Hashing

Rômulo César Silva

Unioeste

Junho de 2016

unioeste
Universidade Estadual do Oeste do Paraná

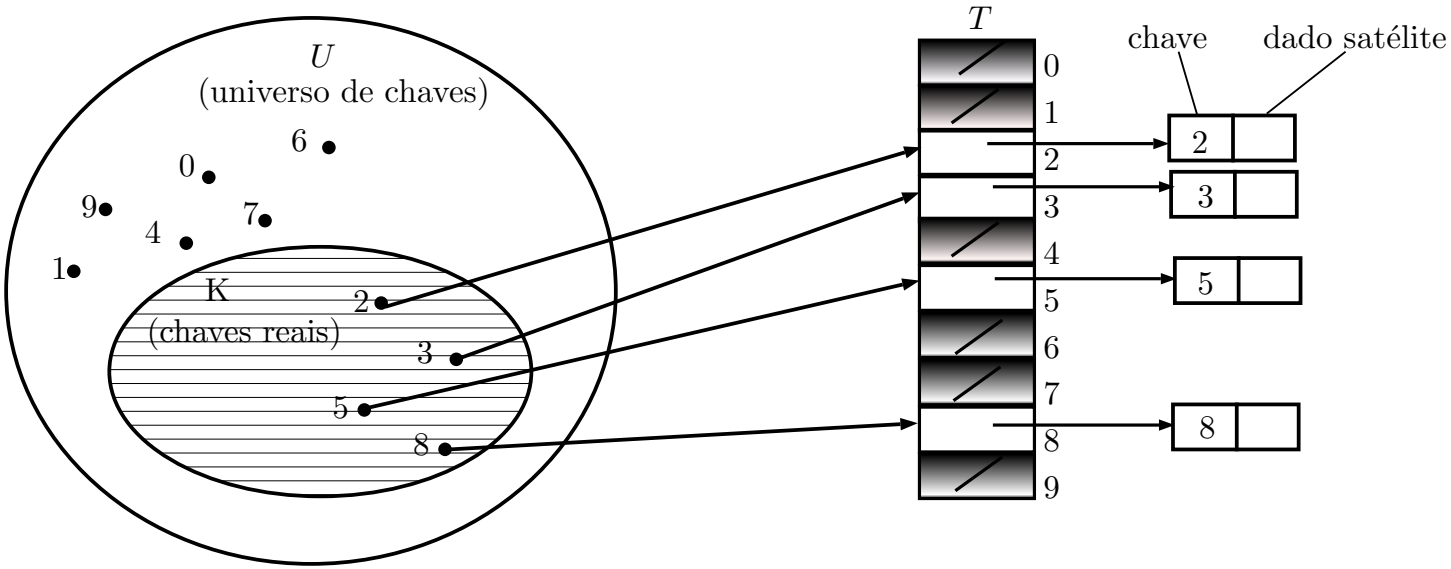
Sumário

Tabelas de Endereçamento Direto

A técnica de **endereçamento direto** para representar um conjunto dinâmico consiste em usar um vetor ou array tal que cada posição ou *slot* corresponde a uma chave do conjunto universo $U = \{0, 1, ..., m-1\}$ de chaves possíveis. Isto é, seja T a tabela de endereçamento direto. $T[k]$ aponta para um elemento no conjunto cuja chave é k. Caso o conjunto não contenha tal elemento, $T[k] = NULL$.

- abordagem interessante para as situações em que m não é muito grande.
- $T[k]$ pode armazenar diretamente o elemento, desde que se tenha alguma maneira de indicar quando a posição está vazia.
- operações de inserção, busca e remoção levam tempo $O(1)$.

Tabela de endereçamento direto - exemplo



Tabelas de Endereçamento Direto

Limitações:

- se o universo U é grande, armazenar uma tabela de tamanho $|U|$ torna-se impraticável
- o conjunto K de chaves realmente armazenadas pode ser tão pequeno em relação a U que a maioria do espaço alocado para T seja desperdiçado.

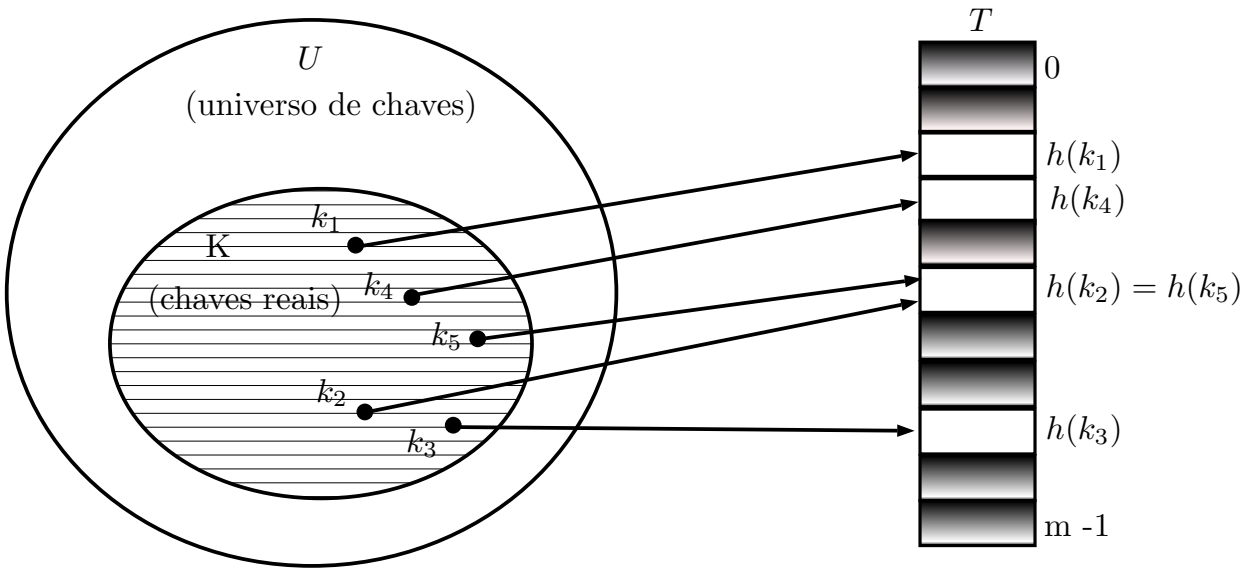
Hash Table

Hash Table

Enquanto com endereçamento direto um elemento de chave k é armazenado na posição k, em uma *hash table* ele é armazenado na posição $h(k)$. Isto é, uma **função hash** é usada para calcular a posição da chave k. A função h mapeia o universo U de chaves em *slots* de uma **tabela hash** $T[0..m-1]$:
$h : U \rightarrow \{0, 1, ..., m-1\}$.

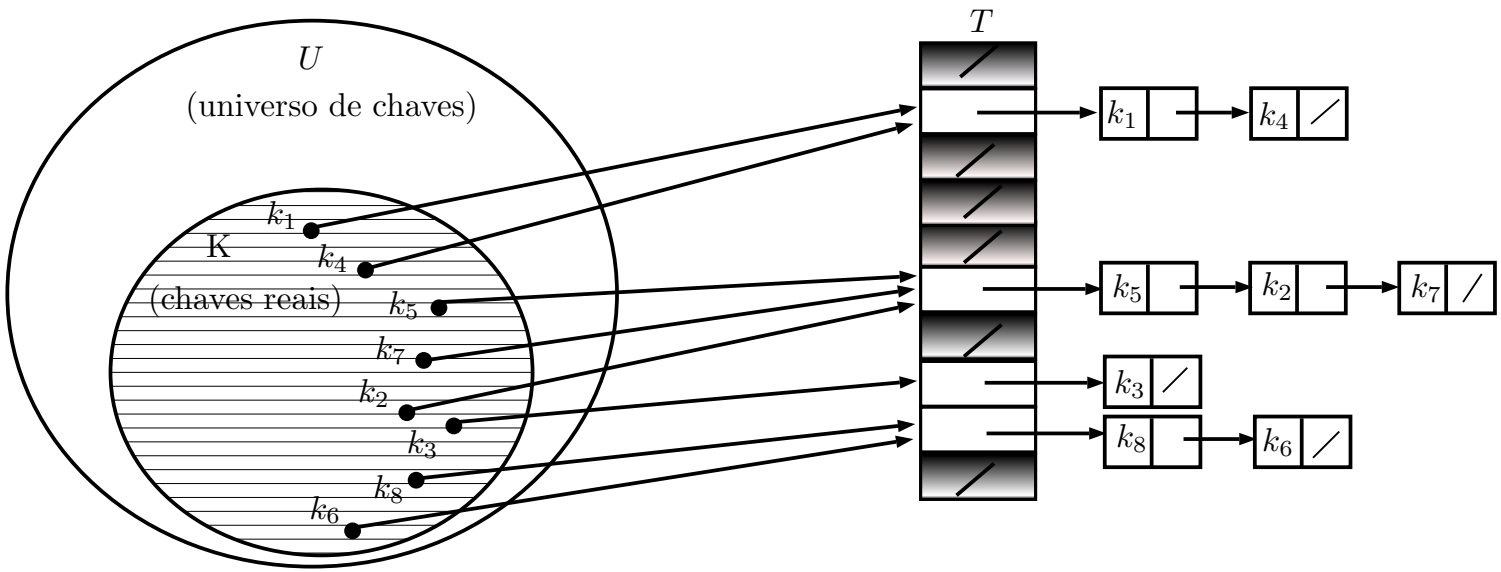
- $h(k)$ é o *valor hash* da chave k
- a função *hash* deve reduzir o intervalo de índices que precisam ser manipulados. Ao invés de $|U|$ valores, precisa-se manipular somente m valores.
- se duas chaves tem o mesmo valor *hash*, há então uma **colisão**

Hash Table - exemplo



Resolução de colisão por encadeamento

Todos os elementos que tem o mesmo valor *hash* são ligados usando lista encadeada.



Resolução de colisão por encadeamento

Fator de carga

Dada uma *hash table* T com m *slots* que armazenam n elementos, o **fator de carga** α para T é definido como sendo n/m, isto é, a média de elementos armazenados em uma lista encadeada.

- pior caso: a função h distribui todas as n chaves no mesmo *slot*, situação em que a complexidade de busca é $O(n)$.
- a performance média depende de como a função h distribui as chaves nos m *slots*.

Funções *hash*

Idealmente uma função *hash*:

- deve produzir um número baixo de colisões
- ser facilmente computável
- ser uniforme: todos os *slots* devem ter a mesma probabilidade de serem escolhidos.

Aplicações de funções *Hash*

- busca de elementos em base de dados: estruturas de dados em memória, bancos de dados e mecanismos de busca na Internet
- verificação de integridade de dados e autenticação de mensagens: envio dos dados junto com o valor do *hash* calculado
- implementação da tabela de símbolos de compiladores
- criptografia: MD5 e família SHA (Secure Hash Algorithm)

Funções *hash*

- A maioria das funções *hash* supõem que o universo de chaves é o conjunto dos números naturais $\mathbb{N} = \{0, 1, 2, ...\}$
- Quando as chaves não são números naturais, precisa ser encontrada uma maneira de interpretá-las como sendo números naturais. Exemplo: para cadeia de caracteres fazer operações usando o código ASCII.
- Há diferentes métodos para se criar funções *hash*:
  - método da divisão
  - método da multiplicação
  - *hashing universal*

Método da Divisão

Mapeia uma chave k em um dos m *slots* através do resto da divisão de k por m. Isto é:

$$h(k) = k \bmod m.$$

Exemplo: se a tabela T tem tamanho $m = 12$ e a chave $k = 100$, então $h(k) = 4$.

- Desde que a operação de divisão é simples, o cálculo é bastante rápido.
- Alguns valores m devem ser evitados:
  - potência de 2: $m = 2^p$, então $h(k)$ é justamente os p bits de mais baixa ordem de k.
  - potência de 10: se a aplicação usa números decimais como chaves, desde que a função não depende de todos os dígitos decimais de k

Método da Divisão

Bons valores para m são primos não muito próximos de potências exatas de 2.
Por exemplo, se $n = 2000$ e supondo que se examine uma média de 3 elementos em busca sem sucesso, o tamanho da tabela *hash* deve ser $m = 701$. Pois 701 é primo próximo a $\alpha = 2000/3$, mas não próximo de qualquer potência de 2. Assim
$h(k) = k \, mod \, 701$.

Método da Multiplicação

Opera em 2 passos:

1. multiplica-se a chave k por uma constante A no intervalo $0 < A < 1$ e extrai-se a parte fracional de kA.
2. multiplica-se este valor por m e faz-se o truncamento do resultado

$$h(k) = \lfloor m(k\,A \bmod 1) \rfloor$$

- o valor de m não é crítico como no método da divisão, pondendo ser escolhido inclusive tipicamente uma potência de 2: $m = 2^p$ para algum inteiro p.

Método da Multiplicação

Exemplo: Suponha que $A = (\sqrt{5} - 1)/2 = 0.6180339887...$, $k = 123456$ e $m = 10000$.

$$
\begin{aligned}
h(k) &= \lfloor 10000 \times (123456 \times 0.61803... \; mod \; 1) \rfloor && (1)\\
&= \lfloor 10000 \times (76300.0041151... \; mod \; 1) \rfloor && (2)\\
&= \lfloor 10000 \times 0.0041151... \rfloor && (3)\\
&= \lfloor 41.151... \rfloor && (4)\\
&= 41 && (5)
\end{aligned}
$$

Hashing universal

Consiste em selecionar uma função *hash* aleatoriamente em tempo de execução dentro de uma classe de funções cuidadosamente escolhida.
Definindo matematicamente: seja $\mathcal{H}$ uma coleção finita de funções *hash* que mapeiam o universo de chaves no intervalo $\{0, 1, ..., m-1\}$. Tal coleção é dita ser **universal** se para cada par distinto de chaves $x, y \in U$, o número de funções $h \in \mathcal{H}$ para os quais $h(x) = h(y)$ é exatamente $|\mathcal{H}|/m$.
Segue dessa definição que se h é escolhida uniformemente de modo aleatório de $\mathcal{H}$, a probabilidade de uma colisão entre x e y é $1/m$.

- a função deve ser escolhida aleatoriamente no início da execução da aplicação (por ex., na criação da tabela *hash*)

Hashing universal

Exemplo de construção de uma família de funções $\mathcal{H}$:

1. selecionar um m primo
2. decompor a chave k em $r+1$ dígitos: $k =< k_0, k_1, ..., k_r >$ onde $k_i \in \{0, 1, ..., m-1\}$. Equivalente a escrever a chave k na base m.
3. Escolher aleatoriamente $a =< a_0, a_1, ..., a_r >$, sendo cada $a_i \in \{0, 1, ..., m-1\}$
4. Fazer $h_a(k) = (\sum_{i=0}^{r} a_i k_i) \, mod \, m$

Endereçamento Aberto

É quando todos os elementos são armazenados na própria tabela *hash*. Isto é, cada entrada contém um elemento ou NULL.

- não há listas encadeadas armazenadas fora da tabela
- o fator de carga α nunca pode exceder 1
- para fazer a inserção é necessário fazer a **sondagem** da tabela *hash* até encontrar um *slot* vazio no qual a chave é colocada.
- para determinar quais *slots* sondar, a função *hash* é extendida para incluir o número de sondagem (iniciando de 0) como uma segunda entrada. Assim a função se torna:

$$h : U \times \{0, 1, ..., m-1\} \rightarrow \{0, 1, ..., m-1\}$$

Endereçamento Aberto

Para cada chave k, é requerido que a sequência de sondagem dada por:

$$< h(k,0), h(k,1), ..., h(k,m-1) >$$

seja uma permutação de $< 0, 1, ..., m-1 >$

```
function INSERE_HASH(T, k)                    ▷ insere chave k na hash T
    i ← 0
    repeat
        j ← h(k, i)
        if T[j] = NIL then
            T[j] ← k
            return j
        else
            i ← i + 1
        end if
    until i = m
    error "hash table overflow"
end function
```

Endereçamento Aberto

Algoritmo de Busca com endereçamento aberto:

```
function BUSCA_HASH(T, k)                    ▷ busca chave k na hash T
    i ← 0
    repeat
        j ← h(k, i)
        if T[j] = k then
            return j
        end if
        i ← i + 1
    until T = NIL or i = m
    return NIL
end function
```

Endereçamento Aberto

Tipos de sondagem

- Linear
- Quadrática
- *Hash* duplo

Endereçamento Aberto

Sondagem Linear

Dada uma função *hash* $h' : U \rightarrow \{0, 1, ..., m-1\}$, o método de **sondagem linear** usa a função *hash*:

$$h(k, i) = (h'(k) + i) \, mod \, m \text{ para } i = 0, 1, ..., m-1.$$

Assim, o primeiro *slot* sondado é $T[h'(k)]$. Em seguida, $T[h'(k) + 1]$, e então até $T[m-1]$.

Endereçamento Aberto

Sondagem Linear

- vantagem: fácil de implementar
- desvantagem: tendência de produzir longos trechos consecutivos de memória ocupados (**agrupamento primário**)

Endereçamento Aberto

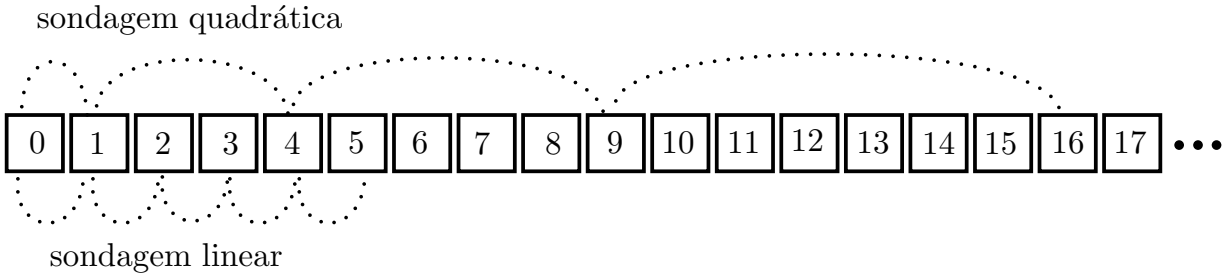
Sondagem Quadrática

O método de **sondagem quadrática** usa uma função *hash* da forma:

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \, mod \, m$$

- Vantagem: evita criação de agrupamentos primários
- Desvantagem: chaves que gerem a mesma posição de *slot* inicial, também produzirão as mesmas posições subsequentes (**agrupamento secundário**). Porém, ainda assim a degradação é menor quando comparado ao agrupamento primário da sondagem linear.

Sondagem Linear x Sondagem Quadrática



Endereçamento Aberto

Hashing Duplo

O método de **hash duplo** usa uma função *hash* da forma:

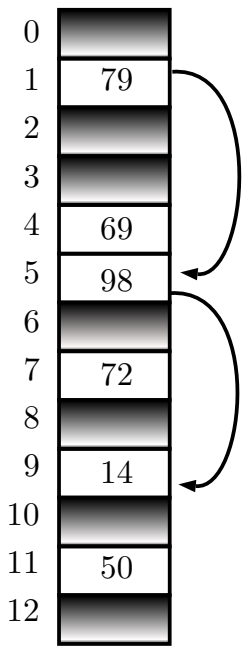
$$h(k, i) = (h_1(k) + i\, h_2(k))\ mod\ m$$

onde h_1 e h_2 são funções *hash* auxiliares.

- A posição inicial sondada é $T[h_1(k)]$, e as posições sondadas seguintes são deslocamentos das posições prévias de quantidade $h_2(k)$ módulo m.

Exemplo de hashing duplo

$m = 13$, $h_1(k) = k \bmod 13$, $h_2(k) = 1 + (k \bmod 11)$.
Como $14 \bmod 13 = 1$ e $14 \bmod 11 = 3$, a chave 14 será inserida no *slot* 9 após sondar os *slots* 1 e 5 e verificar que ambos estão ocupados.



Hashing Duplo

$h_2(k)$ e m devem ser primos entre si para que a tabela *hash* inteira possa ser pesquisada. Se m e $h_2(k)$ tem um máximo divisor comum $d > 1$ para alguma chave k, então a pesquisa pela chave k examinaria somente $1/d$ da tabela *hash*.

Soluções possíveis:

1. m ser potência de 2 e escolher h_2 tal que sempre produza número ímpar
2. m ser primo e escolher h_2 tal que sempre retorne inteiro positivo menor que m

Bibliografia I

[Cormen 1997] Cormen,T.; Leiserson, C.; Rivest, R.
Introduction to Algorithms. McGrawHill, New York, 1997.