ÁRVORES AVL

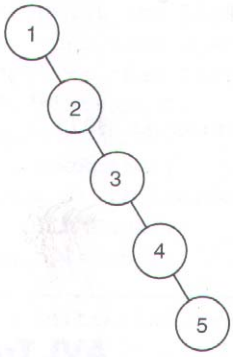
As árvores binárias de pesquisa são projectadas para um acesso rápido à informação. Idealmente a árvore deve ser razoavelmente equilibrada e a sua altura será dada (no caso de estar completa), como já vimos anteriormente por $h=\log_2 (n+1) – 1$, isto é de $O(\log_2 n)$, sendo n o número total de elementos da árvore. Contudo, com alguns dados, a árvore binária de pesquisa pode degenerar, tendo no pior caso altura n -1, tornado-se o acesso muito mais lento, O(n). É o que acontece no caso em que a construção da árvore é feita por inserções sucessivas e os valores são dados com chaves ordenadas (crescentes ou decrescentes).

Assim, vamos analisar um tipo de estrutura que nos dá o poder das árvores binárias de pesquisa, sem ocorrerem as condições do pior caso que esta última apresenta. Estudaremos então as árvores AVL, em que cada nó está equilibrado em altura, significando isto que, em cada nó, a diferença de alturas entre as subárvores esquerda e direita é no máximo 1. Uma outra definição para árvores AVL será aquela em que para qualquer nó P se verifica que o módulo do factor de equilíbrio é sempre < ou = 1. Entende-se por factor de equilíbrio de um nó P a diferença da altura da subárvore esquerda de P e altura da subárvore direita de P.

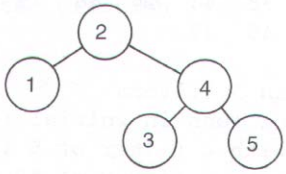
Factor de Equilíbrio(P)= altura(filho_ direito(P))-altura(filho_esquerdo(P))

As árvores abaixo representadas evidenciam o caso da introdução dos valores 1,2,3,4,5 no caso de uma árvore binária de pesquisa e no caso de uma árvore AVL .

Árvore Binária de Pesquisa

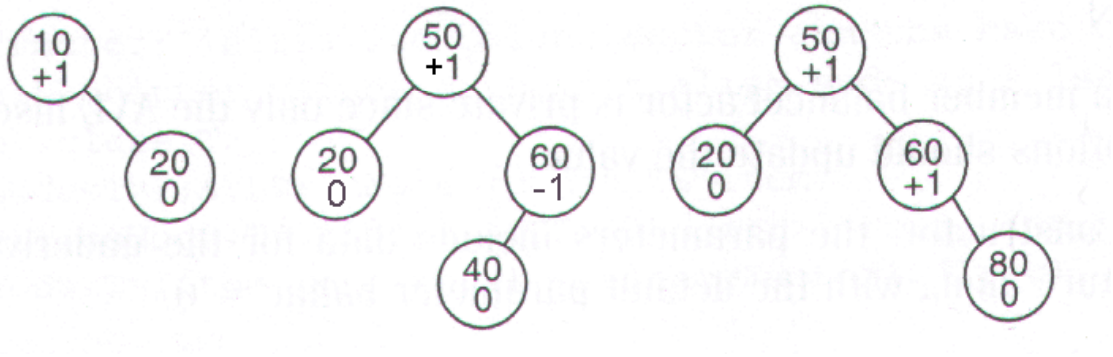


Árvore AVL



Se o factor de equilíbrio num nó é negativo, significa que a altura da subárvore esquerda é maior (em 1) do que a altura da subárvore direita, diremos então que o nó é **pesado à esquerda**.
Se o factor de equilíbrio num nó é positivo, significa que a altura da subárvore direita é maior (em 1) do que a altura da subárvore esquerda, diremos então que o nó é **pesado à direita**.
Se o factor de equilíbrio num nó é nulo, significa que a altura da subárvore esquerda é igual à altura da subárvore direita, diremos então que o nó é **equilibrado**

A seguir desenham-se árvores exemplo, em que a cada nó se adiciona o respectivo factor de equilíbrio.



Assim na definição da classe Nodo, teremos que considerar, além dos campos conteúdo e os apontadores para as subárvores esquerda e direita, mais um campo designado por factor_equilibrio que representará o estado de equilíbrio de cada nó.

Uma árvore AVL terá uma estrutura semelhante a uma árvore binária de pesquisa com a condição de que a árvore de pesquisa mantém-se equilibrada em altura depois de cada operação de inserção e eliminação.

Algoritmo de Inserção
O algoritmo de inserção é semelhante ao das árvores binárias de pesquisa, recursivamente descemos ora para a direita ora para a esquerda até atingirmos uma subárvore nula e então nessa posição colocaremos a folha com o elemento a inserir.

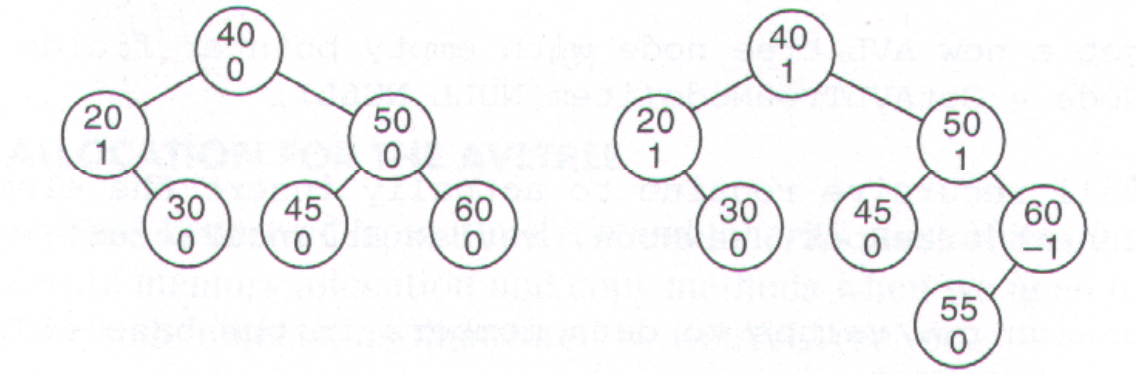
Nesse processo de descida na árvore, definimos um caminho desde a raiz até à posição de inserção do novo nó. Uma vez que o processo é recursivo, temos acesso aos nós por ordem inversa ao caminho percorrido e podemos actualizar o factor de equilíbrio num nó pai depois de termos estudado o efeito de juntar um novo nó numa das subárvores.

Em cada nó do caminho percorrido determinamos se é ou não necessário uma actualização e somos então confrontados com 3 situações.
Em dois dos casos, o novo nó mantem o equilíbrio da árvore e não é necessário a reestruturação da subárvore, só o factor de equilíbrio exige actualização.
O terceiro caso desiquilibra a árvore e é necessário realizar uma rotação simples ou dupla dos nós para fazer o reequilíbrio da árvore.

CASO 1: um nó no caminho percorrido tem facto de equilíbrio 0, depois de juntar o novo nó fica com factor de equilíbrio 1 ou –1dependendo em qual das subárvores juntamos o novo elemento. Consideremos o exemplo abaixo,em que juntamos o nó com valor 55 .

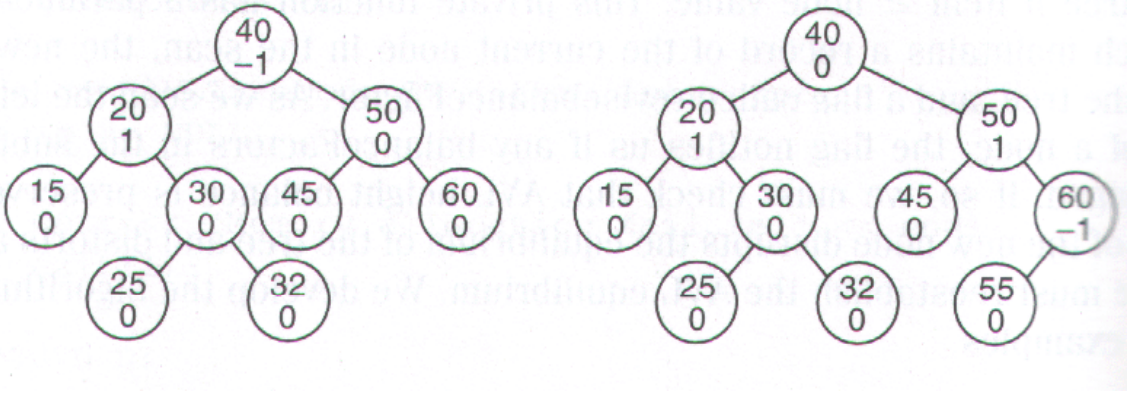
Todos os nós no caminho da descida estavam equilibrados, depois de juntar 55 os factores de equilíbrio foram alterados.

Antes de juntar o nó 55 Depois de juntar o nó 55



CASO 2: Um nó no caminho é pesado à direita ou à esquerda e o novo nó é colocado na outra subárvore(a menos pesada), nesse caso o nó torna-se equilibrado

Antes de juntar o nó 55 Depois de juntar o nó 55



Caso 3: Um nó no caminho é pesado à direita ou à esquerda e o novo elemento é colocado na mesma subárvore, na mais pesada. O nó resultante viola a condição de árvore AVL, uma vez que o factor de equilíbrio ficaria fora da gama de valores –1...1. O algoritmo obriga-nos a rodar os nós para estabelecer o equilíbrio da altura.
Vamos exemplificar e para tal assumimos que a árvore se torna desiquilibrada para a esquerda e nós ajustamos o equilíbrio chamando funções de rotação à direita. Os casos são simétricos quando a árvore se torna desiquilibrada à direita.
Vamos então considerar que o nó da árvore AVL tem a seguinte configuração:

.

esq	conteudo	Factor_equilibrio	dir

O algoritmo que conduz à inserção de um elemento pode ser definido do seguinte modo:

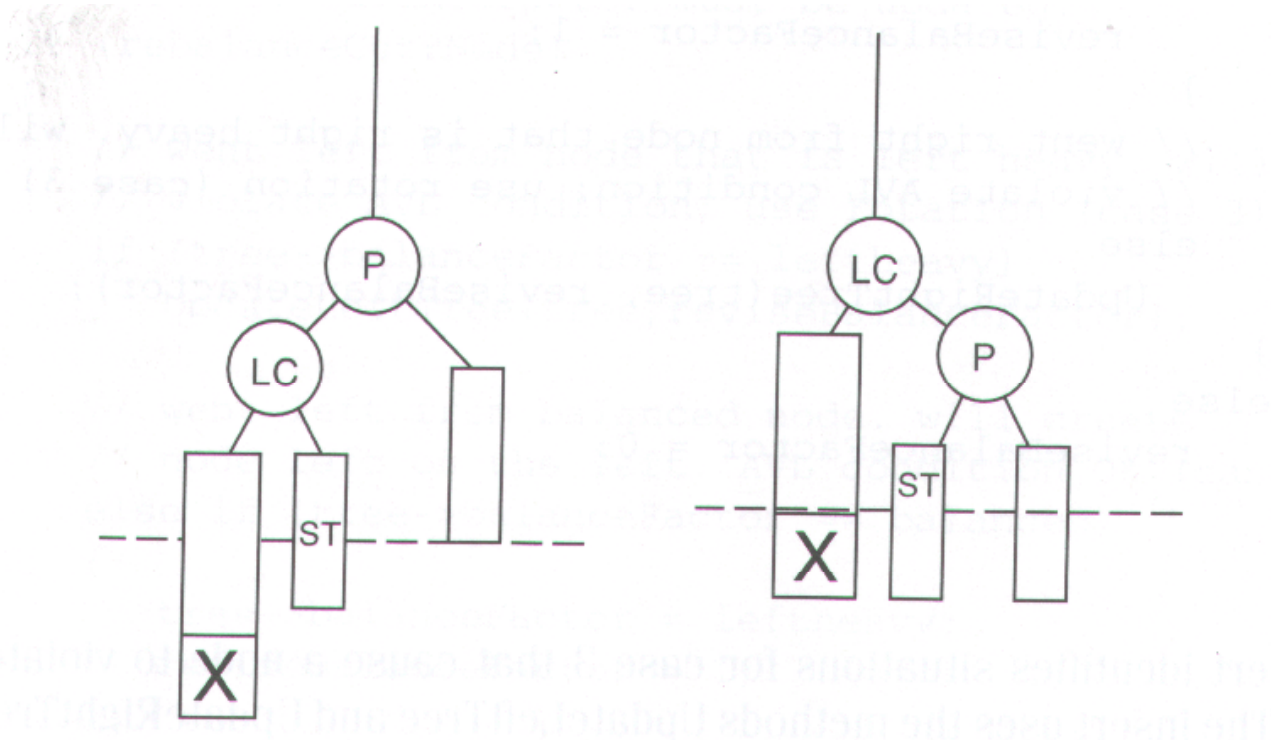
```
insere( item)
Inicio
rz=raiz
factor_rev=0
novo_no=new AVLNodo(item,0,Nulo,Nulo)   // criação do nó
Se raiz==Nulo
        Entao raiz=novo_no
        Senão AVLinsere(rz, novo_no, factor_rev)
                        // rz e factor_rev são passados por referência
Fse
Fim


AVLinsere(rz,novo_no,factor_rev)
Inicio
ReeqNoActual=0    // indica se ocorreu alteração no factor de equilíbrio do nó

Se rz==Nulo
        Então rz=novo_no
                rz->factor_equilibrio=0
                factor_rev=1
        Senão Se novo_no->conteudo < rz->conteudo
                Entao // O elemento será inserido para a esquerda
                        AVLinsere(rz->esq, novo_no, reeqNoActual)
                        Se reeqNoActual !=0
                                Então//há que actualizar o factor_equilíbrio
                                        Se rz->factor_equilibrio==pesado_esq
                                                Então // CASO 3
                                                        actual_esq(rz, factor_rev)
                                                Senão Se rz>factor_equilibrio==equilibrado
                                                                Então // CASO 1
                                                                rz->factor_equilibrio=pesado_esq
                                                                factor_rev=1
                                                                Senão // CASO 2
                                                                rz>factor_equilibrio=equilibrado
                                                                factor_rev=0
                                                        Fse
                                        Fse

                                Senão  factor_rev=0
                        Fse
```

```
        Senão // O elemento será inserido para a direita
                AVLinsere(rz->dir, novo_no, reeqNoActual)
                Se reeqNoActual !=0

                        Então// há que actualizar o factor_equilíbrio
                                Se rz->factor_equilibrio==pesado_esq

                                        Então // CASO 2
                                                rz>factor_equilibrio=equilibrado
                                                factor_rev=0


                                        Senão Se rz>factor_equilibrio==equilibrado

                                                        Então // CASO 1
                                                        rz->factor_equilibrio=pesado_dir
                                                        factor_rev=1

                                                        Senão // CASO 3
                                                                actual_dir(rz, factor_rev)

                                                Fse
                                        Fse

                        Senão factor_rev=0
                Fse
        Fse
Fse
FIM
```

Vejamos agora o algoritmo que faz a actualização da subárvore esquerda, (a da direita é idêntico) há que fazer as rotações dos nós de acordo com os casos de modo a reequilibrar a árvore. Depois de feito o reequilíbrio a flag factor_rev é posta a 0 para notificar o pai de que a subárvore está equilibrada. Os parâmetros são passados por referência.

```
actual_esq (rz, factor_ver)
Inicio
filho=rz->esq
Se filho->factor_equilibrio== pesado_esq
        Então rotaçao_simples_dir ( rz )
                factor_rev=0

        Senão Se filho->factor_equilibrio==pesado_dir
                        Então rotaçao_dupla_dir ( rz )
                                factor_rev=0
                Fse
Fse
Fim
```

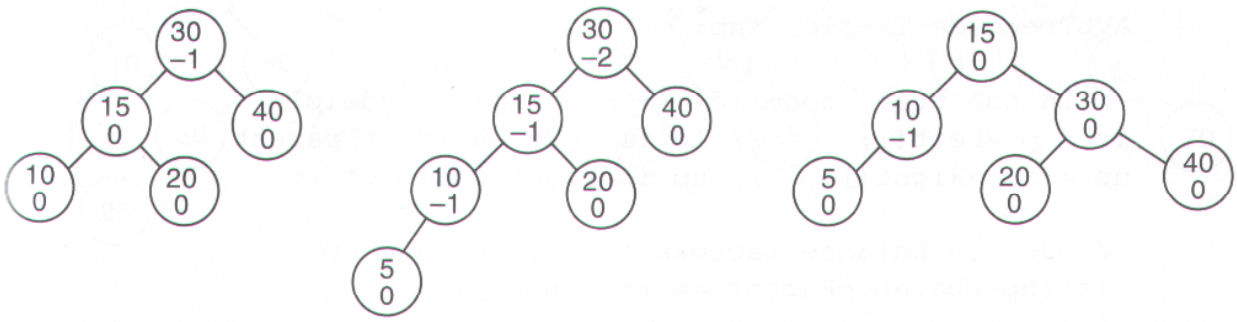
Vejamos agora como se processam as rotações.
1-O caso de **rotação simples** verifica-se então quando o nó pai (P) era pesado à esquerda e o nó filho (LC) depois da inserção do novo nó X, tornou-se pesado também à esquerda. Rodamos os nós de modo que o nó filho substitua o pai e este por sua vez torna-se o seu filho direito . O ajuste de apontadores é bem visível na figura abaixo.



Tudo se passaria de modo idêntico se actualizassemos a subárvore direita, em que nesse caso o nó pai estava pesado à direita e após a inserção do novo nó X, o filho direito ficasse também desiquilibrado à direita. Nesse caso a rotação ter-se-ia que se verificar para o lado esquerdo. As alterações a fazer seria substituir esquerdo por direito e direito por esquerdo

Vejamos agora um exemplo concreto, em que se insere o valor 5

Árvore inicial | Factores de equilibrio depois de juntar 5 | Árvore após rotação



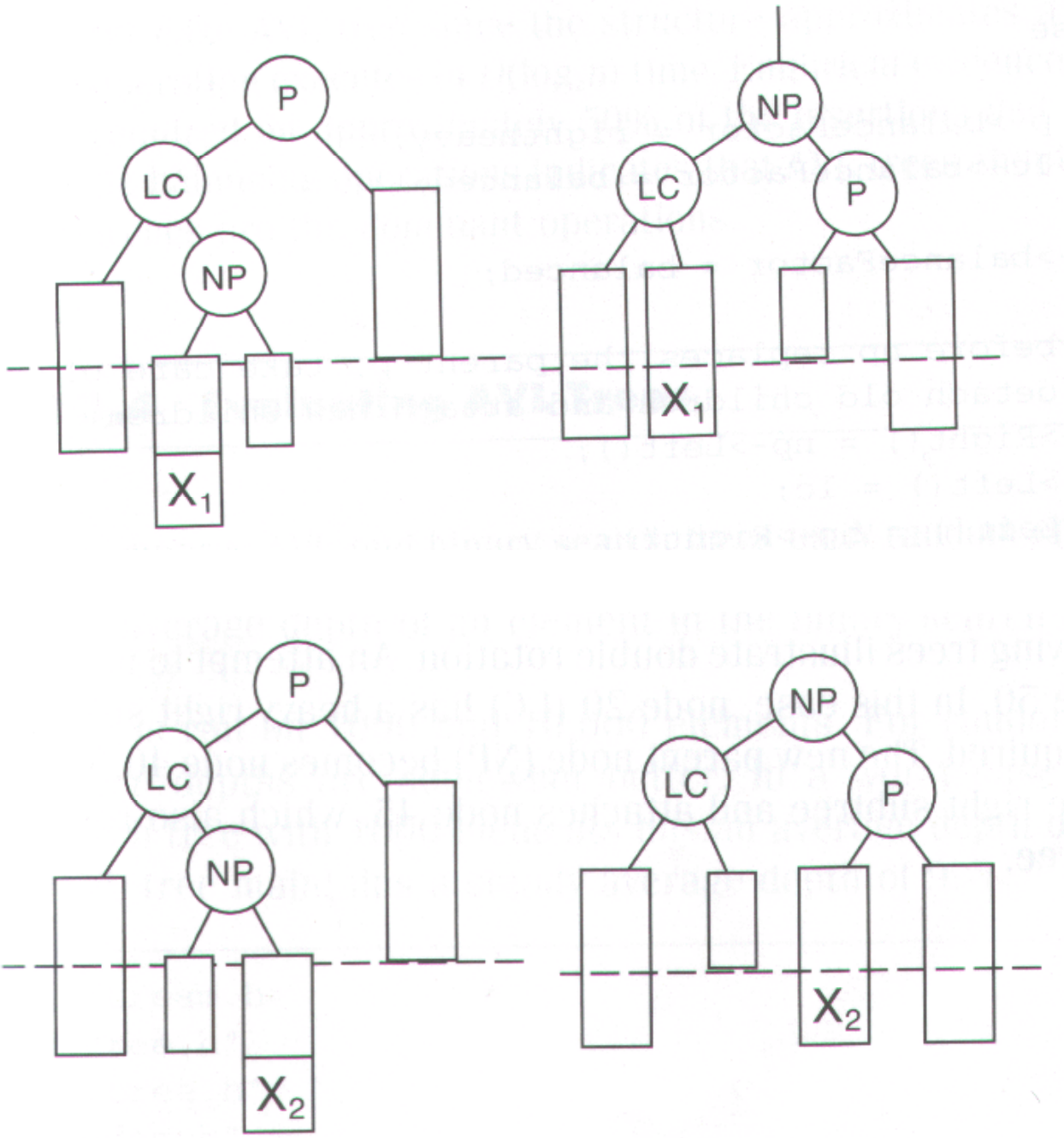
Quanto ao algoritmo referente à rotação simples à direita em que o parâmetro é passado por referência, traduzir-se-á no seguinte:

rotaçao_simples_dir (rz)
Inicio
filho=rz->esq
rz->factor_equilibrio=equilibrado
filho->factor_equilibrio=equilibrado

//ajustar apontadores
rz->esq=filho->dir
filho->dir=rz
rz=filho
Fim

2- O caso de **rotação dupla** à direita, verifica-se então quando o nó está pesado à esquerda e o seu filho esquerdo depois da inserção do novo elemento se torna pesado à direita. Ter-se-á que proceder à actualização dos factores de equilíbrio e para isso há que verificar se o factor de equilíbrio do nó neto (filho direito do filho esquerdo do nó em questão) , é equilibrado, pesado à direita ou pesado à esquerda para fazer a respectiva actualização sendo feito seguidamente o ajuste de apontadores.

As figuras abaixo evidenciam o que se passa, em que a inserção do novo nó se deu na subárvore esquerda do neto ou na subárvore direita do neto. Em qualquer dos casos o neto substitui o avô, ficando com o avô à direita e o pai à esquerda. A subárvore direita do neto passará a ser a esquerda do avô e a esquerda do neto será a direita do pai.

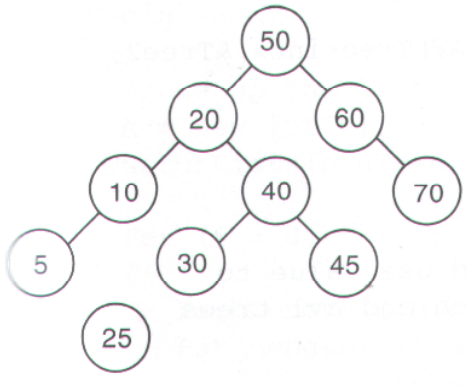


A seguir vamos descrever o algoritmo que executa esta dupla rotação à direita. De modo totalmente idêntico proceder-se-ia à rotação dupla para a esquerda , método esse que seria invocado no método actualizar subárvore direita que nós não descreveremos uma vez que seria igual, ao já descrito, actualizar subárvore esquerda, trocando unicamente os apontadores direitos por esquerdos e vice versa.
O parâmetro , que é um apontador para nó, é passado por referência.

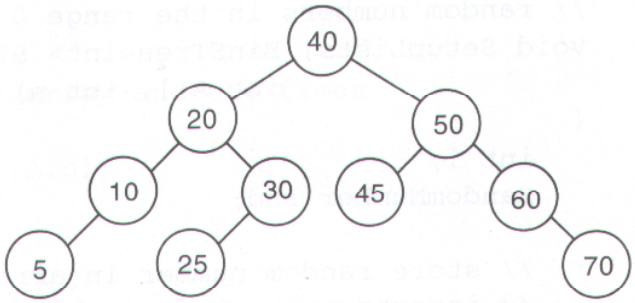
```
dupla_rotaçao_dir ( rz )
Inicio
filho=rz->esp
neto=filho->dir
Se neto->factor_equilibrio==pesado_dir
        Então rz->factor_equilibrio= equilibrado
                filho->factor_equilibrio=pesado_dir

        Senão Se neto->factor_equilibrio==equilibrado
                        Então rz->factor_equilibrio= equilibrado
                                filho->factor_equilibrio=equilibrado

                        Senão rz->factor_equilibrio=pesado_dir
                                filho->factor _equilibrio=equilibrado
                Fse
Fse
neto->factor_equilibrio=equilibrado
filho->dir=neto->esq
neto->esq=filho
rz->esq=neto->dir
neto->dir=rz
rz=neto
Fim
```

Seguidamente apresenta-se um exemplo com valores para melhor se compreender o foi descrito.
Neste exemplo, depois de inserir o nó com valor 25 , houve necessidade de fazer uma dupla rotação para a direita.

Antes de inserir 25



Depois de inserir 25 e de rodar



Há autores que afirmam que a experiência mostra que 50% das inserções e eliminações obrigam a rotações. Evidentemente que as rotações conduzem a uma perda de eficiência nos algoritmos de inserção e eliminação. Assim, a utilização desta estrutura depende das

aplicações e portanto aplicações onde a pesquisa seja a operação dominante deve usar-se árvores AVL, ,já que esta operação garante uma ordem de complexidade $\log_2 n$. Em aplicações em que as inserções ou eliminações são as operações mais frequentes poderemos então usar árvores binárias de pesquisa.