Árvores balanceadas

Prof. Paulo Henrique Pisani

março/2022

Tópicos

- Introdução, Árvore AVL, Fator de balanceamento;
- Rotações;
- Balanceamento;
- Operações: Inserção e Remoção
- Complexidade das operações.

Introdução

Introdução

- O custo das operações em árvores binárias de busca (ABB) é:

Operação	Árvores
Busca	O(h)
Inserção	O(h)
Remoção	O(h)

- Como h pode chegar a ser n-1, o custo no pior caso é O(n).

Introdução

- Com árvores balanceadas, podemos melhorar a eficiência da árvore; Existem vários tipos de árvores balanceadas: AVL, Rubro-negra, 2-3, B, etc.
- Nesta aula, veremos a árvore **AVL**, que permite $h = O(lg(n))$;
- Portanto, em uma árvore AVL, temos que:

Operação	Árvores
Busca	$O(lg(n))$
Inserção	$O(lg(n))$
Remoção	$O(lg(n))$

Árvore AVL

- São árvores binárias balanceadas pela altura:
  - Para todos os nós, o módulo da diferença de altura entre as subárvores esquerda e direita de cada nó é no máximo 1.
- O nome AVL vem dos autores Georgy **A**delson-**V**elsky e Evgenii **L**andis.

Fator de balanceamento

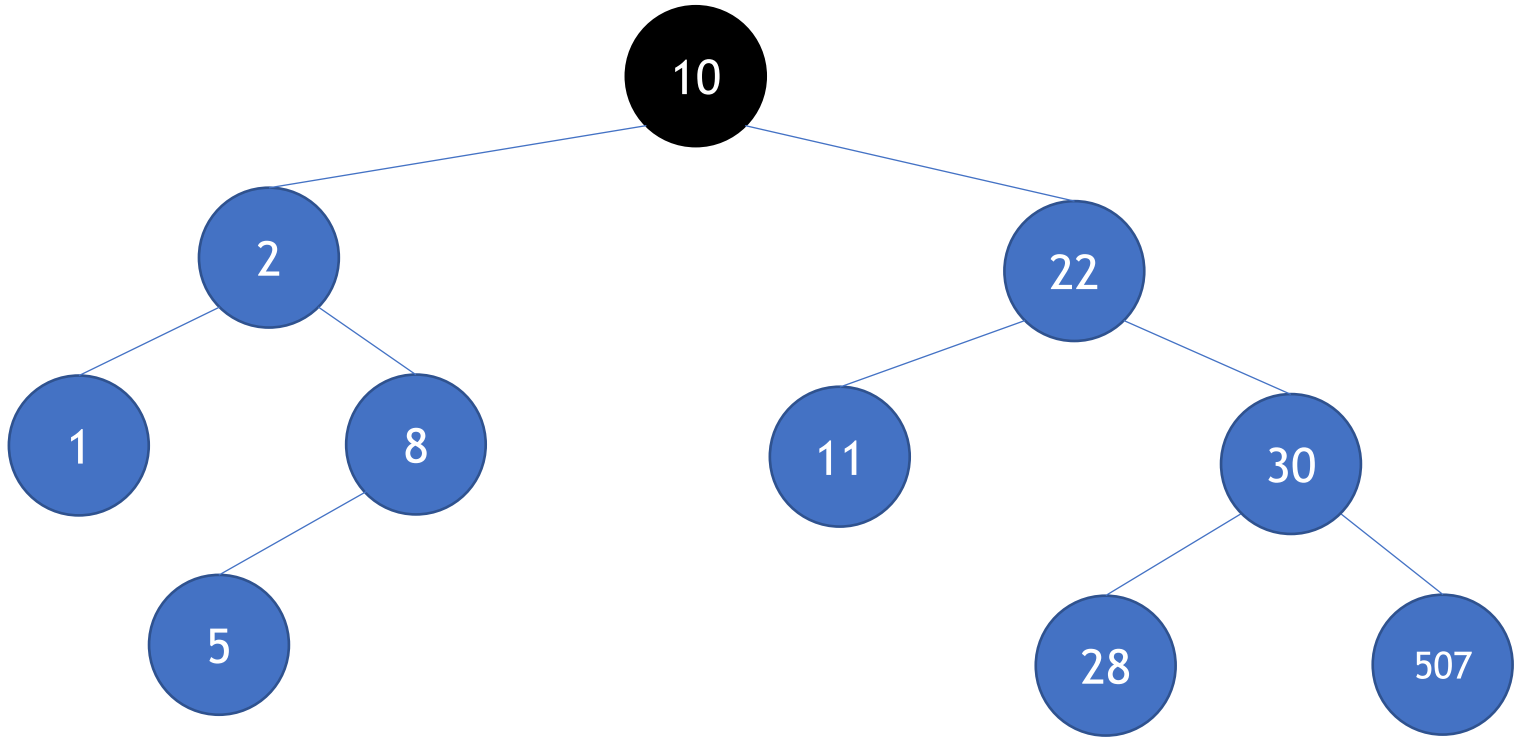
- Para um nó *v*, é a diferença de altura(h) entre as subárvores esquerda e direita:

$$fb(v) = h(v.esq) - h(v.dir)$$

- Em uma AVL, fb deve ser -1, 0 ou 1 para todos os nós.

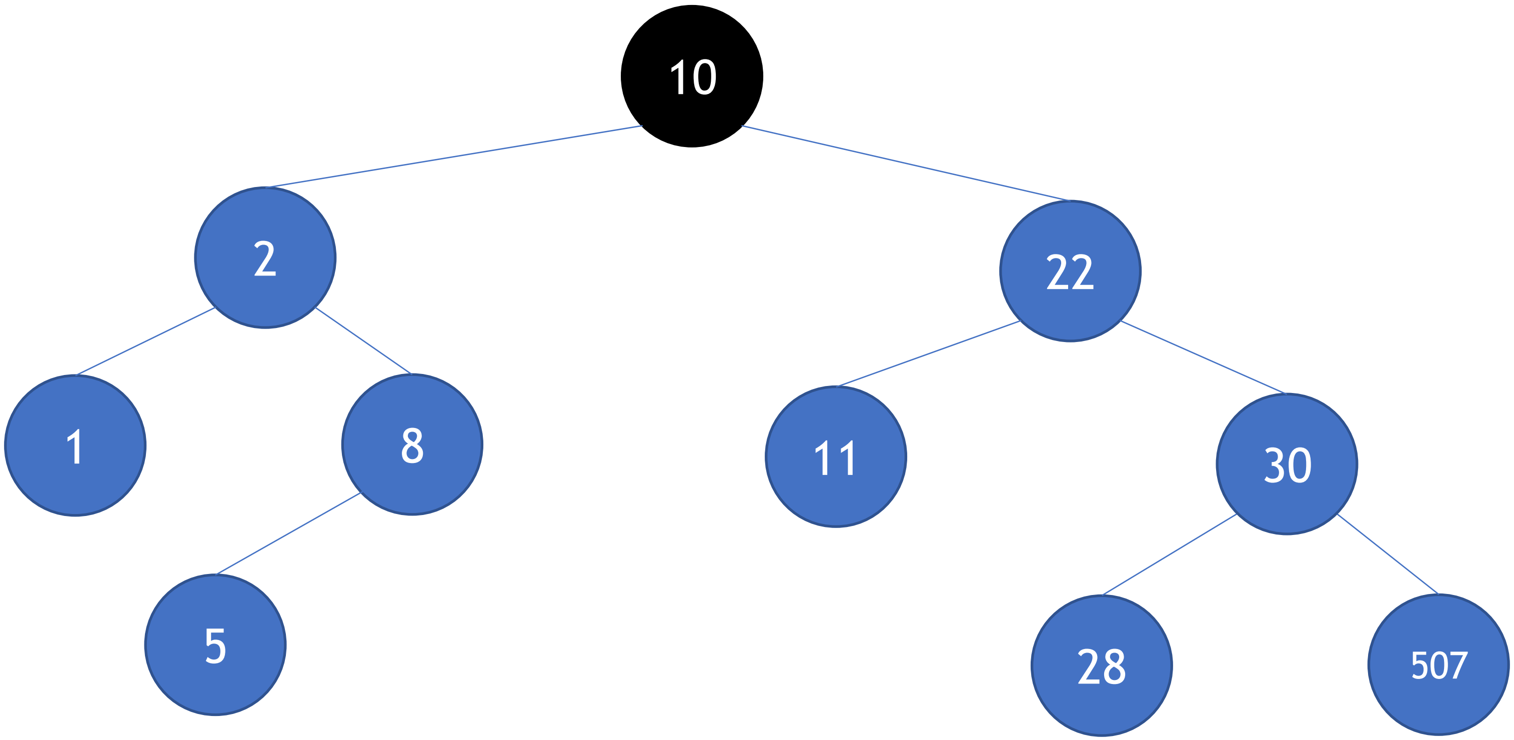
Fator de balanceamento

- Qual o fator de balanceamento no nó com chave 10?



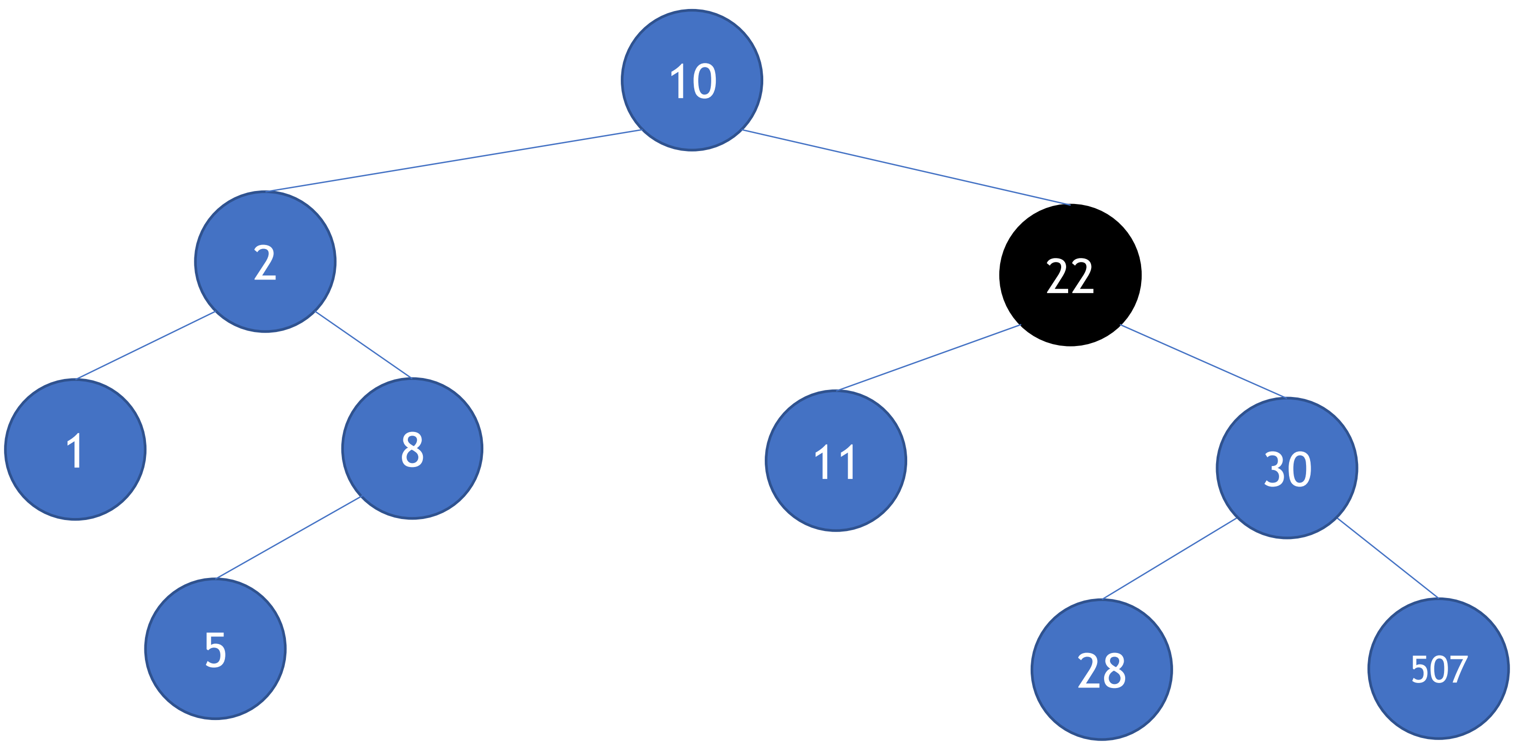
Fator de balanceamento

- Qual o fator de balanceamento no nó com chave 10? fb=0



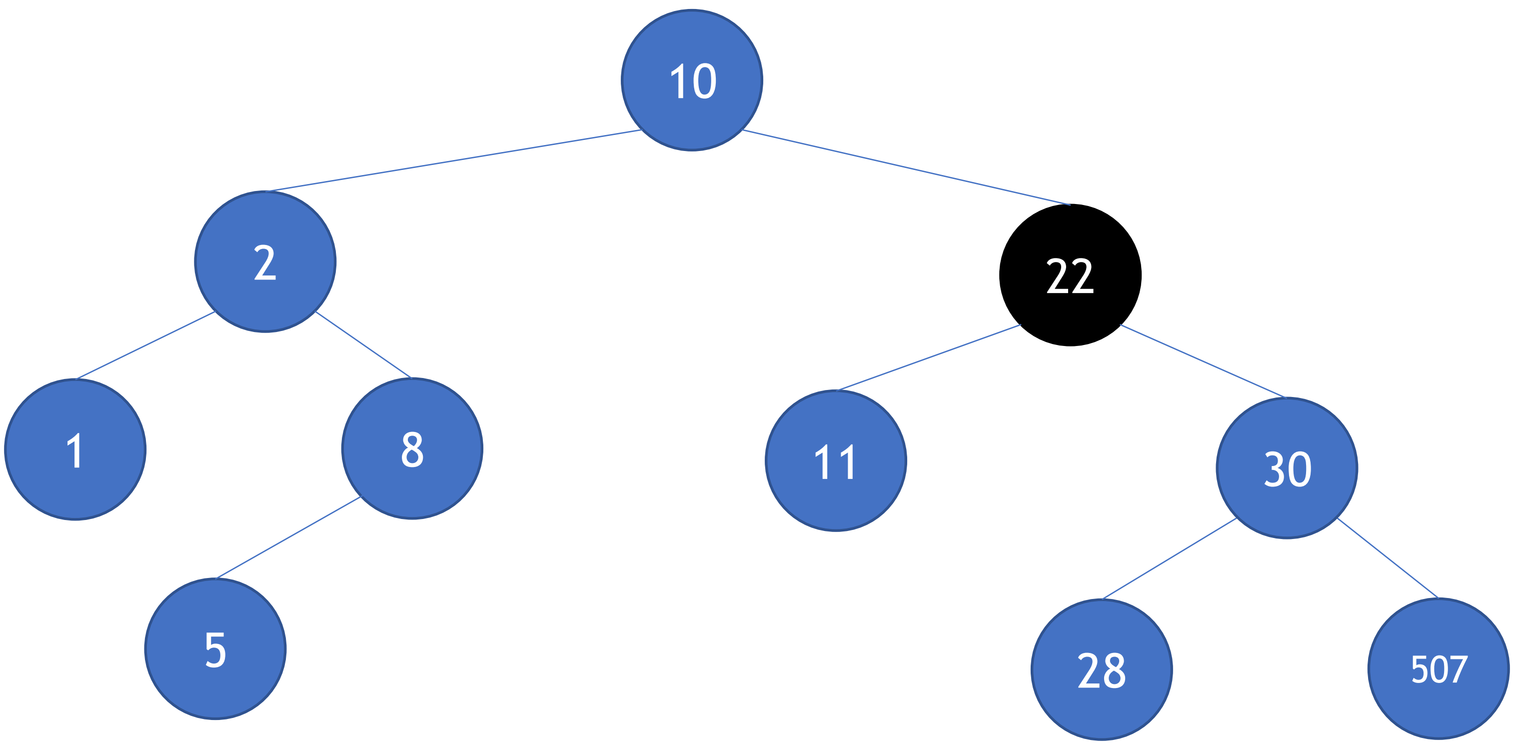
Fator de balanceamento

- Qual o fator de balanceamento no nó com chave 22?



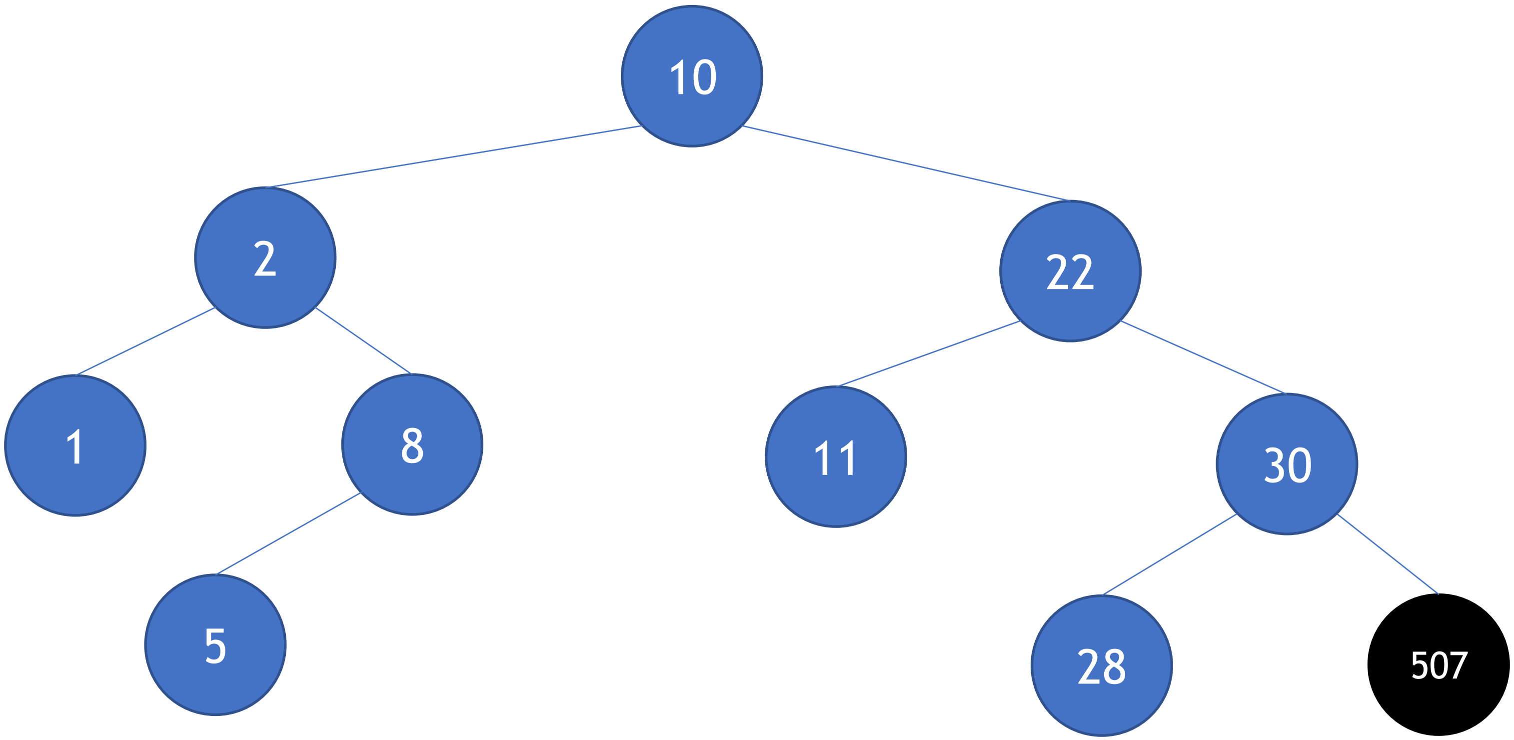
Fator de balanceamento

- Qual o fator de balanceamento no nó com chave 22? fb=-1



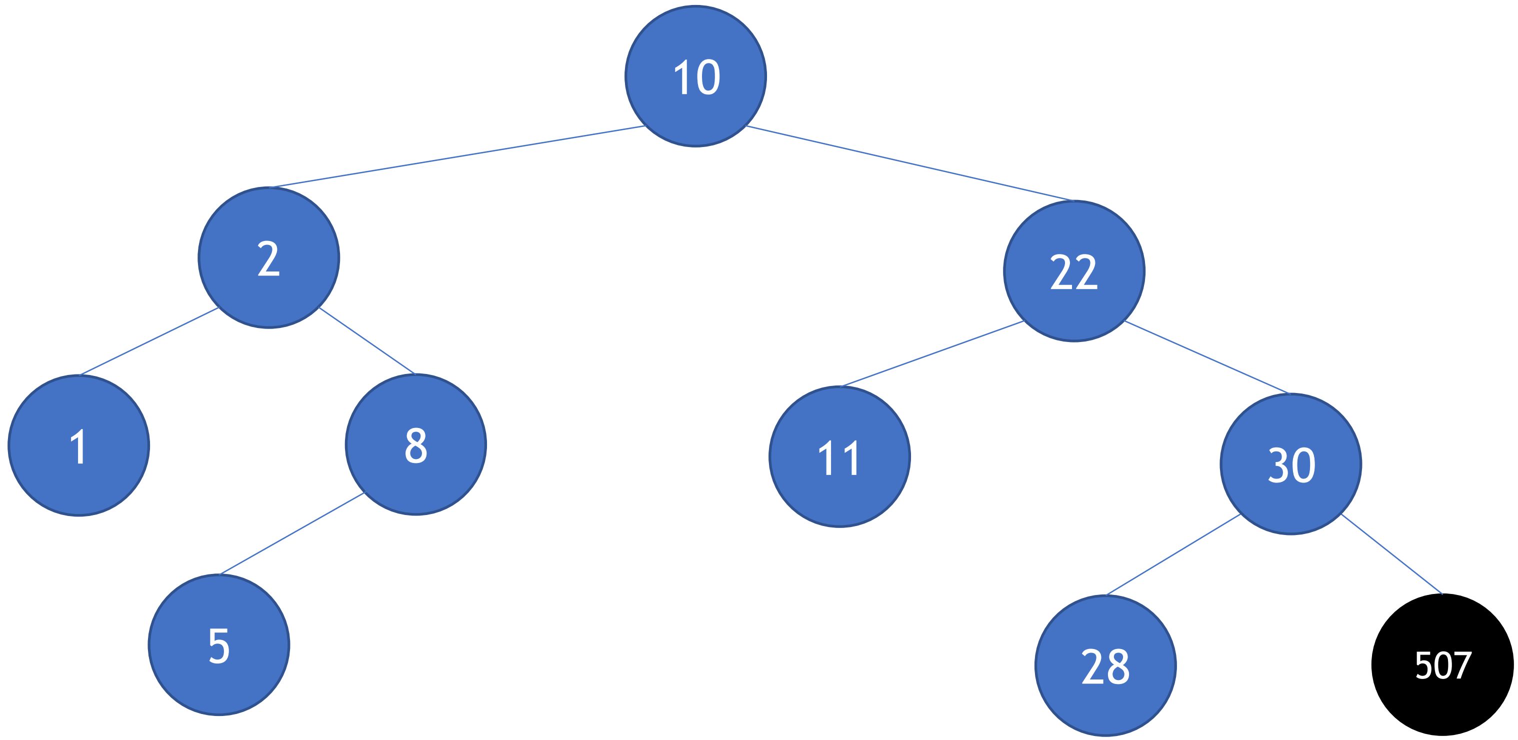
Fator de balanceamento

- Qual o fator de balanceamento no nó com chave 507?



Fator de balanceamento

- Qual o fator de balanceamento no nó com chave 507? fb=0



Fator de balanceamento

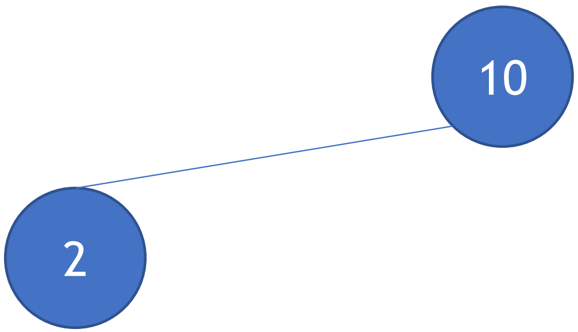
- É a diferença de altura(h) entre as subárvores esquerda e direita:

$$fb(v) = h(v.esq) - h(v.dir)$$

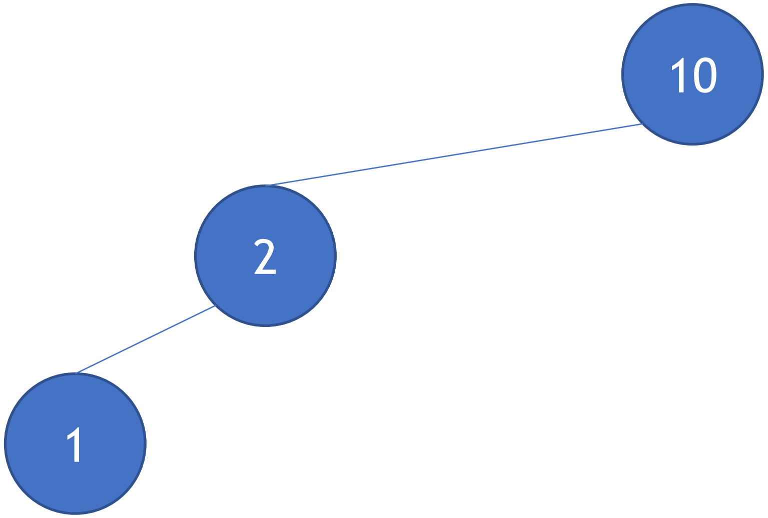
Em uma AVL, fb deve ser -1, 0 ou 1 para todos os nós.

Essa árvore é AVL?

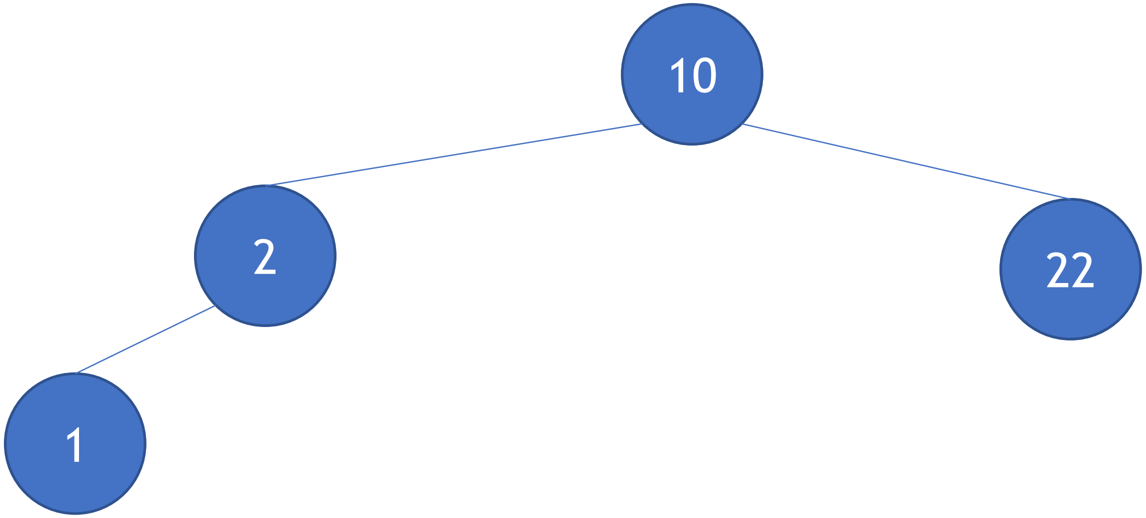
Essa árvore é AVL?



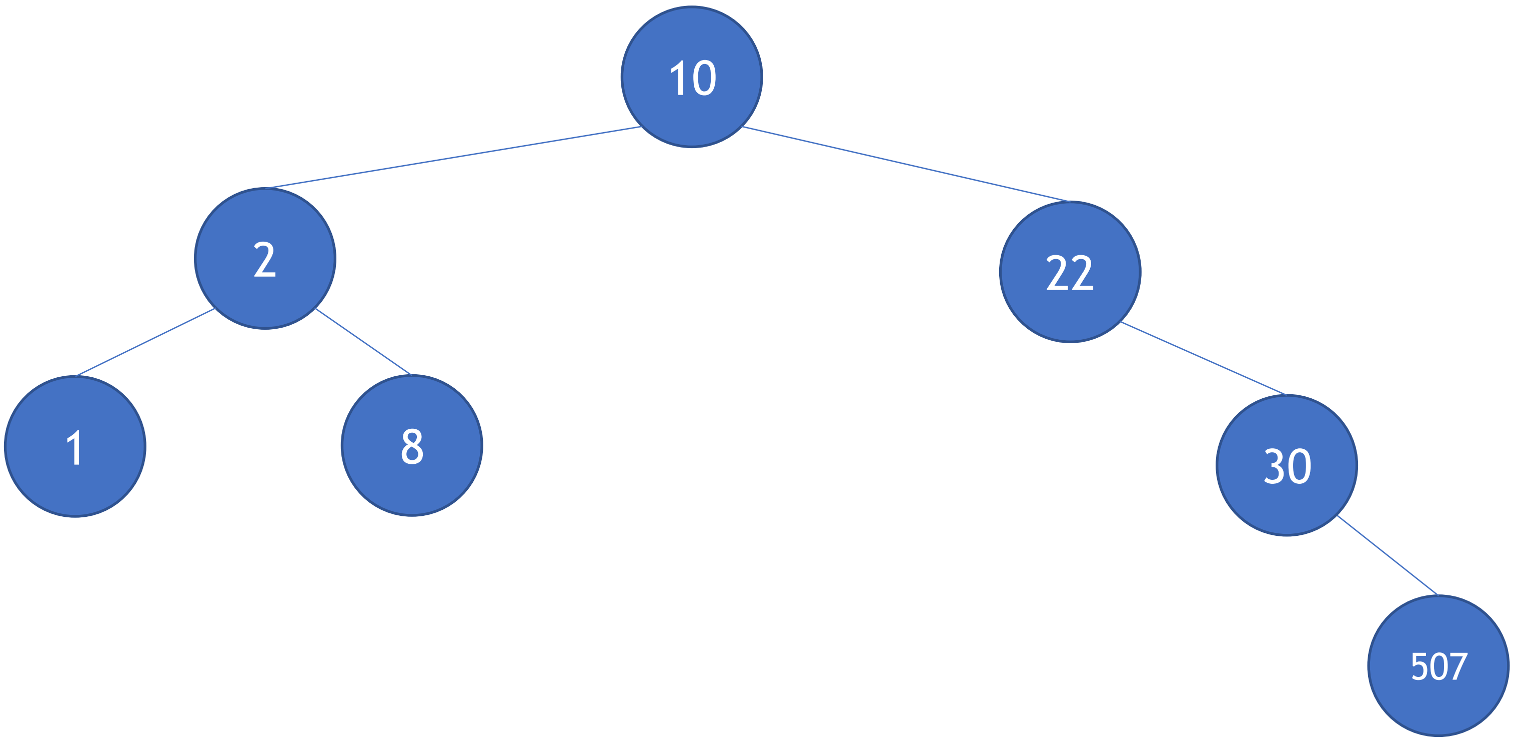
Essa árvore é AVL?



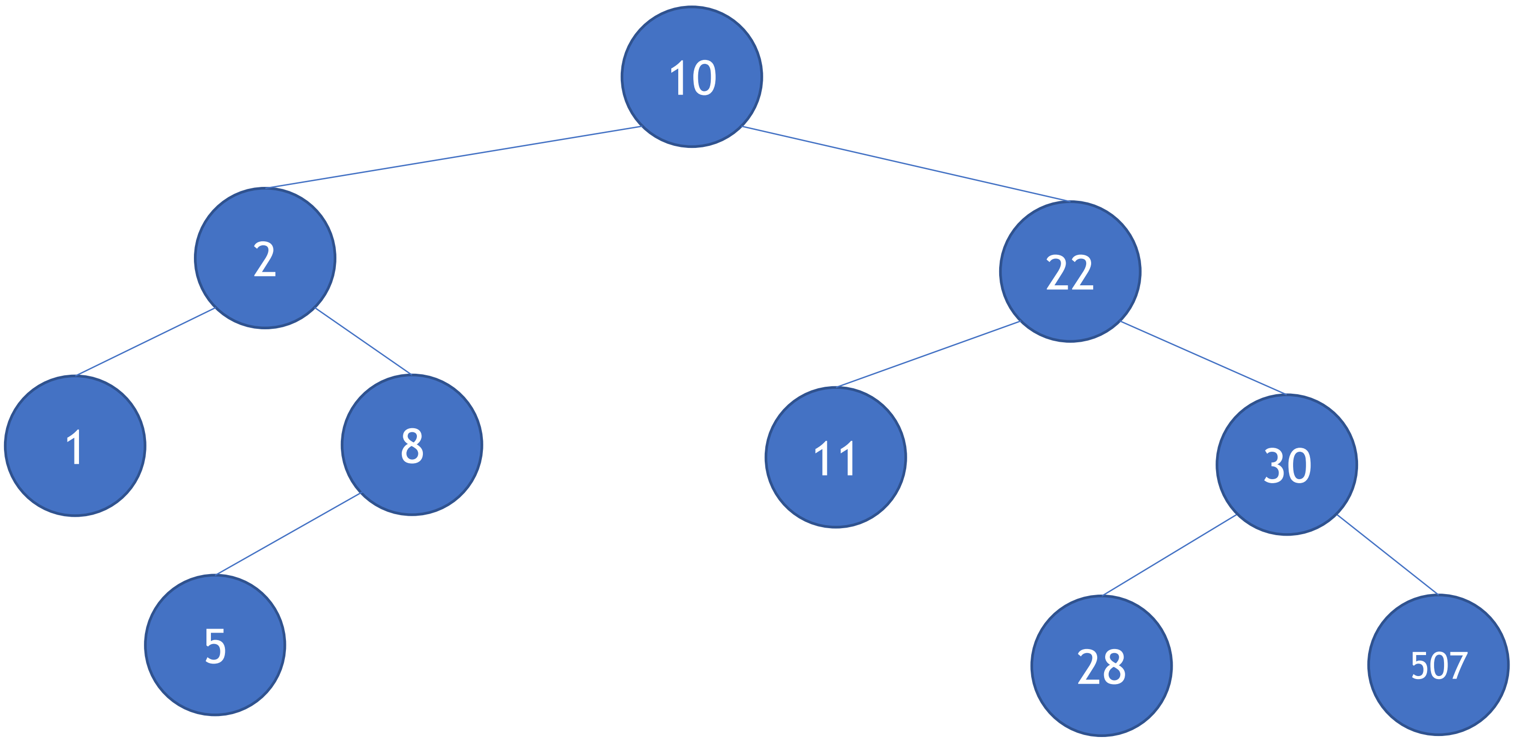
Essa árvore é AVL?



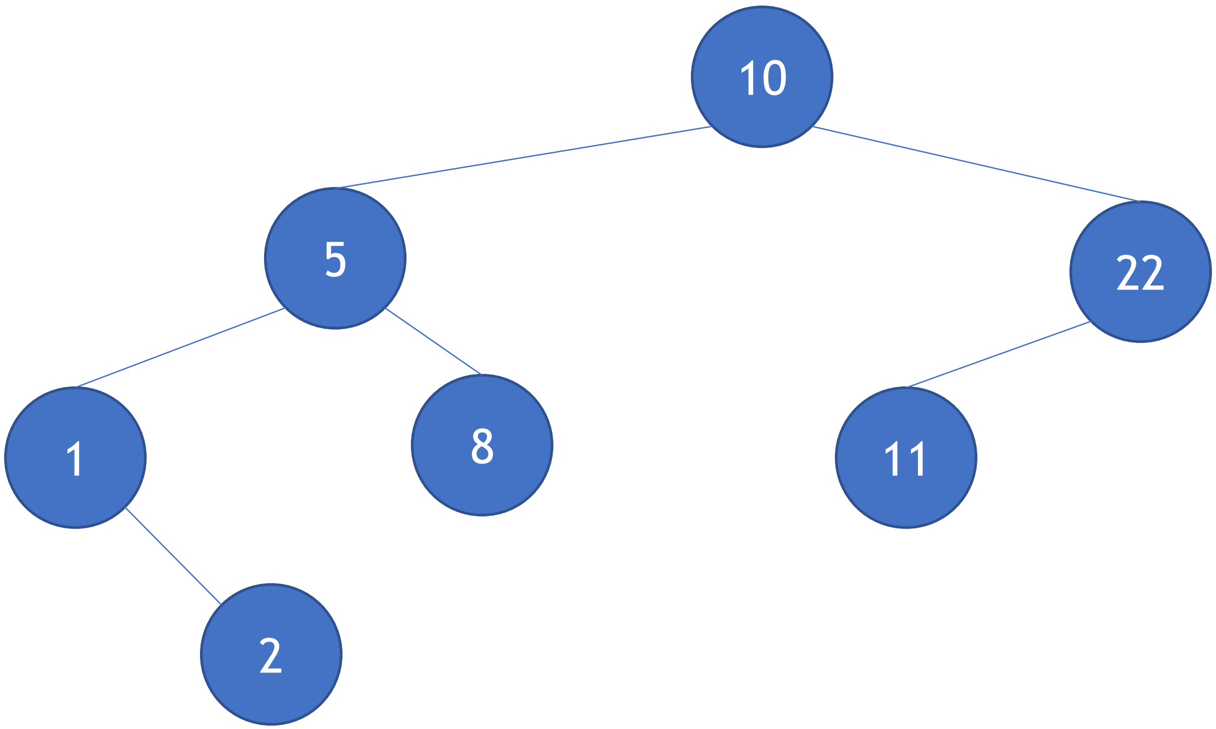
Essa árvore é AVL?



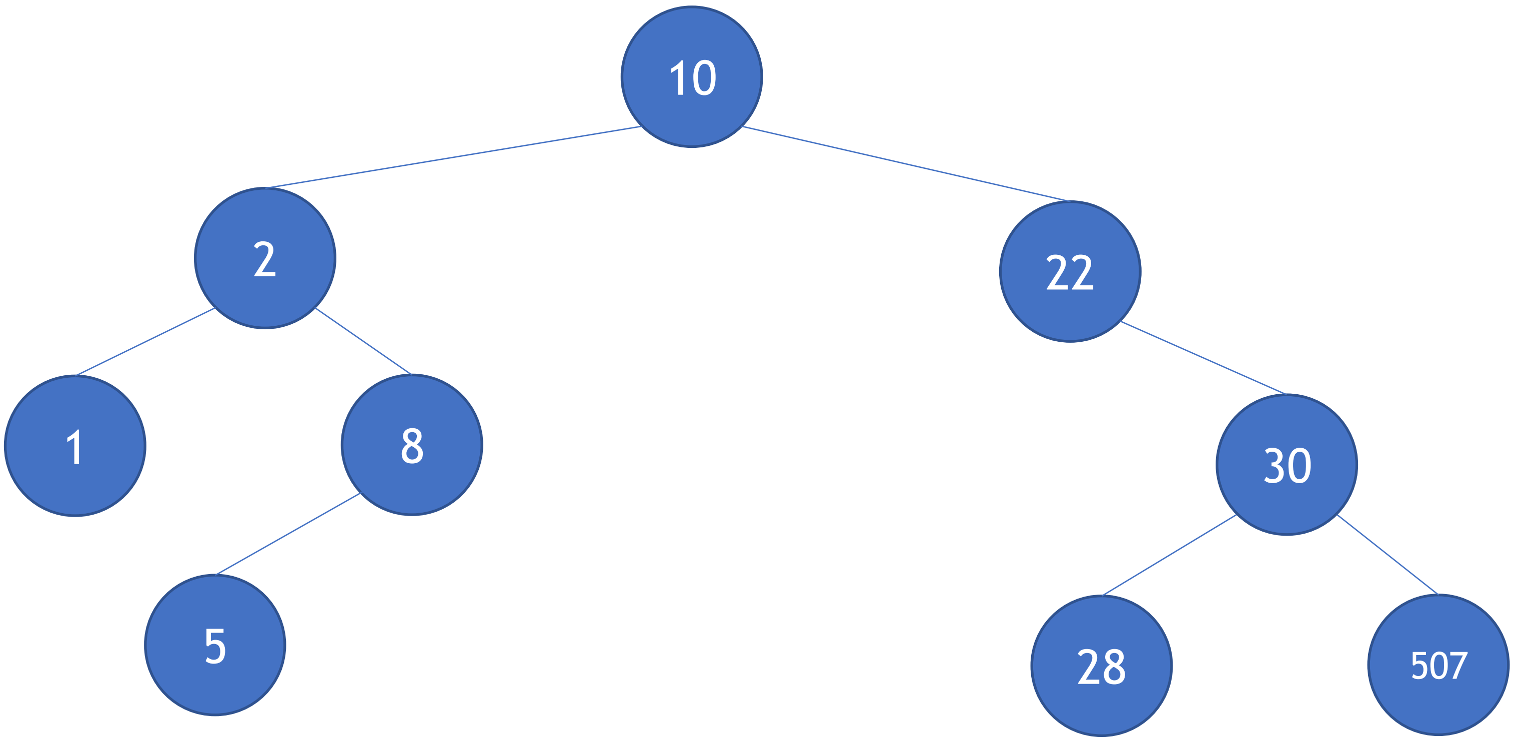
Essa árvore é AVL?



Essa árvore é AVL?



Essa árvore é AVL?



Fator de balanceamento

- Fator de balanceamento **positivo**: subárvore esquerda mais alta;
- Fator de balanceamento **negativo**: subárvore direita mais alta;

$$fb(v) = h(v.esq) - h(v.dir)$$

Fator de balanceamento

- Implementação em C

Chamada:
```
int fb = calcula_fb(no);
```

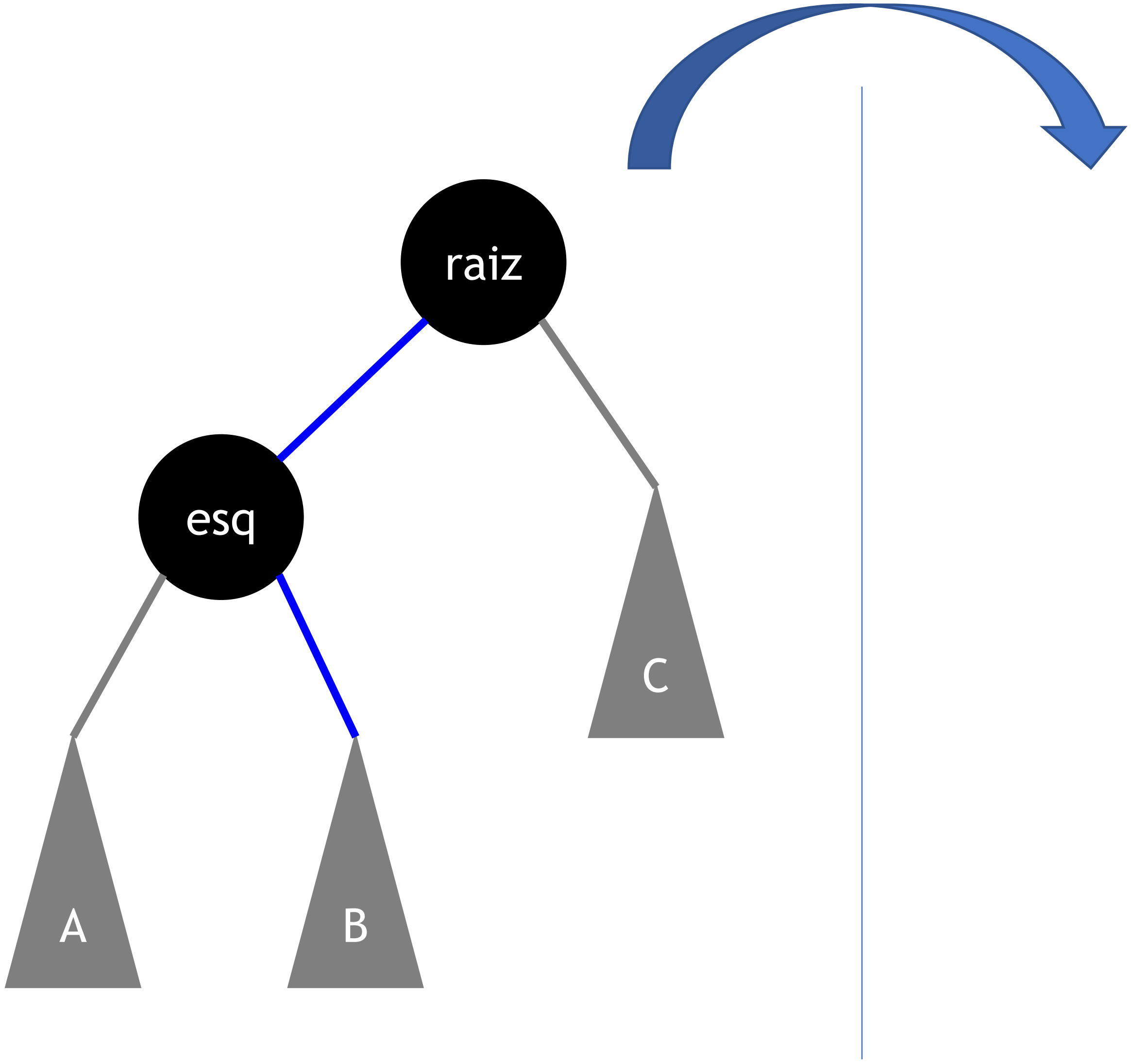
(código no vídeo)

Rotações

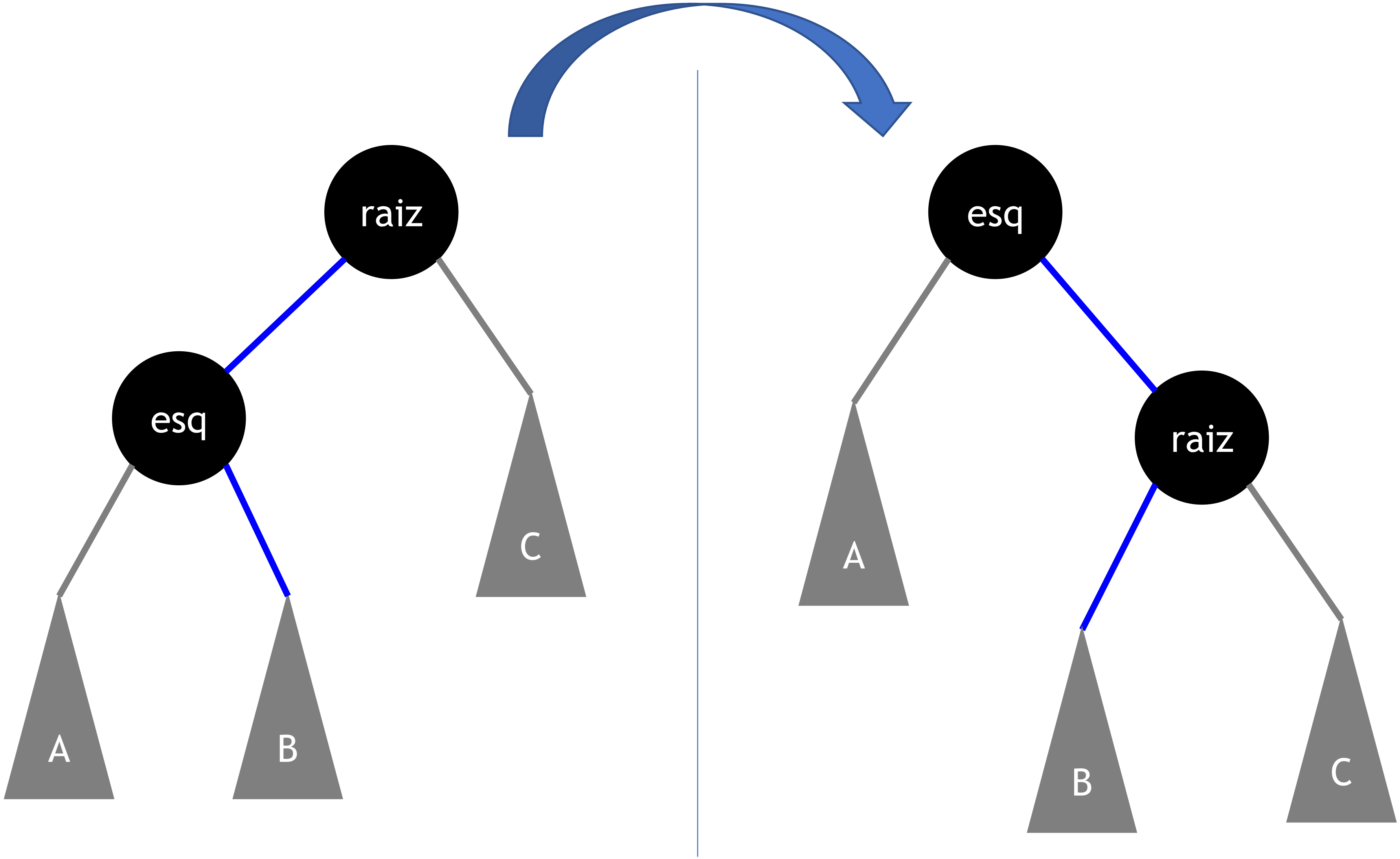
Rotações

- Para manter o balanceamento, podem ser realizadas rotações após a inserção e a remoção de nós na árvore;
- Há dois tipos de rotação em AVL: **esquerda** e **direita**; Mas há 4 casos de aplicação:
  - Rotação simples esquerda;
  - Rotação simples direita;
  - Rotação dupla esquerda;
  - Rotação dupla direita.

Rotação direita

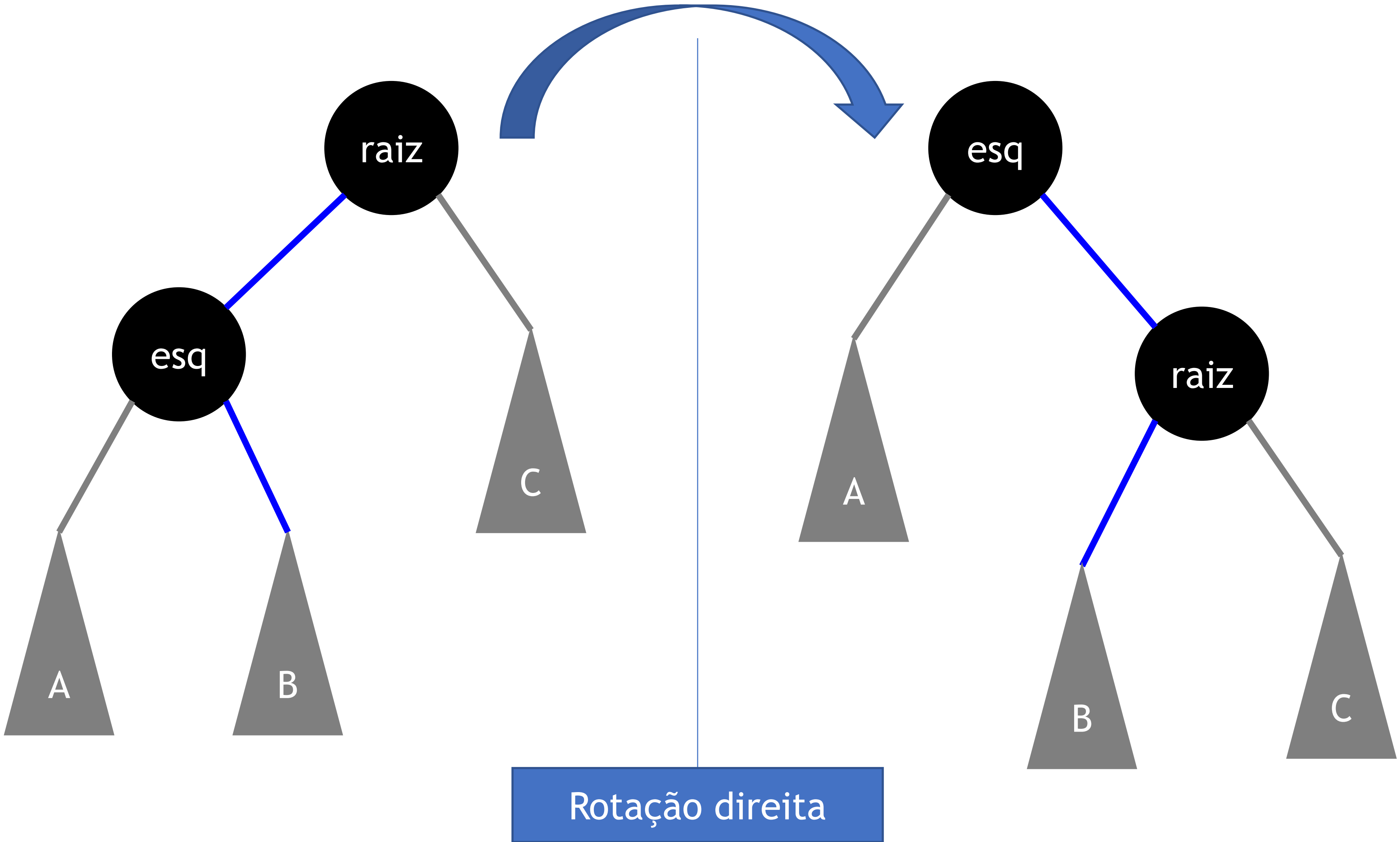


Rotação direita



Como é feita a rotação:
- A raiz passa a ser esq;
- Os dois ponteiros em azul são modificados.

Impacto da rotação:
- As alturas dos nós raiz e esq são alteradas.



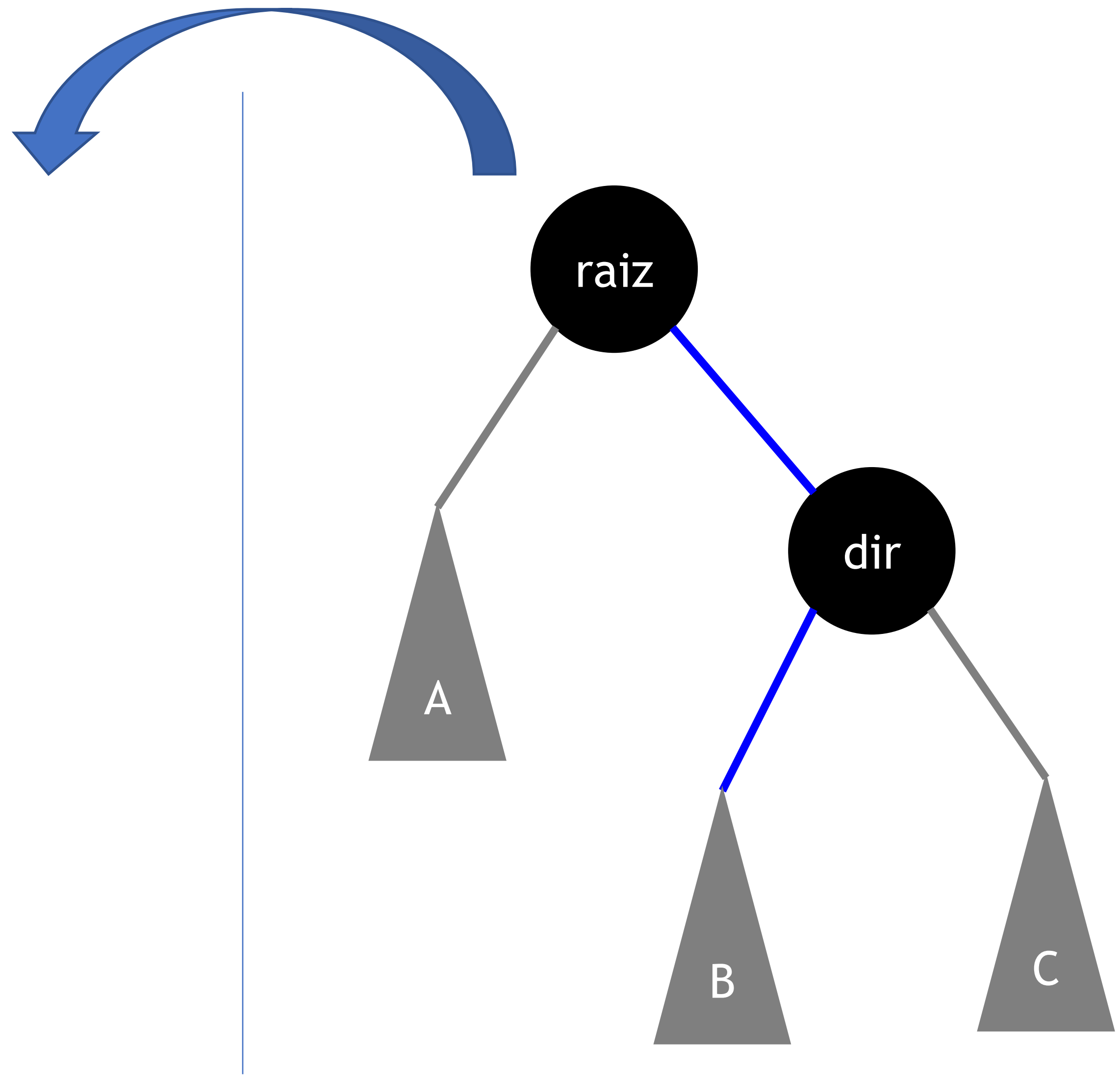
Rotação direita

- Implementação em C

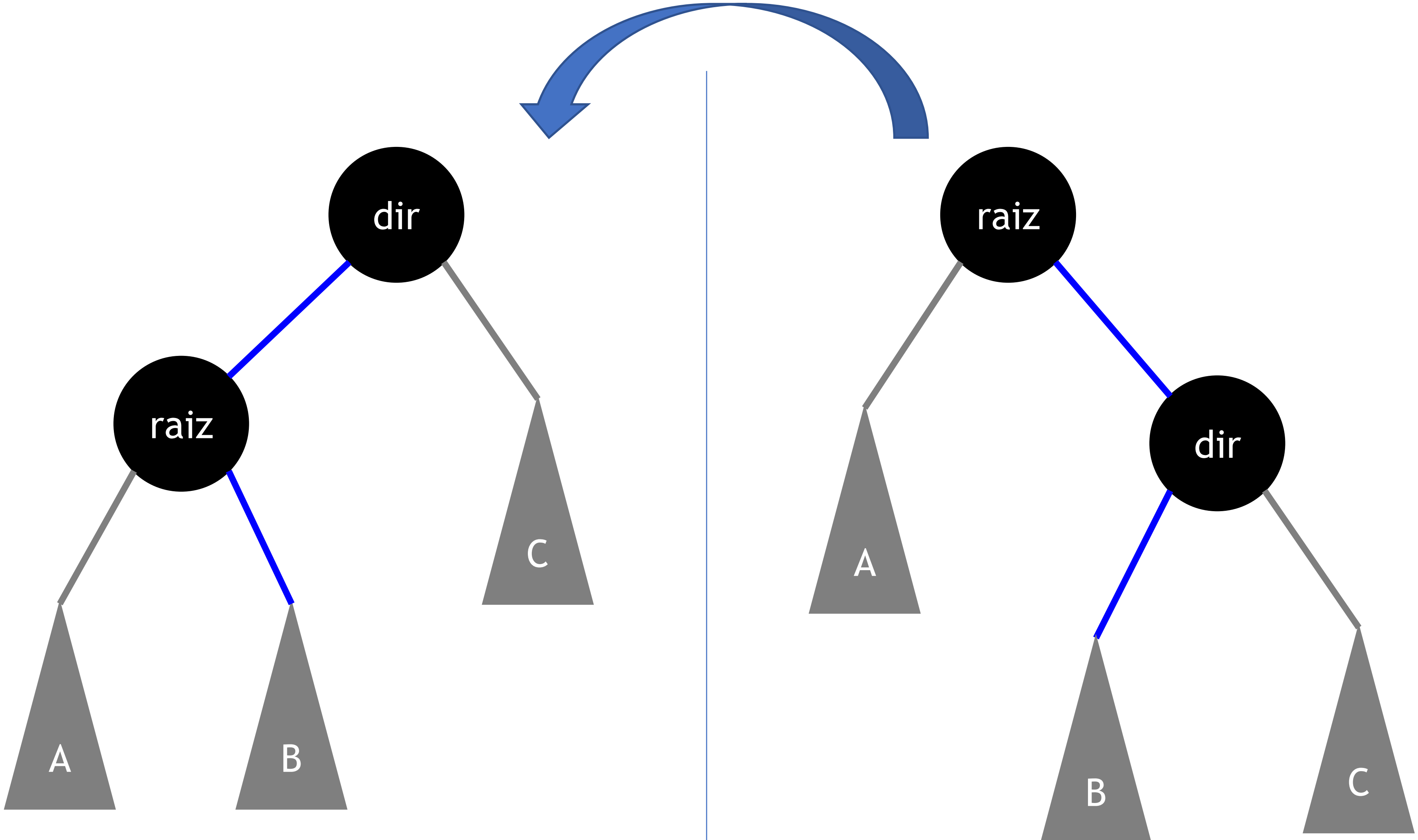
Chamada:
```
raiz = rotaciona_direita(raiz);
```

(código no vídeo)

Rotação esquerda

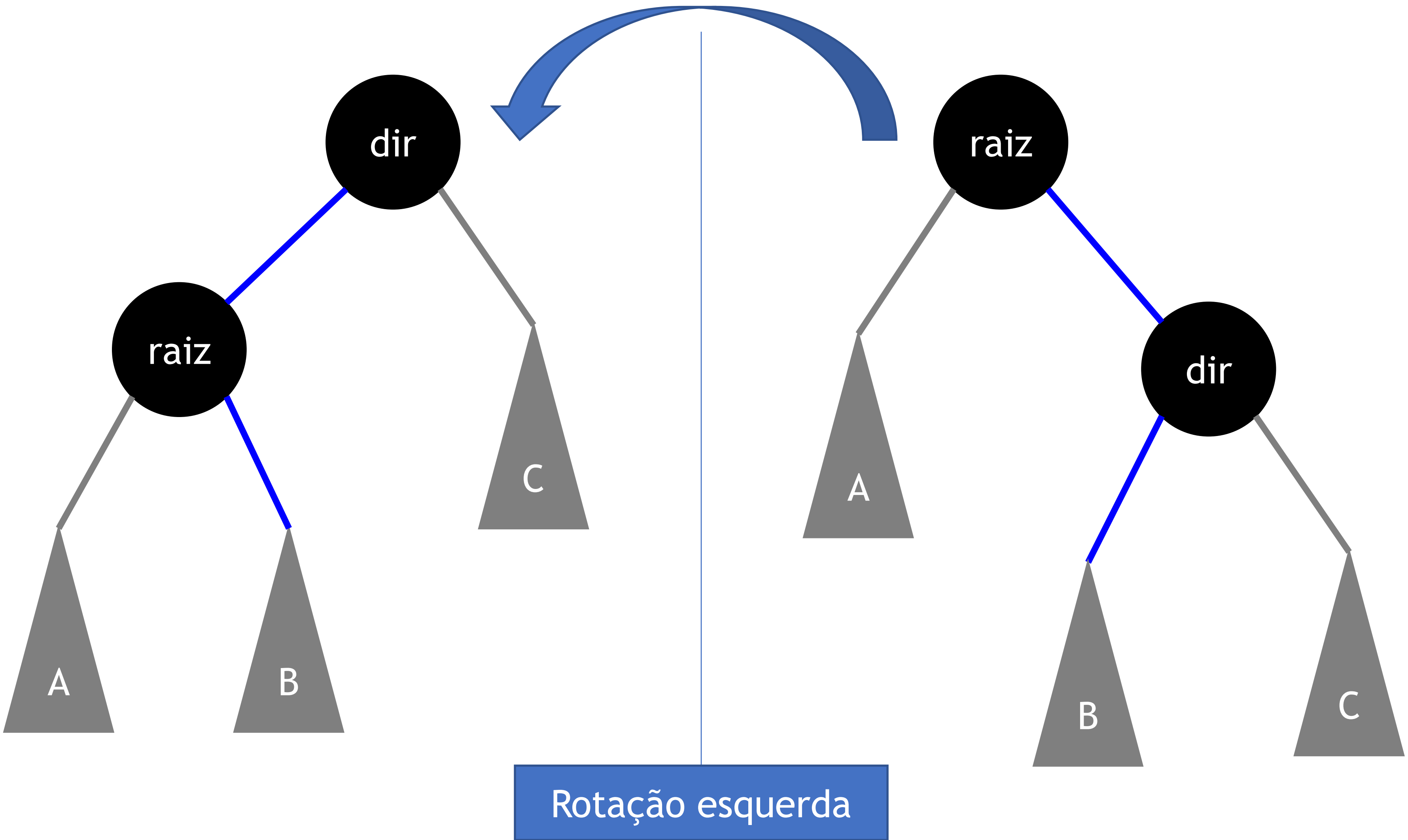


Rotação esquerda



Como é feita a rotação:
- A raiz passa a ser dir;
- Os dois ponteiros em azul são modificados.

Impacto da rotação:
- As alturas dos nós raiz e dir são alteradas.



Rotação esquerda

- Implementação em C

Chamada:
```
raiz = rotaciona_esquerda(raiz);
```

(código no vídeo)

Balanceamento

Balanceamento

- Após a inserção e a remoção, o balanceamento da árvore pode ser impactado;
- <u>Para manter a árvore balanceada na AVL, são aplicadas rotações;</u>
- Cada nó ancestral do nó alterado (até a raiz) deve ser verificado.

Balanceamento

- Se $|fb| > 1$, então há desbalanceamento;
- Existem 4 casos para balanceamento, que serão discutidos nos próximos slides;
- Será usado o exemplo da inserção, mas os 4 casos são os mesmos para remoção.

Rotação direita

- $fb(no) > 1$ → significa que a subárvore esquerda é mais alta; **Portanto, temos que <u>rotacionar para a direita</u> para balancear**;

- Esse caso se divide em dois:
  - 1A: $fb(no.esq) \geq 0$ → rotação simples direita;
  - 1B: $fb(no.esq) < 0$ → rotação dupla direita.

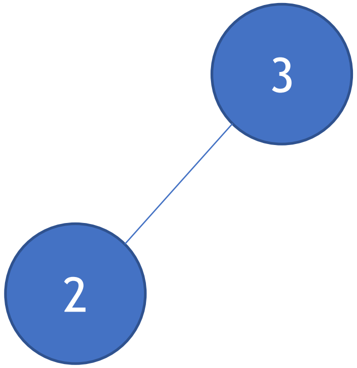
Caso 1A

Rotação simples direita

$fb(no) > 1$ e $fb(no.esq) \geq 0$

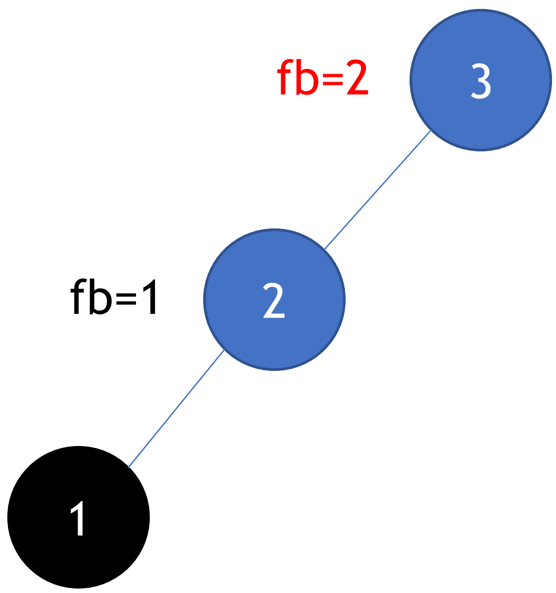
Caso 1A

- Exemplo (inserir chave 1):



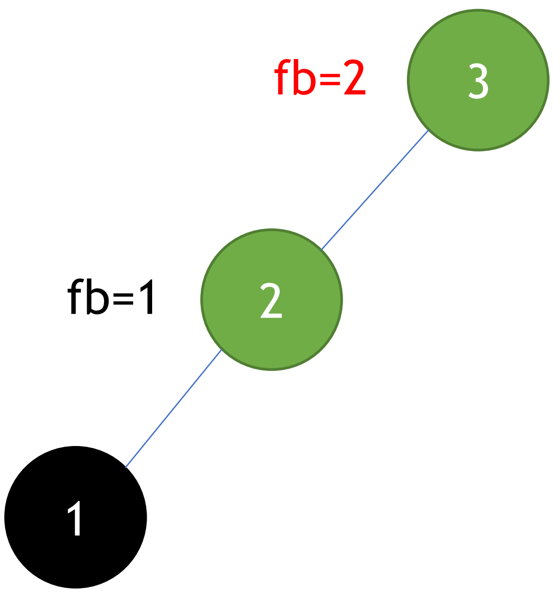
Caso 1A

- Exemplo (inserir chave 1):



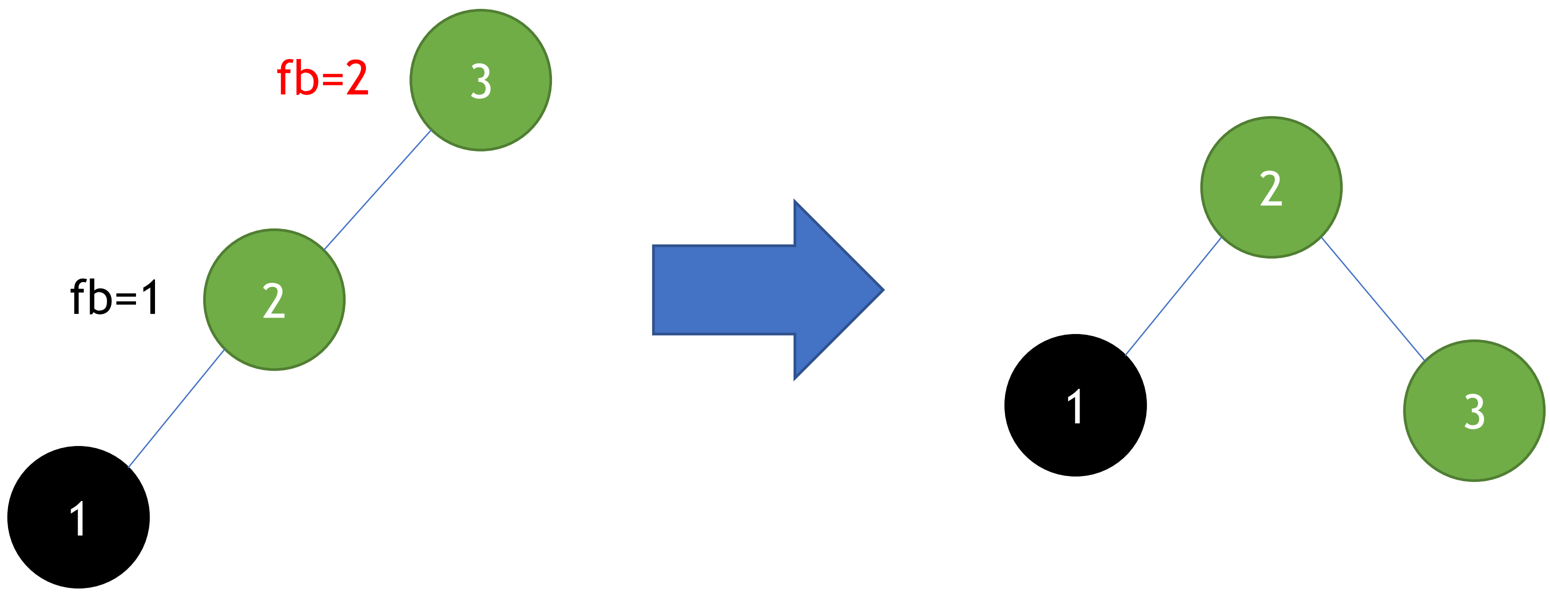
Caso 1A

- Exemplo (inserir chave 1):



Caso 1A

- Exemplo (inserir chave 1):



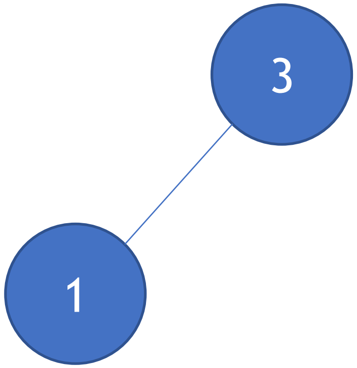
Caso 1B

Rotação dupla direita

$$fb(no) > 1 \text{ e } fb(no.esq) < 0$$

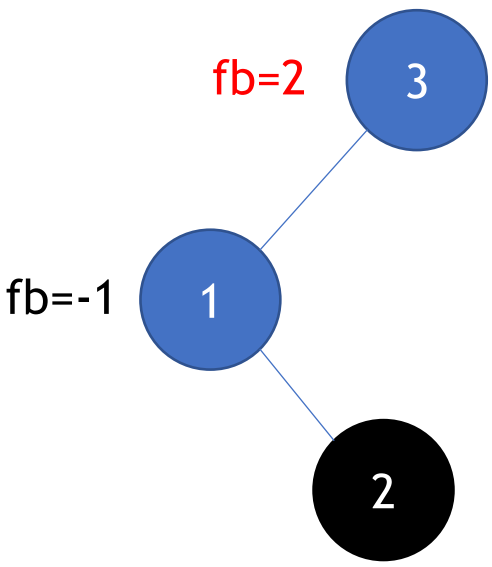
Caso 1B

- Exemplo (inserir chave 2):



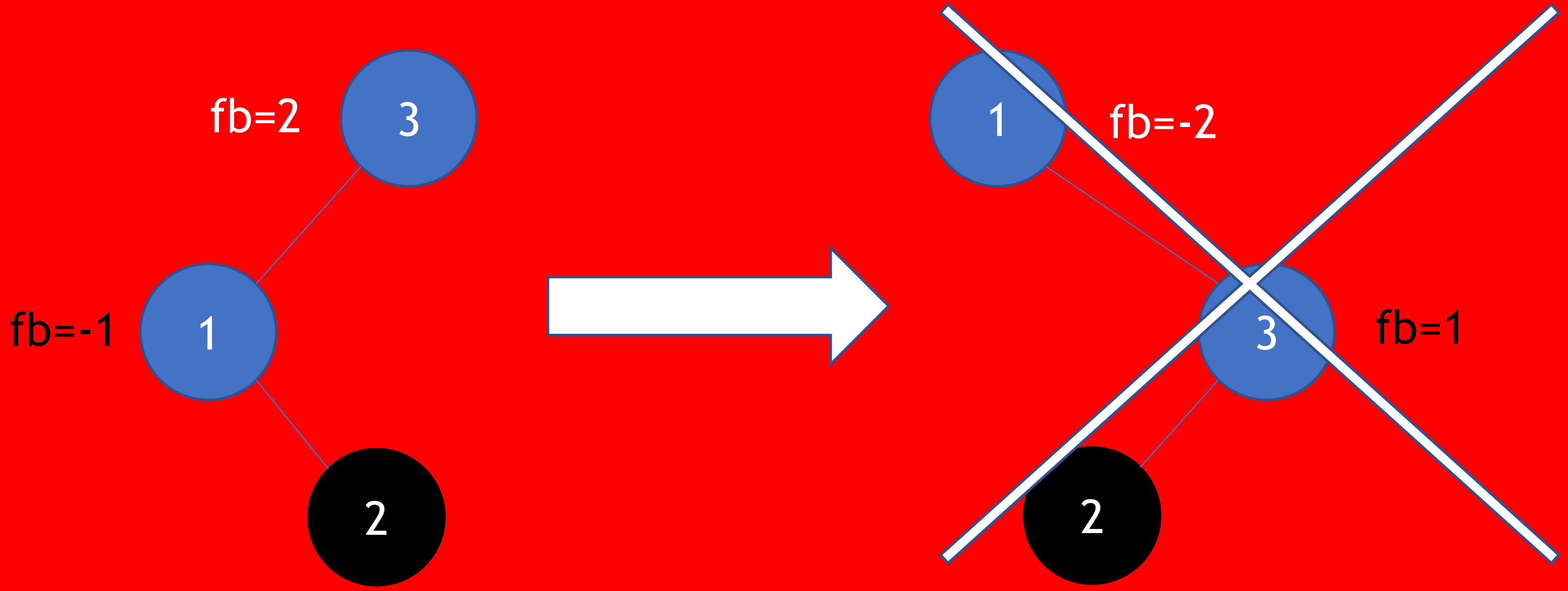
Caso 1B

- Exemplo (inserir chave 2):



Caso 1B

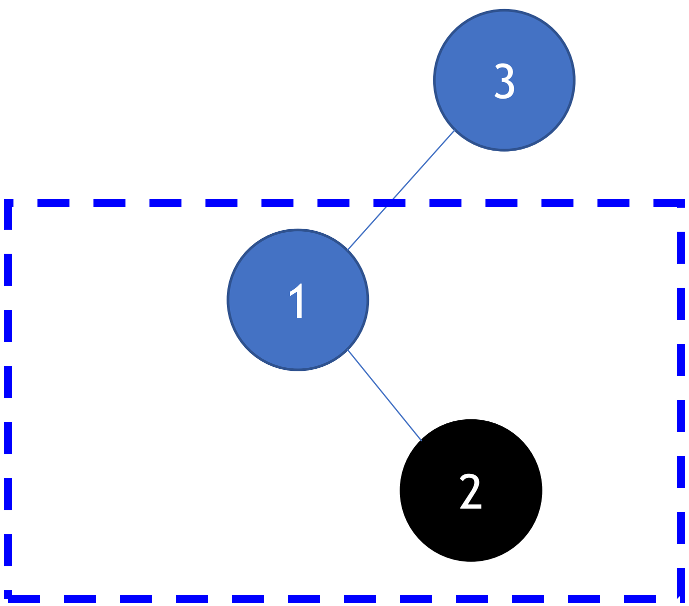
- Exemplo (inserir chave 2):



Rotação simples direita
manteria a árvore desbalanceada!

Caso 1B

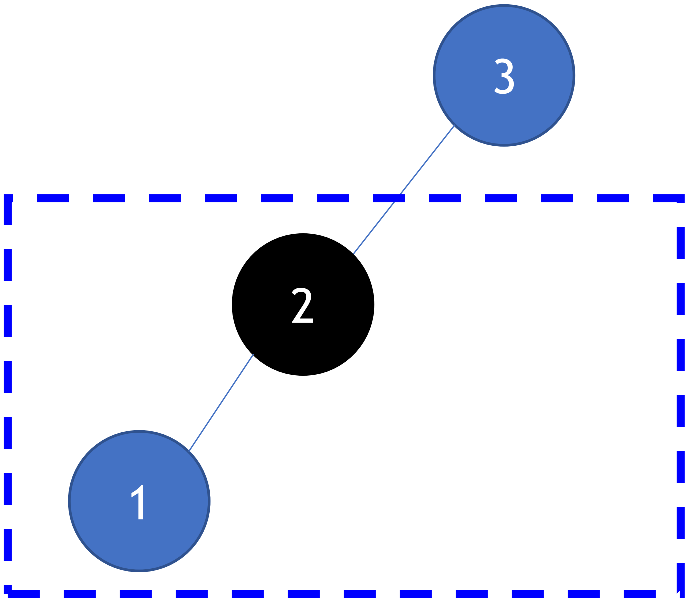
- Exemplo (inserir chave 2):



- Para poder aplicar a rotação direita, temos que primeiro fazer um ajuste;
- O ajuste é **rotacionar a subárvore esquerda para a esquerda**; Após essa rotação, poderemos aplicar a rotação na raiz.

Caso 1B

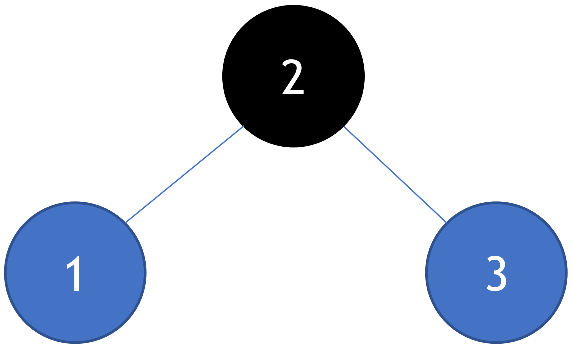
- Exemplo (inserir chave 2):



- Agora podemos aplicar a rotação direita na raiz.

Caso 1B

- Exemplo (inserir chave 2):

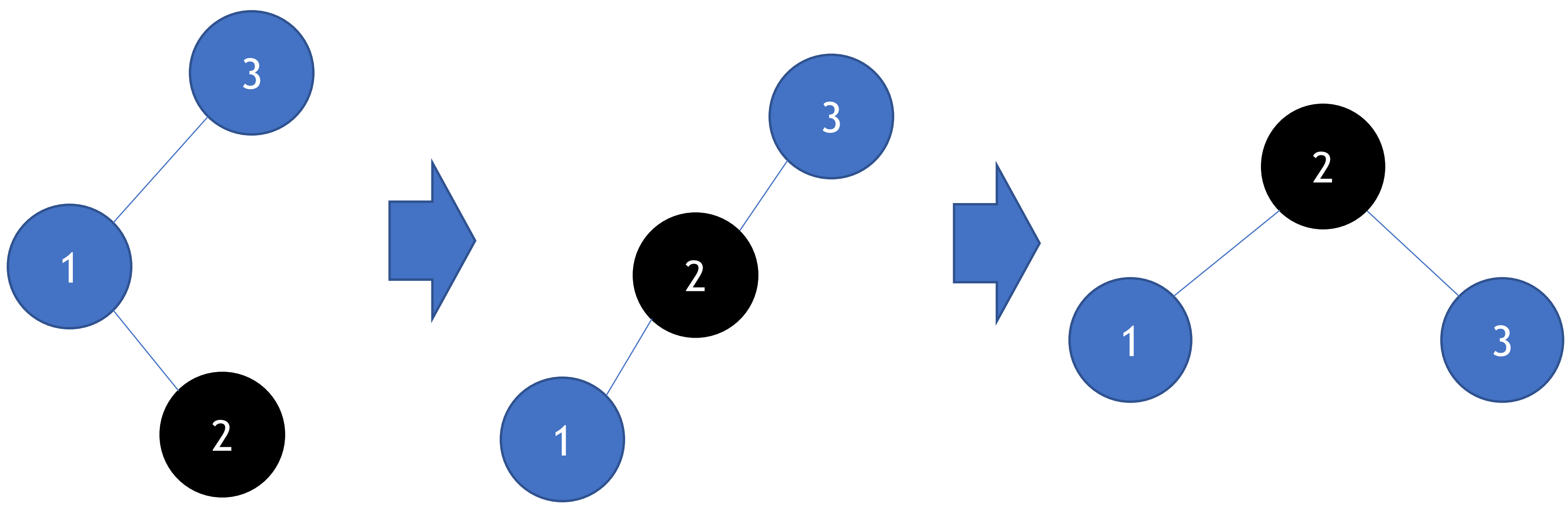


Caso 1B

Rotação dupla direita

$$fb(no) > 1 \text{ e } fb(no.esq) < 0$$

- Exemplo (inserir chave 2):



Inserção inicial   Rotação 1 (esquerda)   Rotação 2 (direita)

Rotação esquerda

- $fb(no) < -1$ → significa que a subárvore direita é mais alta; **Portanto, temos que <u>rotacionar para a esquerda</u> para balancear**;

- Esse caso se divide em dois:
  - 2A: $fb(no.dir) \leq 0$ → rotação simples esquerda;
  - 2B: $fb(no.dir) > 0$ → rotação dupla esquerda;

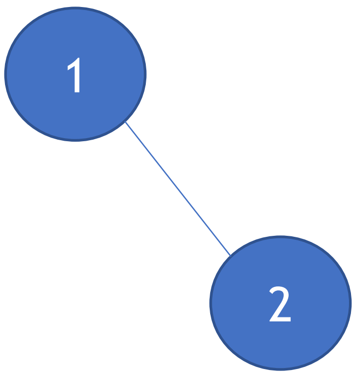
Caso 2A

Rotação simples esquerda

$$fb(no) < -1 \text{ e } fb(no.dir) \leq 0$$

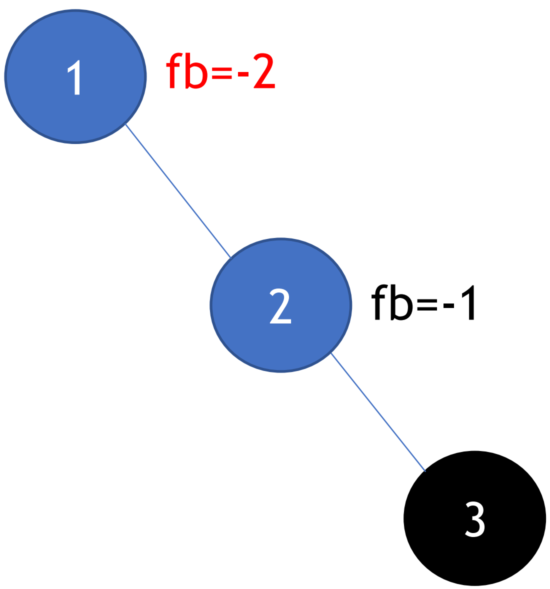
Caso 2A

- Exemplo (inserir chave 3):



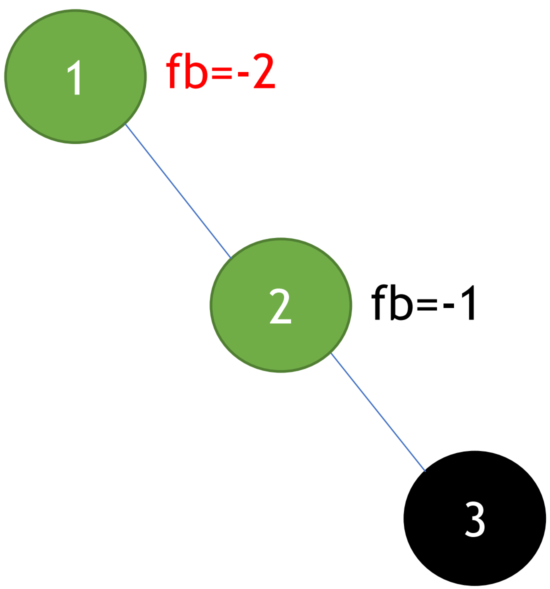
Caso 2A

- Exemplo (inserir chave 3):



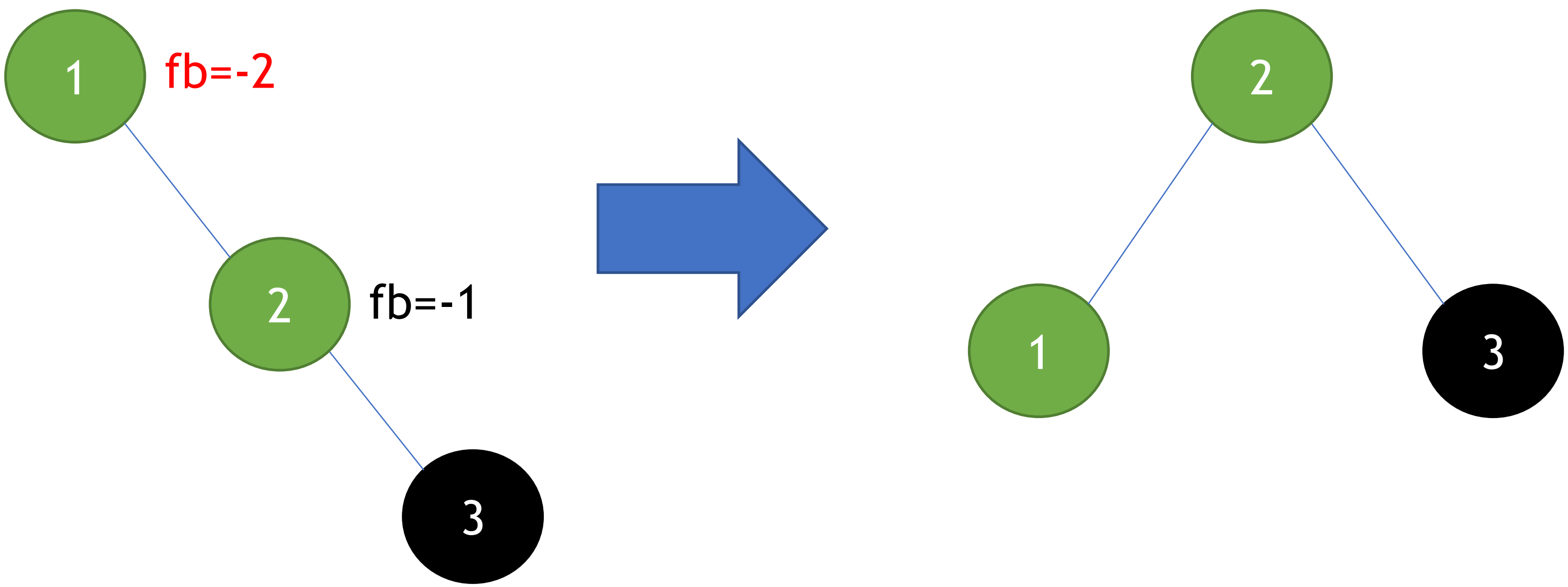
Caso 2A

- Exemplo (inserir chave 3):



Caso 2A

- Exemplo (inserir chave 3):



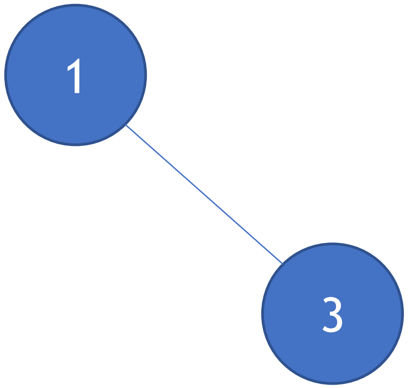
Caso 2B

Rotação dupla esquerda

$fb(no) < -1$ e $fb(no.dir) > 0$

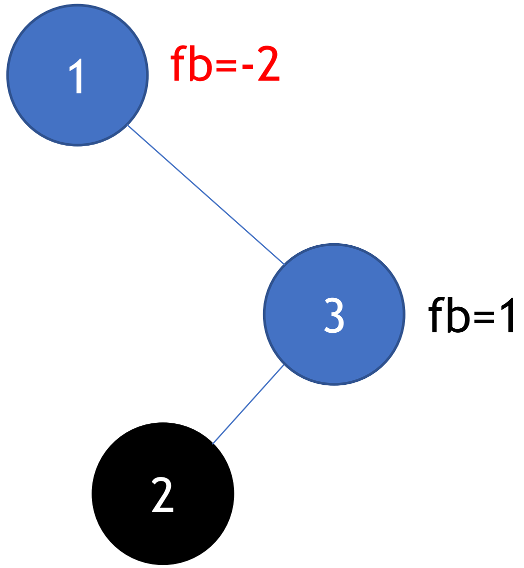
Caso 2B

- Exemplo (inserir chave 2):



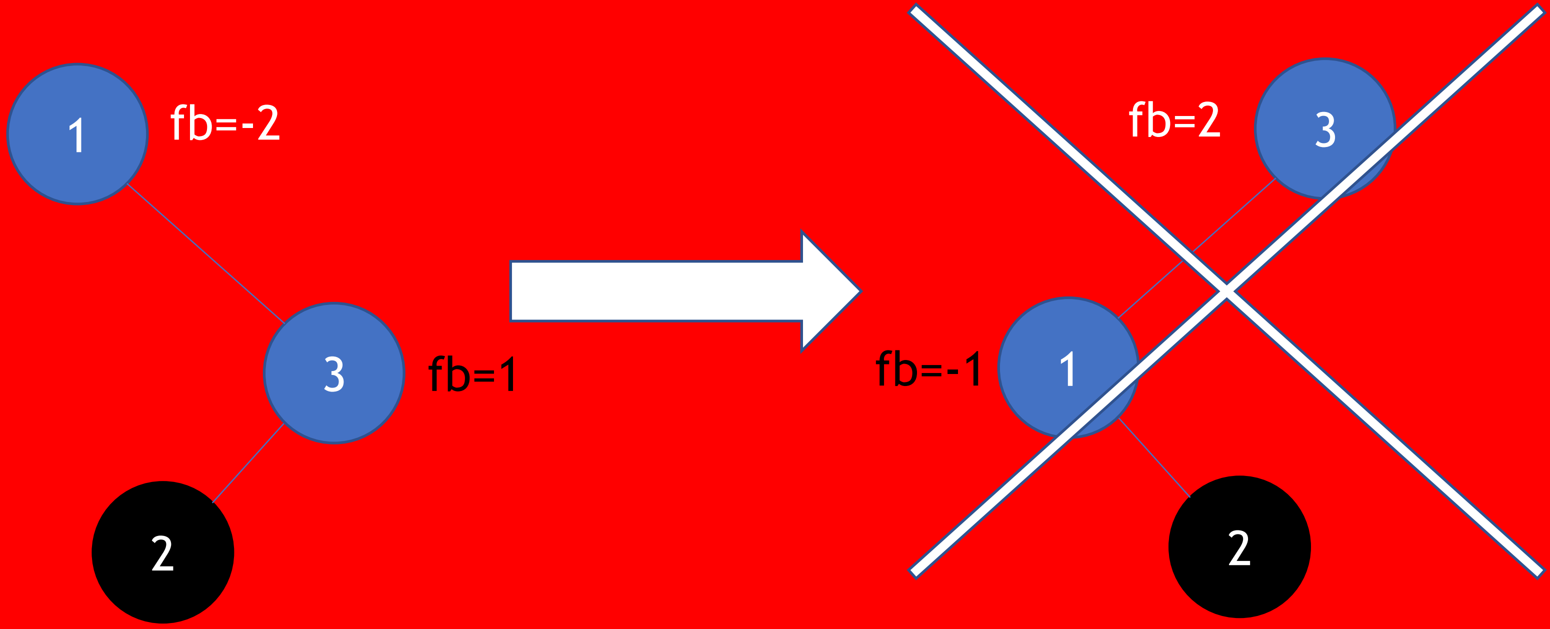
Caso 2B

- Exemplo (inserir chave 2):



Caso 2B

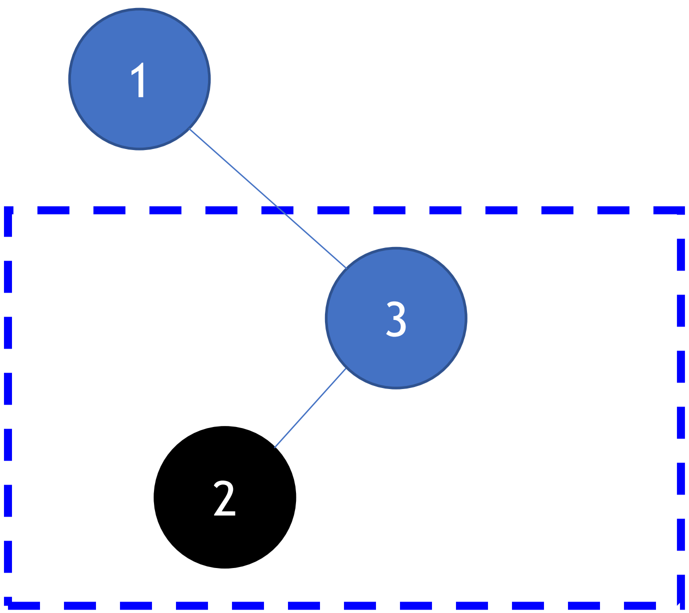
- Exemplo (inserir chave 2):



Rotação simples esquerda
manteria a árvore desbalanceada!

Caso 2B

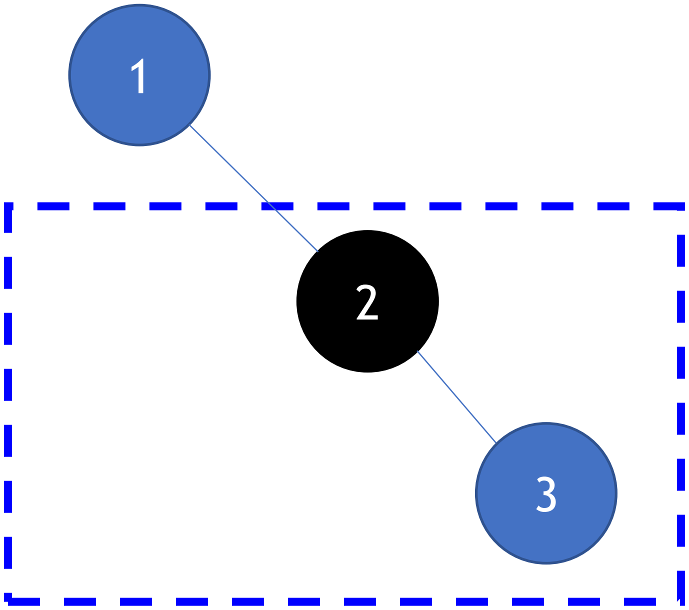
- Exemplo (inserir chave 2):



- Para poder aplicar a rotação esquerda, temos que primeiro fazer um ajuste;
- O ajuste é **rotacionar a subárvore direita para a direita**; Após essa rotação, poderemos aplicar a rotação na raiz.

Caso 2B

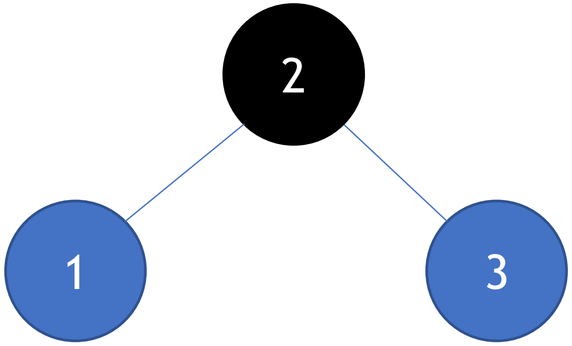
- Exemplo (inserir chave 2):



- Agora podemos aplicar a rotação esquerda na raiz.

Caso 2B

- Exemplo (inserir chave 2):

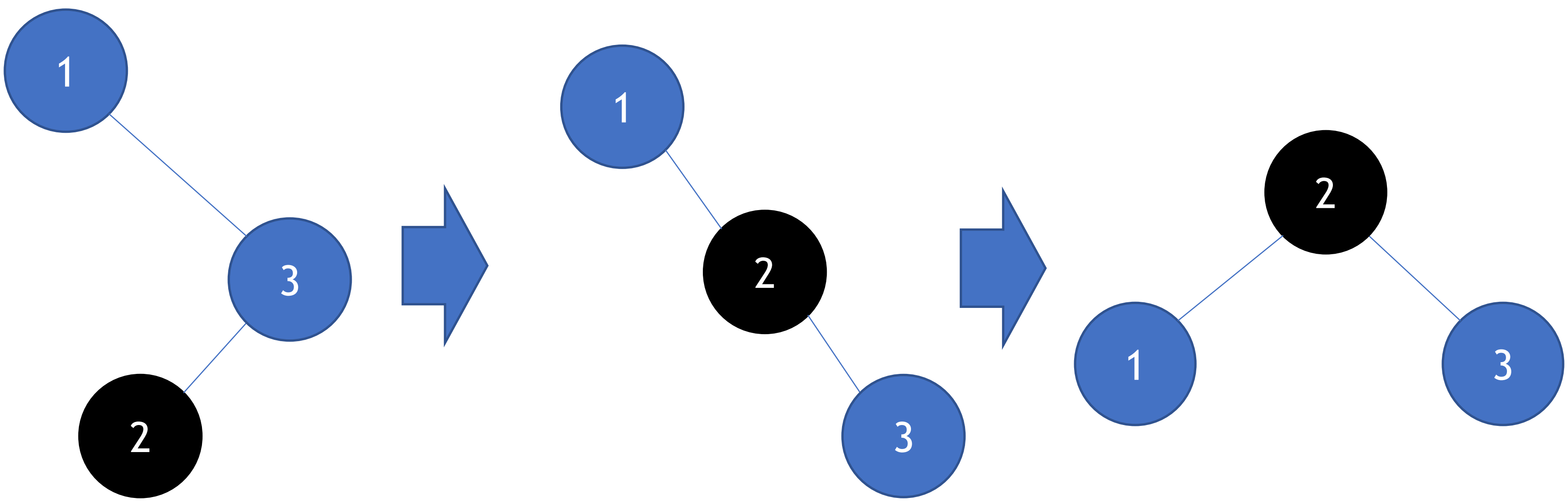


Caso 2B

Rotação dupla esquerda

$fb(no) < -1$ e $fb(no.dir) > 0$

- Exemplo (inserir chave 2):



Inserção inicial

Rotação 1 (direita)

Rotação 2 (esquerda)

Resumo

Quando a árvore está mais alta na esquerda, o filho da esquerda é verificado:

$fb(no) > 1$ e $fb(no.esq) \geq 0$ → Rotação simples direita

$fb(no) > 1$ e $fb(no.esq) < 0$ → Rotação dupla direita

Quando a árvore está mais alta na direita, o filho da direita é verificado:

$fb(no) < -1$ e $fb(no.dir) \leq 0$ → Rotação simples esquerda

$fb(no) < -1$ e $fb(no.dir) > 0$ → Rotação dupla esquerda

Resumo

Quando a árvore está mais alta na esquerda, o filho da esquerda é verificado:

$fb(no) > 1$ e $fb(no.esq) \geq 0$ → Rotação simples direita

$fb(no) > 1$ e $fb(no.esq) < 0$ → Rotação dupla direita

Filho esquerdo com subárvore direita maior.

Quando a árvore está mais alta na direita, o filho da direita é verificado:

$fb(no) < -1$ e $fb(no.dir) \leq 0$ → Rotação simples esquerda

$fb(no) < -1$ e $fb(no.dir) > 0$ → Rotação dupla esquerda

Filho direito com subárvore esquerda maior.

Balanceamento

- Implementação em C

Chamada:
```
raiz = balancear(raiz);
```

(código no vídeo)

Após a inserção/remoção, esta função deve ser chamada para cada um dos nós ancestrais do nó alterado (até a raiz).

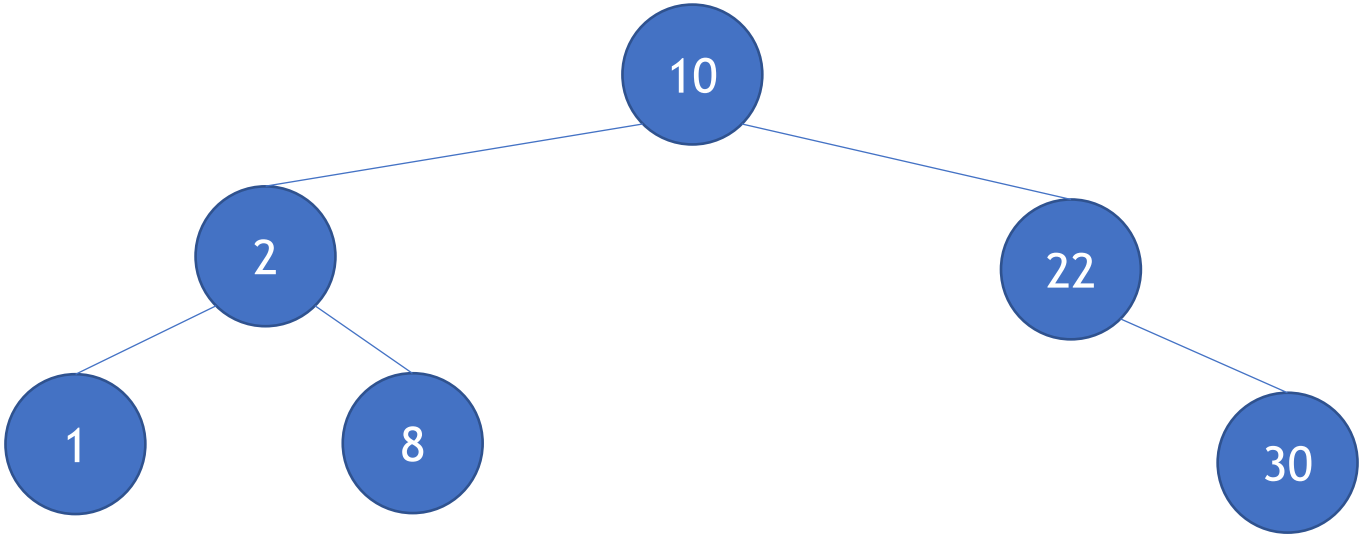
Operações: inserção e remoção

Operações de inserção e remoção

- Primeiro, as operações são realizadas como na árvore binária de busca que vimos;
- Após a inserção ou a remoção, o balanceamento da árvore pode ser impactado;
- <u>Para manter a árvore balanceada na AVL, são aplicadas rotações;</u>
- Cada nó ancestral do nó alterado (até a raiz) deve ser verificado.

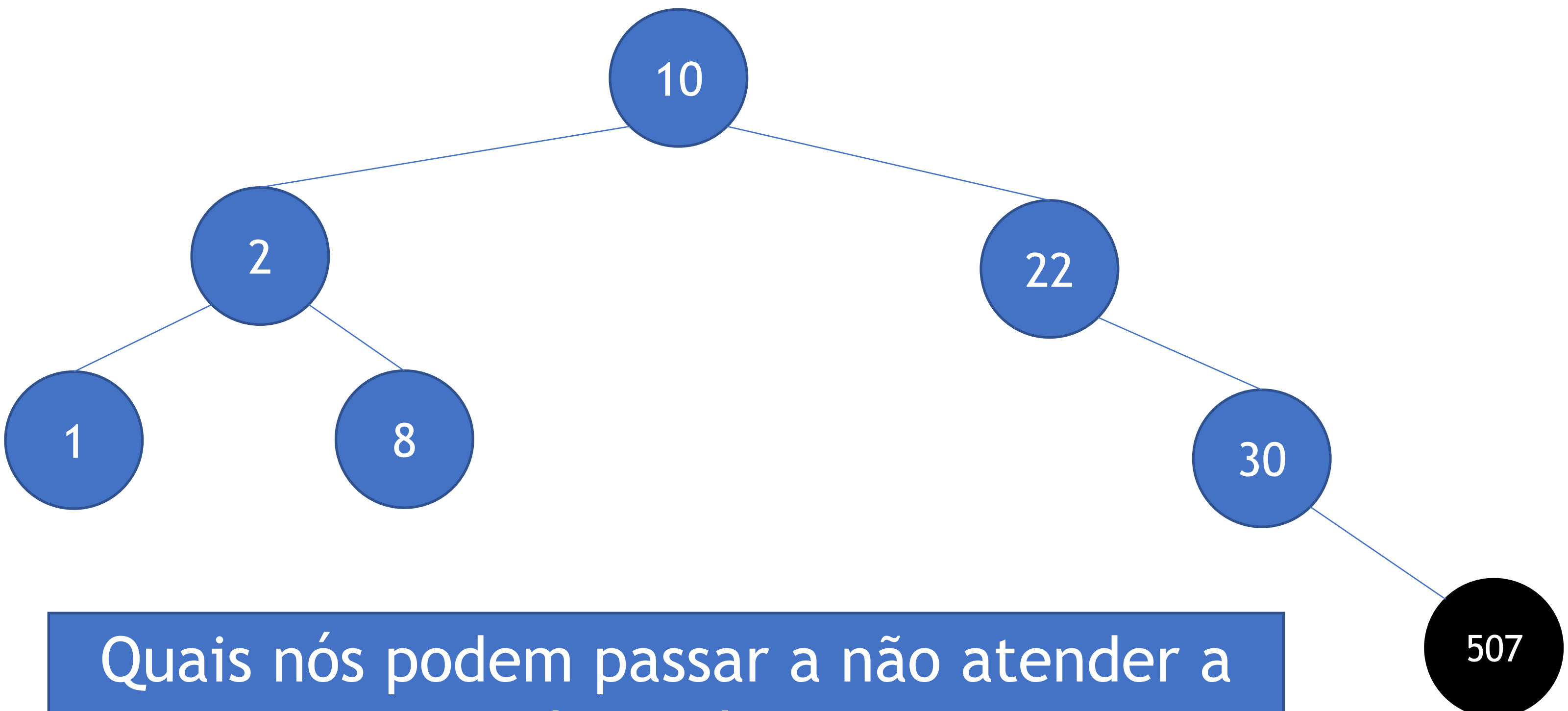
Exemplo: Inserção

- Inserir 507:



Exemplo: Inserção

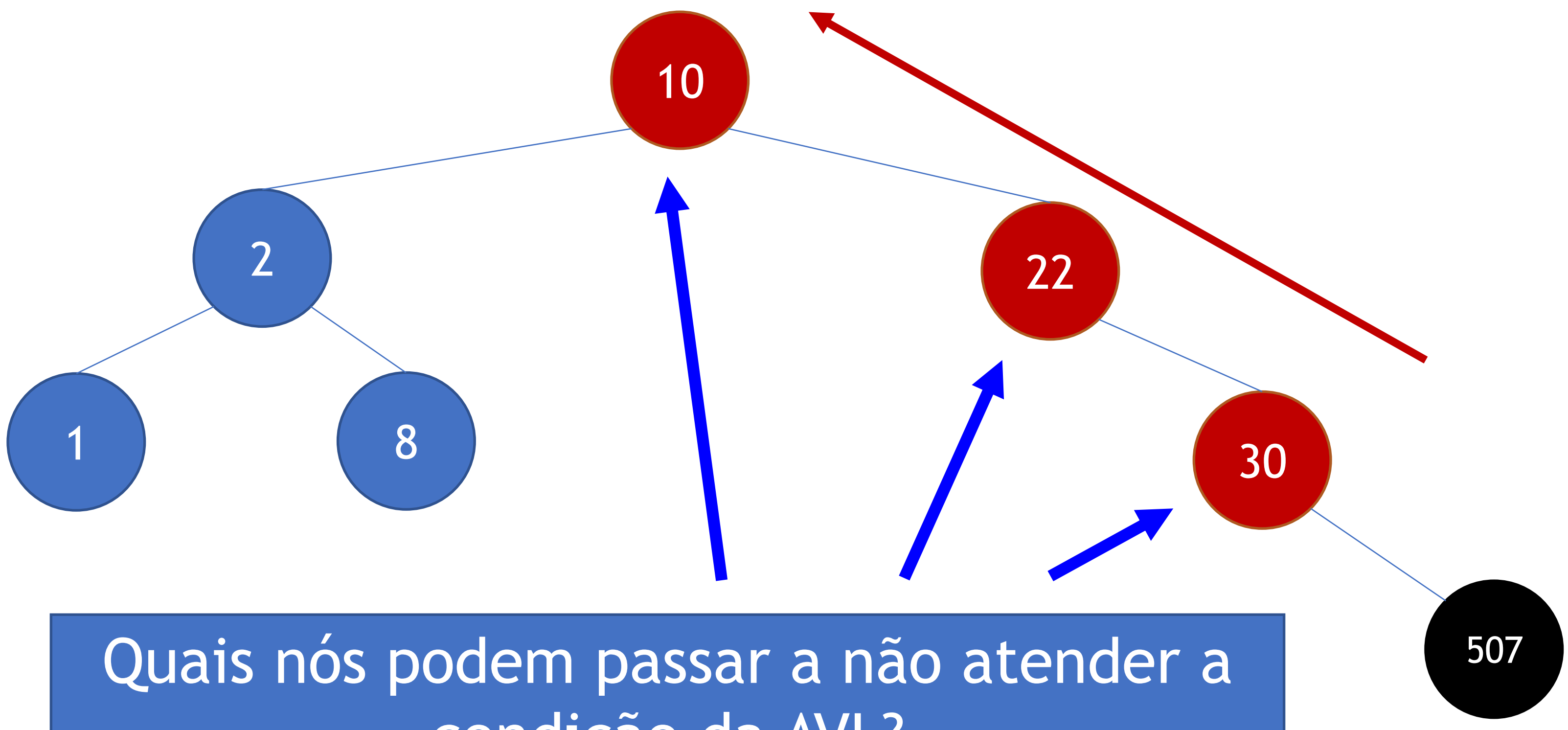
- Inserir 507:



Quais nós podem passar a não atender a condição da AVL?

Exemplo: Inserção

- Inserir 507:

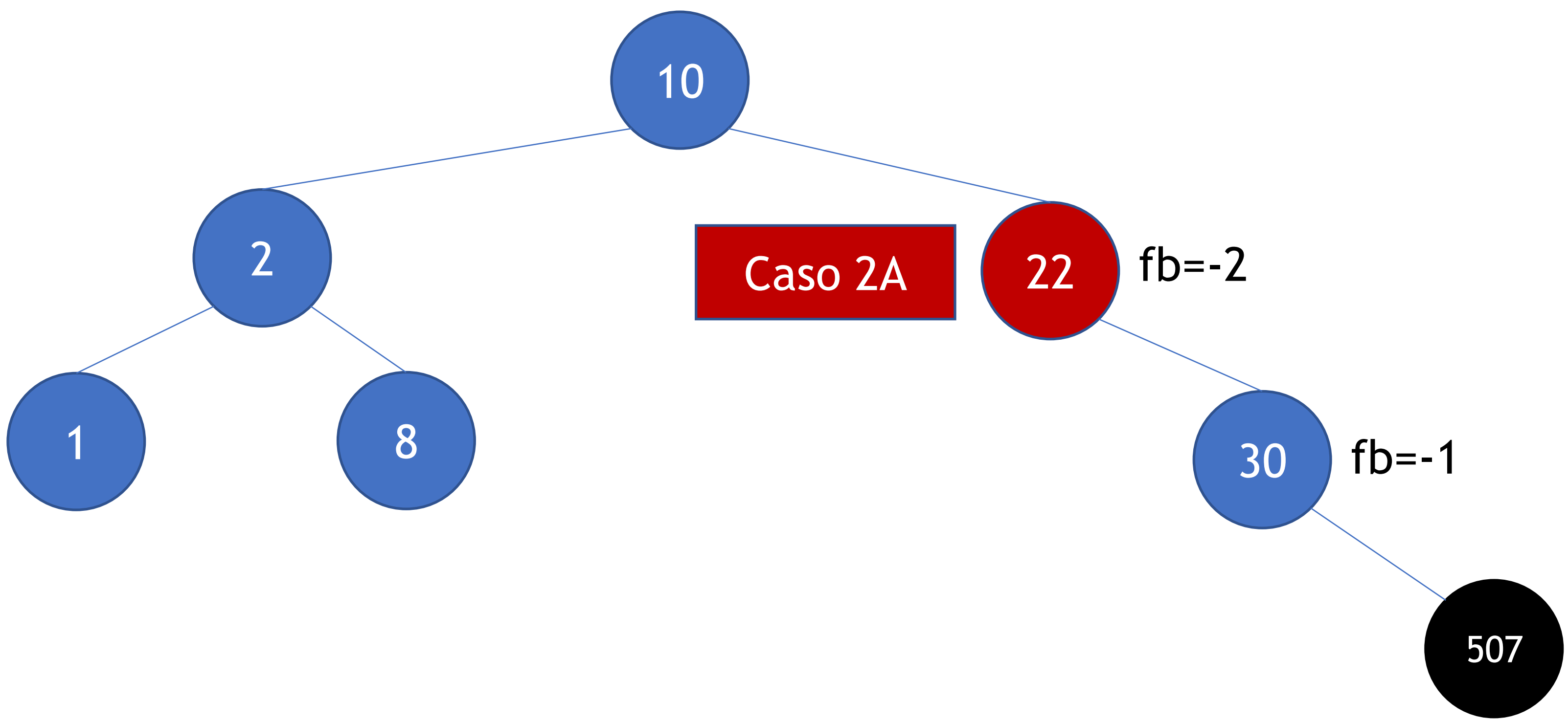


Quais nós podem passar a não atender a condição da AVL?

Quem pode ter o **fb** alterado, ou seja, os nós no caminho do pai até a raiz.

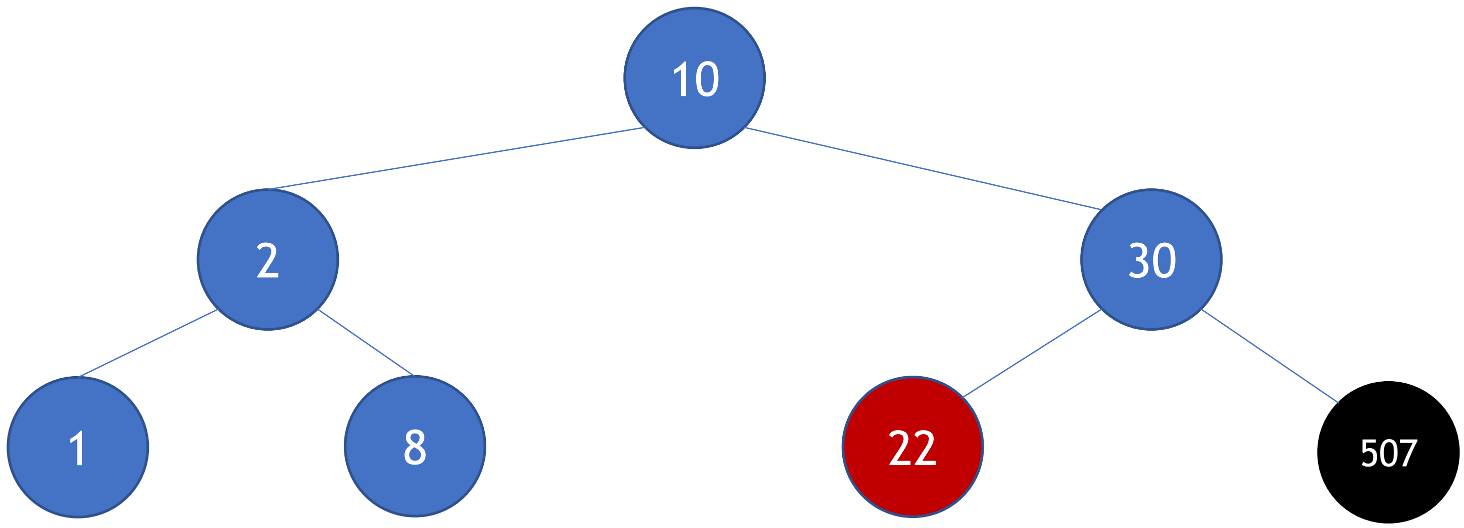
Exemplo: Inserção

- Inserir 507:



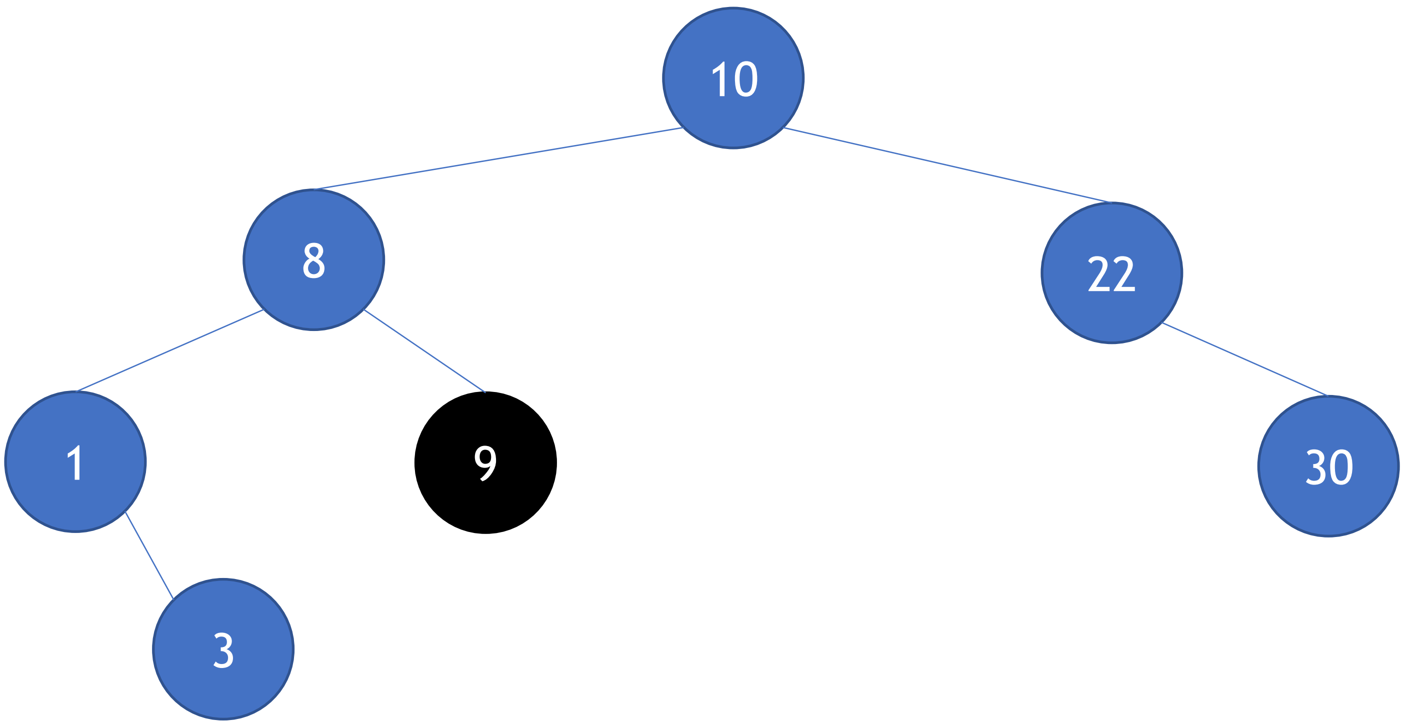
Exemplo: Inserção

- Inserir 507:



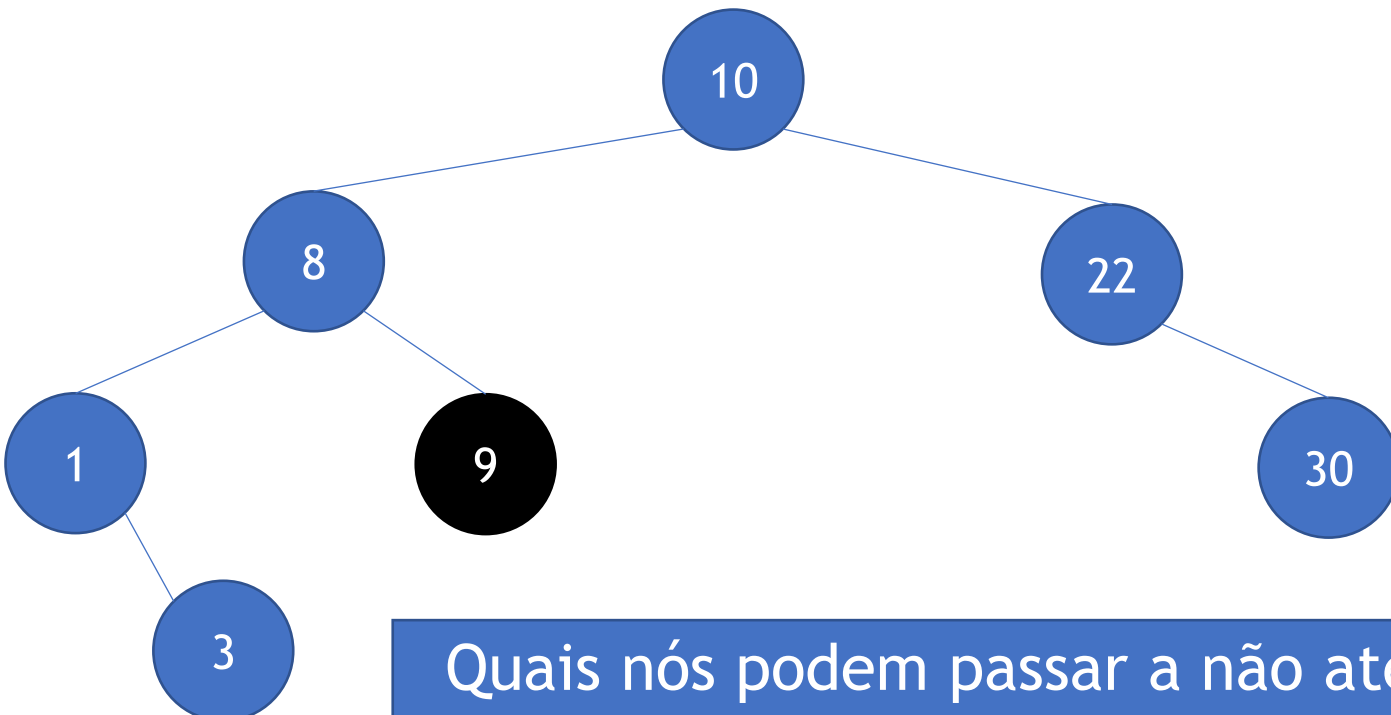
Exemplo: Remoção

- Remover 9:



Exemplo: Remoção

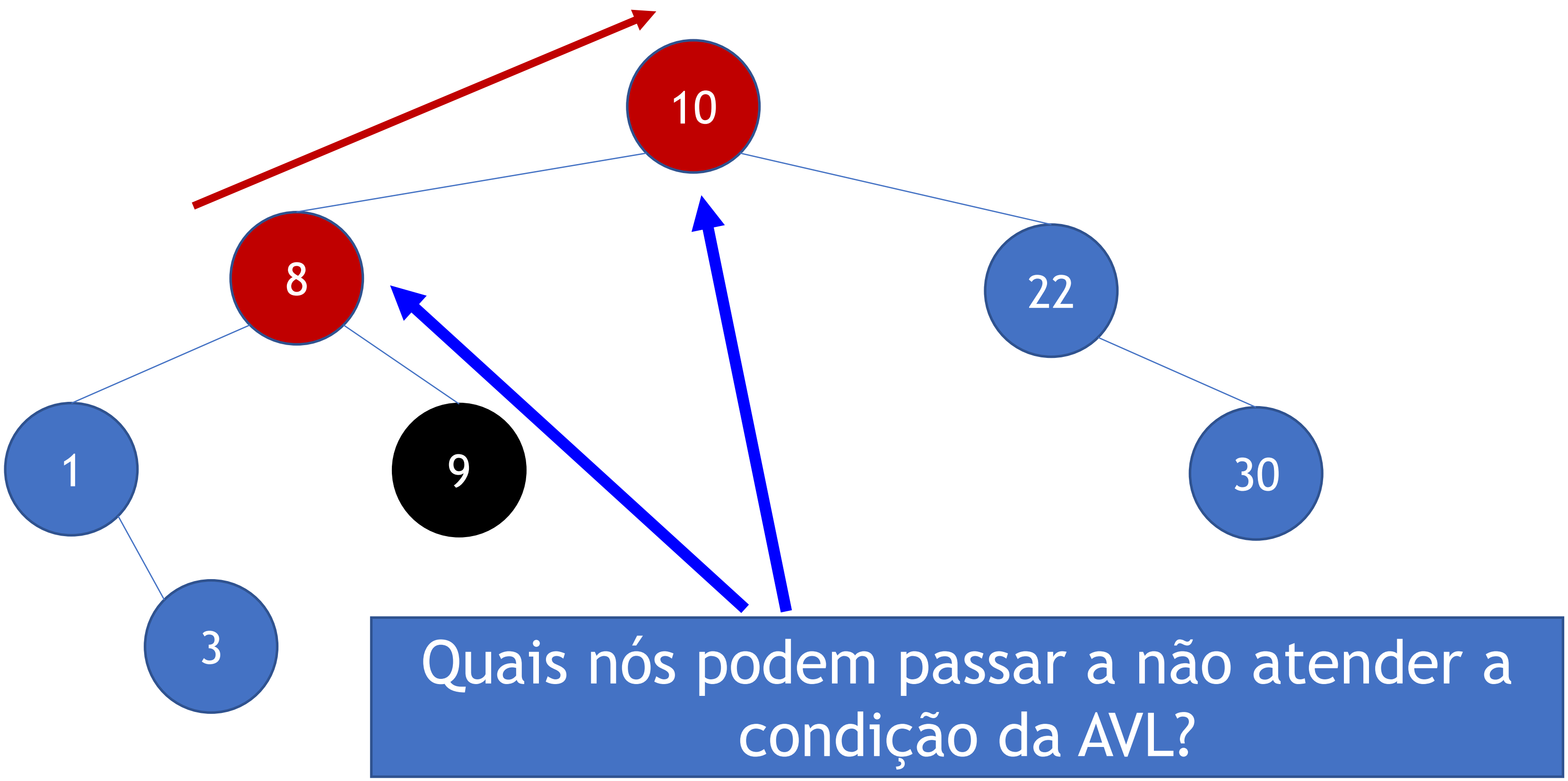
- Remover 9:



Quais nós podem passar a não atender a condição da AVL?

Exemplo: Remoção

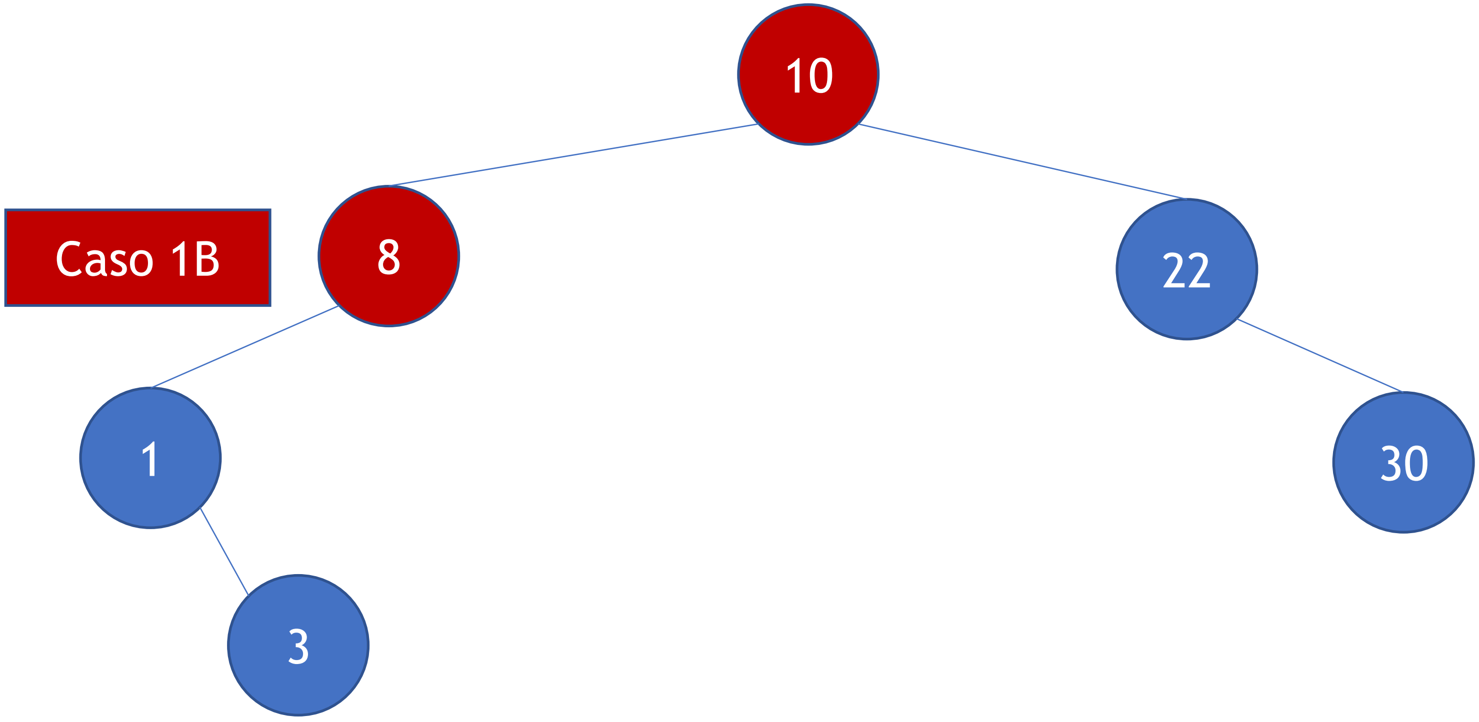
- Remover 9:



Quem pode ter o **fb** alterado, ou seja, os nós no caminho do pai até a raiz.

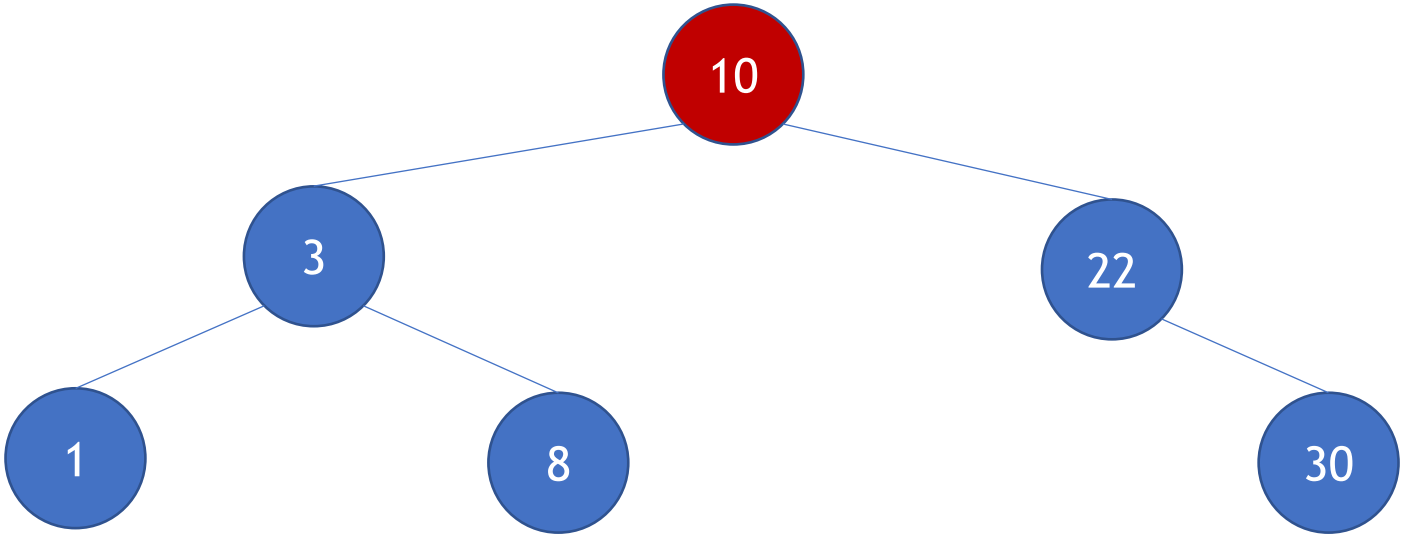
Exemplo: Remoção

- Remover 9:



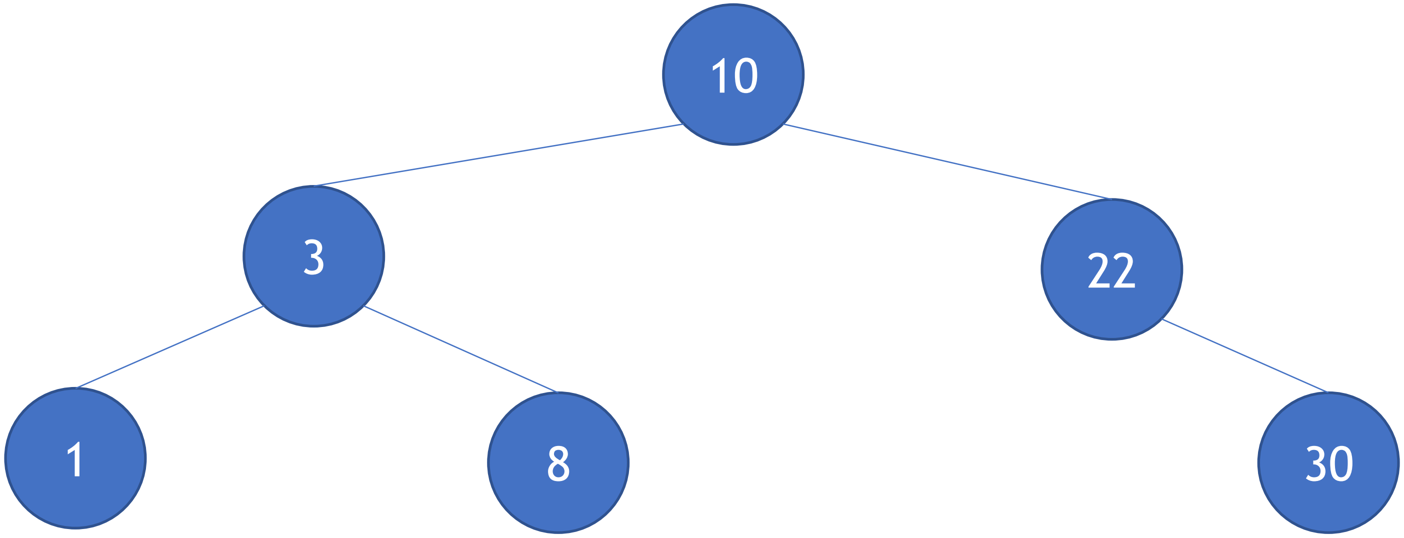
Exemplo: Remoção

- Remover 9:



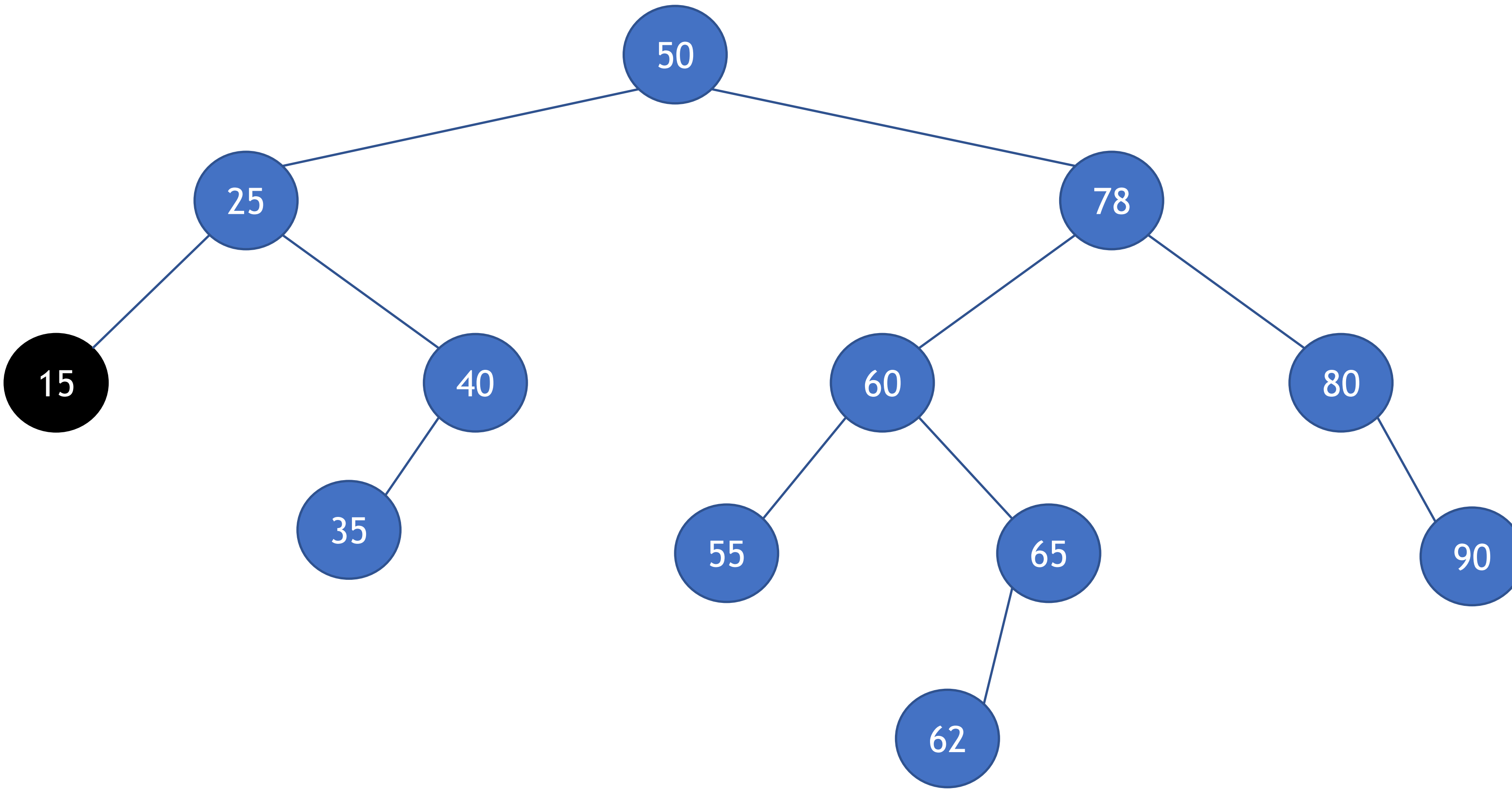
Exemplo: Remoção

- Remover 9:



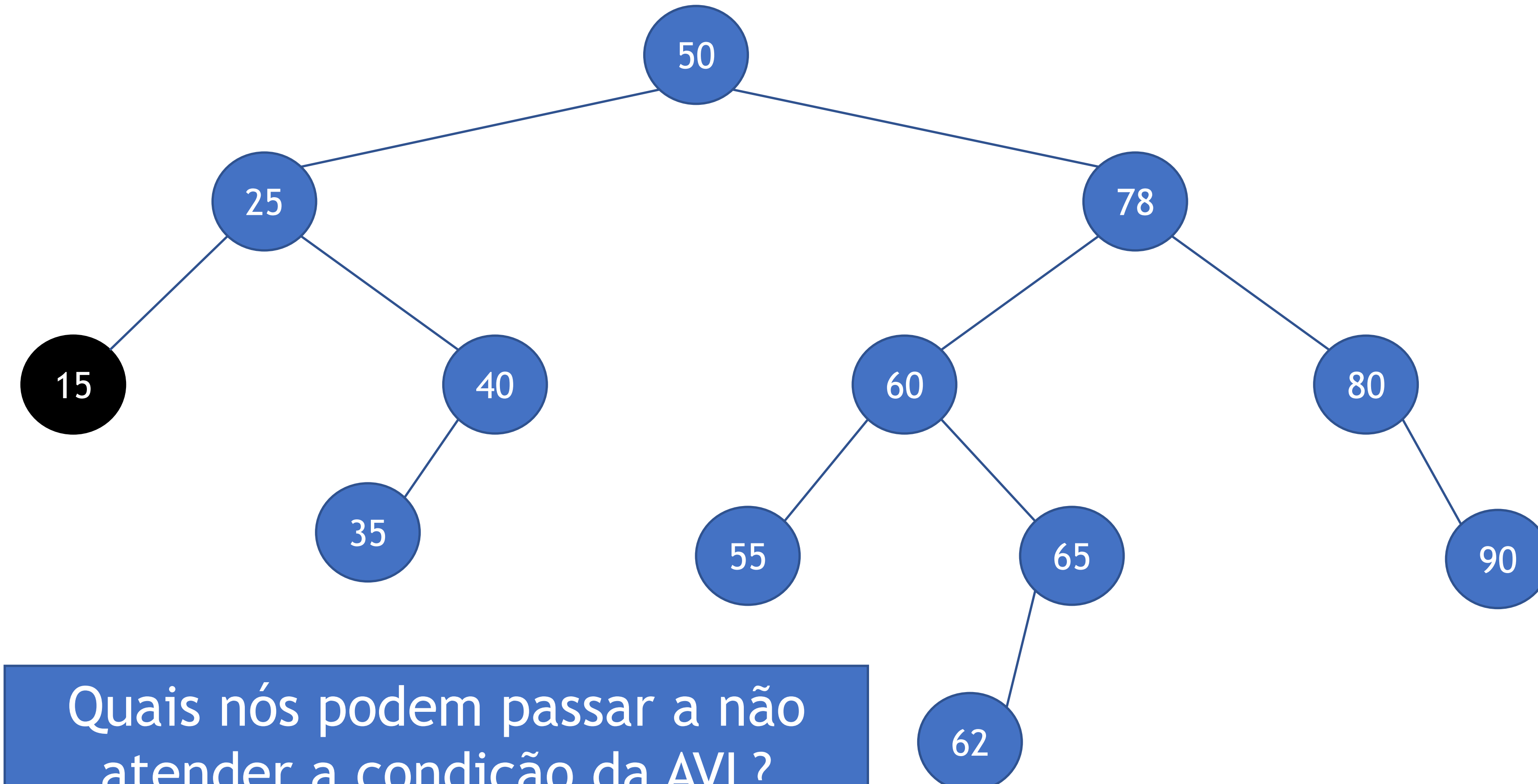
Outro exemplo: Remoção

- Remover 15:



Outro exemplo: Remoção

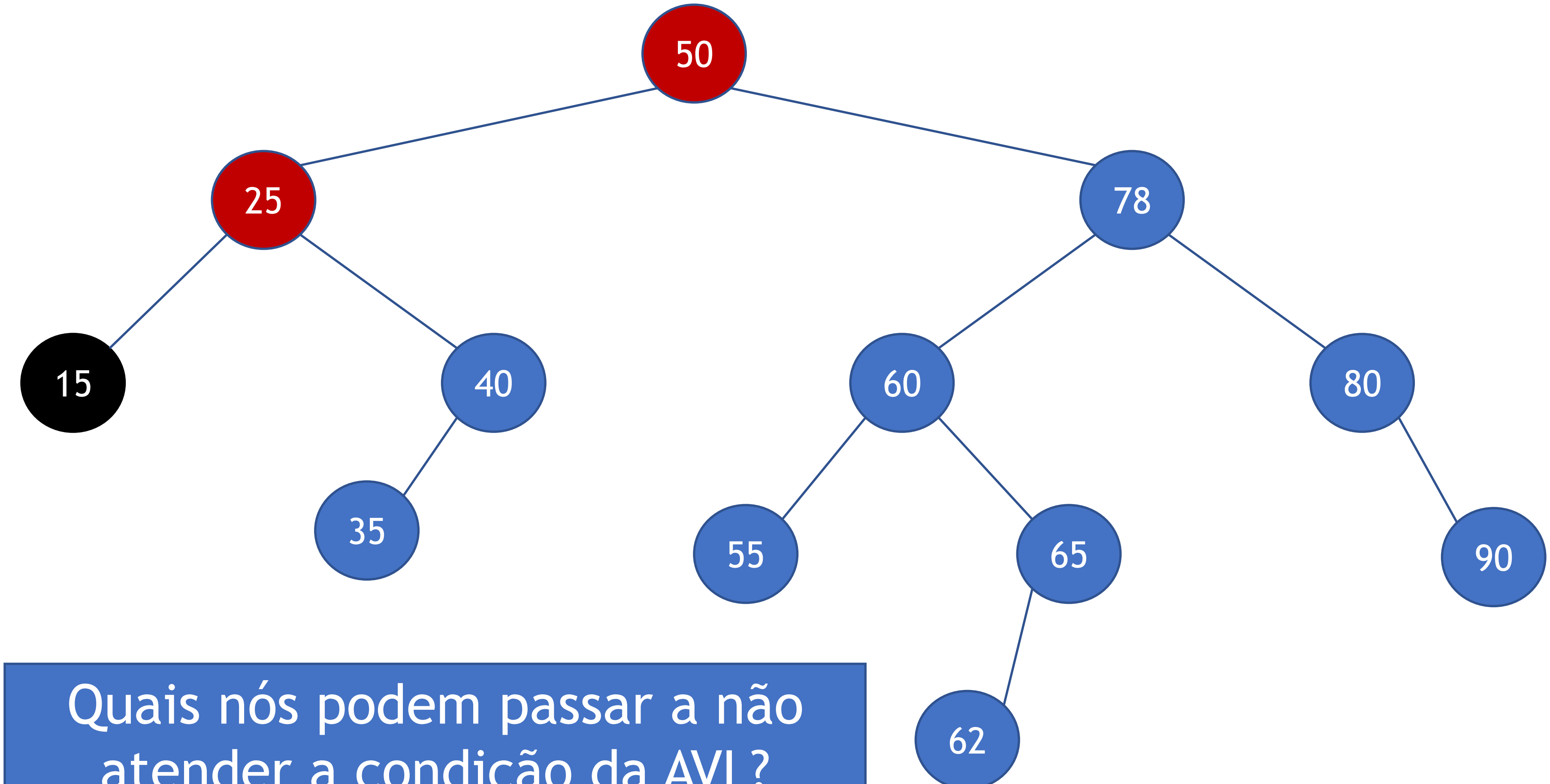
- Remover 15:



Quais nós podem passar a não atender a condição da AVL?

Outro exemplo: Remoção

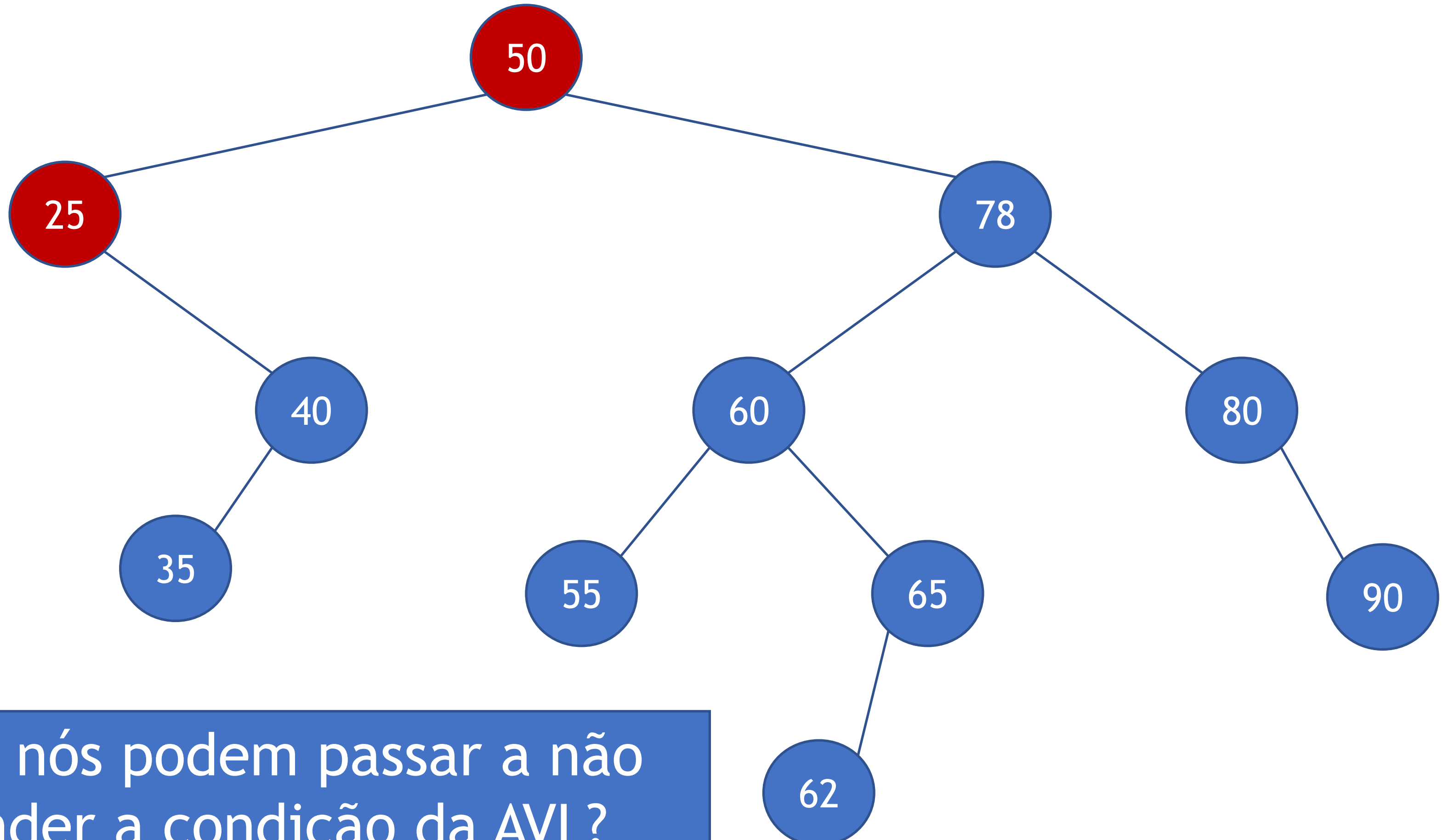
- Remover 15:



Quais nós podem passar a não atender a condição da AVL?

Outro exemplo: Remoção

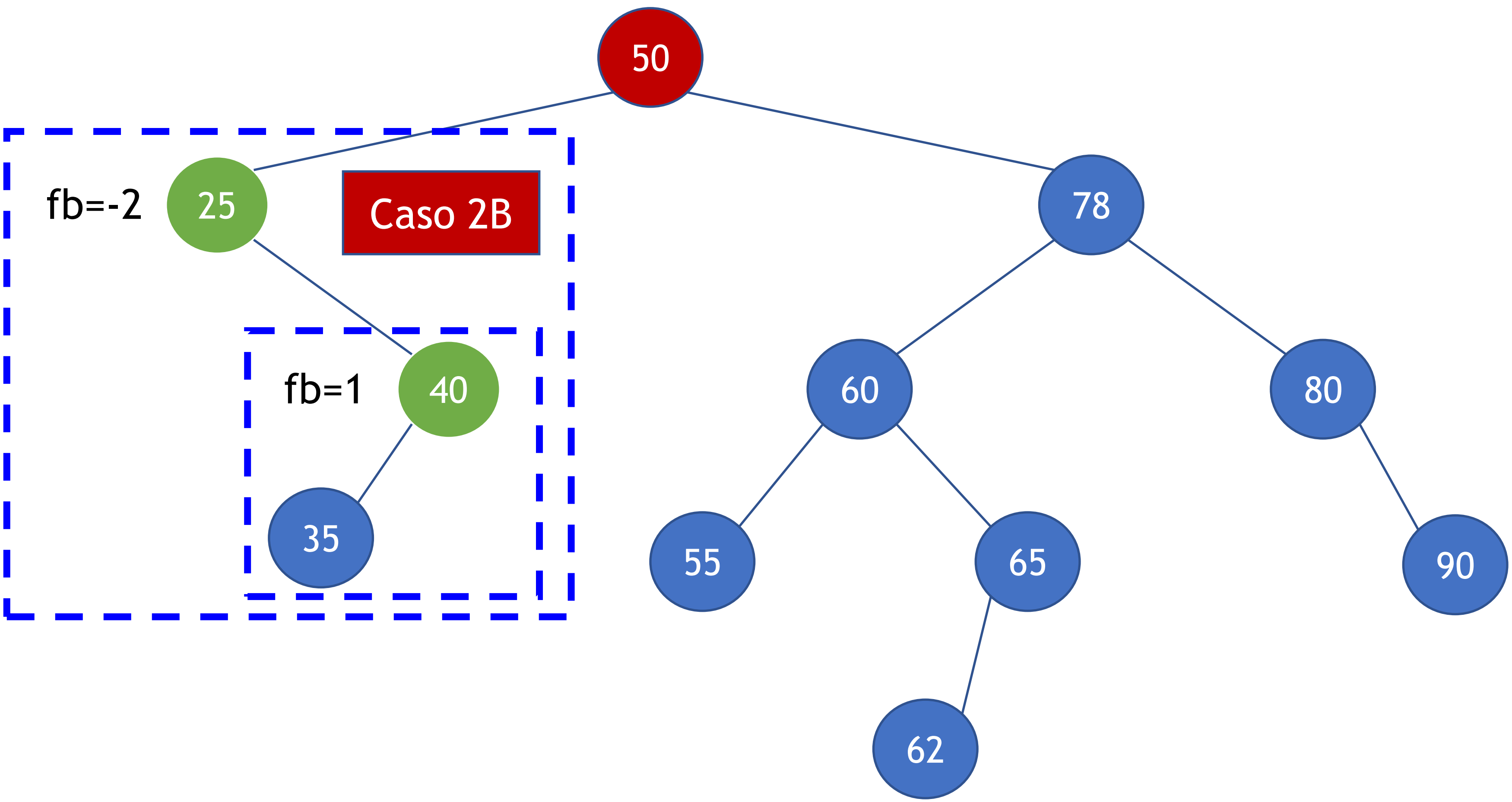
- Remover 15:



Quais nós podem passar a não atender a condição da AVL?

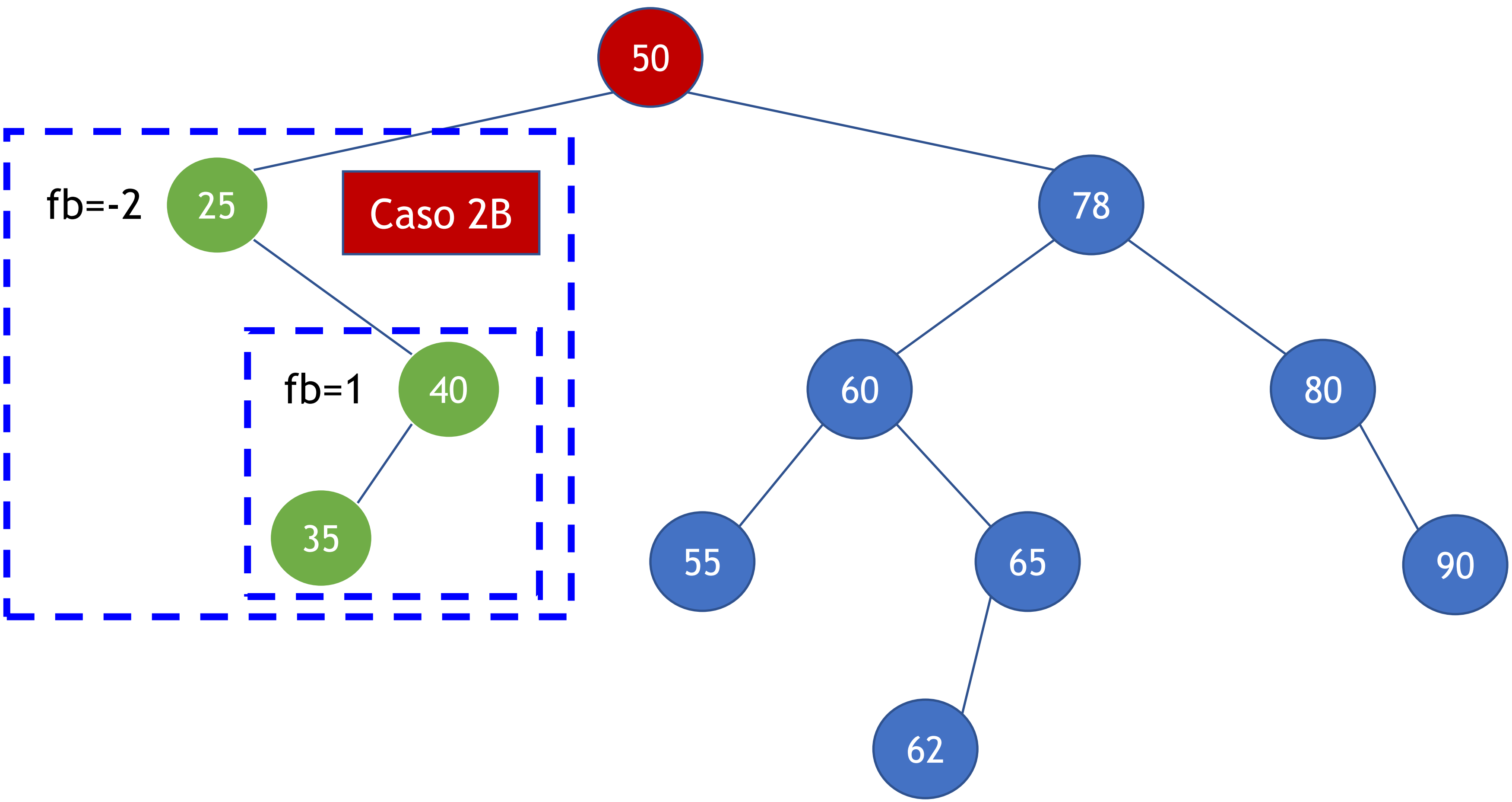
Outro exemplo: Remoção

- Remover 15:



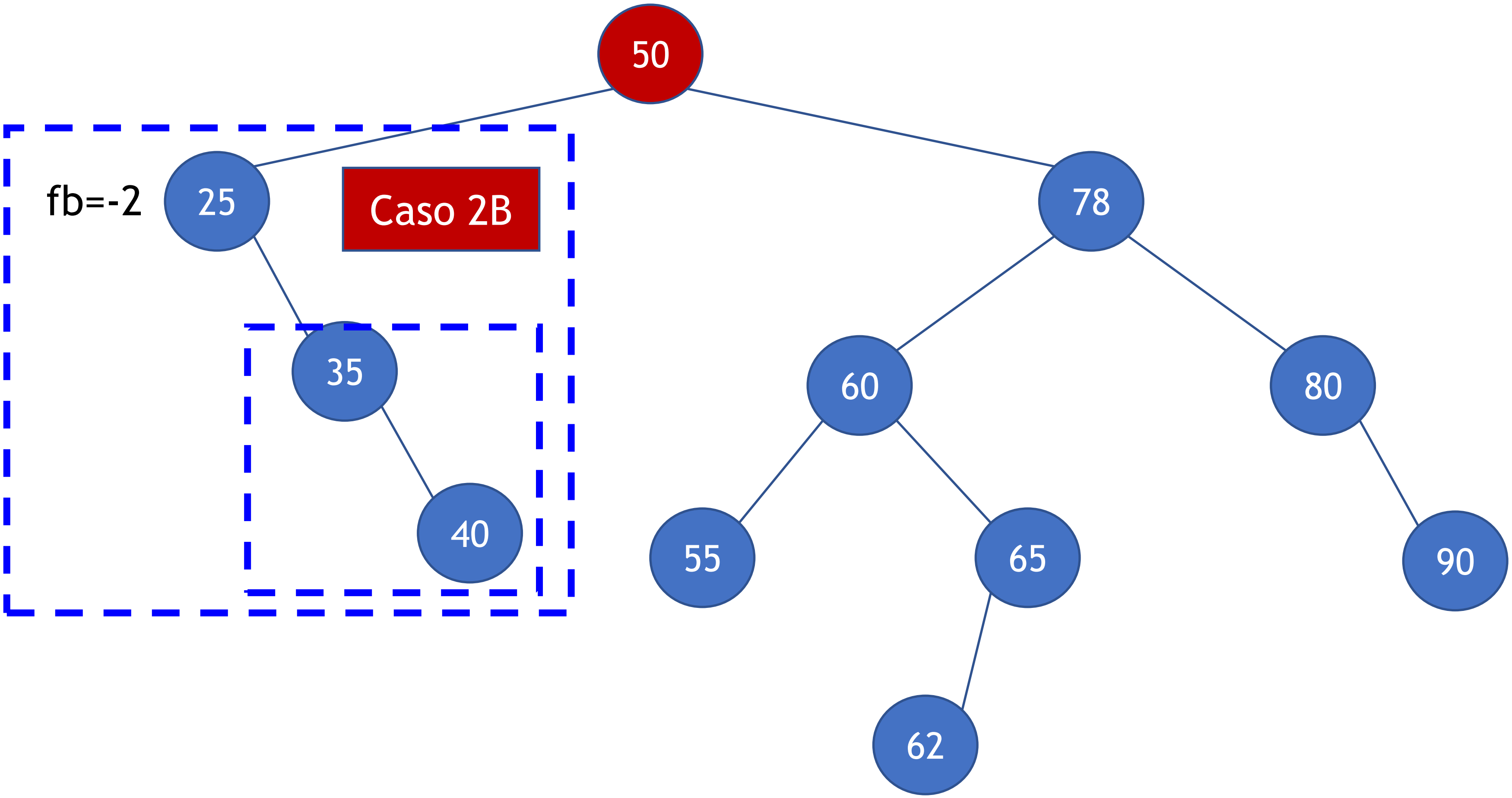
Outro exemplo: Remoção

- Remover 15:



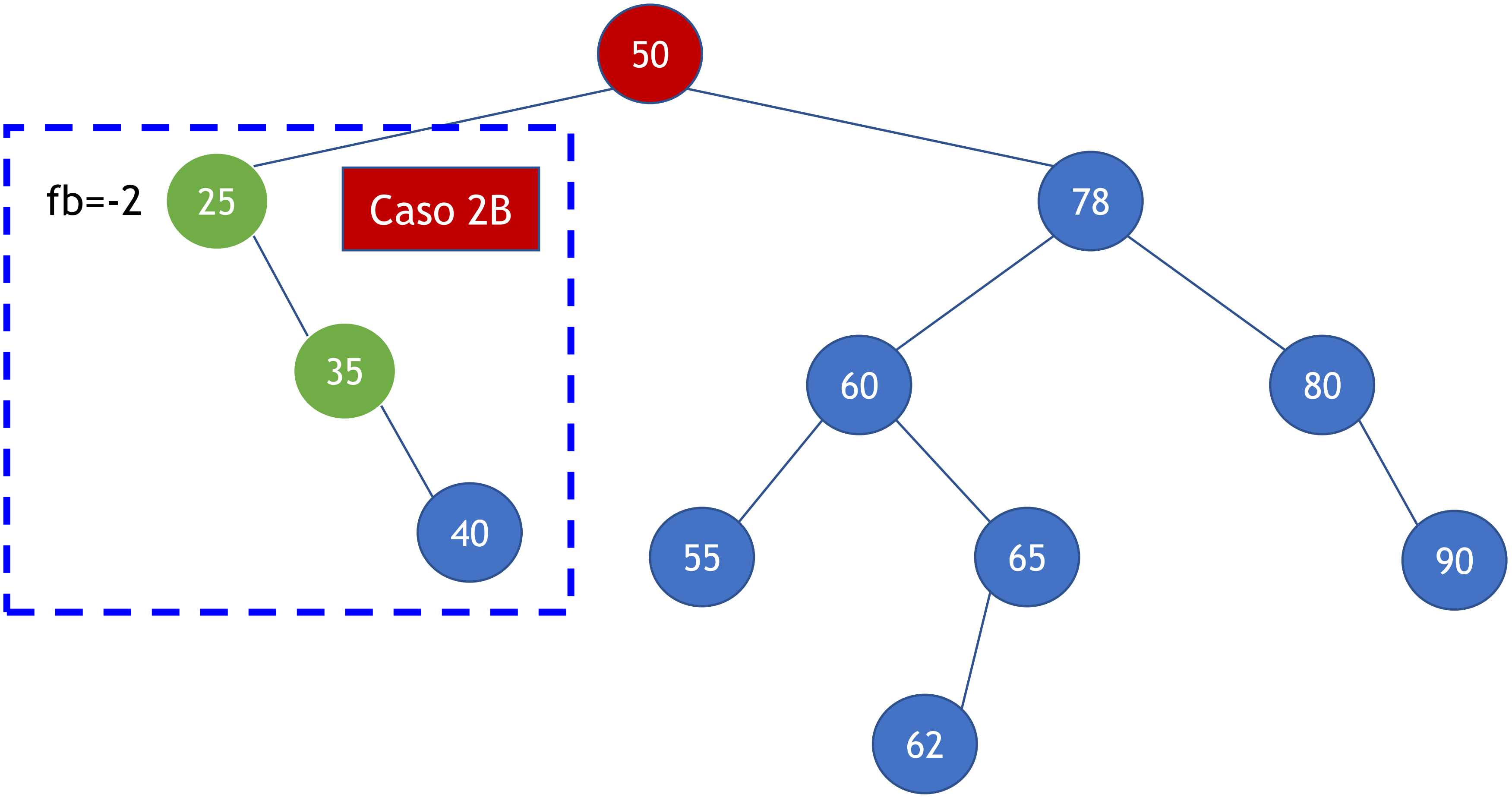
Outro exemplo: Remoção

- Remover 15:



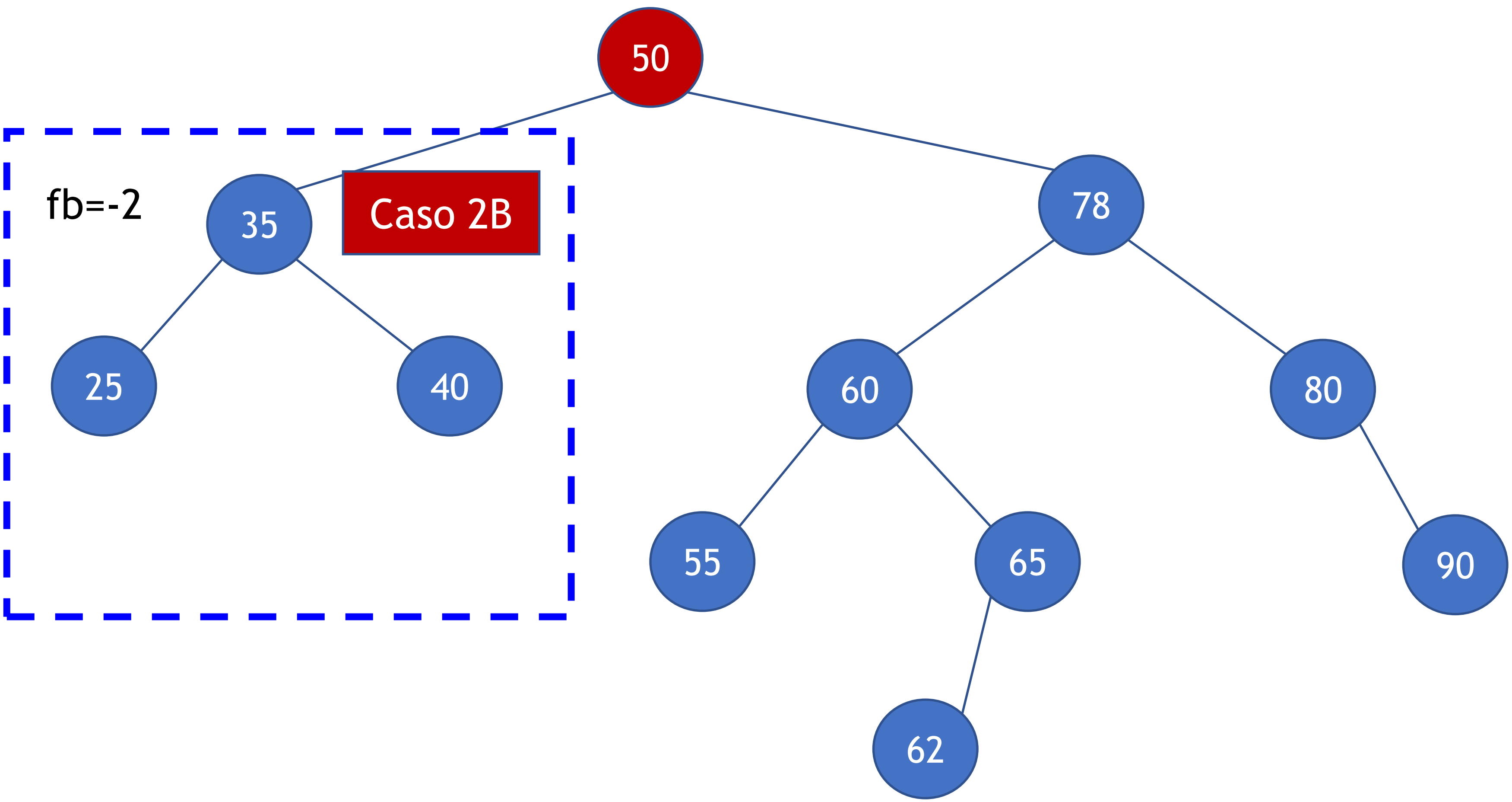
Outro exemplo: Remoção

- Remover 15:



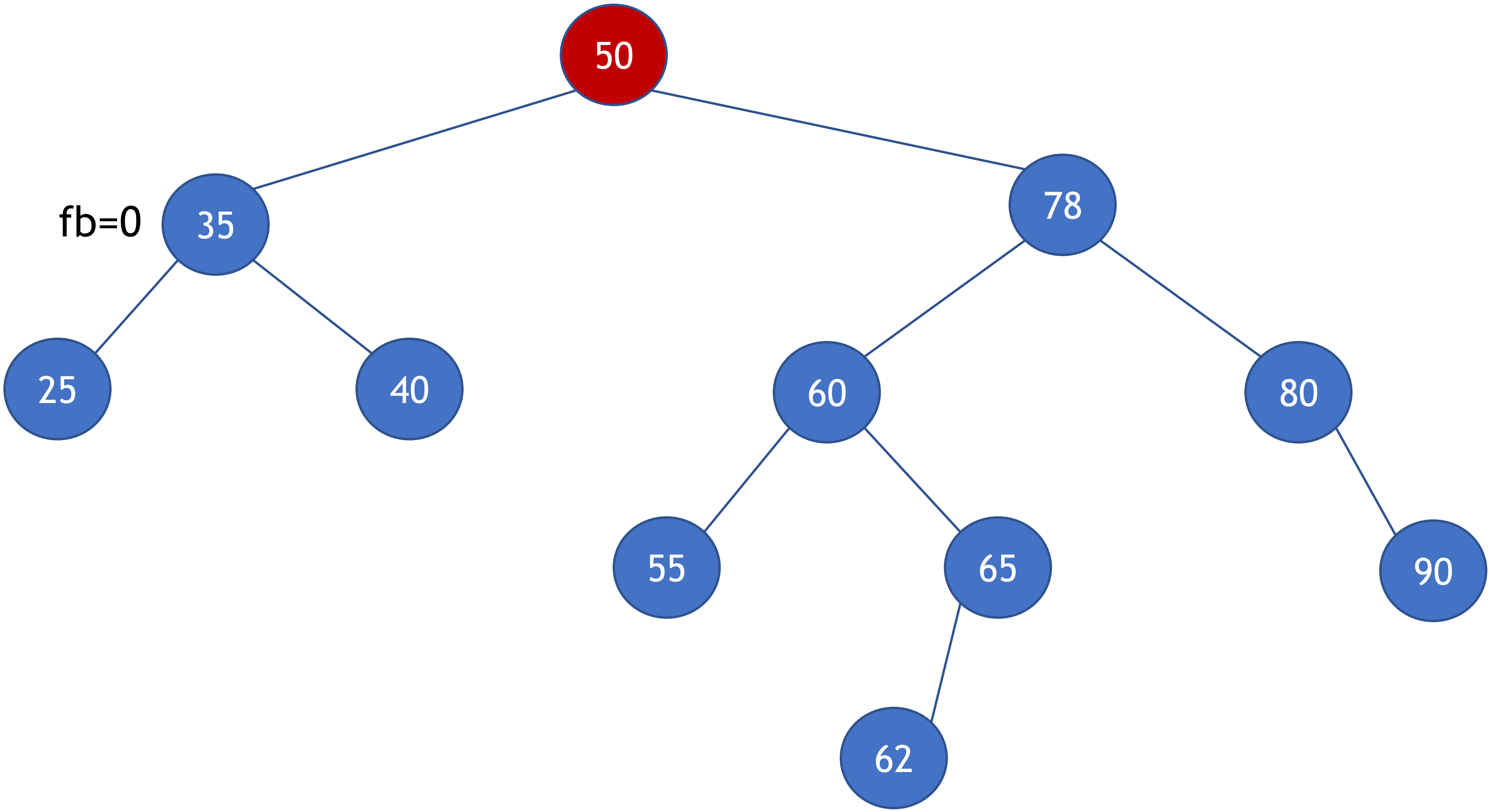
Outro exemplo: Remoção

- Remover 15:



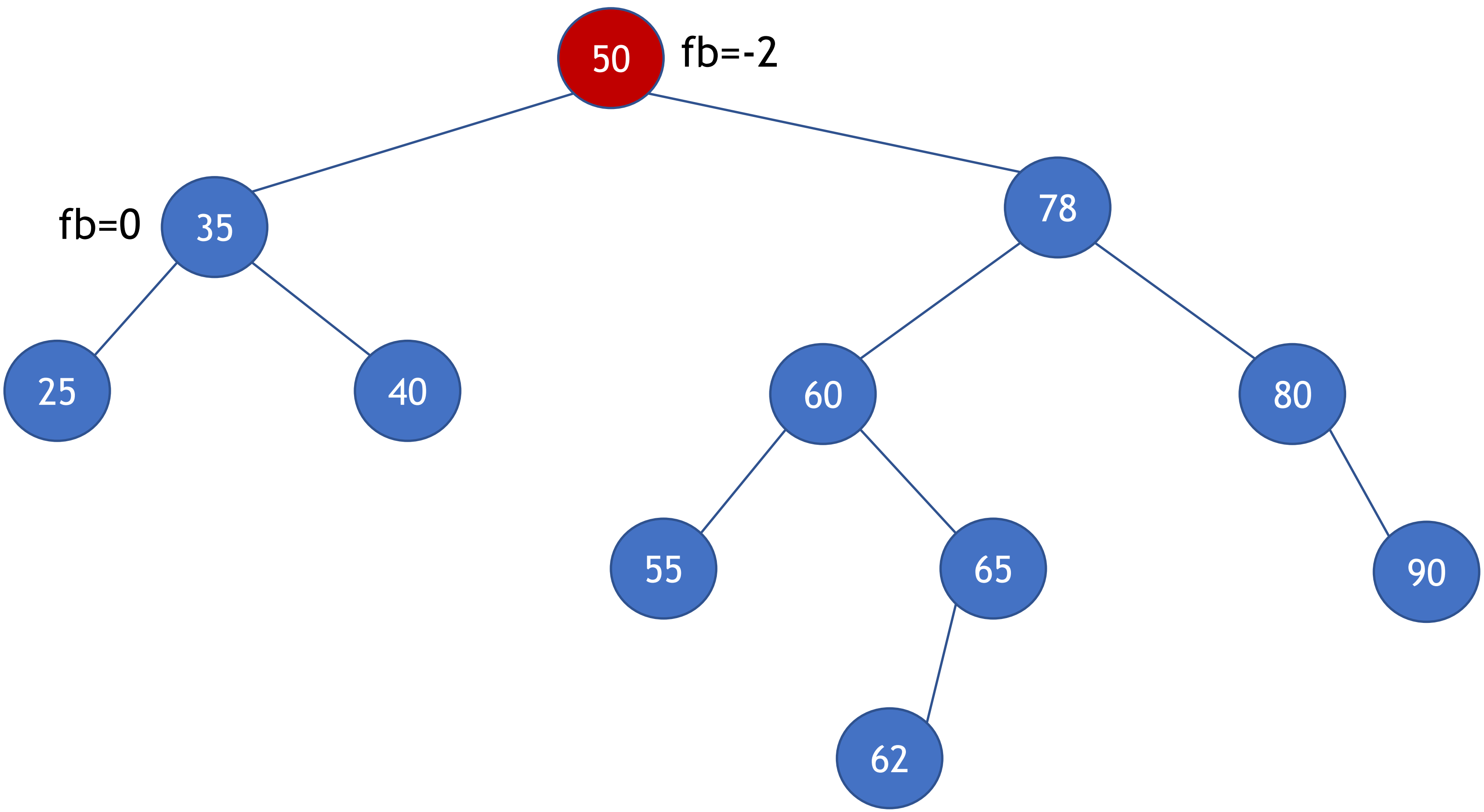
Outro exemplo: Remoção

- Remover 15:



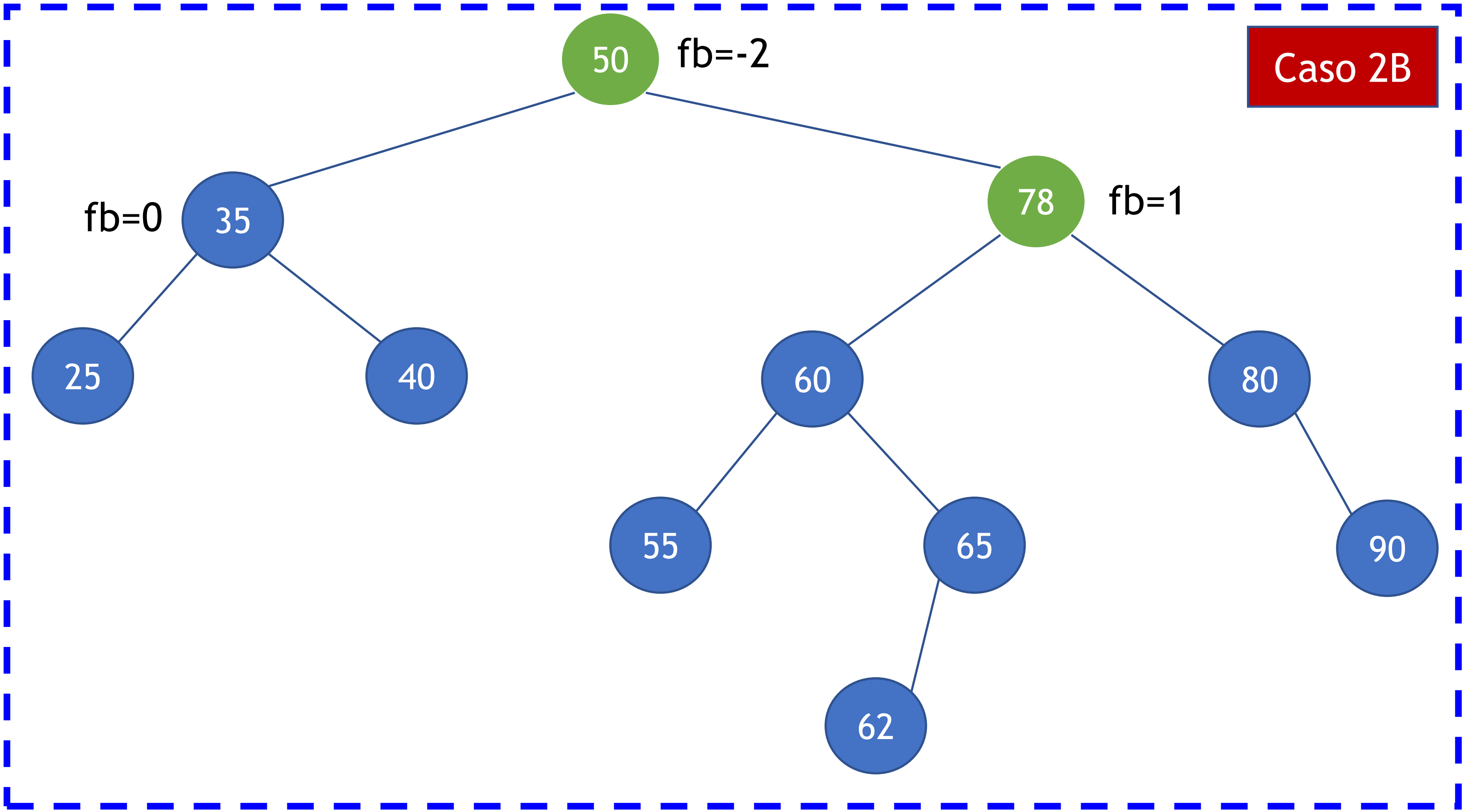
Outro exemplo: Remoção

- Remover 15:



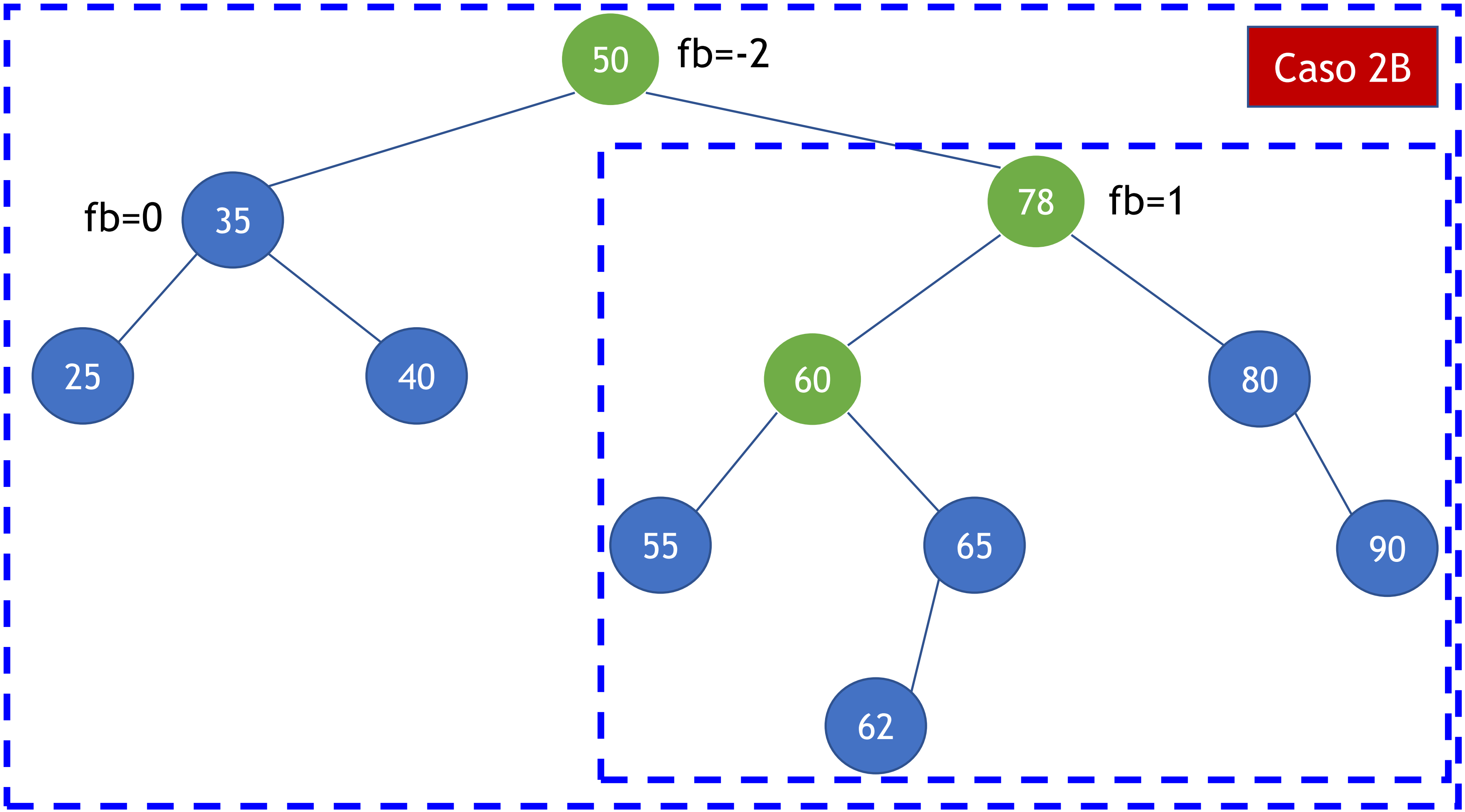
Outro exemplo: Remoção

- Remover 15:



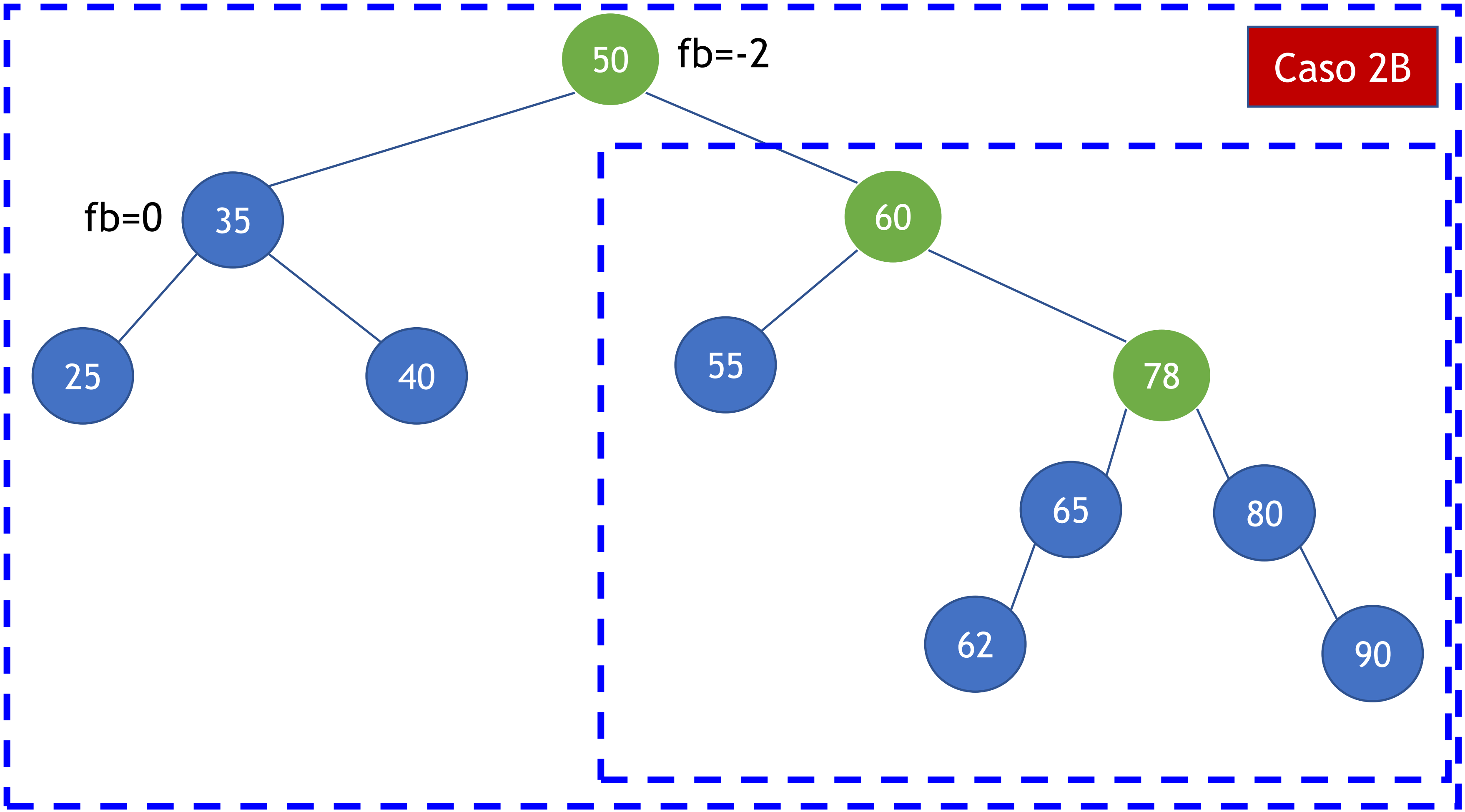
Outro exemplo: Remoção

- Remover 15:



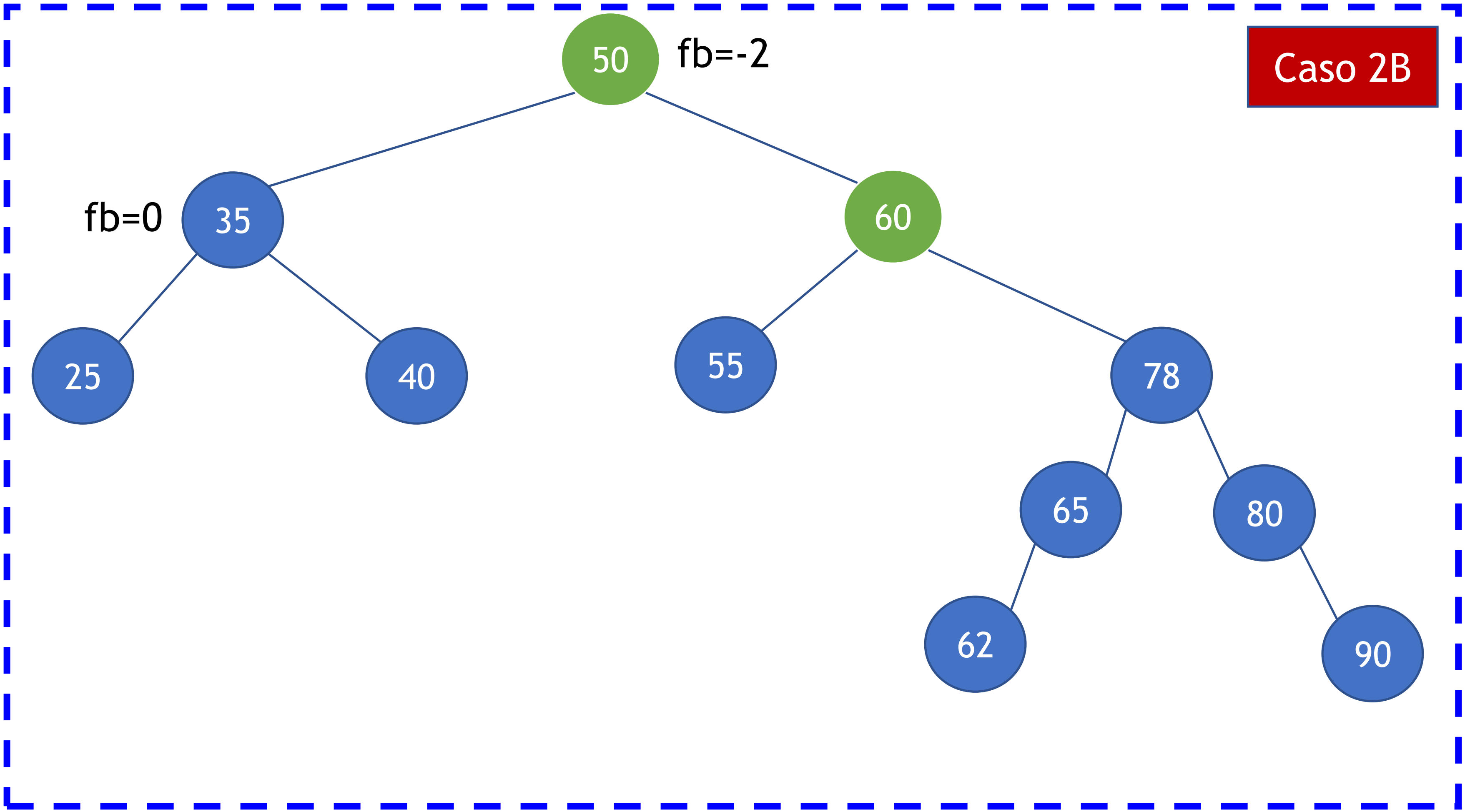
Outro exemplo: Remoção

- Remover 15:



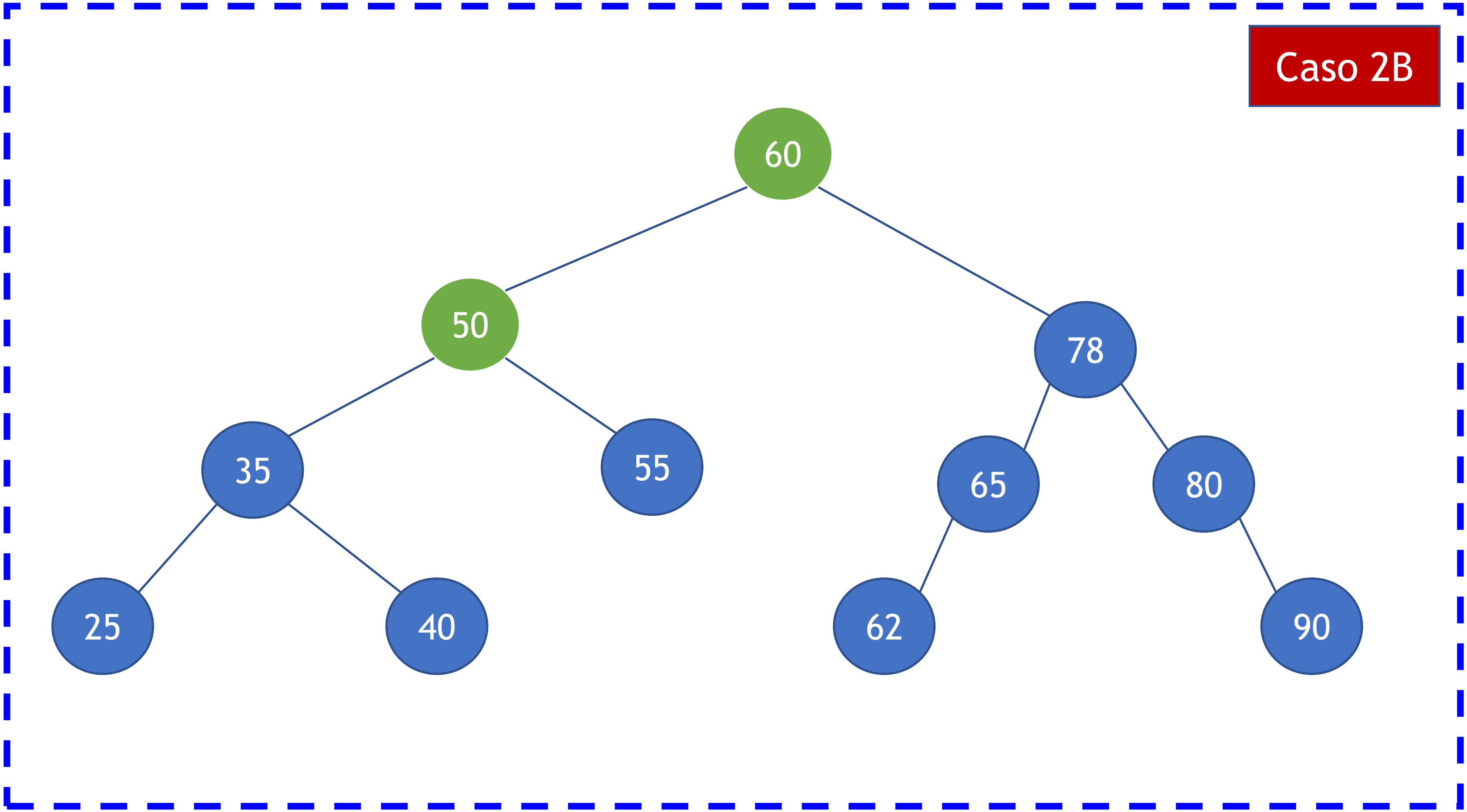
Outro exemplo: Remoção

- Remover 15:



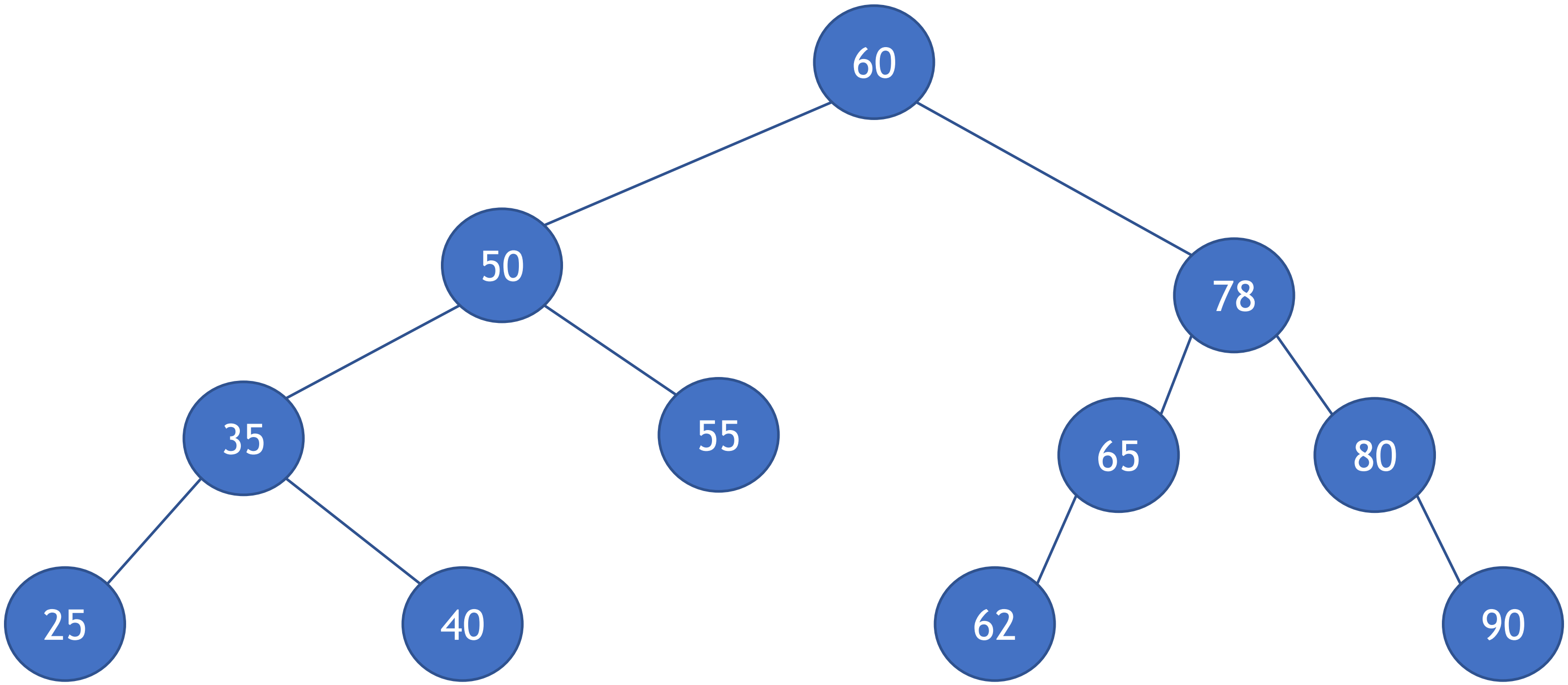
Outro exemplo: Remoção

- Remover 15:



Outro exemplo: Remoção

- Remover 15:



Este exemplo, mostra que pode ser necessário realizar mais de uma rotação. Conforme visto, aplicamos duas vezes o Caso 2B.

Operações: Inserção e Remoção

- Implementação em C

Chamada:
```
raiz = inserir(raiz, chave);
```

(código no vídeo)

Chamada:
```
raiz = remover(raiz, chave);
```

(código no vídeo)

Complexidade das operações

Complexidade das operações

- Como vimos na aula anterior, temos o seguinte custo para diversas operações em ABB:

Operação	Árvores
Busca	O(h)
Inserção	O(h)
Remoção	O(h)

- A AVL tem h = O(lg(n)). Portanto, o custo dessas três operações é O(lg(n)) em uma AVL.

Complexidade das operações

- Sobre as rotações:
  - Cada rotação tem custo constante (não depende do tamanho da árvore);
  - As rotações podem ser executadas apenas no caminho do pai até a raiz. Esse caminho nunca será maior que a altura, que é O(lg(n));
  - Ou seja, o custo das três operações permanece O(lg(n)) em uma AVL.

Referências

- Slides do Prof. Monael Pinheiro Riberio:
    - https://sites.google.com/site/aed2019q1/
- Slides da Profa. Mirtha Lina Fernández Venero
    - Algoritmos e Estruturas de Dados I - 2019
- Slides do Prof. Jesús P. Mena-Chalco:
    - http://professor.ufabc.edu.br/~jesus.mena/courses/aed1-1q-2019/

Bibliografia básica

- PINHEIRO, F. A. C. Elementos de programação em C. Porto Alegre, RS: Bookman, 2012.
- FORBELLONE, A. L. V.; EBERSPACHER, H. F. Lógica de programação: a construção de algoritmos e estruturas de dados. 3ª edição. São Paulo, SP: Prentice Hall, 2005.
- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: teoria e prática. 2ª edição. Rio de Janeiro, RJ: Campus, 2002.

Bibliografia complementar

- AGUILAR, L. J. Programação em C++: algoritmos, estruturas de dados e objetos. São Paulo, SP: McGraw-Hill, 2008.
- DROZDEK, A. Estrutura de dados e algoritmos em C++. São Paulo, SP: Cengage Learning, 2009.
- KNUTH D. E. The art of computer programming. Upper Saddle River, USA: Addison- Wesley, 2005.
- SEDGEWICK, R. Algorithms in C++: parts 1-4: fundamentals, data structures, sorting, searching. Reading, USA: Addison-Wesley, 1998.
- SZWARCFITER, J. L.; MARKENZON, L. Estruturas de dados e seus algoritmos. 3a edição. Rio de Janeiro, RJ: LTC, 1994.
- TEWNENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. Estruturas de dados usando C. São Paulo, SP: Pearson Makron Books, 1995.