AVL

SCC-202 – Algoritmos e Estruturas de Dados I

Prof. Thiago A. S. Pardo

AVL

- Árvore binária de busca balanceada
  - Para cada nó, as alturas das subárvores diferem em 1, no máximo
  - Proposta em 1962 pelos matemáticos russos G.M. Adelson-Velskki e E.M. Landis
    - Métodos de **inserção** e **remoção** de elementos da árvore de forma que ela fique balanceada

AVL

- Relembrando
  - Inserir os elementos 10, 3, 2, 5, 9, 7, 15, 12 e 13, nesta ordem, em uma árvore e balancear quando necessário

AVL

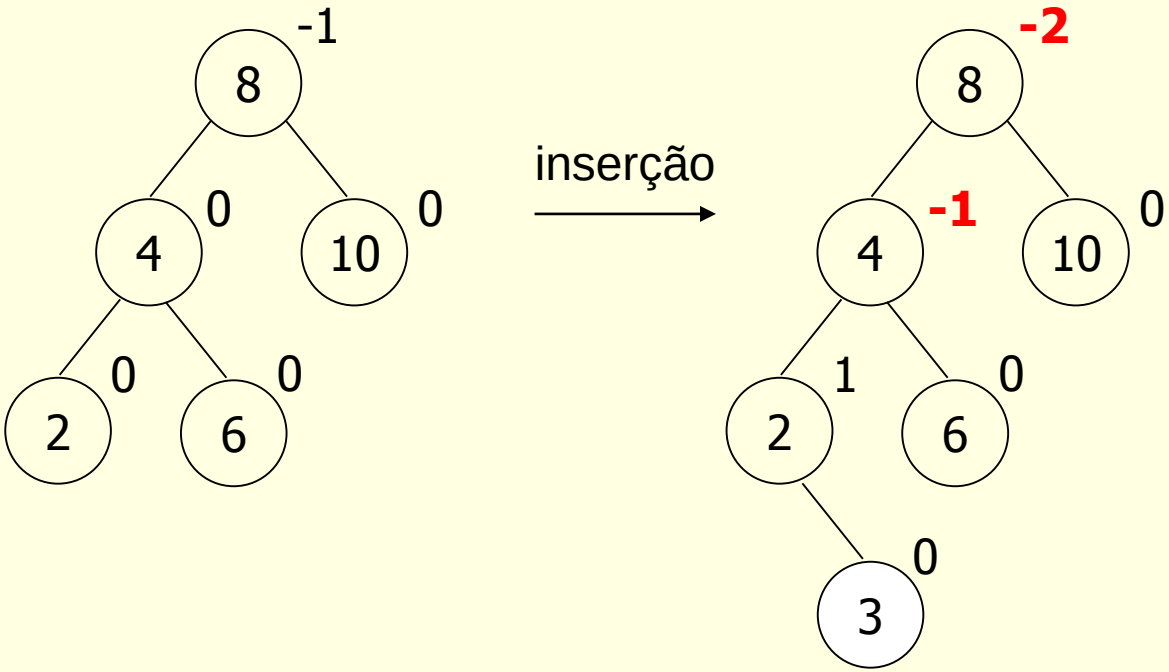
- Novo algoritmo de inserção
  - A cada inserção, verifica-se o balanceamento da árvore
    - Se necessário, fazem-se as rotações de acordo com o caso (sinais iguais ou não)

AVL

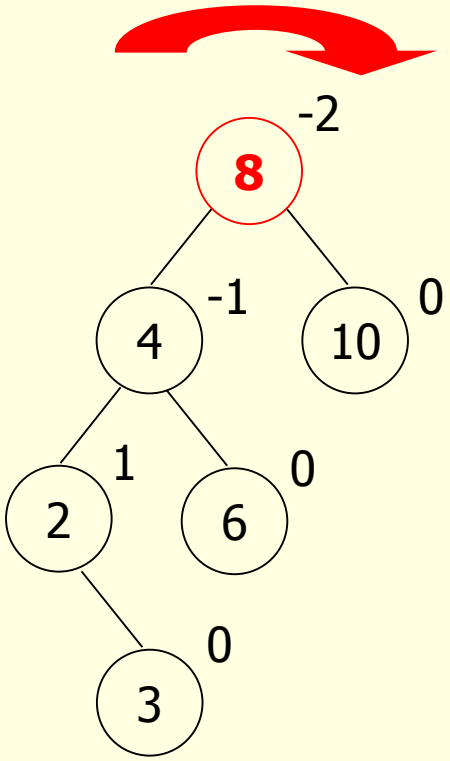
- Controle do balanceamento
  - Altera-se o algoritmo de inserção para balancear a árvore quando ela se tornar desbalanceada após uma inserção (nó com FB 2 ou -2)
    - Rotações
      - Se árvore pende para esquerda (FB negativo), rotaciona-se para a direita
      - Se árvore pende para direita (FB positivo), rotaciona-se para a esquerda
    - 2 casos podem acontecer

AVL: primeiro caso

- Raiz de uma subárvore com FB -2 (ou 2) e um nó filho com FB -1 (ou 1)
  - Os fatores de balanceamento têm sinais iguais: subárvores de nó raiz e filho pendem para o mesmo lado



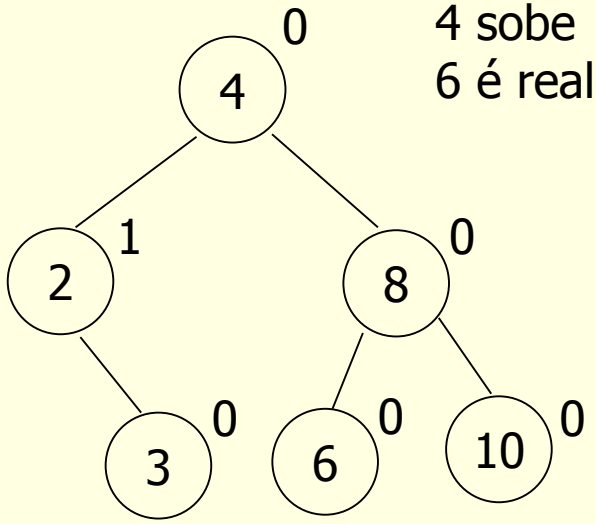
AVL: primeiro caso



Pendendo para
a esquerda

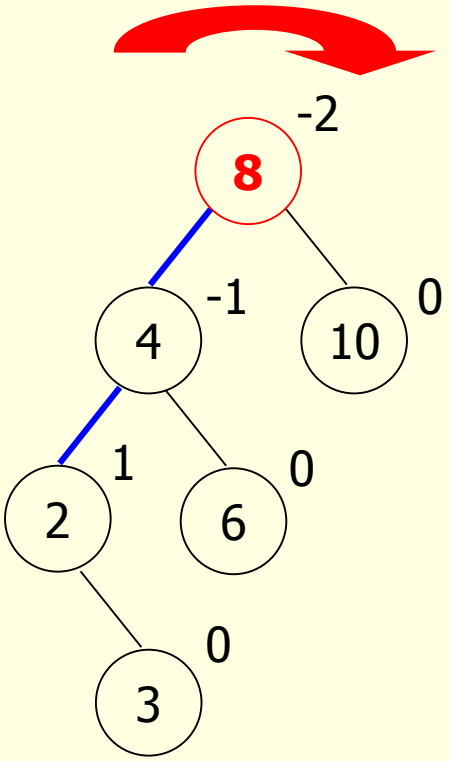
Rotação do nó pai
para a direita
(nó com FB=-2 ou 2)

8 desce
4 sobe
6 é realocado



Árvore balanceada!!!

AVL: primeiro caso

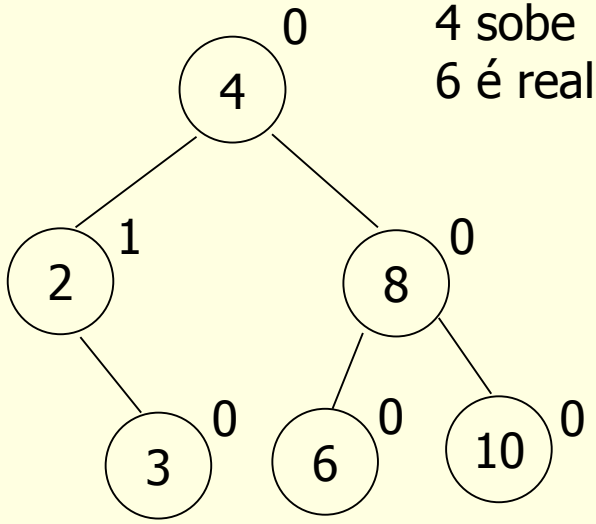


Pendendo para
a esquerda

Rotação do nó pai
para a direita
(nó com FB=-2 ou 2)

Rotação simples EE
(Esquerda-Esquerda)

8 desce
4 sobe
6 é realocado



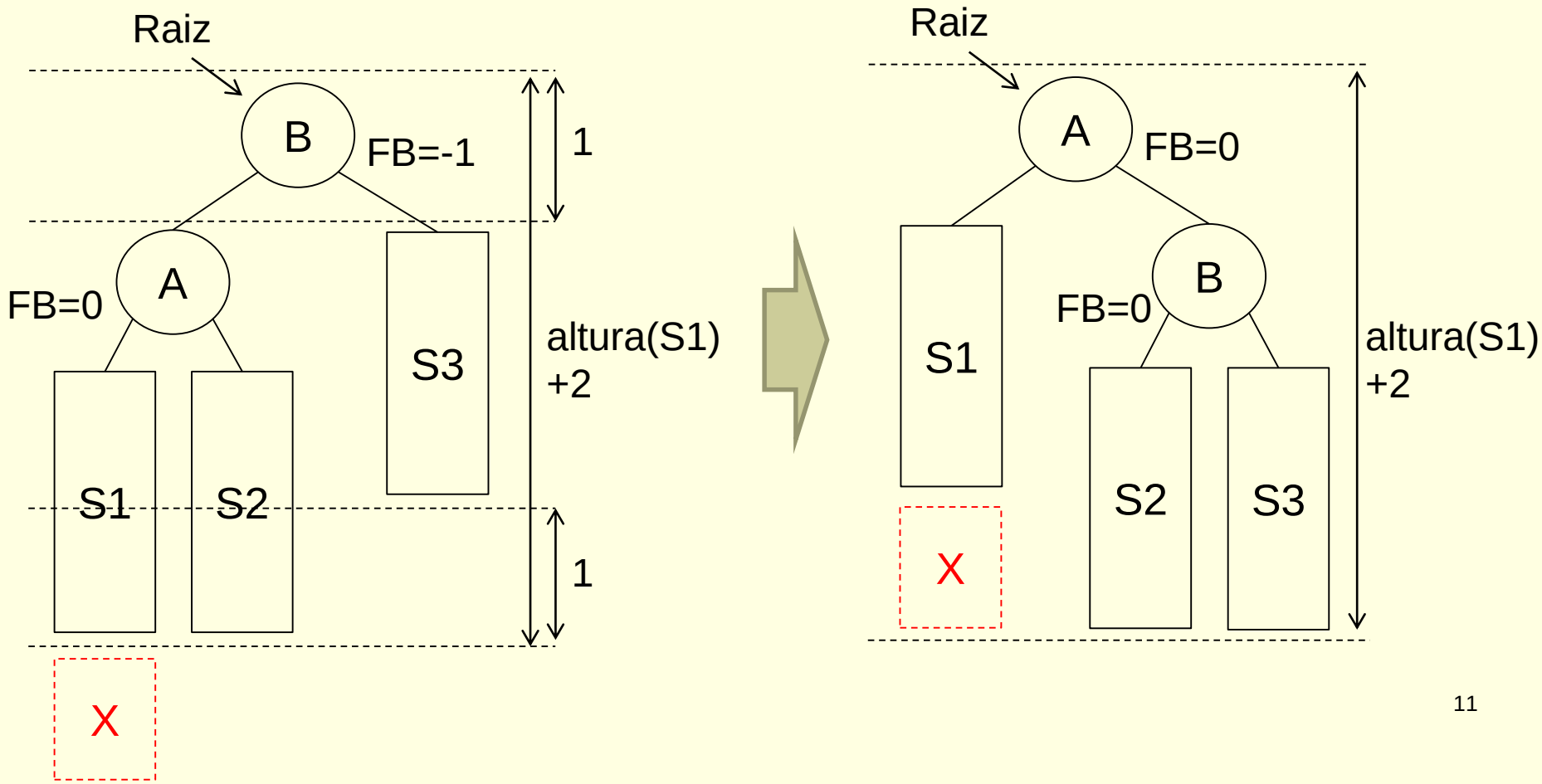
Árvore balanceada!!!

AVL: primeiro caso

- Quando subárvores do pai e filho pendem para um mesmo lado
  - Rotação simples para o lado oposto
    - EE ou DD (Direita-Direita, com raciocínio inverso)
  - Às vezes, é necessário realocar algum elemento, pois ele perde seu lugar na árvore

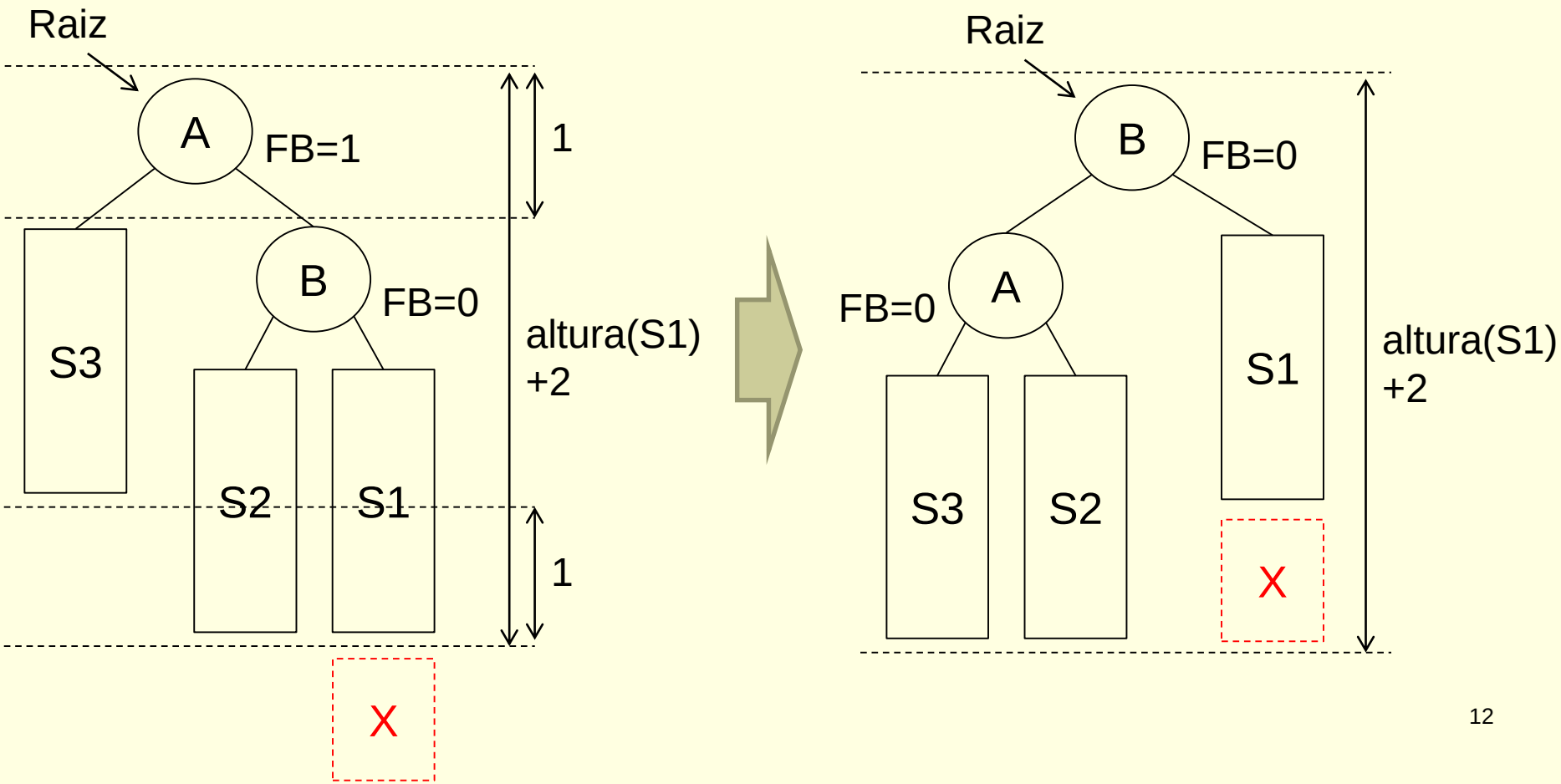
Formalmente: EE

Generalizando



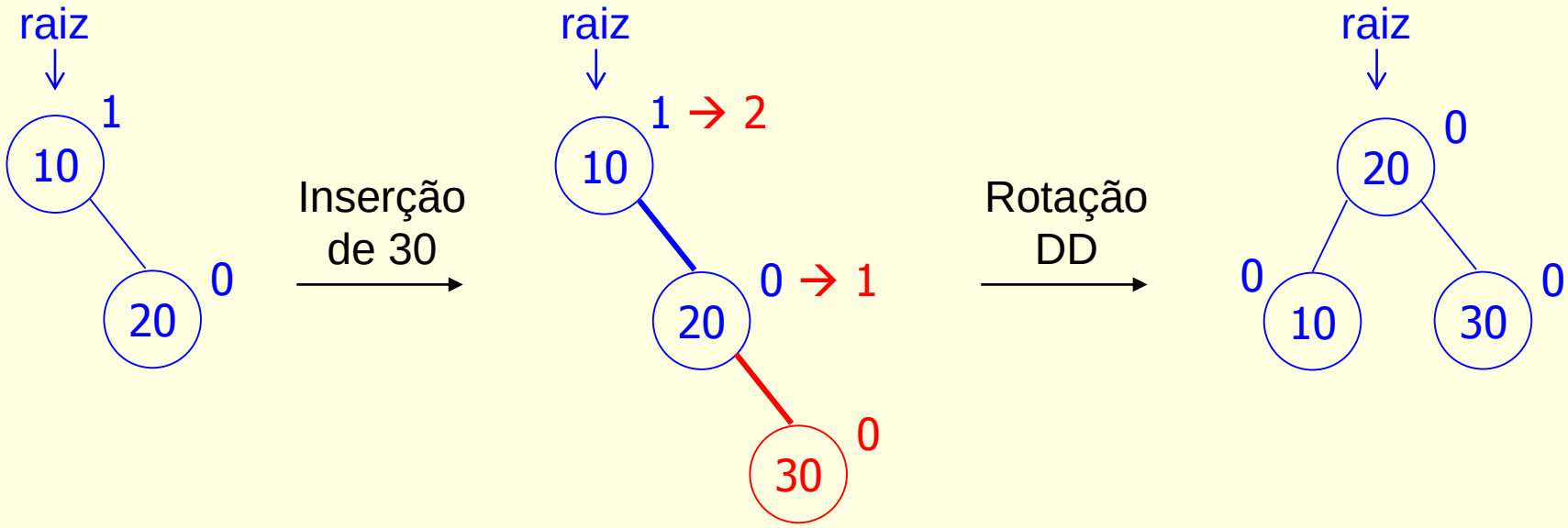
Formalmente: DD (similar)

Generalizando



Exercício

- Passo a passo: implementar rotação DD



Exercício

```
void DD(no **r) {
    no *pai=*r;
    no *filho=pai->dir;
    pai->dir=filho->esq;
    filho->esq=pai;
    pai->fb=0;
    filho->fb=0;
    *r=filho;
}
```

Exercício

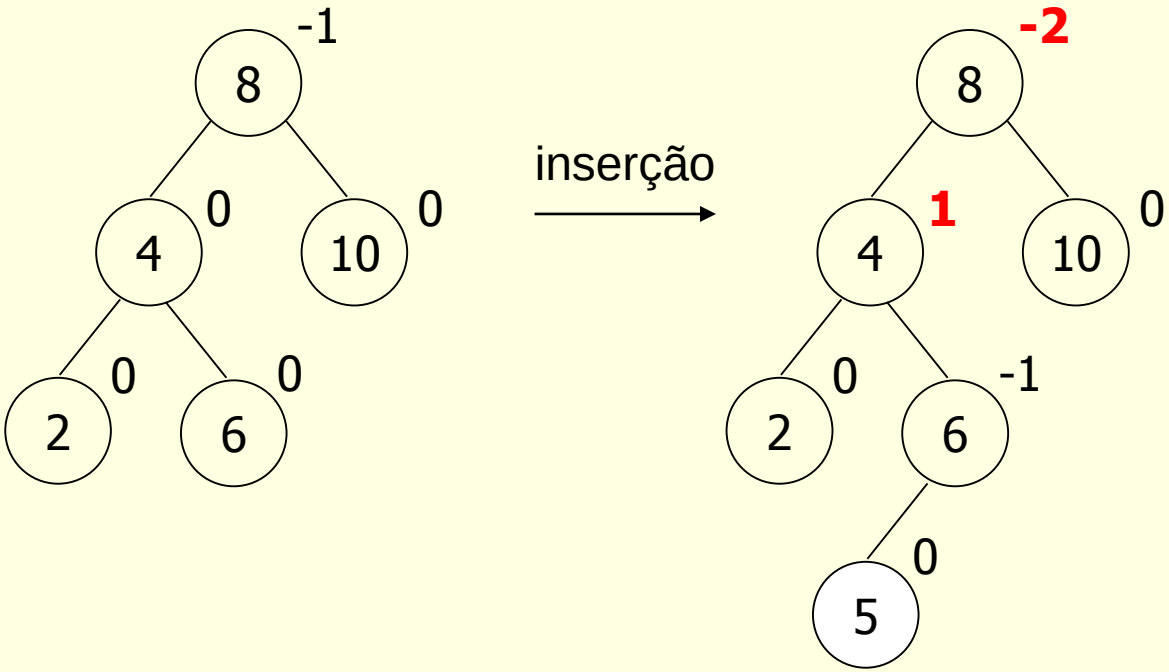
- Passo a passo: implementar rotação EE
  - Raciocínio similar

Exercício

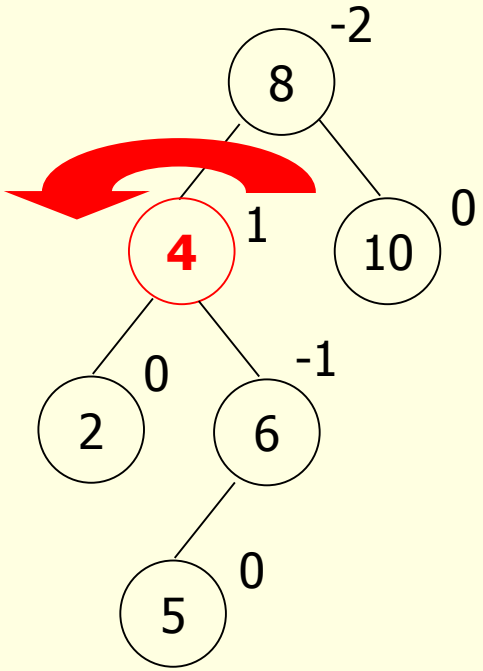
```
void EE(no **r) {
    no *pai=*r;
    no *filho=pai->esq;
    pai->esq=filho->dir;
    filho->dir=pai;
    pai->fb=0;
    filho->fb=0;
    *r=filho;
}
```

AVL: segundo caso

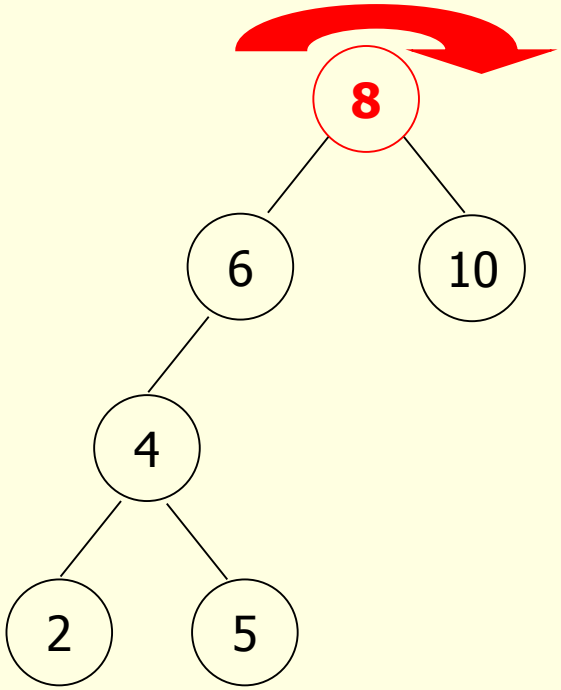
- Raiz de uma subárvore com FB -2 (ou 2) e um nó filho com FB 1 (ou -1)
  - Os fatores de balanceamento têm sinais opostos: subárvore de nó raiz pende para um lado e subárvore de nó filho pende para o outro (ou o contrário)



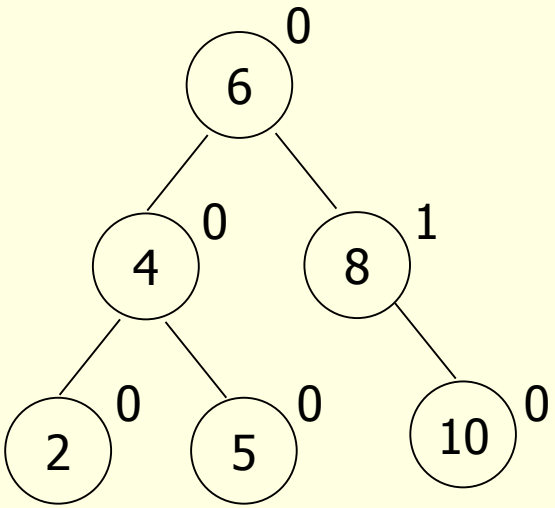
AVL: segundo caso



Rotação do nó filho
para a esquerda
(nó com FB=1 ou -1)

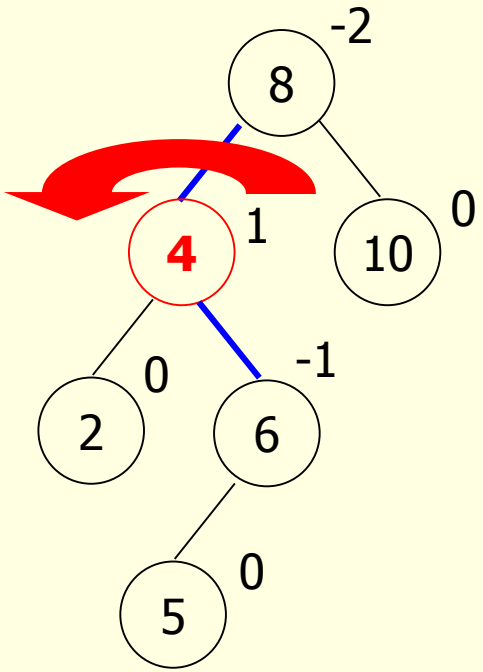


Rotação do nó pai
para a direita
(nó com FB=-2 ou 2)



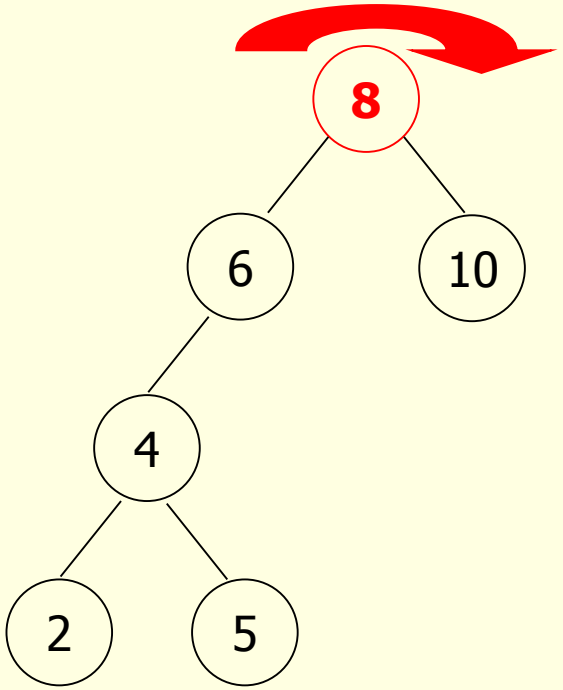
Árvore balanceada!!!

AVL: segundo caso

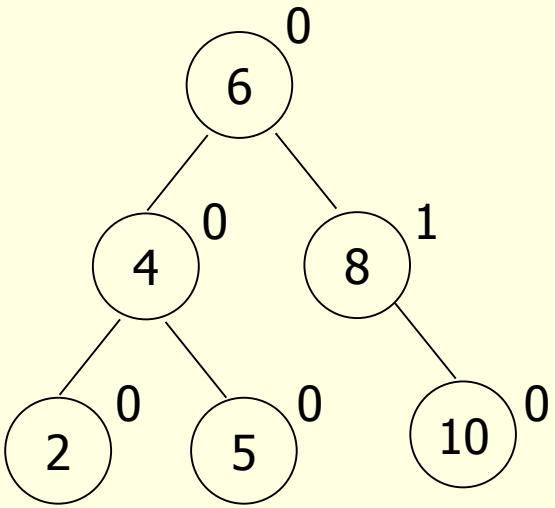


Rotação do nó filho
para a esquerda
(nó com FB=1 ou -1)

Rotação dupla ED
(Esquerda-Direita)



Rotação do nó pai
para a direita
(nó com FB=-2 ou 2)

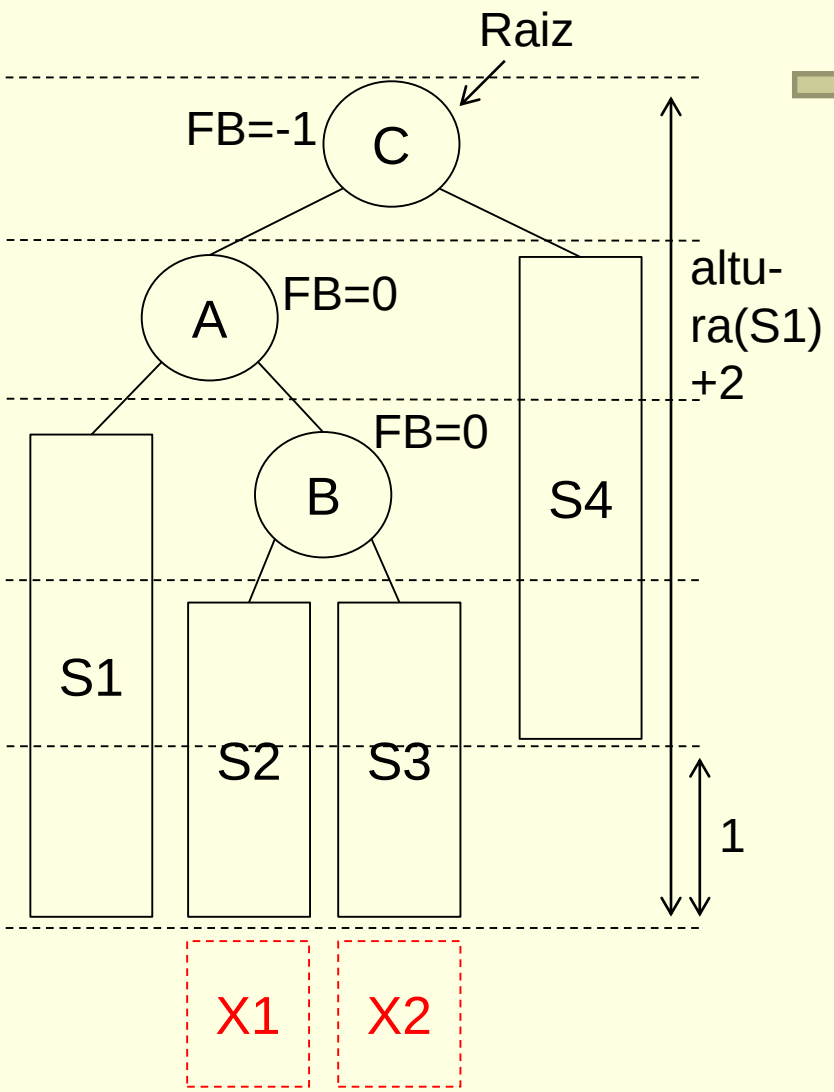


Árvore balanceada!!!

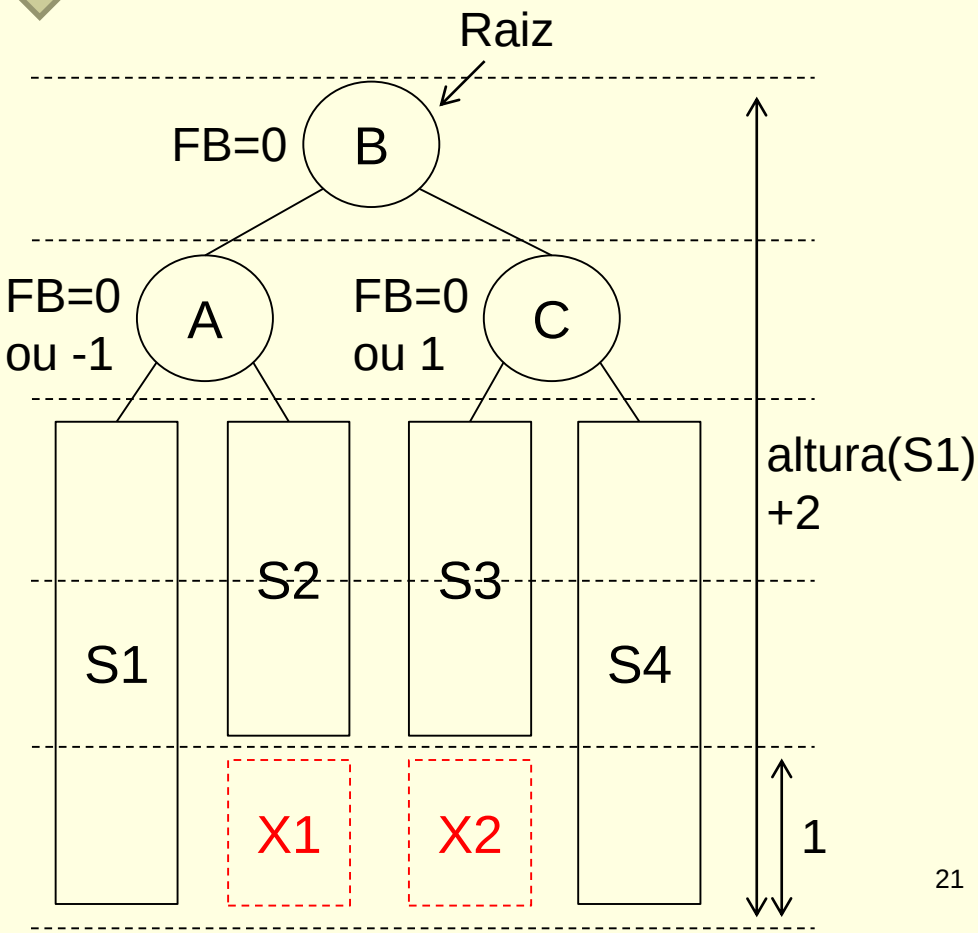
AVL: segundo caso

- Quando subárvores do pai e filho pendem para lados opostos
  - Rotação dupla
    - Primeiro, rotaciona-se o filho para o lado do desbalanceamento do pai
    - Em seguida, rotaciona-se o pai para o lado oposto do desbalanceamento
      - ED ou DE (com raciocínio inverso)
  - Às vezes, é necessário realocar algum elemento, pois ele perde seu lugar na árvore

Formalmente: ED

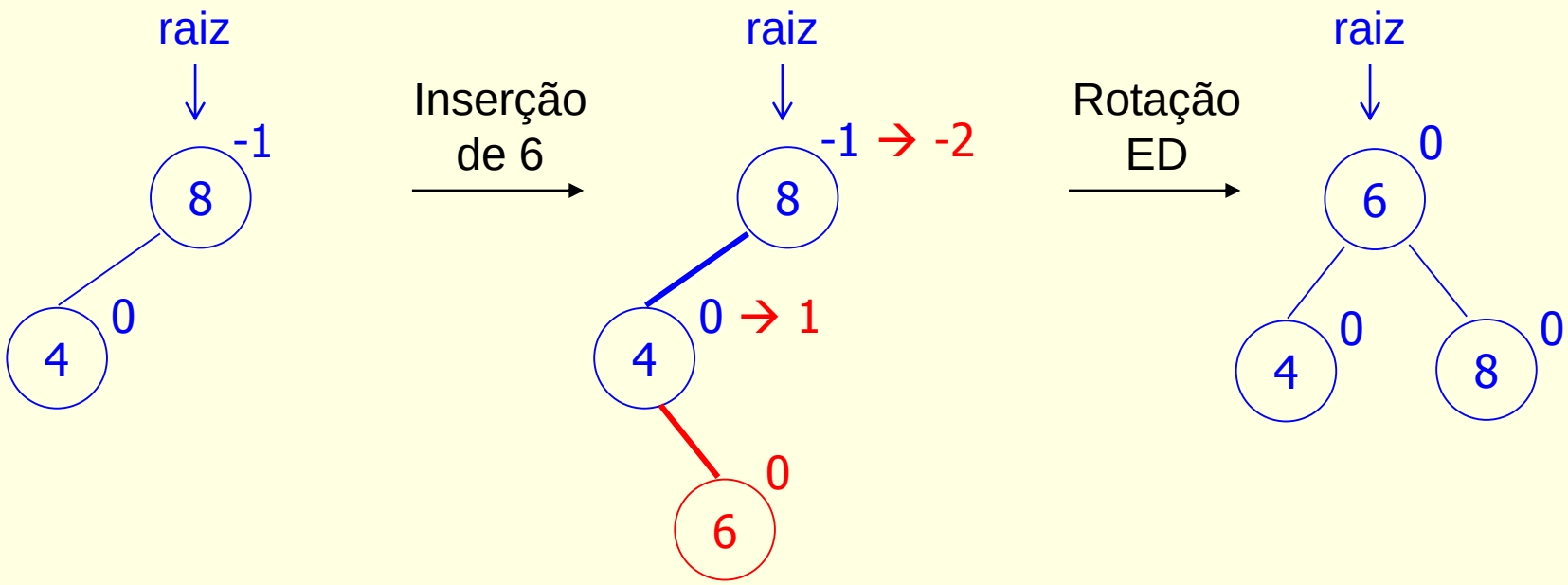


Generalizando



Exercício

- Passo a passo: implementar rotação ED

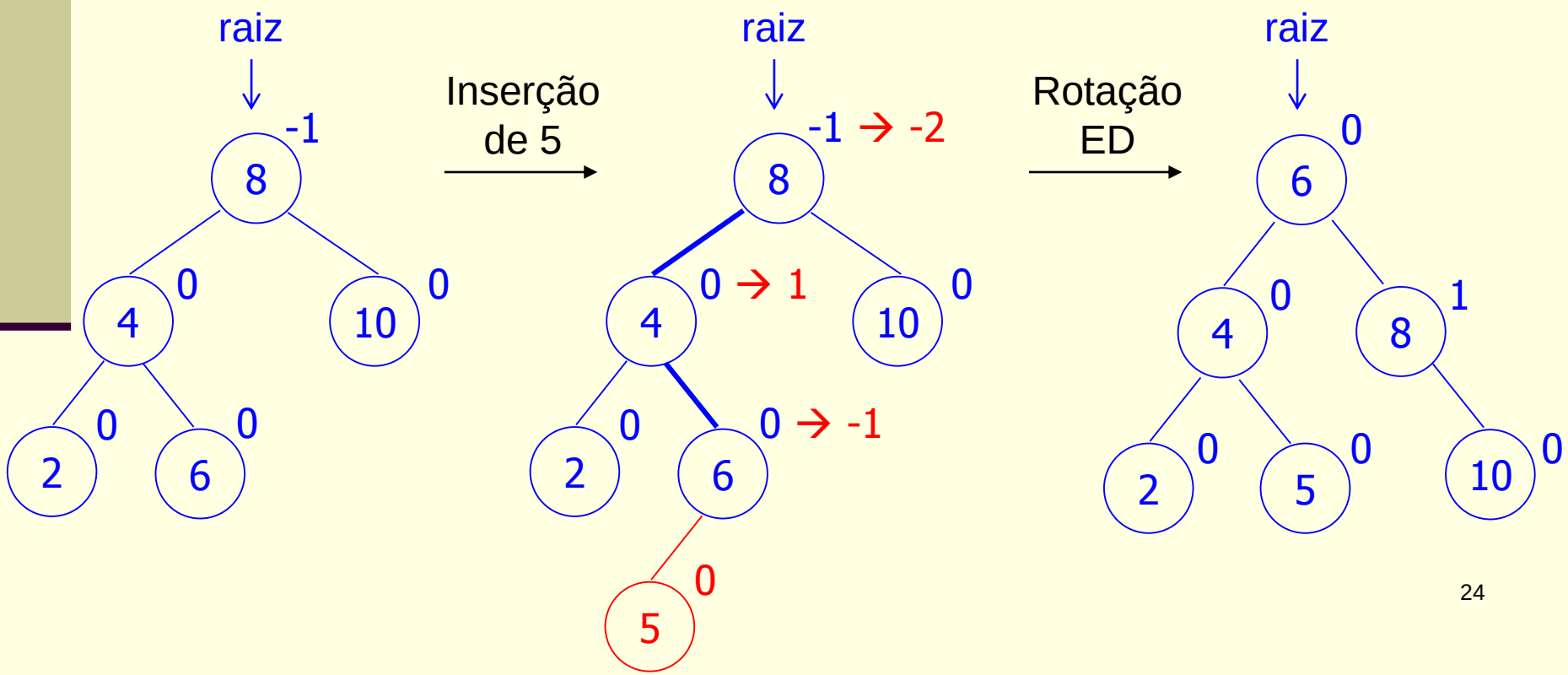


Exercício

```
void ED(no **r) {
    no *pai=*r;
    no *filho=pai->esq;
    no *neto=filho->dir;
    pai->esq=neto->dir;
    filho->dir=neto->esq;
    neto->esq=filho;
    neto->dir=pai;
    switch (neto->fb) {
        case -1:
            pai->fb=1;
            filho->fb=0;
            break;
        case 0:
            pai->fb=0;
            filho->fb=0;
            break;
        case 1:
            pai->fb=0;
            filho->fb=-1;
            break;
    }
    neto->fb=0;
    *r=neto;
}
```

Exercício

- Testar sub-rotina no caso abaixo



Formalmente: DE

- Raciocínio similar

Sub-rotina DE

```
void DE(no **r) {
    no *pai=*r;
    no *filho=pai->dir;
    no *neto=filho->esq;
    filho->esq=neto->dir;
    pai->dir=neto->esq;
    neto->esq=pai;
    neto->dir=filho;
    switch (neto->fb) {
        case -1:
            pai->fb=0;
            filho->fb=1;
            break;
        case 0:
            pai->fb=0;
            filho->fb=0;
            break;
        case 1:
            pai->fb=-1;
            filho->fb=0;
            break;
    }
    neto->fb=0;
    *r=neto;
}
```

AVL

- As transformações dos casos anteriores diminuem em 1 a altura da subárvore com raiz desbalanceada p

- Assegura-se o rebalanceamento de todos os ancestrais de p e, portanto, o rebalanceamento da árvore toda

AVL

- Implementar sub-rotina de inserção de elementos na AVL
  - Função principal de inserção
    - Usando sub-rotinas de rotações e conferindo balanceamento recursivamente

Função principal

```
int inserir(no **p, int *x) {
  int cresceu;
  return aux_insere(p, x, &cresceu);
}
```

```
int aux_insere(no **p, int *x, int *cresceu) {
    if (*p==NULL) {
       *p=(no*) malloc(sizeof(no));
       (*p)->info=*x;
       (*p)->fb=0;
       (*p)->esq=NULL;
       (*p)->dir=NULL;
       *cresceu=1;
       return 1;
    }
    else if (*x==(*p)->info)
         return 0;
    else if (*x<(*p)->info) {
         if (aux_insere(&(*p)->esq,x,cresceu)) {
            if (*cresceu) {
               switch ((*p)->fb) {
                      case -1:
                           if ((*p)->esq->fb==-1)
                              EE(p);
                           else ED(p);
                           *cresceu=0;
                           break;
                      case 0:
                           (*p)->fb=-1;
                           *cresceu=1;
                           break;
                      case 1:
                           (*p)->fb=0;
                           *cresceu=0;
                           break;
               }
            }
            return 1;
         }
         else return 0;
    }
    ... //continua
```

Função principal

```
int inserir(no **p, int *x) {
  int cresceu;
  return aux_insere(p, x, &cresceu);
}
```

```
    else {
        if (aux_insere(&(*p)->dir,x,cresceu)) {
            if (*cresceu) {
                switch ((*p)->fb) {
                        case -1:
                                (*p)->fb=0;
                                *cresceu=0;
                                break;
                        case 0:
                                (*p)->fb=1;
                                *cresceu=1;
                                break;
                        case 1:
                                if ((*p)->dir->fb==1)
                                   DD(p);
                                else DE(p);
                                *cresceu=0;
                                break;
                }
            }
            return 1;
        }
        else return 0;
    }
}
```

AVL

- Exercício: teste a sub-rotina anterior inserindo os elementos **5** e **40** na árvore abaixo