

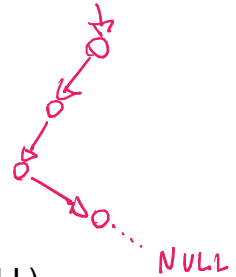
Árvores rubro-negras

Árvores rubro-negras são árvores binárias de busca balanceadas,

- que usam rotações mas não usam fator de balanceamento.

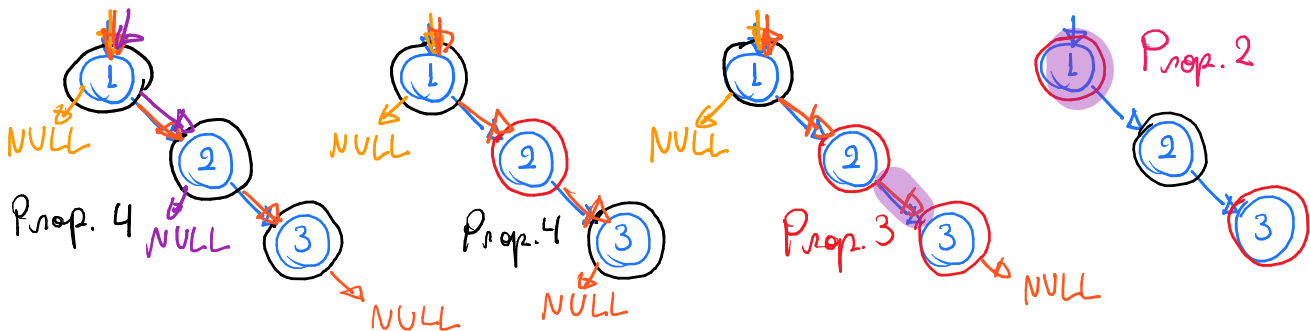
Definição:

- Cada nó é vermelho ou preto.
- Raiz é sempre preta.
- Dois nós vermelhos não podem ser adjacentes,
 - ou seja, um nó vermelho só pode ter filhos pretos.
- Todo caminho da raiz até um apontador NULL (caminho raiz-NULL)
 - tem o mesmo número de nós pretos.
 - Pense nesses caminhos como buscas mal sucedidas.

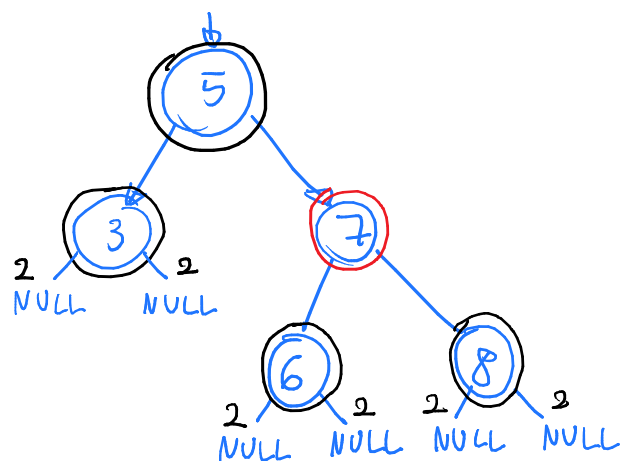
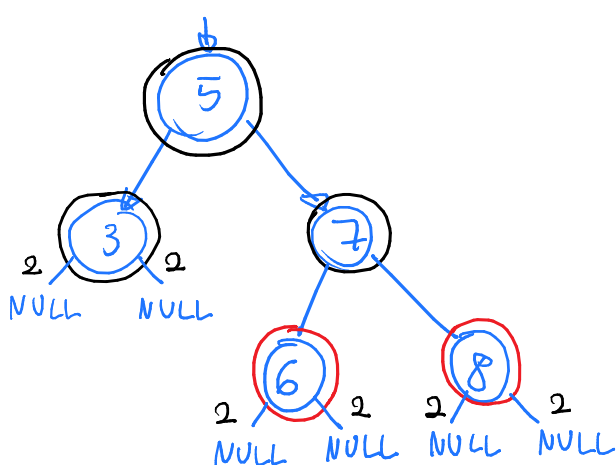


Para ganhar intuição de que essas propriedades levam a uma árvore balanceada

- note que uma lista com três nós já não pode ser uma árvore rubro-negra.



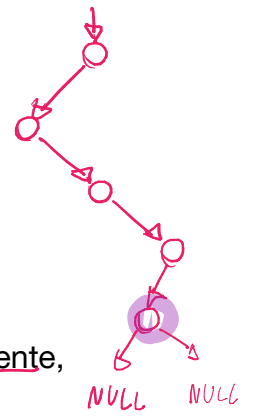
Exemplos de árvores rubro-negras:



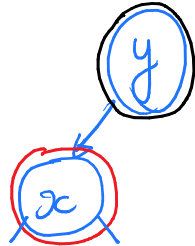
Inserção em árvores rubro-negras

A ideia geral é inserir o novo nó como uma folha vermelha,

- após percorrer um caminho descendente na árvore,
- e usar recolorações e rotações para restabelecer as propriedades,
 - ao percorrer o caminho no sentido ascendente.
- A seguir vamos analisar alguns casos que podem ocorrer,
 - na volta da recursão, quando percorremos o caminho ascendente,
 - sempre considerando que o nó corrente é x

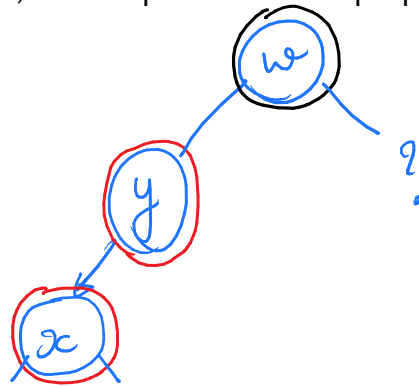


Caso 1: se y , o pai de x , não for vermelho,



- as propriedades da definição estão mantidas e nada precisa ser feito.

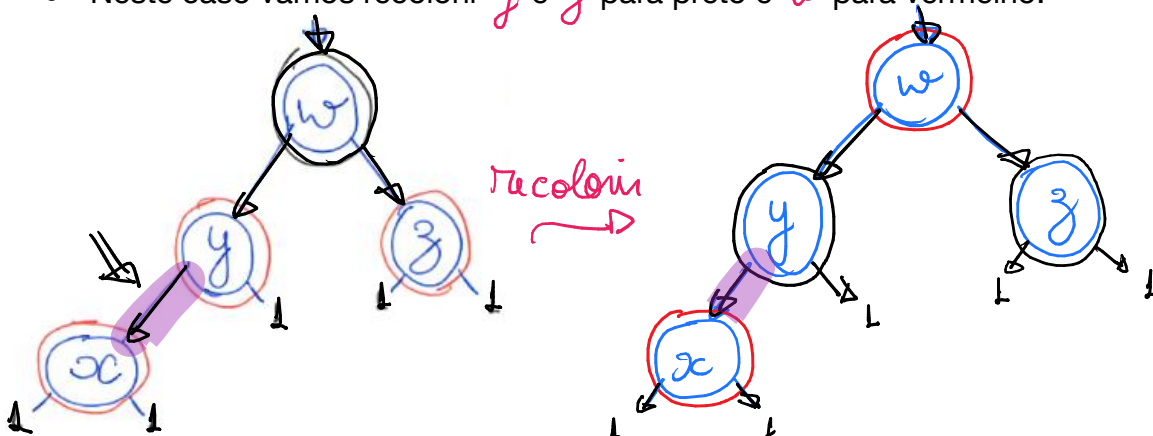
Caso 2: se y é vermelho, temos que restaurar a propriedade 3.



- Sabemos que y não é a raiz, por conta da propriedade 2,
 - e que w , o pai de y , é preto, por conta da propriedade 3.

Caso 2.1: w tem outro filho z que tem cor vermelha.

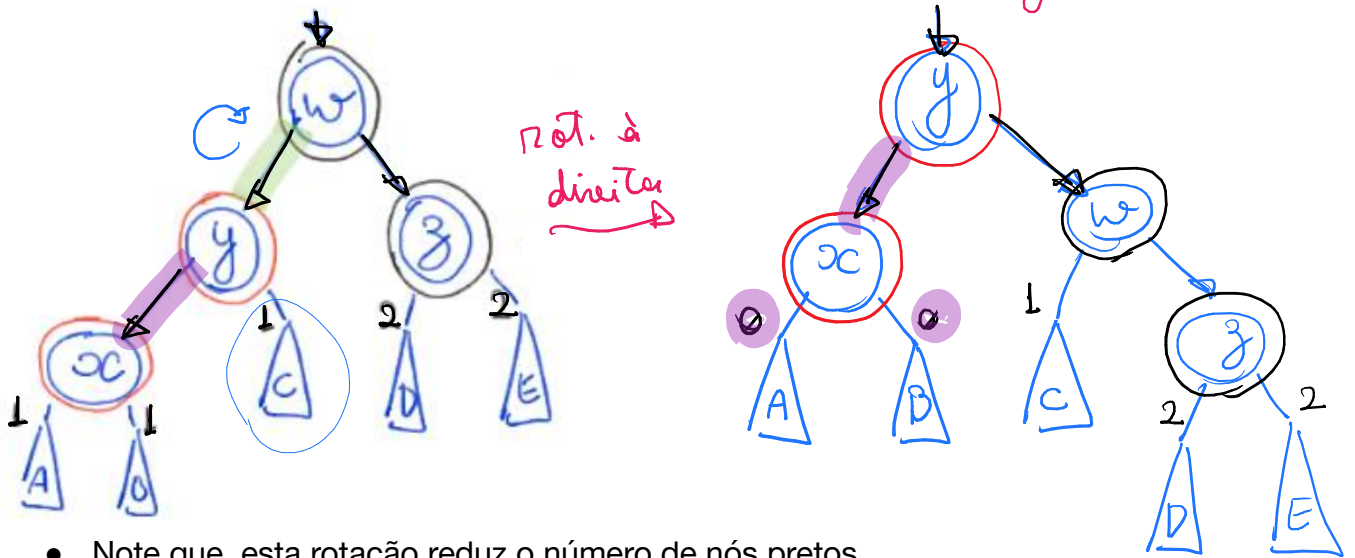
- Neste caso vamos recolorir y e z para preto e w para vermelho.



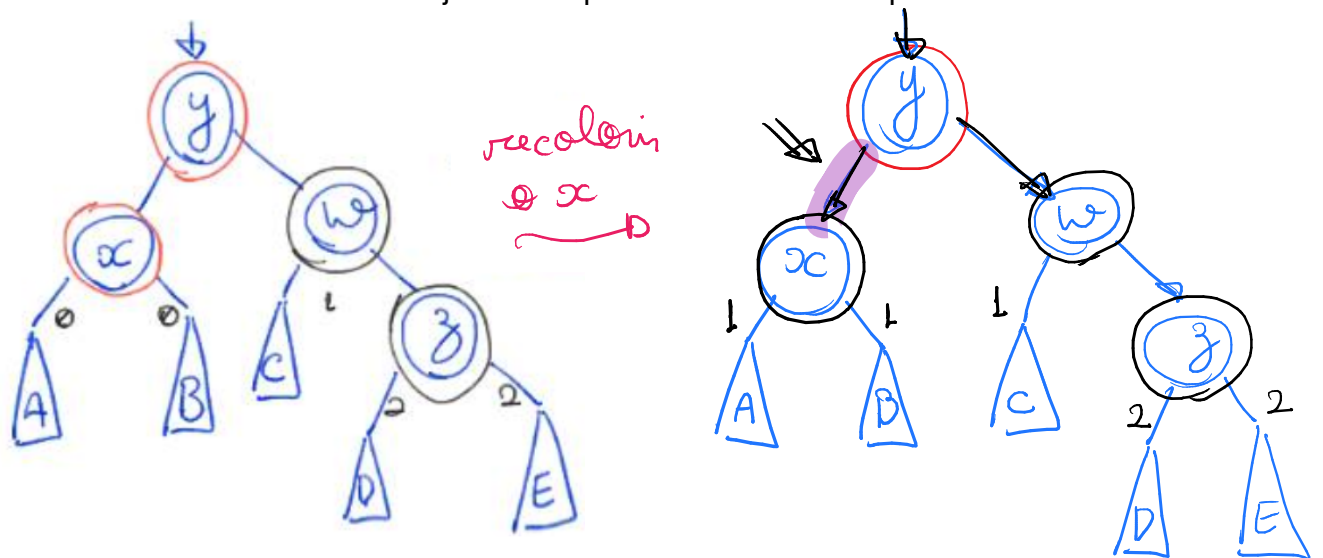
- Note que preservamos a propriedade 4, pois
 - todo caminho que passava por w passa por y ou por z
 - e restauramos a propriedade 3 entre x e y
- Ao continuar a subida na árvore, será necessário analisar a situação de w
 - que ficou vermelho, pois ele pode violar a propriedade 3.
- Se w for a raiz da árvore, basta mudar a cor dele para preto.
 - **Quiz1:** Por que isso não viola as propriedades?

Caso 2.2: w não tem outro filho vermelho.

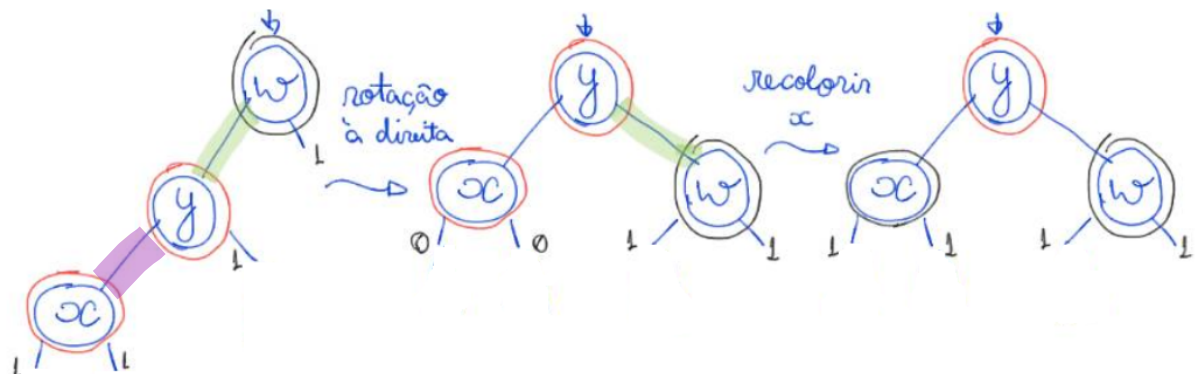
- Neste caso, w pode ter outro filho preto ou não ter outro filho.
 - Primeiro, vamos considerar a presença de um nó preto z
- Fazemos uma rotação à direita que inverte a relação entre w e y z



- Note que, esta rotação reduz o número de nós pretos
 - nos caminhos que vão da raiz até os filhos de x
- Então mudamos a cor de x para preto, o que resolve tanto o problema
 - dos vermelhos adjacentes quanto do número de pretos nos caminhos.



- Ao continuar a subida na árvore, será necessário analisar a situação de y
 - que é vermelho e se tornou a raiz da subárvore,
 - pois ele pode violar a propriedade 3 com relação a seu pai.
 - Se y for a nova raiz da árvore, basta mudar a cor dele para preto.
- Note que, se w não tiver outro filho, a mesma solução funciona.



Quiz2: Como resolver o caso em que x é ~~inserido como~~ filho direito de y ?

Inserção em árvores rubro-negras **esquerdistas**

Embora nossos tratamentos dos casos anteriores estejam corretos,

- uma vez que eles restauram as propriedades da árvore rubro-negra,
 - sua implementação é um tanto complicada,
 - por ser necessário manipular antecessores do nó corrente.
- Por isso, vamos estudar a implementação da inserção
 - de um tipo especial de árvore rubro-negra,
- que é conhecida como **árvore rubro-negra esquerdista**,
 - pois os nós vermelhos sempre são filhos esquerdos.

A seguir o código para a estrutura de um nó de árvore rubro-negra:

```
- typedef int Item;
- typedef int Chave;

typedef struct noh {
    int vermelho;
    Chave chave; -
    Item conteudo; -
    struct noh *pai; -
    struct noh *esq; -
    struct noh *dir; -
} Noh;

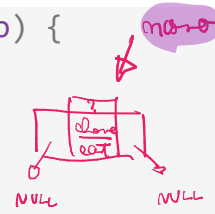
typedef Noh *Arvore;
```

A seguir apresentamos a implementação recursiva,

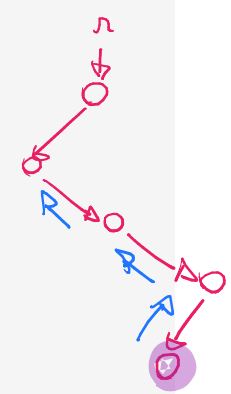
- introduzida por **Sedgewick**, de uma árvore rubro-negra esquerdista.

```
Noh *novaNoh(Chave chave, Item conteudo) {
    Noh *novo;
    novo = (Noh *)malloc(sizeof(Noh)); -
    → novo->vermelho = 1;
    novo->chave = chave; ↵
    novo->conteudo = conteudo; ↵
    novo->esq = NULL; ↵
    novo->dir = NULL; ↵
    return novo;
}
```

```
Arvore inserir(Arvore r, Chave chave, Item conteudo) {
    Noh *novo = novaNoh(chave, conteudo);
    return insereRN(r, novo);
}
```

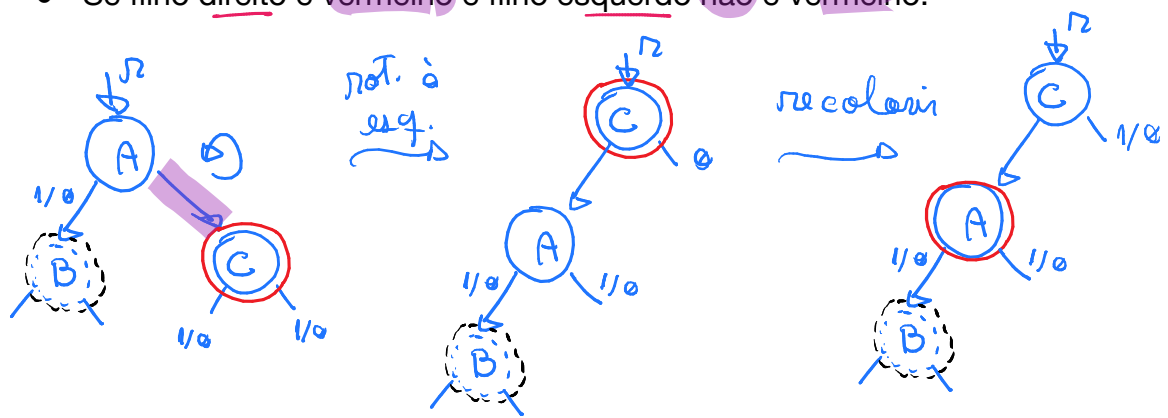


```
Arvore insereRN(Noh *r, Noh *novo) {
    => if (r == NULL) { // subárvore era vazia
        novo->pai = NULL;
        return novo;
    }
    if (novo->chave <= r->chave) { // desce à esquerda
        => r->esq = insereRN(r->esq, novo);
        r->esq->pai = r;
    }
    else { // desce à direita
        r->dir = insereRN(r->dir, novo);
        => r->dir->pai = r;
    }
}
```



Estamos na volta da recursão (caminho ascendente) e vamos tratar alguns casos:

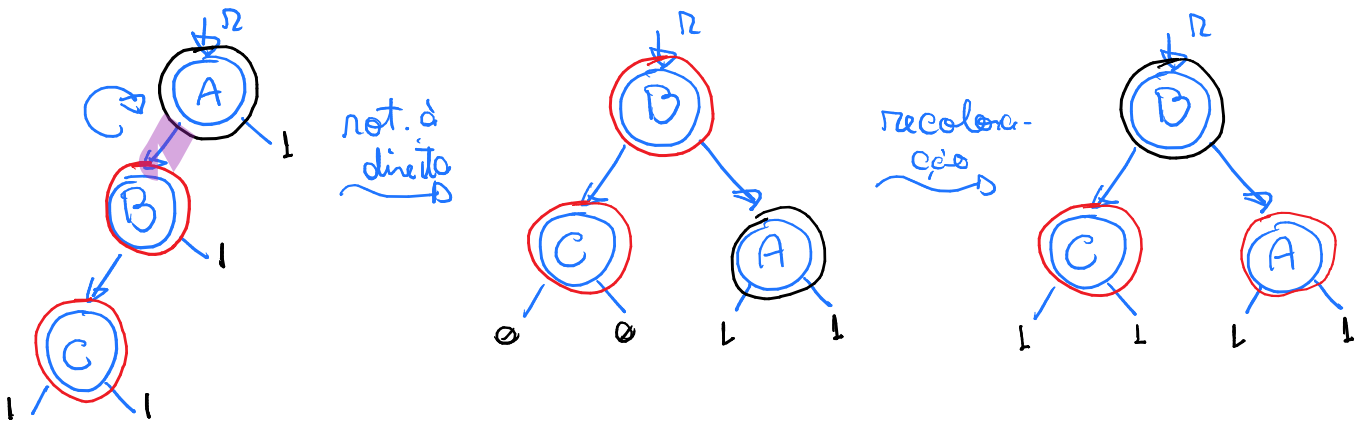
- Se filho direito é vermelho e filho esquerdo não é vermelho.



- Note que A pode ser preto ou vermelho,
 - e que sua cor é herdada por C.

```
if (r->dir != NULL && r->dir->vermelho == 1 && (r->esq == NULL ||
r->esq->vermelho == 0)) {
    => r = rotacaoEsq(r);
    r->vermelho = r->esq->vermelho;
    r->esq->vermelho = 1;
}
```

- Se filho esquerdo é vermelho e filho esquerdo do filho esquerdo é vermelho.

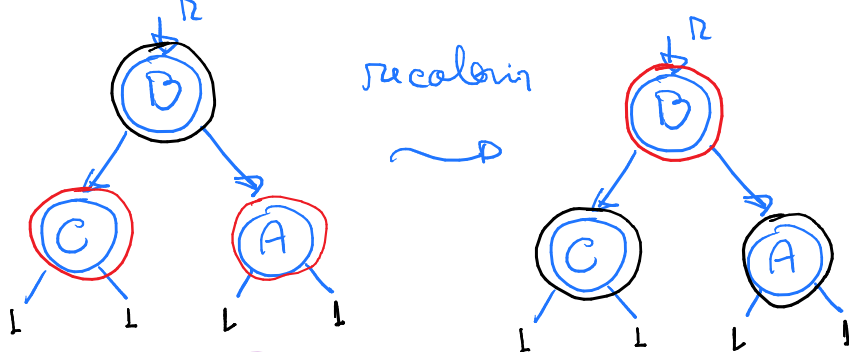


```

if (r->esq != NULL && r->esq->vermelho == 1 && r->esq->esq !=
NULL && r->esq->esq->vermelho == 1) {
    // se filho e neto esquerdos forem vermelhos faz rotação a
    // direita e recolor de acordo
    r = rotacaoDir(r);
    r->vermelho = 0;
    r->dir->vermelho = 1;
}

```

- Se filho esquerdo é vermelho e filho direito é vermelho.



```

if (r->esq != NULL && r->esq->vermelho == 1 && r->dir != NULL &&
r->dir->vermelho == 1) {
    // se os dois filhos são vermelhos, troque a cor com o pai
    // preto
    r->esq->vermelho = 0;
    r->dir->vermelho = 0;
    r->vermelho = 1;
}
return r;
}

```

É interessante analisar como as operações anteriores

- garantem tanto as propriedades tradicionais das árvores rubro-negras,
 - quanto a propriedade adicional da árvore rubro-negra esquerdista,
 - i.e., os nós vermelhos sempre são filhos esquerdos.