

# Aula 22: Árvores chanfradas

## *Splay trees*

David Déharbe  
Programa de Pós-graduação em Sistemas e Computação  
Universidade Federal do Rio Grande do Norte  
Centro de Ciências Exatas e da Terra  
Departamento de Informática e Matemática Aplicada

Download me from <http://DavidDeharbe.github.io>.



Análise amortizada

    Método agregado

    Método do contador

    Método do potencial

Arranjos dinâmicos

Árvores chanfradas

## 1. Avaliação de estruturas de dados: análise amortizada

- ▶ Contador binário
- ▶ Arranjos dinâmicos

# Análise amortizada

- ▶ Medição da complexidade ao longo de  $n$  operações
- ▶ Custo de execução, no pior caso:  $T(n)$
- ▶ Custo amortizado:  $T(n)/n$
- ▶ Exemplos
  1. contador binário
  2. arranjo dinâmico



# Método agregado

- ▶ Considere uma série de  $n$  operações em alguma estrutura de dados,  $n$  qualquer.
- ▶ Determina o custo  $T(n)$  desta série de operações.
- ▶ O custo médio de cada operação é  $T(n)/n$ .
- ▶ Exemplo: contador binário



# Método agregado: um exemplo

INCREMENT( $A$ )

//  $A$  é um arranjo de  $k$  bits

```
1  $i = 0$ 
2 while  $i < \text{length}[A]$  and  $A[i] = 1$ 
3      $A[i] = 0$ 
4      $i = i + 1$ 
5 if  $i < \text{length}[A]$ 
6      $A[i] = 1$ 
```

- ▶ O tamanho da entrada é  $k$ ;
- ▶ O custo da execução da INCREMENT é proporcional ao número de bits invertidos.
- ▶ No pior caso, a complexidade é proporcional a  $k$ ;
- ▶ Logo, o custo de uma sequência de  $n$  operações é  $O(n \times k)$ .
- ▶ Podemos prover uma análise mais precisa?



## Método agregado: um exemplo

Em uma série de  $n$  execuções de INCREMENT:

- ▶  $A[0]$  é invertido  $n$  vezes;
- ▶  $A[1]$  é invertido  $n/2$  vezes;
- ▶  $A[2]$  é invertido  $n/4$  vezes;
- ▶  $A[k - 1]$  é invertido  $n/2^{k-1}$  vezes;

O custo total para  $n$  operações é

$$T(n) = \sum_{i=1}^{\lfloor \lg n \rfloor} n/2^{i-1}$$

$$T(n) < \sum_{i=1}^{\infty} n/2^{i-1}$$

$$T(n) < n \times \sum_{i=1}^{\infty} 1/2^{i-1} = 2n$$

Logo o custo amortizado de INCREMENT é  $T(n)/n = 2 \in \Theta(1)$ : é **constante**.



*Quanto pagar para executar  $n$  operações?*

- ▶ cada operação tem um custo real
- ▶ a cada operação é atribuído um preço ( $\approx$  custo amortizado individual)
- ▶ os preços cobrados devem cobrir os custos reais
- ▶ se o preço é maior que o custo:
  - ▶ crédito é associado a elementos da estrutura de dados
- ▶ se o custo é maior que o preço
  - ▶ a diferença deve poder ser paga com os créditos disponíveis
- ▶ custo amortizado total: soma dos preços cobrados

# Critérios de definição do custo

- ▶ o custo amortizado total deve ser uma cota superior do custo real
  - ▶ a estrutura de dados não “empresta”
- ▶ o crédito total associado aos elementos nunca pode ser negativo



# Método do contador: o contador binário

- ▶ a unidade de custo é inverter um bit
- ▶ preço para  $0 \rightarrow 1$ : 2
  - ▶ 1 é o custo
  - ▶ sobra  $2 - 1 = 1$  de crédito, associado ao bit setado a 1
- ▶ preço para  $1 \rightarrow 0$ : 0
  - ▶ pago com o crédito

# Método do contador: o contador binário

INCREMENT( $A$ )

//  $A$  é um arranjo de  $k$  bits

```
1  $i = 0$ 
2 while  $i < \text{length}[A]$  and  $A[i] = 1$ 
3      $A[i] = 0$ 
4      $i = i + 1$ 
5 if  $i < \text{length}[A]$ 
6      $A[i] = 1$ 
```

- ▶ O custo do laço é pago usando os créditos dos bits setados a 1.
- ▶ no máximo um único bit é setado a 1: custo 1, mais 1 que fica como crédito
- ▶ logo, o custo de uma execução da operação INCREMENT é 2
- ▶ o custo de  $n$  execuções da operação INCREMENT é  $O(n)$ .



# Método do potencial

- ▶ crédito  $\implies$  **potencial** para executar futuras operações
- ▶ estrutura de dados inicia em um estado  $D_0$
- ▶  $n$  operações são executadas,
  - ▶ custo real  $c_1, \dots, c_i, \dots, c_n$
  - ▶ levando aos estados  $D_1, \dots, D_i, \dots, D_n$
- ▶ A função  $\Phi$  associa um número real  $\Phi_i$  (potencial) ao estado  $D_i$
- ▶ **Custo amortizado**  $a_i = c_i + \Phi_i - \Phi_{i-1}$
- ▶ Custo amortizado total

$$\sum_{i=1}^n a_i = \sum_{i=1}^n c_i + \Phi_i - \Phi_{i-1} = \sum_{i=1}^n c_i + \Phi_n - \Phi_0$$

- ▶ se  $\forall i \cdot \Phi_i \geq \Phi_0$ , então  $\sum_{i=1}^n a_i$  é uma cota superior do custo total real.

# Método do potencial

## Relação com o método do contador

- ▶  $\Phi_0 = 0$
- ▶ mostrar que  $\Phi_i \geq 0, \forall i$ .
- ▶  $\Phi_i > \Phi_{i-1} \approx$  crédito (potencial aumenta)
- ▶  $\Phi_i < \Phi_{i-1} \approx$  débito (potencial diminui)

# Método do potencial: o contador binário

- ▶ a  $i$ -ésima chamada de INCREMENT zera  $t_i$  bits
- ▶ custo real  $c_i = t_i + 1$  (zera  $i$ , seta 1)
- ▶ potencial:  $\Phi_i =$  quantidade de bits setados a 1
- ▶ naturalmente,  $\forall i \cdot \Phi_i \geq 0$
- ▶ logo  $\Phi_i \leq \Phi_{i-1} - t_i + 1$
- ▶  $\Phi_i - \Phi_{i-1} \leq 1 - t_i$
- ▶ custo amortizado de uma operação

$$a_i = c_i + \Phi_i - \Phi_{i-1} \leq (t_i + 1) - (1 - t_i) = 2$$

- ▶ custo amortizado de  $n$  operações  $O(n)$



# Arranjos dinâmicos

- ▶ contêiner tabela, cuja capacidade adapta-se ao tamanho da coleção
  - ▶ inserção e remoção: modelo pilha
- ▶ inserção em uma tabela está cheia
  - ▶ aloca uma nova tabela de capacidade maior
  - ▶ copia o conteúdo da tabela original para a nova tabela
  - ▶ libera o espaço ocupado pela tabela original
  - ▶ insere o novo elemento
- ▶ política de expansão: dobra o tamanho
- ▶ remoção, quando o fator de carga fica baixo
  - ▶ aloca uma nova tabela de capacidade menor
  - ▶ copia o conteúdo da tabela original para a nova tabela
  - ▶ libera o espaço ocupado pela tabela original
  - ▶ remove o elemento
- ▶ política de contração: divide a capacidade por dois quando fator de carga chega a  $1/4$ .

# Implementação: dados

- ▶ *T.table* tabela com os elementos
- ▶ *T.num* quantidade de elementos armazenados na tabela
- ▶ *T.size* capacidade máxima



# Inserção

INSERT( $T, k$ )

```
1  if  $T.size == 0$  // alocação inicial
2       $T.table = \text{ALLOC-TABLE}(1)$ 
3       $T.size = 1$ 
4  elseif  $T.num == T.size$  // expansão
5       $tab = T.table$ 
6       $T.table = \text{ALLOC-TABLE}(T.size \times 2)$ 
7      for  $i = 1$  to  $T.size$ 
8           $T.table[i] = tab[i]$ 
9       $T.size = T.size \times 2$ 
10      $\text{FREE-TABLE}(tab)$ 
11   $T.num = T.num + 1$  // inserção
12   $T.table[T.num] = k$ 
```



Cenário:  $n$  inserções em uma tabela inicialmente vazia

- ▶ custo da operação  $i$ 
  - ▶ sem expansão: 1
  - ▶ com expansão:  $i$
- ▶ Pior caso  $O(n^2)$
- ▶ Melhoria desta análise com análise amortizada

# Método agregado

- ▶ Há expansão quando  $i$  é uma potência de 2.
- ▶ Custo total:

$$\begin{aligned}\sum_{i=1}^n c_i &\leq n + \sum_{j=0}^{\lfloor \log_2 n \rfloor} 2^j \\ &< n + 2n \\ &= 3n\end{aligned}$$

# Método do contador

- ▶ Custo 1 para atribuir uma posição de  $T.table$
- ▶ Preço de uma operação de inserção: 3
- ▶ Pagamento de uma operação de inserção, sem expansão:
  - ▶ 1 vai para a inserção (custo real)  $i = T.size/2 + j$
  - ▶ sobra 2 de crédito
  - ▶ 1 crédito na posição de inserção  $i$
  - ▶ 1 crédito para um outro elemento da tabela  $j$
- ▶ Pagamento da expansão:
  - ▶ todos os itens tem um crédito
  - ▶ o crédito paga a expansão

# Método do potencial

- ▶ Após cada expansão:  $\Phi = 0$
- ▶  $\Phi \uparrow$  quando a tabela é preenchida
- ▶ até a tabela ser preenchida, e  $\Phi$  é gasto na expansão
- ▶  $\Phi(T) = 2 \times T.num - T.size$ 
  - ▶ Inicialmente, e após cada expansão  $\Phi(T) = 0$
  - ▶ Antes de uma expansão  $\Phi(T) = 2 \times T.num - T.size = T.size$
  - ▶ A cada momento  $\Phi(T) \geq 0$
  - ▶ Logo, a soma dos custos amortizados é uma cota superior dos custos reais.



# Método do potencial

- ▶  $n_i$ : número de elementos após operação  $i$
- ▶  $s_i$ : capacidade após a operação  $i$
- ▶ inicialmente  $n_0 = s_0 = 0, \Phi_0 = 0$
- ▶  $n_i = n_{i-1} + 1$



se a operação  $i$  não tem expansão:

►  $s_i = s_{i-1}$  e

$$\begin{aligned}a_i &= c_i + \Phi_i - \Phi_{i-1} \\&= 1 + (2 \cdot n_i - s_i) + (2 \cdot n_{i-1} - s_{i-1}) \\&= 1 + (2 \cdot n_i - s_i) + (2 \cdot (n_i - 1) - s_i) \\&= 3\end{aligned}$$

se a operação  $i$  tem expansão:

$$\blacktriangleright s_i = 2 \times s_{i-1}, s_{i-1} = n_{i-1} = n_i - 1 \text{ e}$$

$$\begin{aligned} a_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= n_i + (2 \cdot n_i - s_i) - (2 \cdot n_{i-1} - s_{i-1}) \\ &= n_i + (2 \cdot n_i - 2 \cdot (n_i - 1)) - (2 \cdot (n_i - 1) - (n_i - 1)) \\ &= 3 \end{aligned}$$

# Remoção com contração de capacidade

- ▶ expansão dobra capacidade
- ▶ contração ocorre quando capacidade chega a  $1/4$
- ▶ por quê não escolher  $1/2$ ?

REMOVE( $T$ )

```
1  if  $T.num == 0$ 
2       $T.size = 0$ 
3      FREE-TABLE( $T.table$ )
4  elseif  $T.num \neq 0$  and  $T.num * 4 < T.size$  // contração
5       $tab = T.table$ 
6       $T.table = \text{ALLOC-TABLE}(T.size/2)$ 
7      for  $i = 1$  to  $T.num$ 
8           $T.table[i] = tab[i]$ 
9       $T.size = T.size/2$ 
10     FREE-TABLE( $tab$ )
11   $T.num = T.num - 1$  // remoção
```





# Análise com o método do potencial

- ▶  $\Phi = 0$  logo após contração ou expansão
- ▶ Fator de carga  $\alpha(T) = T.num / T.size$ , se  $T$  não for vazia, 1 se for.
- ▶  $T.num = \alpha(T) \cdot T.size$
- ▶ Função potencial

$$\Phi(T) = \begin{cases} 2 \cdot T.num - T.size & \text{se } \alpha T \geq 1/2 \\ T.size/2 - T.num & \text{se } 1/4 \leq \alpha T < 1/2 \end{cases}$$

A função potencial nunca é negativa: obteremos uma cota superior do custo real

# Análise com o método do potencial

## Justificativa intuitiva

- ▶ antes de uma expansão:
  - ▶  $\alpha(T) = 1$
  - ▶  $T.num = T.size$ ,
  - ▶  $\Phi(T) = T.num$
  - ▶ permite cópia de  $T.num$  elementos.
- ▶ antes de uma contração:
  - ▶  $\alpha(T) = 1/4$
  - ▶  $T.num \cdot 4 = T.size$ ,
  - ▶  $\Phi(T) = T.size/2 - T.num = T.num$
  - ▶ permite cópia de  $T.num$  elementos.

# Análise com o método do potencial

## Notações

- $s_i$  : capacidade após operação  $i$
- $n_i$  : número de elementos após operação  $i$
- $\Phi_i$  : potencial após operação  $i$
- $\alpha_i$  : fator de carga após operação  $i$

Inicialmente:  $s_0 = n_0 = \Phi_0 = 0, \alpha_0 = 1$

# Análise com o método do potencial

## Inserção

A operação  $i$  é uma inserção:

- ▶ já fizemos a análise no caso  $\alpha_{i-1} \geq 1/2$ :  $a_i = 3$
- ▶ se  $\alpha_{i-1} < 1/2$ , não há expansão
- ▶ se  $\alpha_i \geq 1/2$ :

$$\begin{aligned}a_i &= c_i + \Phi_i - \Phi_{i-1} \\&= 1 + (2 \cdot n_i - s_i) - (s_{i-1}/2 - n_{i-1}) \\&= 1 + (2 \cdot (n_{i-1} + 1) - s_{i-1}) - (s_{i-1}/2 - n_{i-1}) \\&= 3 \cdot n_{i-1} - \frac{3}{2} \cdot s_{i-1} + 3 \\&= 3 \cdot \alpha_{i-1} \cdot s_{i-1} - \frac{3}{2} \cdot s_{i-1} + 3 \\&< \frac{3}{2} \cdot s_{i-1} - \frac{3}{2} \cdot s_{i-1} + 3 \\&= 3\end{aligned}$$

# Análise com o método do potencial

## Inserção

A operação  $i$  é uma inserção:

- ▶ já fizemos a análise no caso  $\alpha_{i-1} \geq 1/2$ :  $a_i = 3$
- ▶ se  $\alpha_{i-1} < 1/2$ , não há expansão
- ▶ se  $\alpha_i < 1/2$ :

$$\begin{aligned} a_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (s_i/2 - n_i) - (s_{i-1}/2 - n_{i-1}) \\ &= 1 + (s_i/2 - n_i) - (s_i/2 - (n_i - 1)) \\ &= 0 \end{aligned}$$

O custo amortizado de uma inserção é **3** no máximo.

# Análise com o método do potencial

## Remoção

$$n_i = n_{i-1} - 1$$

- ▶ se  $\alpha_{i-1} < 1/2$ ,
  - ▶ sem contração:  $s_i = s_{i-1}$

$$\begin{aligned}a_i &= c_i + \Phi_i - \Phi_{i-1} \\&= 1 + (s_i/2 - n_i) - (s_{i-1}/2 - n_{i-1}) \\&= 1 + (s_i/2 - (n_{i-1} - 1)) - (s_i/2 - n_{i-1}) \\&= 2\end{aligned}$$

- ▶ com contração:  $n_{i-1} = n_i + 1 = s_i/2 = s_{i-1}/4$  e  $c_i = n_i + 1$

$$\begin{aligned}a_i &= c_i + \Phi_i - \Phi_{i-1} \\&= n_i + 1 + (s_i/2 - n_i) - (s_{i-1}/2 - n_{i-1}) \\&= n_{i-1} + (n_{i-1} - (n_{i-1} - 1)) - (2 \cdot n_{i-1} - n_{i-1}) \\&= 1\end{aligned}$$

# Análise com o método do potencial

## Remoção

$$n_i = n_{i-1} - 1$$

- ▶ se  $\alpha_{i-1} \geq 1/2$ ,
- ▶ não há contração:  $s_i = s_{i-1}$ ,  $n_i = n_{i-1} - 1$
- ▶ se  $\alpha_i \geq 1/2$

$$\begin{aligned}a_i &= c_i + \Phi_i - \Phi_{i-1} \\&= 1 + (2 \cdot n_i - s_i) - (2 \cdot n_{i-1} - s_{i-1}) \\&= 1 + (2 \cdot (n_{i-1} - 1) - s_{i-1}) - (2 \cdot n_{i-1} - s_{i-1}) \\&= -1\end{aligned}$$

# Análise com o método do potencial

## Remoção

$$n_i = n_{i-1} - 1$$

- ▶ se  $\alpha_{i-1} \geq 1/2$ ,
- ▶ não há contração:  $s_i = s_{i-1}$ ,  $n_i = n_{i-1} - 1$
- ▶ se  $\alpha_i < 1/2$

$$\begin{aligned}a_i &= c_i + \Phi_i - \Phi_{i-1} \\&= 1 + (s_i/2 - n_i) - (2 \cdot n_{i-1} - s_{i-1}) \\&= 1 + (s_{i-1}/2 - (n_{i-1} - 1)) - (2 \cdot n_{i-1} - s_{i-1}) \\&= 2 + \frac{3}{2} \cdot s_{i-1} - 3 \cdot n_{i-1} \\&= 2 + \frac{3}{2} \cdot s_{i-1} - 3 \cdot \alpha_{i-1} \cdot s_{i-1} \\&\leq 2\end{aligned}$$

O custo amortizado de uma remoção é **2** no máximo.



# Arranjos dinâmicos

## Síntese da análise de complexidade amortizada

- ▶ inserção tem custo amortizado  $O(1)$
- ▶ remoção tem custo amortizado  $O(1)$



# Arranjos dinâmicos

## Exercício

1. Projetar uma estrutura de dados para arranjos dinâmicos, que tem apenas operação de inserção, tal que esta tenha complexidade constante, *no pior caso*.
  - ▶ hipótese: supõe-se que o custo de uma alocação dinâmica é  $\Theta(1)$ .
  - ▶ dica: pode inspirar-se na aplicação do método do contado desta aula.
2. Estender a estrutura projetada com uma operação de remoção do último elemento (POP-BACK), tal que tanto a inserção quanto a remoção tenham complexidade constante, no pior caso.

# Árvores chanfradas

*splay trees*

- ▶ árvores binárias de busca
- ▶ auto-ajustáveis
- ▶ não necessitam de atributos adicionais
- ▶ complexidade amortizada  $O(\log n)$

# Árvores chanfradas

- ▶ uma única operação básica: chanfrar
  - ▶  $\text{SPLAY}(T, k)$ : chanfra a árvore  $T$  com relação ao valor  $k$
  - ▶ no término da operação  $k$  está na raiz (ou o maior valor  $< k$ , ou o menor valor  $> k$ )
- ▶ busca, inserção e remoção são todas realizadas após chanfrar a árvore



# Realização das operações e o chanframento

- ▶ uma única operação básica: chanfrar
  - ▶  $\text{SPLAY}(T, k)$ : chanfra a árvore  $T$  com relação ao valor  $k$
  - ▶ busca a chave  $k$
  - ▶ remaneja  $T$  tal que  $k$  fica na raiz (ou o maior valor  $< k$ , ou o menor valor  $> k$ )
- ▶ busca, inserção e remoção são todas realizadas após chanfrar a árvore
  - ▶  $\text{SEARCH}(T, k)$ :  $\text{SPLAY}(T, k)$  e consulta a raiz
  - ▶  $\text{INSERT}(T, k)$ :  $\text{SPLAY}(T, k)$  e insere  $k$  na raiz
  - ▶  $\text{REMOVE}(T, k)$ :  $\text{SPLAY}(T, k)$ , remove a raiz,  $T_L$  e  $T_R$  árvores resultantes,  $\text{SPLAY}(T_L, k)$ , e torne  $T_R$  a sub-árvore da nova raiz de  $T_L$ .
  - ▶  $O(1)$  aplicações de  $\text{SPLAY} + O(1)$  operações.

- ▶  $\text{SEARCH}(T, x)$ :  $\text{SPLAY}(T, x)$  e consulta a raiz

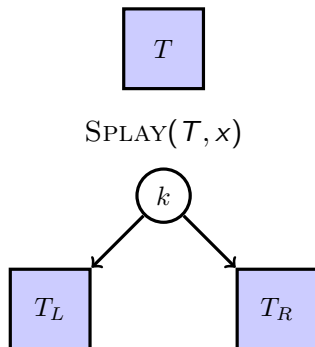


- ▶  $\text{SEARCH}(T, x)$ :  $\text{SPLAY}(T, x)$  e consulta a raiz



$\text{SPLAY}(T, x)$

- $\text{SEARCH}(T, x)$ :  $\text{SPLAY}(T, x)$  e consulta a raiz





- ▶  $\text{INSERT}(T, k)$ :  $\text{SPLAY}(T, k)$  e insere  $k$  na raiz

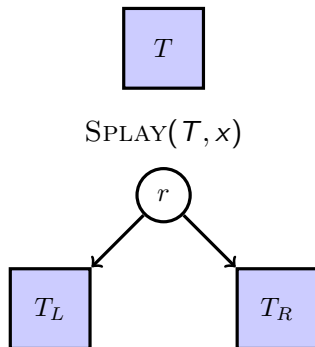


- ▶  $\text{INSERT}(T, k)$ :  $\text{SPLAY}(T, k)$  e insere  $k$  na raiz

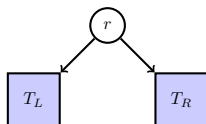


$\text{SPLAY}(T, x)$

- ▶  $\text{INSERT}(T, k)$ :  $\text{SPLAY}(T, k)$  e insere  $k$  na raiz



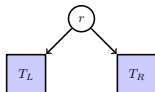
- ▶  $\text{INSERT}(T, k)$ :  $\text{SPLAY}(T, k)$  e insere  $k$  na raiz



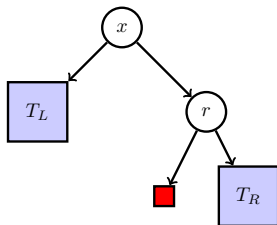
```
n.val = x;  
n.left = T.root.left;  
T.root.left = NIL;  
n.right = T.root;  
T.root = n
```

# Inserção

- ▶  $\text{INSERT}(T, k)$ :  $\text{SPLAY}(T, k)$  e insere  $k$  na raiz



$n.val = x;$   
 $n.left = T.root.left;$   
 $T.root.left = \text{NIL};$   
 $n.right = T.root;$   
 $T.root = n$



- ▶  $\text{REMOVE}(T, k)$ :  $\text{SPLAY}(T, k)$ , remove a raiz,  $T_L$  e  $T_R$  árvores resultantes,  $\text{SPLAY}(T_L, k)$ , e torne  $T_R$  a sub-árvore da nova raiz de  $T_L$ .



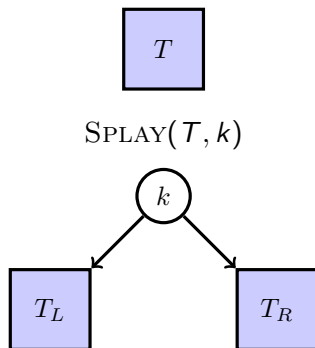
- ▶  $\text{REMOVE}(T, k)$ :  $\text{SPLAY}(T, k)$ , remove a raiz,  $T_L$  e  $T_R$  árvores resultantes,  $\text{SPLAY}(T_L, k)$ , e torne  $T_R$  a sub-árvore da nova raiz de  $T_L$ .



$\text{SPLAY}(T, k)$

# Remoção

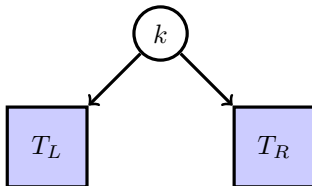
- ▶  $\text{REMOVE}(T, k)$ :  $\text{SPLAY}(T, k)$ , remove a raiz,  $T_L$  e  $T_R$  árvores resultantes,  $\text{SPLAY}(T_L, k)$ , e torne  $T_R$  a sub-árvore da nova raiz de  $T_L$ .





# Remoção

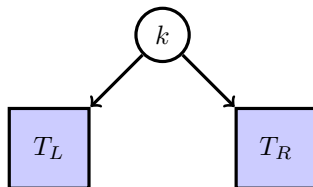
- ▶  $\text{REMOVE}(T, k)$ :  $\text{SPLAY}(T, k)$ , remove a raiz,  $T_L$  e  $T_R$  árvores resultantes,  $\text{SPLAY}(T_L, k)$ , e torne  $T_R$  a sub-árvore da nova raiz de  $T_L$ .



remove root:  $n = T.\text{root}$ ;  $T_L = n.\text{left}$ ;  $T_R = n.\text{right}$ ;  $\text{FREE}(n)$

# Remoção

- ▶  $\text{REMOVE}(T, k)$ :  $\text{SPLAY}(T, k)$ , remove a raiz,  $T_L$  e  $T_R$  árvores resultantes,  $\text{SPLAY}(T_L, k)$ , e torne  $T_R$  a sub-árvore da nova raiz de  $T_L$ .



remove root:  $n = T.\text{root}$ ;  $T_L = n.\text{left}$ ;  $T_R = n.\text{right}$ ;  $\text{FREE}(n)$



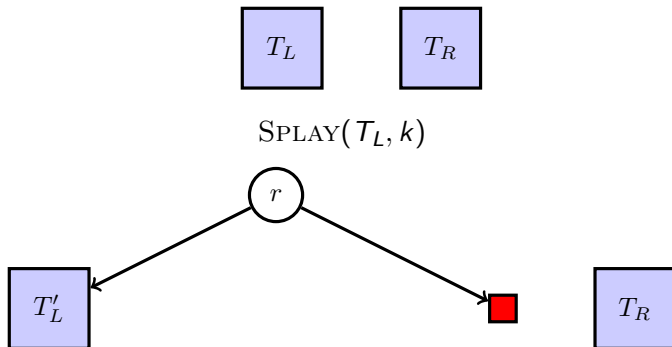
- ▶  $\text{REMOVE}(T, k)$ :  $\text{SPLAY}(T, k)$ , remove a raiz,  $T_L$  e  $T_R$  árvores resultantes,  $\text{SPLAY}(T_L, k)$ , e torne  $T_R$  a sub-árvore da nova raiz de  $T_L$ .



$\text{SPLAY}(T_L, k)$

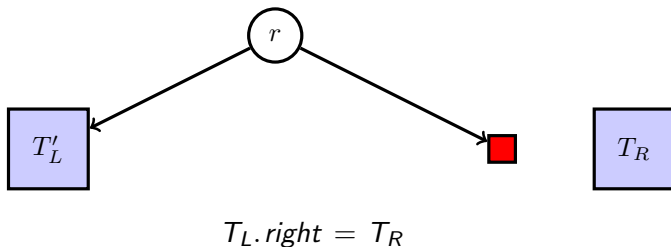
# Remoção

- ▶  $\text{REMOVE}(T, k)$ :  $\text{SPLAY}(T, k)$ , remove a raiz,  $T_L$  e  $T_R$  árvores resultantes,  $\text{SPLAY}(T_L, k)$ , e torne  $T_R$  a sub-árvore da nova raiz de  $T_L$ .



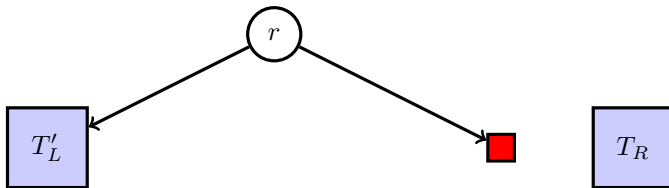
# Remoção

- ▶  $\text{REMOVE}(T, k)$ :  $\text{SPLAY}(T, k)$ , remove a raiz,  $T_L$  e  $T_R$  árvores resultantes,  $\text{SPLAY}(T_L, k)$ , e torne  $T_R$  a sub-árvore da nova raiz de  $T_L$ .

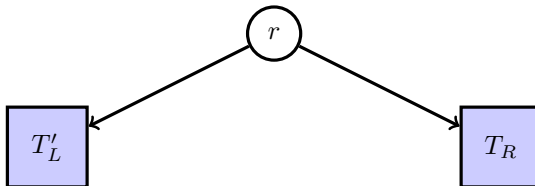


# Remoção

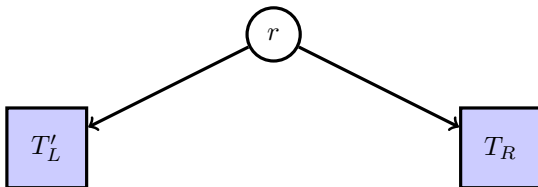
- $\text{REMOVE}(T, k)$ :  $\text{SPLAY}(T, k)$ , remove a raiz,  $T_L$  e  $T_R$  árvores resultantes,  $\text{SPLAY}(T_L, k)$ , e torne  $T_R$  a sub-árvore da nova raiz de  $T_L$ .



$$T_L.\text{right} = T_R$$



- ▶  $\text{REMOVE}(T, k)$ :  $\text{SPLAY}(T, k)$ , remove a raiz,  $T_L$  e  $T_R$  árvores resultantes,  $\text{SPLAY}(T_L, k)$ , e torne  $T_R$  a sub-árvore da nova raiz de  $T_L$ .



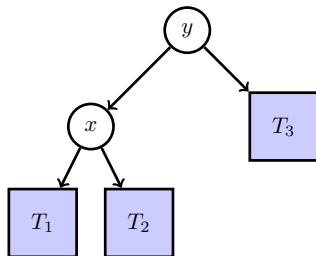
# Chanfrar um valor

- ▶ aplicar rotações até chegar à raiz
- ▶ se  $x$  for um filho da raiz: rotação simples (caso 1)
- ▶  $x.up = y$  e  $y.up = z$ 
  - ▶ se  $x = y.left$  e  $y = z.left$ : rotação simples de  $y$ , rotação simples de  $x$  (caso 2)
  - ▶ se  $x = y.right$  e  $y = z.right$ : rotação simples de  $y$ , rotação simples de  $x$  (caso 2)
  - ▶ se  $x = y.right$  e  $y = z.left$ : rotação dupla de  $x$  (caso 3)
  - ▶ se  $x = y.left$  e  $y = z.right$ : rotação dupla de  $x$  (caso 3)
- ▶ complexidade no pior caso  $O(n)$
- ▶ análise amortizada mostra que todas as operações de chanframento tem custo amortizado  $O(\log n)$

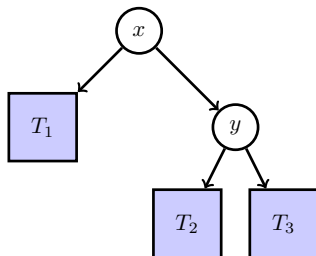


# Chanfrar um valor — caso 1

se  $x$  for um filho da raiz: rotação simples

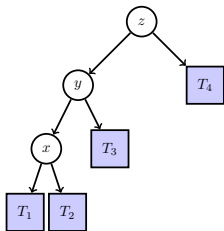


ROTATE-SIMPLE-RIGHT( $x, y$ )



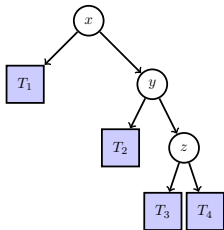
# Chanfrar um valor — caso 2

$x = y.\text{left}$  e  $y = z.\text{left}$  (e simétrico)



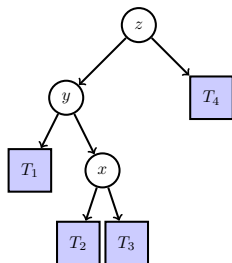
ROTATE-SIMPLE-RIGHT( $y, z$ )

ROTATE-SIMPLE-RIGHT( $x, y$ )

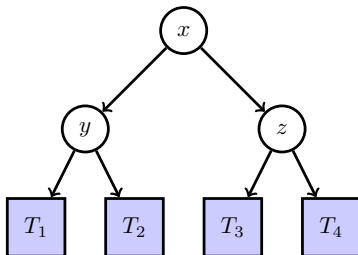


## Chanfrar um valor — caso 3

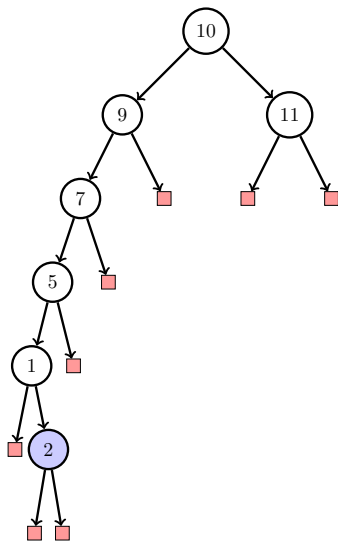
$x = y.right$  e  $y = z.left$



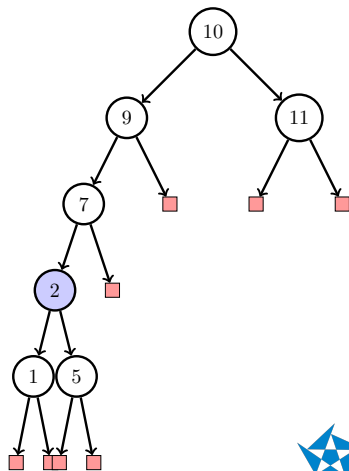
ROTATE-DOUBLE-RIGHT( $x, y, z$ )



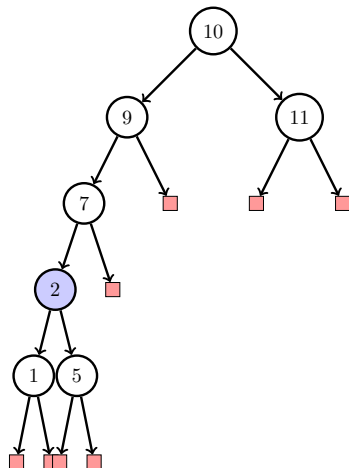
# Chanfrar um valor: exemplo



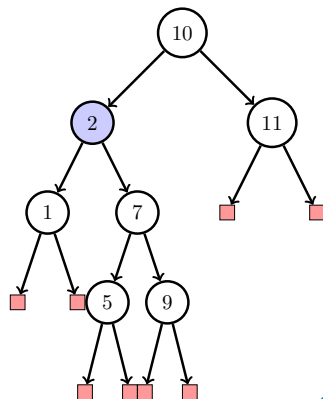
Caso 3



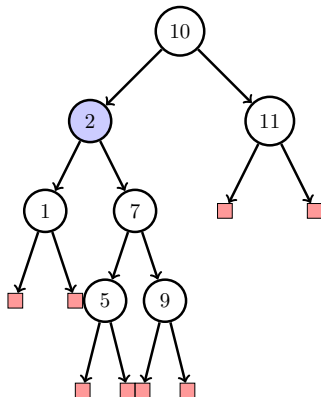
# Chanfrar um valor: exemplo



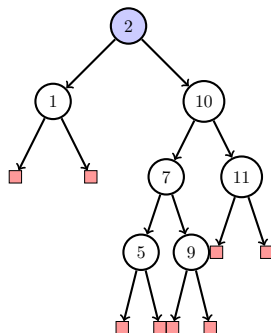
Caso 2



# Chanfrar um valor: exemplo



Caso 1



# Análise amortizada: método do contador

- ▶ os nós guardam créditos ( $\gamma(x)$ )
- ▶ créditos pagam para efetuar rotações
- ▶ o preço de SPLAY é  $O(\log n)$
- ▶ invariante:  $\gamma(x) = \lfloor \log |T(x)| \rfloor$   
 $T(x)$ : sub-árvore enraizada em  $x$
- ▶ propriedade:  $3 \times (\gamma(T) - \gamma(x) + O(1))$  créditos paga SPLAY( $T, x$ ) e mantém o invariante. (vamos mostrar isto depois)
- ▶ logo SPLAY custa no máximo  $3 \lfloor \log n \rfloor + O(1) \in \Theta(\log n)$ .
- ▶ para pagar inserção, busca ou remoção, basta pagar  $O(\log n)$ .



# Análise amortizada: método do contador

- ▶  $\gamma(x)$  e  $\gamma'(x)$ : quantidade de créditos em  $x$  antes e depois das rotações nos casos 1, 2 e 3
- ▶ SPLAY cai  $k \geq 0$  vezes nos casos 2 e 3 e possivelmente uma vez no caso 1.
- ▶ vamos mostrar que o custo em cada caso é
  - ▶ caso 1:  $3 \times (\gamma'(x) - \gamma(x)) + O(1)$
  - ▶ caso 2:  $3 \times (\gamma'(x) - \gamma(x))$
  - ▶ caso 3:  $3 \times (\gamma'(x) - \gamma(x))$
- ▶ somando, obtemos  $3(\gamma(T) - \gamma(x)) + O(1)$   
 $\gamma(x)$ : número de créditos em  $\gamma$  antes do SPLAY



# Caso 1

$y = x.up, y.up = \text{NIL}$

- ▶  $\gamma'(x) = \gamma(y), \gamma'(y) \leq \gamma'(x),$
- ▶ logo

$$\begin{aligned}\gamma'(x) + \gamma'(y) - \gamma(x) - \gamma(y) &= \gamma'(y) - \gamma(x) \\ &\leq \gamma'(x) - \gamma(x) \\ &\leq 3 \times (\gamma'(x) - \gamma(x))\end{aligned}$$

## Caso 2

$y = x.up, x = x.left, z = y.up, y = z.left$

- ▶ temos:  $\gamma'(x) = \gamma(z), \gamma'(y) \leq \gamma'(x), \gamma'(z) \leq \gamma'(x),$   
 $\gamma(y) \geq \gamma(x)$
- ▶ logo

$$\begin{aligned} & \gamma'(x) + \gamma'(y) + \gamma'(z) - \gamma(x) - \gamma(y) - \gamma(z) \\ &= \gamma'(y) + \gamma'(z) - \gamma(x) - \gamma(y) \\ &= (\gamma'(y) - \gamma(x)) + (\gamma'(z) - \gamma(y)) \\ &\leq (\gamma'(x) - \gamma(x)) + (\gamma'(x) - \gamma(x)) \\ &\leq 2 \times (\gamma'(x) - \gamma(x)) \end{aligned}$$

- ▶ resta  $\gamma'(x) - \gamma(x)$  créditos para efetuar a rotação.

## Caso 2 (continuação)

Mostramos que há créditos para a rotação

Resta  $\gamma'(x) - \gamma(x)$ :

- ▶ se  $\gamma(x) \neq \gamma'(x)$ , então resta  $\gamma'(x) - \gamma(x) > 0$  créditos para efetuar a rotação

## Caso 2 (continuação)

Mostramos que há créditos para a rotação

Resta  $\gamma'(x) - \gamma(x)$ :

- ▶ se  $\gamma(x) \neq \gamma'(x)$ , então resta  $\gamma'(x) - \gamma(x) > 0$  créditos para efetuar a rotação
- ▶ se  $\gamma(x) = \gamma'(x)$ , mostramos que  $\gamma'(x) + \gamma'(y) + \gamma'(z) < \gamma(x) + \gamma(y) + \gamma(z)$ .
- ▶ neste caso, é preciso menos créditos na árvore para manter o invariante
- ▶ este crédito é usado para efetuar a rotação
- ▶ roteiro da prova:
  - ▶ supor que  $\gamma'(x) + \gamma'(y) + \gamma'(z) \geq \gamma(x) + \gamma(y) + \gamma(z)$ .
  - ▶ chegar a uma contradição



## Caso 2 (continuação)

o caso 2 pode liberar créditos para a rotação

Supondo  $\gamma'(x) + \gamma'(y) + \gamma'(z) \geq \gamma(x) + \gamma(y) + \gamma(z)$ , temos

- ▶  $\gamma(x) \leq \gamma(y) \leq \gamma(z)$
- ▶  $\gamma'(z) \leq \gamma'(y) \leq \gamma'(x)$
- ▶  $\gamma'(x) = \gamma(z)$
- ▶ logo temos que  $\gamma(x) = \gamma(z) = \gamma(y)$
- ▶ e  $\gamma'(x) = \gamma'(y) = \gamma'(z)$
- ▶ pela definição de  $\gamma$ , temos
$$\lfloor \log_2(a + b + 1) \rfloor = \lfloor \log_2 a \rfloor = \lfloor \log_2 b \rfloor$$
  - ▶ se  $a \leq b$  então
$$\lfloor \log_2(a + b + 1) \rfloor \geq \lfloor \log_2 2a \rfloor = 1 + \lfloor \log_2 a \rfloor > \lfloor \log_2 a \rfloor$$
  - ▶ se  $a > b$  então
$$\lfloor \log_2(a + b + 1) \rfloor \geq \lfloor \log_2 2b \rfloor = 1 + \lfloor \log_2 b \rfloor > \lfloor \log_2 b \rfloor$$
- ▶ em ambos casos chegamos a uma contradição.

## Caso 3

- ▶ Análogo ao caso 2
- ▶ deixado como exercício

- ▶ Análise amortizada pode fornecer resultados mais apurados
- ▶ outro exemplo: árvores rubro-negras
  - ▶ inserção:  $O(\log n)$
  - ▶ mas o custo amortizado de  $m$  inserções em uma árvore de  $n$  nós é  $O(n + m)$