Árvores Binárias Balanceadas (Árvores Rubro-Negras)

Breno Piva Ribeiro – IC/UNICAMP

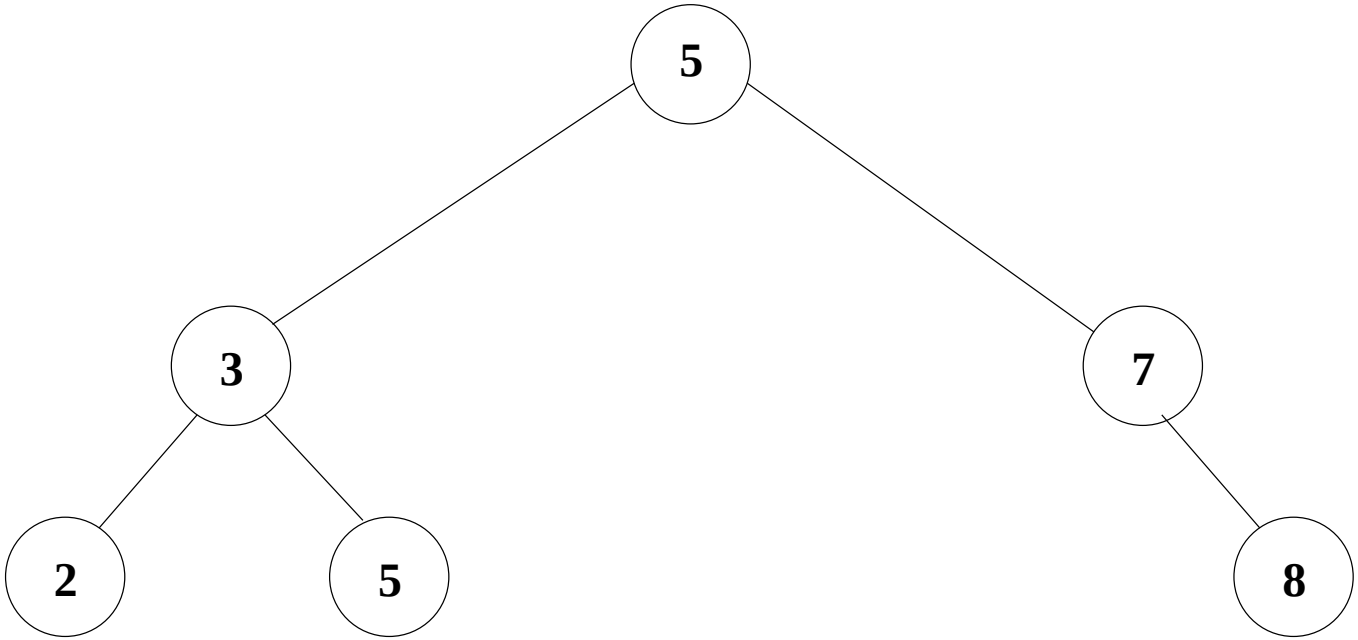
11 de setembro de 2007

Sumário

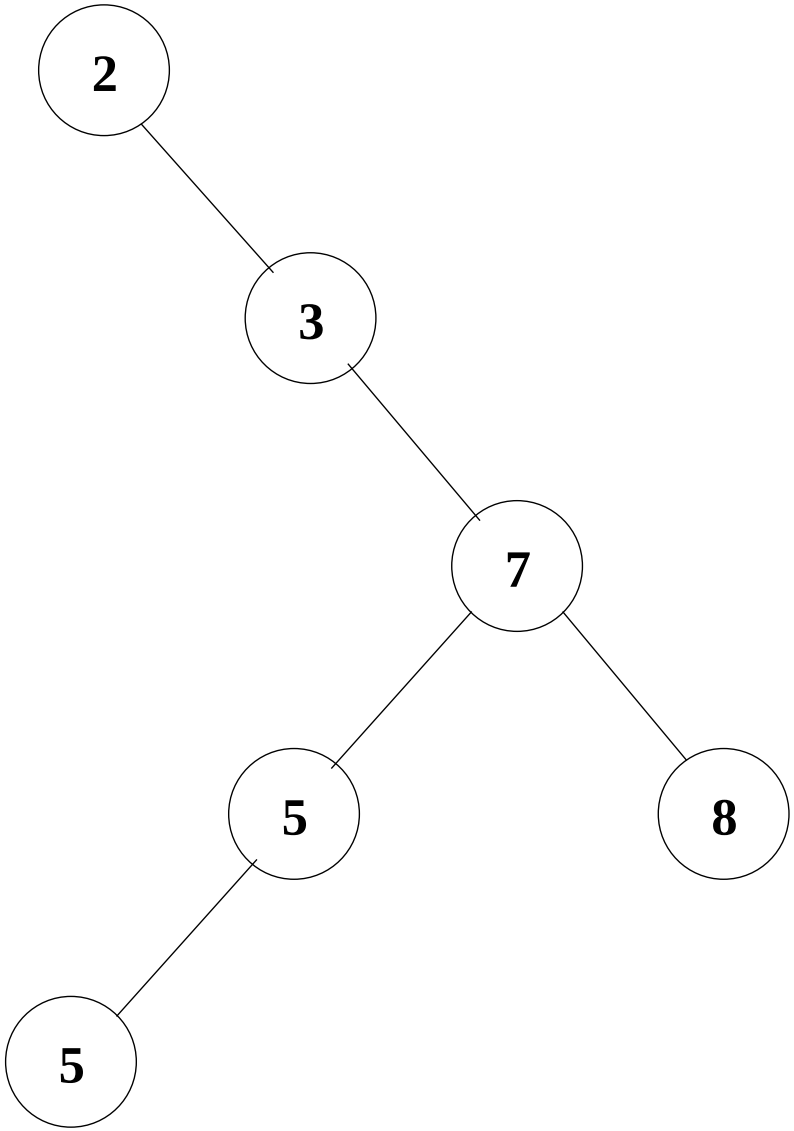
Árvores Binárias de Busca

- Árvores Binárias que possuem a seguinte propriedade:
- Se y é um nó da subárvore esquerda de x, então $Chave[y] \leq Chave[x]$.
- Se y é um nó da subárvore direita de x, então $Chave[x] \leq Chave[y]$.

Árvores Binárias de Busca



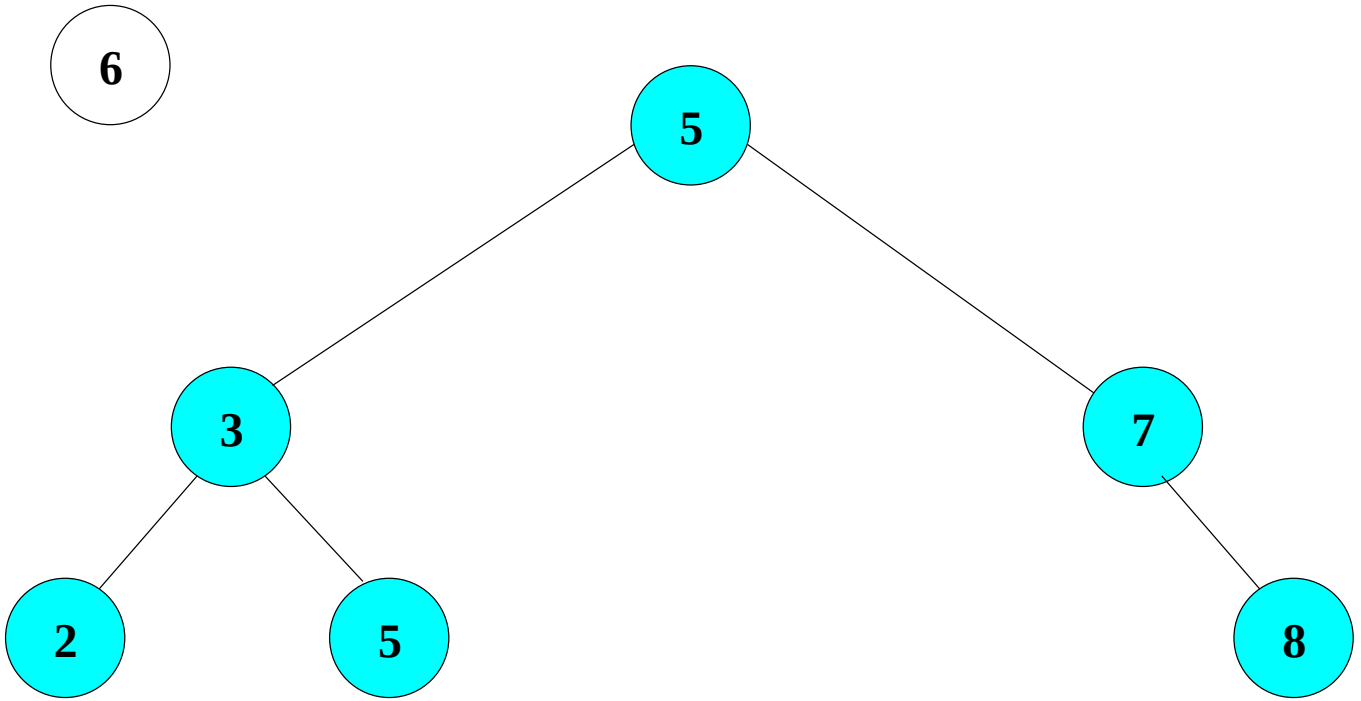
Árvores Binárias de Busca



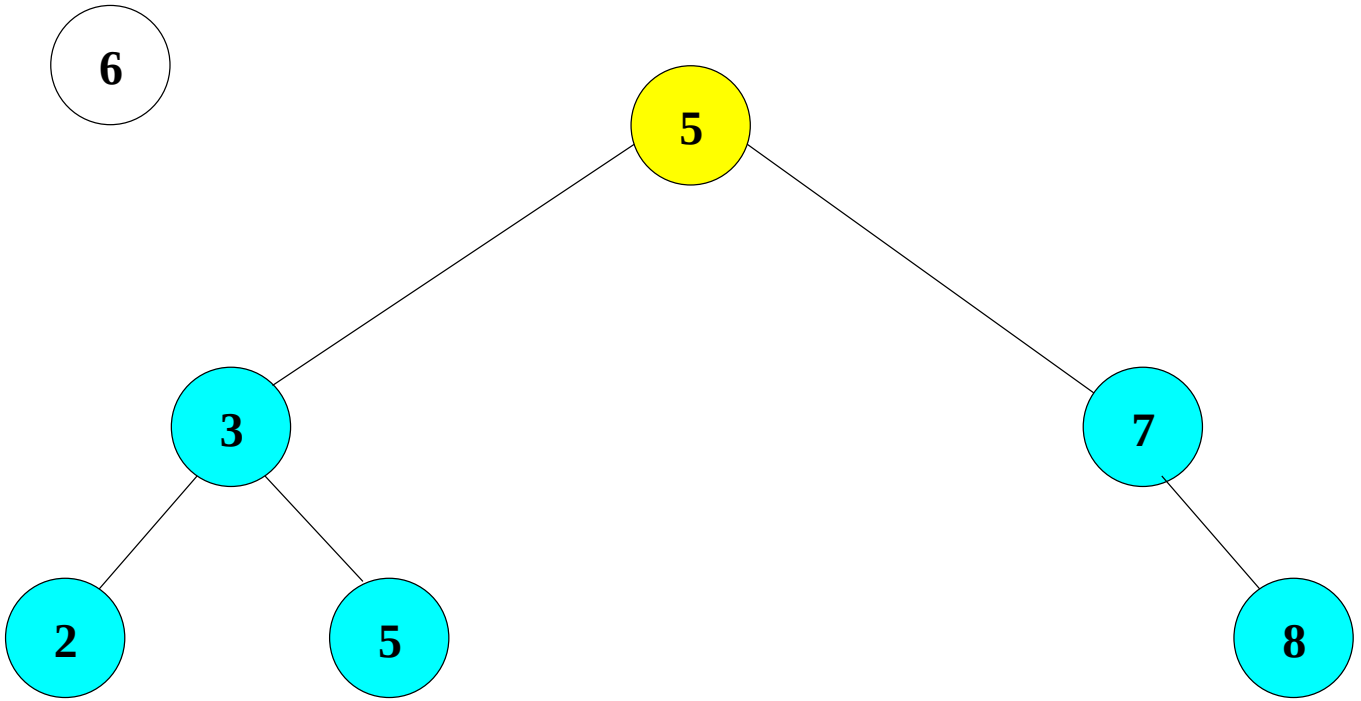
Árvores Binárias de Busca

- Operações sobre Árvores Binárias de Busca:
- Busca, Mínimo, Máximo, Predecessor, Sucessor, Inserir, Remover

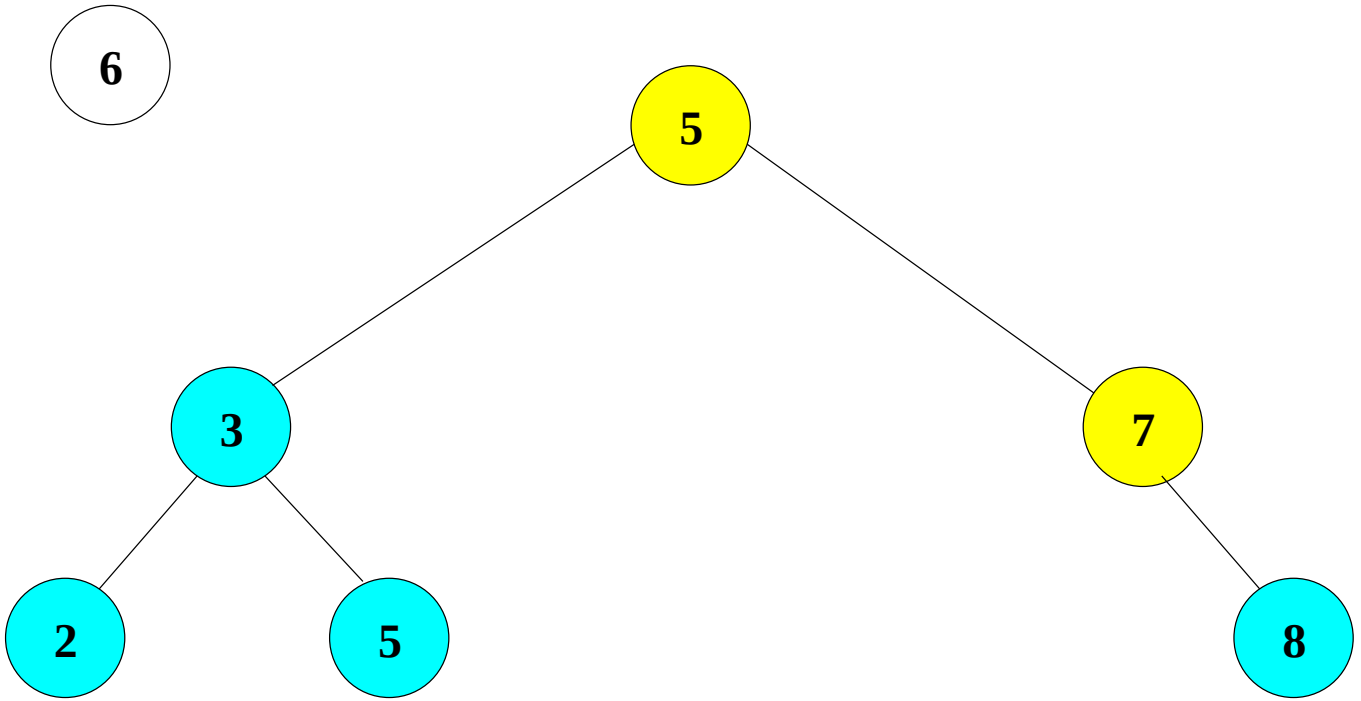
Inserir



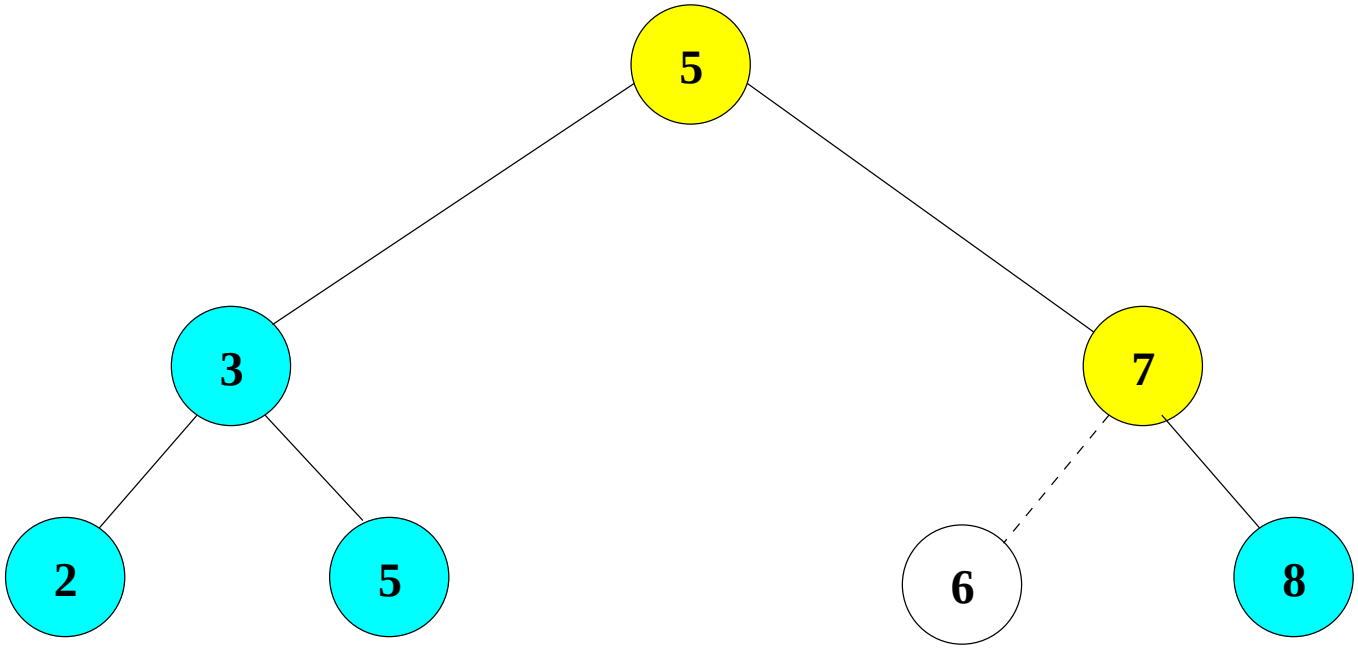
Inserir



Inserir



Inserir



Remover

- Remover um elemento é ligeiramente mais complicado, pois existem três casos a considerar:

Remover

- Remover um elemento é ligeiramente mais complicado, pois existem três casos a considerar:
- Caso 1: O elemento a ser removido é uma folha.

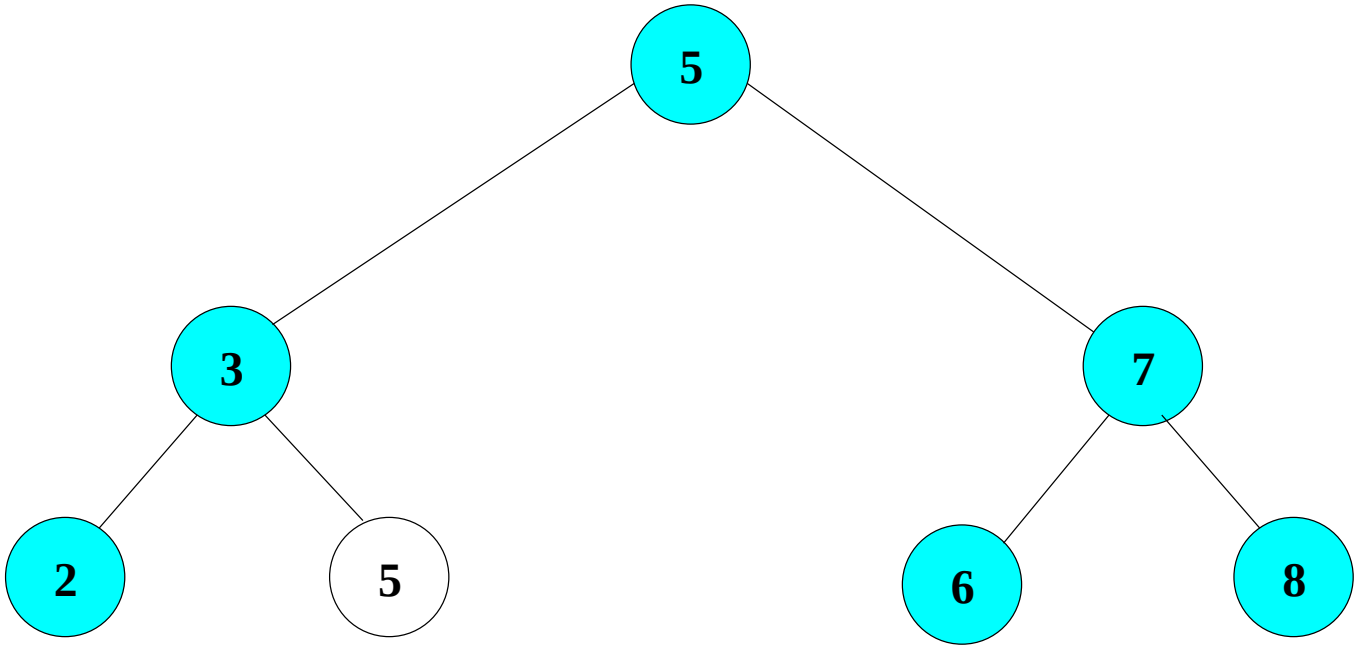
Remover

- Remover um elemento é ligeiramente mais complicado, pois existem três casos a considerar:
- Caso 1: O elemento a ser removido é uma folha.
- Caso 2: O elemento a ser removido possui um único filho.

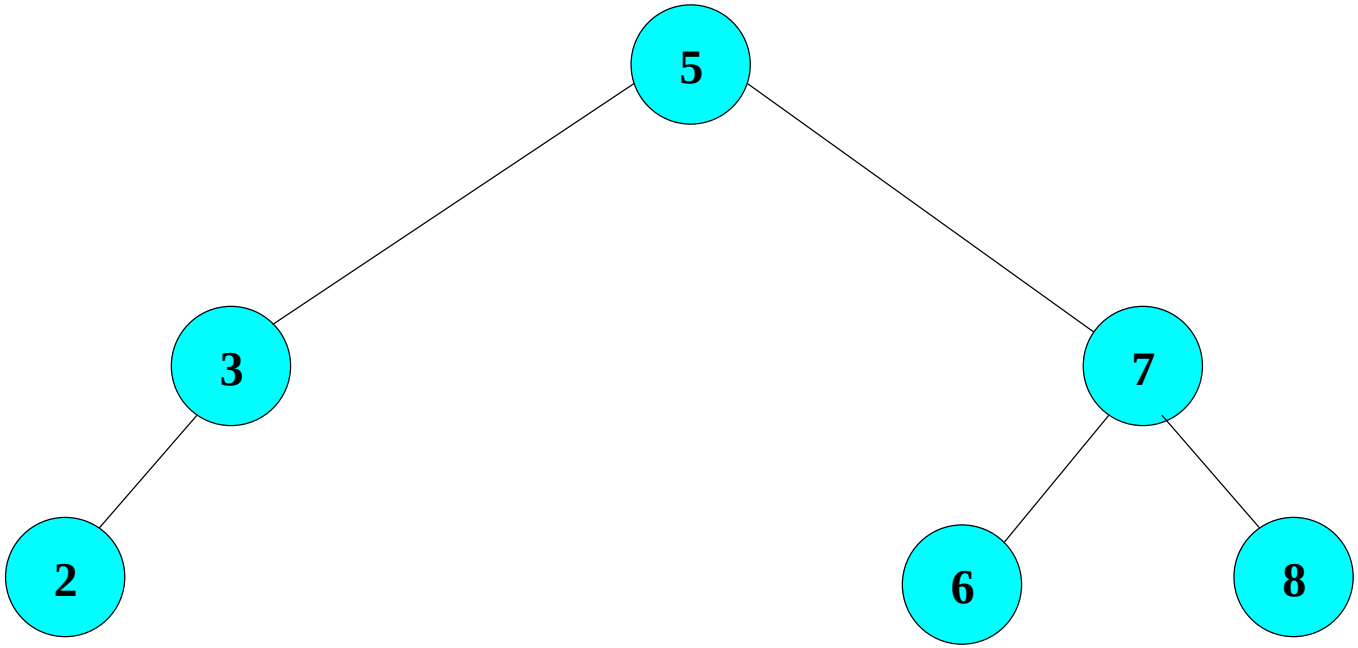
Remover

- Remover um elemento é ligeiramente mais complicado, pois existem três casos a considerar:
- Caso 1: O elemento a ser removido é uma folha.
- Caso 2: O elemento a ser removido possui um único filho.
- Caso 3: O elemento a ser removido possui dois filhos.

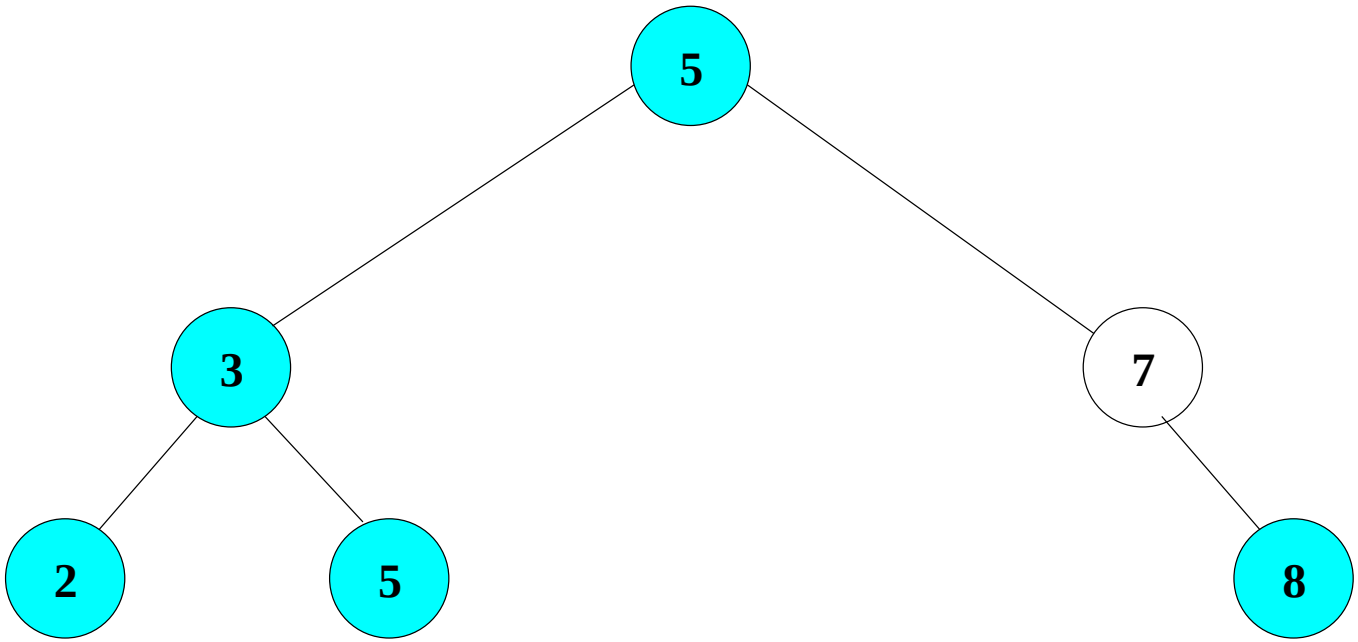
Caso 1: O elemento a ser removido é uma folha



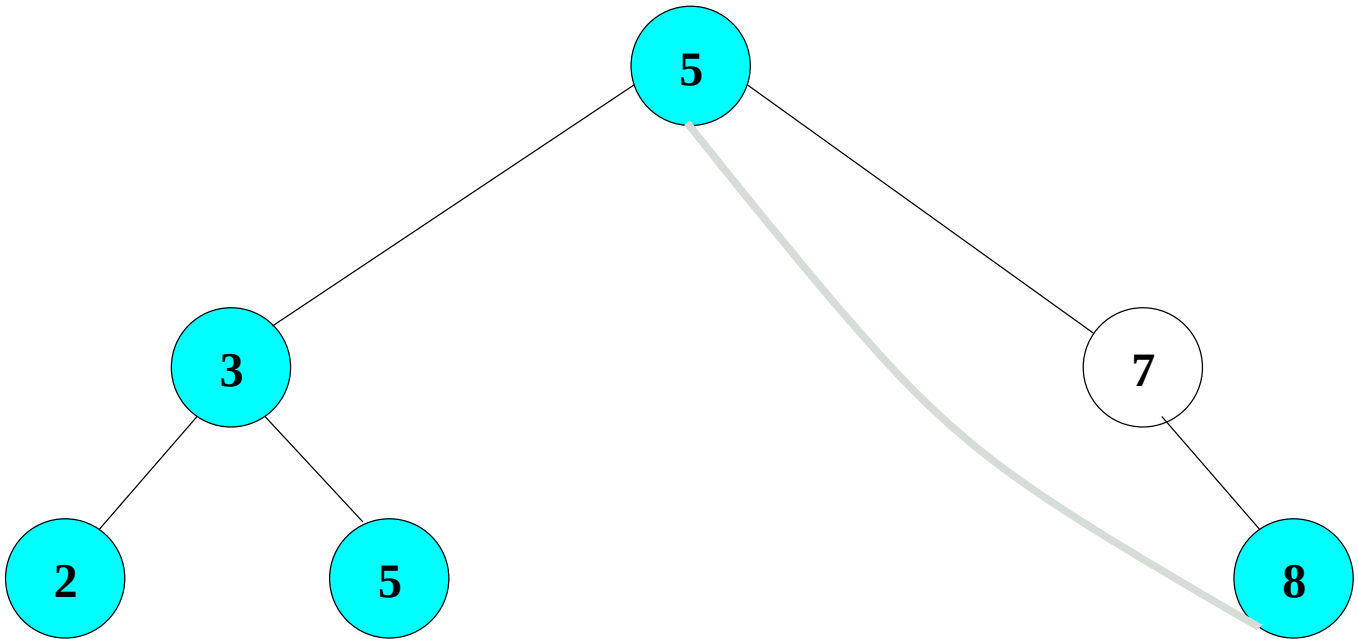
Caso 1: O elemento a ser removido é uma folha



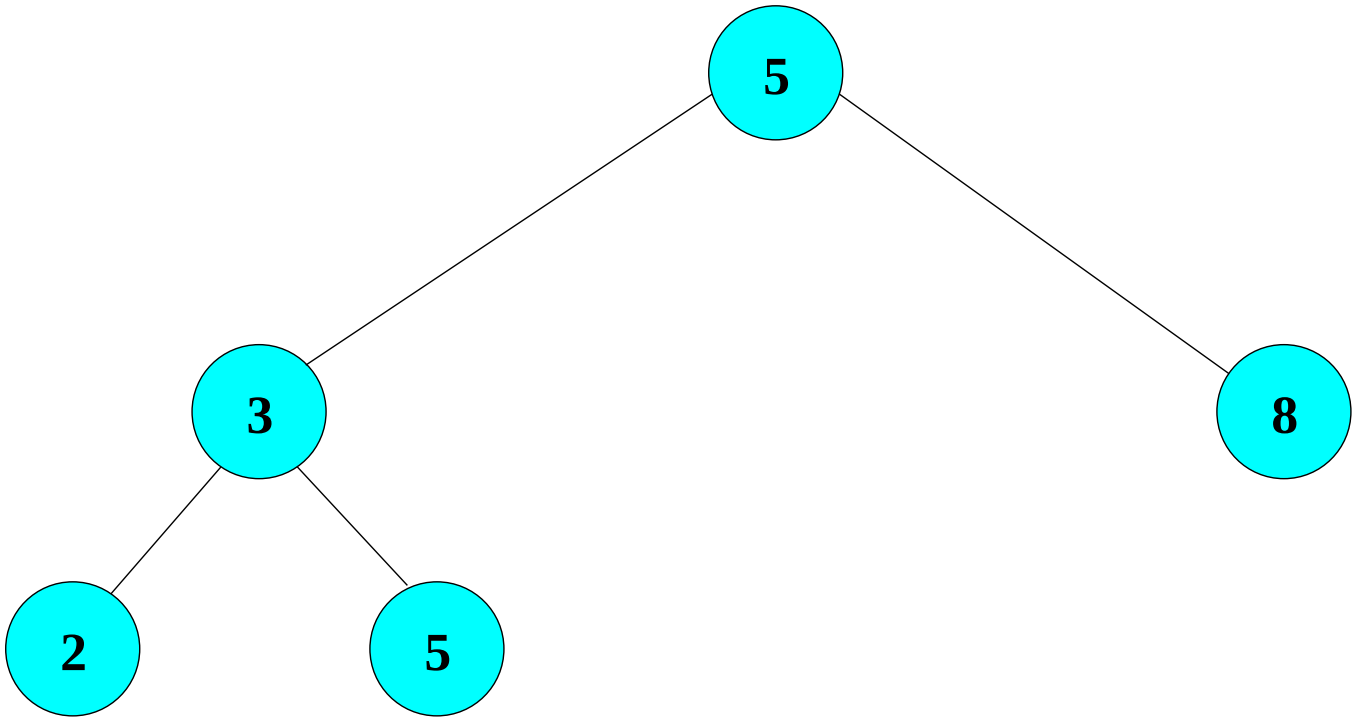
Caso 2: O elemento a ser removido possui um único filho



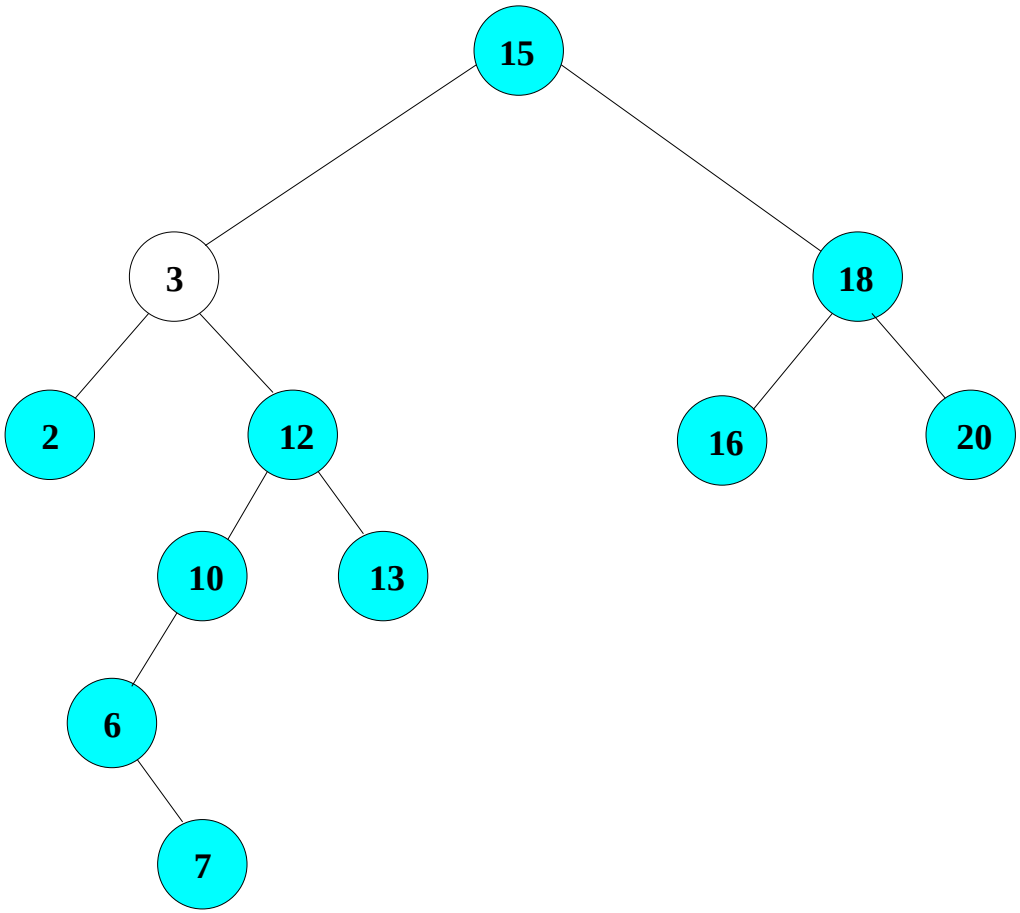
Caso 2: O elemento a ser removido possui um único filho



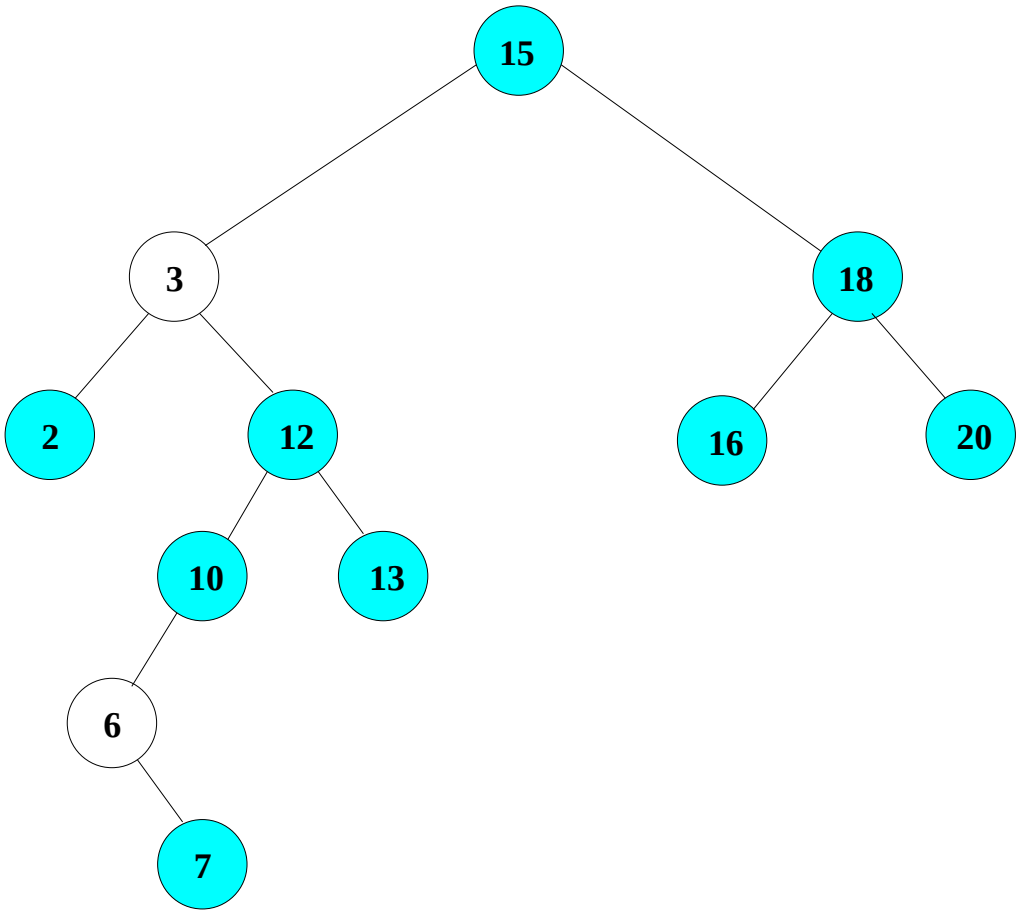
Caso 2: O elemento a ser removido possui um único filho



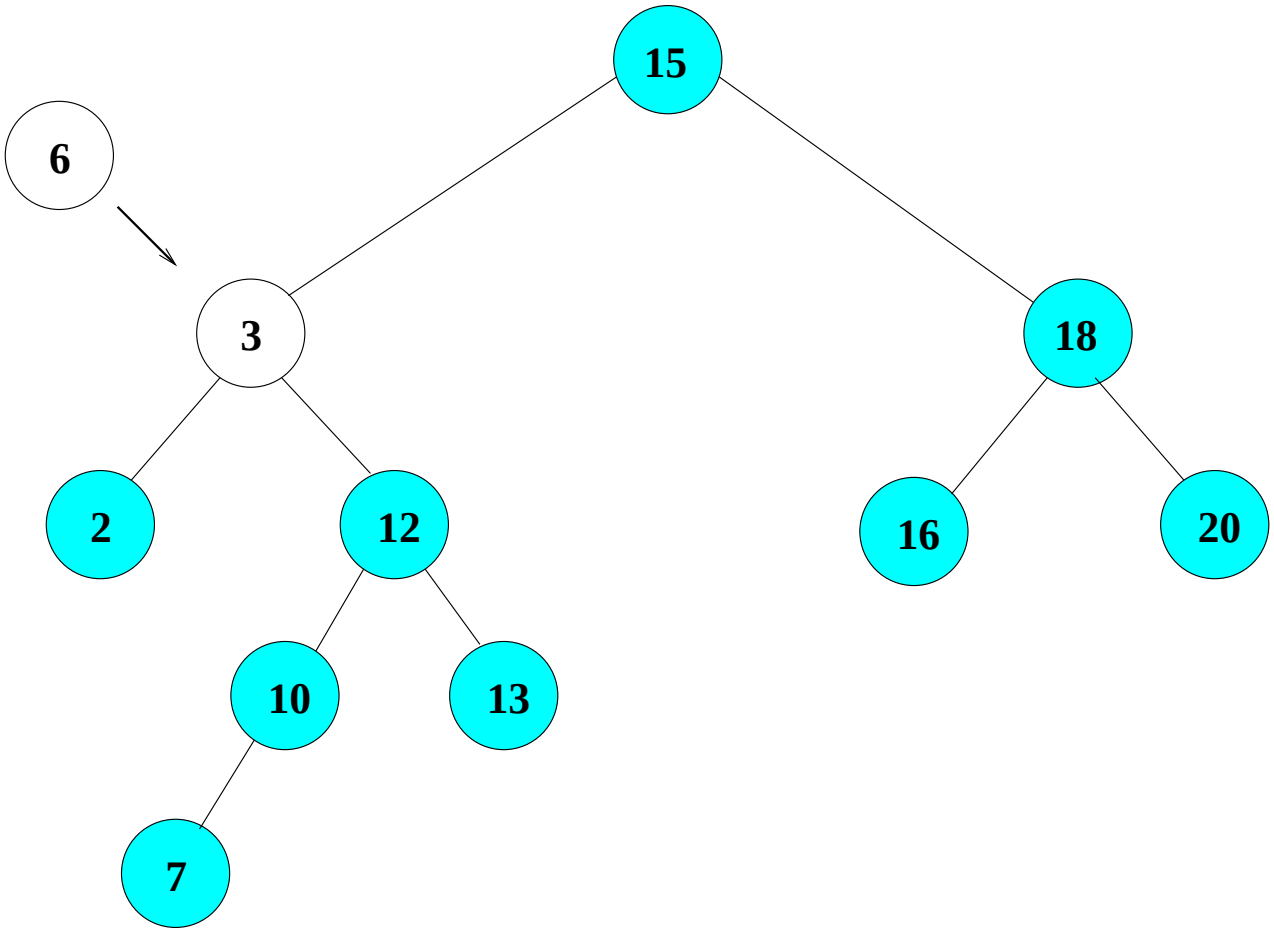
Caso 3: O elemento a ser removido possui dois filhos



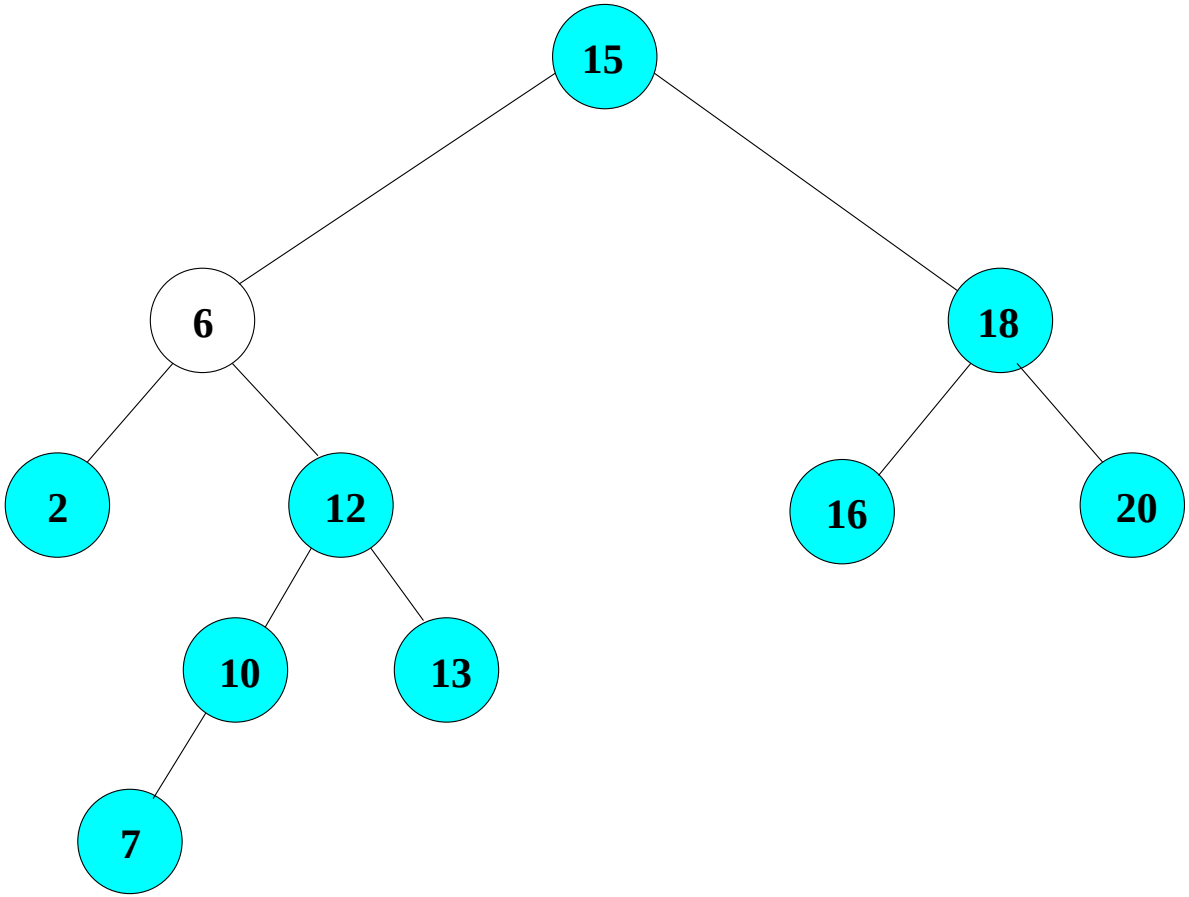
Caso 3: O elemento a ser removido possui dois filhos



Caso 3: O elemento a ser removido possui dois filhos



Caso 3: O elemento a ser removido possui dois filhos



Análise

- Todas operações percorrem um único caminho da raiz até uma folha, logo, a complexidade dessas operações é da ordem O(h), onde h é a altura da árvore.

Sumário

Árvores Rubro-Negras

- Uma Árvore Rubro-Negra é uma Árvore Binária de Busca com um bit extra de armazenamento por nó que indica sua cor, PRETA ou VERMELHA.

Árvores Rubro-Negras

- Uma Árvore Rubro-Negra é uma Árvore Binária de Busca com um bit extra de armazenamento por nó que indica sua cor, PRETA ou VERMELHA.
- A idéia é restringir a forma como os nós podem ser coloridos em qualquer caminho da raiz até as folhas de tal forma que nenhum caminho possa ser mais que duas vezes mais longo que outro caminho, deste modo a árvore fica aproximadamente balanceada.

Árvores Rubro-Negras

- Uma Árvore Binária de Busca é uma Árvore Rubro-Negra se ela satisfaz as seguintes propriedades:
- 1: Todo nó é VERMELHO ou PRETO

Árvores Rubro-Negras

- Uma Árvore Binária de Busca é uma Árvore Rubro-Negra se ela satisfaz as seguintes propriedades:
- 1: Todo nó é VERMELHO ou PRETO
- 2: A raiz é PRETA

Árvores Rubro-Negras

- Uma Árvore Binária de Busca é uma Árvore Rubro-Negra se ela satisfaz as seguintes propriedades:
- 1: Todo nó é VERMELHO ou PRETO
- 2: A raiz é PRETA
- 3: Toda folha (NIL) é PRETA

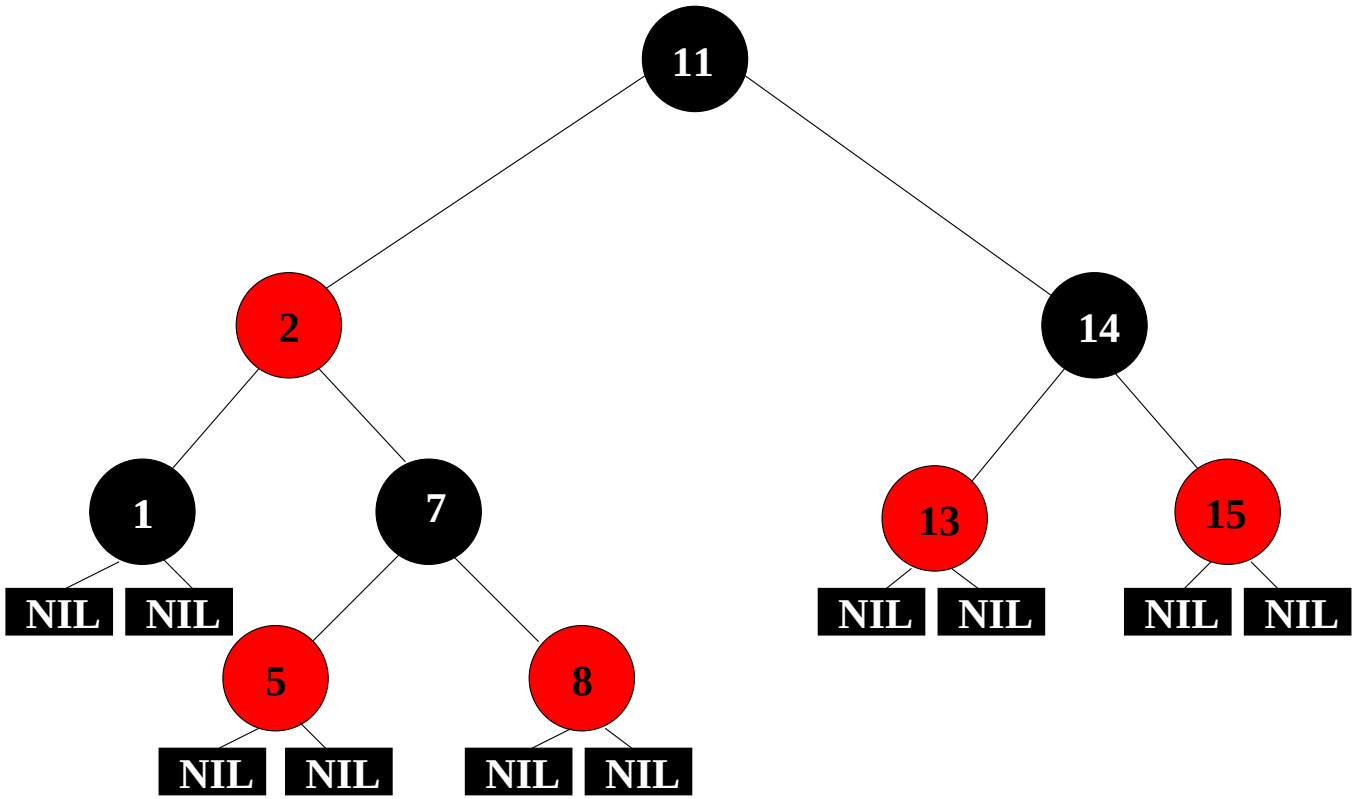
Árvores Rubro-Negras

- Uma Árvore Binária de Busca é uma Árvore Rubro-Negra se ela satisfaz as seguintes propriedades:
- 1: Todo nó é VERMELHO ou PRETO
- 2: A raiz é PRETA
- 3: Toda folha (NIL) é PRETA
- 4: Se um nó é VERMELHO, então ambos os seus filhos são PRETOS

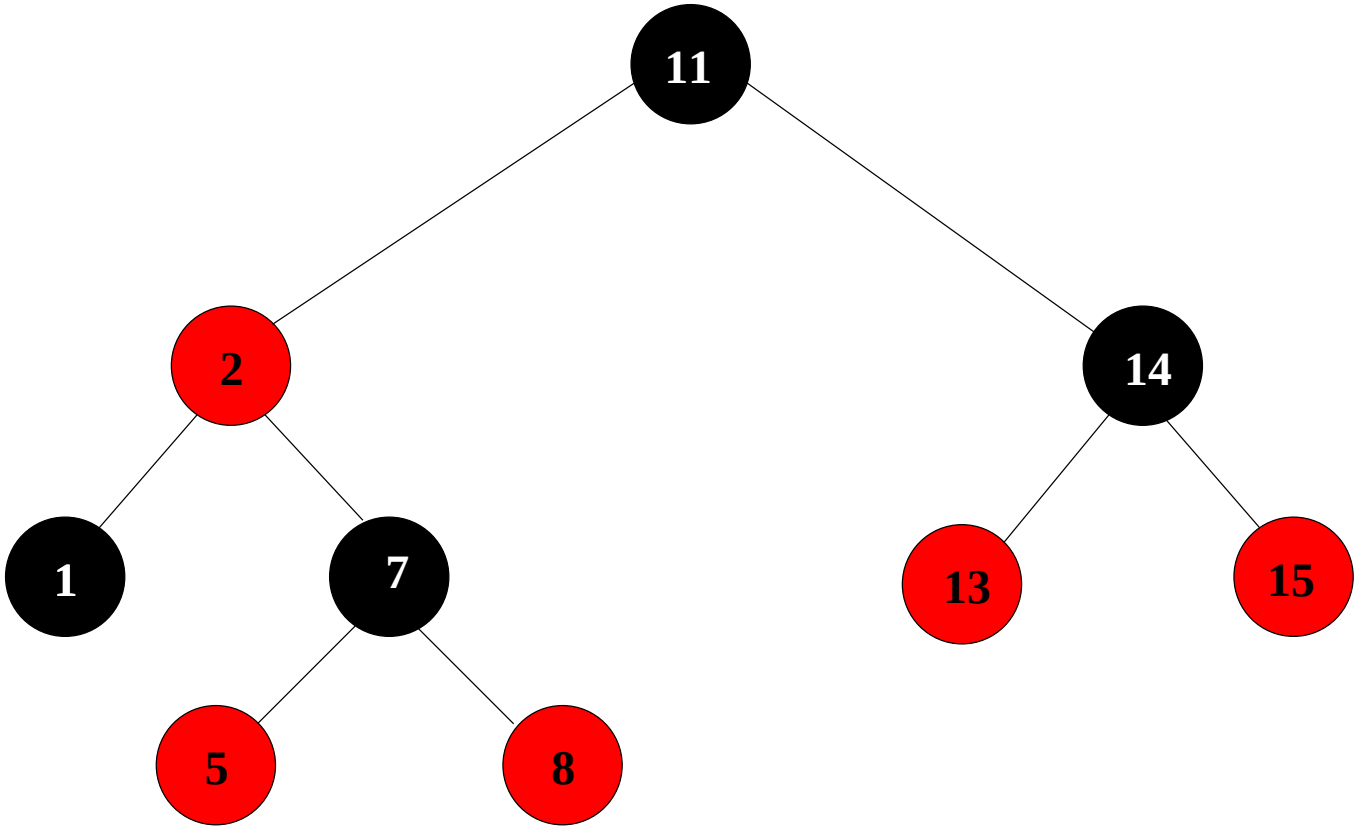
Árvores Rubro-Negras

- Uma Árvore Binária de Busca é uma Árvore Rubro-Negra se ela satisfaz as seguintes propriedades:
- 1: Todo nó é VERMELHO ou PRETO
- 2: A raiz é PRETA
- 3: Toda folha (NIL) é PRETA
- 4: Se um nó é VERMELHO, então ambos os seus filhos são PRETOS
- 5: Para cada nó, todos os caminhos do nó para folhas descendentes contém o mesmo número de nós PRETOS.

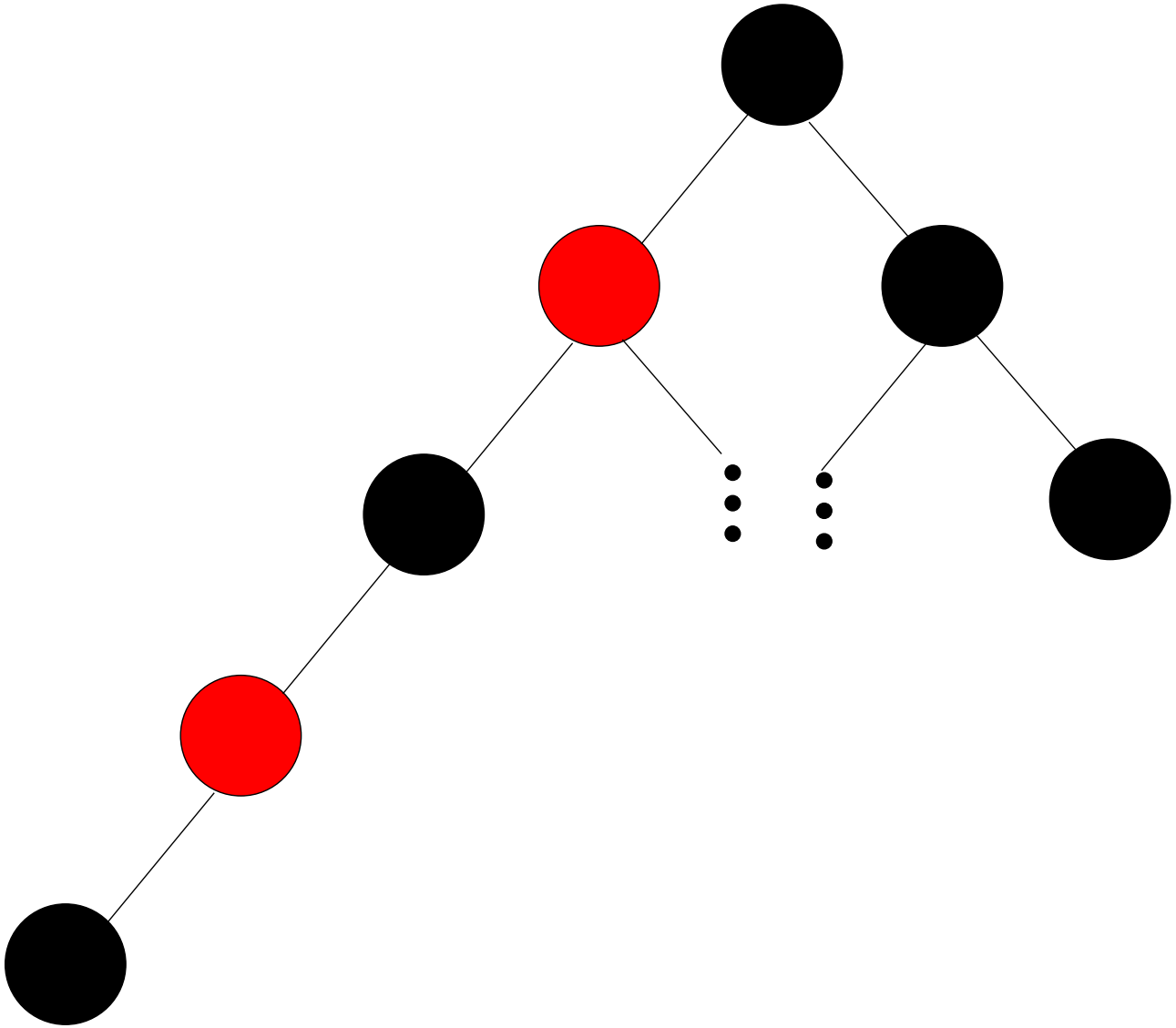
Árvores Rubro-Negras



Árvores Rubro-Negras



Árvores Rubro-Negras



Árvores Rubro-Negras

Lema:

Uma Árvore Rubro-Negra com n nós internos possui altura no máximo $2\lg(n + 1)$.

Prova:

- Primeiro mostramos que a subárvore com raiz em qualquer nó x contém pelo menos $2^{bh(x)}$-1 nós internos. (Por indução em bh(x))

Prova:

- Primeiro mostramos que a subárvore com raiz em qualquer nó x contém pelo menos $2^{bh(x)}$-1 nós internos. (Por indução em bh(x))
- Caso Base (bh(x) = 0): $2^{bh(x)}$-1 = 2^0-1 = 0 nós internos, ou seja, bh(x) = 0 engloba o caso em que x é uma folha.

Prova:

- Primeiro mostramos que a subárvore com raiz em qualquer nó x contém pelo menos $2^{bh(x)}$-1 nós internos. (Por indução em bh(x))
- Caso Base (bh(x) = 0): $2^{bh(x)}$-1 = 2^0-1 = 0 nós internos, ou seja, bh(x) = 0 engloba o caso em que x é uma folha.
- H.I.: A subárvore que tem como raiz o nó com altura-preta bh(x)-1 possui $2^{bh(x)-1}$-1 nós internos.

Prova:

- Primeiro mostramos que a subárvore com raiz em qualquer nó x contém pelo menos $2^{bh(x)}$-1 nós internos. (Por indução em bh(x))
- Caso Base (bh(x) = 0): $2^{bh(x)}$-1 = 2^0-1 = 0 nós internos, ou seja, bh(x) = 0 engloba o caso em que x é uma folha.
- H.I.: A subárvore que tem como raiz o nó com altura-preta bh(x)-1 possui $2^{bh(x)-1}$-1 nós internos.
- P.I.: Os filhos de um nó x com altura-preta bh(x) possuem altura-preta bh(x) se for vermelho ou bh(x)-1 se for preto, logo, a subárvore que tem x como raiz possui pelo menos ($2^{bh(x)-1}$-1) + ($2^{bh(x)-1}$-1) + 1 = $2^{bh(x)}$-1 nós internos.

- Agora que provamos que a subárvore com raiz em qualquer nó x contém pelo menos $2^{bh(x)}$-1 nós internos. Iremos completar a prova que uma Árvore Rubro-Negra com n nós internos possui altura no máximo 2lg(n + 1).

Árvores Rubro-Negras

- Seja h a altura da árvore. De acordo com a propriedade 4, pelo menos metade dos nós em qualquer caminho simples da raiz até uma folha, não incluindo a raiz, deve ser PRETA. Consequentemente, a altura-preta da raiz deve ser pelo menos h/2. Então:

Árvores Rubro-Negras

- Seja h a altura da árvore. De acordo com a propriedade 4, pelo menos metade dos nós em qualquer caminho simples da raiz até uma folha, não incluindo a raiz, deve ser PRETA. Consequentemente, a altura-preta da raiz deve ser pelo menos h/2. Então:
- $n \geq 2^{h/2} - 1$

Árvores Rubro-Negras

- Seja h a altura da árvore. De acordo com a propriedade 4, pelo menos metade dos nós em qualquer caminho simples da raiz até uma folha, não incluindo a raiz, deve ser PRETA. Consequentemente, a altura-preta da raiz deve ser pelo menos h/2. Então:
- $n \geq 2^{h/2} - 1$
- Passando o 1 para o lado esquerdo e tomando os logaritmos de ambos os lados, temos:

Árvores Rubro-Negras

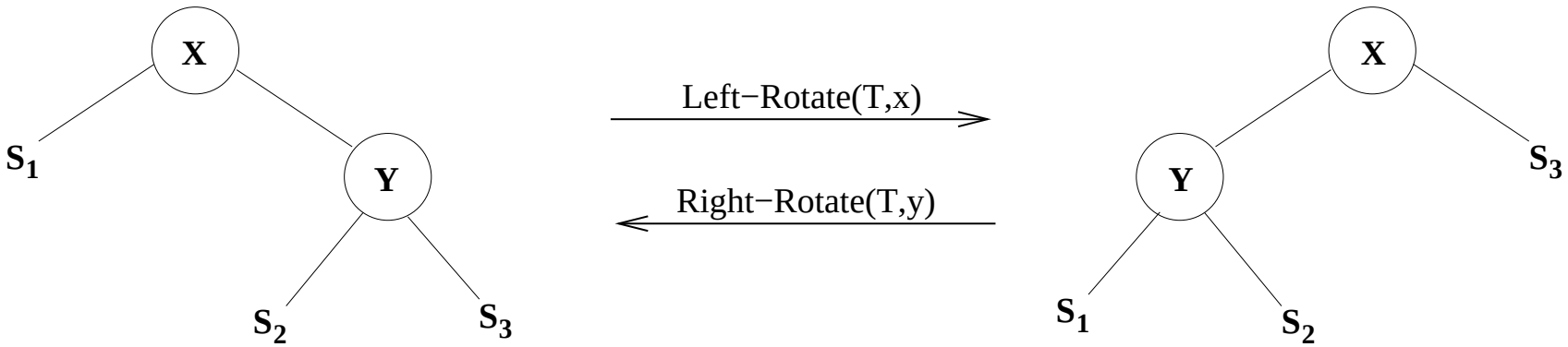
- Seja h a altura da árvore. De acordo com a propriedade 4, pelo menos metade dos nós em qualquer caminho simples da raiz até uma folha, não incluindo a raiz, deve ser PRETA. Consequentemente, a altura-preta da raiz deve ser pelo menos h/2. Então:
- $n \geq 2^{h/2} - 1$
- Passando o 1 para o lado esquerdo e tomando os logaritmos de ambos os lados, temos:
- $lg(n+1) \geq h/2$ ou $h \leq 2lg(n+1)$

Árvores Rubro-Negras

Corolário:

As operações de Busca, Mínimo, Máximo, Sucessor e Predecessor podem ser implementadas em O(lgn).

Rotações



Inserir

- As operações Inserir e Remover são mais complicadas nas Árvores Rubro-Negras porque elas podem ferir alguma propriedade deste tipo de árvore.

Inserir

- As operações Inserir e Remover são mais complicadas nas Árvores Rubro-Negras porque elas podem ferir alguma propriedade deste tipo de árvore.
- Como veremos, essas operações podem ser implementadas de forma bastante parecida com as respectivas operações nas Árvores Binárias de Busca, bastando apenas modificar as cores dos nós para que as propriedades de Árvores Rubro-Negras sejam satisfeitas.

Inserir

- As operações Inserir e Remover são mais complicadas nas Árvores Rubro-Negras porque elas podem ferir alguma propriedade deste tipo de árvore.
- Como veremos, essas operações podem ser implementadas de forma bastante parecida com as respectivas operações nas Árvores Binárias de Busca, bastando apenas modificar as cores dos nós para que as propriedades de Árvores Rubro-Negras sejam satisfeitas.
- Como a inserção e a remoção propriamente ditas já foram vistas para Árvores Binárias de Busca, veremos apenas o que é necessário para acertar as cores da árvore.

Inserir

- Existem três casos para corrigir as cores após uma inserção:

Inserir

- Existem três casos para corrigir as cores após uma inserção:
- Caso 1: O tio do elemento inserido é VERMELHO.

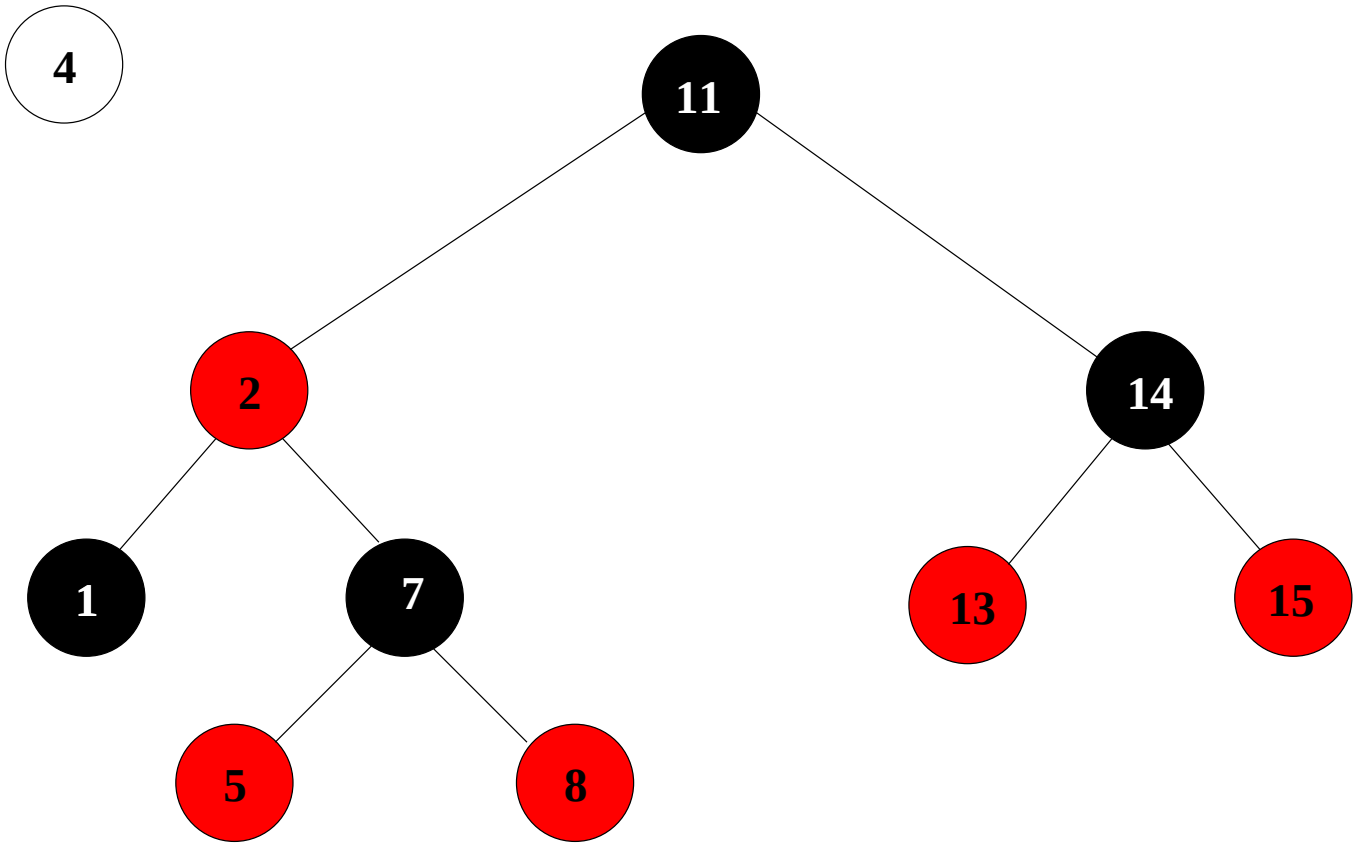
Inserir

- Existem três casos para corrigir as cores após uma inserção:
- Caso 1: O tio do elemento inserido é VERMELHO.
- Caso 2: O tio do elemento inserido é PRETO e o elemento inserido é um filho da direita.

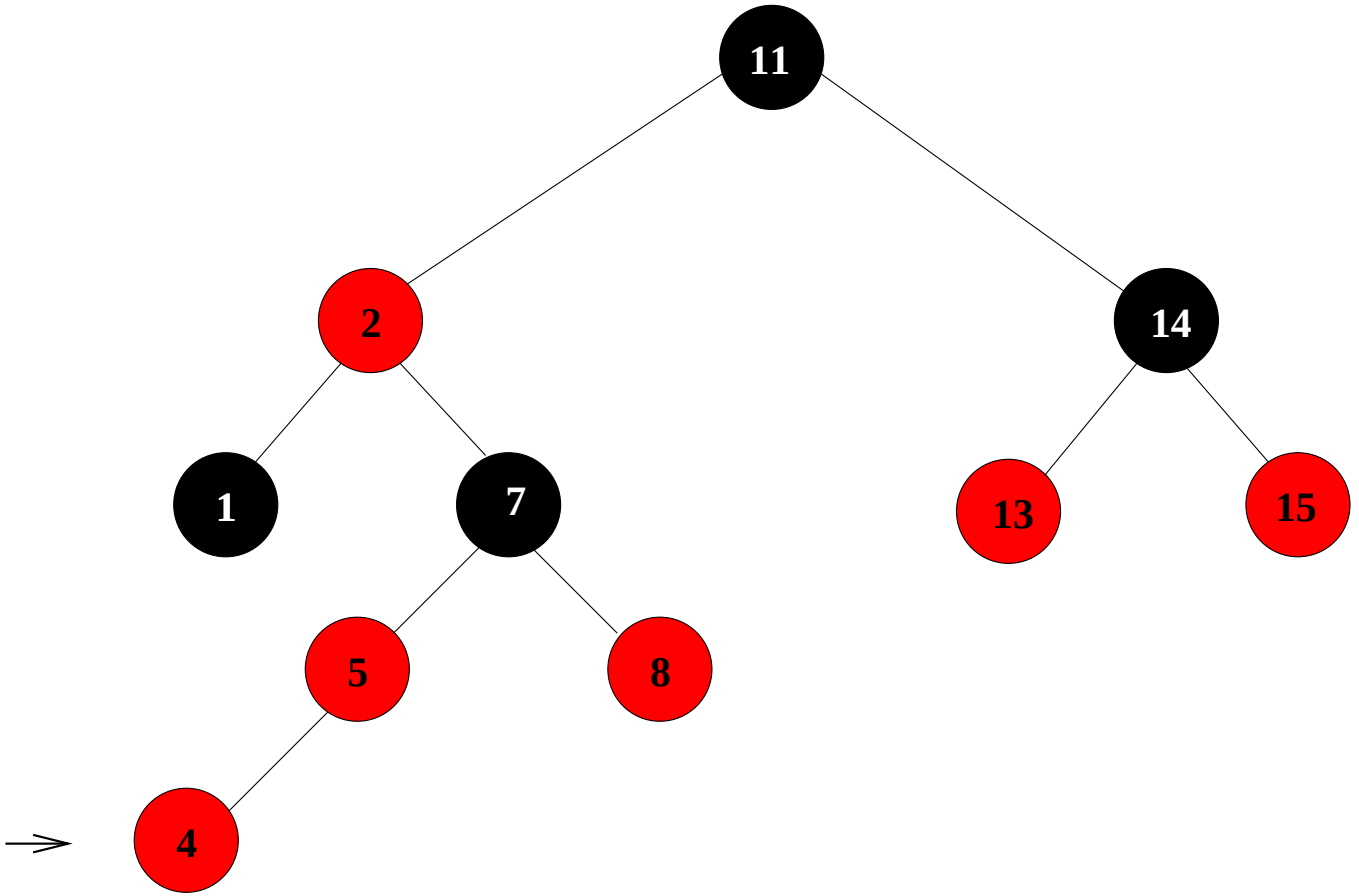
Inserir

- Existem três casos para corrigir as cores após uma inserção:
- Caso 1: O tio do elemento inserido é VERMELHO.
- Caso 2: O tio do elemento inserido é PRETO e o elemento inserido é um filho da direita.
- Caso 3: O tio do elemento inserido é PRETO e o elemento inserido é um filho da esquerda.

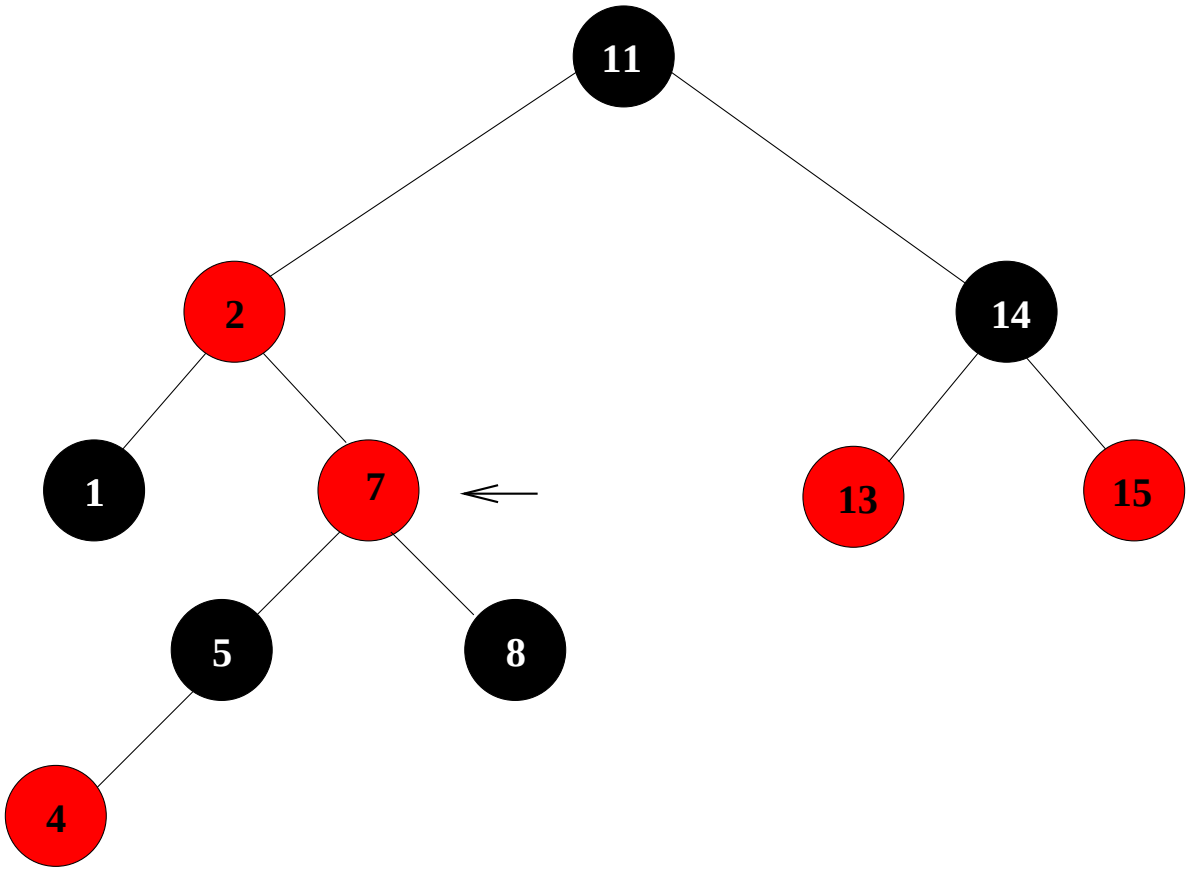
Inserir



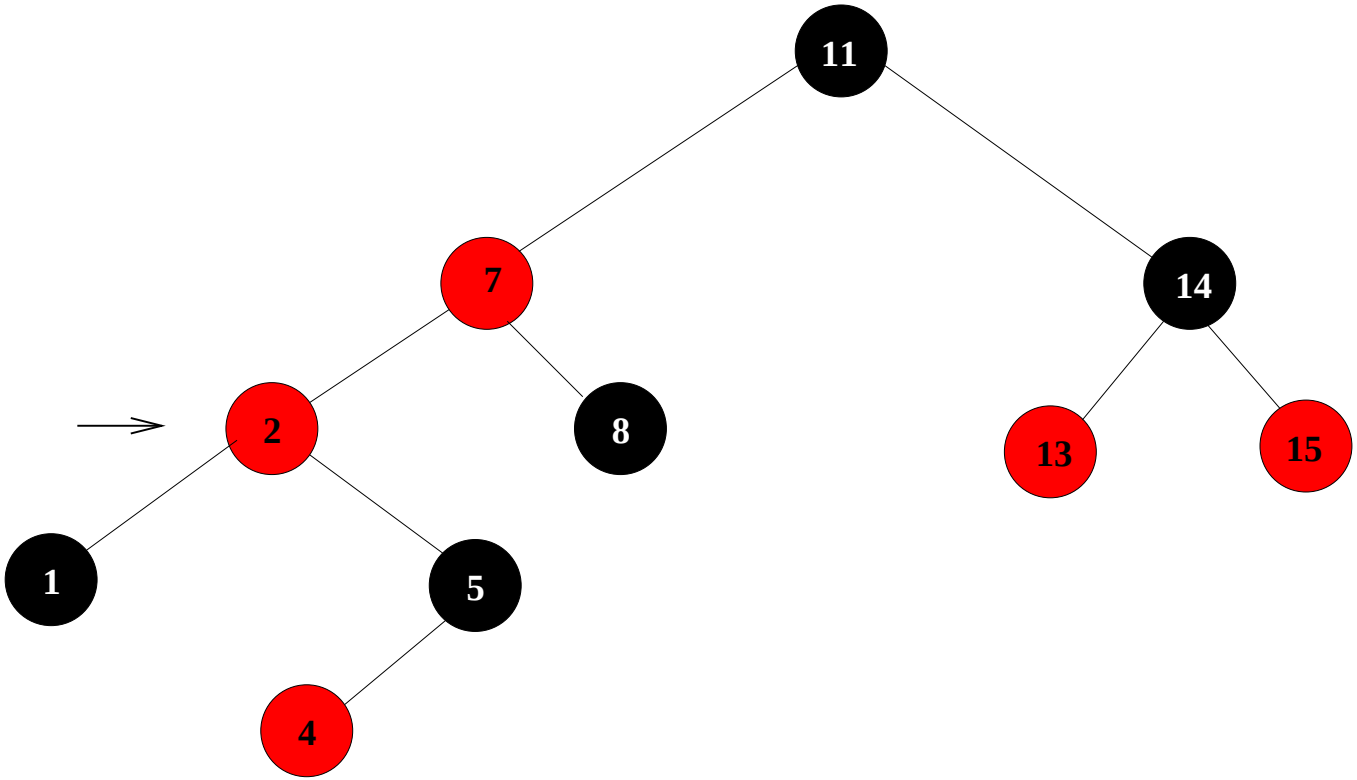
Caso 1: O tio do elemento inserido é VERMELHO



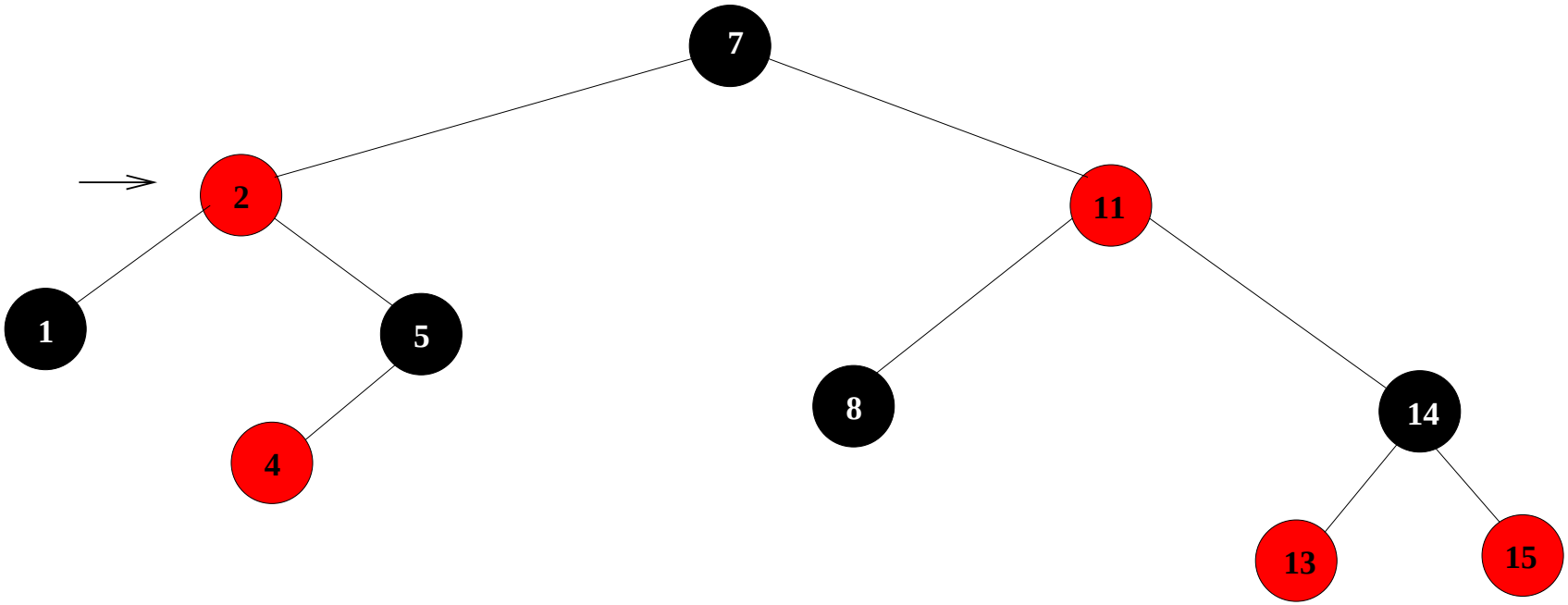
Caso 2: O tio do elemento inserido é PRETO e o elemento inserido é um filho da direita



Caso 3: O tio do elemento inserido é PRETO e o elemento inserido é um filho da esquerda



Inserir



Remover

- Existem quatro casos para corrigir as cores após uma remoção, mas antes de definí-los, identifiquemos alguns nós:

Remover

- Existem quatro casos para corrigir as cores após uma remoção, mas antes de definí-los, identifiquemos alguns nós:
- Seja z o nó a ser removido.

Remover

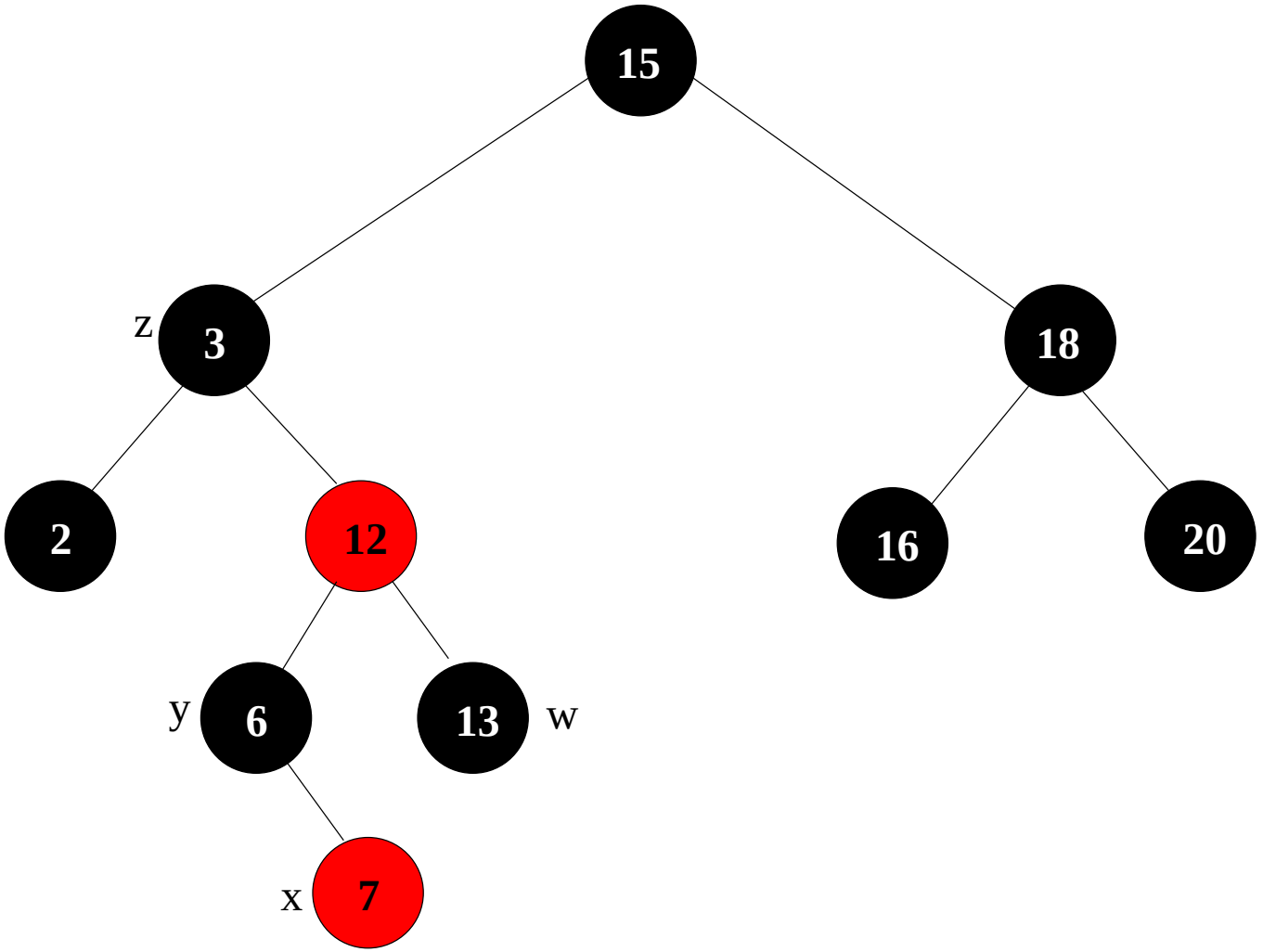
- Existem quatro casos para corrigir as cores após uma remoção, mas antes de definí-los, identifiquemos alguns nós:
- Seja z o nó a ser removido.
- Seja y = z se z possuia um ou nenhum filho ou y = succ(z) se z possuia dois filhos.

Remover

- Existem quatro casos para corrigir as cores após uma remoção, mas antes de definí-los, identifiquemos alguns nós:
- Seja z o nó a ser removido.
- Seja y = z se z possuia um ou nenhum filho ou y = succ(z) se z possuia dois filhos.
- Seja x o filho de y antes da remoção de z ou NIL caso y não possuisse filho.

Remover

- Existem quatro casos para corrigir as cores após uma remoção, mas antes de definí-los, identifiquemos alguns nós:
- Seja z o nó a ser removido.
- Seja y = z se z possuia um ou nenhum filho ou y = succ(z) se z possuia dois filhos.
- Seja x o filho de y antes da remoção de z ou NIL caso y não possuisse filho.
- Seja w o tio de x antes da remoção de z.


15
z
3
18
2
12
16
20
y
6
13
w
x
7

- Vamos considerar x sempre como sendo duplo-PRETO, a idéia é que o x recebe a “PRETIDÃO” do pai para manter a propriedade 5 (nos casos em que x é PRETO e VERMELHO basta fazer x simplesmente PRETO e todas as propriedades estarão respeitadas).

Remover

- Caso 1: w é VERMELHO.

Remover

- Caso 1: w é VERMELHO.
- Caso 2: w é PRETO, seus dois filhos são PRETOS.

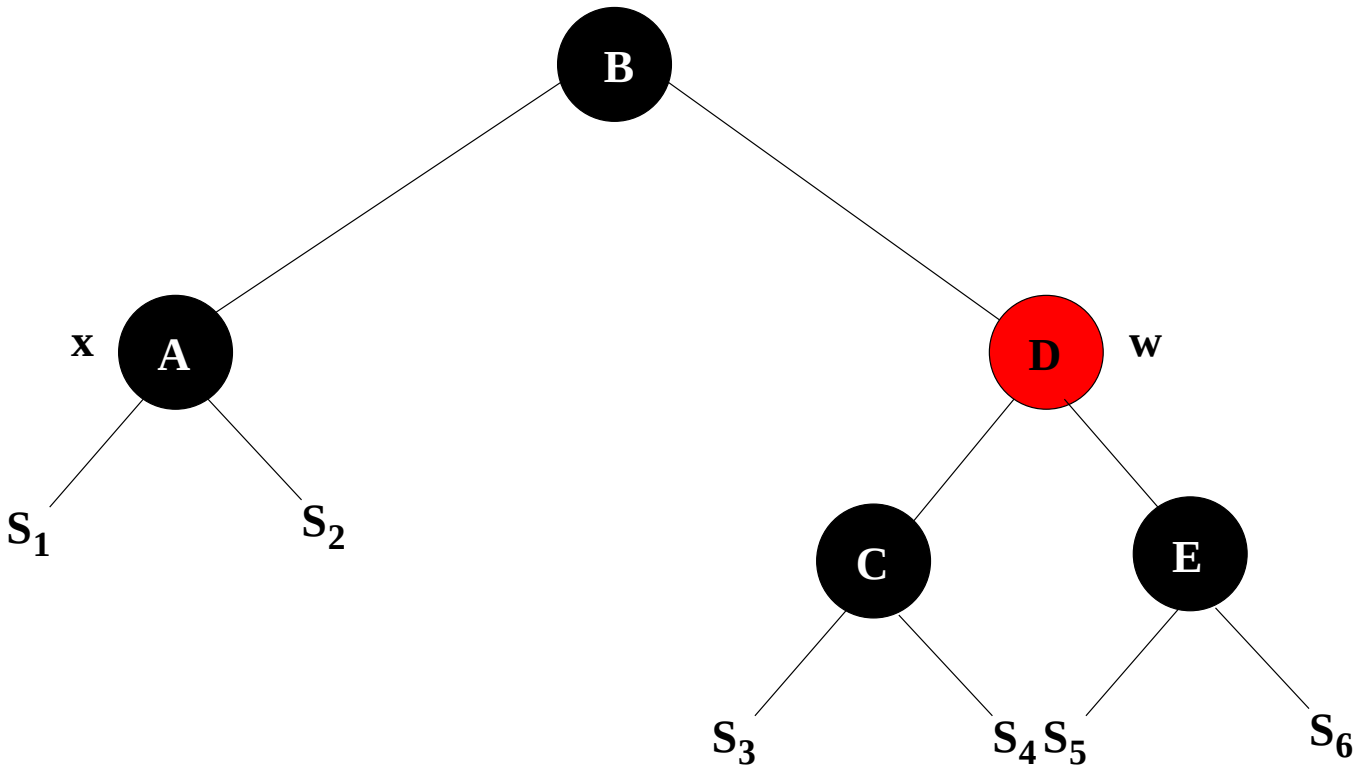
Remover

- Caso 1: w é VERMELHO.
- Caso 2: w é PRETO, seus dois filhos são PRETOS.
- Caso 3: w é PRETO, seu filho da esquerda é VERMELHO e se filho da direita é PRETO.

Remover

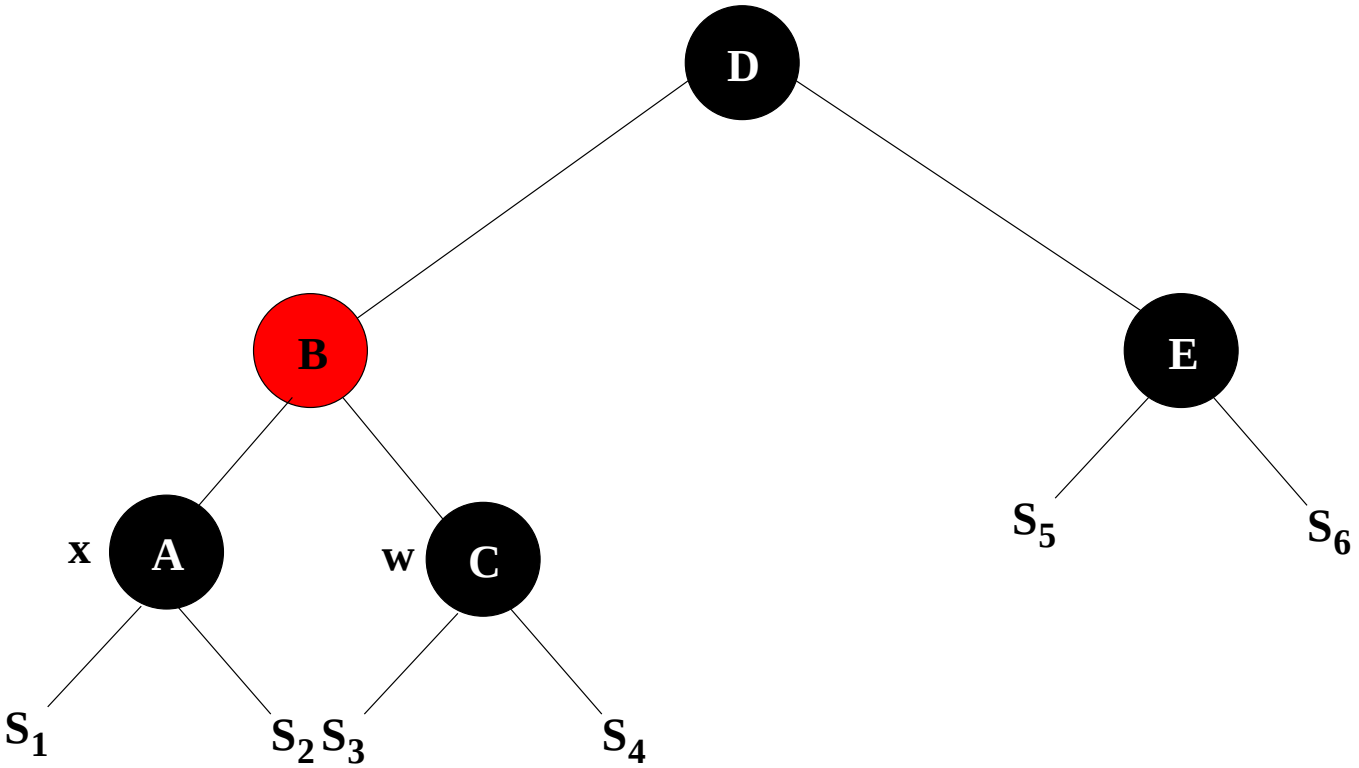
- Caso 1: w é VERMELHO.
- Caso 2: w é PRETO, seus dois filhos são PRETOS.
- Caso 3: w é PRETO, seu filho da esquerda é VERMELHO e se filho da direita é PRETO.
- Caso 4: w é PRETO e seu filho da direita é VERMELHO.

Caso 1: w é VERMELHO

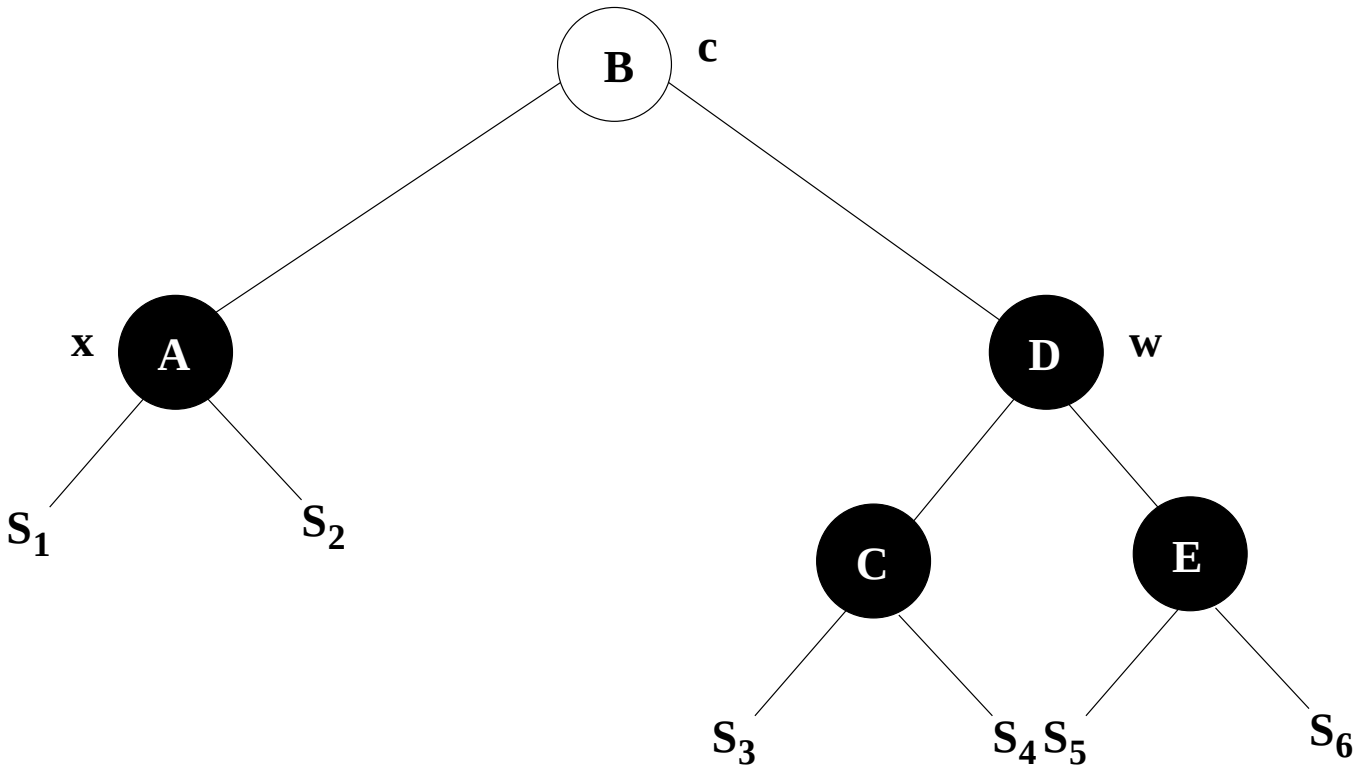


Caso 1: w é VERMELHO

Invertendo as cores do pai e de w obtemos:

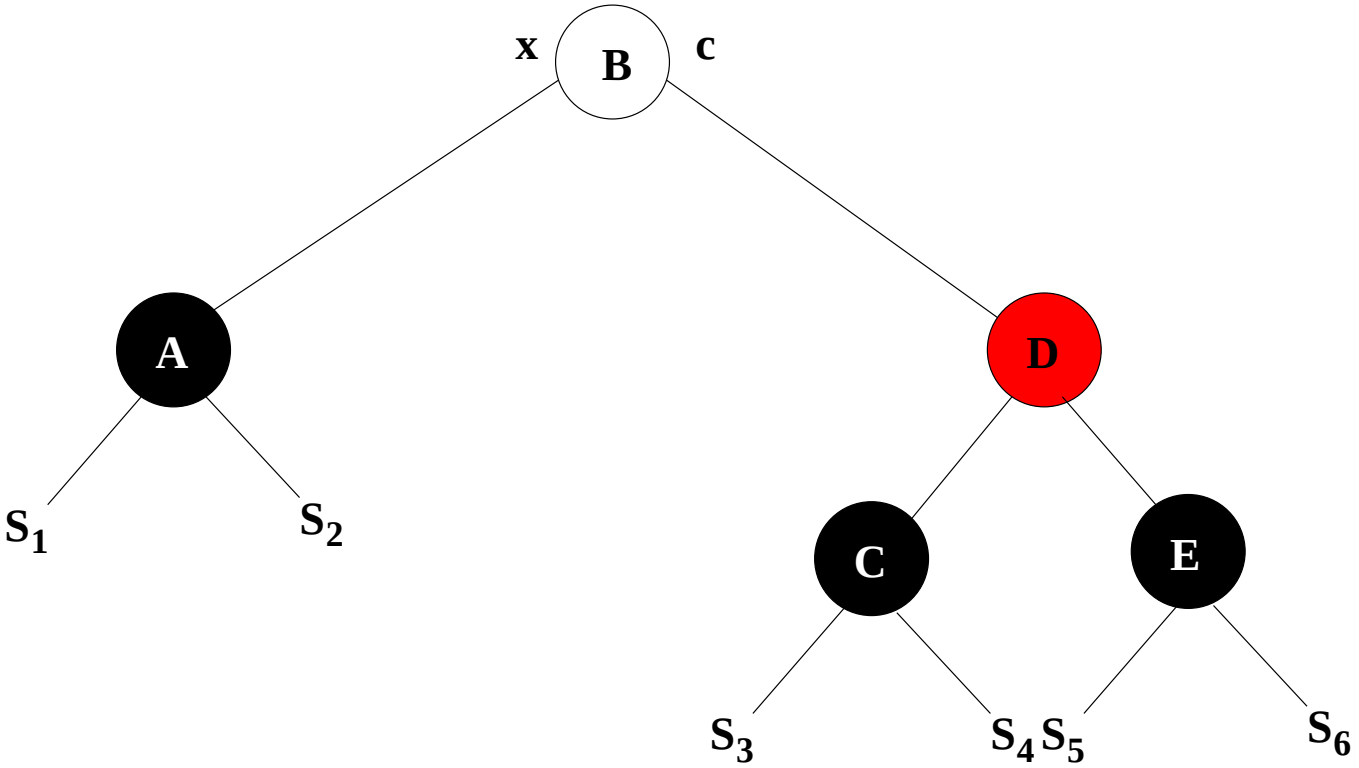


Caso 2: w é PRETO, seus dois filhos são PRETOS

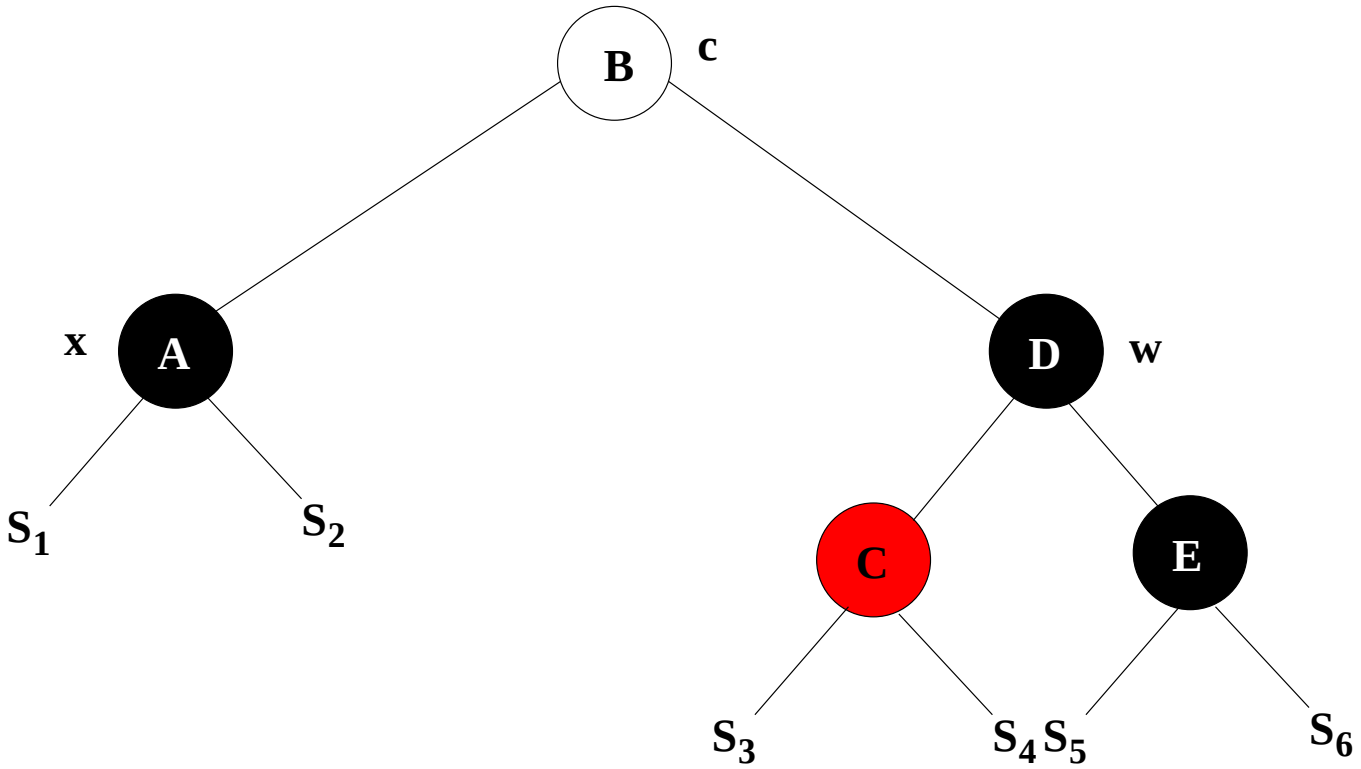


Caso 2: w é PRETO, seus dois filhos são PRETOS

"Removendo" um PRETO de x, fazendo w VERMELHO, "adicionando" PRETO ao pai e fazendo pai novo x:

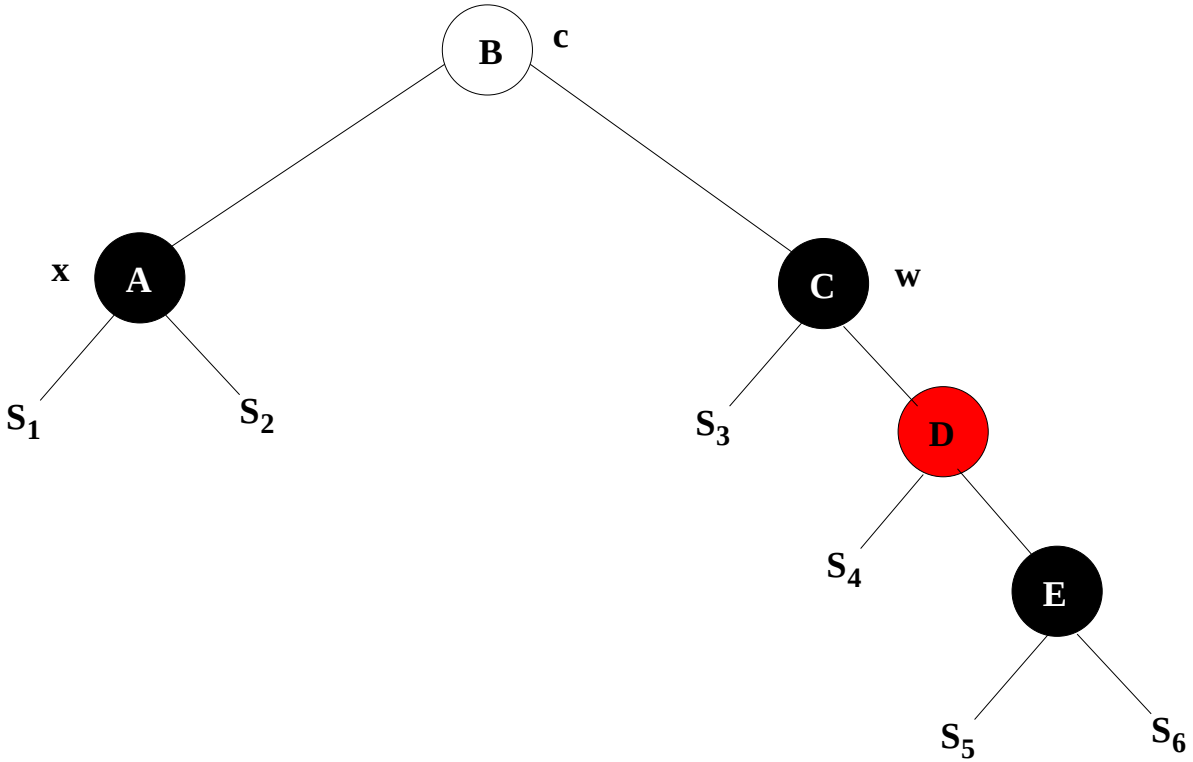


Caso 3: w é PRETO, seu filho da esquerda é VERMELHO e se filho da direita é PRETO

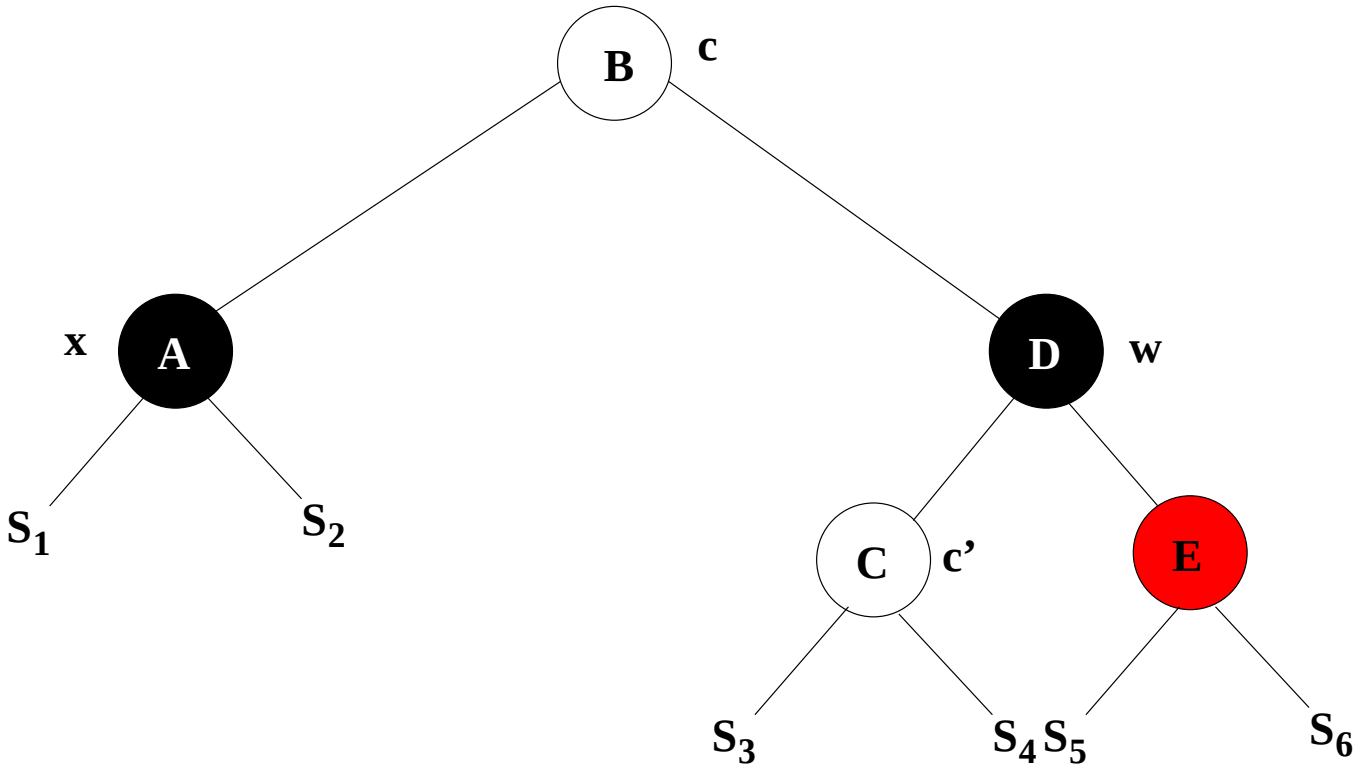


Caso 3: w é PRETO, seu filho da esquerda é VERMELHO e se filho da direita é PRETO

Invertendo as cores de w e do seu filho da esquerda e fazendo uma rotação para a direita:

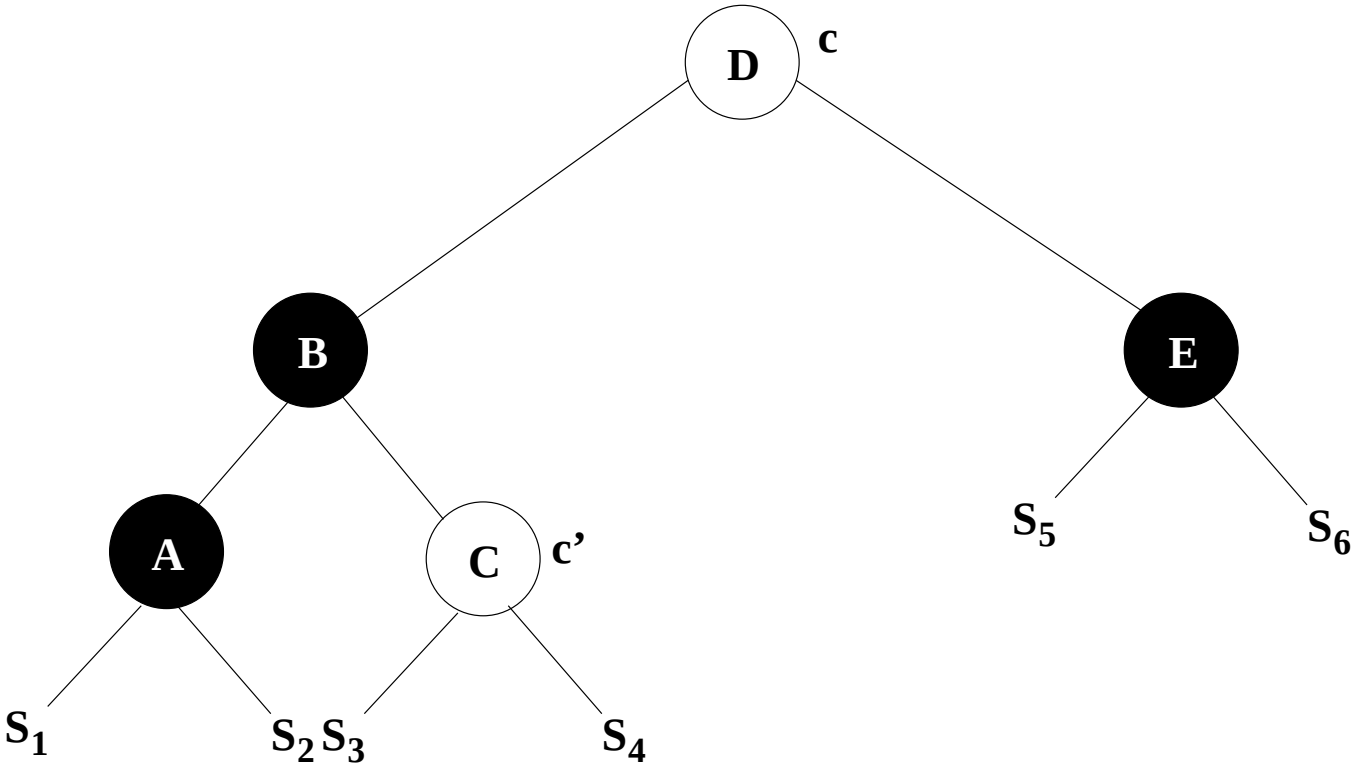


Caso 4: w é PRETO e seu filho da direita é VERMELHO



Caso 4: w é PRETO e seu filho da direita é VERMELHO

Atribuindo a cor do pai a w, fazendo pai PRETO, filho da direta de w PRETO e fazendo uma rotação para a esquerda, temos:



Análise

- As operações de inserção e remoção propriamente ditas gastam cada uma, tempo O(lgn), assim como as demais operações de Árvores Binárias de Busca, portanto, resta saber quanto gastam as operações de corrigir as cores em Inserir e Remover, porém é fácil perceber que cada operação não gasta mais que O(lgn) e, portanto, Inserir e Remover podem ser executadas em tempo O(lgn).

Sumário

Resumindo

Árvores Binárias de Busca:

- Para todo nó da árvore: nós menores ou iguais à esquerda e nós maiores ou iguais à direita.

Resumindo

Árvores Binárias de Busca:

- Para todo nó da árvore: nós menores ou iguais à esquerda e nós maiores ou iguais à direita.
- Operações: Busca, Mínimo, Máximo, Predecessor, Sucessor, Inserir e Remover.

Resumindo

Árvores Binárias de Busca:

- Para todo nó da árvore: nós menores ou iguais à esquerda e nós maiores ou iguais à direita.
- Operações: Busca, Mínimo, Máximo, Predecessor, Sucessor, Inserir e Remover.
- Inserir: basta manter a propriedade básica das Árvores Binárias de Busca. Insere sempre como uma folha.

Resumindo

Árvores Binárias de Busca:

- Para todo nó da árvore: nós menores ou iguais à esquerda e nós maiores ou iguais à direita.
- Operações: Busca, Mínimo, Máximo, Predecessor, Sucessor, Inserir e Remover.
- Inserir: basta manter a propriedade básica das Árvores Binárias de Busca. Insere sempre como uma folha.
- Remover: três casos dependendo de quantos filhos possui o nó.
  - nenhum filho: simplesmente remove o nó.
  - um filho: o filho passa a ser filho do avô.
  - dois filhos: encontrar o sucessor para assumir o lugar do nó.

Resumindo

Árvores Binárias de Busca:

- Para todo nó da árvore: nós menores ou iguais à esquerda e nós maiores ou iguais à direita.
- Operações: Busca, Mínimo, Máximo, Predecessor, Sucessor, Inserir e Remover.
- Inserir: basta manter a propriedade básica das Árvores Binárias de Busca. Insere sempre como uma folha.
- Remover: três casos dependendo de quantos filhos possui o nó.
  - nenhum filho: simplesmente remove o nó.
  - um filho: o filho passa a ser filho do avô.
  - dois filhos: encontrar o sucessor para assumir o lugar do nó.
- Complexidade das operações: O(h).

Resumindo

Árvores Rubro-Negras:

- Árvores Binárias de Busca com bit adicional para representar a cor (PRETA ou VERMELHA).

Resumindo

Árvores Rubro-Negras:

- Árvores Binárias de Busca com bit adicional para representar a cor (PRETA ou VERMELHA).
- Cinco propriedades básicas:
  - Todo nó é colorido PRETO ou VERMELHO.
  - A raiz é PRETA.
  - As folhas (NIL) são PRETAS.
  - Se um nó é VERMELHO, seus filhos são PRETOS.
  - Para cada nó, todos os caminhos até as folhas descendentes contém o mesmo número de nós.

Resumindo

Árvores Rubro-Negras:

- Árvores Binárias de Busca com bit adicional para representar a cor (PRETA ou VERMELHA).
- Cinco propriedades básicas:
  - Todo nó é colorido PRETO ou VERMELHO.
  - A raiz é PRETA.
  - As folhas (NIL) são PRETAS.
  - Se um nó é VERMELHO, seus filhos são PRETOS.
  - Para cada nó, todos os caminhos até as folhas descendentes contém o mesmo número de nós.
- Lema: Uma Árvore Rubro-Negra com n nós internos tem altura no máximo $2\lg(n+1)$.

Resumindo

Árvores Rubro-Negras:

- Árvores Binárias de Busca com bit adicional para representar a cor (PRETA ou VERMELHA).
- Cinco propriedades básicas:
  - Todo nó é colorido PRETO ou VERMELHO.
  - A raiz é PRETA.
  - As folhas (NIL) são PRETAS.
  - Se um nó é VERMELHO, seus filhos são PRETOS.
  - Para cada nó, todos os caminhos até as folhas descendentes contém o mesmo número de nós.
- Lema: Uma Árvore Rubro-Negra com n nós internos tem altura no máximo 2lg(n+1).
- Possui todas as operações que uma árvore binária balanceada comum possui.

Resumindo

Árvores Rubro-Negras:

- Inserir: tês casos a considerar.
  - O tio é VERMELHO: inverte cores do avô, pai e tio.
  - O tio é PRETO e filho da direita: faz rotação para a esquerda.
  - O tio é PRETO e filho da esquerda: inverte cores do avô e do pai e faz rotação para a direita.

Resumindo

Árvores Rubro-Negras:

- Remover: quatro casos a considerar.
  - w é VERMELHO: inverte cores do pai e de w.
  - w é PRETO e ambos os filhos de w são pretos: “remove” um PRETO de x, faz w VERMELHO e “adiciona” PRETO ao pai. Faz pai novo x.
  - w é PRETO, o filho da esquerda é VERMELHO e o da direita PRETO: inverte cores de w e do seu filho da esquerda e faz uma rotação para a direita.
  - w é PRETO e o filho da direita é VERMELHO: atribui a cor do pai a w, faz pai PRETO, faz filho da direta de w PRETO e faz uma rotação para a esquerda.

Resumindo

Árvores Rubro-Negras:

- Remover: quatro casos a considerar.
  - w é VERMELHO: inverte cores do pai e de w.
  - w é PRETO e ambos os filhos de w são pretos: “remove” um PRETO de x, faz w VERMELHO e “adiciona” PRETO ao pai. Faz pai novo x.
  - w é PRETO, o filho da esquerda é VERMELHO e o da direita PRETO: inverte cores de w e do seu filho da esquerda e faz uma rotação para a direita.
  - w é PRETO e o filho da direita é VERMELHO: atribui a cor do pai a w, faz pai PRETO, faz filho da direta de w PRETO e faz uma rotação para a esquerda.
- Complexidade das operações: O(lgn).

Exercícios:

Questão 12.3-5:

A operação de remoção é “comutativa” no sentido que removendo x e depois y de uma árvore binária de busca resulta na mesma árvore que removendo y e depois x? Argumente porque é ou dê um contra-exemplo.

Exercícios:

Questão 13.3-5:

Considere uma árvore rubro-negra formada pela inserção de n nós com a operação Inserir. Argumente que se $n > 1$, a árvore terá pelo menos um nó VERMELHO.

Exercícios:

Questão 13.4-6:

Os professores Skelton e Baron então preocupados que no início do caso 1 do procedimento para acertar as cores após uma remoção, o pai do nó x pode não ser PRETO. Se os professores estiverem corretos, então o procedimento pode estar errado (pois w é colorido VERMELHO). Mostre que o pai do nó x deve ser PRETO no início do caso 1, para que os professores não tenham nada com que se preocupar.