AED2 - Aulas 06 e 07
Árvores AVL e rubro-negras

Árvores AVL

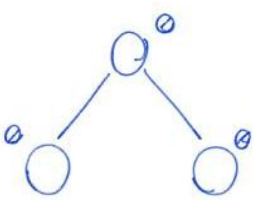
AVL vem dos nomes dos seus inventores: Adelson-Velsky and Landis.

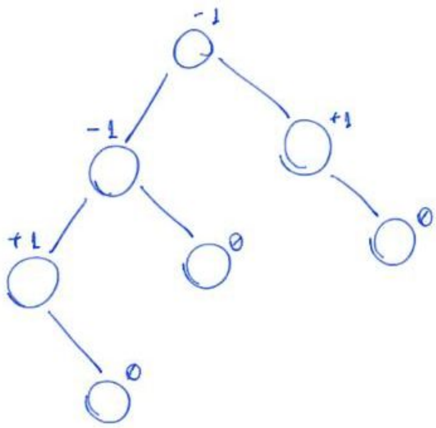
Definições:

- a altura de uma subárvore é o comprimento do caminho mais longo da raíz até alguma folha
- o fator de balanceamento de um nó é a diferença entre a altura de sua subárvore direita menos a altura de sua subárvore esquerda
- uma árvore é dita AVL se todos os seus nós tem fator de balanceamento entre -1 e +1.
    - intuitivamente, essa propriedade garante que uma árvore AVL é pouco desbalanceada.
    - veremos que de fato ela limita o pior caso do desbalanceamento dessas árvores.

```
typedef int Item;
typedef int Chave;

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

typedef Noh *Arvore;
```

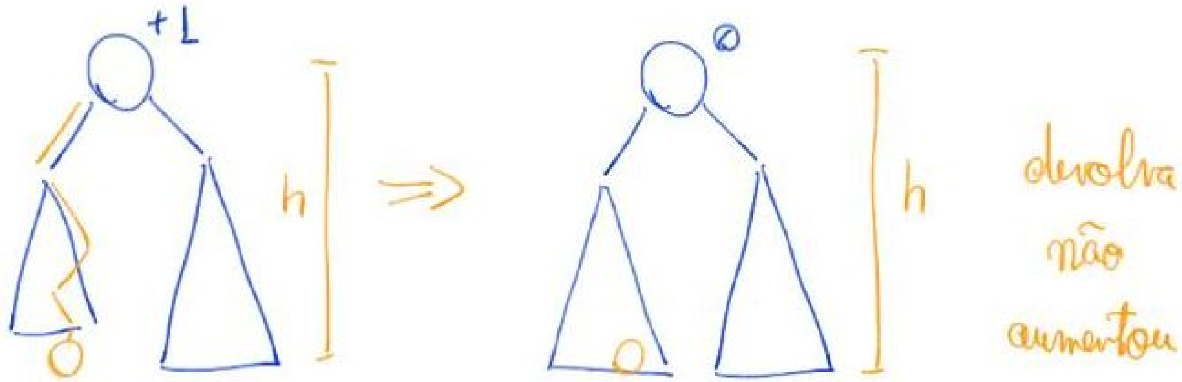
Exemplos:



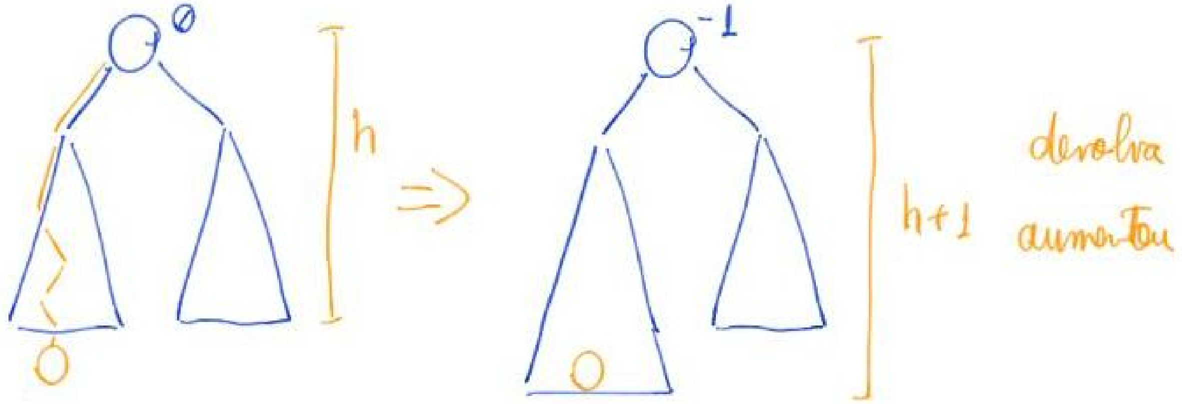


Inserção:

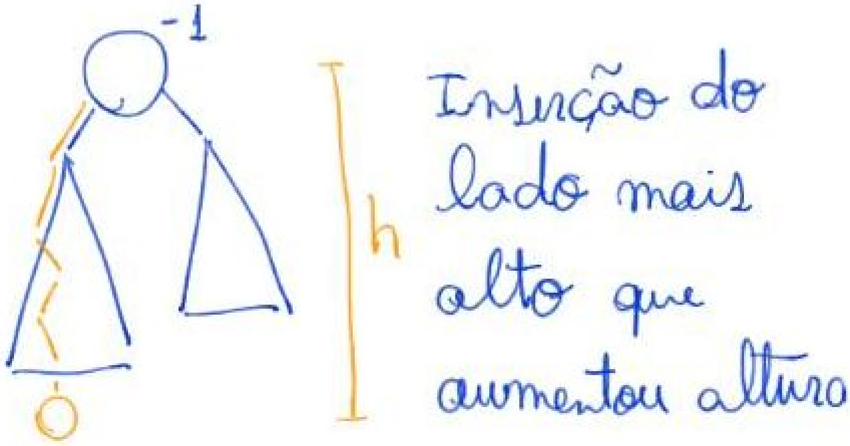
- a seguir são descritos os casos que um algoritmo de inserção recursivo analisa quando a altura de uma de suas subárvores aumenta após a inserção.
- se a altura não aumentar, o algoritmo não precisa fazer mais nada.
- caso 1: se a árvore era vazia crie um nó com dois filhos NULL e balanceamento 0.
- caso 2: se inseriu na subárvore mais baixa e aumentou a altura mude o balanceamento para zero e devolva que a altura da sua subárvore não aumentou.



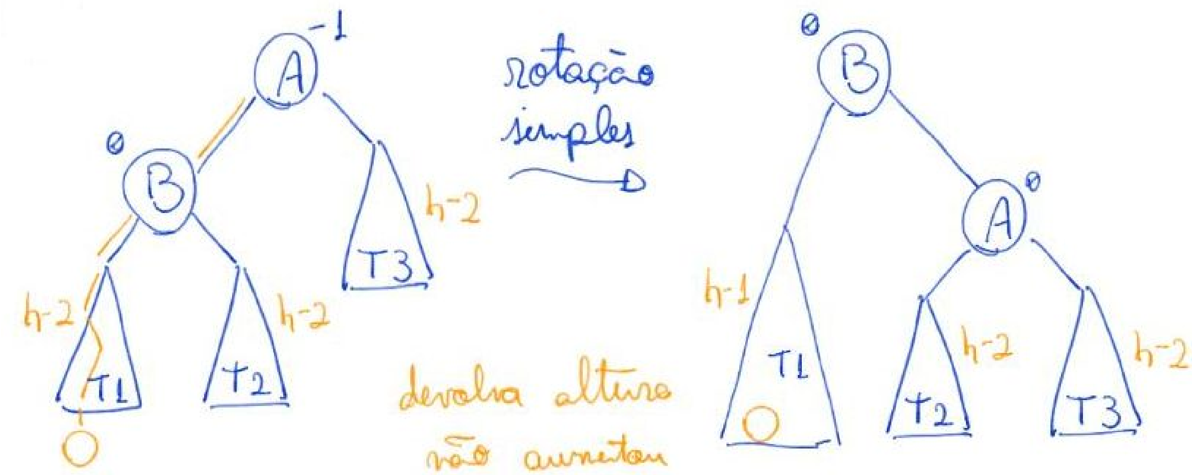
- caso 3: se inseriu em qualquer lado quando as alturas eram iguais (balanceamento 0) e aumentou a altura atualize o balanceamento para -1 ou +1 (dependendo do lado da inserção) e devolva que a altura da sua subárvore aumentou.



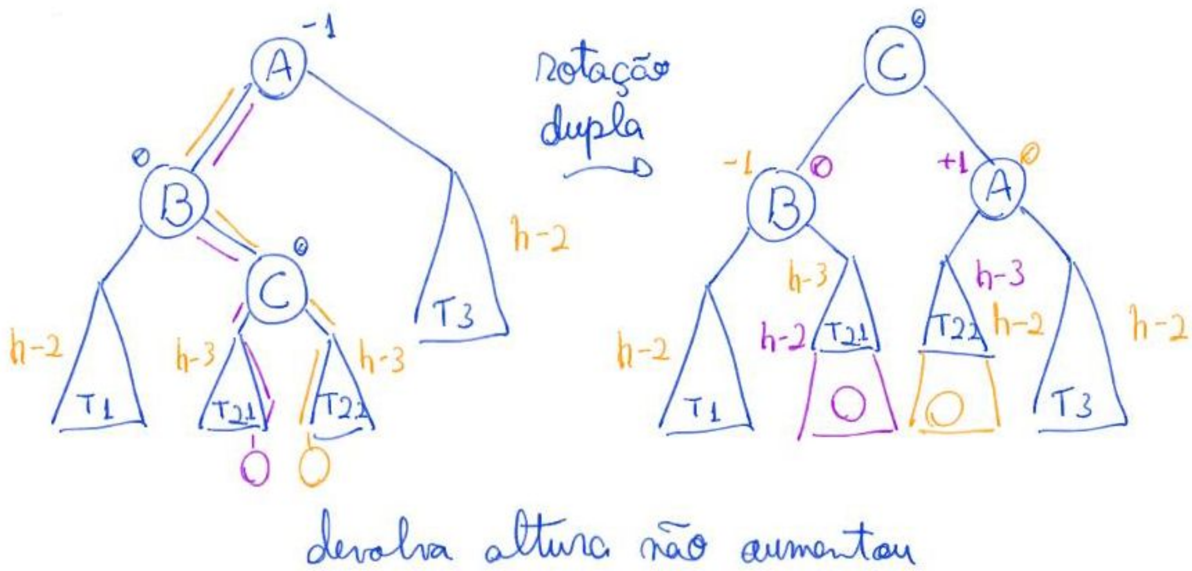
- caso 4: se inseriu na subárvore mais alta e aumentou a altura será necessário realizar uma ou mais rotações para manter a propriedade AVL.



- Caso 4.1:



- Caso 4.2:




```
Noh *novoNoh(Chave chave, Item conteudo)
{
```

```c
    Noh *novo;
    novo = (Noh *)malloc(sizeof(Noh));
    novo->bal = 0;
    novo->chave = chave;
    novo->conteudo = conteudo;
    novo->esq = NULL;
    novo->dir = NULL;
    //    novo->pai = ??
    return novo;
}

Arvore insereAVL(Noh *r, Noh *novo, int *h)
{
    if (r == NULL) // subárvore era vazia
    {
        novo->pai = NULL;
        *h = 1;
        return novo;
    }
    if (novo->chave <= r->chave) // desce à esquerda
    {
        r->esq = insereAVL(r->esq, novo, h);
        r->esq->pai = r;
        if (*h == 1) // altura da subárvore esquerda aumentou após inserção
        {
            if (r->bal == +1) // inseriu do lado mais baixo
            {
                r->bal = 0;
                *h = 0;
            }
            else if (r->bal == 0) // dois lados tinham a mesma altura
            {
                r->bal = -1;
                *h = 1;
            }
            else if (r->bal == -1) // inseriu do lado mais alto
            {
                if (r->esq->bal == -1) // inseriu à esquerda do filho esquerdo
                {
                    // rotação simples a direita
                    r = rotacaoDir(r);
                    r->dir->bal = 0;
                }
                else // r->esq->bal == +1 - inseriu à direita do filho esquerdo
                {
                    // rotação dupla
                    r->esq = rotacaoEsq(r->esq);
                    r = rotacaoDir(r);
```

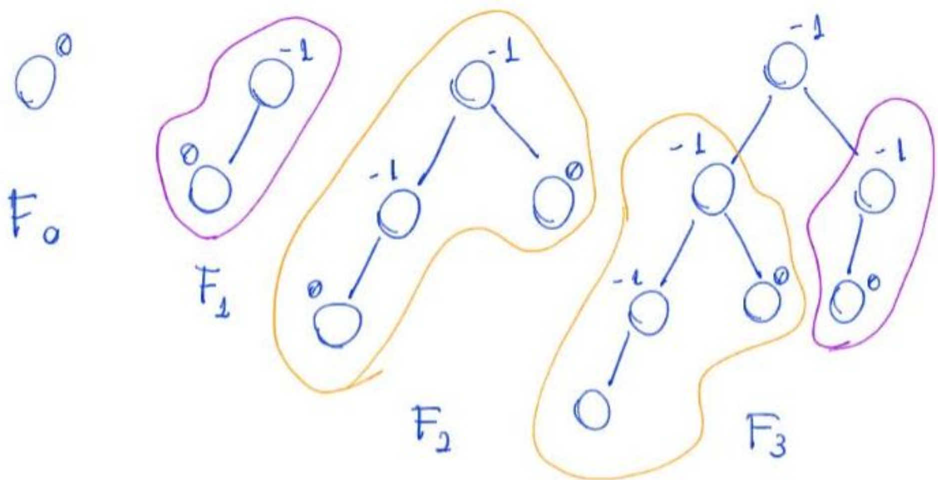
```
                    if (r->bal == 0)
                    {
                        r->esq->bal = 0;
                        r->dir->bal = 0;
                    }
                    else if (r->bal == -1)
                    {
                        r->esq->bal = 0;
                        r->dir->bal = +1;
                    }
                    else // r->bal == +1
                    {
                        r->esq->bal = -1;
                        r->dir->bal = 0;
                    }
                }
                r->bal = 0;
                *h = 0;
            }
        }
    }
    else // desce à direita
    {
        // complementar à inserção à esquerda
    }
    return r;
}
```

Remoção:

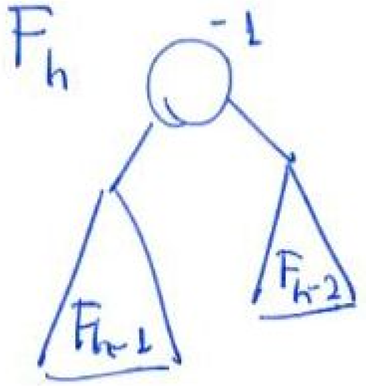
- similar aos casos da inserção, mas um tanto mais complexo.

Altura máxima de árvores AVL:

- quão desbalanceada pode ser uma árvore AVL?
    - Considere os seguintes exemplos:



- observe que a regra de formação dessas árvore é



  - ou seja, F_h é composta por um nó raiz cujo
    - filho esquerdo esquerdo é F_h-1
    - filho direito é F_h-2
- note que F_h é a árvore AVL de altura h com o menor número de nós possível. Isso porque, pensando recursivamente, tal árvore precisa ter:
  - uma subárvore com altura h-1,
  - outra com altura h-2,
  - e ambas as subárvores devem ter o menor número de nós possíveis.
- seja N(h) o número de nós da árvore F_h

  N(h) = N(h-1) + N(h-2) + 1, para h >= 2
  N(0) = 1
  N(1) = 2
- resolvendo essa recorrência temos que

  N(h) = Fibonacci(h+2) - 1 >= 2^(h/2)
  h/2 <= lg N(h)
  h <= 2 lg N(h)
- por fim, como o número de nós n de qualquer árvore AVL de altura h é maior ou igual a N(h), temos

  h <= 2 lg N(h) <= 2 lg n

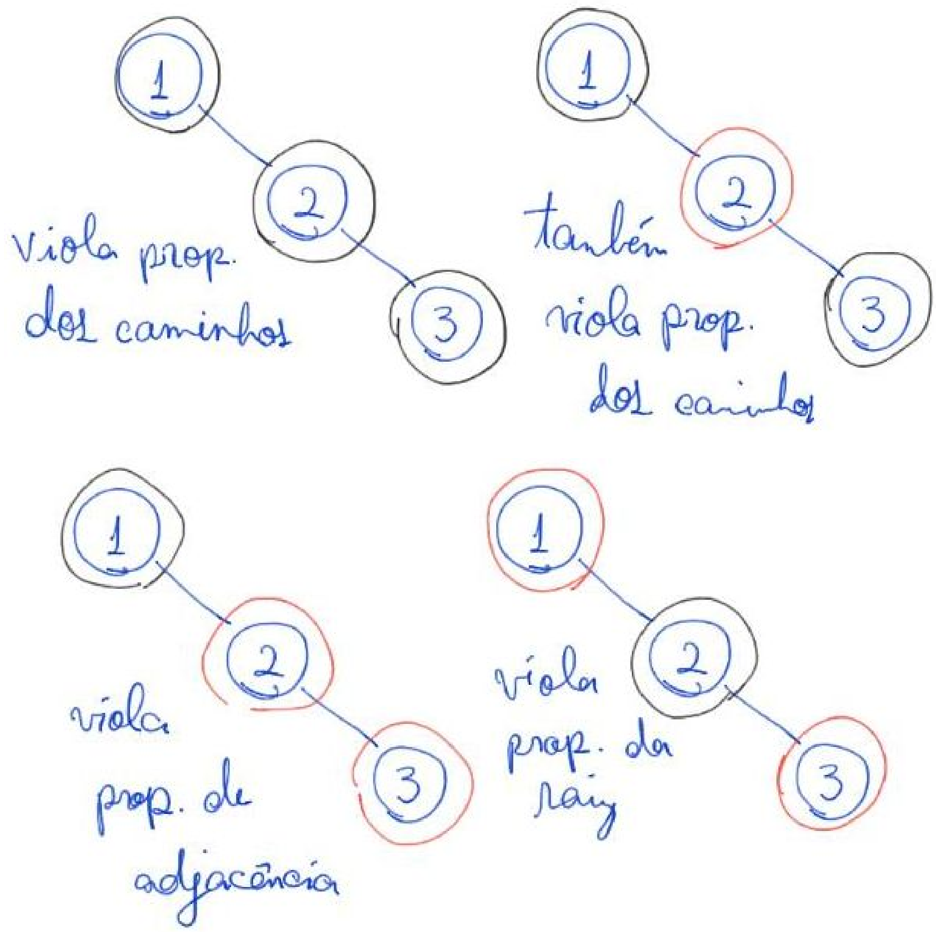
Árvores Rubro-Negras

Definição:

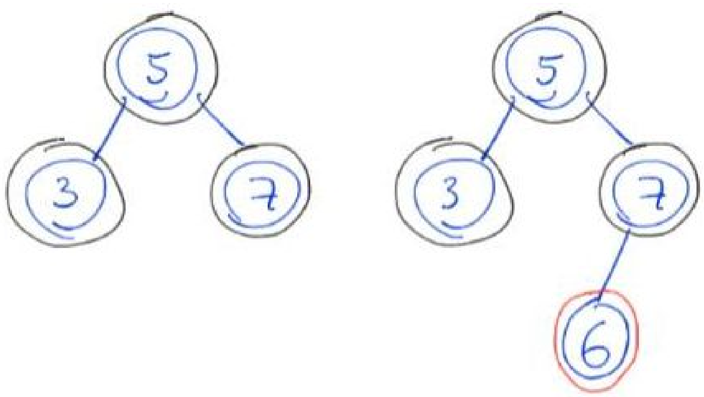
1. cada nó é vermelho ou preto.
2. raiz é sempre preta.
3. dois nós vermelhos não podem ser adjacentes,
   - ou seja, um nó vermelho só pode ter filhos pretos.
4. todo caminho da raiz até um apontador NULL (caminho raiz-NULL) tem o mesmo número de nós pretos.
   - pense nesses caminhos como buscas mal sucedidas.

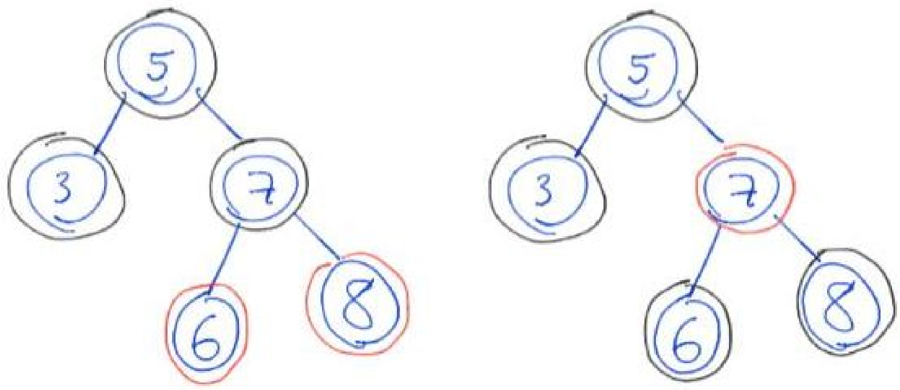
Para ganhar intuição de que essas propriedades levam a uma árvore balanceada

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



Exemplos:





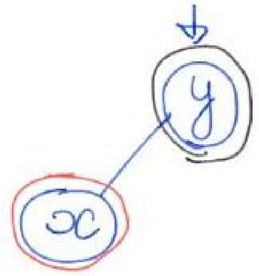
Altura Máxima:

- toda árvore rubro-negra tem altura <= 2 lg (n+1)
- Demonstração:
    - observe que, se todo caminho raiz-NULL de uma árvore tem pelo menos k nós,
        - então os primeiros k níveis da árvore devem estar completos.
        - caso contrário haveria um caminho da raiz até o nó ausente em um dos k níveis iniciais, resultando em um caminho raiz-NULL de comprimento menor que k.
    - decorre disso que, o número de nós n desta árvore é maior ou igual que o número de nós numa árvore binária completa de altura k,
        - ou seja, n >= 2^k - 1.
    - portanto,
        - k <= lg (n + 1)
    - note que, numa árvore rubro-negra
        - pela propriedade 4 da definição, todo caminho raiz-NULL tem um mesmo número de nós pretos k.
            - lembre que k <= lg (n + 1)
            - e que k só está contando os nós pretos do caminho.
        - mas, pelas propriedades 1 e 3 da definição, todo caminho tem no máximo um nó vermelho para cada nó preto.
            - assim, o número total de nós em qualquer caminho raiz-NULL <= 2 * k <= 2 lg (n + 1).

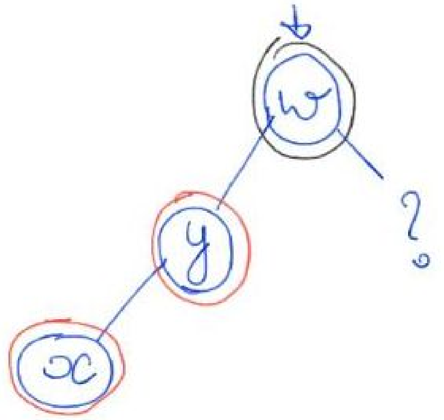
Inserção:

- a ideia geral é inserir normalmente (percorrendo um caminho descendente na árvore), então usar recoloração e rotações (ao percorrer o caminho no sentido ascendente) para restabelecer as propriedades da definição.
- inserir novo nó x como uma folha e colori-lo de vermelho.
- caso 1: se y, o pai de x, não for vermelho,
    - as propriedades da definição estão mantidos

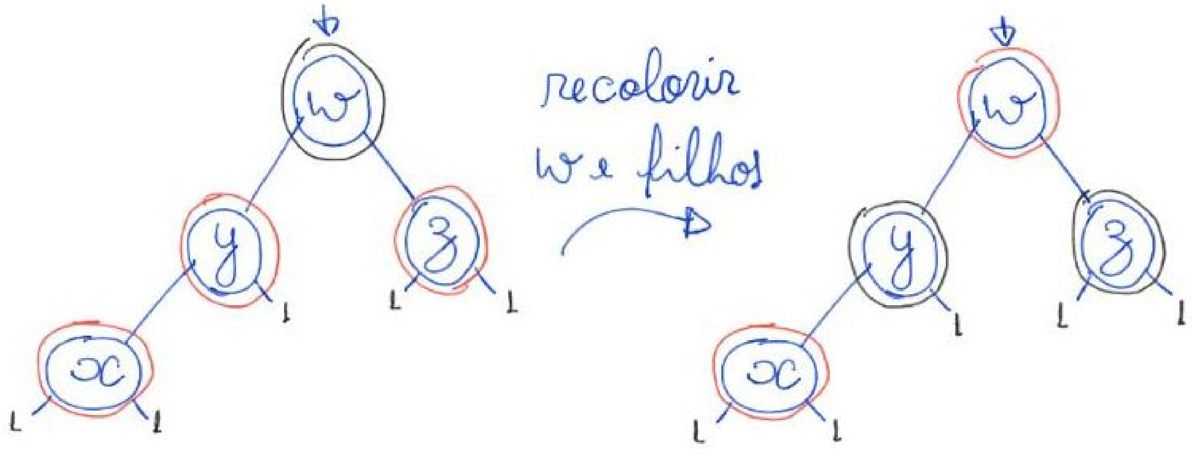
○ e a inserção pode terminar.



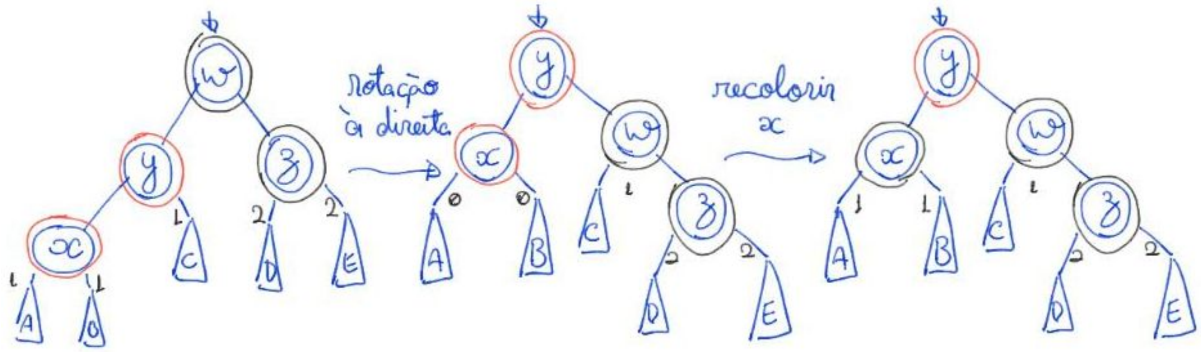
- caso 2: se y é vermelho,
    - então sabemos que ele não é a raiz e que w, o pai de y, é preto.

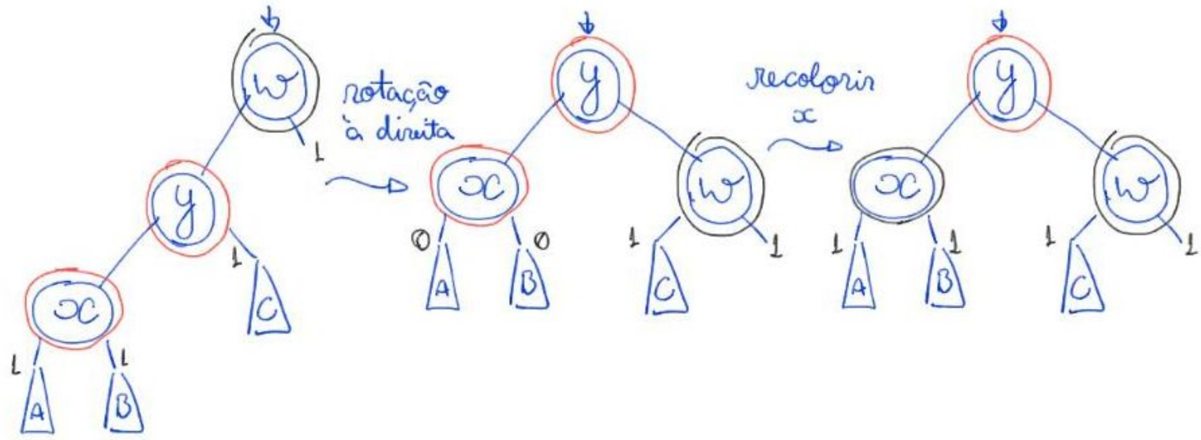


    - caso 2.1: w tem outro filho z que tem cor vermelha. Neste caso vamos
        - recolorir z e y para preto
        - recolorir w para vermelho
            - assim mantemos a propriedade 4
                - pois o número de nós pretos nos caminhos raiz-NULL é preservado
            - e resolvemos a propriedade 3 entre x e y
        - resta propagar o problema para o pai de w
            - já que w ficou vermelho, ele pode violar a propriedade 3 com relação a seu pai
            - se w for a raiz da árvore voltamos a cor dele para preto



- caso 2.2: w tem não tem outro filho vermelho
  - note que w pode ter outro filho preto ou não ter outro filho
  - fazemos uma rotação à direita a partir de w
    - 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
    - resolvendo tanto o problema dos vermelhos adjacentes
    - quanto do número de pretos nos caminhos raiz-NULL
  - resta propagar o problema para o pai de y
    - já que y é a nova raiz da subárvore e é vermelho, ele pode violar a propriedade 3 com relação a seu pai
    - se y for a raiz da árvore voltamos a cor dele para preto





- a seguir é apresentado código baseado nas funções recursivas para inserção na versão da árvore rubro-negra apresentada por Sedgewick

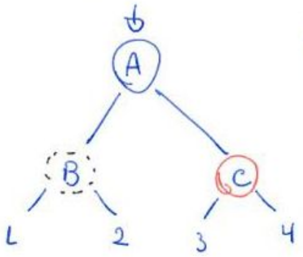
```
Noh *novoNoh(Chave chave, Item conteudo)
{
   Noh *novo;
   novo = (Noh *)malloc(sizeof(Noh));
   novo->vermelho = 1;
   novo->chave = chave;
   novo->conteudo = conteudo;
   novo->esq = NULL;
   novo->dir = NULL;
   //    novo->pai = ??
```

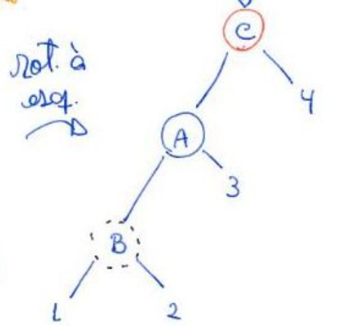
```
    return novo;
}

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

- seguem figuras exemplificando as operações do código anterior

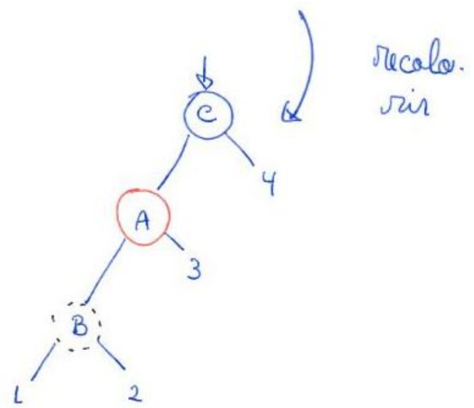
- se f.d. é vermelho e f.e. ñ é vermelho
  ↳ preto ou ausente

  - rotação à esq.



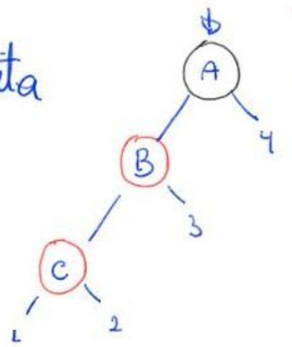


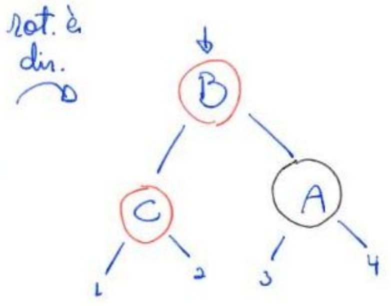
  - recolore de acordo p/ manter inalterado o número de nós pretos nos caminhos
    - note que A pode ter cor preta ou vermelha e que sua cor é herdada por C

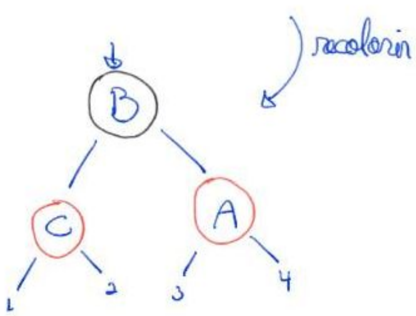


- se f.e. é vermelho e f.e. do f.e. é vermelho

  - rotação à direita

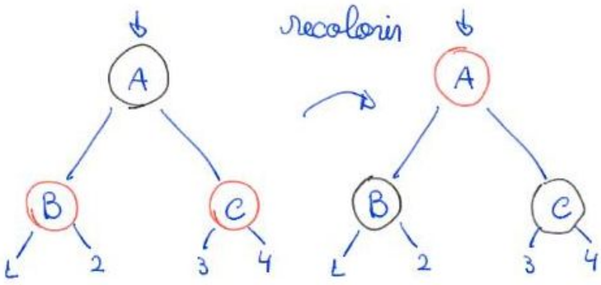






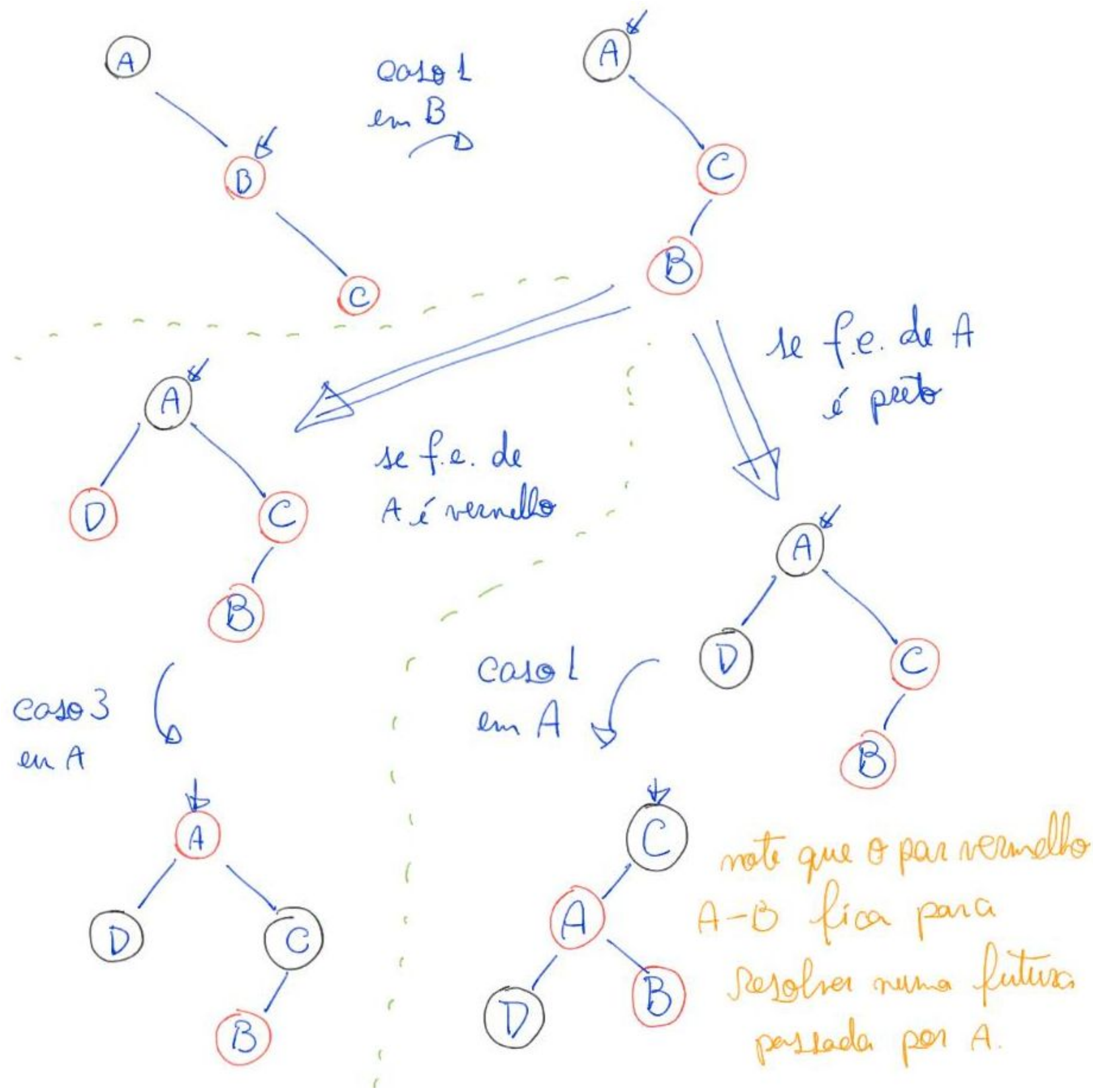
  - recolore de acordo p/ manter inalterado o número de nós pretos nos caminhos
    - note que neste caso A só pode ser preto (por quê?)

- se f.e. é vermelho e f.d. é vermelho
  - recolore de modo que o pai passe a ser vermelho e os filhos fiquem pretos



    - note que neste caso A certamente é preto (por que?)
    - note que a recoloração preserva o número de nós pretos em todos os caminhos.

  - uma observação é que nesta variante da árvore rubro-negra a raiz nem sempre fica vermelha
  - outra é que vermelhos podem ficar adjacentes ao final de uma inserção,
    - como mostra o exemplo a seguir



Remoção:

- similar aos casos da inserção, mas um tanto mais complexo.