Árvores rubro-negra

Estruturas de dados II
Prof. Allan Rodrigo Leite

Árvores rubro-negra

- Árvore rubro-negra é uma árvore de busca binária autobalanceada
  - 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 Sedgewick 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

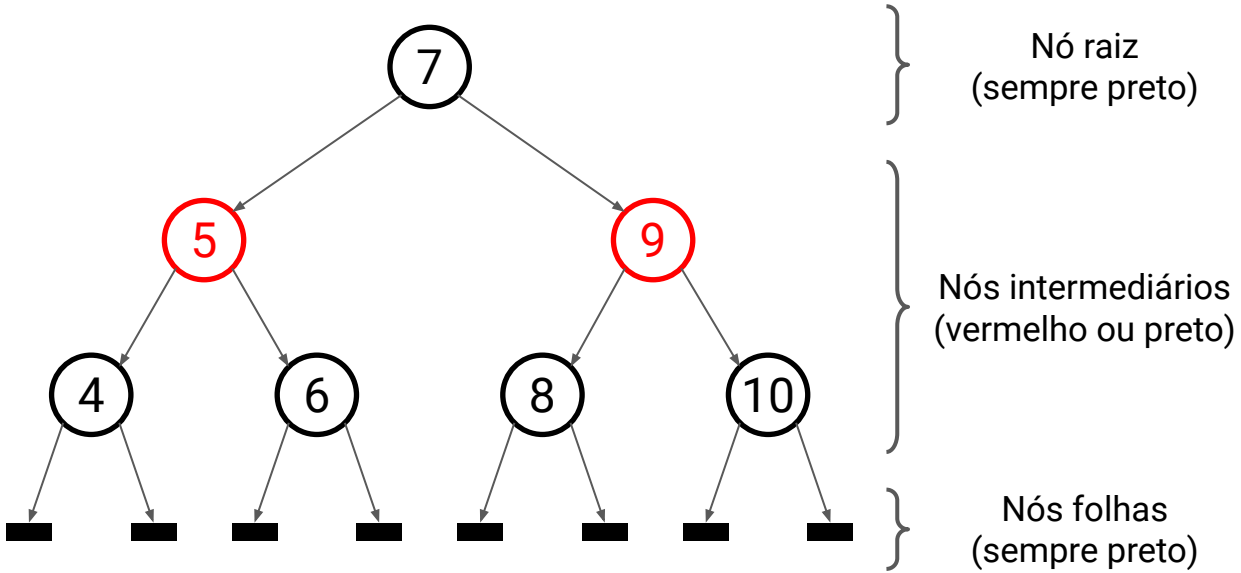
- Uma árvore rubro-negra deve respeitar as seguintes as regras
  - Regra 1: Todo nó possui cor vermelha ou é preta
  - Regra 2: a raiz sempre será preta
  - Regra 3: todo nó folha possui cor preta
  - Regra 4: se um nó é vermelho então ambos os seus filhos são pretos
  - Regra 5: para cada nó, todos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

Propriedades da árvore rubro-negra

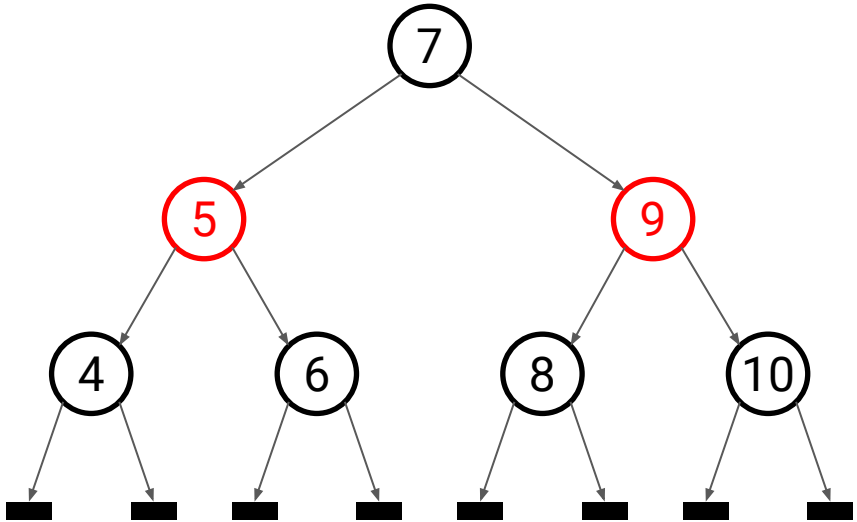
- Ao restringir a maneira que os nós podem ser coloridos do caminho da raiz até qualquer folhas, assegura-se que
  - Nenhum caminho será maior que o dobro do comprimento de outro
  - A árvore será aproximadamente balanceada

- 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

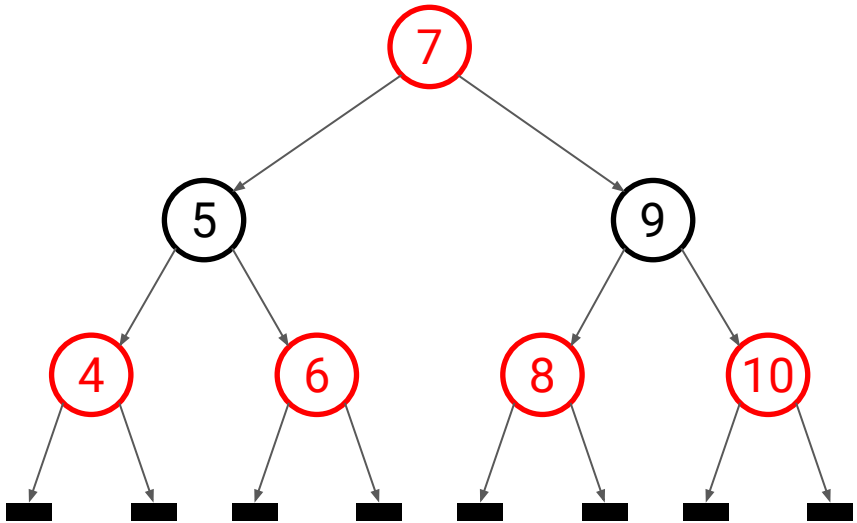


Coloração dos nós



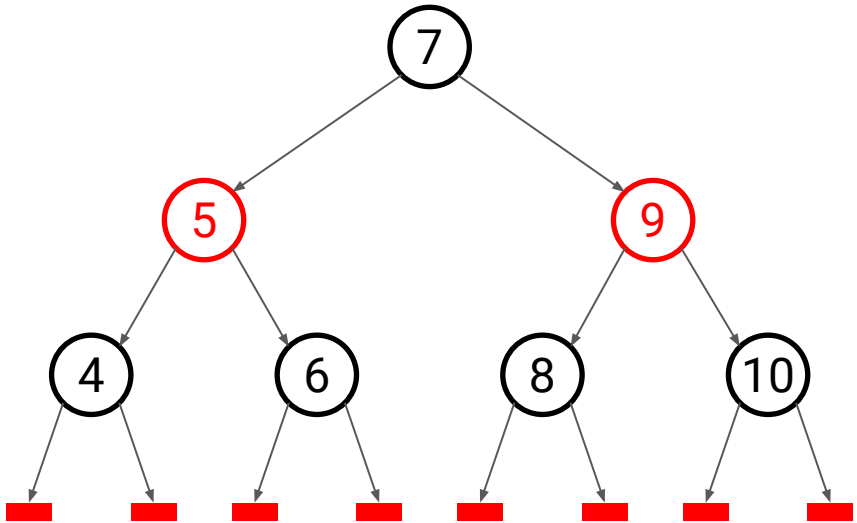
- Regra 1: Todo nó é vermelho ou é preto
- Regra 2: a raiz sempre será preto
- Regra 3: todo nó folha é preto
- Regra 4: se um nó é vermelho então ambos os seus filhos são pretos
- Regra 5: todos os caminhos de um nó até as folhas possuem o mesmo número de nós pretos

Coloração dos nós



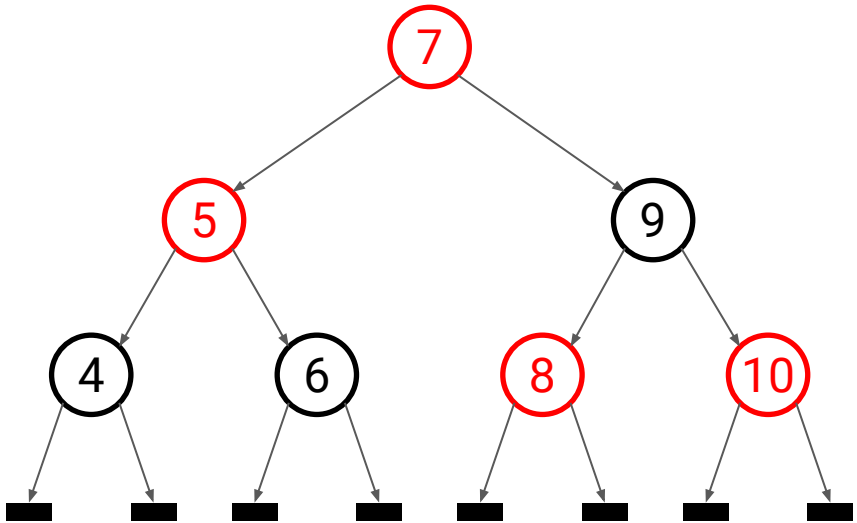
- ✓ Regra 1: Todo nó é vermelho ou é preto
- ✗ Regra 2: a raiz sempre será preto
- ✓ Regra 3: todo nó folha é preto
- ✓ Regra 4: se um nó é vermelho então ambos os seus filhos são pretos
- ✓ Regra 5: todos os caminhos de um nó até as folhas possuem o mesmo número de nós pretos

Coloração dos nós



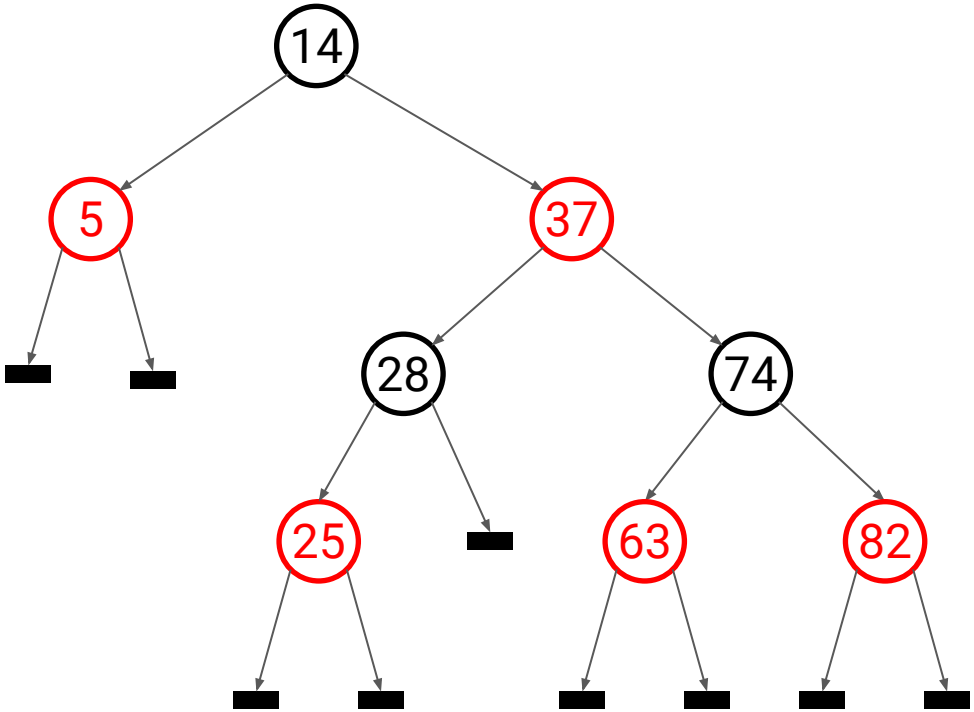
- Regra 1: Todo nó é vermelho ou é preto
- Regra 2: a raiz sempre será preto
- Regra 3: todo nó folha é preto
- Regra 4: se um nó é vermelho então ambos os seus filhos são pretos
- Regra 5: todos os caminhos de um nó até as folhas possuem o mesmo número de nós pretos

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

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

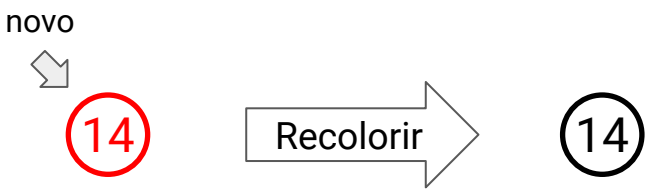
Balanceamento da árvore rubro-negra

- 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

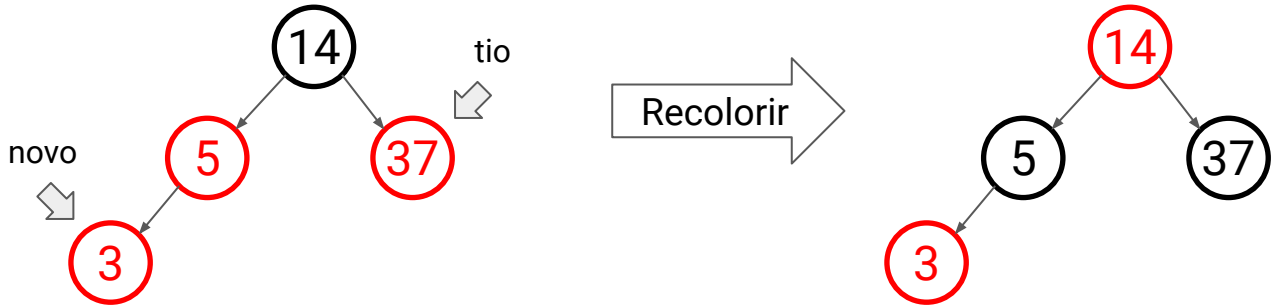
- Caso 1: árvore vazia
  - Ocorre após adicionar o nó raiz, a cor do nó será vermelha
    - Este caso viola a regra 2 (o nó raiz precisa ser preto)
    - Para corrigir, deve ser atribuída a cor preta à raiz da árvore

  - Exemplo
    - Adicionar o nó 14
    - Recolorir nó 14



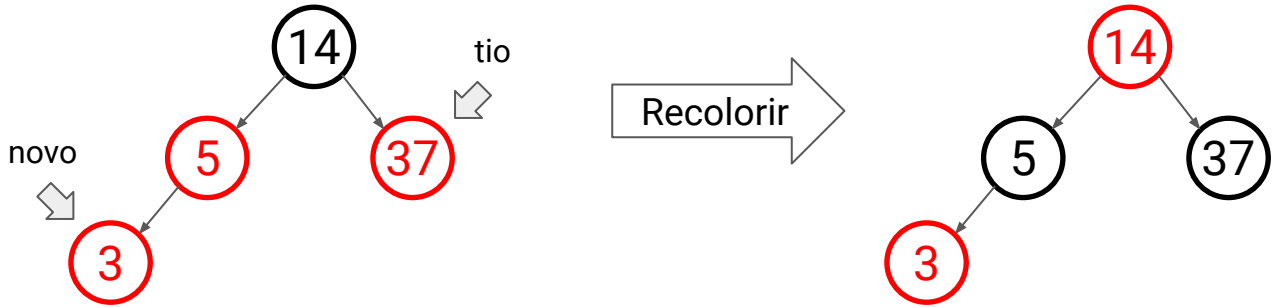
Balanceamento da árvore rubro-negra

- 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

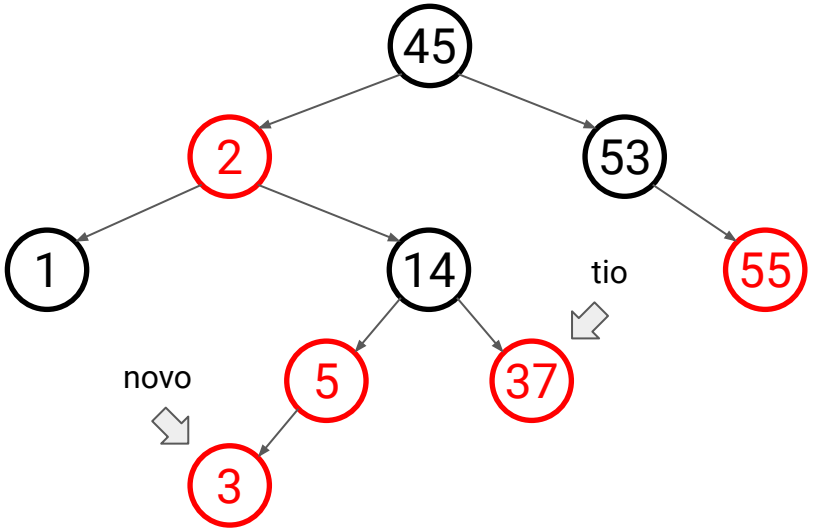
- Caso 2: nós pai e filho vermelhos com tio vermelho (cont.)
  - Exemplo
    - Adicionar o nó 3
    - Recolorir nós 5 e 37
  - E se o nó 14 for filho de um nó vermelho?
    - O processo deve ser feito iterativamente nos níveis anteriores da árvore
    - A mudança de cor de um nó pode afetar os níveis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



Balanceamento da árvore rubro-negra

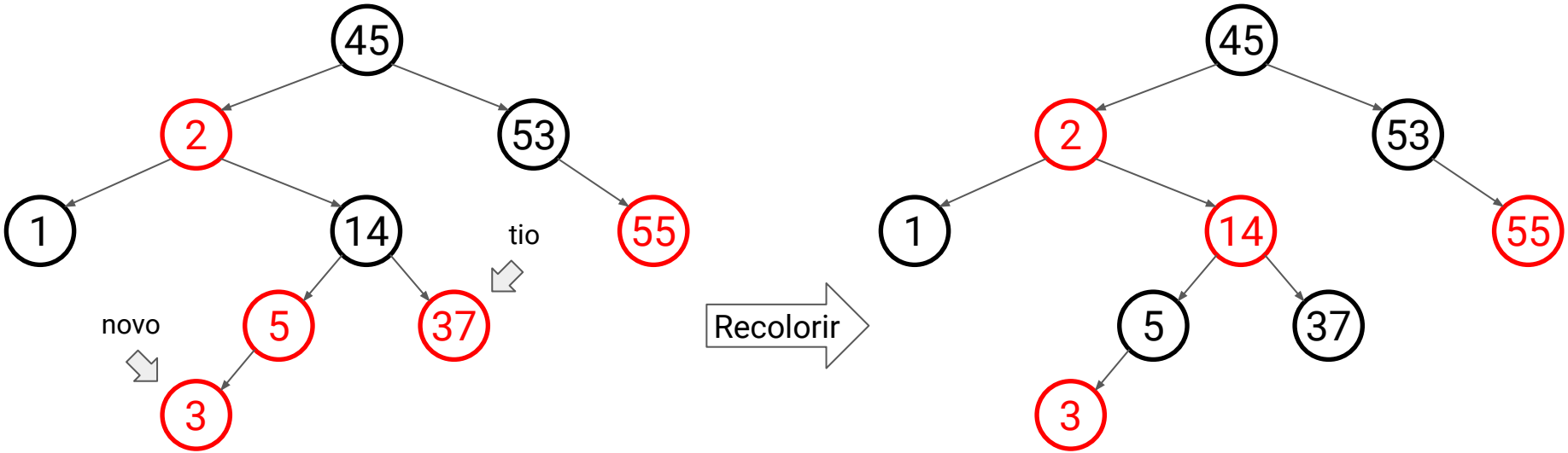
- Caso 3: nó pai e filho à direita vermelho com tio preto
  - Ocorre após recoloração para resolver o caso 2
    - Este caso também viola a regra 4 (nó vermelho possui ambos filhos pretos)
    - Para corrigir, deve ser realizada uma rotação à esquerda

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



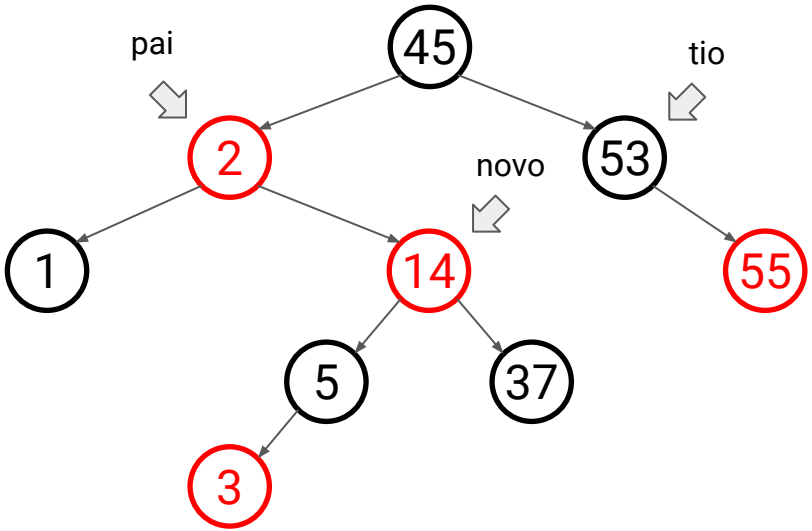
Balanceamento da árvore rubro-negra

- Caso 3: nó pai e filho à direita vermelho com tio preto (cont.)
  - Exemplo
    - Resolver caso 2



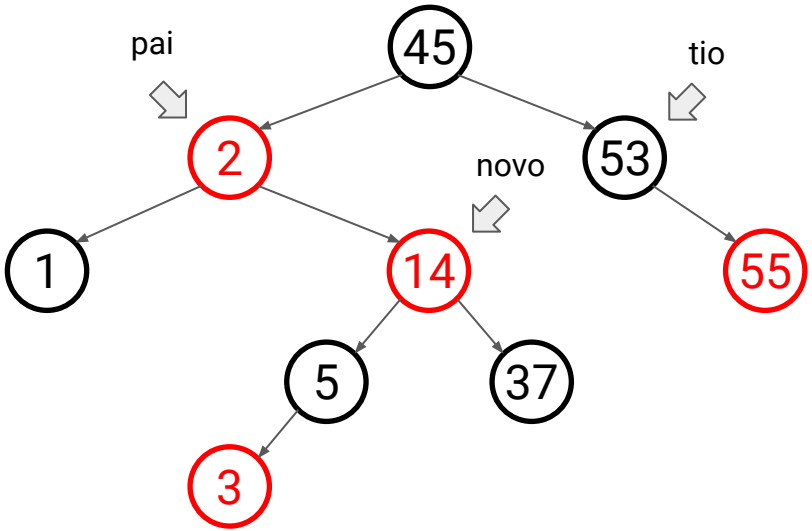
Balanceamento da árvore rubro-negra

- Caso 3: nó pai e filho à direita vermelho com tio preto (cont.)
  - Exemplo
    - Ir para o nível anterior

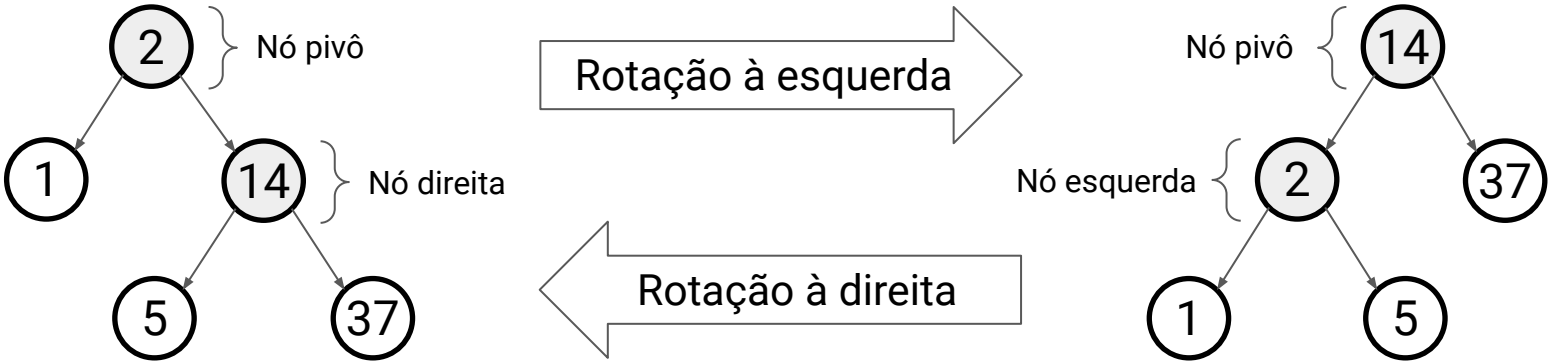


Balanceamento da árvore rubro-negra

- Caso 3: nó pai e filho à direita vermelho com tio preto (cont.)
  - Exemplo
    - Rotação à esquerda em 2

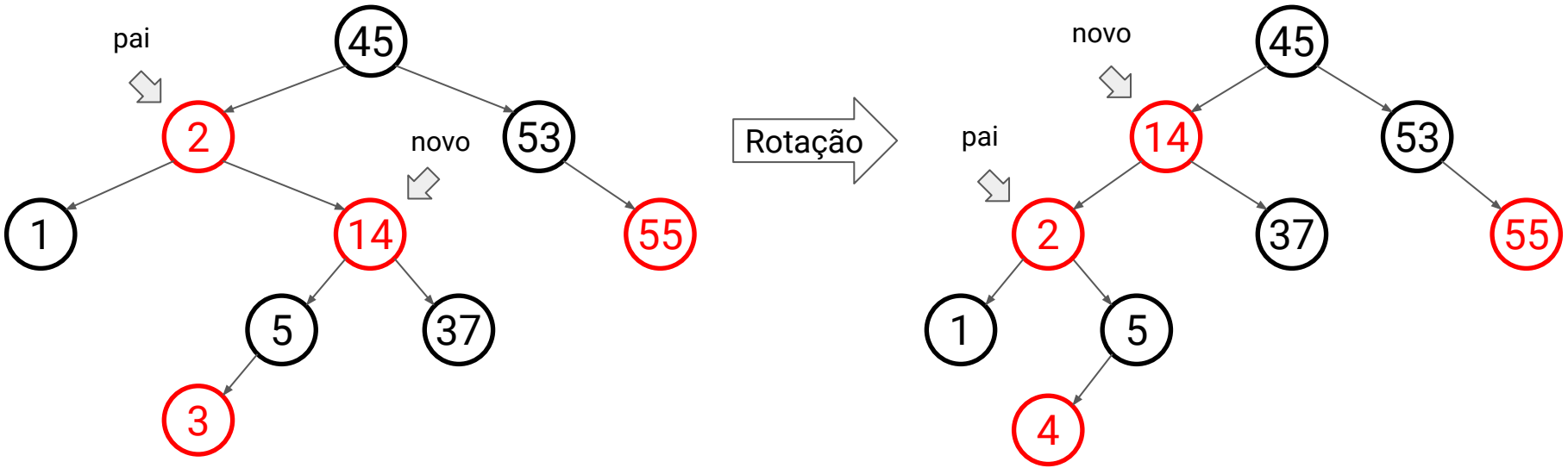


Balanceamento da árvore rubro-negra



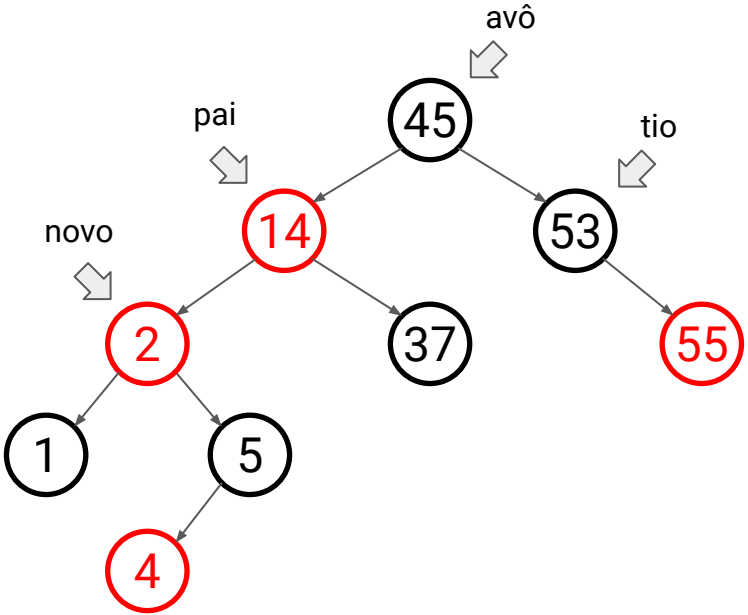
Balanceamento da árvore rubro-negra

- Caso 3: nó pai e filho à direita vermelho com tio preto (cont.)
  - Exemplo
    - Rotação à esquerda em 2



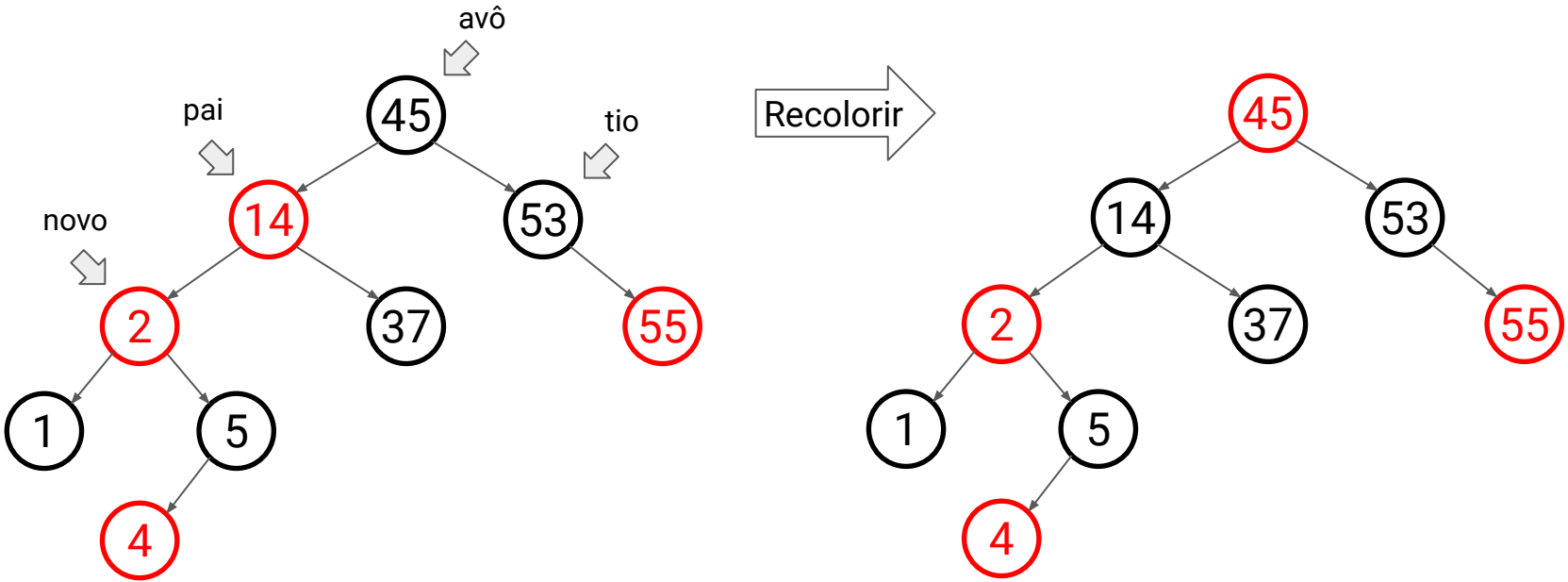
Balanceamento da árvore rubro-negra

- Caso 4: nó pai e filho à esquerda vermelho com tio preto (cont.)
  - Ocorre após rotação para resolver o caso 3
    - Este caso também viola a regra 4 (nó vermelho possui ambos filhos pretos)
    - Para corrigir, deve ser
      - Trocar a cor do pai para preto
      - Trocar a cor do avô para vermelho
      - Realizar uma rotação à direita no avô
  - Exemplo
    - Resolver caso 3
    - Ir para o nível anterior
    - Recolorir 14 e 45
    - Rotação à direita em 45



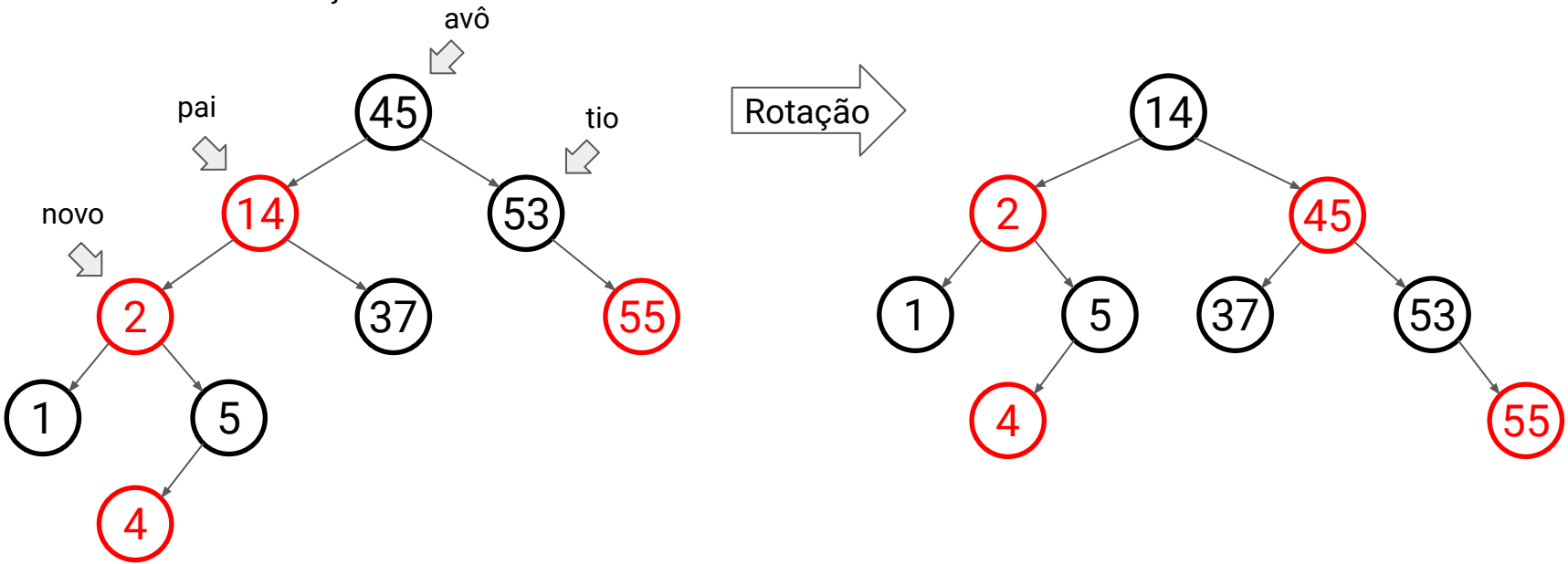
Balanceamento da árvore rubro-negra

- Caso 4: nó pai e filho à esquerda vermelho com tio preto (cont.)
  - Exemplo
    - Recolorir 14 e 45



Balanceamento da árvore rubro-negra

- Caso 4: nó pai e filho à esquerda vermelho com tio preto (cont.)
  - Exemplo
    - Rotação à direita em 45



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 != arvore->nulo)
        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 == arvore->nulo)
        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;
}
```

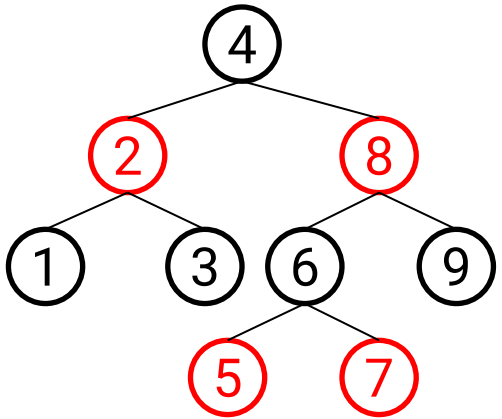
Balanceamento da árvore rubro-negra

- Rotação à esquerda

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

    if (esquerda->direita != arvore->nulo)
        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 == arvore->nulo)
        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;
}
```

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 AVL e valide removendo o nó 6 do exercício anterior.

Árvores rubro-negra

Estruturas de dados II
Prof. Allan Rodrigo Leite