CI1057: Algoritmos e Estruturas de Dados III
Árvores 2-3-4

Profa. Carmem Hara

Departamento de Informática/UFPR

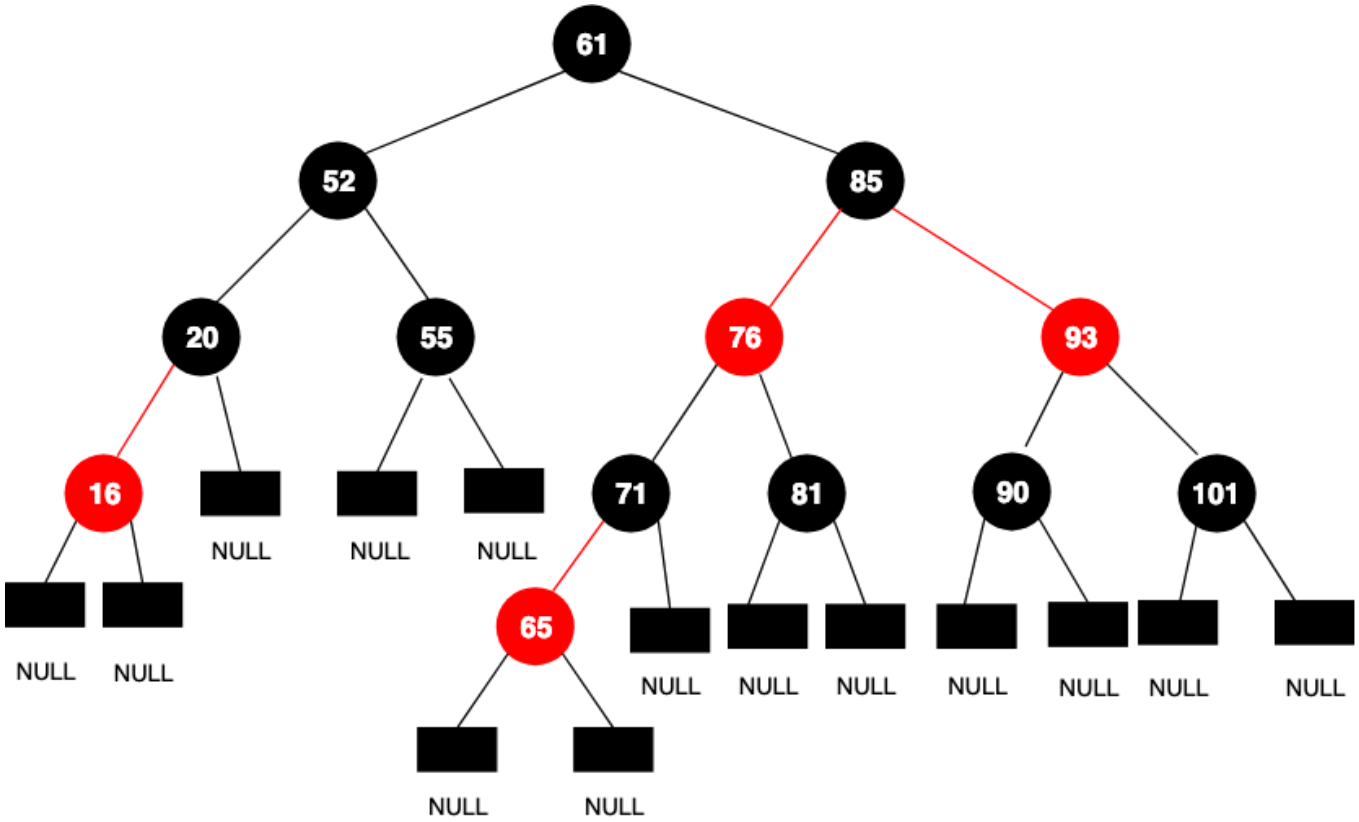
5 de abril de 2024

Árvore Rubro-Negra

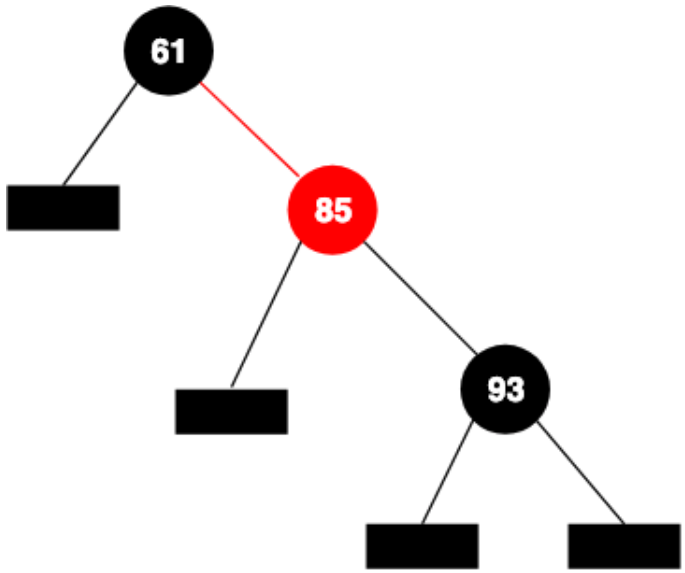
É uma árvore binária de busca na qual:

- nodos são vermelhos ou pretos
- a raiz é preta
- todos os nodos externos são pretos
- se um nodo é vermelho, seus dois filhos são pretos
  (não podem existir 2 nodos vermelhos consecutivos em um caminho)
- todo caminho de um nodo até um nodo externo contém o mesmo número de nodos pretos
  (caminho preto de mesmo tamanho)

Exemplo de Árvore Rubro-Negra



Exemplo de Árvore que NÃO É Rubro-Negra



Árvore Rubro-Negra - Estrutura de Dados

```
1  typedef int ItemArv;
2
3  enum tipoCor {RED, BLACK};
4
5  typedef struct nodo *ApNodo;
6  typedef struct nodo{
7    ApNodo esq, dir;
8    ApNodo pai;
9    ItemArv item;
10   tipoCor cor;
11 } Nodo;
12
13 typedef ApNodo ArvRB;
14
15 static ApNodo NodoNull;
```

- a cor do `NodoNull` é BLACK
- o pai da raiz de uma árvore não vazia é o `nodoNull`
- todos os nodos externos são representados pelo mesmo `nodoNull`

Árvore Rubro-Negra - Interface

```
1 void criaArvRB( ArvRB* );
2 void freeArvRB( ArvRB );
3 void escreveArvRB( ArvRB );
4 ArvRB insereArvRB( ItemArv, ArvRB );
5 ArvRB buscaArvRB( ItemArv, ArvRB );
6 int removeArvRB( ItemArv, ArvRB* );
```

Árvore Rubro-Negra - Busca

- Idêntica a busca na árvore binária de busca

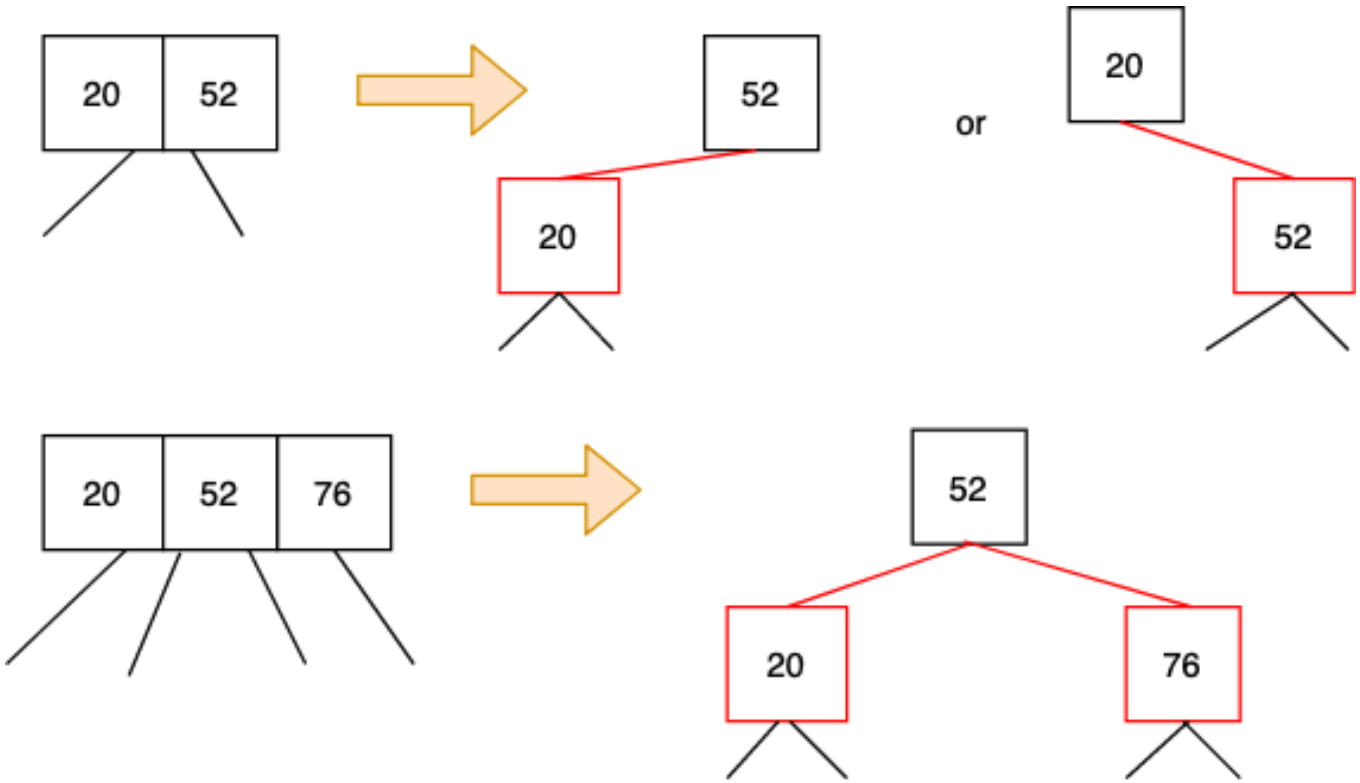
```
1  ArvRB buscaArvRB( ItemArv v, ArvRB p ){
2    int i;
3
4    if( p == NodoNull )
5      return NULL;
6    if( v == p->item   )
7      return p;
8    if( lt( v, p->item ))
9      return buscaArvRB( v, p->esq );
10   return buscaArvRB( v, p->dir );
11 }
```

Árvore Rubro-Negra - Inserção

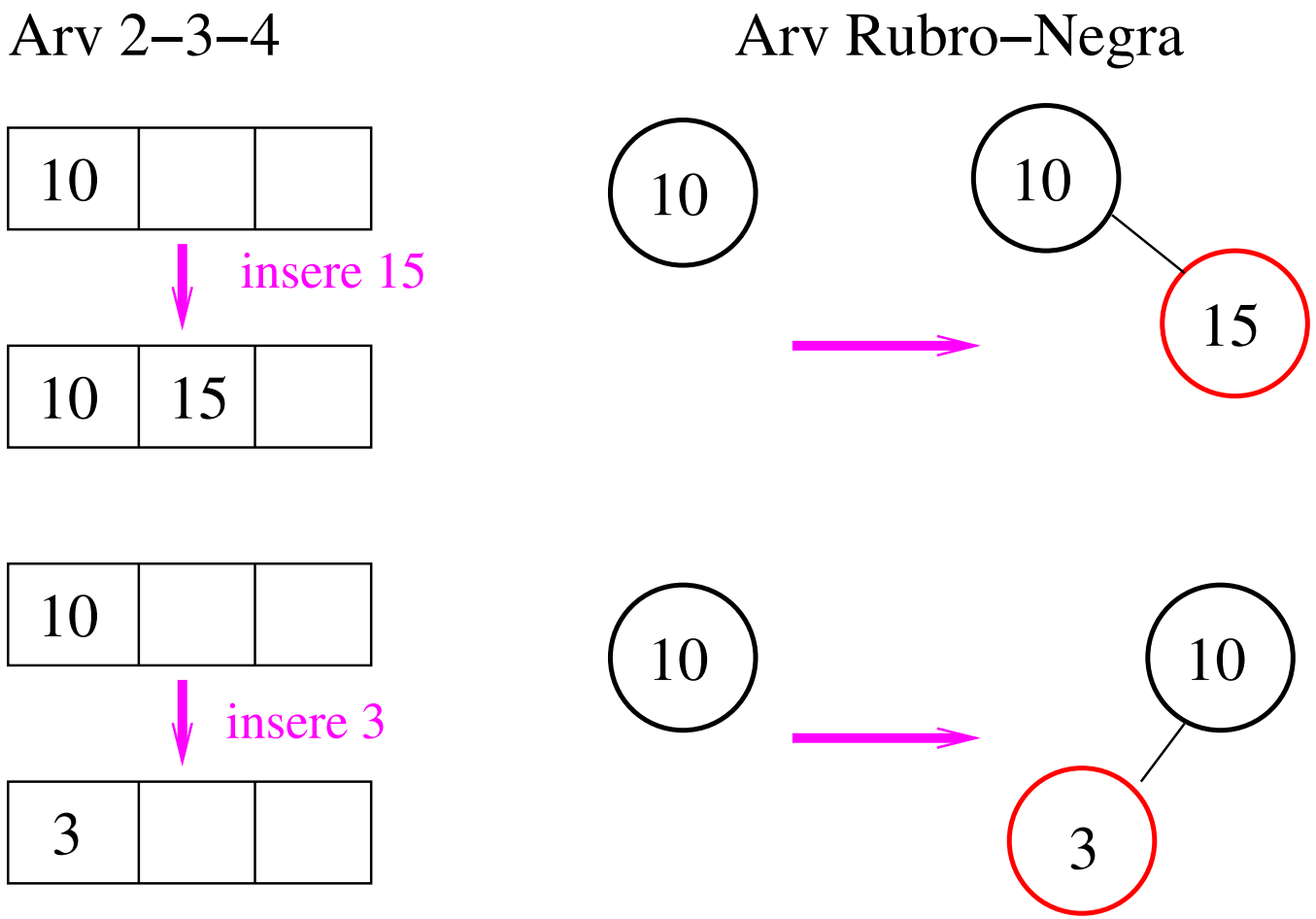
1. executa a inserção como em uma árvore binária de busca
2. colore o nodo de vermelho
3. se a inserção violar alguma propriedade, arruma a árvore com:
   - troca de cores dos nodos
   - rotações

Árvore Rubro-Negra - Inserção

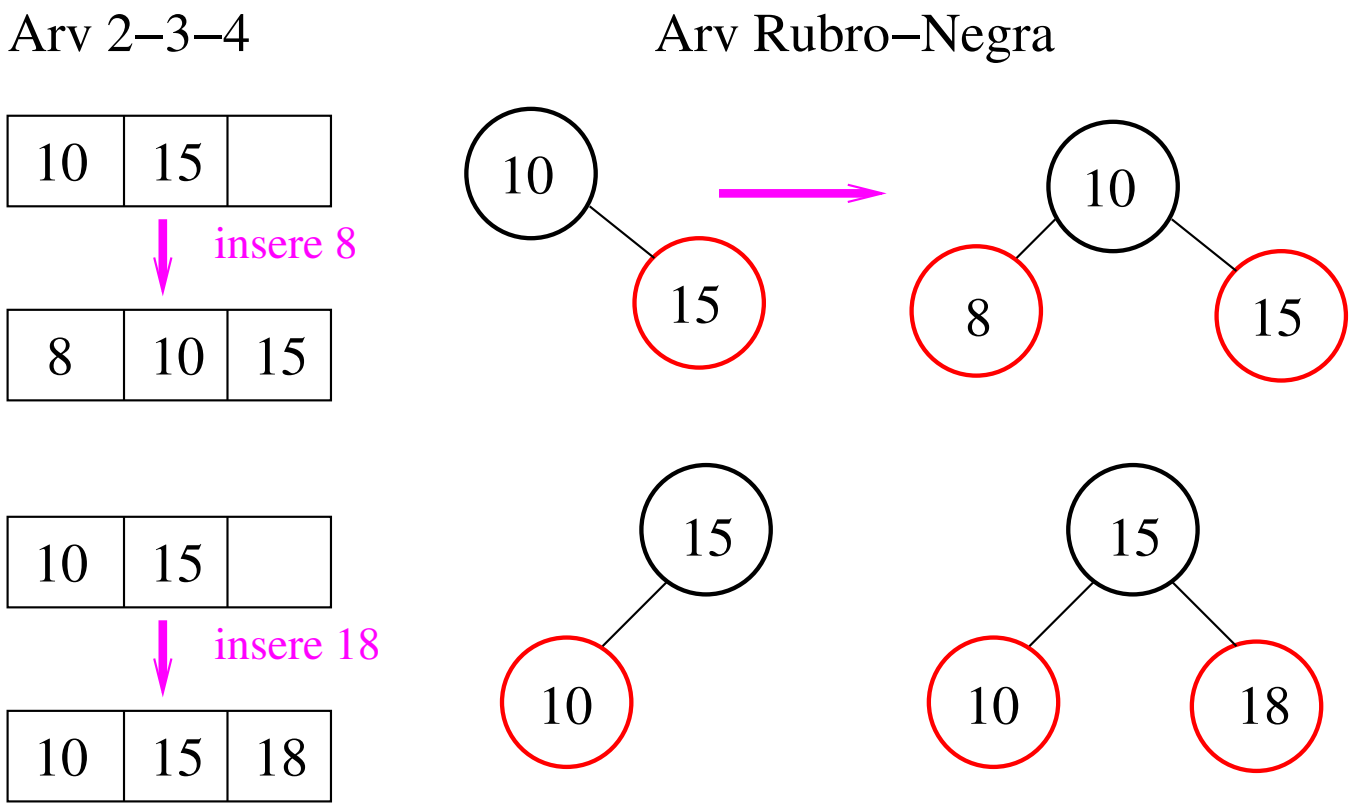
Correspondência entre Árvore 2-3-4 e Árvore Rubro-Negra



Árvore Rubro-Negra - Inserção em Nodo-2

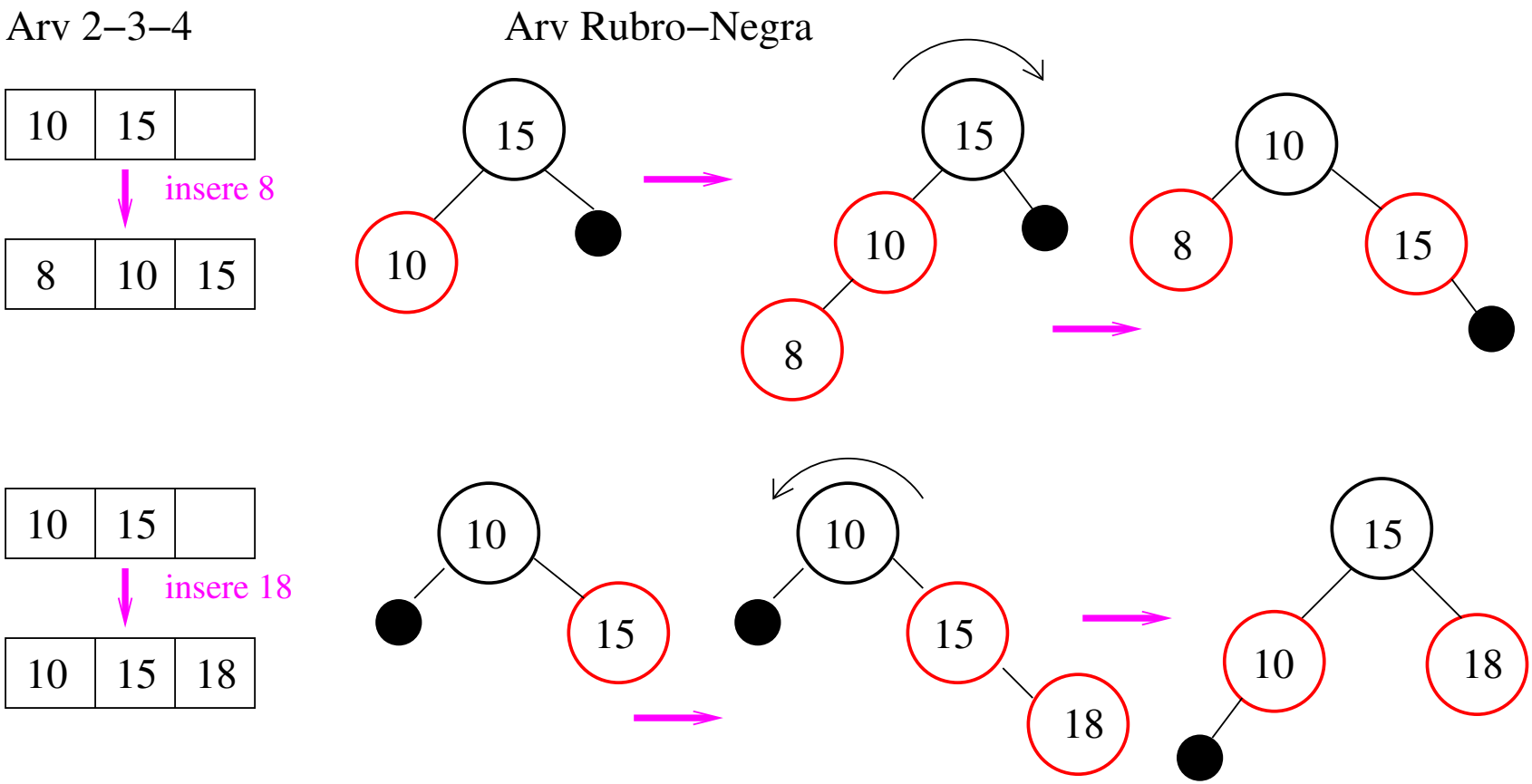


Árvore Rubro-Negra - Inserção em Nodo-3 - Simples



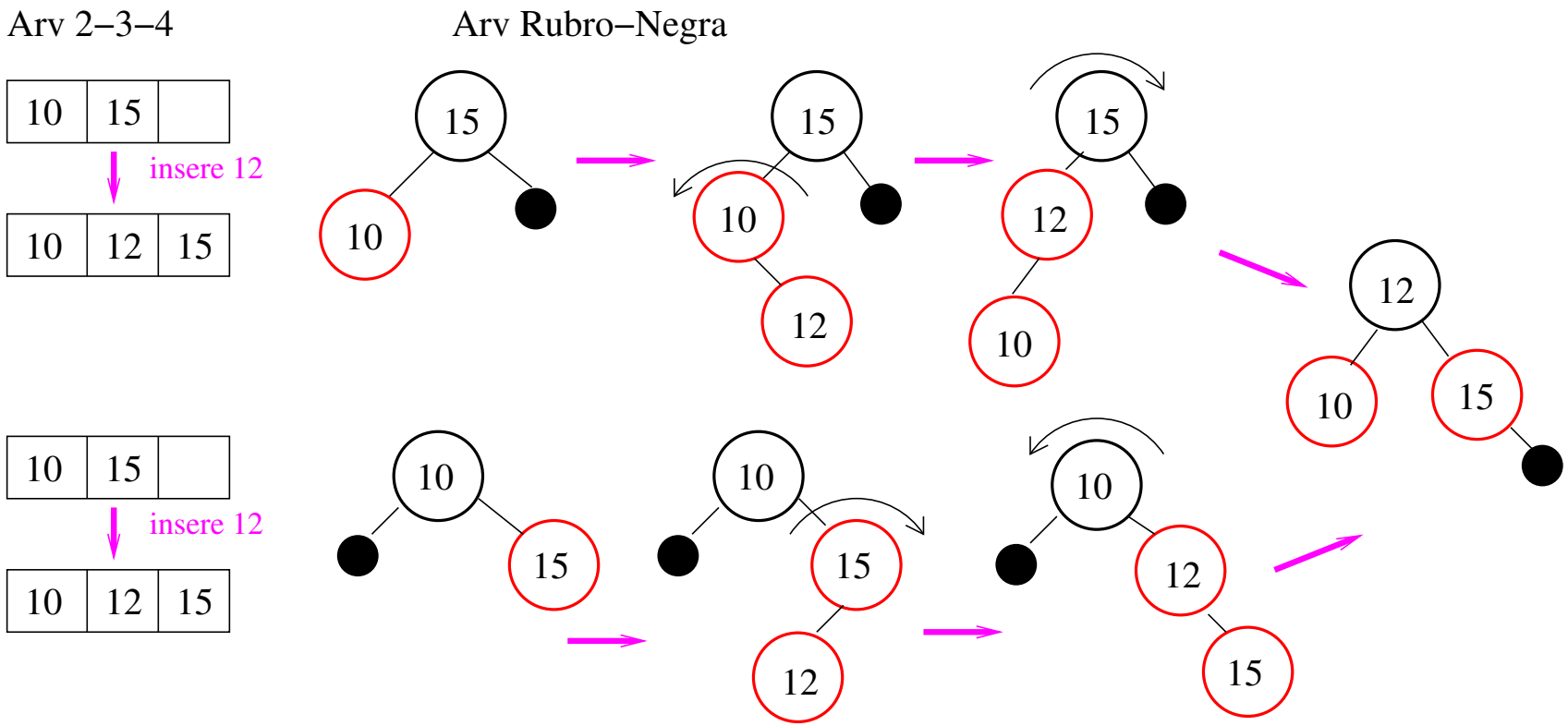
Árvore Rubro-Negra - Inserção em Nodo-3 - Direito-Direito ou Esquerdo-Esquerdo

Caso 3: Transformação em Nodo-4



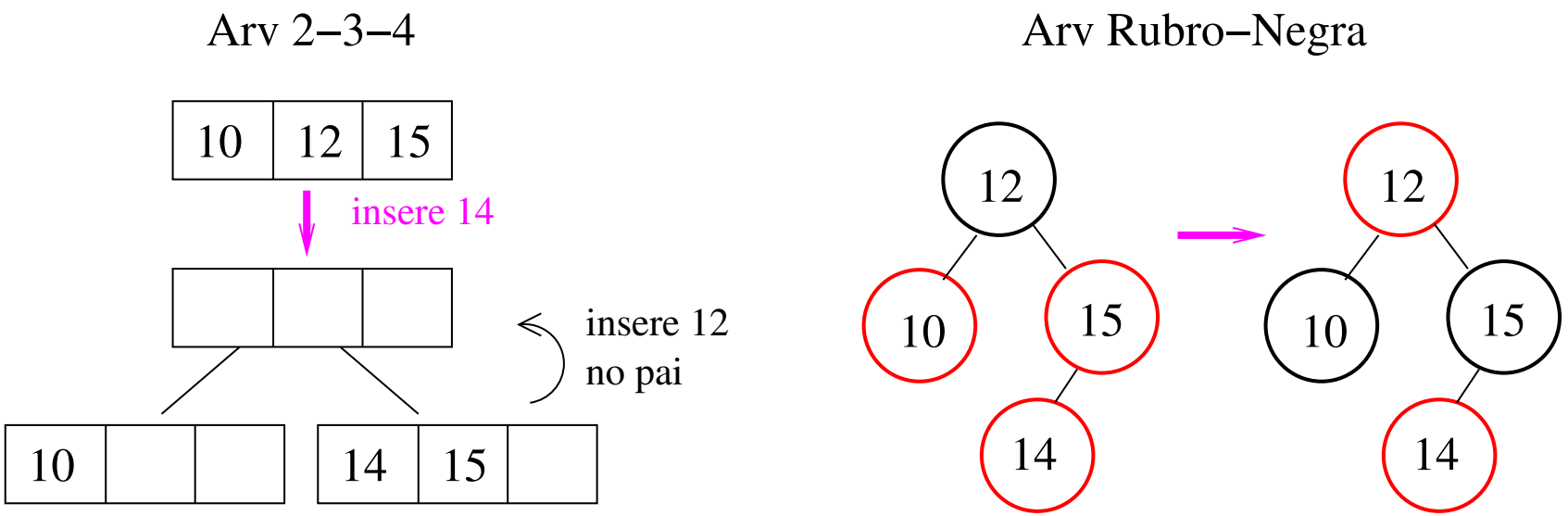
Árvore Rubro-Negra - Inserção em Nodo-3 - Direito-Esquerdo ou Esquerdo-Direito

Caso 2: Transformação em Nodo-4 após alteração da representação do Nodo-3



Árvore Rubro-Negra - Inserção em Nodo-4

Caso 1: Split



Árvore Rubro-Negra - Inserção Exemplo

Construa a árvore rubro-negra com os seguintes valores:
50 30 20 10 25 60 55 22 52 51 48

Árvore Rubro-Negra - Inserção - Implementação

```
1  ArvRB insereArvRB( ItemArv v, ArvRB raiz ){
2    ArvRB novoNodo;
3
4    if( raiz == NodoNull )
5      return criaNodo( v, NodoNull, NodoNull, NodoNull, BLACK );
6    if( eq( v, raiz->item ))
7      return raiz;
8    if( lt( v, raiz->item ))
9      raiz->esq = insereR( v, raiz->esq, raiz, &novoNodo );
10   else
11     raiz->dir = insereR( v, raiz->dir, raiz, &novoNodo );
12   arrumaRBInsercao( &raiz, novoNodo );
13   return raiz;
14 }
```

Árvore Rubro-Negra - Inserção - Balanceamento

```
1  void arrumaRBInsercao( ArvRB *raiz, ArvRB p ){
2    ArvRB pai, avo, tio;
3
4    pai= p->pai;    avo= pai->pai;
5    while( pai->cor == RED && p != *raiz){
6      if( pai == avo->esq ){                    /* insercao na subarv esq */
7        tio = avo->dir;
8        if( tio->cor == RED ){                  /* Caso 1 */
9          pai->cor = BLACK;  tio->cor = BLACK; avo->cor = RED;
10         p = avo;
11       } else {
12         if( p == pai->dir ){                  /* Caso 2: esq-dir */
13           p = p->pai;
14           rotacaoEsq(p, raiz);
15           pai= p->pai;
16         }
17         pai->cor = BLACK; avo->cor = RED; /* Caso 3  esq-esq*/
18         rotacaoDir( avo, raiz );
19       }
20     }
21     else { }                                 /* insercao na subarv. direita */
22     (*raiz)->cor = BLACK;
23 }
```

Altura de uma Árvore Rubro-Negra

A árvore rubro-negra é uma árvore balanceada.
Sua altura é no máximo $2 * lg(n+1)$.

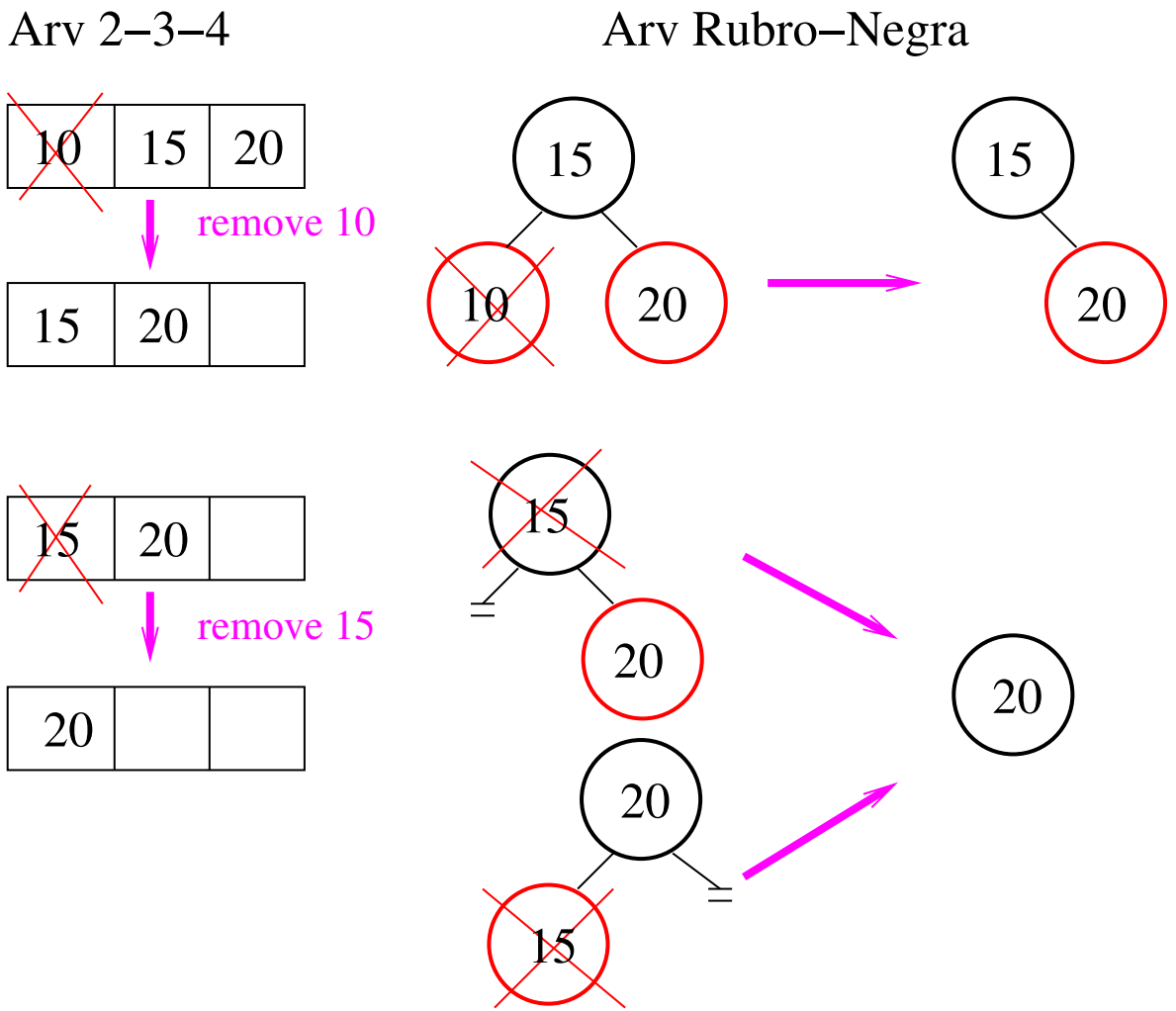
Já vimos que a altura de uma árvore binária completa com n nodos internos é $lg(n+1)$. Considere uma árvore rubro-negra com apenas nodos pretos. Pelas propriedades da árvore, ela corresponde a uma árvore binária completa para manter a igualdade das alturas pretas (bh). Ou seja, $n = 2^{bh} - 1$. A adição de nodos vermelhos pode dobrar a altura da árvore (h). Ou seja, $bh \geq h/2$.
Assim, $n \geq 2^{h/2} - 1$. Aplicando lg em ambos os lados, obtemos $h \leq 2 * lg(n+1)$.

Árvore Rubro-Negra - Remoção

1. caso o nodo (*nodoK*) for folha ou tiver apenas um filho, remove o próprio nodo *nodoK*
2. caso contrário, remove o nodo contendo o valor sucessor ou predecessor (*nodoRem* (que tem no máximo um filho)
3. se o nodo removido for preto, balancear a árvore
4. se o *nodoRem* for diferente de *nodoK*, alterar o valor do *nodoK* para conter o valor do *nodoRem*

Árvore Rubro-Negra - Remoção - Casos Simples

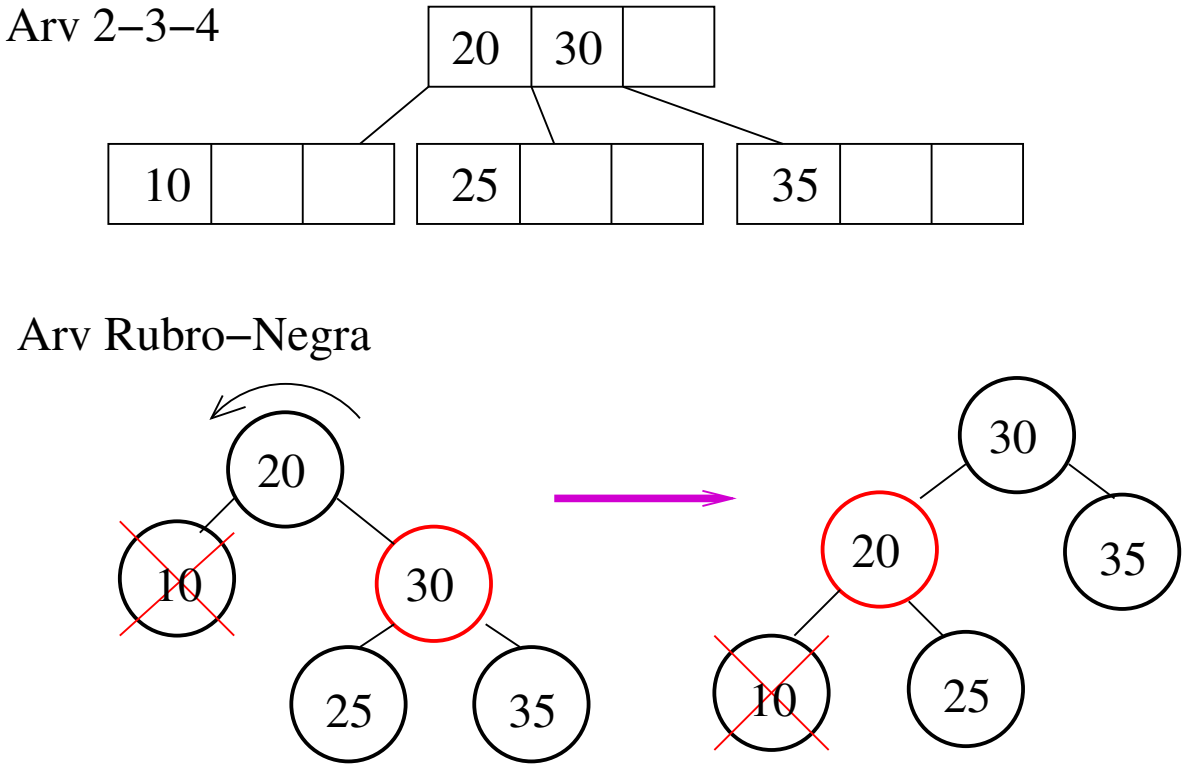
Casos simples: remoção de Nodo-4 ou Nodo-3



Árvore Rubro-Negra - Remoção de Nodo-2 - Balanceamento

Caso 1: Mudança de representação do Nodo-3

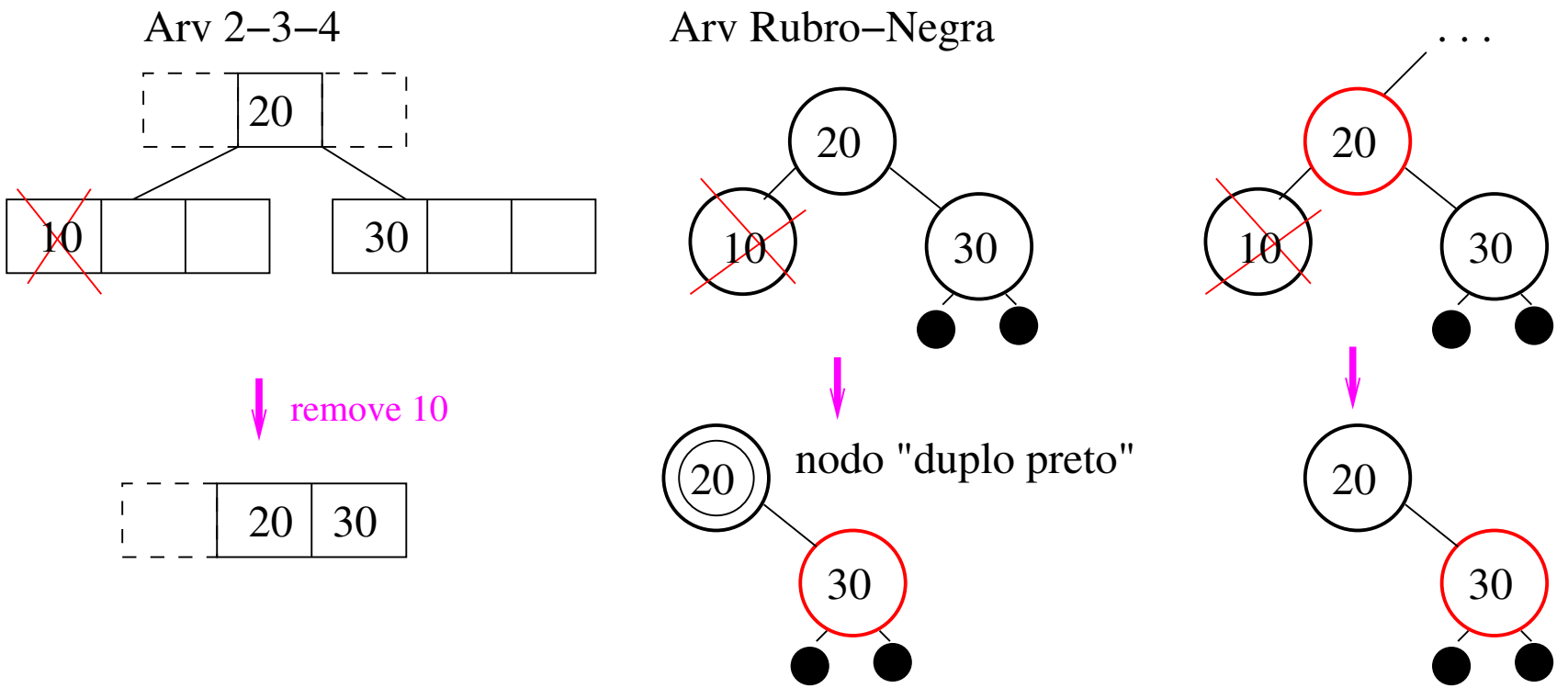
Objetivo: na árvore rubro-negra o irmão do nodo a ser removido é raiz da subárvore que representa o nodo na árvore 2-3-4



Árvore Rubro-Negra - Remoção de Nodo-2 - Balanceamento

Caso 2: Irmão é Nodo-2

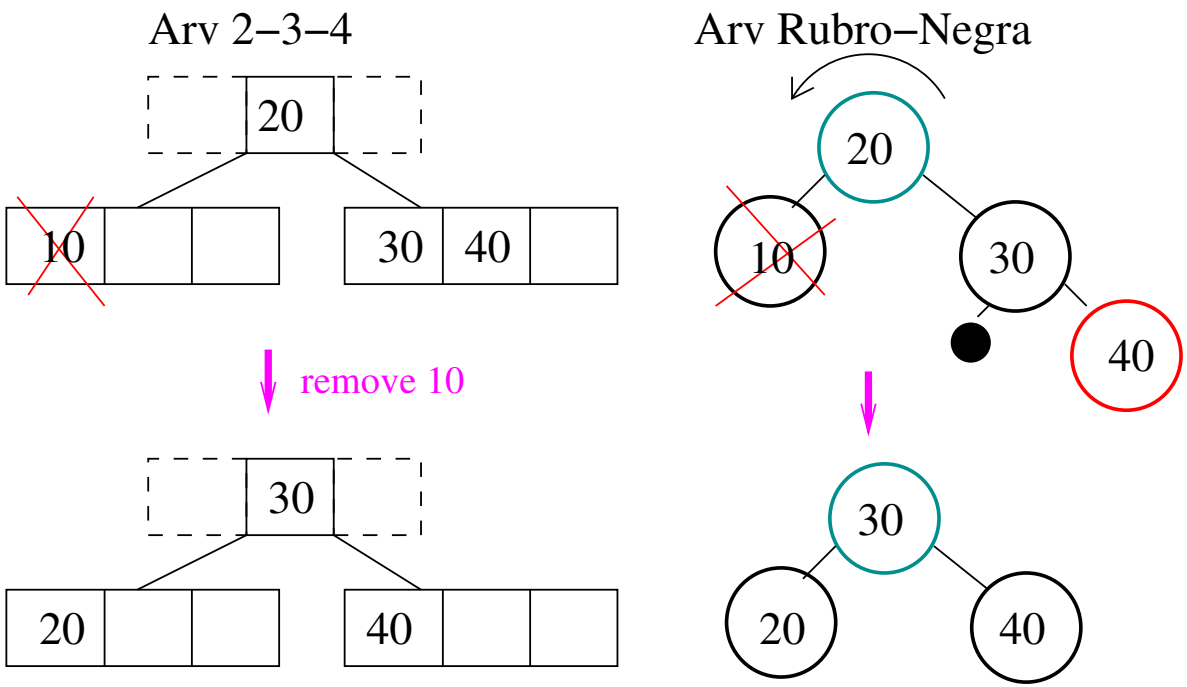
Faz o merge do irmão com o pai



Árvore Rubro-Negra - Remoção de Nodo-2 - Balanceamento

Caso 4: Irmão é Nodo-3 ou Nodo-4

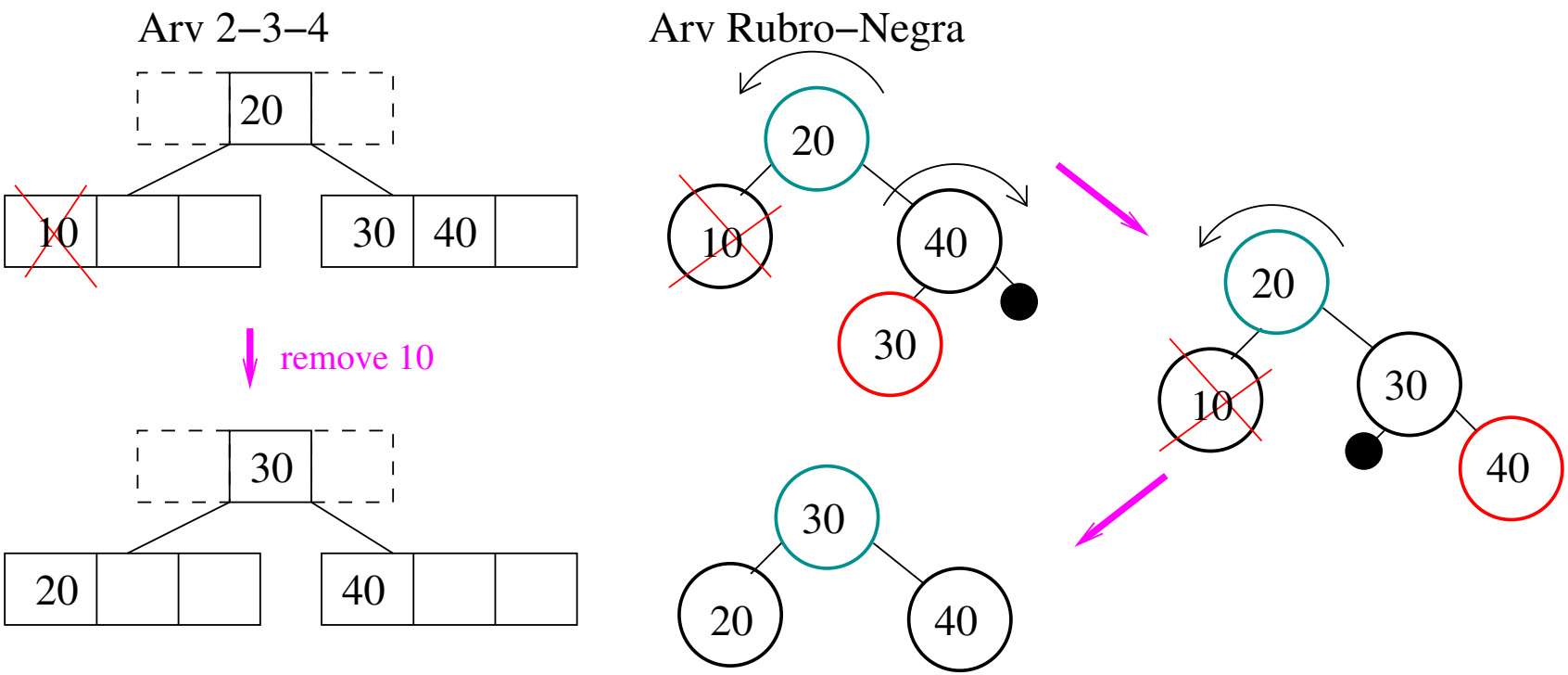
Empréstimo da chave do irmão - Casos direito-direito ou esquerdo-esquerdo



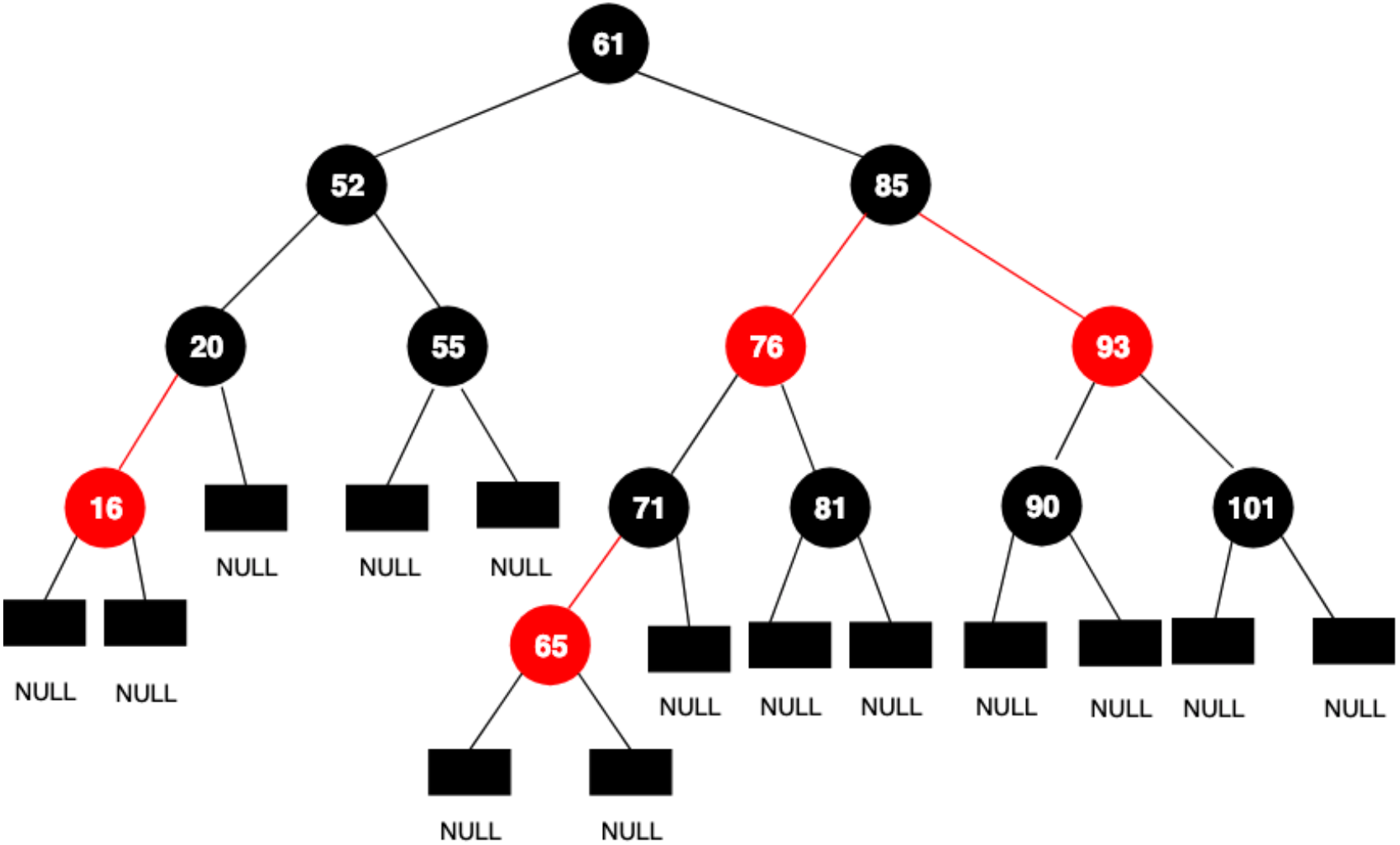
Árvore Rubro-Negra - Remoção de Nodo-2 - Balanceamento

Caso 3: Irmão é Nodo-3

Empréstimo da chave do irmão - Casos direito-esquerdo ou esquerdo-direito



Árvore Rubro-Negra - Remoção - Exemplo



Árvore Rubro-Negra - Remoção - Implementação

```
1  int removeArvRB( ItemArv v, ArvRB *raiz ){
2    ArvRB nodoV, nodoRem, pai, filho;
3
4    nodoV= buscaArvRB( v, *raiz );
5    if( nodoV == NULL )
6      return 0;
7    if( nodoV->esq == NodoNull || nodoV->dir == NodoNull )
8      nodoRem = nodoV;                  /* se nodoV tem no maximo 1 filho remove
           nodoV */
9    else                                /* senao remove o sucessor */
10     nodoRem = sucessor( nodoV );
11   if( nodoRem->esq != NodoNull )
12     filho= nodoRem->esq;
13   else filho= nodoRem->dir;
14   filho->pai= nodoRem->pai;
15   if( nodoRem->pai == NodoNull )
16     *raiz= filho;
17   else
18     trocaFilho( nodoRem->pai, nodoRem, filho );
19   if( nodoV != nodoRem )
20     nodoV->item= nodoRem->item;
21   if( nodoRem->cor == BLACK )
22     arrumaRBRemocao( raiz, filho );
23   free( nodoRem );
24   return 1;
25 }
```

Árvore Rubro-Negra - Remoção - Balanceamento

```
1  void arrumaRBRemocao( ArvRB *raiz,  ArvRB p ){
2    ArvRB pai, irmao;
3
4    while( p != *raiz && p->cor == BLACK ){
5      pai= p->pai;
6      if( p == pai->esq ){      /* duplo BLACK a esquerda */
7        irmao= pai->dir;
8        if( irmao->cor == RED ){          /* Caso 1 */
9          irmao->cor= BLACK; pai->cor= RED;
10         rotacaoEsq (pai, raiz );
11         irmao= p->pai->dir;
12       }
13       if( irmao->esq->cor == BLACK && irmao->dir->cor == BLACK ){
14         irmao->cor= RED;                  /* Caso 2 */
15         p= p->pai;
16       }
17       else{
18         if( irmao->esq->cor == RED ){  /* Caso 3 - direito-esquerdo */
19           irmao->cor= RED; irmao->esq->cor= BLACK;
20           rotacaoDir( irmao, raiz );
21           irmao= p->pai->dir;
22         }
23         irmao->cor= p->pai->cor; p->pai->cor= BLACK; irmao->dir->cor= BLACK;
24         rotacaoEsq( p->pai, raiz );
25         p= *raiz;
26       }
27     } else {}                  /* duplo BLACK a direita -- similar */
28   p->cor= BLACK;   return;
29 }
```

Referências

- Cormen & Leiserson & Rivest (Cap. 14)
- Sedgewick (Seção 13.4)
- `https://algorithmtutor.com/Data-Structures/Tree/Red-Bla`