AED III
Árvores Rubro Negra

Ciência da Computação – IFSULDEMINAS

Primeiro Semestre de 2014

Roteiro

Características

- A árvore rubro-negra tem esse nome devido a *coloração* de seus nós.
- Uma árvore rubro negra é uma árvore binária de busca com um campo adicional na estrutura do nó: sua cor — VERMELHO ou **PRETO**.
- A coloração do nó é usada como fator de balanceamento da árvore Rubro Negra.
- **Árvores Aproximadamente Balanceadas**.

Características

- A árvore rubro-negra tem esse nome devido a *coloração* de seus nós.
- Uma árvore rubro negra é uma árvore binária de busca com um campo adicional na estrutura do nó: sua cor — VERMELHO ou **PRETO**.
- A coloração do nó é usada como fator de balanceamento da árvore Rubro Negra.
- **Árvores Aproximadamente Balanceadas**.

Características

- A árvore rubro-negra tem esse nome devido a *coloração* de seus nós.
- Uma árvore rubro negra é uma árvore binária de busca com um campo adicional na estrutura do nó: sua cor — VERMELHO ou **PRETO**.
- A coloração do nó é usada como fator de balanceamento da árvore Rubro Negra.
- Árvores Aproximadamente Balanceadas.

Características

- A árvore rubro-negra tem esse nome devido a *coloração* de seus nós.
- Uma árvore rubro negra é uma árvore binária de busca com um campo adicional na estrutura do nó: sua cor — VERMELHO ou **PRETO**.
- A coloração do nó é usada como fator de balanceamento da árvore Rubro Negra.
- **Árvores Aproximadamente Balanceadas**.

Características

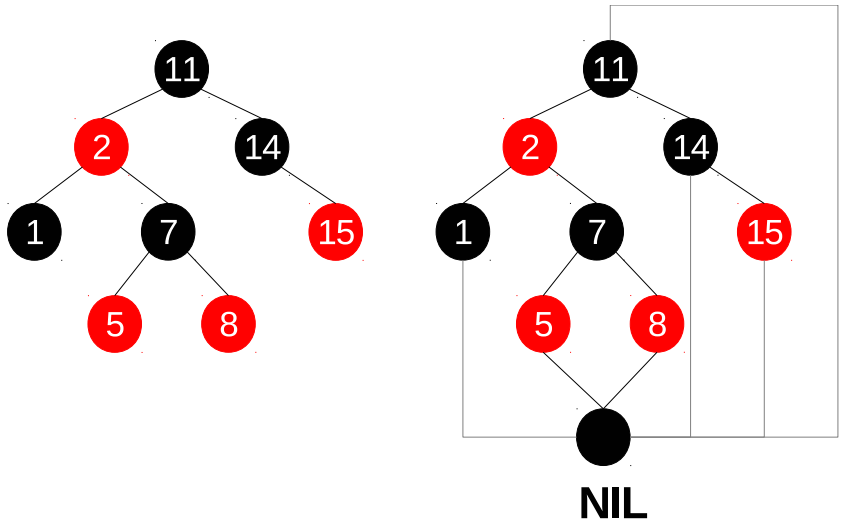
- Cada nó passa a ter: *cor*, *chave*, *esquerda*, *direita* e *pai*.
- Se um filho ou pai de um nó não existir faremos estes ponteiros apontarem para um nó especial, denominado NIL.

Características

- Cada nó passa a ter: *cor*, *chave*, *esquerda*, *direita* e *pai*.
- Se um filho ou pai de um nó não existir faremos estes ponteiros apontarem para um nó especial, denominado NIL.

Características

- Cada nó passa a ter: *cor*, *chave*, *esquerda*, *direita* e *pai*.
- Se um filho ou pai de um nó não existir faremos estes ponteiros apontarem para um nó especial, denominado NIL.



Propriedades

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Considera-se os nós NIL como nós fictícios. Estes nós sempre são folhas, não contém chaves e sua cor é preta.

Propriedades

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Considera-se os nós NIL como nós fictícios. Estes nós sempre são folhas, não contém chaves e sua cor é preta.

Propriedades

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Considera-se os nós NIL como nós fictícios. Estes nós sempre são folhas, não contém chaves e sua cor é preta.

Propriedades

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Considera-se os nós NIL como nós fictícios. Estes nós sempre são folhas, não contém chaves e sua cor é preta.

Propriedades

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Considera-se os nós NIL como nós fictícios. Estes nós sempre são folhas, não contém chaves e sua cor é preta.

Propriedades

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Considera-se os nós NIL como nós fictícios. Estes nós sempre são folhas, não contém chaves e sua cor é preta.

Altura Negra

Altura Negra

A altura negra de uma árvore rubro-negra A, é o número de nós negros que se encontram nos caminhos da raiz até uma folha.

Exemplo:

- Qual a altura negra do nó 7?

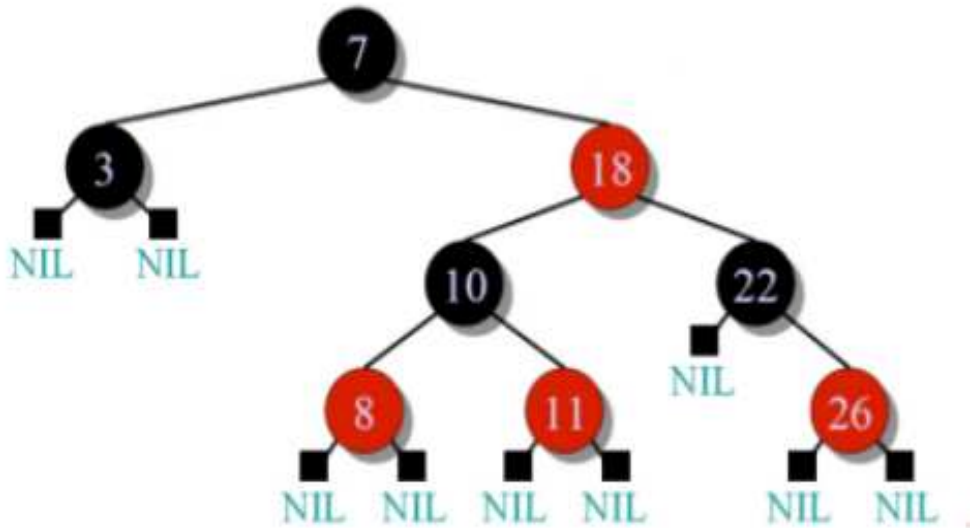
Altura Negra

Altura Negra

A altura negra de uma árvore rubro-negra A, é o número de nós negros que se encontram nos caminhos da raiz até uma folha.

Exemplo:

- Qual a altura negra do nó 7?



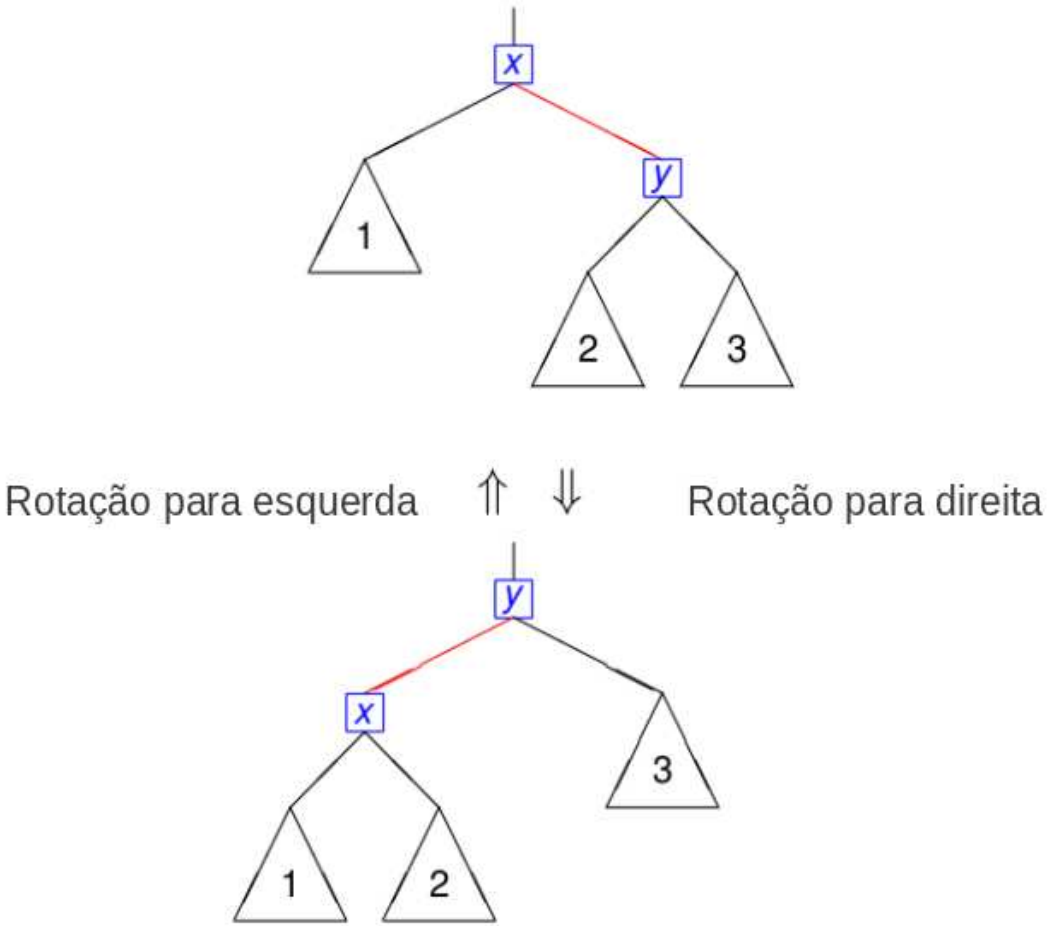
Estrutura para Árvore Rubro Negra

Nó

```
Nó:
    chave valor armazenado no nó;
    cor cor do nó (vermelho ou preto);
    direita ponteiro para o filho da direita;
    esquerda ponteiro para o filho da esquerda;
    pai ponteiro para o pai;
```

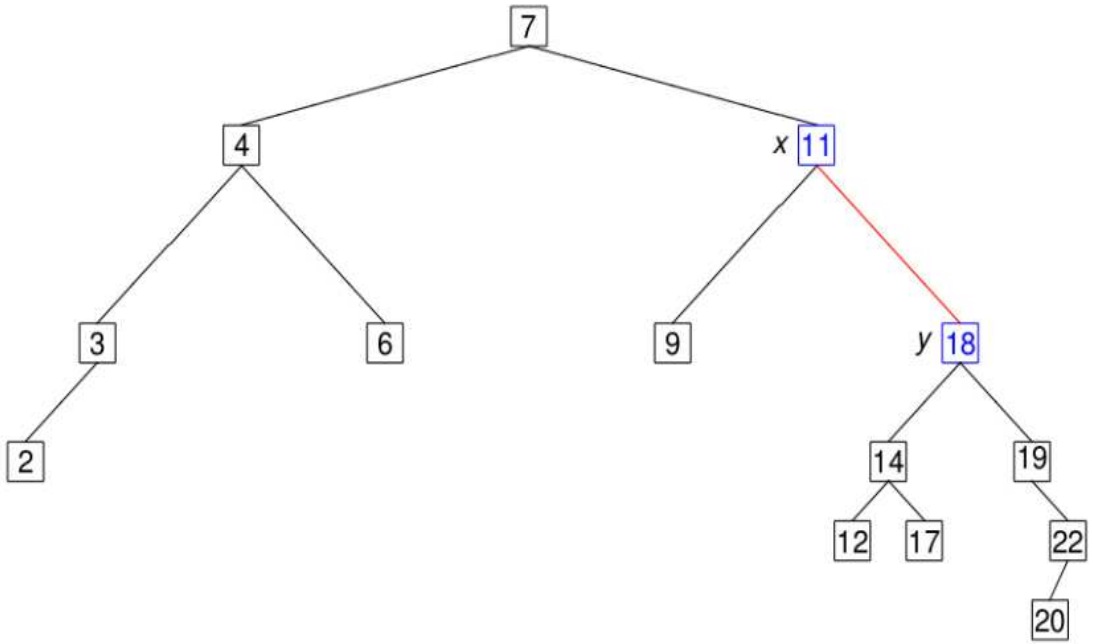
Rotações

As rotações para a direita e para a esquerda são utilizadas no balanceamento das árvores.



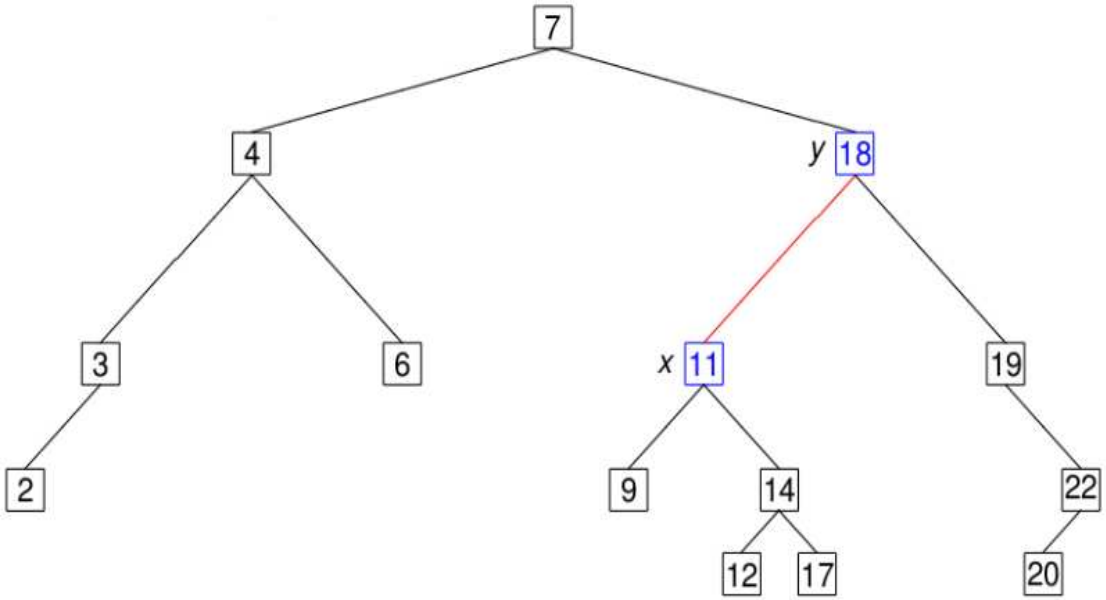
Exemplo Rotações

Antes:



Exemplo Rotações

Depois:



Algoritmo Rotação Esquerda

Rotação Esquerda

```
LEFT_ROTATE(T, x) {
    y = x.direita;
    x.direita = y.esquerda;

    if y.esquerda != NIL            // ajusta pai de x
        y.esquerda.pai = x;

    y.pai = x.pai;                  // ajusta pai de y

    if x.pai == NIL                 // x é raiz
        T.raiz = y;
    else if x == x.pai.esquerda     // x é filho esquerdo
        x.pai.esquerda = y;
    else                            // x é filho direito
        x.pai.direita = y;

    y.esquerda = x;
    x.pai = y;
```

Inserção em Árvore Rubro Negra

- Dados o ponteiro para a raiz, T, e um nó z.
- Obs: o nó z já foi alocado e seus valores são:
  - z.chave = v
  - z.esquerda = z.direita = z.pai = NIL
  - z.cor = VERMELHO
- x aponta para a raiz, começando da raiz da árvore, x traça um caminho descendente em busca do local de inserção de z.
- y é um ponteiro acompanhante, no percurso é o pai de x.

Inserção em Árvore Rubro Negra

- Dados o ponteiro para a raiz, T, e um nó z.
- Obs: o nó z já foi alocado e seus valores são:
  - z.chave = v
  - z.esquerda = z.direita = z.pai = NIL
  - z.cor = VERMELHO
- x aponta para a raiz, começando da raiz da árvore, x traça um caminho descendente em busca do local de inserção de z.
- y é um ponteiro acompanhante, no percurso é o pai de x.

Inserção em Árvore Rubro Negra

- Dados o ponteiro para a raiz, T, e um nó z.
- Obs: o nó z já foi alocado e seus valores são:
  - z.chave = v
  - z.esquerda = z.direita = z.pai = NIL
  - z.cor = VERMELHO
- x aponta para a raiz, começando da raiz da árvore, x traça um caminho descendente em busca do local de inserção de z.
- y é um ponteiro acompanhante, no percurso é o pai de x.

Inserção em Árvore Rubro Negra

- Dados o ponteiro para a raiz, T, e um nó z.
- Obs: o nó z já foi alocado e seus valores são:
  - z.chave = v
  - z.esquerda = z.direita = z.pai = NIL
  - z.cor = VERMELHO
- x aponta para a raiz, começando da raiz da árvore, x traça um caminho descendente em busca do local de inserção de z.
- y é um ponteiro acompanhante, no percurso é o pai de x.

Inserção em Árvore Rubro Negra

- Dados o ponteiro para a raiz, T, e um nó z.
- Obs: o nó z já foi alocado e seus valores são:
  - z.chave = v
  - z.esquerda = z.direita = z.pai = NIL
  - z.cor = VERMELHO
- x aponta para a raiz, começando da raiz da árvore, x traça um caminho descendente em busca do local de inserção de z.
- y é um ponteiro acompanhante, no percurso é o pai de x.

Inserção em Árvore Rubro Negra

- Dados o ponteiro para a raiz, T, e um nó z.
- Obs: o nó z já foi alocado e seus valores são:
  - z.chave = v
  - z.esquerda = z.direita = z.pai = NIL
  - z.cor = VERMELHO
- x aponta para a raiz, começando da raiz da árvore, x traça um caminho descendente em busca do local de inserção de z.
- y é um ponteiro acompanhante, no percurso é o pai de x.

Inserção em Árvore Rubro Negra

- Dados o ponteiro para a raiz, T, e um nó z.
- Obs: o nó z já foi alocado e seus valores são:
  - z.chave = v
  - z.esquerda = z.direita = z.pai = NIL
  - z.cor = VERMELHO
- x aponta para a raiz, começando da raiz da árvore, x traça um caminho descendente em busca do local de inserção de z.
- y é um ponteiro acompanhante, no percurso é o pai de x.

Inserção em Árvore Rubro Negra

Inserção

```
RB_INSERT(T, z) {
    y = NIL;
    x = T;

    while x != NIL              // busca local da inserção
        y = x;
        if z.chave < x.chave
            x = x.esquerda;
        else x = x.direita;

    z.pai = y;                  // ajusta ponteiro para o pai

    if y == NIL                 // ajusta ponteiro para o novo nó
        T.raiz = z;
    else if z.chave < y.chave
        y.esquerda = z;
    else y.direita = z;
```

Inserção em Árvore Rubro Negra

Inserção

```
    z.cor = VERMELHO;
    RB_INSERT_FIXUP(T, z);
}
```

- Sempre inserimos nós vermelhos.
- Após a inserção devemos balancear a árvore.

Inserção em Árvore Rubro Negra

Inserção

```
    z.cor = VERMELHO;
    RB_INSERT_FIXUP(T, z);
}
```

- Sempre inserimos nós vermelhos.
- Após a inserção devemos balancear a árvore.

Inserção em Árvore Rubro Negra

Inserção

```
    z.cor = VERMELHO;
    RB_INSERT_FIXUP(T, z);
}
```

- Sempre inserimos nós vermelhos.
- Após a inserção devemos balancear a árvore.

Propriedades Violadas

Após a inserção, mas antes da execução a função RB_INSERE_FIXUP, apenas as seguintes propriedades podem ser violadas:

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Propriedades Violadas

Após a inserção, mas antes da execução a função RB_INSERE_FIXUP, apenas as seguintes propriedades podem ser violadas:

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Propriedades Violadas

Após a inserção, mas antes da execução a função RB_INSERE_FIXUP, apenas as seguintes propriedades podem ser violadas:

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Propriedades Violadas

Após a inserção, mas antes da execução a função RB_INSERE_FIXUP, apenas as seguintes propriedades podem ser violadas:

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Propriedades Violadas

Após a inserção, mas antes da execução a função RB_INSERE_FIXUP, apenas as seguintes propriedades podem ser violadas:

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

A função RB_INSERE_FIXUP

- A propriedade 1 é violada quando a árvore é vazia.
- A propriedade 4 é violada quando o nó a ser inserido (z) e seu pai (z.pai) são vermelhos.
- A função RB_INSERE_FIXUP corrige as cores dos nós e faz rotações na árvore.
- Quando a propriedade 4 é violada, a função RB_INSERE_FIXUP faz o balanceamento a partir do pai do nó recém inserido (z):
  - Levando em consideração se o pai de z é filho da direita ou da esquerda.
  - Há três casos para cada situação.

Obs: analisaremos os casos em que o pai de z é filho à esquerda.

A função RB_INSERE_FIXUP

- A propriedade 1 é violada quando a árvore é vazia.
- A propriedade 4 é violada quando o nó a ser inserido (z) e seu pai (z.pai) são vermelhos.
- A função RB_INSERE_FIXUP corrige as cores dos nós e faz rotações na árvore.
- Quando a propriedade 4 é violada, a função RB_INSERE_FIXUP faz o balanceamento a partir do pai do nó recém inserido (z):
  - Levando em consideração se o pai de z é filho da direita ou da esquerda.
  - Há três casos para cada situação.

Obs: analisaremos os casos em que o pai de z é filho à esquerda.

A função RB_INSERE_FIXUP

- A propriedade 1 é violada quando a árvore é vazia.
- A propriedade 4 é violada quando o nó a ser inserido (z) e seu pai (z.pai) são vermelhos.
- A função RB_INSERE_FIXUP corrige as cores dos nós e faz rotações na árvore.
- Quando a propriedade 4 é violada, a função RB_INSERE_FIXUP faz o balanceamento a partir do pai do nó recém inserido (z):
  - Levando em consideração se o pai de z é filho da direita ou da esquerda.
  - Há três casos para cada situação.

Obs: analisaremos os casos em que o pai de z é filho à esquerda.

A função RB_INSERE_FIXUP

- A propriedade 1 é violada quando a árvore é vazia.
- A propriedade 4 é violada quando o nó a ser inserido (z) e seu pai (z.pai) são vermelhos.
- A função RB_INSERE_FIXUP corrige as cores dos nós e faz rotações na árvore.
- Quando a propriedade 4 é violada, a função RB_INSERE_FIXUP faz o balanceamento a partir do pai do nó recém inserido (z):
  - Levando em consideração se o pai de z é filho da direita ou da esquerda.
  - Há três casos para cada situação.

Obs: analisaremos os casos em que o pai de z é filho à esquerda.

A função RB_INSERE_FIXUP

- A propriedade 1 é violada quando a árvore é vazia.
- A propriedade 4 é violada quando o nó a ser inserido (z) e seu pai (z.pai) são vermelhos.
- A função RB_INSERE_FIXUP corrige as cores dos nós e faz rotações na árvore.
- Quando a propriedade 4 é violada, a função RB_INSERE_FIXUP faz o balanceamento a partir do pai do nó recém inserido (z):
  - Levando em consideração se o pai de z é filho da direita ou da esquerda.
  - Há três casos para cada situação.

Obs: analisaremos os casos em que o pai de z é filho à esquerda.

A função RB_INSERE_FIXUP

- A propriedade 1 é violada quando a árvore é vazia.
- A propriedade 4 é violada quando o nó a ser inserido (z) e seu pai (z.pai) são vermelhos.
- A função RB_INSERE_FIXUP corrige as cores dos nós e faz rotações na árvore.
- Quando a propriedade 4 é violada, a função RB_INSERE_FIXUP faz o balanceamento a partir do pai do nó recém inserido (z):
  - Levando em consideração se o pai de z é filho da direita ou da esquerda.
  - Há três casos para cada situação.

Obs: analisaremos os casos em que o pai de z é filho à esquerda.

A função RB_INSERE_FIXUP

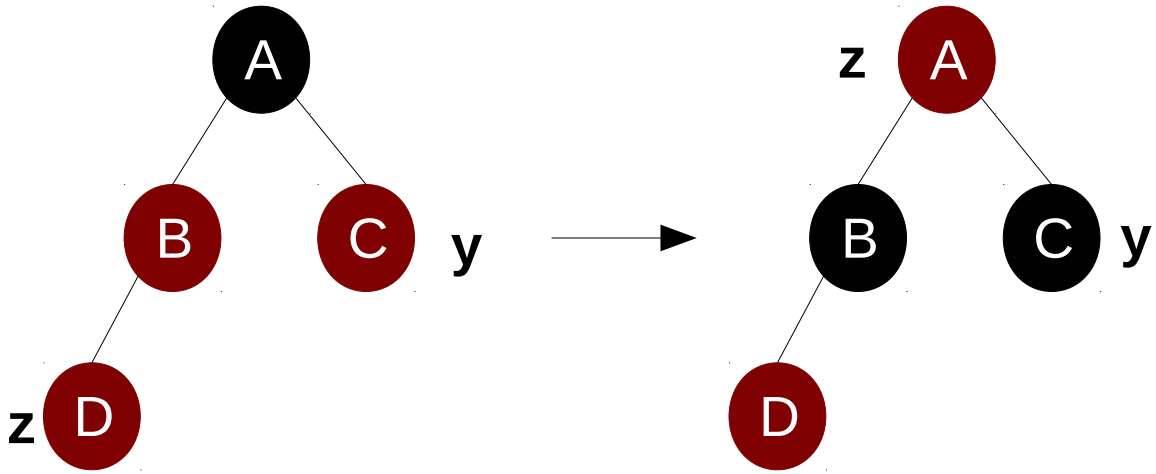
- A propriedade 1 é violada quando a árvore é vazia.
- A propriedade 4 é violada quando o nó a ser inserido (z) e seu pai (z.pai) são vermelhos.
- A função RB_INSERE_FIXUP corrige as cores dos nós e faz rotações na árvore.
- Quando a propriedade 4 é violada, a função RB_INSERE_FIXUP faz o balanceamento a partir do pai do nó recém inserido (z):
  - Levando em consideração se o pai de z é filho da direita ou da esquerda.
  - Há três casos para cada situação.

Obs: analisaremos os casos em que o pai de z é filho à esquerda.

Caso 1

Caso 1: o tio y de z é vermelho

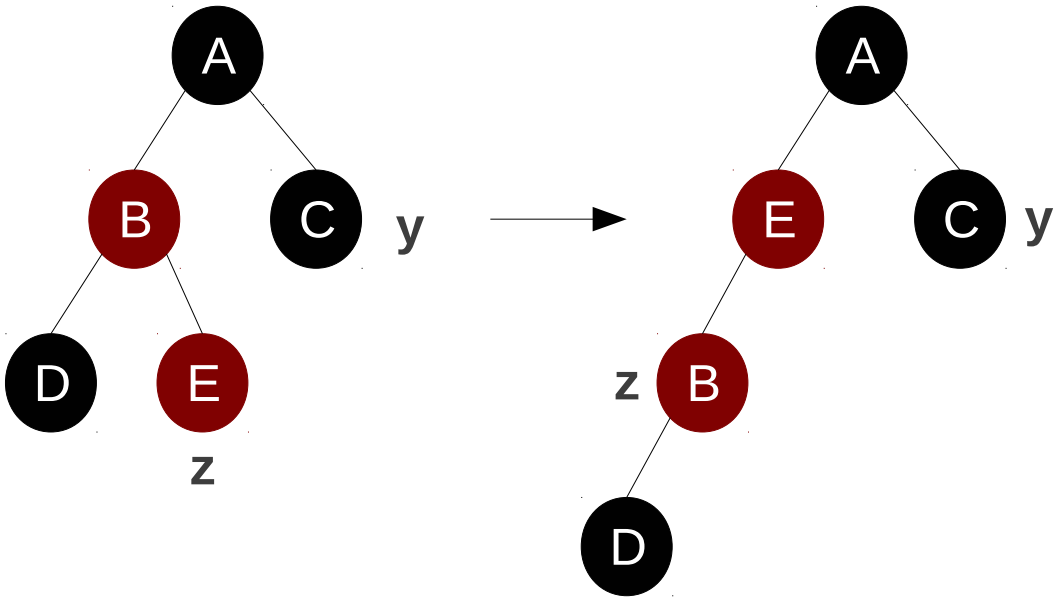
- Colorimos o pai de z e o seu tio y de pretos e o avô de z de vermelho.
- O avô de z passa a ser o novo z.



Caso 2

Caso 2: o tio y de z é preto e z é filho à direita

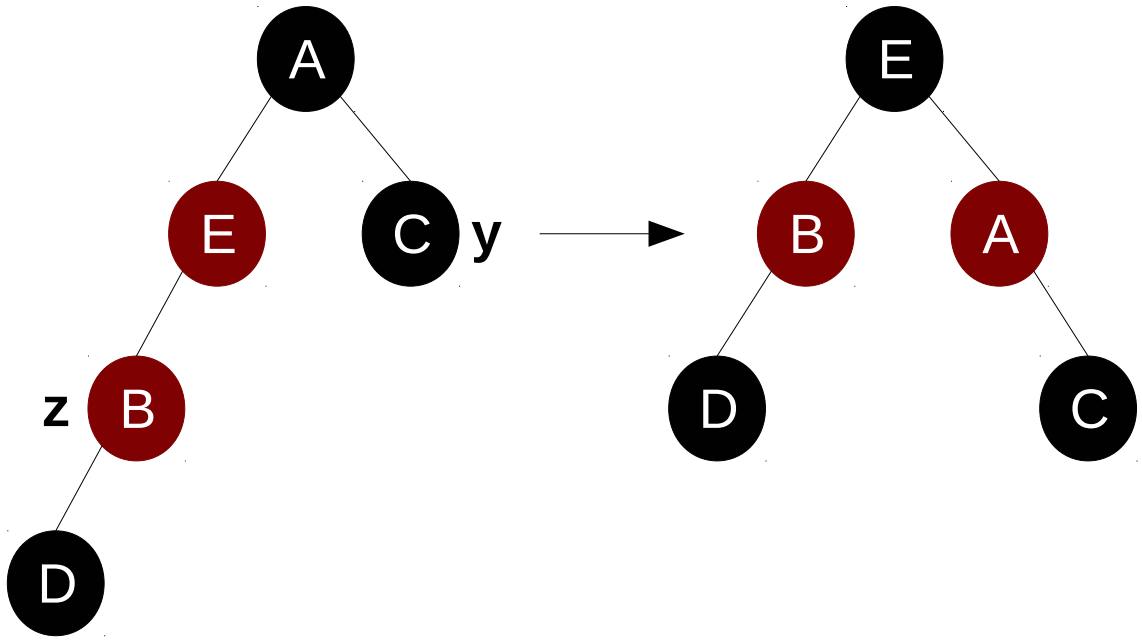
- z passa a ser o pai de z.
- Executamos rotação à esquerda em z, caindo no caso 3.



Caso 3

Caso 3: o tio y de z é preto e z é filho à esquerda

- Colorimos o pai de z de preto e o avô de z de vermelho.
- Executamos rotação à direita no avô de z.

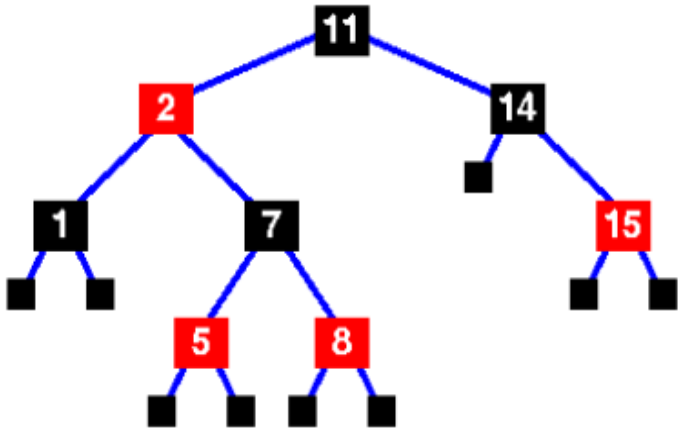


Balanceamento em Árvore Rubro Negra

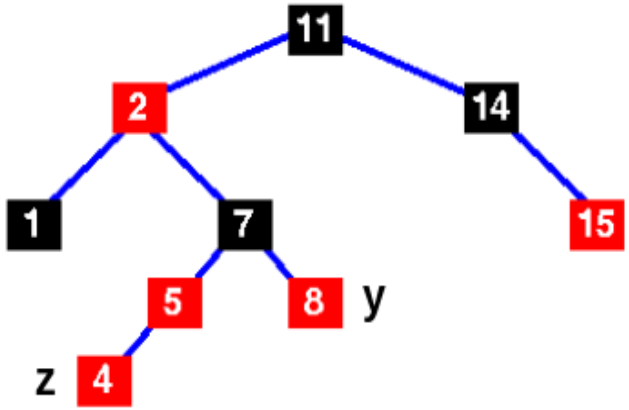
Balanceamento

```
RB_INSERT_FIXUP(T, z)
    while z.pai.cor == VERMELHO
        if z.pai == z.pai.pai.esquerda
            y = z.pai.pai.direita;
            if y.cor == VERMELHO
                z.pai.cor = PRETO;                  // caso 1
                y.cor = PRETO;                      // caso 1
                z.pai.pai.cor = VERMELHO;           // caso 1
                z = z.pai.pai;                      // caso 1
            else
                if z = z.pai.direita
                    z = z.pai;                      // caso 2
                    LEFT_ROTATE(T, z);              // caso 2
                z.pai.cor = PRETO;                  // caso 3
                z.pai.pai.cor = VERMELHO;           // caso 3
                RIGHT_ROTATE(T, z.pai.pai);         // caso 3
        else ( igual à cláusula então com dir. e esq. trocadas)
    T.raiz.cor = PRETO;
```

Exemplo

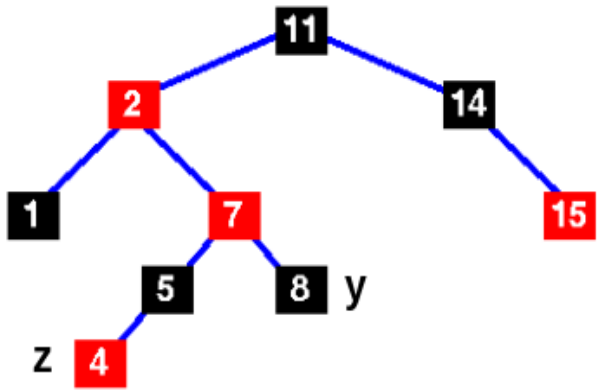


Árvore Original

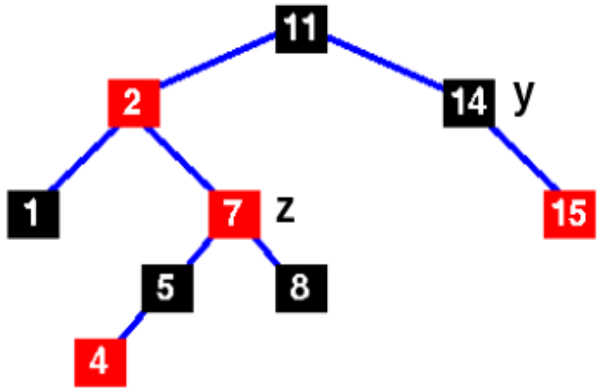


Inserção do nó 4.
O tio de z é vermelho (Caso 1).

Exemplo

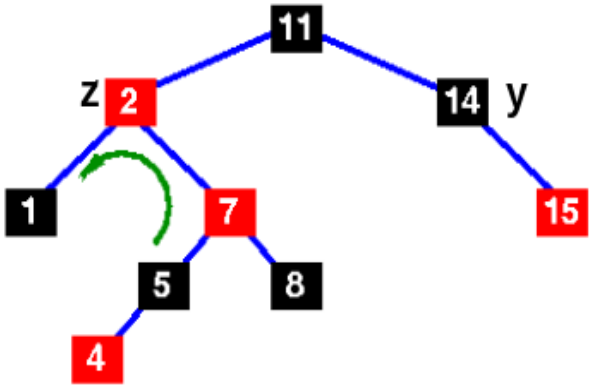


Trocamos a cor dos nós 5, 7 e 8.

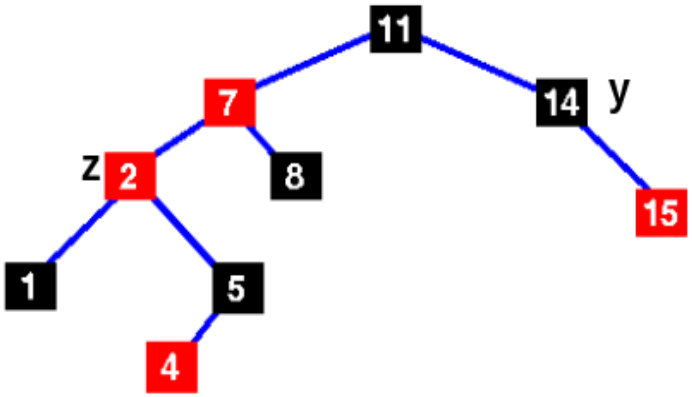


O avô de z passa a ser o novo z.
O tio de z é preto e z é filho da direita
(Caso 2).

Exemplo

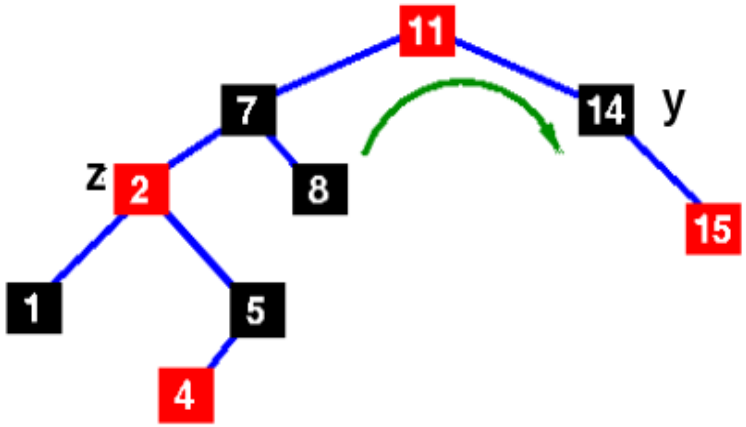


z passa a ser o pai de z.
Rotação a esquerda.

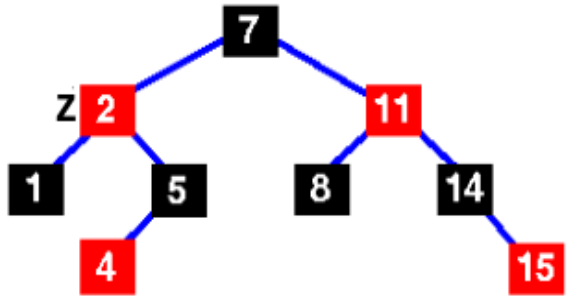


O tio de z é preto e z é filho da esquerda
(Caso 3).

Exemplo



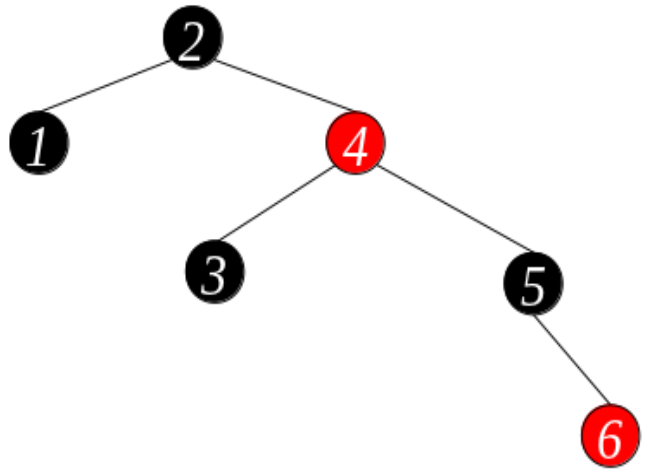
Troca a cor dos nós 7 e 11.
Rotação a direta no avô de z.



Árvore final.

Exercício

1. Inserir os elementos 41, 38, 31, 12, 19 e 8 em uma árvore rubro negra.
2. Dado a árvore rubro negra insira o nó 7.



Remoção em Árvore Rubro Negra

- Dados o ponteiro para a raiz, T, e um nó z a ser removido.
- O nó y é:
  - y é o próprio nó z, quando z não tem filhos ou tem um filho.
  - y é o nó antecessor de z, quando z tem dois filhos.
  - y é o nó a ser removido.
- O nó x é o **nó que ocupará o lugar do nó z**. Ou seja, é um filho de y.

Remoção em Árvore Rubro Negra

- Dados o ponteiro para a raiz, T, e um nó z a ser removido.
- O nó y é:
  - y é o próprio nó z, quando z não tem filhos ou tem um filho.
  - y é o nó antecessor de z, quando z tem dois filhos.
  - y é o nó a ser removido.
- O nó x é o **nó que ocupará o lugar do nó z**. Ou seja, é um filho de y.

Remoção em Árvore Rubro Negra

- Dados o ponteiro para a raiz, T, e um nó z a ser removido.
- O nó y é:
  - y é o próprio nó z, quando z não tem filhos ou tem um filho.
  - y é o nó antecessor de z, quando z tem dois filhos.
  - y é o nó a ser removido.
- O nó x é o **nó que ocupará o lugar do nó z**. Ou seja, é um filho de y.

Remoção em Árvore Rubro Negra

- Dados o ponteiro para a raiz, T, e um nó z a ser removido.
- O nó y é:
  - y é o próprio nó z, quando z não tem filhos ou tem um filho.
  - y é o nó antecessor de z, quando z tem dois filhos.
  - y é o nó a ser removido.
- O nó x é o **nó que ocupará o lugar do nó z**. Ou seja, é um filho de y.

Remoção em Árvore Rubro Negra

- Dados o ponteiro para a raiz, T, e um nó z a ser removido.
- O nó y é:
  - y é o próprio nó z, quando z não tem filhos ou tem um filho.
  - y é o nó antecessor de z, quando z tem dois filhos.
  - y é o nó a ser removido.
- O nó x é o **nó que ocupará o lugar do nó z**. Ou seja, é um filho de y.

Remoção em Árvore Rubro Negra

- Dados o ponteiro para a raiz, T, e um nó z a ser removido.
- O nó y é:
  - y é o próprio nó z, quando z não tem filhos ou tem um filho.
  - y é o nó antecessor de z, quando z tem dois filhos.
  - y é o nó a ser removido.
- O nó x é o **nó que ocupará o lugar do nó z**. Ou seja, é um filho de y.

Remoção em Árvore Rubro Negra

RB_DELETE

```
RB_DELETE(T, z) {
    y = z;
    y_cor = y.cor;

    // z nao tem filhos, z tem um filho a direita
    if z.esquerda == NIL
        x = z.direita
        TRANSPLANT(T, z, z.direita);

    // z tem um filho a esquerda
    else if z.direita == NIL
        x = z.esquerda
        TRANSPLANT(T, z, z.esquerda);
```

Remoção em Árvore Rubro Negra

RB_DELETE

```
    // z tem dois filhos
    else
        y = TREE_MAXIMUM(z.esquerda); // y e o antecessor de z
        y_cor = y.cor
        x = y.esquerda
        z.chave = y.chave
        TRANSPLANT(T, y, y.esquerda);
        transpor(arv, y, y->esq);

    if y_cor == PRETO
        RB_DELETE_FIXUP(T, x)

    desaloca y
```

Propriedades Violadas

- A remoção de nós vermelhos não alteram as propriedades da árvore.
- Mas a remoção de nós pretos alteram.Após a remoção, mas antes da execução a função RB_DELETE_FIXUP, apenas as seguintes propriedades podem ser violadas:

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Propriedades Violadas

- A remoção de nós vermelhos não alteram as propriedades da árvore.
- Mas a remoção de nós pretos alteram. Após a remoção, mas antes da execução a função RB_DELETE_FIXUP, apenas as seguintes propriedades podem ser violadas:

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Propriedades Violadas

- A remoção de nós vermelhos não alteram as propriedades da árvore.
- Mas a remoção de nós pretos alteram.Após a remoção, mas antes da execução a função RB_DELETE_FIXUP, apenas as seguintes propriedades podem ser violadas:

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Propriedades Violadas

- A remoção de nós vermelhos não alteram as propriedades da árvore.
- Mas a remoção de nós pretos alteram.Após a remoção, mas antes da execução a função RB_DELETE_FIXUP, apenas as seguintes propriedades podem ser violadas:

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Propriedades Violadas

- A remoção de nós vermelhos não alteram as propriedades da árvore.
- Mas a remoção de nós pretos alteram.Após a remoção, mas antes da execução a função RB_DELETE_FIXUP, apenas as seguintes propriedades podem ser violadas:

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Propriedades Violadas

- A remoção de nós vermelhos não alteram as propriedades da árvore.
- Mas a remoção de nós pretos alteram.Após a remoção, mas antes da execução a função RB_DELETE_FIXUP, apenas as seguintes propriedades podem ser violadas:

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Propriedades Violadas

- A remoção de nós vermelhos não alteram as propriedades da árvore.
- Mas a remoção de nós pretos alteram.Após a remoção, mas antes da execução a função RB_DELETE_FIXUP, apenas as seguintes propriedades podem ser violadas:

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Propriedades Violadas

- A remoção de nós vermelhos não alteram as propriedades da árvore.
- Mas a remoção de nós pretos alteram.Após a remoção, mas antes da execução a função RB_DELETE_FIXUP, apenas as seguintes propriedades podem ser violadas:

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 seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até as folhas descendentes contêm o mesmo número de nós pretos (altura negra).

Remoção em Árvore Rubro Negra

- Após uma remoção a árvore pode estar com suas propriedades violadas.
- Quando o nó removido é preto as propriedades da árvore devem ser restauradas.
- A função RB_DELETE_FIXUP é responsavel por restaurar as propriedades da árvore por meio da troca de cores e das rotações.
  - A função recebe como parâmetro o nó x.
  - Após a remoção do nó y, o nó x passou a ocupar o lugar de y.
  - A função trata quatro casos para restauração da árvores, levando em consideração se x é filho da direita ou da esquerda.

Obs: analisaremos os casos quando x é filho da esquerda.

Remoção em Árvore Rubro Negra

- Após uma remoção a árvore pode estar com suas propriedades violadas.
- Quando o nó removido é preto as propriedades da árvore devem ser restauradas.
- A função RB_DELETE_FIXUP é responsavel por restaurar as propriedades da árvore por meio da troca de cores e das rotações.
  - A função recebe como parâmetro o nó x.
  - Após a remoção do nó y, o nó x passou a ocupar o lugar de y.
  - A função trata quatro casos para restauração da árvores, levando em consideração se x é filho da direita ou da esquerda.

Obs: analisaremos os casos quando x é filho da esquerda.

Remoção em Árvore Rubro Negra

- Após uma remoção a árvore pode estar com suas propriedades violadas.
- Quando o nó removido é preto as propriedades da árvore devem ser restauradas.
- A função RB_DELETE_FIXUP é responsavel por restaurar as propriedades da árvore por meio da troca de cores e das rotações.
  - A função recebe como parâmetro o nó x.
  - Após a remoção do nó y, o nó x passou a ocupar o lugar de y.
  - A função trata quatro casos para restauração da árvores, levando em consideração se x é filho da direita ou da esquerda.

Obs: analisaremos os casos quando x é filho da esquerda.

Remoção em Árvore Rubro Negra

- Após uma remoção a árvore pode estar com suas propriedades violadas.
- Quando o nó removido é preto as propriedades da árvore devem ser restauradas.
- A função RB_DELETE_FIXUP é responsavel por restaurar as propriedades da árvore por meio da troca de cores e das rotações.
  - A função recebe como parâmetro o nó x.
  - Após a remoção do nó y, o nó x passou a ocupar o lugar de y.
  - A função trata quatro casos para restauração da árvores, levando em consideração se x é filho da direita ou da esquerda.

Obs: analisaremos os casos quando x é filho da esquerda.

Remoção em Árvore Rubro Negra

- Após uma remoção a árvore pode estar com suas propriedades violadas.
- Quando o nó removido é preto as propriedades da árvore devem ser restauradas.
- A função RB_DELETE_FIXUP é responsavel por restaurar as propriedades da árvore por meio da troca de cores e das rotações.
  - A função recebe como parâmetro o nó x.
  - Após a remoção do nó y, o nó x passou a ocupar o lugar de y.
  - A função trata quatro casos para restauração da árvores, levando em consideração se x é filho da direita ou da esquerda.

Obs: analisaremos os casos quando x é filho da esquerda.

Remoção em Árvore Rubro Negra

- Após uma remoção a árvore pode estar com suas propriedades violadas.
- Quando o nó removido é preto as propriedades da árvore devem ser restauradas.
- A função RB_DELETE_FIXUP é responsavel por restaurar as propriedades da árvore por meio da troca de cores e das rotações.
  - A função recebe como parâmetro o nó x.
  - Após a remoção do nó y, o nó x passou a ocupar o lugar de y.
  - A função trata quatro casos para restauração da árvores, levando em consideração se x é filho da direita ou da esquerda.

Obs: analisaremos os casos quando x é filho da esquerda.

Remoção em Árvore Rubro Negra

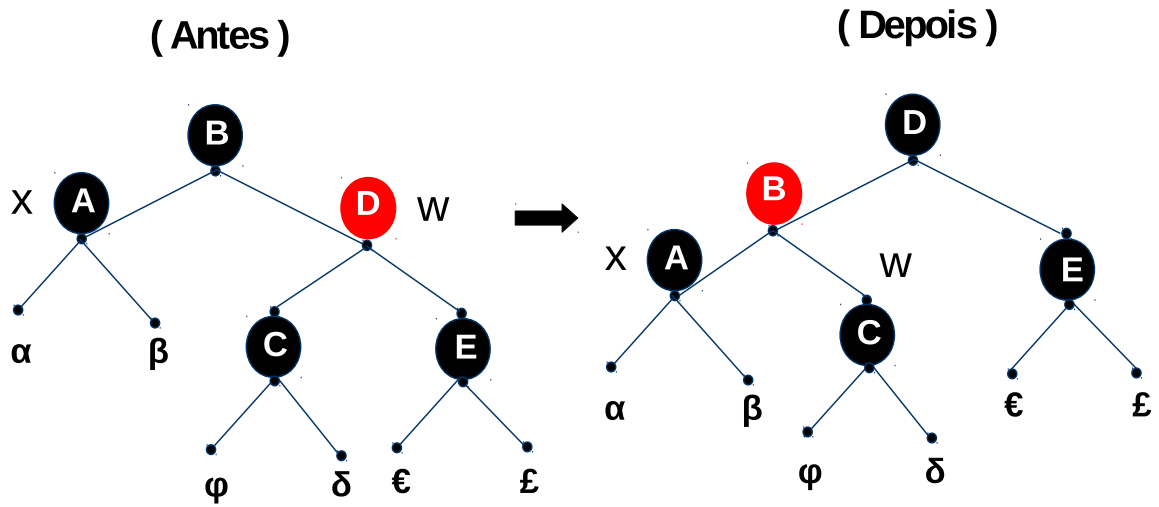
- Após uma remoção a árvore pode estar com suas propriedades violadas.
- Quando o nó removido é preto as propriedades da árvore devem ser restauradas.
- A função RB_DELETE_FIXUP é responsavel por restaurar as propriedades da árvore por meio da troca de cores e das rotações.
  - A função recebe como parâmetro o nó x.
  - Após a remoção do nó y, o nó x passou a ocupar o lugar de y.
  - A função trata quatro casos para restauração da árvores, levando em consideração se x é filho da direita ou da esquerda.

Obs: analisaremos os casos quando x é filho da esquerda.

Caso 1

Caso 1: o irmão w de x é vermelho

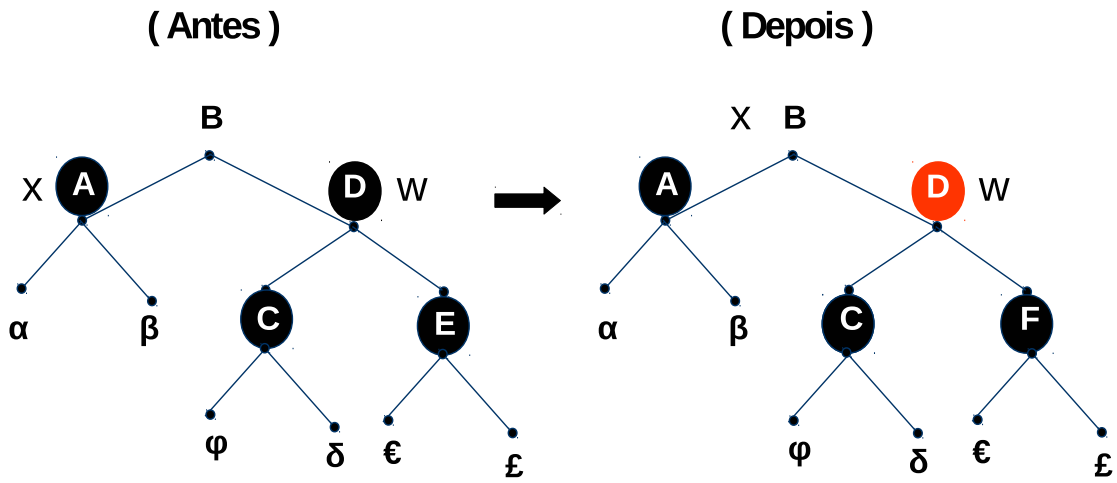
- A cor de w passa a ser preta
- A cor do pai de x passa a ser vermelha
- Rotação à esquerda no pai de x
- w passa a ser o irmão de x à direita



Caso 2

Caso 2: o irmão w de x é preto e os filhos de w são pretos

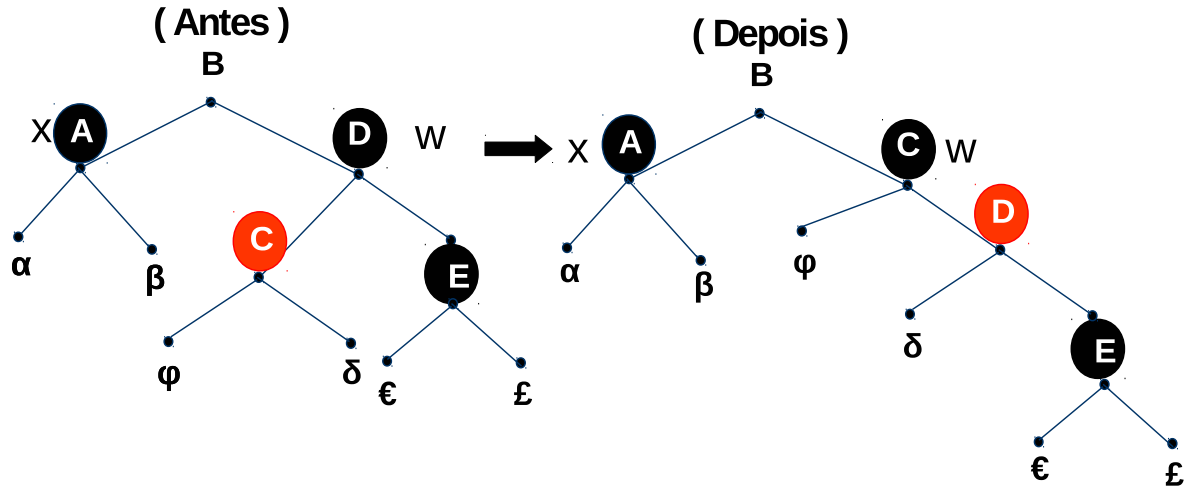
- A cor de w passa a ser vermelha
- x passa a ser o pai de x



Caso 3

Caso 3: o irmão w de x é preto, o filho à esquerda de w é vermelho e o filho à direita de w é preto

- A cor de w a esquerda passa a ser preta
- A cor de w passa a ser vermelha
- Rotação à direita em w
- w passa a ser o pai de x à direita

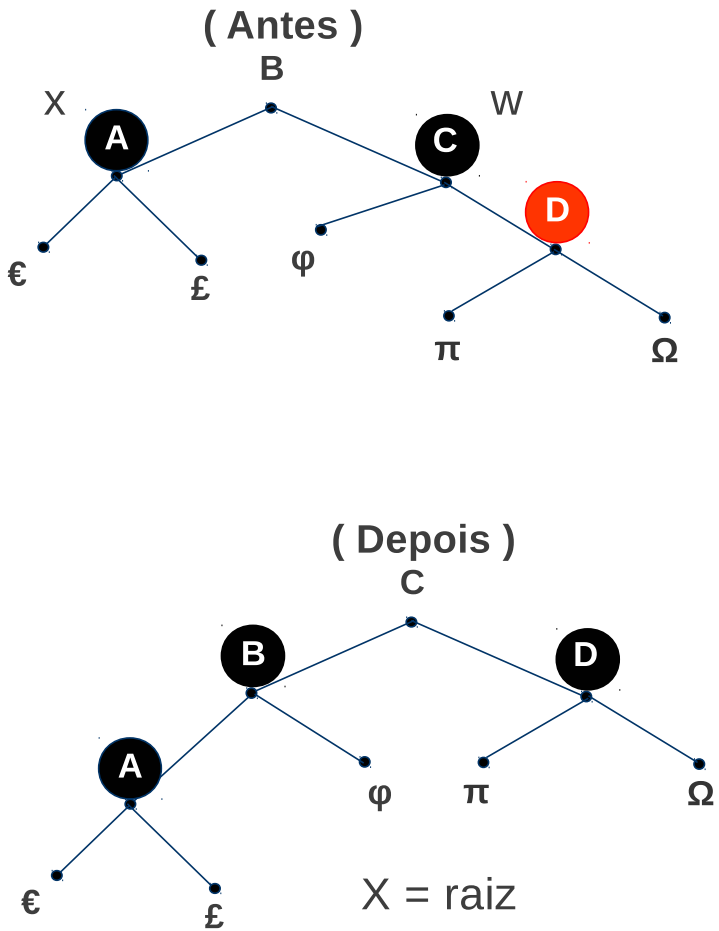


Caso 4

Caso 4: o irmão w de x é preto e o filho à direita de w é vermelho

- A cor de w passa a ser a cor do pai de x
- A cor do pai de x passa a ser negra
- A cor de w à direita passa a ser negra
- Rotação esquerda no pai de x
- x passa a ser a raiz

Caso 4



Remoção em Árvore Rubro Negra

RB_DELETE_FIXUP

```
RB_DELETE_FIXUP(T, x) {
    while x != T && x.cor == PRETO
        // trata remocao a esquerda
        if x == x.pai.esquerda
            w = x.pai.direita;

            // CASO 1
            if w.cor == VERMELHO
                w.cor = PRETO;
                x.pai.cor = VERMELHO;
                LEFT_ROTATE(T, x.pai);
                w = x.pai.dir;
```

Remoção em Árvore Rubro Negra

RB_DELETE_FIXUP

```
            // CASO 2
            if w.esquerda.cor == PRETO && w.dir.cor == PRETO
                w.cor = VERMELHO;
                x = x.pai;
            else
                // CASO 3
                if w.dir.cor == PRETO
                    w.esq.cor = PRETO;
                    w.cor = VERMELHO;
                    RIGHT_ROTATE(T, w);
                    w = x.pai.dir;
```

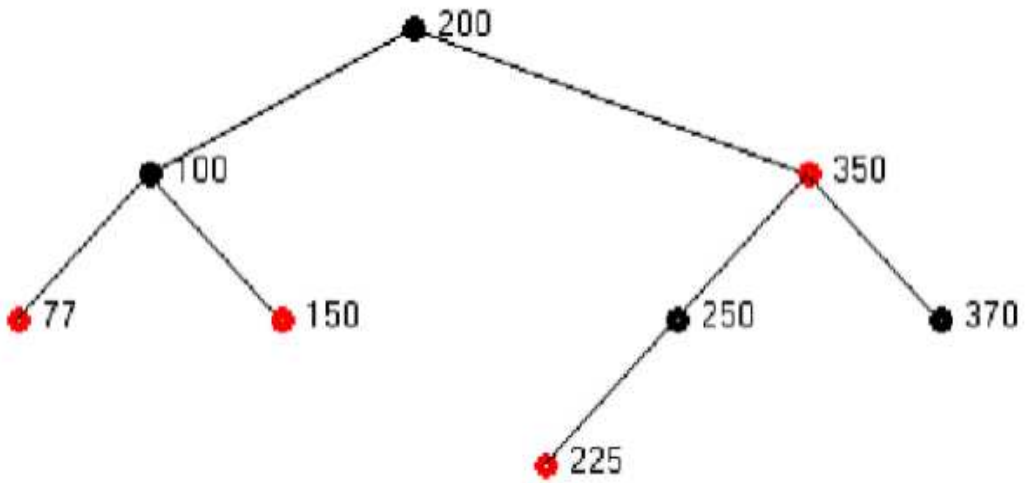
Remoção em Árvore Rubro Negra

RB_DELETE_FIXUP

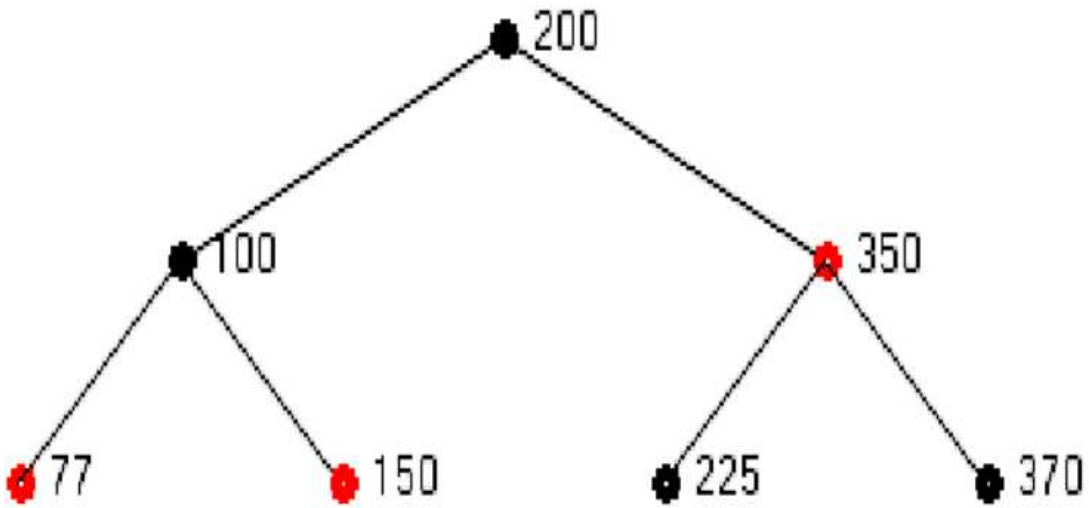
```
                // CASO 4
                w.cor = x.pai.cor;
                x.pai.cor = PRETO;
                w.dir.cor = PRETO;
                LEFT_ROTATE(T, x.pai);
                x = T;
        else
            // trata remocao a direita
            // trocando direita e esquerda

    x.cor = PRETO;
}
```

Exemplo

Removendo o nó 250:

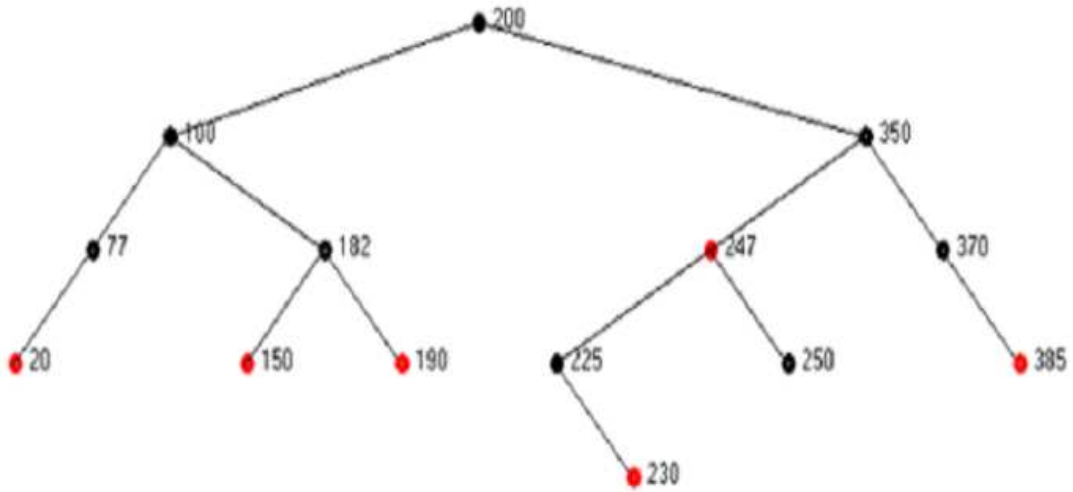


Exemplo

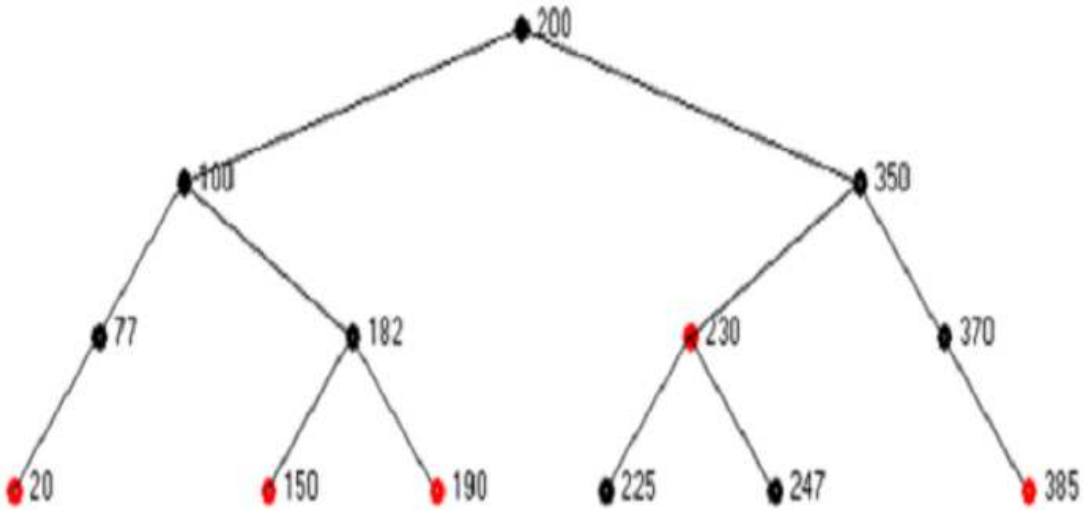


Exemplo

Removendo o nó 250:



Exemplo



Exercícios

1. Após a inserção dos elementos 41, 38, 31, 12, 19 e 8 em uma árvore rubro negra faça a remoção dos elementos 19 e 8.