INF 1010
Estruturas de Dados Avançadas

Árvores AVL

voltando às árvores binárias de busca

- problemas com desbalanceamento!

diversas propostas de estruturas que se mantêm balanceadas

Balanceamento de Árvores Binárias de Busca

motivação:

busca em O(log(n))

estratégia:

diminuir a diferença de altura entre a sub-árvore à esquerda e a sub-árvore à direita de cada nó

árvores binárias de busca

tempo de acesso depende de altura da árvore

```
tdados* busca (Mapa* r, int c)
{
   if (r == NULL)
         return NULL;
   else if (c < r->chave)
         return busca (r->esq, c);
   else if (c > r->chave)
         return busca (r->dir, c);
   else return r->dados;
}
```

árvores binárias de busca

tempo de acesso depende de altura da árvore

```
tdados* busca (Mapa* r, int c)
{
   if (r == NULL)
         return NULL;
   else if (c < r->chave)
         return busca (r->esq, c);
   else if (c > r->chave)
         return busca (r->dir, c);
   else return r->dados;
}
```

quantas chamadas podem ocorrer no pior caso?

árvores binárias de busca

diversas propostas de estruturas que se mantêm balanceadas

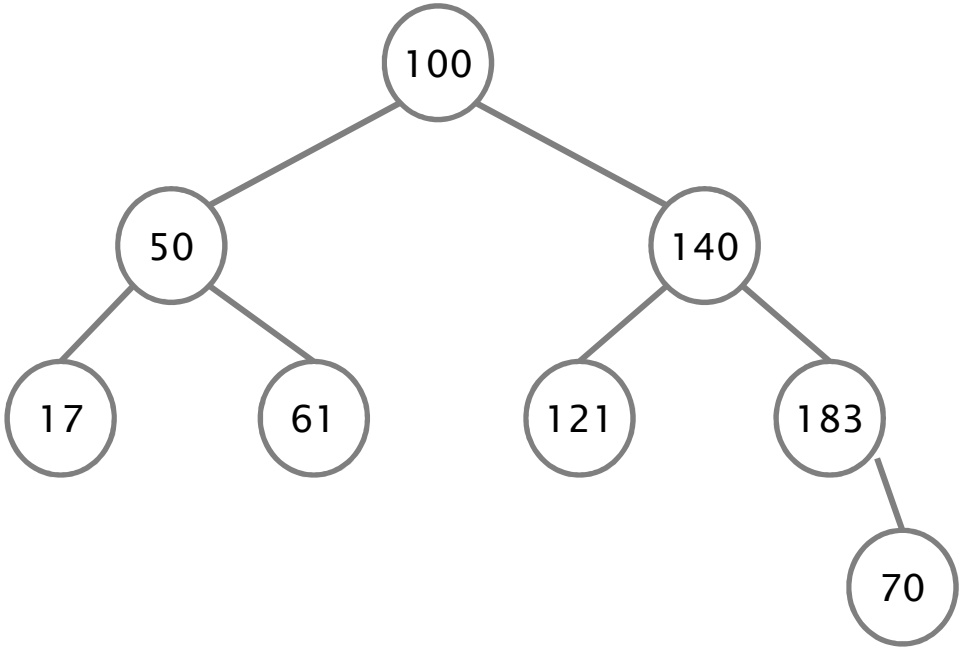
- árvores AVL
- árvores rubro-negras

Balanceamento de Árvores Binárias de Busca

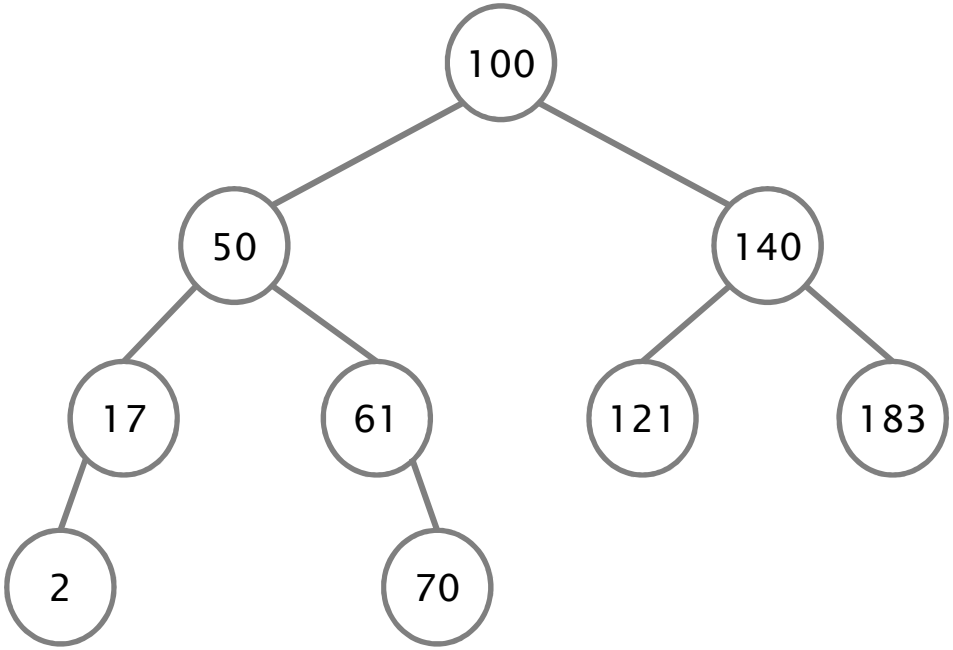
1962: AVLs
Adelson-Velsky e Landis

árvore binária de busca onde, em cada nó, a altura de sua sub-árvore à direita difere da altura da sub-árvore à esquerda de no máximo 1 unidade

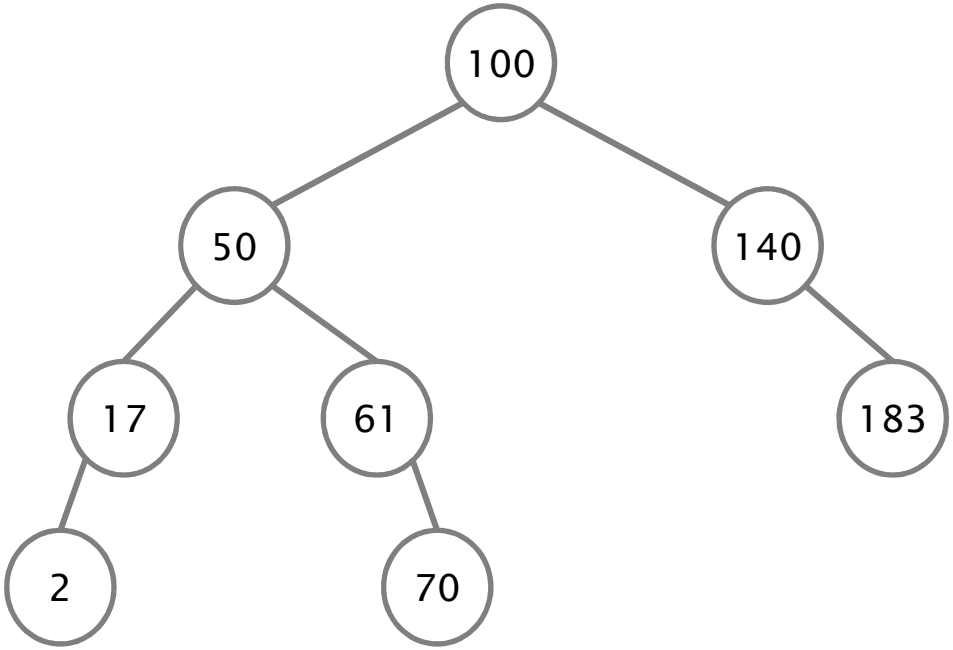
exemplos: árvore AVL



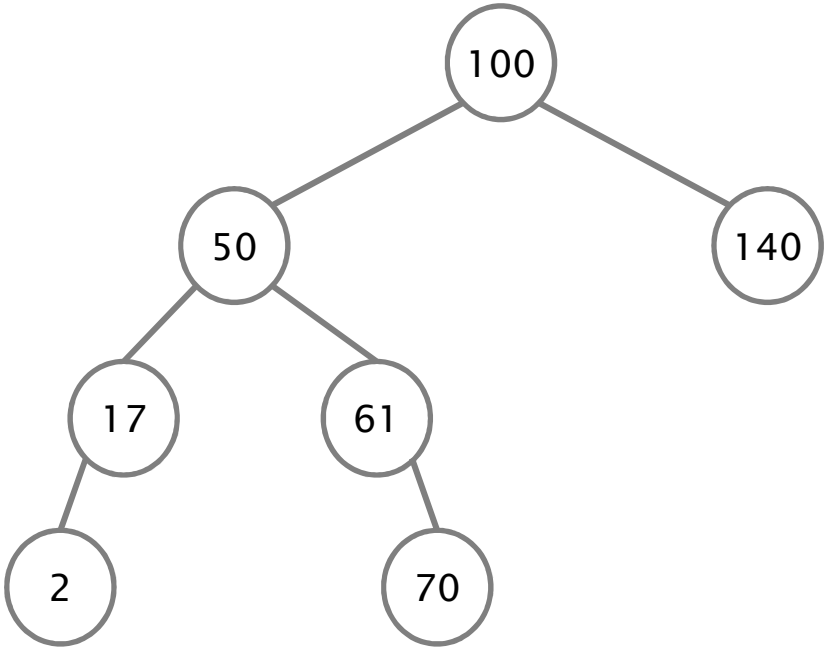
exemplos: árvore AVL



exemplos: árvore AVL



exemplos: árvore não AVL



Fator de balanceamento

fb = hd – he

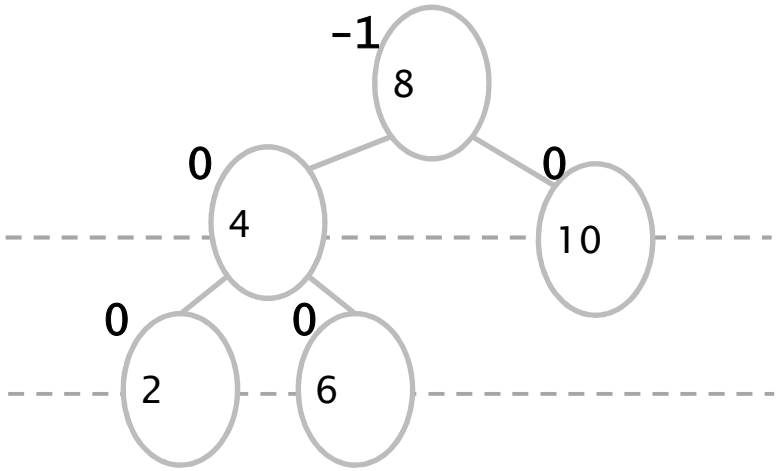
onde he = altura da sub-árvore à esquerda

hd = altura da sub-árvore à direita

```
struct _avl {
    int chave;
    int fb; /*fator de balanceamento*/
    struct _avl *esq;
    struct _avl *dir;
    tdados *dados;
};
```

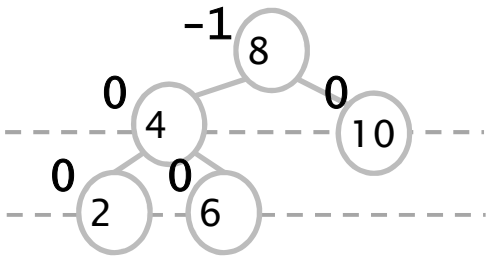
Árvore AVL – desequilíbrio (caso 1a)

árvore balanceada (equilibrada)

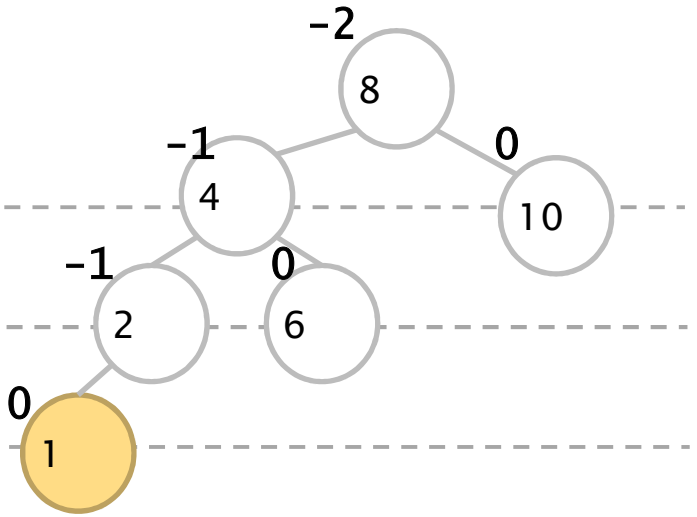


Árvore AVL – desequilíbrio (caso 1a)

árvore balanceada (equilibrada)



desequilíbrio após inserir 1

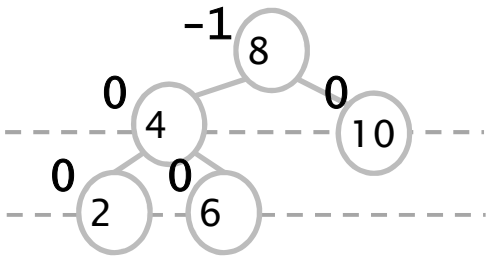


Árvore AVL – esboço inserção

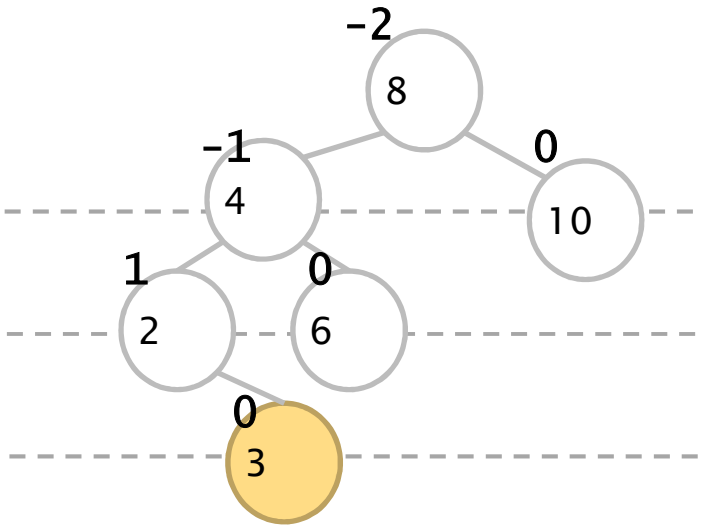
```
Avl* avl_insere (Avl* r,int chave) {
  if (r==NULL) {
     r = (Avl*) malloc(sizeof(Avl));
     r->esq = r->dir = NULL;
     r->chave = chave;
     r->fb = 0;
  }
  else if (r->chave > chave) {
     r->esq = avl_insere (r->esq, chave, flag);
     verifica se precisa consertar
   }
  else if (r->chave < chave) {
     r->dir = avl_insere (r->dir, chave, flag);
     verifica se precisa consertar
  }
  return r;
}
```

Árvore AVL – desequilíbrio (caso 1a)

árvore balanceada (equilibrada)

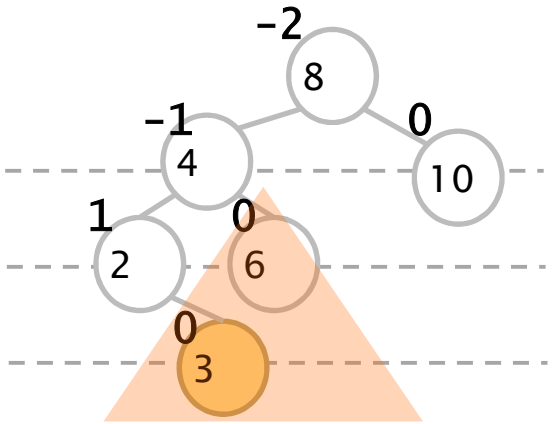


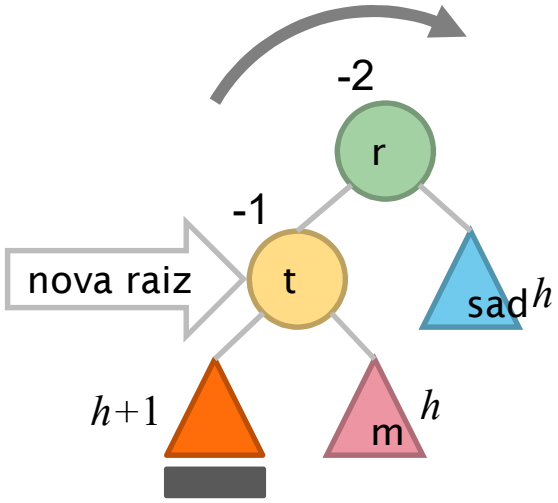
desequilíbrio após inserir 3



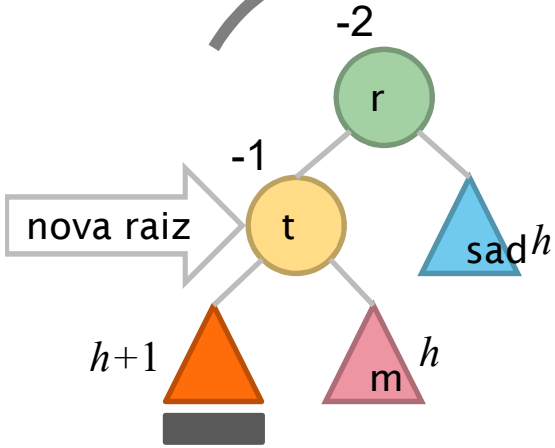
Árvore AVL – Caso 1a: rotação à direita

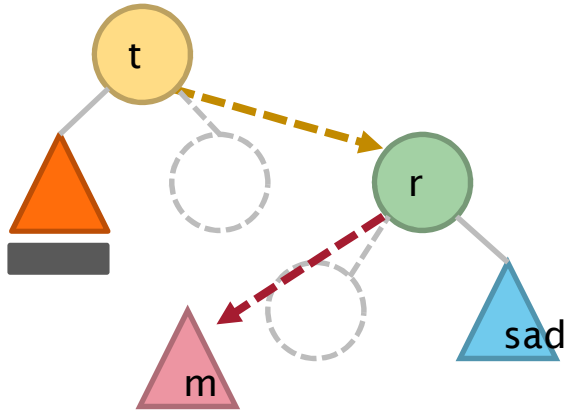
desequilíbrio após inserir 3

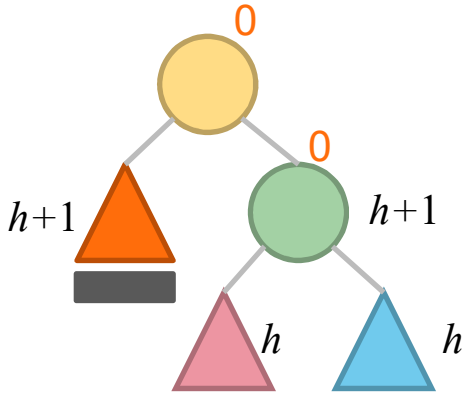




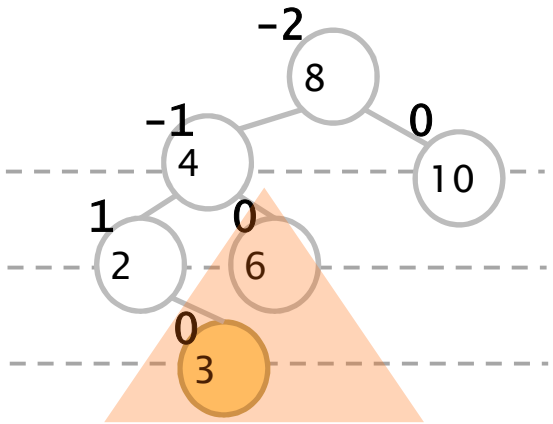
Árvore AVL – Caso 1a: rotação à direita







desequilíbrio após inserir 3

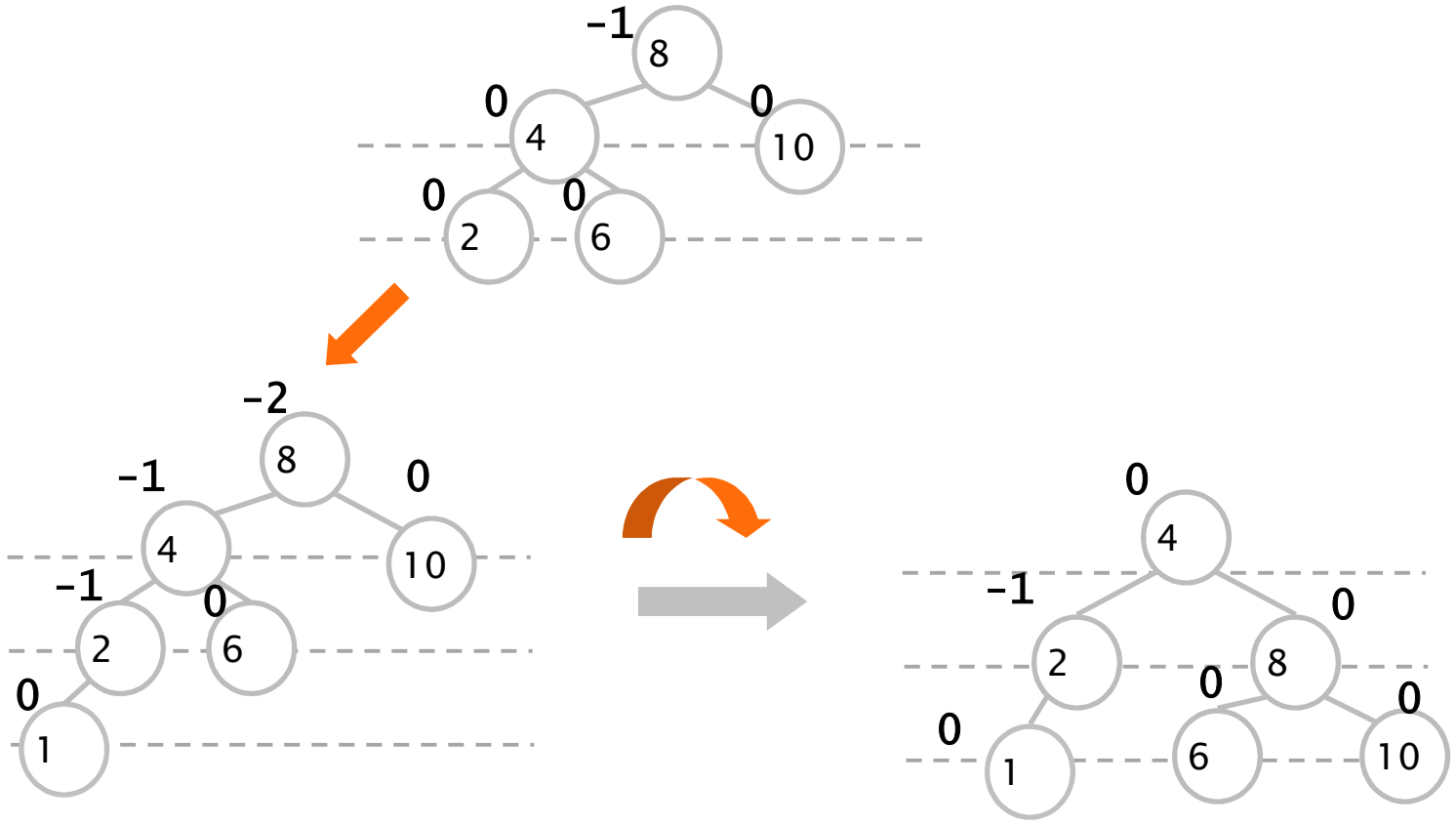


Caso 1a - Exemplo de rotação à direita

Uma rotação à direita ☑

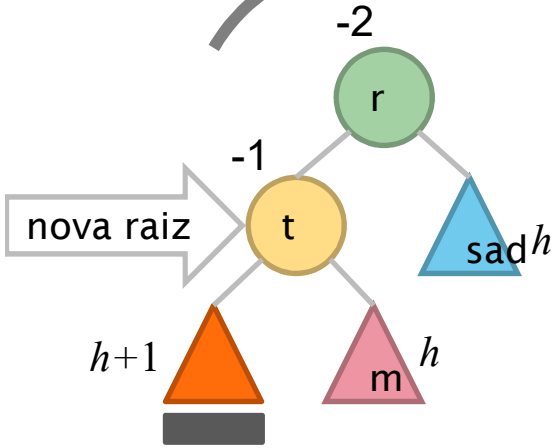
Caso 1a - Exemplo de rotação à direita

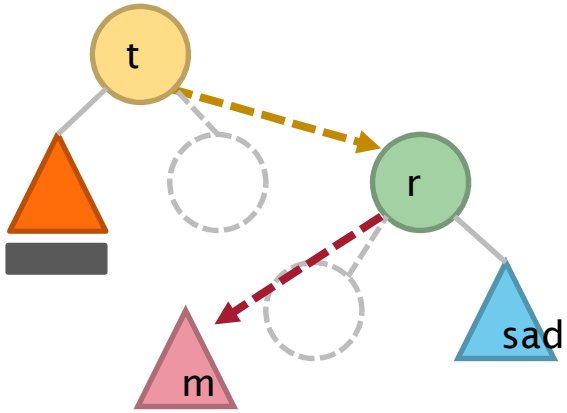
árvore balanceada (equilibrada)

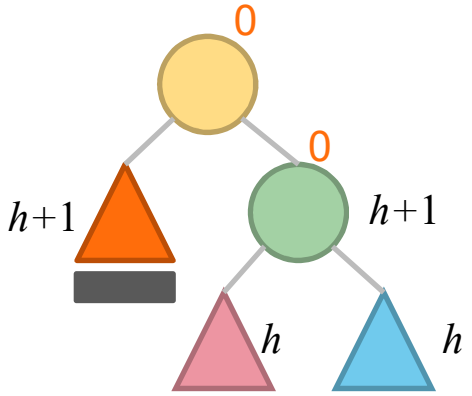


Uma rotação à direita ☑

Árvore AVL – Caso 1a: rotação à direita

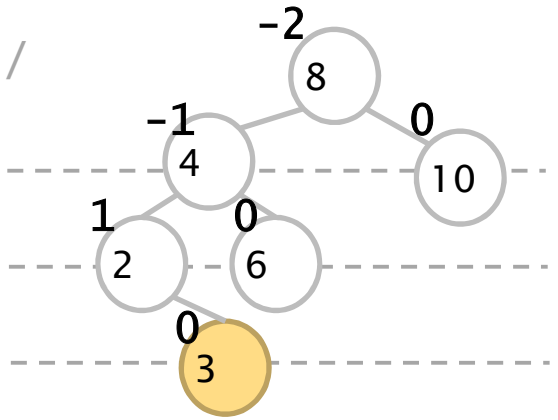






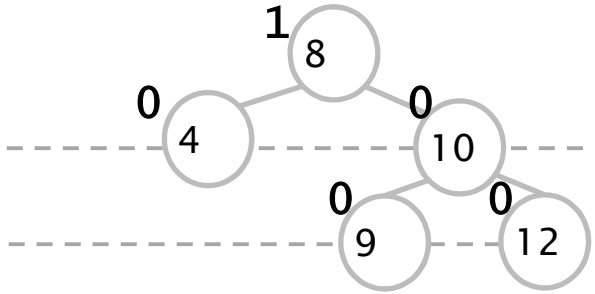

```
Avl* rotacao_direita(Avl *r) {
/* falta testar se esses ponteiros existem */
    Avl *t=r->esq, *m=t->dir;
    t->dir = r;
    r->esq = m;
    t->fb = r->fb = 0;
    return t;
}
```

desequilíbrio após inserir 3

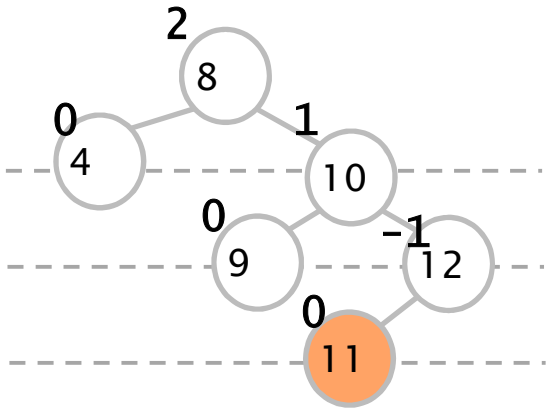


Árvore AVL – desequilíbrio (caso 1b)

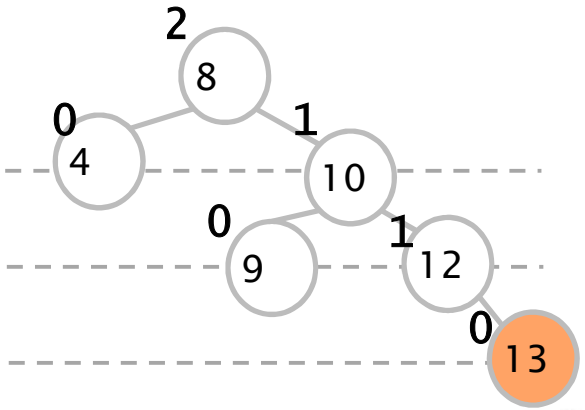
árvore balanceada (equilibrada)



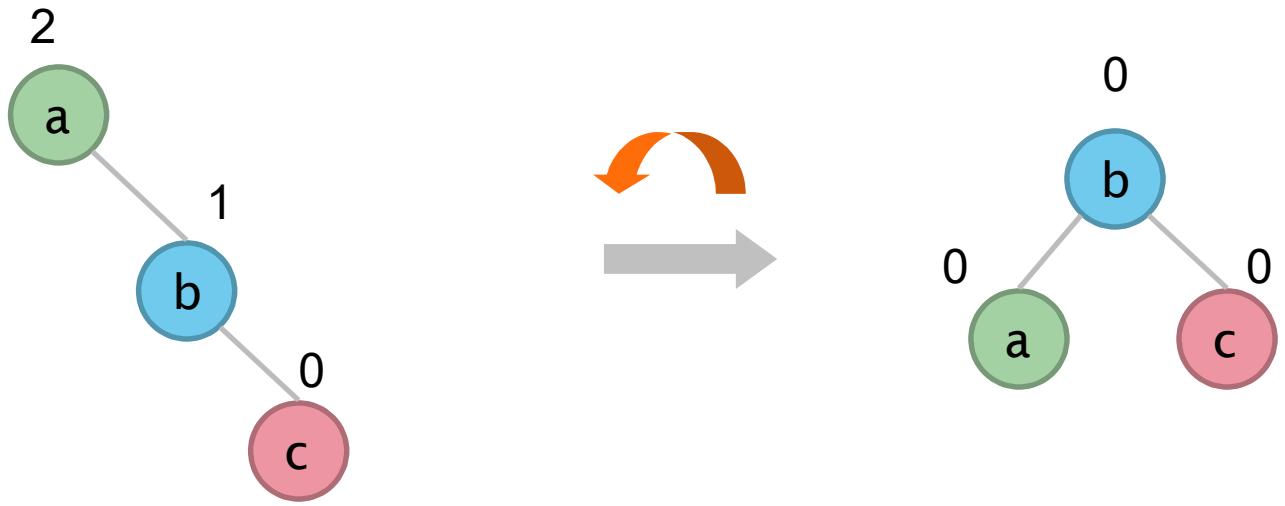
desequilíbrio após inserir 11



desequilíbrio após inserir 13

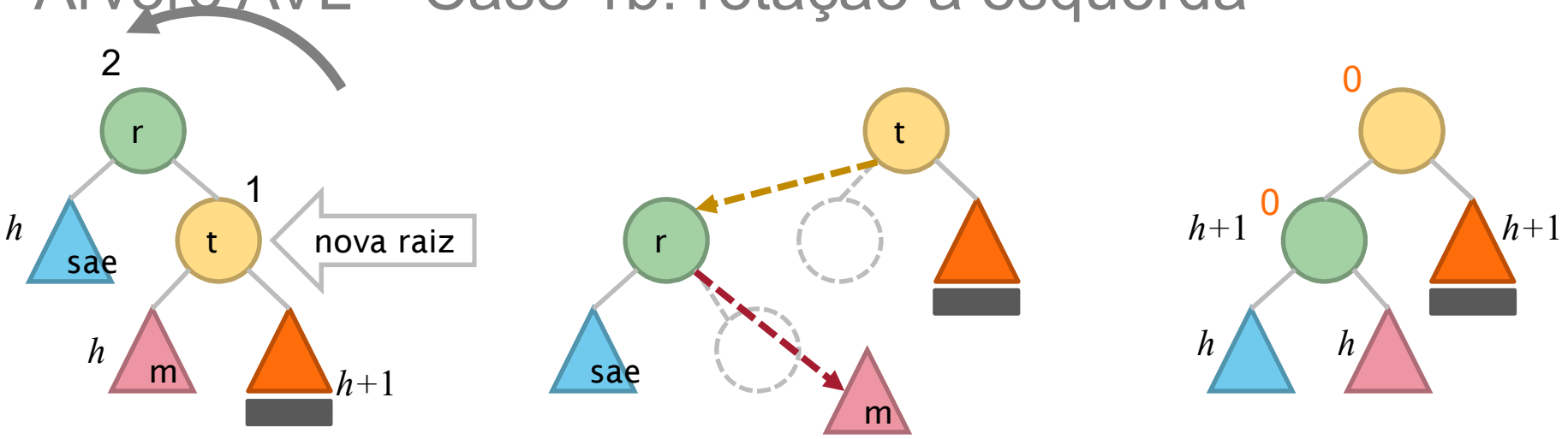


Caso 1b - Exemplo de rotação à esquerda



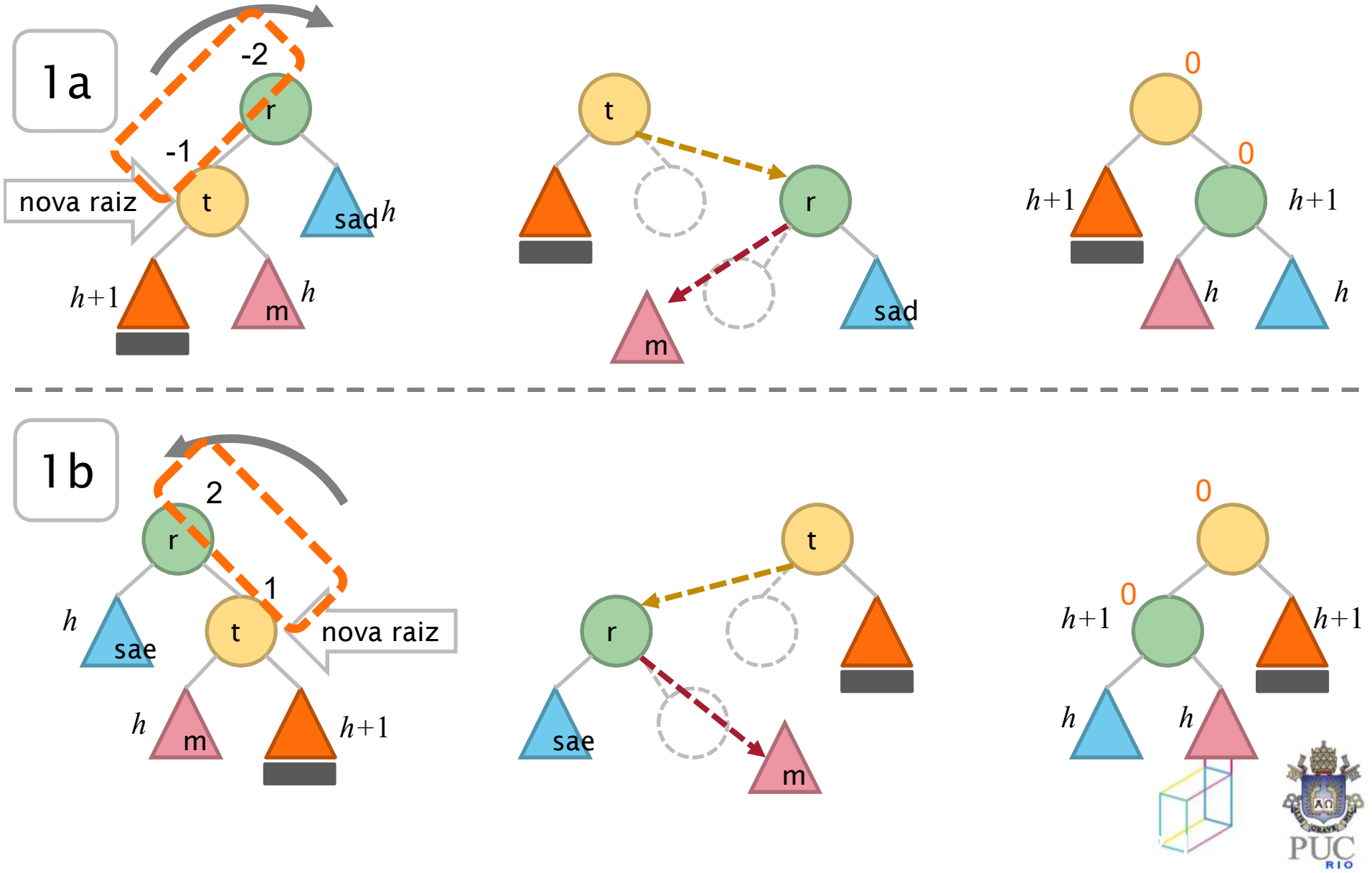
uma rotação à esquerda ☑

Árvore AVL – Caso 1b: rotação à esquerda

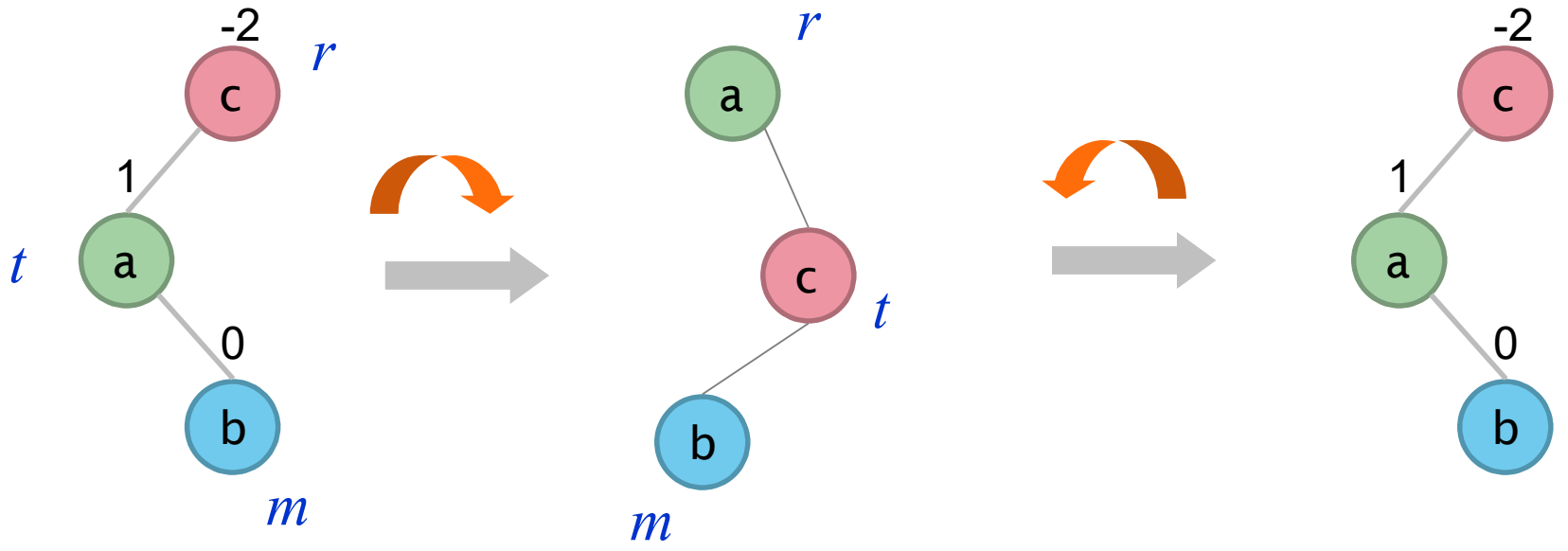



```
static Avl* rotacao_esquerda(Avl *r) {
    /* falta testar se esses ponteiros existem */
    Avl *t=r->dir, *m=t->esq;
    t->esq = r;
    r->dir = m;
    t->fb = r->fb = 0;
    return t;
}
```

Árvore AVL – Casos simples: fb com mesmo sinal



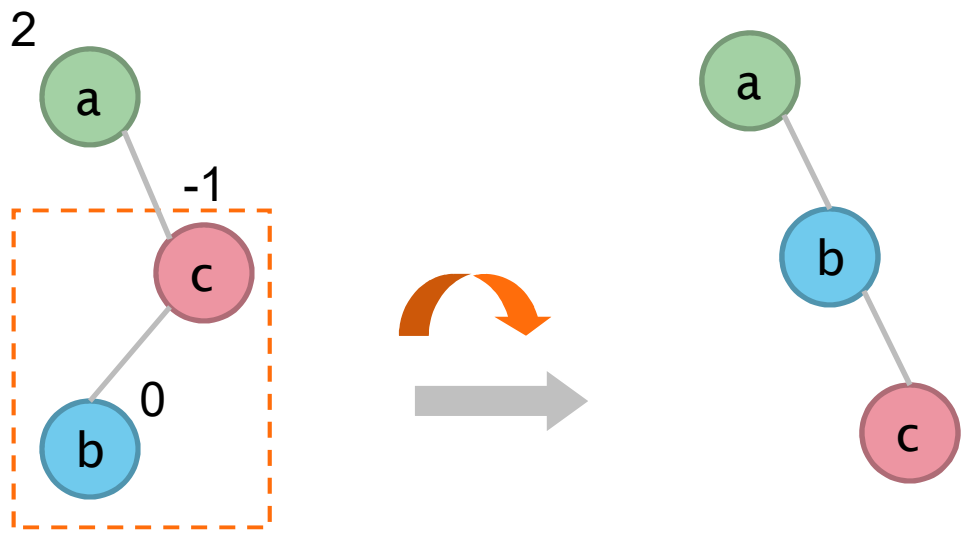
Exemplo: caso em que rotação simples não resolve



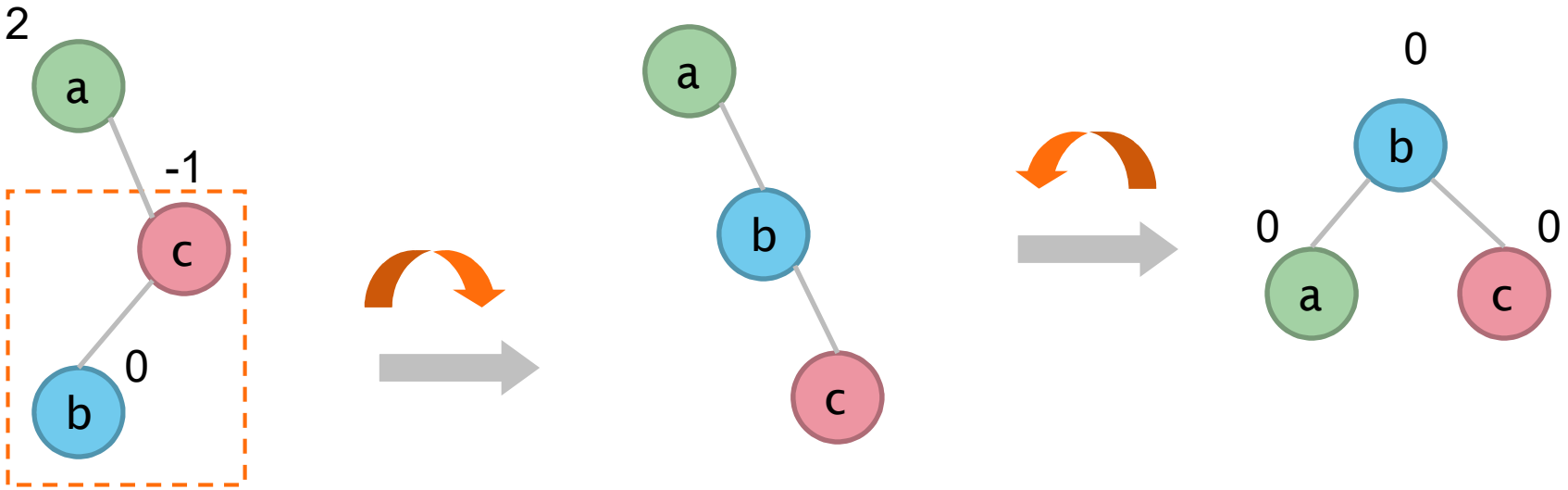
Não resolve!

Caso 2a - Balanceamento com rotação dupla

fazemos uma rotação à direita na sub-árvore à direita...

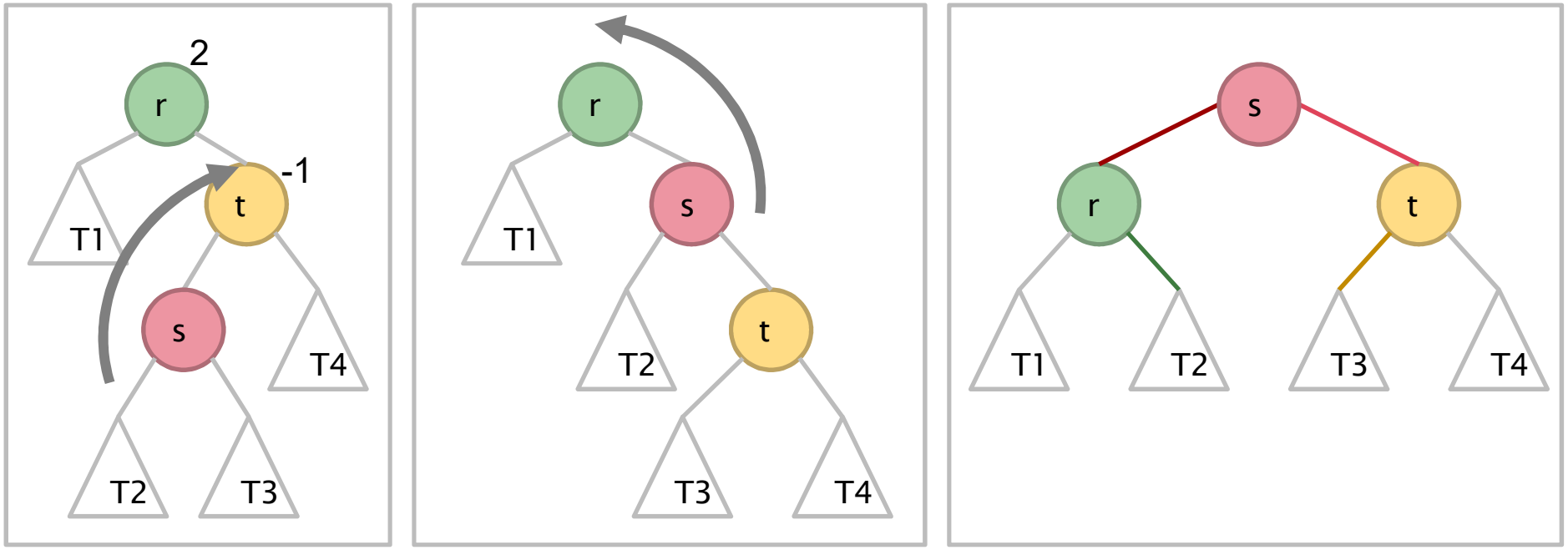


Caso 2a - Balanceamento com rotação dupla



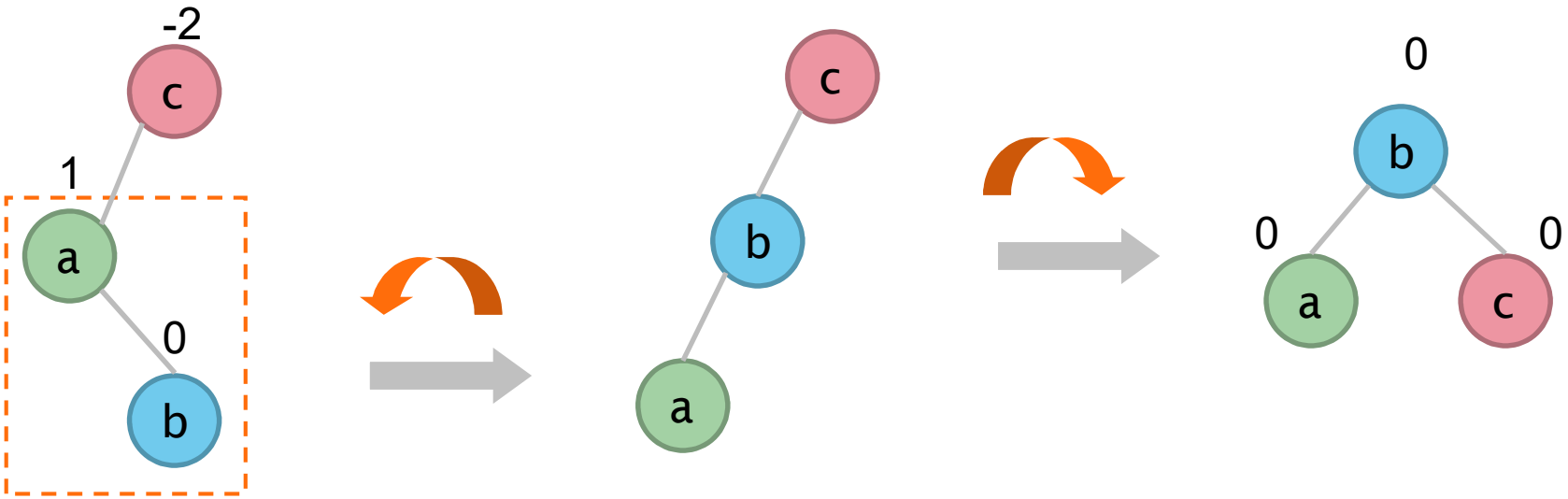
...seguida de uma rotação à esquerda

Caso 2a – Rotação direita-esquerda



Caso 2b - Balanceamento com rotação dupla

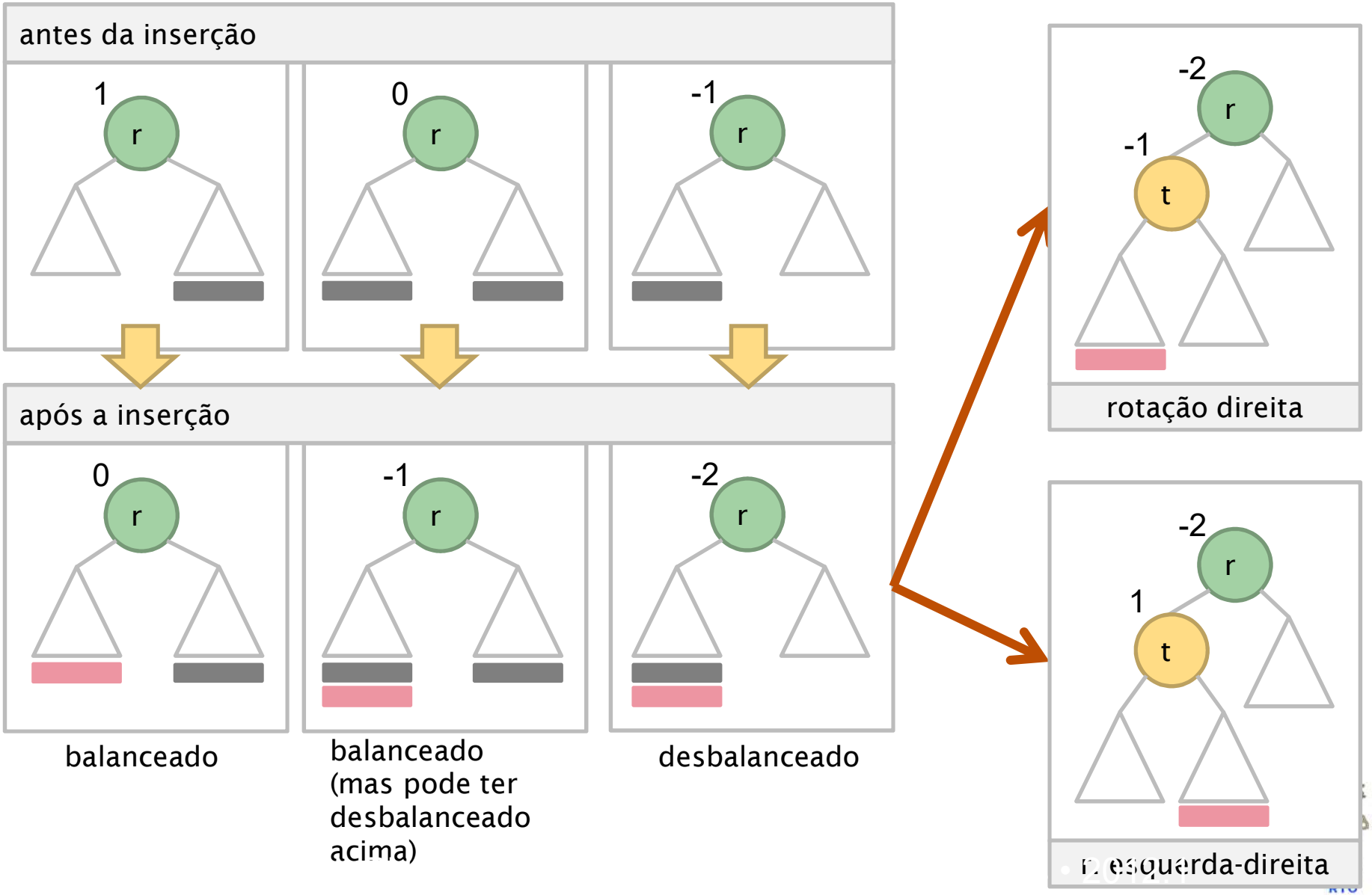
Uma rotação à esquerda na sub-árvore à esquerda...



...seguida de uma rotação à direita

Inserção

Inserção à esquerda



```
Avl* avl_insere2(Avl* r,int chave, int* flag) {
  if (r==NULL) {
     r = (Avl*) malloc(sizeof(Avl));
     r->esq = r->dir = NULL;
     r->chave = chave;
     r->fb = 0;
     *flag = 1;
  }
  else if (r->chave > chave) {
     r->esq = avl_insere2 (r->esq, chave, flag);
     if (*flag) …
   }
  else if (r->chave < chave) {
     r->dir = avl_insere2 (r->dir, chave, flag);
     if (*flag) …
  }
  return r;
}
Avl* avl_insere(Avl* r, int chave) {
    int flag=0;
    return avl_insere2 (r,chave,&flag);
}
```

```
Avl* avl_insere2(Avl* r,int chave, int* flag) {
  if (r==NULL) {
     r = (Avl*) malloc(sizeof(Avl));
     r->esq = r->dir = NULL;
     r->chave = chave;
     r->fb = 0;
     *flag = 1;
  }
  else if (r->chave > chave) {
     r->esq = avl_insere2 (r->esq, chave, flag);
```

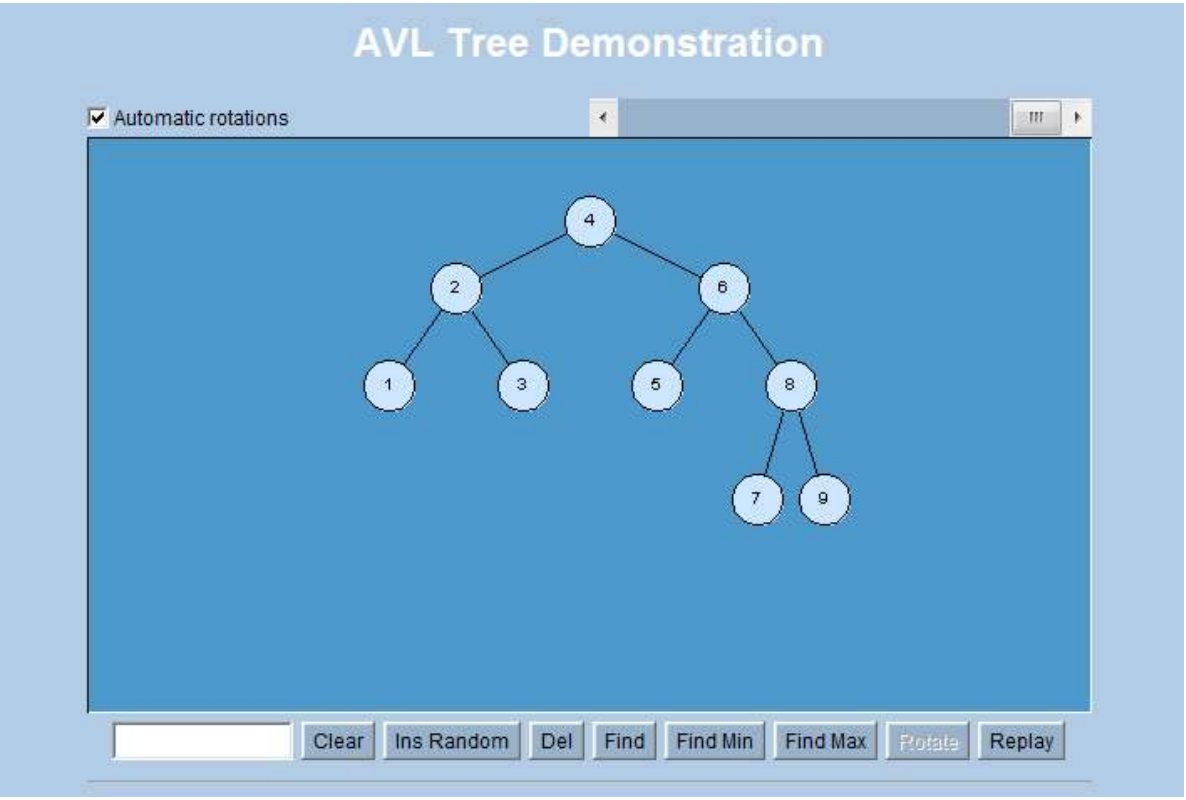
```
if (*flag) {       /* r cresceu à esquerda (ou seja, he aumentou)*/
  switch(r->fb) { /* análise do fator de balanceamento de r */
    case  1: /* antes: hd>he xxx depois: hd=he pois he aumentou */
      …
    case  0: /* antes: hd=he xxx depois: hd<he pois he aumentou */
      …
    case -1: /* antes: hd<he xxx depois: hd-he=-2 pois he aumentou */
      …
  }
}
```

```
}
```

```
Avl* avl_insere2(Avl* r,int chave, int* flag) {
  if (r==NULL) {
     r = (Avl*) malloc(sizeof(Avl));
     r->esq = r->dir = NULL;
if (*flag) {      /* r cresceu à esquerda (ou seja, he
aumentou)*/
  switch(r->fb) { /* análise do fator de balanceamento de r */
    case  1: /* antes: hd>he - depois: hd=he pois he aumentou
*/
      r->fb =  0; *flag = 0; break;
    case  0: /* antes: hd=he - depois: hd<he pois he aumentou
*/
      r->fb = -1; break;
    case -1: /* antes: hd<he - depois: hd-he=-2 pois he
aumentou */
      if (r->esq->fb == -1) r = avl_rotacao_direita(r);
             else r = avl_rotacao_esquerda_direita(r);
      *flag = 0; break;
  }
}
}
```

Animação de Árvores AVL

http://www.strille.net/works/media_technology_projects/avl-tree_2001/

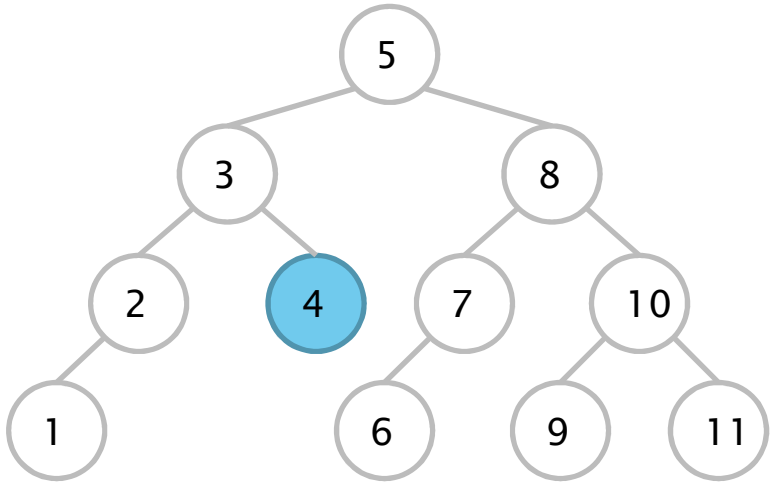


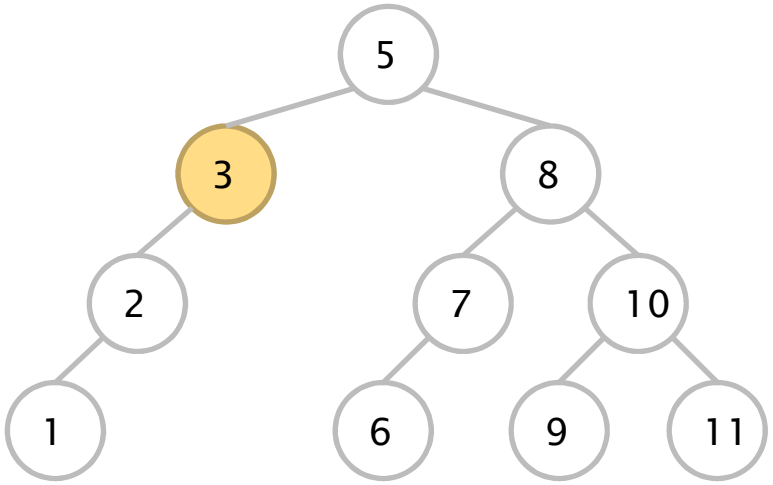
Remoções

- Problemas
  - semelhantes aos das inserções, ou seja, pode ocorrer desbalanceamento

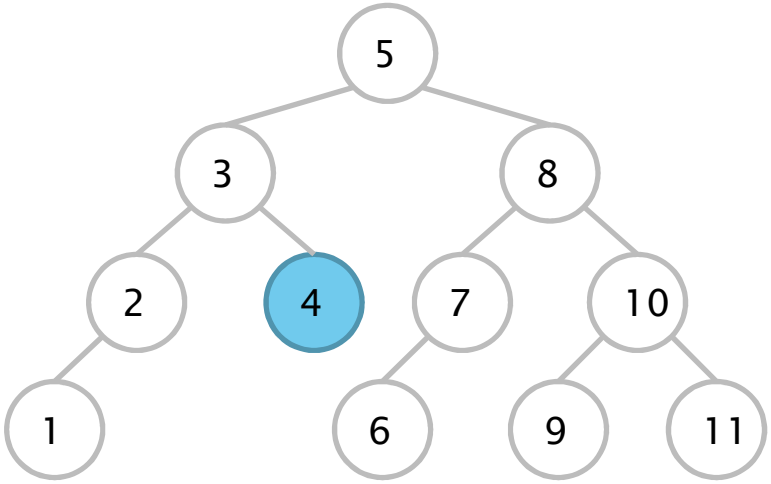
- Análise de casos:
  - Caso 1:
    - o nó removido é uma folha, ou
    - o nó removido possui apenas 1 descendente
  - Caso 2:
    - o nó removido possui as duas sub-árvores

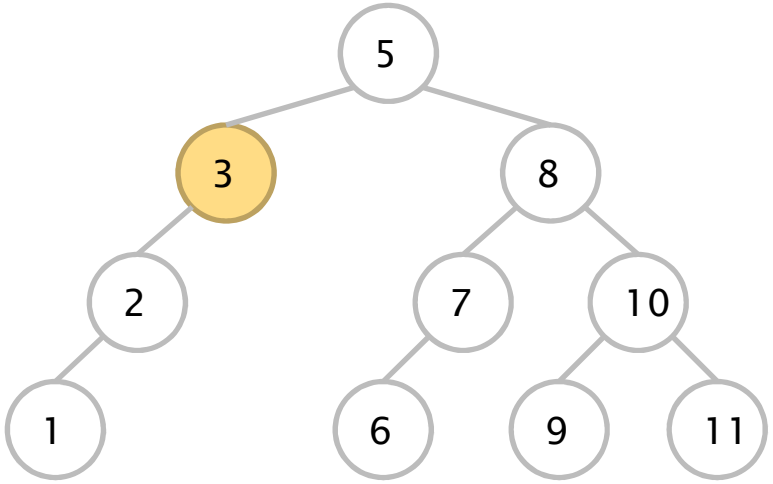
Exemplo: Remove 4



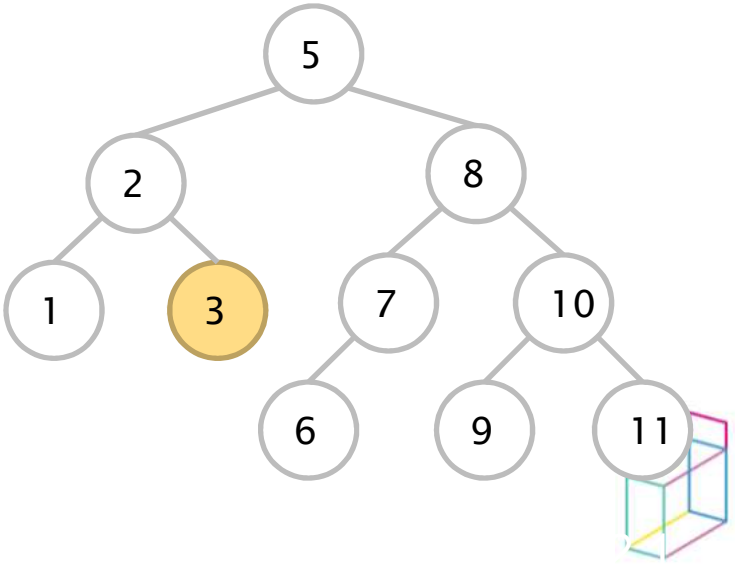


Exemplo: Remove 4

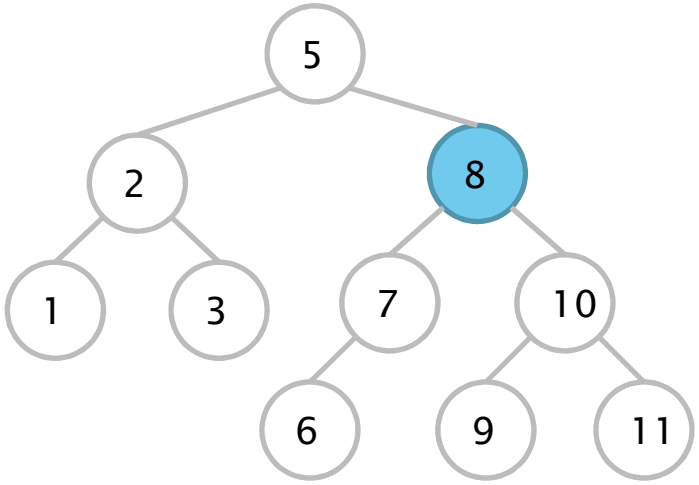


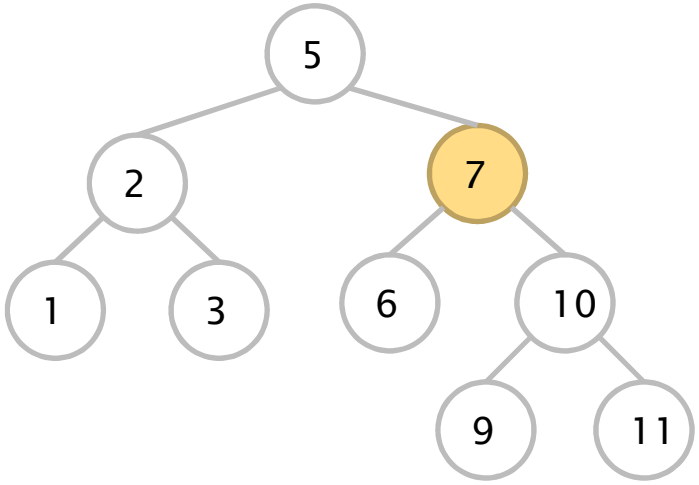


rotação à direita

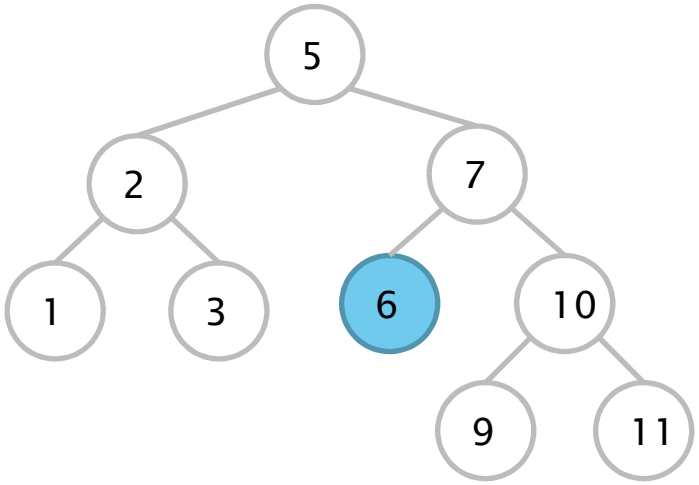


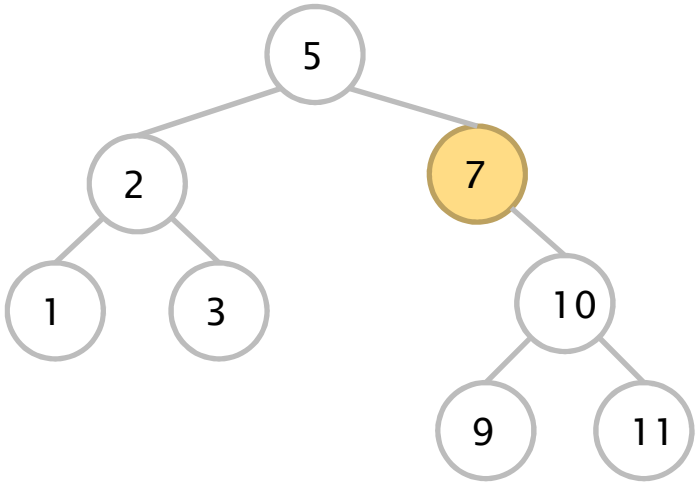
Exemplo: Remove 8



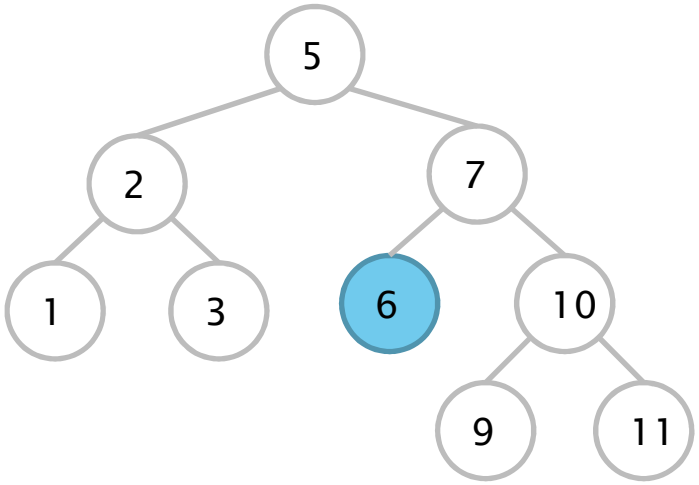


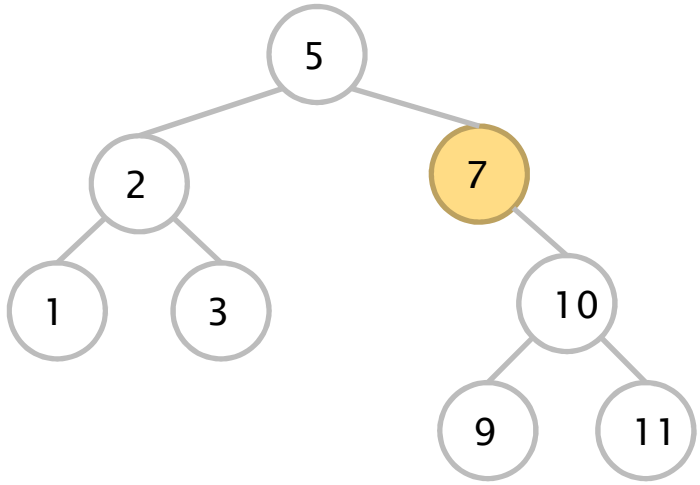
Exemplo: Remove 6



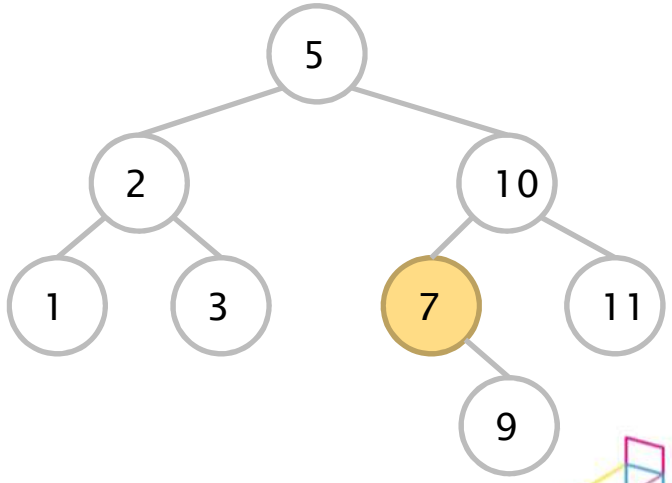


Exemplo: Remove 6





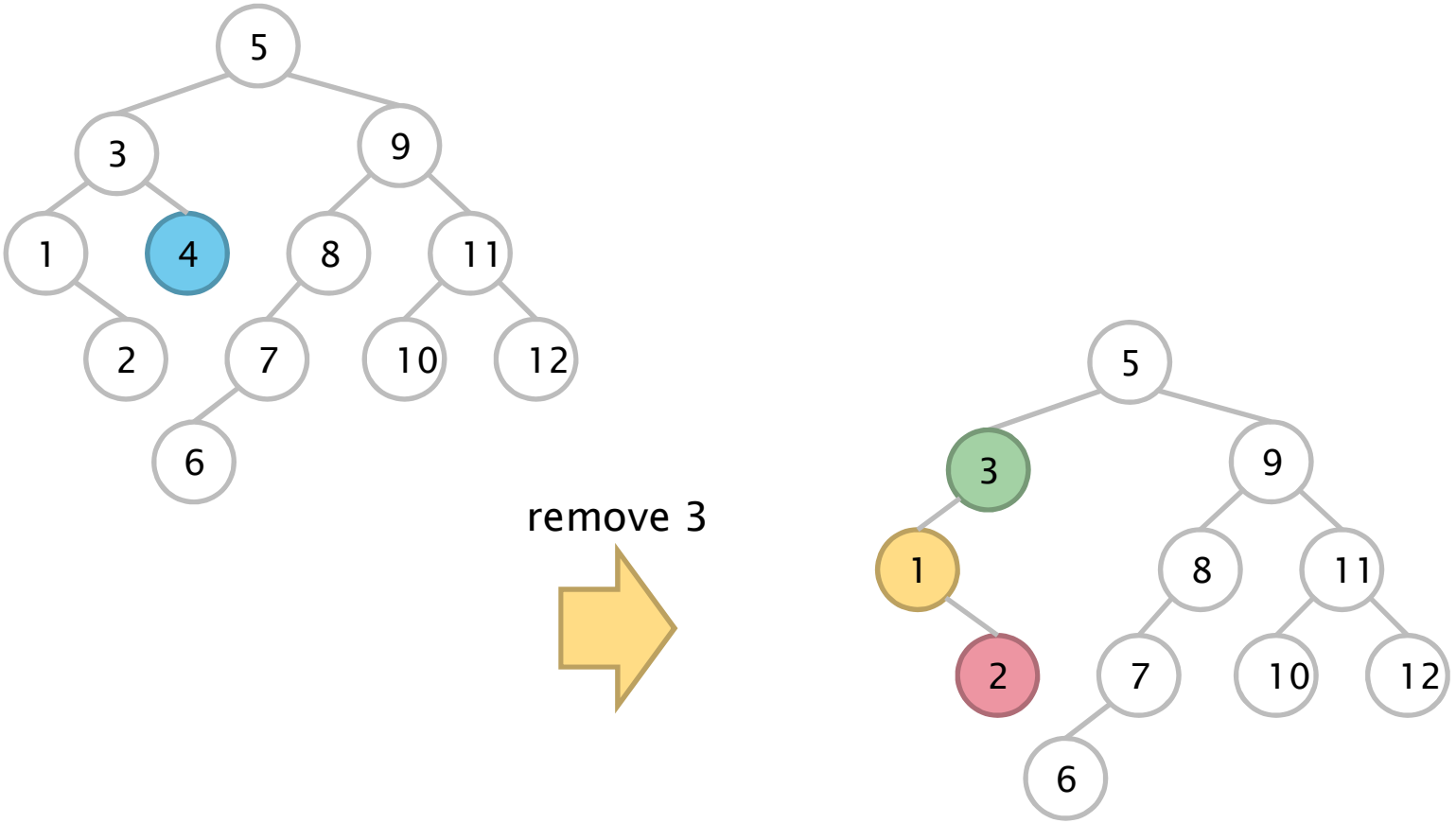
rotação à esquerda



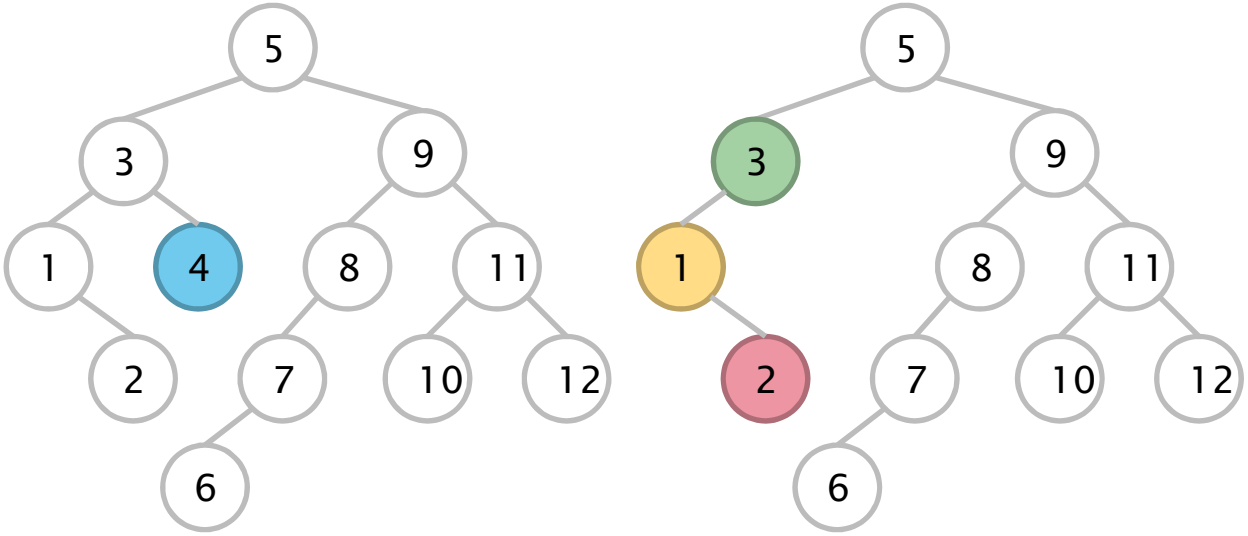
Problemas com as Árvores AVL

- Problema:
  - Nem sempre uma ÚNICA rotação (simples ou dupla) resolve o problema de desbalanceamento dos nós
  - Há casos em que O(log n) rotações são necessárias para tornar a árvore balanceada
- Exemplo (a seguir):
  - exclusão do nó provoca diversas rotações

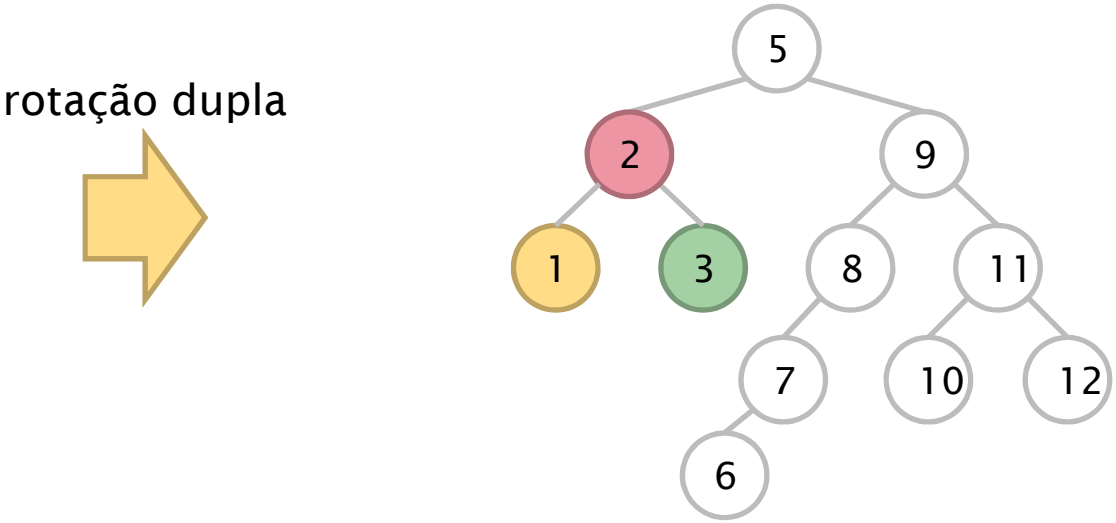
Múltiplas Rotações na Árvore AVL



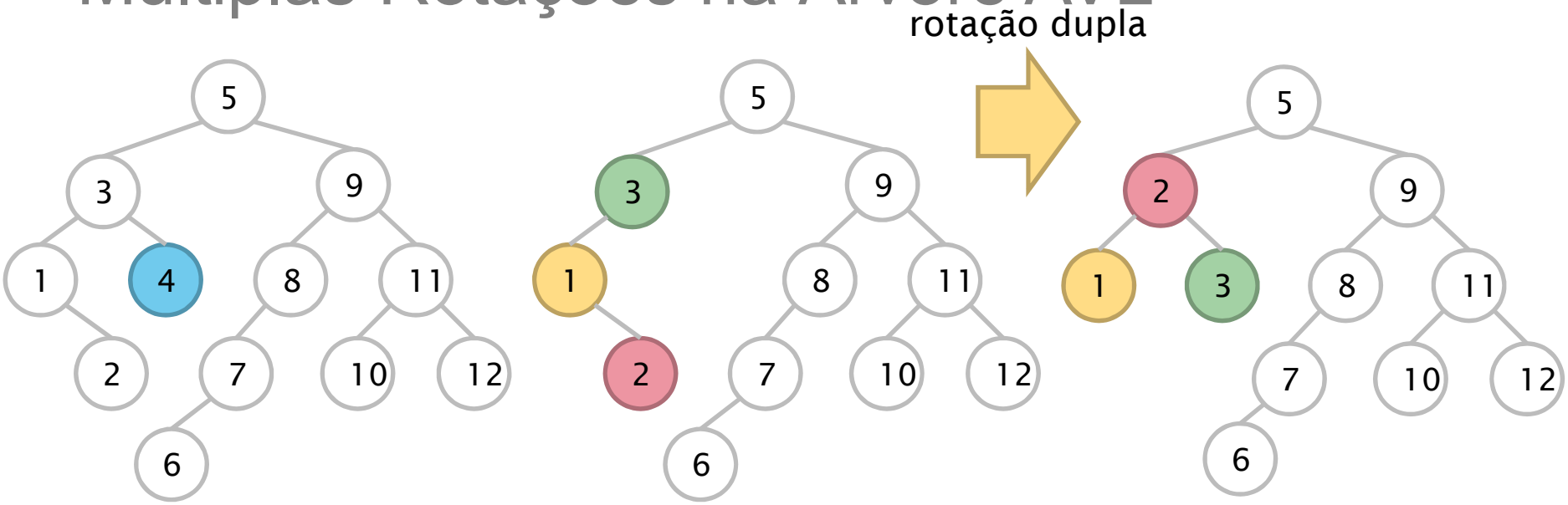
Múltiplas Rotações na Árvore AVL



rotação dupla



Múltiplas Rotações na Árvore AVL



Esboço do algoritmo de remoção

- Pesquise o nó que contém a chave procurada,
aplicando recursivamente o algoritmo de remoção
- Quando achar o nó
  - faça a remoção
    (semelhante à remoção em árvore de busca binária)
  - analise o balanceamento
  - Se o nó estiver desbalanceado (|fb| >1),
    faça as devidas rotações
- Recursivamente, reporte uma mudança na altura de um nó
ao seu pai para que ele faça as devidas correções

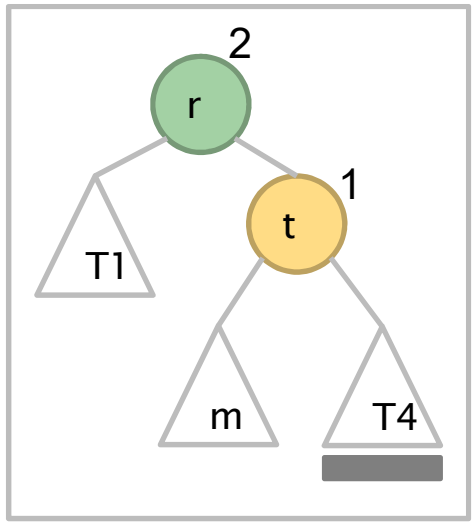
Esboço do algoritmo de remoção

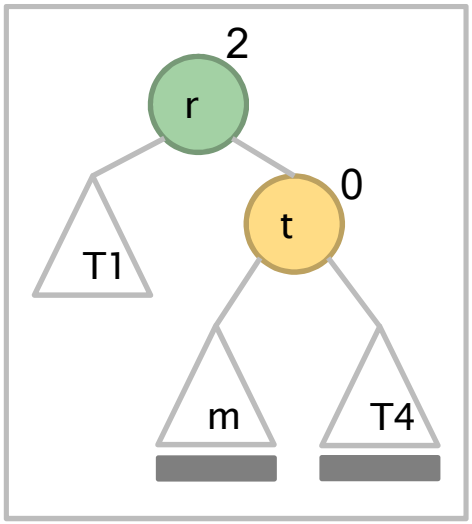
- Pontos adicionais da remoção

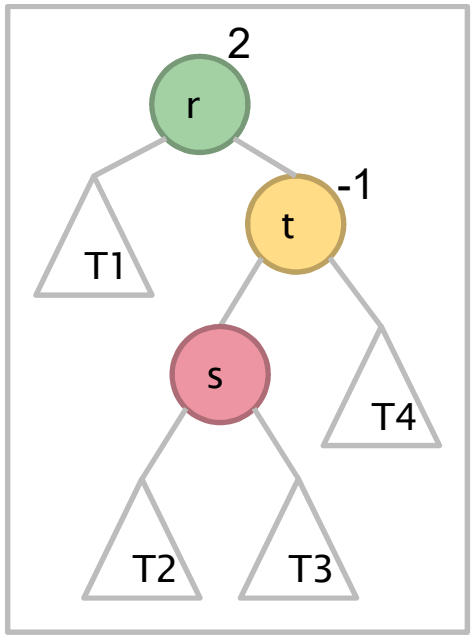
(que não ocorrem na inserção)

  - tratar todos os casos possíveis de altura nas rotações
  - analisar quando uma sub-árvore muda de altura para reportar esta informação ao pai

Casos de rotação à esquerda (a s.a.e. perdeu altura)



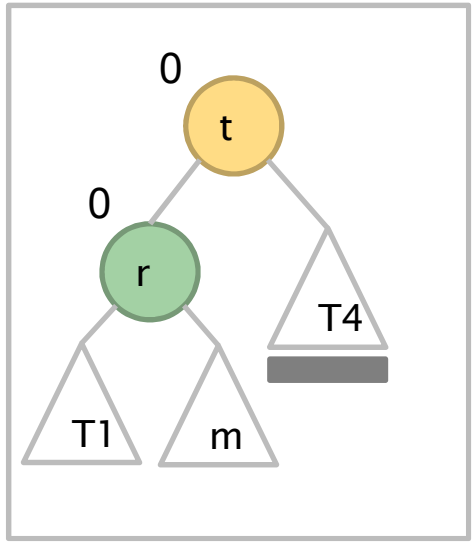


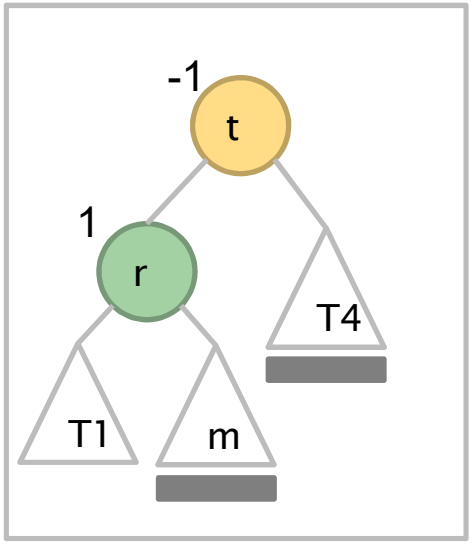


rotação dupla

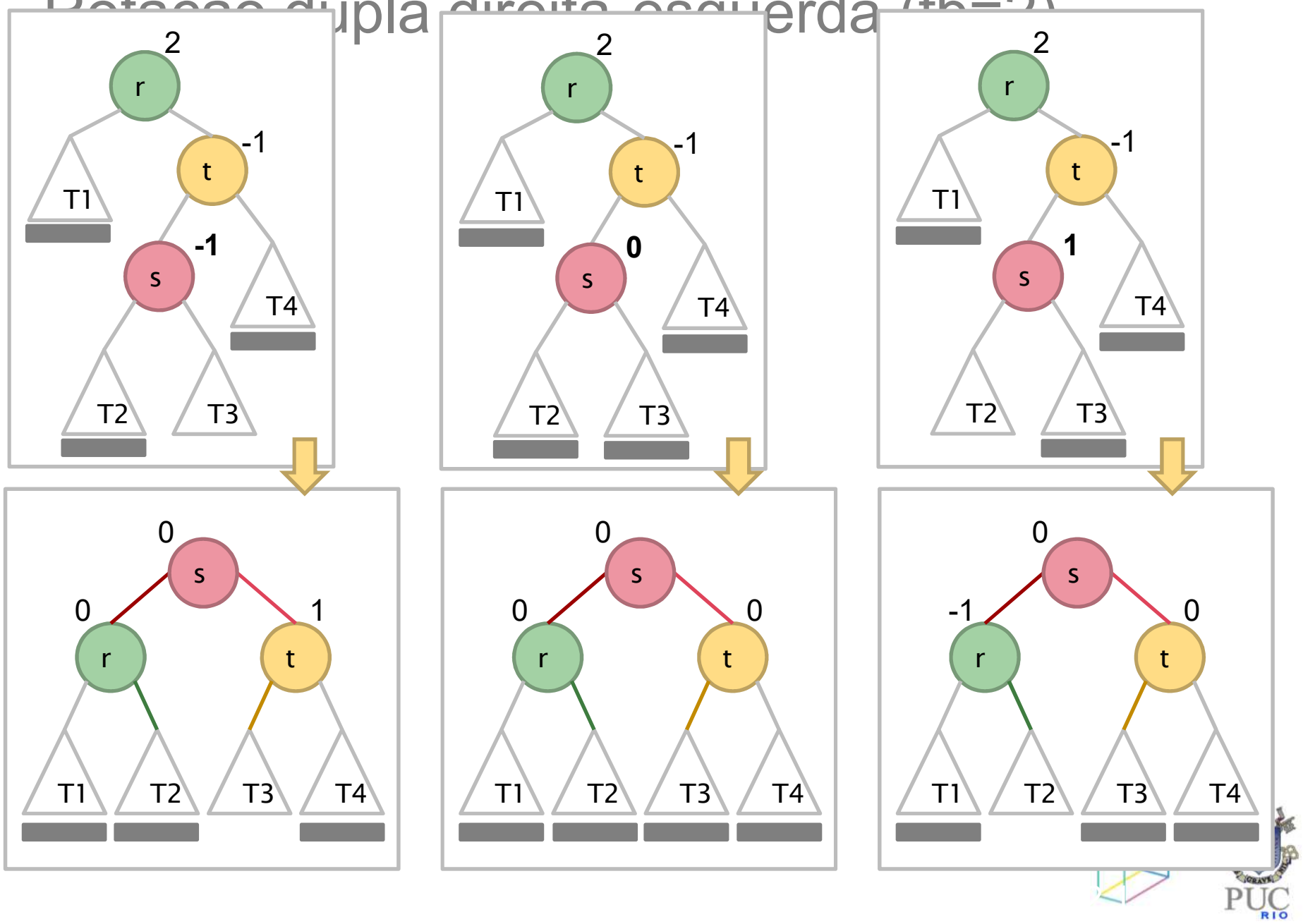
rotação à esquerda

rotação à esquerda

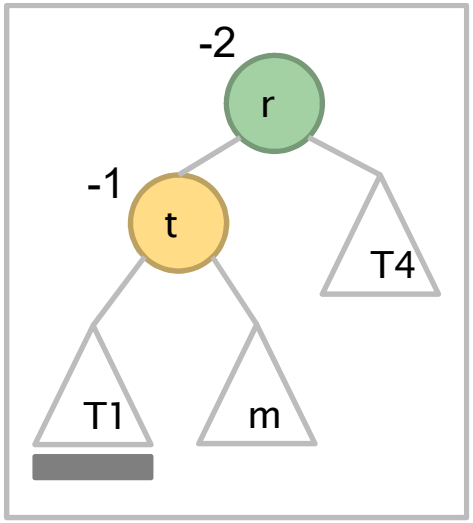


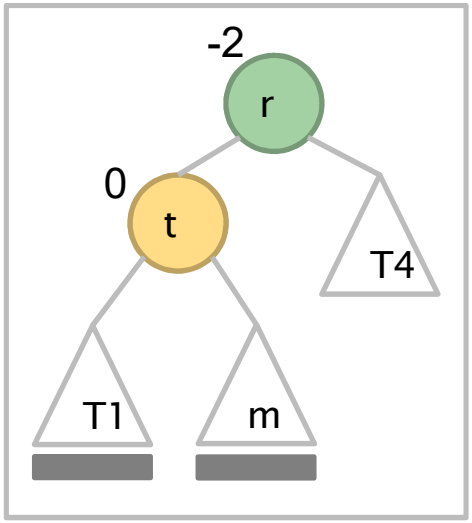


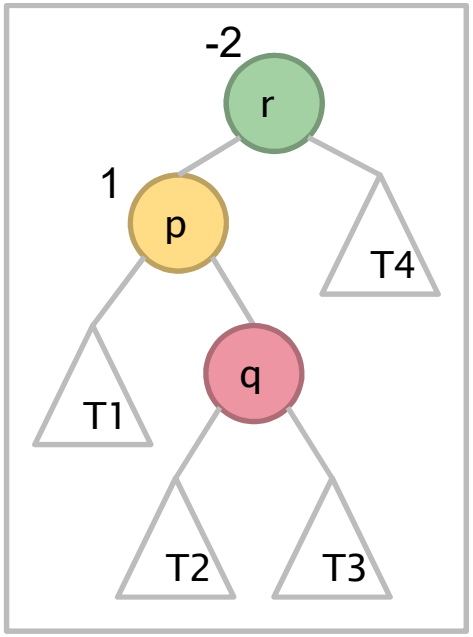
Rotação dupla direita-esquerda (fb=2)



Casos de rotação à direita (a s.a.d. perdeu altura)



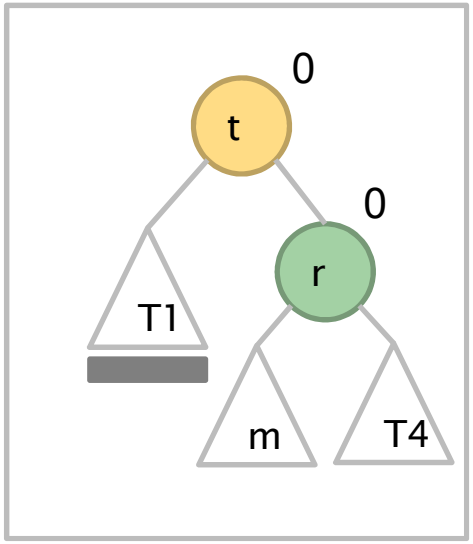


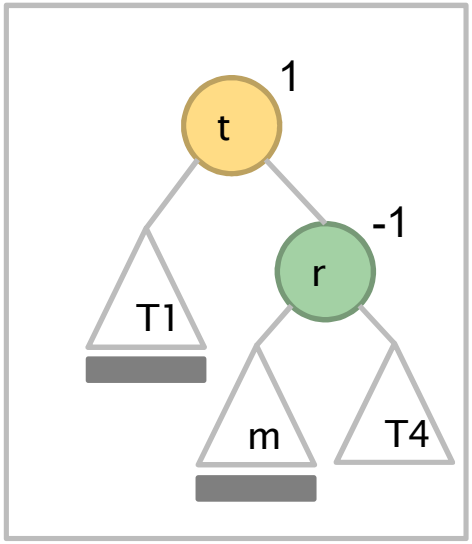


rotação dupla

rotação à direita

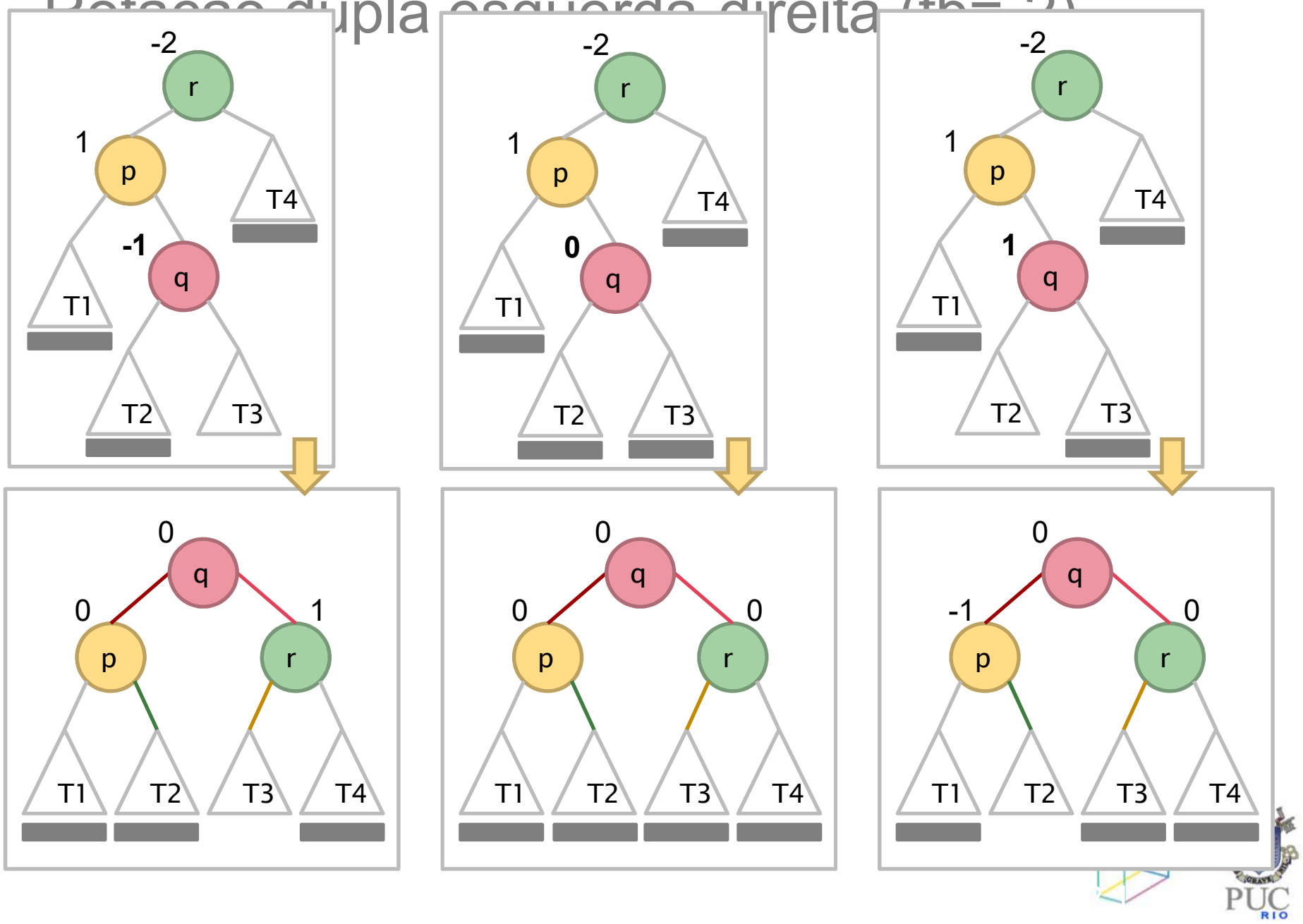
rotação à direita






```
static Avl* rot_direita(Avl *r) {
  Avl *t=r->esq, *m=t->dir;
  t->dir = r;
  r->esq = m;
  if (t->fb == -1) {
    t->fb = 0;  r->fb =  0; }
  else          {
    t->fb = 1;  r->fb = -1; }
  return t;
}
```

Rotação dupla esquerda-direita (fb= 2)



Relação entre altura e número de nós numa AVL

Seja N(h) o número mínimo de nós de uma AVL de altura h.

Então, a altura de uma das sub-árvores é, no mínimo, h-1 e a altura da outra é, no mínimo, h-2.

$$N(h) = N(h-1) + N(h-2) + 1$$

$$[N(h)+1] = [N(h-1)+1] + [N(h-2)+1]$$

$[N(h)+1] =$ *são números de Fibonacci*

$$N(h)+1 \approx \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+3}$$

$$h \approx 1.44\log(n))$$