Estruturas de Dados Aumentantes

Rosemberg André da Silva – IC/UNICAMP

14 de setembro de 2007

Agenda

Agenda

- *(i)* Árvores Rubro-Negras

Agenda

- *(i)* Árvores Rubro-Negras
- *(ii)* Estatísticas de Ordem em Conjuntos Dinâmicos

Agenda

- *(i)* Árvores Rubro-Negras
- *(ii)* Estatísticas de Ordem em Conjuntos Dinâmicos
- *(iii)* Aumentando uma Estrutura de Dados

Agenda

- *(i)* Árvores Rubro-Negras
- *(ii)* Estatísticas de Ordem em Conjuntos Dinâmicos
- *(iii)* Aumentando uma Estrutura de Dados
- *(iv)* Árvores de Intervalo

Motivação

Motivação

Necessidades de Engenharia / Estruturas de dados:

Motivação

Necessidades de Engenharia / Estruturas de dados:

(a) Estruturas existentes.

Motivação

Necessidades de Engenharia / Estruturas de dados:

(a) Estruturas existentes.

(b) Estruturas novas.

Motivação

Necessidades de Engenharia / Estruturas de dados:

(a) Estruturas existentes.

(b) Estruturas novas.

(c) Modificações de estruturas existentes.

Árvores Rubro-Negras

Árvores Rubro-Negras

Definição 1: Árvore Rubro-Negra

Tipo de árvore binária com um bit extra de informação em cada nó para indicar sua cor: **preta** ou **vermelha**. Restringindo-se a seqüência de coloração dos nós, consegue-se garantir que tais árvores não possuam nenhum caminho da raiz às folhas que seja duas vezes mais longo que qualquer outro. Ou seja, a árvore é quase balanceada.

Árvores Rubro-Negras

Definição 1: Árvore Rubro-Negra

Tipo de árvore binária com um bit extra de informação em cada nó para indicar sua cor: **preta** ou **vermelha**. Restringindo-se a seqüência de coloração dos nós, consegue-se garantir que tais árvores não possuam nenhum caminho da raiz às folhas que seja duas vezes mais longo que qualquer outro. Ou seja, a árvore é quase balanceada.

Propriedades:

Árvores Rubro-Negras

Definição 1: Árvore Rubro-Negra

Tipo de árvore binária com um bit extra de informação em cada nó para indicar sua cor: **preta** ou **vermelha**. Restringindo-se a seqüência de coloração dos nós, consegue-se garantir que tais árvores não possuam nenhum caminho da raiz às folhas que seja duas vezes mais longo que qualquer outro. Ou seja, a árvore é quase balanceada.

Propriedades:

(a) Os nós são vermelhos ou pretos.

Árvores Rubro-Negras

Definição 1: Árvore Rubro-Negra

Tipo de árvore binária com um bit extra de informação em cada nó para indicar sua cor: **preta** ou **vermelha**. Restringindo-se a seqüência de coloração dos nós, consegue-se garantir que tais árvores não possuam nenhum caminho da raiz às folhas que seja duas vezes mais longo que qualquer outro. Ou seja, a árvore é quase balanceada.

Propriedades:

(a) Os nós são vermelhos ou pretos.

(b) A raiz é preta.

Árvores Rubro-Negras

Definição 1: Árvore Rubro-Negra

Tipo de árvore binária com um bit extra de informação em cada nó para indicar sua cor: **preta** ou **vermelha**. Restringindo-se a seqüência de coloração dos nós, consegue-se garantir que tais árvores não possuam nenhum caminho da raiz às folhas que seja duas vezes mais longo que qualquer outro. Ou seja, a árvore é quase balanceada.

Propriedades:

(a) Os nós são vermelhos ou pretos.

(b) A raiz é preta.

(c) Toda folha é preta.

Árvores Rubro-Negras

Definição 1: Árvore Rubro-Negra

Tipo de árvore binária com um bit extra de informação em cada nó para indicar sua cor: **preta** ou **vermelha**. Restringindo-se a seqüência de coloração dos nós, consegue-se garantir que tais árvores não possuam nenhum caminho da raiz às folhas que seja duas vezes mais longo que qualquer outro. Ou seja, a árvore é quase balanceada.

Propriedades:

(a) Os nós são vermelhos ou pretos.

(b) A raiz é preta.

(c) Toda folha é preta.

(d) Se um nó for vermelho, seus filhos serão pretos.

Árvores Rubro-Negras

Definição 1: Árvore Rubro-Negra

Tipo de árvore binária com um bit extra de informação em cada nó para indicar sua cor: **preta** ou **vermelha**. Restringindo-se a seqüência de coloração dos nós, consegue-se garantir que tais árvores não possuam nenhum caminho da raiz às folhas que seja duas vezes mais longo que qualquer outro. Ou seja, a árvore é quase balanceada.

Propriedades:

(a) Os nós são vermelhos ou pretos.

(b) A raiz é preta.

(c) Toda folha é preta.

(d) Se um nó for vermelho, seus filhos serão pretos.

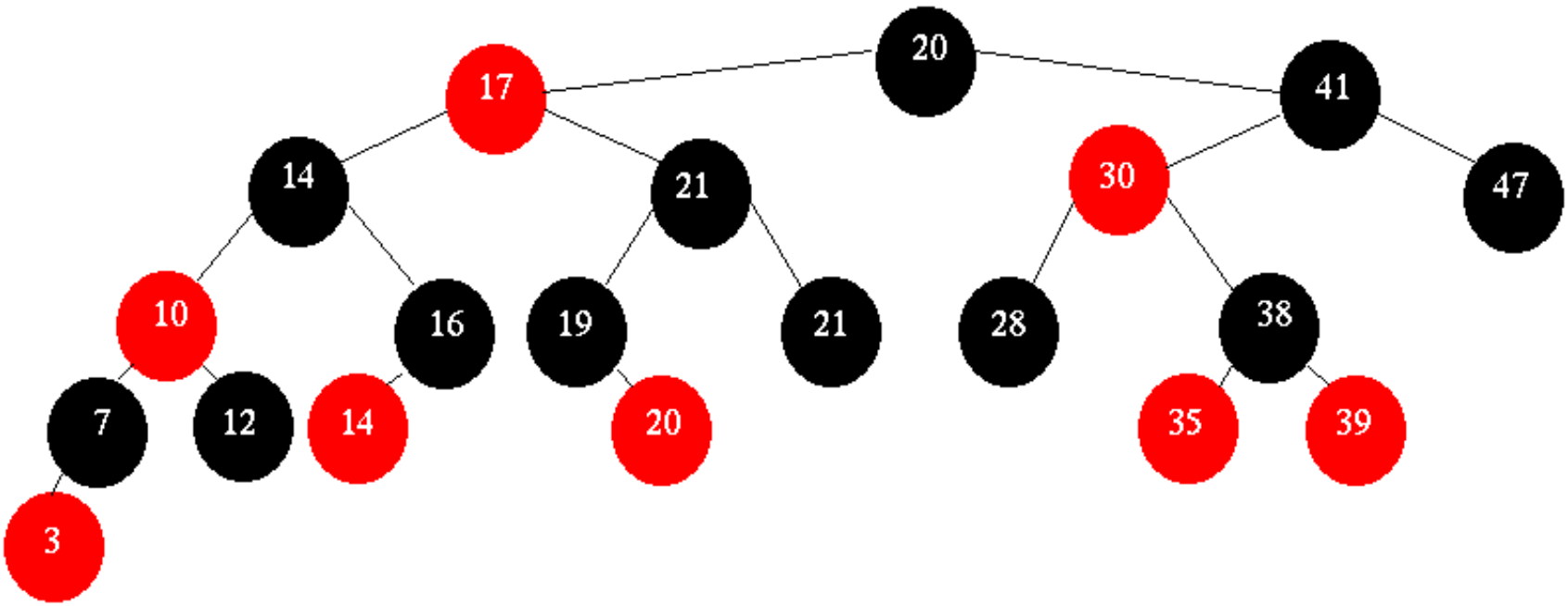
(e) Todos os caminhos de um nó até as folhas descendentes contêm o mesmo número de nós pretos.

Exemplo de Árvore Rubro-Negra

Exemplo de Árvore Rubro-Negra

Lema 1: Altura da Árvore Rubro-Negra

Uma árvore rubro-negra com n nós internos tem uma altura no máximo igual a $2log(n+1)$. Demonstração está descrita detalhadamente em [Cormen 2003], seção 13.1.



Operações numa Árvore Rubro-Negra

Operações numa Árvore Rubro-Negra

- *(i)* Rotações

Operações numa Árvore Rubro-Negra

- *(i)* Rotações
- *(ii)* Inserções

Operações numa Árvore Rubro-Negra

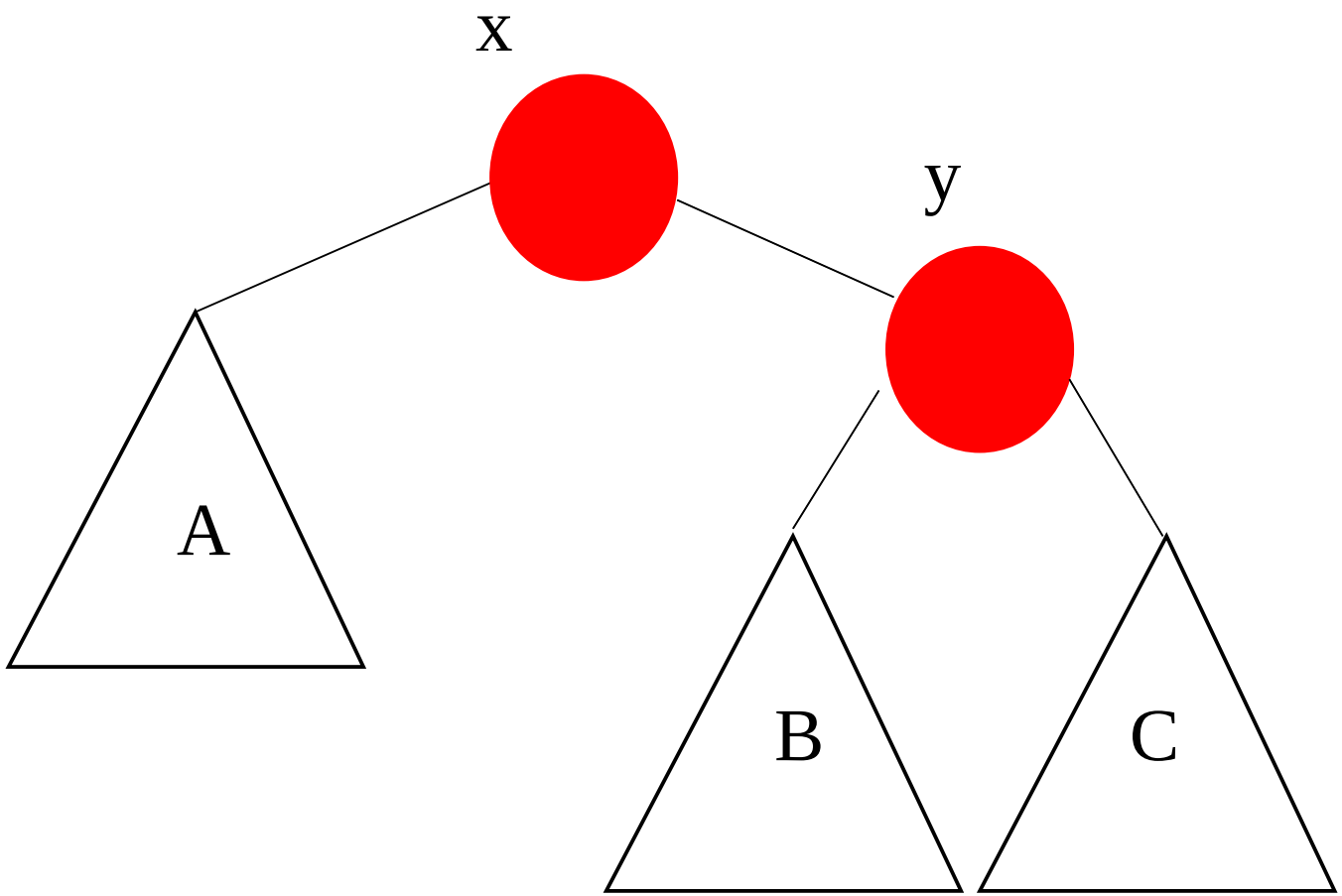
- *(i)* Rotações
- *(ii)* Inserções
- *(iv)* Remoções

Rotação em Árvores Rubro-Negras

Rotação em Árvores Rubro-Negras

Rotações

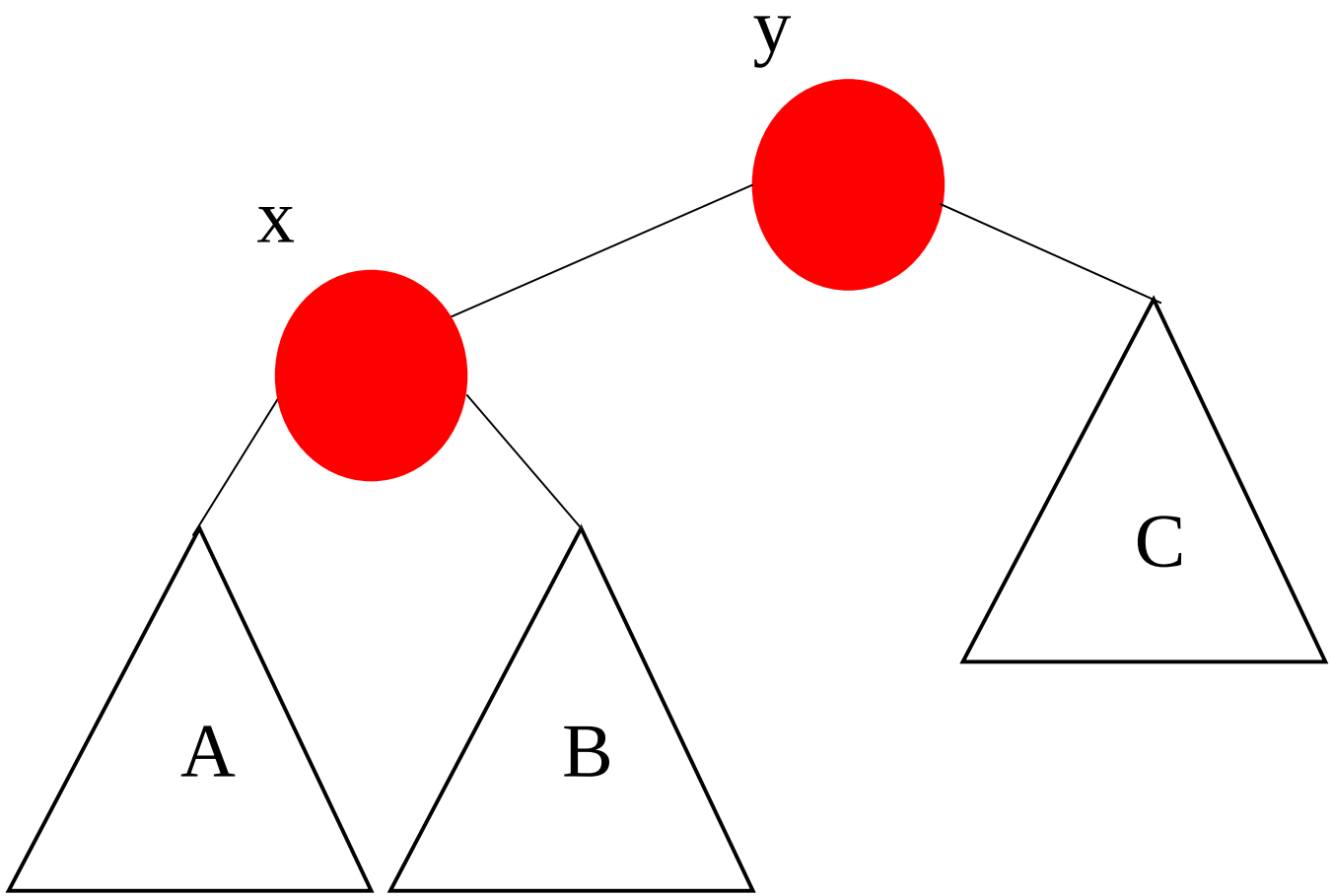
Mudanças nas estruturas de ponteiros com o intuito de recuperar as propriedades das árvores rubro-negras que eventualmente tenham sido violadas por uma inserção ou uma remoção.



Rotação em Árvores Rubro-Negras

Rotações: para a esquerda

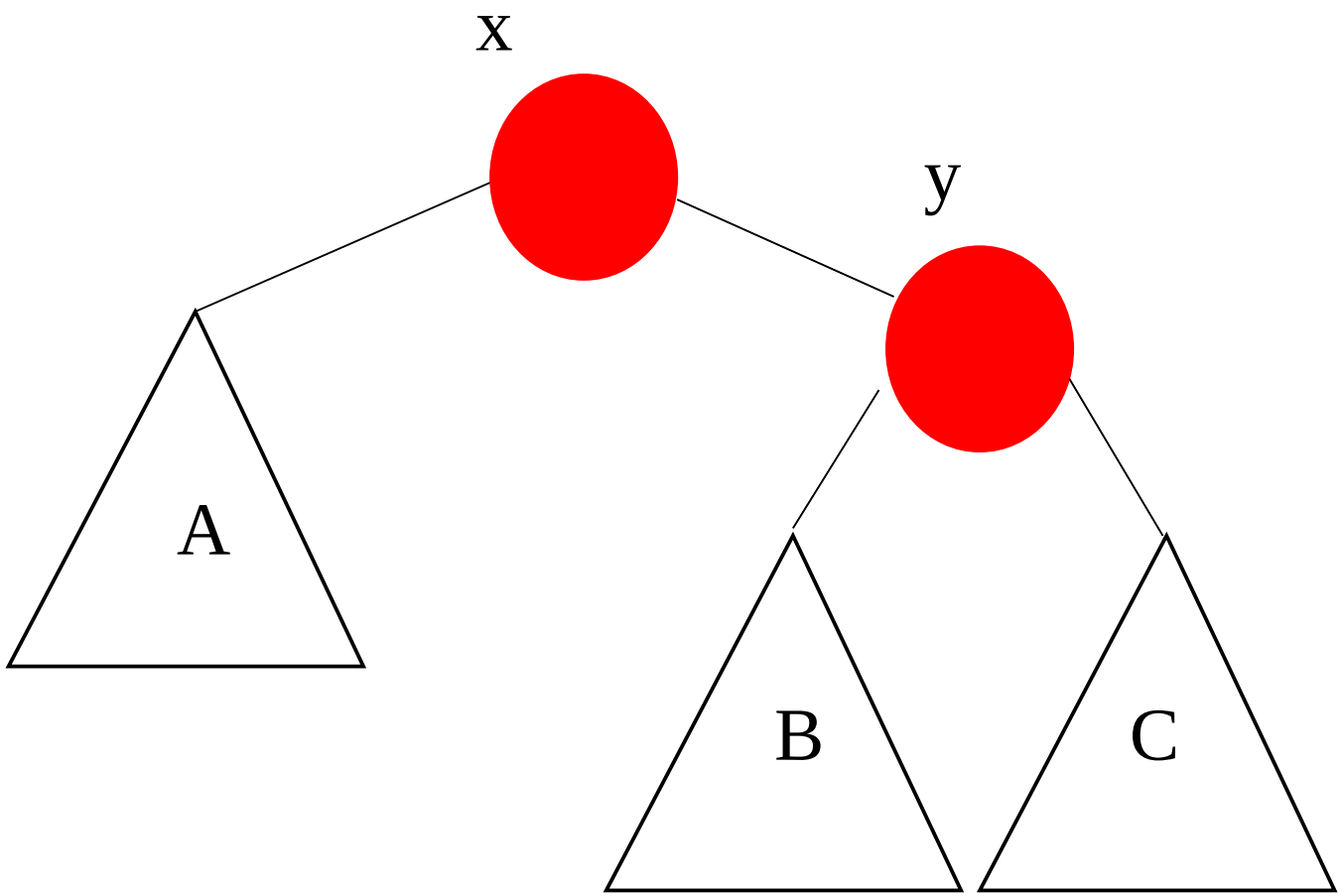
Mudanças nas estruturas de ponteiros com o intuito de recuperar as propriedades das árvores rubro-negras que eventualmente tenham sido violadas por uma inserção ou uma remoção.



Rotação em Árvores Rubro-Negras

Rotações: para a direita

Mudanças nas estruturas de ponteiros com o intuito de recuperar as propriedades das árvores rubro-negras que eventualmente tenham sido violadas por uma inserção ou uma remoção.

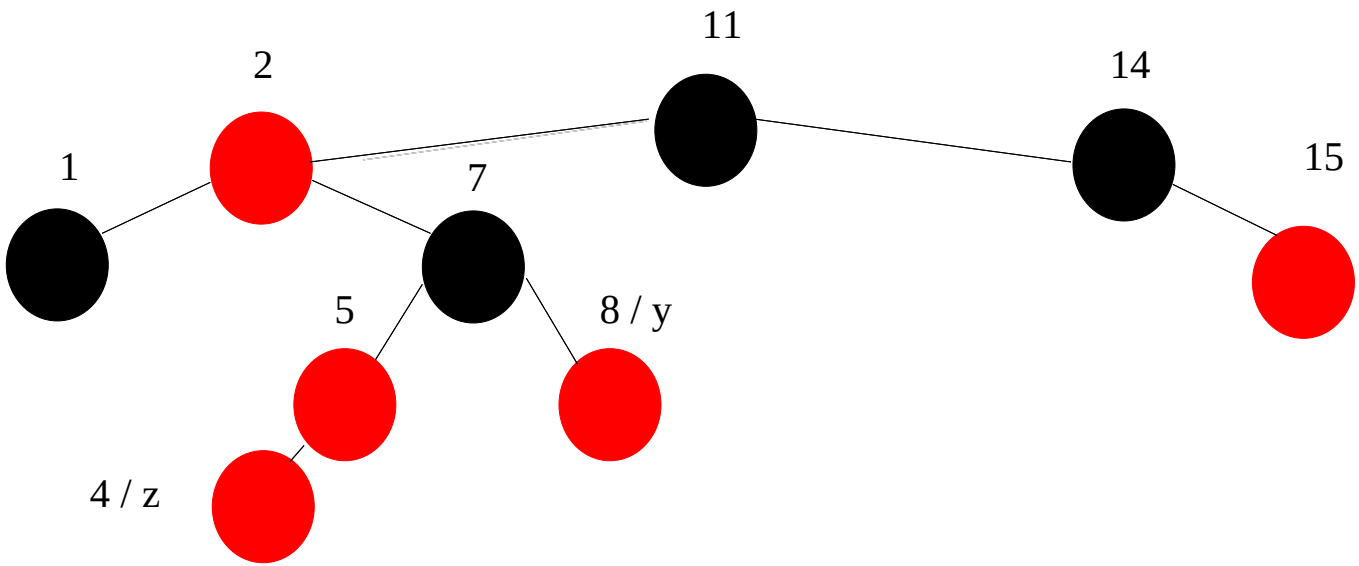


Inserção de Nós em Árvores Rubro-Negras

Inserção de Nós em Árvores Rubro-Negras

Inserções

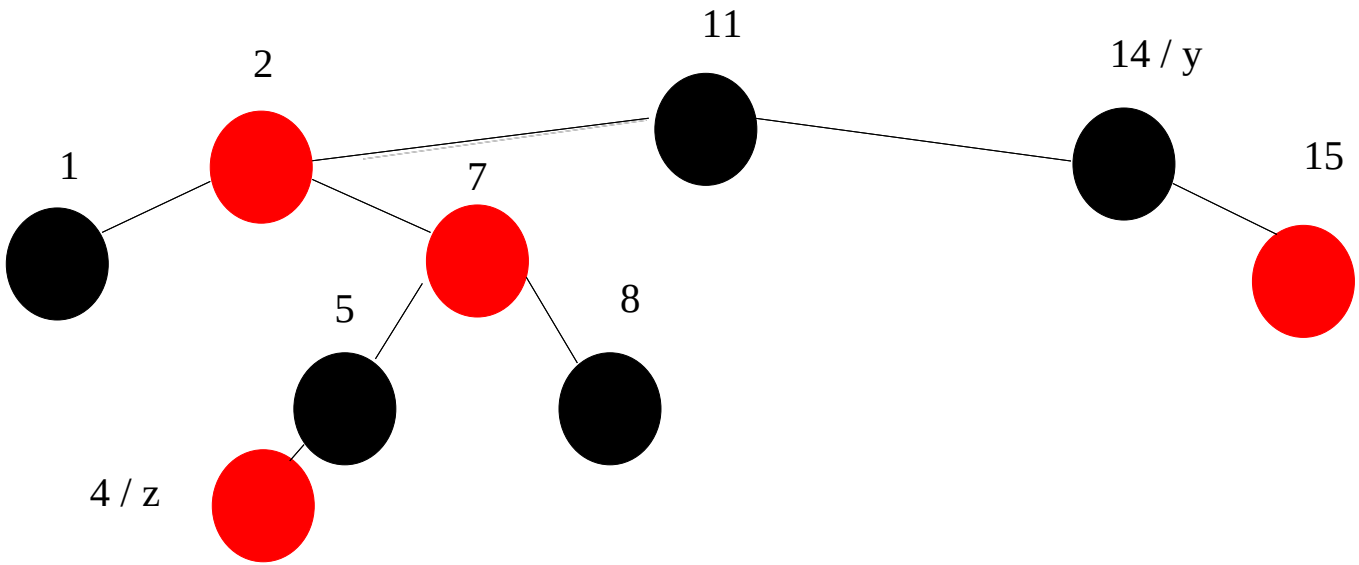
Adição de nós a uma árvore de tamanho n em tempo $O(log(n))$ seguida de ajustes que garantam a manutenção das propriedades de uma árvore rubro-negra.



Inserção de Nós em Árvores Rubro-Negras

Inserções

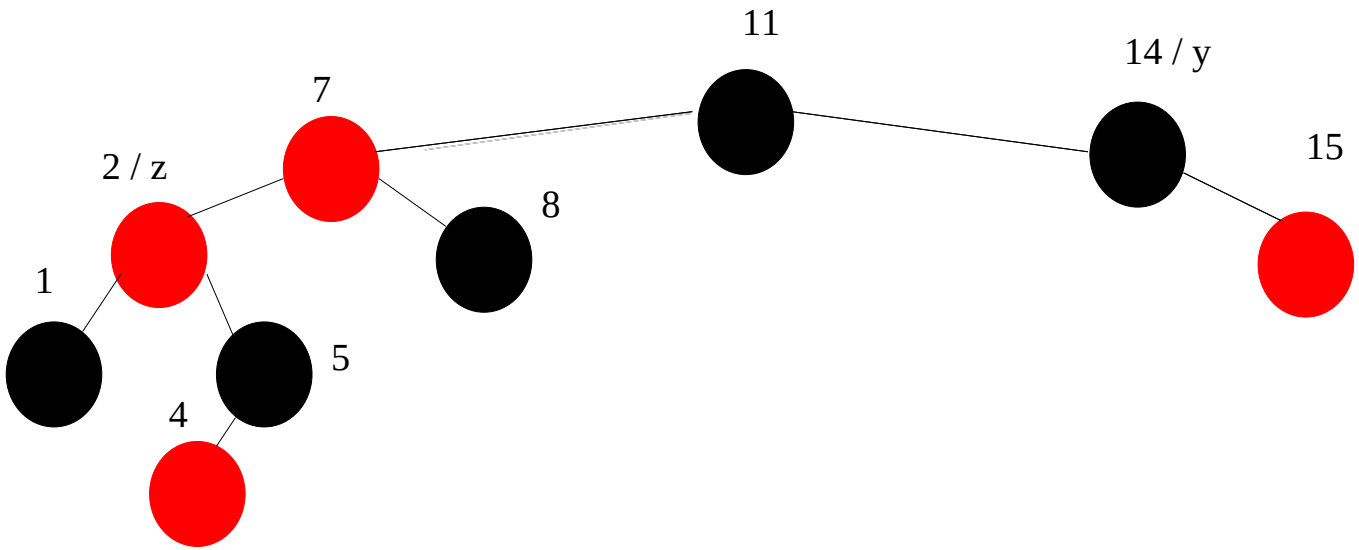
Adição de nós a uma árvore de tamanho n em tempo $O(log(n))$ seguida de ajustes que garantam a manutenção das propriedades de uma árvore rubro-negra.



Inserção de Nós em Árvores Rubro-Negras

Inserções

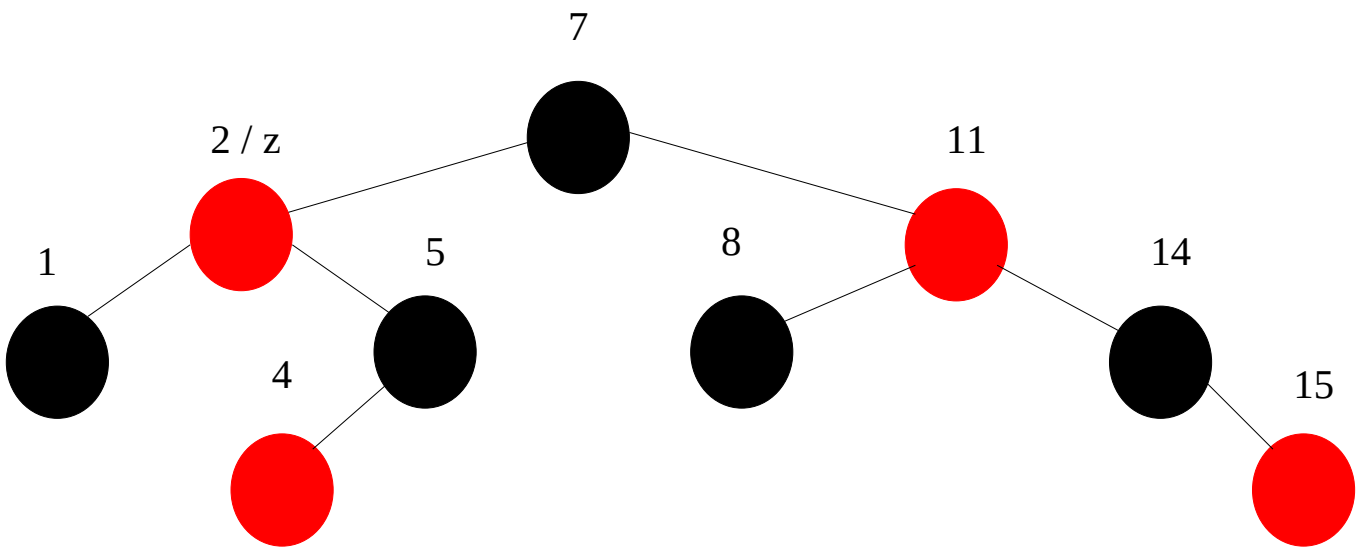
Adição de nós a uma árvore de tamanho n em tempo $O(log(n))$ seguida de ajustes que garantam a manutenção das propriedades de uma árvore rubro-negra.



Inserção de Nós em Árvores Rubro-Negras

Inserções

Adição de nós a uma árvore de tamanho n em tempo $O(log(n))$ seguida de ajustes que garantam a manutenção das propriedades de uma árvore rubro-negra.



Remoção de Nós em Árvores Rubro-Negras

Remoção de Nós em Árvores Rubro-Negras

Remoções

Remoção de nós a uma árvore de tamanho n em tempo $O(log(n))$ seguida de ajustes que garantam a manutenção das propriedades de uma árvore rubro-negra.

Remoção de Nós em Árvores Rubro-Negras

Remoções

Remoção de nós a uma árvore de tamanho n em tempo $O(log(n))$ seguida de ajustes que garantam a manutenção das propriedades de uma árvore rubro-negra.

Nota

De forma semelhante ao que ocorreu com a inserção, a remoção de nós da árvore pode levar a ajustes através de rotações. Entretanto, o tempo assintótico $O(log(n))$ é mantido.

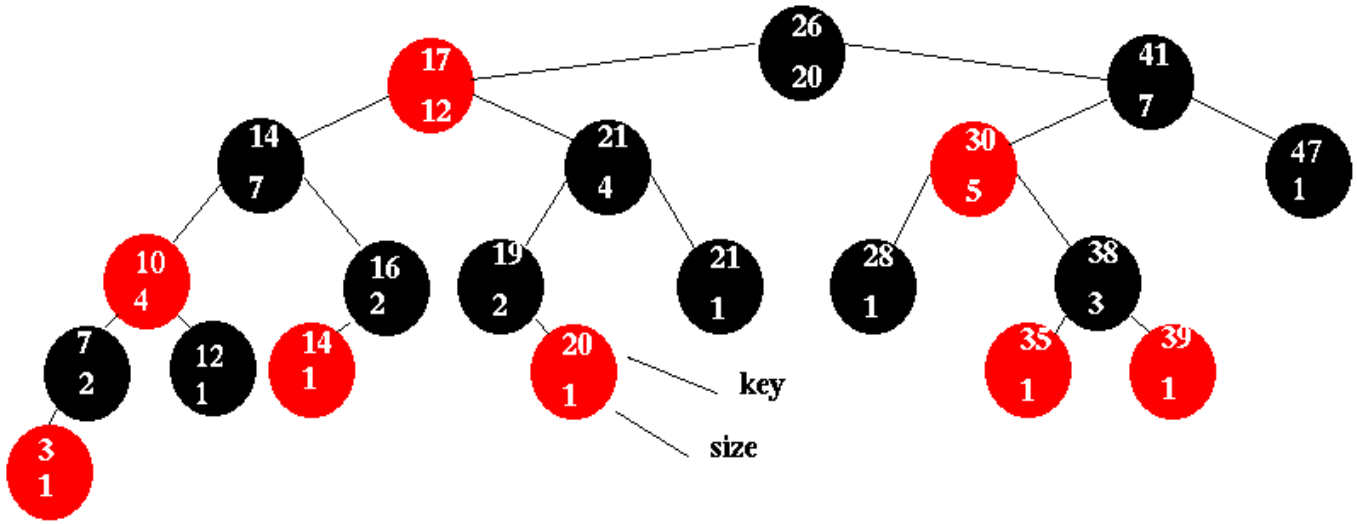
Agenda

- *(i)* Árvores Rubro-Negras
- *(ii)* Estatísticas de Ordem em Conjuntos Dinâmicos
- *(iii)* Aumentando uma Estrutura de Dados
- *(iv)* Árvores de Intervalo

Estatísticas de Ordem em Conjuntos Dinâmicos

i-ésimo menor elemento

Estatística de ordem i, com $i \in \{1, 2, ..., n\}$, de um dado conjunto de n elementos consiste no elemento com a i-ésima menor chave. Árvores rubro-negras possibilitam este cálculo em tempo $O(log(n))$. Na árvore ilustrada abaixo foram acrescentados aos campos usuais $key[x], color[x], p[x], left[x]$ e $right[x]$ o campo $size[x]$. A seguinte restrição deve ser observada:
$size[x] = size[left[x]] + size[right[x]] + 1$



Estatísticas de Ordem em Conjuntos Dinâmicos

Obtenção de um elemento de uma dada ordem

O algoritmo $OS - SELECT(x, i)$ implementa uma pesquisa que retorna a chave do i-ésimo menor elemento da sub-árvore que possui x como raiz.

OS-SELECT(x, i)

```
1  r ← size[left[x]] + 1
2  if i = r
3      then return x
4  elseif i < r
5      then return OS-SELECT(lelft[x], i)
6  else return Os-SELECT(right[x], i − r)
```

Estatísticas de Ordem em Conjuntos Dinâmicos

Determinação da ordem de um dado elemento

O algoritmo $OS - RANK(T, x)$ implementa uma pesquisa que retorna a posição de x na ordem linear ao se percorrer a árvore T em ordem.

OS-RANK(T, x)

```
1  r ← size[left[x]] + 1
2  y ← x
3  while y ≠ root[T]
4        do if y = right[p(y)]
5              then r ← r + size[left[p[y]]] + 1
6           y ← p[y]
7  return r
```

Estatísticas de Ordem em Conjuntos Dinâmicos

Estatísticas de Ordem em Conjuntos Dinâmicos

Manutenção dos tamanhos das sub-árvores

Valores de *size* de cada nó da árvore precisam ser eficientemente mantidos para que $OS - RANK(x, i)$ e $OS - SELECT(x, i)$ mantenham o comportamento assintótico $O(log(n))$.

Estatísticas de Ordem em Conjuntos Dinâmicos

Manutenção dos tamanhos das sub-árvores

Valores de *size* de cada nó da árvore precisam ser eficientemente mantidos para que $OS - RANK(x, i)$ e $OS - SELECT(x, i)$ mantenham o comportamento assintótico $O(log(n))$.

Fases da inserção:

Estatísticas de Ordem em Conjuntos Dinâmicos

Manutenção dos tamanhos das sub-árvores

Valores de *size* de cada nó da árvore precisam ser eficientemente mantidos para que $OS - RANK(x, i)$ e $OS - SELECT(x, i)$ mantenham o comportamento assintótico $O(log(n))$.

Fases da inserção:

(a) Adicionar novo nó como filho de um já existente (sentido descendente).

Estatísticas de Ordem em Conjuntos Dinâmicos

Manutenção dos tamanhos das sub-árvores

Valores de *size* de cada nó da árvore precisam ser eficientemente mantidos para que $OS - RANK(x, i)$ e $OS - SELECT(x, i)$ mantenham o comportamento assintótico $O(log(n))$.

Fases da inserção:

(a) Adicionar novo nó como filho de um já existente (sentido descendente).

(b) Alterações de cores para manter propriedades da árvore (sentido ascendente).

Estatísticas de Ordem em Conjuntos Dinâmicos

Manutenção dos tamanhos das sub-árvores

Valores de *size* de cada nó da árvore precisam ser eficientemente mantidos para que $OS - RANK(x, i)$ e $OS - SELECT(x, i)$ mantenham o comportamento assintótico $O(log(n))$.

Fases da inserção:

(a) Adicionar novo nó como filho de um já existente (sentido descendente).

(b) Alterações de cores para manter propriedades da árvore (sentido ascendente).

(c) Rotações para manter propriedades da árvore.

Estatísticas de Ordem em Conjuntos Dinâmicos

Manutenção dos tamanhos das sub-árvores

Valores de *size* de cada nó da árvore precisam ser eficientemente mantidos para que $OS - RANK(x, i)$ e $OS - SELECT(x, i)$ mantenham o comportamento assintótico $O(log(n))$.

Fases da inserção:

(a) Adicionar novo nó como filho de um já existente (sentido descendente).

(b) Alterações de cores para manter propriedades da árvore (sentido ascendente).

(c) Rotações para manter propriedades da árvore.

Fases da remoção:

Estatísticas de Ordem em Conjuntos Dinâmicos

Manutenção dos tamanhos das sub-árvores

Valores de *size* de cada nó da árvore precisam ser eficientemente mantidos para que $OS - RANK(x, i)$ e $OS - SELECT(x, i)$ mantenham o comportamento assintótico $O(log(n))$.

Fases da inserção:

(a) Adicionar novo nó como filho de um já existente (sentido descendente).

(b) Alterações de cores para manter propriedades da árvore (sentido ascendente).

(c) Rotações para manter propriedades da árvore.

Fases da remoção:

(a) Remoção do nó.

Estatísticas de Ordem em Conjuntos Dinâmicos

Manutenção dos tamanhos das sub-árvores

Valores de *size* de cada nó da árvore precisam ser eficientemente mantidos para que $OS - RANK(x, i)$ e $OS - SELECT(x, i)$ mantenham o comportamento assintótico $O(log(n))$.

Fases da inserção:

(a) Adicionar novo nó como filho de um já existente (sentido descendente).

(b) Alterações de cores para manter propriedades da árvore (sentido ascendente).

(c) Rotações para manter propriedades da árvore.

Fases da remoção:

(a) Remoção do nó.

(b) Rotações

Agenda

- *(i)* Árvores Rubro-Negras
- *(ii)* Estatísticas de Ordem em Conjuntos Dinâmicos
- *(iii)* Aumentando uma Estrutura de Dados
- *(iv)* Árvores de Intervalo

Aumentando uma Estrutura de Dados

Aumentando uma Estrutura de Dados

Passos envolvidos:

Aumentando uma Estrutura de Dados

Passos envolvidos:

(a) Escolha da estrutura de dados subjacente.

Aumentando uma Estrutura de Dados

Passos envolvidos:

(a) Escolha da estrutura de dados subjacente.

(b) Determinar informação adicional a ser mantida.

Aumentando uma Estrutura de Dados

Passos envolvidos:

(a) Escolha da estrutura de dados subjacente.

(b) Determinar informação adicional a ser mantida.

(c) Verificar se manutenção da informação adicional pode ser feita pelas operações de modificação da estrutura de dados.

Aumentando uma Estrutura de Dados

Passos envolvidos:

(a) Escolha da estrutura de dados subjacente.

(b) Determinar informação adicional a ser mantida.

(c) Verificar se manutenção da informação adicional pode ser feita pelas operações de modificação da estrutura de dados.

(d) Desenvolver novas operações.

Aumentando uma Estrutura de Dados

Passos envolvidos:

(a) Escolha da estrutura de dados subjacente.

(b) Determinar informação adicional a ser mantida.

(c) Verificar se manutenção da informação adicional pode ser feita pelas operações de modificação da estrutura de dados.

(d) Desenvolver novas operações.

Teorema 1: Aumentando uma Árvore Rubro-Negra

Seja f um campo que aumenta uma árvore rubro-negra T de n nós tal que $f(x)$ possa ser calculado a partir de x, $left[x]$, $right[x]$, $f[left[x]]$ e $f[right[x]]$. Então, podemos manter os valores de f para todos nos nós de T durante inserção e remoção sem afetar o comportamento assintótico $O(log(n))$ destas operações. Demonstração está descrita detalhadamente em [Cormen 2003], seção 14.1.

Aumentando uma Estrutura de Dados

Aumentando uma Estrutura de Dados

Passos da Demonstração do Teorema

Aumentando uma Estrutura de Dados

Passos da Demonstração do Teorema

(a) Mostrar que mudanças em f propagam-se para ancestrais.

Aumentando uma Estrutura de Dados

Passos da Demonstração do Teorema

(a) Mostrar que mudanças em f propagam-se para ancestrais.

(b) Altura limita o número de mudanças propagadas.

Aumentando uma Estrutura de Dados

Passos da Demonstração do Teorema

(a) Mostrar que mudanças em f propagam-se para ancestrais.

(b) Altura limita o número de mudanças propagadas.

(c) Na inserção, f é calculado para o elemento inserido e propagado até a raiz. No máximo duas rotações ocorrerão.

Aumentando uma Estrutura de Dados

Passos da Demonstração do Teorema

(a) Mostrar que mudanças em f propagam-se para ancestrais.

(b) Altura limita o número de mudanças propagadas.

(c) Na inserção, f é calculado para o elemento inserido e propagado até a raiz. No máximo duas rotações ocorrerão.

(d) Na remoção, mudanças ocorrerão se o nó removido for substituído por seu sucessor. A propagação destas mudanças é limitada pela altura da árvore. No máximo três rotações ocorrerão.

Agenda

- *(i)* Árvores Rubro-Negras
- *(ii)* Estatísticas de Ordem em Conjuntos Dinâmicos
- *(iii)* Aumentando uma Estrutura de Dados
- *(iv)* Árvores de Intervalo

Árvores de Intervalo

Árvores de Intervalo

Definição 2: Árvore de Intervalos

Tipo de árvore rubro-negra que armazena um conjunto dinâmico de elementos, com cada nó contendo um intervalo $int[x]$.

Árvores de Intervalo

Definição 2: Árvore de Intervalos

Tipo de árvore rubro-negra que armazena um conjunto dinâmico de elementos, com cada nó contendo um intervalo $int[x]$.

Operações Suportadas:

Árvores de Intervalo

Definição 2: Árvore de Intervalos

Tipo de árvore rubro-negra que armazena um conjunto dinâmico de elementos, com cada nó contendo um intervalo $int[x]$.

Operações Suportadas:

(a) *InserirIntervalo*(T, x): adiciona à árvore o elemento x cujo campo *int* contém o intervalo em questão.

Árvores de Intervalo

Definição 2: Árvore de Intervalos

Tipo de árvore rubro-negra que armazena um conjunto dinâmico de elementos, com cada nó contendo um intervalo $int[x]$.

Operações Suportadas:

(a) *InserirIntervalo*(T, x): adiciona à árvore o elemento x cujo campo *int* contém o intervalo em questão.

(b) *RemoverIntervalo*(T, x): remove o elemento x da árvore.

Árvores de Intervalo

Definição 2: Árvore de Intervalos

Tipo de árvore rubro-negra que armazena um conjunto dinâmico de elementos, com cada nó contendo um intervalo $int[x]$.

Operações Suportadas:

(a) *InserirIntervalo*(T, x): adiciona à árvore o elemento x cujo campo *int* contém o intervalo em questão.

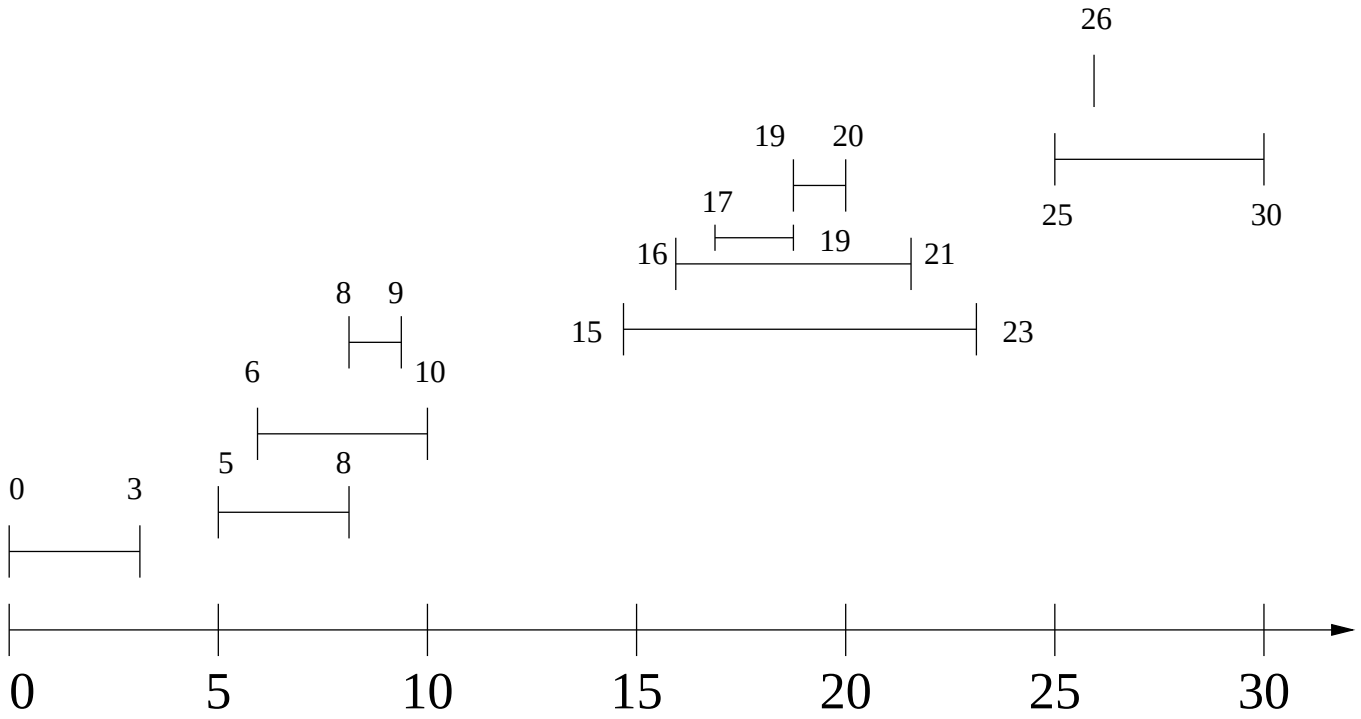
(b) *RemoverIntervalo*(T, x): remove o elemento x da árvore.

(c) *BuscarIntervalo*(T, i): retorna um ponteiro para um elemento x na árvore tal que $int[x]$ sobreponha o intervalo procurado

Árvores de Intervalo

Representação de Intervalos

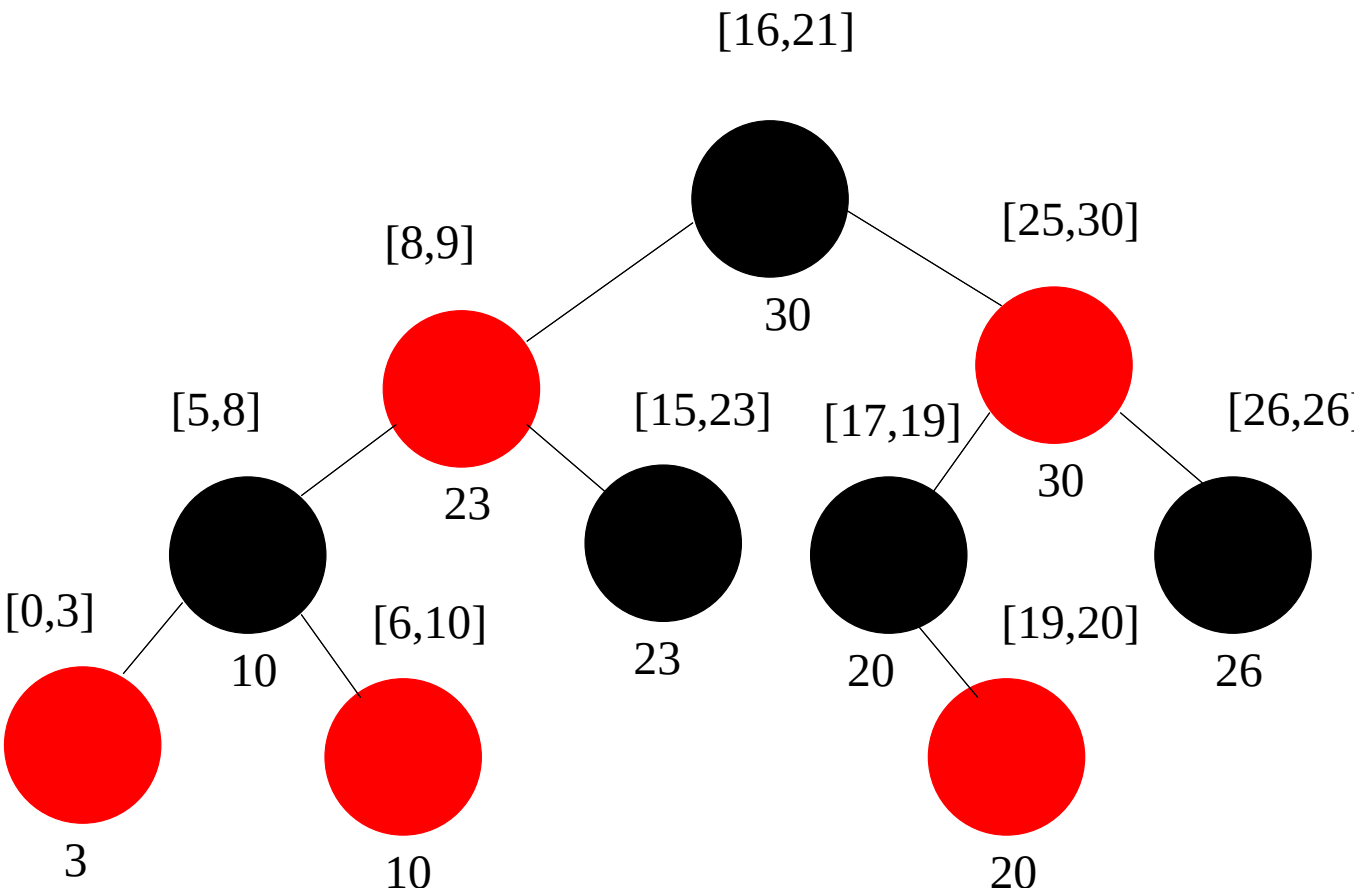
Dado o conjunto de intervalos abaixo, será dada uma representação na forma de árvore.



Árvores de Intervalo

Representação de Intervalos

Árvore representando um conjunto de intervalos.



Árvores de Intervalo

Árvores de Intervalo

Operações Suportadas:

Árvores de Intervalo

Operações Suportadas:

(a) Estrutura subjacente: cada nó x deve ter um campo adicional $int[x]$. A chave de x é o limite inferior do intervalo $low[int[x]]$.

Árvores de Intervalo

Operações Suportadas:

(a) Estrutura subjacente: cada nó x deve ter um campo adicional $int[x]$. A chave de x é o limite inferior do intervalo $low[int[x]]$.

(b) Informação adicional: $max[x]$ que informa o valor máximo de qualquer intervalo armazenado na sub-árvore que tem x por raiz.

Árvores de Intervalo

Operações Suportadas:

(a) Estrutura subjacente: cada nó x deve ter um campo adicional $int[x]$. A chave de x é o limite inferior do intervalo $low[int[x]]$.

(b) Informação adicional: $max[x]$ que informa o valor máximo de qualquer intervalo armazenado na sub-árvore que tem x por raiz.

(c) Manutenção da informação:
$max[x] = max(high[int[x]], max[left[x]], max[right[x]])$.

Árvores de Intervalo

Operações Suportadas:

(a) Estrutura subjacente: cada nó x deve ter um campo adicional $int[x]$. A chave de x é o limite inferior do intervalo $low[int[x]]$.

(b) Informação adicional: $max[x]$ que informa o valor máximo de qualquer intervalo armazenado na sub-árvore que tem x por raiz.

(c) Manutenção da informação:
$max[x] = max(high[int[x]], max[left[x]], max[right[x]])$.

(d) Novas operações: algoritmo de busca de intervalos mostrado a seguir.

Árvores de Intervalo

Teorema 2: Busca de Intervalos

Qualquer execução do algoritmo de busca de intervalos (T, i) retorna um nó cujo intervalo sobreponha-se ao procurado i ou retorna *nil*, caso não haja nenhum intervalo com sobreposição a i.

Teorema 2: Busca de Intervalos

Qualquer execução do algoritmo de busca de intervalos (T, i) retorna um nó cujo intervalo sobreponha-se ao procurado i ou retorna *nil*, caso não haja nenhum intervalo com sobreposição a i.

Esquema da Demonstração

Teorema 2: Busca de Intervalos

Qualquer execução do algoritmo de busca de intervalos (T, i) retorna um nó cujo intervalo sobreponha-se ao procurado i ou retorna *nil*, caso não haja nenhum intervalo com sobreposição a i.

Esquema da Demonstração

(a) Invariante: Se a árvore T contém um intervalo que se sobrepõe a i, então tal intervalo está numa sub-árvore com raiz em x.

Teorema 2: Busca de Intervalos

Qualquer execução do algoritmo de busca de intervalos (T, i) retorna um nó cujo intervalo sobreponha-se ao procurado i ou retorna *nil*, caso não haja nenhum intervalo com sobreposição a i.

Esquema da Demonstração

(a) Invariante: Se a árvore T contém um intervalo que se sobrepõe a i, então tal intervalo está numa sub-árvore com raiz em x.

(b) Inicialização: antes da primeira iteração, x aponta para a raiz de T. Portanto, a invariante é respeitada.

Teorema 2: Busca de Intervalos

Qualquer execução do algoritmo de busca de intervalos (T, i) retorna um nó cujo intervalo sobreponha-se ao procurado i ou retorna *nil*, caso não haja nenhum intervalo com sobreposição a i.

Esquema da Demonstração

(a) Invariante: Se a árvore T contém um intervalo que se sobrepõe a i, então tal intervalo está numa sub-árvore com raiz em x.

(b) Inicialização: antes da primeira iteração, x aponta para a raiz de T. Portanto, a invariante é respeitada.

(c) Manutenção: Em cada iteração, ou a linha 4 ou a linha 5 são executadas. Ambas mantêm a invariante.

Teorema 2: Busca de Intervalos

Qualquer execução do algoritmo de busca de intervalos (T, i) retorna um nó cujo intervalo sobreponha-se ao procurado i ou retorna *nil*, caso não haja nenhum intervalo com sobreposição a i.

Esquema da Demonstração

(a) Invariante: Se a árvore T contém um intervalo que se sobrepõe a i, então tal intervalo está numa sub-árvore com raiz em x.

(b) Inicialização: antes da primeira iteração, x aponta para a raiz de T. Portanto, a invariante é respeitada.

(c) Manutenção: Em cada iteração, ou a linha 4 ou a linha 5 são executadas. Ambas mantêm a invariante.

(d) Término: x aponta elemento procurado.

Árvores de Intervalo

Algoritmo de Busca de Intervalos

Dada uma árvore T e um intervalo i o algoritmo $Interval - Search$ retorna o nó da árvore que contém um intervalo que se sobreponha a i, caso exista. Do contrário, retorna NIL

INTERVAL-SEARCH(T, i)

```
1  x ← root[T]
2  y ← x
3  while x ≠ NIL and i does not overlap int[x]
4        do if left[x] ≠ NIL and max[left[x]] >= low[i]
5              then x ← left[x]
6              else x ← right[x]
7  return x
```

Comentário sobre o algoritmo *Interval – Search*

Comentário sobre o algoritmo *Interval – Search*

Algoritmo de Busca de Intervalos

Comentário sobre o algoritmo *Interval – Search*

Algoritmo de Busca de Intervalos

(a) Entradas: $i = [INI, FIM]$ é o intervalo de interesse e x a árvore onde se busca um intervalo que se sobreponha a i. Suponhamos que a raiz da sub-árvore x contenha o intervalo $[a, b]$.

Comentário sobre o algoritmo *Interval – Search*

Algoritmo de Busca de Intervalos

(a) Entradas: $i = [INI, FIM]$ é o intervalo de interesse e x a árvore onde se busca um intervalo que se sobreponha a i. Suponhamos que a raiz da sub-árvore x contenha o intervalo $[a, b]$.

(b) Se houver sobreposição entre $[INI, FIM]$ e $[a, b]$, o algoritmo termina.

Comentário sobre o algoritmo *Interval – Search*

Algoritmo de Busca de Intervalos

(a) Entradas: $i = [INI, FIM]$ é o intervalo de interesse e x a árvore onde se busca um intervalo que se sobreponha a i. Suponhamos que a raiz da sub-árvore x contenha o intervalo $[a, b]$.

(b) Se houver sobreposição entre $[INI, FIM]$ e $[a, b]$, o algoritmo termina.

(c) Caso tal sobreposição não ocorra, chega-se ao teste da linha 4 para que se decida por qual das sub-árvores a busca continua. Uma das duas situações seguintes pode ter ocorrido.

Comentário sobre o algoritmo *Interval – Search*

Comentário sobre o algoritmo *Interval – Search*

Algoritmo de Busca de Intervalos

Comentário sobre o algoritmo *Interval – Search*

Algoritmo de Busca de Intervalos

(c.1) $INI > b$, implicando que $i = [INI, FIM]$ está à direita de $[a, b]$. Neste caso, se $max[left[x]] >= INI$, então há algum intervalo que possui sobreposição com i na sub-árvore esquerda, já que todos os intervalos desta sub-árvore devem começar antes de a e pelo menos um termina depois de INI. Se $max[left[x]] < INI$, então não há como existir sobreposição com o intervalo i na sub-árvore esquerda.

Comentário sobre o algoritmo *Interval – Search*

Algoritmo de Busca de Intervalos

(c.1) $INI > b$, implicando que $i = [INI, FIM]$ está à direita de $[a, b]$. Neste caso, se $max[left[x]] >= INI$, então há algum intervalo que possui sobreposição com i na sub-árvore esquerda, já que todos os intervalos desta sub-árvore devem começar antes de a e pelo menos um termina depois de INI. Se $max[left[x]] < INI$, então não há como existir sobreposição com o intervalo i na sub-árvore esquerda.

(c.2) $FIM < a$, implicando que $i = [INI, FIM]$ está à esquerda de $[a, b]$. Neste caso, se houver algum intervalo com sobreposição ele tem que estar na sub-árvore esquerda, já que todo intervalo da sub-árvore direita começam depois de a e por consequência depois de FIM.

Comentário sobre o algoritmo *Interval – Search*

Algoritmo de Busca de Intervalos

(c.1) $INI > b$, implicando que $i = [INI, FIM]$ está à direita de $[a, b]$. Neste caso, se $max[left[x]] >= INI$, então há algum intervalo que possui sobreposição com i na sub-árvore esquerda, já que todos os intervalos desta sub-árvore devem começar antes de a e pelo menos um termina depois de INI. Se $max[left[x]] < INI$, então não há como existir sobreposição com o intervalo i na sub-árvore esquerda.

(c.2) $FIM < a$, implicando que $i = [INI, FIM]$ está à esquerda de $[a, b]$. Neste caso, se houver algum intervalo com sobreposição ele tem que estar na sub-árvore esquerda, já que todo intervalo da sub-árvore direita começam depois de a e por consequência depois de FIM.

(d) x inicia na raiz da árvore e vai iterativamente descendo até que se encontre uma sobreposição ou que um nó sentinela seja atingido, indicando a inexistência de sobreposição.

Comentário sobre o algoritmo *Interval – Search*

Comentário sobre o algoritmo *Interval – Search*

Algoritmo de Busca de Intervalos

Comentário sobre o algoritmo *Interval – Search*

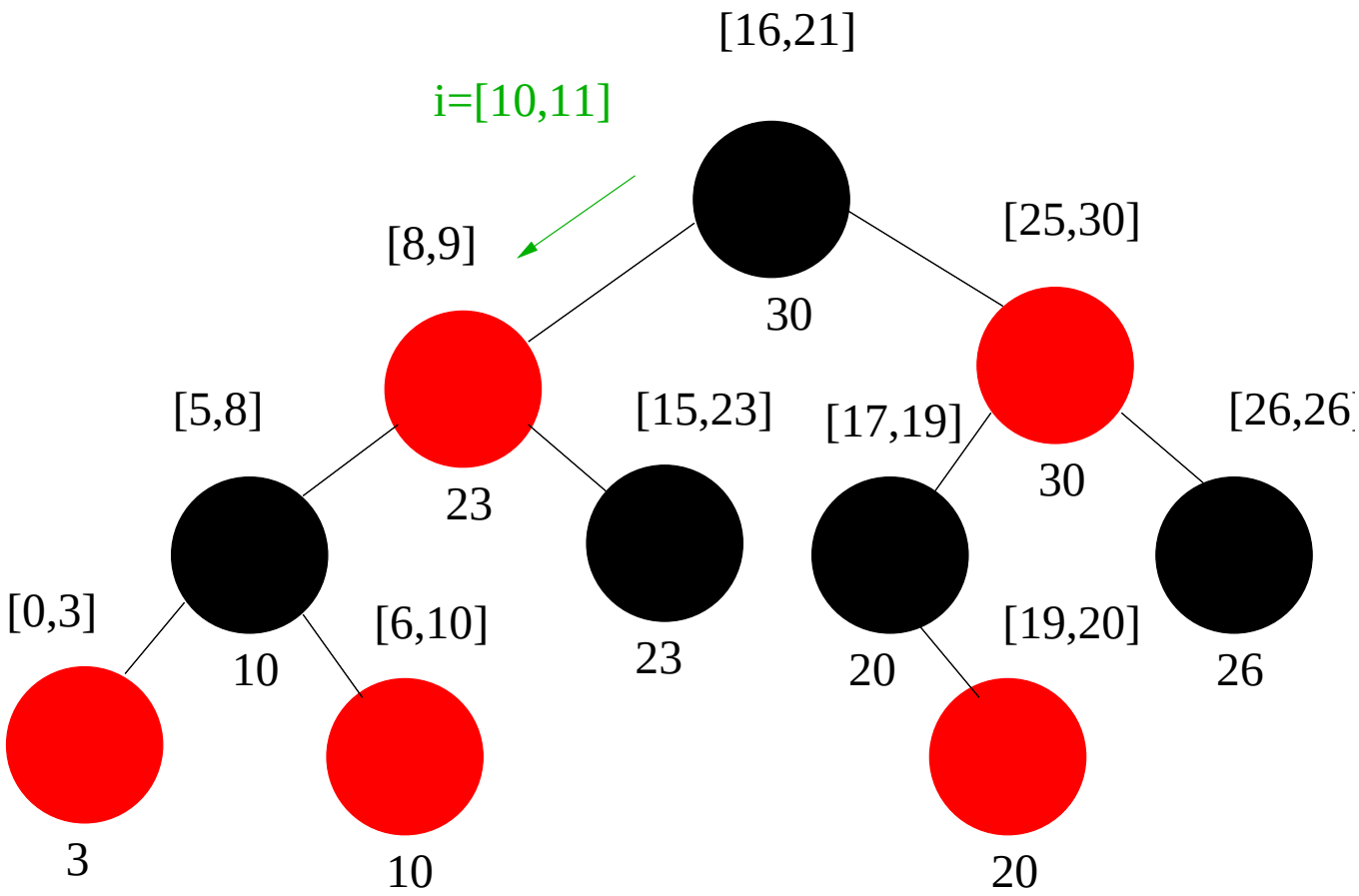
Algoritmo de Busca de Intervalos

(e) O algoritmo em questão localiza uma sobreposição, caso exista uma. Caso haja mais de uma, no entanto, ele localiza a que estiver mais à esquerda na árvore.

Exemplo - *Interval – Search*

Exemplo - *Interval – Search*

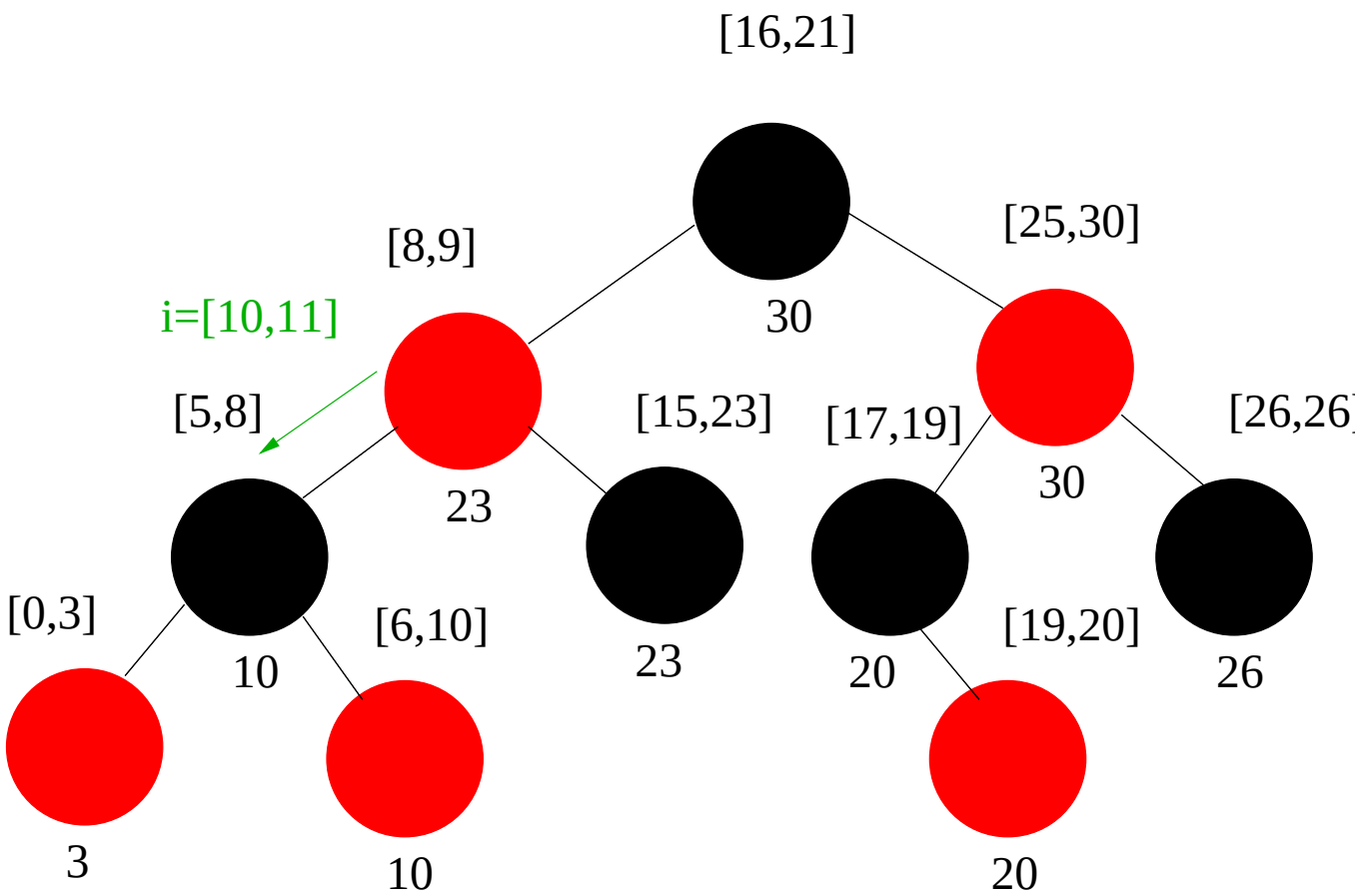
Busca pelo intervalo $[10, 11]$ na árvore dada como exemplo nesta seção.



Exemplo - *Interval – Search*

Exemplo - *Interval – Search*

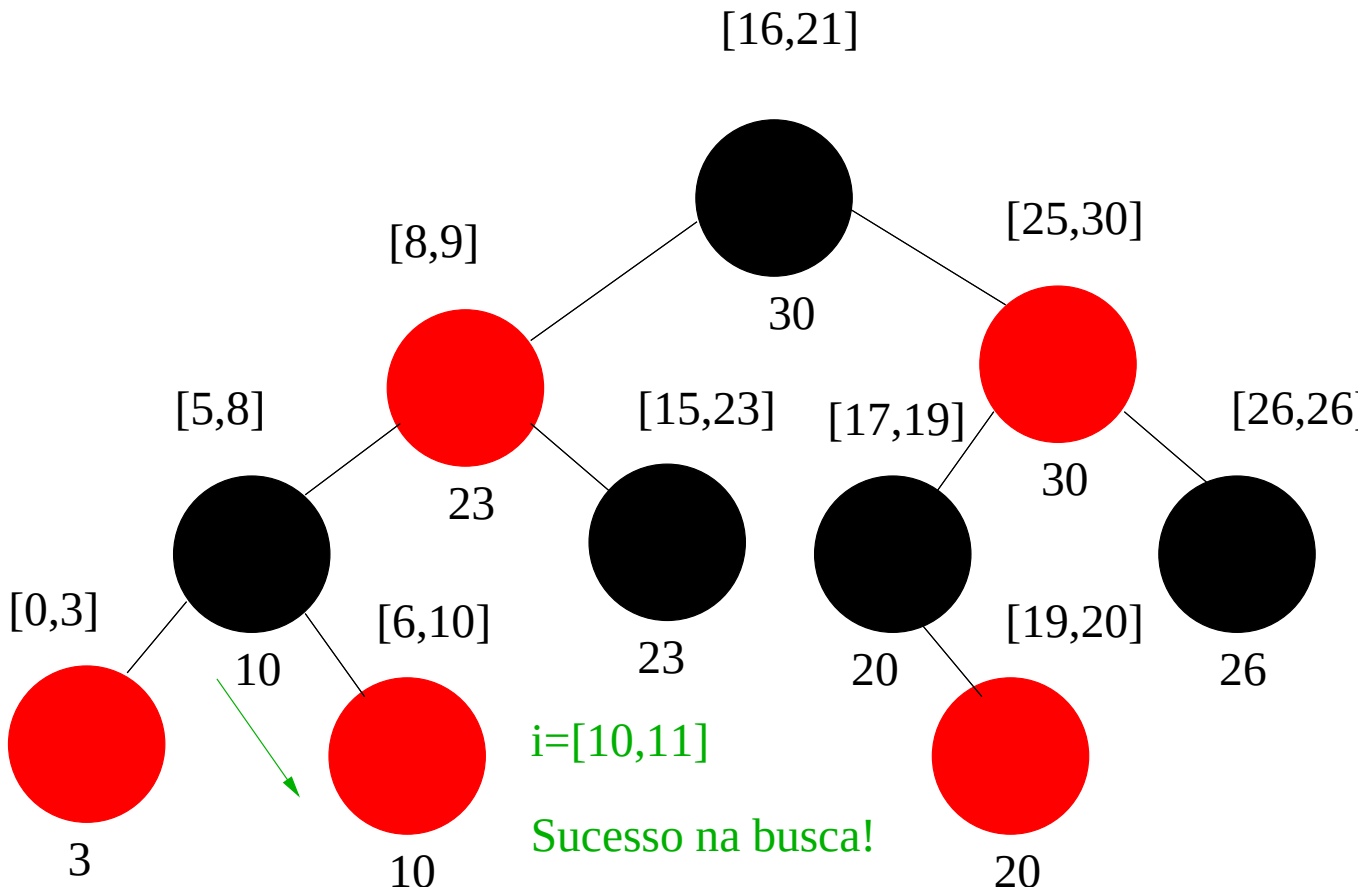
Descendo pela sub-árvore esquerda.



Exemplo - *Interval – Search*

Exemplo - *Interval – Search*

Descendo pela sub-árvore direita e encontrando um intervalo. Se *i* fosse [11,14], por exemplo, a seqüência seria a mesma, mas não teríamos encontrado nenhum intervalo.



Conclusões

Conclusões

Conclusões

(a) Dependendo do problema específico sendo resolvido, pode-se obter uma solução mais eficiente e elegante ao se optar por aumentar uma estrutura existente, ao invés de simplesmente usar uma estrutura conhecida ou partir-se para a criação de uma totalmente nova.

Conclusões

(a) Dependendo do problema específico sendo resolvido, pode-se obter uma solução mais eficiente e elegante ao se optar por aumentar uma estrutura existente, ao invés de simplesmente usar uma estrutura conhecida ou partir-se para a criação de uma totalmente nova.

(b) O campo a se aumentar na estrutura deve ser tal que sua atualização seja bastante eficiente.

Conclusões

(a) Dependendo do problema específico sendo resolvido, pode-se obter uma solução mais eficiente e elegante ao se optar por aumentar uma estrutura existente, ao invés de simplesmente usar uma estrutura conhecida ou partir-se para a criação de uma totalmente nova.

(b) O campo a se aumentar na estrutura deve ser tal que sua atualização seja bastante eficiente.

(c) Os algoritmos que manipulavam a estrutura original devem agora levar em consideração a atualização do novo campo.

Conclusões

(a) Dependendo do problema específico sendo resolvido, pode-se obter uma solução mais eficiente e elegante ao se optar por aumentar uma estrutura existente, ao invés de simplesmente usar uma estrutura conhecida ou partir-se para a criação de uma totalmente nova.

(b) O campo a se aumentar na estrutura deve ser tal que sua atualização seja bastante eficiente.

(c) Os algoritmos que manipulavam a estrutura original devem agora levar em consideração a atualização do novo campo.

(d) O novo campo geralmente leva à criação de novos algoritmos que estendam o comportamento da estrutura original para a solução de nosso problema específico.