Algoritmos e Estruturas de Dados I

Árvores 2-3

Prof. Tiago Eugenio de Melo
tmelo@uea.edu.br

www.tiagodemelo.info

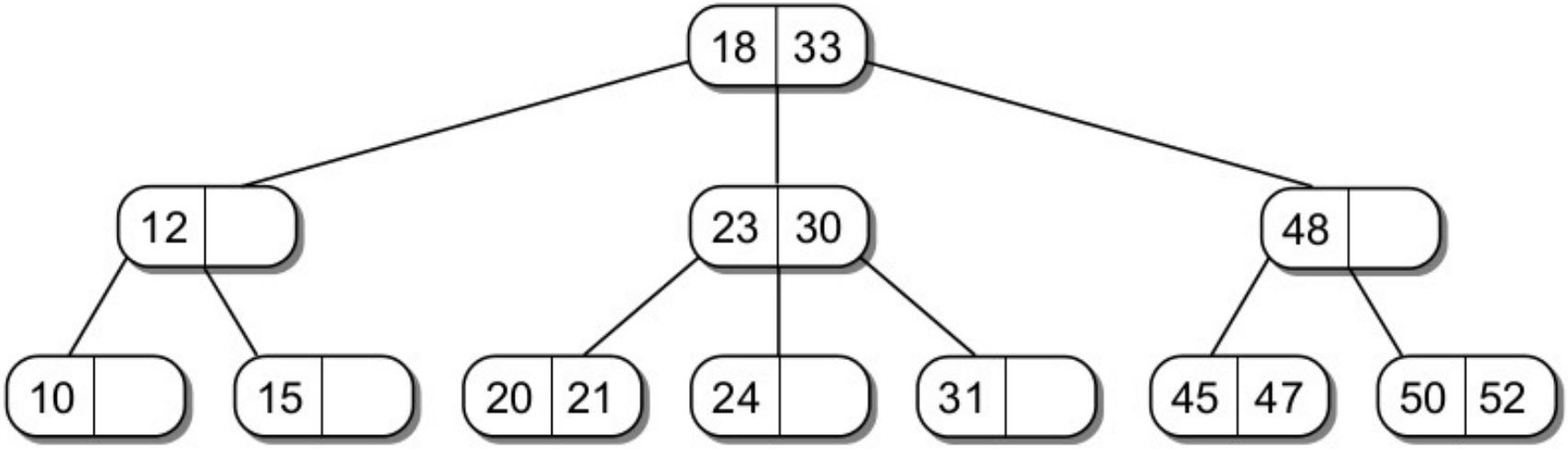
Observações

- O conteúdo dessa aula é parcialmente proveniente do Capítulo 14 do livro “*Data Structure and Algorithms Using Python*”.
- As palavras com a fonte `Courier` indicam uma palavra-reservada da linguagem de programação.

Árvores 2-3

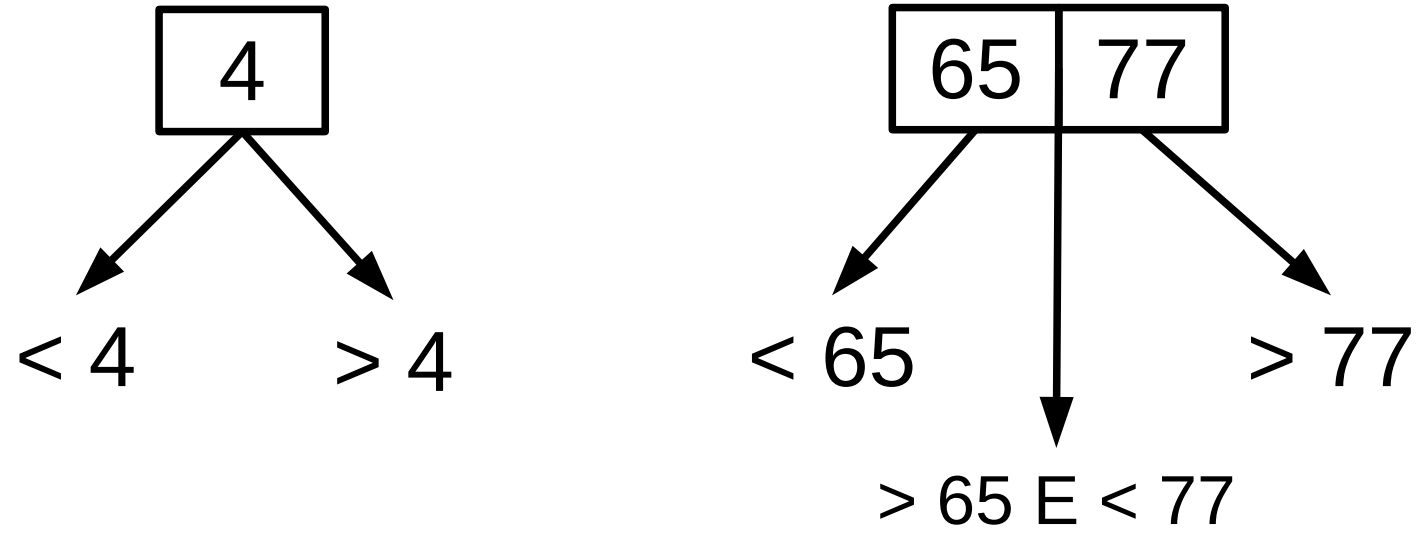
Introdução

- É uma estrutura alternativa para realização de busca de modo eficiente.
- É uma árvore que pode ter até 3 (três) filhos.



Propriedades

- É uma árvore de busca que está **sempre balanceada** e que tem a seguinte definição:
  - Cada nó pode ter uma ou duas chaves.
  - Cada nó não folha pode ter 2 ou 3 filhos.

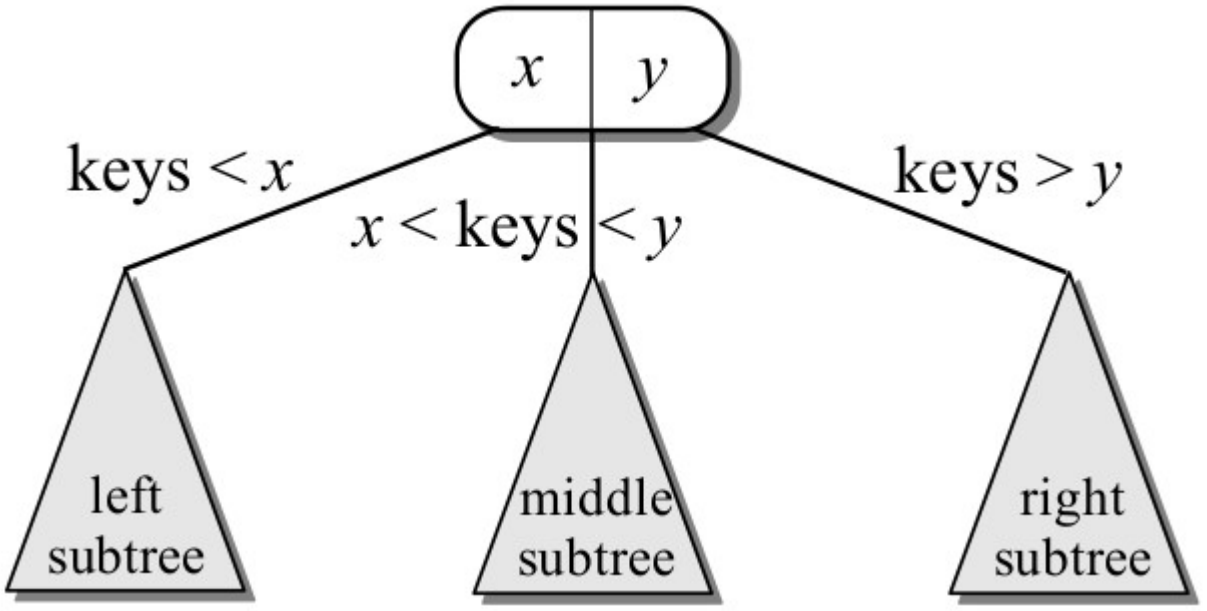


Propriedades

- Todos os nós folhas estão no mesmo nível.
- Portanto, toda árvore 2-3 é **perfeitamente balanceada**.
- Todo nó interno deve conter dois ou três filhos.
  - Se o nó tem 1 chave, então ele possui 2 filhos.
  - Se o nó tem 2 chaves, então ele possui 3 filhos.

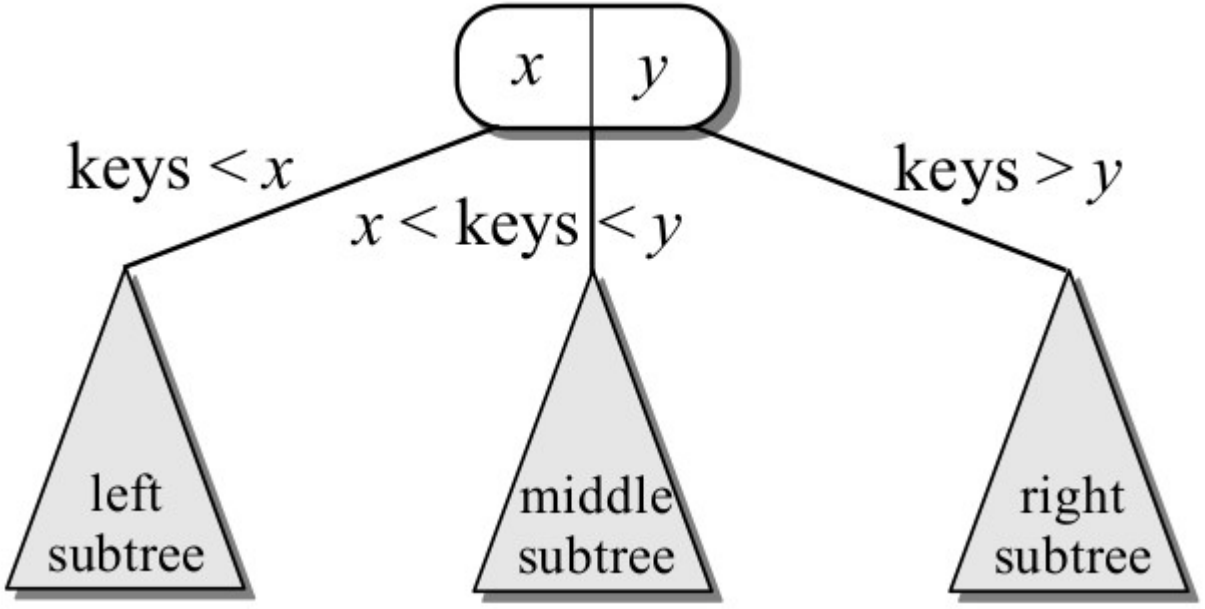
Propriedades (cont.)

- As árvores 2-3 mantêm as propriedades das árvores binárias de busca.
  - Todas as chaves que sejam menores do que a primeira chave de V são armazenadas na subárvore à esquerda de V.



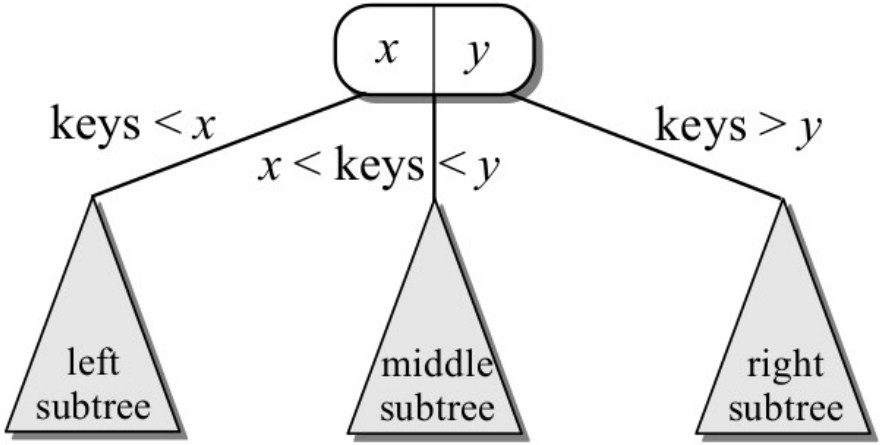
Propriedades (cont.)

- As árvore 2-3 mantêm as propriedades das árvores binárias de busca.
  - Se o nó tem dois filhos, então todas as chaves maiores do que a primeira chave do nó V e menores do que a segunda chave são armazenadas no meio da subárvore de V.



Propriedades (cont.)

- As árvore 2-3 mantêm as propriedades das árvores binárias de busca.
  - Se o nó tiver três filhos:
    - Todas as chaves que sejam maiores que a primeira chave do V, mas menores do que o segundo nó, são armazenados no meio da subárvore V.
    - Todas as chaves maiores do que a segunda chave são armazenados na subárvore à direita.



Busca

- A busca nas árvores 2-3 é bastante similar da busca nas ABBs.
- Nós devemos iniciar na raiz e seguir a ramificação apropriada baseada no valor da chave alvo.
- A única diferença é que nós temos que comparar o alvo contra duas chaves, caso o nó possua duas chaves, e então devemos escolher entre as três possíveis subárvores.

Busca

- Assim como numa ABB, uma busca com sucesso levará a chave em um dos nós da árvore, enquanto uma busca sem sucesso levará a um link *null*.
- Este link *null* será sempre em um nó folha.
- A razão para isso é que se um nó interior contém uma chave, ele sempre contém dois nós.

Busca

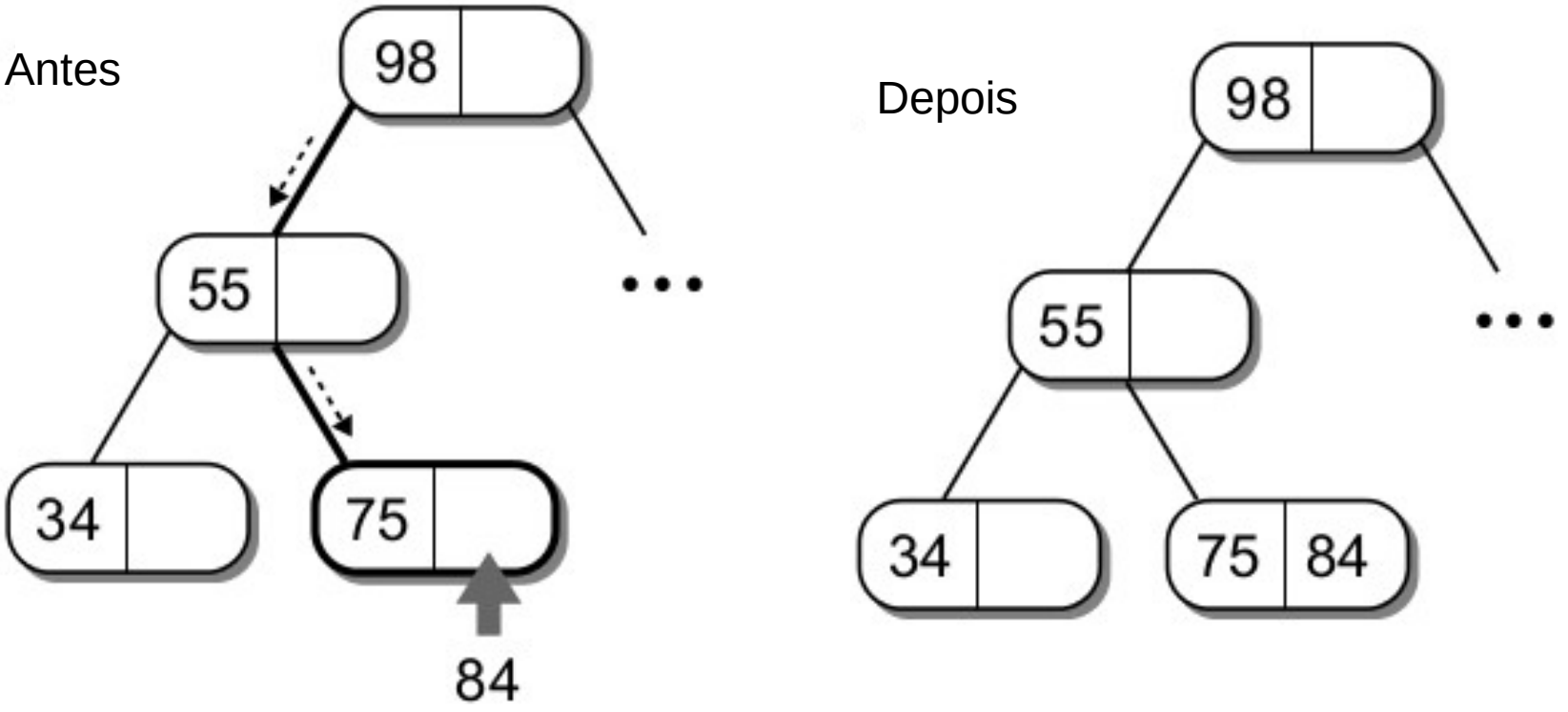
- Algoritmo (busca k):
  - Se k for igual a alguma chave, então encontramos.
  - Se k é menor do que a, então realizamos a busca no filho à esquerda.
  - Se k está entre a e b, então nós realizamos a busca no filho do meio.
  - Se k é maior que b, então realizamos a busca no filho à direita.

Inserção

- O processo de inserção é parecido com o método de inserção das árvores binárias, mas com algumas adaptações.
- O primeiro passo é buscar pelo elemento na árvore.
- A busca por um nó não-existente (novo nó) levará a um nó folha.
- O próximo passo é verificar se existe espaço nesse nó folha.

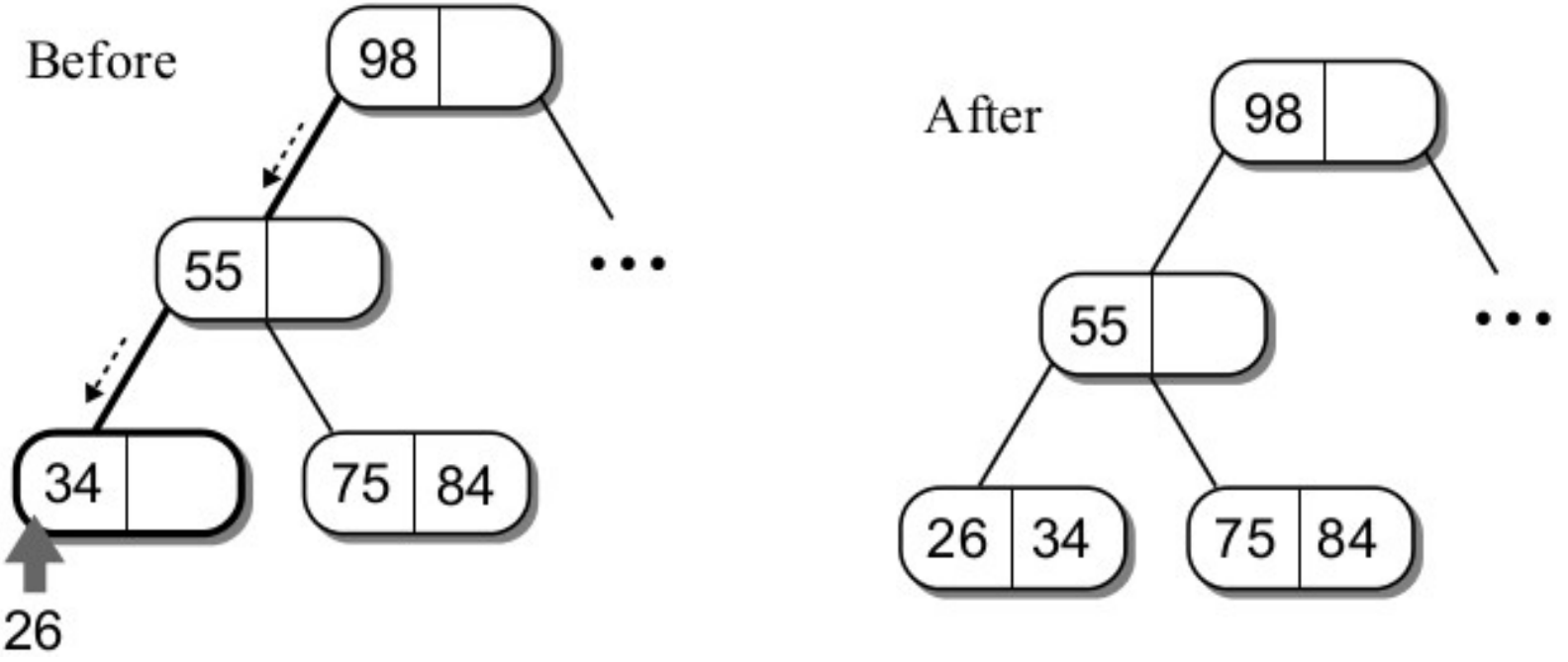
Inserção

- Se o nó folha contém apenas uma chave, então podemos inserir a chave nesse nó.
- Exemplo: inserir o elemento 84



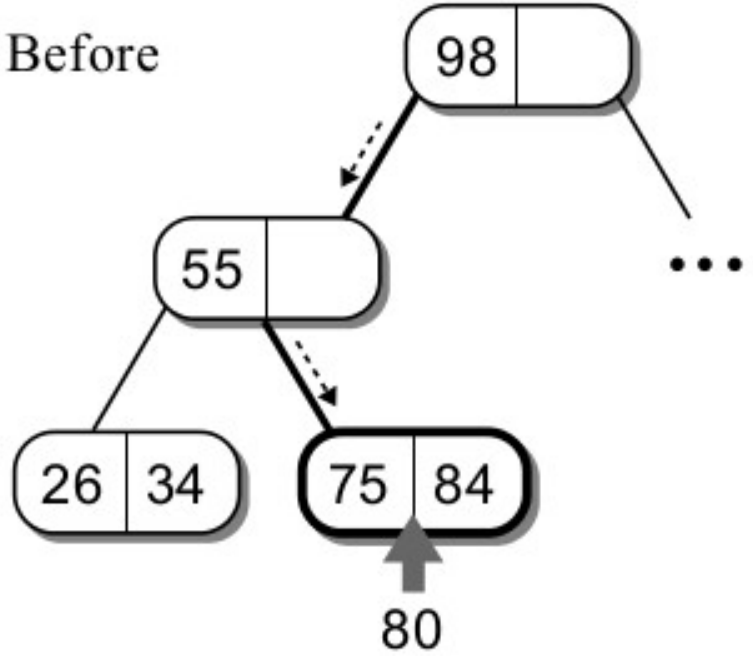
Inserção

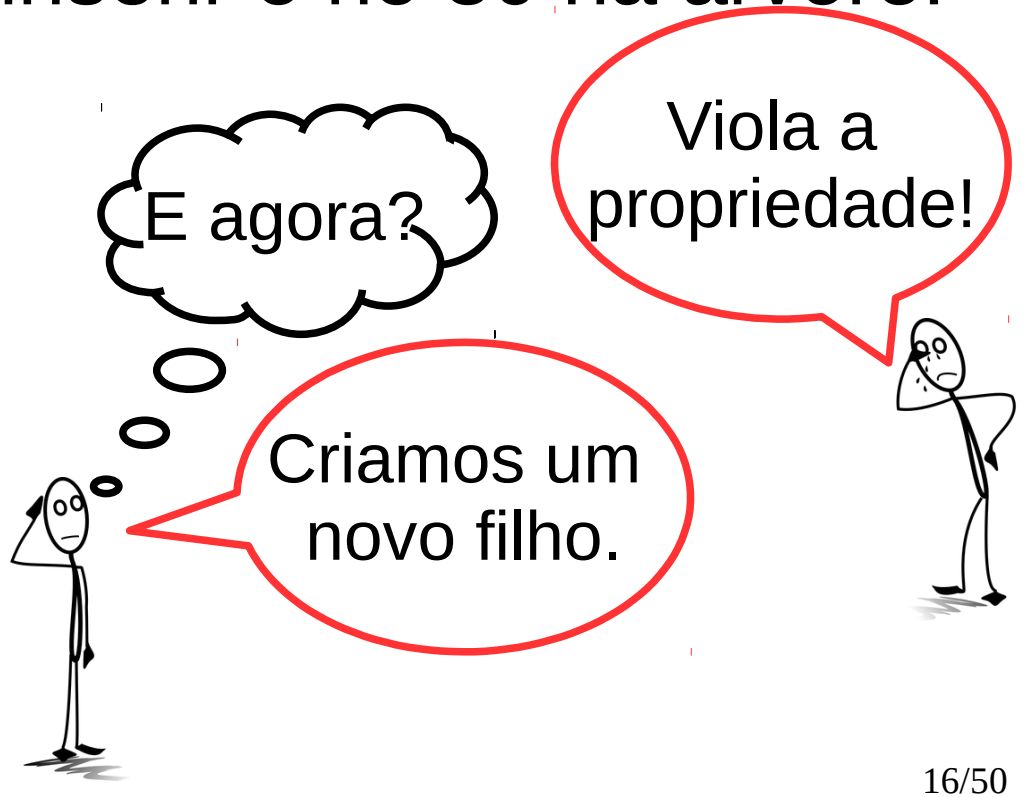
- Mas o que aconteceria se a chave do novo elemento fosse menor do que a chave armazenada no nó folha?
- Vamos inserir o elemento 26 na árvore abaixo:



Inserção

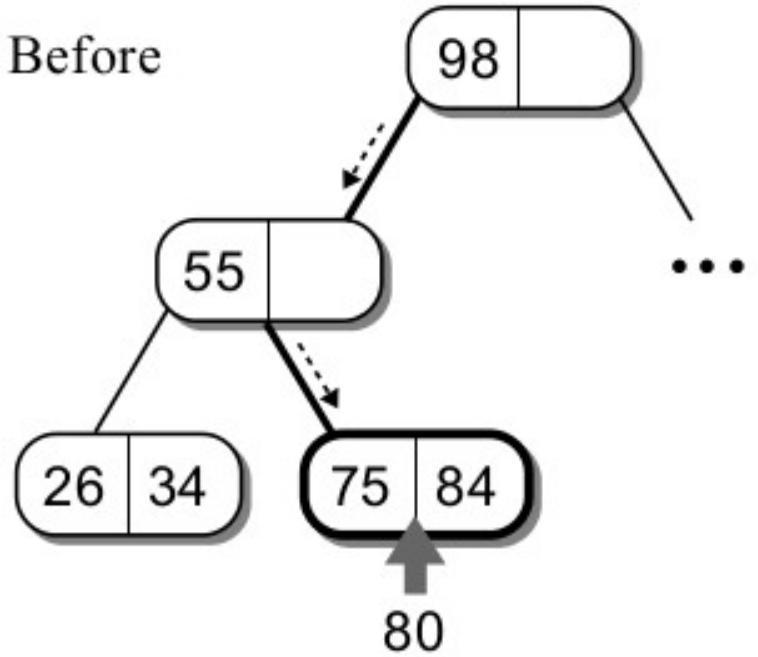
- As coisas se tornam um pouco mais complicadas (interessantes) quando o nó folha está cheio.
- Suponha que vamos inserir o nó 80 na árvore:





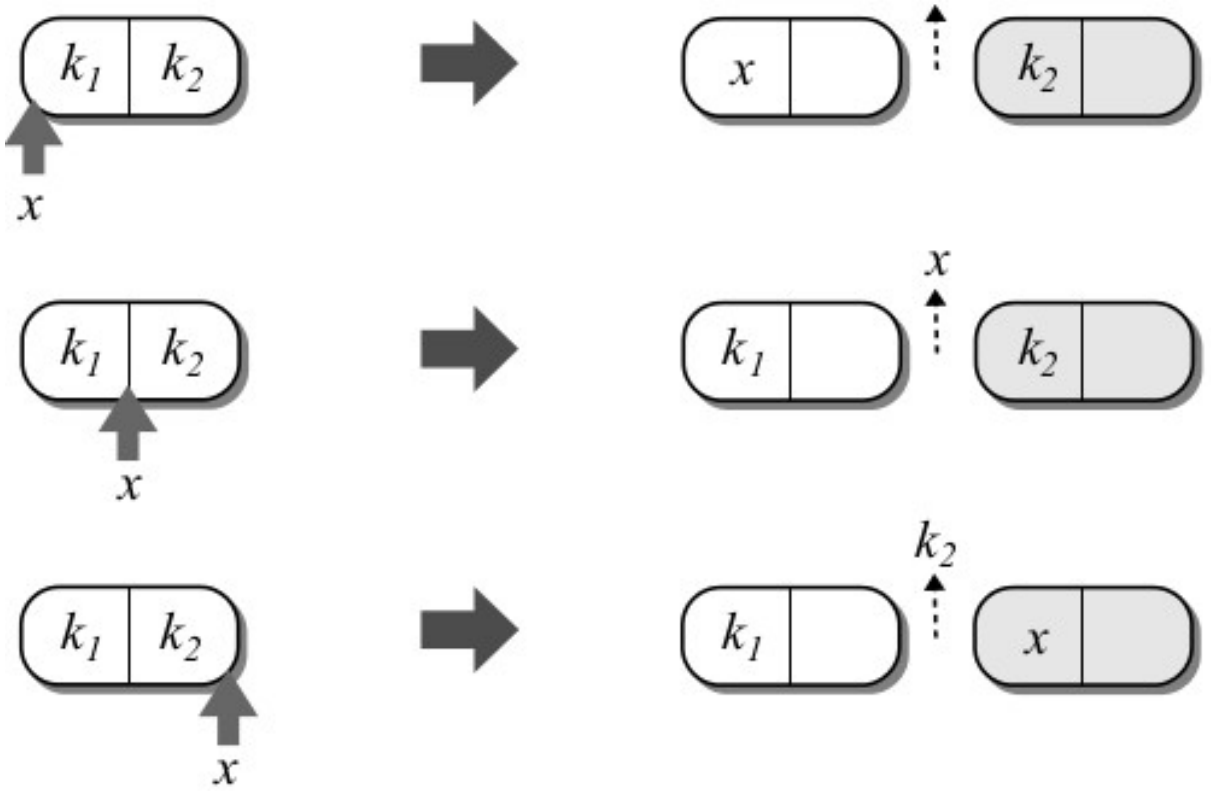
Inserção

- Será necessária uma operação de divisão em duas etapas.
- Primeiro, criamos um novo nó e então comparamos a nova chave com as duas outras chaves do nó folha (75 e 84).



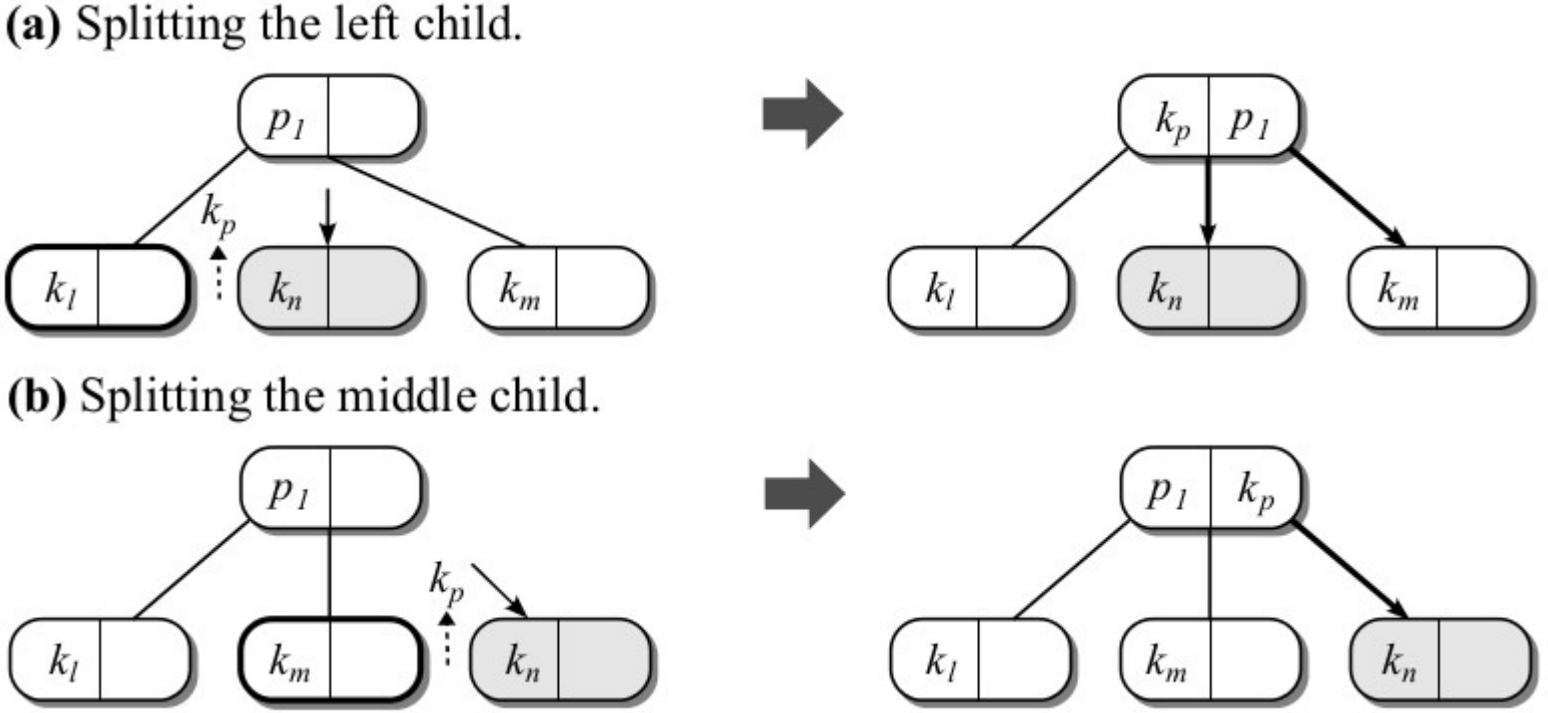
Inserção

- O menor valor é inserido no nó original.
- O maior valor é inserido no novo nó.
- O nó do meio passará a ser o nó pai.



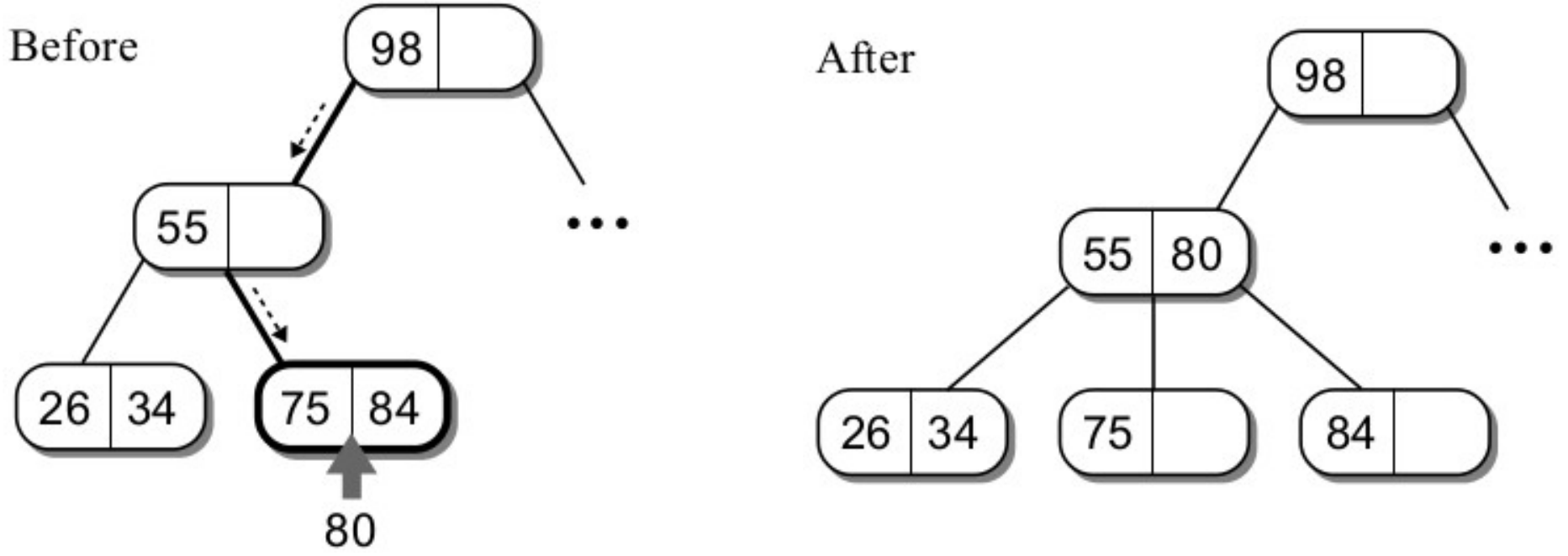
Inserção

- Quando um nó é promovido para o nível de pai, ele pode ser inserido de modo similar a inserção de nó folha.
- O procedimento é simples, se houver espaço.



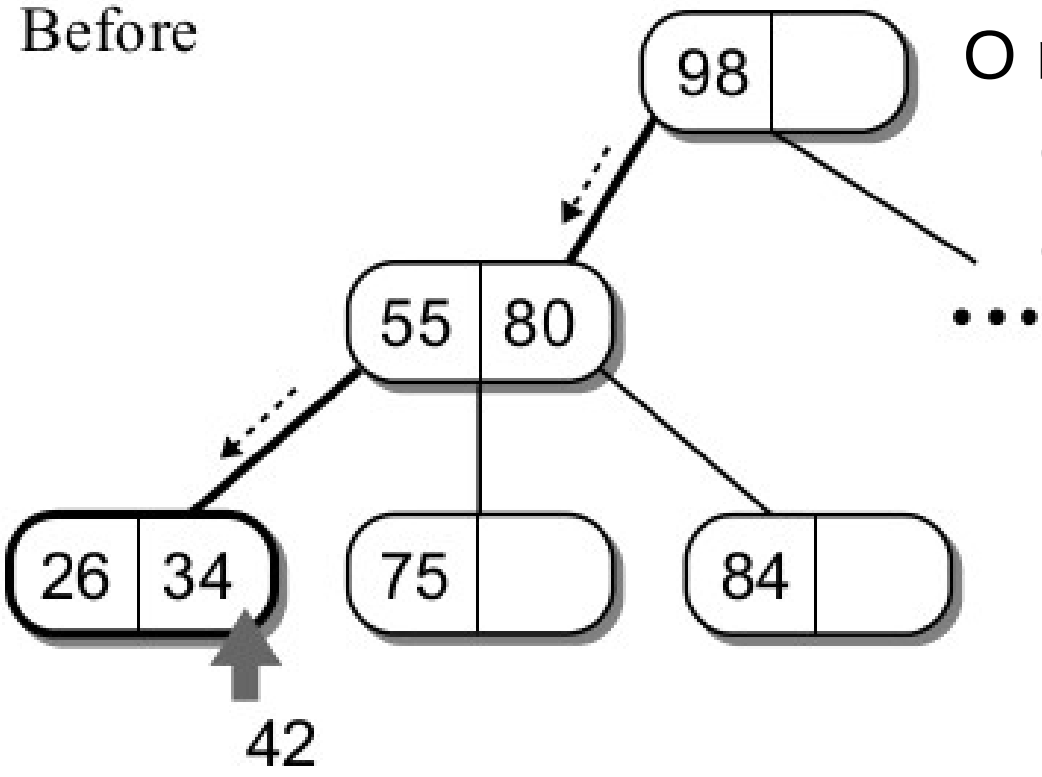
Inserção

- Inserir o elemento 80 (continuação):



Inserção

- O que aconteceria se o nó superior (pai) já estivesse cheio?
- Suponha a inserção do nó 42 na árvore abaixo:



O nó (26 e 34) deve ser dividido.

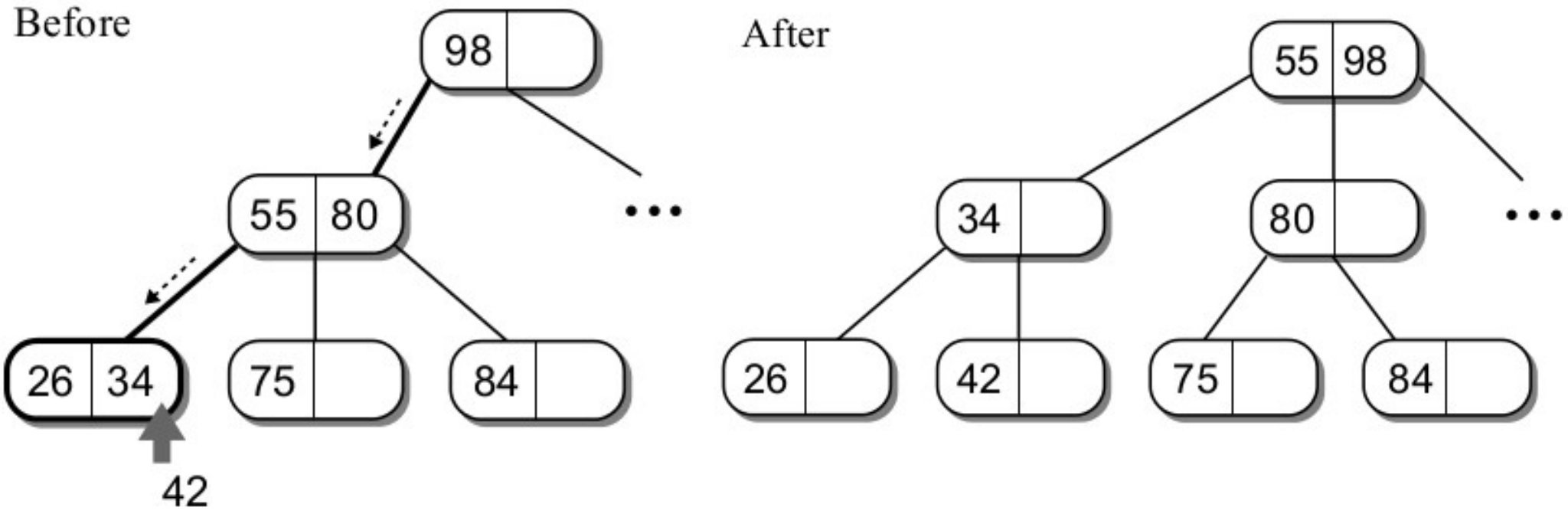
O nó 34 deve ser promovido.

O nó pai tem duas chaves (55,80).

Esse nó deverá ser dividido.

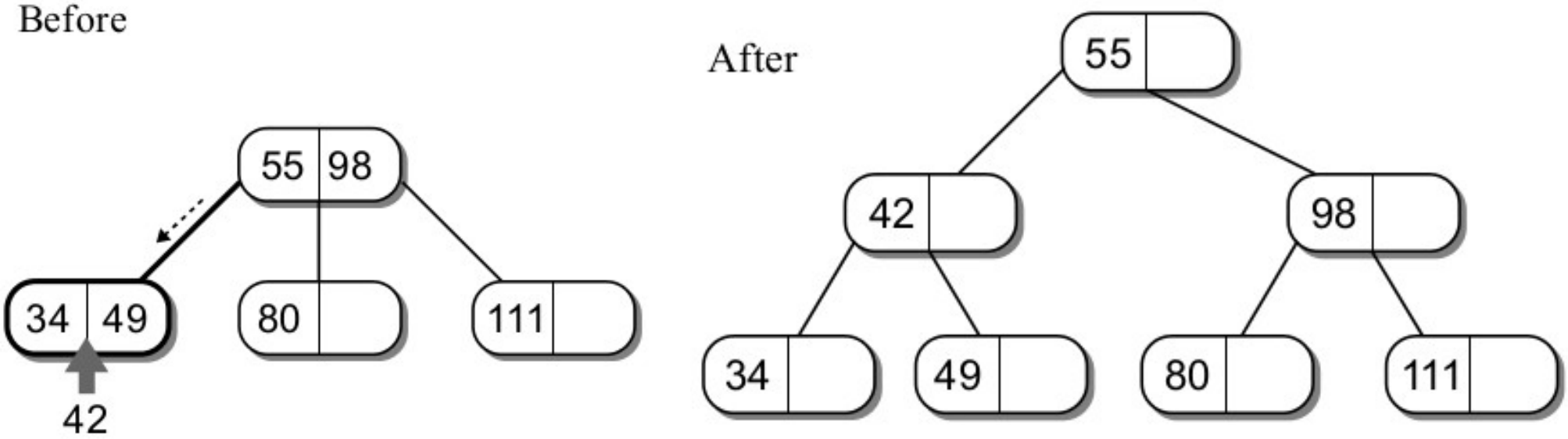
Processo será repetido até a raiz.

Inserção



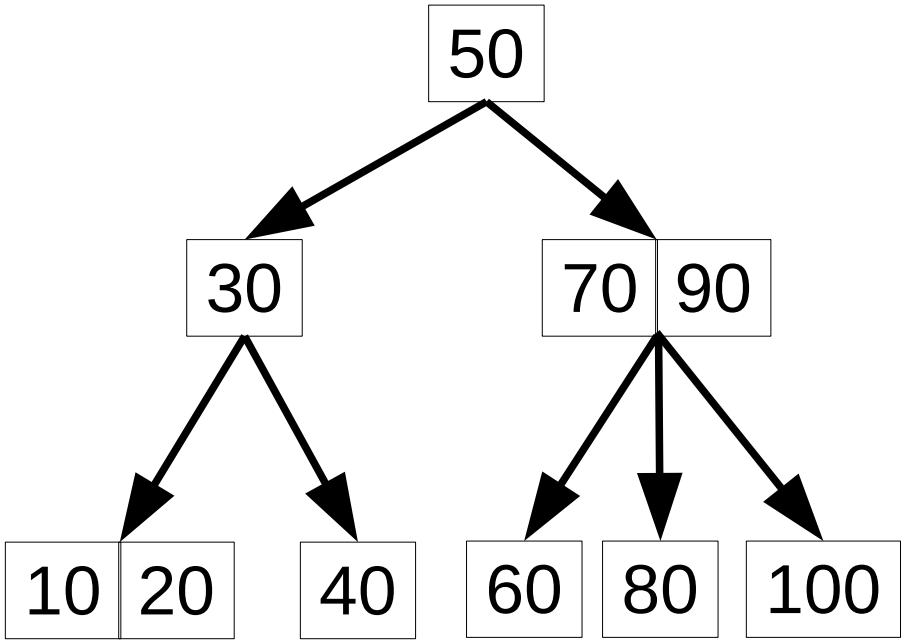
Inserção

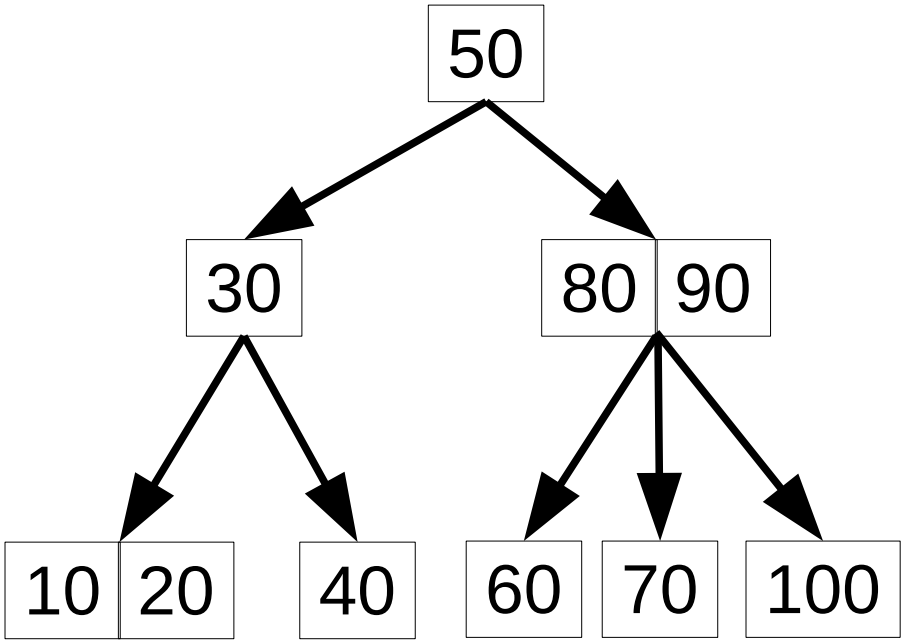
- Quando o novo nó precisa ser dividido, um novo nó raiz é criado.
- O nó raiz original se torna o filho esquerdo e o novo nó se torna o filho do meio.



Remoção

- Remover nó 70.

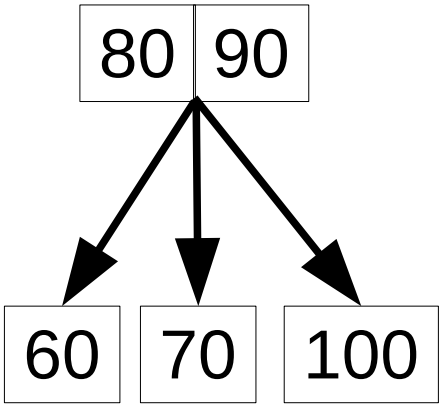




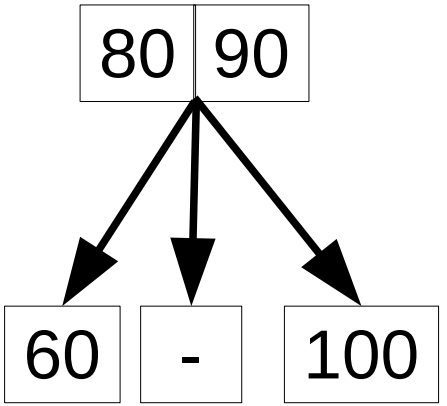
Trocar a chave (70) pelo seu sucessor (80).

Remoção

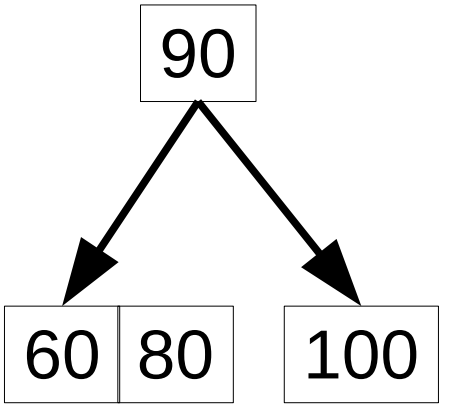
- Remover nó 70 (continuação)...



Remover a folha (70).

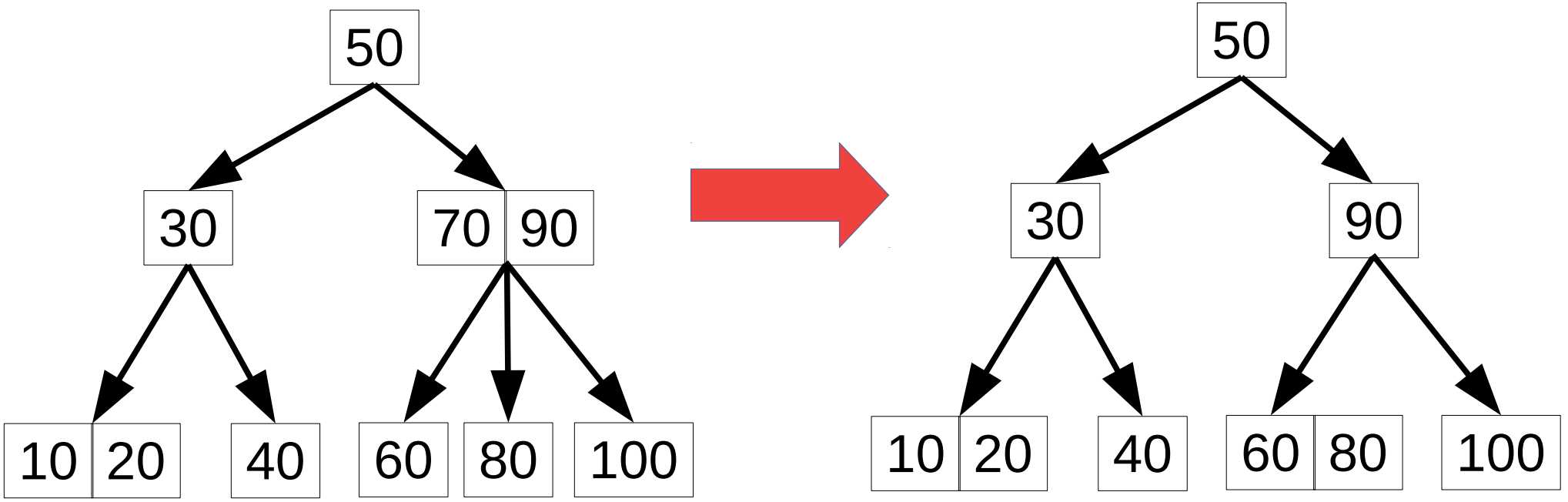


Unir os nós da folha vazia
e mover o nó 80 para baixo.



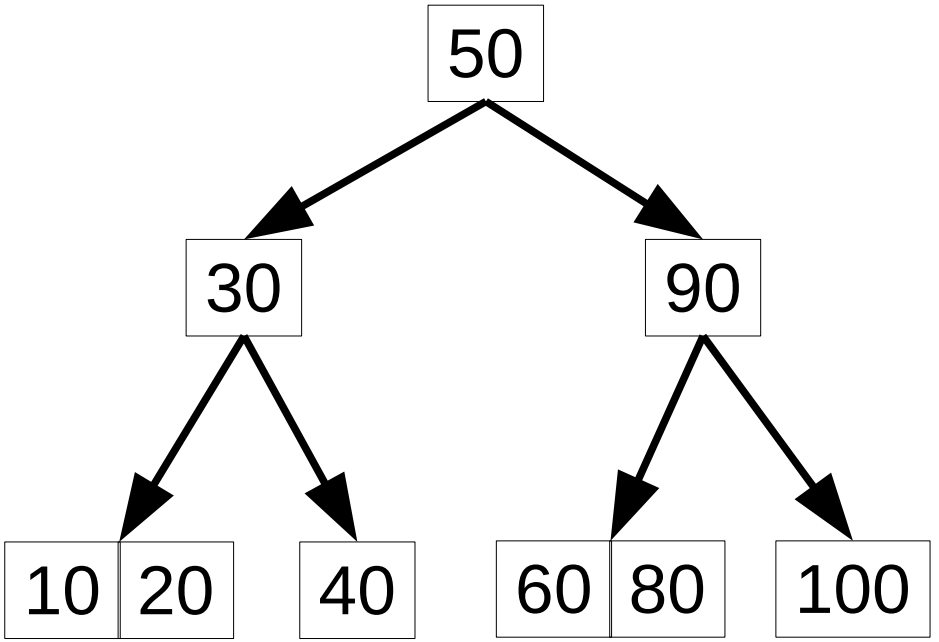
Remoção

- Remover nó 70 (continuação)...



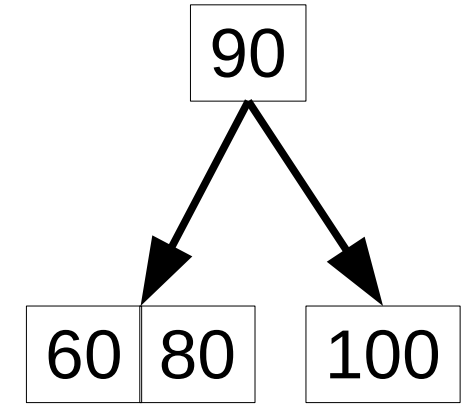
Remoção

- Remover nó 100

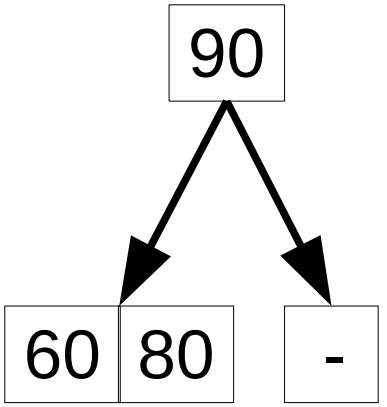


Remoção

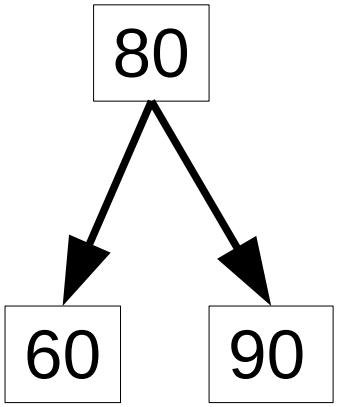
- Remover nó 100 (continuação)...



Remover a folha (100).

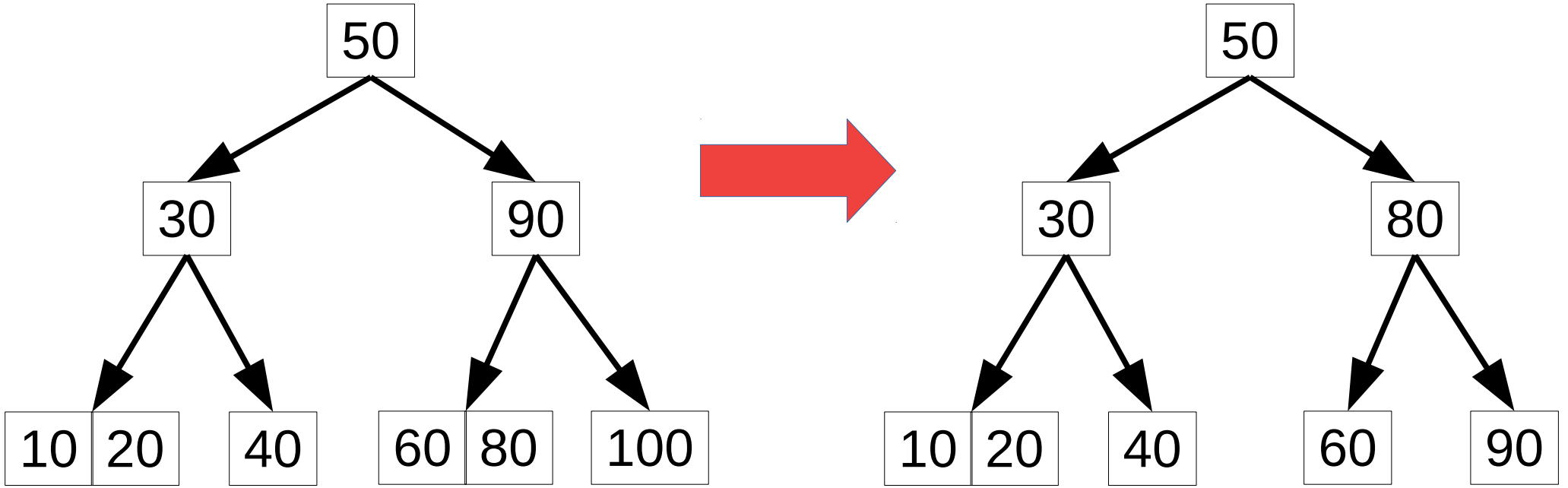


Redistribuição



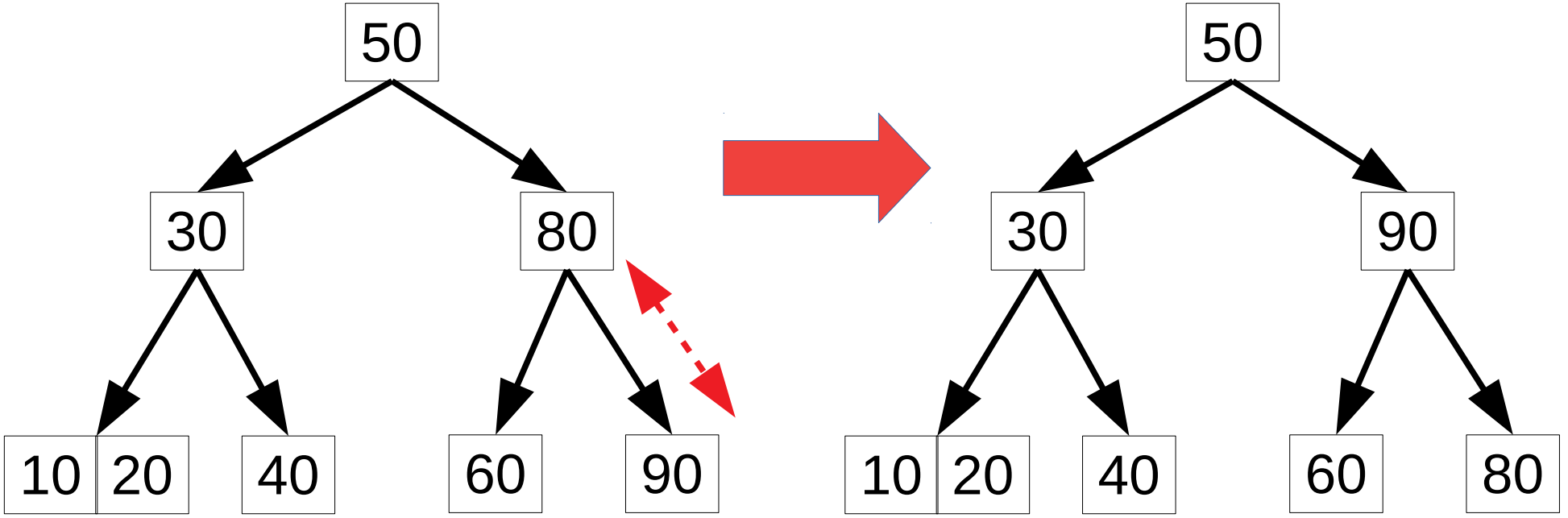
Remoção

- Remover nó 100 (continuação)...



Remoção

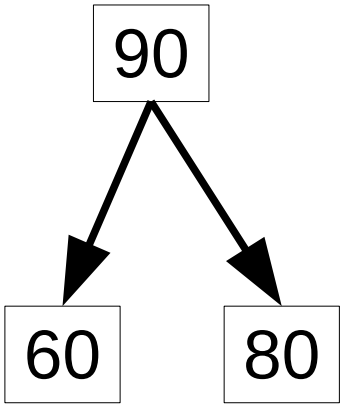
- Remover nó 80.



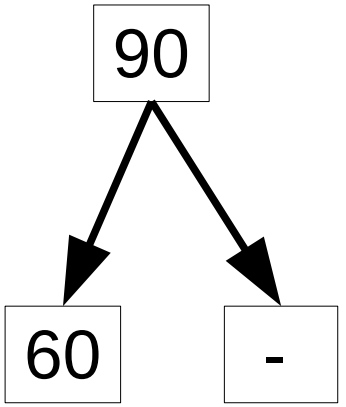
Trocar a ordem com o seu sucessor!

Remoção

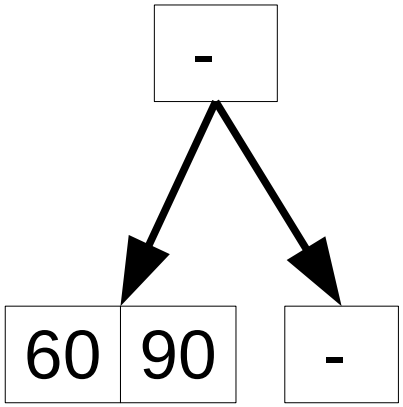
- Remover nó 80 (continuação)...



Remover a folha (80).

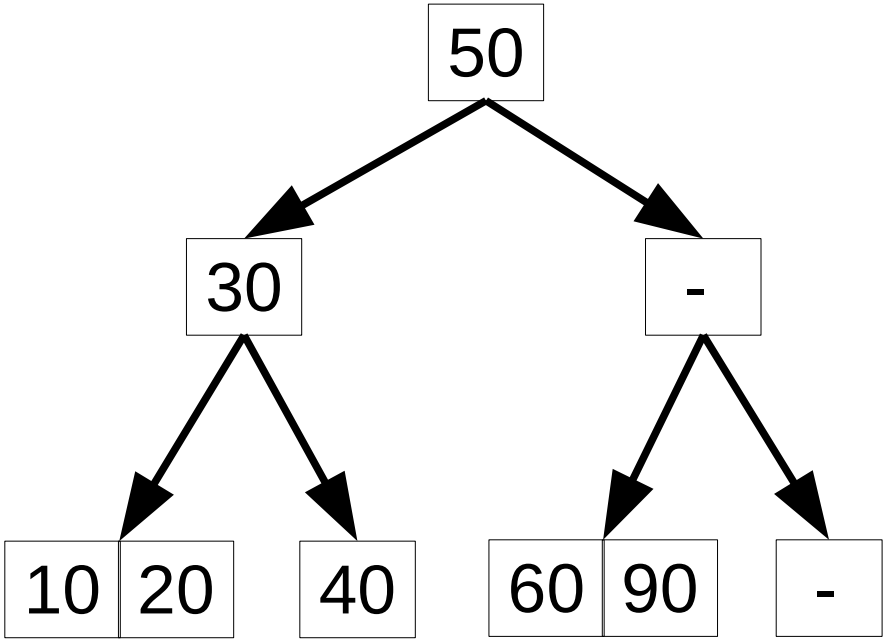


Unir os nós da folha vazia
e mover o nó 90 para baixo.

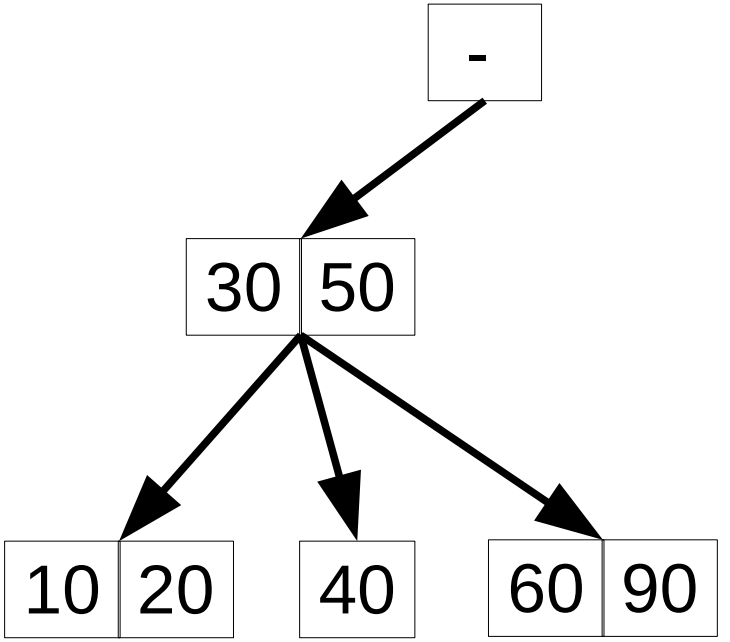


Remoção

- Remover nó 80 (continuação)...



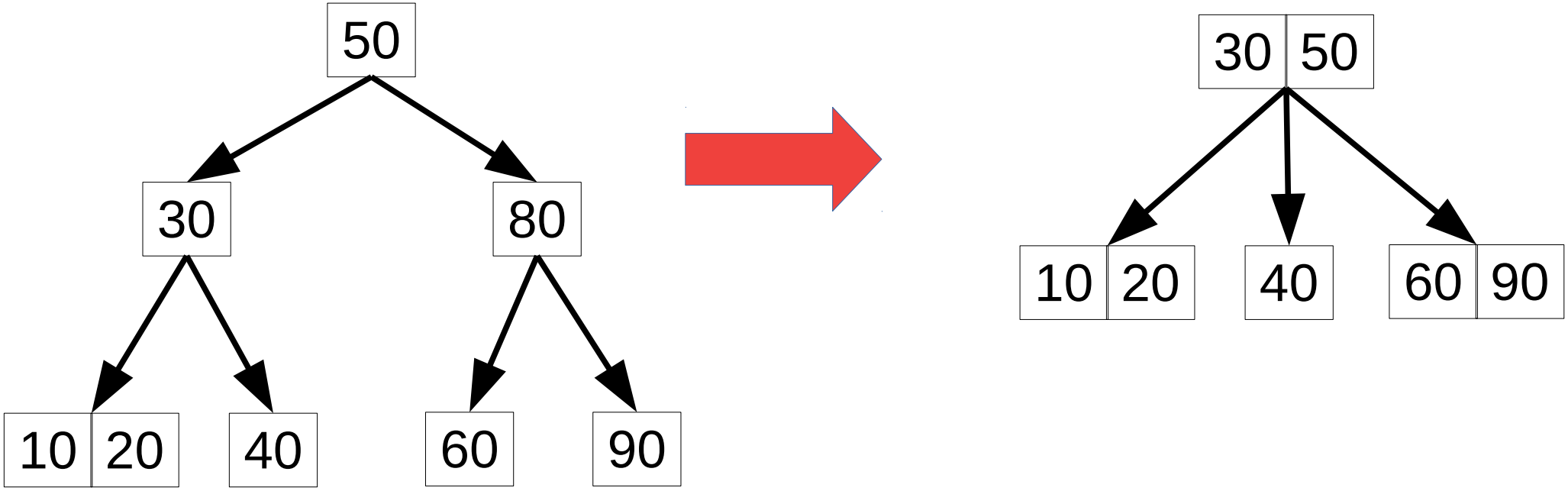
Unir os nós da folha vazia
e mover o nó 50 para baixo.



Remove o nó raiz.

Remoção

- Remover nó 80 (continuação)...

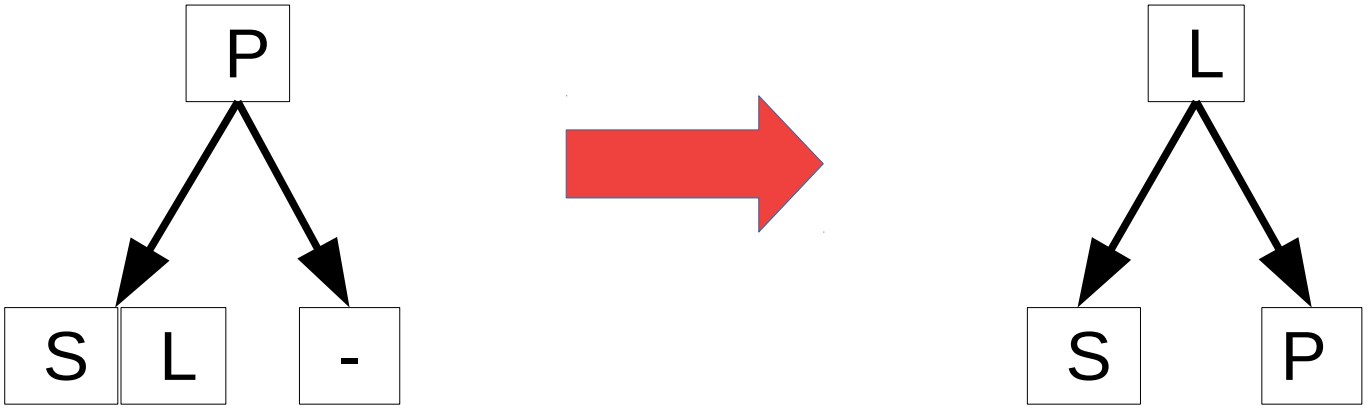


Remoção

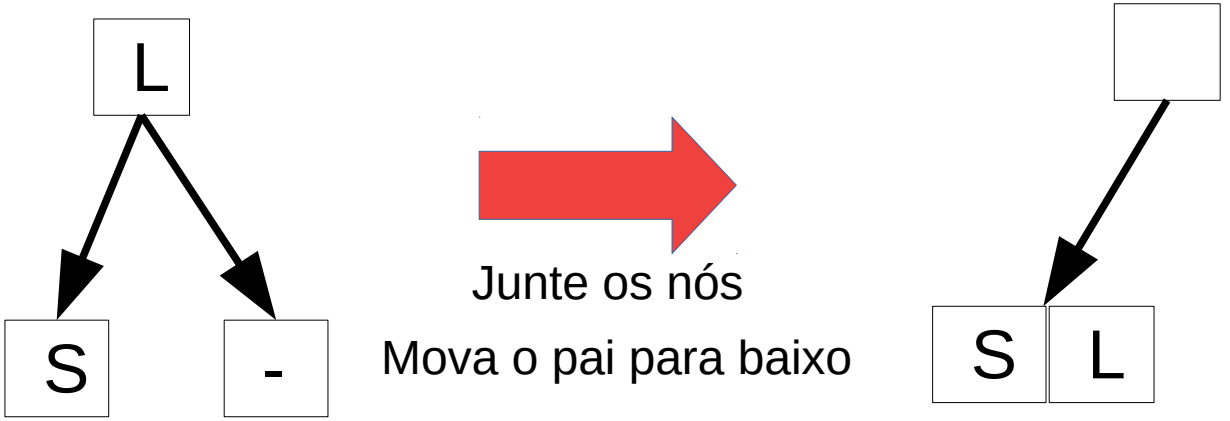
- Algoritmo Remover X
  1. Localizar o nó N que contém o nó X.
  2. Se N não é uma folha
     1. Trocar X por seu sucessor.
     2. Remoção sempre será nas folhas.
  3. Se o nó folha N contém outro item, apague X, senão, tente redistribuir os nós irmãos, se não for possível, junte os nós.

Remoção

- Redistribuição

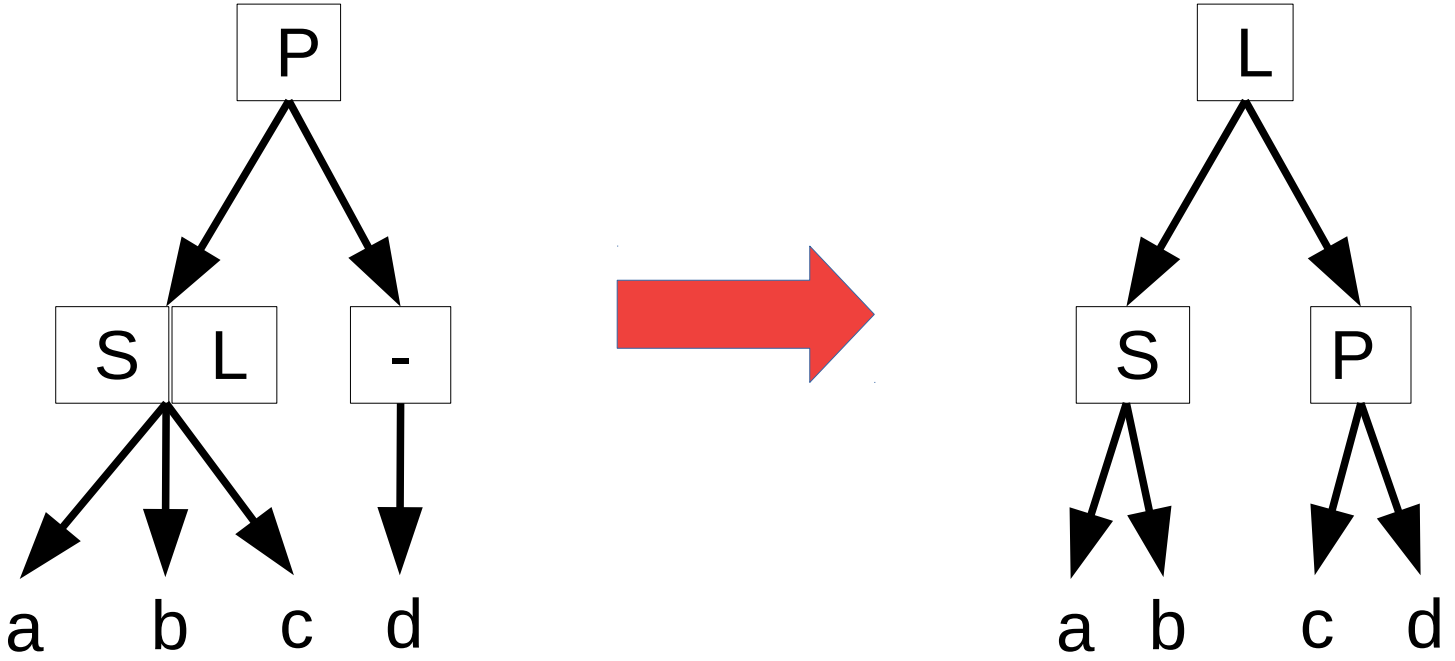


- Juntando (*merge*)



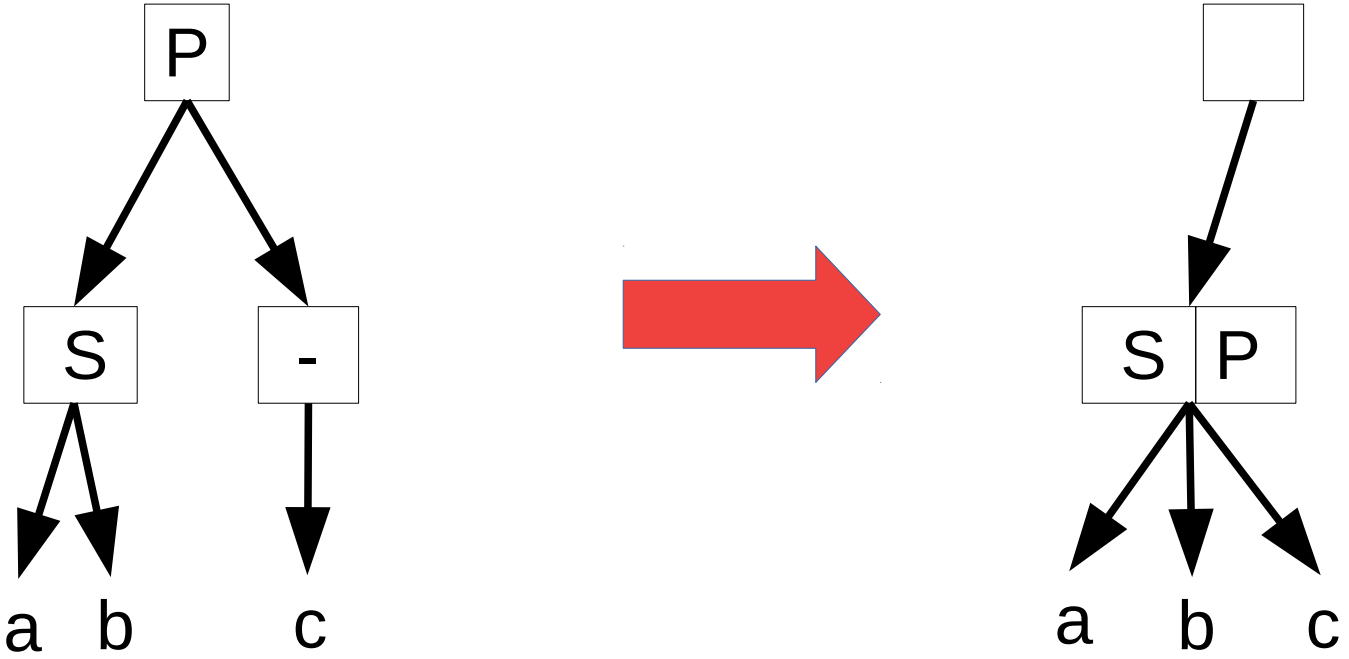
Remoção

- Redistribuição
  - Nó interno não tem item a esquerda.



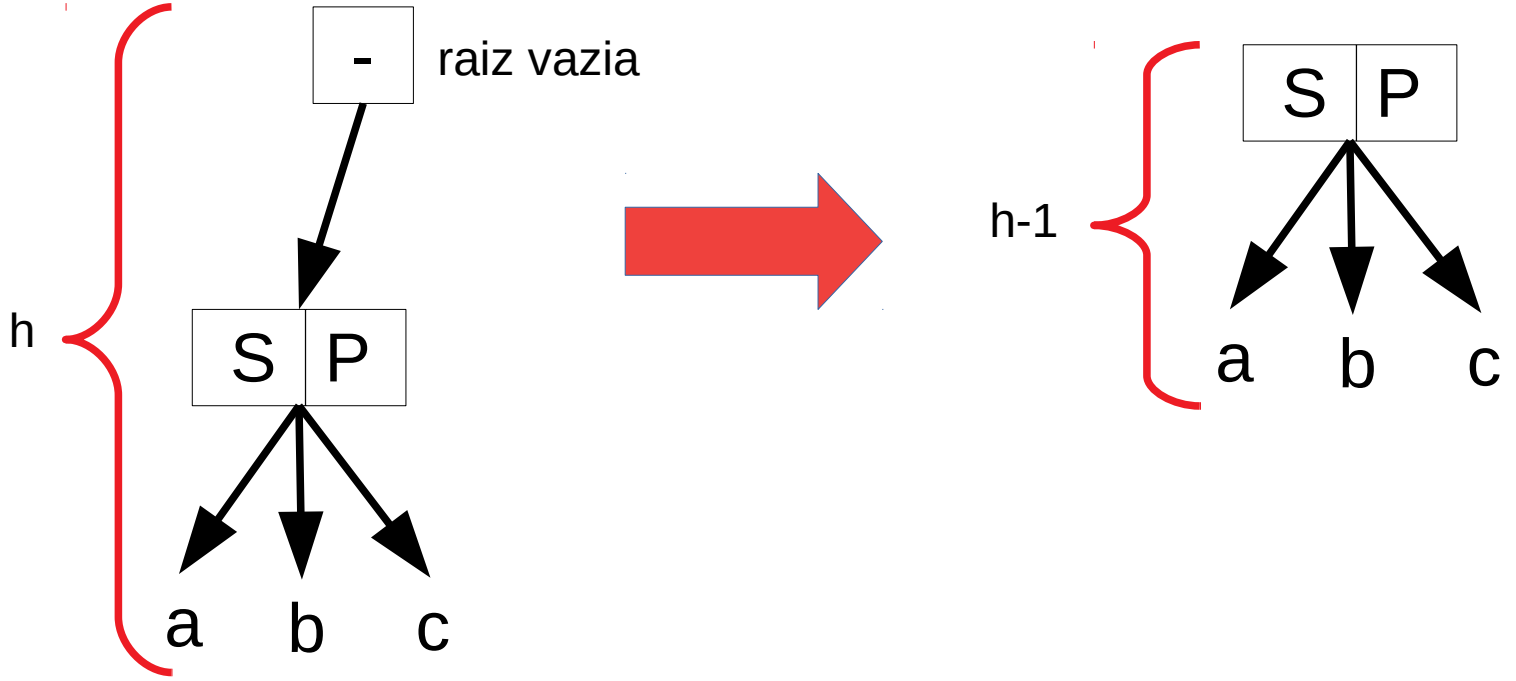
Remoção

- Juntando (*merge*)
  - Redistribuição não é possível.

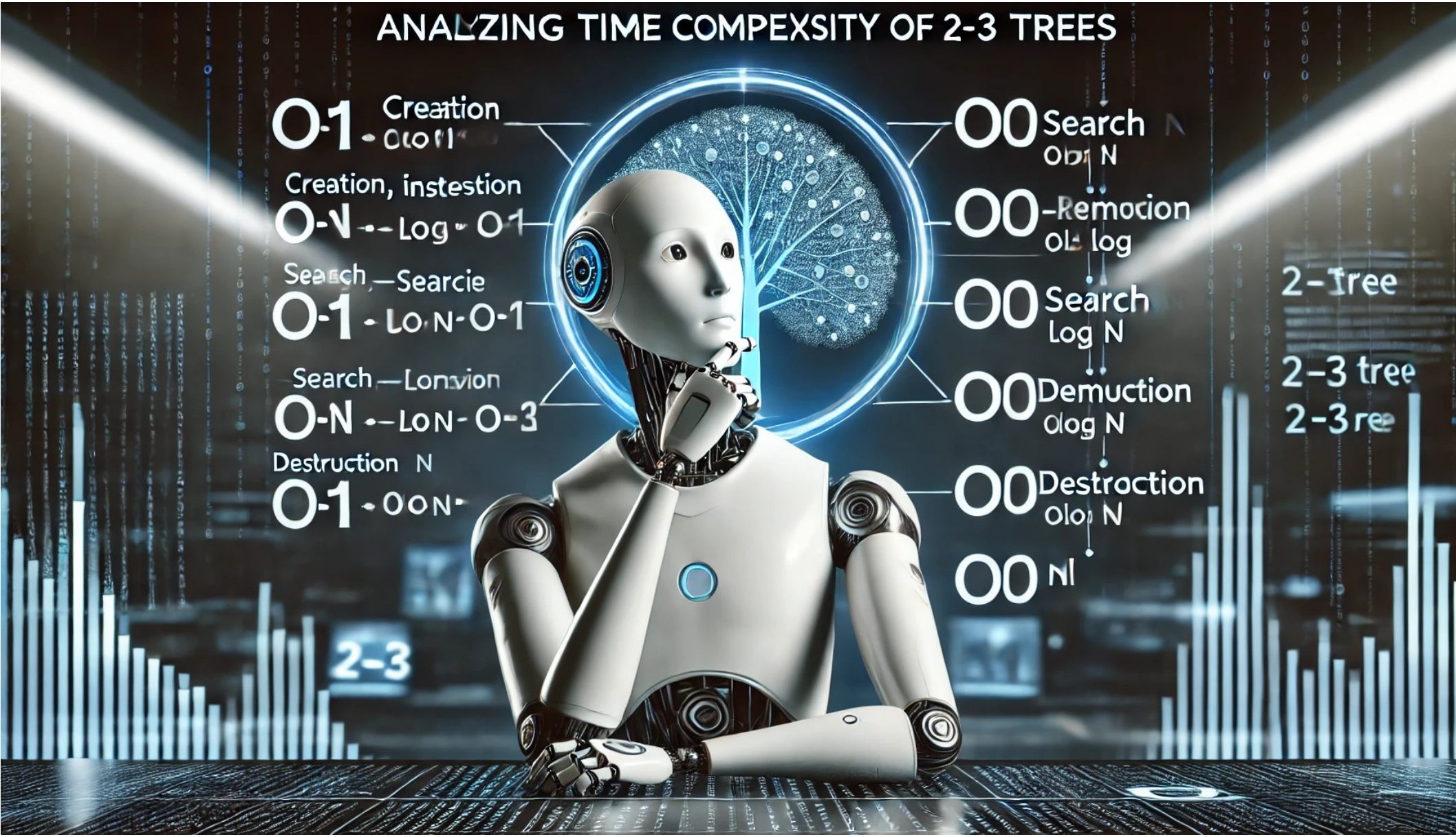


Remoção

- Se o processo de *merging* chegar até a raiz e a raiz estiver vazia, então remova a raiz.



Análise de Ordem de Complexidade



Ordem de Complexidade

- Criação da árvore
  - Complexidade: O(1)
  - Explicação:
    - A criação de uma árvore 2-3 vazia envolve apenas a inicialização de uma estrutura vazia.
    - Não há elementos para processar ou alocar.

Ordem de Complexidade

- Busca
  - Complexidade: O(log n).
  - Explicação:
    - A busca segue o caminho da raiz até um nó folha.
    - Devido ao balanceamento da árvore, a altura é O(log n).
    - Portanto, a busca requer O(log n) comparações no pior caso.

Ordem de Complexidade

- Inserção
  - Complexidade: O(log n).
  - Explicação:
    - A árvore 2-3 mantém-se balanceada, então a altura da árvore é O(log n).
    - A inserção requer uma busca para encontrar a posição correta, o que leva O(log n) no pior caso.
    - Se a inserção causa um transbordo, pode ser necessário dividir nós, mas cada divisão e propagação para cima também é O(log n).

Ordem de Complexidade

- Remoção
  - Complexidade: O(log n).
  - Explicação:
    - A remoção pode envolver uma busca inicial para localizar o elemento, que é O(log n).
    - A remoção pode necessitar de fusões ou redistribuições para manter as propriedades da árvore, mas essas operações também são limitadas pela altura da árvore, O(log n).
    - No pior caso, a remoção e todas as operações de reequilíbrio são O(log n).

Ordem de Complexidade

- Destruição
  - Complexidade: O(n).
  - Explicação:
    - Destruir uma árvore 2-3 envolve liberar a memória de todos os nós.
    - Cada nó deve ser visitado uma vez, e como há n nós, a complexidade é O(n).

Ordem de Complexidade

- Resumo das complexidades:

Operação	Complexidade
Criação	O(1)
Inserção	O(log n)
Busca	O(log n)
Remoção	O(log n)
Destruição	O(n)

Exercícios

Exercícios

- Quais são as propriedades de uma árvore 2-3?
- Descreva as diferenças entre um nó 2-nó e um nó 3-nó em uma árvore 2-3.
- Quantas chaves, no máximo, pode conter uma árvore 2-3 de altura 2? Qual o número mínimo de chaves em uma árvore 2-3 de altura 2? Pense e/ou desenhe exemplos dessas árvores.
- Desenhe a árvore 2-3 que resulta da inserção das chaves [10, 1, 20, 30, 18, 25, 24, 11, 3], nesta ordem, em uma árvore inicialmente vazia.
- O que representa o 2-3 no nome da árvore 2-3?
- Os dados numa árvore 2-3 são mantidos ordenados? Justifique a sua resposta.

Exercícios

- Qual é a vantagem do uso de árvores 2-3 quando comparadas com árvores binárias de busca?
- Marque alternativa correta a respeito da ordem de complexidade da operação de busca em uma árvore 2-3:

  a) $\Theta(1)$

  b) $\Theta(N)$

  c) $\Theta(\log_2 N)$

  d) $\Theta(\log_3 N)$

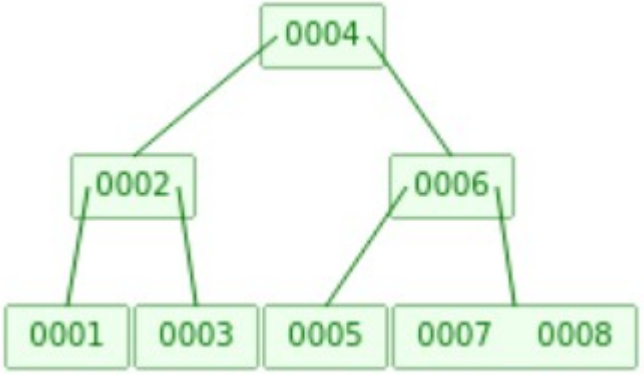
Exercícios

- Considere a árvore 2-3 abaixo e responda ao que se pede:

  a) Como ficaria a árvore após a remoção do nó 0008?

  b) Como ficaria a árvore após a remoção do nó 0006?

  c) Se eu realizar a operação de remoção do elemento 0006 e, em seguida, a operação de inserção do elemento 0006, as árvores antes e depois das operações seriam idênticas?



Exercícios

- Qual é o número máximo de elementos que podem ser inseridos na árvore abaixo sem aumentar a sua altura?