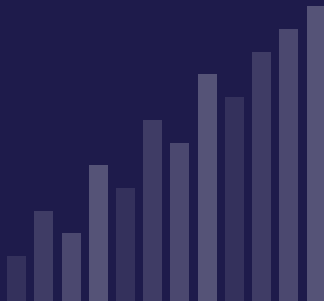

AULA 21 · BLOCO 3: TABELAS HASH E ÁRVORES

Endereçamento aberto

Seção 11.4

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Guardar todas as chaves no próprio vetor, sem listas
- ▶ Comparar sondagem linear, quadrática e hash duplo
- ▶ Explicar por que a remoção exige marcas
- ▶ Medir as sondagens e confrontar com $1/(1 - \alpha)$ e com a fórmula de Knuth



Teoria

Sequência de sondagem, hash duplo, teoremas 11.6 e 11.8



Prática em C

Hash duplo com remoção por marca em C



Experimento

Sondagens: linear contra duplo, de $\alpha = 0,05$ a $0,9$



Conexões

Hardware, sistemas embarcados, linguagens e bancos de dados

Sem listas: sondar a própria tabela

- ▶ Todas as chaves ficam no vetor $T[0..m-1]$: $\alpha = n/m \leq 1$.
- ▶ Para inserir k , testa-se a **sequência de sondagem** $h(k, 0), h(k, 1), \dots$ até achar uma posição vazia.
- ▶ A busca segue a mesma sequência até achar k ou uma posição **vazia**.
- ▶ **Linear:** $h(k, i) = (h'(k) + i) \bmod m$. Simples, amigável ao cache, mas forma **aglomerados**.
- ▶ **Quadrática:** $h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$.
- ▶ **Hash duplo:** $h(k, i) = (h_1(k) + i h_2(k)) \bmod m$, com $h_2(k)$ primo com m para visitar todas as posições.

Hash duplo

- ▶ h_1 escolhe o início; h_2 , o **passo**.
- ▶ Chaves com o mesmo início seguem caminhos diferentes: os aglomerados somem.
- ▶ m primo e $h_2(k) \in [1, m - 1]$, ou $m = 2^p$ e $h_2(k)$ ímpar.
- ▶ $\Theta(m^2)$ sequências diferentes, contra $\Theta(m)$ da linear.

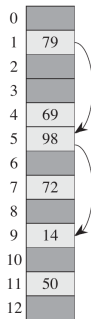


Figura 11.5 de Cormen et al. (2009) — inserção de 14 com $h_1(k) = k \bmod 13$ e $h_2(k) = 1 + (k \bmod 11)$: sondagens 1, 5 e 9.

Quantas sondagens?

Com **hash uniforme** (cada sequência de sondagem igualmente provável):

- ▶ **Teorema 11.6** (sem sucesso): no máximo $\frac{1}{1-\alpha}$ sondagens. Com $\alpha = 0,9$: 10.
- ▶ **Teorema 11.8** (com sucesso): no máximo $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$. Com $\alpha = 0,9$: 2,56.
- ▶ **Sondagem linear** (Knuth): $\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2}\right)$ sem sucesso e $\frac{1}{2} \left(1 + \frac{1}{1-\alpha}\right)$ com sucesso. Com $\alpha = 0,9$: 50,5 e 5,5.

Remoção: não se pode esvaziar a posição (a busca pararia antes da hora). Usa-se uma marca REMOVIDA, que a busca pula e a inserção reaproveita.



Prática em C

- ▶ Tabela de 13 posições, como na Figura 11.5
- ▶ $h_2(k) = 1 + (k \bmod 11)$: nunca zero, primo com 13
- ▶ Marca REMOVIDA para a remoção

Hash duplo em C (1/3)

```
1  #include <stdio.h>
2
3  #define M 13                                // tamanho primo, como na Figura 11.5
4  #define VAZIA -1
5  // marca: a sondagem continua por cima dela
6  #define REMOVIDA -2
7
8  int T[M];
9
10 int h1(int k) { return k % M; }
11 int h2(int k) { return 1 + k % (M - 2); }    // nunca 0 e primo com M: visita tudo
12
13 int sonda(int k, int i) { return (h1(k) + i * h2(k)) % M; }
14
15 int insere(int k) {                          // HASH-INSERT
16     for (int i = 0; i < M; i++) {
```

Hash duplo em C (2/3)

```
17     int j = sonda(k, i);
18     if (T[j] == VAZIA || T[j] == REMOVIDA) { T[j] = k; return j; }
19 }
20 return -1;                                // tabela cheia
21 }
22
23 int busca(int k) {                          // HASH-SEARCH: para numa posição VAZIA
24     for (int i = 0; i < M; i++) {
25         int j = sonda(k, i);
26         if (T[j] == VAZIA) return -1;
27         if (T[j] == k) return j;
28     }
29     return -1;
30 }
31
32 void remove chave(int k) {
```

Hash duplo em C (3/3)

```
33     int j = busca(k);
34     if (j >= 0) T[j] = REMOVIDA;           // não pode virar VAZIA: quebraria buscas
35 }
36
37 int main(void) {
38     for (int j = 0; j < M; j++) T[j] = VAZIA;
39     char c;
40     int k;
41     while (scanf(" %c %d", &c, &k) == 2) {
42         if (c == 'i') printf("%d na posição %d\n", k, insere(k));
43         else if (c == 'b')
44             printf(busca(k) >= 0 ? "%d encontrado\n" : "%d ausente\n", k);
45         else if (c == 'r') remove_chave(k);
46     }
47     return 0;
48 }
```



Experimento

- ▶ $m = 2^{20}$ posições, chaves aleatórias
- ▶ Hash duplo com passo ímpar (primo com 2^{20})
- ▶ Teoremas 11.6 e 11.8 e as fórmulas de Knuth para a linear

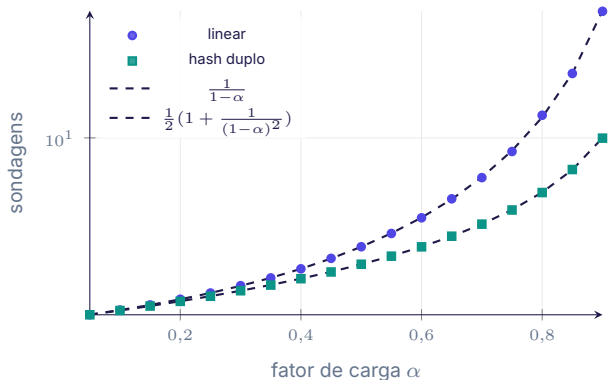
Linear contra hash duplo

Hipótese. O hash duplo se comporta quase como hash uniforme: $1/(1 - \alpha)$ sondagens sem sucesso e $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$ com sucesso. A sondagem linear sofre com aglomerados: $\frac{1}{2}(1 + 1/(1 - \alpha)^2)$ sem sucesso, **muito pior** quando a tabela enche.

Método

1. $m = 2^{20}$; $n = \alpha m$ chaves de 64 bits sorteadas
2. α de 0,05 a 0,90
3. Com sucesso: busca de todas as chaves; sem sucesso: 200.000 sorteadas
4. Mesmas chaves para as duas estratégias

Busca sem sucesso



<i>alfa</i>	medido	razão
0,050	1,052	0,999
0,200	1,248	0,998
0,350	1,535	0,998
0,500	2,002	1,001
0,650	2,860	1,001
0,900	9,970	0,997

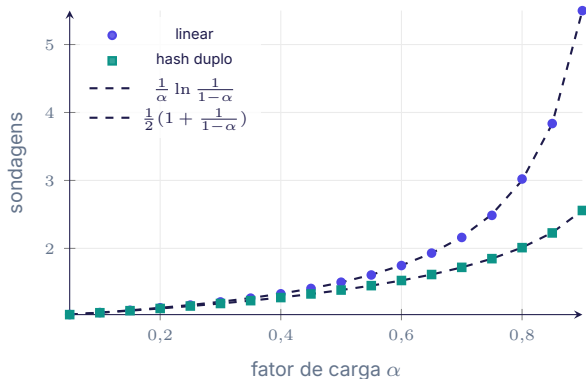
razão = medido $\div$ $\frac{1}{1-\alpha}$

✓ desvio máx. 0,3%

Com 90% de ocupação: 10 sondagens no hash duplo, 50 na linear.

EXPERIMENTO

Busca com sucesso



<i>alfa</i>	medido	razão
0,050	1,025	0,999
0,200	1,116	1,000
0,350	1,231	1,000
0,500	1,387	1,000
0,650	1,615	1,000
0,900	2,558	1,000

razão = medido $\div$ $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

✓ desvio máx. 0,2%

Com sucesso é mais barato: a chave foi inserida quando a tabela estava mais vazia.

EXPERIMENTO

O que o experimento mostrou

As quatro fórmulas se confirmam com desvios de poucos por cento. Até $\alpha \approx 0,5$, linear e duplo empatam; acima disso, os **aglomerados** da linear explodem o custo. Regra prática: endereçamento aberto com $\alpha \leq 0,7$; acima disso, redimensione.

A linear ainda é muito usada: suas sondagens são vizinhas na memória e aproveitam o cache (aula 14). Meça o **tempo**, não só as sondagens.

➤ Reproduza: `gcc -O2 aula21_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Hardware
- ▶ Linguagens
- ▶ Sistemas embarcados
- ▶ Bancos de dados

Onde isso aparece na computação

Hardware

Caches de CPU são tabelas hash em hardware: o endereço escolhe o conjunto, e cada conjunto tem poucas “posições (associatividade).

Linguagens

O HashMap do Rust (SwissTable), o dict do Python e o `absl::flat_hash_map` do C++ usam endereçamento aberto pela localidade de memória.

Sistemas embarcados

Sem malloc nem listas: uma tabela de tamanho fixo com endereçamento aberto cabe em memória estática, ideal para microcontroladores.

Bancos de dados

Agregações (GROUP BY) e *joins* em memória usam tabelas com sondagem linear vetorizada (SIMD) em sistemas analíticos.

Para casa

Exercícios

1. (11.4-1) Insira 10, 22, 31, 4, 15, 28, 17, 88, 59 com $m = 11$: linear, quadrática ($c_1 = 1$, $c_2 = 3$) e duplo.
2. (11.4-2) Escreva HASH-Delete com a marca REMOVIDA e ajuste HASH-INSERT.
3. (11.4-3) Calcule as sondagens esperadas para $\alpha = 3/4$ e $\alpha = 7/8$.
4. Acrescente a sondagem quadrática ao experimento. Onde ela fica entre linear e duplo?
5. Meça o **tempo** de busca da linear e do duplo para $\alpha = 0,5$. Quem vence? Por quê?

Leitura

Cormen et al., seção 11.4.

Código

josedihego.net/estruturas