Aula 23: 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

Download me from `http://DavidDeharbe.github.io`

Plano

Introdução

O problema da seleção

Receita

Coleções de intervalos

Referência: Cormen, cap 15.

Motivação

- 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

PPgSC

Descrição

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

PPgSC

Descrição

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

propostas?

PPgSC

Propostas

1. arranjo
2. árvore balanceada

Arranjo

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

PPgSC

Árvore balanceada

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

```
SIZE(n)
1  if n == NIL
2        return 0
3  else return n.size
```

PPgSC

Seleção do dado na posição k

```
SELECT(n, k)
1   sl = SIZE(n.left)
2   if k ≤ sl
3       return SELECT(n.left, k)
4   elseif k == sl + 1
5       return n
6   else return SELECT(n.right, k − 1 − sl)
```

Seleção do dado na posição k

$\textsc{Select}(n, k)$

1. $sl = \textsc{Size}(n.left)$
2. **if** $k \leq sl$
3. **return** $\textsc{Select}(n.left, k)$
4. **elseif** $k == sl + 1$
5. **return** n
6. **else return** $\textsc{Select}(n.right, k - 1 - sl)$

Complexidade?

Cálculo da posição de um dado x

$\textsc{Rank}(n, x)$

1 **if** $x == n.key$
2   **return** $\textsc{Size}(n.left) + 1$
3 **elseif** $x < n.key$
4   **return** $\textsc{Rank}(n.left, x)$
5 **else return** $\textsc{Size}(n.left) + 1 + \textsc{Rank}(n.right, x)$

Cálculo da posição de um dado x

$\textsc{Rank}(n, x)$

1. **if** $x == n.key$
2. **return** $\textsc{Size}(n.left) + 1$
3. **elseif** $x < n.key$
4. **return** $\textsc{Rank}(n.left, x)$
5. **else return** $\textsc{Size}(n.left) + 1 + \textsc{Rank}(n.right, x)$

Complexidade?

Como atualizar o atributo *size*?

- Invariant: $x.\mathit{size} = 1 + \textsc{Size}(x.\mathit{left}) + \textsc{Size}(x.\mathit{right})$
- 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

PPgSC

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 = ALLOC-NODE()
2. $n.val = x$
3. $n.color$ = 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

$\textsc{Make-Node}(x)$

1. $n = \textsc{Alloc-Node}()$
2. $n.val = x$
3. $n.color = \textsc{Red}$
4. $n.size = 1$

- subida até a raiz, com rotações

$\textsc{Rotate-Simple-Right}(x, y)$

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

1. $y.left = x.right$
2. $x.left = y$
3. $x.size = y.size$
4. $y.size = \textsc{Size}(y.left) + \textsc{Size}(y.right) + 1$

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 = 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

Complexidade?

PPgSC

Exercício

- Escrever o algoritmo de rotação dupla com atualização do atributo *size*.

PPgSC

Como atualizar o atributo *size*?

Remoção

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

Receita para extender uma estrutura de dados

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

PPgSC

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.

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

PPgSC

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.

PPgSC

Exemplo: operações

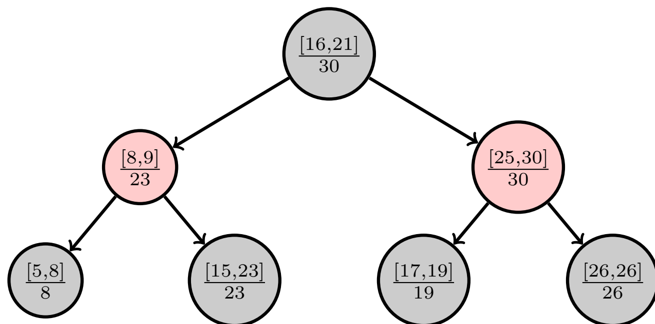
Operações:

- Interval-Insert(T, x): x com $x.int$ um intervalo;
- Interval-Delete(T, x): remove x;
- 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



PPgSC

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)

PPgSC

Novas operações

- INTERVAL-SEARCH(T, i)

Busca

```
INTERVAL-SEARCH(T, i)
1  x = T.root
2  while x ≠ NIL and not OVERLAP(i, x.int)
3      if x.left ≠ NIL and x.left.max ≥ i.low
4          x = x.left
5      else x = x.right
6  return x
```

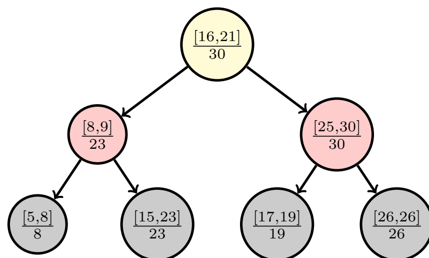
PPgSC

Busca

INTERVAL-SEARCH(T, i)

```
1  x = T.root
2  while x ≠ NIL and not OVERLAP(i, x.int)
3      if x.left ≠ NIL and x.left.max ≥ i.low
4          x = x.left
5      else x = x.right
6  return x
```

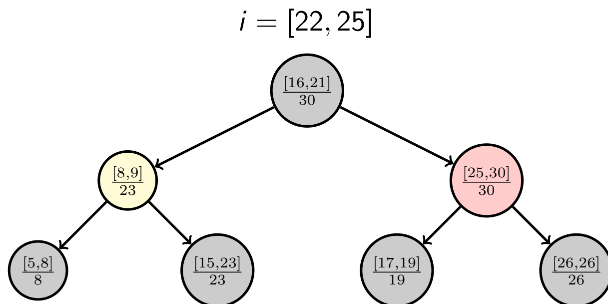
$i = [22, 25]$



Busca

INTERVAL-SEARCH(T, i)

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

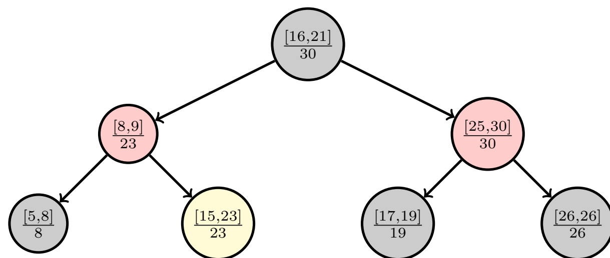


PPgSC

Busca

INTERVAL-SEARCH(T, i)

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



Busca

```
INTERVAL-SEARCH(T, i)
1  x = T.root
2  while x ≠ NIL and not OVERLAP(i, x.int)
3      if x.left ≠ NIL and x.left.max ≥ i.low
4          x = x.left
5      else x = x.right
6  return x
```

$$i = [22, 25]$$

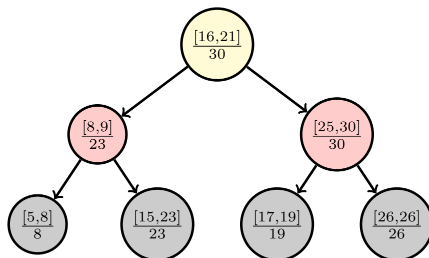
PPgSC

Busca

INTERVAL-SEARCH(T, i)

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

$i = [11, 14]$

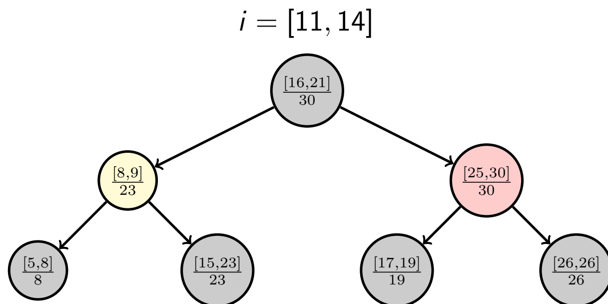


PPgSC

Busca

INTERVAL-SEARCH(T, i)

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



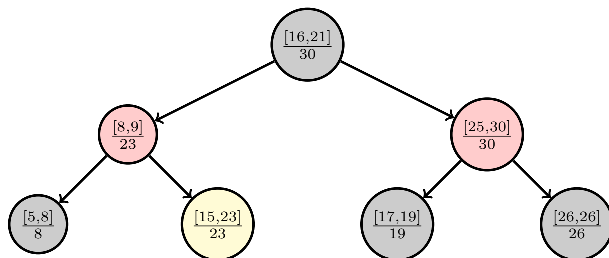
PPgSC

Busca

INTERVAL-SEARCH(T, i)

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

$i = [11, 14]$



PPgSC

Busca

```
INTERVAL-SEARCH(T, i)
1  x = T.root
2  while x ≠ NIL and not OVERLAP(i, x.int)
3      if x.left ≠ NIL and x.left.max ≥ i.low
4          x = x.left
5      else x = x.right
6  return x
```

Complexidade: $O(\log n)$

Busca

INTERVAL-SEARCH(T, i)

```
1  x = T.root
2  while x ≠ NIL and not OVERLAP(i, x.int)
3      if x.left ≠ NIL and x.left.max ≥ i.low
4          x = x.left
5      else x = x.right
6  return x
```

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

PPgSC

Exercícios

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/busca para ter a operação que retorna o número de elementos de um conjunto.

PPgSC