

Aula 22: Extensão de estruturas de dados

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

06 de maio de 2015



Introdução

O problema da seleção

Receita

Coleções de intervalos

Referência: Cormen, cap 15.

- ▶ Estruturas de dados prontas não resolvem todos os problemas práticos
- ▶ Discernimento para saber adaptar uma estrutura de dados ao problema
- ▶ Conservar as boas propriedades da estrutura de dados
 1. complexidade
 2. propriedades de ordenação

- ▶ Coleção dinâmica de dados ordenados
 - ▶ inserção
 - ▶ busca
 - ▶ remoção
- ▶ + seleção do dado na k -ésima posição da classificação
- ▶ + cálculo da classificação de um dado

- ▶ Coleção dinâmica de dados ordenados
 - ▶ inserção
 - ▶ busca
 - ▶ remoção
- ▶ + seleção do dado na k -ésima posição da classificação
- ▶ + cálculo da classificação de um dado

propostas?

1. arranjo
2. árvore balanceada

1. ordenado: acesso direto
 - ▶ inserção, remoção são ineficientes
2. não ordenado: algoritmo similar ao *quick sort*
 - ▶ busca, remoção são ineficientes

Árvore balanceada

1. árvore rubro-negra
2. associar a cada nó o tamanho da sub-árvore enraizada nele

SIZE(n)

```
1  if  $n == \text{NIL}$ 
2      return 0
3  else return  $n.size$ 
```


Seleção do dado na posição k

SELECT(n, k)

1 $sl = \text{SIZE}(n.\text{left})$

2 **if** $k \leq sl$

3 **return** SELECT($n.\text{left}, k$)

4 **elseif** $k == sl + 1$

5 **return** n

6 **else return** SELECT($n.\text{right}, k - 1 - sl$)

Seleção do dado na posição k

```
SELECT( $n, k$ )  
1   $sl = \text{SIZE}(n.\text{left})$   
2  if  $k \leq sl$   
3      return SELECT( $n.\text{left}, k$ )  
4  elseif  $k == sl + 1$   
5      return  $n$   
6  else return SELECT( $n.\text{right}, k - 1 - sl$ )
```

Complexidade?

Cálculo da posição de um dado x

$\text{RANK}(n, x)$

```
1  if  $x == n.\text{key}$   
2      return  $\text{SIZE}(n.\text{left}) + 1$   
3  elseif  $x < n.\text{key}$   
4      return  $\text{RANK}(n.\text{left}, x)$   
5  else return  $\text{SIZE}(n.\text{left}) + 1 + \text{RANK}(n.\text{right}, x)$ 
```



Cálculo da posição de um dado x

$\text{RANK}(n, x)$

```
1  if  $x == n.\text{key}$   
2      return  $\text{SIZE}(n.\text{left}) + 1$   
3  elseif  $x < n.\text{key}$   
4      return  $\text{RANK}(n.\text{left}, x)$   
5  else return  $\text{SIZE}(n.\text{left}) + 1 + \text{RANK}(n.\text{right}, x)$ 
```

Complexidade?



Como atualizar o atributo *size*?

- ▶ Invariant: $x.size = 1 + \text{SIZE}(x.left) + \text{SIZE}(x.right)$
- ▶ Inserção:
 1. descida recursiva na árvore até o ponto de inserção
 2. criação de um novo vértice
 3. subida até a raiz, com rotações
- ▶ Remoção:
 1. descida recursiva na árvore até o ponto de remoção
 2. possivelmente remoção auxiliar
 3. subida até a raiz, com rotações

Como atualizar o atributo *size*?

Inserção

- ▶ descida até o ponto de inserção: incremento de *size*

Como atualizar o atributo *size*?

Inserção

- ▶ descida até o ponto de inserção: incremento de *size*
- ▶ criação de um novo vértice

MAKE-NODE(x)

1 $n = \text{ALLOC-NODE}()$

2 $n.val = x$

3 $n.color = \text{RED}$

4 $n.size = 1$



Como atualizar o atributo *size*?

Inserção

- ▶ descida até o ponto de inserção: incremento de *size*
- ▶ criação de um novo vértice

MAKE-NODE(*x*)

1 *n* = ALLOC-NODE()

2 *n.val* = *x*

3 *n.color* = RED

4 *n.size* = 1

- ▶ subida até a raiz, com rotações

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

// *y.left* == *x* and *x.up* == *y*

1 *y.left* = *x.right*

2 *x.left* = *y*

3 *x.size* = *y.size*

4 *y.size* = SIZE(*y.left*) + SIZE(*y.right*) + 1



Como atualizar o atributo *size*?

Inserção

- ▶ descida até o ponto de inserção: incremento de *size*
- ▶ criação de um novo vértice

MAKE-NODE(x)

```
1   $n = \text{ALLOC-NODE}()$   
2   $n.val = x$   
3   $n.color = \text{RED}$   
4   $n.size = 1$ 
```

- ▶ subida até a raiz, com rotações

ROTATE-SIMPLE-RIGHT(x, y)

// $y.left == x$ and $x.up == y$

```
1   $y.left = x.right$   
2   $x.left = y$   
3   $x.size = y.size$   
4   $y.size = \text{SIZE}(y.left) + \text{SIZE}(y.right) + 1$ 
```

Complexidade?

Como atualizar o atributo *size*?

Remoção

- ▶ descida até o ponto de inserção: decremento de *size*
- ▶ possivelmente remoção auxiliar
- ▶ eliminação de um vértice
- ▶ número finito de rotações

Como atualizar o atributo *size*?

Remoção

- ▶ descida até o ponto de inserção: decremento de *size* $O(\log n)$
- ▶ possivelmente remoção auxiliar
- ▶ eliminação de um vértice $O(1)$
- ▶ número finito de rotações $O(1)$

Receita para estender uma estrutura de dados

1. escolha da estrutura mais adequada
2. identificação dos novos atributos
3. adaptar operações existentes
4. novas operações



Teorema

Seja a um novo atributo para uma árvore rubro-negra com n vértices.

Se, para qualquer nó x , $x.a$ pode ser calculado a partir de x , $x.left$ e $x.right$, então podemos atualizar os valores de a para todos os nós da árvore sem alterar a complexidade assintótica das operações básicas.

Extensão de árvores rubro-negra

Teorema

Seja a um novo atributo para uma árvore rubro-negra com n vértices.

Se, para qualquer nó x , $x.a$ pode ser calculado a partir de x , $x.left$ e $x.right$, então podemos atualizar os valores de a para todos os nós da árvore sem alterar a complexidade assintótica das operações básicas.

Justificativa:

- ▶ Mudar $x.a$ só implica em alterar $x.up.a$, etc. até $T.root.a$, ou seja $O(\log n)$ alterações.
- ▶ Rotações só implicam em alterar a em um número limitado de vértices.
- ▶ Cada alteração de a é $O(1)$.



Exemplo: coleção de intervalos

i : intervalo

- ▶ $i.low$: limite inferior
- ▶ $i.upp$: limite superior

comparação de intervalos

1. $i.low \leq i'.high$ e $i'.low \leq i.high$: i e i' tem sobreposição
2. $i.high < i'.low$: i antes de i'
3. $i'.low < i.high$: i depois de i'

Exemplo: operações

Operações:

- ▶ $\text{INTERVAL-INSERT}(T, x)$: x com $x.int$ um intervalo;
- ▶ $\text{INTERVAL-DELETE}(T, x)$: remove x ;
- ▶ $\text{INTERVAL-SEARCH}(T, i)$: retorna um elemento x de T tal que i e $x.int$ se sobrepõem.

Exemplo: operações

Operações:

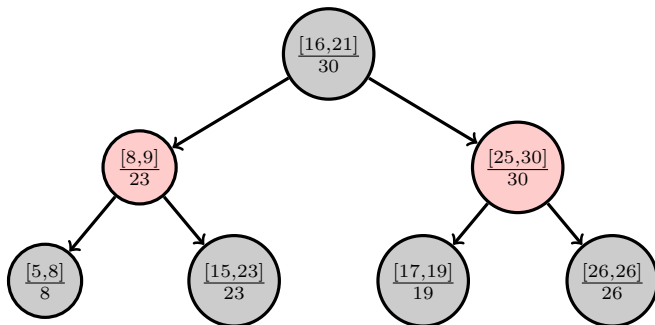
- ▶ $\text{INTERVAL-INSERT}(T, x)$: x com $x.int$ um intervalo;
- ▶ $\text{INTERVAL-DELETE}(T, x)$: remove x ;
- ▶ $\text{INTERVAL-SEARCH}(T, i)$: retorna um elemento x de T tal que i e $x.int$ se sobrepõem.

Árvores rubro-negras



Árvores rubro-negras extendidas para intervalos

- ▶ chave: $x.int.low$
- ▶ percurso em ordem: intervalos em ordem crescente de limite inferior
- ▶ atributo extra: $x.max$
 - ▶ valor máximo de qualquer intervalo na sub-árvore enraizada em x



Atualização de novos atributos

- ▶ $x.max = \max\{x.left.max, x.right.max, x.int.upp\}$
- ▶ complexidade: $O(1)$
- ▶ preserva complexidade assintótica das operações básicas (Teorema)

Novas operações

INTERVAL-SEARCH(T, i)

```
1   $x = T.root$ 
2  while  $x \neq \text{NIL}$  and not OVERLAP( $i, x.int$ )
3      if  $x.left \neq \text{NIL}$  and  $x.left.max \geq i.low$ 
4           $x = x.left$ 
5      else  $x = x.right$ 
6  return  $x$ 
```

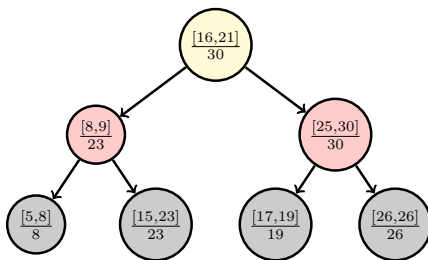
INTERVAL-SEARCH(T, i)

```
1   $x = T.root$ 
2  while  $x \neq \text{NIL}$  and not OVERLAP( $i, x.int$ )
3      if  $x.left \neq \text{NIL}$  and  $x.left.max \geq i$ 
4           $x = x.left$ 
5      else  $x = x.right$ 
6  return  $x$ 
```

INTERVAL-SEARCH(T, i)

```
1   $x = T.root$   
2  while  $x \neq \text{NIL}$  and not  $\text{OVERLAP}(i, x.int)$   
3      if  $x.left \neq \text{NIL}$  and  $x.left.max \geq i$   
4           $x = x.left$   
5      else  $x = x.right$   
6  return  $x$ 
```

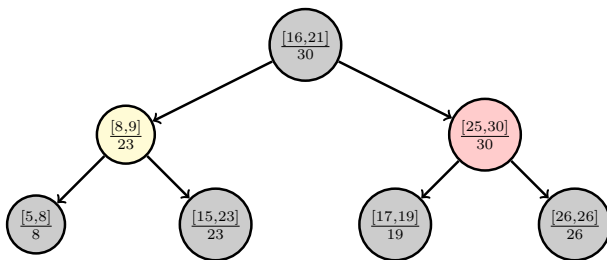
$i = [22, 25]$



INTERVAL-SEARCH(T, i)

```
1   $x = T.root$   
2  while  $x \neq \text{NIL}$  and not  $\text{OVERLAP}(i, x.int)$   
3      if  $x.left \neq \text{NIL}$  and  $x.left.max \geq i$   
4           $x = x.left$   
5      else  $x = x.right$   
6  return  $x$ 
```

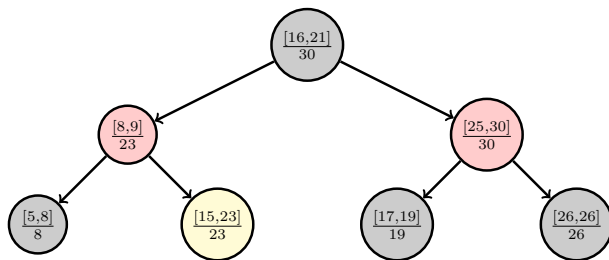
$i = [22, 25]$



INTERVAL-SEARCH(T, i)

```
1  $x = T.root$   
2 while  $x \neq \text{NIL}$  and not  $\text{OVERLAP}(i, x.int)$   
3   if  $x.left \neq \text{NIL}$  and  $x.left.max \geq i$   
4      $x = x.left$   
5   else  $x = x.right$   
6 return  $x$ 
```

$i = [22, 25]$



INTERVAL-SEARCH(T, i)

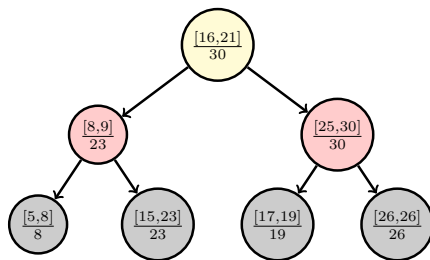
```
1   $x = T.root$ 
2  while  $x \neq \text{NIL}$  and not  $\text{OVERLAP}(i, x.int)$ 
3      if  $x.left \neq \text{NIL}$  and  $x.left.max \geq i$ 
4           $x = x.left$ 
5      else  $x = x.right$ 
6  return  $x$ 
```

$i = [22, 25]$

INTERVAL-SEARCH(T, i)

```
1   $x = T.root$   
2  while  $x \neq \text{NIL}$  and not  $\text{OVERLAP}(i, x.int)$   
3      if  $x.left \neq \text{NIL}$  and  $x.left.max \geq i$   
4           $x = x.left$   
5      else  $x = x.right$   
6  return  $x$ 
```

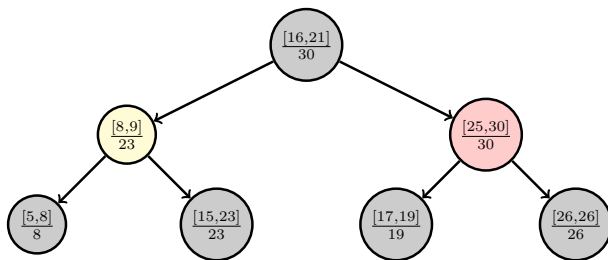
$i = [11, 14]$



INTERVAL-SEARCH(T, i)

```
1   $x = T.root$   
2  while  $x \neq \text{NIL}$  and not  $\text{OVERLAP}(i, x.int)$   
3      if  $x.left \neq \text{NIL}$  and  $x.left.max \geq i$   
4           $x = x.left$   
5      else  $x = x.right$   
6  return  $x$ 
```

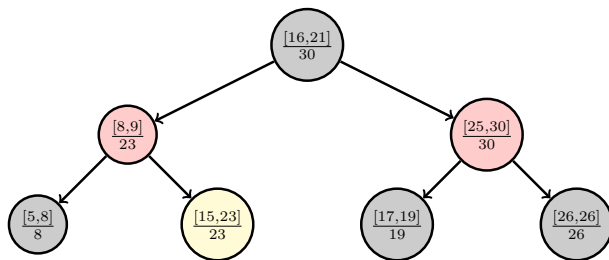
$i = [11, 14]$



INTERVAL-SEARCH(T, i)

```
1  $x = T.root$   
2 while  $x \neq \text{NIL}$  and not  $\text{OVERLAP}(i, x.int)$   
3   if  $x.left \neq \text{NIL}$  and  $x.left.max \geq i$   
4      $x = x.left$   
5   else  $x = x.right$   
6 return  $x$ 
```

$i = [11, 14]$



INTERVAL-SEARCH(T, i)

```
1   $x = T.root$ 
2  while  $x \neq \text{NIL}$  and not  $\text{OVERLAP}(i, x.int)$ 
3      if  $x.left \neq \text{NIL}$  and  $x.left.max \geq i$ 
4           $x = x.left$ 
5      else  $x = x.right$ 
6  return  $x$ 
```

Complexidade: $O(\log n)$

INTERVAL-SEARCH(T, i)

```
1   $x = T.root$ 
2  while  $x \neq \text{NIL}$  and not  $\text{OVERLAP}(i, x.int)$ 
3      if  $x.left \neq \text{NIL}$  and  $x.left.max \geq i$ 
4           $x = x.left$ 
5      else  $x = x.right$ 
6  return  $x$ 
```

Complexidade: $O(\log n)$ Corretude?

1. Adaptar a busca para funcionar com intervalos abertos
2. Escrever uma operação que
 - ▶ tem como parâmetro um intervalo i e,
 - ▶ retorna um intervalo que se sobrepõe a i , com o menor limite inferior, ou NIL se não existir.
3. Adapte a estrutura de dados união/fusão para ter a operação que retorna o número de elementos de um conjunto.