

ÁRVORES DE ADELSON-VELSKII E LANDIS (AVL)

Prof. Joaquim Uchôa
Profa. Juliana Gregghi
Prof. Renato Ramos



- Visão geral
- Rotações
- Implementação da AVL

VISÃO GERAL



ÁRVORES BINÁRIAS E BALANCEAMENTO

- Árvores binárias de busca têm por objetivo prover acesso rápido à informação.
- O ideal é que a árvore esteja o mais equilibrada possível, ou seja, com aproximadamente a mesma altura nas subárvores direita e esquerda.

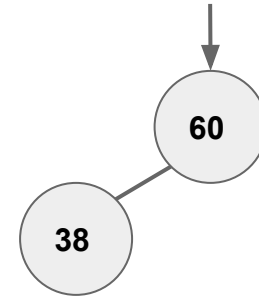
UM EXEMPLO PROBLEMÁTICO - I

Imagine a construção de uma ABB, com a inserção dos seguintes valores, nessa ordem:

60, 38, 55, 44, 52, 48, 50, 49

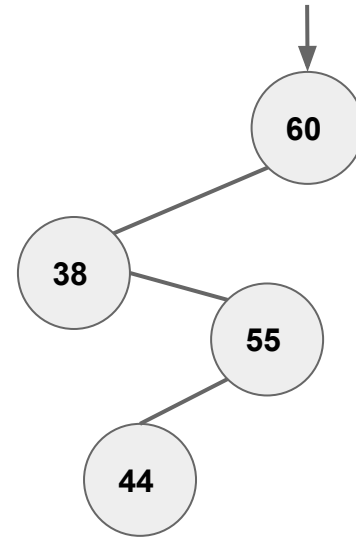
UM EXEMPLO PROBLEMÁTICO - II

~~60~~, ~~38~~, 55, 44,
52, 48, 50, 49



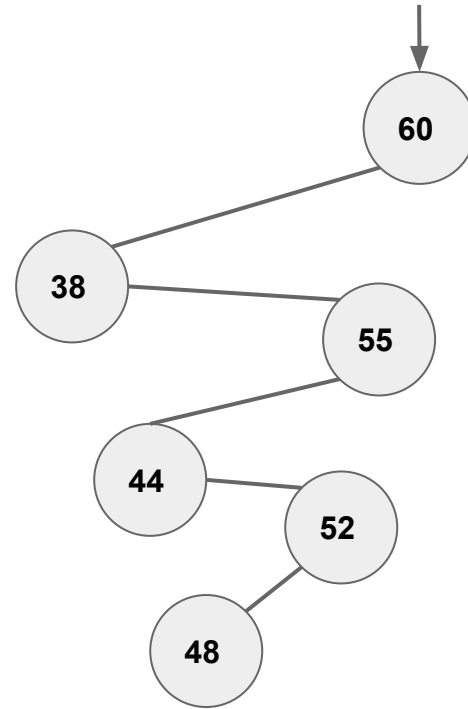
UM EXEMPLO PROBLEMÁTICO - III

~~60~~, ~~38~~, ~~55~~, ~~44~~,
52, 48, 50, 49



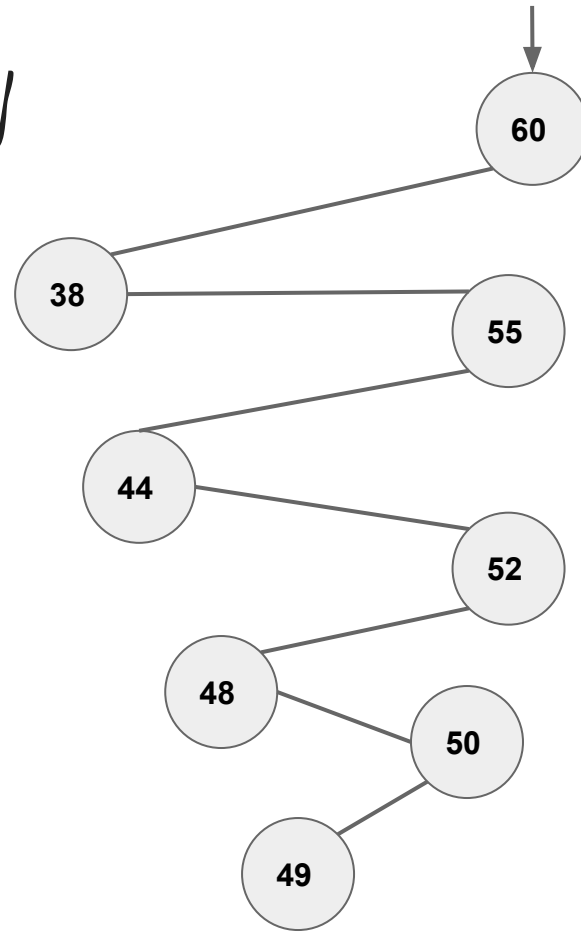
UM EXEMPLO PROBLEMÁTICO - IV

~~60~~, ~~38~~, ~~55~~, ~~44~~,
~~52~~, ~~48~~, 50, 49



UM EXEMPLO PROBLEMÁTICO - V

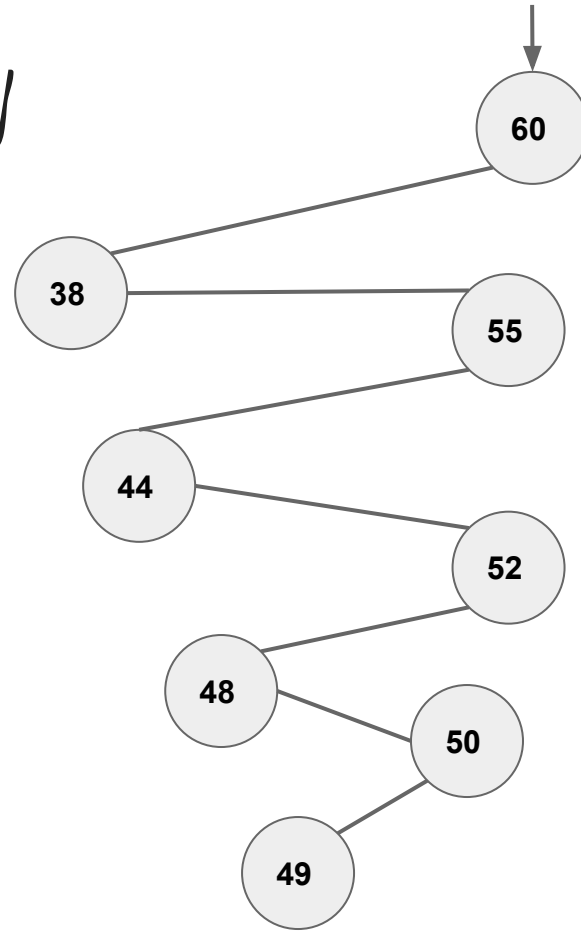
~~60~~, ~~38~~, ~~55~~, ~~44~~,
~~52~~, ~~48~~, ~~50~~, ~~49~~



UM EXEMPLO PROBLEMÁTICO - V

~~60~~, ~~38~~, ~~55~~, ~~44~~,
~~52~~, ~~48~~, ~~50~~, ~~49~~

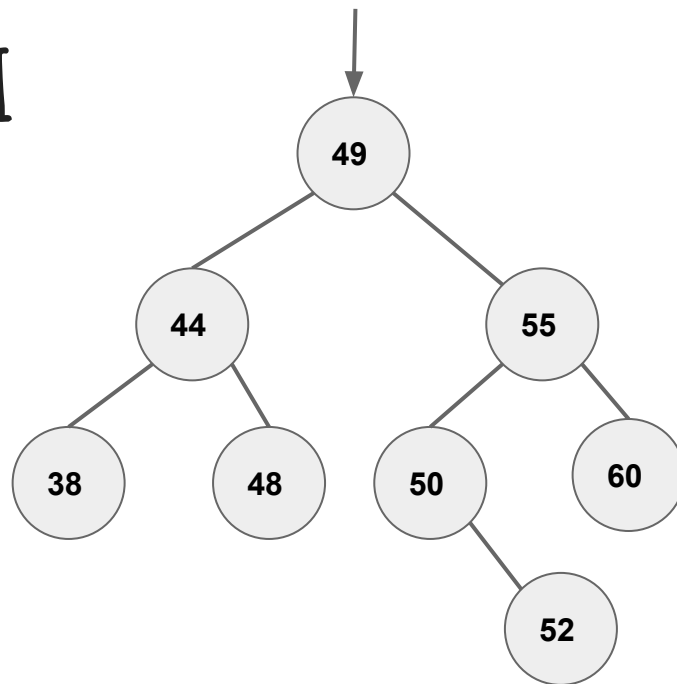
A árvore se degenerou em uma lista, a busca é bem menos ineficiente que uma abb balanceada, se tornando uma busca em uma lista encadeada.



UM EXEMPLO PROBLEMÁTICO - VI

~~60~~, ~~38~~, ~~55~~, ~~44~~,
~~52~~, ~~48~~, ~~50~~, ~~49~~

Esses mesmos dados poderiam gerar a árvore ao lado, se tivessem sido inseridos de outra forma, ou se fossem rearranjados. A busca nessa árvore é bem mais eficiente!

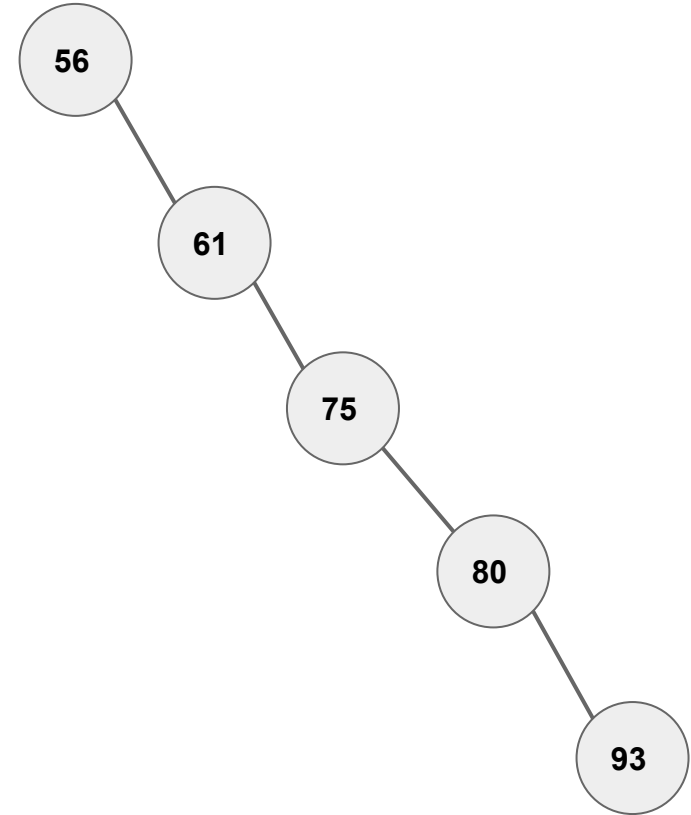
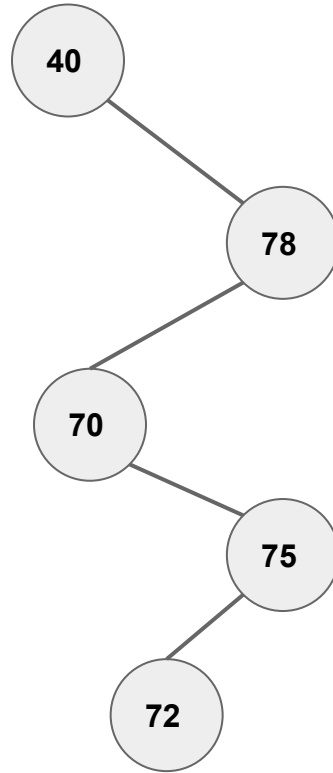
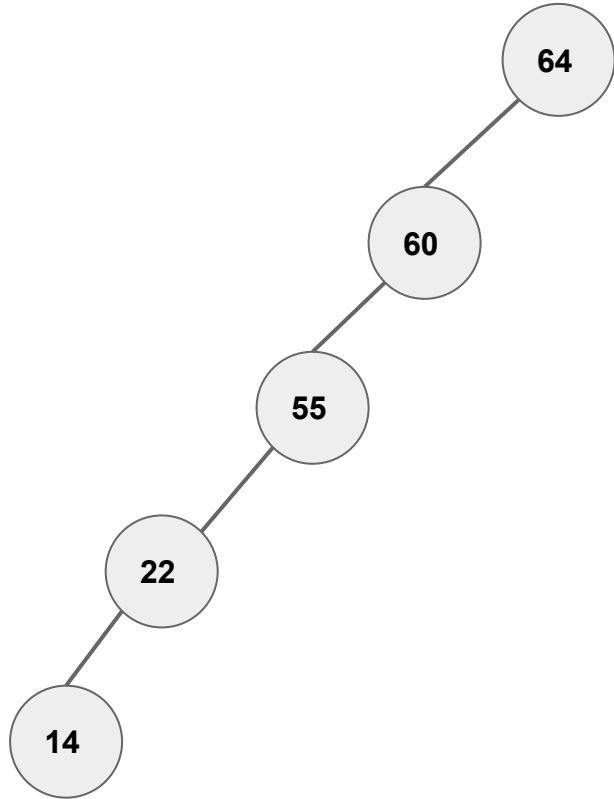


ÁRVORES BINÁRIAS E BALANCEAMENTO

Após muitas inclusões e remoções de elementos a árvore resultante pode ser degenerada, ou seja, desequilibrada.

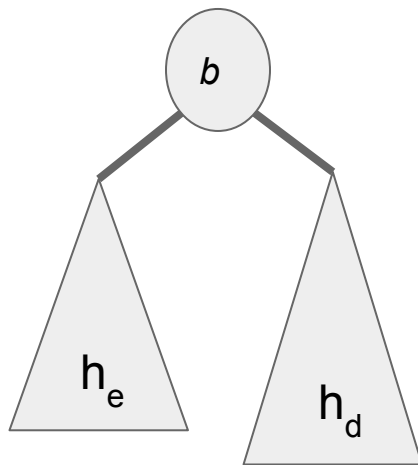
Para garantir que isso não ocorra, é preciso que os nós da árvore sejam reorganizados.

EXEMPLOS DE ÁRVORES BINÁRIAS DEGENERADAS



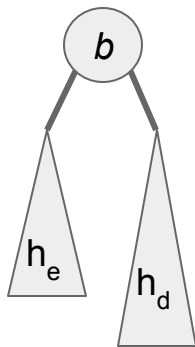
CONCEITO DE BALANCEAMENTO

O balanceamento de um nó diz respeito à diferença de altura das subárvores esquerda(h_e) e direita(h_d).



CONCEITO DE BALANCEAMENTO

O processo de reorganização de uma árvore pode ser bastante “caro” no que diz respeito à quantidade de operações e geralmente envolve rotação entre os nós.



ALTURA DE UM NÓ - I

A altura de um nó é dada pela altura da subárvore em que ele é raiz.

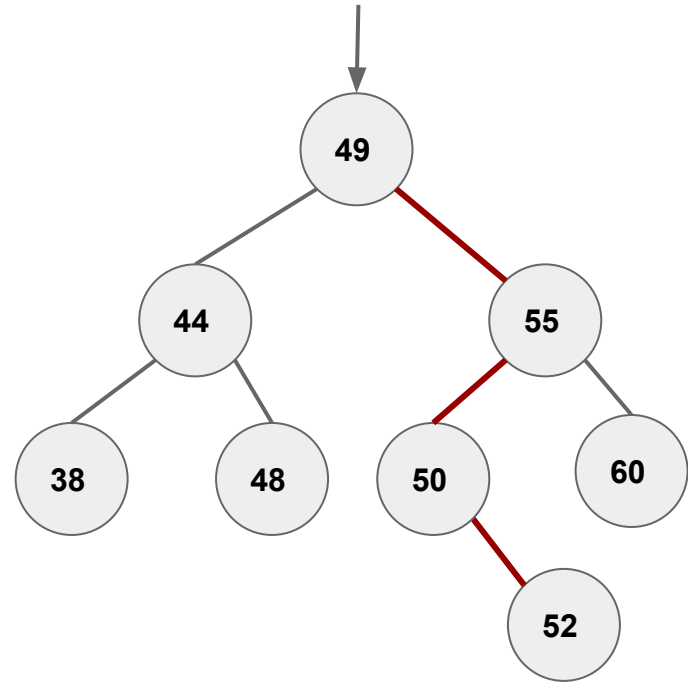
Nesse caso a altura de um nó é o tamanho do maior caminho (o número de arestas) dele até suas folhas.

Em algumas situações, consideram-se que as folhas são nulas, portanto os nós folhas não-nulos tem altura 1.

ALTURA DE UM NÓ - II

No exemplo ao lado, o 49 tem duas arestas até o 38, o 48 e o 60, mas tem três arestas até o 52.

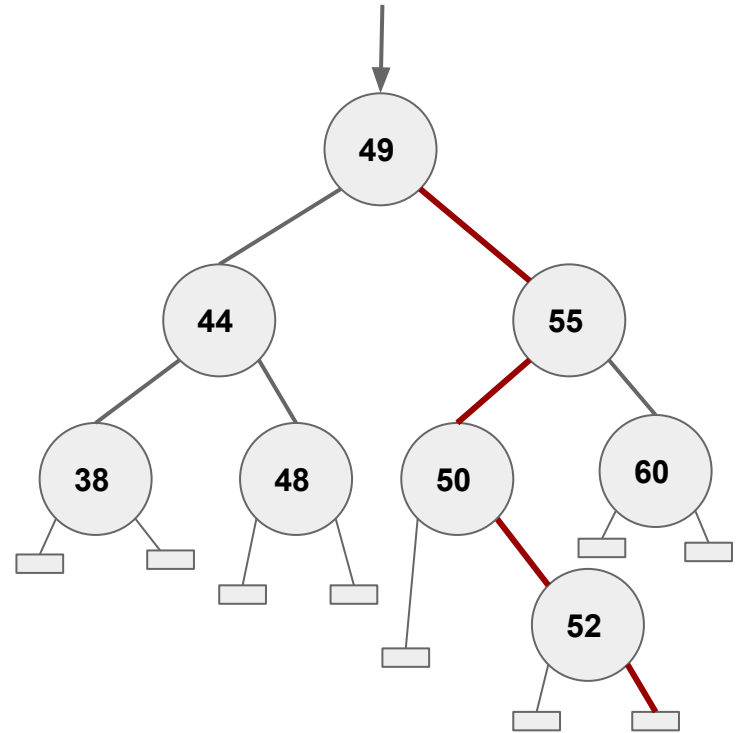
Assim, sua altura, não considerando folhas nulas, seria 3.



ALTURA DE UM NÓ - III

Considerando as folhas nulas, tem-se uma aresta a mais. Nesse caso, sua altura seria 4.

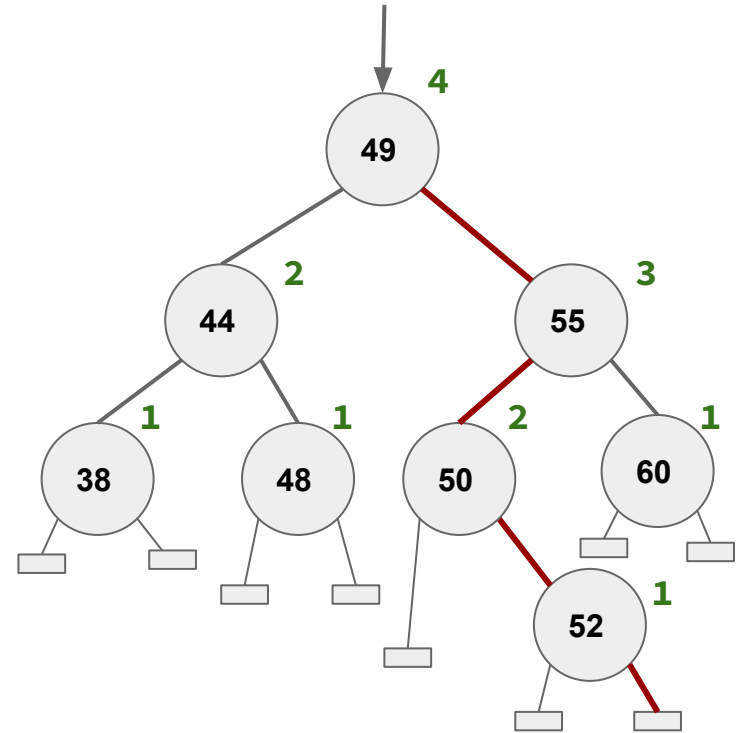
Em geral, não se consideram as folhas nulas, mas alguns algoritmos (como as AVL) fazem uso desse recurso para simplificar a implementação.



ALTURA DE UM NÓ - IV

Uma forma prática de calcular a altura de um nó é dada pelo fato que sua altura é dada pela maior altura entre seus dois filhos somado a uma unidade.

Na figura ao lado, tem-se a altura de cada nó, considerando-se folhas nulas.



AVL - ADELSON-VELSKII E LANDIS (1962)

Uma árvore AVL é uma árvore binária de busca altamente balanceada, em que a cada inclusão ou remoção de elementos, procura-se reorganizar os nós de forma a mantê-la balanceada.

AVL - ADELSON-VELSKII E LANDIS (1962)

A diferença entre as alturas da subárvores esquerda(H_{esq}) e direita(H_{dir}) deve ser de, no máximo, uma unidade.

A diferença entre as alturas da subárvores esquerda e direita a partir de um determinado nó é denominado *fator de balanceamento*(**FB**) do nó, e não pode ser menor que -1 ou maior que 1.

$$\mathbf{Fb} = H_{\text{esq}} - H_{\text{dir}}$$

AVL - ADELSON-VELSKII E LANDIS (1962)

A diferença entre as alturas da subárvores esquerda(H_{esq}) e direita(H_{dir}) deve ser de, no máximo, uma unidade.

A diferença entre as alturas da subárvores esquerda e direita a cada nó é denominado *fator de balanceamento*. Este fator não pode ser menor que -1 nem maior que 1.

Algumas implementações utilizam

$$Fb = H_{\text{dir}} - H_{\text{esq}}$$

para cálculo do fator de balanceamento. Isso é irrelevante desde que isso seja mantido em toda a implementação, com os devidos ajustes efetuados.

$$Fb = H_{\text{esq}} - H_{\text{dir}}$$

IMPLEMENTAÇÃO DE AVLs - 1/3

A implementação de AVLs faz uso de um atributo de altura em cada nó, para garantir o balanceamento entre subárvores esquerda e direita.

Para tornar o código mais simples e eficiente, implementações em geral consideram que qualquer nó não-nulo possui no mínimo altura 1. Nesse caso, as implementações geralmente consideram que os nós folhas são nulos, e possuem, nesse caso, altura 0.

IMPLEMENTAÇÃO DE AVLs - 2/3

Operações de alteração na árvore, como inserção e remoção de nós, implicam em atualizar a altura dos nós envolvidos no processo de alteração.

A atualização da altura é feita do nó folha até o raiz. Por conta disso, inclusive, a maior parte das implementações de AVL utilizam-se de métodos recursivos, com encadeamento simples.

IMPLEMENTAÇÃO DE AVLs - 3/3

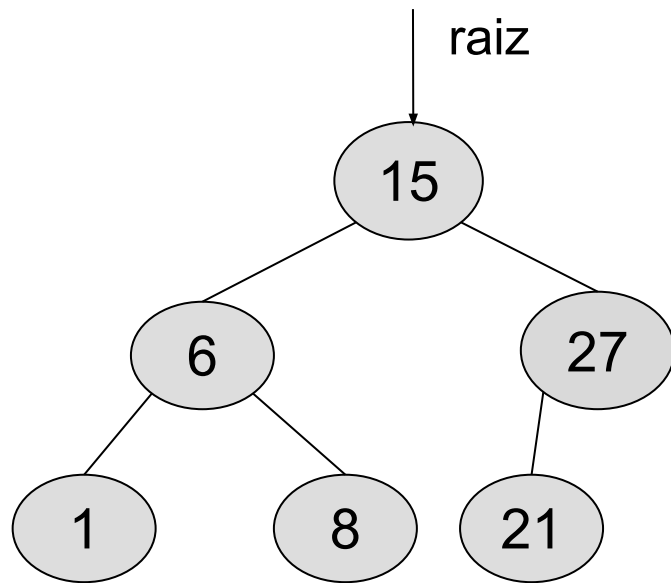
Por exemplo, após a inserção de um novo nó, recursivamente vai se ajustando as alturas dos nós envolvidos.

À medida que se sobe na árvore, até a raiz, cada nó atualiza sua altura e verifica se precisa ou não fazer algum ajuste para manter o balanceamento.

COMO ARRUMAR O DESBALANCEAMENTO? - I

A principal forma de resolver problemas de desbalanceamento é utilizando-se de rotações entre os nós envolvidos.

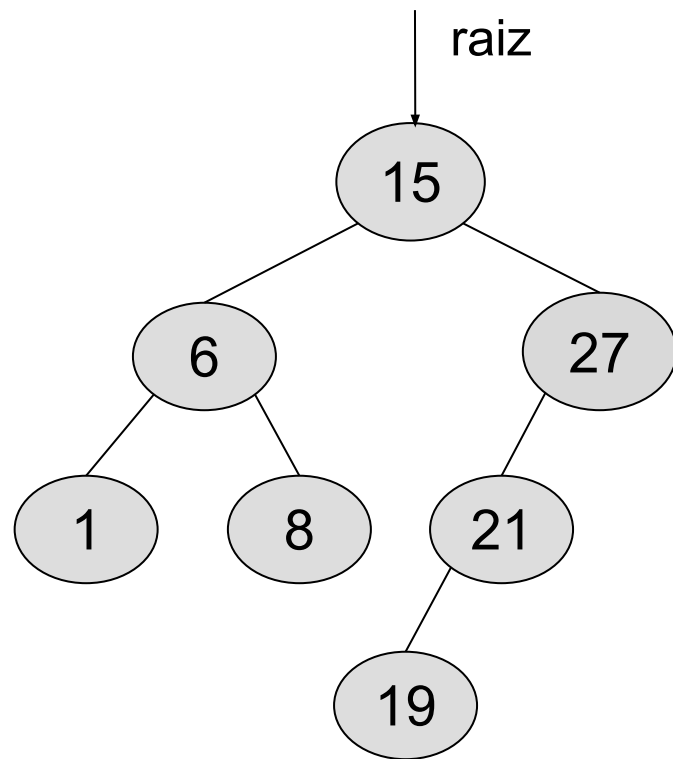
Suponha a árvore ao lado e queremos inserir o valor 19.



COMO ARRUMAR O DESBALANCEAMENTO? - II

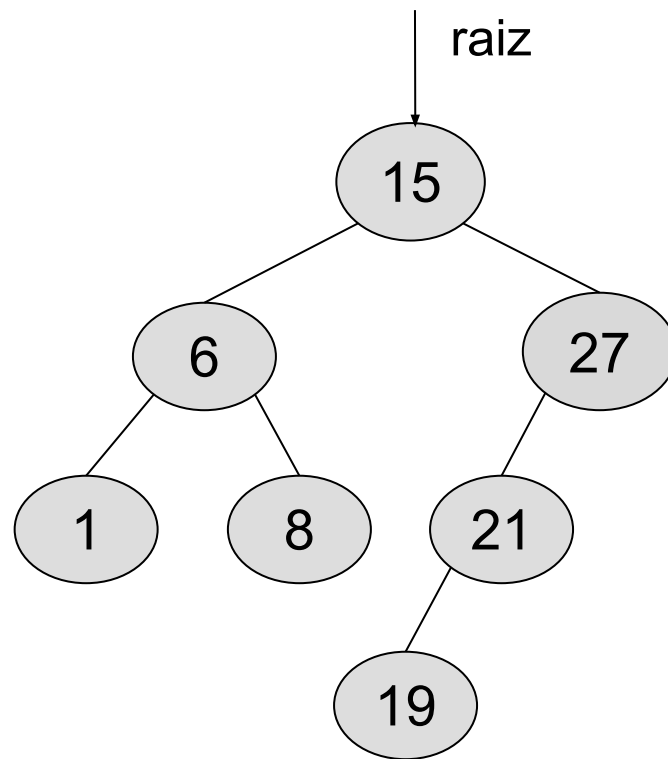
A inserção do 19 irá desbalancear a árvore, uma vez que o nó 27 terá uma árvore de altura 2 à esquerda e de altura 0 à direita...

Como resolver isso?



COMO ARRUMAR O DESBALANCEAMENTO? - III

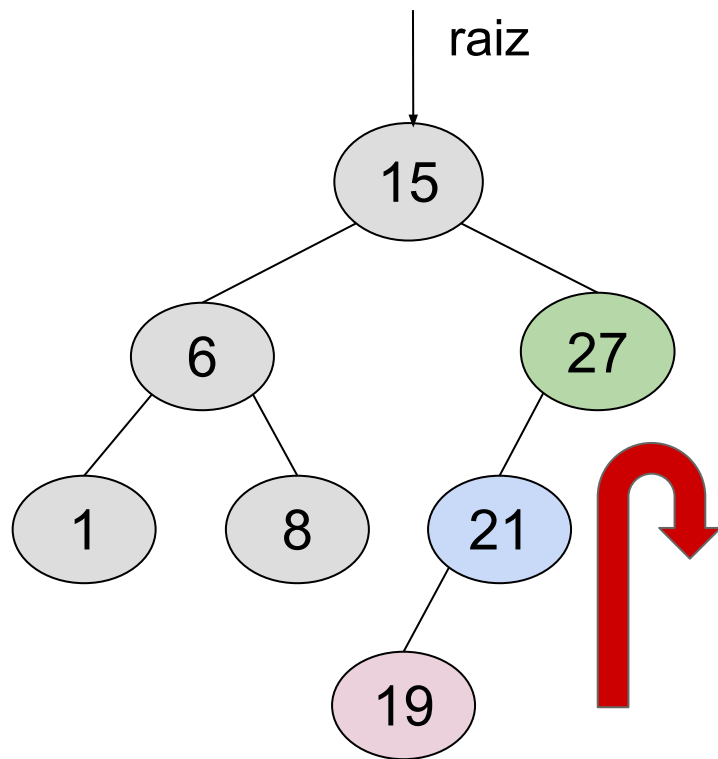
Percebam que o 21 é maior que 19 e menor que 27...



COMO ARRUMAR O DESBALANCEAMENTO? - III

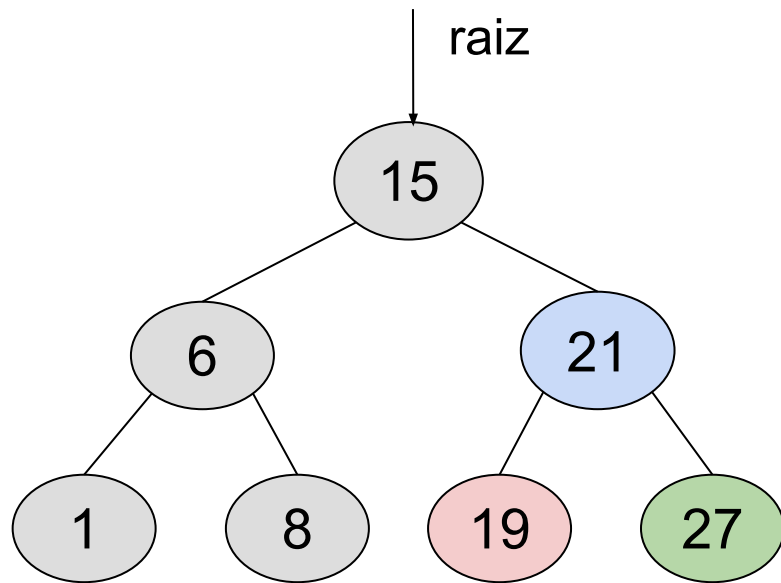
Percebam que o 21 é maior que 19 e menor que 27...

Ou seja, poderíamos colocar o 21 no lugar do 27, fazendo uma rotação!



COMO ARRUMAR O DESBALANCEAMENTO? - IV

Com a rotação, a árvore ficou balanceada novamente, e, ainda assim, foi mantida a propriedade básica da ABB (filhos menores à esquerda e filhos maiores à direita).



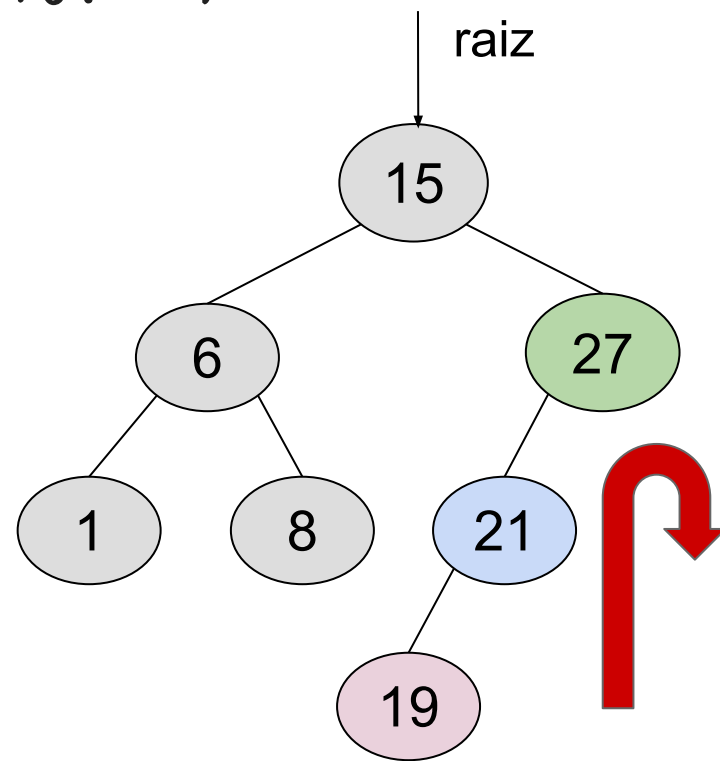
ROTAÇÕES



COMO ARRUMAR O DESBALANCEAMENTO? - V

No exemplo apresentado, tínhamos um desbalanceamento à esquerda e resolvemos isso fazendo uma rotação à direita.

Outros tipos de desbalanceamento irão precisar de outras rotações.

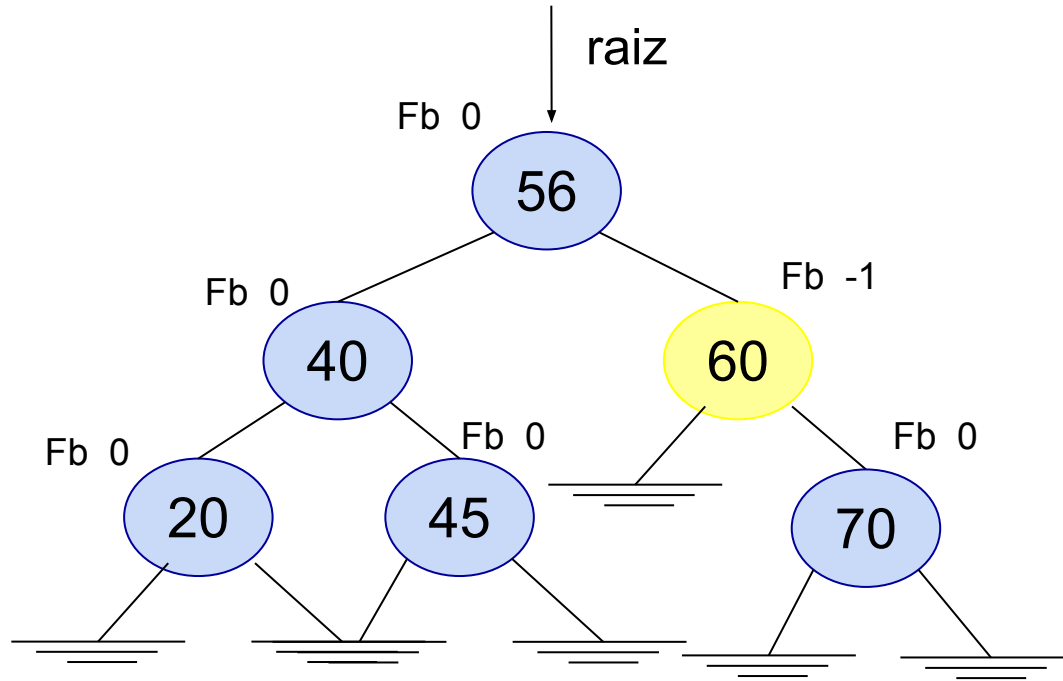


BALANCEAMENTO EM AVL

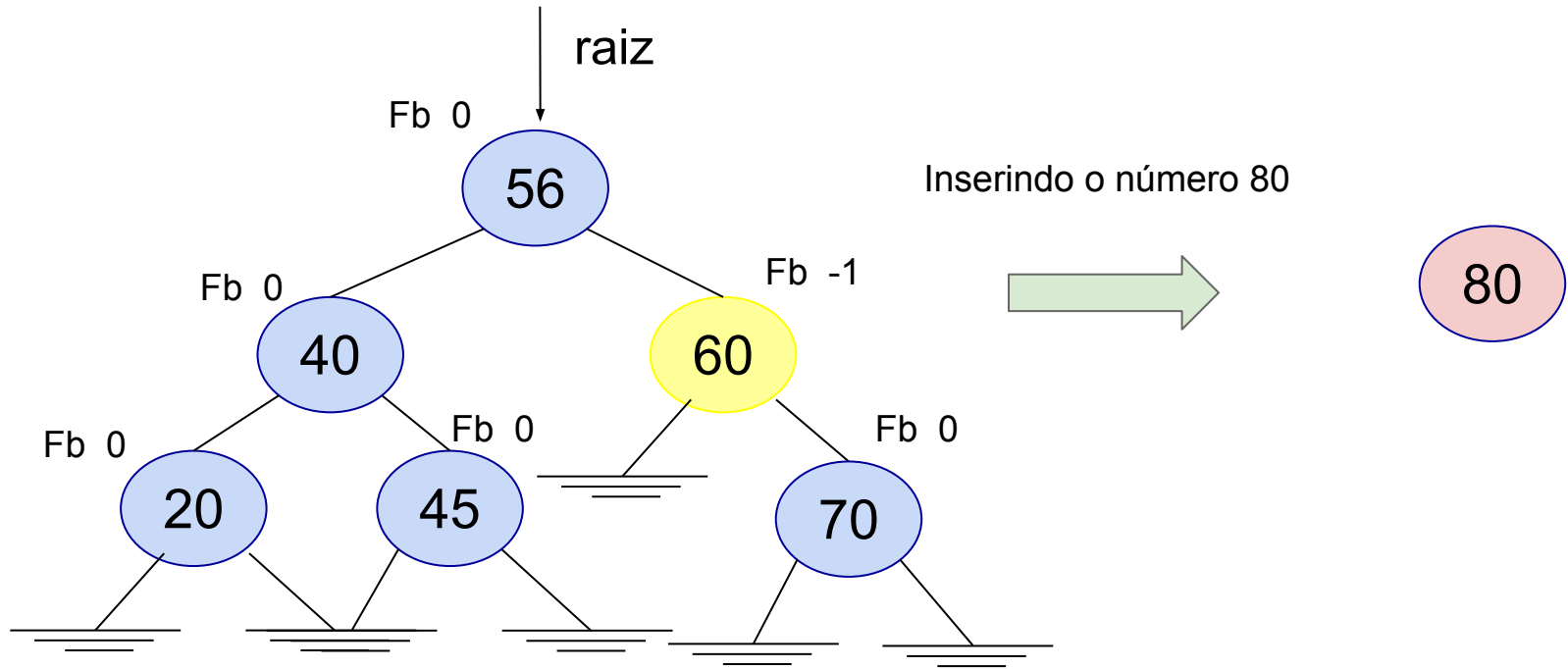
Caso ocorra um desbalanceamento, é necessário reorganizar a árvore AVL. O balanceamento dos nós é obtido através de operações de rotação dos nós:

- Rotação simples à esquerda
- Rotação simples à direita
- Rotação esquerda - direita (também denominada dupla à direita)
- Rotação direita - esquerda (também denominada dupla à esquerda)

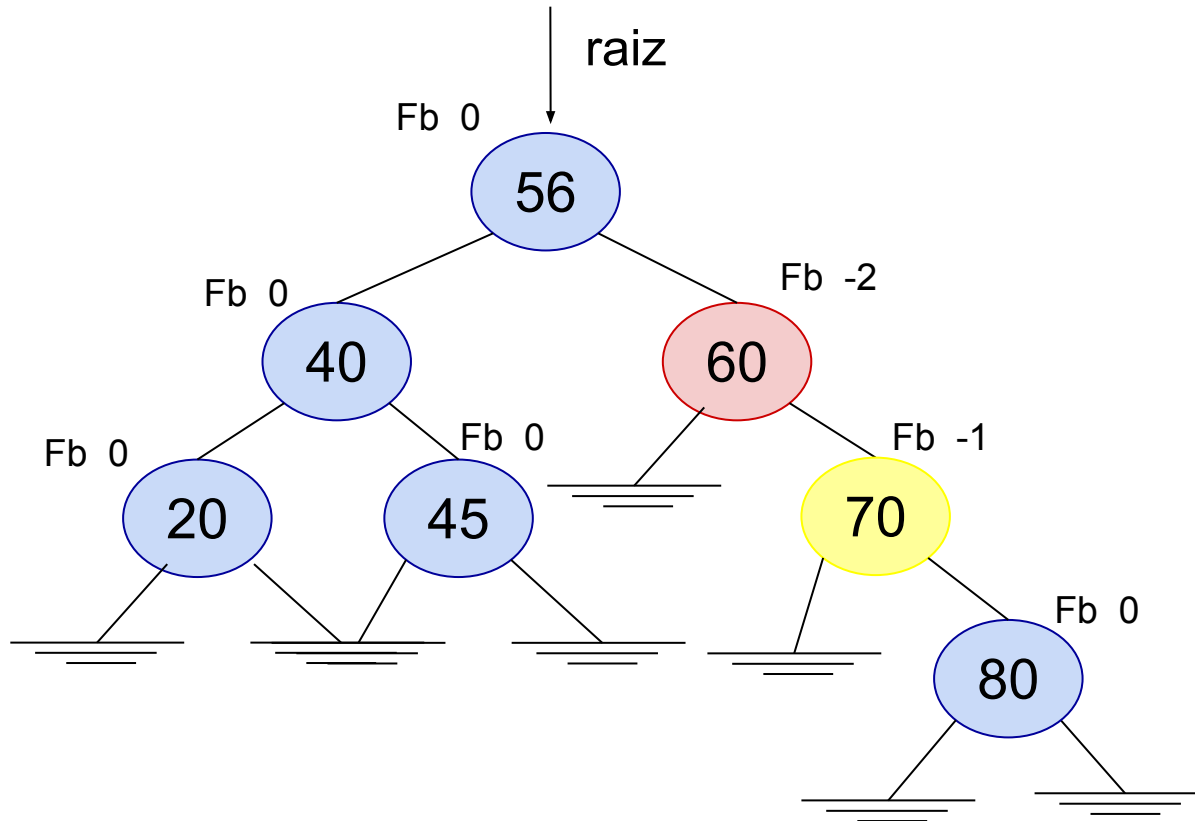
ROTAÇÃO SIMPLES À ESQUERDA - I



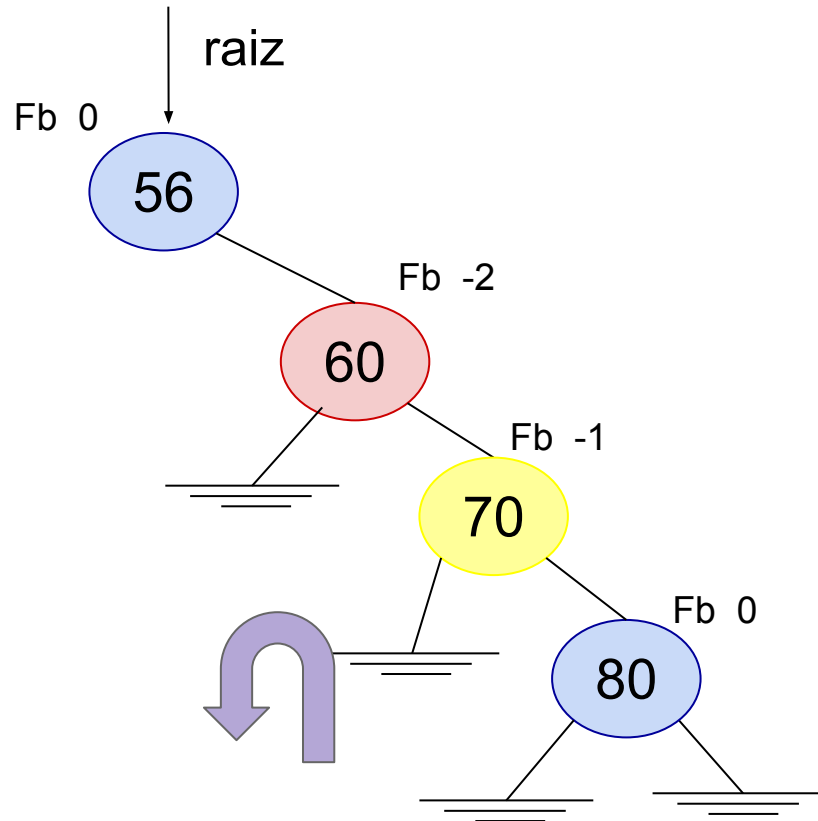
ROTAÇÃO SIMPLES À ESQUERDA - II



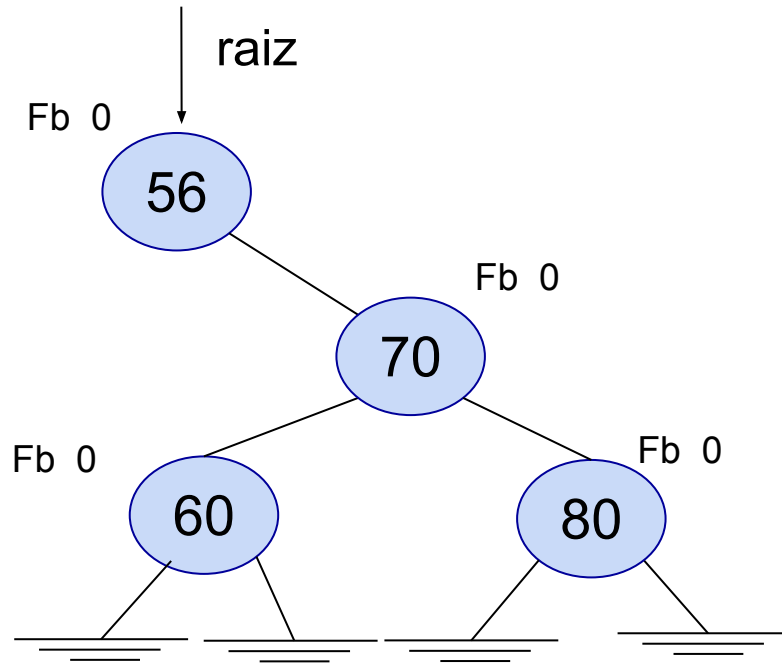
ROTAÇÃO SIMPLES À ESQUERDA - III



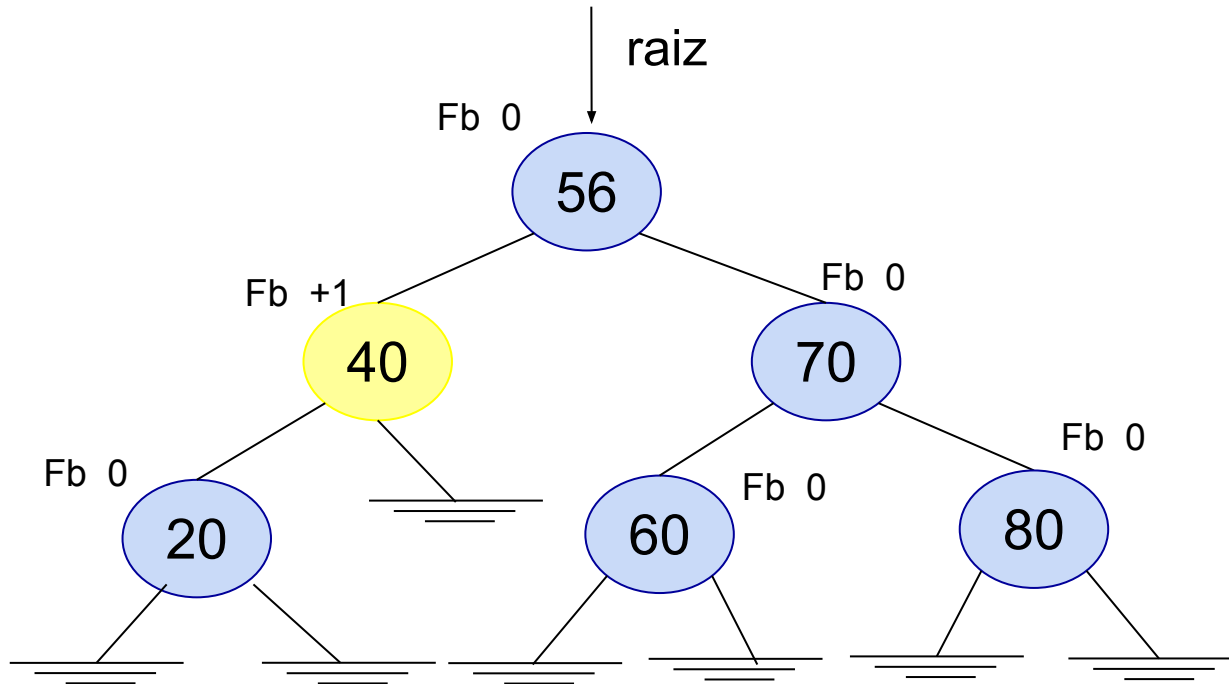
ROTAÇÃO SIMPLES À ESQUERDA - IV



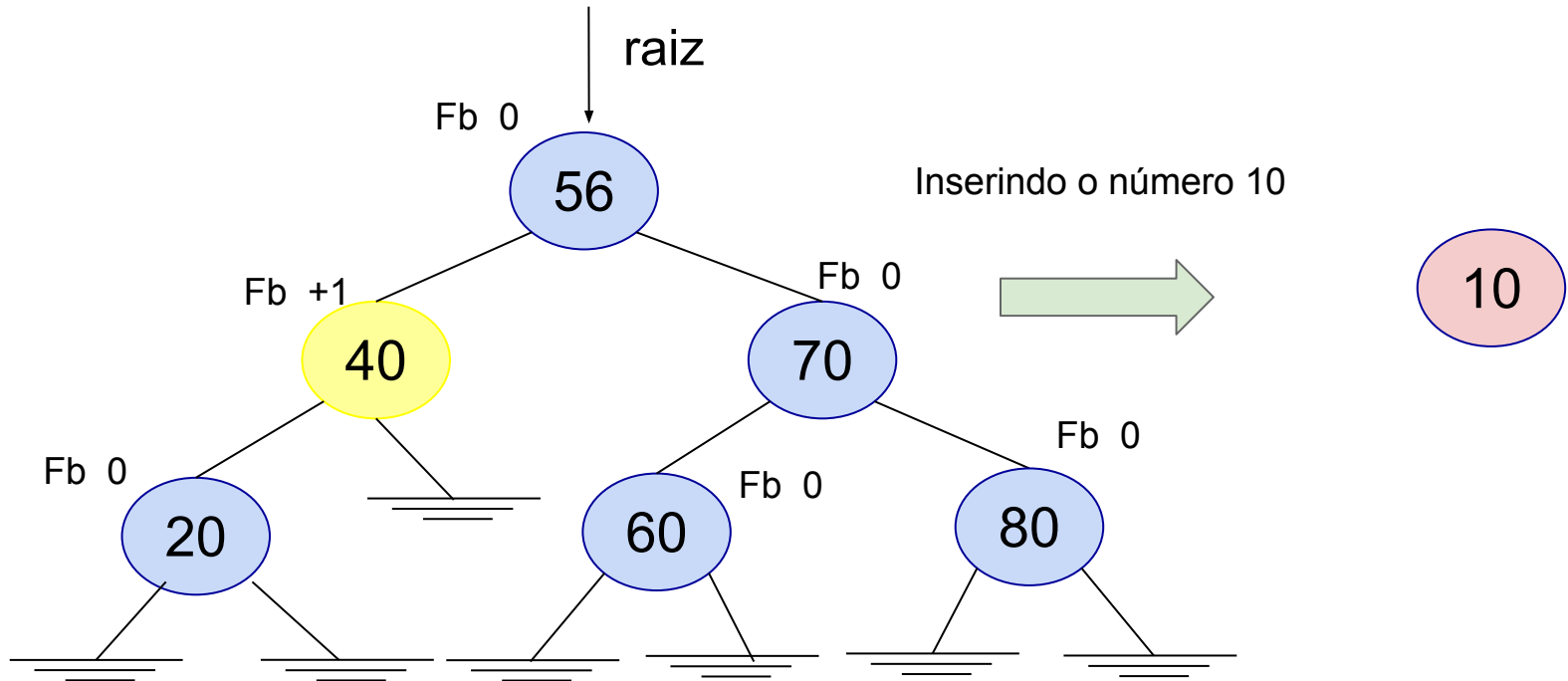
ROTAÇÃO SIMPLES À ESQUERDA - V



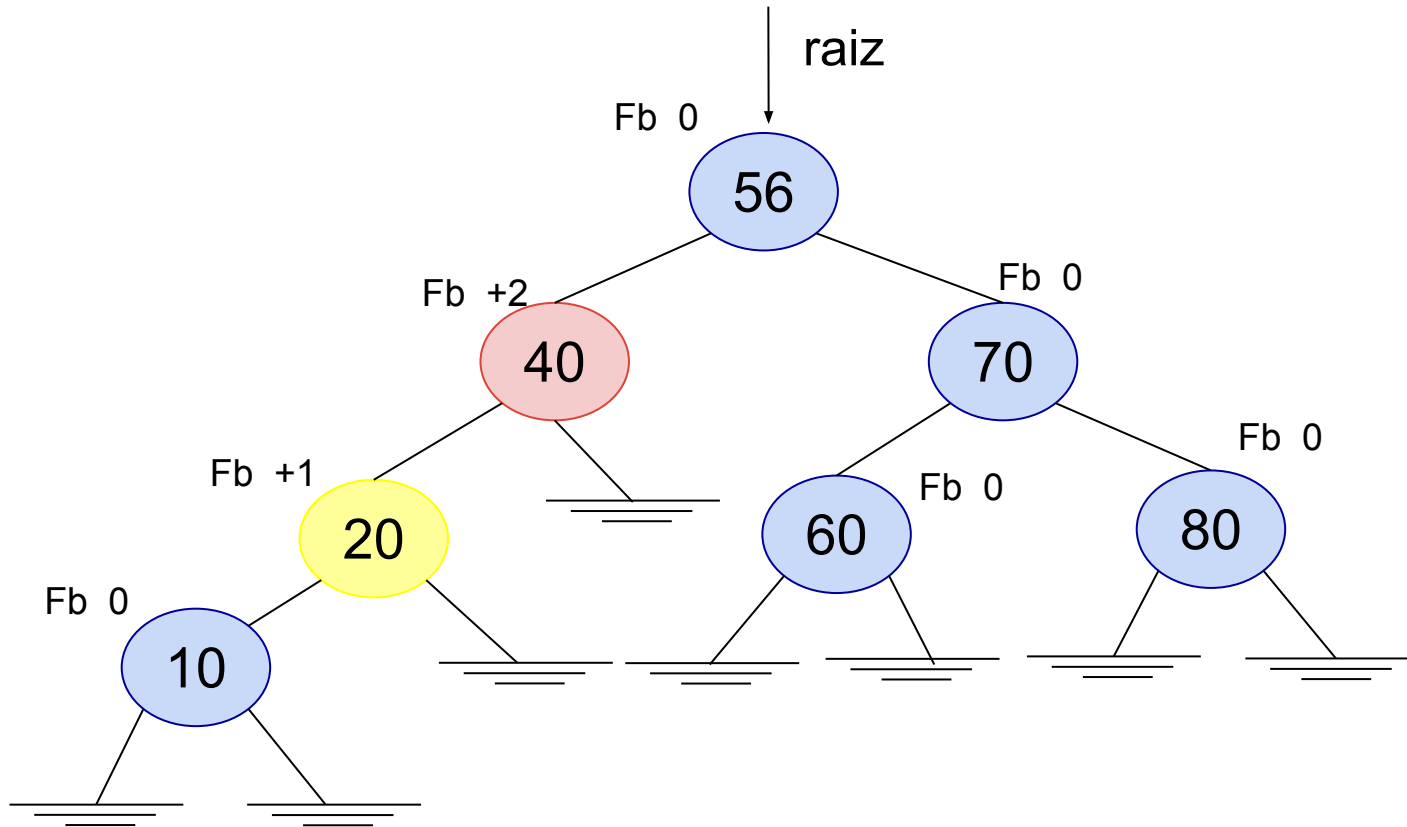
ROTAÇÃO SIMPLES À DIREITA - I



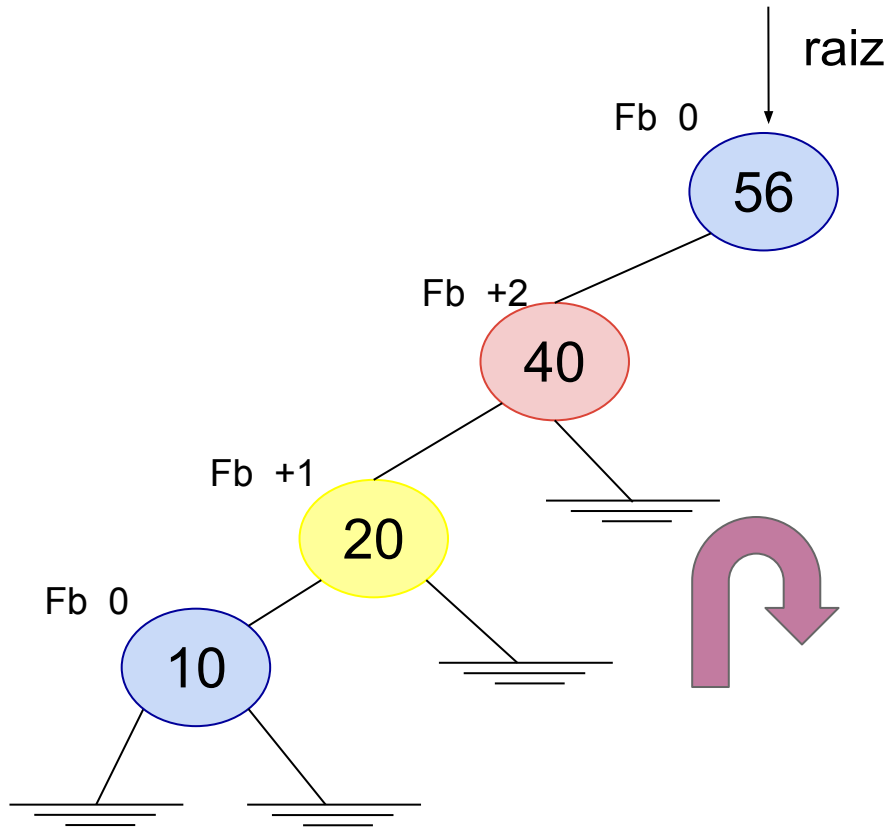
ROTAÇÃO SIMPLES À DIREITA - II



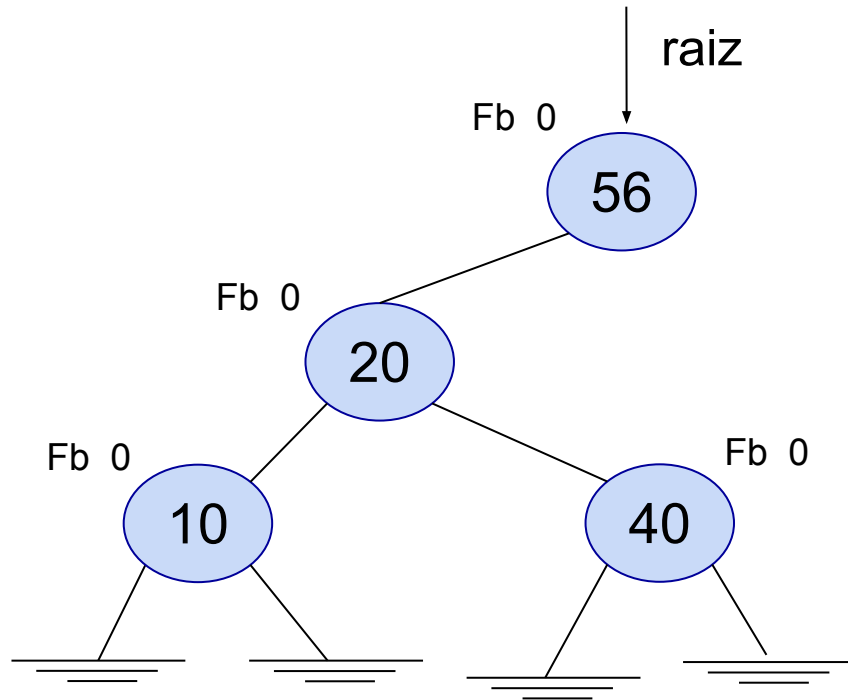
ROTAÇÃO SIMPLES À DIREITA - III



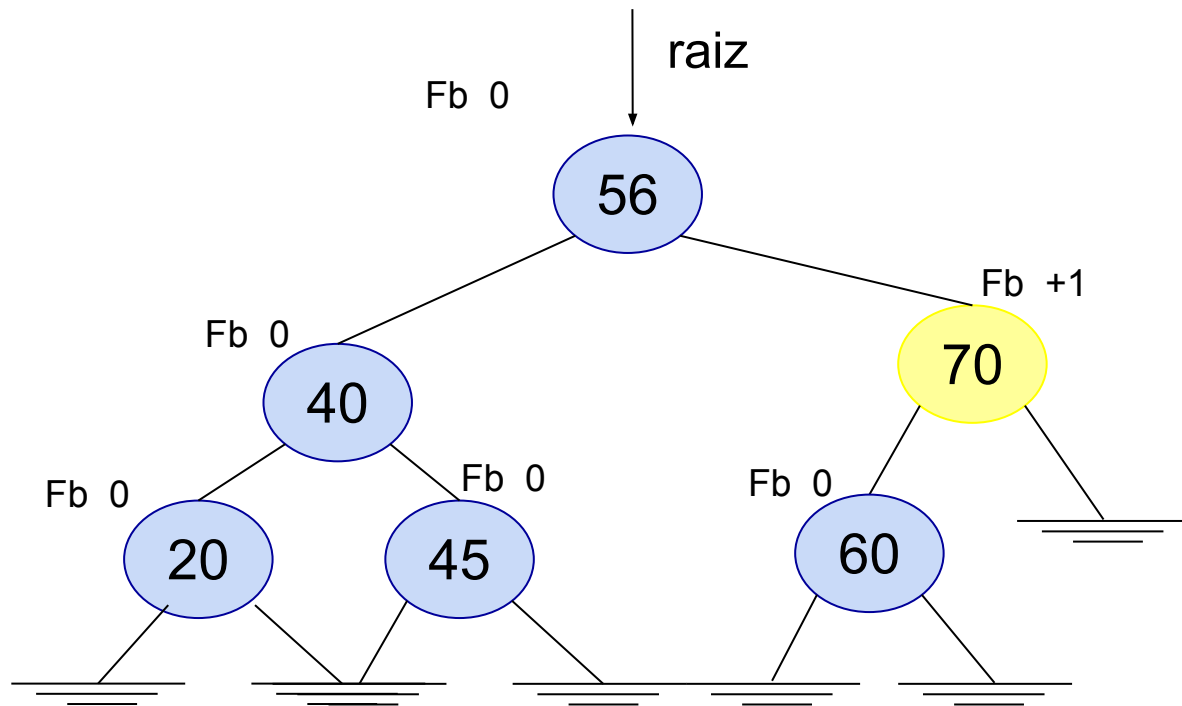
ROTAÇÃO SIMPLES À DIREITA - IV



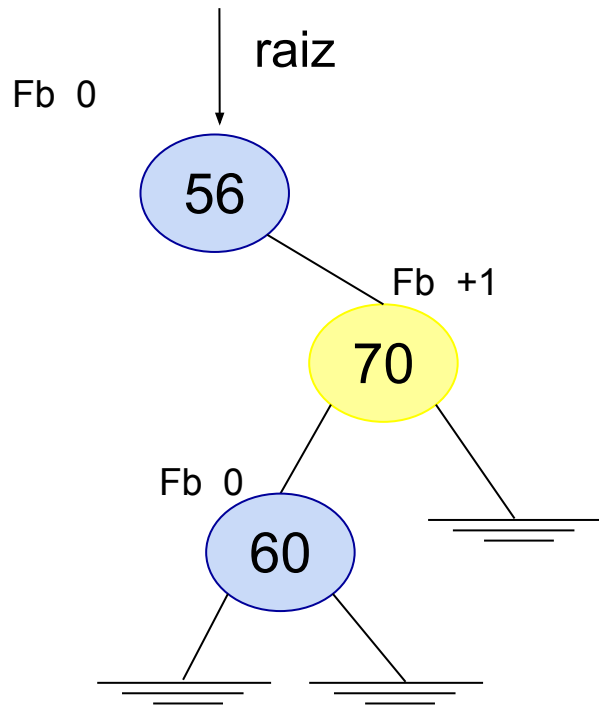
ROTAÇÃO SIMPLES À DIREITA - V



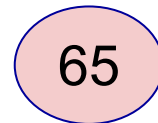
ROTAÇÃO ESQUERDA - DIREITA - I



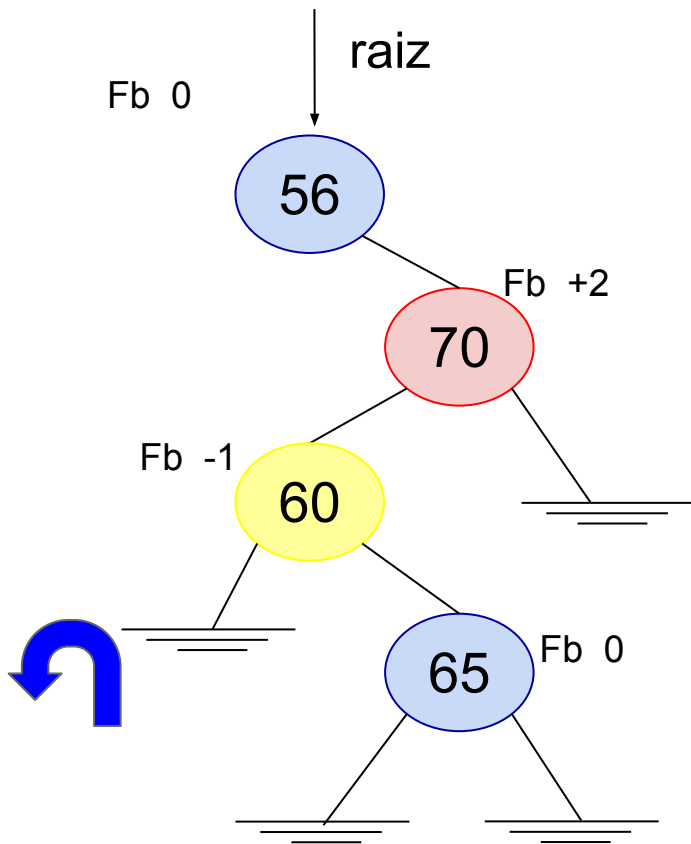
ROTAÇÃO ESQUERDA - DIREITA - II



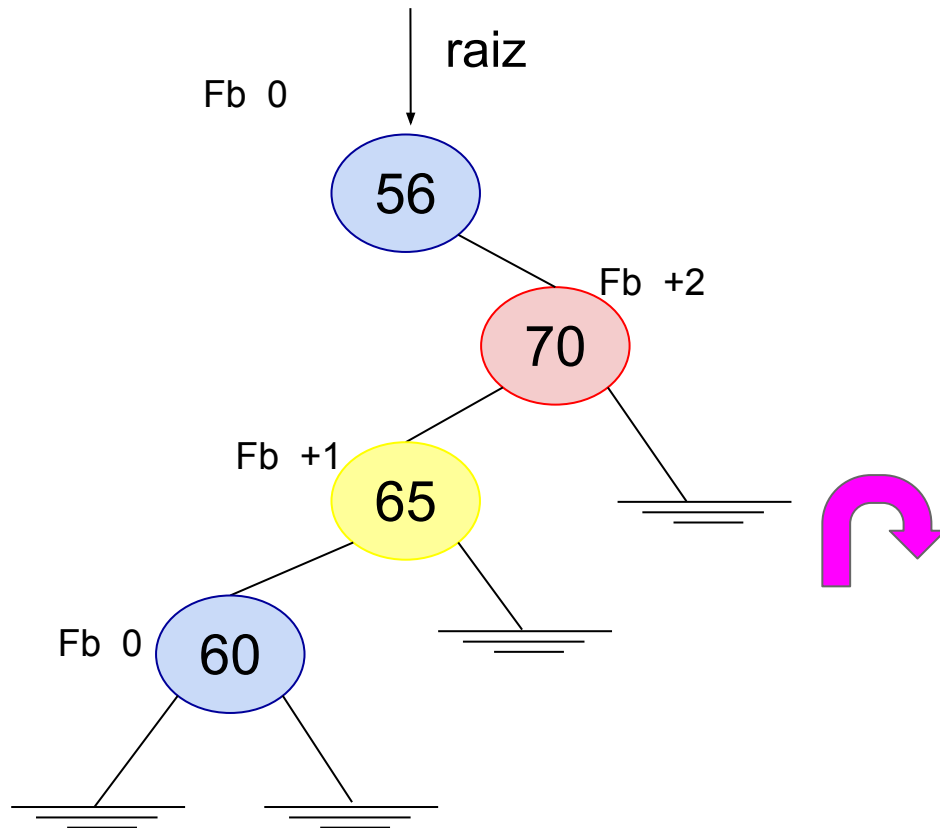
Inserindo o número 65



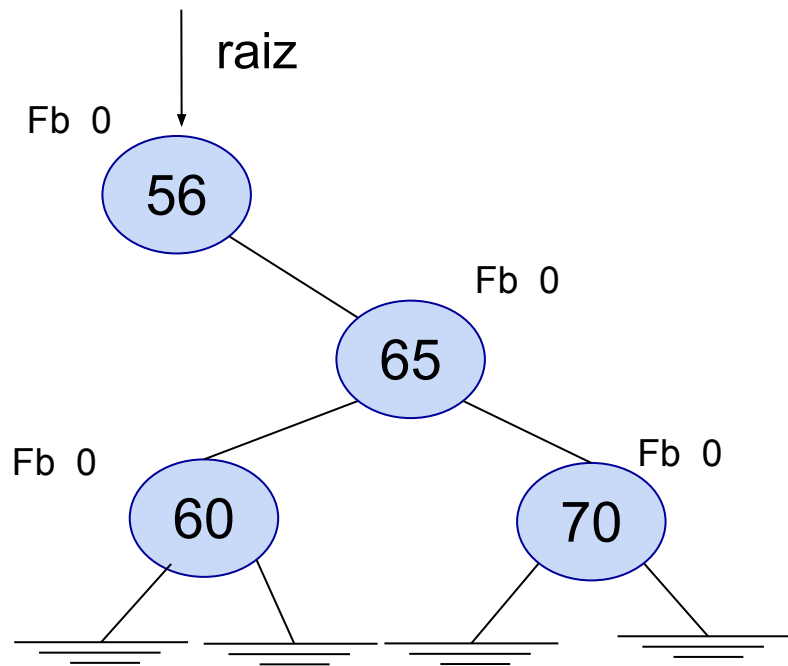
ROTAÇÃO ESQUERDA - DIREITA - III



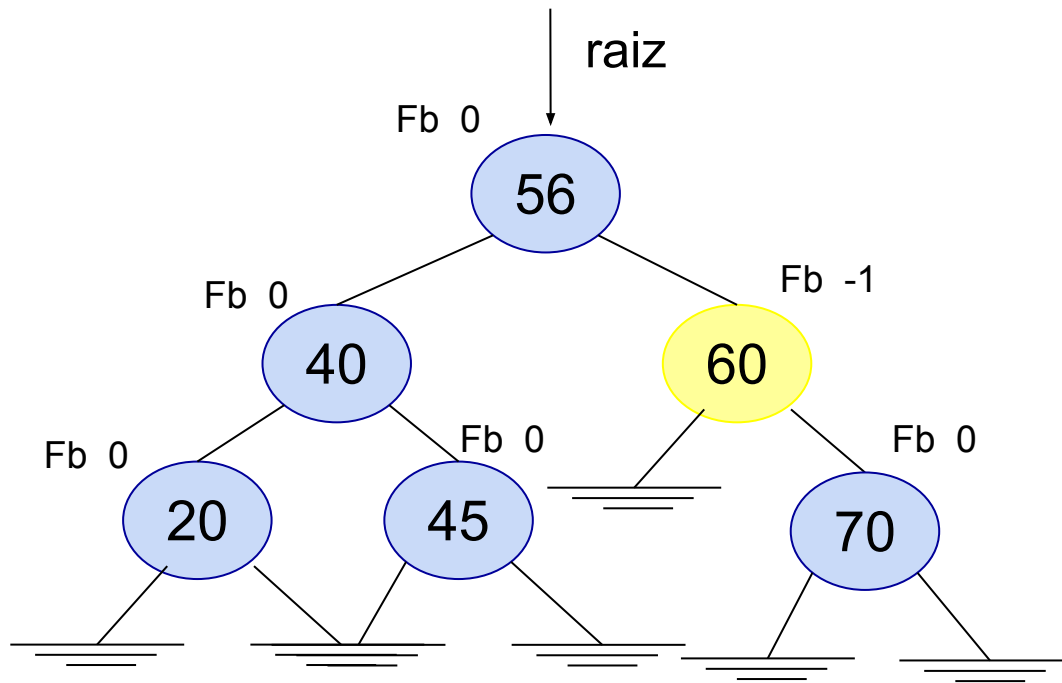
ROTAÇÃO ESQUERDA - DIREITA - IV



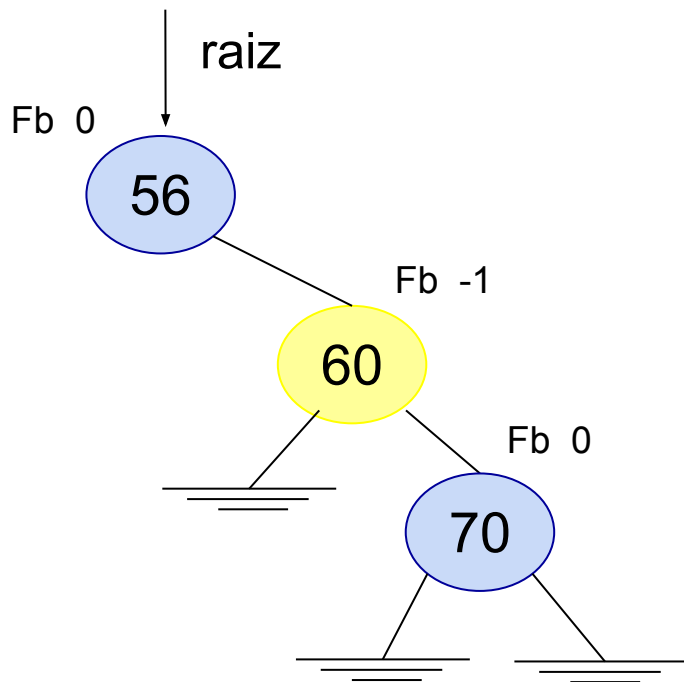
ROTAÇÃO ESQUERDA - DIREITA - V



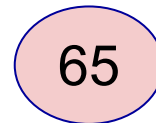
ROTAÇÃO DIREITA - ESQUERDA - I



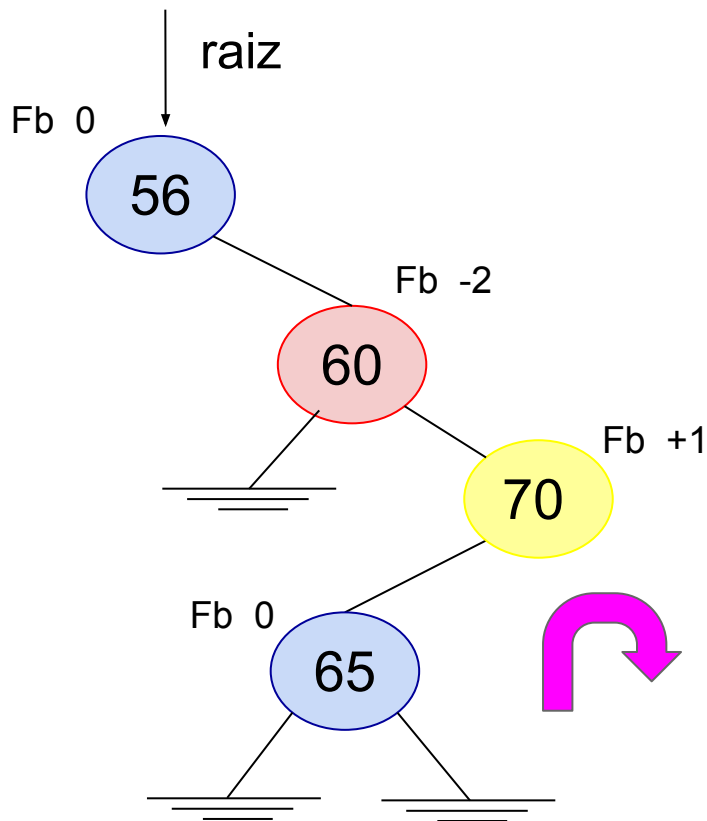
ROTAÇÃO DIREITA - ESQUERDA - II



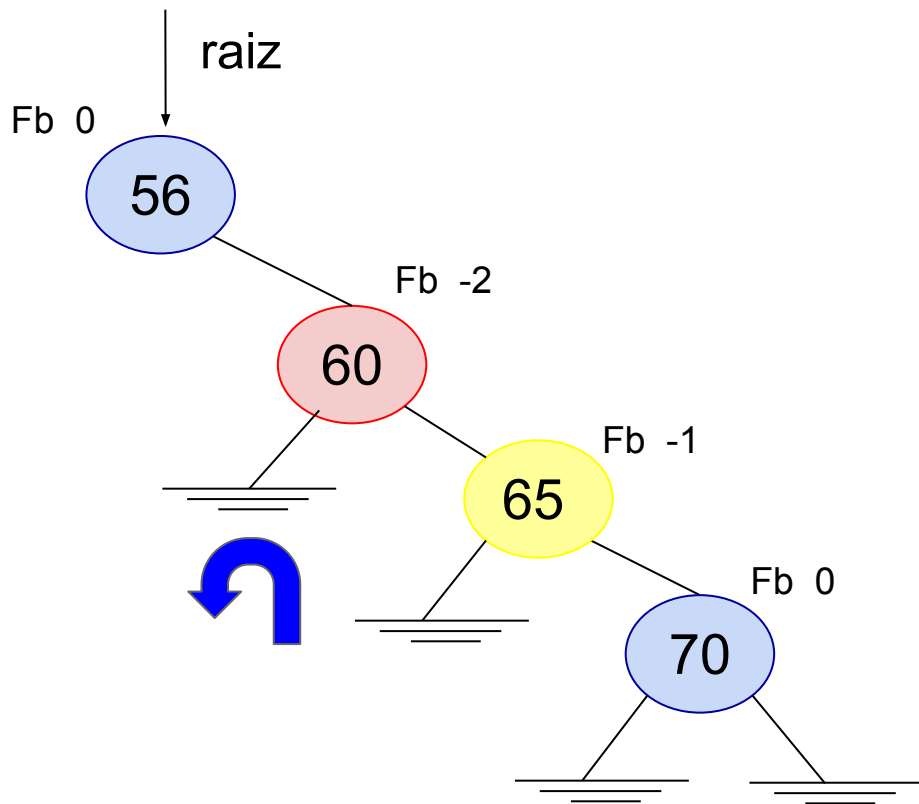
Inserindo o número 65



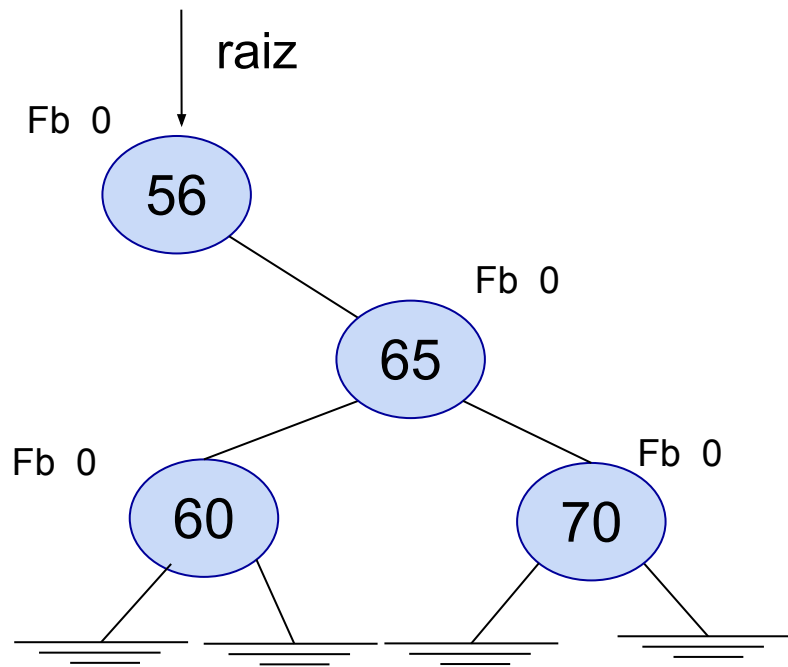
ROTAÇÃO DIREITA - ESQUERDA - III



ROTAÇÃO DIREITA - ESQUERDA - IV



ROTAÇÃO DIREITA - ESQUERDA - V



QUANDO BALANCEAR - 1/3

Quando o fator de balanceamento de um nó for menor que -1 ou maior que 1 , há necessidade de balancear a árvore.

Se o fator de balanceamento for positivo, as rotações são à direita; se for negativo as rotações são à esquerda.

QUANDO E COMO BALANCEAR - 2/3

Se o fator de balanceamento Fb de um nó for

$Fb = -2$ ou $Fb = 2$

e o fator de balanceamento de nó filho Fbf tiver o mesmo sinal, ou seja,

$Fb = -2$ e $Fbf = -1$ ou $Fbf = 0$

ou

$Fb = 2$ e $Fbf = 1$ ou $Fbf = 0$

a rotação a ser realizada é simples.

QUANDO E COMO BALANCEAR - 3/3

Se os sinais do fator de balanceamento do nó desbalanceado e de seu filho forem trocados, por exemplo,

$$Fb = -2 \text{ e } Fbf = 1$$

ou

$$Fb = 2 \text{ e } Fbf = -1$$

deve ser realizada uma rotação dupla.

IMPLEMENTAÇÃO DA AVL



OPERAÇÕES BÁSICAS EM AVLs

As árvores AVLs possuem métodos básicos das árvores binárias de busca (ABBs), como busca, percorrimto, etc.

As operações de inserção e remoção seguem o mesmo padrão das ABBs, com ajustes para manutenção do balanceamento, usando rotações.

OPERAÇÕES BÁSICAS EM AVLs

- Inserção
- Remoção
- Informar e atualizar altura
- Calcular o fator de balanceamento
- Rotação simples à direita
- Rotação simples à esquerda
- Rotação esquerda-direita
- Rotação direita-esquerda

OPERAÇÕES BÁSICAS EM ÁRVORES AVL

Informar altura é uma função que retorna a altura de um determinado nó.

Atualizar altura é uma função que ajusta o valor de um nó a medida da necessidade.

A função fator de balanceamento calcula o fator de balanceamento entre duas subárvores.

ATUALIZAÇÃO DE ALTURA - PSEUDOCÓDIGO - I

informarAltura(umNoh):

```
se (umNoh = NULO) {  
    retorna 0;  
} senão {  
    retornar umNoh.altura;  
}
```

ATUALIZAÇÃO DE ALTURA - PSEUDOCÓDIGO - II

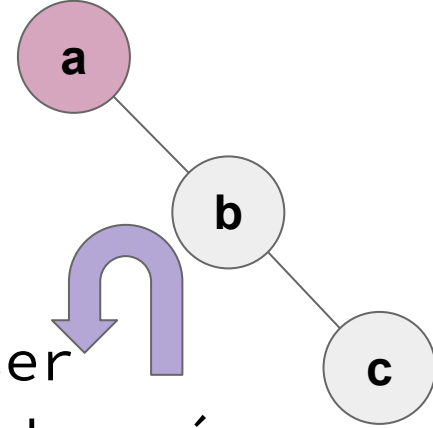
atualizaAltura(umNoh):

```
altArvEsq = informarAltura(umNoh.esquerdo);  
altArvDir = informarAltura(umNoh.direito);  
umNoh.altura = 1 + maximo(altArvEsq, altArvDir);
```

fatorBalanceamento(umNoh):

```
altArvEsq = informarAltura(umNoh.esquerdo);  
altArvDir = informarAltura(umNoh.direito);  
fatorBal = altArvEsq - altArvDir;  
retornar fatorBal;
```

CONSIDERAÇÕES SOBRE ROTAÇÕES SIMPLES - I

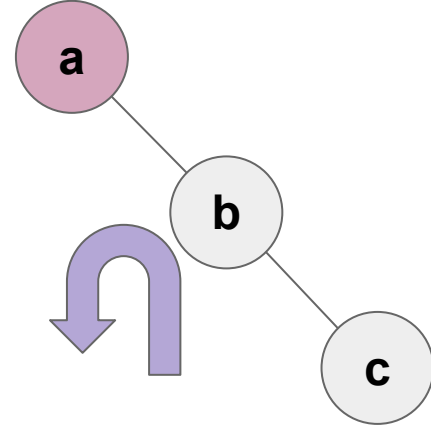


Os algoritmos de rotação simples podem ser implementados como métodos da árvore ou dos nós.

Podem ser implementados a partir do nó que se tornará a nova raiz da subárvore (nó **b**) ou a partir da raiz atual (nó **a**).

Os algoritmos a seguir implementam utilizando a raiz atual da subárvore (nó **a**) como referência.

CONSIDERAÇÕES SOBRE ROTAÇÕES SIMPLES - II



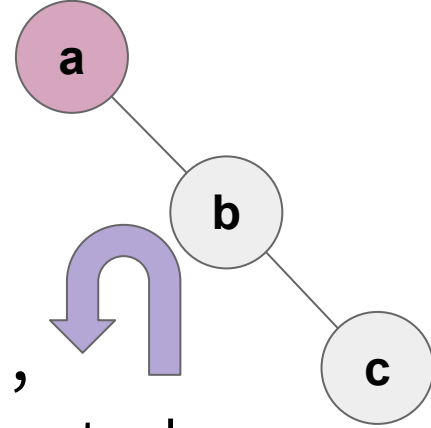
Os algoritmos de rotação podem ser implementados de maneira mais simples caso a árvore não possua duplo encadeamento (apontador para o nó pai).

Os slides a seguir utilizam esta abordagem, deixaremos apenas um exemplo de remoção com o duplo encadeamento para destacar o aumento na complexidade.

CONSIDERAÇÕES SOBRE ROTAÇÕES SIMPLES - III

Com a utilização do encadeamento simples, é necessário que as funções sejam implementadas de maneira recursiva, uma vez que o pai do nó rotacionado precisa alterar o filho, após a rotação.

Assim, o nó pai é quem chama o método, recebendo como retorno seu novo filho.



ROTAÇÃO À ESQUERDA - PSEUDOCÓDIGO - I

rotacaoEsquerda(umNoh):

// acha filho à direita da raiz da subárvore

nohAux ← umNoh.direito;

// armazena subárvore à esquerda do nó auxiliar

// à direita da raiz atual

umNoh.direito ← nohAux.esquerdo;

// posiciona umNoh como filho à esquerda de nohAux

nohAux.esquerdo ← umNoh;

ROTAÇÃO À ESQUERDA - PSEUDOCÓDIGO - II

```
// atualiza alturas
```

```
atualizaAltura(umNoh);
```

```
atualizaAltura(nohAux);
```

```
// atualiza a nova raiz da subárvore
```

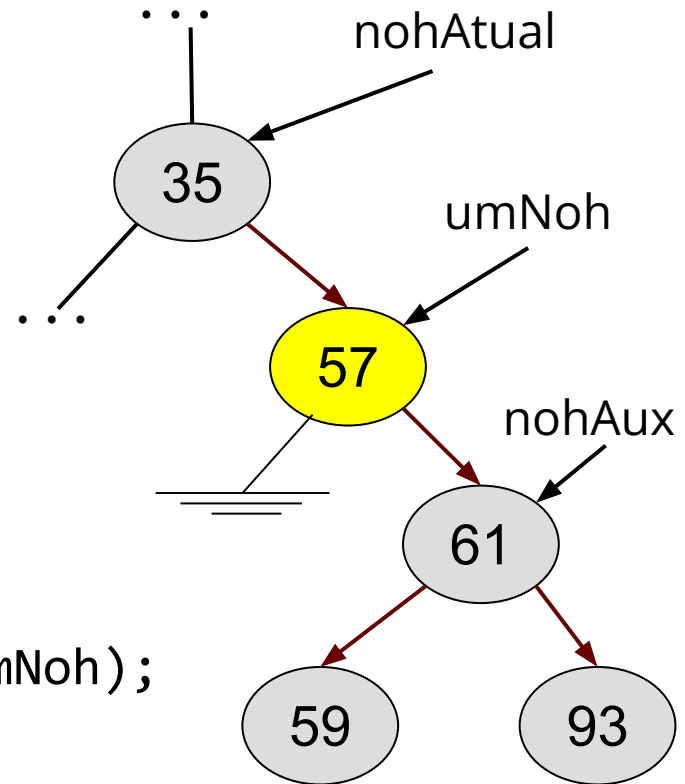
```
retornar nohAux;
```

ROTAÇÃO À ESQUERDA - EXEMPLO/PSEUDOCÓDIGO - I

Considere a o trecho de árvore ao lado, em que o nó 57 deverá ser rotacionado à esquerda.

O método será chamado no nó 35 (vamos aqui chamá-lo de `nohAtual`), da seguinte forma:

```
nohAtual.direito ← rotacaoEsquerda(umNoh);
```



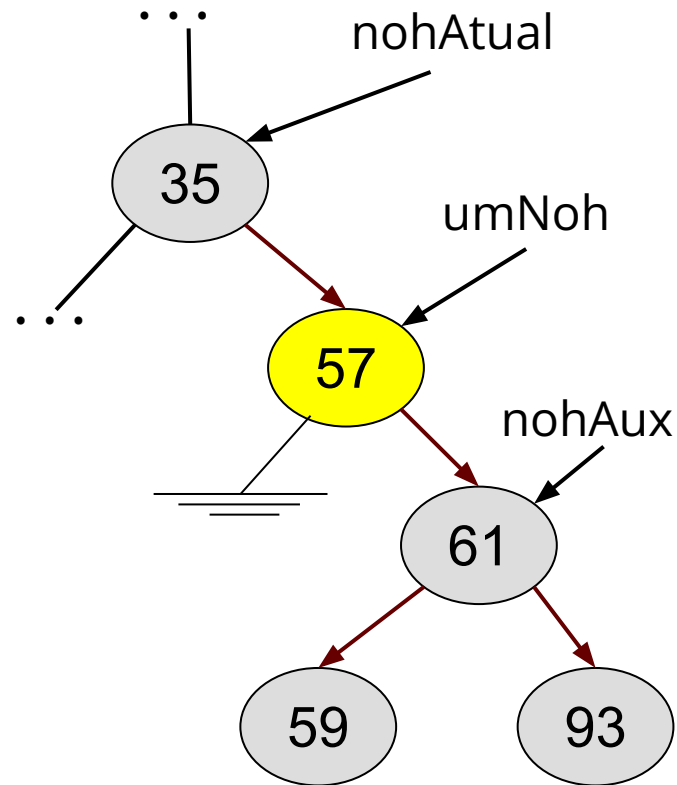
ROTAÇÃO À ESQUERDA - EXEMPLO/PSEUDOCÓDIGO - II

rotacaoEsquerda(umNoh):

nohAux \leftarrow *umNoh.direito*;

umNoh.direito \leftarrow *nohAux.esquerdo*;

nohAux.esquerdo \leftarrow *umNoh*;



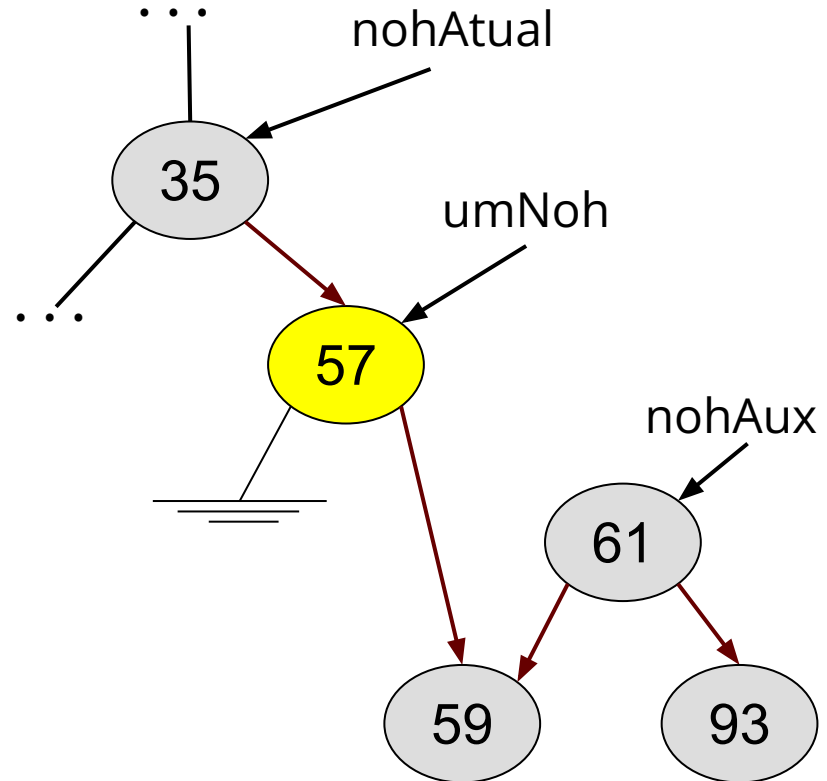
ROTAÇÃO À ESQUERDA - EXEMPLO/PSEUDOCÓDIGO - III

rotacaoEsquerda(umNoh):

nohAux ← umNoh.direito;

umNoh.direito ← nohAux.esquerdo;

nohAux.esquerdo ← umNoh;

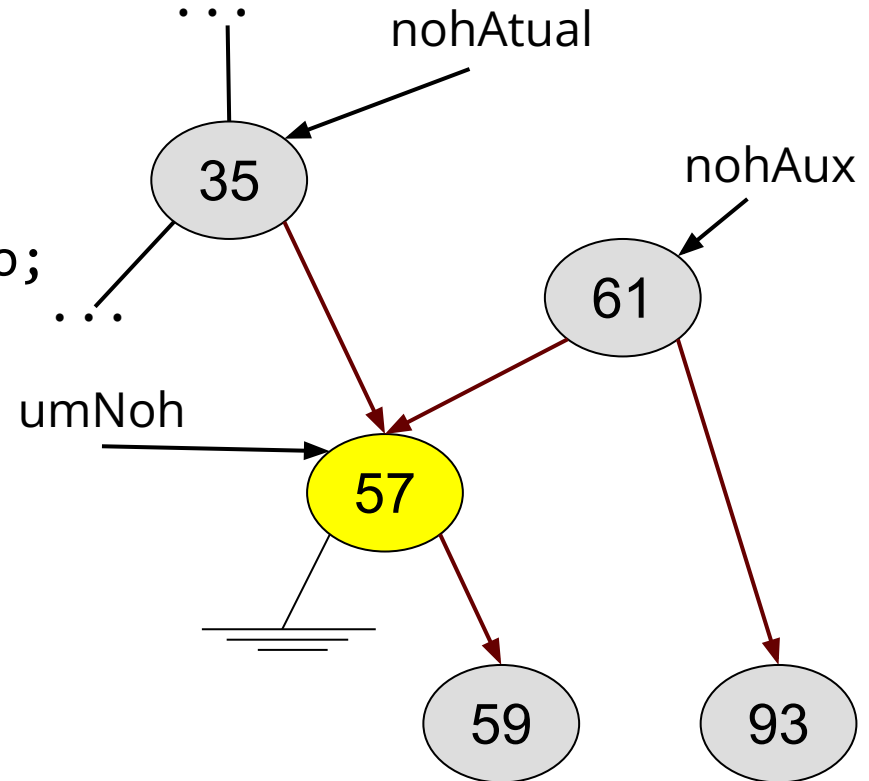


ROTAÇÃO À ESQUERDA - EXEMPLO/PSEUDOCÓDIGO - IV

rotacaoEsquerda(umNoh):

nohAux \leftarrow umNoh.direito;
umNoh.direito \leftarrow nohAux.esquerdo;

nohAux.esquerdo \leftarrow umNoh;



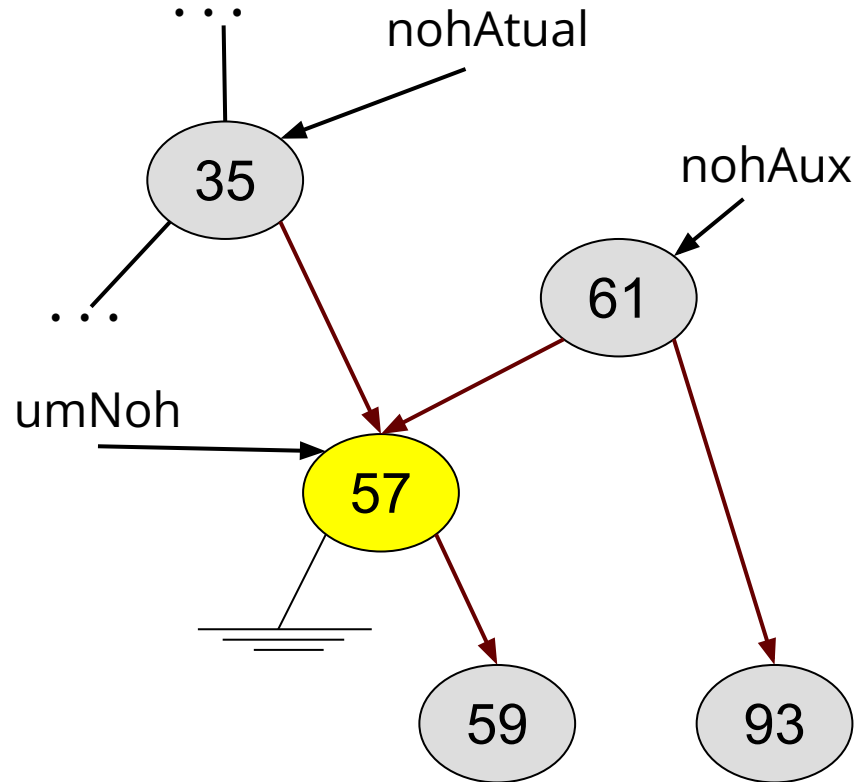
ROTAÇÃO À ESQUERDA - EXEMPLO/PSEUDOCÓDIGO - V

rotacaoEsquerda(umNoh):

atualizaAltura(umNoh);
atualizaAltura(nohAux);

retornar nohAux;

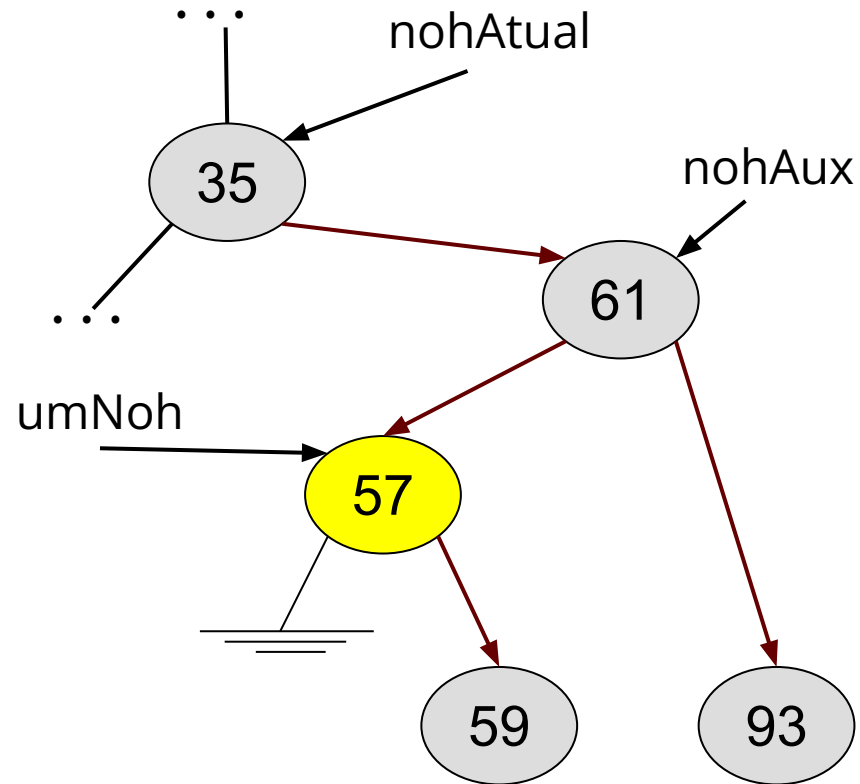
Não indicaremos aqui o processo de atualização da altura, para não sobrecarregar o desenho.



ROTAÇÃO À ESQUERDA - EXEMPLO/PSEUDOCÓDIGO - VI

nohAtual.direito

← rotacaoEsquerda(umNoh);



ROTAÇÃO À ESQUERDA - DUPLO ENCADEAMENTO - I

rotacaoEsquerda(umNoh):

// acha filho à direita da raiz da subárvore

nohAux ← umNoh.direito;

// armazena subárvore à esquerda do nó auxiliar

// à direita da raiz atual

umNoh.direito ← nohAux.esquerdo;

// atualiza o pai do nó à esquerda da raiz da subárvore

se (nohAux.esquerdo ≠ NULO) {

 nohAux.esquerdo.pai ← umNoh;

}

ROTAÇÃO À ESQUERDA - DUPLO ENCADEAMENTO - I

rotacaoEsquerda(umNoh):

// acha filho à direita da raiz

nohAux ← umNoh.direito;

// armazena subárvore à esquerda

// à direita da raiz atual

umNoh.direito ← nohAux.esquerdo;

// atualiza o pai do nó à esquerda da raiz da subárvore

se (nohAux.esquerdo ≠ NULO) {

 nohAux.esquerdo.pai ← umNoh;

}

Trechos em amarelo, neste e nos próximos slides, para destacar mudanças para o duplo encadeamento.

ROTAÇÃO À ESQUERDA - DUPLO ENCADEAMENTO - II

```
// Atualiza o pai de nohAux
```

```
nohAux.pai ← umNoh.pai;
```

```
// se procedimento:
```

```
// se raizAtual era raiz da árvore, muda raiz para nohAux
```

```
se (umNoh = raiz) raiz ← nohAux;
```

```
// caso contrário, coloque nohAux no lugar de umNoh
```

```
senão se (umNoh = umNoh.pai.esquerdo) {
```

```
    umNoh.pai.esquerdo ← nohAux;
```

```
} senão {
```

```
    umNoh.pai.direito ← nohAux;
```

```
}
```

ROTAÇÃO À ESQUERDA - DUPLO ENCADEAMENTO - III

```
// faz umNoh como filho à esquerda de nohAux
```

```
nohAux.esquerdo ← umNoh;
```

```
umNoh.pai ← nohAux;
```

```
// atualiza alturas
```

```
atualizaAltura(umNoh);
```

```
atualizaAltura(nohAux);
```

```
// se função: retornar a nova raiz da subárvore
```

```
retornar nohAux;
```

ROTAÇÃO À ESQUERDA - DUPLO ENCADEAMENTO - II × III

// se raizAtual era raiz da árvore, muda raiz para
nohAux

```
se (umNoh = raiz) {
```

```
    raiz ← nohAux;
```

```
// caso contrário, coloque
```

```
} senão se (umNoh = umNoh.
```

```
...
```

```
...
```

```
// opcional: retornar a raiz
```

```
retornar nohAux;
```

Caso o método seja implementado como procedimento, o ajuste manual da raiz (trecho em amarelo) é necessário.

Implementações com métodos com retorno (trecho em verde) fazem o ajuste da raiz nas chamadas recursivas, por meio de atribuição direta na chamada.

ROTAÇÃO À DIREITA - PSEUDOCÓDIGO - I

rotacaoDireita(umNoh):

// acha filho à esquerda da raiz da subárvore

nohAux ← umNoh.esquerdo;

// armazena subárvore à direita de nohAux

// à esquerda da raiz atual

umNoh.esquerdo ← nohAux.direito;

// posiciona umNoh como filho à direita de nohAux

nohAux.direito ← umNoh;

ROTAÇÃO À DIREITA - PSEUDOCÓDIGO - II

// atualiza alturas

atualizaAltura(umNoh);

atualizaAltura(nohAux);

// se implementado como função:

// retornar a nova raiz da subárvore

retornar nohAux;

ROTAÇÕES DUPLAS - PSEUDOCÓDIGO

rotacaoEsquerdaDireita(umNoh):

```
umNoh.esquerdo ← rotacaoEsquerda(umNoh.esquerdo);  
retornar rotacaoDireita(umNoh);
```

rotacaoDireitaEsquerda(umNoh):

```
umNoh.direito ← rotacaoDireita(umNoh.direito);  
retornar rotacaoEsquerda(umNoh);
```

BALANCEAMENTO - 1/2

Processo de verificação da necessidade de balanceamento é acionado sempre que uma alteração (inserção/remoção) ocorre na árvore. A verificação é feita no ramo da árvore em que ocorreu a alteração, a partir do nó de altura mais baixa até a raiz, processo que ocorre em todo esse caminho.

Geralmente esse processo é feito por meio de um método próprio para essa verificação, denominado `arrumarBalanceamento()` ou similar.

BALANCEAMENTO - 2/2

O método é invocado pelos métodos tradicionais de inserção e remoção, que, em geral, só possuem essa chamada de diferença significativa para uma árvore binária tradicional implementada por métodos recursivos.

AVL - INSERÇÃO - PSEUDOCÓDIGO - I

inserirRecursivamente(umValor):

raiz \leftarrow inserirRecAux(raiz, umValor);

inserirRecAux(umNoh, umValor):

se (umNoh = NULO) {

 novo \leftarrow criar_noh(umValor); // cria um nó com o valor

 retornar novo;

}senão { ...

AVL - INSERÇÃO - PSEUDOCÓDIGO - II

```
}senão {  
  // não é folha nula, checa inserção à esquerda ou direita  
  se (umValor < umNoh.valor) {  
    umNoh.esquerdo ← inserirRecAux(umNoh.esquerdo, umValor);  
  } senão {  
    umNoh.direito ← inserirRecAux(umNoh.direito, umValor);  
  }  
}  
retornar arrumarBalanceamento(umNoh);
```

AVL - INSERÇÃO - PSEUDOCÓDIGO - II

```
}senão {  
  // não é folha nula, checa inserção à esquerda ou direita  
  se (umValor < umNoh.valor) {  
    umNoh.esquerdo ← inserirRecAux(umNoh.esquerdo, umValor);  
  } senão {  
    umNoh.direito ← inserirRecAu  
  }  
}  
retornar arrumarBalanceamento(umNoh);
```

Alteração de código da árvore binária para suporte ao balanceamento AVL

AVL - ARRUMARBALANCEAMENTO() - PSEUDOCÓDIGO - I

arrumarBalanceamento(umNoh):

```
se (umNoh = NULO) { retornar umNoh; }  
// Inicialmente atualiza a altura de umNoh  
atualizaAltura(umNoh);  
// Checa o balanceamento no nó  
fatorBal ← fatorBalanceamento(umNoh);  
// retorna o nó acima na árvore, caso esteja balanceado  
se ((fatorBal ≥ -1) e (fatorBal ≤ 1)) {  
    retornar umNoh;  
}
```

AVL - ARRUMARBALANCEAMENTO() - PSEUDOCÓDIGO - II

```
// Caso o nó esteja desbalanceado, há 4 situações
// 1. Desbalanceamento Esquerda Esquerda
se ( (fatorBal > 1)
    e (fatorBalanceamento(umNoh.esquerdo) ≥ 0) ) {
    retornar rotacaoDireita(umNoh);
}
```


AVL - ARRUMARBALANCEAMENTO() - PSEUDOCÓDIGO - II

// 2. Desbalanceamento Esquerda Direita

```
se ( (fatorBal > 1)
    e (fatorBalanceamento(umNoh.esquerdo) < 0) ) {
    umNoh.esquerdo = rotacaoEsquerda(umNoh.esquerdo);
    retornar rotacaoDireita(umNoh);
}
```

AVL - ARRUMARBALANCEAMENTO() - PSEUDOCÓDIGO - III

// Caso o nó esteja desbalanceado, há 4 situações

// 3. Desbalanceamento Direita Direita

se ((fatorBal < -1)

 e (fatorBalanceamento(umNoh.direito) \leq 0)) {

 retornar rotacaoEsquerda(umNoh);

}

AVL - ARRUMARBALANCEAMENTO() - PSEUDOCÓDIGO - III

// 4. Desbalanceamento Direita Esquerda

se ((fatorBal < -1)

 e (fatorBalanceamento(umNoh.direito) > 0)) {

 umNoh.direito = rotacaoDireita(umNoh.direito);

 retornar rotacaoEsquerda(umNoh);

}

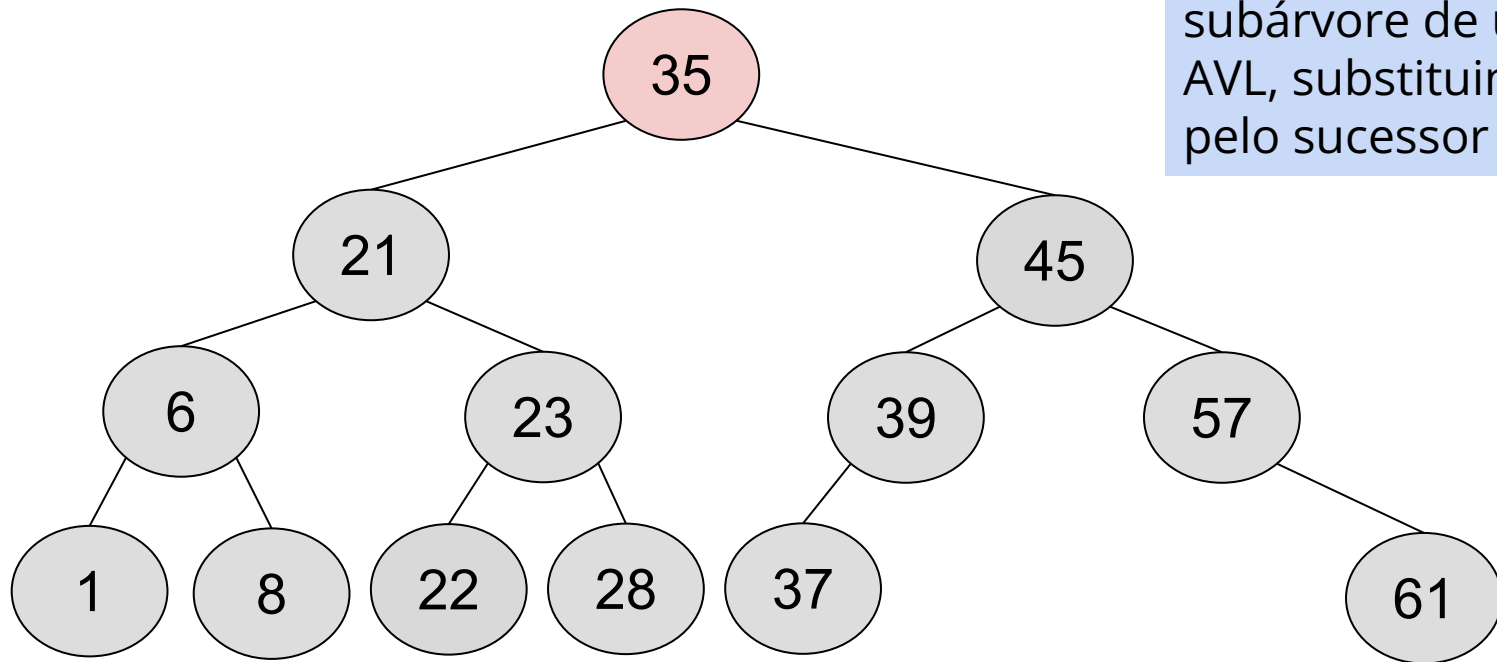
AVL - REMOÇÃO

A remoção na AVL é feita de maneira mais simples e eficiente usando a remoção totalmente recursiva.

Isso ocorre porque o uso da transplanta obrigaria a ter que ajustar o balanceamento no caminho, tornando necessário, inclusive, o duplo encadeamento.

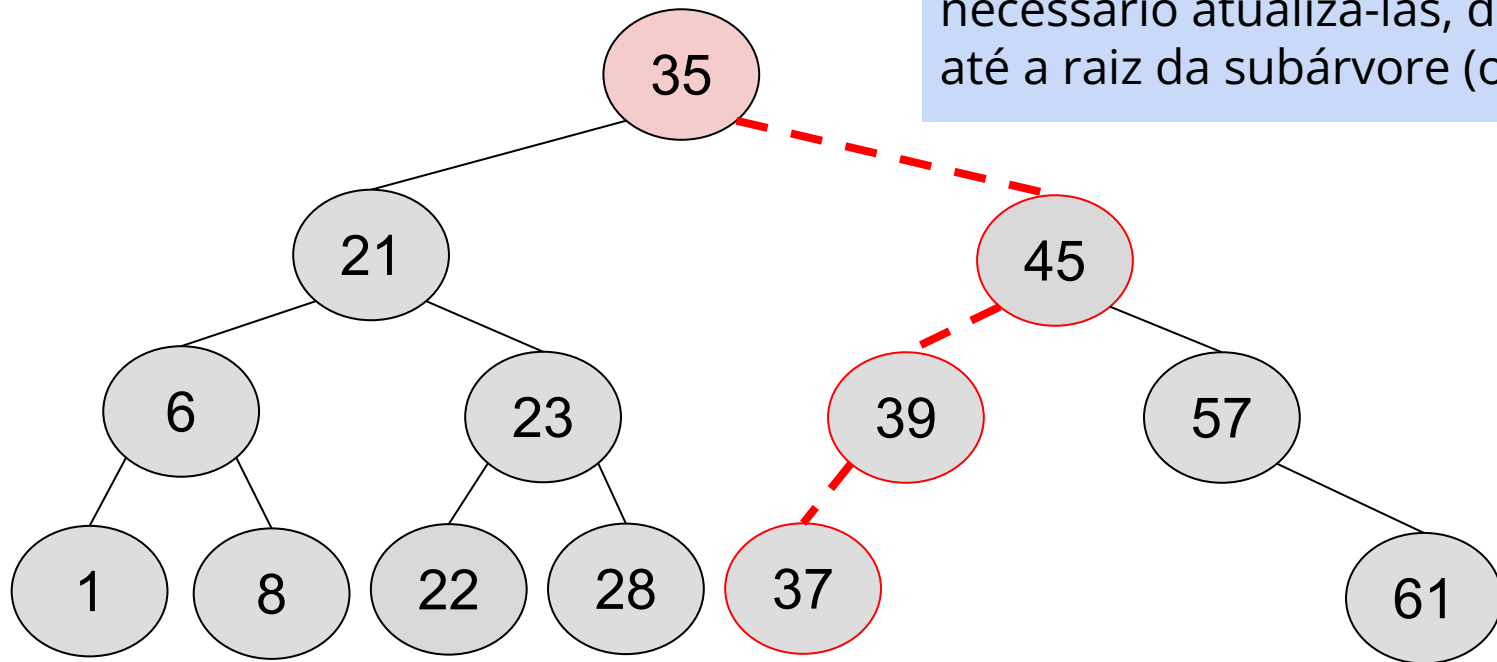
AVL - REMOÇÃO -1

Suponha remoção do 35 em uma subárvore de uma AVL, substituindo pelo sucessor



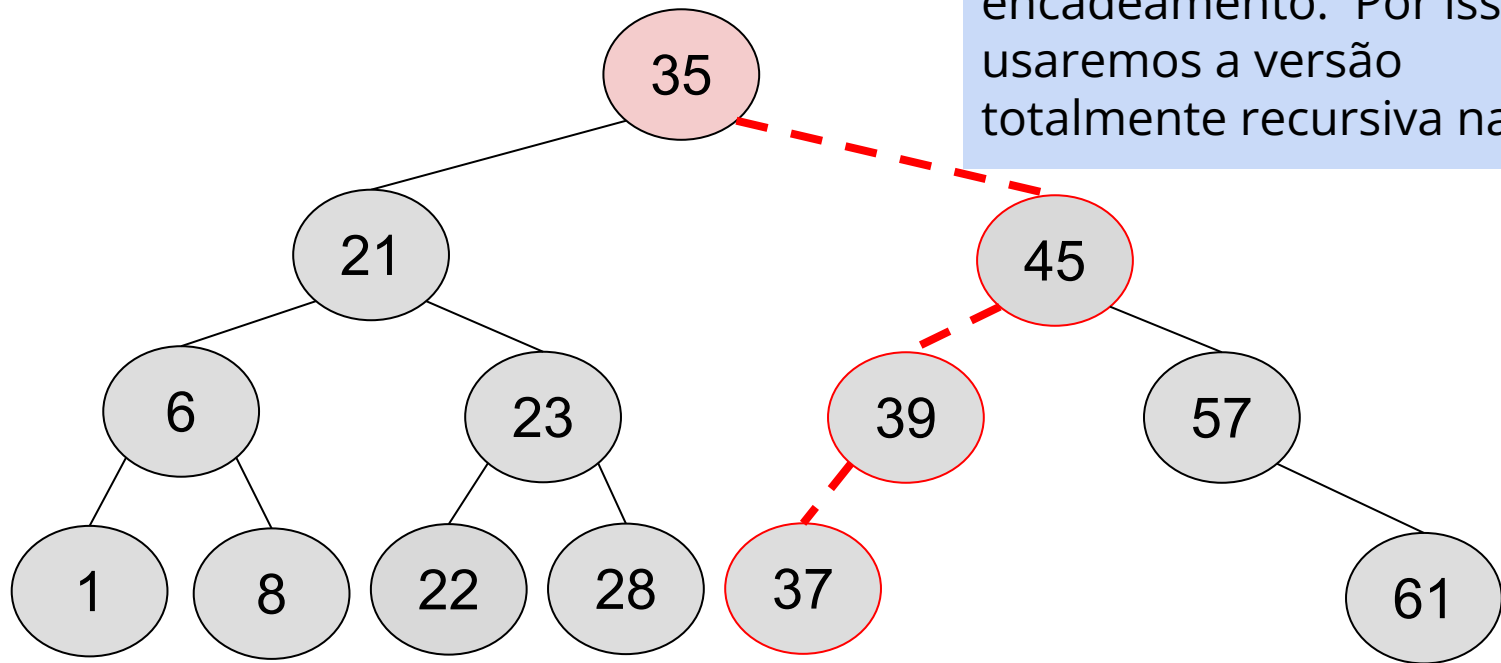
AVL - REMOÇÃO - 2

Todos no caminho até o sucessor (37) podem ter sua altura modificada, sendo necessário atualizá-las, da folha até a raiz da subárvore (o 35).



AVL - REMOÇÃO - 3

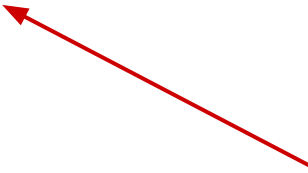
Sem usar recursão, é necessário voltar na árvore, o que exigiria o duplo encadeamento. Por isso, usaremos a versão totalmente recursiva na AVL.



AVL - REMOVEMENOR - PSEUDOCÓDIGO

removeMenor(raizSub):

```
// procedimento auxiliar para remover o sucessor substituindo-o
// pelo seu filho à direita
se (raizSub.esquerdo = NULO) { // encontrou o sucessor
    retorna raizSub.direito;
} senão { // não achou ainda, desce mais na subárvore
    raizSub.esquerdo ← removeMenor(raizSub->esquerdo);
    retorna arrumaBalanceamento(raizSub);
}
```



*Alteração de código da árvore
binária para suporte ao
balanceamento AVL*

AVL - REMOÇÃO 2 - PSEUDOCÓDIGO - I

removerRecursivamente(umValor):

raiz \leftarrow removerRecAux(raiz, umValor);

removerRecAux(umNoh, umValor):

```
se (umNoh = NULO) {  
    geraErro("Nó não encontrado!");  
}
```

AVL - REMOÇÃO - PSEUDOCÓDIGO - II

```
novaRaizSubArvore ← umNoh;  
// valor é menor que nó atual, vai para subárvore esquerda  
se ( umValor < umNoh.valor ) {  
    umNoh.esq ← removerRecAux(umNoh.esquerdo, umValor);  
// valor é maior que nó atual, vai para subárvore direita  
} senão se ( umValor > umNoh->valor ) {  
    umNoh.dir ← removerRecAux(umNoh.direito, umValor);  
// valor igual ao do nó atual, que deve ser apagado  
} senão {
```

AVL - REMOÇÃO - PSEUDOCÓDIGO - III

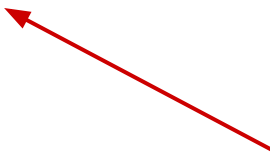
```
} senão {  
  // nó não tem filhos à esquerda  
  se (umNoh.esquerdo = NULO) {  
    novaRaizSubArvore ← umNoh.direito;  
  // nó não tem filhos à direita  
} senão se (umNoh.direito = NULO) {  
  novaRaizSubArvore ← umNoh.esquerdo;  
} senão { // nó tem dois filhos
```

AVL - REMOÇÃO - PSEUDOCÓDIGO - IV

```
} senão { // nó tem dois filhos
    // trocando pelo sucessor
    novaRaizSubArvore ← minimoAux(umNoh.direito);
    // onde antes estava o sucessor fica agora seu filho
    // à direita
    novaRaizSubArvore.direita ←
removeMenor(umNoh.direito);
    // filho à esquerda de umNoh torna-se filho à esquerda
    // de sucessor
    novaRaizSubArvore.esquerda ← umNoh.esquerdo;
}
```

AVL - REMOÇÃO - PSEUDOCÓDIGO - VI

```
// ponteiros ajustados, apagamos o nó  
apagar(umNoh);  
}  
// retorna o balanceamento na nova raiz da subárvore  
retornar arrumaBalanceamento(novaRaizSubArvore);
```



*Alteração de código da árvore
binária para suporte ao
balanceamento AVL*

SOBRE O MATERIAL



SOBRE ESTE MATERIAL

Material produzido coletivamente, principalmente pelos seguintes professores do DCC/UFLA:

- Joaquim Quinteiro Uchôa
- Juliana Galvani Greggi
- Renato Ramos da Silva

Inclui contribuições de outros professores do setor de Fundamentos de Programação do DCC/UFLA.

Esta obra está licenciado com uma Licença [Creative Commons Atribuição 4.0 Internacional](https://creativecommons.org/licenses/by/4.0/).