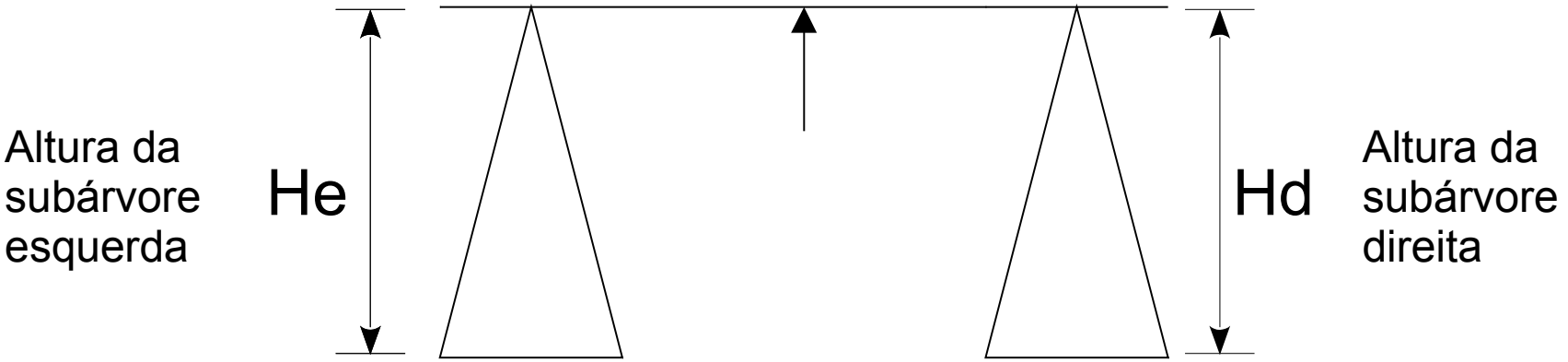
ABB-balanceada

Uma árvore binária é dita balanceada se todos os seus nós apresentarem um nível de equilíbrio entre o par de subárvores.

Analogamente, seria como um nó correspondesse ao “fiel” de uma balança e suas subárvores fossem as bandejas apresentando um certo equilíbrio.

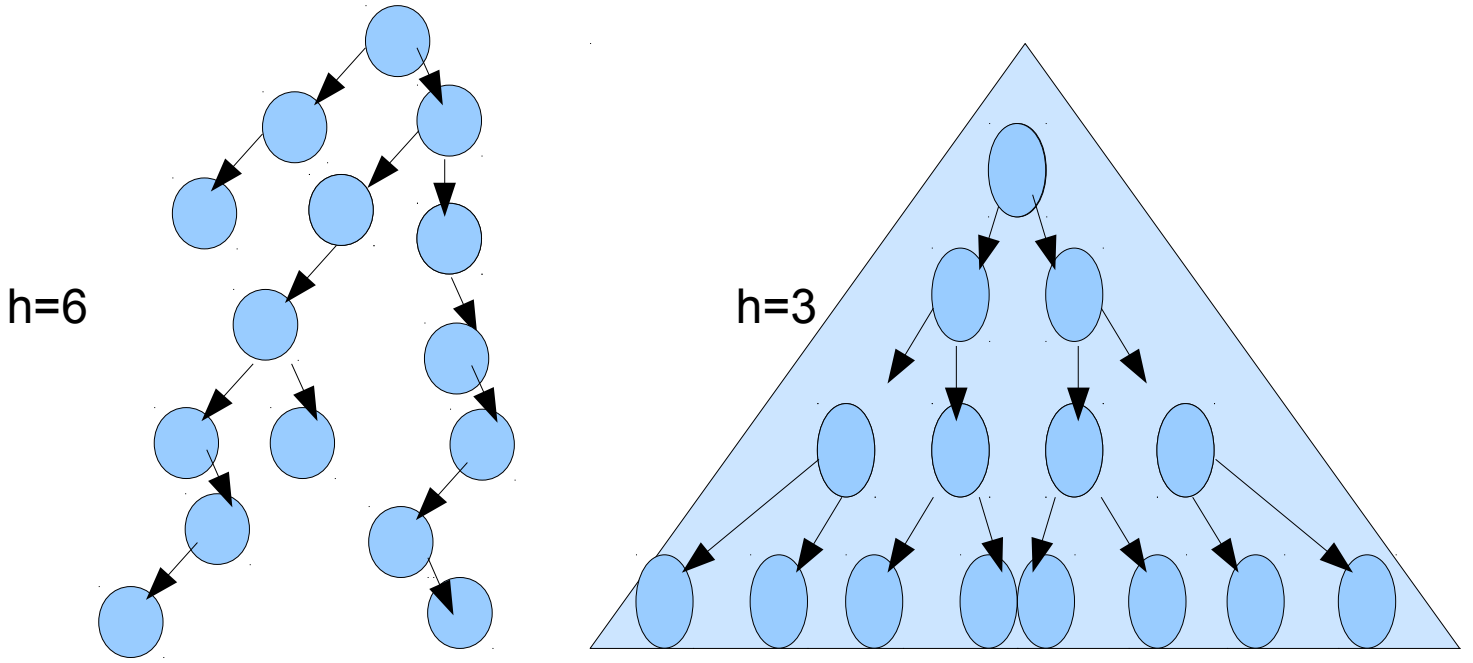
Na ABB-balanceada, o nível de equilíbrio de um nó é representado pelo Fator de Balanceamento associado a esse nó FB=He-Hd (alguns autores consideram FB=Hd-He)



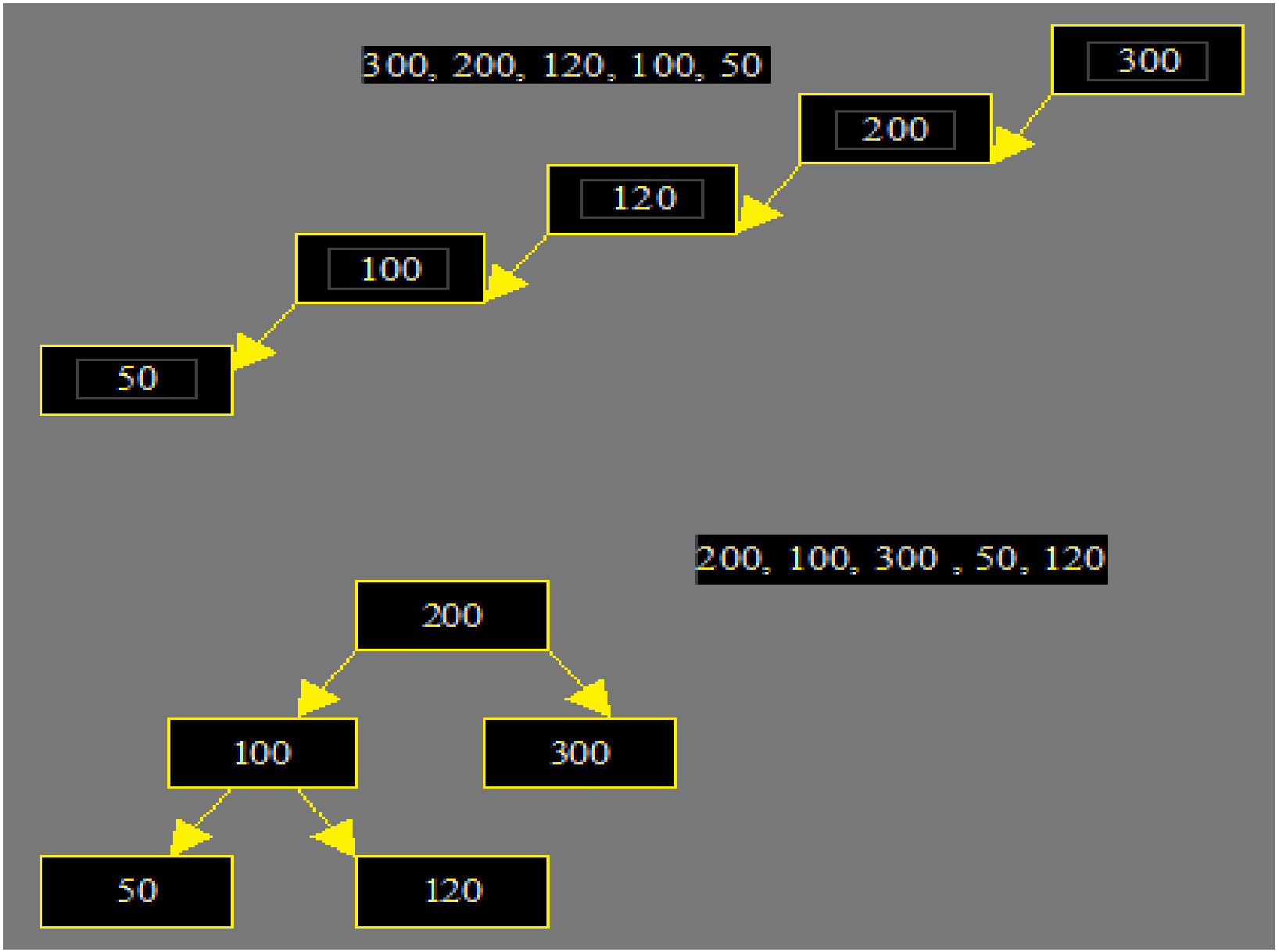
Árvores mais altas implicam em potencial lentidão nas operações de busca, inserção ou remoção;

Por outro lado, as árvores mais balanceadas apresentam uma distribuição de elementos mais equilibrada e...

...para um mesmo conjunto de itens, o balanceamento reduz a altura final da árvore otimizando o tempo de uma busca, inserção ou remoção:



A sequência de inserções ou remoções na ABB influencia no nível de balanceamento da ABB a ponto de poder até degenerá-la à uma lista.



Ou seja, a priori, para garantir o balanceamento teríamos que controlar a ordem das entradas de itens ou a ordem das remoções, o que na prática é inviável.

Deve-se então providenciar uma forma de “rebalancear” a árvore a cada momento que esta se desequilibra.

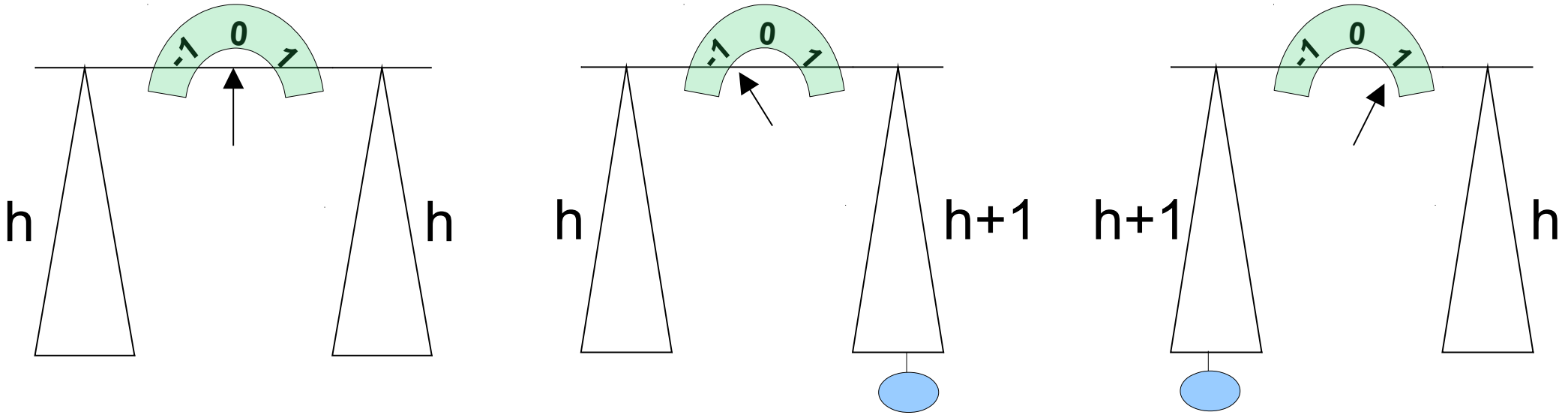
Para tanto existem algoritmos de inserção e remoção [2,3] que garantem reequilibrar os nós caso tais operações provoquem desbalanceamento.

Aqui será dado destaque a uma das estratégias mais conhecidas: a AVL.

A propósito... o nome AVL deriva dos seus criadores Adelson-Velskii e Landis.

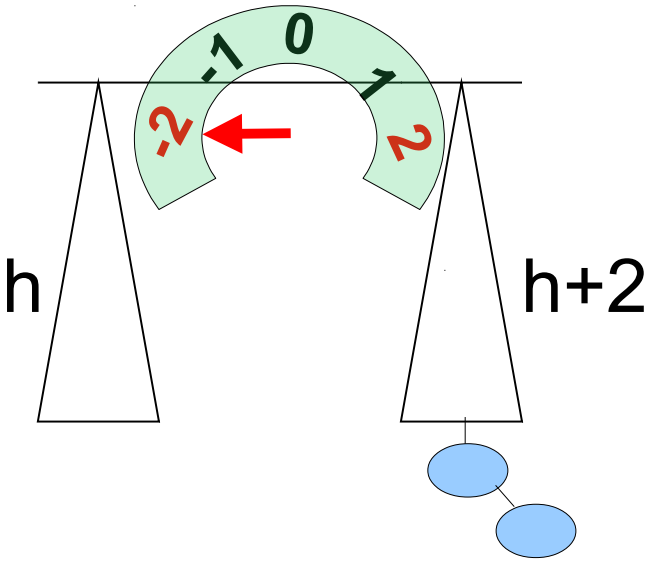
Fator de Balanceamento: FB=He-Hd

Para considerarmos a árvore como AVL-balanceada todos os seus nós terão que apresentar -1 ≤ FB ≤ 1.

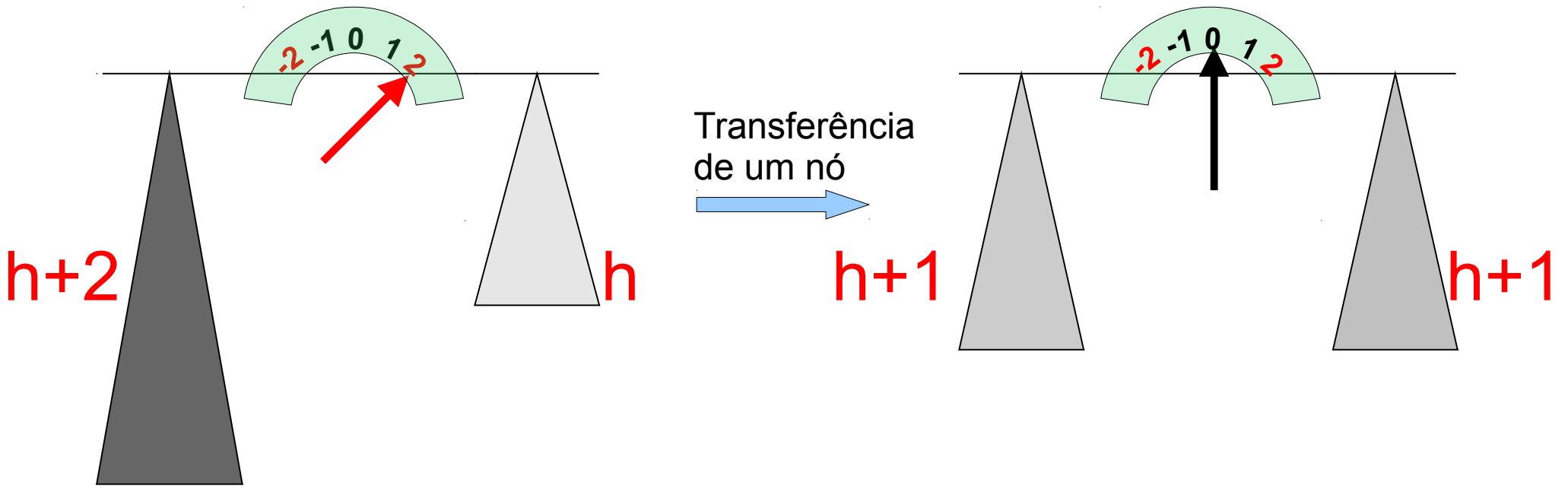


No entanto, um nó pode ser desbalanceado em decorrência de uma inserção ou remoção...

Nesses casos, é necessário redistribuir os “pesos na balança”, transferindo nós entre subárvores por meio de operações chamadas rotações...



Rotação: “transferência de pesos (nós)” entre subárvores:



Pode ser necessária mais de uma rotação para <u>evitar o efetivo desbalanceamento</u>.

→→ Exemplos serão tratados na lista exercícios.

O custo computacional das rotações é compensado pela maior eficiência na busca de dados, ocasionada pelo rebalanceamento.

Inserção na ABB-AVL:

Considere uma implementação recursiva da inserção na ABB comum.

Após a inserção de um novo nó (K) há o retorno da recursão, para cada nó P visitado nesse retorno é avaliado o FB(P).

Se FB(P) indicar desbalanceamento em P, então deve ser realizada a devida rotação.

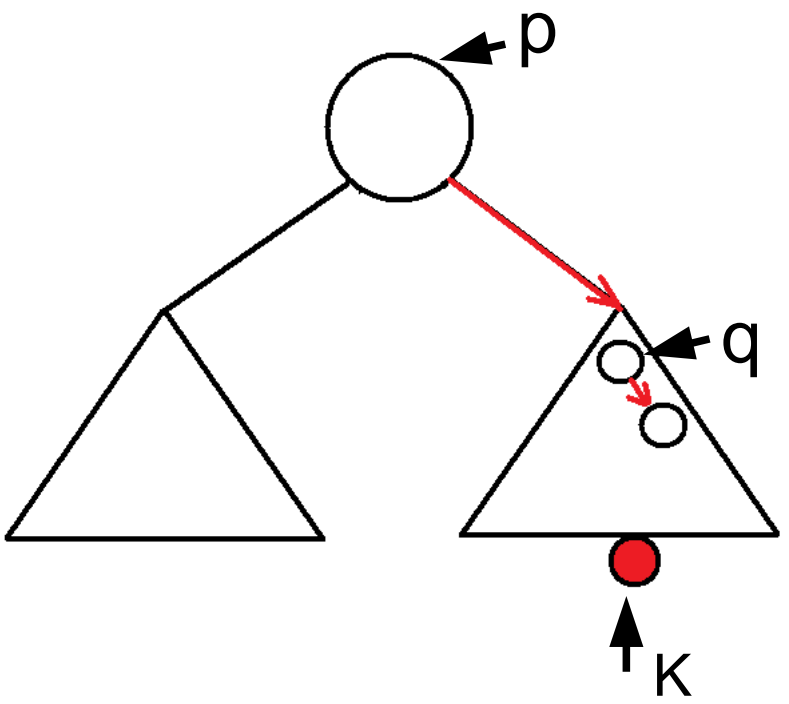
As rotações são de quatro tipos (LL, RR, RL e LR) e dependem do caminho tomado para a inserção do novo nó a partir de P e seu filho. Esse caminho é rastreado no retorno da recursão.

Rotação devido a inserções: RR

Considerando:

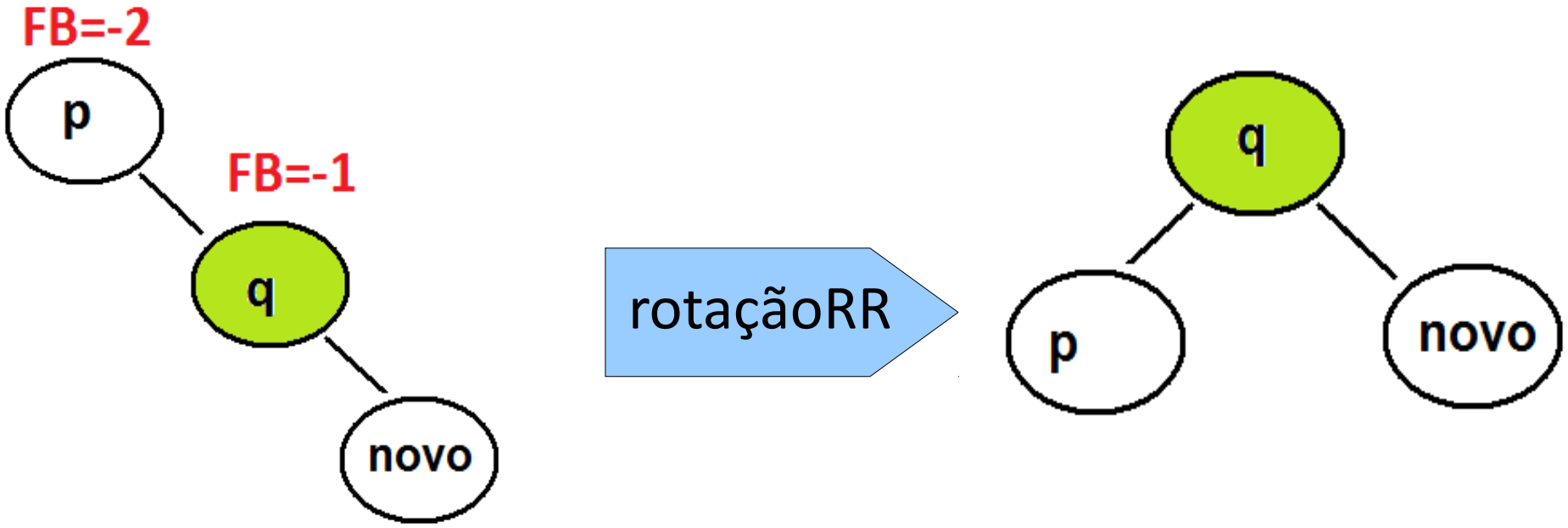
- P o nó atual visitado no retorno da recursão e FB(P)=±2;
- K foi inserido na subárvore da direita do filho direito de P.

Se o nó P foi desbalanceado devido a inserção do nó K, então a rotação de tratamento será: rotaçãoRR.




```
no * rotacaoRR(p) //giro à esq
{    q=p->dir;
     tmp= q->esq;
     q->esq = p;
     p->dir= tmp;
     return(q); /* q: nova raiz */
}
```

Rotações devido a inserções: RR

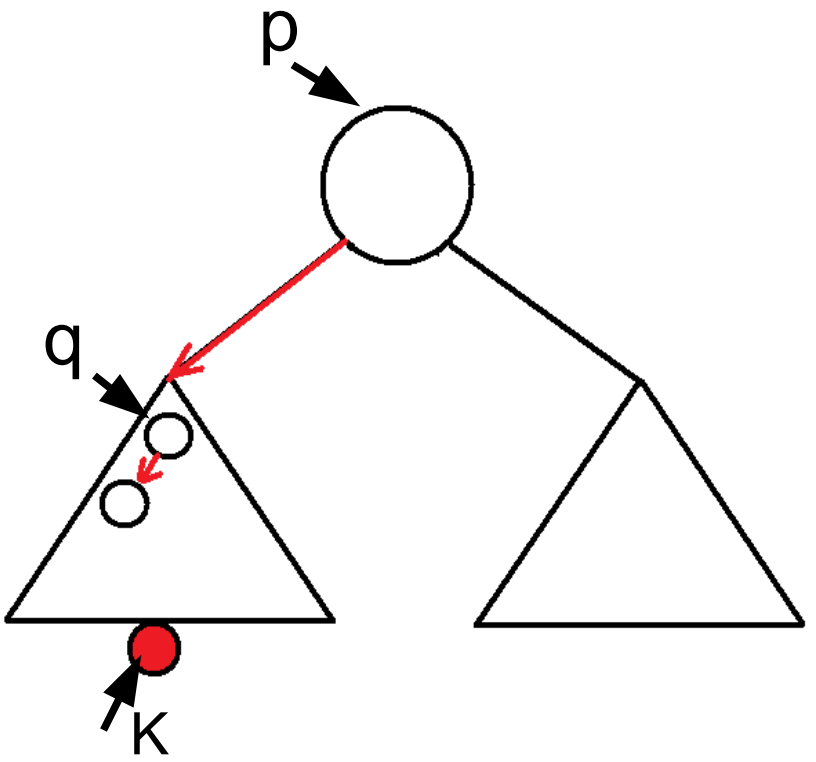


Rotações devido a inserções: LL

Considerando:

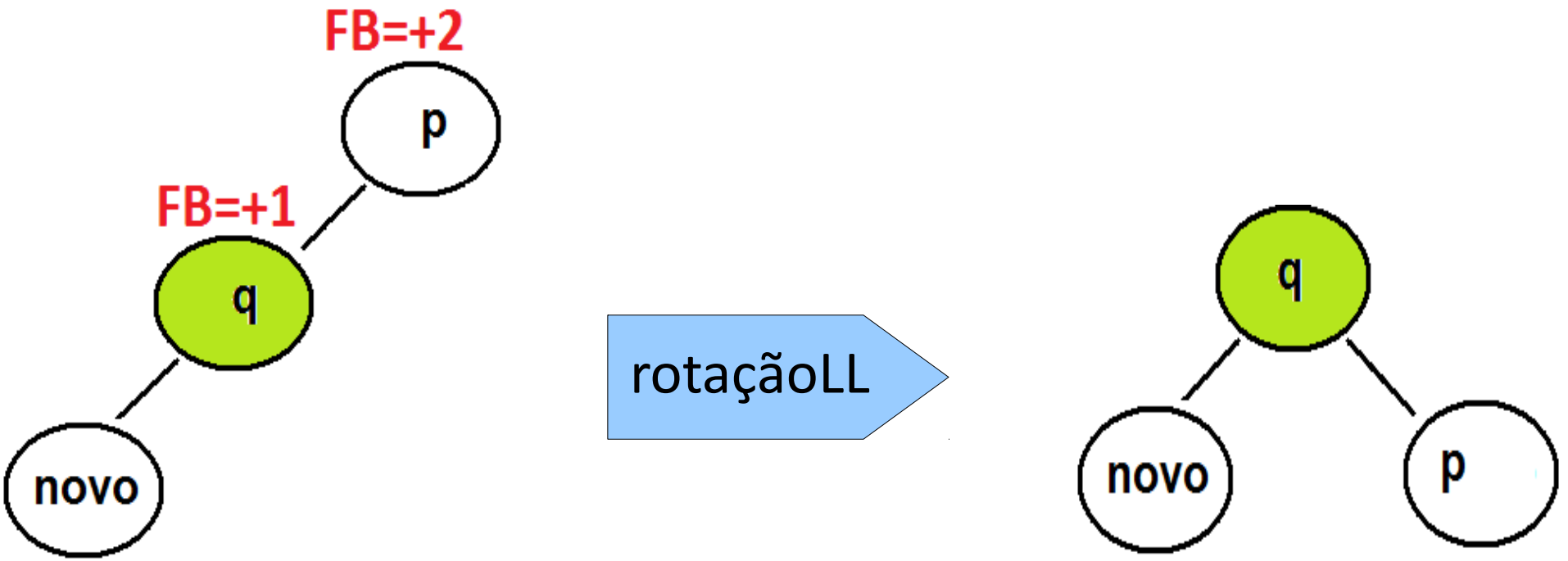
- P o nó atual visitado no retorno da recursão e FB(P)=±2;
- K foi inserido na subárvore esquerda do filho esquerdo de P.

Se o nó P foi desbalanceado devido a inserção do nó K, então a rotação de tratamento será: rotaçãoLL.




```
no * rotacaoLL(p) // giro à dir
{    q=p->esq;
     tmp= q->dir;
     q->dir = p;
     p->esq= tmp;
     return(q); /* q: nova raiz */
}
```

Rotações devido a inserções: LL

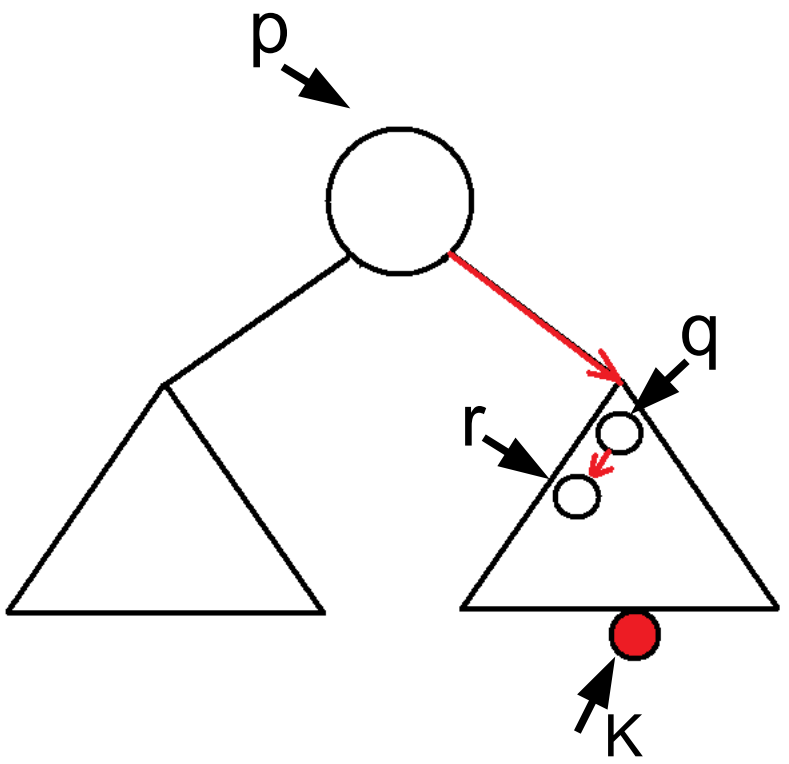


Rotações devido a inserções: RL

Considerando:

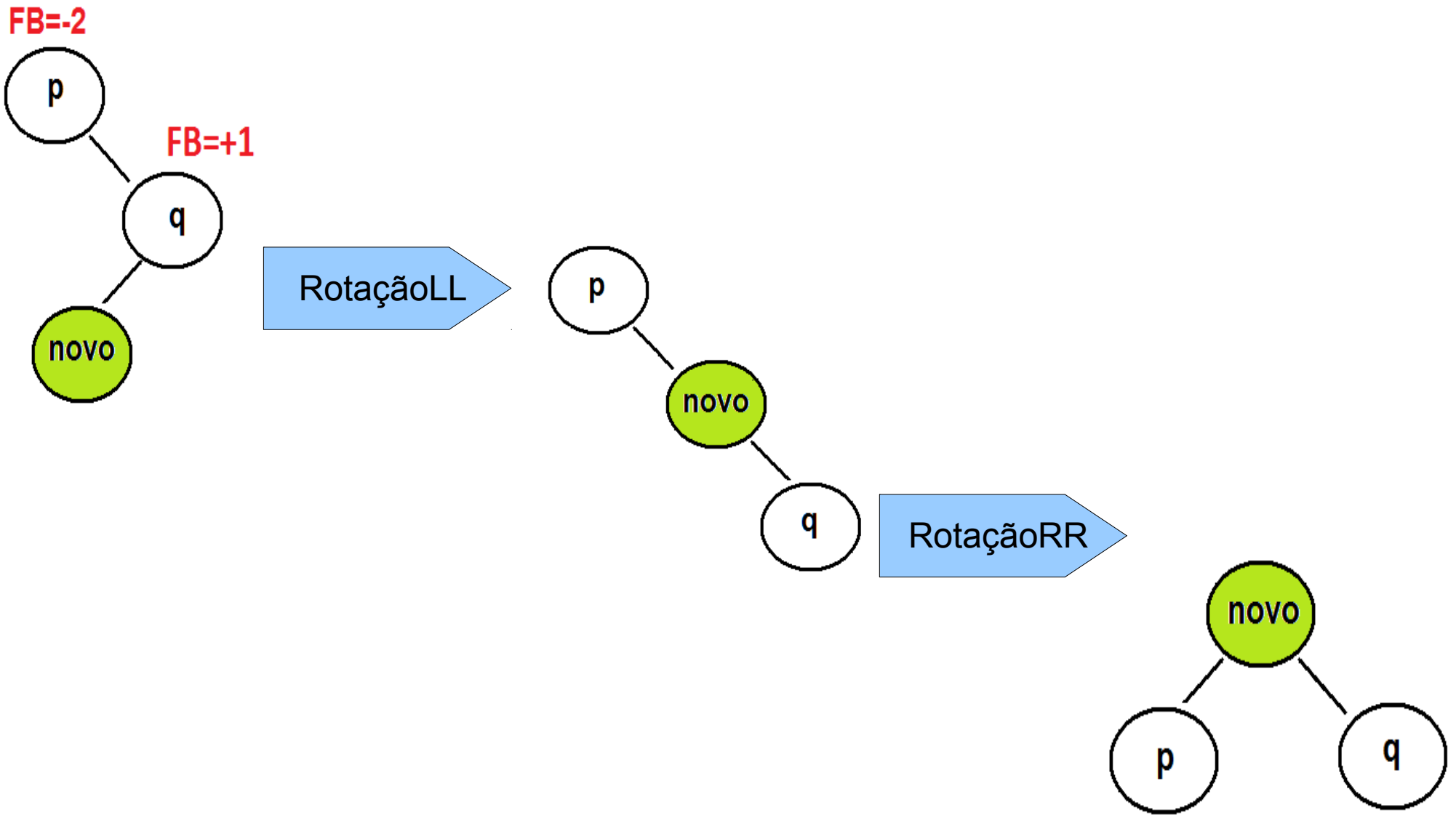
- P o nó atual visitado no retorno da recursão e FB(P)=±2;
- K foi inserido na subárvore da esquerda do filho direito de P.

Se o nó P foi desbalanceado devido a inserção do nó K, então a rotação (dupla) de tratamento será: rotaçãoRL.




```
no * rotacaoRL(p) //r gira à dir depois à esq
{    q=p->dir;
     rotacaoLL(q);
     raiz=rotacaoRR(p);
     return(raiz); /* r: nova raiz */
}
```

Rotações devido a inserções: RL

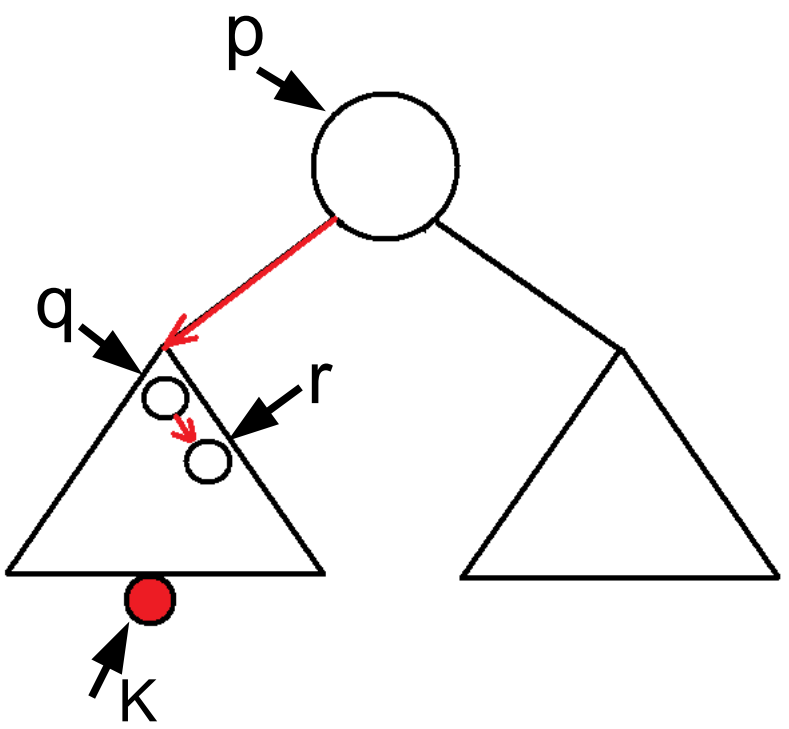


Rotações devido a inserções: LR

Considerando:

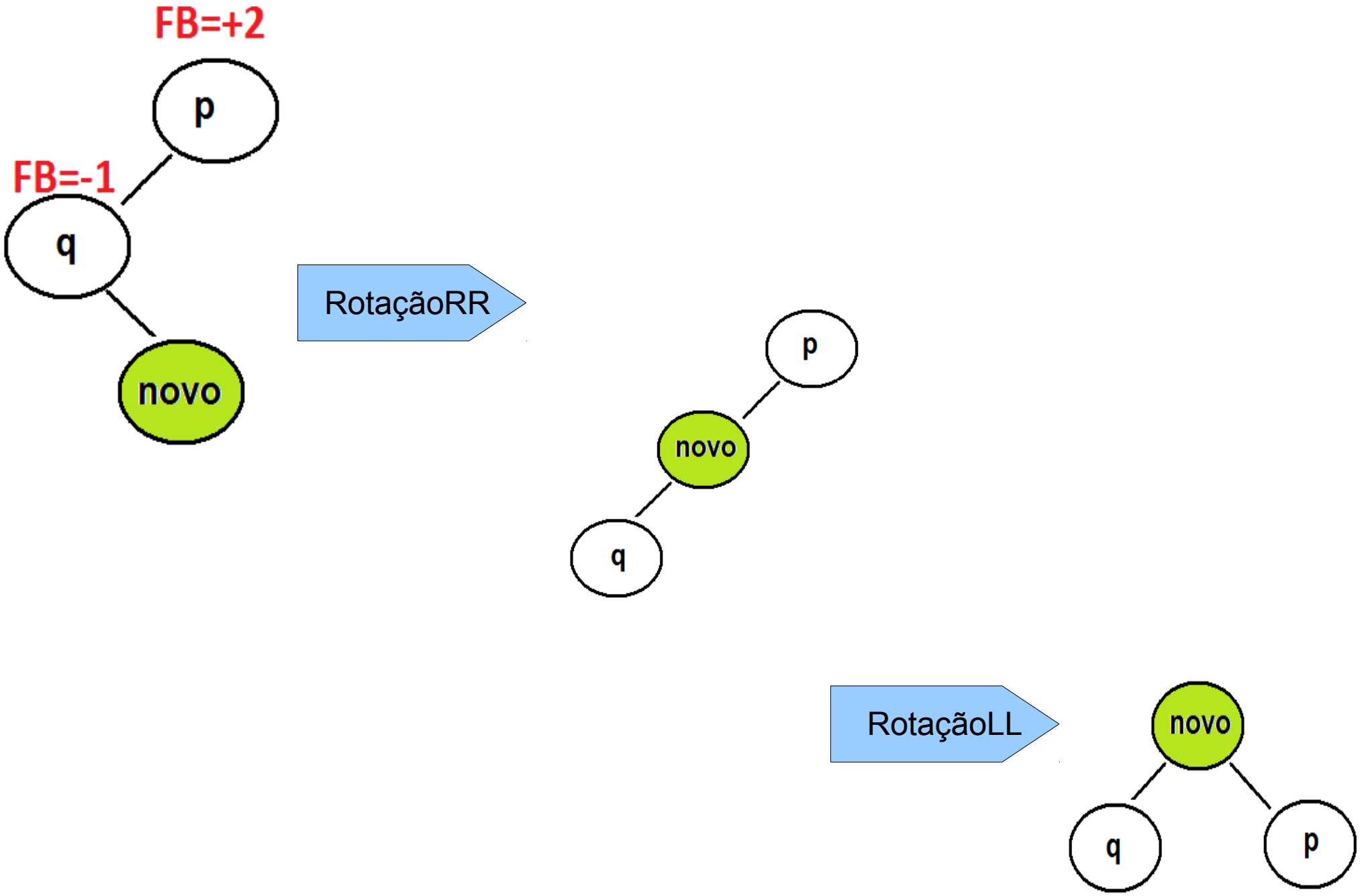
- P o nó atual visitado no retorno da recursão e FB(P)=±2;
- K foi inserido na subárvore da direita do filho esquerdo de P.

Se o nó P foi desbalanceado devido a inserção do nó K, então a rotação (dupla) de tratamento será: rotaçãoLR.




```
no * rotacaoLR(p) //r gira à esq depois à dir
{    q=p->esq;
     rotacaoRR(q);
     raiz=rotacaoLL(p);
     return(raiz); /* r: nova raiz */
}
```

Rotações devido a inserções: LR



Remoção na ABB-AVL:

Considere uma implementação recursiva da remoção na ABB comum.

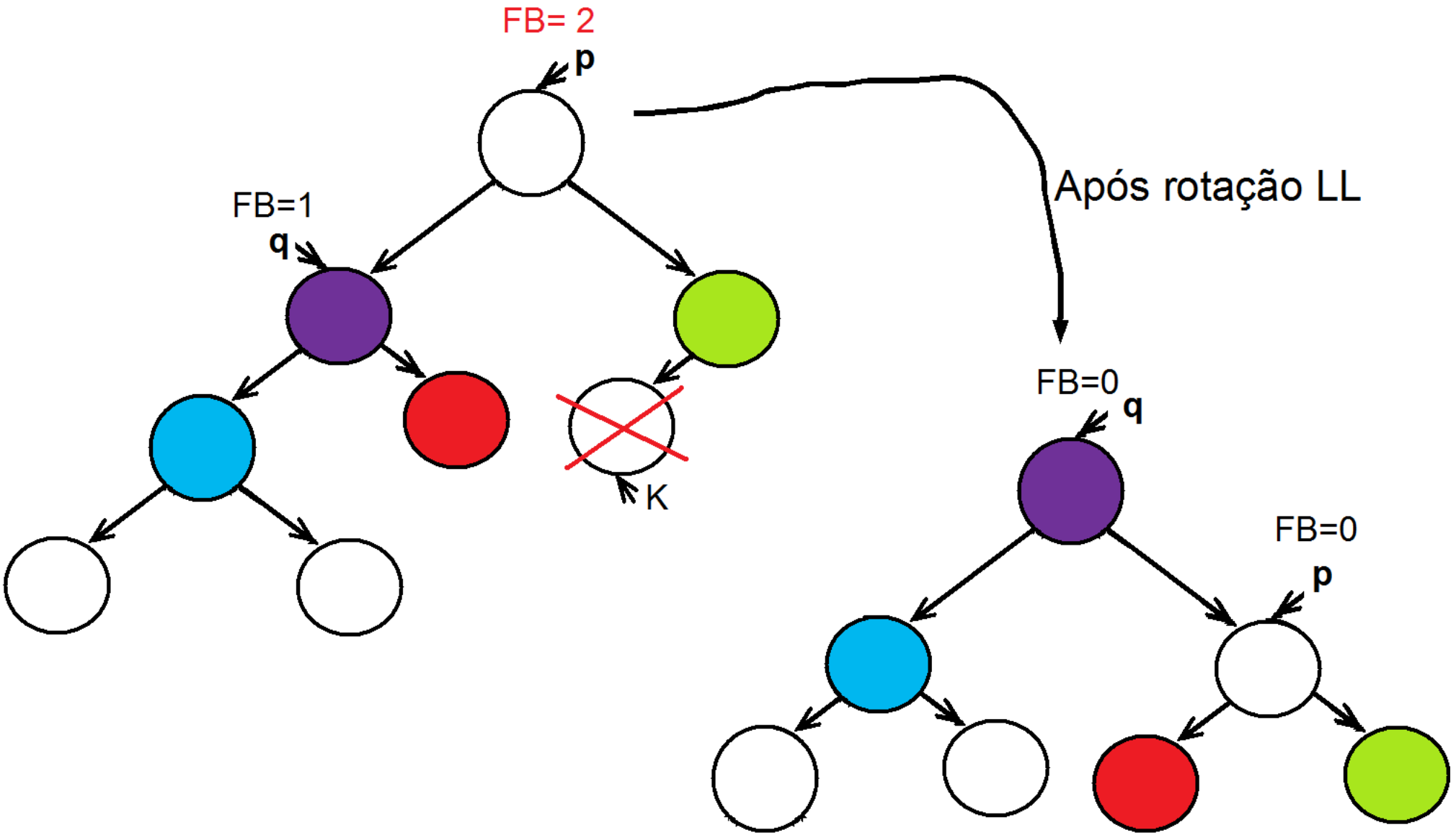
Após a remoção de um nó **folha K** há o retorno da recursão, o FB(P) é avaliado para cada nó P visitado nesse retorno.

Se FB(P) indicar desbalanceamento em P, então deve ser realizada a devida rotação (LL, RR, RL e LR).

A escolha do tipo de rotação dependem do caminho tomado pela remoção (rastreado no retorno da recursão) e da análise de qual filho de P poderá contribuir no balanceamento.

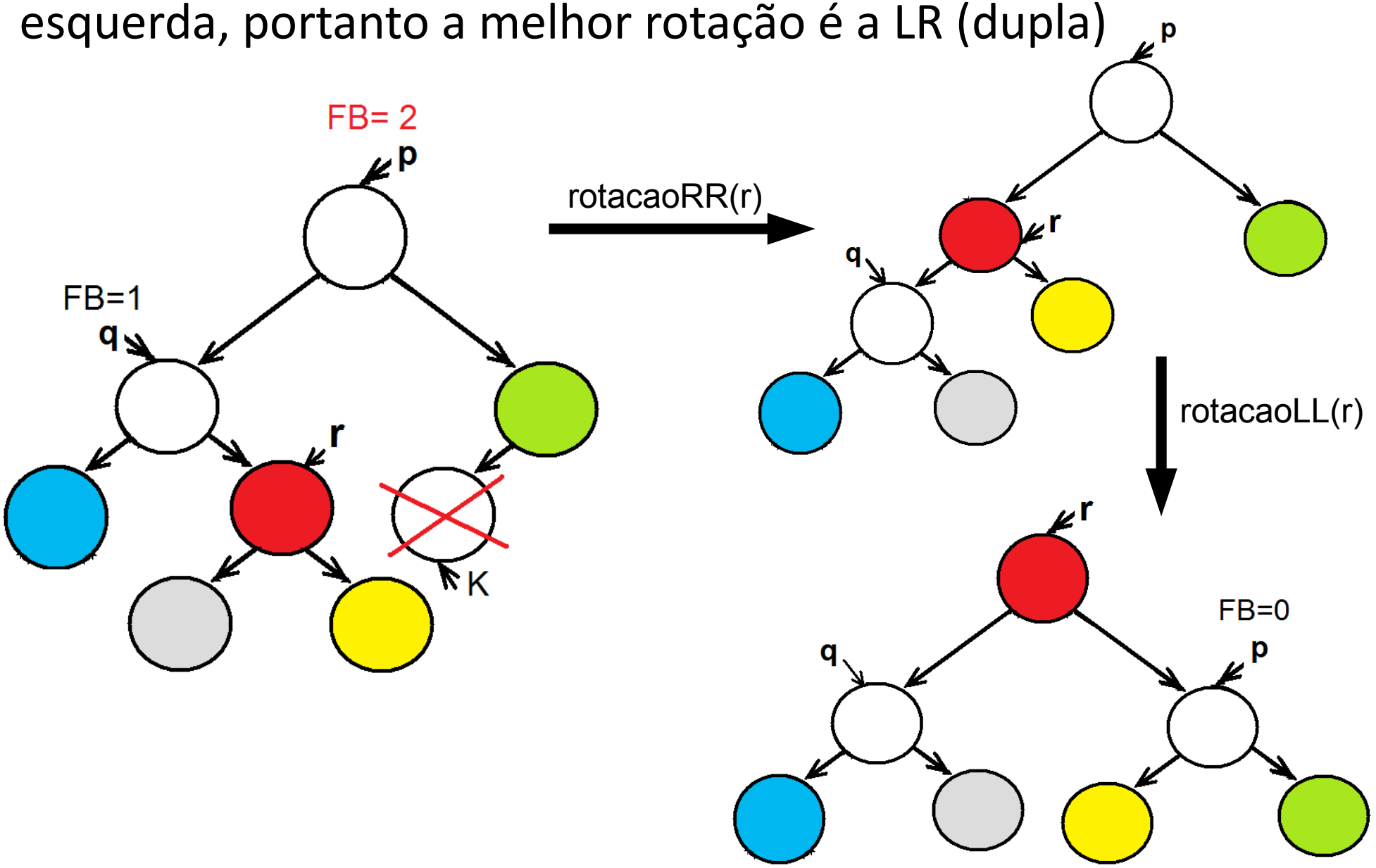
Rotações devido a remoções de **folhas**:

Exemplo 1: a subárvore esquerda do nó q é mais alta que a direita, portanto a melhor rotação é a LL (simples)



Rotações devido a remoções de **folhas**:

Exemplo 2: a subárvore direita do nó *q* é mais alta que a esquerda, portanto a melhor rotação é a LR (dupla)

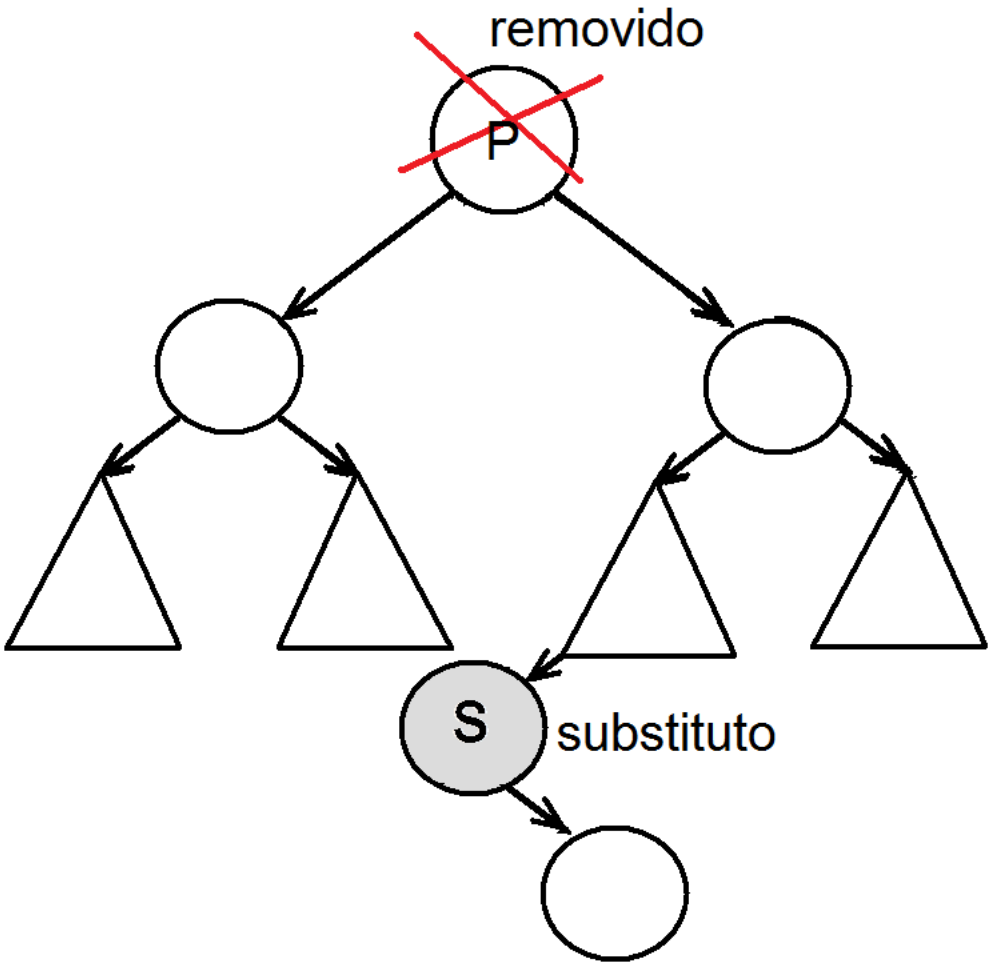


Rotações devido a remoções de **não-folhas**:

Caso seja necessário, a determinação da rotação é baseada na análise da substituição do nó removido (P).

Na prática o nó substituto (S) é removido da sua posição original e transferido para a posição “desocupada” pelo nó P, o qual não mais pertencerá a árvore.

O movimento de S pode desbalancear algum ascendente seu, esse caso deve ser tratado com rotação(ções) similarmente aos casos anteriores.



TDA ABB_AVL

O TDA-ABB-AVL é obtido a partir do TDA-ABB, já estudado, simplesmente pela modificação do nó de dados (acréscimo do campo FB) bem como das funções de inserção e remoção.

Estas passam a realizar as rotações adequadas ao balanceamento. Também é preciso inserir os códigos das próprias rotações.

Alguns autores trabalham com o acréscimo de ponteiro para o nó pai.

Arquivo TDA_ABB_PRIV.H

```
#include “TDA_ABB.H”
typedef struct noABB
    {   void *dados;
        int FB;  // ←←←
        struct noABB *esq;
        struct noABB *dir;
} NoABB, *pNoABB;
```

TDA ABB_AVL

Os códigos (inserção/remoção) são clássicos, podendo ser acessados em diversas fontes, tais como:

i) Material do prof. Nonato do icmc [1], onde um nó da árvore possui ponteiro para o seu pai;

ii) O algoritmo de Szwarcfiter [1] e Wirth [2] considera a altura calculada através da contagem de nós e não de arestas, a altura de uma folha é igual a 1 e FB(i) = He(i) – Hd(i).

[1] http://www.lcad.icmc.usp.br/~nonato/ED/AVL/node67.html
[2] Szwarcfiter, Jayme L. & Markenzon L. “Estruturas de Dados e Seus Algoritmos”. Rio de Janeiro. Ed. LTC. 1994.
[3] Wirth, Niklaus. “Algoritmos e Estruturas de Dados”, Ed. LTC, 1986.