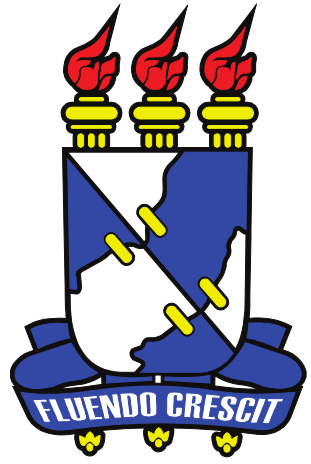


Estrutura de Dados I

Árvores rubro-negras

Bruno Prado

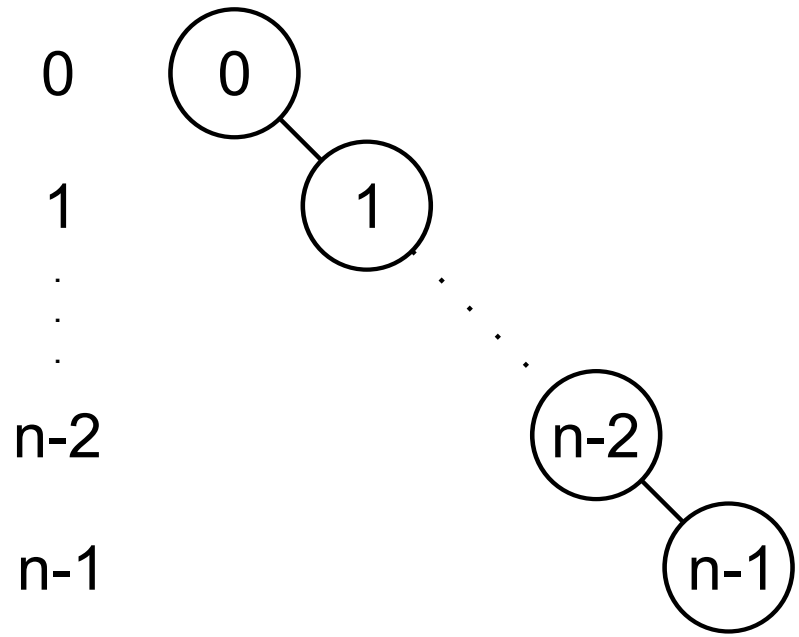
Departamento de Computação / UFS

Introdução

- Conceitos chave de árvores binárias
  - Cada nó possui até 2 filhos
  - Para uma árvore com altura h existem $\lceil 2^{h+1} - 1 \rceil$ nós
  - Com n nós possui altura entre $\log_2 n - 1 \leq h \leq n - 1$

Introdução

- Conceitos chave de árvores binárias
  - As operações na árvore tem custo $O(h)$
  - A degeneração da árvore binária leva a uma altura $\lceil h = n - 1 \rceil$ e ocorre devido ao desbalanceamento



Introdução

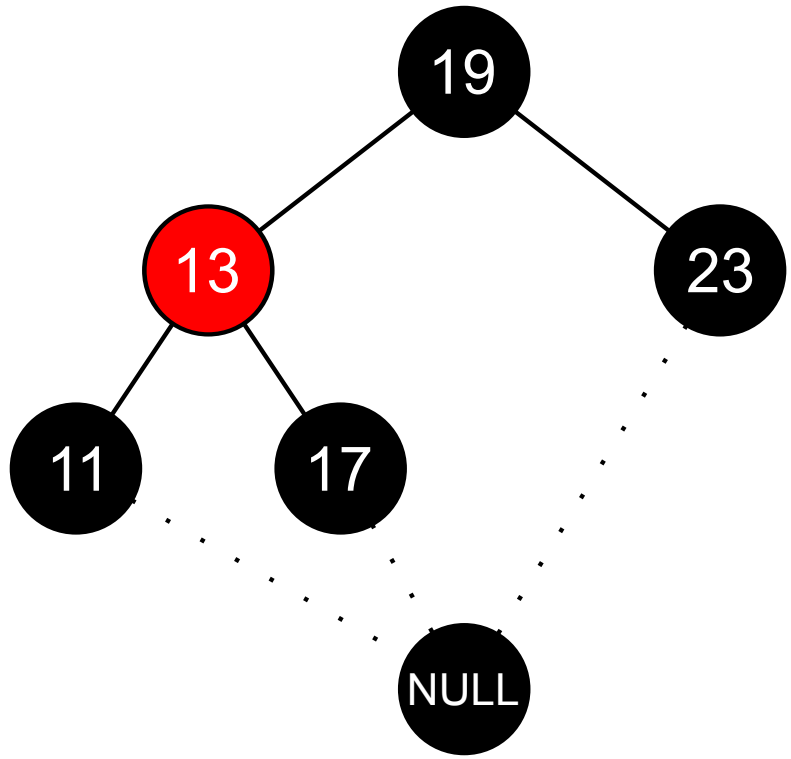
- O que é balanceamento de uma árvore binária?
  - É a aplicação de restrições estruturais na realização das operações para garantir que a árvore resultante possua uma altura logarítmica
  - Evita o processo de degeneração e garante a eficiência computacional da estrutura

Introdução

- O que é balanceamento de uma árvore binária?
  - É a aplicação de restrições estruturais na realização das operações para garantir que a árvore resultante possua uma altura logarítmica
  - Evita o processo de degeneração e garante a eficiência computacional da estrutura
- Árvore Rubro-Negra: Rudolf Bayer
  - É uma árvore binária balanceada proposta em 1972
  - Seu funcionamento é baseado em coloração dos nós em cores rubro (red) e negra (black) para limitar a altura máxima e evitar o processo de degeneração

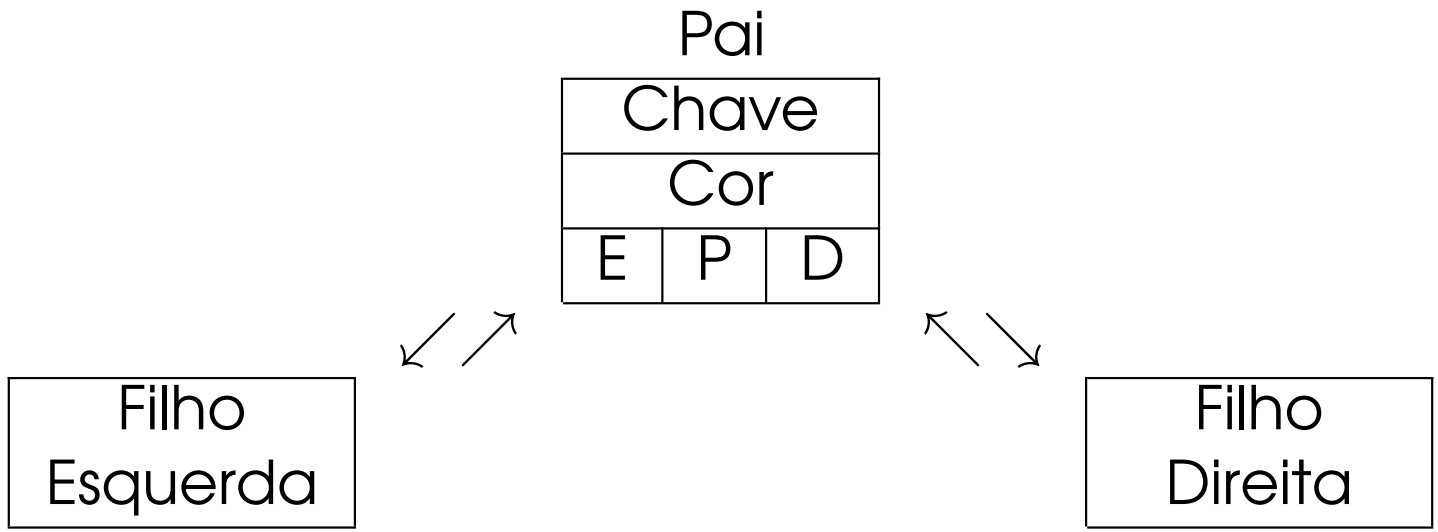
Árvores Rubro-Negras

- Propriedades da árvore rubro-negra
  - Todo nó possui coloração rubro ou negra
  - O nó raiz e todos os nós folha são negros
  - Se um nó é rubro, seus filhos são negros
  - Para cada nó da árvore, os percursos realizados até as folhas possuem o mesmo número de nós negros



Árvores Rubro-Negras

- Implementação em C
  - Estruturas e ponteiros



Árvores Rubro-Negras

- Implementação em C
  - Estruturas e ponteiros

```
// Representação da árvore rubro-negra
typedef struct no {
        // Chave do nó
        int chave;
        // Cor do nó
        char cor;
        // Pai
        struct no* P;
        // Filho da esquerda
        struct no* E;
        // Filho da direita
        struct no* D;
} no;
```

Árvores Rubro-Negras

- Como é feito o balanceamento da árvore?
  - O balanceamento é feito através do ajuste da coloração dos nós, ao invés da calcular a diferença de altura das subárvores como ocorre em árvores AVL
  - Para manter as propriedades da árvore são realizadas operações de rotação para esquerda, para direita e inversão de cores

Árvores Rubro-Negras

- Operações de rotação para esquerda e para direita

```
void rotacao_esquerda(no* x) {
        no* y = x->D;
        x->D = y->E;
        if(y->E != NULL)
                y->E->P = x;
        y->P = x->P;
        if(x->P == NULL)
                raiz = y;
        else if(x == x->P->E)
                x->P->E = y;
        else
                x->p->D = y;
        y->E = x;
        x->P = y;
}
```

```
void rotacao_direita(no* y) {
        no* x = y->E;
        y->E = x->D;
        if(x->D != NULL)
                x->D->P = y;
        x->P = y->P;
        if(y->P == NULL)
                raiz = x;
        else if(y == y->P->D)
                y->P->D = x;
        else
                y->p->D = x;
        x->D = y;
        y->P = x;
}
```

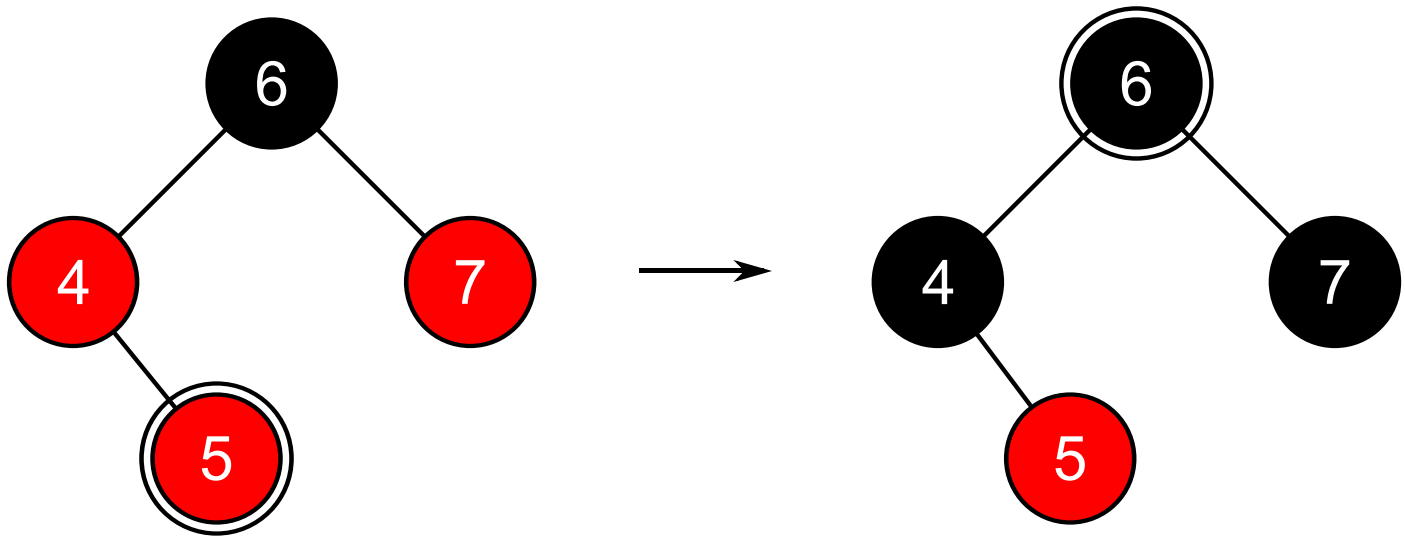
Árvores Rubro-Negras

- Manutenção das propriedades
  - Caso 1: nós pai e tio são rubros
    - É aplicado quando ocorre a violação da propriedade que define que nós rubros só possuem filhos negros
    - A cor do nó pai e do nó tio são ajustadas para negra
    - O ponteiro do nó é atualizado para o avô que é raiz da árvore, com ajuste de cor para negra

```
while(rubro(x->P) {
        if(x->P == x->P->P->E) {
                no* tio = x->P->P->D;
                if(rubro(tio)) {
                        x->P->cor = NEGRA;
                        tio->cor = NEGRA;
                        x->P->P = RUBRO;
                        x = x->P->P;
                }
                ...
        }
        ...
}
raiz->cor = NEGRA;
```

Árvores Rubro-Negras

- Manutenção das propriedades
  - Caso 1: nós pai e tio são rubros
    - É aplicado quando ocorre a violação da propriedade que define que nós rubros só possuem filhos negros
    - A cor do nó pai e do nó tio são ajustadas para negra
    - O ponteiro do nó é atualizado para o avô que é raiz da árvore, com ajuste de cor para negra



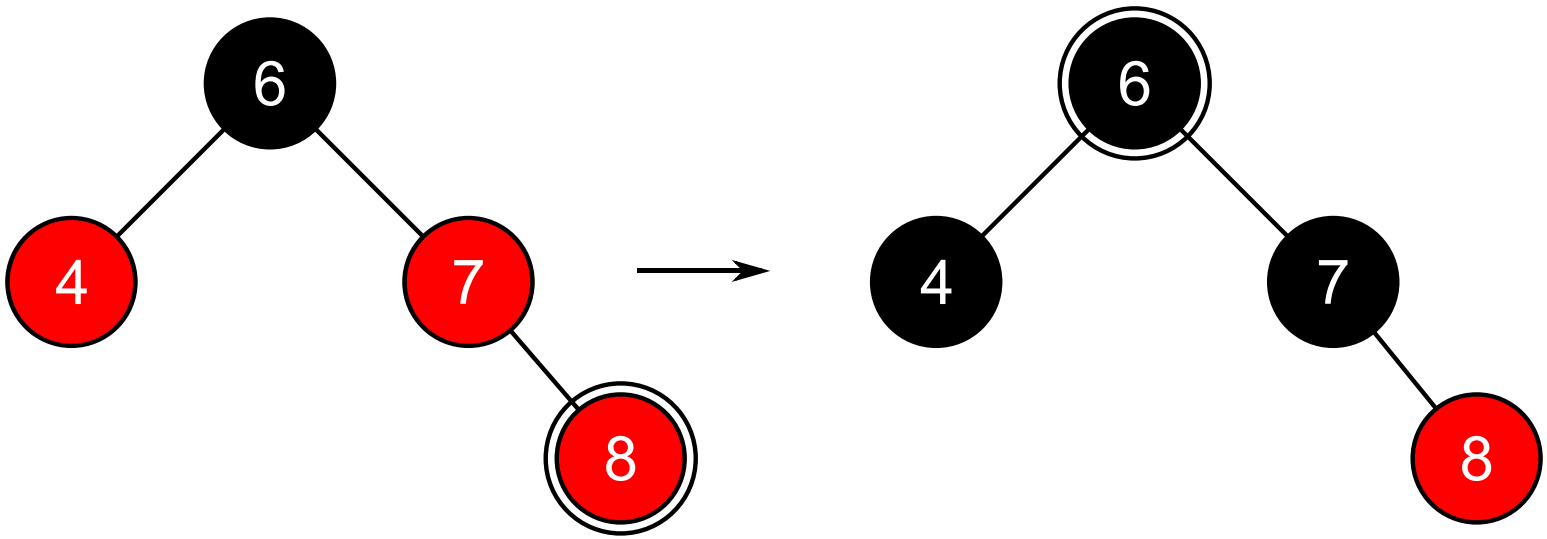
Árvores Rubro-Negras

- Manutenção das propriedades
  - Caso 1: nós pai e tio são rubros
    - É aplicado quando ocorre a violação da propriedade que define que nós rubros só possuem filhos negros
    - A cor do nó pai e do nó tio são ajustadas para negra
    - O ponteiro do nó é atualizado para o avô que é raiz da árvore, com ajuste de cor para negra

```
while(rubro(x->P)) {
    if(x->P == x->P->P->E) {
        ...
        else {
            no* tio = x->P->P->E;
            if(rubro(tio)) {
                x->P->cor = NEGRA;
                tio->cor = NEGRA;
                x->P->P = RUBRO;
                x = x->P->P;
            }
        ...
    }
}
raiz->cor = NEGRA;
```

Árvores Rubro-Negras

- Manutenção das propriedades
  - Caso 1: nós pai e tio são rubros
    - É aplicado quando ocorre a violação da propriedade que define que nós rubros só possuem filhos negros
    - A cor do nó pai e do nó tio são ajustadas para negra
    - O ponteiro do nó é atualizado para o avô que é raiz da árvore, com ajuste de cor para negra



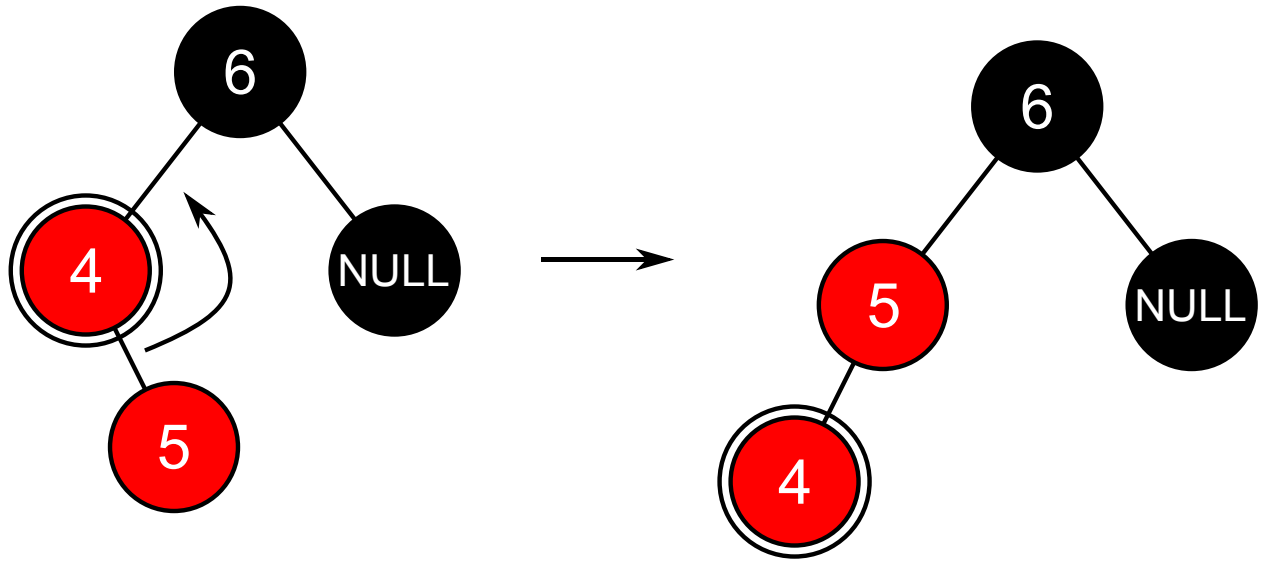
Árvores Rubro-Negras

- Manutenção das propriedades
  - Caso 2: nó filho direito e nó tio é negro
    - É aplicada uma rotação para esquerda com eixo no pai do nó inserido

```
while(rubro(x->P)) {
    if(x->P == x->P->P->E) {
        no* tio = x->P->P->D;
        if(rubro(tio)) {
        ...
        else {
            if(x == x->P->D) {
                x = x->P;
                rotacao_esquerda(x);
            }
            x->P->cor = NEGRA;
            x->P->P->cor = RUBRO;
            rotacao_direita(x->P->P);
        }
    ...
}
raiz->cor = NEGRA;
```

Árvores Rubro-Negras

- Manutenção das propriedades
  - Caso 2: nó filho direito e nó tio é negro
    - É aplicada uma rotação para esquerda com eixo no pai do nó inserido



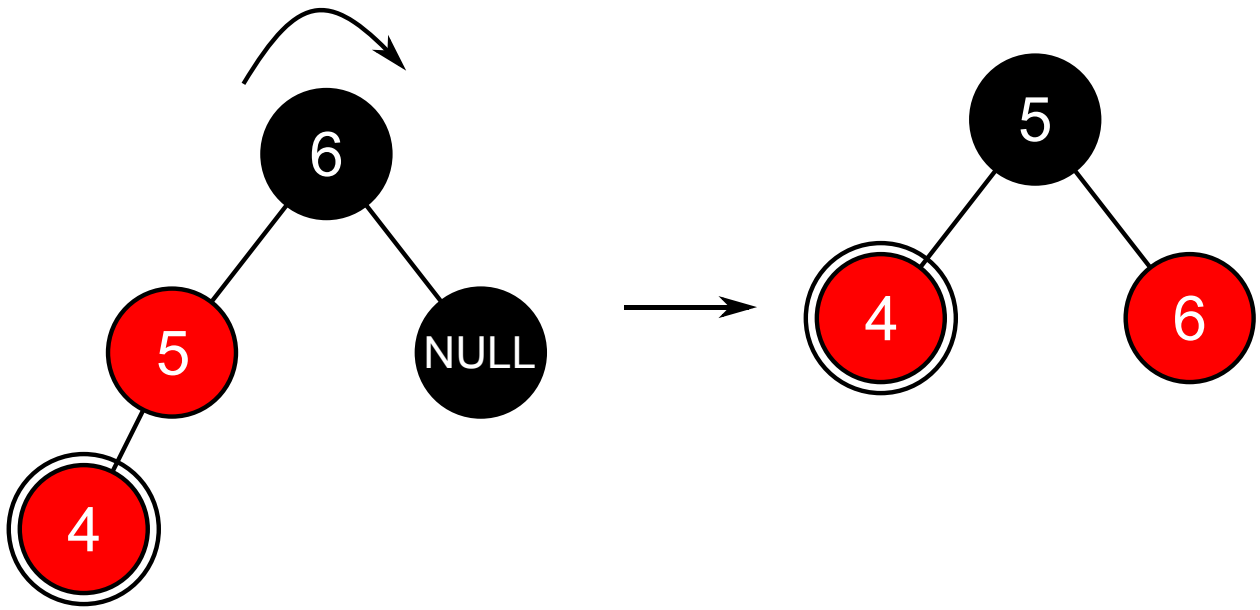
Árvores Rubro-Negras

- Manutenção das propriedades
  - Caso 3: nó filho esquerdo e nó tio é negro
    - O nó pai recebe cor negra e o avô cor rubro com rotação para direita, com rotação para direta no avô

```
while(rubro(x->P)) {
    if(x->P == x->P->P->E) {
        no* tio = x->P->P->D;
        if(rubro(tio)) {
        ...
        else {
            if(x == x->P->D) {
                x = x->P;
                rotacao_esquerda(x);
            }
            x->P->cor = NEGRA;
            x->P->P->cor = RUBRO;
            rotacao_direita(x->P->P);
        }
    ...
}
raiz->cor = NEGRA;
```

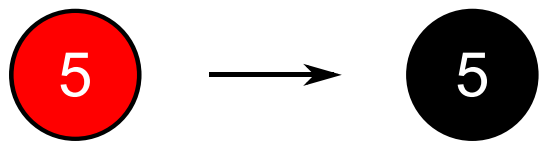
Árvores Rubro-Negras

- Manutenção das propriedades
  - Caso 3: nó filho esquerdo e nó tio é negro
    - O nó pai recebe cor negra e o avô cor rubro com rotação para direita, com rotação para direta no avô



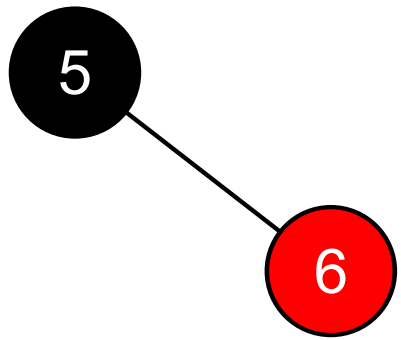
Árvores Rubro-Negras

- Operação de inserção
  - Parâmetro de chave: 5
  - Os nós sempre são inseridos na árvore com a coloração rubra, entretanto para satisfazer as propriedades da árvore, o nó raiz é ajustado para coloração negra



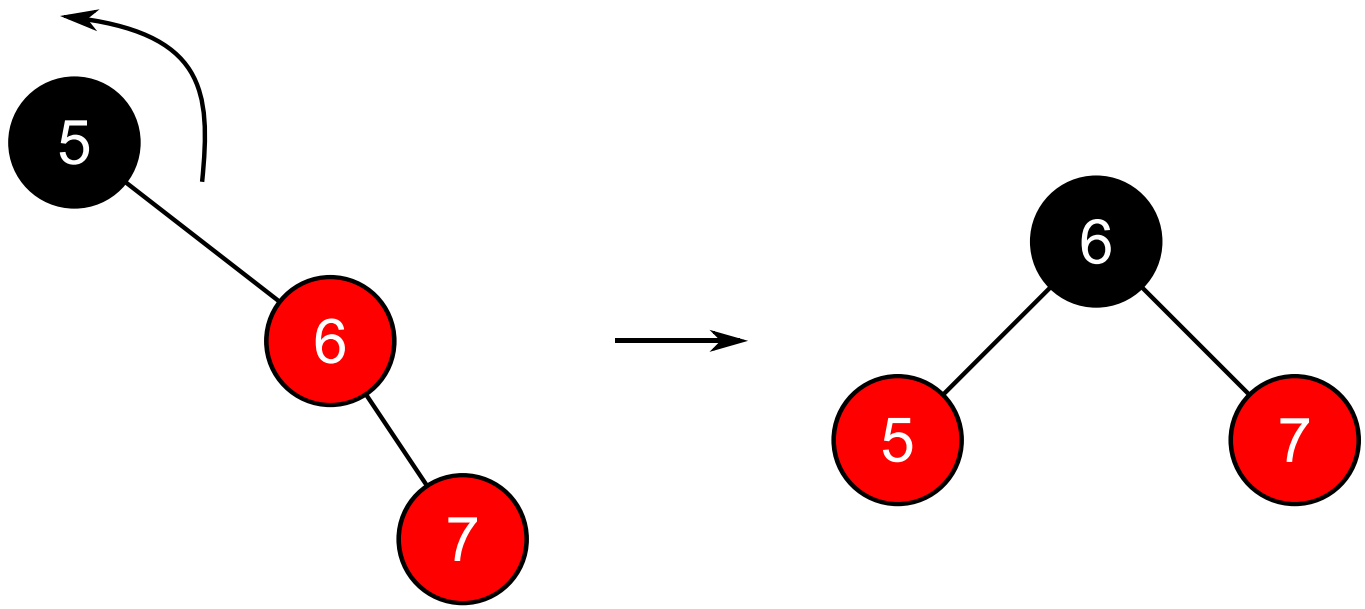
Árvores Rubro-Negras

- Operação de inserção
  - Parâmetro de chave: 6
  - Todas as propriedades da árvore são satisfeitas



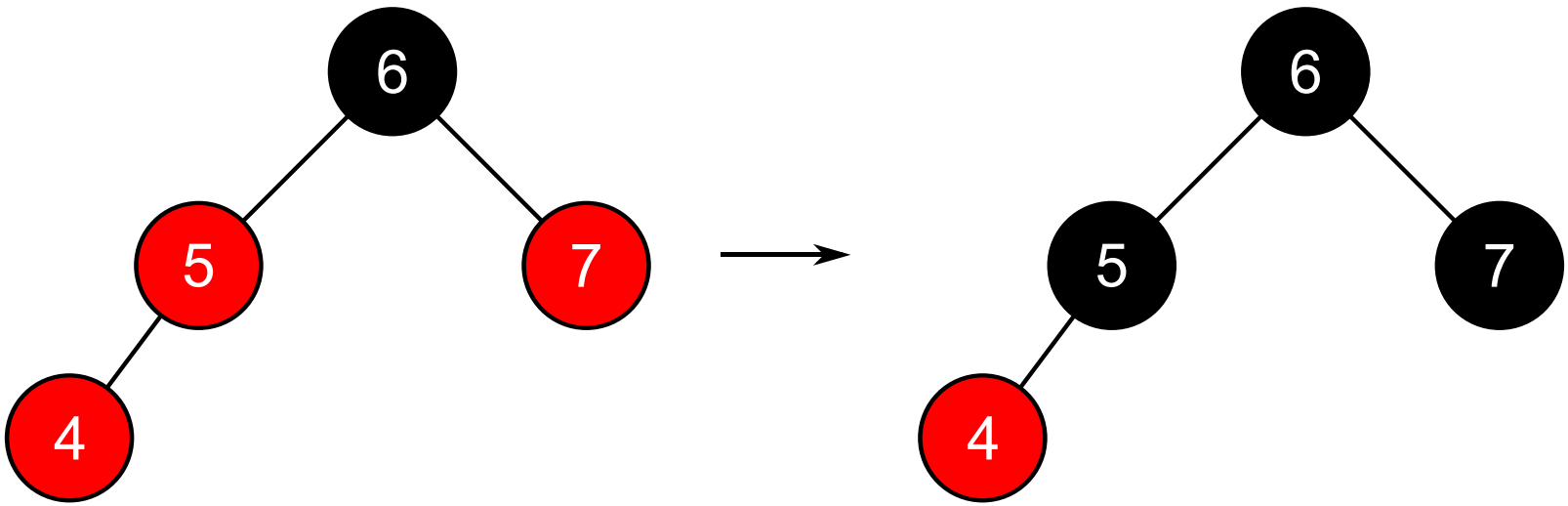
Árvores Rubro-Negras

- Operação de inserção
  - Parâmetro de chave: 7
  - O caso 2 é aplicado para manutenção das propriedades, uma vez que o nó filho é inserido a direita, com nó pai rubro e nó tio negro
  - É realizada uma rotação para esquerda



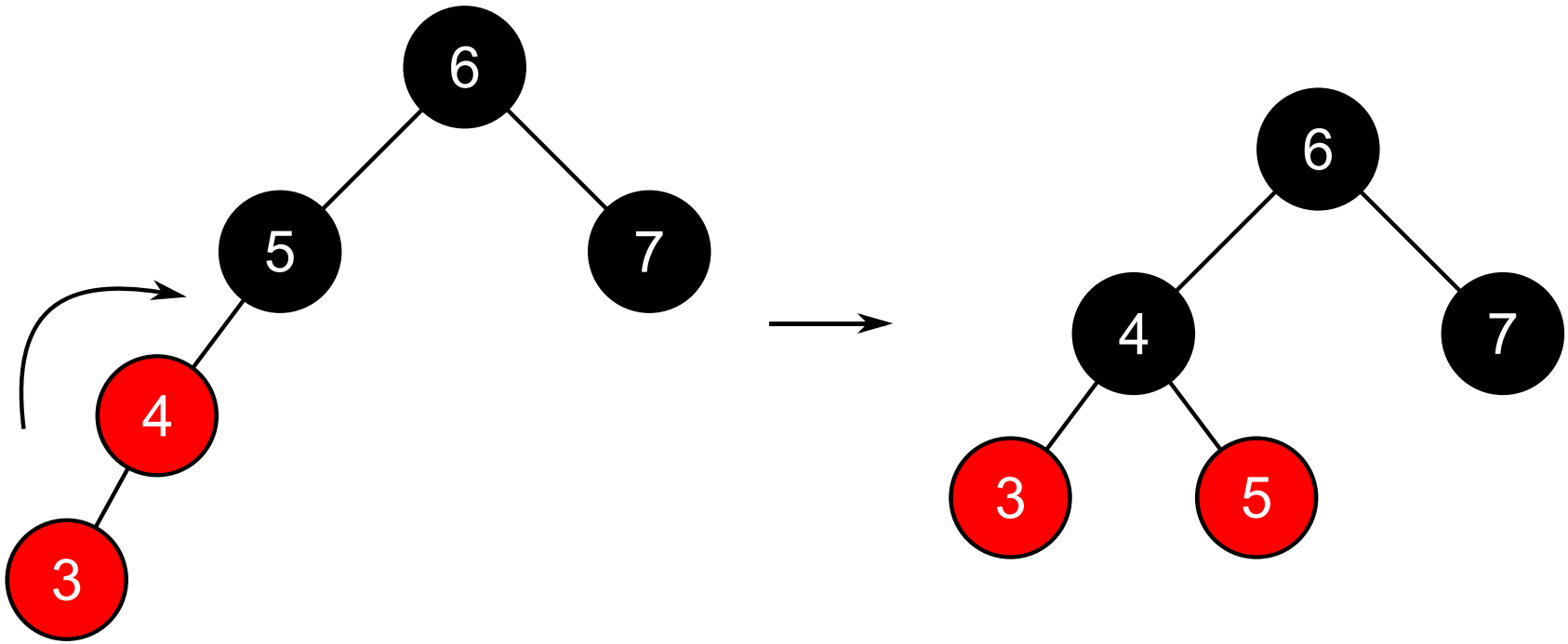
Árvores Rubro-Negras

- Operação de inserção
  - Parâmetro de chave: 4
  - É aplicado o caso 1 para atender a propriedade que um nó rubro possui somente filhos negros, sendo feito o ajuste de colocação do nó pai e do nó tio



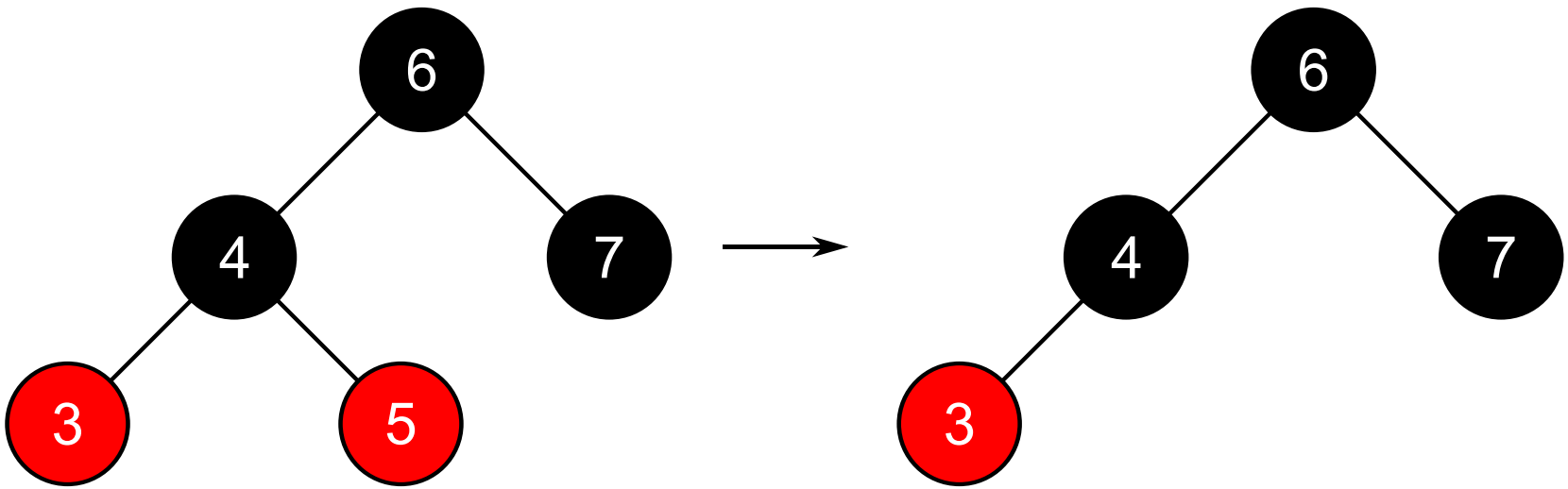
Árvores Rubro-Negras

- Operação de inserção
  - Parâmetro de chave: 3
  - O caso 3 é aplicado para manutenção das propriedades, uma vez que o nó filho é inserido a esquerda, com nó pai rubro e nó tio negro
  - É realizada uma rotação para direita



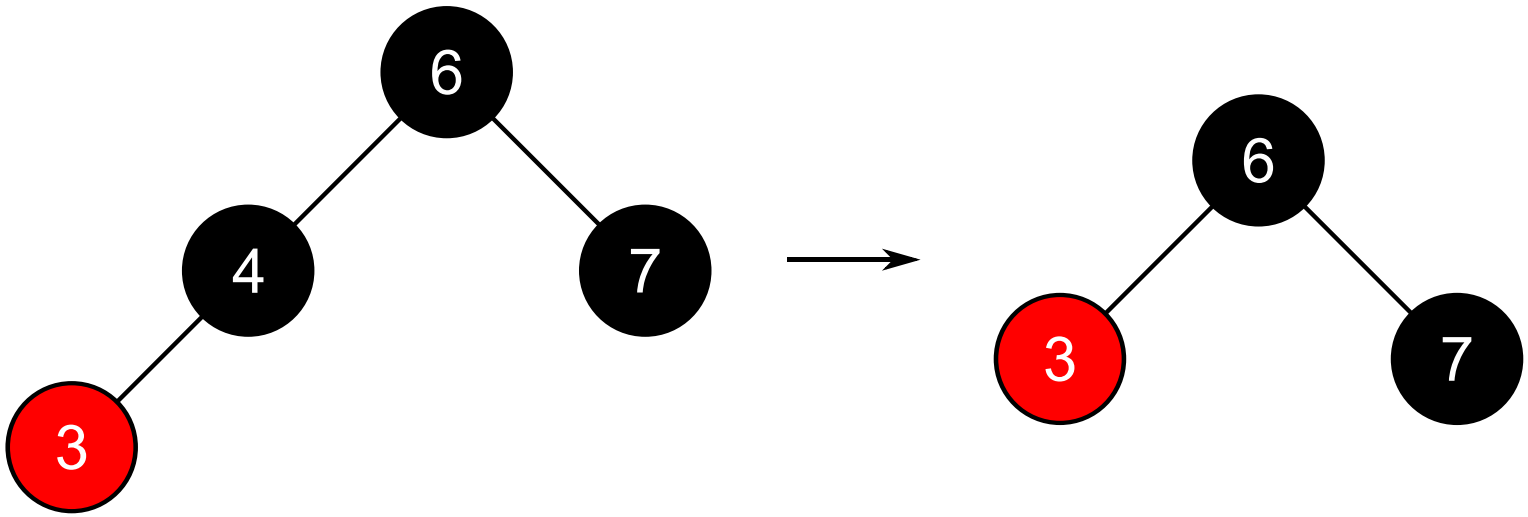
Árvores Rubro-Negras

- Operação de remoção
  - Parâmetro de chave: 5
  - Todas as propriedades da árvore são satisfeitas



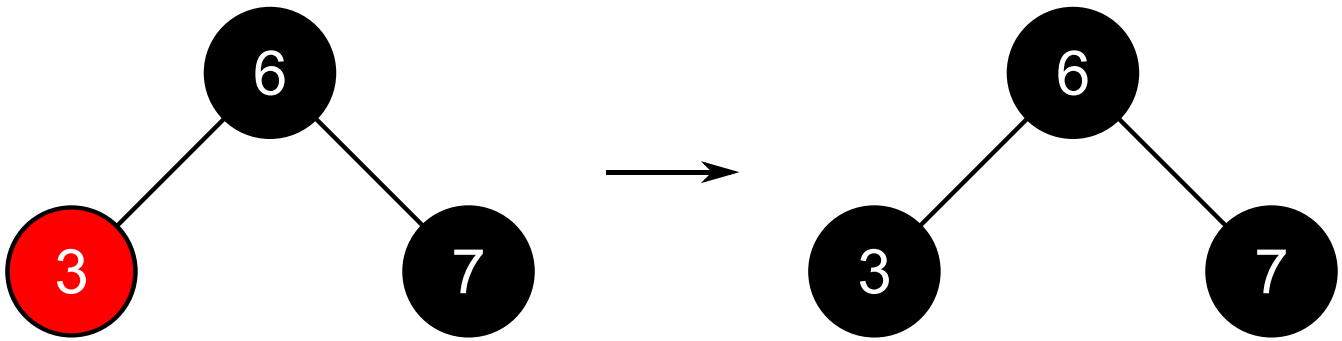
Árvores Rubro-Negras

- Operação de remoção
  - Parâmetro de chave: 4
  - É feita a substituição do nó pela subárvore



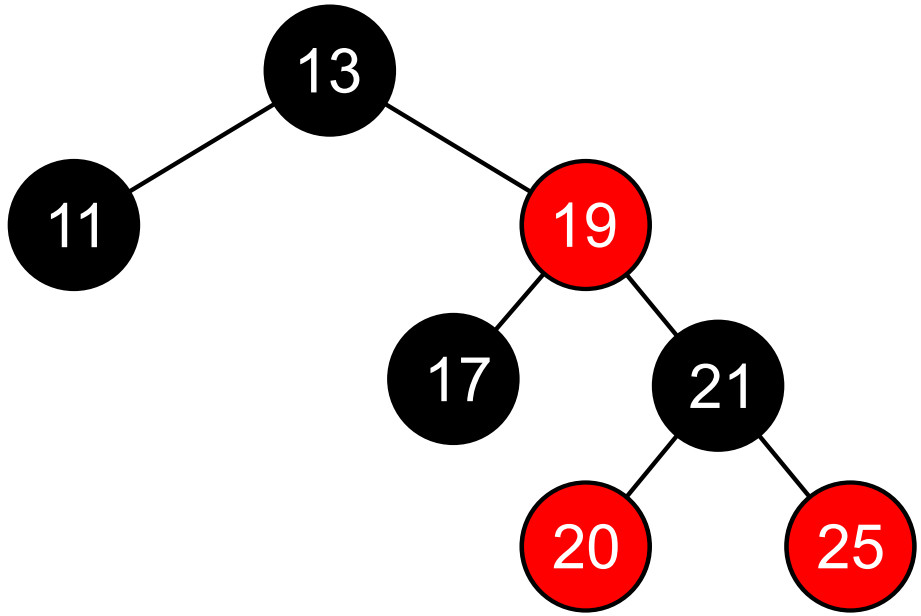
Árvores Rubro-Negras

- Operação de remoção
  - Parâmetro de chave: 4
  - Para atender a propriedade que os percursos realizados possuem o mesmo número de nós negros, é feito o ajuste da coloração dos nós



Árvores Rubro-Negras

- Por que a árvore rubro-negra é balanceada?
  - O balanceamento é obtido pela propriedade do percurso partindo de qualquer nó da árvore possui o mesmo número ou altura de nós negros



  - Pela propriedade que define que um nó rubro só pode ter filhos negros, em uma árvore com altura h, é possível afirmar que a altura h_N dos nós negros é de pelo menos $h/2$

Árvores Rubro-Negras

- Análise de complexidade
  - Considerando uma árvore de altura h, existem no máximo $2^{h+1} - 1$ nós rubros e negros
  - Pelas propriedades da árvore rubro-negra, a altura h_N dos nós negros em cada percurso possui pelo menos $h/2$ de altura

$$
\begin{aligned}
n &\geq 2^{h_N+1} - 1 \\
n+1 &\geq 2^{\frac{h}{2}+1} \\
\log_2(n+1) &\geq \frac{h}{2} + 1
\end{aligned}
$$

$$\downarrow$$

$$h \leq 2\log_2(n+1) - 2$$

Árvores Rubro-Negras

- Análise de complexidade
  - No pior caso, a busca percorre a altura h da árvore que possui n nós, entretanto a aplicação das técnicas de balanceamento garante que $h \approx 2 \log_2 n$
  - Espaço $\Theta(n)$
  - Tempo $O(\log_2 n)$

Exemplo

- Construa uma árvore binária rubro-negra
  - Insira os elementos com chaves 13, 2, 34, 11, 7, 43 e 9
  - Realize a remoção dos elementos de chave 7 e 9
  - Faça uma análise comparativa com as árvores AVL