

Árvores 2-3 e 2-3-4

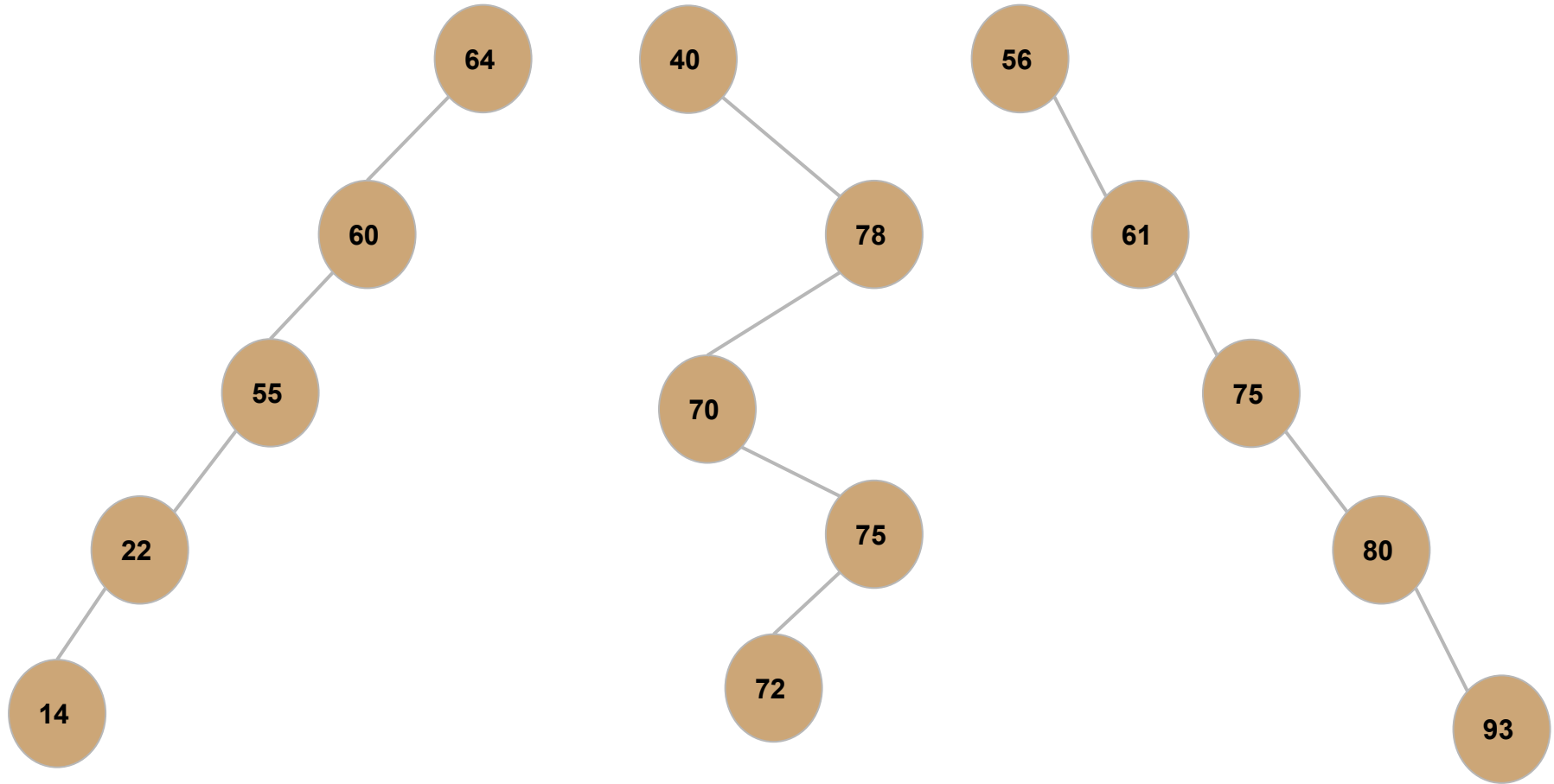
Prof. Joaquim Uchôa
Prof. Renato Ramos



Relembrando: 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.
- Após muitas inclusões e remoções de elementos a árvore resultante pode ser degenerada, ou seja, desequilibrada e, para garantir que isso não ocorra, é preciso que os nós da árvore sejam reorganizados.

Exemplos de árvores binárias degeneradas



Balanceamento em árvores binárias

O balanceamento em árvores binárias pode ser implementado de várias maneiras, sendo que as abordagens mais populares são:

- Árvores AVL
- Árvores rubro-negras
- Árvores 2-3 e 2-3-4

Árvores 2-3 e 2-3-4 são equivalentes em termos de eficiência a implementações de rubro-negras para operações de inserção, remoção e busca.

Árvores B - 1/2

Árvores 2-3 e 2-3-4 são tipos especiais de árvores B que são utilizados para uso em memória em substituição à árvores binárias.

Árvores B podem ser entendidas como uma generalização de árvores binárias, em que cada nó pode ter mais que dois filhos e mais que uma chave.

Árvores B foram propostas por Bayer e McCreight em 1972, sendo que Bayer também propôs a árvore que se tornaria depois a rubro-negra.

Árvores B - 1/2

Árvores B são otimizadas para operações de leitura e escrita em disco, ao contrário dos outros modelos de árvores balanceadas.

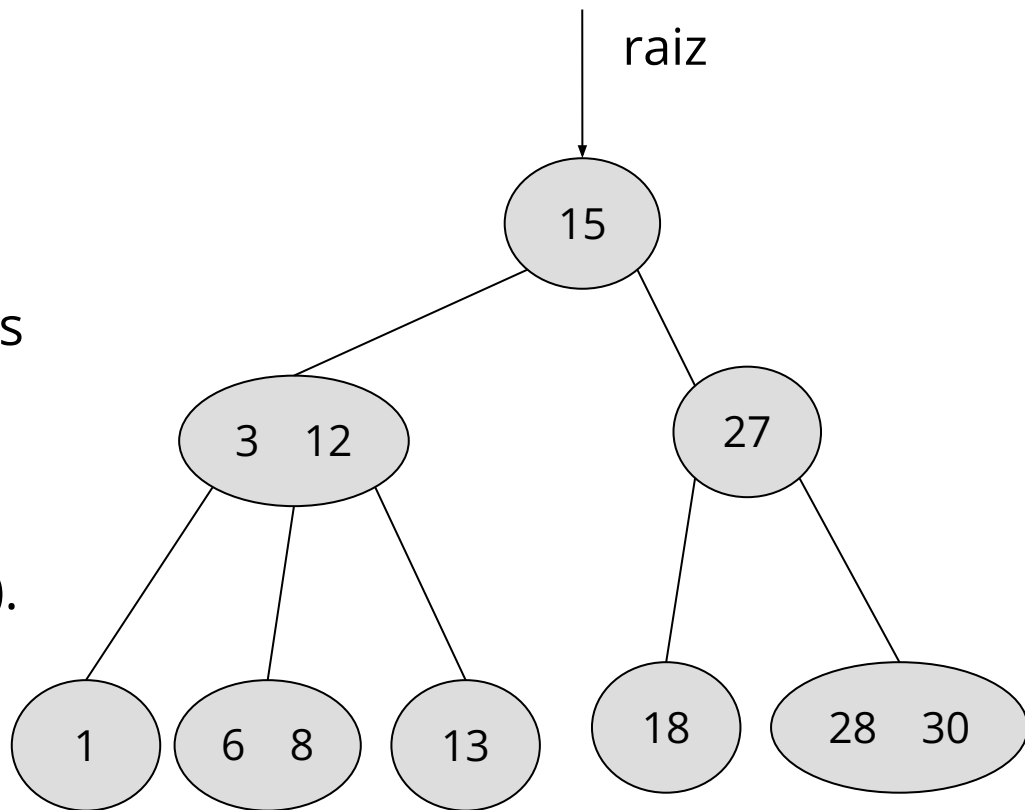
Árvores B e suas variantes (como B* e B+) são usadas para armazenamento de dados em sistemas de arquivos e bancos de dados diversos.

Apesar do uso extensivo de árvores B em disco, há também situações que justificam seu uso em memória, o que é o caso das árvores 2-3 e 2-3-4.

Árvore 2-3

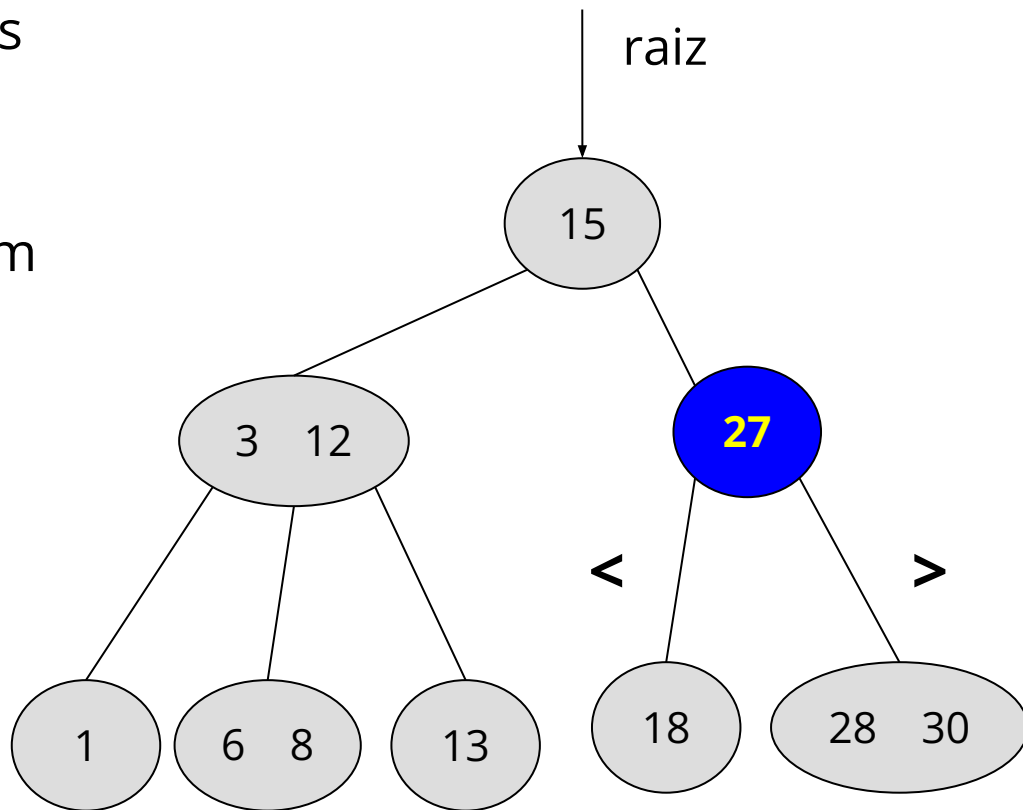
Em uma árvore 2-3, cada nó interno pode ter dois ou três filhos. Nesse caso, cada nó interno armazena, respectivamente, uma ou duas chaves.

Todos os nós folhas estão no mesmo nível (balanceamento).



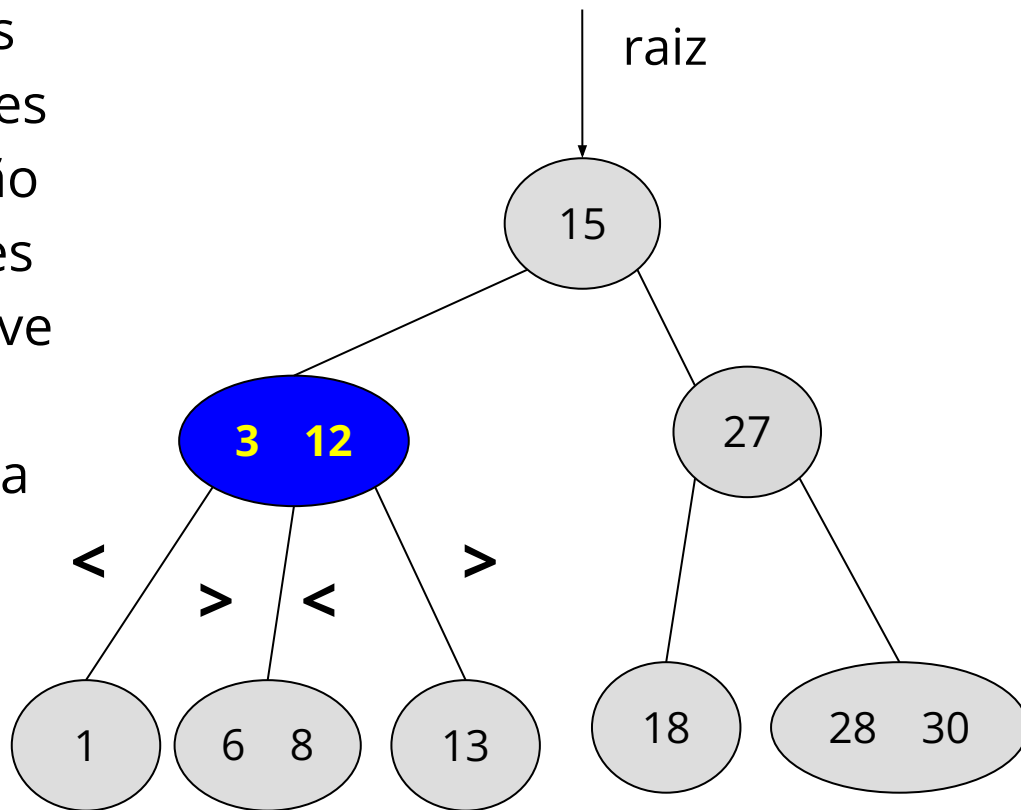
Árvore 2-3 - Propriedades 1/3

Caso um nó interno tenha dois filhos, as chaves menores que ele estão à sua esquerda e os maiores à sua direita, como em uma árvore binária normal.



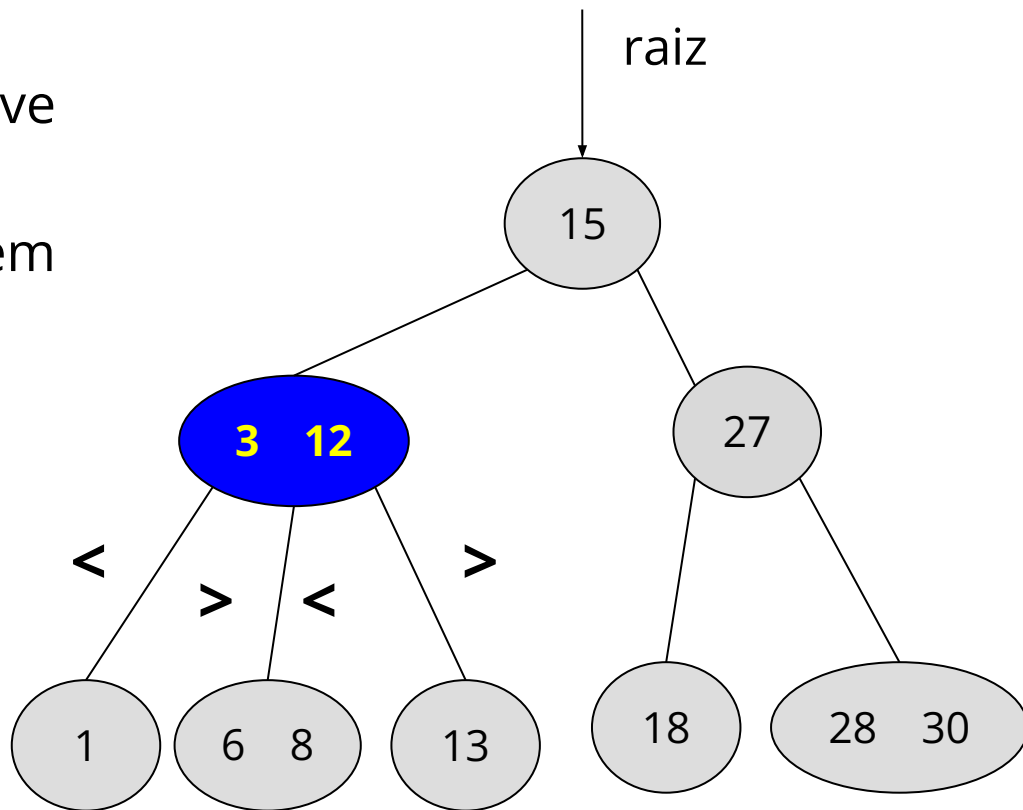
Árvore 2-3 - Propriedades 2/3

Caso um nó interno tenha três filhos, então: i) valores menores que sua primeira chave estarão à esquerda desse nó; ii) valores maiores que sua segunda chave estarão à direita do nó; iii) valores maiores que a primeira chave mas menores que a segunda chave estarão no nó central.



Árvore 2-3 - Propriedades 3/3

Dada uma chave qualquer, valores menores que esta chave estarão à esquerda (seja em uma chave no mesmo nó ou em um dos nós à esquerda). O mesmo raciocínio vale para valores maiores.



Árvores 2-3 - Inserção 1/2

O processo de inserção em árvores 2-3 é relativamente simples:

1. Procura-se o nó folha em que será feita a inserção.
2. Caso o nó tenha uma única chave, procura-se a posição de inserção da chave (antes ou depois da existente) e o processo se encerra.
3. Caso o nó já possua duas chaves, é necessário dividi-lo para realizar a inserção.

Árvore 2-3 - Inserção 2/2

O processo de divisão implica que uma das chaves irá subir para o nó pai, separando os dois novos nós.

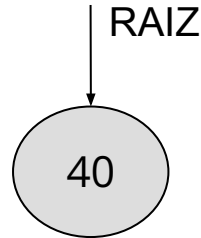
Nesse caso, o nó pai é avaliado para verificar se não deve também ser dividido.

Parte das implementações realizam a divisão antes da inserção, enquanto outras inserem o valor para dividir posteriormente.

Iremos utilizar a divisão após a inserção, o que implica que um nó poderá armazenar, temporariamente, três chaves.

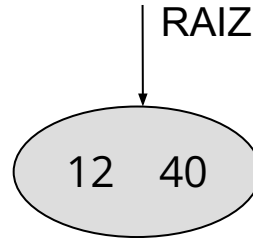
Exemplo - Inserção - 1/19

Sequência de inserção: **40** 12 68 36 38 60 48 55 50 62 65 22 90



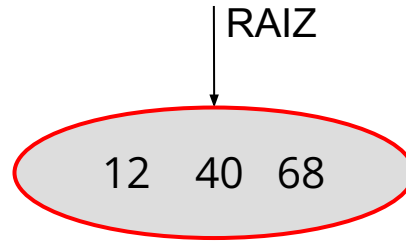
Exemplo - Inserção - 2/19

Sequência de inserção: 40 **12** 68 36 38 60 48 55 50 62 65



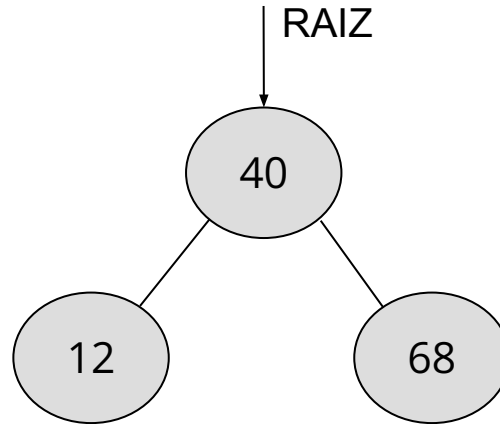
Exemplo - Inserção - 3/19

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65



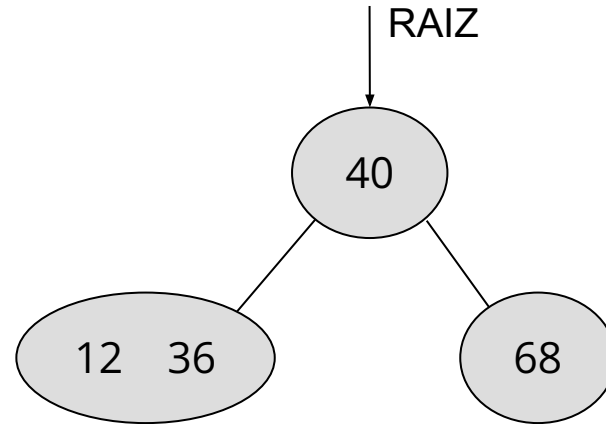
Exemplo - Inserção - 4/19

Sequência de inserção: 40 12 **68** 36 38 60 48 55 50 62 65



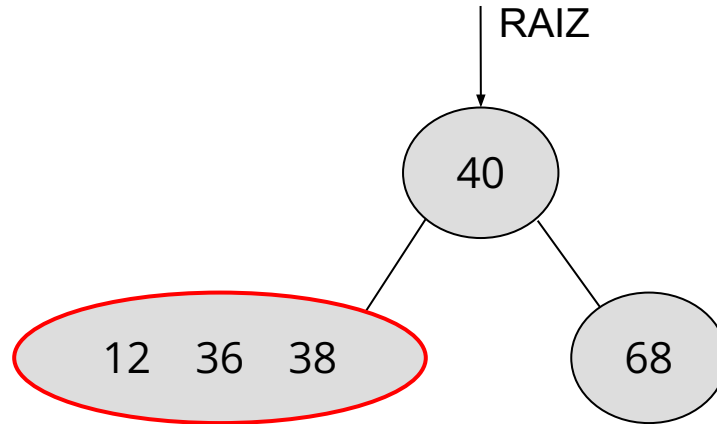
Exemplo - Inserção - 5/19

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65



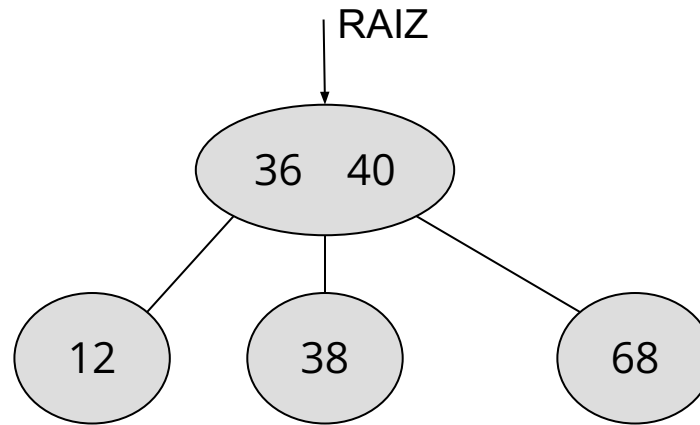
Exemplo - Inserção - 6/19

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65



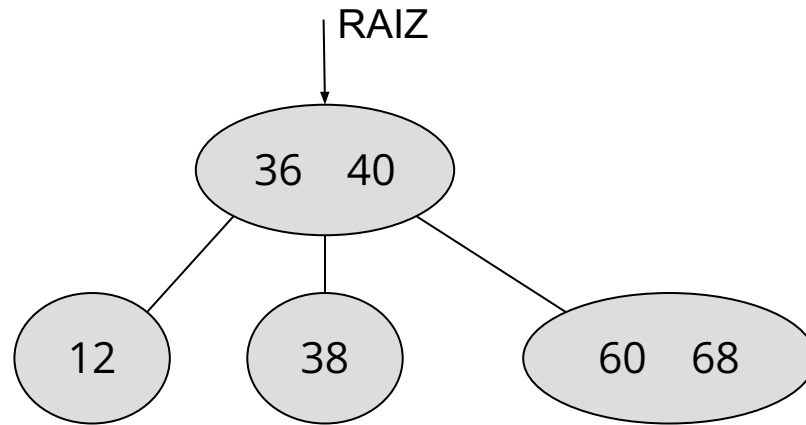
Exemplo - Inserção - 7/19

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65



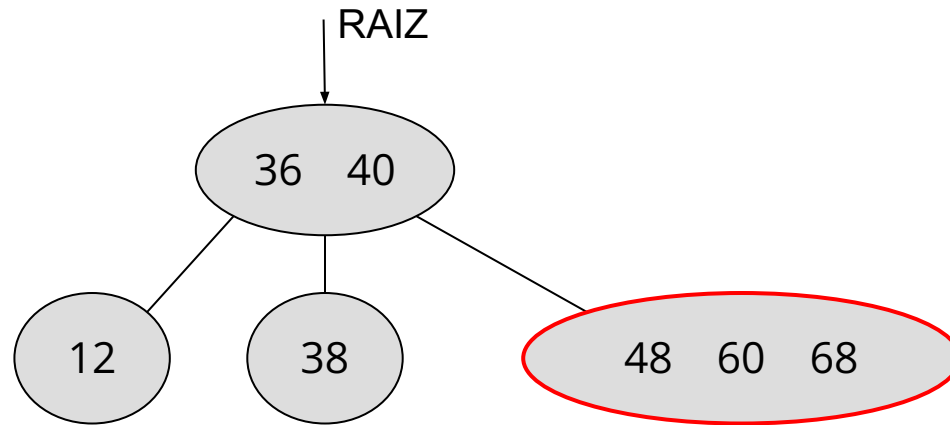
Exemplo - Inserção - 8/19

Sequência de inserção: 40 12 68 36 38 **60** 48 55 50 62 65



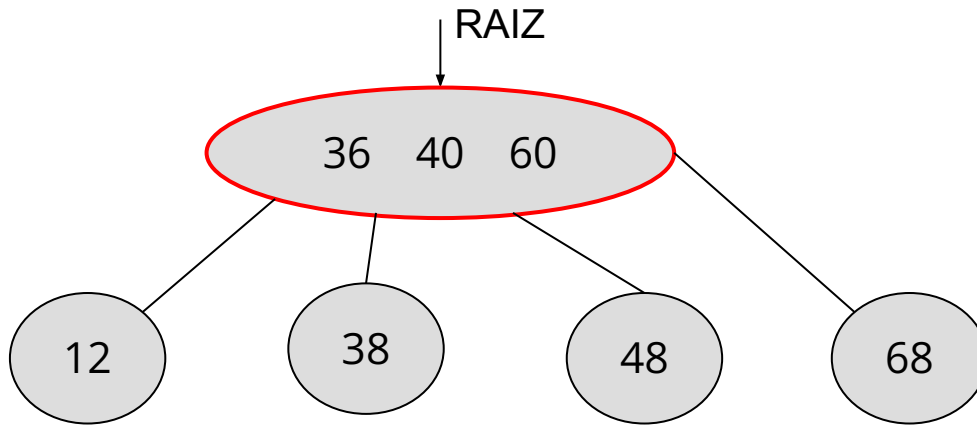
Exemplo - Inserção - 9/19

Sequência de inserção: 40 12 68 36 38 60 **48** 55 50 62 65



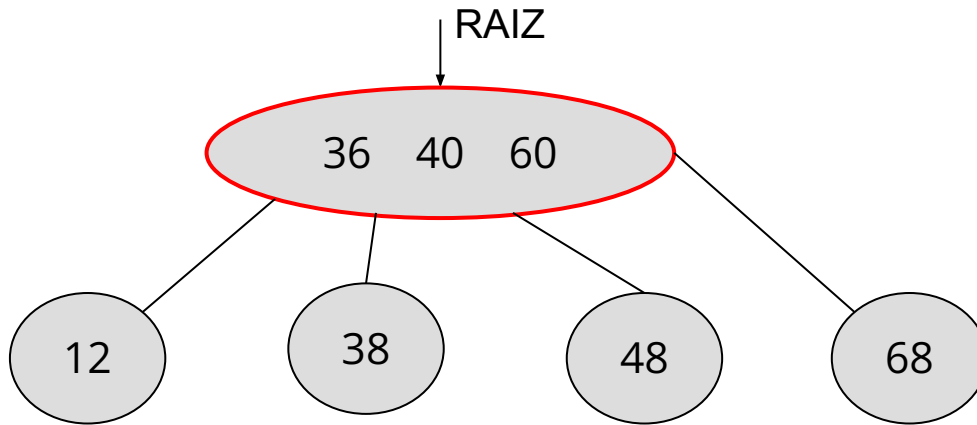
Exemplo - Inserção - 10/19

Sequência de inserção: 40 12 68 36 38 60 **48** 55 50 62 65



Exemplo - Inserção - 11/19

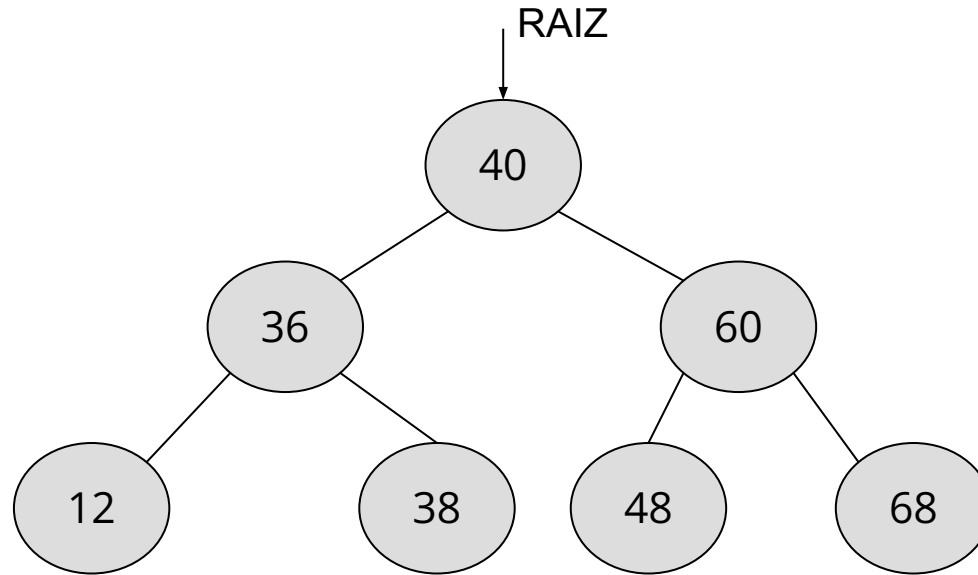
Sequência de inserção: 40 12 68 36 38 60 **48** 55 50 62 65



Um dos problemas da abordagem de divisão após inserção é que ajustes acabam sendo feitos em um nó que será dividido na etapa posterior.

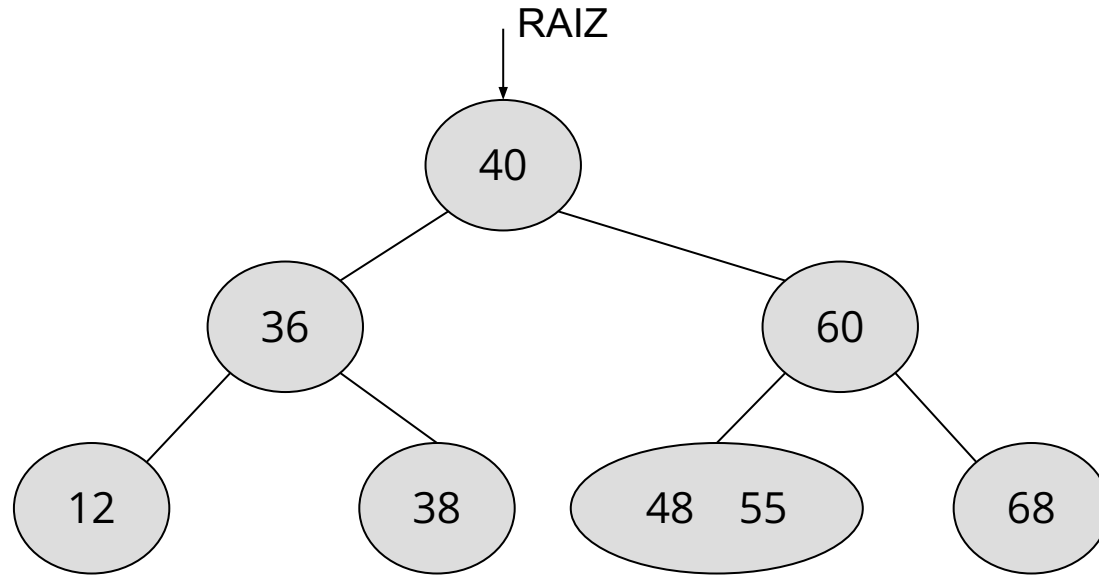
Exemplo - Inserção - 12/19

Sequência de inserção: 40 12 68 36 38 60 **48** 55 50 62 65



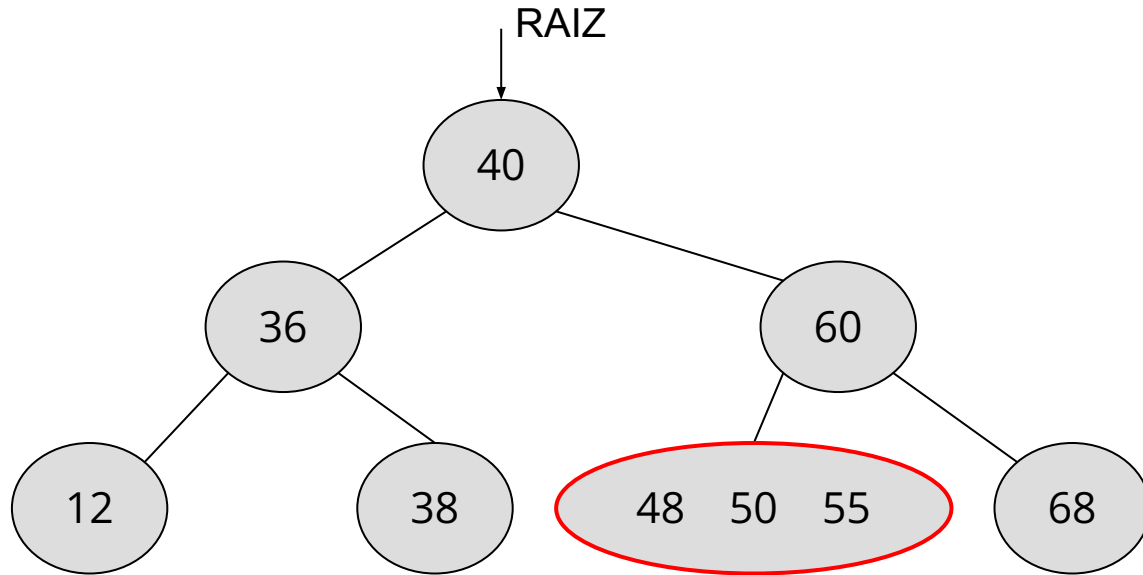
Exemplo - Inserção - 13/19

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65



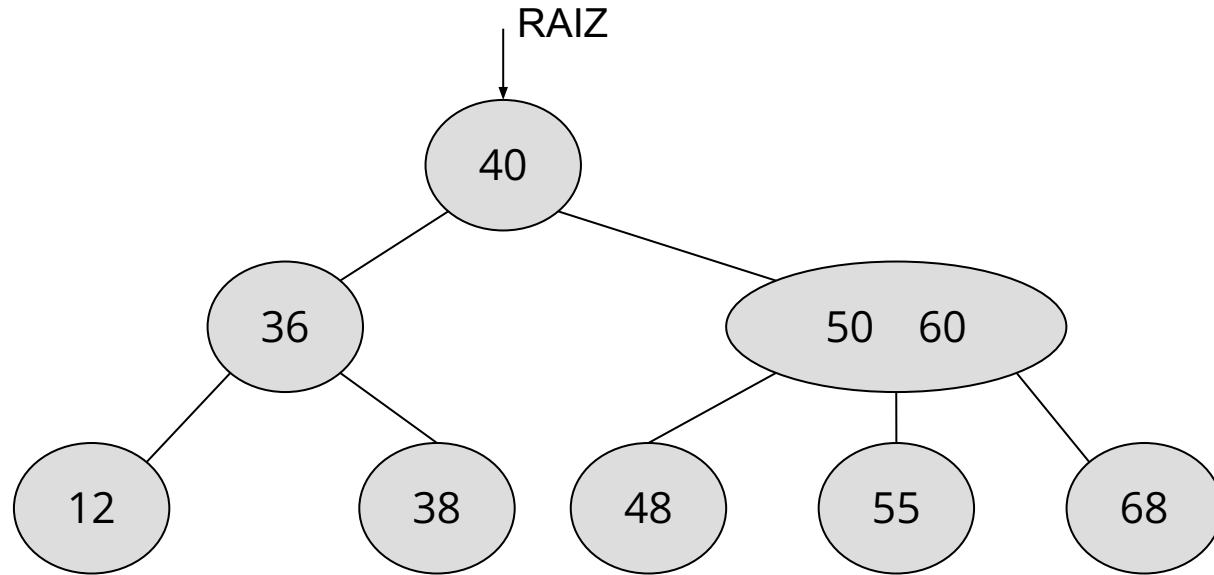
Exemplo - Inserção - 14/19

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65



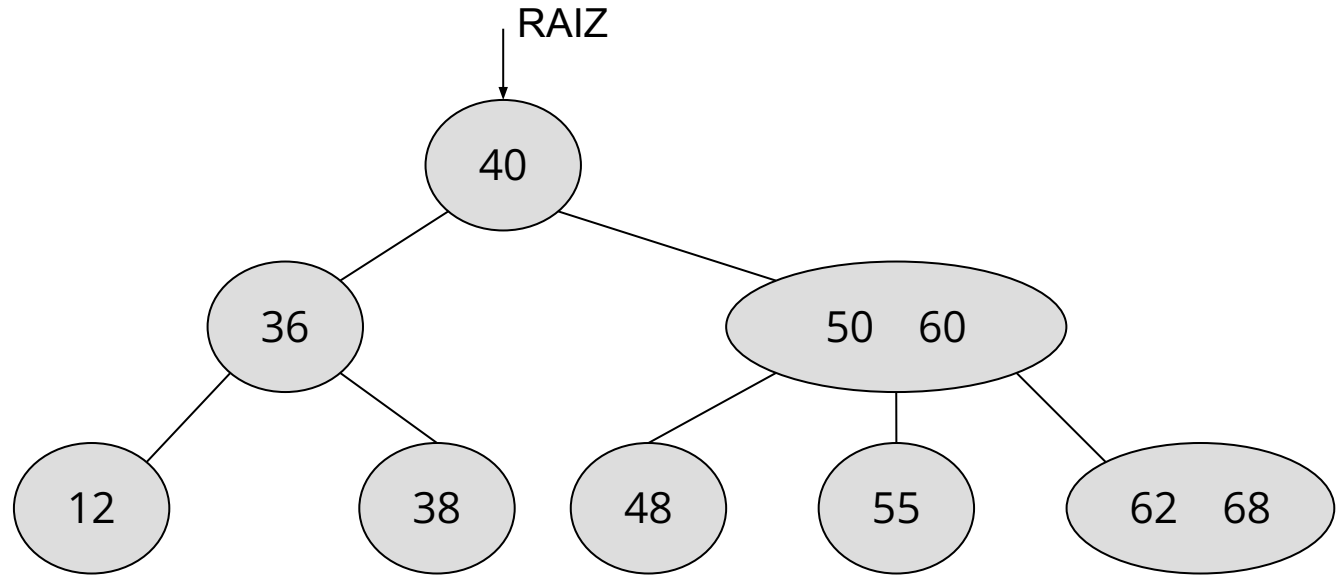
Exemplo - Inserção - 15/19

Sequência de inserção: 40 12 68 36 38 60 48 55 **50** 62 65



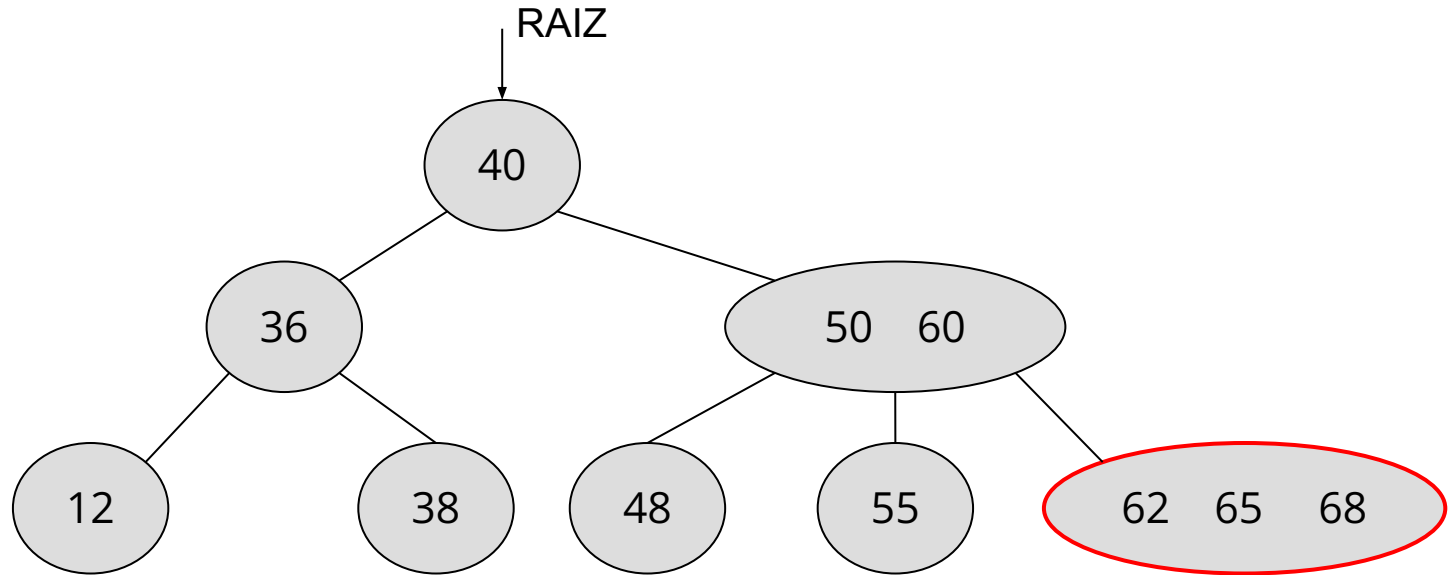
Exemplo - Inserção - 16/19

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65



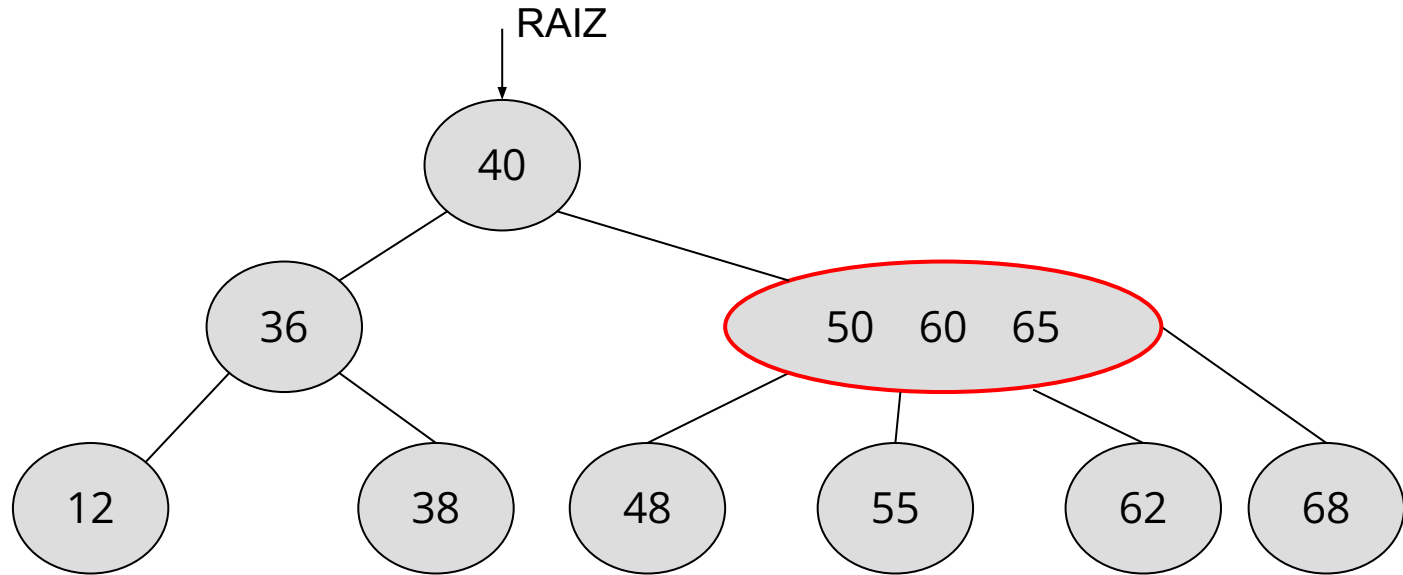
Exemplo - Inserção - 17/19

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65



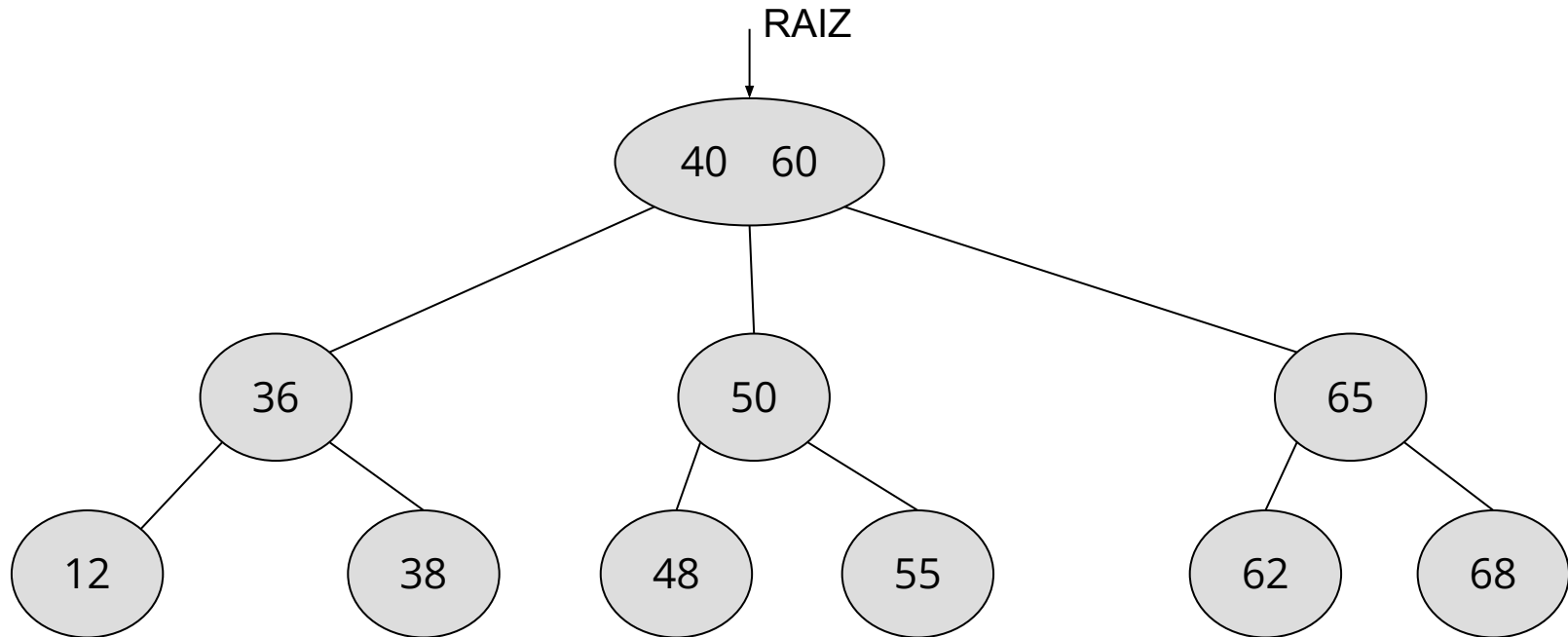
Exemplo - Inserção - 18/19

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65



Exemplo - Inserção - 19/19

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65



Árvore 2-3 - Remoção 1/2

O processo de remoção é similar ao da árvore binária, entretanto, alguns cuidados adicionais precisam ser tomados para manter o balanceamento:

1. A remoção é sempre feita em nós folhas, caso uma chave em um nó interno precise ser removida, ela é substituída por sua *sucessora* ou *antecessora*.
2. Caso a remoção na folha (da chave ou sua sucessora/antecessora) ocorra em um nó com duas chaves, o processo se encerra, caso contrário será necessário rotacionar chaves ou fundir nós.

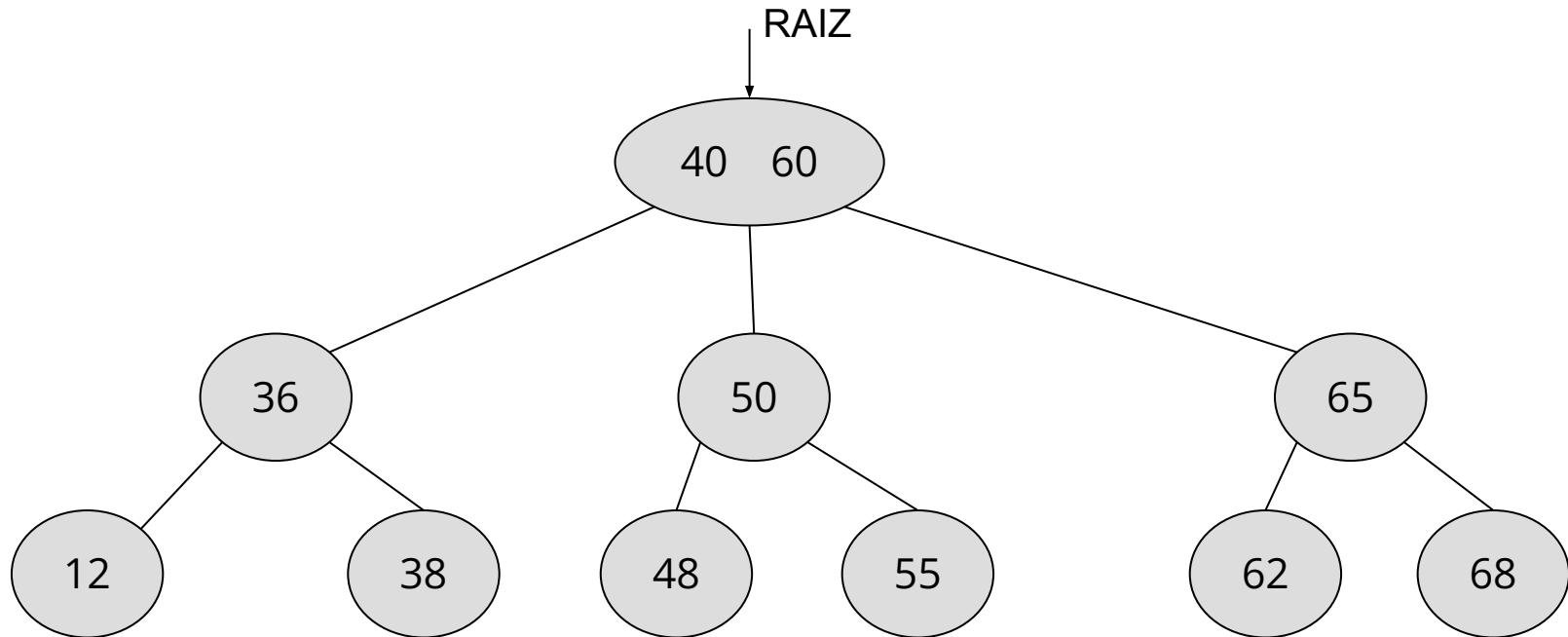
Árvore 2-3 - Remoção 2/2

Quando um nó fica vazio, mas seu irmão possui duas chaves, então é possível rotacionar chaves (envolvendo o nó pai) para manter o equilíbrio.

Quando um nó fica vazio e seus irmãos possuem apenas uma chave, é necessário efetuar a fusão de nós, envolvendo nó pai e nó irmão: a chave do nó pai que separa o nó vazio com o seu irmão irá descer para ficar no nó irmão. Com isso, é necessário verificar se o nó pai também não irá ficar vazio.

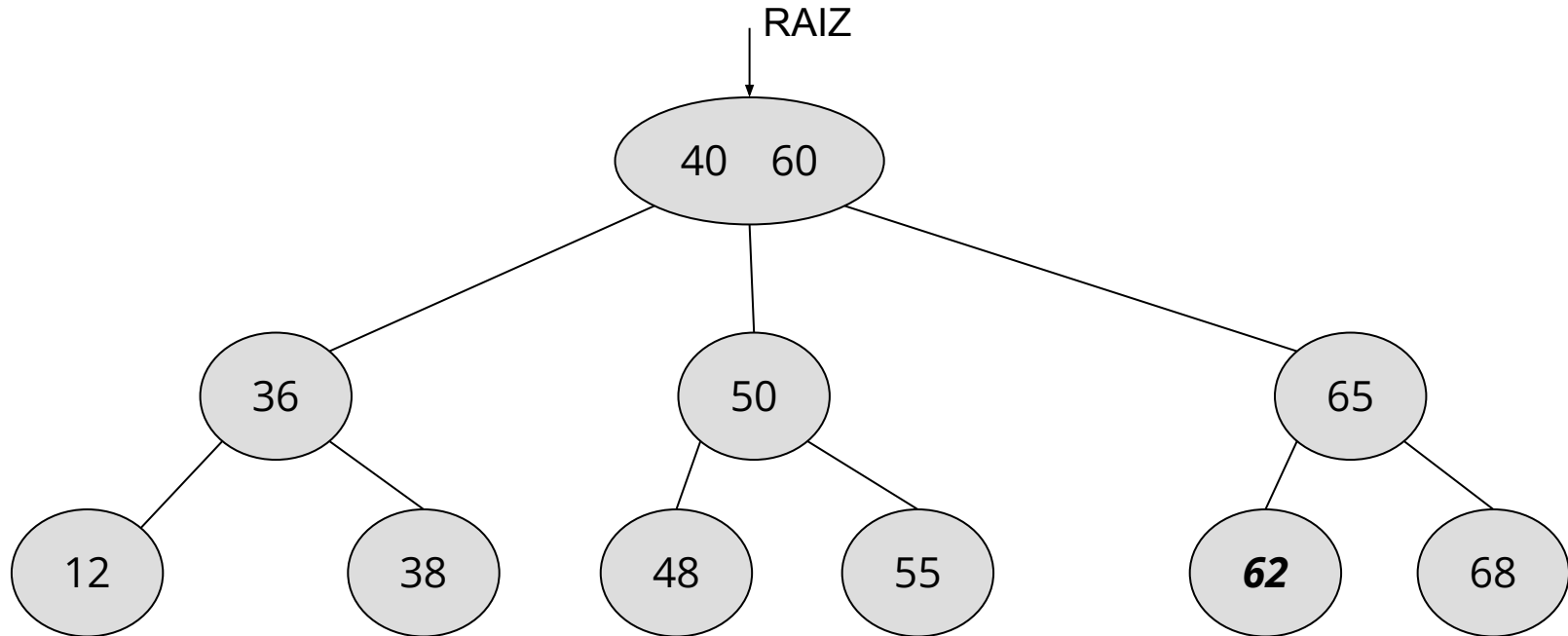
Exemplo - Remoção - 1/23

Sequência de remoção: ~~60~~ 36 38 40 12 55



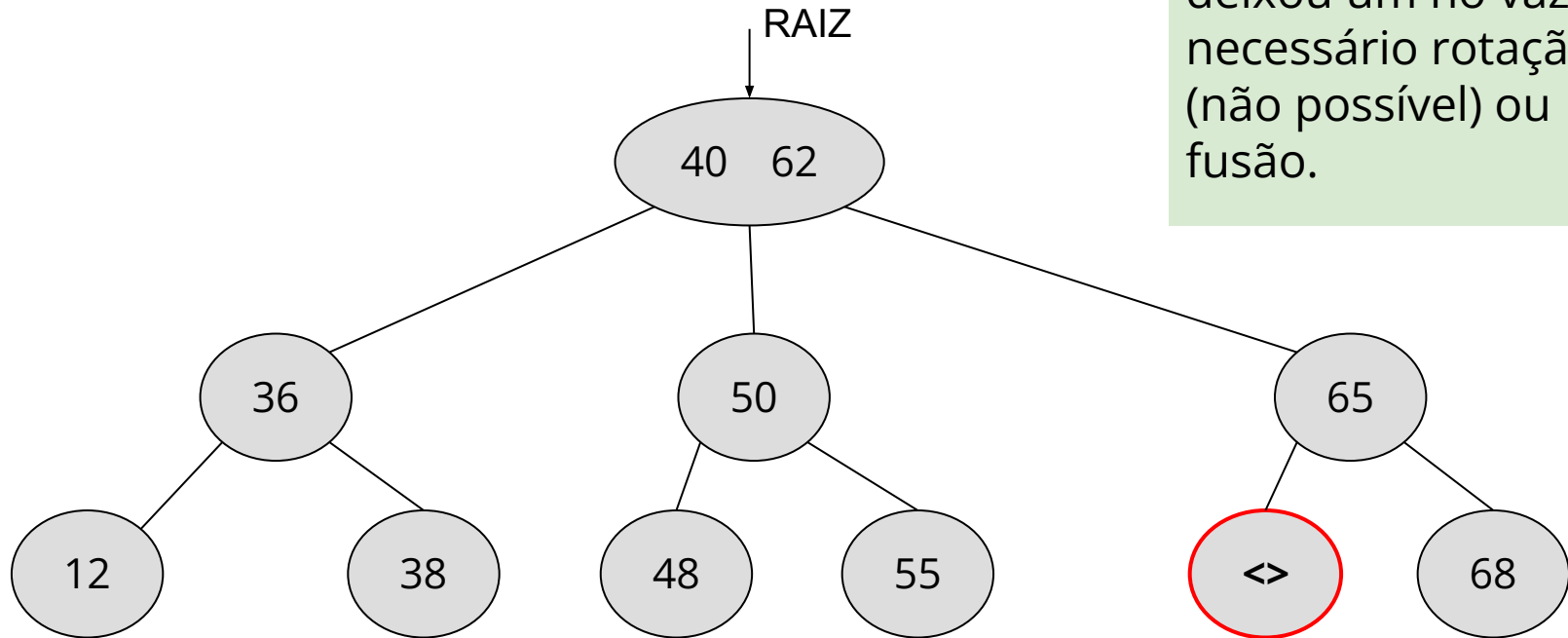
Exemplo - Remoção - 2/23

Sequência de remoção: **60** 36 38 40 12 55



Exemplo - Remoção - 3/23

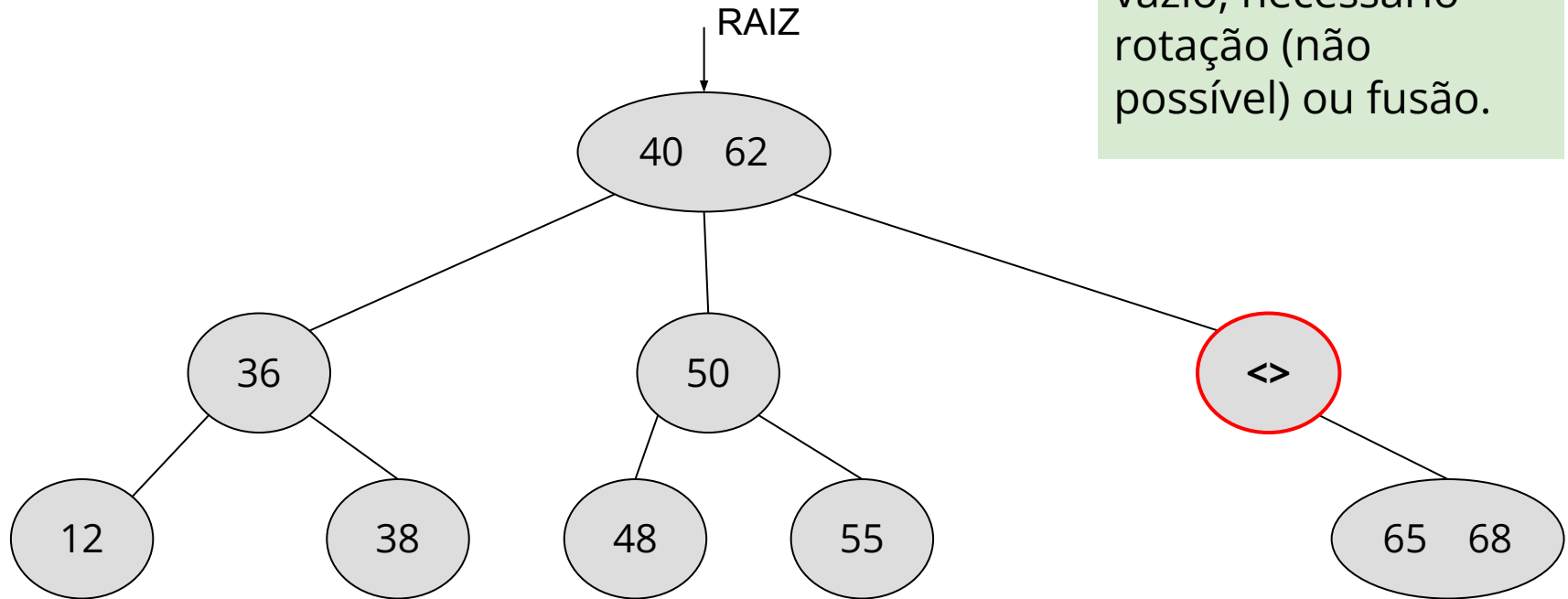
Sequência de remoção: ~~60~~ 36 38 40 12 55



Remoção da chave deixou um nó vazio, necessário rotação (não possível) ou fusão.

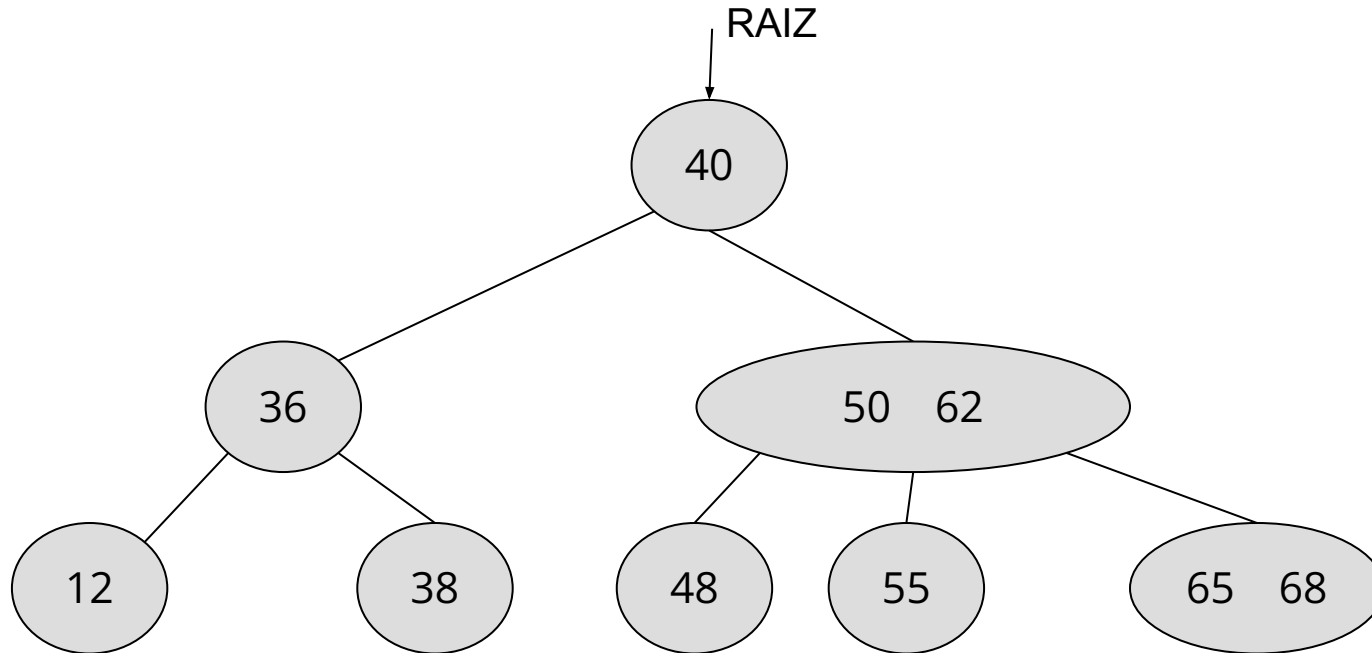
Exemplo - Remoção - 4/23

Sequência de remoção: ~~60~~ 36 38 40 12 55



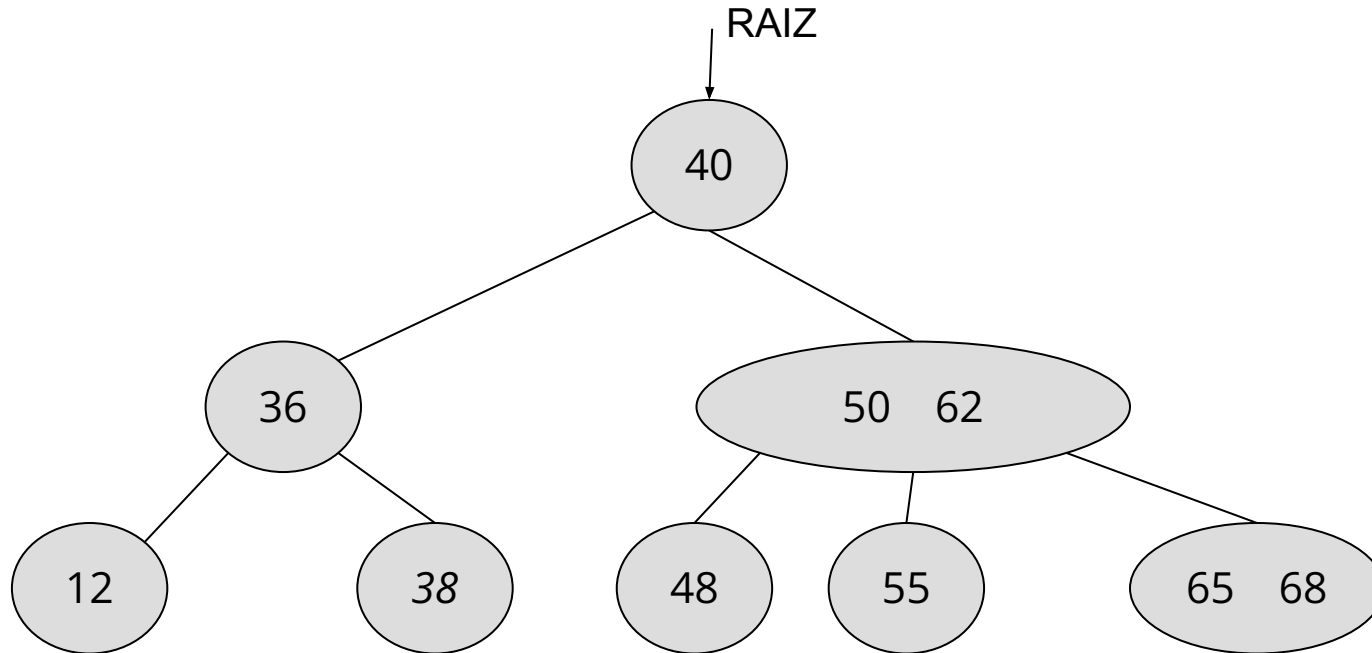
Exemplo - Remoção - 5/23

Sequência de remoção: ~~60~~ 36 38 40 12 55



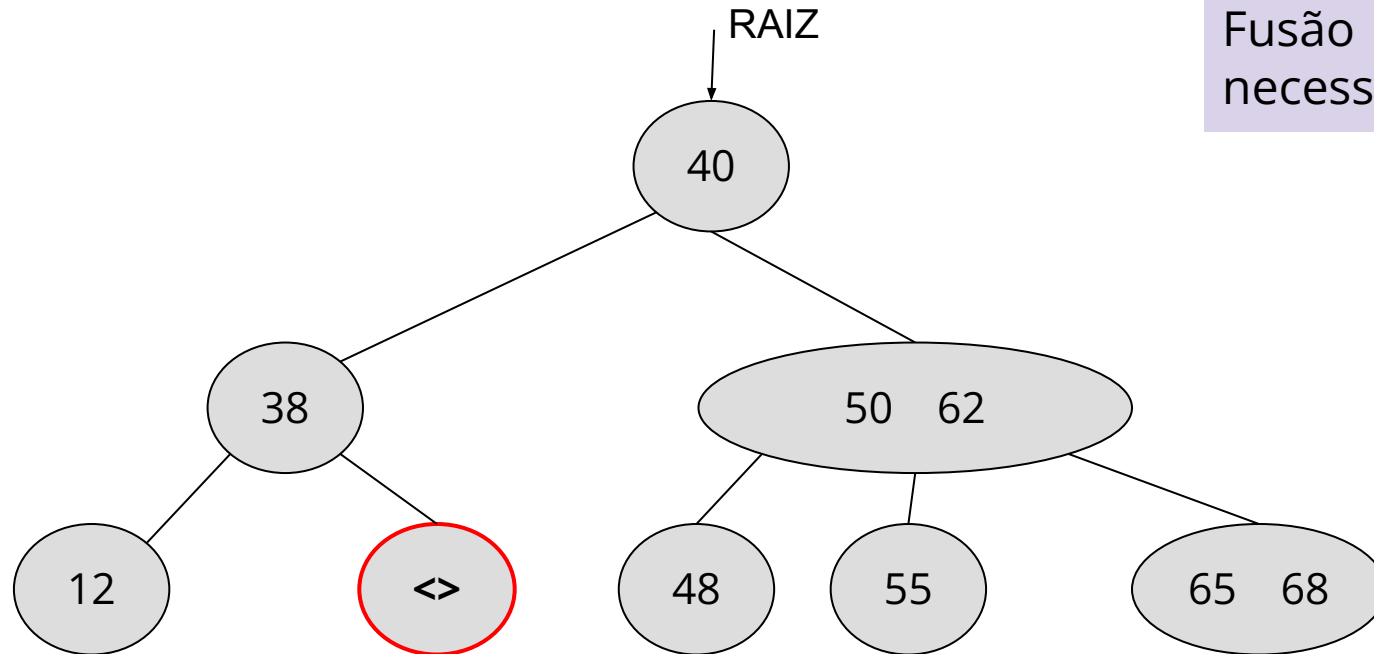
Exemplo - Remoção - 6/23

Sequência de remoção: 60 36 38 40 12 55



Exemplo - Remoção - 7/23

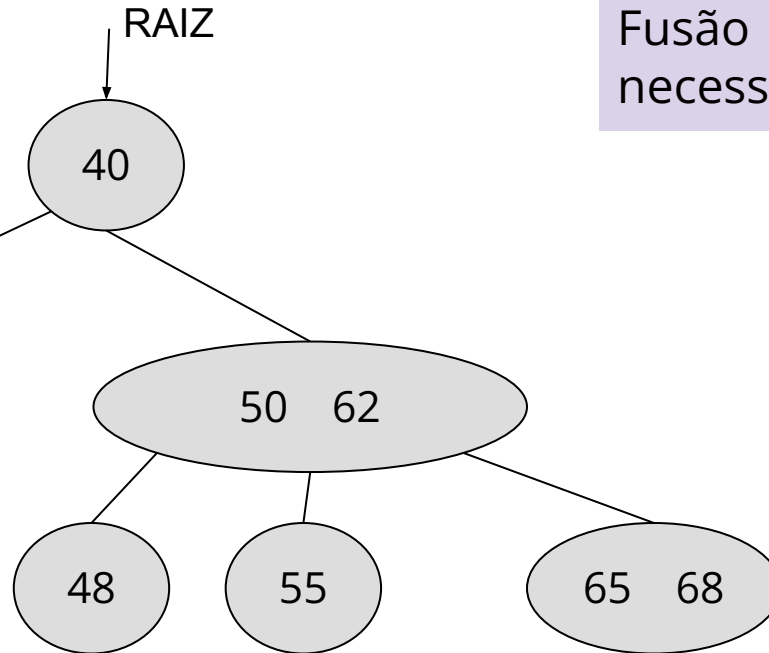
Sequência de remoção: 60 36 38 40 12 55



Fusão
necessária

Exemplo - Remoção - 7/23

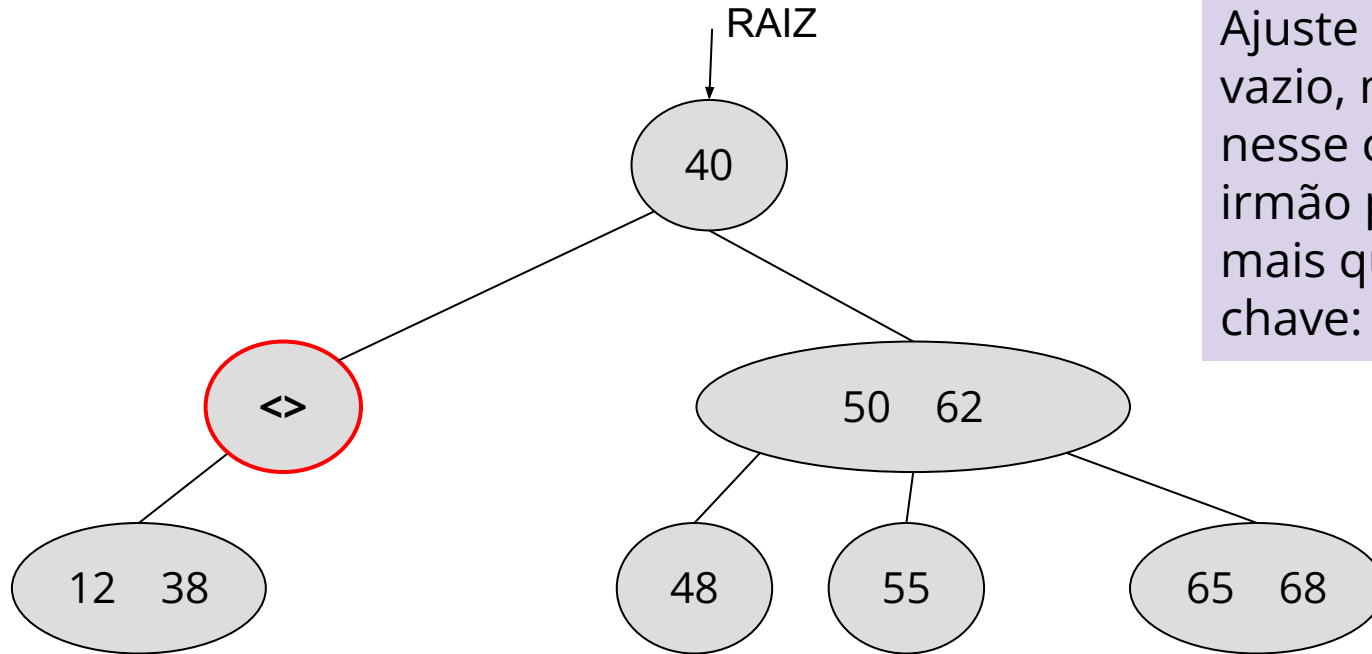
Sequência de remoção: 60 36 38 40 12 55



Fusão
necessária

Exemplo - Remoção - 8/23

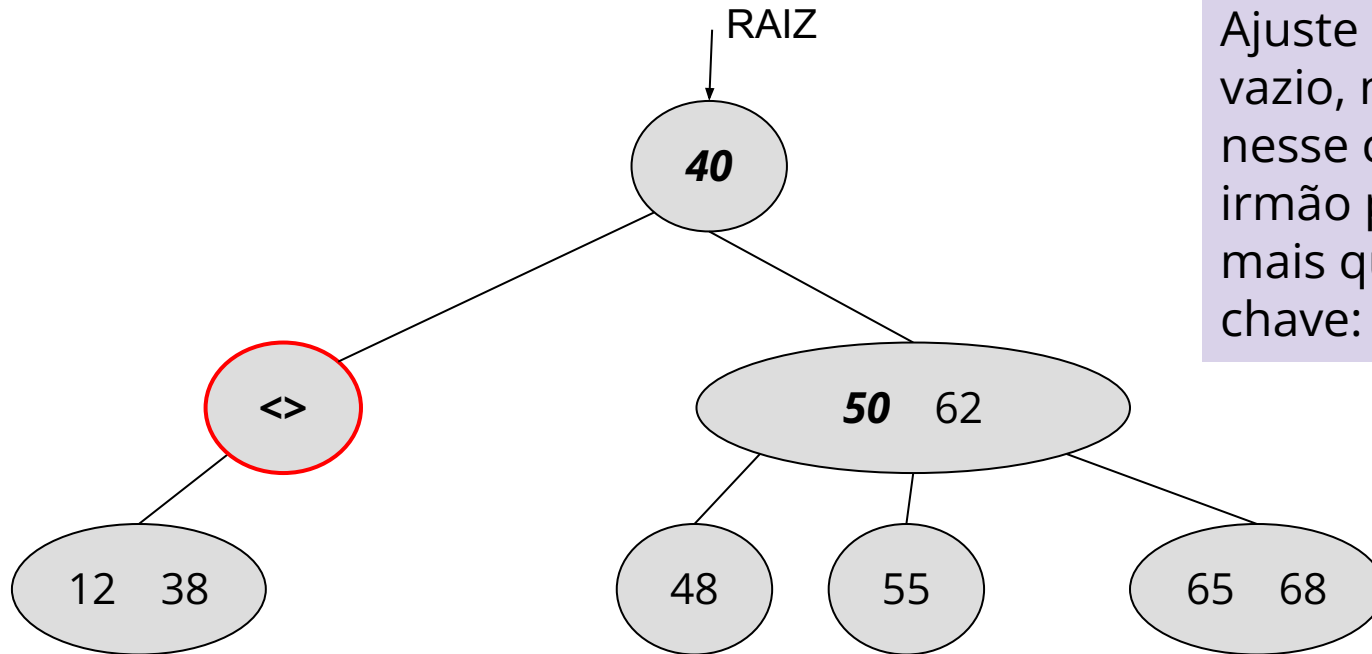
Sequência de remoção: 60 36 38 40 12 55



Ajuste deixou nó vazio, mas, nesse caso, irmão possui mais que uma chave: *rotação!*

Exemplo - Remoção - 9/23

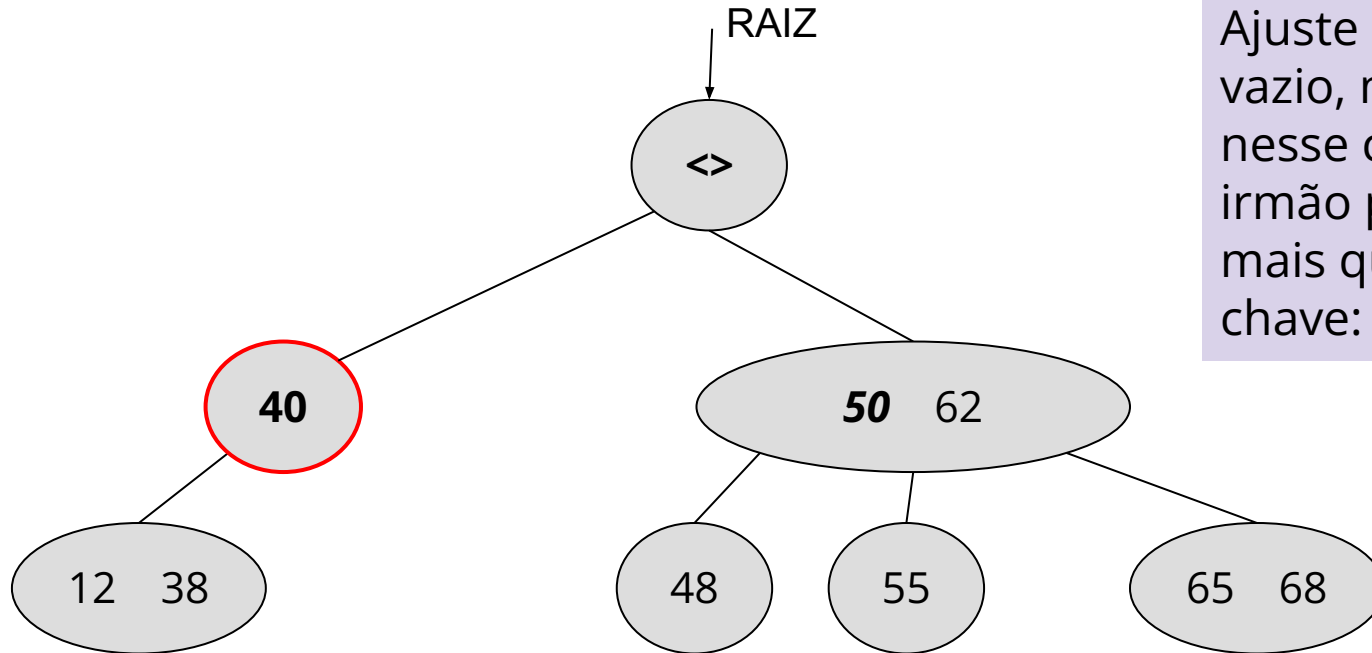
Sequência de remoção: 60 36 38 40 12 55



Ajuste deixou nó vazio, mas, nesse caso, irmão possui mais que uma chave: *rotação!*

Exemplo - Remoção - 10/23

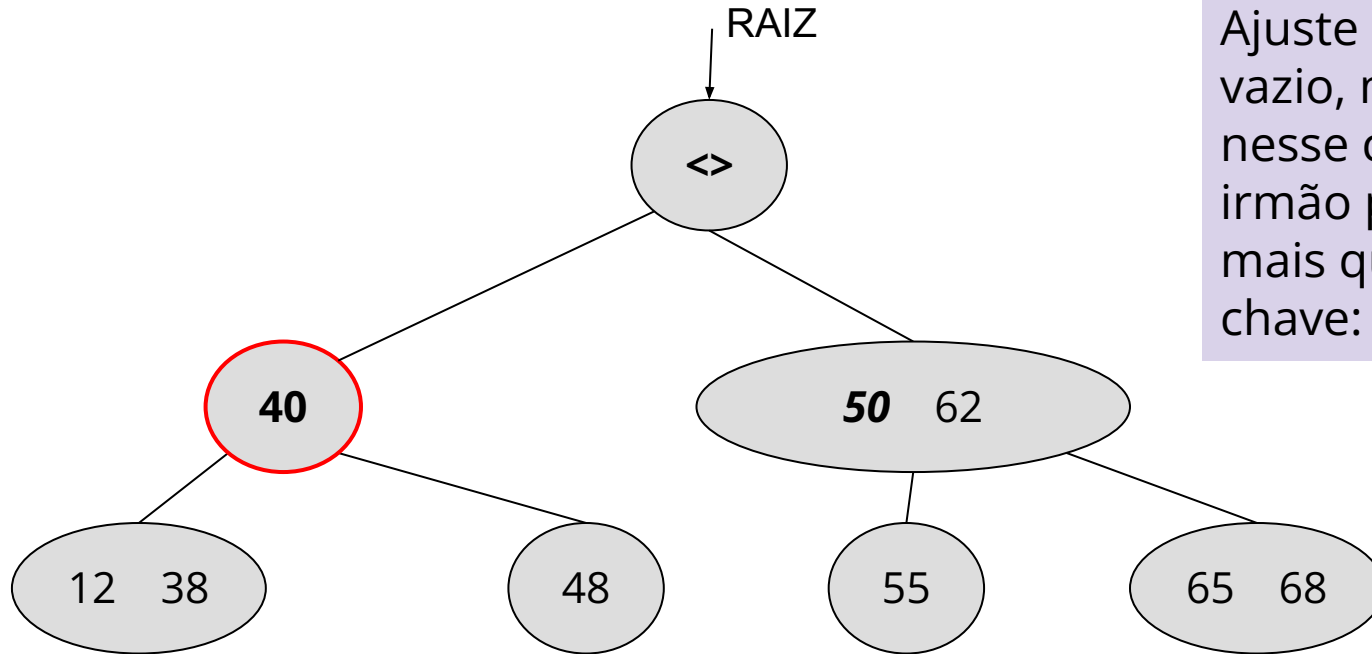
Sequência de remoção: 60 36 38 40 12 55



Ajuste deixou nó vazio, mas, nesse caso, irmão possui mais que uma chave: *rotação!*

Exemplo - Remoção - 11/23

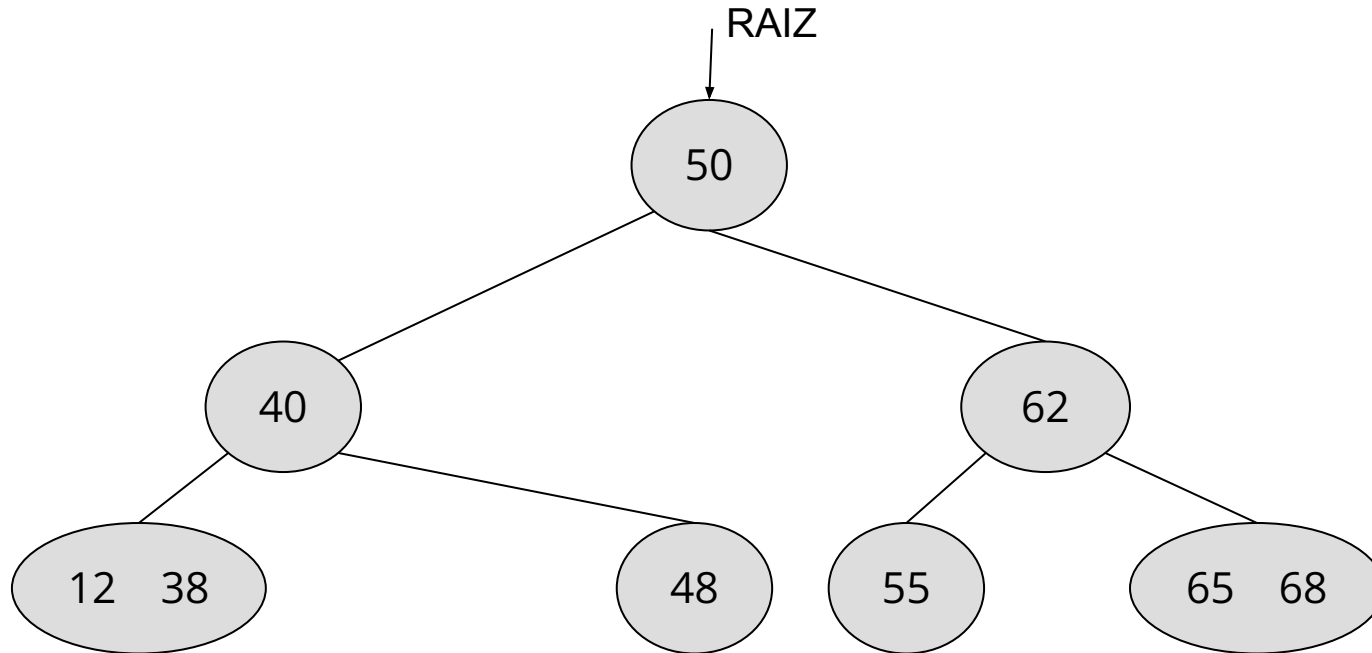
Sequência de remoção: 60 36 38 40 12 55



Ajuste deixou nó vazio, mas, nesse caso, irmão possui mais que uma chave: *rotação!*

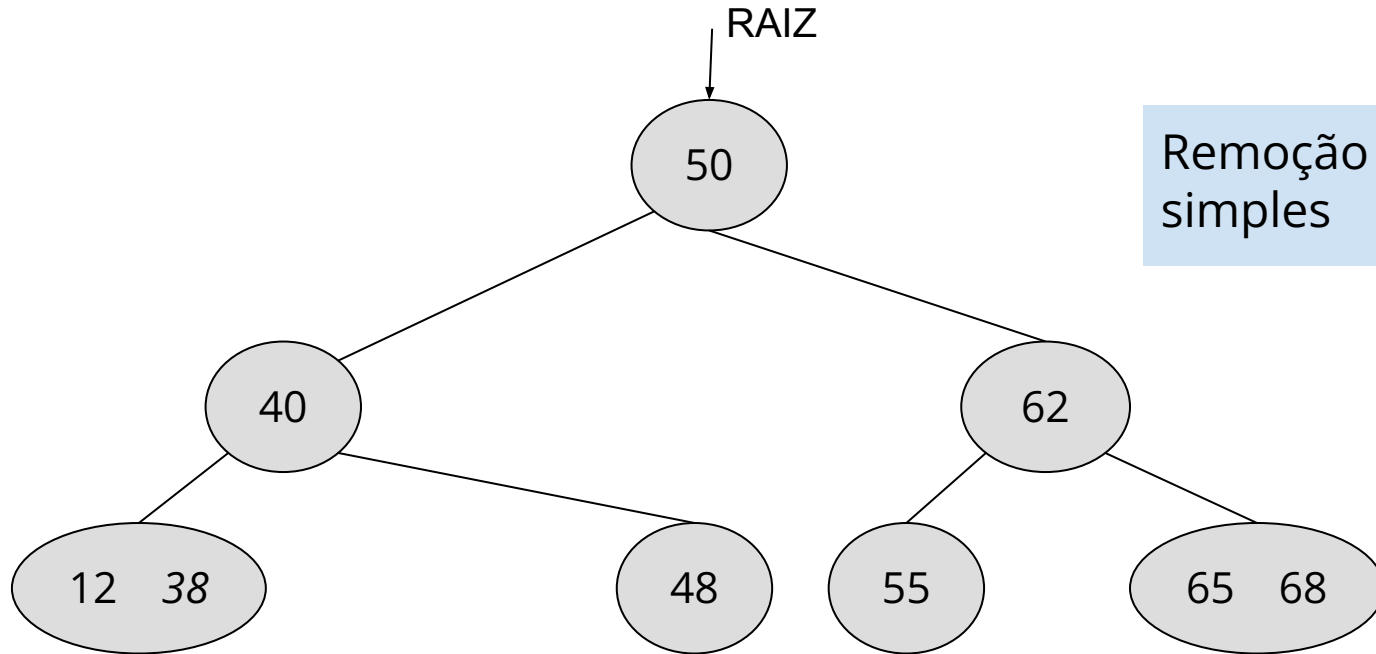
Exemplo - Remoção - 12/23

Sequência de remoção: 60 36 38 40 12 55



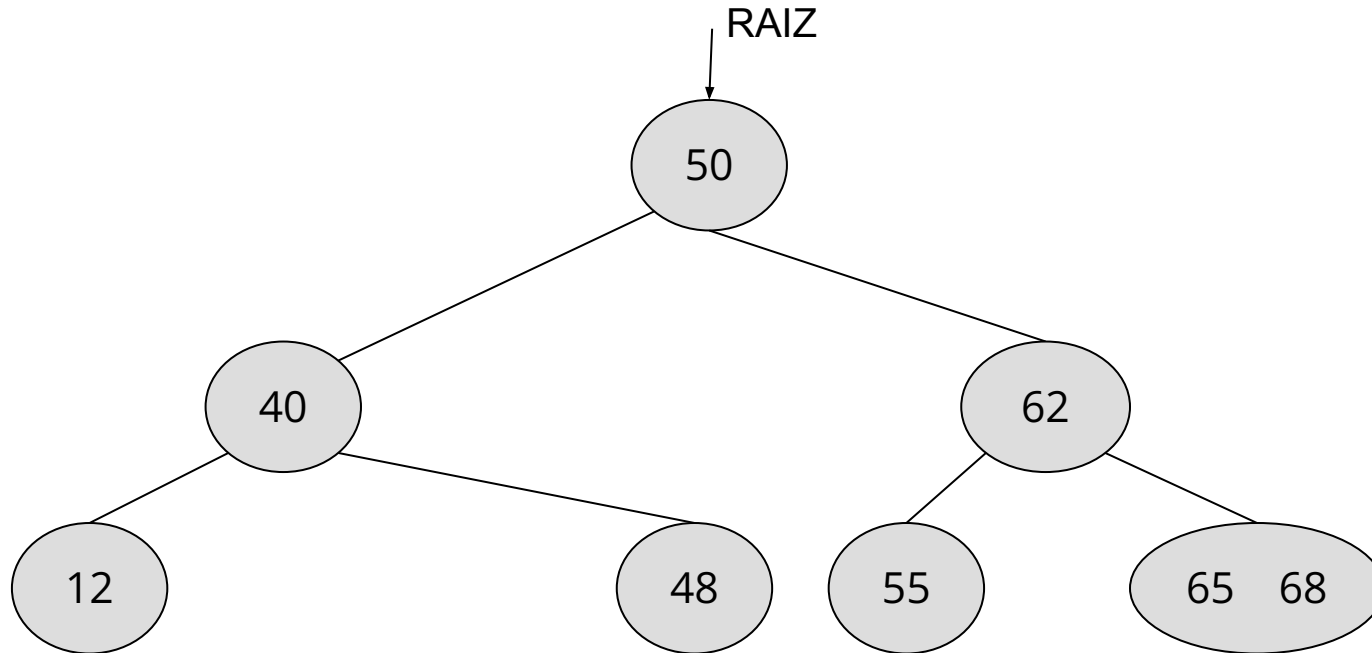
Exemplo - Remoção - 13/23

Sequência de remoção: 60 36 38 40 12 55



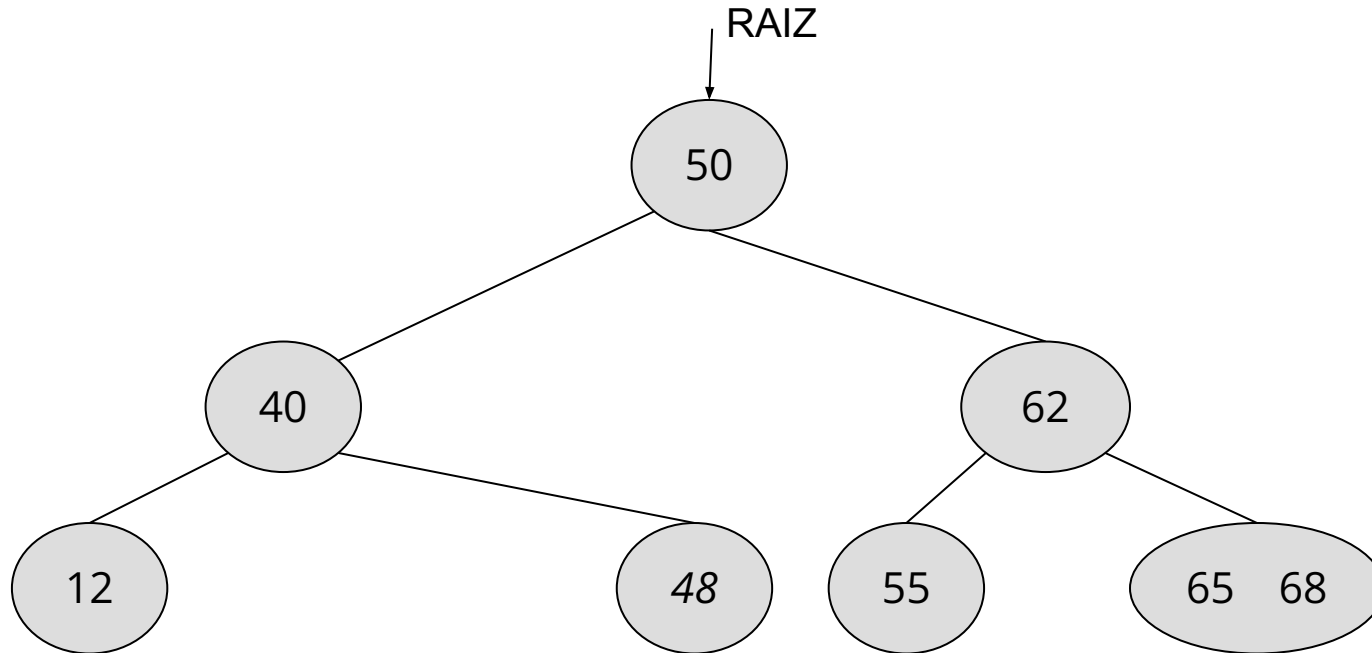
Exemplo - Remoção - 14/23

Sequência de remoção: 60 36 38 40 12 55



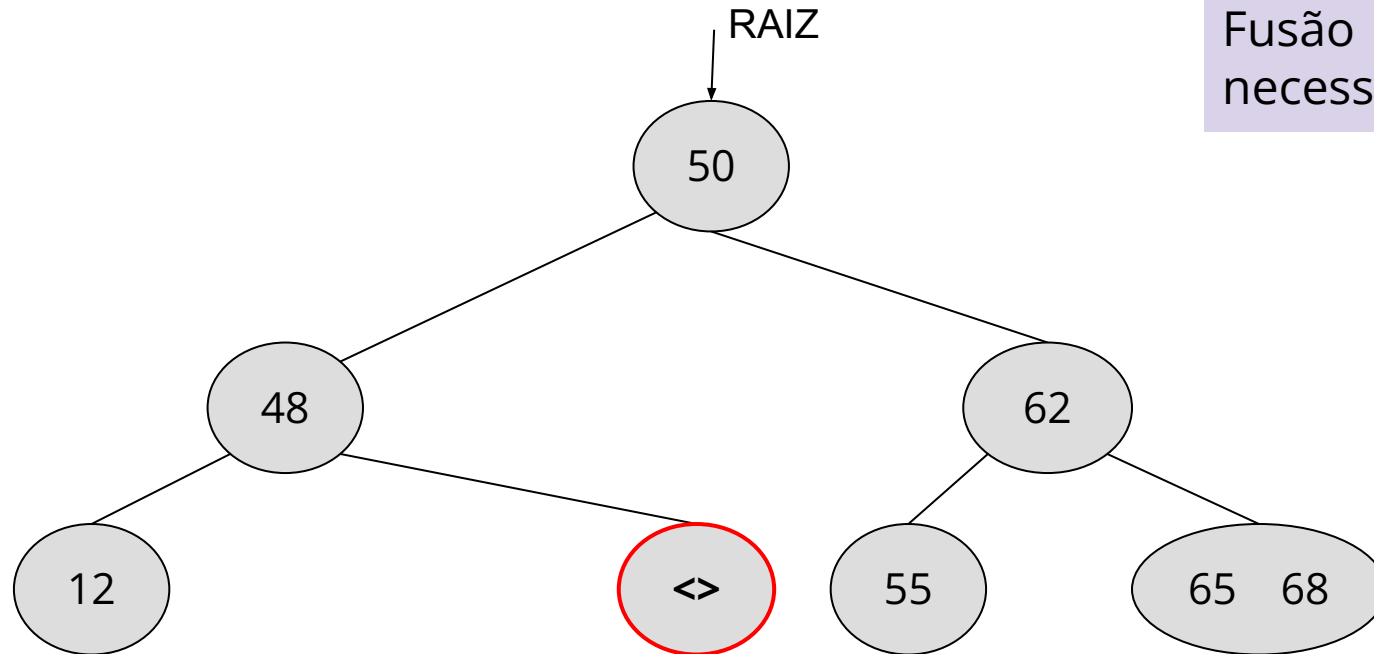
Exemplo - Remoção - 15/23

Sequência de remoção: 60 36 38 **40** 12 55



Exemplo - Remoção - 16/23

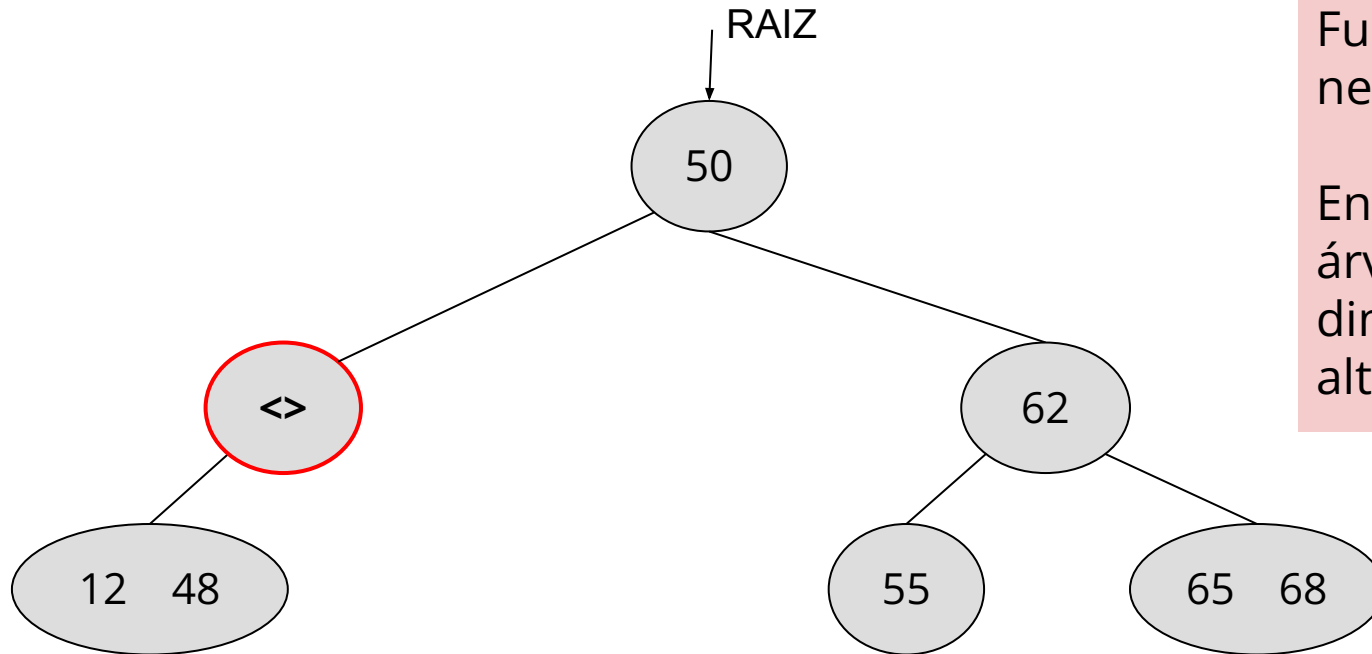
Sequência de remoção: 60 36 38 **40** 12 55



Fusão
necessária

Exemplo - Remoção - 17/23

Sequência de remoção: 60 36 38 **40** 12 55

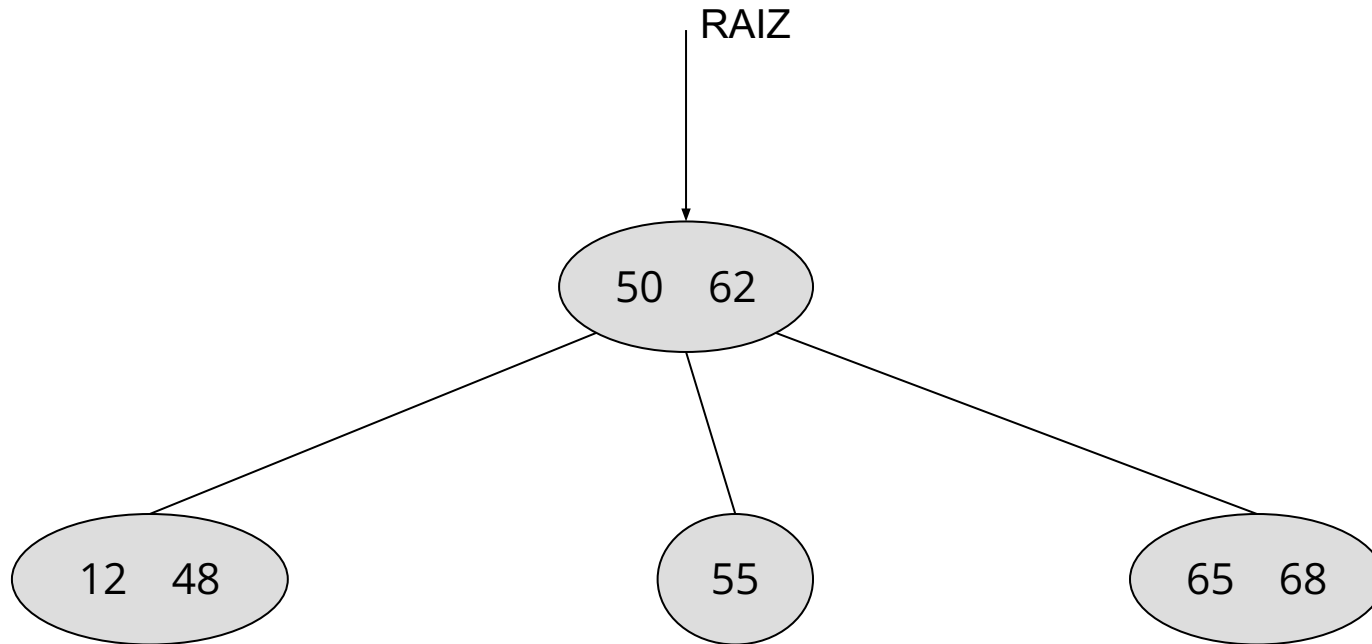


Fusão
necessária

Envolve raiz:
árvore
diminui
altura

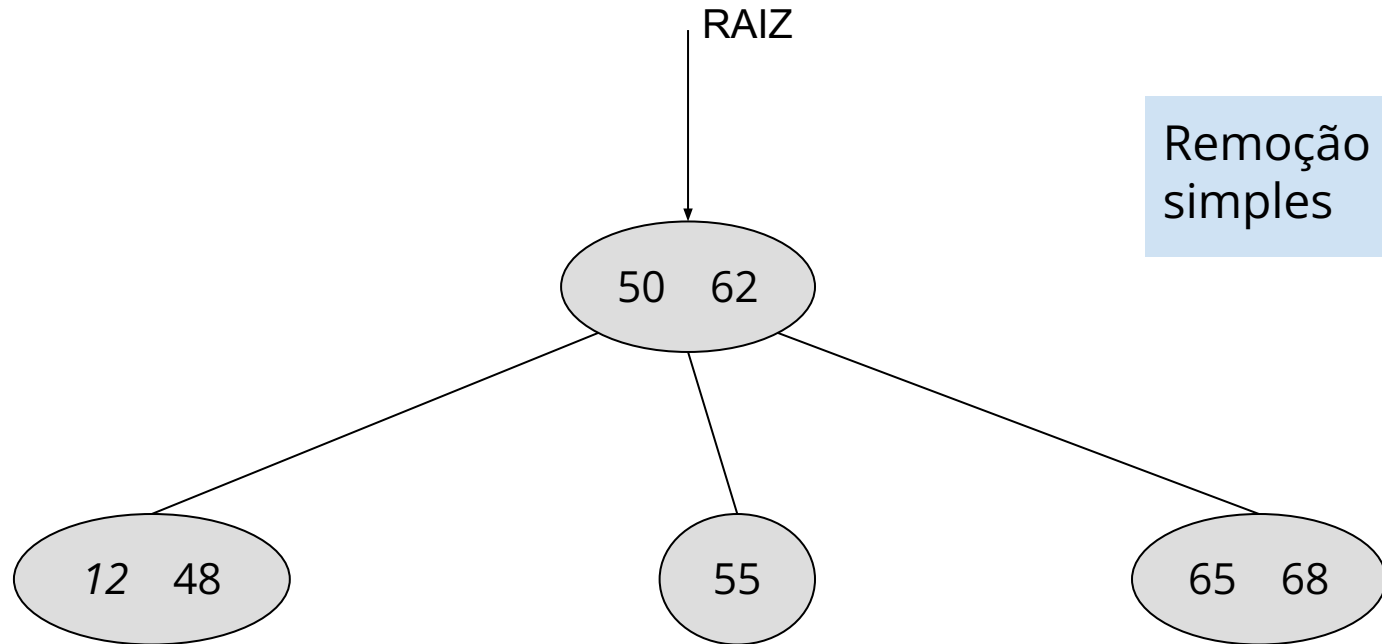
Exemplo - Remoção - 18/23

Sequência de remoção: 60 36 38 **40** 12 55



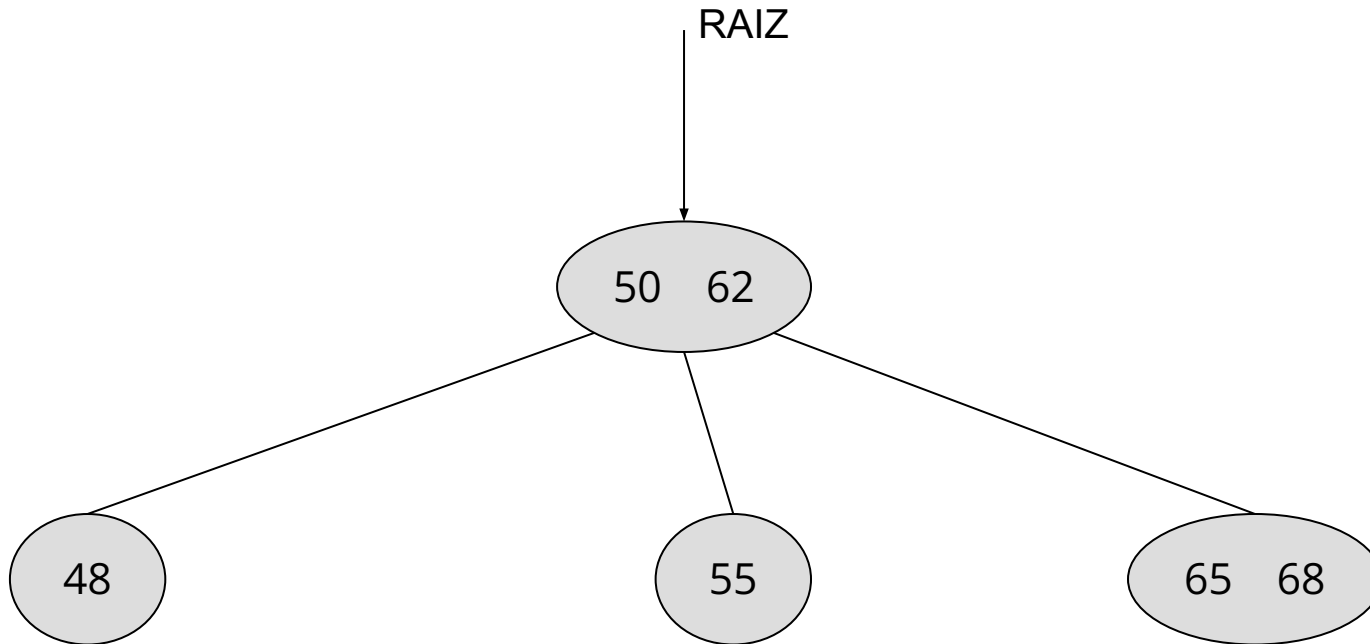
Exemplo - Remoção - 19/23

Sequência de remoção: 60 36 38 40 **12** 55



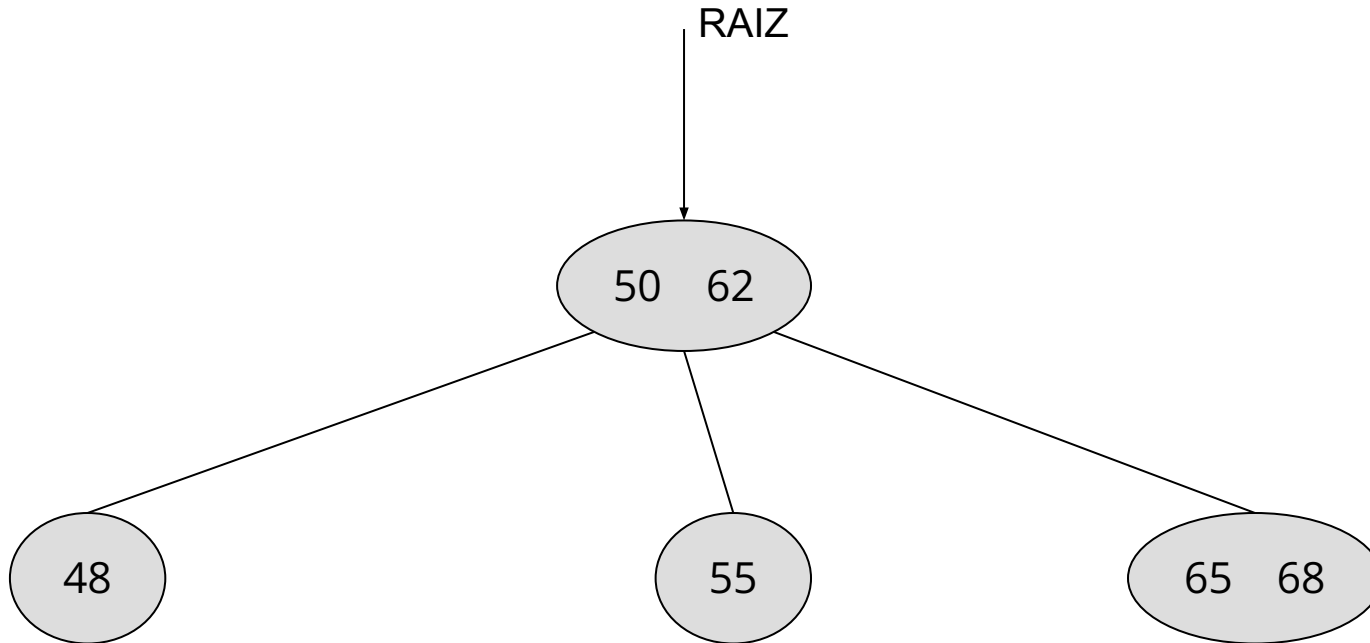
Exemplo - Remoção - 20/23

Sequência de remoção: 60 36 38 40 **12** 55



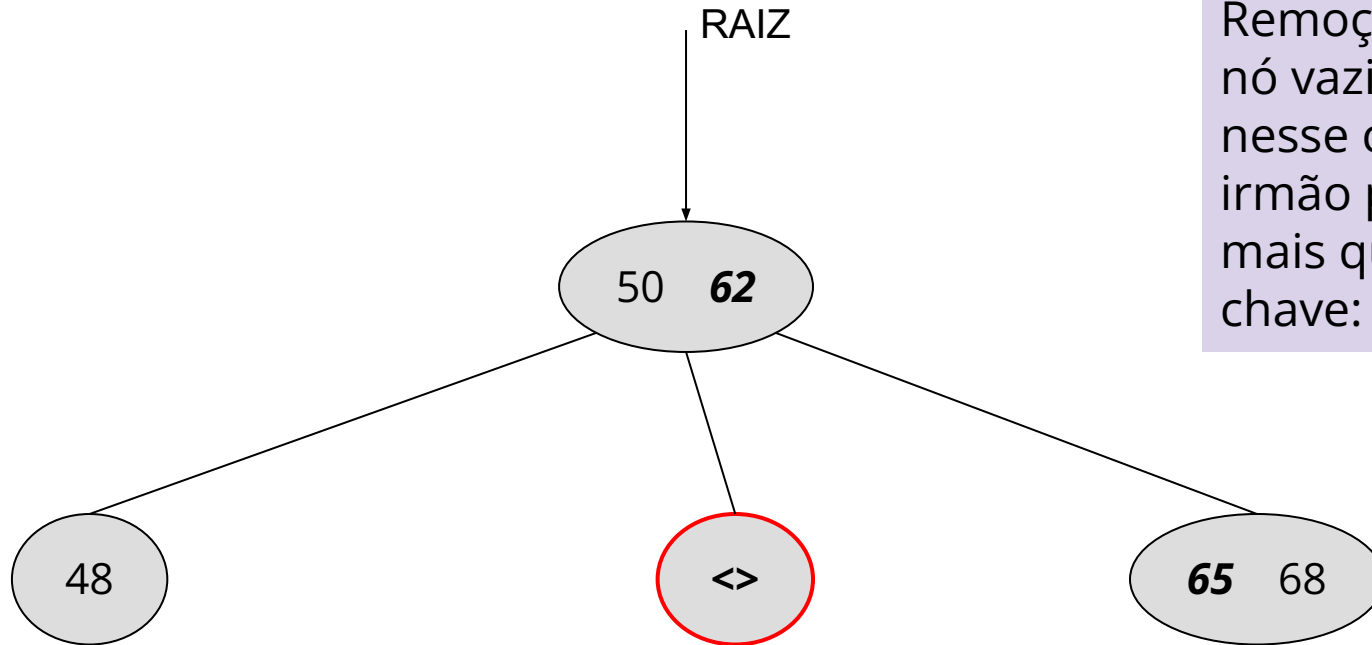
Exemplo - Remoção - 21/23

Sequência de remoção: 60 36 38 40 12 55



Exemplo - Remoção - 22/23

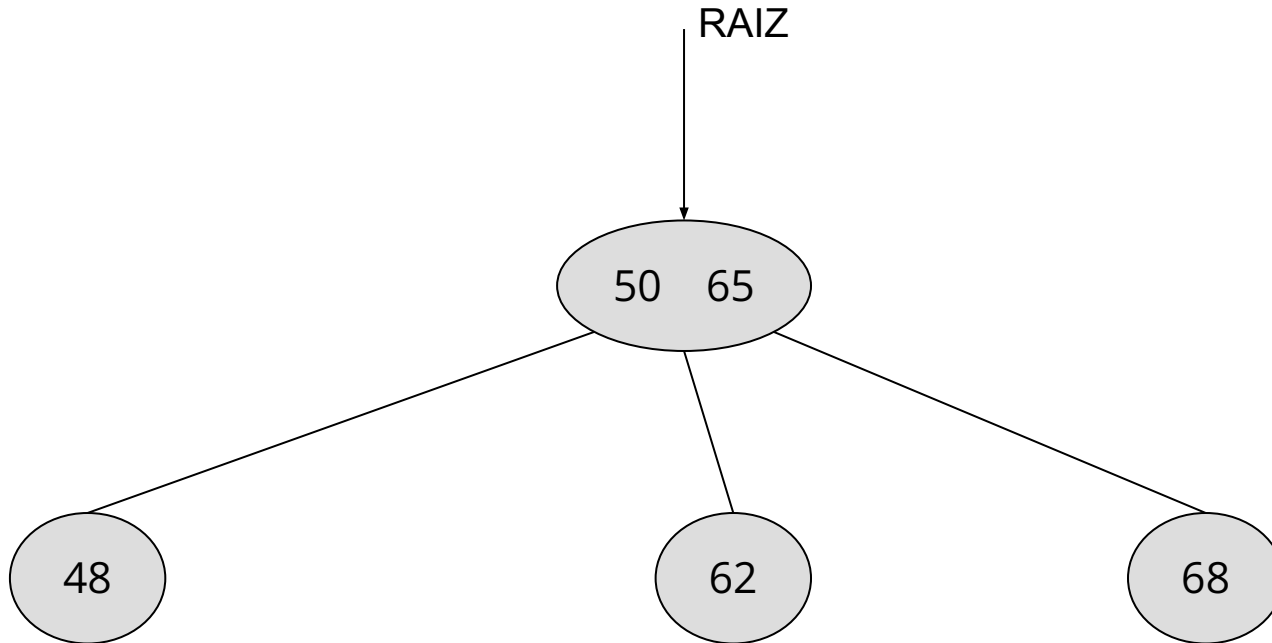
Sequência de remoção: 60 36 38 40 12 55



Remoção deixou
nó vazio, mas,
nesse caso,
irmão possui
mais que uma
chave: *rotação!*

Exemplo - Remoção - 23/23

Sequência de remoção: 60 36 38 40 12 55



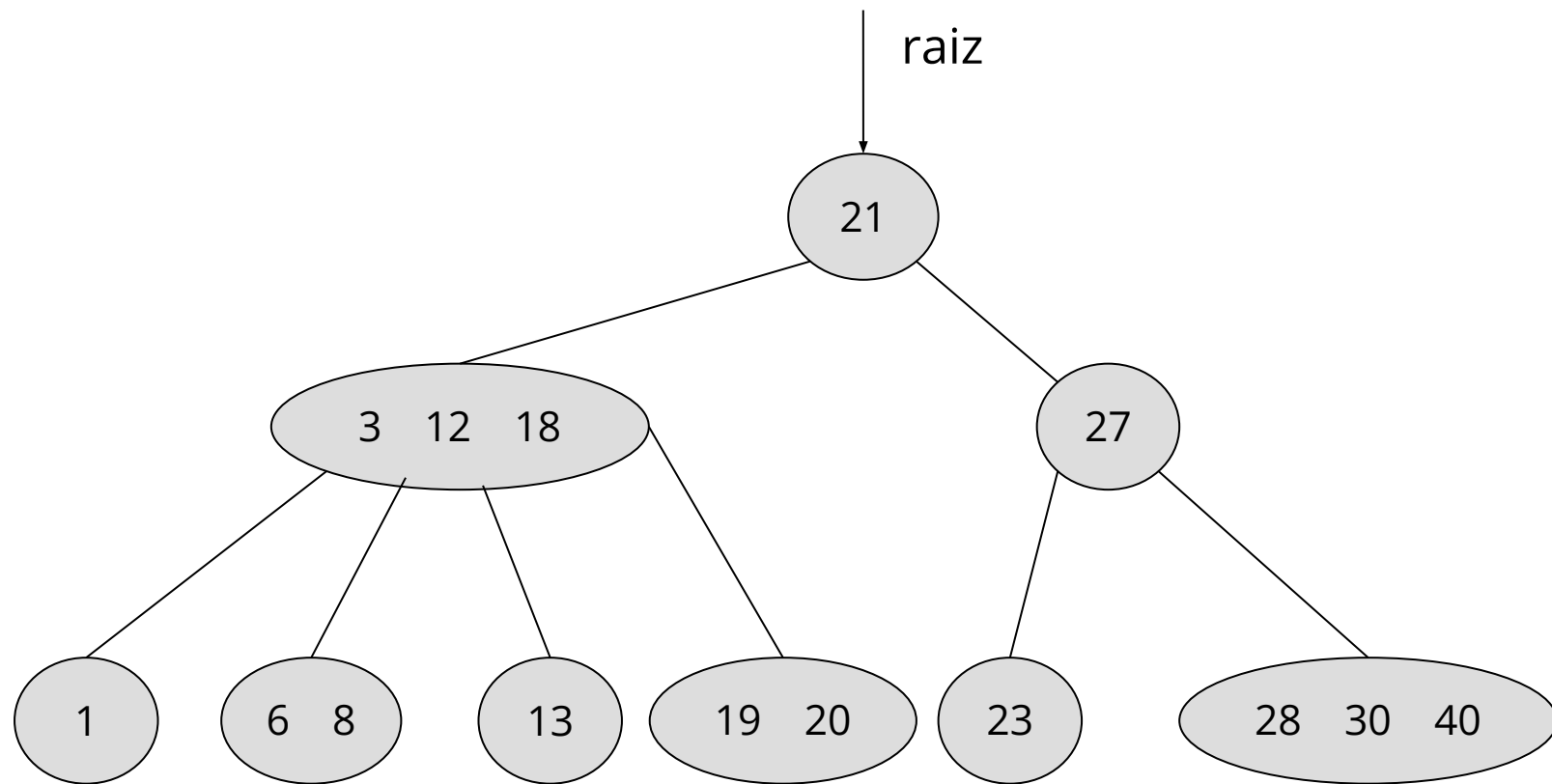
Árvore 2-3-4

Em uma árvore 2-3-4, também chamada de árvore 2-4, cada nó interno pode ter dois, três ou quatro filhos. Nesse caso, cada nó interno armazena, respectivamente, uma, duas ou três chaves.

Todos os nós folhas estão no mesmo nível (balanceamento).

São estruturas de dados equivalentes/isométricas às árvores rubro-negras (ou seja, uma árvore 2-3-4 pode ser implementada como rubro-negra e vice-versa).

Exemplo de árvore 2-3-4



Árvores 2-3-4 - Inserção

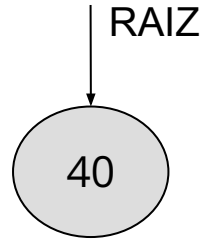
O processo de inserção em árvores 2-3-4 é similar ao de árvores 2-3:

1. Procura-se o nó folha em que será feita a inserção.
2. Caso o nó tenha uma ou duas chaves, procura-se a posição de inserção da chave e o processo se encerra.
3. Caso o nó já possua três chaves, é necessário dividi-lo para realizar a inserção. Uma das chaves irá subir para o nó pai, separando os dois novos nós. Nesse caso, o nó pai é avaliado para verificar se não deve também ser dividido.

Iremos utilizar a divisão antes da inserção, para comparação.

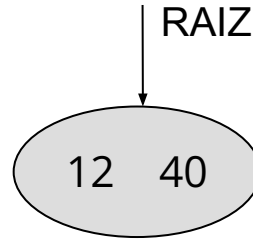
Exemplo - Inserção - 1/24

Sequência de inserção: **40** 12 68 36 38 60 48 55 50 62 65 22 90



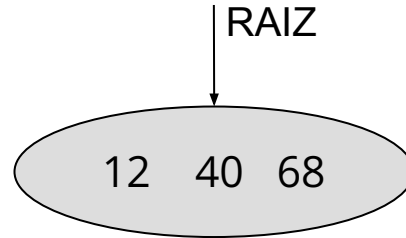
Exemplo - Inserção - 2/24

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 90



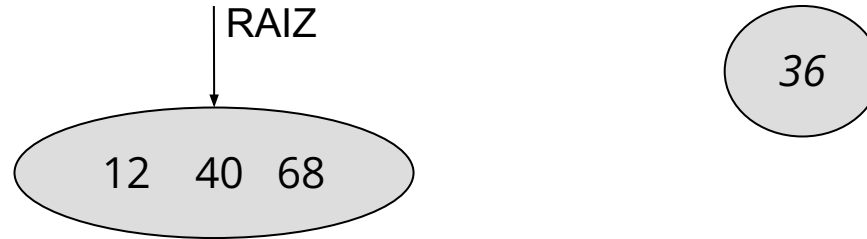
Exemplo - Inserção - 3/24

Sequência de inserção: 40 12 **68** 36 38 60 48 55 50 62 65 22 90



Exemplo - Inserção - 4/24

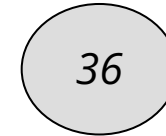
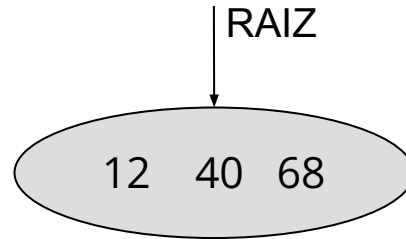
Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 90



Exemplo - Inserção - 5/24

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 90

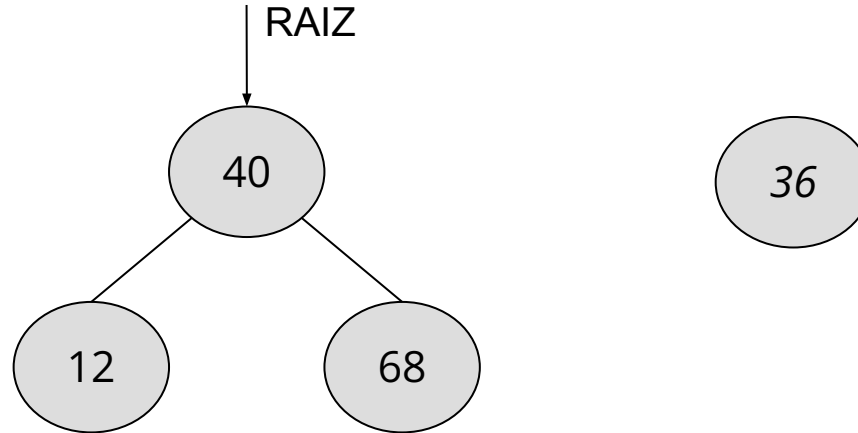
Nó está cheio, é
necessário
dividi-lo antes!



Exemplo - Inserção - 6/24

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 90

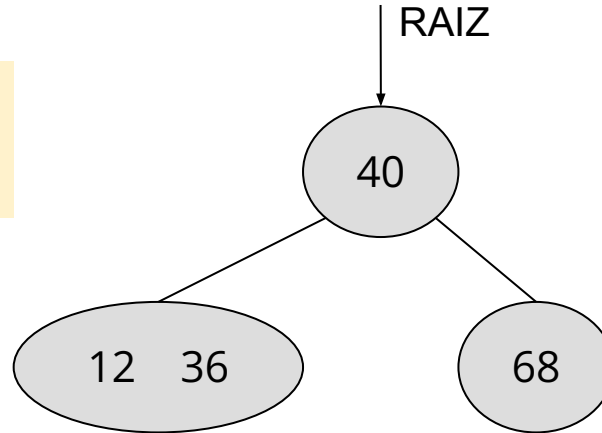
Chave central sobe
ao nó pai (caso pai
não exista, torna-se
a nova raiz).



Exemplo - Inserção - 7/24

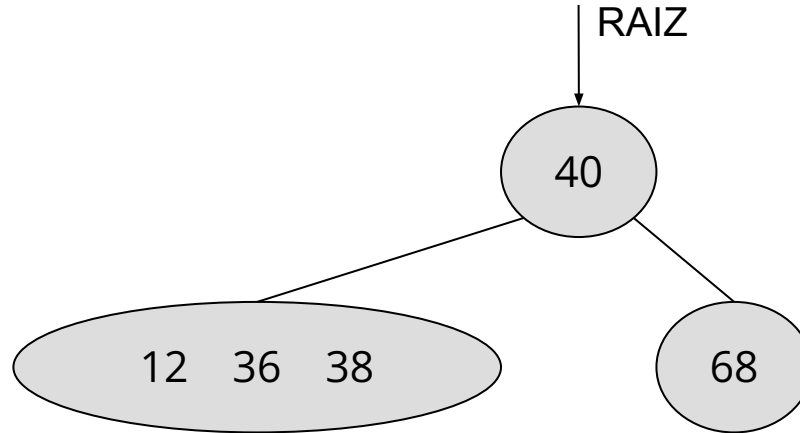
Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 90

Inserção agora é
feita normalmente



Exemplo - Inserção - 8/24

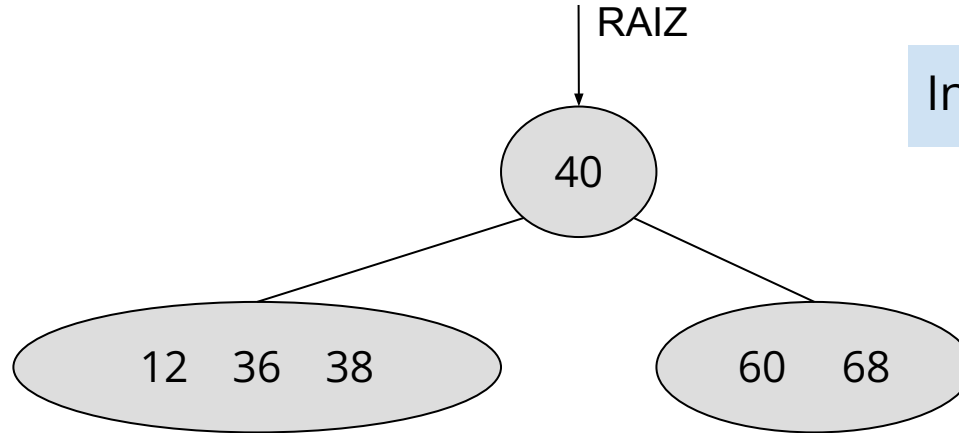
Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 90



Inserção normal

Exemplo - Inserção - 9/24

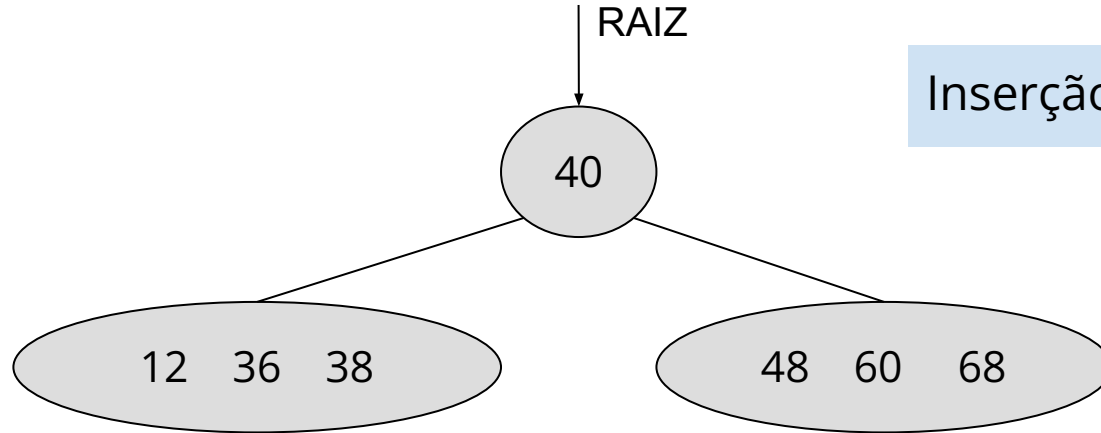
Sequência de inserção: 40 12 68 36 38 **60** 48 55 50 62 65 22 90



Inserção normal

Exemplo - Inserção - 10/24

Sequência de inserção: 40 12 68 36 38 60 **48** 55 50 62 65 22 90

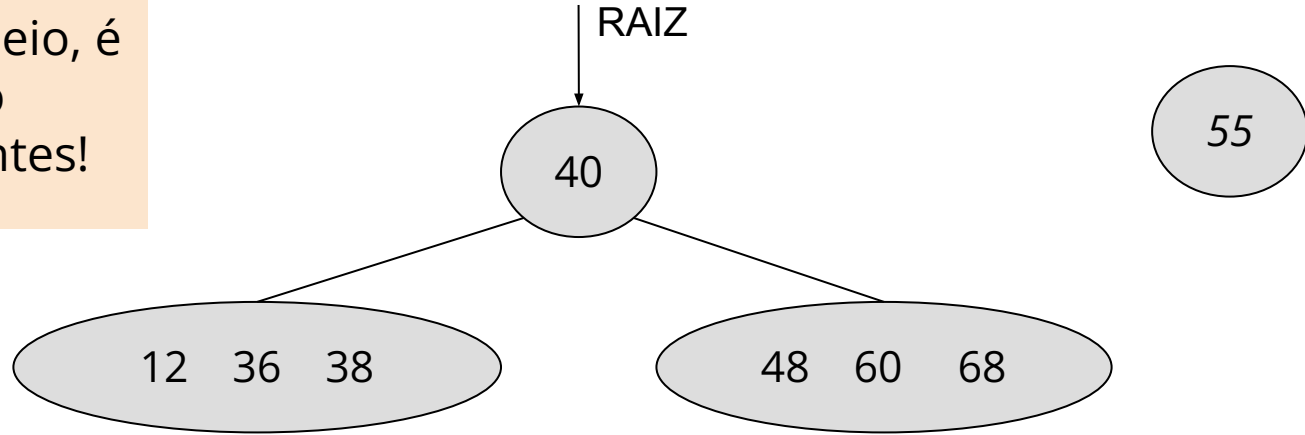


Inserção normal

Exemplo - Inserção - 11/24

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 90

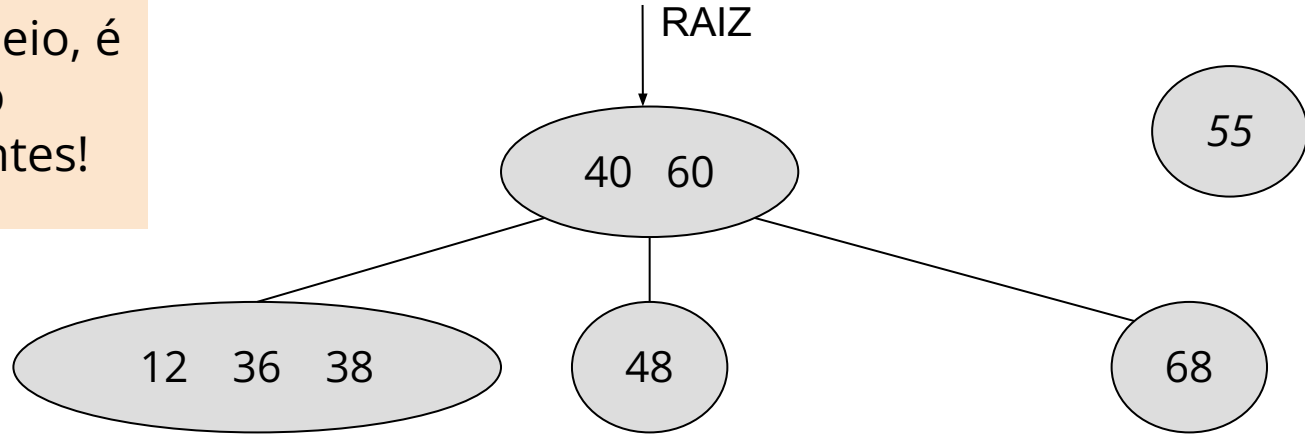
Nó está cheio, é necessário dividi-lo antes!



Exemplo - Inserção - 12/24

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 90

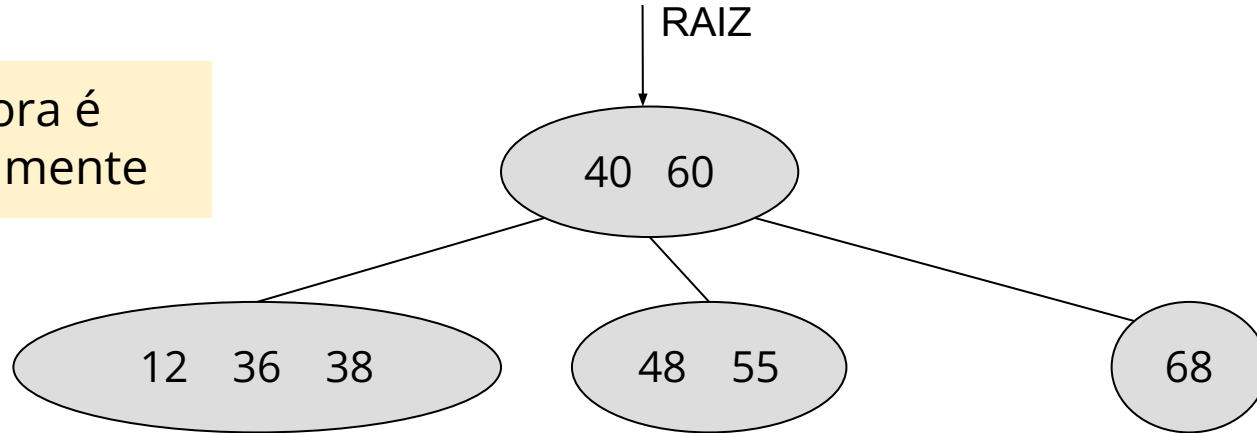
Nó está cheio, é necessário dividi-lo antes!



Exemplo - Inserção - 13/24

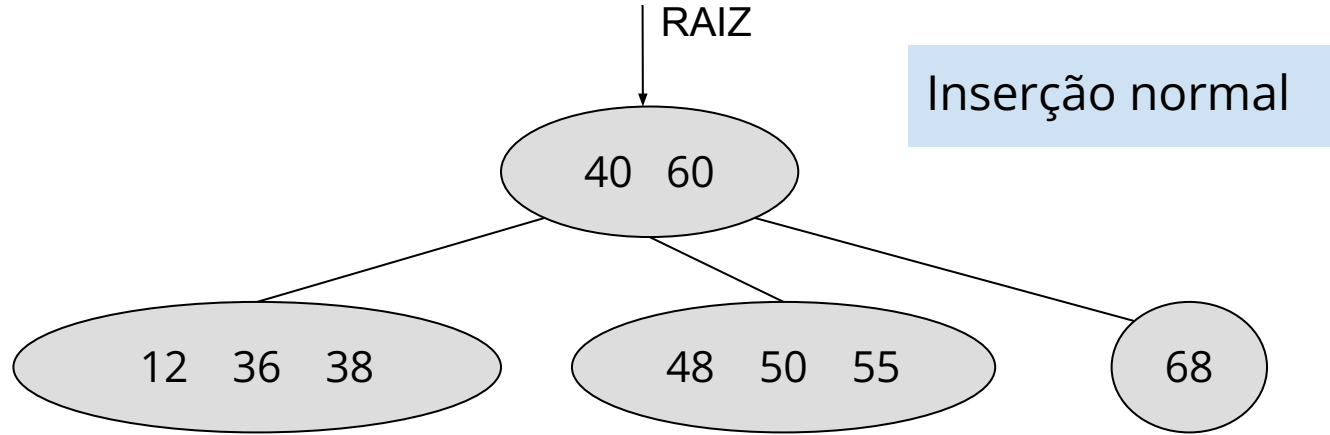
Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 90

Inserção agora é
feita normalmente



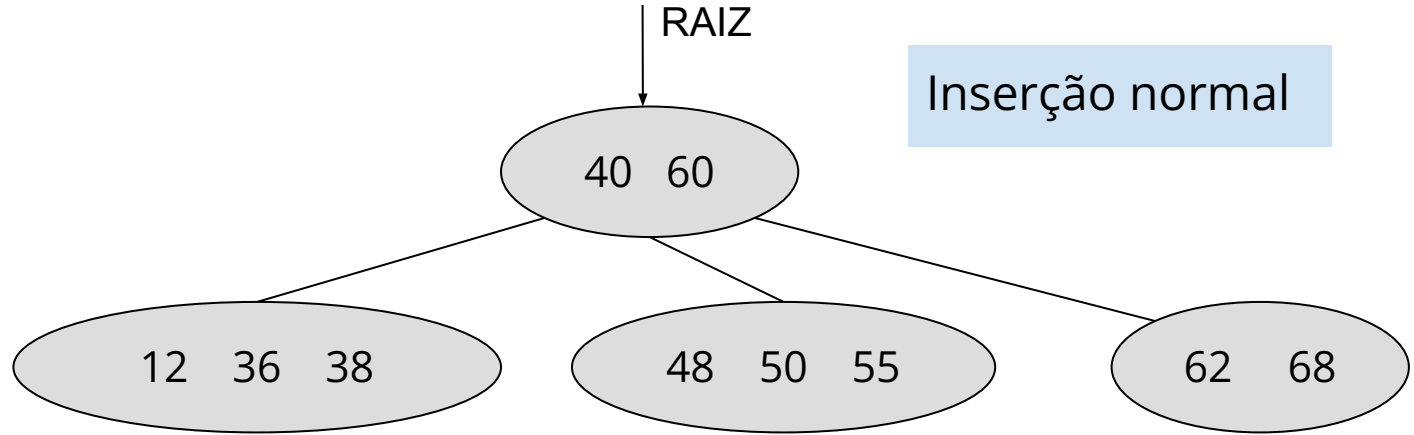
Exemplo - Inserção - 14/24

Sequência de inserção: 40 12 68 36 38 60 48 55 **50** 62 65 22 90



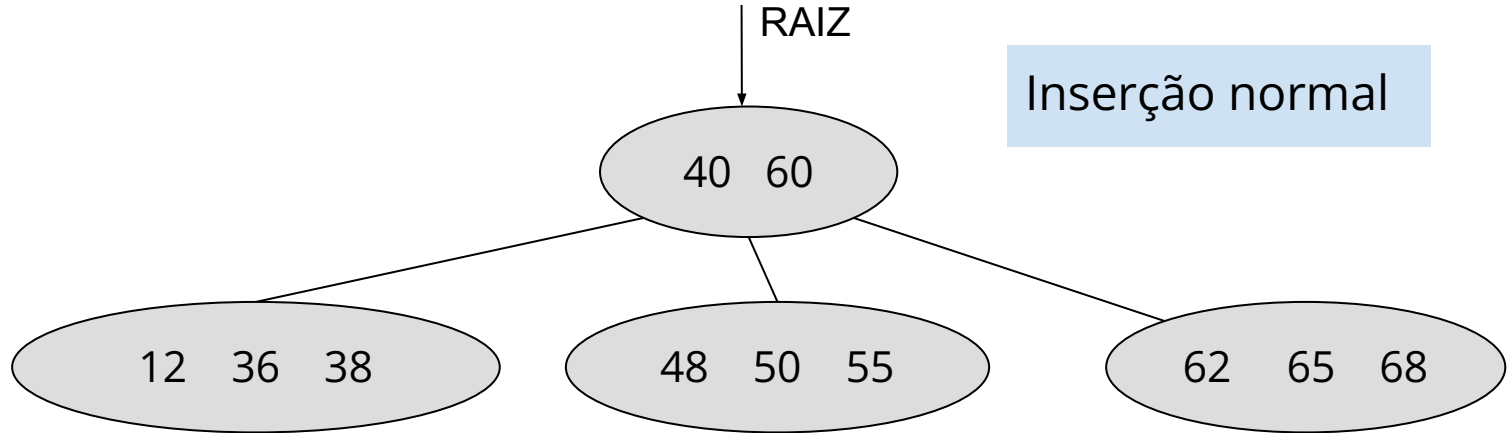
Exemplo - Inserção - 15/24

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 90



Exemplo - Inserção - 16/24

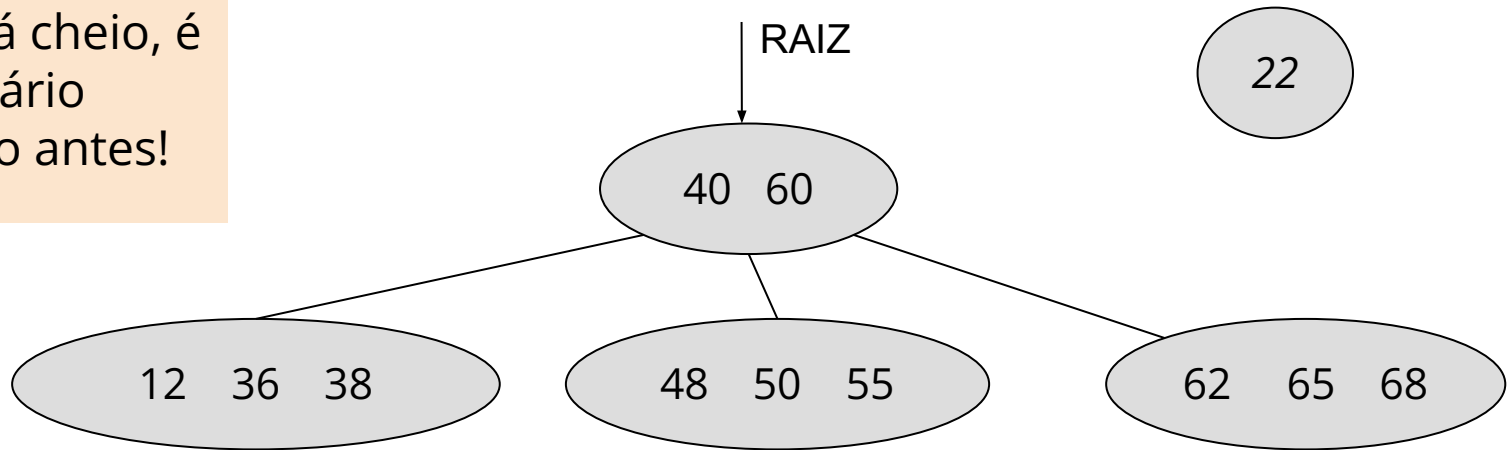
Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 90



Exemplo - Inserção - 17/24

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 90

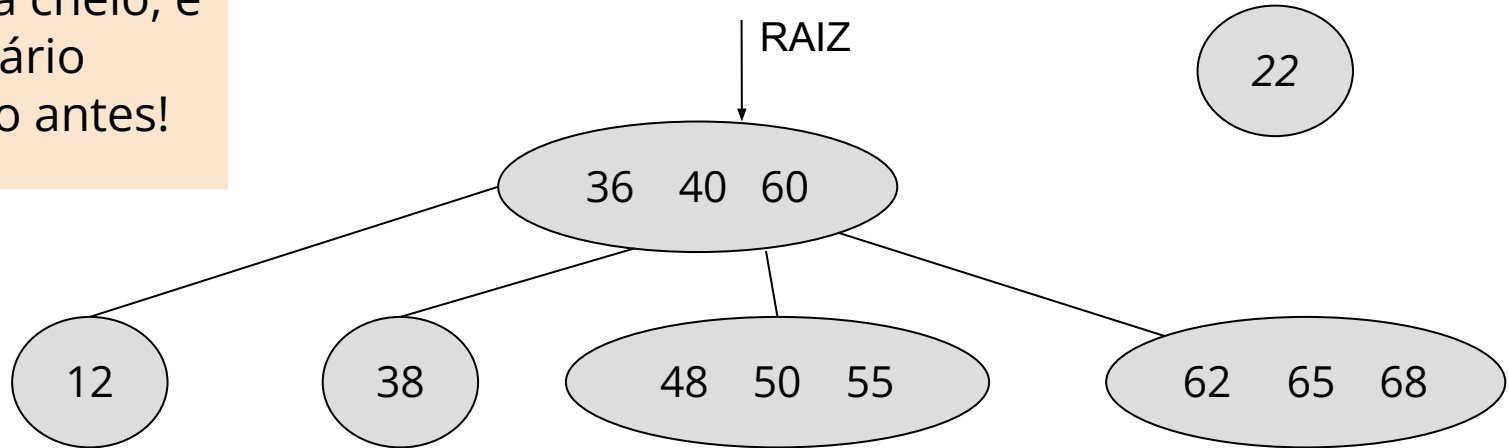
Nó está cheio, é necessário dividi-lo antes!



Exemplo - Inserção - 18/24

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 90

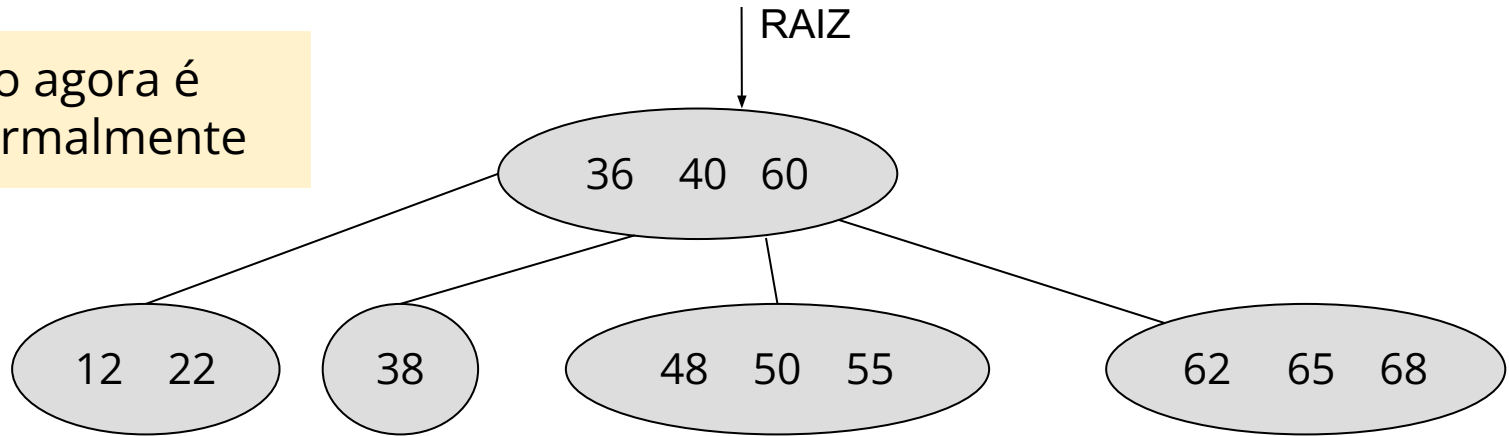
Nó está cheio, é necessário dividi-lo antes!



Exemplo - Inserção - 19/24

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 90

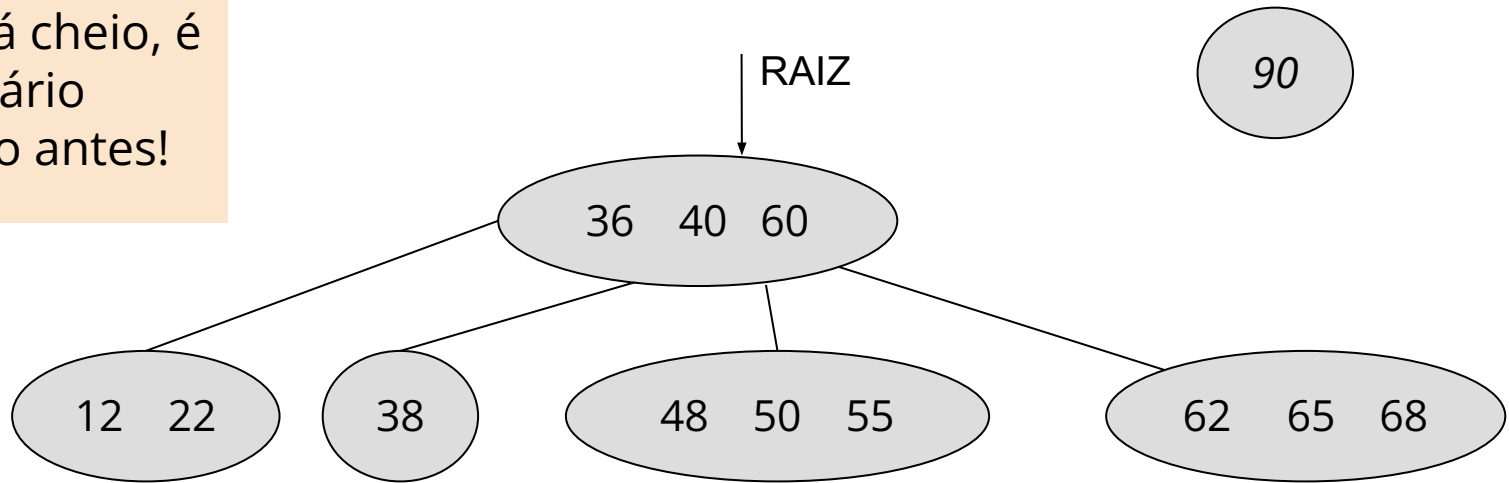
Inserção agora é
feita normalmente



Exemplo - Inserção - 20/24

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 **90**

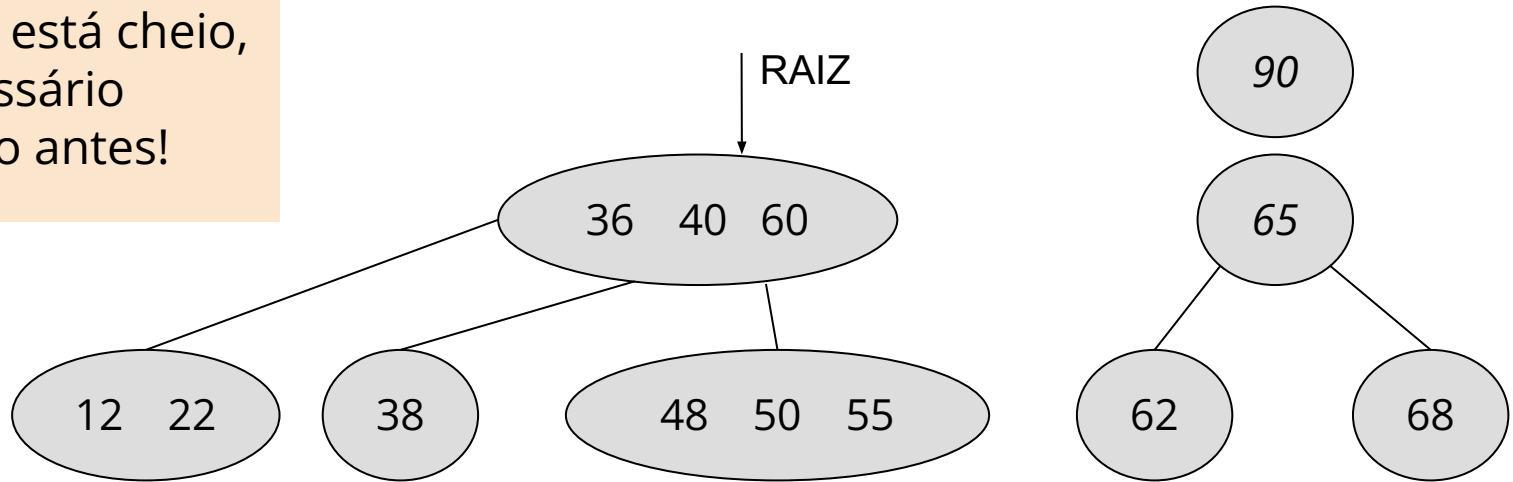
Nó está cheio, é necessário dividi-lo antes!



Exemplo - Inserção - 21/24

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 **90**

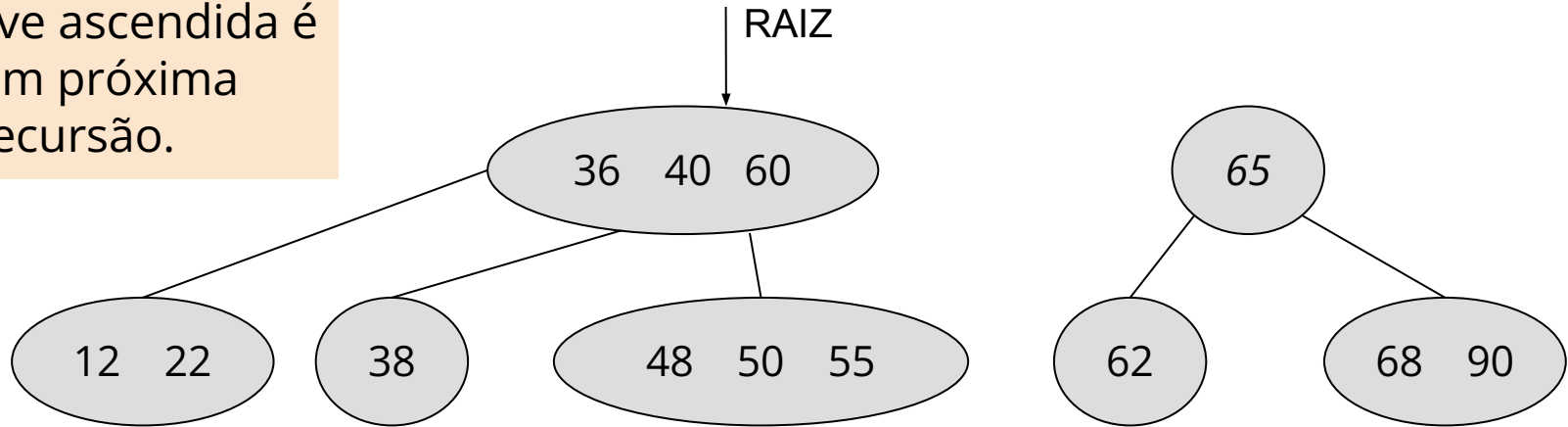
Nó **pai** está cheio,
é necessário
dividi-lo antes!



Exemplo - Inserção - 22/24

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 **90**

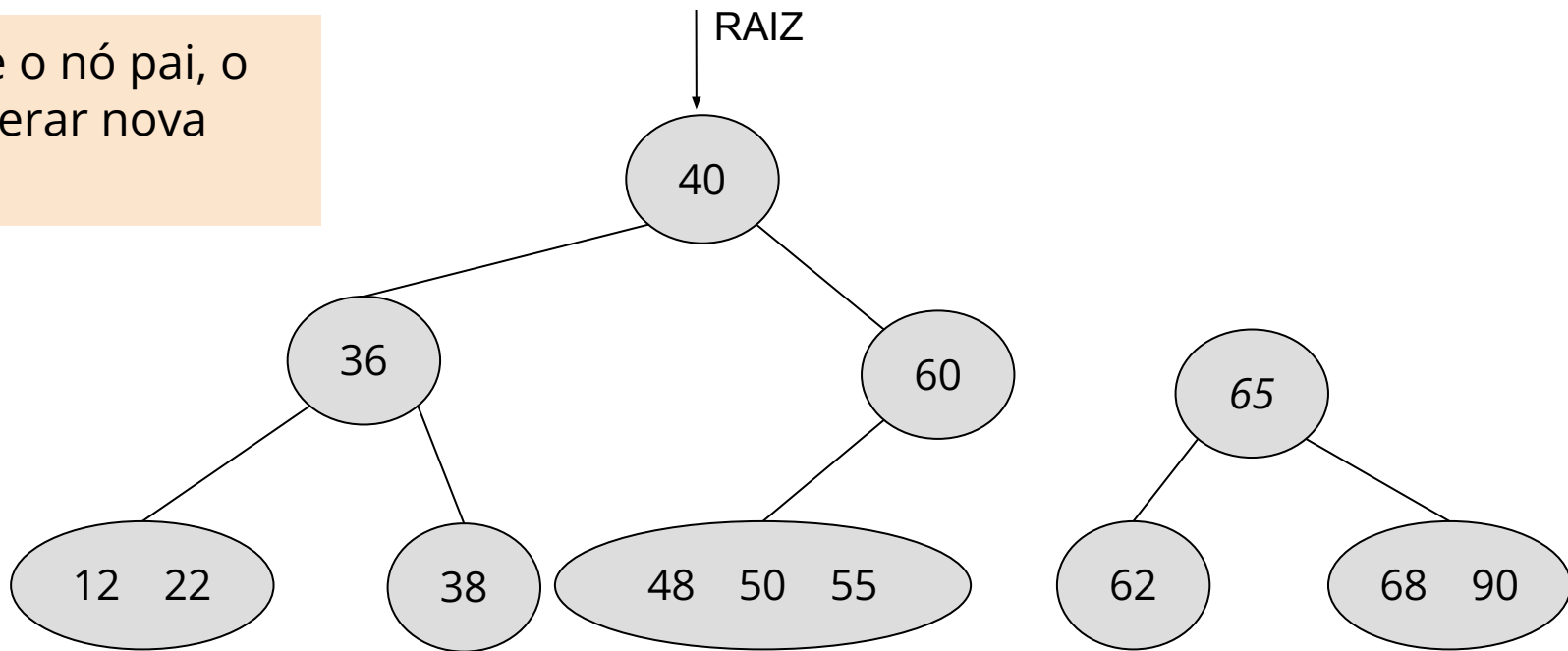
Efetuamos a inserção do 90 e a chave ascendida é ajustada em próxima iteração/recursão.



Exemplo - Inserção - 23/24

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 **90**

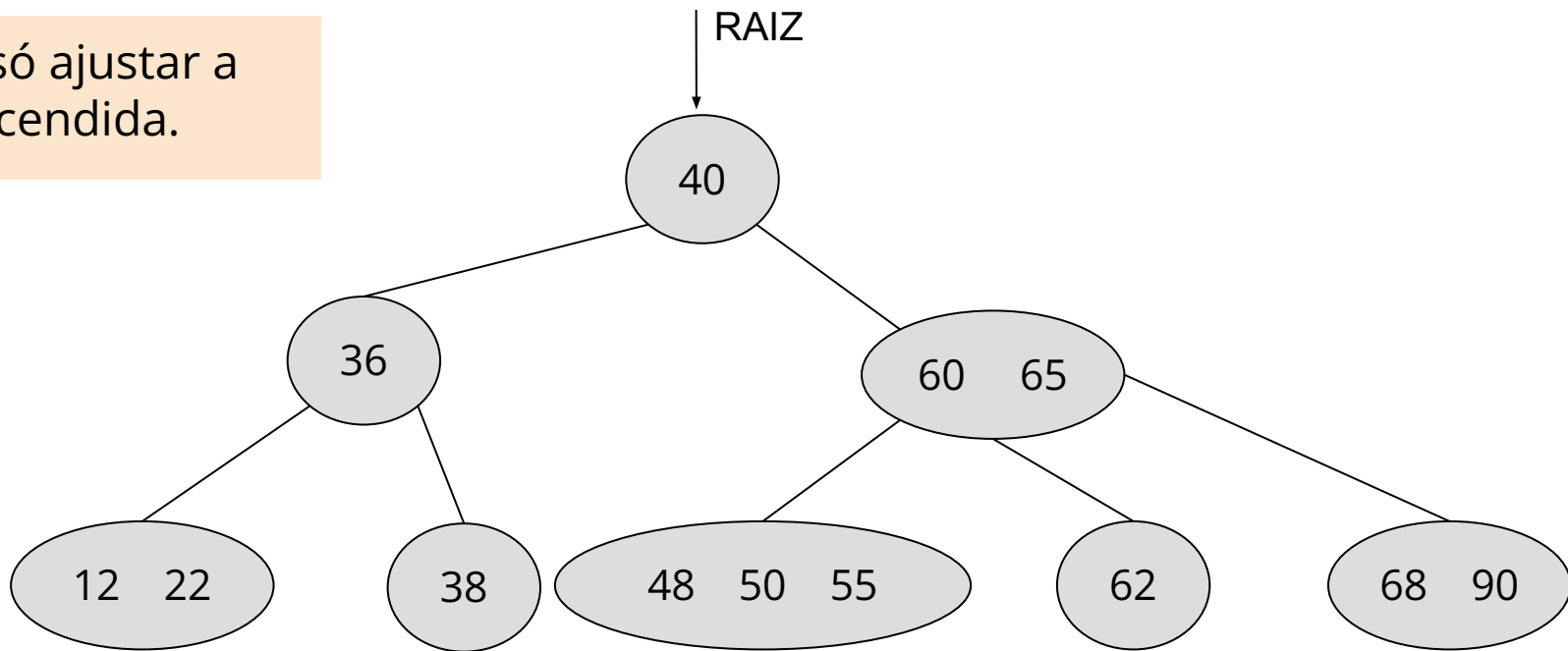
Divide-se o nó pai, o que irá gerar nova raiz.



Exemplo - Inserção - 24/24

Sequência de inserção: 40 12 68 36 38 60 48 55 50 62 65 22 **90**

Agora é só ajustar a chave ascendida.



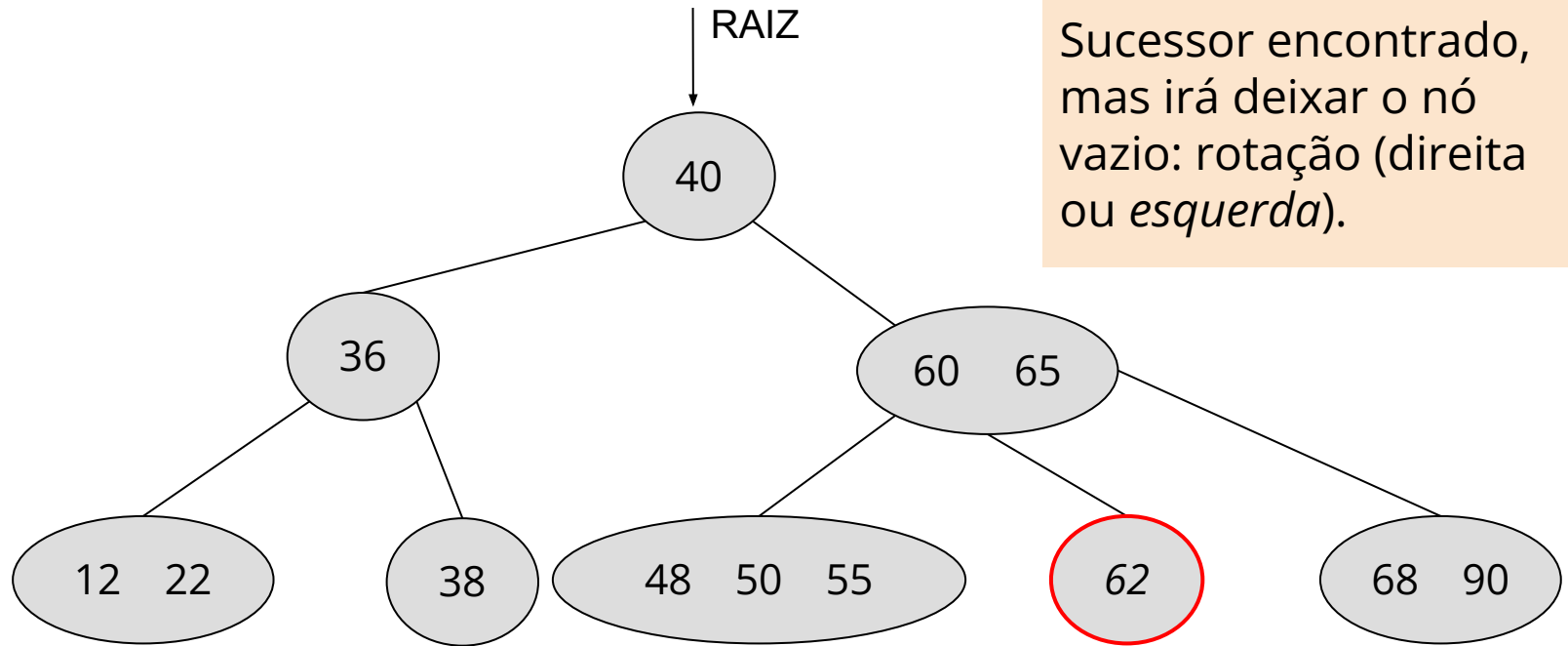
Árvore 2-3-4 - Remoção

O processo de remoção é muito similar ao de árvore 2-3, :

1. A remoção é sempre feita em nós folhas, caso uma chave em um nó interno precise ser removida, ela é substituída por sua *sucessora* ou antecessora.
2. Caso a remoção na folha (da chave ou sua sucessora/antecessora) ocorra em um nó com duas ou três chaves, o processo se encerra, caso contrário será necessário rotacionar chaves ou fundir nós.

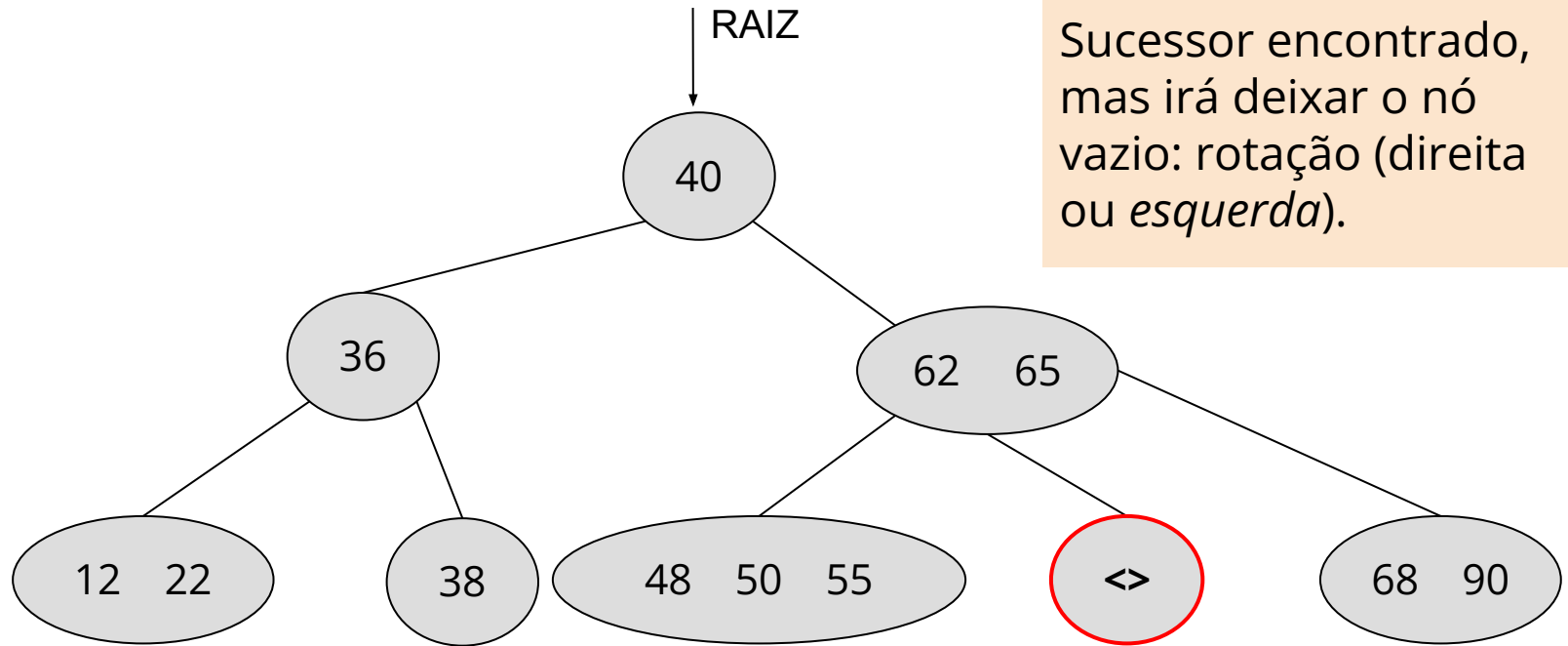
Exemplo - Remoção - 1/27

Sequência de remoção: ~~60~~ 36 38 40 12 55 62



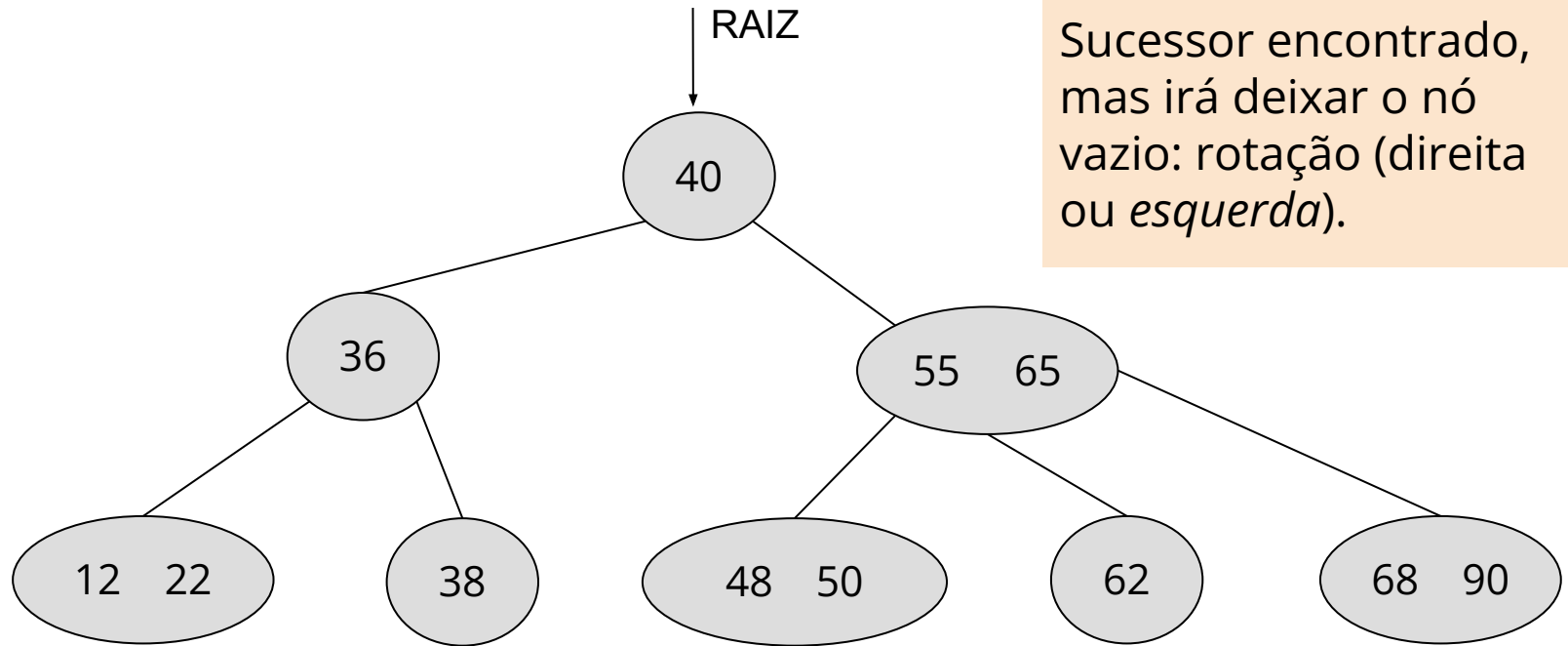
Exemplo - Remoção - 2/27

Sequência de remoção: ~~60~~ 36 38 40 12 55 62



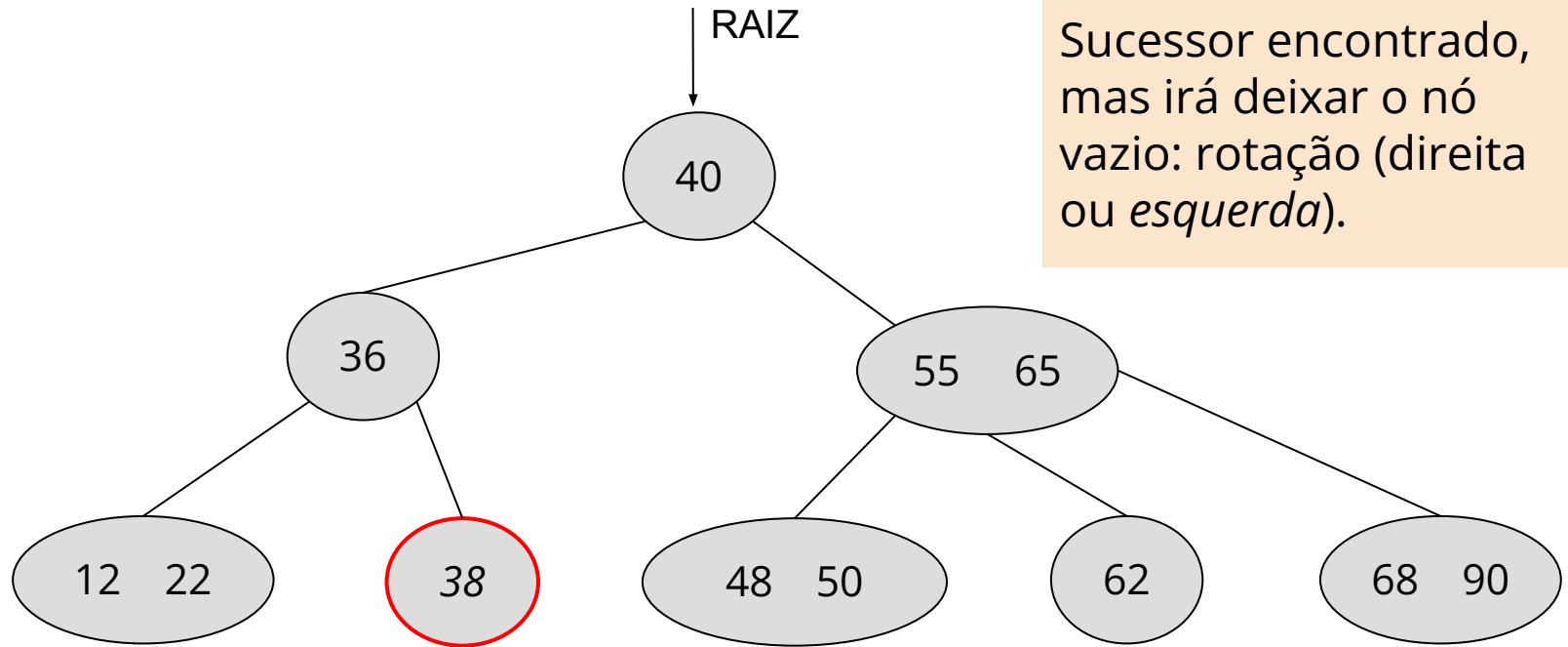
Exemplo - Remoção - 3/27

Sequência de remoção: ~~60~~ 36 38 40 12 55 62



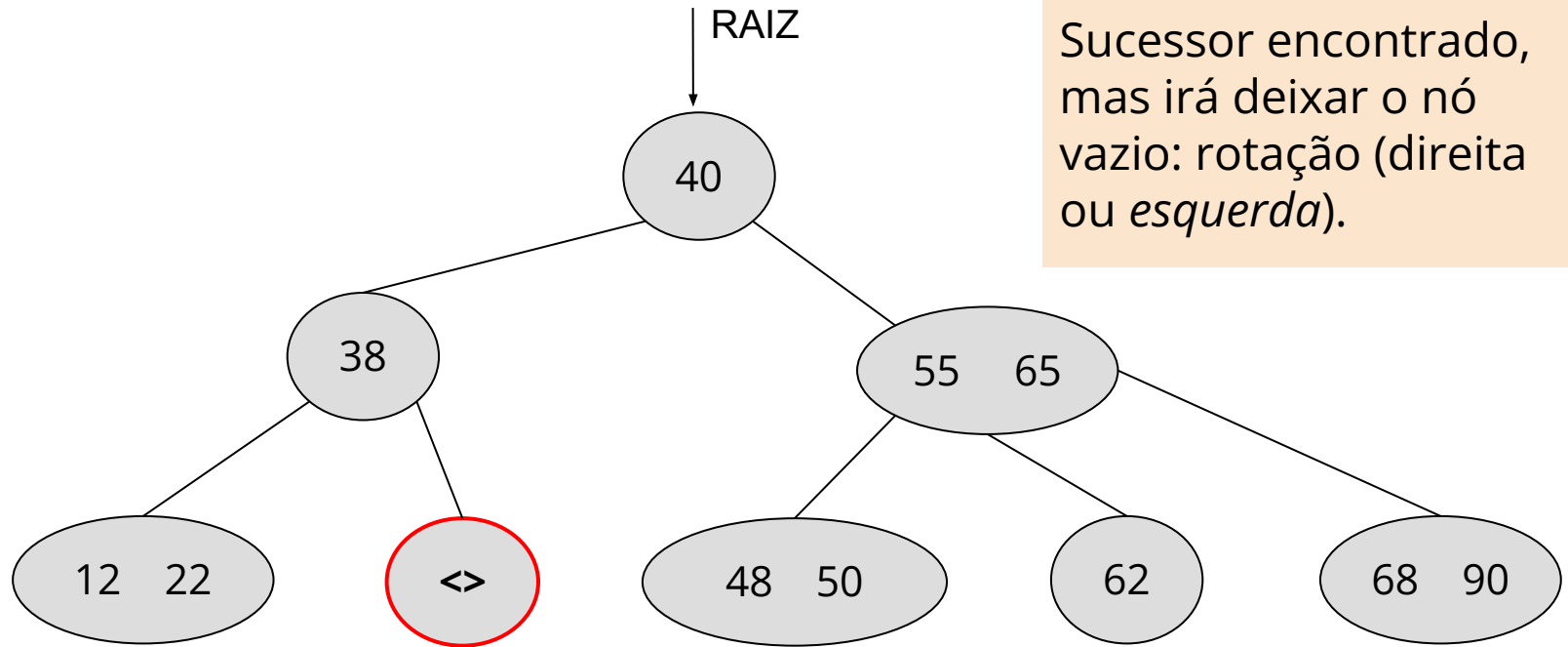
Exemplo - Remoção - 4/27

Sequência de remoção: 60 36 38 40 12 55 62



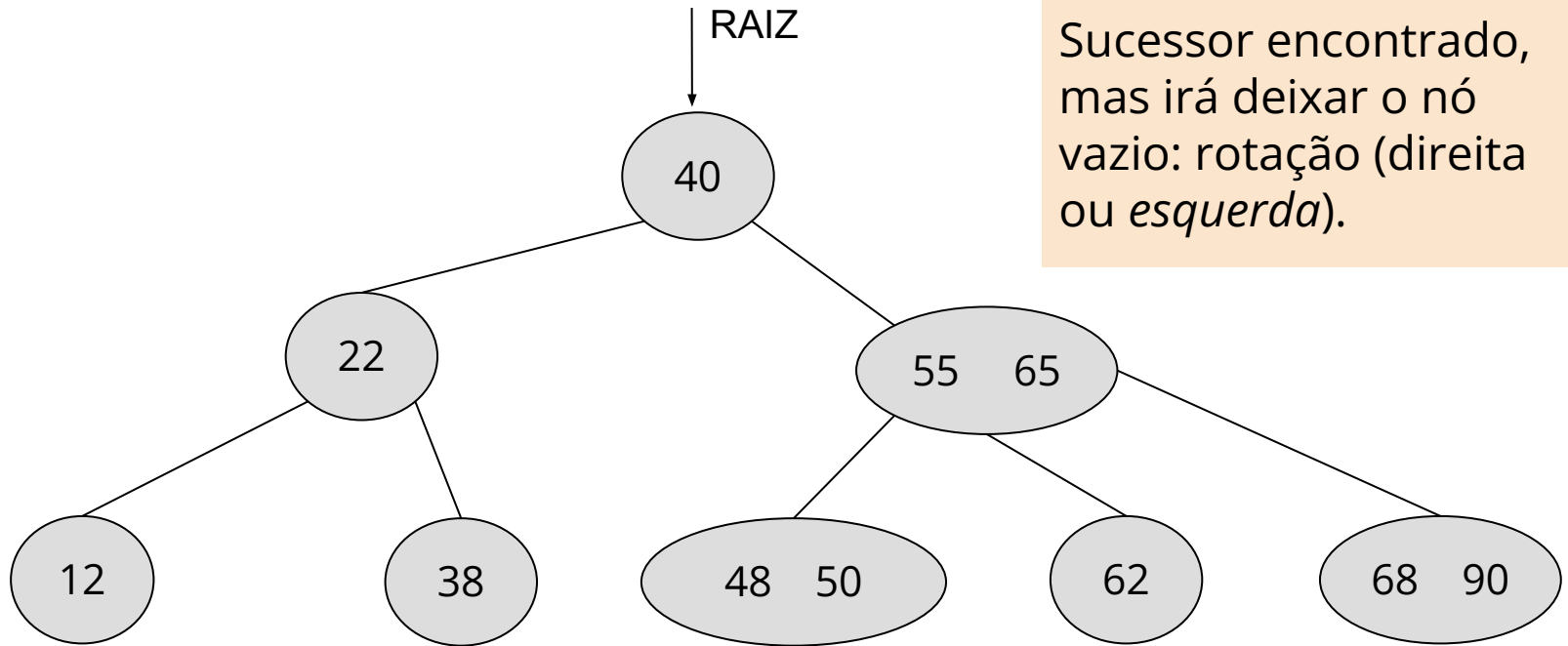
Exemplo - Remoção - 5/27

Sequência de remoção: 60 36 38 40 12 55 62



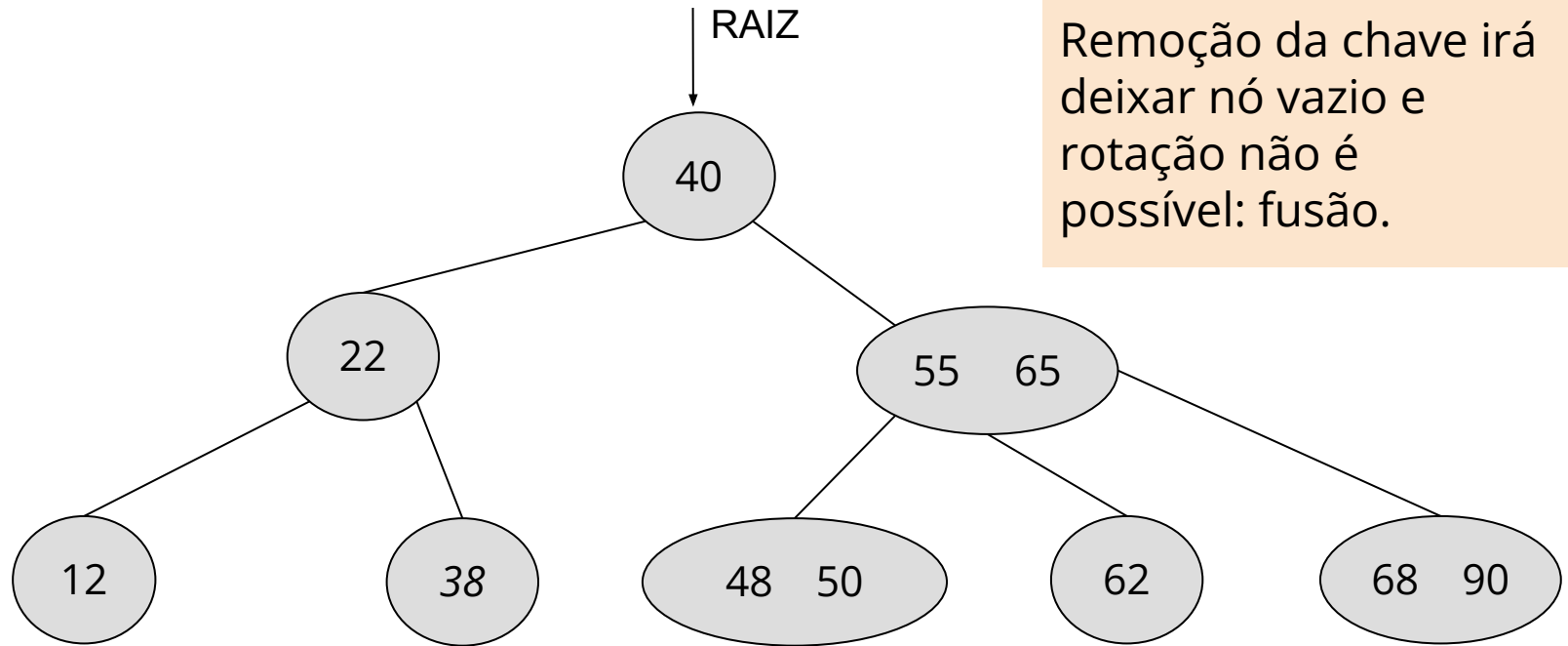
Exemplo - Remoção - 6/27

Sequência de remoção: 60 36 38 40 12 55 62



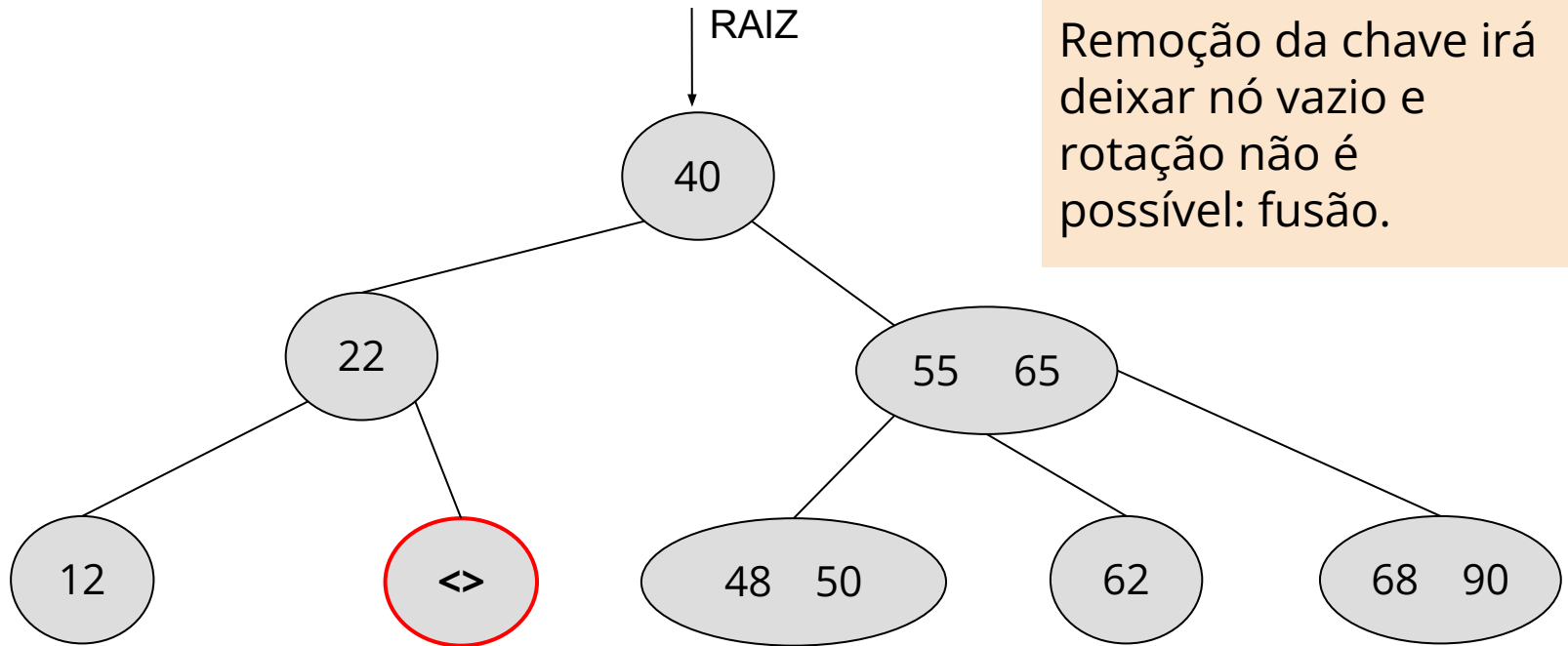
Exemplo - Remoção - 7/27

Sequência de remoção: 60 36 38 40 12 55 62



Exemplo - Remoção - 8/27

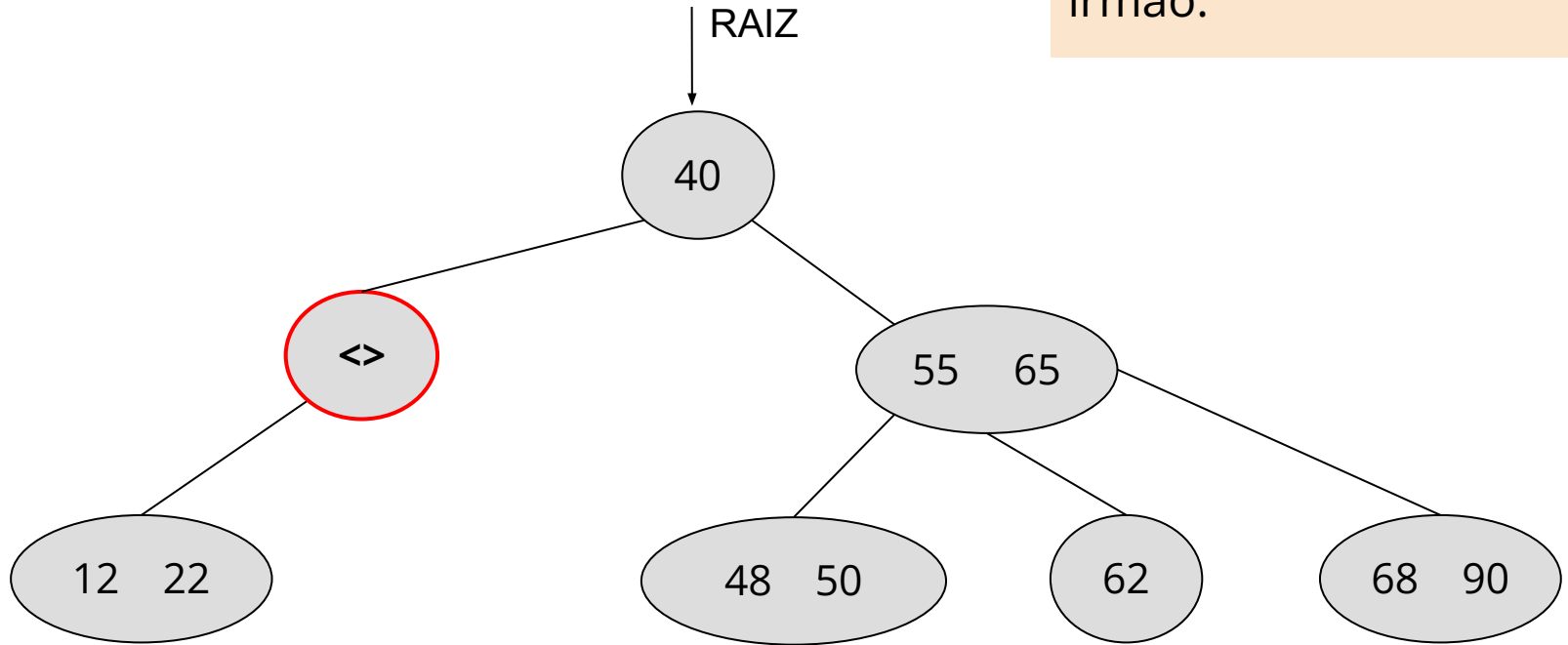
Sequência de remoção: 60 36 38 40 12 55 62



Exemplo - Remoção - 9/27

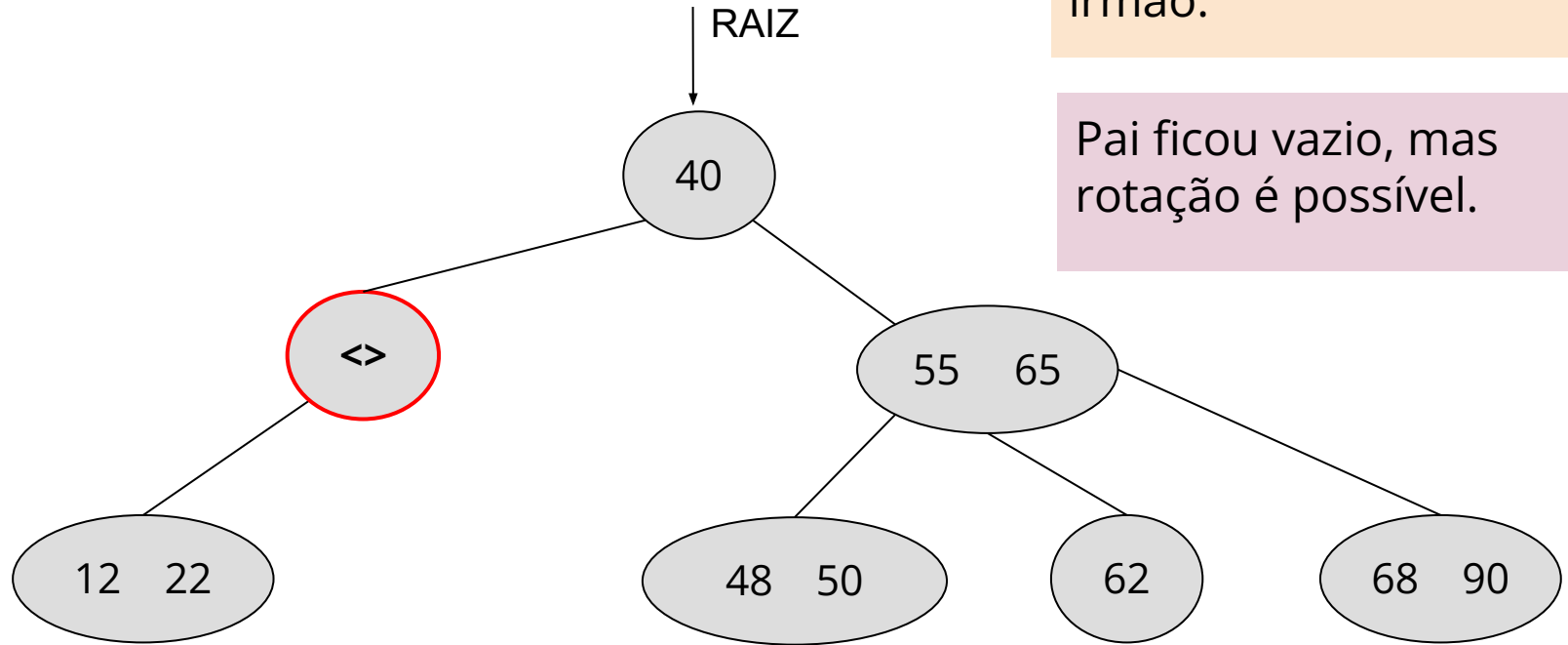
Sequência de remoção: 60 36 38 40 12 55

Fusão efetuada (chave do pai que separava nó do irmão desce ao irmão).



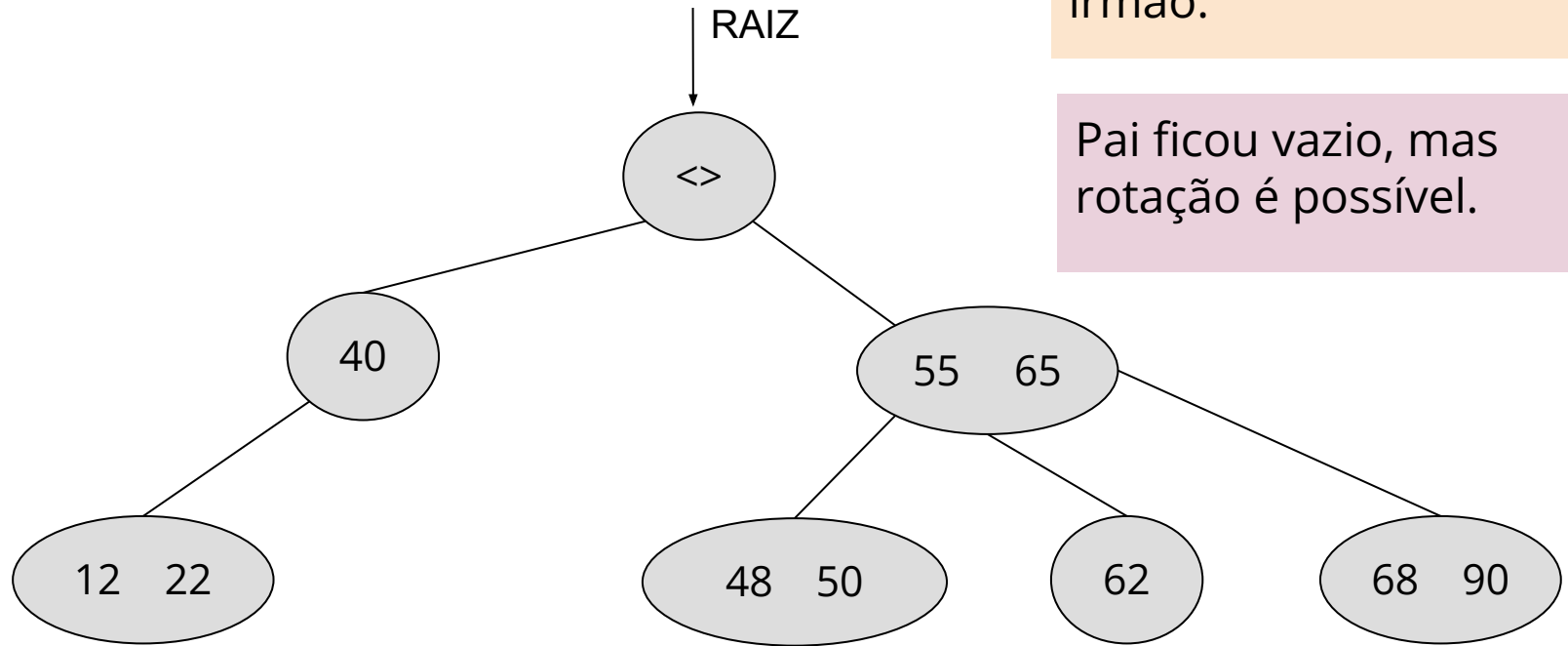
Exemplo - Remoção - 10/27

Sequência de remoção: 60 36 38 40 12 55 62



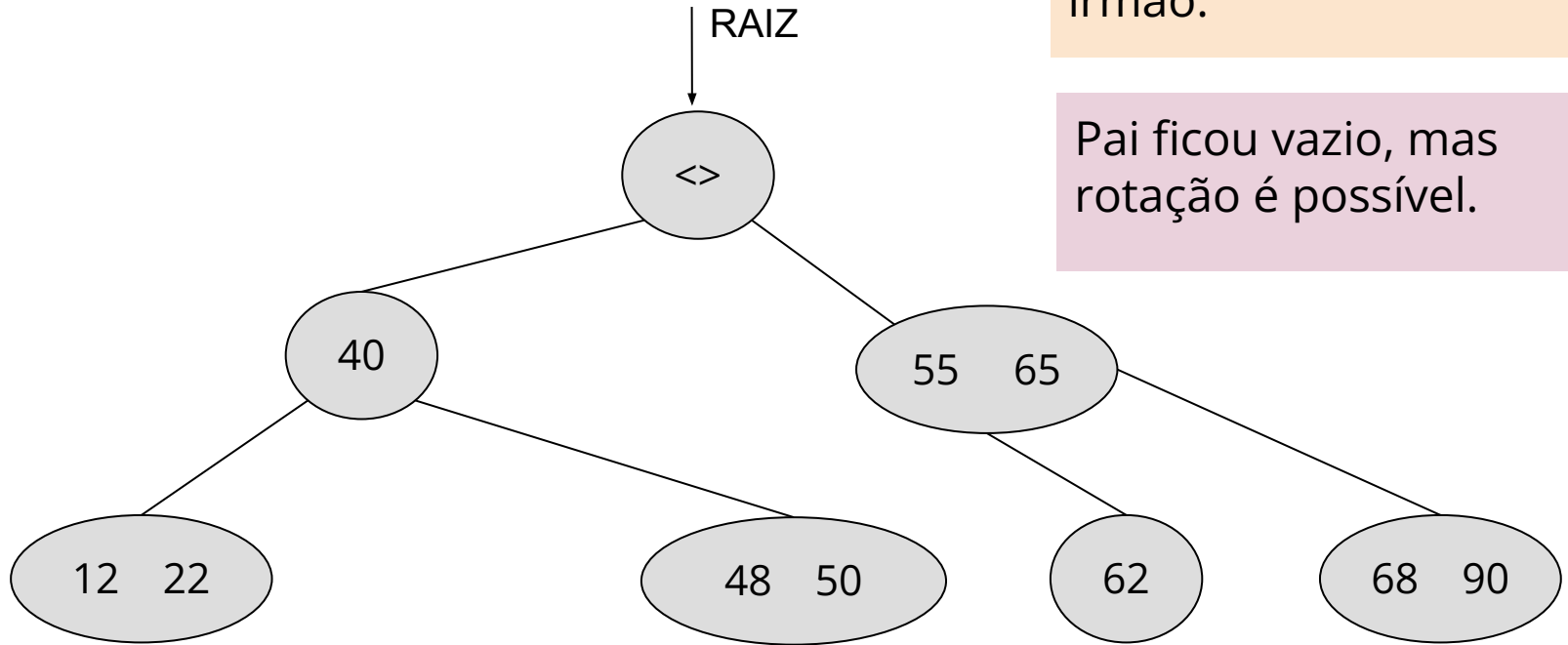
Exemplo - Remoção - 11/27

Sequência de remoção: 60 36 38 40 12 55 62



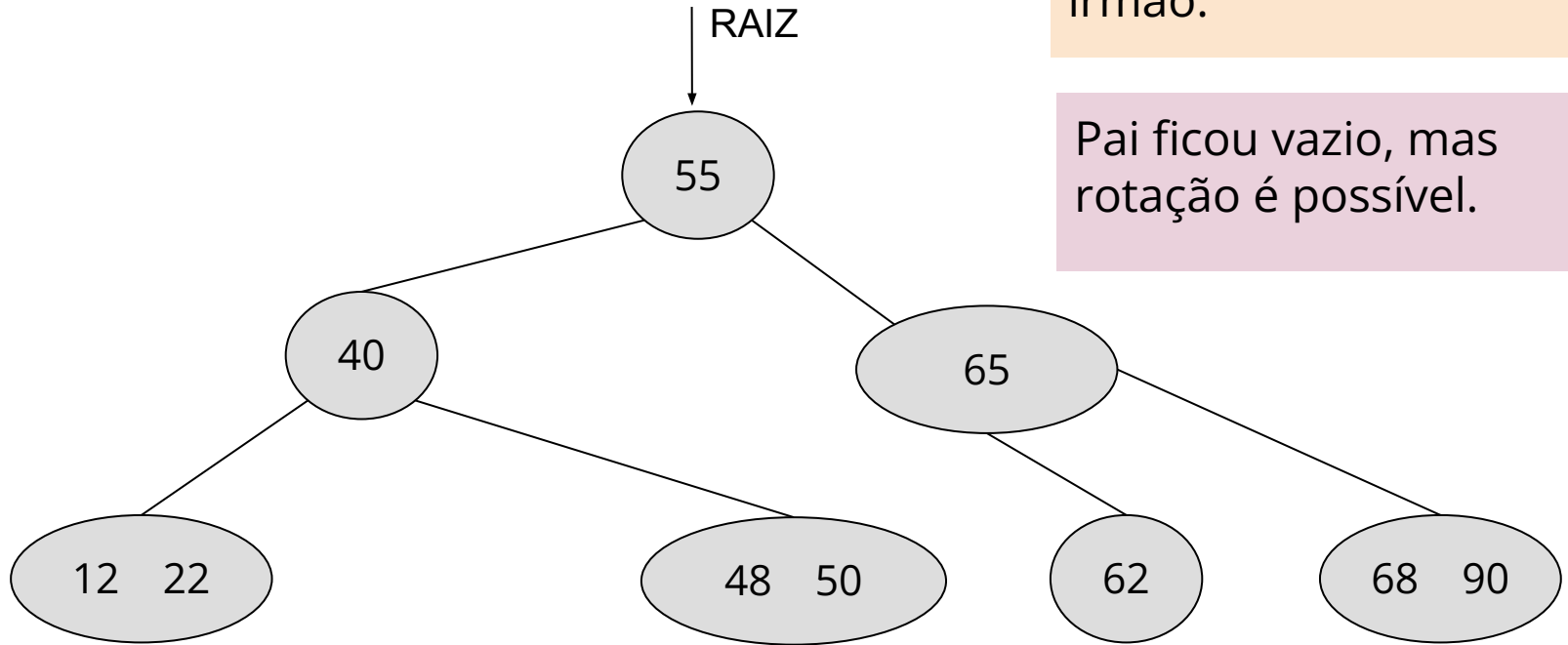
Exemplo - Remoção - 12/27

Sequência de remoção: 60 36 38 40 12 55 62



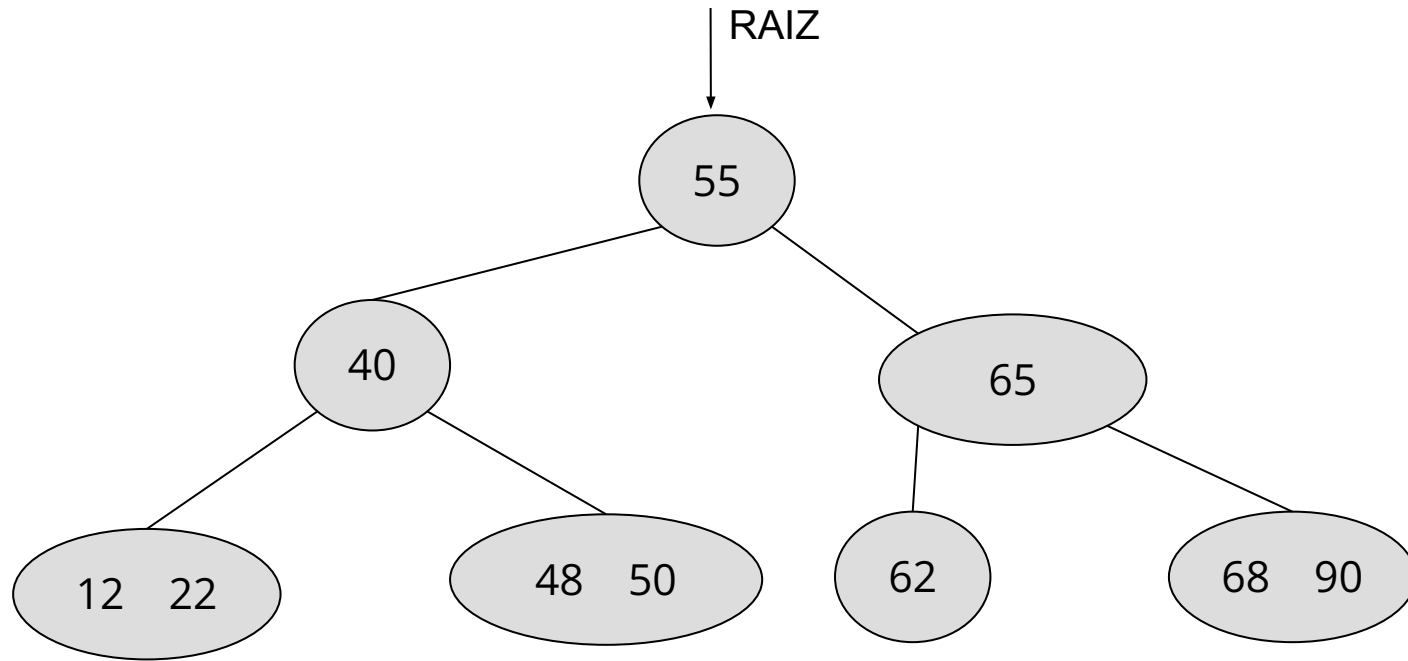
Exemplo - Remoção - 13/27

Sequência de remoção: 60 36 38 40 12 55 62



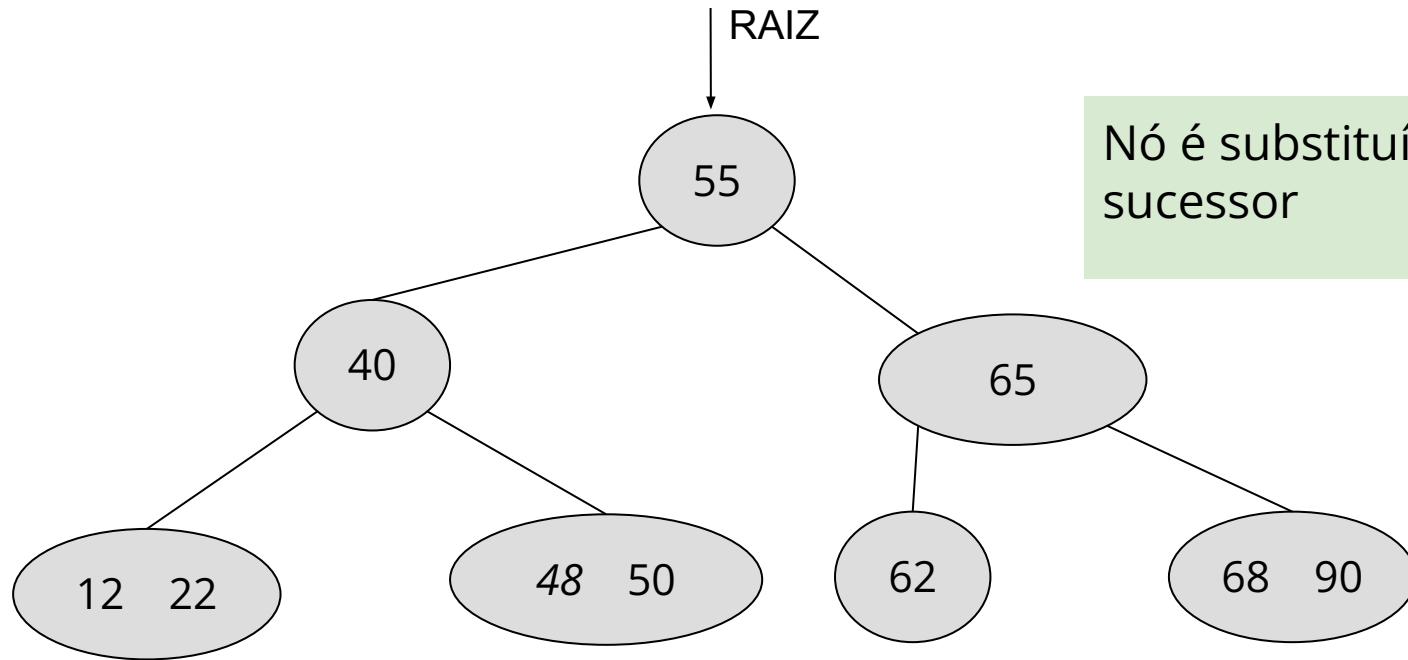
Exemplo - Remoção - 14/27

Sequência de remoção: 60 36 38 40 12 55 62



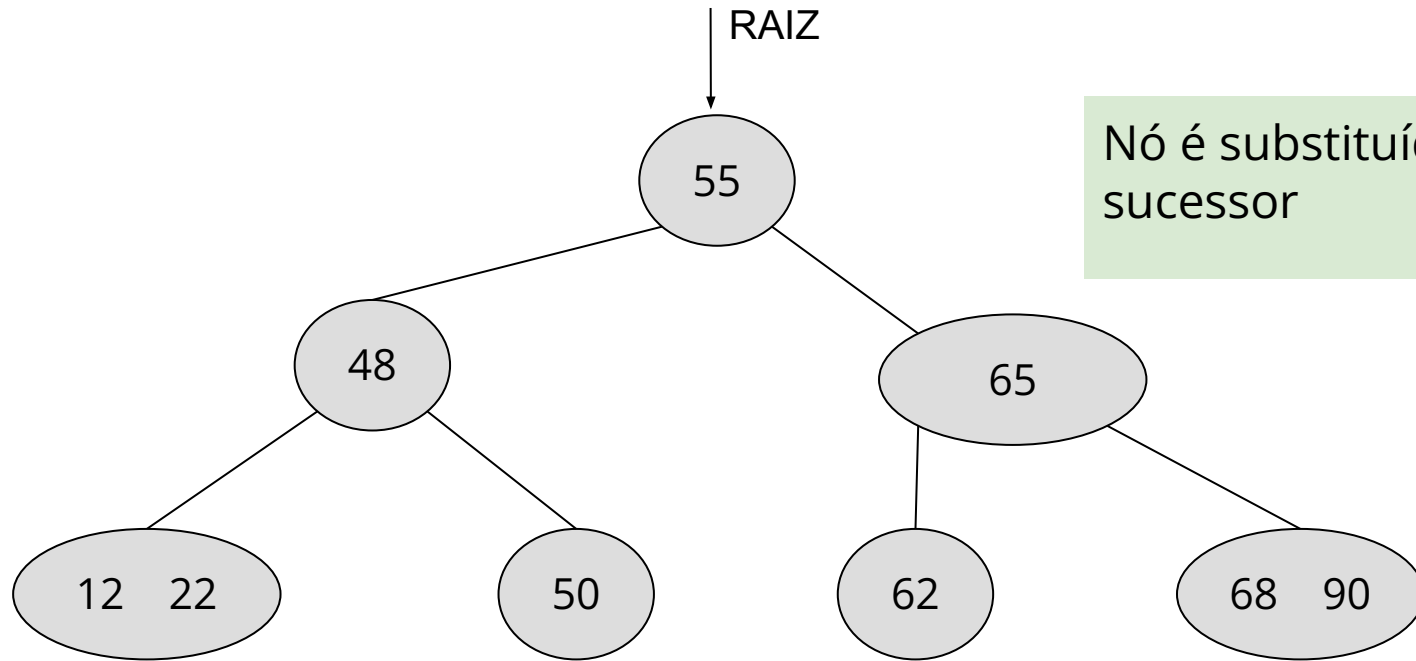
Exemplo - Remoção - 15/27

Sequência de remoção: 60 36 38 **40** 12 55 62



Exemplo - Remoção - 16/27

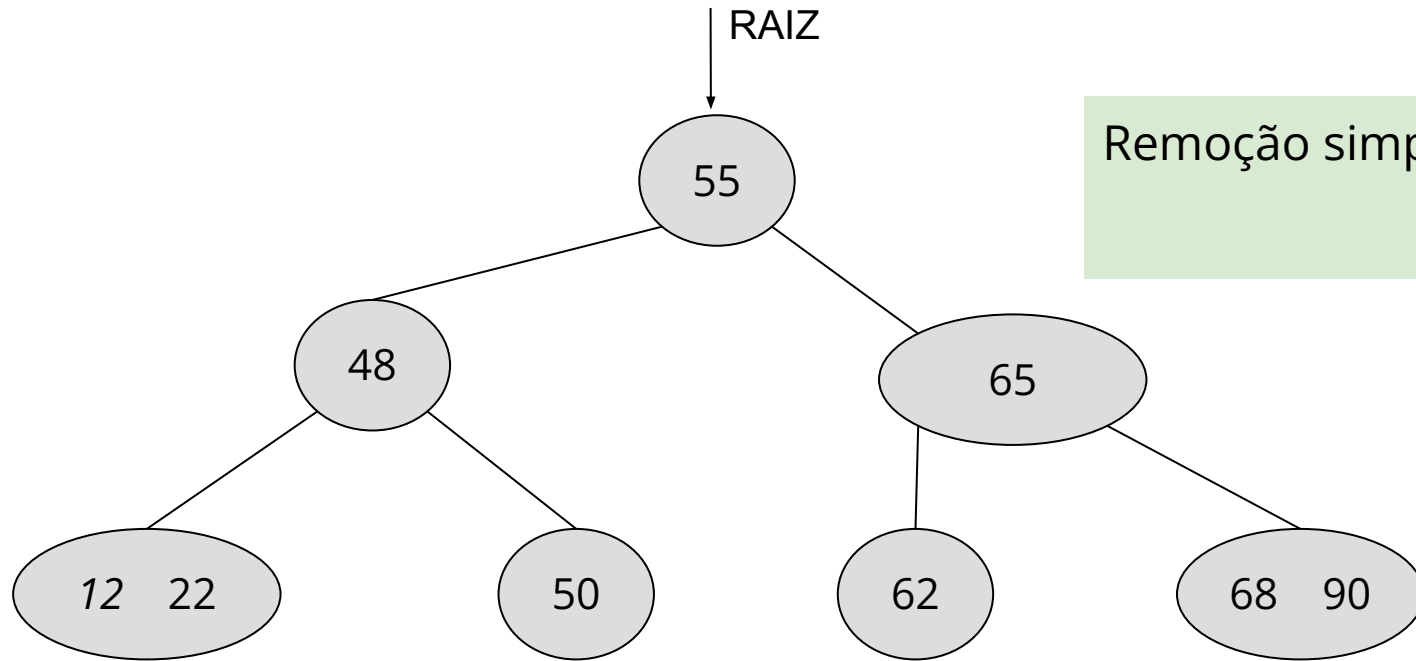
Sequência de remoção: 60 36 38 **40** 12 55 62



Nó é substituído pelo
sucessor

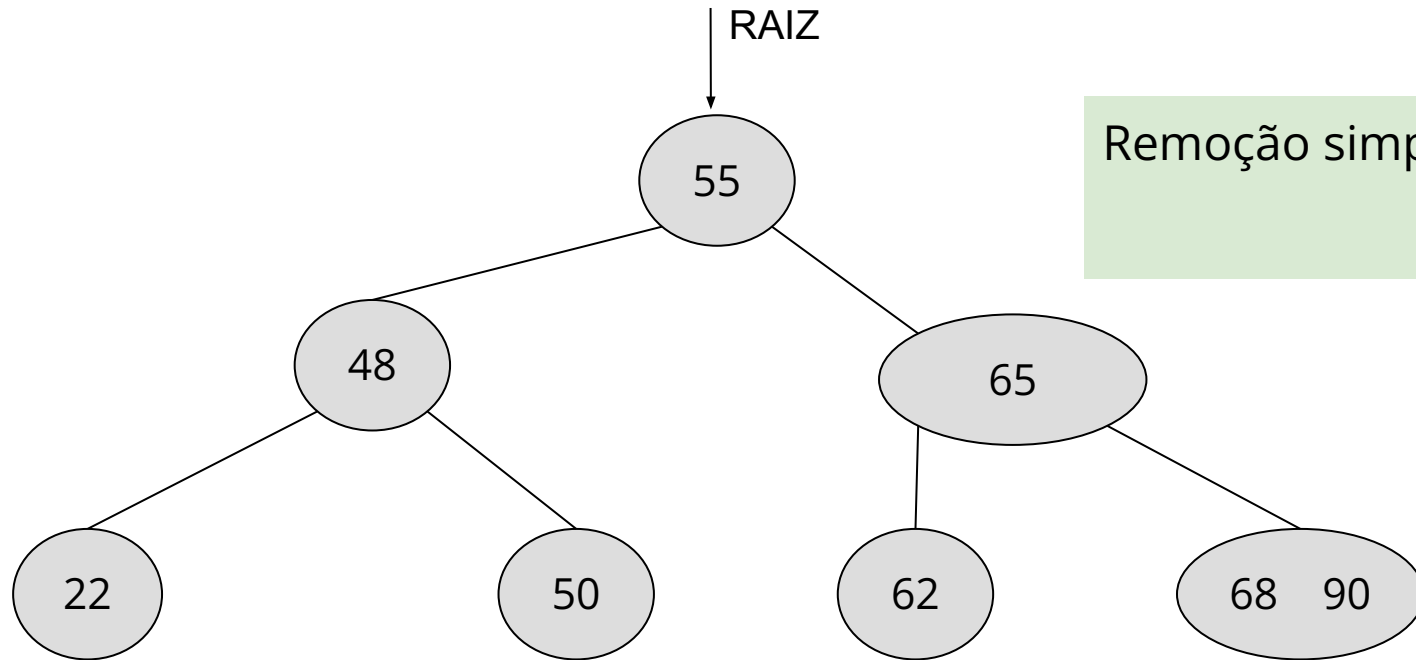
Exemplo - Remoção - 17/27

Sequência de remoção: 60 36 38 40 **12** 55 62



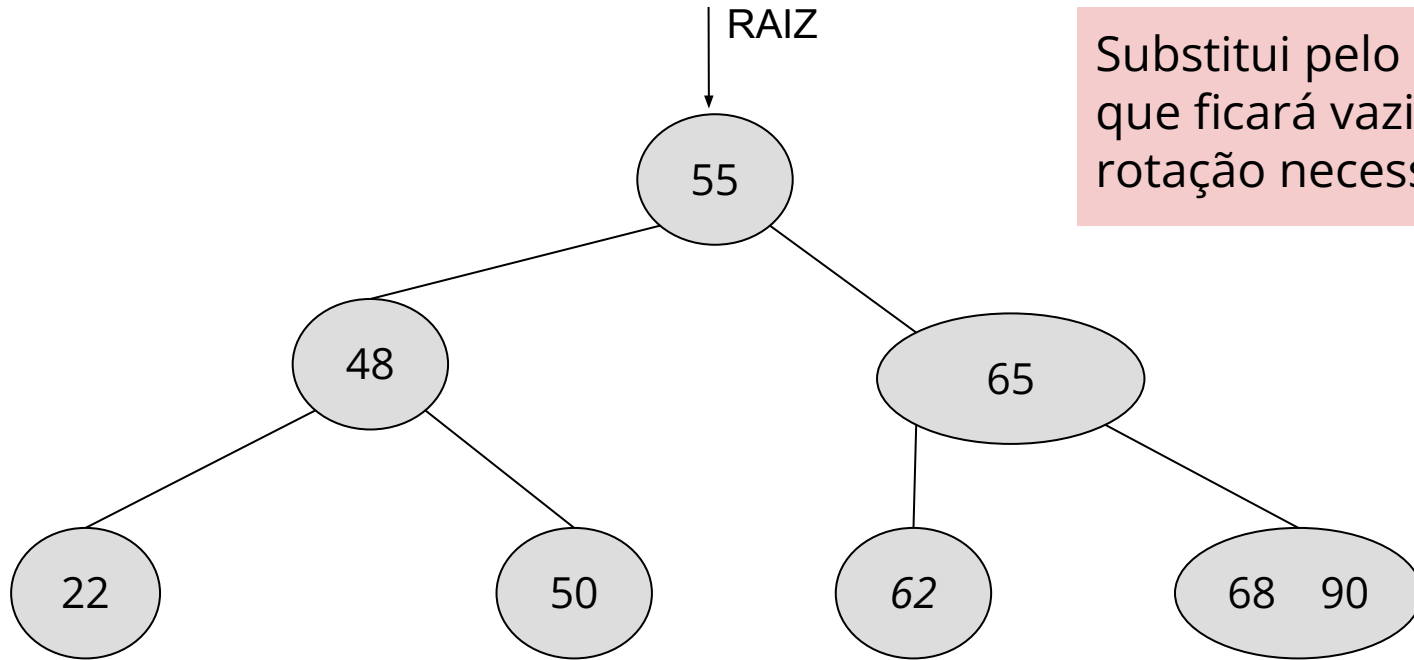
Exemplo - Remoção - 18/27

Sequência de remoção: 60 36 38 40 **12** 55 62



Exemplo - Remoção - 19/27

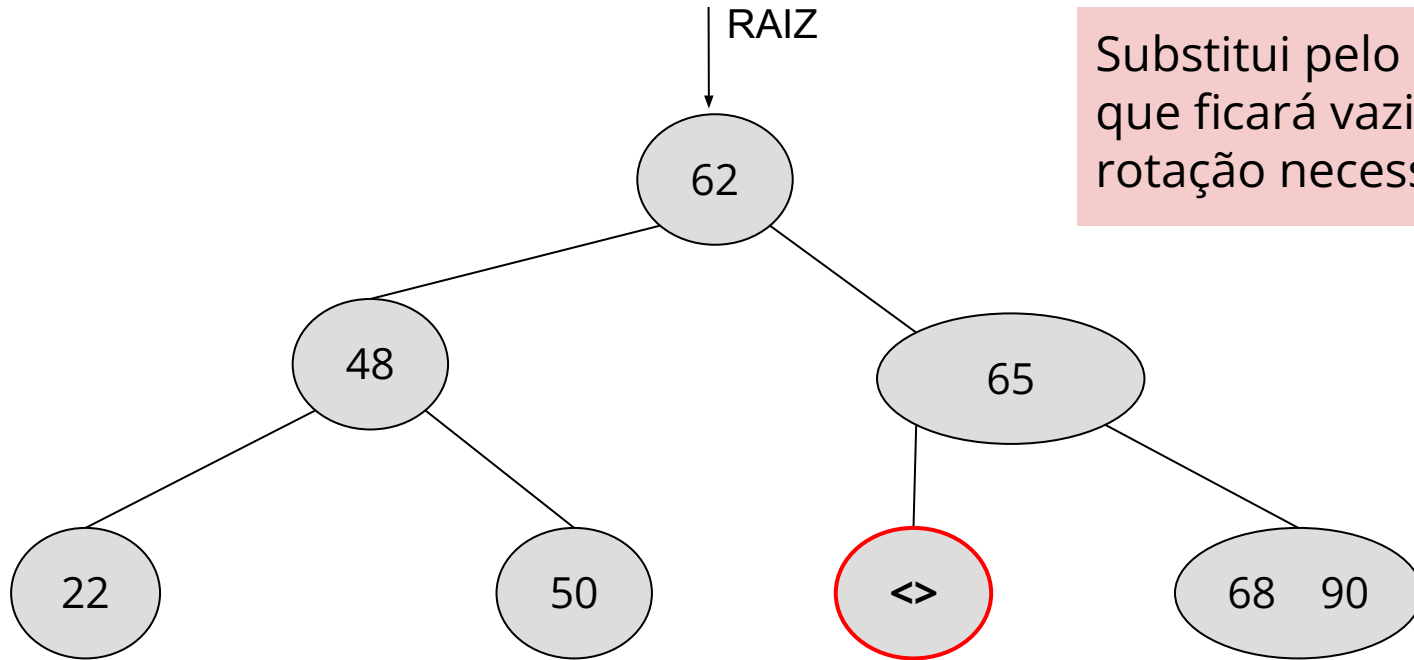
Sequência de remoção: 60 36 38 40 12 55 62



Substitui pelo sucessor,
que ficará vazio:
rotação necessária.

Exemplo - Remoção - 20/27

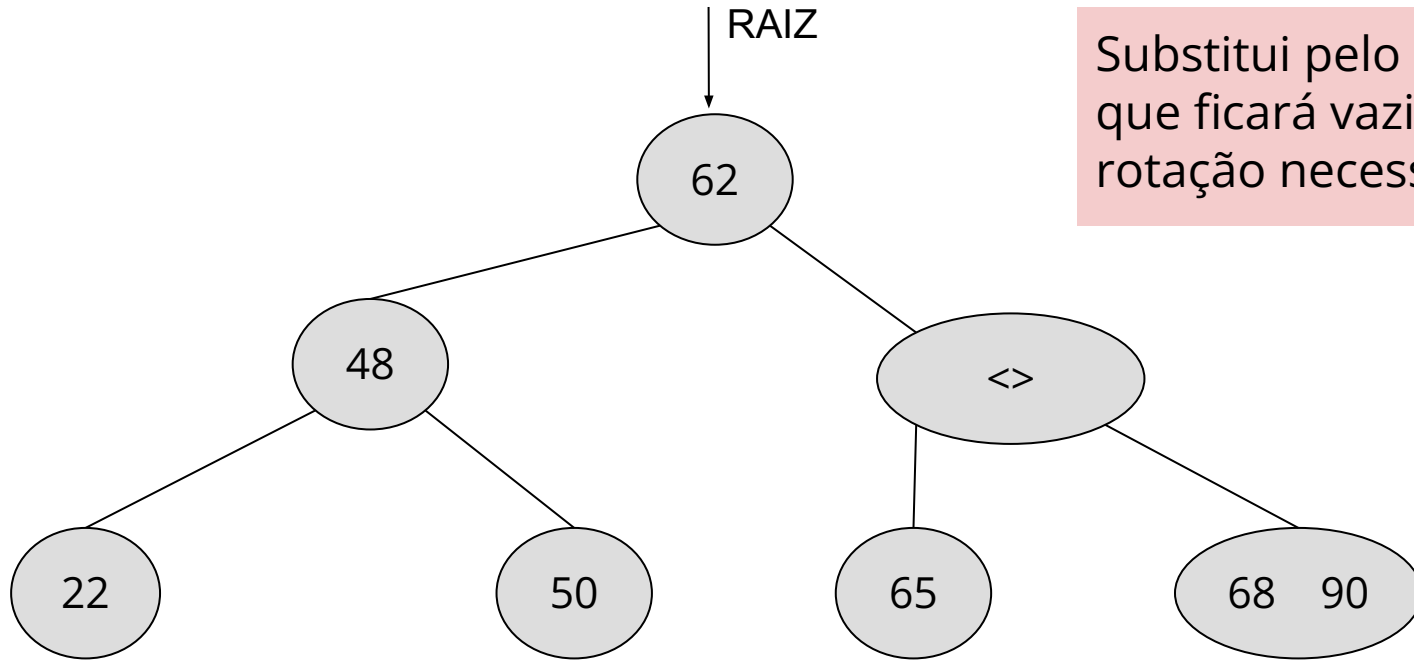
Sequência de remoção: 60 36 38 40 12 55 62



Substitui pelo sucessor,
que ficará vazio:
rotação necessária.

Exemplo - Remoção - 21/27

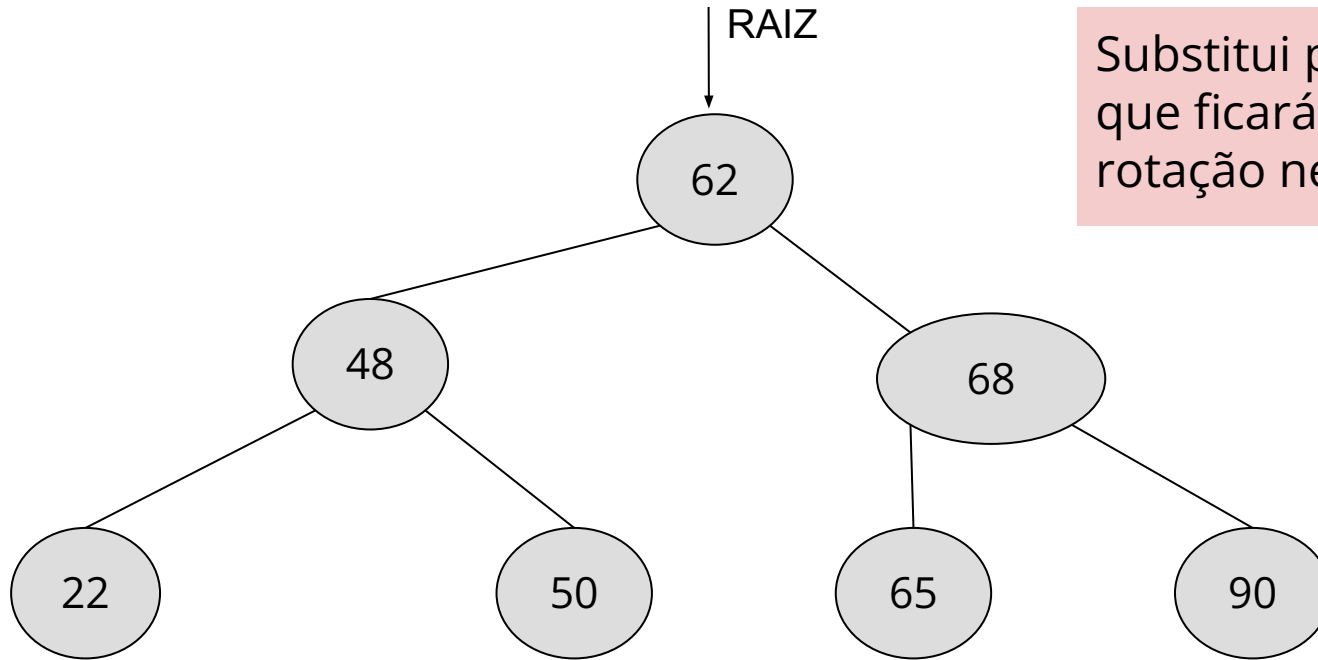
Sequência de remoção: 60 36 38 40 12 55 62



Substitui pelo sucessor,
que ficará vazio:
rotação necessária.

Exemplo - Remoção - 22/27

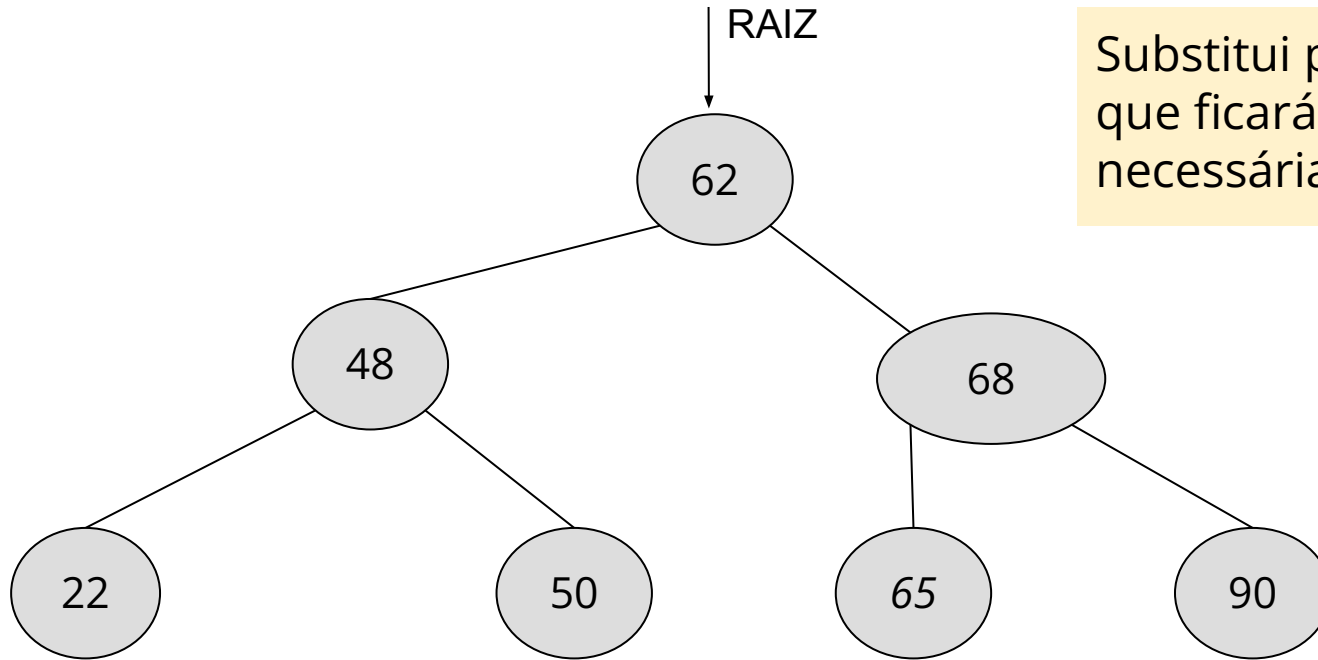
Sequência de remoção: 60 36 38 40 12 55 62



Substitui pelo sucessor,
que ficará vazio:
rotação necessária.

Exemplo - Remoção - 23/27

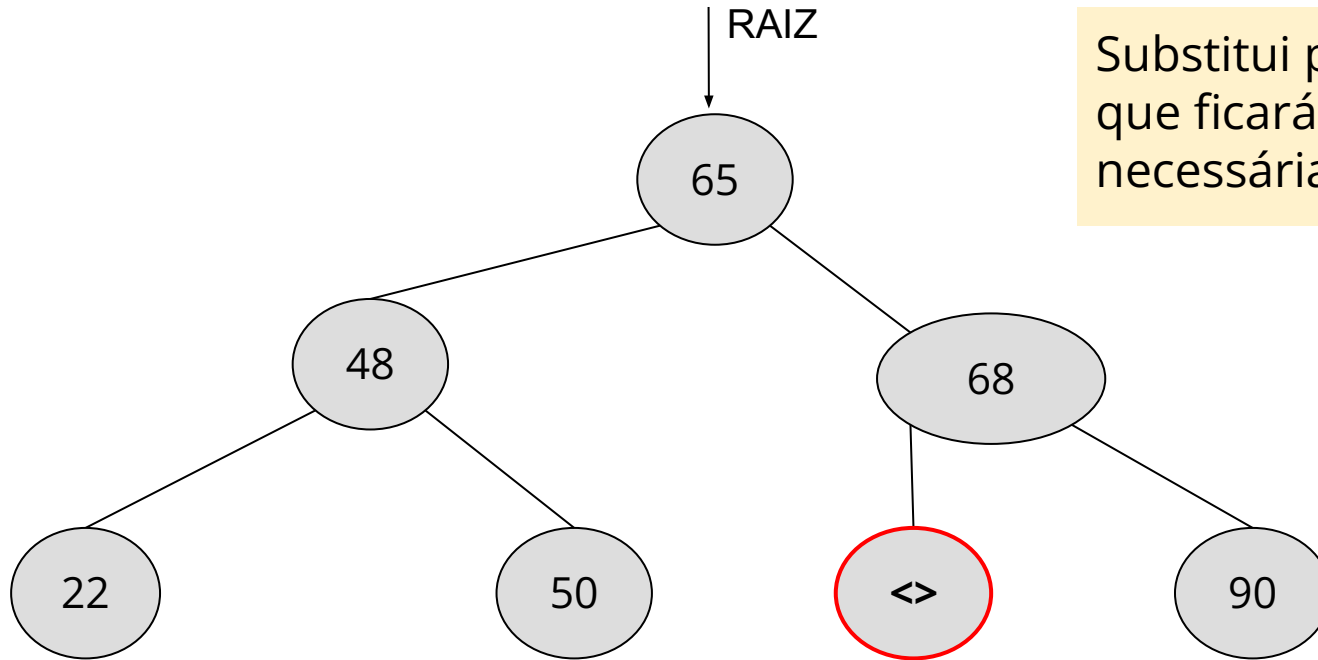
Sequência de remoção: 60 36 38 40 12 55 **62**



Substitui pelo sucessor, que ficará vazio: fusão necessária.

Exemplo - Remoção - 24/27

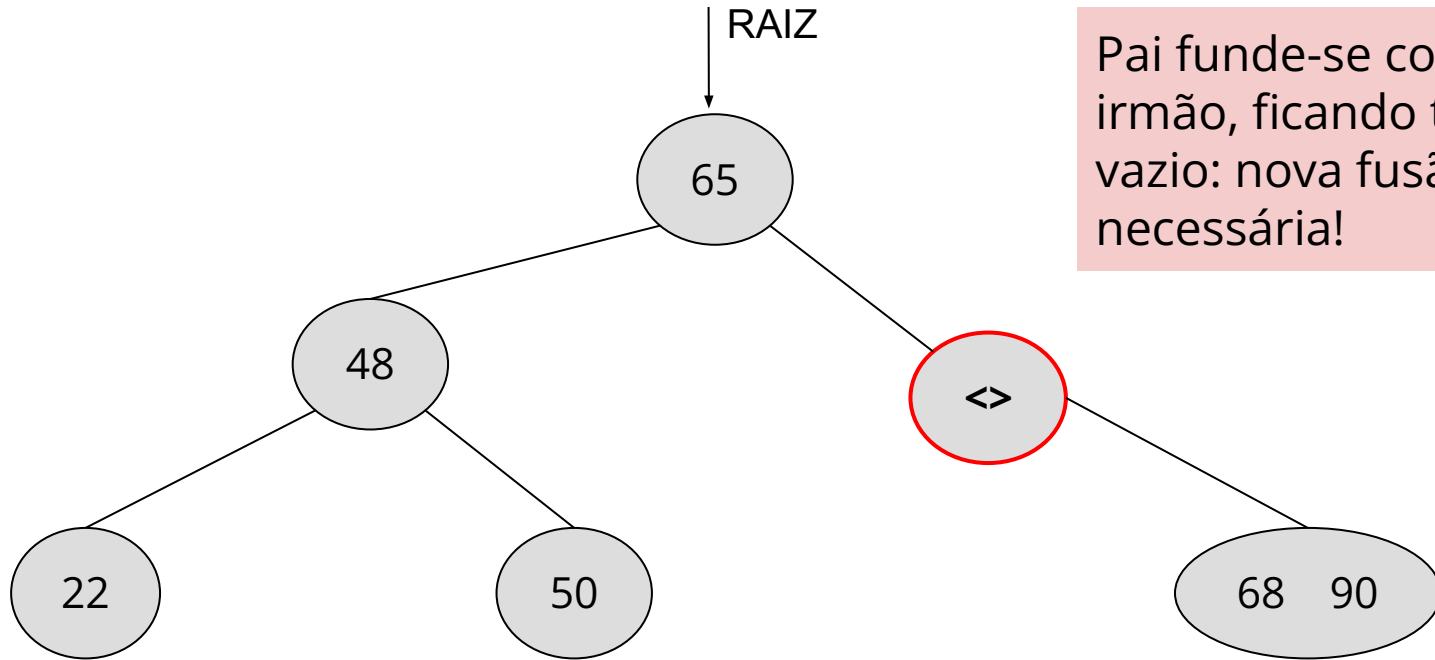
Sequência de remoção: 60 36 38 40 12 55 **62**



Substitui pelo sucessor,
que ficará vazio: fusão
necessária.

Exemplo - Remoção - 25/27

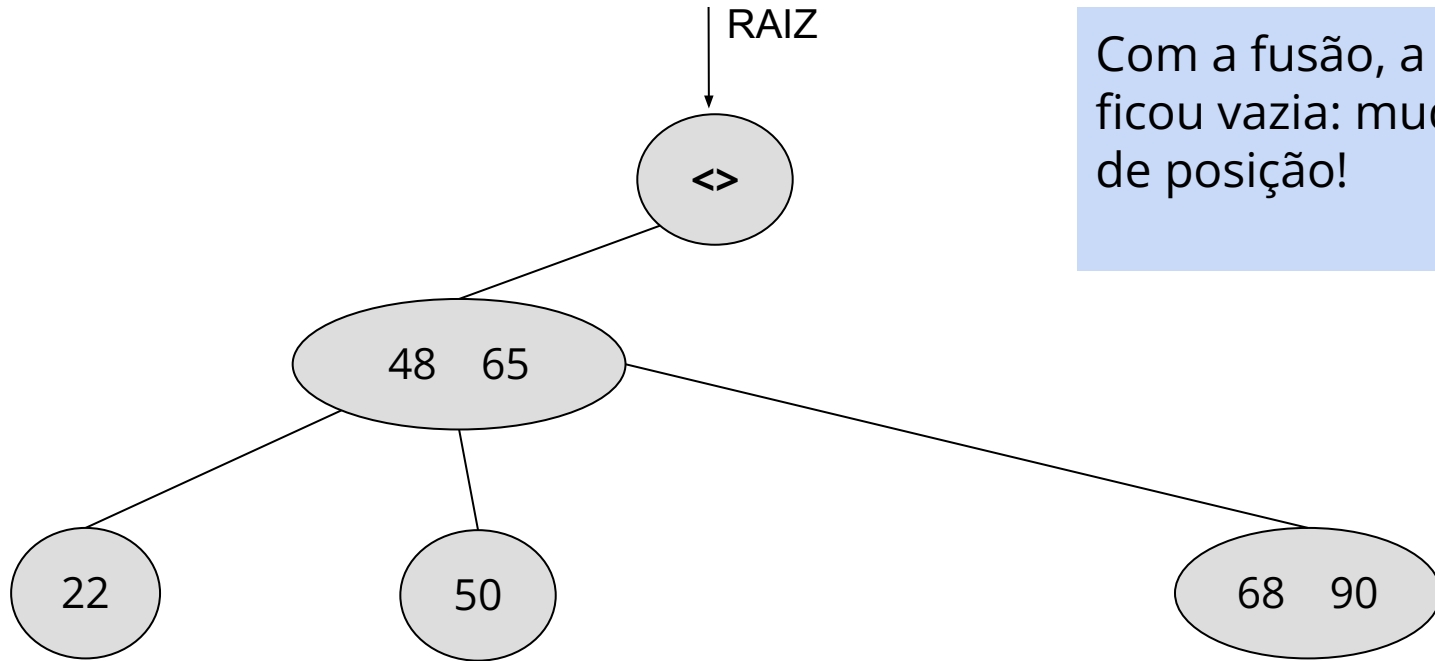
Sequência de remoção: 60 36 38 40 12 55 **62**



Pai funde-se com irmão, ficando também vazio: nova fusão necessária!

Exemplo - Remoção - 26/27

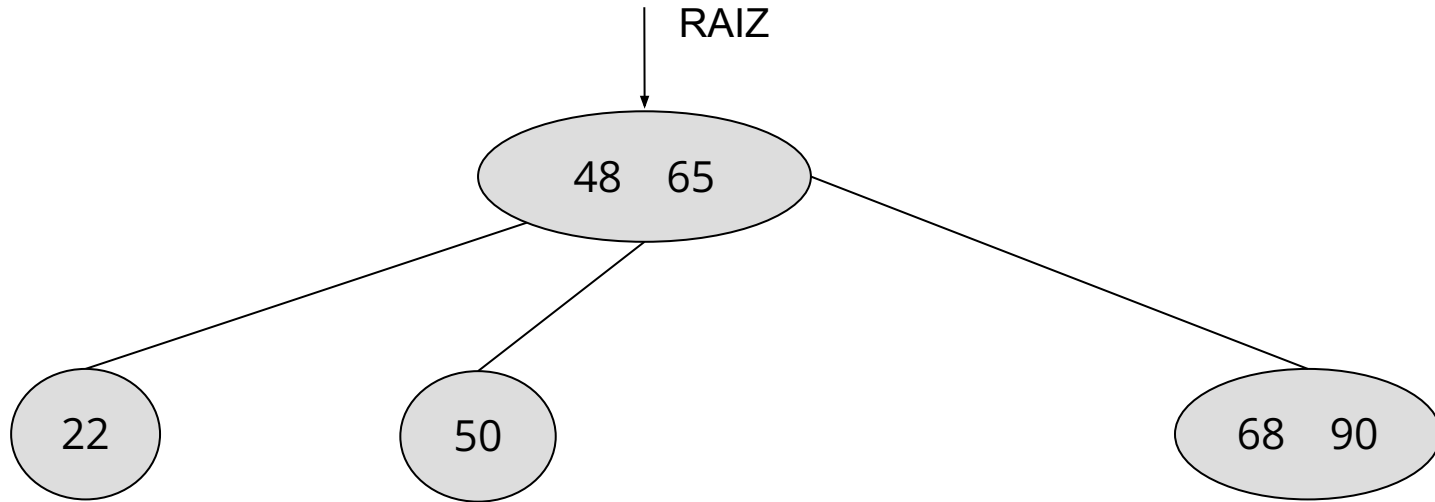
Sequência de remoção: 60 36 38 40 12 55 **62**



Com a fusão, a raiz ficou vazia: muda a raiz de posição!

Exemplo - Remoção - 27/27

Sequência de remoção: 60 36 38 40 12 55 **62**



Implementando Árvores 2-3 e 2-3-4 - i/ii

Existem várias formas e abordagens para implementação de árvores 2-3 e 2-3-4, que são bastante similares. Aqui iremos mostrar para árvores 2-3-4.

Algumas das abordagens criam entidades específicas para cada tipo de nó: nó-2, nó-3, nó-4, por exemplo.

Uma implementação baseada em árvores B utiliza um nó com um vetor de chaves e um vetor de filhos, com um atributo para controle das posições utilizadas, bem como se é ou não folha.

Implementando Árvores 2-3 e 2-3-4 - i/ii

Em uma implementação estilo árvore B, como não há mudança na capacidade, os vetores de dados e de filhos podem ser alocados estaticamente.

Algumas constantes podem ser utilizadas para facilitar a implementação (ex: para árvore 2-3-4):

MAXITENS = 3;

MINITENS = 1;

MAXFILHOS = 4;

MEIO = 1;

METADE = 1;

Árvore 2-3-4 - Implementação da Inserção - 1/2

A inserção em uma árvore 2-3-4 envolve uma série de passos menores e pode ser feita de maneira recursiva ou iterativa. Neste curso, iremos utilizar a forma recursiva.

Quando da inserção, busca-se um nó folha. Caso este não esteja cheio, efetua-se uma inserção em um nó folha não cheio (que basicamente insere a chave no vetor de chaves).

Se o nó folha estiver cheio, é necessário dividi-lo, criando um novo nó e verificando a chave que será utilizada para separar o nó atual e o novo nó. Com isso, é necessário retornar recursivamente a chave promovida e o nó criado.

Árvore 2-3-4 - Implementação da Inserção - 2/2

Pode ocorrer de, quando da divisão de um nó folha, ser necessário também dividir o nó pai (e outros antecessores), recursivamente.

Em todos os casos de divisão, a chave promovida e o nó criado para dividir os dados deverão ser ajustados no sucessor do nó dividido. Dessa maneira, é necessário efetuar uma inserção em um nó intermediário não cheio. Neste caso, além de inserir a chave no vetor de chaves, é necessário também inserir o nó criado no vetor de nós filhos.

Árvore 2-3-4 - Inserção 1/14

inserirEmNohFolhaNaoCheio(umNoh, umItem):

```
// método auxiliar, insere item em nó folha com capacidade
pos ← umNoh.num - 1; // num é o número de elementos em umNoh
enquanto (pos ≥ 0 e umNoh.itens[pos] > umItem) {
    // move item uma posição à direita
    umNoh.itens[pos+1] ← umNoh.itens[pos];
    pos--;
}
// Insere novo item no local encontrado
umNoh.itens[i+1] ← umItem;
umNoh.num++;
```

Árvore 2-3-4 - Inserção 2/14

divideNoh(umNoh):

```
// divide um nó, retornando o novo nó e o item promovido
itemPromovido ← umNoh->itens[MEIO];
novoNoh ← criar_noh();
novoNoh.folha ← umNoh.folha;
novoNoh.itens[0] ← umNoh.itens[MEIO+1];
// agora cada nó tem metade dos elementos
novoNoh.num ← METADE;
umNoh.num ← METADE;
// se nó não é folha, divide os filhos
se (não(umNoh->folha)) { ...
```

Árvore 2-3-4 - Inserção 3/14

```
// se nó não é folha, divide os filhos
se (não(umNoh->folha)) {
    para (i de 0 até MEIO+1) {
        novoNoh.filhos[i] ← umNoh.filhos[MEIO+1+i];
    }
}

retornar novoNoh, itemPromovido;
```

Árvore 2-3-4 - Inserção 3/14

```
// se nó não é folha, divide os filhos
```

```
se (não(umNoh->folha)) {
```

```
    para (i de 0 até MEIO+1) {
```


```
        novoNoh.filhos[i] ← umNoh.filhos[MEIO+1+i];
```

```
    }
```

```
}
```

```
retornar novoNoh, itemPromovido;
```

Função retorna dois valores.



Árvore 2-3-4 - Inserção 4/14

insereEmNohIntermediarioNaoCheio(umNoh, novoNoh, itemPromovido):

```
// insere em nó intermediário (após divisão de nó filho)
pos ← umNoh.num - 1; // num é o número de elementos em umNoh
enquanto (pos ≥ 0 e umNoh.itens[pos] > itemPromovido) {
    // move item uma posição à direita
    umNoh.itens[pos+1] ← umNoh.itens[pos];
    // move filho à direita de item uma posição à direita
    umNoh.filhos[pos+2] ← umNoh.filhos[pos+1];
    pos--;
}
...
```

Árvore 2-3-4 - Inserção 5/14

```
// Insere novo item no local encontrado
```

```
umNoh.itens[pos+1] ← itemPromovido;
```

```
// Insere novo nó (uma posição à frente no vetor de filhos)
```

```
umNoh.filhos[pos+2] ← novoNoh;
```

```
umNoh.num++;
```

Árvore 2-3-4 - Inserção 6/14

inserirRecursivamente(umItem):

```
// se árvore estiver vazia, aloca nó folha para a raiz
// e insere objeto na posição inicial
se (raiz = NULO) {
    raiz ← criar_noh();
    raiz.folha ← VERDADEIRO;
    raiz.itens[0] ← umItem;
    raiz.qtdade ← 1;
} senão { // já tem algo na raiz
    ...
}
```

Árvore 2-3-4 - Inserção 6/14

inserirRecursivamente(umItem):

```
// se árvore estiver vazia, aloca nó folha para a raiz  
// e insere objeto na posição inicial
```

```
se (raiz = NULO) {  
    raiz ← criar_noh();  
    raiz.folha ← VERDADEIRO;  
    raiz.itens[0] ← umItem;  
    raiz.qtdade ← 1;  
} senão { // já tem algo na raiz  
    ...
```

Nó criado vazio,
sem qualquer item

Quantidade de
itens (se não for
folha, então tem
um filho a mais)

Árvore 2-3-4 - Inserção 7/14

```
} else { // já tem algo na raiz
// itemPromovido refere-se a item de filho que foi dividido
// novoNoh é gerado em caso de divisão
    novoNoh, itemPromovido ← insereRecAux(raiz, umItem);
    // verifica se houve divisão na raiz
    se (novoNoh) { // se novoNoh não é nulo, houve divisão
        // cria nova raiz apontando com antiga raiz e novoNoh
        // como filhos
        ...
    }
```

Árvore 2-3-4 - Inserção 7/14

Função deve retornar dois valores.

```
} else { // já tem algo na raiz
// itemPromovido refere-se a item de filho que foi dividido
// novoNoh é gerado em caso de divisão
novoNoh, itemPromovido ← insereRecAux(raiz, umItem);
// verifica se houve divisão na raiz
se (novoNoh) { // se novoNoh não é nula, houve divisão
    // cria nova raiz apontando com antiga raiz e novoNoh
    // como filhos
    ...
}
```

Árvore 2-3-4 - Inserção 8/14

```
se (novoNoh) { // se novoNoh não é nulo, houve divisão
    // cria nova raiz apontando com antiga raiz e novoNoh
    // como filhos
    antigaRaiz ← raiz;
    raiz ← criar_noh();
    raiz.itens[0] ← itemPromovido;
    raiz.qtdade ← 1;
    raiz.filhos[0] ← antigaRaiz;
    raiz.filhos[1] ← novoNoh;
}
}
```

Árvore 2-3-4 - Inserção 9/14

inserirRecAux(umNoh, umItem):

```
// Caso umNoh seja folha, encontre o local para inserir
se (umNoh.folha) {
    // verificando se umNoh não está cheio
    se (umNoh->num < MAXITENS) {
        // não está cheio, basta inserir
        insereEmNohFolhaNaoCheio(umNoh, umItem);
        retornar NULO,NULO; // indica que não houve divisão
    } senão {
        // umNoh está cheio, precisa dividir
        ...
    }
}
```


Árvore 2-3-4 - Inserção 10/14

```
} senão {  
    // umNoh está cheio, precisa dividir  
    novoNoh, itemPromovido ← divideNoh(umNoh);  
    // verifica se umNoh ou novoNoh recebe umItem  
    if (umItem ≤ umNoh.itens[MEIO]) {  
        // item fica em umNoh  
        insereEmNohFolhaNaoCheio(umNoh, umItem);  
    } senão {  
        // item fica em novoNoh  
        insereEmNohFolhaNaoCheio(novoNoh, umItem)  
    }  
    retornar novoNoh, itemPromovido;  
}
```

Árvore 2-3-4 - Inserção 11/14

```
} senão { // nó não é folha
    // Encontra filho que irá receber novo item
    // vai do final ao começo
    int i ← umNoh.num - 1;
    enquanto ((i ≥ 0) e (umNoh.itens[i] > umItem)) {
        i--;
    }
    nohAux, itemPromovido ←
        insereRecAux(umNoh->filhos[i+1], umItem);
    // verifica se não houve estouro no filho
    se (nohAux) { // novoNoh não é nulo, houve divisão
        ...
    }
```

Árvore 2-3-4 - Inserção 11/14

```
} senão { // nó não é folha
    // Encontra filho que irá receber novo item.
    // vai do final ao começo
    int i ← umNoh.num - 1;
    enquanto ((i ≥ 0) e (umNoh.itens[i] > umItem)) {
        i--;
    }
    nohAux, itemPromovido ←
        insereRecAux(umNoh->filhos[i+1], umItem);
    // verifica se não houve estouro no filho
    se (nohAux) { // novoNoh não é nulo, houve divisão
        ...
    }
```

Ponto de recursão



Árvore 2-3-4 - Inserção 12/14

Considerando que itemPromovido esteja sendo passado por referência, o objetivo aqui é manter o valor antes de ser alterado, para comparações.

```
se (nohAux) { // novoNoh não é nulo, houve
// antes de inserir o item promovido,
// verifica se não deve dividir o noh
// armazena antes itemPromovido em var
itemAux <- itemPromovido;
se (umNoh->num < MAXITENS) {
    // umNoh não está cheio, arrumar o estouro do filho
    insereEmNohIntermediarioNaoCheio(umNoh, nohAux,
        itemPromovido);
    retornar NULO, NULO;
} senão {
    ...
}
```

Árvore 2-3-4 - Inserção 12/14

```
se (nohAux) { // novoNoh não é nulo, houve divisão
    // antes de inserir o item promovido,
    // verifica se não deve dividir o noh atual
    // armazena antes itemPromovido em variável auxiliar
    itemAux <- itemPromovido;
    se (umNoh->num < MAXITENS) {
        // umNoh não está cheio, arrumar o estouro do filho
        insereEmNohIntermediarioNaoCheio(umNoh,nohAux,
            itemPromovido);
        retornar NULO, NULO;
    } senão {
        ...
    }
```

Árvore 2-3-4 - Inserção 13/14

```
} senão {  
    // umNoh está cheio,  
    // divide antes de arrumar estouro do filho  
    itemPromovFilho ← itemPromovido;  
    novoNoh, itemPromovido ← divideNoh(umNoh, umItem);  
    // verifica quem vai receber novo nó  
    // e item promovido do filho, se umNoh ou novoNoh  
    se (itemAux ≤ umNoh.itens[MEIO]) {  
        // nó e item ficam em umNoh  
        insereEmNohIntermediarioNaoCheio(umNoh, nohAux,  
            itemPromovFilho);  
    } senão { ...
```

Árvore 2-3-4 - Inserção 14/14

```
    } senão {  
        // nó e item ficam em novoNoh  
        insereEmNohIntermediarioNaoCheio(novoNoh, nohAux,  
            itemPromovFilho);  
    }  
    retornar novoNoh,itemPromovido;  
} // fim do se..senão para teste se nó atual está cheio  
} // fim do teste se houve estouro no nó filho  
retornar NULO, NULO; // inserção sem estouro  
}
```

Visão Geral de Busca e Percorrimento

O percorrimto em árvores 2-3 ou 2-3-4 é relativamente simples, comparado à árvore binária padrão. Nesse caso apenas, basta levar em conta que cada nó pode possuir mais que dois filhos.

Com a busca, é necessário percorrer o vetor de chaves de um nó, para encontrar o filho adequado para descer na árvore, caso seja necessário.

Árvore 2-3-4 - Busca 1/3

busca(chave):

```
se (raiz = NULO) {  
    gerar_erro("Árvore vazia!");  
} senão {  
    itemBuscado ← buscaAux(raiz, chave);  
    efetuaAcao(itemBuscado);  
}
```

Árvore 2-3-4 - Busca 2/3

buscaAux(raizSub, chave):

i \leftarrow 0;

// percorre raizSub até achar um item

// com chave maior ou igual à procurada

enquanto ((*i* < raizSub.num) e (raizSub.itens[*i*] ≤ chave)) {

i++;

}

// retorna à posição anterior (desfaz o último incremento)

i--;

// se item é igual à chave, terminou a busca

se (raizSub.itens[*i*] = chave) { ...

Árvore 2-3-4 - Busca 3/3

```
se (raizSub.itens[i] = chave) {  
    retornar raizSub.itens[i];  
} senão {  
    //se nó é folha, então árvore não tem o elemento buscado  
    se (raizSub.folha) {  
        retornar NULO;  
    } senão {  
        // não é folha, desce em nó filho à esquerda do item  
        // com chave maior que a procurada  
        retornar buscaAux(raizSub.filhos[i+1],chave);  
    }  
}
```

Visão Geral para Implementação da Remoção 1/2

A remoção em árvores 2-3 e 2-3-4 é baseada na remoção em árvores binárias. Uma vez encontrada a chave, é necessário verificar se a mesma encontra-se em nó folha. Se não se encontra, ela deve ser substituída pela chave antecessora ou sucessora (que encontra-se em um nó folha).

Quando da remoção da chave ou de sua substituta no nó folha, é necessário verificar se o nó não ficará vazio. Se o nó ficar vazio, será necessário verificar se o(s) nós irmão(s) não possuem duas ou três chaves. Nesse caso, é feita uma rotação, passando um chave do nó irmão para o nó pai e do nó pai para o nó em que houve a remoção de chave.

Visão Geral para Implementação da Remoção 2/2

Caso o nó esteja para ficar vazio e seus irmãos possuem apenas uma chave, é necessário efetuar a fusão de nós, envolvendo nó pai e nó irmão: a chave do nó pai que separa o nó vazio com o seu irmão irá descer para ficar no nó irmão.

Nesse processo de fusão, é necessário verificar se o nó pai também não irá ficar vazio, repetindo o processo no nível superior na árvore. Pode ocorrer da fusão ir da folha até a raiz.



Para depois (LLRB)



Árvore RN-Arrumar Inserção- Pseudocódigo - i

arrumarBalanceamentoVP(umNoh):

```
se ( cor(umNoh.direito) = RED e cor(umNoh.esquerdo)=BLACK){
```

```
    umNoh = rotacionaEsquerda(umNoh);
```

```
}
```

```
se ( cor(umNoh.esquerdo)=RED e cor(umNoh.esquerdo.esquerdo)=RED){
```

```
    umNoh = rotacionaDireita(umNoh);
```

```
}
```

Árvore RN-Arrumar Inserção- Pseudocódigo - ii

```
se ( cor(umNoh.direito) = RED e cor(umNoh.esquerdo)=RED){  
    TrocaCor(umNoh);  
}
```


Árvore RN-Arrumar Inserção- Pseudocódigo - iii

cor(umNoh):

```
se ( (umNoh)=NILL ) { retornar Black;  
} senão { retornar umNoh.cor;}  
  
}
```

trocaCor(umNoh):

```
umNoh.cor = !umNoh.cor;  
se ((umNoh.direito)!=NILL){umNoh.direito.cor = !umNoh.direito.cor}  
se ((umNoh.esquerdo)!=NILL){umNoh.esquerdo.cor =  
!umNoh.esquerdo.cor}  
}
```