Estruturas de Dados

aula 28: O truque do passarinho

1 Introdução

Nós vimos que os mecanismos de resolução de colisão por encadeamento externo e endereçamento aberto permitem implementar as operações de acesso aos dados em tempo médio $O(1)$.

Mas para os dois mecanismos, o pior caso da busca ainda é $O(\log n)$ — (*assumindo que as funções de dispersão tem comportamento aleatório*)

Na aula passada, nós vimos uma técnica de dispersão perfeita que resolve esse problema para o mecanismo de encadeamento externo

Quer dizer, a ideia chave foi trocar memória por eficiência na operação de busca.

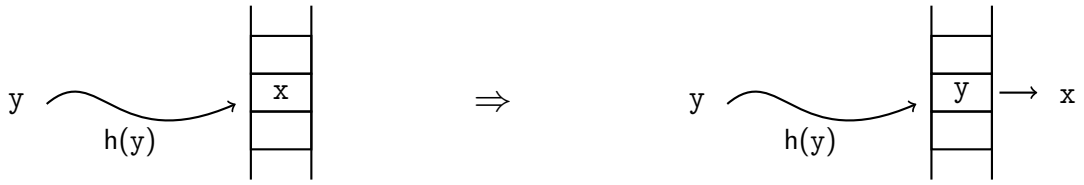
E hoje nós vamos fazer a mesma coisa, para obter outra técnica que reduz o pior caso da busca para $O(1)$, no mecanismo de endereçamento aberto.

2 O truque do passarinho

A técnica do cuco é baseada na seguinte ideia

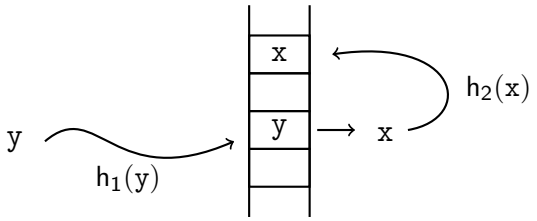
→ *Nenhum elemento é o dono da sua posição na tabela*

Quer dizer, se um novo elemento `y` encontra a sua posição ocupada pelo elemento `x`, ele pode tirá-lo dali



E agora é o `x` quem tem o problema de encontrar um lugar para ele.

Daí, nós usamos uma outra função de dispersão $\mathsf{h}_2(.)$ para encontrar o lugar de `x`

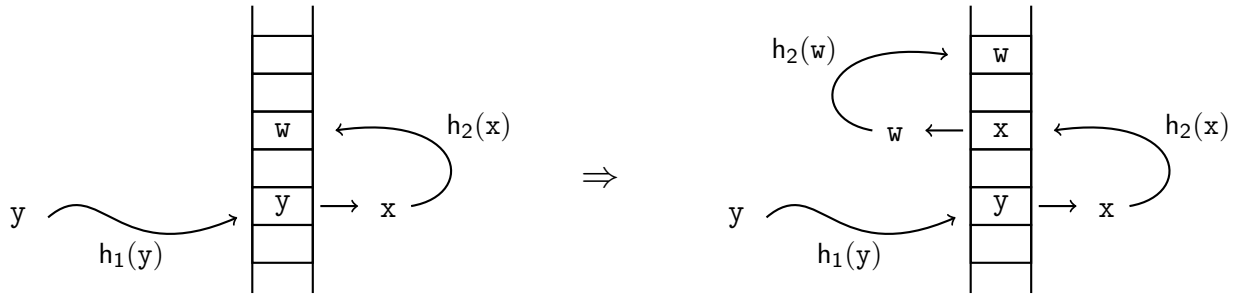


E a coisa acaba aí.

Quer dizer, só existem dois lugares para o elemento `x` na tabela: $\mathsf{h}_1(\mathtt{x})$ e $\mathsf{h}_2(\mathtt{x})$.

— (*e é isso que vai garantir a busca em tempo $O(1)$, no pior caso*)

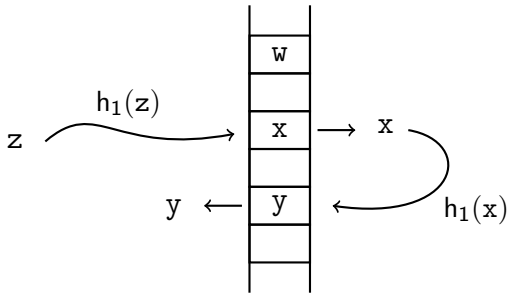
Daí, se o x encontra a sua segunda posição ocupada pelo elemento w, ele pode tirá-lo dali



E daí, pode acontecer uma coisa engraçada ...

Quer dizer, imagine que nesse ponto um novo elemento z quer ocupar o lugar onde o x está.

Daí, o x dá o lugar ao elemento z, e volta para a sua posição original tirando o y dali



E agora o y vai ter que procurar outra posição para ele, utilizando a função $h_2(.)$ ...

Essa técnica é conhecida como *cuckoo hashing* — (*ou o esquema de dispersão do cuco*)

Mas, vejamos um exemplo concreto para entender melhor como a coisa funciona.

Exemplo: Suponha que nós queremos inserir os seguintes elementos em uma tabela de tamanho 11

$$20, \quad 50, \quad 53, \quad 75, \quad 89, \quad 39, \quad 3, \quad 67$$

utilizando as funções de dispersão

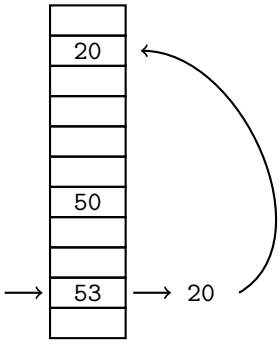
$$h_1(x) \;=\; x \mod 11 \qquad\qquad h_2(x) \;=\; \left\lfloor \frac{x}{11} \right\rfloor \mod 11$$

Bom, o primeiro passo é pré-calcular as duas posições de cada elemento na tabela

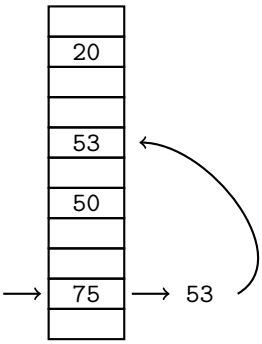
	h_1	h2
20	9	1
50	6	4
53	9	4
75	9	6

	h_1	h2
89	1	8
39	6	3
3	3	0
67	1	6

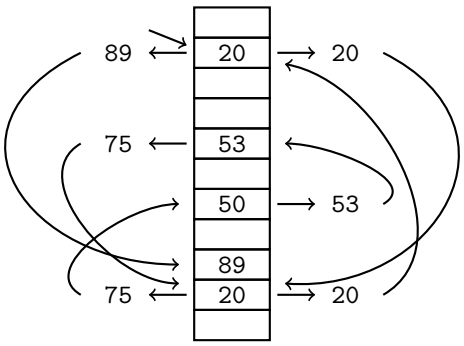
Daí, a primeira colisão ocorre quando o 53 toma o lugar do 20



Depois o 75 toma o lugar do 53

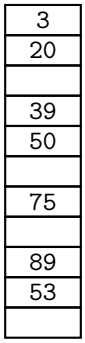


A seguir, a chegada do 89 provoca uma grande confusão



- o 89 tira o lugar do 20
- o 20 tira o lugar do 75
- o 75 tira o lugar do 50
- o 50 tira o lugar do 53
- o 53 tira o lugar do 20
- o 20 tira o lugar do 89
- e o 89 entra na posição 8

Daí, acontecem coisas semelhantes com a inserção do 39 e do 3, deixando a tabela assim



E finalmente chega o 67.

Quer dizer, quando ele chega acontece o seguinte

- o 67 tira o lugar do 20
- o 20 tira o lugar do 53
- o 53 tira o lugar do 50
- o 50 tira o lugar do 75
- o 75 tira o lugar do 20
- o 20 tira o lugar do 67
- o 67 tira o lugar do 50
- o 50 tira o lugar do 53
- o 53 tira o lugar do 75
- o 75 tira o lugar do 67
- o 67 tira ...

e a coisa entrou em ciclo ...

E agora?

Bom, agora não tem jeito.

Quer dizer, o que está acontecendo é que existem 5 elementos

$$20, \quad 50, \quad 53, \quad 67, \quad 75$$

que só podem ocupar 4 posições da tabela

$$1, \quad 4, \quad 6, \quad 9$$

e daí a coisa não pode dar certo — (*sempre fica alguém de fora ...*)

3 Implementação

(. . .)

```
função  busca-CH ( x, T )
 {
     k1  <--  h1(x);    k2  <--  h2(x);

     Se ( T[k1] = x )   Retorna (k1)

     Se ( T[k2] = x )   Retorna (k2)

     Retorna (-1)
 }
```

(. . .)

```
função  remoção-CH ( x, T )
 {
     k  <--  busca-CH (x,T)

     Se ( k = -1 )   Retorna

     T[k] = vazio
 }
```

(. . .)

```
função  inserção-CH ( x, T )
 {
     elem  <--  x;      pos  <--  h1 (elem)

     Repita ( n vezes )
     {
          Se ( T[pos] == vazio )
          {
              T[pos] = elem;   Retorna
          }

          aux = T[pos]
          T[pos] = elem
          elem = aux

          Se ( pos == h1(elem) )      pos  <--  h2 (elem)
          Senão                       pos  <--  h1 (elem)
     }

     Rehash()
 }
```

Nota: `Rehash()` é uma função que escolhe duas novas funções de dispersão `h1()` e `h2()`, e reconstrói a tabela de dispersão.

(. . .)

```
função  rehash-CH ( T )
 {
     h1  <--  nova função de dispersão
     h2  <--  nova função de dispersão

     Para  k <-- 1  Até  n
     {
         Se ( T[k] não está vazio )
         {
             elem  <--  T[k]

             Se ( h1(elem) != k  E  h2(elem) != k )
             {
                  T[k]  <--  vazio

                  inserção-CH ( elem, T )
     }   }   }

 }
```

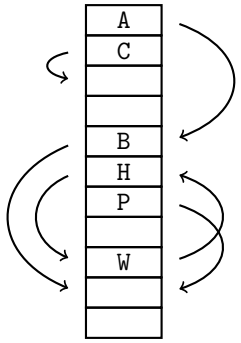
(. . .)

4 Análise

Nós já vimos que as operações de busca e remoção executam em tempo $O(1)$, no pior caso.

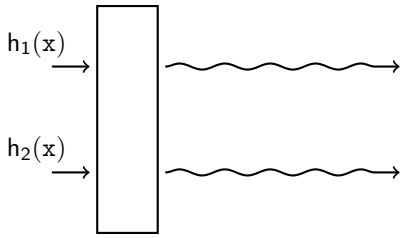
Mas, qual é o tempo médio da operação de inserção ?

Bom, para responder essa pergunta, é útil considerar o *grafo de vizinhanças* da tabela cuco



Quer dizer, nesse esquema as setas indicam para onde o elemento vai, quando ele é tirado da sua posição.

Daí, não é difícil ver que a busca percorre dois caminhos nesse grafo



E o tempo médio da inserção é proporcional ao tamanho dos caminhos que ela percorre.

Em outras palavras, a operação de inserção é eficiente se os caminhos que surgem no grafo de vizinhanças da tabela cuco são curtos.

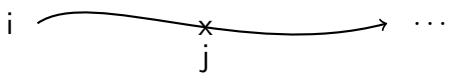
A seguir, nós vamos ver que se a tabela contém muitas posições vazias, então os caminhos são curtos e a inserção executa em tempo médio $O(1)$.

Especificamente, considere uma tabela de tamanho n.

E suponha que ao menos $2n/3$ posições estão vazias.

Para determinar o tamanho médio dos caminhos que surgem nessa tabela, nós fazemos a seguinte pergunta

- *Qual é a probabilidade de que o caminho que começa na posição i passe pela posição j ?*



Bom, a situação mais simples possível é aquela onde existe uma seta de i para j.

Isso acontece quando existe algum elemento $\mathtt{x}$ na tabela tal que

$$\mathsf{h}_1(\mathtt{x}) \;=\; \mathsf{i} \qquad\qquad \text{e} \qquad\qquad \mathsf{h}_2(\mathtt{x}) \;=\; \mathsf{j}$$

Mas a probabilidade de que isso aconteça para um certo elemento $\mathtt{x}$ é igual a

$$\frac{1}{\binom{n}{2}} \quad = \quad \frac{2}{n(n-1)}$$

— (*assumindo que as funções* $\mathsf{h}_1, \mathsf{h}_2$ *tem comportamento aleatório*)

Por outro lado, existem até $n/3$ elementos na tabela.

Mas ainda assim, a probabilidade de que algum desses elementos coloque uma seta de i para j é no máximo

$$\frac{n}{3} \cdot \frac{2}{n(n-1)} \quad \simeq \quad \frac{2}{3n}$$

— (*o que é bem pequenininho, quando* n *é bem grande ...*)

A seguir, nós podemos considerar a possibilidade de que exista um caminho de tamanho 2 entre i e j

$$\mathsf{i} \longrightarrow \mathsf{k} \longrightarrow \mathsf{j}$$

Como indicado no esquema, para que isso aconteça é preciso que exista uma posição k tal que

- existe uma seta entre i e k
- existe uma seta entre k e j

Mas, nós acabamos de ver que a probabilidade de uma seta qualquer é de $2/3n$.

Daí que, a probabilidade de que as duas setas acima existam, para uma certa posição k, é igual a

$$\frac{2}{3n} \cdot \frac{2}{3n} \quad = \quad \left(\frac{2}{3}\right)^2 \cdot \frac{1}{n^2}$$

Por outro lado, existem n posições na tabela que podem fazer o papel de k.

Mas ainda assim, a probabilidade de que exista uma caminho de tamanho 2 entre i e j é no máximo

$$n \cdot \left(\frac{2}{3n}\right)^2 \cdot \frac{1}{n^2} \quad = \quad \left(\frac{2}{3}\right)^2 \cdot \frac{1}{n}$$

— (*o que é bem pequenininho, quando* n *é bem grande ...*)

Agora já deu para o ver o padrão, não é?

Quer dizer, a probabilidade de que exista um caminho de tamanho 3 entre i e j é

$$\left(\frac{2}{3}\right)^3 \cdot \frac{1}{n}$$

e a probabilidade de um caminho de tamanho 4 é de

$$\left(\frac{2}{3}\right)^4 \cdot \frac{1}{n}$$

Daí, a gente observa que

$$\begin{aligned} P(\text{existe um caminho entre i e j}) \quad &= \quad P(\text{existe caminho de tamanho 1}) \\ &+ \; P(\text{existe caminho de tamanho 2}) \\ &+ \; P(\text{existe caminho de tamanho 3}) \\ &+ \; P(\text{existe caminho de tamanho 4}) \\ &+ \; \ldots \end{aligned}$$

E agora nós podemos escrever

$$P(\text{existe um caminho entre i e j}) \quad = \quad \left(\frac{2}{3}\right) \cdot \frac{1}{n} \quad + \quad \left(\frac{2}{3}^2\right) \cdot \frac{1}{n} \quad + \quad \left(\frac{2}{3}^3\right) \cdot \frac{1}{n} \quad + \quad \ldots$$

o que a gente reescreve assim

$$P(\text{existe um caminho entre i e j}) \quad \leq \quad \frac{1}{n} \cdot \sum_{k=1}^{\infty} \left(\frac{2}{3}\right)^k$$

e depois assim

$$P(\text{existe um caminho entre i e j}) \quad \leq \quad \frac{1}{n} \cdot \frac{2/3}{1 - 2/3} \quad = \quad \frac{2}{n}$$

— (*o que é bem pequenininho, quando n é bem grande ...*)

Quer dizer, a probabilidade de que a posição j apareça no caminho que começa em i é bem pequena.

E isso vale para qualquer posição tabela.