].chave ← x
  T[j].estado ← ocupado
Senão PARE / CHAVE JÁ EXISTENTE
```

Ex: insere 27
end = -1
T[end] nunca foi ocupado



temp = 6
T[4].point = j = 7
T[4].point = 6
T[7].chave = 27
T[7].estado = ocupado

Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Remove(T, x, end, a):

Busca(T, x, end, a) /* **a** indica presença/ausência da chave; **end** indica em que slot está a chave*/

Se a = 1 /* chave existente no slot **end** */

T[end].estado ← não ocupado

Senão

CHAVE NÃO EXISTENTE

Tratamento de colisões - 1) Endereçamento fechado
1.2) Encadeamento interior – opção 2

Complexidades:

Busca / Remove / Insere: O(n) no pior caso

Tratamento de colisões

Estratégias:

A) Hashing estático (tamanho da tabela é constante)

1) Encadeamento ou endereçamento fechado – colisões vão para uma lista ligada

1.1) Encadeamento exterior (fora da tabela)

1.2) Encadeamento interior (dentro da tabela)

2) Endereçamento aberto (chaves dentro da tabela, sem ponteiros)

2.1) Tentativa/Sondagem linear

2.2) Tentativa/Sondagem quadrática

2.3) Dispersão dupla / Hash duplo

B) Hashing dinâmico (tabela pode expandir/encolher)

3) Hashing extensível (estrutura de dados adicional)

4) Hashing linear

Tratamento de colisões
2) Endereçamento aberto

Conceitos gerais

Tratamento de colisões
2) Endereçamento aberto

Características:

- todas as chaves dentro da tabela (**espaço constante**)
- sem uso de ponteiros (não há listas): **economiza espaço**
- endereço de uma mesma chave pode ser diferente dependendo de quando h(x) é calculada (cálculo em **aberto**)
- pode ficar cheia inviabilizando novas inserções (assim como no encadeamento interno)

Tratamento de colisões
2) Endereçamento aberto

Vantagens:

- evita por completo o uso de listas encadeadas;
- ao invés de seguir os ponteiros nas listas, *calculamos* a seqüência de posições a serem examinadas;
- uso mais eficiente do espaço alocado para a tabela hash;
- o espaço não alocado para as listas pode ser usado para aumentar o tamanho da tabela hash, o que implica menor número de colisões.

Referências

Conceitos gerais de Hashing:

SZWARCFITER, J. L.; MARKENZON, L. Estruturas de Dados e Seus Algoritmos. Ed. LTC, 3ª ed, 2013. Capítulo 10 (figuras do livro)

Slides dos Profs. M. Chaim, Delano Beder e L. Digiampietri