



Estruturas de dados II

Árvore Rubro-Negra

André Tavares da Silva
andre.silva@udesc.br

Árvores rubro-negra

- Árvore rubro-negra é uma árvore de busca binária auto balanceada
 - Assim como em árvore AVL, a rubro-negra possui altura $O(\log n)$
 - No entanto, o pior caso de tempo na prática é mais eficiente que a AVL
 - Operações de busca, inserção e remoção ocorrem em tempo $O(\log n)$
 - Para alcançar este desempenho, a árvore se mantém aproximadamente balanceada ao inserir ou remover nós
- A árvore rubro-negra foi inventada em 1972 por Rudolf Bayer
 - Inicialmente foi chamada de árvores binárias B simétricas
 - Popularizou com este nome após artigo de Guibas e Sedgwick em 1978

Árvores rubro-negra

- Comparação com árvore AVL
 - Em teoria possuem a mesma complexidade computacional
 - Inserção, remoção e busca é $O(\log N)$
 - Na prática a árvore AVL é
 - Mais rápida na operação de busca
 - Mais lenta nas operações de inserção e remoção de nós
 - A árvore AVL possui o balanceamento mais rígido do que a rubro-negra
 - Isto acelera a operação de busca
 - Porém, exige mais esforço computacional para inserção e remoção de nós

Aplicação de árvores rubro-negra

- As árvores rubro-negra são de uso mais geral do que as árvores AVL
 - Bibliotecas em Java
 - `java.util.TreeMap`
 - `java.util.TreeSet`
 - Bibliotecas C++ STL
 - `map`
 - `multimap`
 - `multiset`
 - Linux kernel
 - `scheduler`
 - `filesystem`
 - `linux/rbtree.h`

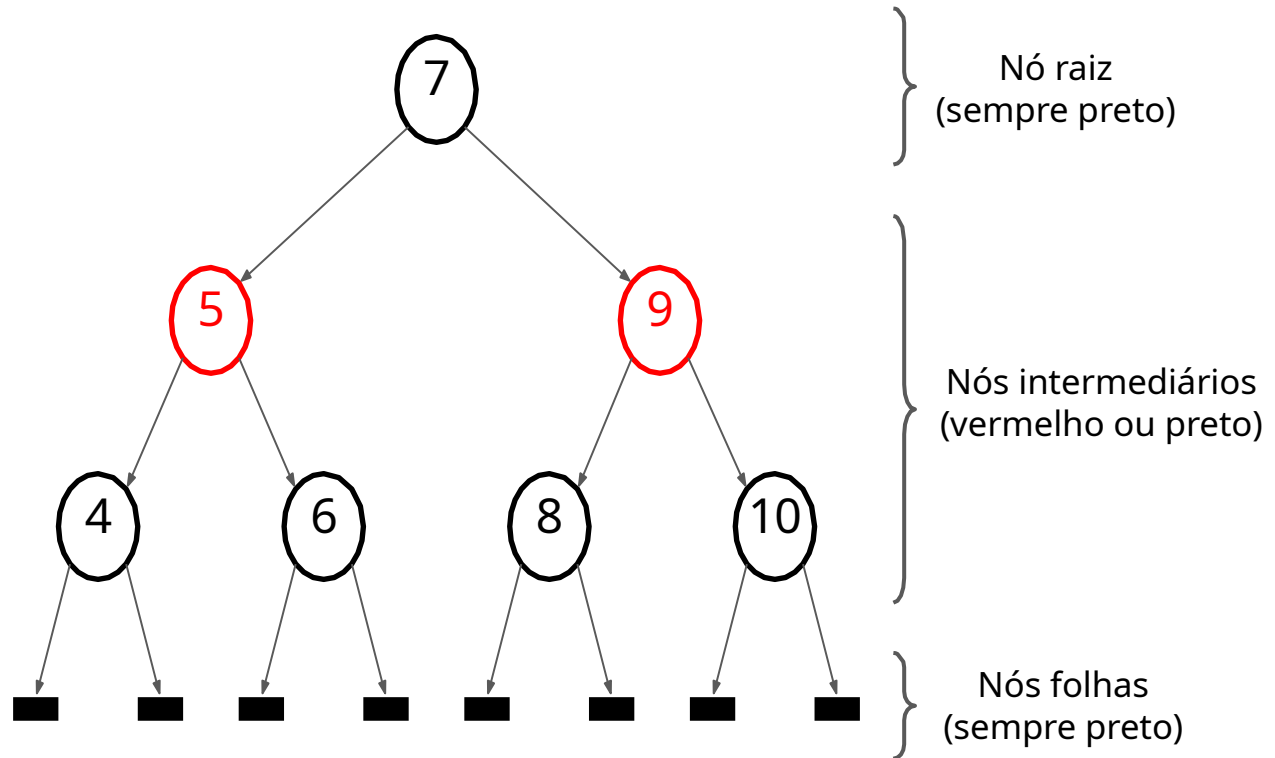
Propriedades da árvore rubro-negra

- Todo nó da árvore possui um atributo adicional que define a cor
 - A cor do nó será utilizada para o balanceamento dos nós
 - Ou seja, as operações de balanceamento baseiam-se na cor dos nós
- Uma árvore rubro-negra deve respeitar as seguintes regras
 - Regra 1: Todo nó possui cor vermelha ou é preta
 - Regra 2: a raiz sempre será preta
 - Regra 3: todo nó folha possui cor preta
 - Regra 4: se um nó é vermelho então ambos os seus filhos são pretos
 - Regra 5: para cada nó, todos os caminhos deste nó até os nós folha da sua subárvore possuem o mesmo número de nós pretos

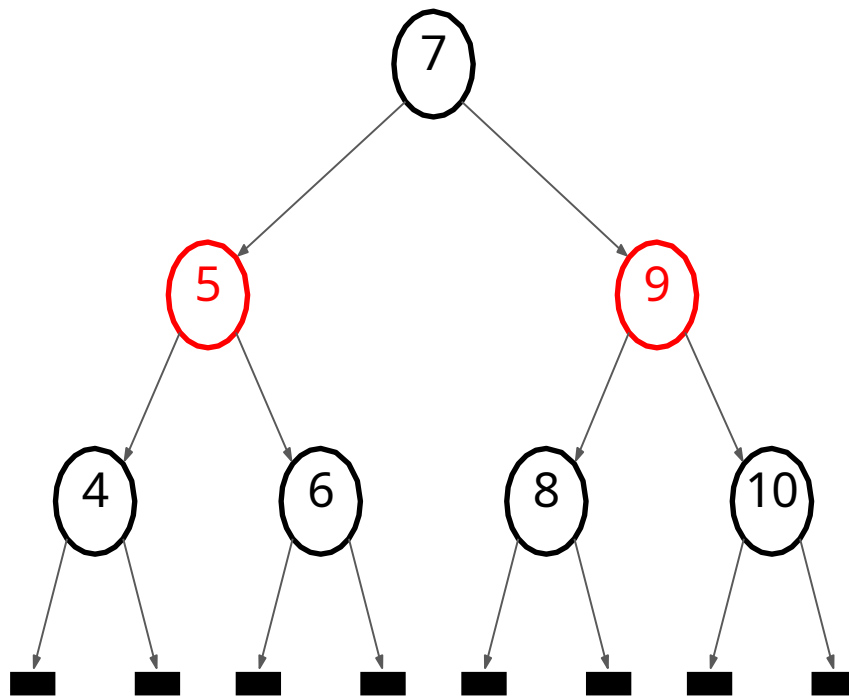
Propriedades da árvore rubro-negra

- Ao restringir a maneira que os nós podem ser coloridos do caminho da raiz até qualquer folhas, assegura-se que
 - Nenhum caminho será maior que o dobro do comprimento de outro
 - A árvore será aproximadamente balanceada
- Em árvores rubro-negra, os nós folha são irrelevantes e sem dados
 - É comum os nós folhas serem representados com ponteiros nulos
 - Porém algumas operações são simplificadas ao explicitar os nós folha
 - Usa-se um nó sentinela e todos os ponteiros para as folhas apontam para ele

Coloração dos nós

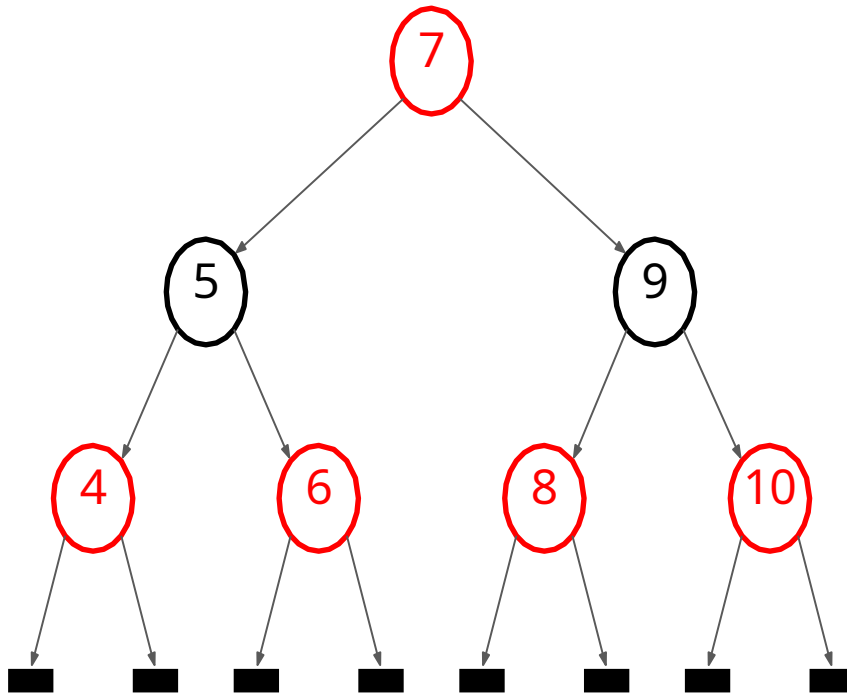


Coloração dos nós



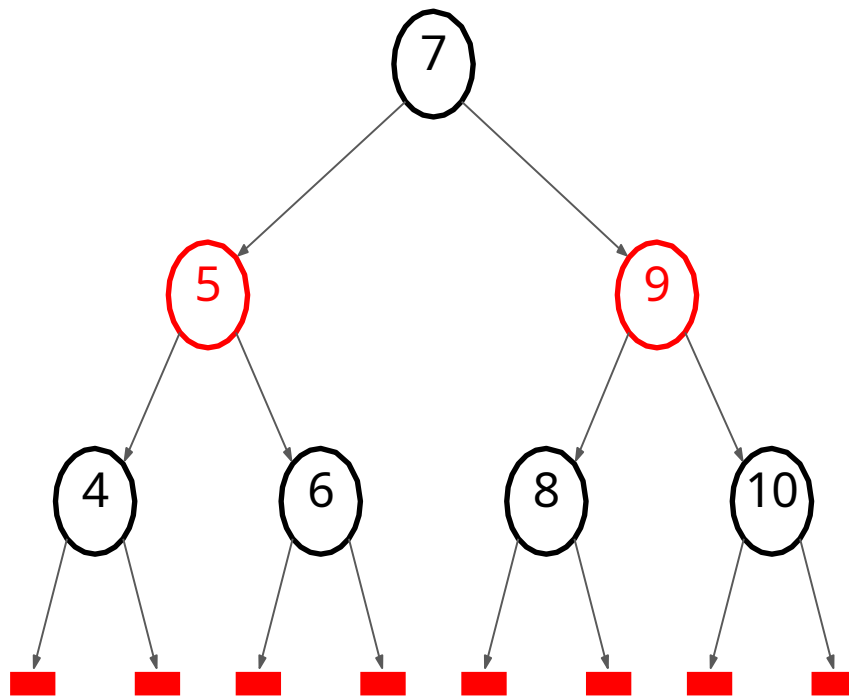
- ✓ Regra 1: Todo nó é vermelho ou é preto
- ✓ Regra 2: a raiz sempre será preto
- ✓ Regra 3: todo nó folha é preto
- ✓ Regra 4: se um nó é vermelho então ambos os seus filhos são pretos
- ✓ Regra 5: todos os caminhos de um nó até as folhas possuem o mesmo número de nós pretos

Coloração dos nós



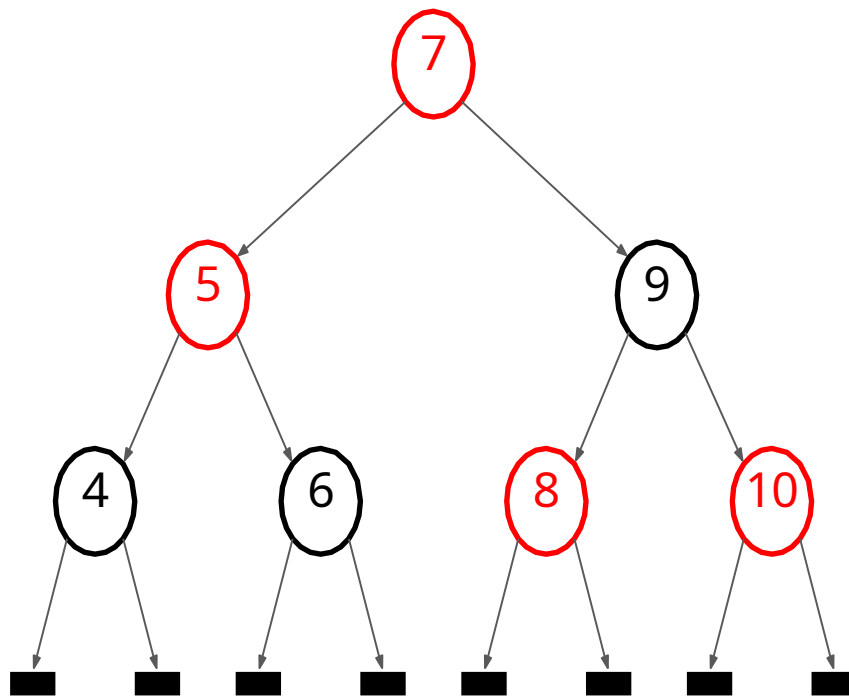
- ✓ Regra 1: Todo nó é vermelho ou é preto
- ✗ Regra 2: a raiz sempre será preto
- ✓ Regra 3: todo nó folha é preto
- ✓ Regra 4: se um nó é vermelho então ambos os seus filhos são pretos
- ✓ Regra 5: todos os caminhos de um nó até as folhas possuem o mesmo número de nós pretos

Coloração dos nós



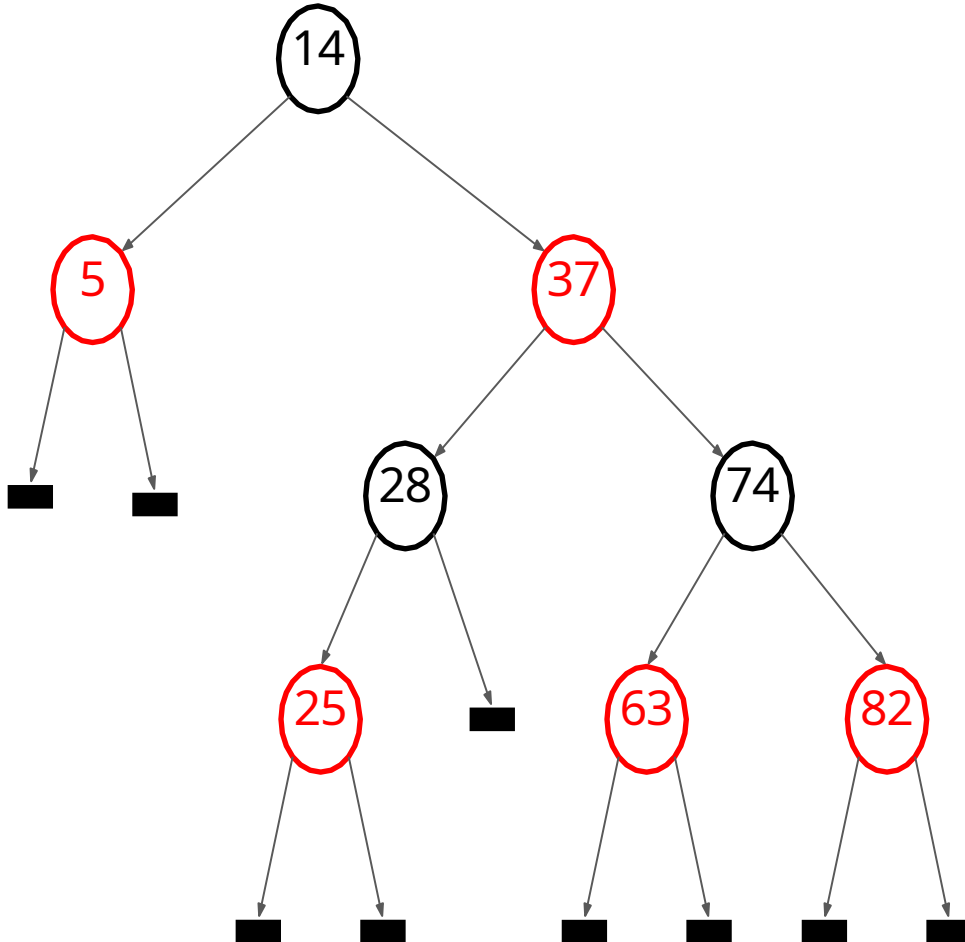
- ✓ Regra 1: Todo nó é vermelho ou é preto
- ✓ Regra 2: a raiz sempre será preto
- ✗ Regra 3: todo nó folha é preto
- ✓ Regra 4: se um nó é vermelho então ambos os seus filhos são pretos
- ✓ Regra 5: todos os caminhos de um nó até as folhas possuem o mesmo número de nós pretos

Coloração dos nós



- ✓ Regra 1: Todo nó é vermelho ou é preto
- ✓ Regra 2: a raiz sempre será preto
- ✓ Regra 3: todo nó folha é preto
- ✗ Regra 4: se um nó é vermelho então ambos os seus filhos são pretos
- ✓ Regra 5: todos os caminhos de um nó até as folhas possuem o mesmo número de nós pretos

Coloração dos nós



- ✓ Regra 1: Todo nó é vermelho ou é preto
- ✓ Regra 2: a raiz sempre será preto
- ✓ Regra 3: todo nó folha é preto
- ✓ Regra 4: se um nó é vermelho então ambos os seus filhos são pretos
- ✗ Regra 5: todos os caminhos de um nó até as folhas possuem o mesmo número de nós pretos

Estrutura da árvore rubro-negra

- Representação em C

```
typedef enum coloracao { Vermelho, Preto } Cor;
```

```
typedef struct no {  
    struct no* pai;  
    struct no* esquerda;  
    struct no* direita;  
    Cor cor; //cor do nó (Vermelho ou Preto)  
    int valor;  
} No;
```

Operações em árvore rubro-negra

- Adicionar novos elementos
 - Cria um novo nó
 - Atribui a cor vermelha ao nó
 - Associa o nó à chave que deseja ser adicionada
 - Localiza onde o novo nó que será adicionado
 - Realiza uma busca binária para localizar o nó
 - Realiza o balanceamento prevendo o nó recém inserido
 - Corrige possíveis violações das regras da árvore rubro-negra

Balanceamento da árvore rubro-negra

- Existem duas operações para rebalancear uma árvore rubro-negra
 - Rotação a esquerda ou à direita
 - São operações locais que alteram alguns ponteiros dos nós relacionados
 - Recoloração de nós envolvidos
 - A fim de evitar dois nós vizinhos com cor vermelha
- O rebalanceamento pode ocorrer em 4 casos distintos
 - Visam corrigir possíveis violações das regras da árvore rubro-negra

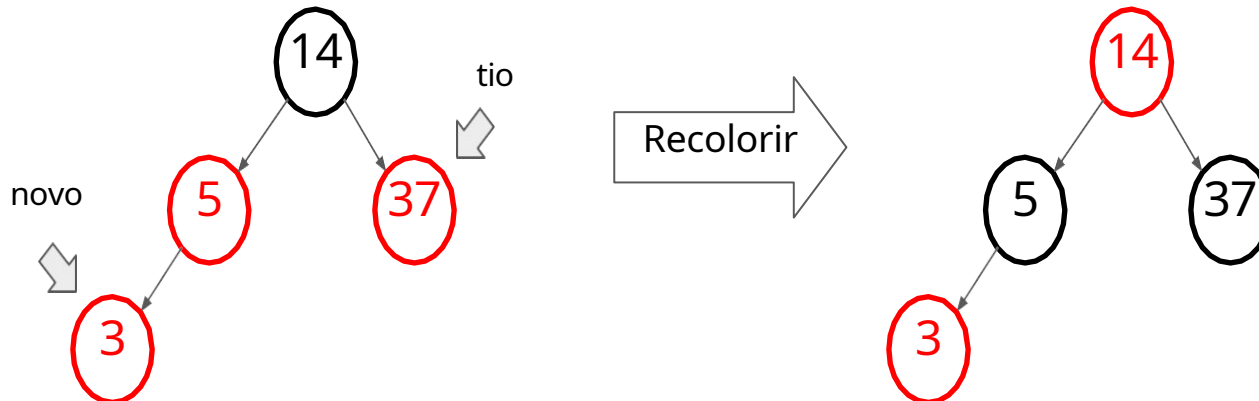
Balanceamento da árvore rubro-negra

- Caso 1: árvore vazia
 - Ocorre após adicionar o nó raiz, a cor do nó será vermelha
 - Este caso viola a regra 2 (o nó raiz precisa ser preto)
 - Para corrigir, deve ser atribuída a cor preta à raiz da árvore
 - Exemplo
 - Adicionar o nó 14
 - Recolorir nó 14



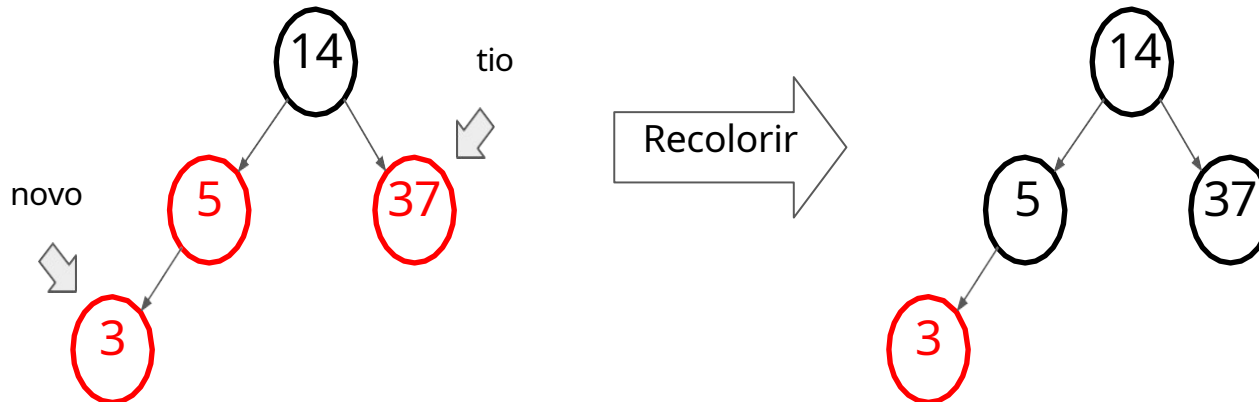
Balanceamento da árvore rubro-negra

- Caso 2: nós pai e filho vermelhos com **tio vermelho**
 - Ocorre após adicionar um nó como filho de outro de cor vermelha
 - Este caso viola a regra 4 (nó vermelho possui ambos os filhos pretos)
 - Para corrigir, deve ser atribuída a cor preta à raiz da árvore
 - Exemplo
 - Adicionar o nó 3
 - Recolorir nós 5 e 37



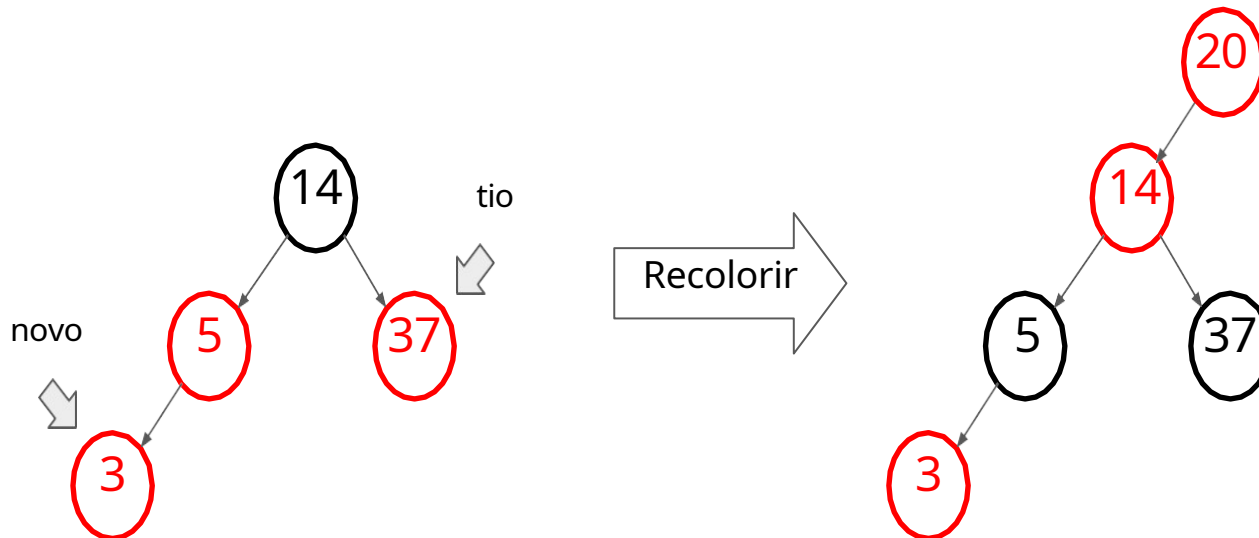
Balanceamento da árvore rubro-negra

- Caso 2: nós pai e filho vermelhos com **tio vermelho** (cont.)
 - Exemplo
 - Adicionar o nó 3
 - Recolorir nós 5 e 37
 - E se o nó 14 for raiz da árvore?
 - Nesse caso, basta recolorir o nó raiz



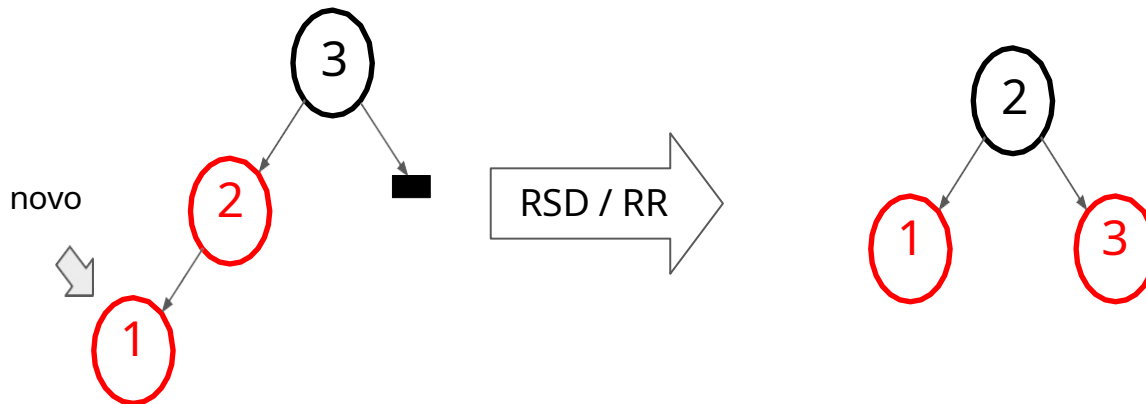
Balanceamento da árvore rubro-negra

- Caso 2: nós pai e filho vermelhos com **tio vermelho** (cont.)
 - Exemplo
 - Adicionar o nó 3
 - Recolorir nós 5 e 37
 - E se o nó 14 for filho de um nó vermelho?
 - O processo deve ser feito iterativamente nos níveis anteriores da árvore
 - A mudança de cor de um nó pode afetar os níveis anteriores



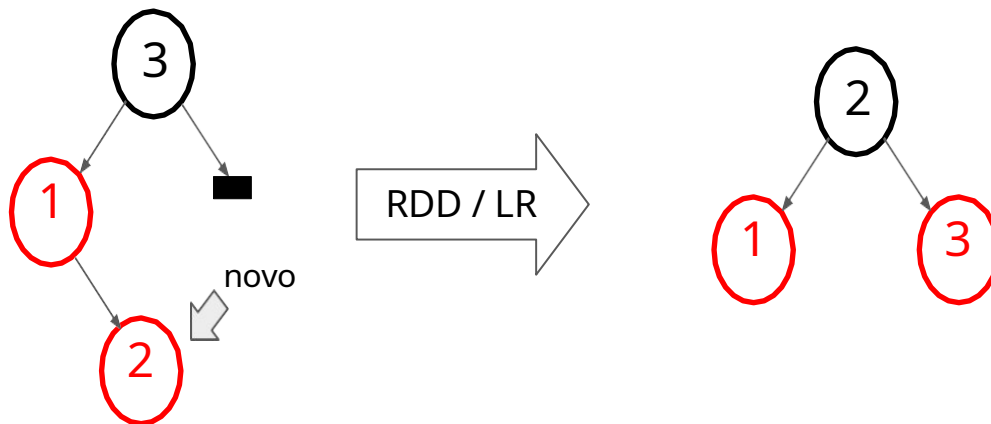
Balanceamento da árvore rubro-negra

- Caso 3: **tio** do elemento inserido é **preto**
 - Quando o nó pai do elemento inserido também é vermelho e o tio é preto, a solução envolve a rotação (como na árvore AVL)
 - Rotação simples à esquerda
 - Rotação simples à direita
 - Rotação dupla à esquerda
 - Rotação dupla à direita

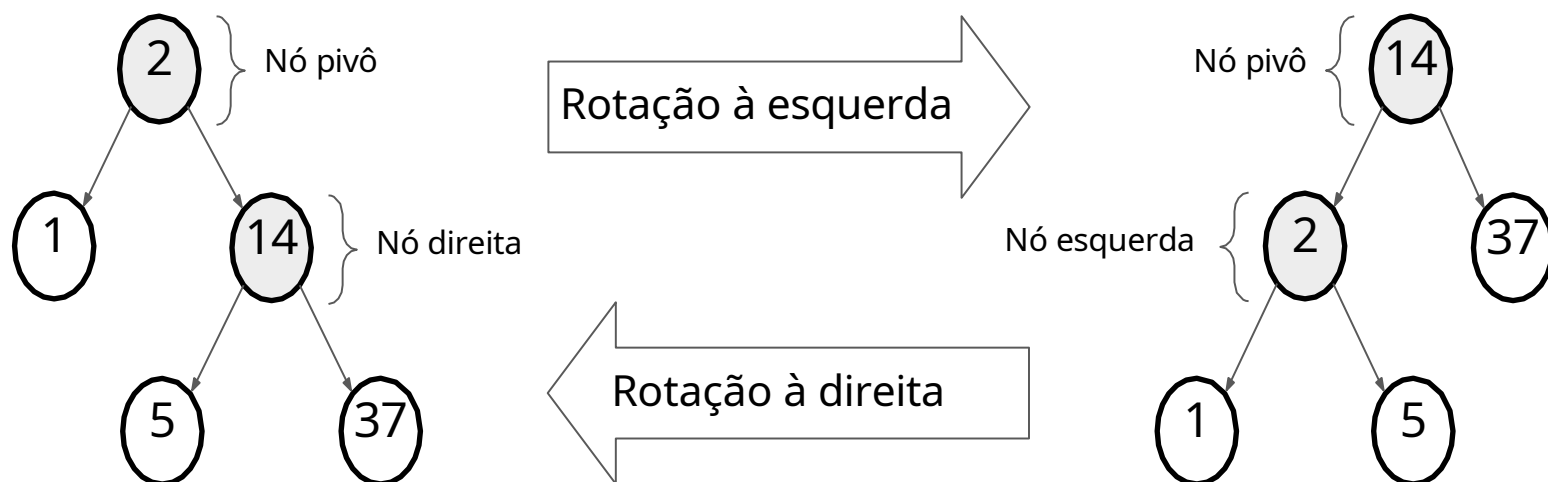


Balanceamento da árvore rubro-negra

- Caso 3: **tio** do elemento inserido é **preto**
 - Quando o nó pai do elemento inserido também é vermelho e o tio é preto, a solução envolve a rotação (como na árvore AVL)
 - Rotação simples à esquerda
 - Rotação simples à direita
 - Rotação dupla à esquerda
 - Rotação dupla à direita



Balanceamento da árvore rubro-negra

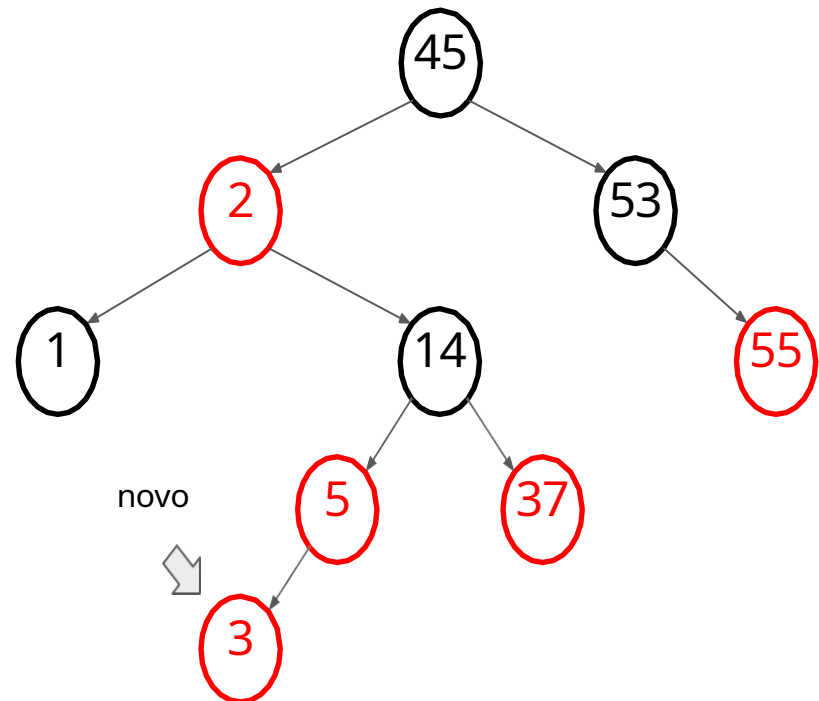


Balanceamento da árvore rubro-negra

- Caso 3: **tio** do elemento inserido é **preto**
 - Ocorre após recoloração para resolver o caso 2
 - Este caso também viola a regra 4 (nó vermelho possui ambos filhos pretos)
 - Para corrigir, deve ser realizada uma rotação à esquerda

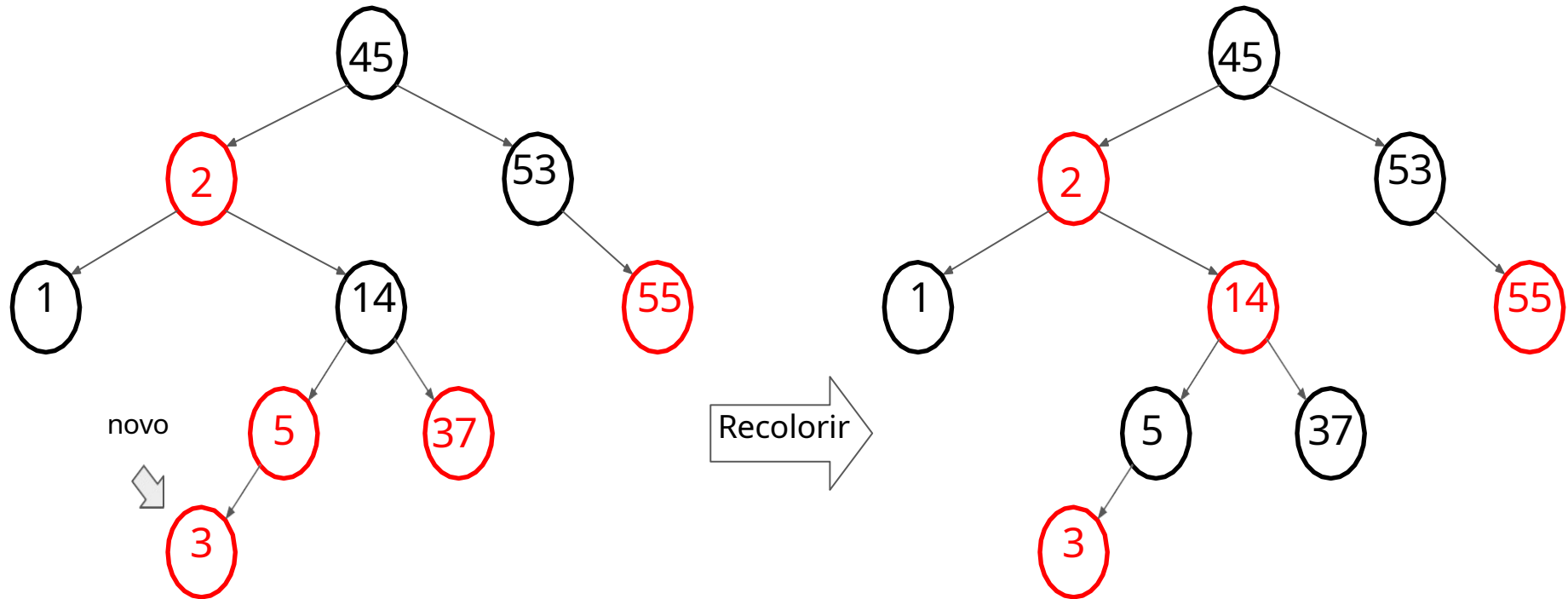
- Exemplo

- Adicionar o nó 3
- Resolver o caso 2
- Ir para o nível anterior
- Rotação à esquerda em 2



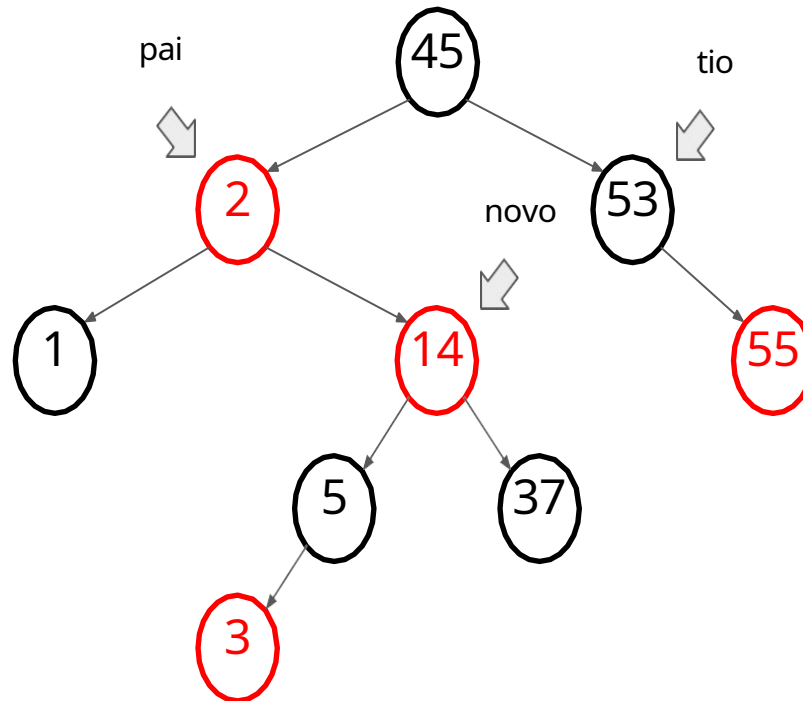
Balanceamento da árvore rubro-negra

- Caso 3: **tio** do elemento inserido é **preto** (cont.)
 - Exemplo
 - Resolver caso 2



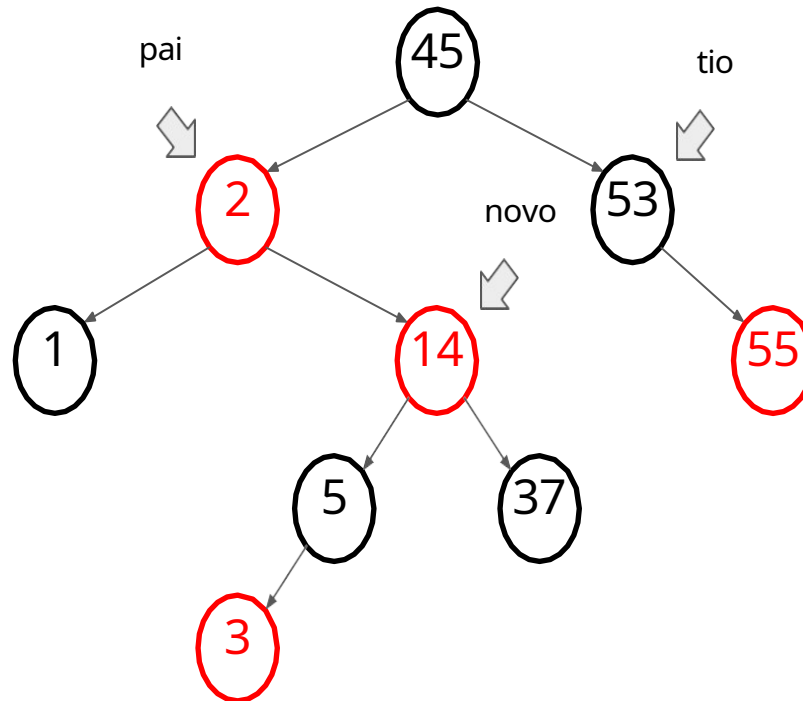
Balanceamento da árvore rubro-negra

- Caso 3: **tio** do elemento inserido é **preto** (cont.)
 - Exemplo
 - Ir para o nível anterior



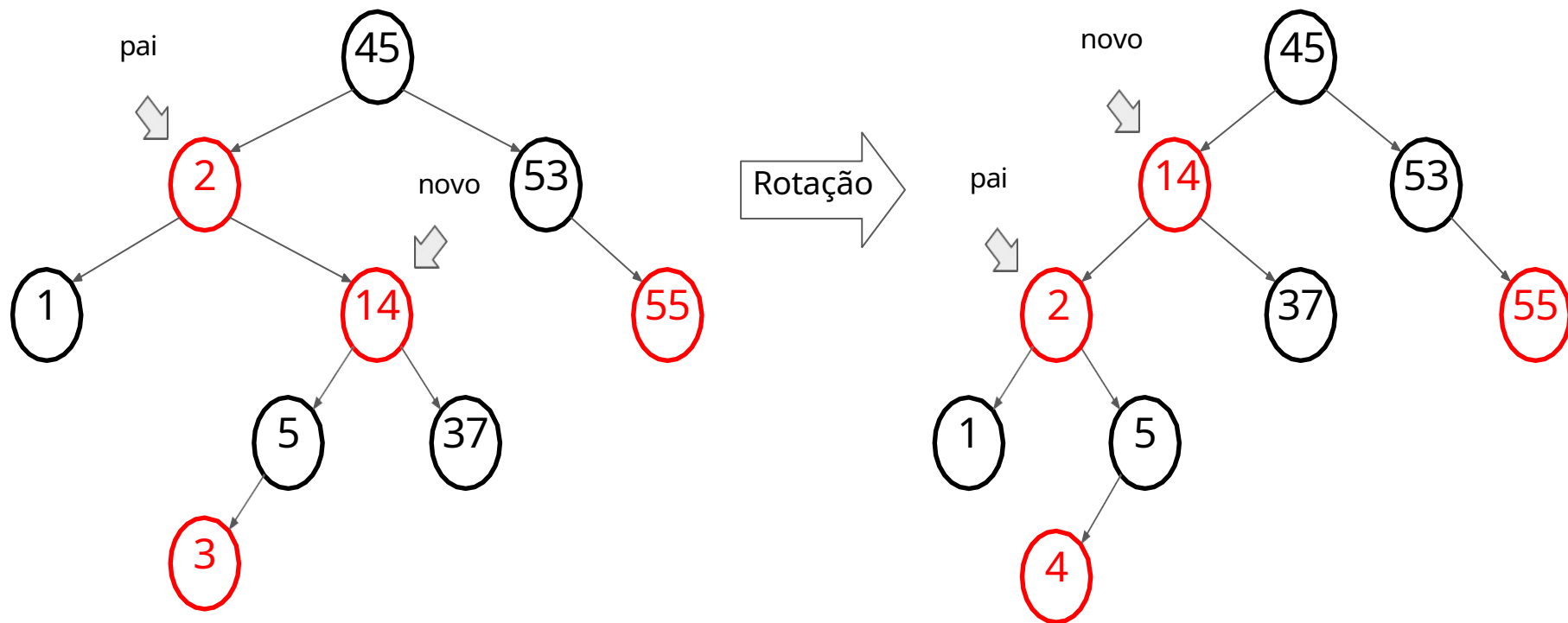
Balanceamento da árvore rubro-negra

- Caso 3: **tio** do elemento inserido é **preto** (cont.)
 - Exemplo
 - Rotação à esquerda em 2



Balanceamento da árvore rubro-negra

- Caso 3: **tio** do elemento inserido é **preto** (cont.)
 - Exemplo
 - Rotação à esquerda em 2



Balanceamento da árvore rubro-negra

- Balanceamento

```
void balancear(Arvore* arvore, No* no) {
    while (no->pai->cor == Vermelho) { //Garante que todos os níveis foram balanceados
        if (no->pai == no->pai->pai->esquerda) {
            No *tio = no->pai->pai->direita;

            if (tio->cor == Vermelho) {
                tio->cor = Preto; //Resolve caso 2
                no->pai->cor = Preto;
                no->pai->pai->cor = Vermelho;
                no = no->pai->pai; //Vai para o nível anterior
            } else {
                if (no == no->pai->direita) {
                    no = no->pai; //Vai para o nível anterior
                    rotacionarEsquerda(arvore, no); //Resolve caso 3
                } else {
                    no->pai->cor = Preto; //Resolve caso 4
                    no->pai->pai->cor = Vermelho;
                    rotacionarDireita(arvore, no->pai->pai);
                }
            }
        } else {
            //Repete o mesmo código do bloco if, invertendo o lado dos direita e esquerda
        }
    }
    arvore->raiz->cor = Preto; //Resolve caso 1
}
```

Balanceamento da árvore rubro-negra

- Rotação à esquerda

```
void rotacionarEsquerda(Arvore* arvore, No* no) {
    No* direita = no->direita;
    no->direita = direita->esquerda;

    if (direita->esquerda != NULL)
        direita->esquerda->pai = no; //Se houver filho à esquerda em direita, ele será pai do nó

    direita->pai = no->pai; //Ajusta no pai do nó à direita

    if (no->pai == NULL)
        arvore->raiz = direita; //Se nó for raiz, o nó direita será a nova raiz da árvore
    else if (no == no->pai->esquerda)
        no->pai->esquerda = direita; //Corrige relação pai-filho do novo pai (esquerda)
    else
        no->pai->direita = direita; //Corrige relação pai-filho do novo pai (direita)

    direita->esquerda = no; //Corrige relação pai-filho entre o nó pivô e o nó à direita
    no->pai = direita;
}
```

Balanceamento da árvore rubro-negra

- Rotação à direita

```
void rotacionarDireita(Arvore* arvore, No* no) {
    No* esquerda = no->esquerda;
    no->esquerda = esquerda->direita;

    if (esquerda->direita != NULL)
        esquerda->direita->pai = no; //Se houver filho à direita em esquerda, ele será pai do nó

    esquerda->pai = no->pai; //Ajusta no pai do nó à esquerda

    if (no->pai == NULL)
        arvore->raiz = esquerda; //Se nó for raiz, o nó esquerda será a nova raiz da árvore
    else if (no == no->pai->esquerda)
        no->pai->esquerda = esquerda; //Corrige relação pai-filho do novo pai (esquerda)
    else
        no->pai->direita = esquerda; //Corrige relação pai-filho do novo pai (direita)

    esquerda->direita = no; //Corrige relação pai-filho entre o nó pivô e o nó à esquerda
    no->pai = esquerda;
}
```

Remoção na árvore rubro-negra

- O procedimento para remoção de um nó é similar aos algoritmos utilizados nas árvores ABB e AVL. Primeiro se busca o elemento a ser removido e depois remove ele mantendo a árvore "ordenada";
- Após a remoção, precisamos verificar se as regras da árvore Rubro-Negra ainda continuam válidas.

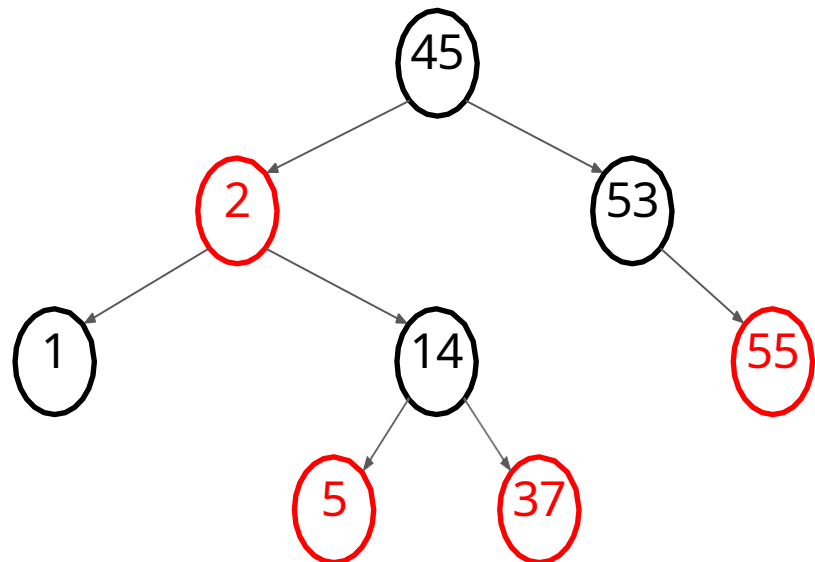
Remoção na árvore rubro-negra

- O procedimento para remoção de um nó é similar aos algoritmos utilizados nas árvores ABB e AVL. Primeiro se busca o elemento a ser removido e depois remove ele mantendo a árvore "ordenada";
- Após a remoção, precisamos verificar se as regras da árvore Rubro-Negra ainda continuam válidas.
- O primeiro caso ocorre quando se remove um nó vermelho que é folha. Neste caso, todas as regras são mantidas sem alteração. Basta, neste caso remover o nó e ajustar os ponteiros do nó pai.

Remoção na árvore rubro-negra

- O procedimento para remoção de um nó é similar aos algoritmos utilizados nas árvores ABB e AVL. Primeiro se busca o elemento a ser removido e depois remove ele mantendo a árvore "ordenada";
- Após a remoção, precisamos verificar se as regras da árvore Rubro-Negra ainda continuam válidas.
- O primeiro caso ocorre quando se remove um nó vermelho que é folha. Neste caso, todas as regras são mantidas sem alteração. Basta, neste caso remover o nó e ajustar os ponteiros do nó pai.

Nesse caso, a remoção dos nós **5**, **37** e **55** não violam as regras da árvore Rubro-Negra.

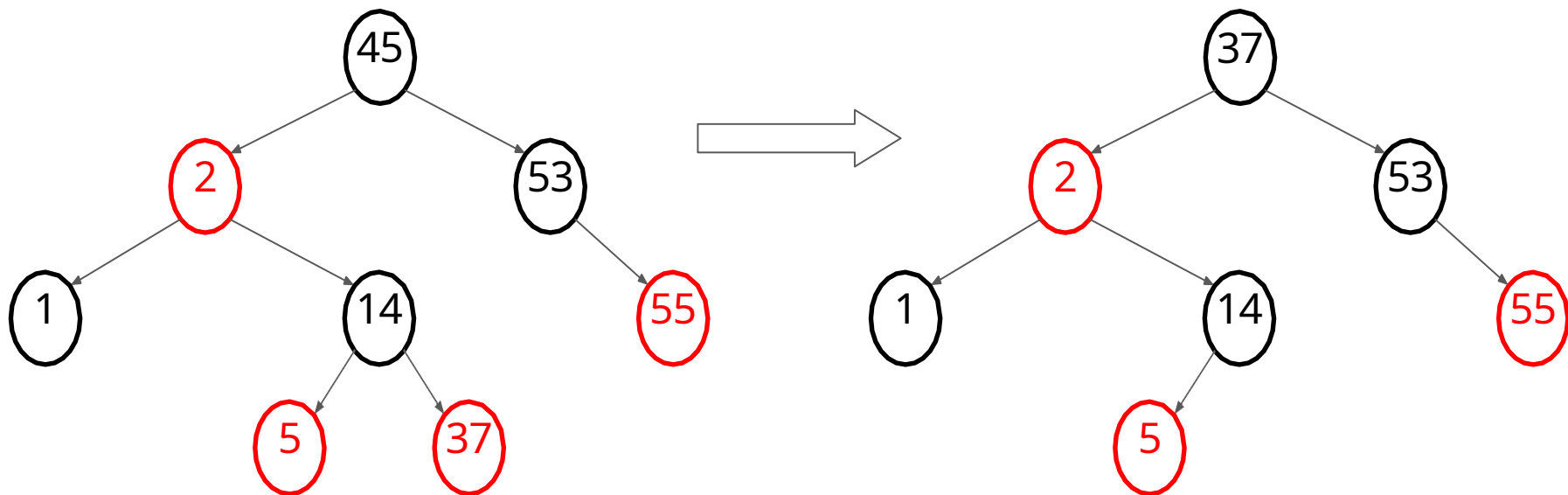


Remoção na árvore rubro-negra

- Lembremos que a remoção de um nó com dois filhos na verdade é apenas uma substituição:
 - O maior valor da esquerda (antecessor) ou menor da direita (sucessor) é quem é removido de fato.
 - Neste nó é que deve-se verificar as regras de balanceamento após a remoção.

Remoção na árvore rubro-negra

- Lembremos que a remoção de um nó com dois filhos na verdade é apenas uma substituição:
- O maior valor da esquerda (antecessor) ou menor da direita (sucessor) é quem é removido de fato.
- Neste nó é que deve-se verificar as regras de balanceamento após a remoção. *Ao remover o nó 45, substituímos ele por 37 e removemos o nó folha (vermelho no caso), mantendo a cor preta original.*



Remoção na árvore rubro-negra

- A remoção de um nó preto reduz o número de nós pretos em qualquer caminho que continha o nó que foi removido;
- Casos que não precisamos nos preocupar, pois não causa alteração nas regras da árvore Rubro-Negra:
 - Nó preto substituído por um nó vermelho (lembrando que mantém-se a cor preta na substituição);
 - Nó vermelho substituído por outro nó vermelho;
 - Nó vermelho folha removido (substituído por nó nulo).

Remoção na árvore rubro-negra

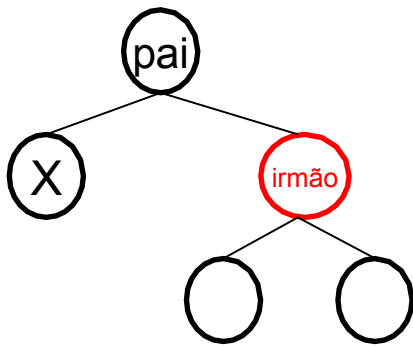
- A remoção de um nó preto reduz o número de nós pretos em qualquer caminho que continha o nó que foi removido;
- Casos que **precisamos** nos preocupar, pois causa alteração nas regras da árvore Rubro-Negra:
 - Nó preto substituído por um nó preto;
 - Nó preto substituído por uma folha nula;

Remoção na árvore rubro-negra

- A remoção de um nó preto reduz o número de nós pretos em qualquer caminho que continha o nó que foi removido;
- Casos que **precisamos** nos preocupar, pois causa alteração nas regras da árvore Rubro-Negra:
 - Nó preto substituído por um nó preto;
 - Nó preto substituído por uma folha nula;
- Se o nó é raiz, o caminho da raiz até as folhas não muda (já que muda na árvore toda).

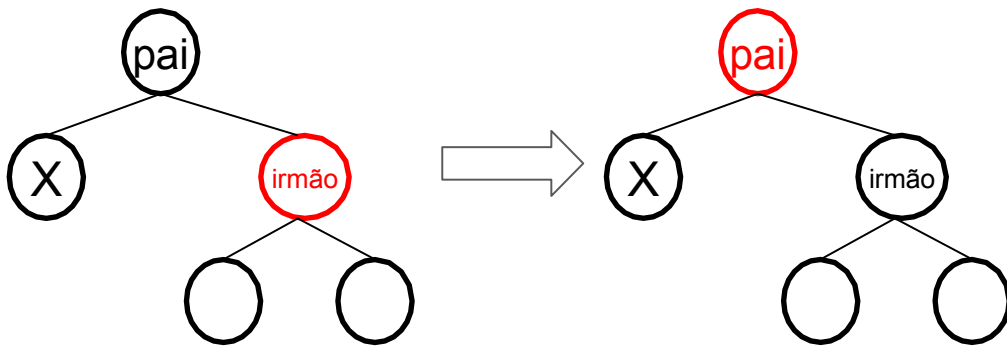
Remoção na árvore rubro-negra

- A remoção de um nó preto reduz o número de nós pretos em qualquer caminho que continha o nó que foi removido;
- Casos 1: irmão é vermelho
 - Pai fica vermelho;
 - Irmão fica preto;
 - Rotaciona pai à esquerda;



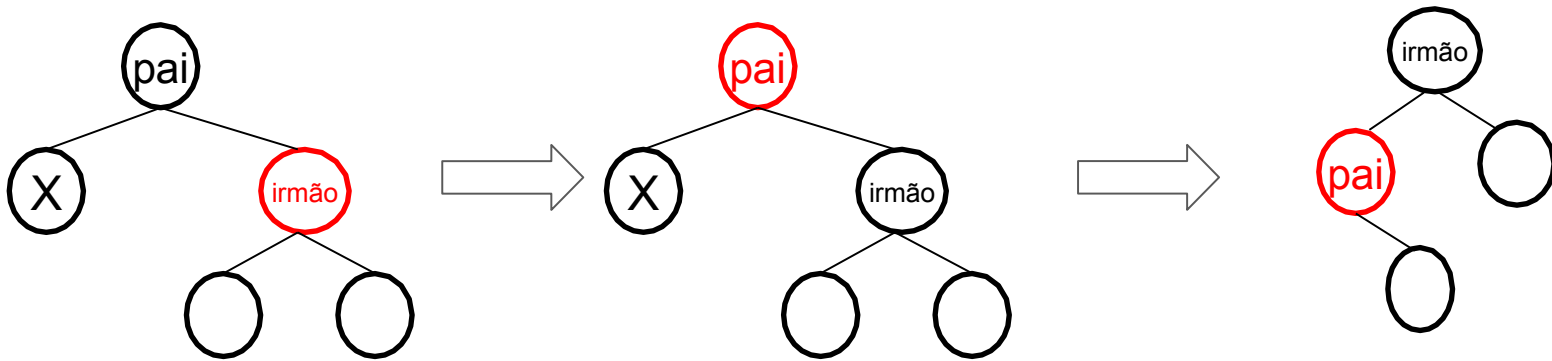
Remoção na árvore rubro-negra

- A remoção de um nó preto reduz o número de nós pretos em qualquer caminho que continha o nó que foi removido;
- Casos 1: irmão é vermelho
 - Pai fica vermelho;
 - Irmão fica preto;
 - Rotaciona pai à esquerda;



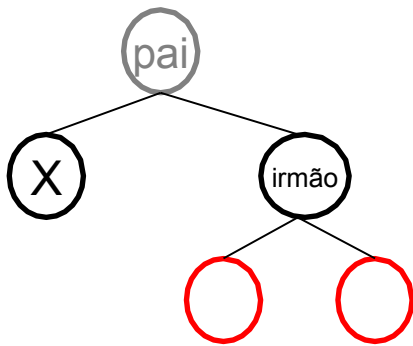
Remoção na árvore rubro-negra

- A remoção de um nó preto reduz o número de nós pretos em qualquer caminho que continha o nó que foi removido;
- Casos 1: irmão é vermelho
 - Pai fica vermelho;
 - Irmão fica preto;
 - Rotaciona pai à esquerda;



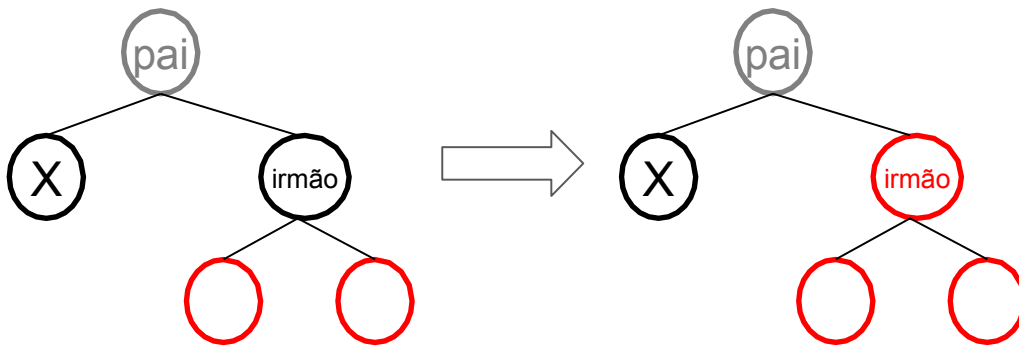
Remoção na árvore rubro-negra

- Casos 2: irmão é preto e filhos do irmão (primos) são vermelhos
 - Irmão fica vermelho;
 - Nó passa a apontar para o pai (sobe na árvore para testar próximos casos);
 - Se o pai era vermelho, basta trocar para preto;
 - Senão inicia uma nova verificação.



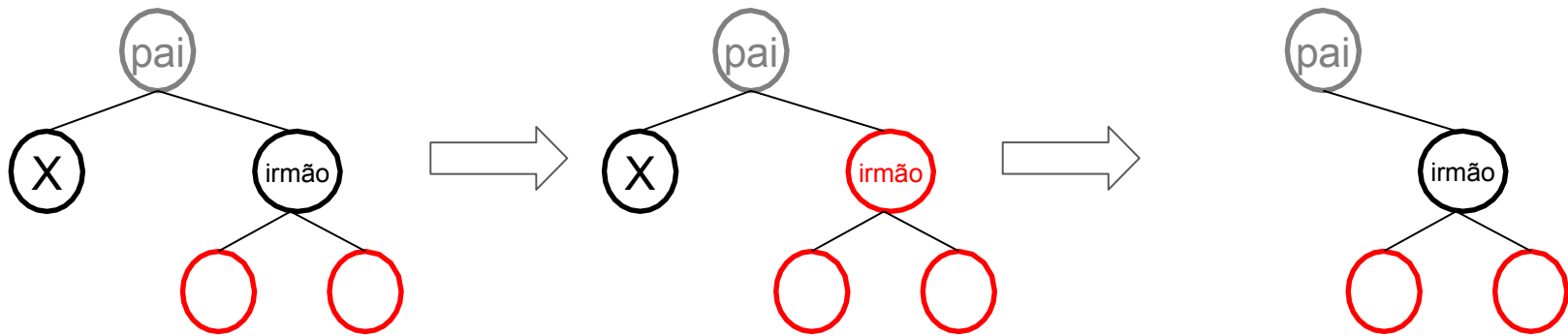
Remoção na árvore rubro-negra

- Casos 2: irmão é preto e filhos do irmão (primos) são vermelhos
 - Irmão fica vermelho;
 - Nó passa a apontar para o pai (sobe na árvore para testar próximos casos);
 - Se o pai era vermelho, basta trocar para preto;
 - Senão inicia uma nova verificação.



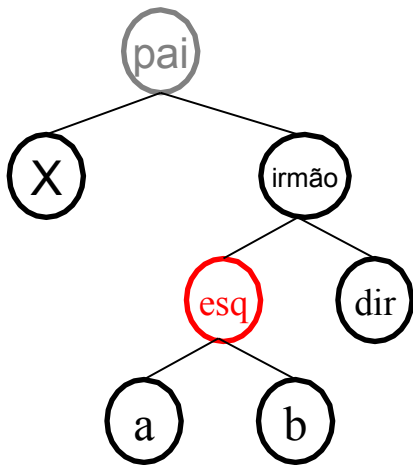
Remoção na árvore rubro-negra

- Casos 2: irmão é preto e filhos do irmão (primos) são vermelhos
 - Irmão fica vermelho;
 - Nó passa a apontar para o pai (sobe na árvore para testar próximos casos);
 - Se o pai era vermelho, basta trocar para preto;
 - Senão inicia uma nova verificação.



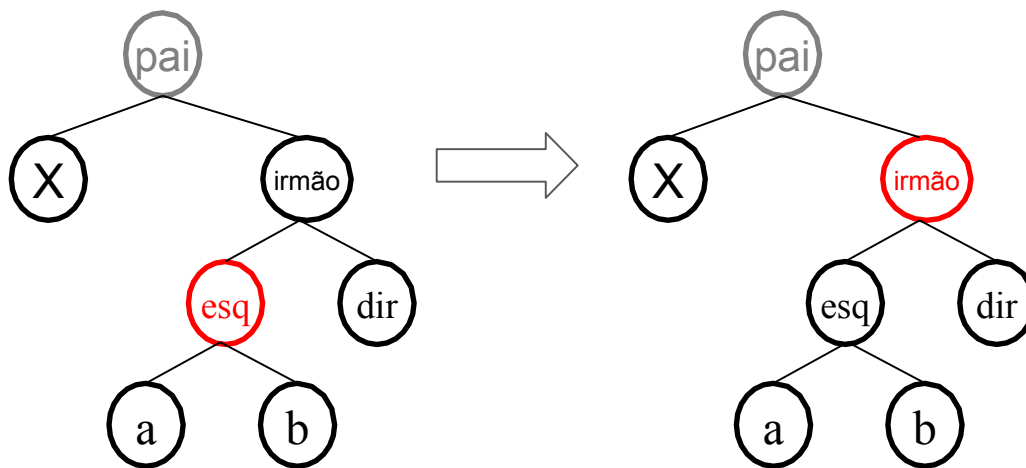
Remoção na árvore rubro-negra

- Casos 3: irmão da direita é preto, filho direito do irmão é preto e filho esquerdo é vermelho
 - Filho esquerdo fica preto;
 - Irmão fica vermelho;
 - Rotação à direita do irmão esquerdo.



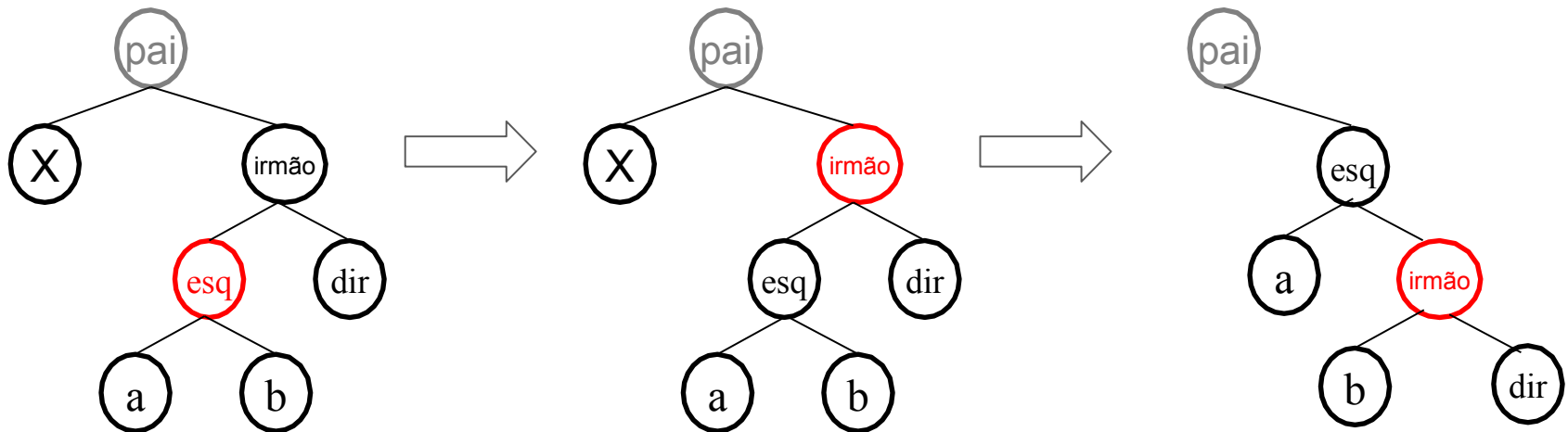
Remoção na árvore rubro-negra

- Casos 3: irmão da direita é preto, filho direito do irmão é preto e filho esquerdo é vermelho
 - Filho esquerdo fica preto;
 - Irmão fica vermelho;
 - Rotação à direita do irmão esquerdo.



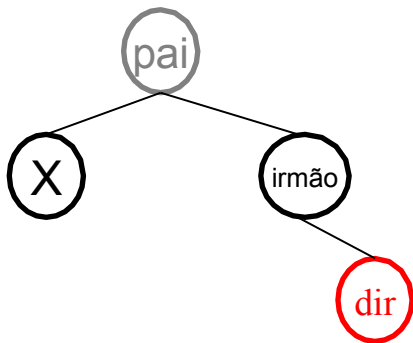
Remoção na árvore rubro-negra

- Casos 3: irmão da direita é preto, filho direito do irmão é preto e filho esquerdo é vermelho
 - Filho esquerdo fica preto;
 - Irmão fica vermelho;
 - Rotação à direita do irmão esquerdo.



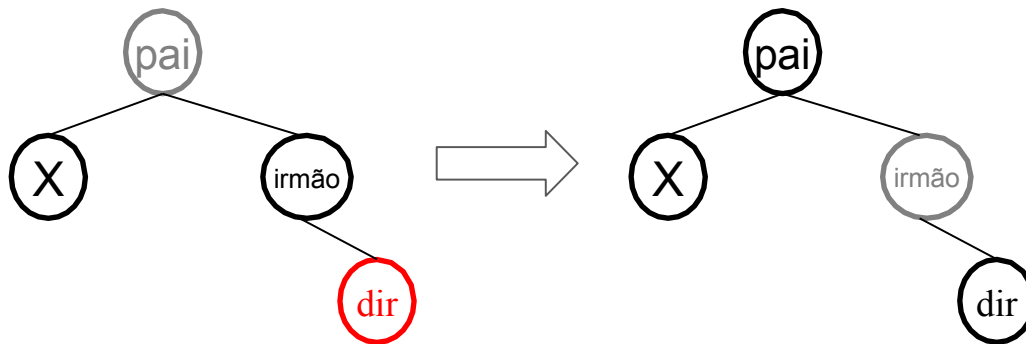
Remoção na árvore rubro-negra

- Casos 4: irmão da direita é preto e filho direito do irmão é vermelho
 - Irmão copia cor do pai
 - Pai e filho direito (dir) ficam pretos
 - Rotação pai à esquerda.
 - Se raiz for vermelha, trocar para preto.



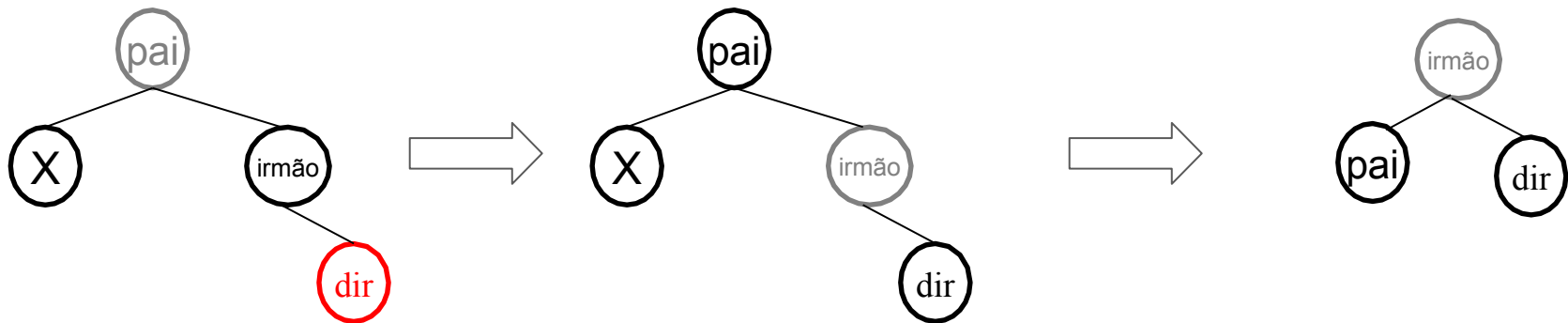
Remoção na árvore rubro-negra

- Casos 4: irmão da direita é preto e filho direito do irmão é vermelho
 - Irmão copia cor do pai
 - Pai e filho direito (dir) ficam pretos
 - Rotação pai à esquerda.
 - Se raiz for vermelha, trocar para preto.



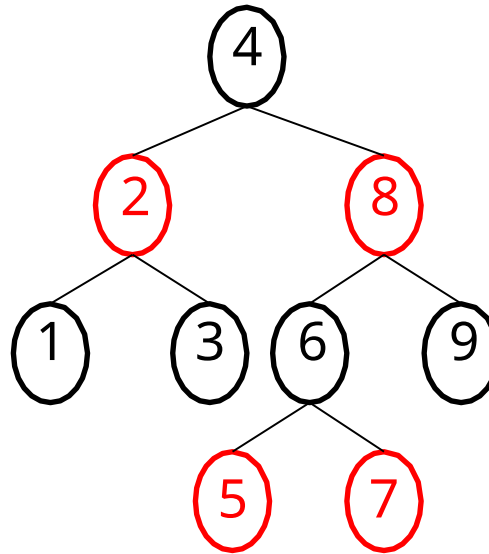
Remoção na árvore rubro-negra

- Casos 4: irmão da direita é preto e filho direito do irmão é vermelho
 - Irmão copia cor do pai
 - Pai e filho direito (dir) ficam pretos
 - Rotação pai à esquerda.
 - Se raiz for vermelha, trocar para preto.



Exercícios

1. Implemente uma árvore rubro-negra e adicione os nós de modo que a árvore apresente a respectiva topologia abaixo.



2. Implemente a operação de remoção de nós em uma árvore Rubro-Negra e valide removendo o nó 6 do exercício anterior.



Estruturas de dados II

Árvore Rubro-Negra

André Tavares da Silva
andre.silva@udesc.br