MC3305
Algoritmos e Estruturas de Dados II

Aula 11 –
Árvores Adelson-Velskii e Landis

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

2Q-2014

AVL

G.M. Adelson-Velskii y E.M. Landis
"**An algorithm for the organization of information**".
Proceedings of the USSR Academy of Sciences, vol. 146, pp. 263–266, **1962**

AN ALGORITHM FOR THE ORGANIZATION OF INFORMATION

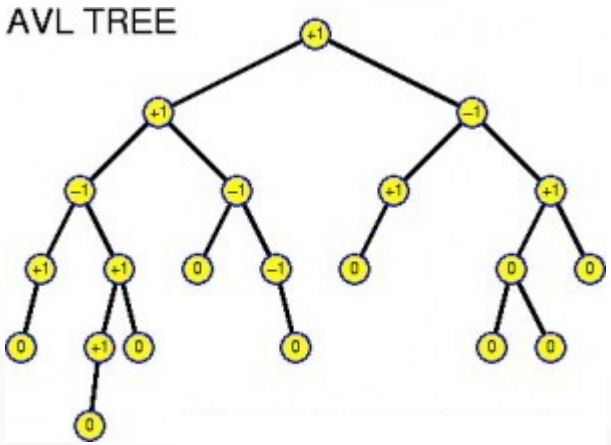
G. M. ADEL'SON-VEL'SKIĬ AND E. M. LANDIS

In the present article we discuss the organization of information contained in the cells of an automatic calculating machine. A three-address machine will be used for this study.

Statement of the problem. The information enters a machine in sequence from a certain reserve. The information element is contained in a group of cells which are arranged one after the other. A certain number (the information estimate), which is different for different elements, is contained in the information element. The information must be organized in the memory of the machine in such a way that at any moment a very large number of operations is not required to scan the information with the given evaluation and to record the new information element.

An algorithm is proposed in which both the search and the recording are carried out in $C \lg N$ operations, where N is the number of information elements which have entered at a given moment.

A part of the memory of the machine is set aside to store the incoming information. The information elements are arranged there in their order of entry. Moreover, in another part of the memory a "reference board" [1] is formed, each cell of which corresponds to one of the information elements.



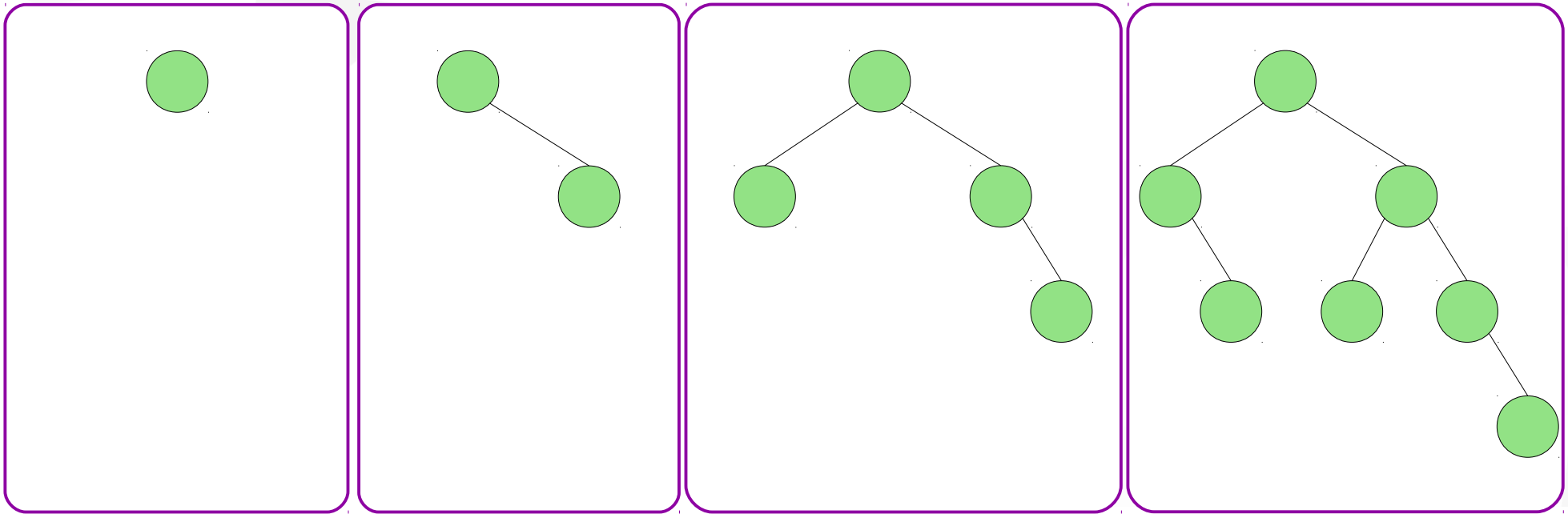
AVL foi a primeira estrutura (conhecida) de árvore de altura balanceada/equilibrada.

AVL é uma árvores balanceada?

Balanceamento de árvores AVL

- Uma árvore AVL de altura h é balanceada se h = O(log(n))
- Outra forma de pensar: **Dada uma árvore AVL de altura h, qual seria o valor mínimo possivel para n?**
- Seja T_h uma árvore AVL com altura h e número mínimo de nós.

Nesta definição → h=4



T1 T2 T3 T4

Balanceamento de árvores AVL

- Basta calcular um limite inferior do número de nós de Th. Seja |Th| o número de nós de Th.

$$\begin{cases} |T_h| = 0 & , \text{para } h = 0 \\ |T_h| = 1 & , \text{para } h = 1 \\ |T_h| = 1 + |T_{h-1}| + |T_{h-2}| & , \text{para } h > 1 \end{cases}$$

h	$\lvert T_h \rvert$
0	0
1	1
2	2
3	4
4	7
5	12
6	20
7	33
8	54
9	88
10	143

Analogia com a sequência de Fibonacci

$$\begin{cases} f_h = 0 & , \text{para } h = 0 \\ f_h = 1 & , \text{para } h = 1 \\ f_h = f_{h-1} + f_{h-2} & , \text{para } h > 1 \end{cases}$$

$$f_h = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^h - \left(\frac{1-\sqrt{5}}{2} \right)^h \right]$$

$$|T_h| \geq f_h$$

AVL

$|T_h| \geq f_h$

$$f_h = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^h - \left(\frac{1-\sqrt{5}}{2}\right)^h\right]$$

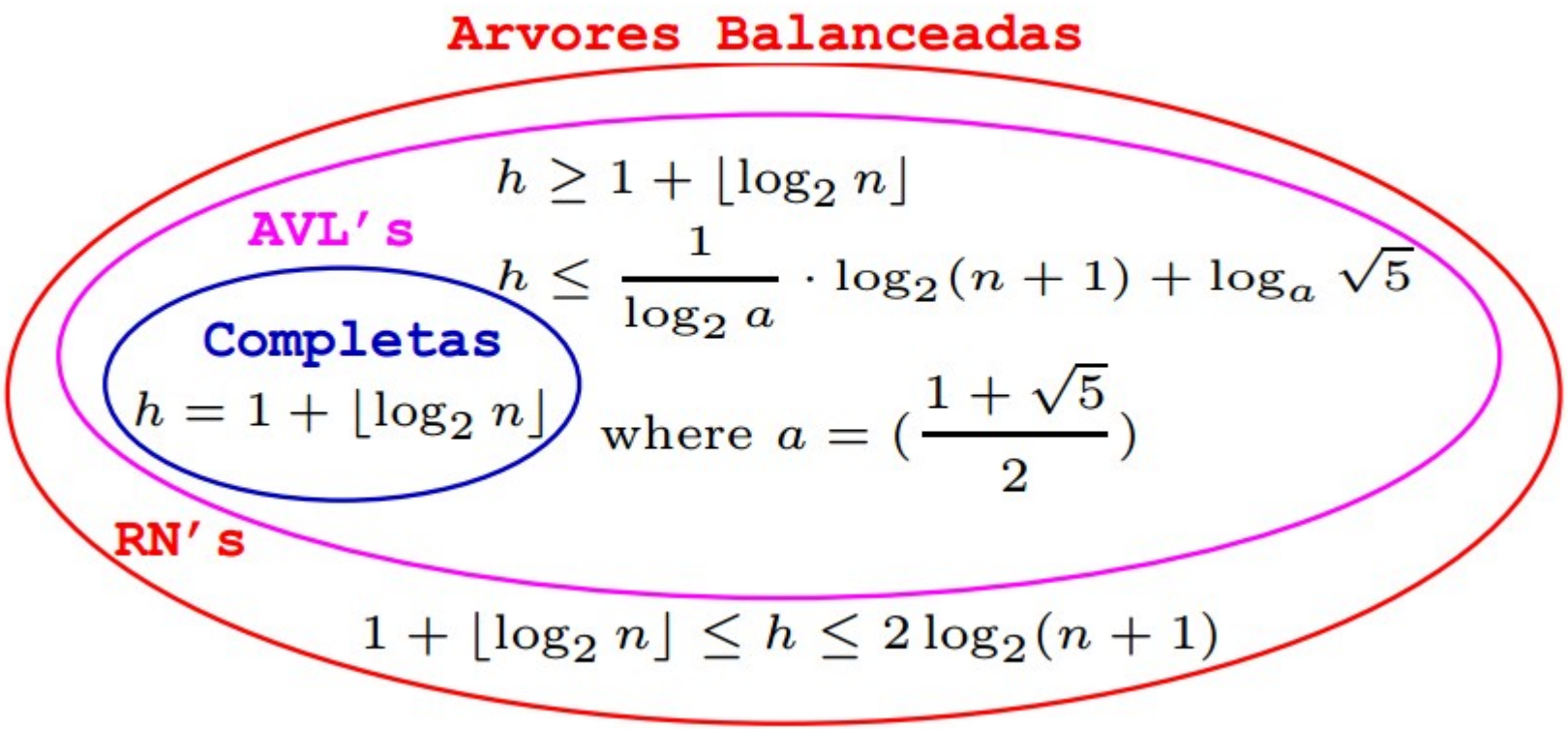
Como $\frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^h < 1$ Temos $|T_h| > \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^h - 1$

$$h < \frac{1}{\log_2 a}\log_2(|T_h|+1) + \log_a(\sqrt{5}) \qquad a = \frac{1+\sqrt{5}}{2}$$

$$h = O(\log n)$$

AVL é uma árvore balanceada!

AVL



A **árvore balanceada nunca será 45% mais alta que a correspondente árvore perfeitamente balanceada**, independentemente do número de nós existentes.

AVL

n	Completa = C	AVL = A	A/C
10	4.322	6.655	1.540
200	8.644	12.693	1.468
3000	12.551	18.311	1.459
40000	16.288	23.693	1.455
500000	19.932	28.942	1.452
6000000	23.517	34.106	1.450
70000000	27.061	39.211	1.449
800000000	30.575	44.273	1.448
9000000000	34.067	49.303	1.447
100000000000	37.541	54.307	1.447
1100000000000	41.001	59.290	1.446
12000000000000	44.448	64.256	1.446
130000000000000	47.886	69.207	1.445
1400000000000000	51.314	74.146	1.445
15000000000000000	54.736	79.074	1.445
160000000000000000	58.151	83.994	1.444
1700000000000000000	61.560	88.904	1.444
18000000000000000000	64.965	93.808	1.444
190000000000000000000	68.365	98.706	1.444
2000000000000000000000	71.760	103.597	1.444

Árvores balanceadas

- Árvores balanceadas são muito utilizadas em problemas reais:
  - JAVA: TreeMap, TreeSet.
  - C++: Map, Set do STL.
- Custo de busca, inserção, remoção da árvore AVL: O(log n)

Considerações de avaliações empíricas

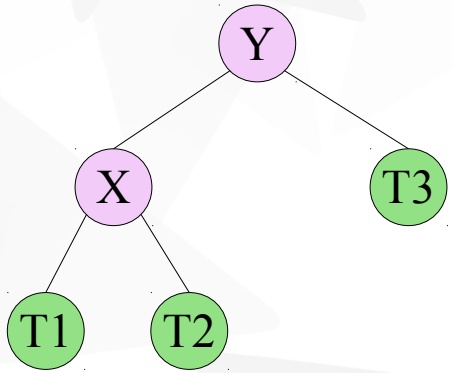
- Há um custo adicional para manter uma árvore balanceada, mesmo assim garantindo O(log2 n), mesmo no pior caso, para todas as operações.

- Em testes empíricos:
  - **Uma rotação é necessária a cada duas inserções.**
  - Uma rotação é necessária a cada cinco remoções.

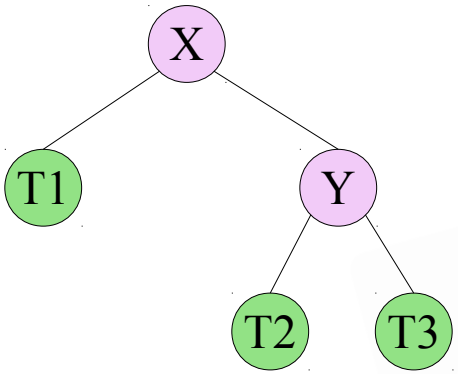
- A remoção em árvore balanceada é tão simples (ou tão complexa) quanto a inserção.

Sobre as rotações

Rotação: Esquerda & Direita

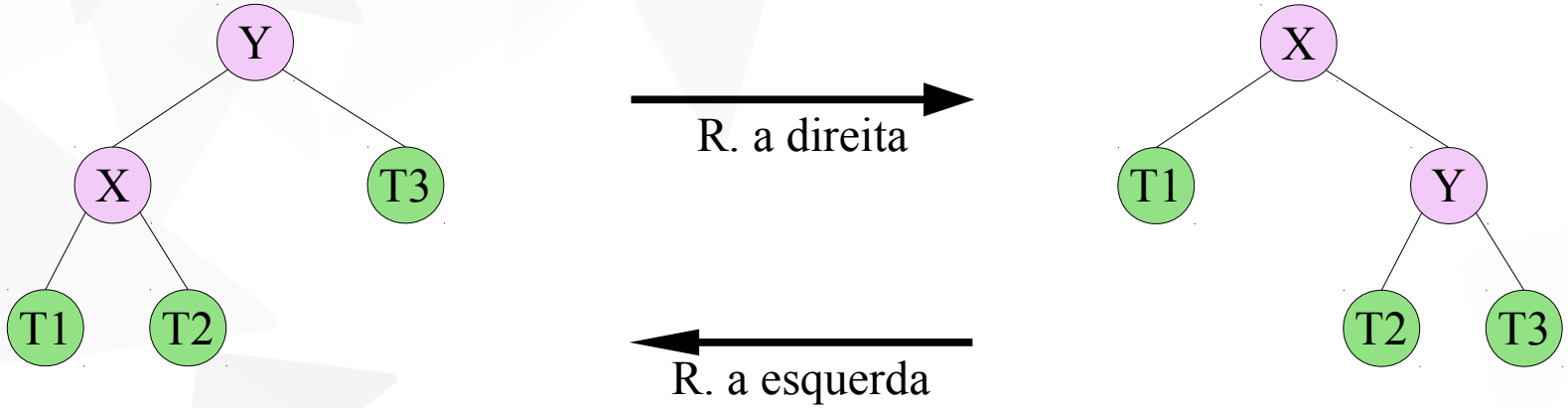


Varredura e-r-d: T1, X, T2, Y, T3



Varredura e-r-d: T1, X, T2, Y, T3

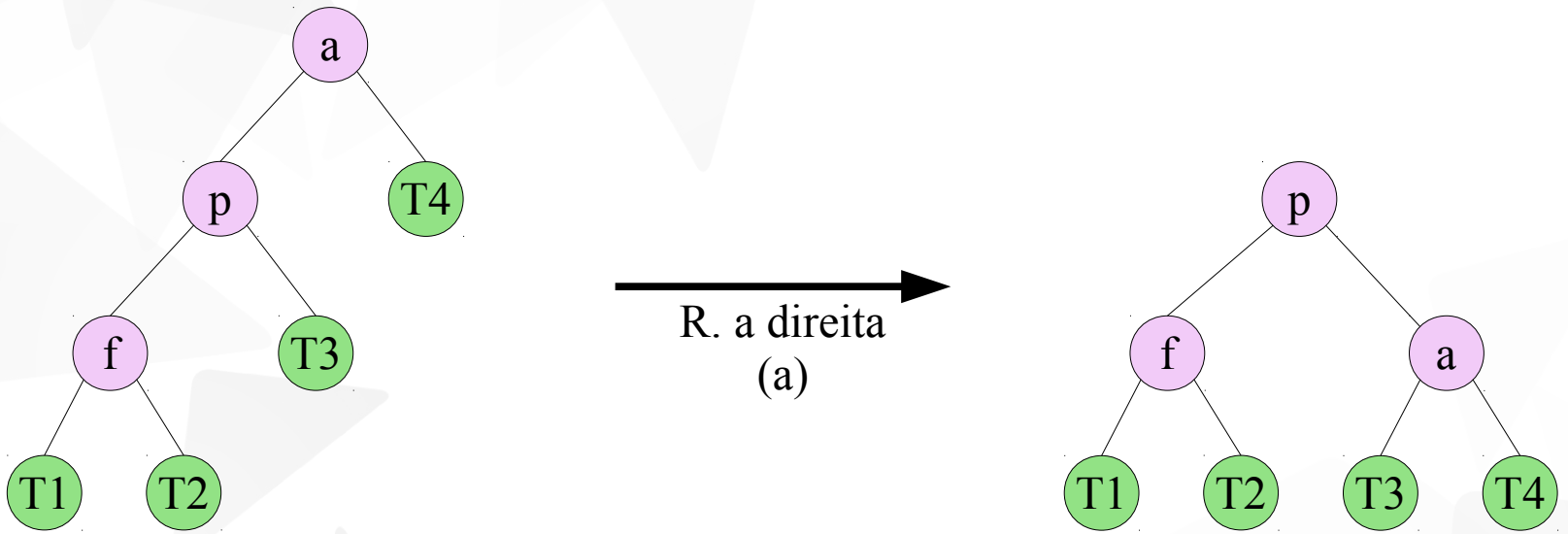
Rotação: Esquerda & Direita



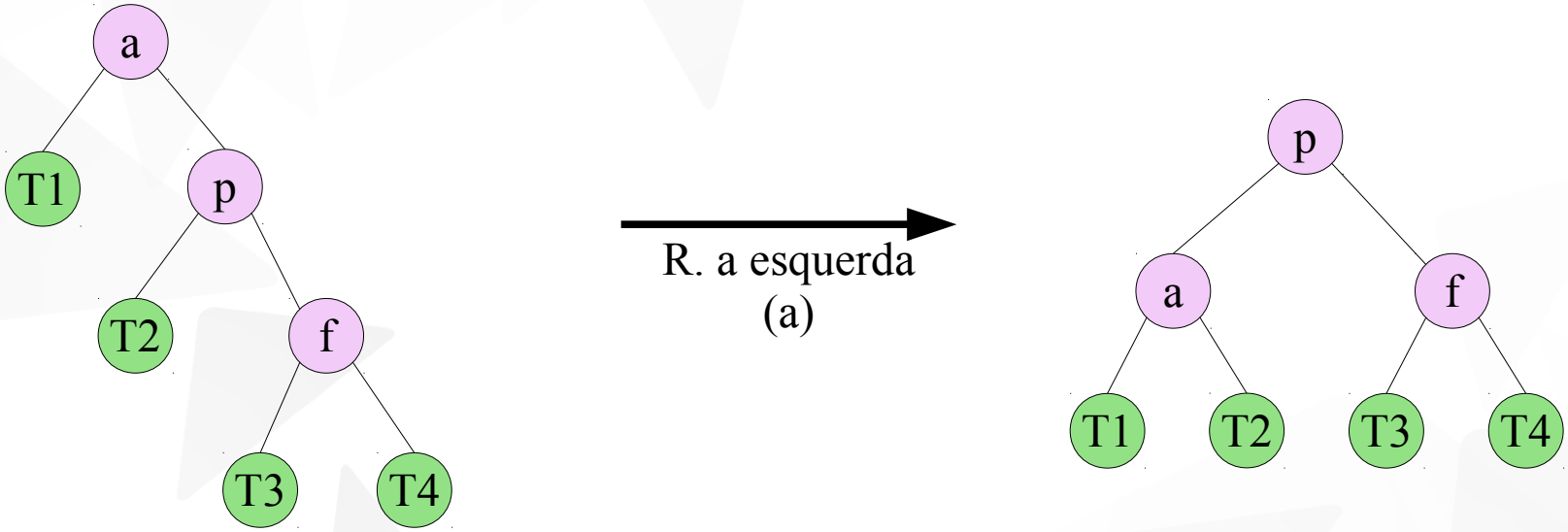
Varredura e-r-d: T1, X, T2, Y, T3

Varredura e-r-d: T1, X, T2, Y, T3

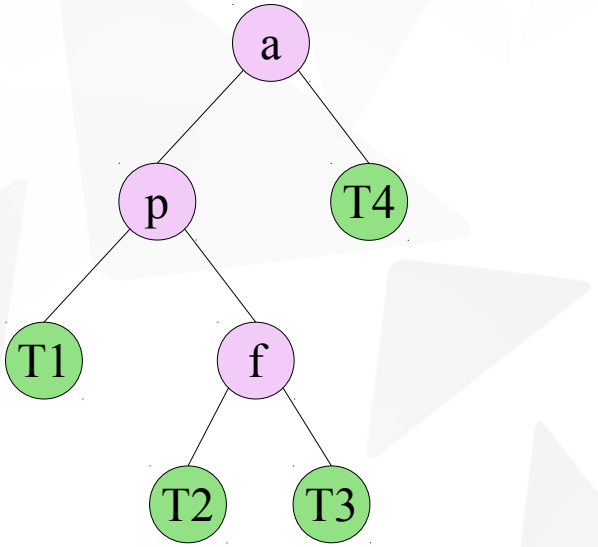
Rotação: LL

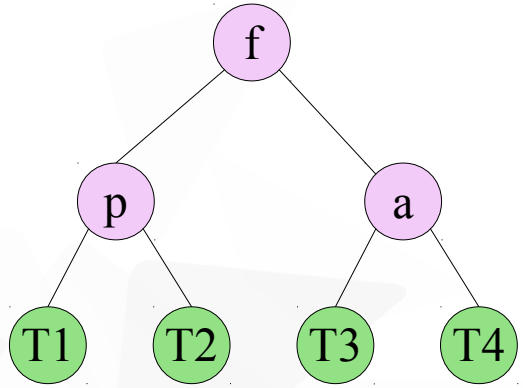


Rotação: RR

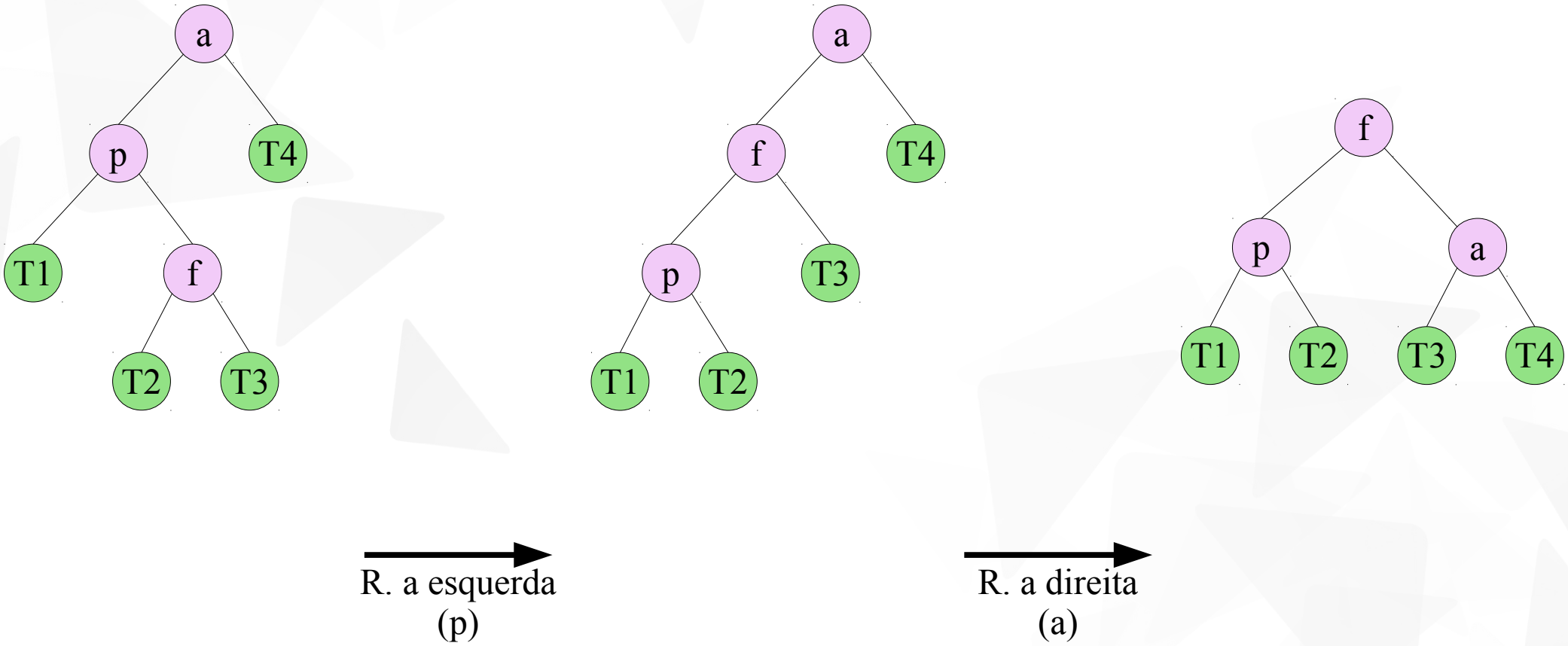


Rotação: LR

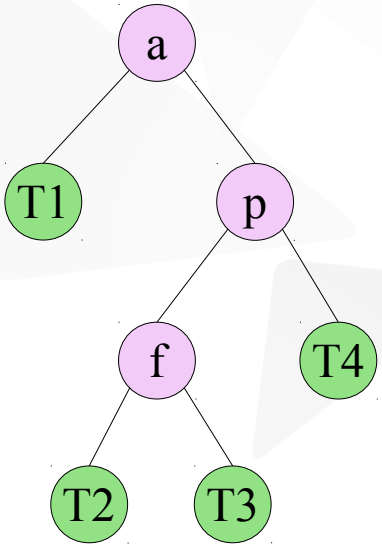


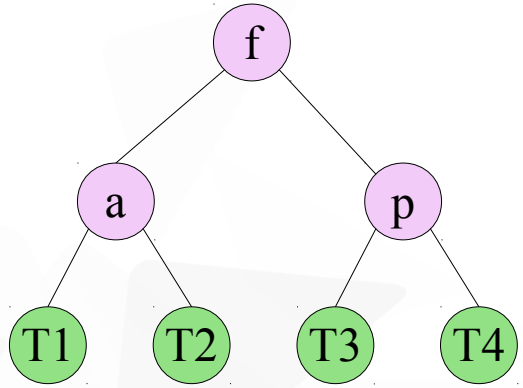


Rotação: LR

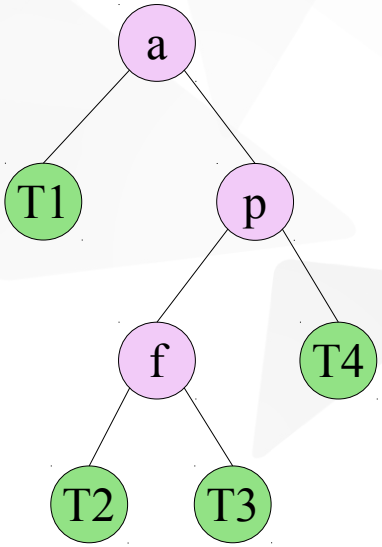

a
p
T4
T1
f
T2
T3
a
f
T4
p
T3
T1
T2
f
p
a
T1
T2
T3
T4
R. a esquerda
(p)
R. a direita
(a)

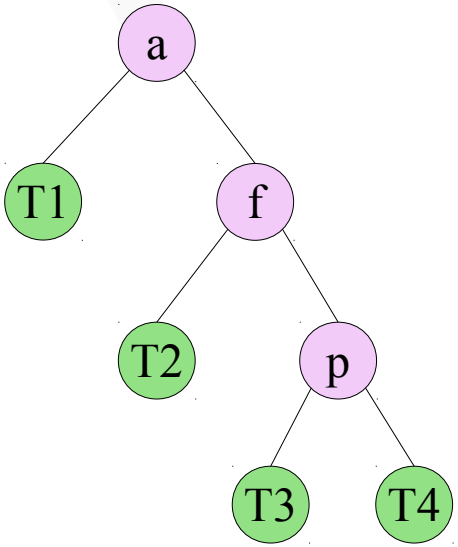
Rotação: RL

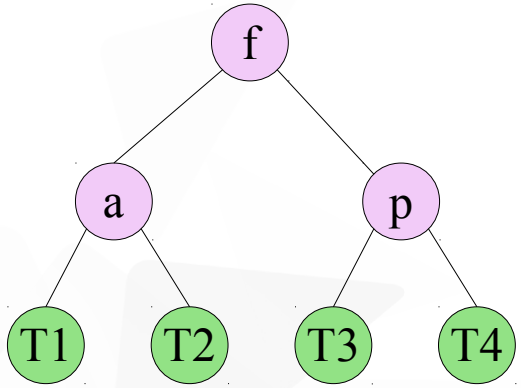




Rotação: RL



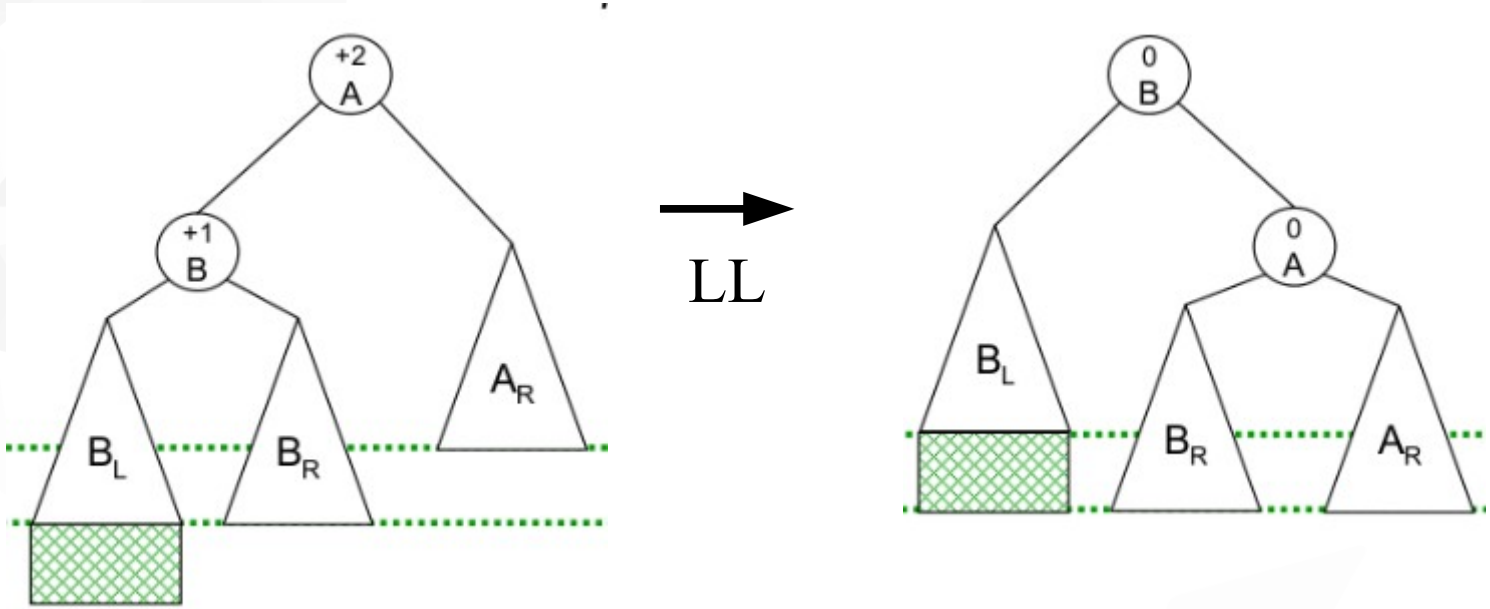


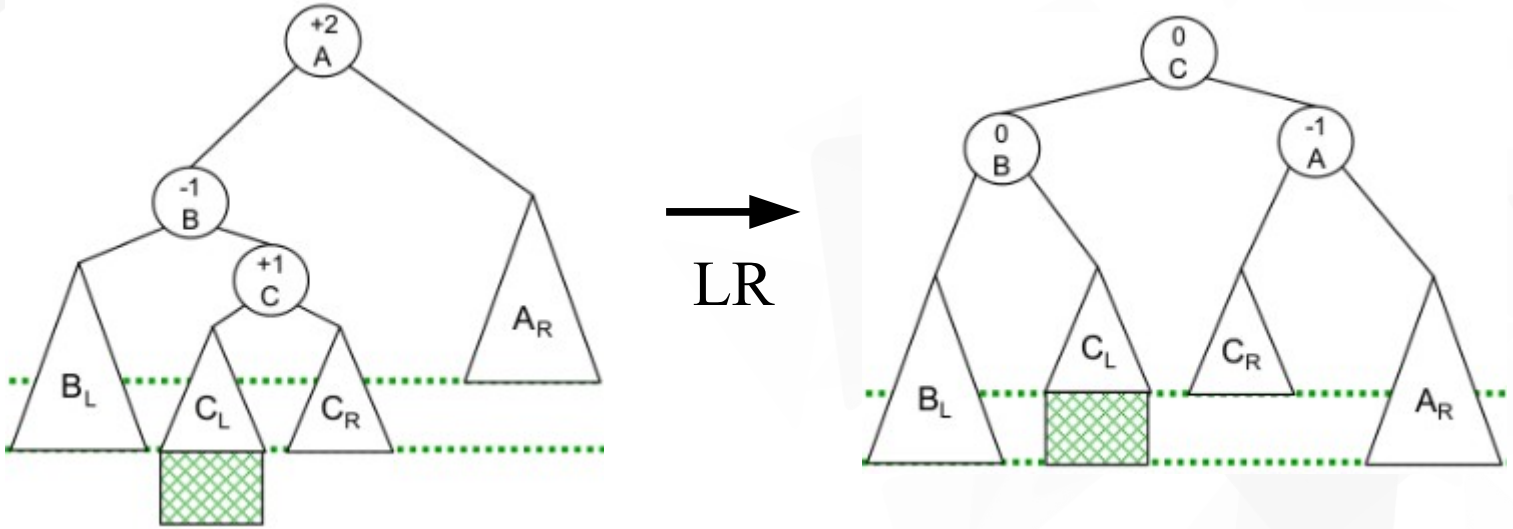


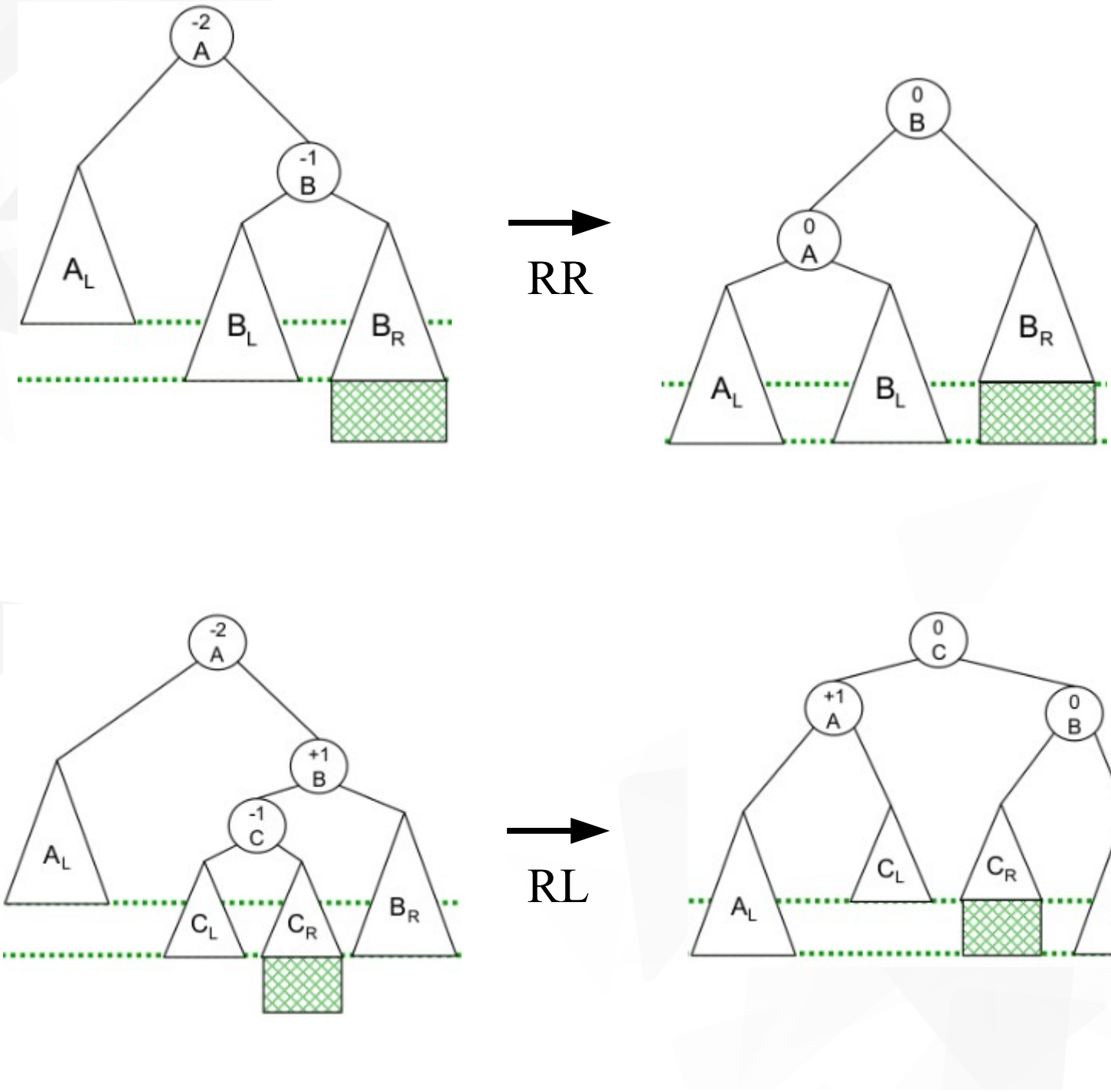
R. a direita
(p)

R. a esquerda
(a)

Como determinar o tipo de rotação?


+2
A
+1
B
B_L
B_R
A_R
LL
0
B
B_L
0
A
B_R
A_R


+2
A
-1
B
+1
C
B_L
C_L
C_R
A_R
LR
0
C
0
B
-1
A
B_L
C_L
C_R
A_R


-2
A
-1
B
A_L
B_L
B_R
RR
0
B
0
A
A_L
B_L
B_R
-2
A
+1
B
-1
C
A_L
C_L
C_R
B_R
RL
0
C
+1
A
0
B
A_L
C_L
C_R
B_R

Atividade prática

avl.c (tidia: aed2-11.tgz)

```c
struct cel {
    int chave;
    struct cel *esq;
    struct cel *dir;
    int altura;
};

typedef struct cel no;
```

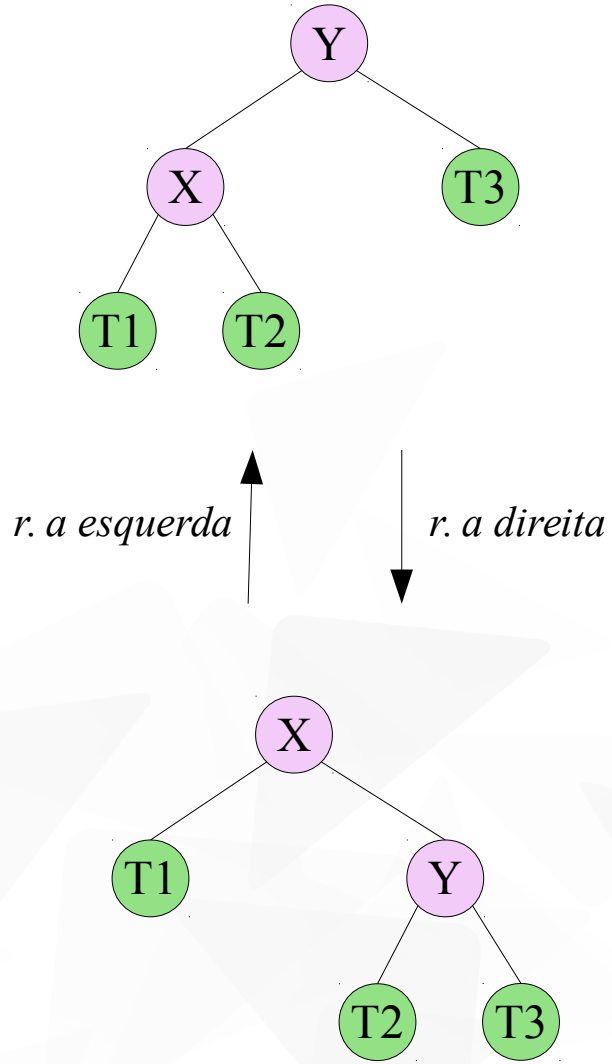
```c
int altura(no* r) {
    if (r==NULL)
        return 0;
    else
        return r->altura;
}
```

```c
no* criarNo(int k) {
    no* novoElemento = (no*)malloc(sizeof(no));
    novoElemento->chave = k;
    novoElemento->esq = novoElemento->dir = NULL;
    novoElemento->altura = 1;
    return novoElemento;
}
```

avl.c (tidia: aed2-11.tgz)

```
no* rotacaoADireita(no* y) {
    no* x  = y->esq;
    no* T2 = x->dir;

    // rotacao
    x->dir = y;
    y->esq = T2;

    // atualizacao de altura
    y->altura = 1 + max( altura(y->esq) , altura(y->dir) );
    x->altura = 1 + max( altura(x->esq) , altura(x->dir) );

    return x;
}
```

```
no* rotacaoAEsquerda(no* x) {
    no* y  = x->dir;
    no* T2 = y->esq;

    // rotacao
    y->esq = x;
    x->dir = T2;

    // atualizacao de altura
    x->altura = 1 + max( altura(x->esq) , altura(x->dir) );
    y->altura = 1 + max( altura(y->esq) , altura(y->dir) );

    return y;
}
```



avl.c (tidia: aed2-11.tgz)

```
int fatorDeBalanceamento(no* r) {
    if (r==NULL)
        return 0;
    else
        return altura(r->esq) - altura(r->dir);
}
```

avl.c (tidia: aed2-11.tgz)

```
no* inserirChave(no *r, int k) {

    // insercao na ABB -- versao recursiva
    if (r==NULL)
        return criarNo(k);

    if (k == r->chave)
        return r;
    else {
        if (k < r->chave)
            r->esq = inserirChave(r->esq, k);
        else
            r->dir = inserirChave(r->dir, k);
    }

    // atualizacao da altura do no antecessor
    r->altura = 1 + max( altura(r->esq), altura(r->dir) );

    // fator de balanceamento
    int fb = fatorDeBalanceamento(r);
```

avl.c (tidia: aed2-11.tgz)

```
    // ------------------------------------------------------------
    // se o no estiver desbalanceado, entao temos 4 alternativas

    // Caso LL:
    if ( fb>1  && k < r->esq->chave) {
        printf("\n* Rotacao LL");
        return rotacaoADireita(r);
    }

    // Caso RR:
    if ( fb<-1 && k > r->dir->chave) {
        printf("\n* Rotacao RR");
        return rotacaoAEsquerda(r);
    }

    // Caso LR:
```

TO-DO

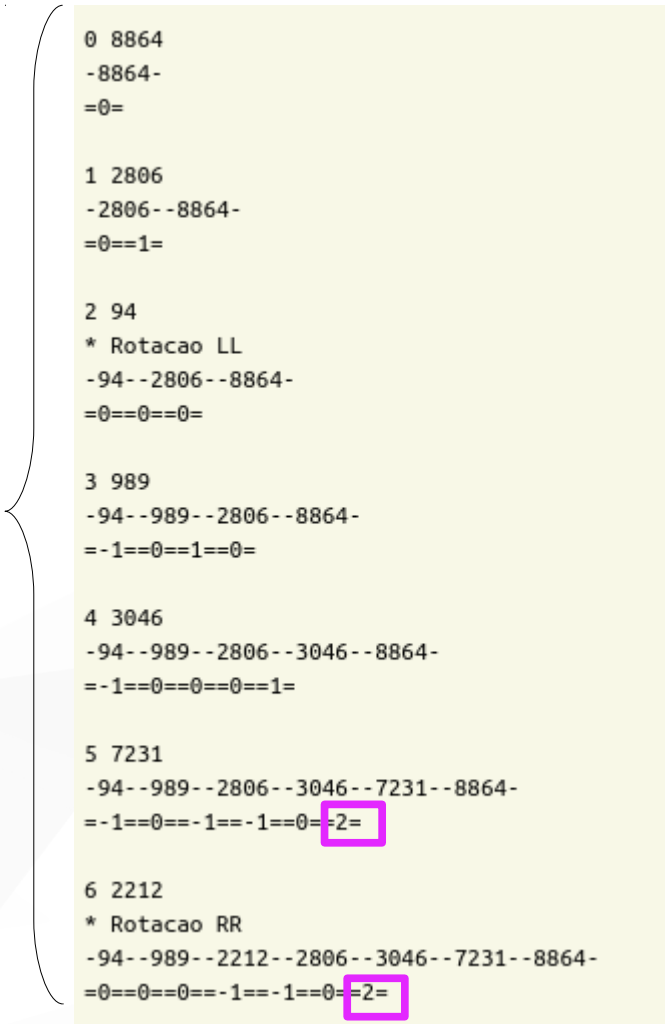
```
    // Caso RL:
```

TO-DO

```
    return r;
}
```

avl.c

```
$ gcc avl.c -o avl.exe

$ ./avl.exe 20 < vetor1.dat

$ ./avl.exe 20 < vetor2.dat

$ ./avl.exe 20 < vetor3.dat

$ shuf -i0-10000 -n20 | ./avl.exe 20



$ ./avl.exe 200 < vetor4.dat
```



avl.c

```
$ ./avl.exe 40000 < vetor4.dat
...
Altura da arvore:18



$ shuf -i0-1000000000 -n40000 | ./avl.exe 40000
```

n	Completa = C	AVL = A	A/C
10	4.322	6.655	1.540
200	8.644	12.693	1.468
3000	12.551	18.311	1.459
40000	16.288	23.693	1.455
500000	19.932	28.942	1.452
6000000	23.517	34.106	1.450
70000000	27.061	39.211	1.449
800000000	30.575	44.273	1.448
9000000000	34.067	49.303	1.447
100000000000	37.541	54.307	1.447
1100000000000	41.001	59.290	1.446
12000000000000	44.448	64.256	1.446
130000000000000	47.886	69.207	1.445
1400000000000000	51.314	74.146	1.445
15000000000000000	54.736	79.074	1.445
160000000000000000	58.151	83.994	1.444
1700000000000000000	61.560	88.904	1.444
18000000000000000000	64.965	93.808	1.444
190000000000000000000	68.365	98.706	1.444
2000000000000000000000	71.760	103.597	1.444