Hashing

Marcelo K. Albertini

19 de junho de 2018

Estrutura abstrata: dicionário

- Dicionário: operações
  - put()
  - get()
  - contains()
  - delete()
  - ...
- Implementações:

  `Map<>(), Set<>()`

  - Usar TreeMap, TreeSet se ordem for importante
  - Usar HashMap, HashSet se ordem não for importante

Tabelas de espalhamento = Tabela hash

- Tabelas de espalhamento = Tabelas hash = Hashtable
- Hash significa "misturar, bagunçar"

Porquê "espalhar" ajudaria na busca?

Espalhamos elementos em um espaço limitado e montamos uma tabela para saber a posição.

A tabela é definida por uma **função pré-definida** que associa chaves com a posição na tabela.

Exemplo de tabela hash

Exemplo: método da divisão ("bubblesort do hashing")

Usando: `hash(x) = x % 11`

pos	chave
0	11
1	23
2	
3	
4	
5	38
6	
7	29
8	19
9	
10	

Hashing

Valores são mantidos em uma tabela $T[\cdot]$ com m posições indexada de acordo com uma chave.

Função hash

Um método para computar a posição de uma chave em uma tabela: $h : \mathcal{U} \to \{0, 1, \ldots, m-1\}$.

Aspectos de projeto

- Como computar a função hash?
- Como solucionar uma colisão, quando $h(x) = h(y)$ e $x \neq y$?

Elementos de uma tabela hash

- M espaços de memória
- Função de pré-hashing
  - conversão do tipo composto em um inteiro (ou long)
- Hashing
  - cálculo da posição do inteiro do pré-hashing em uma das M posições da tabela hash
- Resolução de colisões
  - Verificação e posicionamento da chave na tabela de acordo com colisões
- Método de Rehashing
  - Expansão da tabela (aumento de M) e reposicionamento de chaves caso esteja muito cheia (fator de carga alto)

Análise de Tabelas hash

- Custos importantes: M, computação de hashings, verificação da igualdade de elementos em caso de colisão
- Objetivo da análise: como manter M pequeno e reduzir outros custos?

Pré-hashing

Meta

- computação da hash deve ser barata
- boa distribuição das chaves nas posições da tabela (cada posição da tabela tem igual chance de receber chaves)

Exemplo: números de telefones

- Ruim: três primeiros dígitos
- Melhor: três últimos dígitos

Necessário

- Um cálculo de Hash diferente para cada tipo de chave

Pré-hashing em Java: `hashCode()`

- Todas as class herdam o método `int hashCode()`
- Obrigatório: Se `x.equals(y)` então `x.hashCode() == y.hashCode()`
- Meta: Se `!x.equals(y)` então `x.hashCode() != y.hashCode()`

Implementação padrão

Uso do endereço de memória do objeto

Implementações por tipo

Integer, Double, String, File, Url, Date,

Revisão: números e operadores do Java

- `Integer.MIN_VALUE == 0b10000000000000000000000000000000`
- `-1 == 0b11111111111111111111111111111111`
- `-2 == 0b11111111111111111111111111111110`
- `0 == 0b00000000000000000000000000000000`
- `Byte.MIN_VALUE == 0b10000000 == -128`
- `Byte.MAX_VALUE == 0b01111111 == 127`
- Deslocamentos
  - `0b00010000 << 2 == 0b01000000` (à esquerda)
  - `0b00010000 >> 2 == 0b00000100` (à direita)
  - `0b10000000 >> 2 == 0b11100000` (com sinal)
  - `0b10000000 >>> 2 == 0b00100000` (sem sinal)
- Lógica bit a bit
  - `0b10101011 == (~0b01010100)` (negação)
  - `0b01001111 & 0b10101010 == 0b00001010` (AND)
  - `0b01001111 | 0b10101010 == 0b11101111` (OR)
  - `0b01001111 ^ 0b10101010 == 0b11100101` (XOR)

Pré-hashing: implementações do Java

Integer

```
1 int hashCode() {
2  return value;
3 }
```

Double

```
1 int hashCode() {
2  // xor das metades do binario
3  long bits = doubleToLongBits(this.
       doubleValue());
4  return (int)(bits^(bits >>> 32));
5 }
```

Boolean

```
1 int hashCode() {
2  if (value)
3   return 1231;
4  else
5   return 1237;
6 }
```

String

```
1 //s[0]*31^(n-1)+s[1]*31^(n-2)+...+s
      [n-1]
2 public int hashCode() {
3  int h = 0;
4  for (int i = 0; i < s.length; i++)
5   h = s[i] + (31 * h);
6  return h;
7 }
```

Pré-hashing: exemplo para tipo composto

```
1  class Transacao {
2    String quem;
3    Date quando;
4    double quanto;
5
6    int hashCode() {
7      int hash = 17;
8      hash = 31*hash + quem.hashCode();
9      hash = 31*hash + quando.hashCode();
10     hash = 31*hash + ((Double) quanto).hashCode();
11     return hash;
12   }
13 }
```

Pré-hashing: "receita" para tipo composto

Como fazer

- Se campo for primitivo, usar o `hashCode()` do `wrapper`
  - `Integer` para `int`
  - `Double` para `double`
- Se campo for tipo composto, usar `hashCode()`
- Se campo for `array`, usar hash para cada campo
- Se campo for `null`, usar 0
- Combinar cada campo x na hash com a operação $31 * hash + x.hashCode()$

Funções de hashing: tipos

- Funções simples para hashing (não recomendável usar)
  - Método da divisão: $h(x) = x\%m$ (ad hoc)
  - Método da multiplicação (ad hoc)
- Famílias de Funções de Hashing (modelos)
  - Família Uniforme
  - Família Universal
  - Família Quasi-universal
  - Família k-uniforme
  - ...

Funções hash: método da divisão

Sendo

- chave é um inteiro não-negativo k
- temos m posições na tabela
- a posição na tabela pela chave k é obtida pela função hash $h(k)$

Método da divisão

$h(k) = k \mod m$

Funções hash: método da divisão

Método da divisão

$h(k) = k \mod m$

- Se tabela tem $m = 1699$ posições e chave for $k = 25657$, a posição da tabela é $h(k) = 172$
- Evitar o uso de valores para m que sejam potência de 2
  - Se $m = 2^p$, h se torna somente p bits de baixa ordem de k
- Sugestão frequente: "Escolher para m um valor primo não próximo de uma potência de 2"
- Na prática: NÃO USAR PARA CÓDIGOS REAIS pois é fácil obter colisões

Implementação de hashing em Java: `hash()`

- Recebe pré-hashing de `hashcode()`: `int` entre -2^{31} e $2^{31}-1$
- Função hash: índice da tabela de 0 a $M-1$ para tabela com M posições

```
1 hash(Chave c) { // Código ERRADO
2   return c.hashCode() % M; // BUG
3 }
```

Código errado! Problema com negativos.

```
1 hash(Chave c) {
2   return Math.abs(key.hashCode()) % M; // BUG
3 }
```

Código errado! Problema com -2^{31}. `Math.abs(`-2^{31}`) < 0`!

```
1 int hash(Chave c) { // Código CERTO
2   return (c.hashCode() & 0x7fffffff) % M;
3 }
```

Funções hash: método da multiplicação

Método da multiplicação

Multiplicar a chave $k \geq 0$ por uma constante $0 < A < 1$, extrair a parte fracional, multiplicar esse valor pelo número m de posições na tabela e pegar a parte inteira arrendondada para baixo com uma função *floor*.

$h(k) = floor(m(kA mod 1))$, onde $A \approx (\sqrt{5} - 1)/2 \approx 0.618$

```
1 static double A = (Math.sqrt(5.0) -1.0) /2.0;
2 int hash2(Chave c) {
3   int k = c.hashCode() & 0x7fffffff;
4   double fracional = (k*A) % 1;
5   return (int) Math.floor(M * fracional);
6 }
```

Exemplo

Se $m = 2000$ e chave for 6341, a posição obtida pela função hash $h(k)$ é 1476.

Funções hash: métodos da divisão e multiplicação

- Vantagens:
  - Simplicidade
- **Desvantagens:**
  - **para todos inteiros a e x temos $h(x + a \times m) = h(x)$ de maneira predizível**

Resolução de colisões

Uma boa função hash deve espalhar uniformemente valores no array.

Colisões

Valores distintos mapeados na mesma posição de um array são chamados de colisões

Exemplo: em uma tabela de tamanho `M` e método divisivo

$\text{hash}(2.1) == 1$
$\text{hash}(1.4) == 1$
$\text{hash}(7.1) == 1$

Colisões

Desafio

Tratar colisões eficientemente

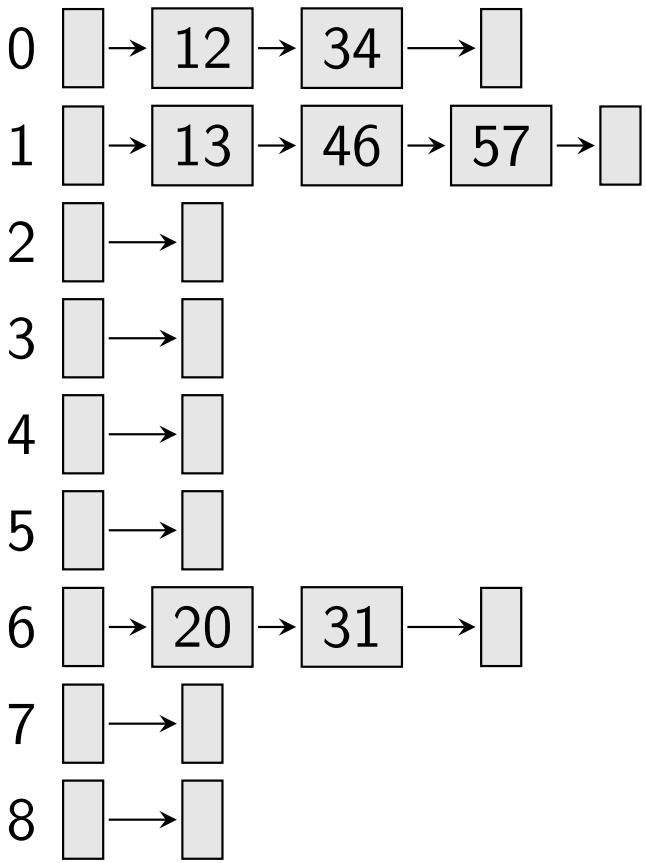
- é muito caro evitar colisões (memória $O(n^2)$)
- suposição comum: colisões são uniformemente distribuídas

Resolução de colisões

- Dois estilos principais de tratar de colisões
  - Encadeamento das colisões ou endereçamento fechado (Estilo `HashMap` do Java)
  - Sondagem ou endereçamento aberto (Estilo `dict` do Python)

Tabela Hash com encadeamento

- Tabela T tem M posições
- $T[1]$ tem lista de chaves colidindo: $h(13) = h(46) = h(57)$



Análise exata de custo de busca mal-sucedida

- Hash table com resolução por encadeamento em M listas
- Seja $|\mathbf{T[i]}|$ o número de chaves em $T[i]$ (colisões em i)
- Custos:
  - computação pré-hashing e hashing custa C_h
  - verificação de igualdade custa C_i
- Custo de busca mal-sucedida:

$$C_h + |\mathbf{T[i]}| \times C_i$$

Análises de custos

- Problema: balancear uso da memória com o número de colisões
- Qual é o número médio de colisões $E[|T[i]|]$?
- Como obter um limitante superior para o número de colisões $[\max |T[i]|]$?
- Como escolher tamanho da tabela M para reduzir chance de colisões?
- Como escolher o tamanho de redimensionamento da tabela para fazer rehashing?

Modelo de distribuição de entradas

- É difícil caracterizar a distribuição de entradas
- É mais fácil usar famílias de Hashing que garantem propriedades independentemente das entradas
  - Famílias de Hashing Uniforme
  - Famílias de Hashing Universal

Análise de Hashing: exemplo

- Também conhecido com "Paradoxo do aniversário"
- Temos M dias, grupo de N pessoas
- Seja i,j um par de pessoas, definir

$$X_{i,j} = \begin{cases} 1 & \text{niver}(i) = \text{niver}(j) \\ 0 & \text{caso contrário} \end{cases}$$

- Suposição: aniversários acontecem uniformemente ao longo do ano: $P[X_{i,j}] = 1/M$
- Em grupo de N pessoas, quantos aniversários acontecem no mesmo dia?

$$E\left[\sum_{\text{pares } i,j} X_{i,j}\right] = \sum_{\text{pares } i,j} E[X_{i,j}] = \binom{N}{2}\frac{1}{M}$$

- Tamanho de um grupo para 2 fazerem aniversário juntos?
  - Ou seja, qual é o N em $1 = \binom{N}{2}\frac{1}{M} = \frac{N(N-1)}{2M}$? $N \approx 27$

Premissa da Família de Hashing Uniforme

- Seja um conjunto de funções de hashing $\mathcal{H}$

$$P_{h \in \mathcal{H}}[h(x) = i] = \frac{1}{m}$$

para todo x e todo i

- Ideia: Todas as posições da tabela são igualmente prováveis para "espalhar chaves uniformemente"

Hashing uniforme NÃO É suficiente: exemplo de família

- Parece razoável mas não é suficiente para termos boas funções hash
- Seja $K = \{const_a | 0 \leq a \leq m-1\}$ tal que $const_a(x) = a$
- Temos que $P_{h \in K}[h(x) = i] = \frac{1}{m}$ pois existem m opções para o valor a

  e $P(h(x) = i) = P(a) \cdot P(h(x) = a) = \frac{1}{m} \cdot 1$

Família de Hashing Universal

- Ideia: "minimizar colisões"

$$P_{h \in H}[h(x) = h(y)] = \frac{1}{m}$$

  para $x \neq y$, $\forall x, y$
- Quase-universal: $P[h(x) = h(y)] \leq \frac{c}{m}$ com c constante

Hashing universal

Se há m para serem selecionados aleatoriamente e colisões tem probabilidade de $1/m$ então essa função hash é universal.

$$P(h = i) = \frac{1}{m} \tag{1}$$

- Aplicações
  - Evitar ataques a uma função hash – exploração do pior caso
  - Reconstrução da hash quando fator de carga mostra risco de colisão

Família de Hashing Universal: exemplo 1

- Família de Hashing Multiplicativo Primal é $MP = \{multp_a | a \in [1 \ldots p-1]\}$
- Definir número primo $p > |U|$, onde U é o conjunto de todos os valores de hashing
- Definir inteiro $a \in [1 \ldots p-1]$ chamado salt
- A função de hashing é

$$multp_a(x) = (ax \mod p) \mod m$$

Família de Hashing Universal: exemplo 2

- Dietzfelbinger et al. (1997) "A Reliable Randomized Algorithm for the Closest-Pair Problem"
- Hashing binário multiplicativo
- Tabela tem $m = 2^l$ posições e pré-hashing produz inteiros com w bits
- Usar um valor aleatório de salt $a \in [0, 2^w[$
- $multb_a(x) = \lfloor \frac{(a \cdot x) \mod 2^w}{2^{w-l}} \rfloor$
- `#define hash(a,x) ((a)*(x) >> (WORDSIZE - HASHBITS))`

Custo médio de busca mal-sucedida

- Tabela hash com encadeamento e família universal $\mathcal{H}$
  - Custo exato é $c_h + |T[i]|c_i$
  - Custo médio é $c_h + E\left[|T[i]|\right] c_i$
- Temos que o tamanho da lista de colisões em $h(x)$ é, para cada chave k na tabela T,

$$|T[h(x)]| = \sum_{k \in T} \delta_{h(k),h(x)}$$

- Portanto queremos achar $E\left[|T\left[h(x)\right]|\right]$

Custo médio de busca mal-sucedida: continuação

- Como busca é mal-sucedida temos que $x \notin T$ e $x \neq k$, então

$$E\left[|\,T\left[h(x)\right]\,|\right] = E\left[\sum_{k \in T} \delta_{h(k),h(x)}\right] = \sum_{k \in T} E\left[\delta_{h(k),h(x)}\right]$$

- Sendo $h \in H$ uma Família Universal, temos

$$E[\delta_{h(k),h(x)}] = \begin{cases} 1 & \text{se } k = x \\ 1/M & \text{se } k \neq x \end{cases}$$

- Dessa forma, com $N = |T|$, o custo é

$$\sum_{k \in T} E\left[\delta_{h(k),h(x)}\right] = \left(\sum_{k \in T} \frac{1}{M}\right) = \frac{N}{M}$$

- Definimos a quantidade $\alpha = \frac{N}{M}$ como o fator de carga

Análise do maior número de colisões no caso médio

- Objetivo: obter limitante superior com alta probabilidade para número de colisões
- Resultado da análise: se tabela tem carga $\alpha = 1$, para N chaves, o número de colisões é $O(\log N / \log \log N)$ com alta probabilidade
- Chance de uma lista X_j ter pelo menos K chaves é:

$$P[X_j \geq K] = \binom{N}{K} \left(\frac{1}{N}\right)^K$$

onde $\binom{N}{K}$ é o número de formas de distribuir K chaves em N posições, $\left(\frac{1}{N}\right)^K$ é chance de cada chave cair em j

- A partir de $P[X_j \geq K]$ queremos achar uma função $f(N) = K$ tal que a probabilidade $P[\max_j X_j \geq f(N)]$ seja pequena

Análise do maior número de colisões no caso médio (1)

$$
\begin{aligned}
P[X_j \geq K] &= \binom{N}{K} \left(\frac{1}{N}\right)^K \\
&= \frac{N!}{K!(N-K)!} \left(\frac{1}{N}\right)^K \\
&\leq \frac{N(N-1)(N-2)\cdots(N-(K-1))}{K!} \left(\frac{1}{N}\right)^K \\
&\leq \frac{N^K}{K!} \left(\frac{1}{N}\right)^K = \frac{1}{K!}
\end{aligned}
$$

Assim, $P[X_j \geq K] \leq \frac{1}{K!}$ e como

$$
\begin{aligned}
K! &= K(K-1)(K-2)(K-3)\ldots \\
K(K-1)(K-2)\ldots 3\cdot 2\cdot 1 &= (K\cdot 1)((K-1)\cdot 2)((K-2)\cdot 3)\ldots \\
K! &\geq K^{K/2} \\
1/K! &\leq 1/K^{K/2}
\end{aligned}
$$

Então

$$
P[X_j \geq K] \leq \frac{1}{K!} \leq \frac{1}{K^{K/2}}
$$

Análise do maior número de colisões no caso médio (2)

- Fazendo $K = 4c \lg N / \lg \lg N$ em $P[X_j \geq K] \leq \frac{1}{K^{K/2}}$

$$K^{K/2} = \left(\frac{4c \lg N}{\lg \lg N} \right)^{2c \lg N / \lg \lg N} \geq \left(\sqrt{\lg N} \right)^{2c \lg N / \lg \lg N}$$

Pois temos que $4c \frac{\lg N}{\lg \lg N} = w(\sqrt{\lg N})$ ao fazer $x = \lg N$ em

$$\lim_{N \to \infty} \frac{4c \frac{\lg N}{\lg \lg N}}{\sqrt{\lg N}} = \frac{\frac{4cx}{\lg x}}{x^{1/2}} = \frac{4c\sqrt{x}}{\lg x} = \infty$$

Por sua vez

$$(\sqrt{\lg N})^{\frac{2c \lg N}{\lg \lg N}} = \left[(\lg N)^{\frac{1}{\lg \lg N}} \right]^{c \lg N} = 2^{c \lg N} = N^c$$

- sendo que $(x)^{\frac{1}{\lg x}} = (x)^{\frac{1}{\log_x x / \log_x 2}} = (x)^{\frac{1}{1/\log_x 2}} = x^{\log_x 2} = 2$
- Portanto $P\left[X_j \geq \frac{4c \lg N}{\lg \lg N} \right] \leq \frac{1}{N^c}$

Análise do maior número de colisões no caso médio (3)

- Até agora vimos que $P\left[X_j \geq \frac{4c \lg N}{\lg \lg N}\right] \leq \frac{1}{N^c}$
- Queremos achar $f(N)$ para quando $P[\max_j X_j \geq f(N)]$ tem baixa probabilidade.
- Usando o limitante da união: $P(A \cup B) \leq P(A) + P(B)$ e que $P\left[\max_j X_j \geq f(N)\right] = P\left[\cup_{j=0}^{N-1} (X_j \geq f(N))\right]$
- temos $P\left[\cup_{j=0}^{N-1} X_j \geq f(N)\right] \leq \sum_{j=0}^{N-1} P\left[X_j \geq \frac{4c \lg N}{\lg \lg N}\right] \leq N \frac{1}{N^c} = \frac{1}{N^{c-1}}$
- Portanto, $\max_j X_j = O(\lg N / \lg \lg N)$ com alta probabilidade $1 - O(N^{-c})$

Controle do número de colisões usando M

- Usando a análise do Paradoxo do aniversário, o número médio de colisões é

$$E\left[\sum_{\text{pares } i,j} X_{i,j}\right] = \binom{N}{2}\frac{1}{M}$$

- Se usarmos bastante espaço, $M = N^2$, então o número esperado de colisões será

$$\frac{N(N-1)}{2}\frac{1}{N^2} \leq 1/2$$

- Então o custo médio de busca mal-sucedida será $c_h + c_i/2 = O(1)$ ao pagarmos um preço de $O(N^2)$ de espaço

Controle do número de colisões usando M e alta prob.

- Seja X_j o número de colisões na posição j
- Queremos achar uma fórmula que liga o número de colisões com M para $\alpha < 1$
- Ou seja, buscamos uma fórmula para fazer $P\left[\max_j X_j \leq 1\right]$ ter alta probabilidade

Controle do número de colisões usando M e alta prob. (1)

- Primeiro devemos achar
  $P[X_j > 1] = 1 - P[X_j = 0] - P[X_j = 1]$
- Como a chance de uma chave não cair na posição j é
  $P[X_j = 0] = (1 - 1/M)$, então, para N chaves,

$$P[X_j = 0] = \left(1 - \frac{1}{M}\right)^N = \sum_{i=0}^{\infty} \binom{N}{i} \left(\frac{-1}{M}\right)^i = 1 - \frac{N}{M} + \Theta\left(\frac{N^2}{M^2}\right)$$

usando expansão binomial de Newton

$$P[X_j = 1] = \frac{N}{M}\left(1 - \frac{1}{M}\right)^{N-1}$$

$$= \sum_{i=0}^{\infty} \binom{N-1}{i} \left(\frac{-1}{M}\right)^i$$

$$= \frac{N}{M}\left(1 - \frac{N-1}{M} + \Theta\left(\frac{N^2}{M^2}\right)\right)$$

Controle do número de colisões usando M e alta prob. (2)

- Queremos achar $P[X_j > 1] = 1 - P[X_j = 0] - P[X_j = 1]$

$$P[X_j > 1] = 1 - \left[1 - \frac{N}{M} + \Theta\left(\frac{N^2}{M^2}\right)\right] - P[X_j = 1]$$

$$P[X_j > 1] = \frac{N}{M} + \Theta\left(\frac{N^2}{M^2}\right) - P[X_j = 1]$$

$$P[X_j > 1] = \frac{N}{M} + \Theta\left(\frac{N^2}{M^2}\right) - \frac{N}{M}\left(1 - \frac{N-1}{M} + \Theta\left(\frac{N^2}{M^2}\right)\right)$$

$$P[X_j > 1] = \Theta\left(\frac{N^2}{M^2}\right) + \frac{N}{M}\frac{N-1}{M} - \frac{N}{M}\Theta\left(\frac{N^2}{M^2}\right)$$

- Usando $N < M$, $P[X_j > 1] < \frac{N(N-1)}{M^2}$

Controle do número de colisões usando M e alta prob. (3)

- Usando o limitante da probabilidade da união

$$P\left[\max_j X_j > 1\right] = P\left[X_j > 1 \text{ para } \forall j\right]$$
$$< \sum_{j=0}^{M-1} \frac{N(N-1)}{M^2} = N(N-1)/M$$

- Portanto, se usarmos $M = N^{2+\epsilon}$ temos probabilidade menor que $1/N^{\epsilon}$ de não haver colisões
- Se usarmos $M = cN^2$, o número esperado de colisões é $E_{\forall \text{ pair } x,y \in T}\left[h(x) = h(y)\right] < 1/c$

Controle de colisões: método da matriz usando hashing universal

Método da matriz

Chave x tem u bits. Tamanho da tabela M é uma potência de 2. Assim, um índice tem b bits tal que $M = 2^b$. É universal pois a probabilidade de colisão é igual ou menor que $1/M$.

Passos do algoritmo de hash

1. obter representacao binária x da chave
2. definir aleatoriamente uma matriz h de tamanho $b \times u$ com valores 0 ou 1
3. obter o hash usando a multiplicação $h(x) = hx$ e fazendo soma resto 2.

Exemplo:

$$h = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}, x = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, h(x) = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \quad (2)$$

Existe uma chance de $1/2^b$ de colisão.

Hashing perfeito

- No hashing perfeito queremos custo de busca seja $O(1)$
- Também queremos que o uso do espaço total seja $O(N)$
- Como fazer? Podemos usar no segundo nível da hash table outras hash tables encadeadas
  - Seja $N_j = |T[j]|$
  - Tamanho da tabela no primeiro nível é $M = N$
  - Tamanho M_j da tabela de segundo nível na posição j será $M_j = 2N_j^2$
  - Portanto, o número esperado de colisões nas tabelas secundárias será $< 1/2$
  - O custo médio de busca será $c_{\text{prehashing}} + 2c_{\text{hashing}} + \frac{1}{2}c_i$
- Qual é o espaço total médio usado? Calcular $E\left[\sum_{j=0}^{M-1} 2\,|T[j]|^2\right]$

Hashing perfeito com espaço $O(N)$

- Calcular $E\left[\sum_{j=0}^{M-1} 2N_j^2\right] = 2\sum_{j=0}^{M-1} E\left[N_j^2\right]$

$$
\begin{aligned}
\sum_{j=0}^{M-1} N_j^2 &= \sum_{j=0}^{M-1}\left(\sum_{x\in T}\delta_{h(x),j}\right)^2 \\
&= \sum_{j=0}^{M-1}\left(\sum_{x\in T}\delta_{h(x),j}\right)\left(\sum_{y\in T}\delta_{h(y),j}\right) \\
&= \sum_{j=0}^{M-1}\sum_{x\in T}\sum_{y\in T}\delta_{h(x),j}\delta_{h(y),j} \\
&= \sum_{j=0}^{M-1}\left(\sum_{x=y\in T}\left[\delta_{h(x),j}\right]^2 + \sum_{x\neq y\in T}\delta_{h(x),j}\delta_{h(y),j}\right) \\
&= \sum_{x=y\in T}\sum_{j=0}^{M-1}\left[\delta_{h(x),j}\right]^2 + \sum_{x\neq y\in T}\sum_{j=0}^{M-1}\delta_{h(x),j}\delta_{h(y),j} \\
&= \sum_{x\in T}\sum_{j=0}^{M-1}\left[\delta_{h(x),j}\right]^2 + \sum_{x\neq y\in T}\delta_{h(x),h(y)} \\
&= N + \sum_{x\neq y\in T}\delta_{h(x),h(y)}
\end{aligned}
$$

- com $\sum_{x\in T}\sum_{j=0}^{M-1}\left[\delta_{h(x),j}\right]^2 = N$ pois é a contagem de 1^2 das chaves em suas posições na tabela T

Hashing perfeito com espaço $O(N)$ – continuação

- Calcular $E\left[\sum_{j=0}^{M-1} 2N_j^2\right] = 2\sum_{j=0}^{M-1} E\left[N_j^2\right]$

$$
\begin{aligned}
E\left[\sum_{j=0}^{M-1} N_j^2\right] &= E[N] + E\left[\sum_{x\neq y\in T} \delta_{h(x),h(y)}\right] \\
&= N + \left(N^2 - N\right)\frac{1}{M} \quad \text{(hashing universal)} \\
&= N + (N-1) = 2N - 1 \quad (1^{\text{o}} \text{ nível } M = N) \\
2E\left[\sum_{j=0}^{M-1} N_j^2\right] &= 4N - 2
\end{aligned}
$$

- Portanto, o custo de espaço do Hashing Perfeito usando 2 níveis é $4N - 2 = O(N)$ com uma família universal de funções hash.

Uso de outras estruturas para tratar colisões

- Árvore de Busca Binária Balanceada
  - Custo de busca será $c_h + \lg |T[x]| c_i$
  - Usado como última linha de defesa contra degradação da hash
- Outras hash tables com M posições
  - Similar a árvore com bifurcação M
  - Custo de busca será $c_h + \log_M |T[x]| c_i$
  - Se $M = \sqrt{N}$, no pior caso, será $c_h + \log_{\sqrt{N}} N c_i = c_h + 2c_i$

Exemplo de Tabela com Encadeamento: HashMap Java

- java.util.HashMap (Java 9)
- Fator de carga máximo padrão: 0.75
- Endereçamento fechado – uso de buckets
  - Buckets são listas se número de colisões < 8
  - Lista vira árvore Rubro-Negra para ≥ 8 colisões
  - Usa pre-hashing ou função de comparação
- Mapeamento em posição é feito com

  `hash(key) & (length -1)`

```
1 static final int hash(Object key) {
2  int h;
3  return (key == null) ?
4              0 : (h = key.hashCode()) ^ (h >>> 16);
5 }
```

Tabela de símbolos: sondagem linear

Endereçamento aberto

Usar apenas o array da tabela. Não usar listas ligadas.

Ideia: inserção

- Se houver uma colisão, buscar algum lugar vazio nas posições seguintes

Busca

- Verificar se o elemento na posição da hash da chave é o procurado
- Procurar nas posições seguintes até encontrar um vazio

Array tem que ser maior que o número de elementos a serem inseridos.

Tabela hash com sondagem linear

```
1  class TabelaHash<Chave, Valor> {
2    int M; // tamanho
3    Chave[] chaves;
4    Valor[] valores;
5
6    public TabelaHash(int M) {
7      chaves  = (Chave[]) new Object[M];
8      valores = (Valor[]) new Object[M];
9      this.M = M;
10   }
11
12   void put(Chave c, Valor v) {
13     ...
14   }
15   void get(Chave c) {
16     ...
17   }
18 }
```

Hashing com sondagem linear

```
1  void put(Chave c, Valor v){
2   int h = hash(c);
3
4   if (chaves[h] == null) {
5    chaves[h] = c;
6    valores[h] = v;
7   } else { // colisão
8    while(++h < M &&
9    chaves[h] != null) {}
10   if (h == M) {
11     // tabela cheia
12     // reestruturar tabela
13   }
14   chaves[h] = c;
15   valores[pos] = v;
16  }
17 }
```

Chave	Hash
2.1	1
1.4	1
1.3	11
5.1	3
6.0	6
7.1	1
-1.1	2
0.0	0

M = 13

pos	chave
0	0
1	2.1
2	1.4
3	5.1
4	7.1
5	-1.1
6	6
7	
8	
9	
10	
11	1.3
12	

Tabela Hash com resolução de colisões por sondagem

- Tabela de M posições de memória
- Cada posição contém só 1 chave
- Se ocorrer colisão, uma recorrência indicará próxima posição que será verificada para inserção
- Define uma sequência de funções hash:

$$\langle h_0, h_1, h_2, \ldots, h_{M-1} \rangle$$

- Para uma chave x, temos uma sequência de sondagem

$$\langle h_0(x), h_1(x), h_2(x), \ldots \rangle$$

  - Idealmente, a sequência de sondagem deve ser uma permutação das posições da tabela $[0, M-1]$

Tabela Hash com resolução de colisões por sondagem: operações

- Inserção
  - Inserir na primeira posição vazia indicada pela sequência de sondagem
- Busca
  - Verificar cada posição indicada pela sequência de sondagem até atingir posição vazia
- Remoção
  - Usar marcador de remoção
  - Alternativa: reposicionar cada colisão da mesma sequência de sondagem

Resolução de colisões por sondagem: tipos de recorrência

- Sondagem Linear

$$h_i(x) = (h(x) + i) \mod M$$

  bom para caching, mas tem efeito de "clustering"

- Sondagem Binária

$$h_i(x) = h(x) \oplus i$$

Tabela Hash com resolução de colisões por sondagem: análise

- Premissa Forte de Hashing Uniforme (PFHU)
  - Para qualquer chave x, a sequência de sondagem é um sorteio de uma permutação das posições da tabela $[0, M-1]$ com probabilidade uniforme do conjunto de todas as permutações
  - Independencia: a sequência de sondagem $\langle h_i(x), h_{i+1}(x), h_{i+2}(x), \ldots \rangle$ continua sendo uma permutação uniforme após a sondagem de $\langle h_0(x), \ldots, h_{i-1} \rangle$
- Sob a PFHU temos:
  - P [colisão em $h_0(x)$] $= \frac{N}{M}$ para tabela com N chaves e M posições
- Premissa facilita análise, mas não existe (sequência) hash que comprovamente é PFHU

Tabela Hash com resolução de colisões por sondagem: análise

- Recorrência do número médio de sondagens necessárias, com mínimo de uma sondagem para tabela vazia $T(M, 0) = 1$,
- Usando indução provamos que $T(M, N) < \frac{M}{M-N}$

$$
\begin{aligned}
T(M, N) &= 1 + \tfrac{N}{M} T(M-1, N-1) \\
&< 1 + \tfrac{N}{M} \tfrac{M-1}{(M-1)-(N-1)} \quad \text{(aplicando } \tfrac{M}{M-N}\text{)} \\
&< 1 + \tfrac{N}{M} \tfrac{M-1}{M-N} \\
&< 1 + \tfrac{N}{M} \tfrac{M}{M-N} \quad \text{(troca } M-1 \text{ por } M\text{)} \\
&= 1 + \tfrac{N}{M-N} = \tfrac{M-N+N}{M-N} = \tfrac{M}{M-N} \\
&= \left(\tfrac{M-N}{M}\right)^{-1} = (1-\alpha)^{-1}
\end{aligned}
$$

- Com fator de carga $\alpha = M/N$
- Portanto temos que $T(M, N) < \frac{1}{1-\alpha}$ e $T(M, N) = O(1)$

Implementação Tabela Hash: Python

- Pre-hashing: link para pyhash.c
  - Tipos compostos: Função Hash Fowler Noll Vo (FNV) Modificada

```
1 hash = FNV offset basis // uma constante pré-definida
2 for each byte of data to be hashed
3     hash = hash * FNV prime (mod 2^N)
4     hash = hash XOR byte of data
5 return hash
```

- Dicionário (hash table): link para dictobject.c

  `my_dict = {1: 'apple', 2: 'ball'}`

- Endereçamento aberto, pseudo-random probing pela recorrência: $j = ((5 * j + 1) \mod 2 * *i$
- Fator de carga máximo: 0.66

Análise: sondagem linear

Knuth (1962): problema do estacionamento

Sob a premissa de espalhamento uniforme ($P\left[h(x) = j\right] = 1/M$), a média de sondagem de uma tabela hash de tamanho M que contém $N = \alpha M$ chaves, $0 < \alpha < 1$ é:

Para busca (encontrando)

$$\leq \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$$

Para inserção (ou busca sem encontrar)

$$\leq \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$$

Análise: sondagem linear

Avaliação de carga de uma hash table: risco de colisões

taxa de ocupação $\alpha = \frac{\text{número de elementos}}{\text{tamanho da tabela}}$

Quando taxa de ocupação $\alpha = N/M$

- Se M alto, muitas posições vazias no array
- Se M pequeno, tempo de busca aumenta devido a agrupamento
- Com $\alpha = 1/2$, sondagens para busca é cerca de $3/2$.
- Com $\alpha = 1/2$, sondagens para inserção é cerca de $5/2$

Endereçamento aberto: premissa do espalhamento uniforme

Temos espalhamento perfeitamente uniforme quando

Cada chave é igualmente provável a ser mapeada para um inteiro entre 0 e M-1.

Propriedades conhecidas

- Duas chaves terão a mesma posição na tabela (colidirão) após cerca de $\sqrt{\pi M/2}$ chaves aleatórias
- Cada inteiro terá sido utilizado para mapear pelo menos uma vez após $\approx M \ln M$ chaves
- Após M chaves, o inteiro mais mapeado foi usado $\Theta(\log M / \log \log M)$ vezes

Variações de hashing

- Hashing encadeado duplo
  - Obter 2 hashes para uma chave e inserir na cadeia menor
  - Reduz tamanho esperado da maior cadeia para $\log \log N$
- Hashing com sondagem variável
  - Usar sondagem linear com saltos variáveis maiores que 1
  - Reduz agrupamento
- Cuckoo hashing
  - Faz 2 hashes por chave; insere chave em uma das posições; se ocupado reinsere a chave deslocada na posição alternativa (com recorrência) (com recorrência)
  - Pior caso garantido constante para busca

Cuckoo hashing

Estratégia

- Usar duas funções hashes
  - Exemplo 1: método da divisão
  - Exemplo 2: método da multiplicação

Na busca `get(c)`

- c pode estar na posição da hash 1 ou da hash 2

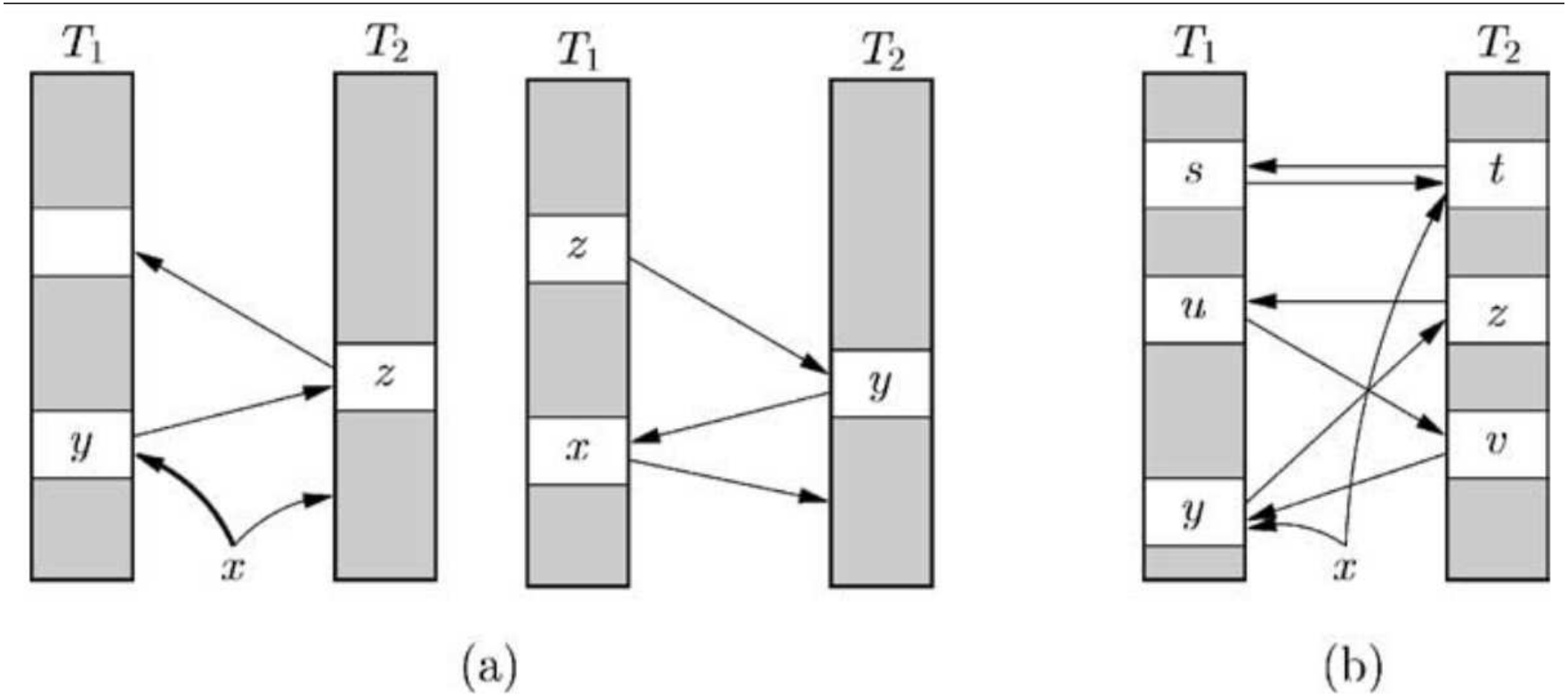
Na inserção `put(c,v)`

1. Tentar a hash 1, se estiver ocupado tentar hash 2
2. Se hash 2 ocupado, substituir o que estava ali e reposicioná-lo
3. Se houver até $6 \log n$ substituições, parar e reconstruir o hash com mais espaço

Video: `http://www.youtube.com/watch?v=dAU5MTXmAPY`

put(x)

Artigo: *Cuckoo hashing, Pagh & Rodler. 2004*



(a) Operação bem sucedida

(b) Operação mal sucedida: rearranjar

Hashing com encadeamento

- Tamanho da tabela:

Cuckoo hashing: análise

- Tempo de construção esperado: $O(n)$

Cuckoo hashing: tempo de construção

- Condições:
  - 2 tabelas de endereçamento aberto com $(1+\epsilon)n$, ϵ chaves

Cuckoo hashing: operação put

```
1  // assume que chave não está na tabela
2  void put(Chave c, Valor v) {
3   int cuckoos = 0; int limite = (int)(log(++N)*6+1);
4   Integer h1 = hash1(c), h2 = hash2(c);
5   Chave ctmp = null; Valor vtmp = null;
6
7   while (cuckoos++ <= limite) {//garantia probabilística
8     if (h1 != null) {// insere com hash1
9       ctmp = chaves1[h1]; vtmp = valores1[h1];
10      chaves1[h1] = c; valores1[h1] = v;
11      h1 = null; h2 = hash2(ctmp);//prepara o cuckoo
12    } else if (h2 != null) {//insere com hash2
13      ctmp = chaves2[h2]; vtmp = valores2[h2];
14      chaves2[h2] = c; valores2[h2] = v;
15      h1 = hash1(ctmp); h2 = null;//prepara o cuckoo
16    }
17    if (ctmp == null) return;//sem chave deslocada.fim.
18    else { c = ctmp; v = vtmp;}//CUCKOO!
19  }
20  if (cuckoos > limite) rehashing(c, v);//obtém espaço
21 }
```

Inserindo (chave, valor) == (15.07, 1.0)

Hash 1

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Inserindo (chave, valor) == (15.07, 1.0)

Hash 1

pos	chave	valor
0	null	null
1	15.07	1.0
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Inserindo (chave, valor) == (0.08, 2.0)

Hash 1

pos	chave	valor
0	null	null
1	15.07	1.0
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Inserindo (chave, valor) == (0.08, 2.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	15.07	1.0
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Inserindo (chave, valor) == (9.01, 3.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	15.07	1.0
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Inserindo (chave, valor) == (9.01, 3.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	15.07	1.0
2	null	null
3	9.01	3.0
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Inserindo (chave, valor) == (6.04, 4.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	15.07	1.0
2	null	null
3	9.01	3.0
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Inserindo (chave, valor) == (6.04, 4.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	15.07	1.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Inserindo (chave, valor) == (27.06, 5.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	15.07	1.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Operação CUCKOO (chave, valor) = (15.07, 1.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Número de operações CUCKOO = 1

Limite atual de operações CUCKOO = 10.0

Inserindo (chave, valor) == (27.06, 5.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Inserindo (chave, valor) == (16.01, 6.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Operação CUCKOO (chave, valor) = (0.08, 2.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	null	null
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 1

Limite atual de operações CUCKOO = 11.0

Inserindo (chave, valor) == (16.01, 6.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	0.08	2.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Inserindo (chave, valor) == (28.02, 7.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	null	null
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	0.08	2.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Inserindo (chave, valor) == (28.02, 7.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	28.02	7.0
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	0.08	2.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Inserindo (chave, valor) == (24.04, 8.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	28.02	7.0
8	null	null
9	null	null

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	0.08	2.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Inserindo (chave, valor) == (24.04, 8.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	28.02	7.0
8	null	null
9	24.04	8.0

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	0.08	2.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Inserindo (chave, valor) == (13.08, 9.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	28.02	7.0
8	null	null
9	24.04	8.0

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	0.08	2.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Inserindo (chave, valor) == (13.08, 9.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	0.08	2.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Inserindo (chave, valor) == (26.07, 10.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	0.08	2.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Operação CUCKOO (chave, valor) = (24.04, 8.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	28.02	7.0
8	13.08	9.0
9	26.07	10.0

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	0.08	2.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 1

Limite atual de operações CUCKOO = 14.0

Operação CUCKOO (chave, valor) = (0.08, 2.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	28.02	7.0
8	13.08	9.0
9	26.07	10.0

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	24.04	8.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 2

Limite atual de operações CUCKOO = 14.0

Operação CUCKOO (chave, valor) = (16.01, 6.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	28.02	7.0
8	13.08	9.0
9	26.07	10.0

Hash 2

pos	chave	valor
0	null	null
1	null	null
2	null	null
3	24.04	8.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 3

Limite atual de operações CUCKOO = 14.0

Inserindo (chave, valor) == (26.07, 10.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	28.02	7.0
8	13.08	9.0
9	26.07	10.0

Hash 2

pos	chave	valor
0	null	null
1	16.01	6.0
2	null	null
3	24.04	8.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Inserindo (chave, valor) == (11.04, 11.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	null	null
7	28.02	7.0
8	13.08	9.0
9	26.07	10.0

Hash 2

pos	chave	valor
0	null	null
1	16.01	6.0
2	null	null
3	24.04	8.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Inserindo (chave, valor) == (11.04, 11.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	28.02	7.0
8	13.08	9.0
9	26.07	10.0

Hash 2

pos	chave	valor
0	null	null
1	16.01	6.0
2	null	null
3	24.04	8.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Inserindo (chave, valor) == (21.06, 12.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	28.02	7.0
8	13.08	9.0
9	26.07	10.0

Hash 2

pos	chave	valor
0	null	null
1	16.01	6.0
2	null	null
3	24.04	8.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Operação CUCKOO (chave, valor) = (28.02, 7.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	21.06	12.0
8	13.08	9.0
9	26.07	10.0

Hash 2

pos	chave	valor
0	null	null
1	16.01	6.0
2	null	null
3	24.04	8.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 1

Limite atual de operações CUCKOO = 15.0

Operação CUCKOO (chave, valor) = (16.01, 6.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	21.06	12.0
8	13.08	9.0
9	26.07	10.0

Hash 2

pos	chave	valor
0	null	null
1	28.02	7.0
2	null	null
3	24.04	8.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 2

Limite atual de operações CUCKOO = 15.0

Operação CUCKOO (chave, valor) = (0.08, 2.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	21.06	12.0
8	13.08	9.0
9	26.07	10.0

Hash 2

pos	chave	valor
0	null	null
1	28.02	7.0
2	null	null
3	24.04	8.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 3

Limite atual de operações CUCKOO = 15.0

Operação CUCKOO (chave, valor) = (24.04, 8.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	21.06	12.0
8	13.08	9.0
9	26.07	10.0

Hash 2

pos	chave	valor
0	null	null
1	28.02	7.0
2	null	null
3	0.08	2.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 4

Limite atual de operações CUCKOO = 15.0

Operação CUCKOO (chave, valor) = (26.07, 10.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	21.06	12.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	null	null
1	28.02	7.0
2	null	null
3	0.08	2.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 5

Limite atual de operações CUCKOO = 15.0

Inserindo (chave, valor) == (21.06, 12.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	21.06	12.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	28.02	7.0
2	null	null
3	0.08	2.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Inserindo (chave, valor) == (23.08, 13.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	21.06	12.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	28.02	7.0
2	null	null
3	0.08	2.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Operação CUCKOO (chave, valor) = (11.04, 11.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	23.08	13.0
7	21.06	12.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	28.02	7.0
2	null	null
3	0.08	2.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 1

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (0.08, 2.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	23.08	13.0
7	21.06	12.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	28.02	7.0
2	null	null
3	11.04	11.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 2

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (16.01, 6.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	23.08	13.0
7	21.06	12.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	28.02	7.0
2	null	null
3	11.04	11.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 3

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (28.02, 7.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	23.08	13.0
7	21.06	12.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	16.01	6.0
2	null	null
3	11.04	11.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 4

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (21.06, 12.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	23.08	13.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	16.01	6.0
2	null	null
3	11.04	11.0
4	null	null
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 5

Limite atual de operações CUCKOO = 16.0

Inserindo (chave, valor) == (23.08, 13.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	23.08	13.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	16.01	6.0
2	null	null
3	11.04	11.0
4	21.06	12.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Inserindo (chave, valor) == (25.08, 14.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	23.08	13.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	16.01	6.0
2	null	null
3	11.04	11.0
4	21.06	12.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Operação CUCKOO (chave, valor) = (0.08, 2.0)

Hash 1

pos	chave	valor
0	25.08	14.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	23.08	13.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	16.01	6.0
2	null	null
3	11.04	11.0
4	21.06	12.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 1

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (11.04, 11.0)

Hash 1

pos	chave	valor
0	25.08	14.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	23.08	13.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	16.01	6.0
2	null	null
3	0.08	2.0
4	21.06	12.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 2

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (23.08, 13.0)

Hash 1

pos	chave	valor
0	25.08	14.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	16.01	6.0
2	null	null
3	0.08	2.0
4	21.06	12.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 3

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (21.06, 12.0)

Hash 1

pos	chave	valor
0	25.08	14.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	16.01	6.0
2	null	null
3	0.08	2.0
4	23.08	13.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 4

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (28.02, 7.0)

Hash 1

pos	chave	valor
0	25.08	14.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	21.06	12.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	16.01	6.0
2	null	null
3	0.08	2.0
4	23.08	13.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 5

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (16.01, 6.0)

Hash 1

pos	chave	valor
0	25.08	14.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	21.06	12.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	28.02	7.0
2	null	null
3	0.08	2.0
4	23.08	13.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 6

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (25.08, 14.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	21.06	12.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	28.02	7.0
2	null	null
3	0.08	2.0
4	23.08	13.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 7

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (28.02, 7.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	21.06	12.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	25.08	14.0
2	null	null
3	0.08	2.0
4	23.08	13.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 8

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (21.06, 12.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	25.08	14.0
2	null	null
3	0.08	2.0
4	23.08	13.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 9

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (23.08, 13.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	25.08	14.0
2	null	null
3	0.08	2.0
4	21.06	12.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 10

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (11.04, 11.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	23.08	13.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	25.08	14.0
2	null	null
3	0.08	2.0
4	21.06	12.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 11

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (0.08, 2.0)

Hash 1

pos	chave	valor
0	16.01	6.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	23.08	13.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	25.08	14.0
2	null	null
3	11.04	11.0
4	21.06	12.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 12

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (16.01, 6.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	23.08	13.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	25.08	14.0
2	null	null
3	11.04	11.0
4	21.06	12.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 13

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (25.08, 14.0)

Hash 1

pos	chave	valor
0	0.08	2.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	23.08	13.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	16.01	6.0
2	null	null
3	11.04	11.0
4	21.06	12.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 14

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (0.08, 2.0)

Hash 1

pos	chave	valor
0	25.08	14.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	23.08	13.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	16.01	6.0
2	null	null
3	11.04	11.0
4	21.06	12.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 15

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (11.04, 11.0)

Hash 1

pos	chave	valor
0	25.08	14.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	23.08	13.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	16.01	6.0
2	null	null
3	0.08	2.0
4	21.06	12.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 16

Limite atual de operações CUCKOO = 16.0

Operação CUCKOO (chave, valor) = (23.08, 13.0)

Hash 1

pos	chave	valor
0	25.08	14.0
1	27.06	5.0
2	null	null
3	9.01	3.0
4	6.04	4.0
5	null	null
6	11.04	11.0
7	28.02	7.0
8	13.08	9.0
9	24.04	8.0

Hash 2

pos	chave	valor
0	26.07	10.0
1	16.01	6.0
2	null	null
3	0.08	2.0
4	21.06	12.0
5	null	null
6	null	null
7	15.07	1.0
8	null	null
9	null	null

Número de operações CUCKOO = 17

Limite atual de operações CUCKOO = 16.0

Complexidade computacional: Cuckoo Hashing

- Espaço: $\approx 2N$
- `get(c)`: no máximo 2 verificações por chave - determinístico

Para hash uniforme

- Reconstrução esperada para n^2 inserções
- Probabilidade de falha de inserção $O(\frac{1}{N})$
- Pior caso esperado amortizado: $O(1)$
  - O custo de reconstrução é amortizado no custo de n^2 inserções

Tabela hash vs. ABB balanceada

Tabela Hash

- Mais fácil para codificar
- Bom para chaves sem ordem
- Rápido para chaves simples

ABB balanceada

- Garantia de desempenho
- Permite operações ordenadas
- Mais fácil implementar `compareTo()` que `hashCode()`