Estruturas de Dados 1

481440

Julho/2018

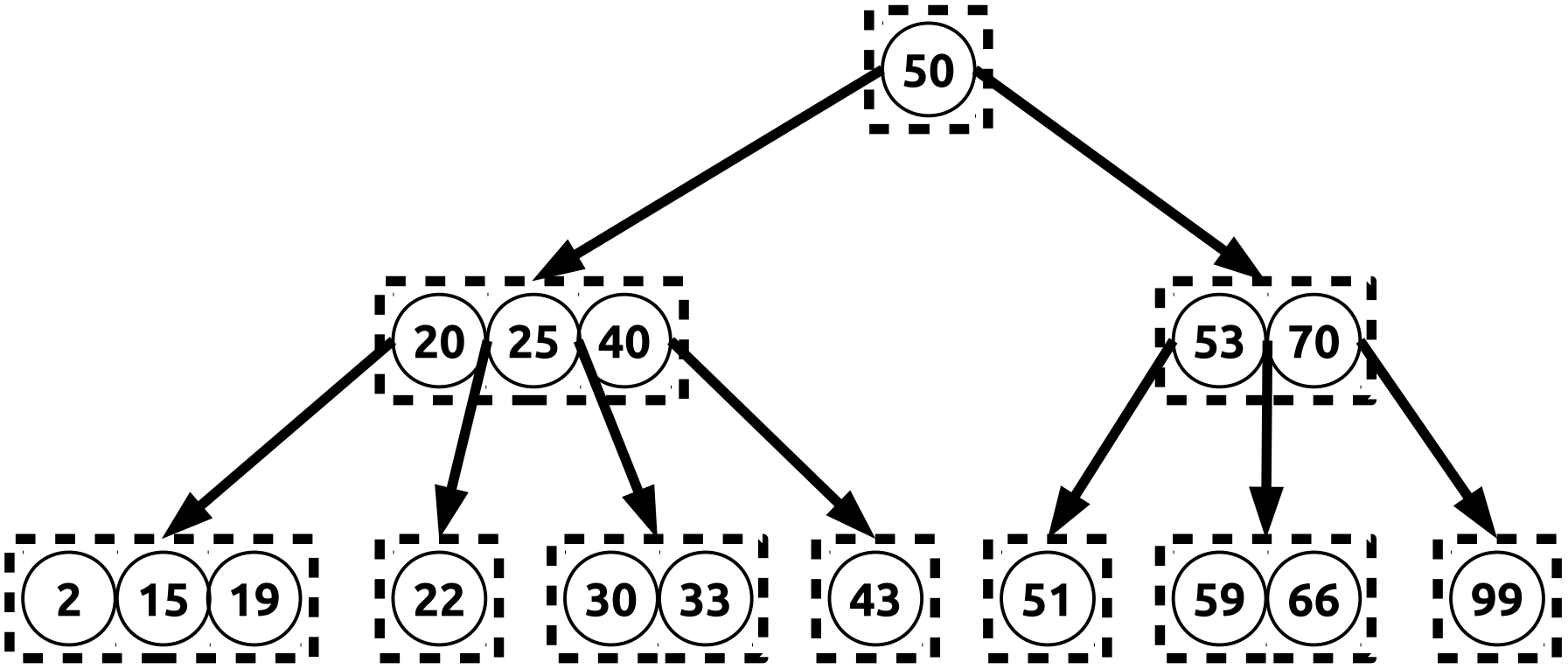
Mario Liziér
lizier@ufscar.br

Árvores balanceadas

- AVL
  - Binária de busca
  - Define um critério de balanceamento: -1, 0 e 1
  - Inserção / Remoção
- Árvore 2-3 (ou 2-4)
  - Árvore *n-ária de busca (multi-way)*
  - Base para árvores RB e B (para HDs)
  - Auxilia no entendimento da RB
- Árvore *Red-Black* (RB)
  - Binária de busca
  - Correspondência com a árvore 2-3
  - Padrão do C++ e Java

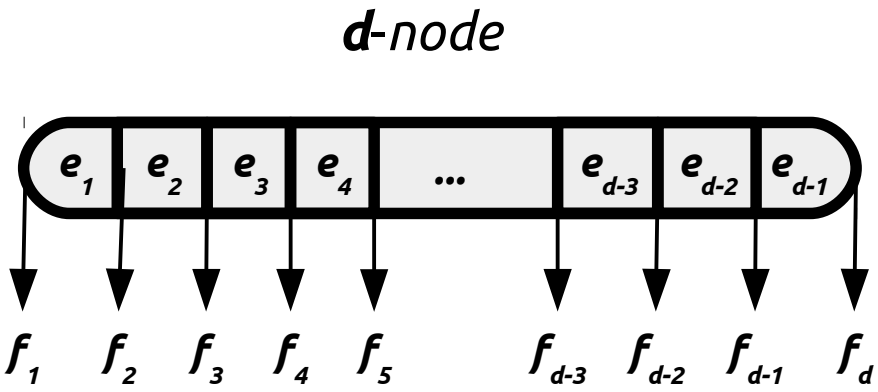
Árvores *multi-way*

- Árvores *n-árias* de busca (grau >= 2)



Árvores *multi-way* (***d*-*node***)

- Um ou mais elementos por nó:
  - $e_1, e_2, e_3, \ldots, e_{d-1}$
  - $e_1 \leq e_2 \leq e_3 \leq \ldots \leq e_{d-1}$
- Dois ou mais filhos por nó:
  - $f_1, f_2, f_3, \ldots, f_d$
- Árvore de busca:
  - Sub-árvore f_i contém elementos maiores que e_{i-1} e menores que e_i, sendo $e_0 = -\infty$ e $e_d = +\infty$

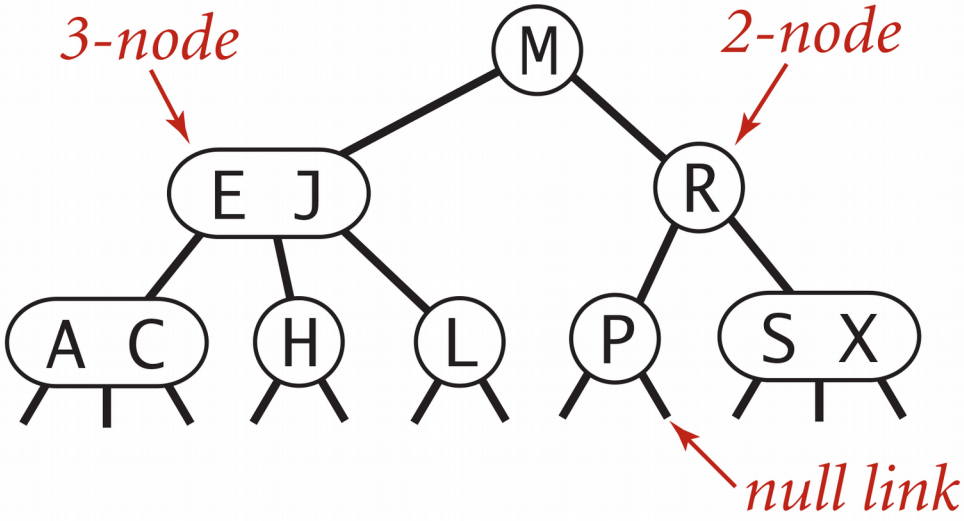


Árvores *multi-way* (***d*-node**)

- Podemos conseguir “facilmente”
  - Balanceamento “Natural”
  - Crescimento de “baixo” para “cima”

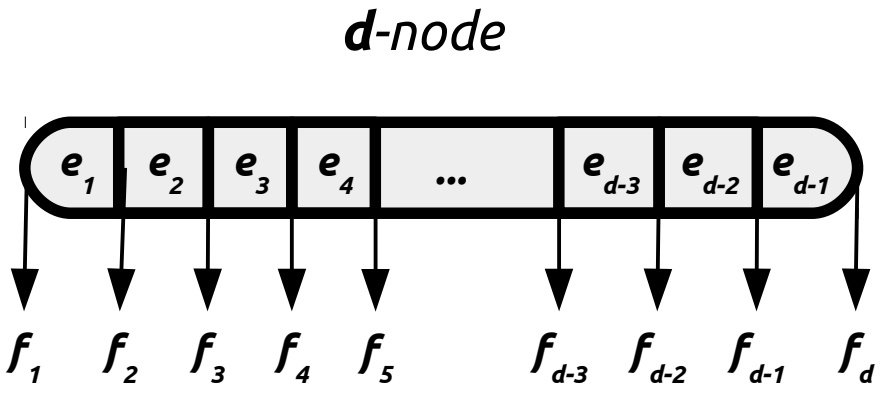
Veremos isso
na inserção/remoção

- Exemplo de árvore com:
  - *2-nodes* e *3-nodes*
  - chamada de árvore 2-3



Árvores *multi-way* (***d*-node**)

- Buscas:
  - Semelhante a árvore binária de busca, mas …
    - Podemos ter mais de 1 elemento por nós
    - Se não encontrarmos o elemento procurado naquele nó, precisamos definir uma subárvore para prosseguir



Árvore 2-4 ou 2-3-4

- Árvore formada apenas por:
  - *2-nodes*
  - *3-nodes*
  - *4-nodes*
- Todos os nós folhas possuem a mesma profundidade
  - Balanceamento perfeito!

Basta limitarmos em 3 o número de elementos por nó

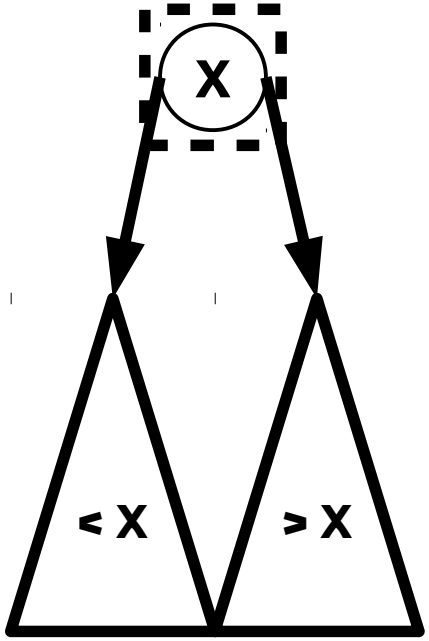
Como conseguimos isso?

- Base para a árvore *Red-Black*
  - A árvore RB é uma binarização da árvore 2-4
- Outro caminho: estudar as árvores 2-3, para depois binarizar para a RB

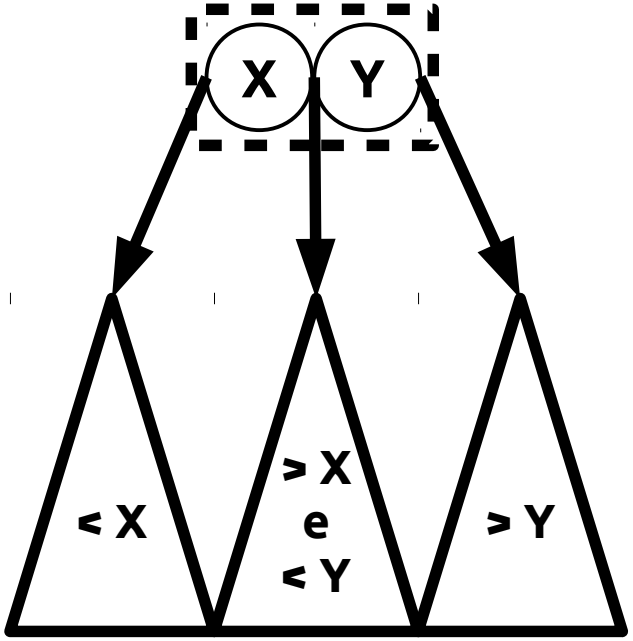
Árvore 2-4 - Busca

- Buscas

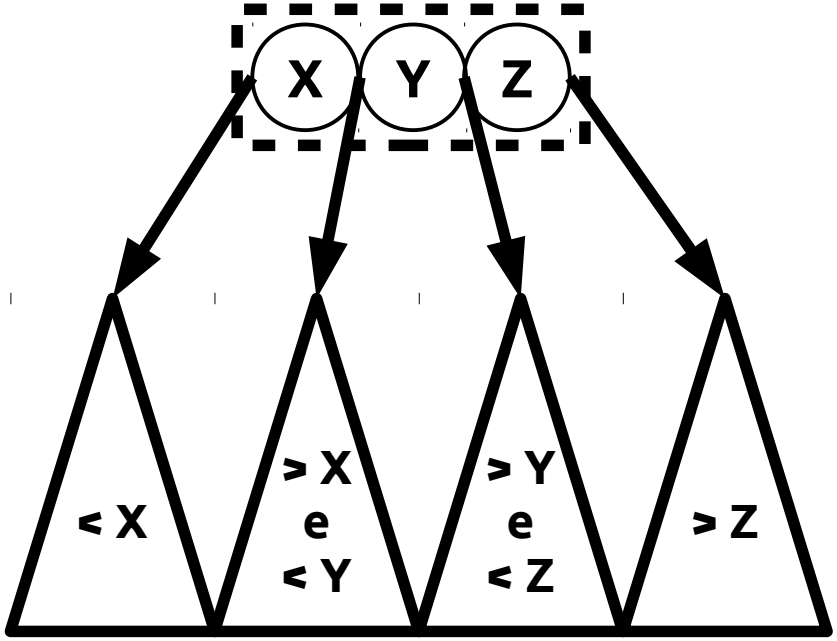
Para um *2-node* temos:



Para um *3-node*,
onde X < Y, temos:

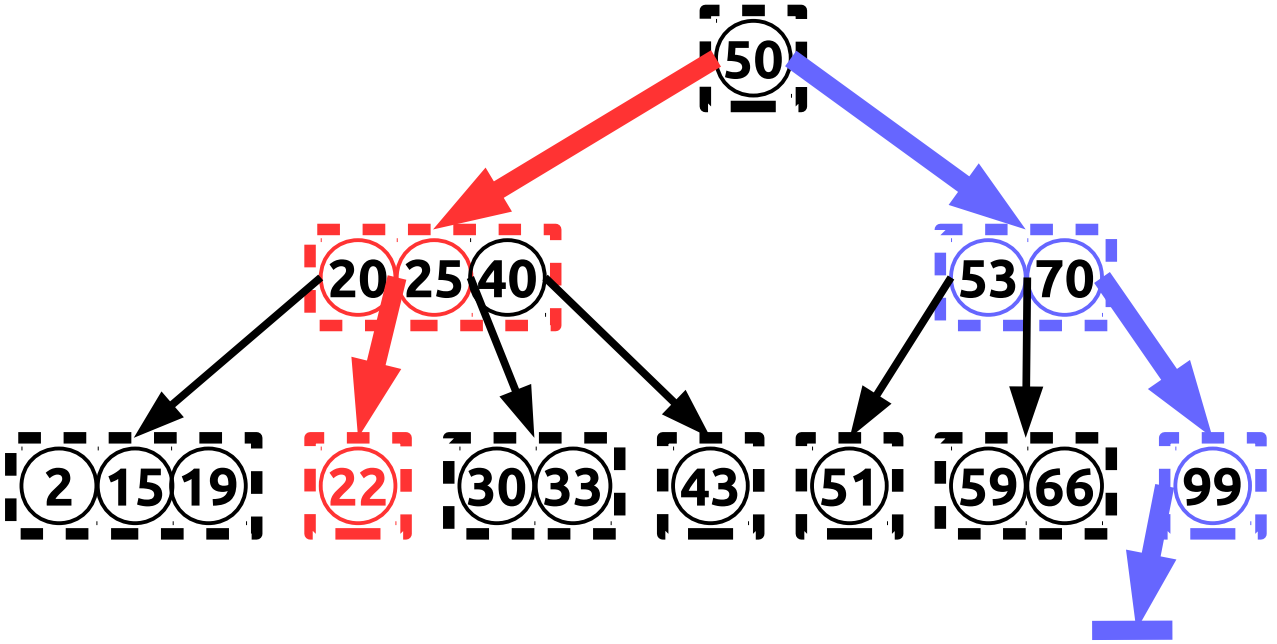


Para um *4-node*,
onde X < Y < Z, temos:

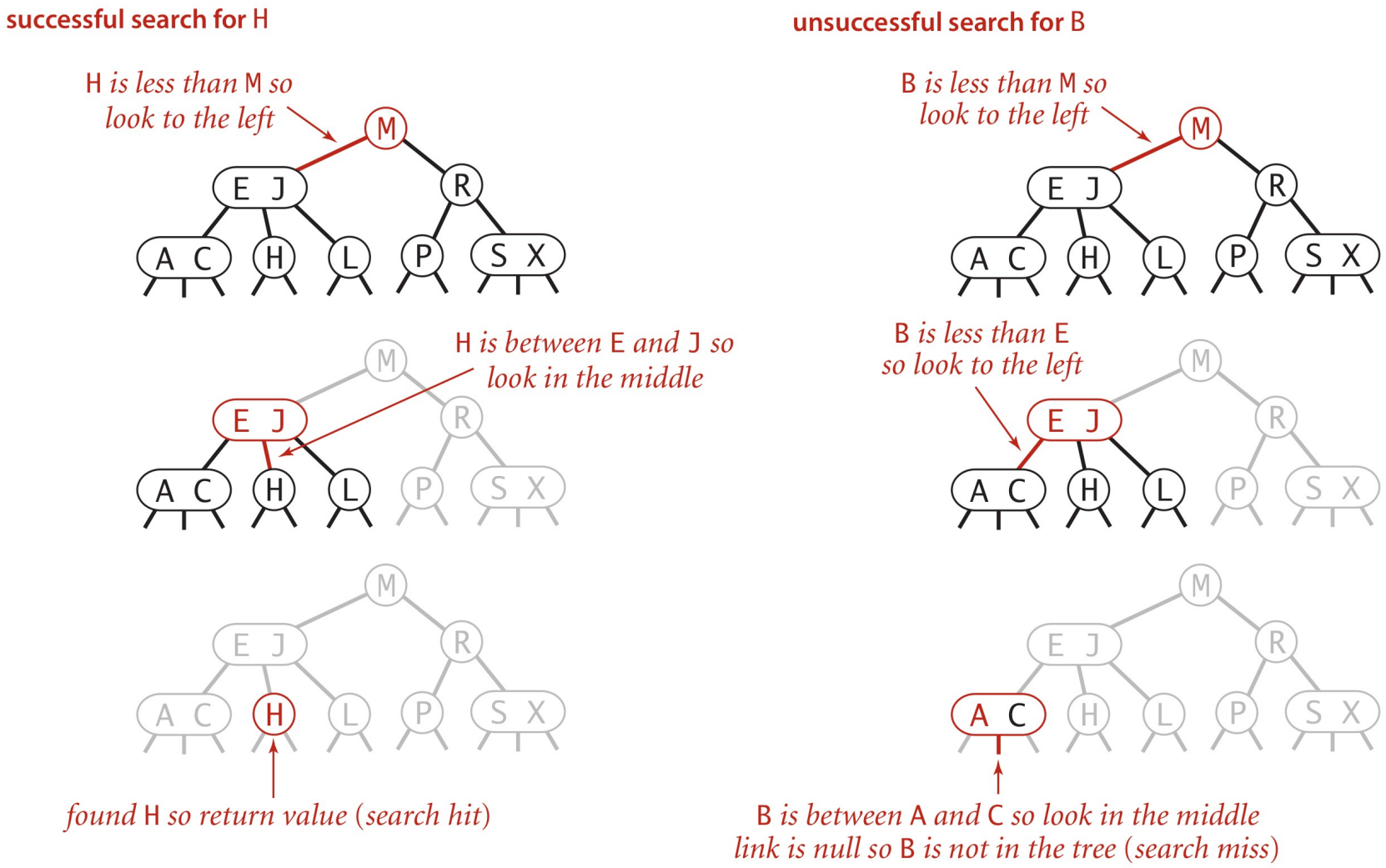


Árvore 2-4 - Busca

- Buscas:
  - Exemplos:
    - find(22) → encontrado!
    - find(80) → não encontrado!



Exemplo: Busca em árvore 2-3



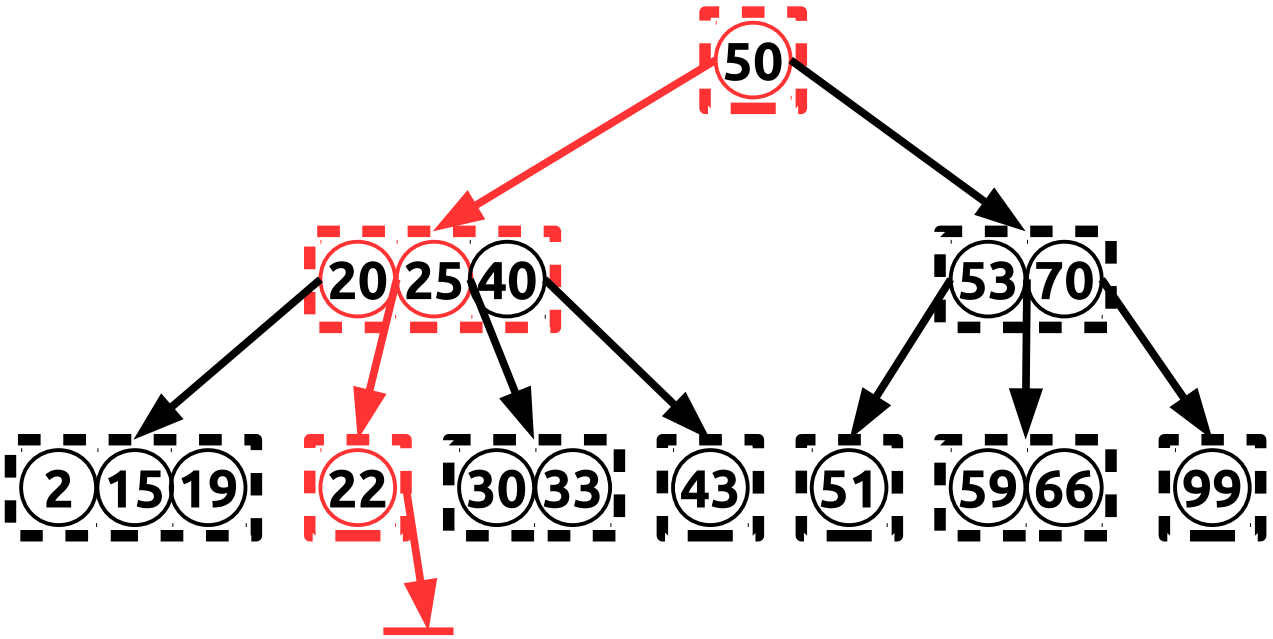
Search hit (left) and search miss (right) in a 2-3 tree

Árvore 2-4 - Inserção

- Inserção (algoritmo parcial):
  - Se a árvore está vazia:
    - Criamos um *2-node* com o elemento a ser inserido
  - Senão:
    - Buscamos pela posição correta (algoritmo de busca comum)
    - E inserimos no nó folha (e não no filho *null*)

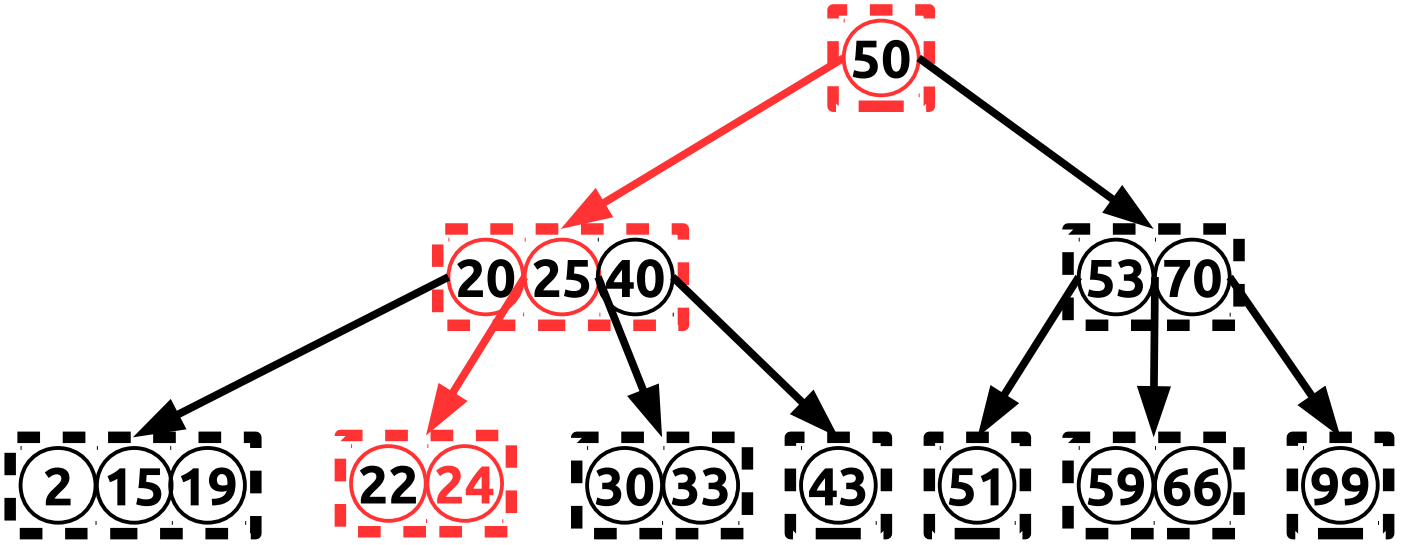
Árvore 2-4 - Inserção

- Caso 1: o nó folha é um *2-node*
  - Exemplo:
    - inserir(24)
      - Ida: Busca recursiva



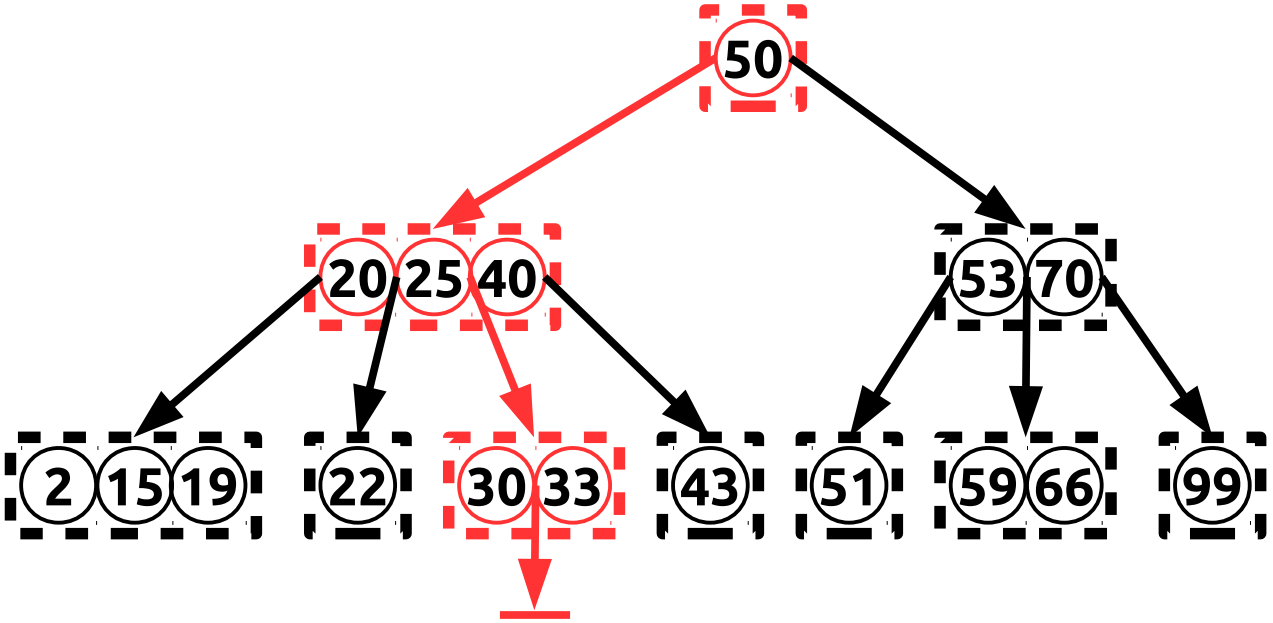
Árvore 2-4 - Inserção

- Caso 1: o nó folha é um *2-node*
  - Exemplo:
    - inserir(24)
      - volta: inserir no nó folha



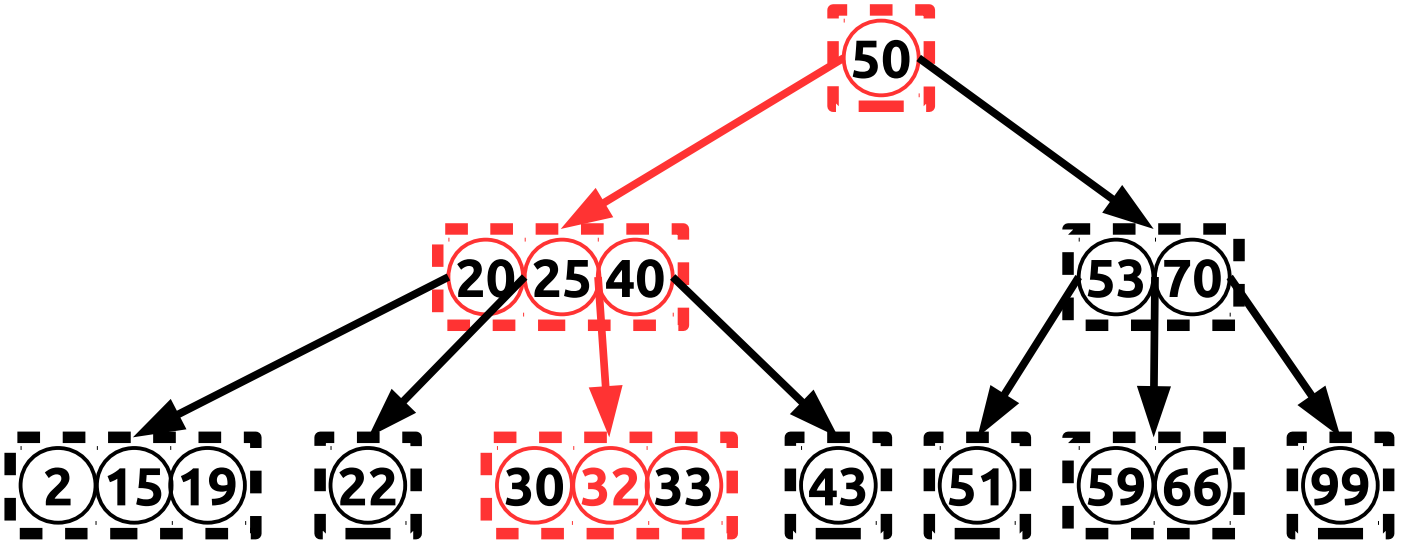
Árvore 2-4 - Inserção

- Caso 2: o nó folha é um *3-node*
  - Exemplo:
    - inserir(32)
      - Ida: Busca recursiva



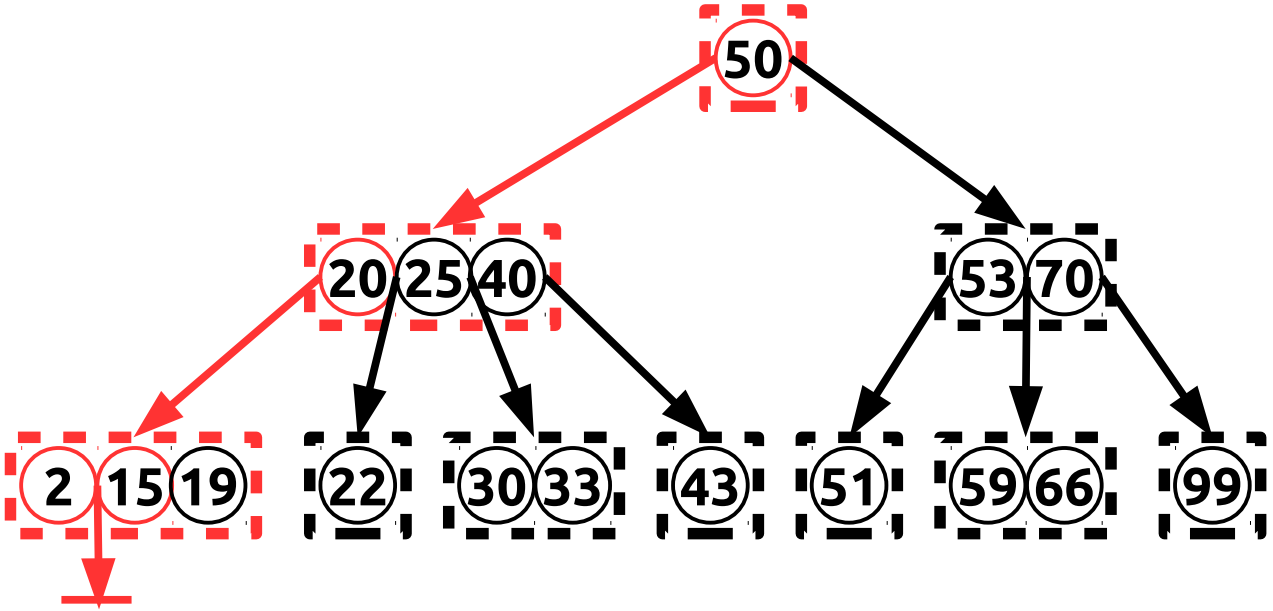
Árvore 2-4 - Inserção

- Caso 2: o nó folha é um *3-node*
  - Exemplo:
    - inserir(32)
      - volta: inserir no nó folha



Árvore 2-4 - Inserção

- Caso 3: o nó folha é um *4-node*
  - Exemplo:
    - inserir(3)
      - Ida: Busca recursiva

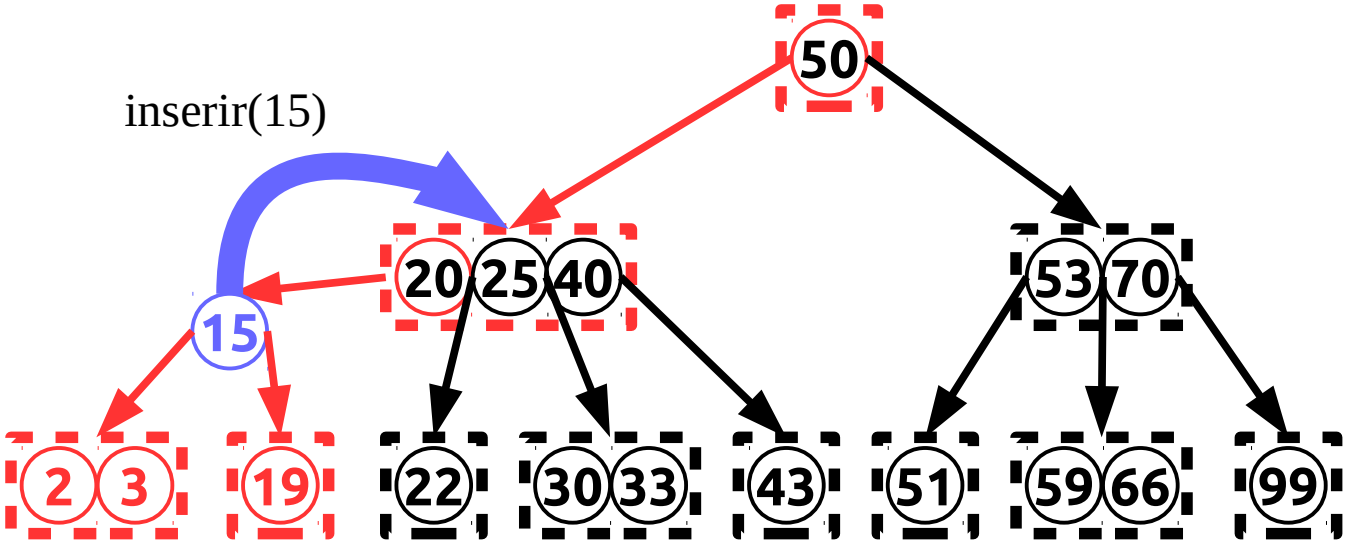


Árvore 2-4 - Inserção

- Caso 3: o nó folha é um *4-node*
  - Exemplo:
    - inserir(3)
      - Volta da recursão:
        - Não podemos ter um *5-node* (temos 4 elementos!)
        - Criamos dois novos nós (*split*):
          - *2-node* (1 elemento)
          - *3 -node* (2 elementos)
        - O elemento intermediário inserimos no nó pai
          - Ou seja, na volta da recursão o processo de inserção se repete! o nó pai pode ser um *2-node* ou *3-node* ou *4-node* (o que provocaria um novo split!)
          - A árvore cresce de baixo para cima!

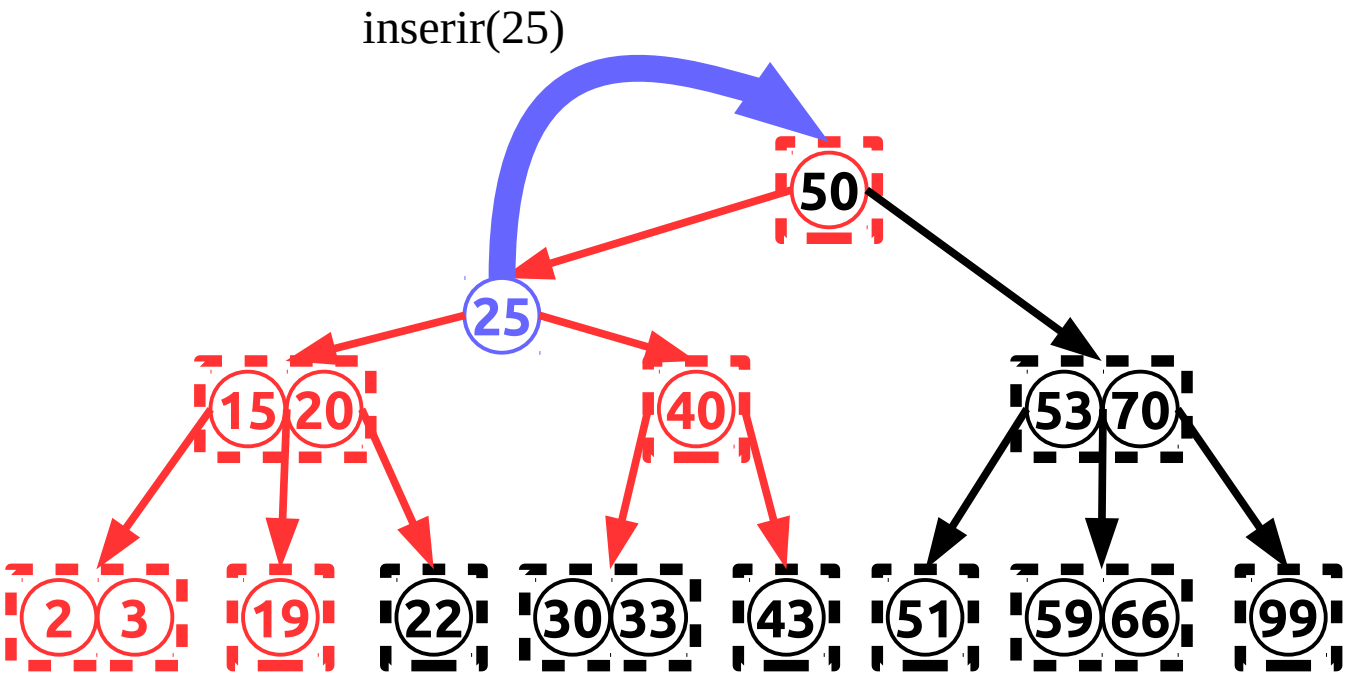
Árvore 2-4 - Inserção

- Caso 3: o nó folha é um *4-node*
  - Exemplo:
    - inserir(3)
      - Volta da recursão!



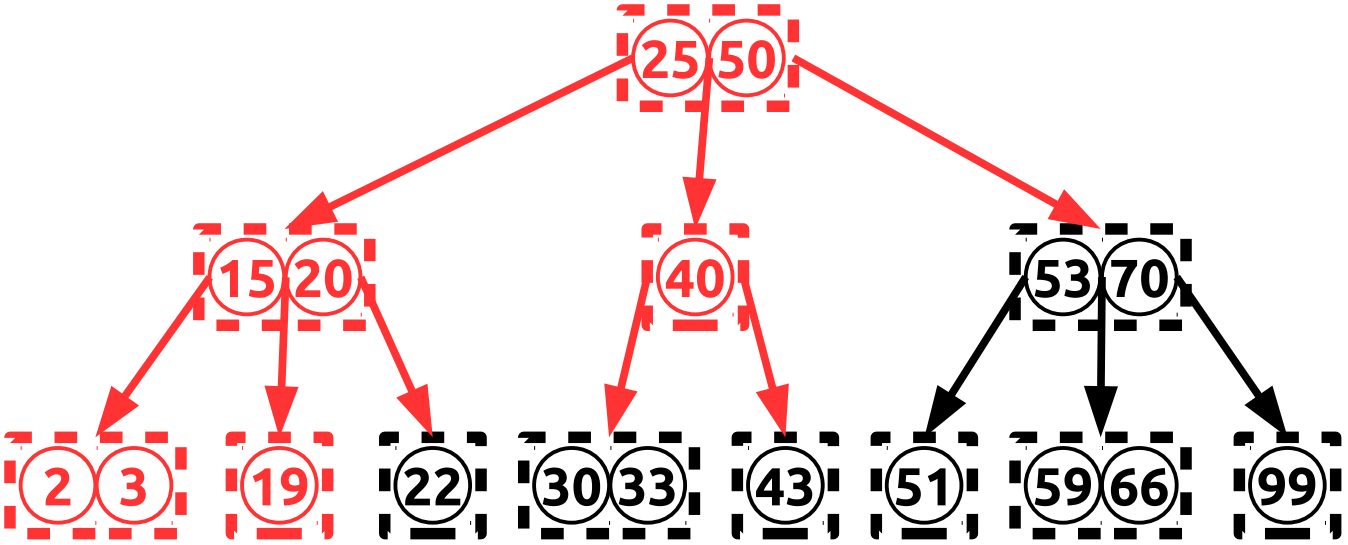
Árvore 2-4 - Inserção

- Caso 3: o nó folha é um *4-node*
  - Exemplo:
    - inserir(3)
      - Volta da recursão!



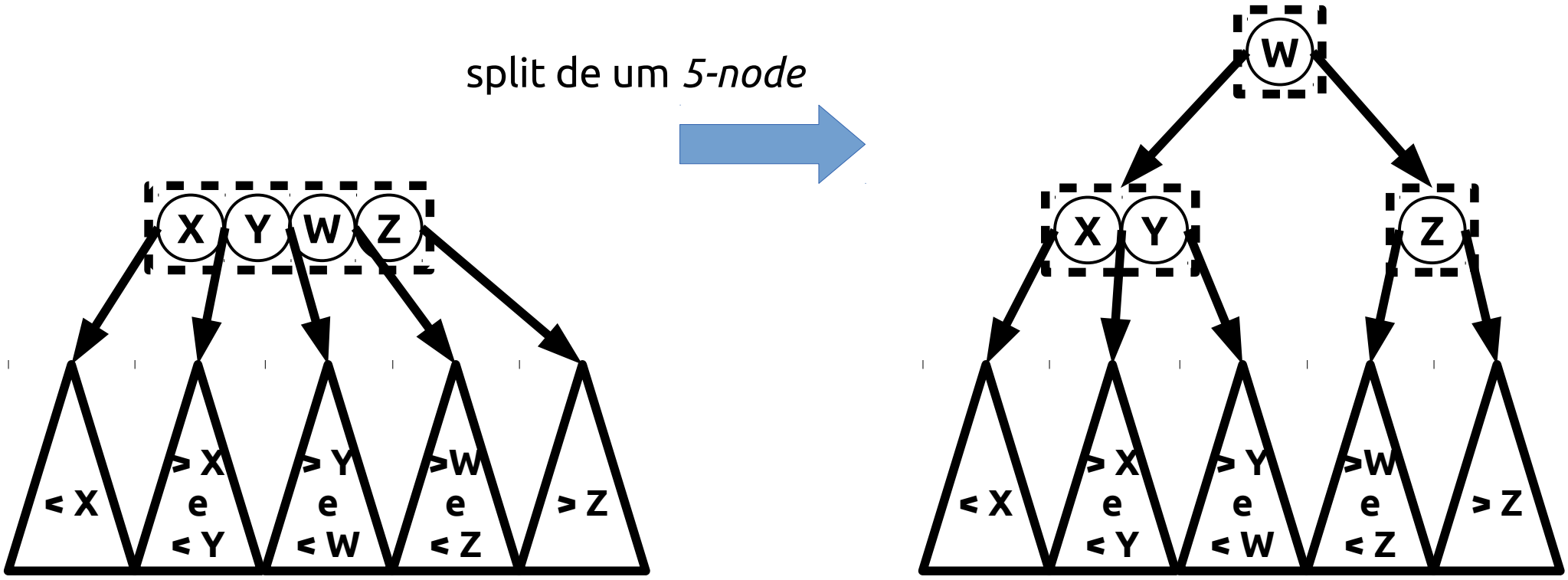
Árvore 2-4 - Inserção

- Caso 3: o nó folha é um *4-node*
  - Exemplo:
    - inserir(3)
      - Volta da recursão!

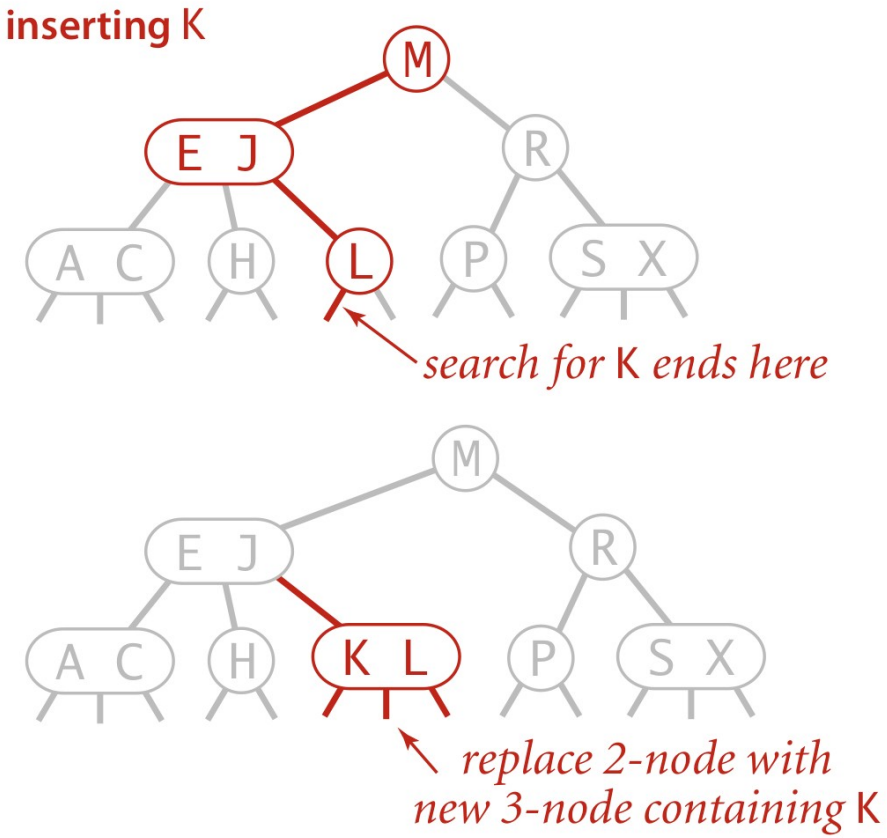


Árvore 2-4

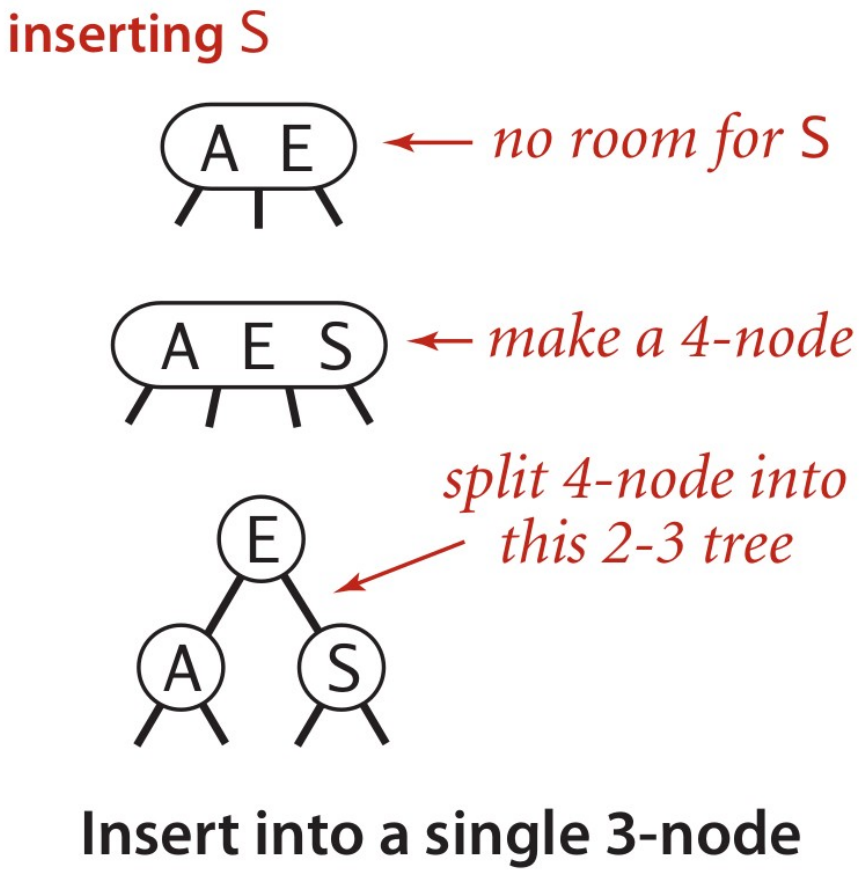
- Quando encontramos o caso 3 em toda volta da recursão:
  - A árvore cresce!



Exemplo: Inserção em árvore 2-3

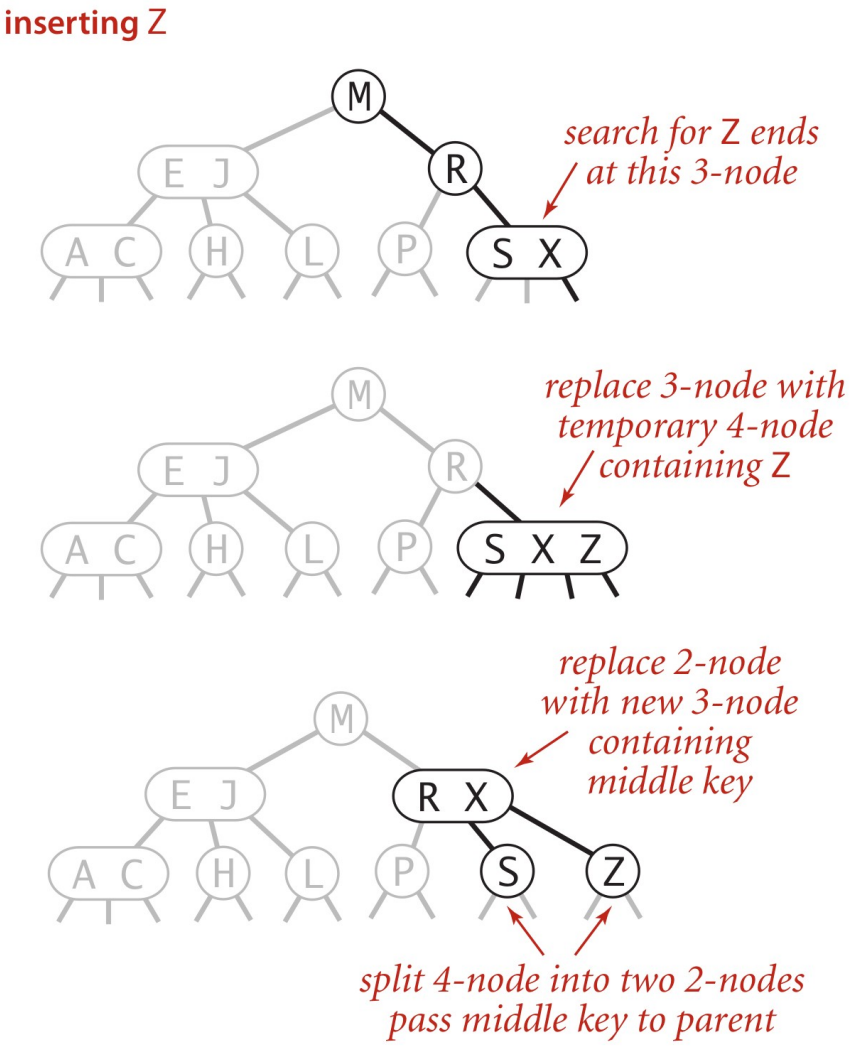


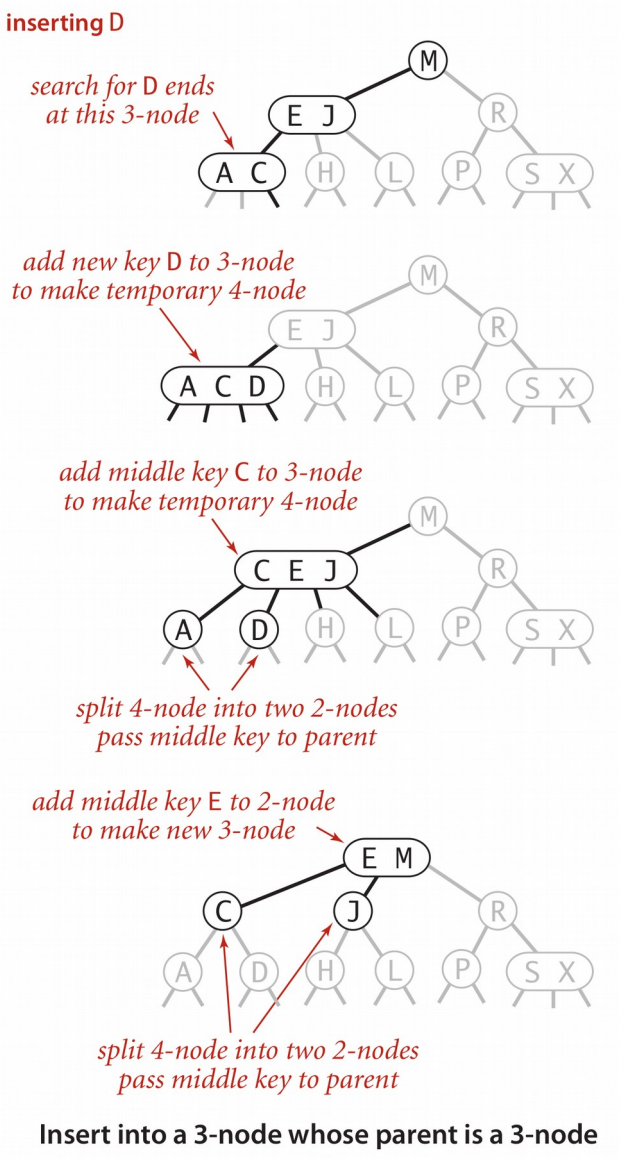
Insert into a 2-node



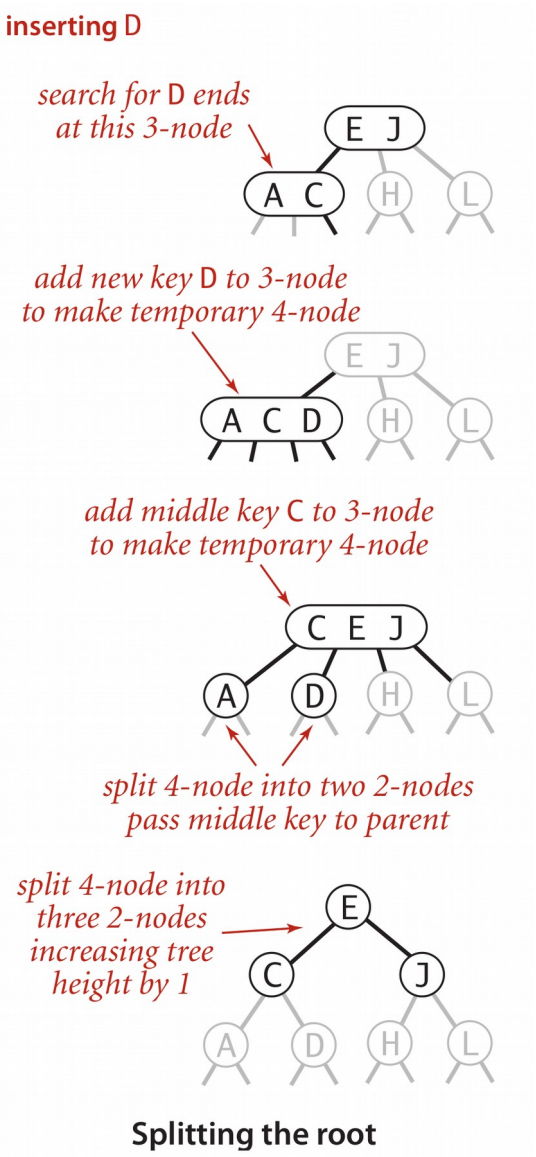
Insert into a single 3-node

Exemplo: Inserção em árvore 2-3

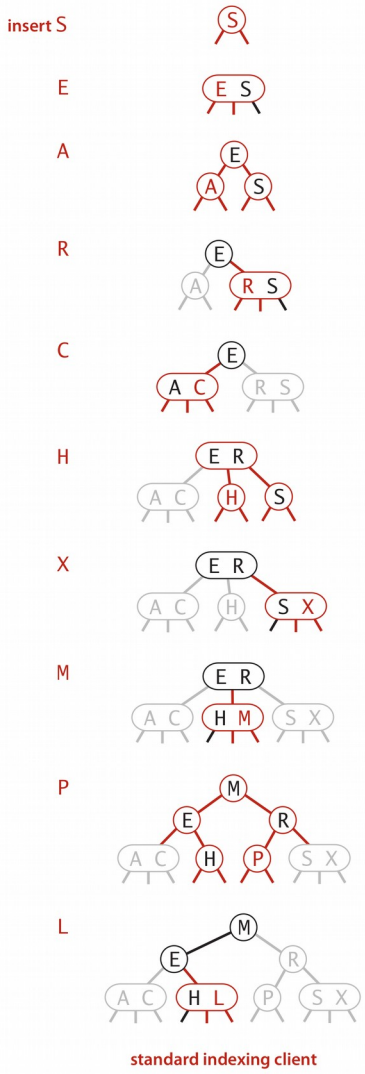




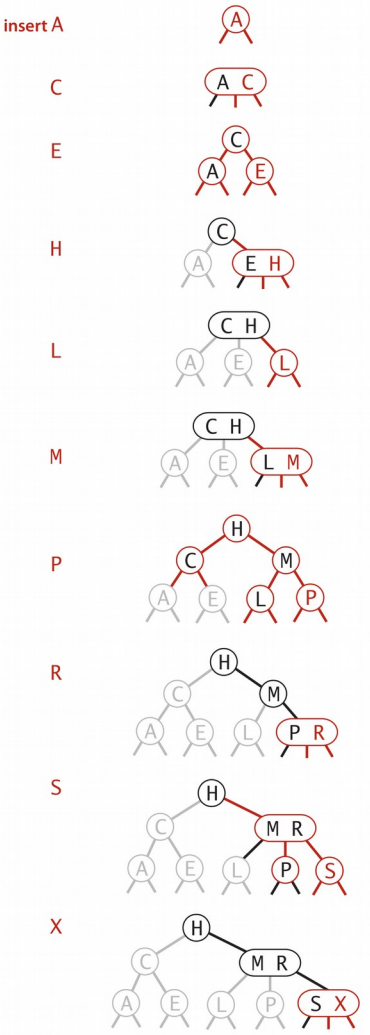
Exemplo: Inserção em árvore 2-3



Splitting the root



standard indexing client



same keys in increasing order

2-3 construction traces

Exemplo: Inserção em árvore 2-4

Insert(4);

Insert(6);

Insert(12);

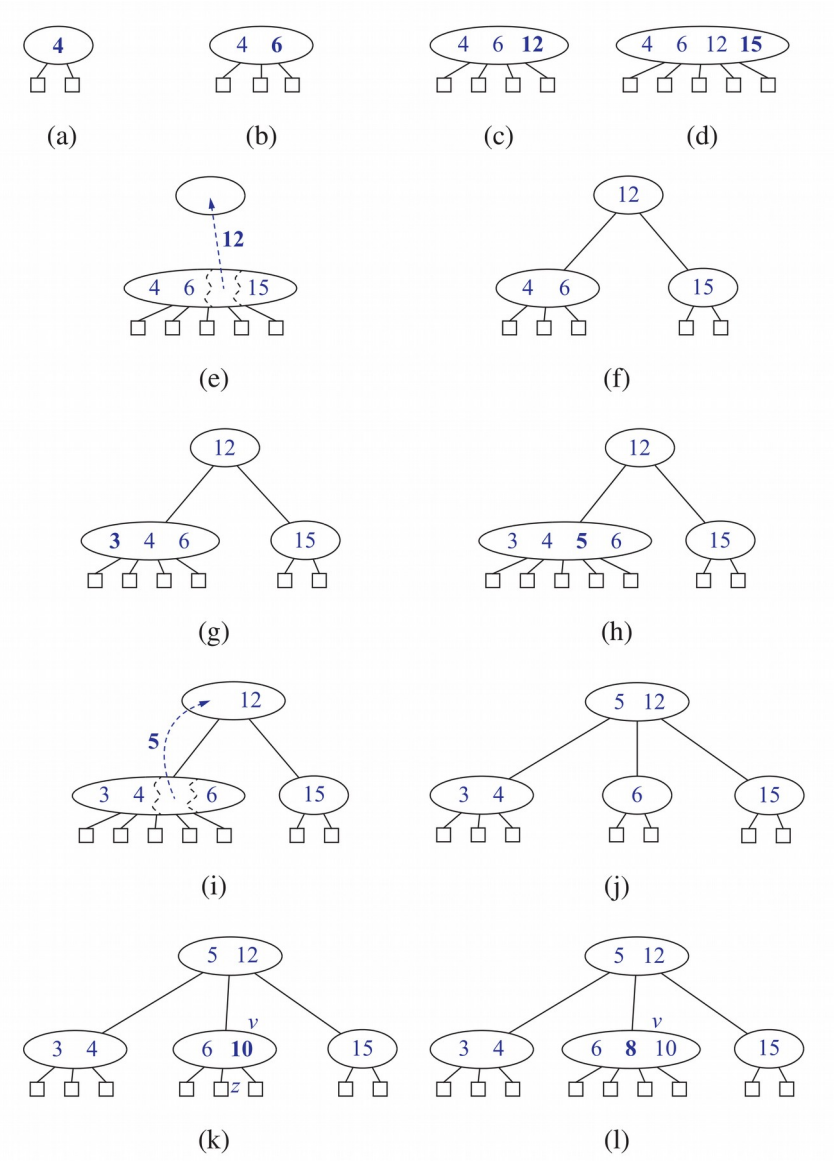
Insert(15);

Insert(3);

Insert(5);

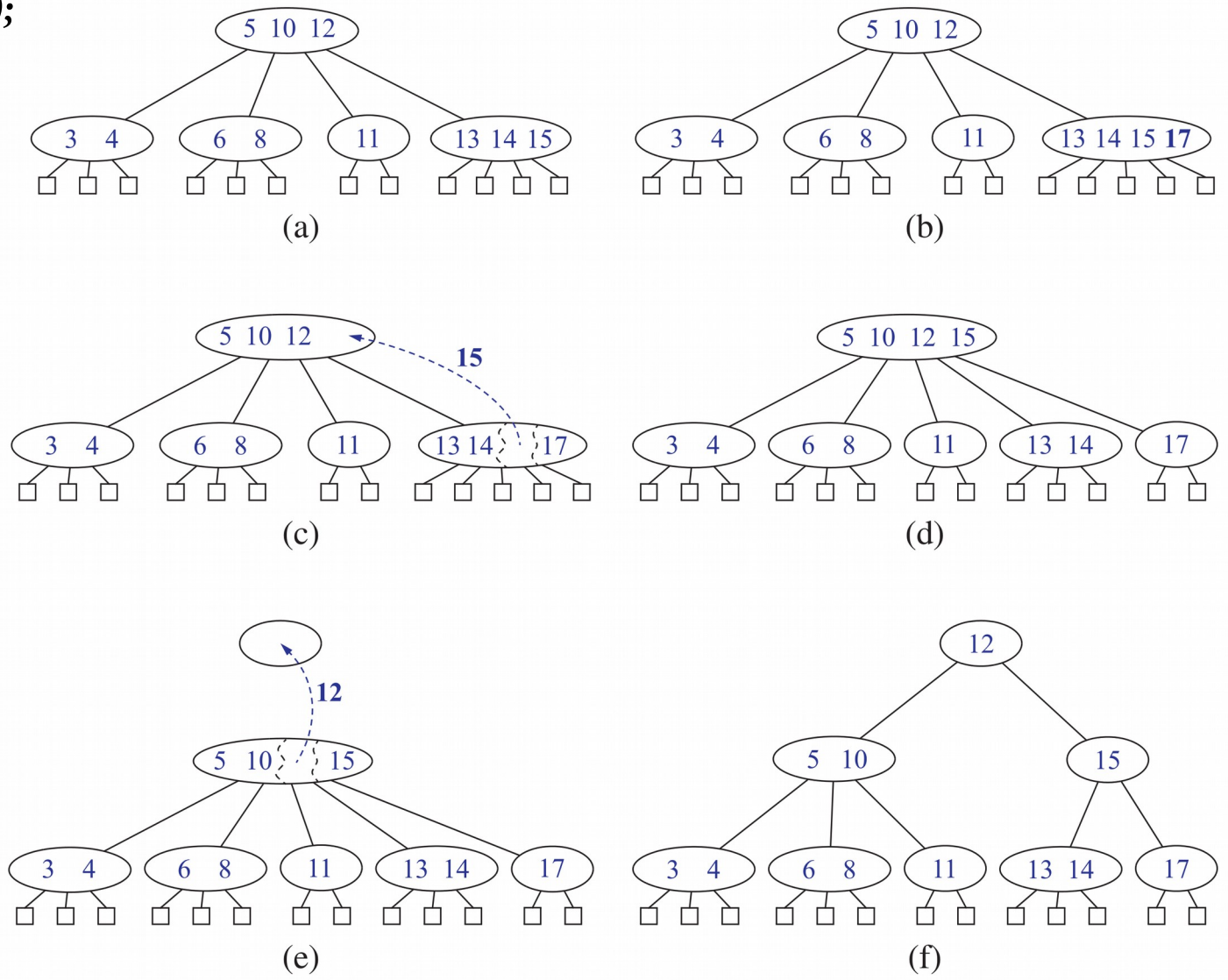
Insert(10);

Insert(8);



Exemplo: Inserção em árvore 2-4

Insert(17);



Árvore 2-4

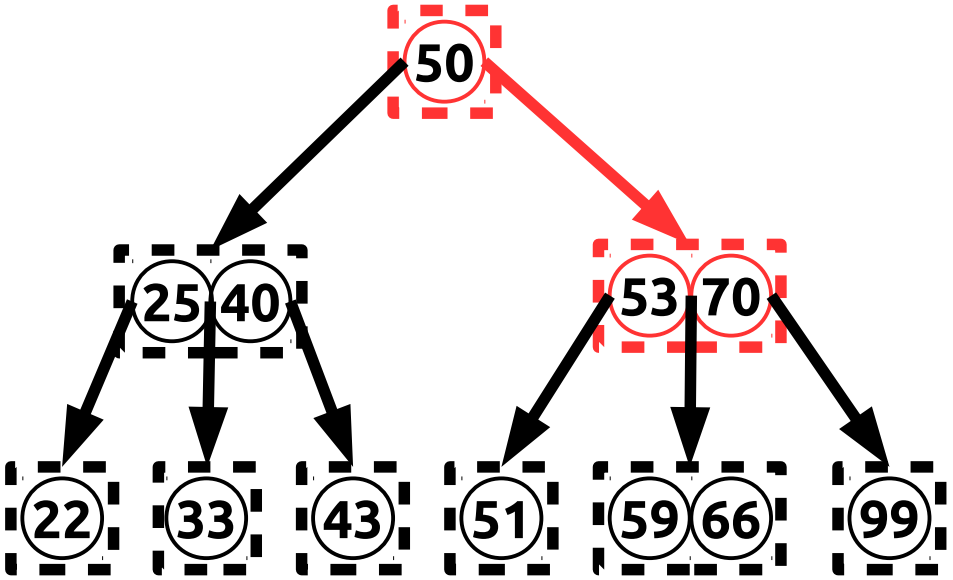
- Altura para N elementos:
  - Melhor caso: $\log_4 N$
    - Todos os nós são *4-nodes*
  - Pior caso: $\log_2 N$
    - Todos os nós são *2-nodes*
- Entre 10 e 20 para 1 milhão de elementos
- Entre 15 e 30 para 1 bilhão de elementos
- Performance logarítmica garantida!!!

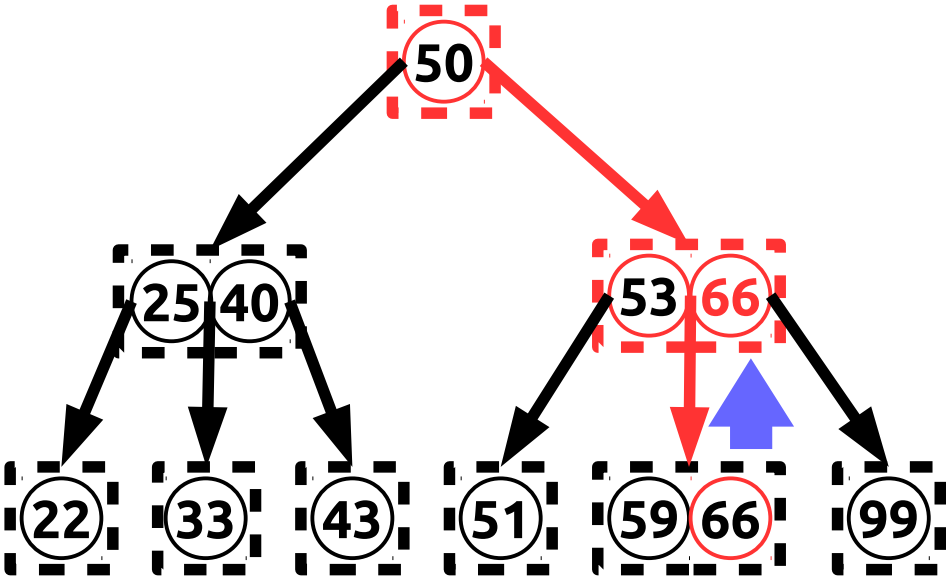
Árvore 2-4 - Remoção

- Buscamos pelo elemento a ser removido: (como na remoção vista anteriormente)
  - Se ele não estiver em uma folha, substituímos por uma das duas opções:
    - maior elemento da subárvore da esquerda; ou
    - menor elemento da subárvore da direita;
  - Continuamos a remoção pelo elemento que o substituiu

Árvore 2-4 - Remoção

- Exemplo:
  - remover(70)

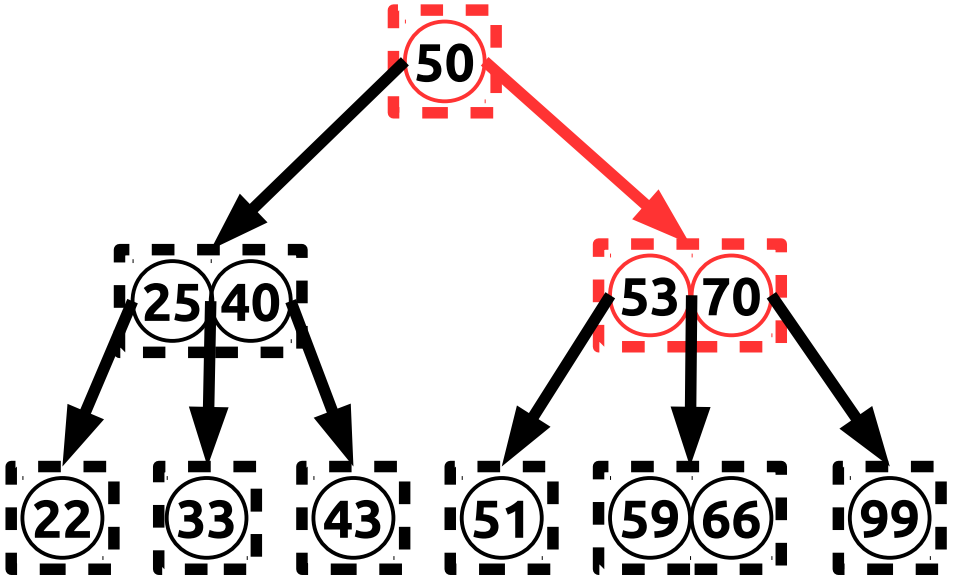


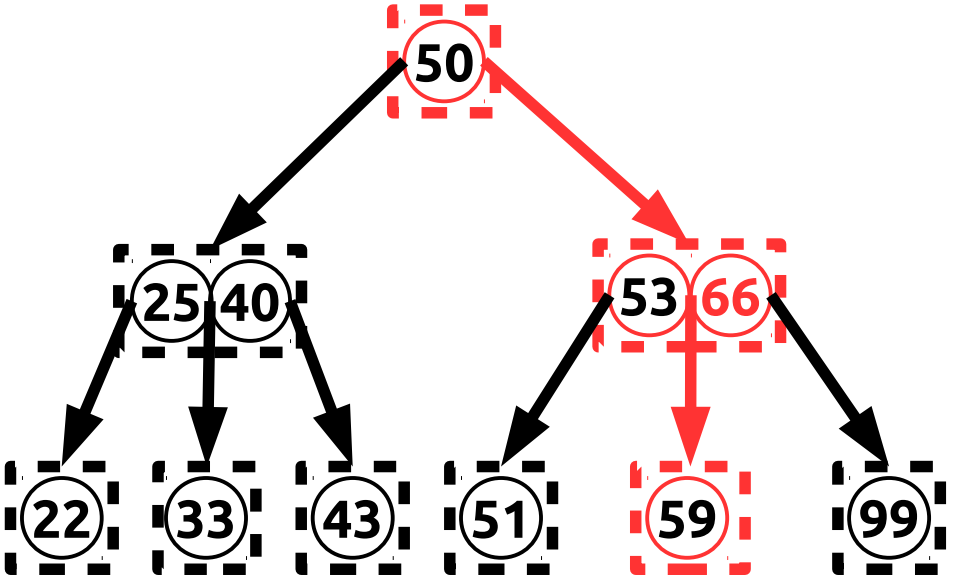


Árvore 2-4 - Remoção

- Para remover um elemento em uma folha:
  - Caso 1: o nó folha é um *4-node* ou *3-node*
    - Remoção direta! simples!

Exemplo:
- remover(70)

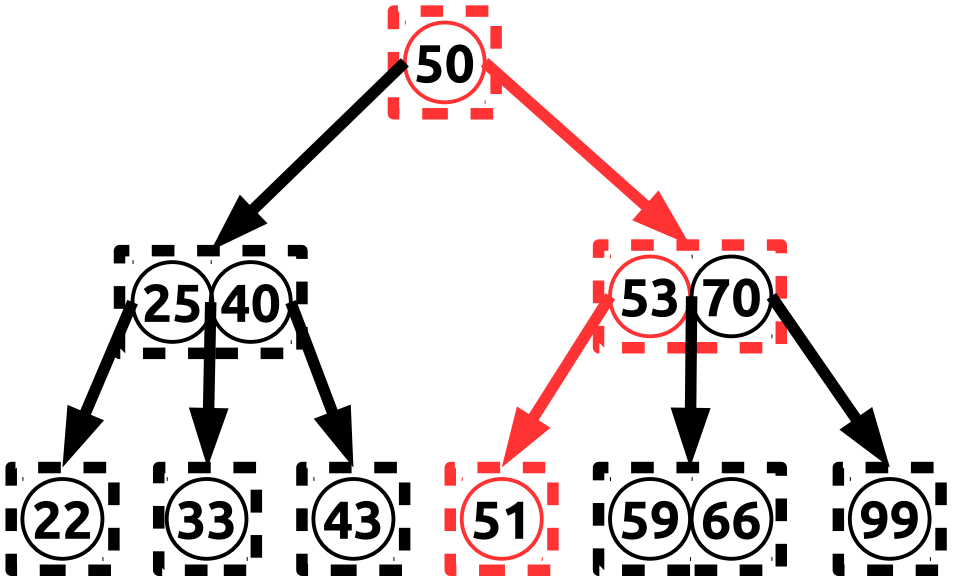




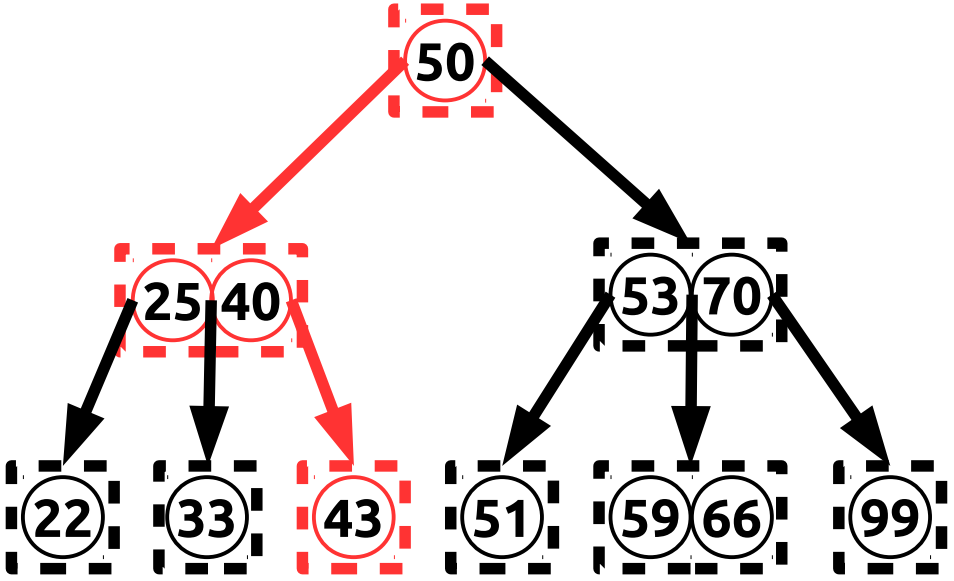
Árvore 2-4 - Remoção

- Para remover um elemento em uma folha:
  - Caso 2: o nó folha é um *2-node*
    - Não podemos simplesmente remover o nó!!!

Exemplo:
- remover(51)



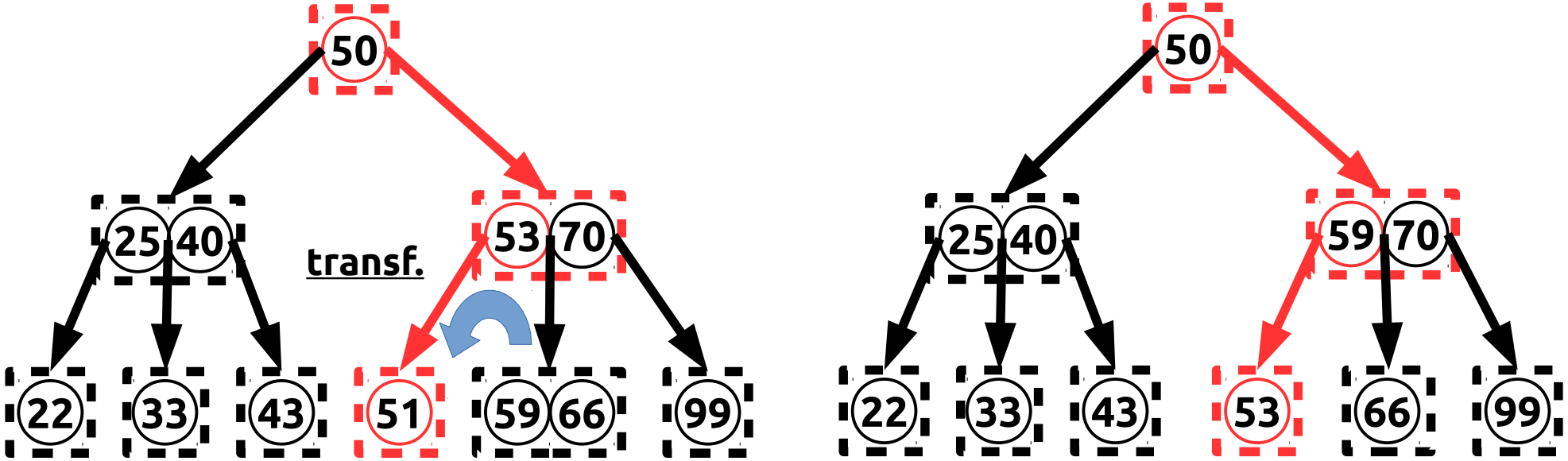
Exemplo:
- remover(43)



Árvore 2-4 - Remoção

- Caso 2.1:
  - Quando podemos transferir um elemento de um nó "irmão"

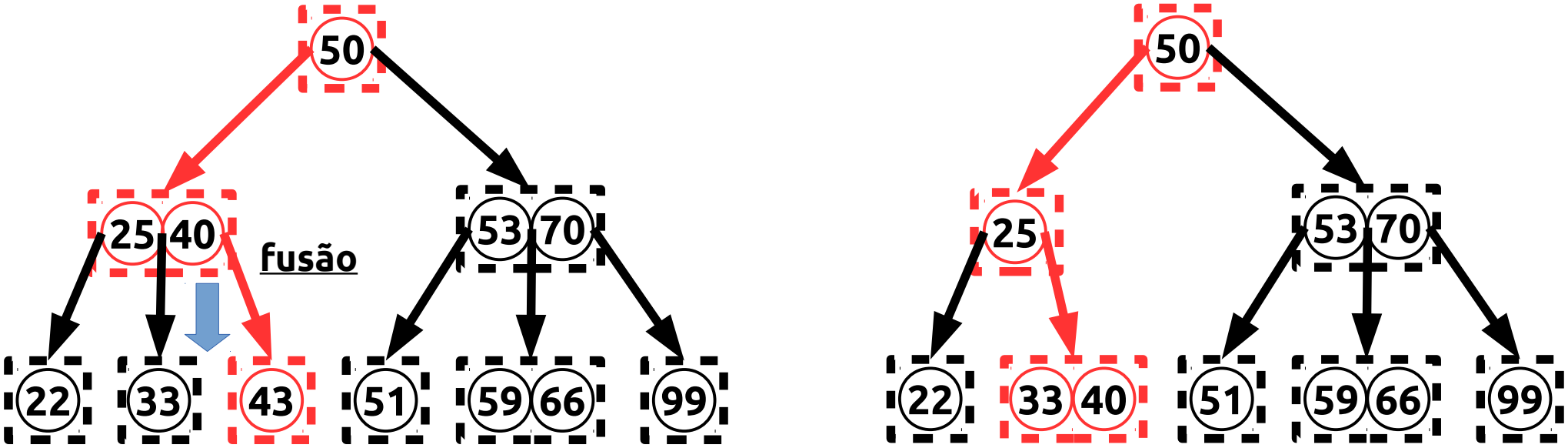
Exemplo:
- remover(51)



Árvore 2-4 - Remoção

- Caso 2.2:
  - Quando não podemos transferir um elemento de um nó "irmão", fazemos a fusão com um nó irmão

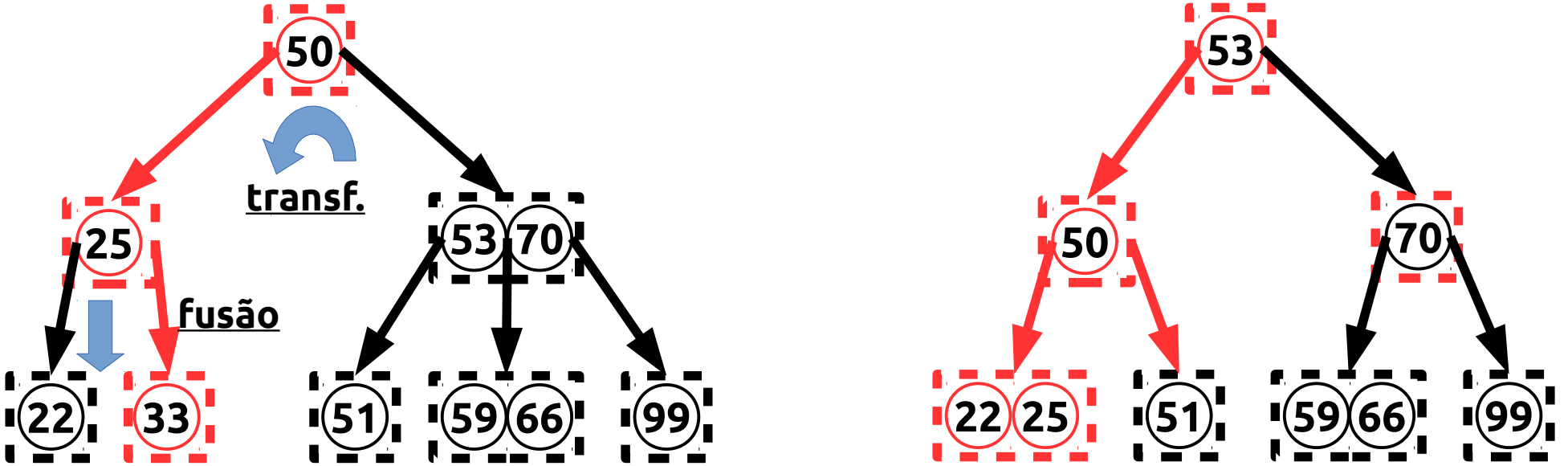
Exemplo:
- remover(43)



Árvore 2-4 - Remoção

- Quando o pai também é um *2-node*:
  - O processo se repete:
    - se tiver um irmão *3-node* ou *4-node*: transfere!
    - se o irmão for *2-node*: fusão!

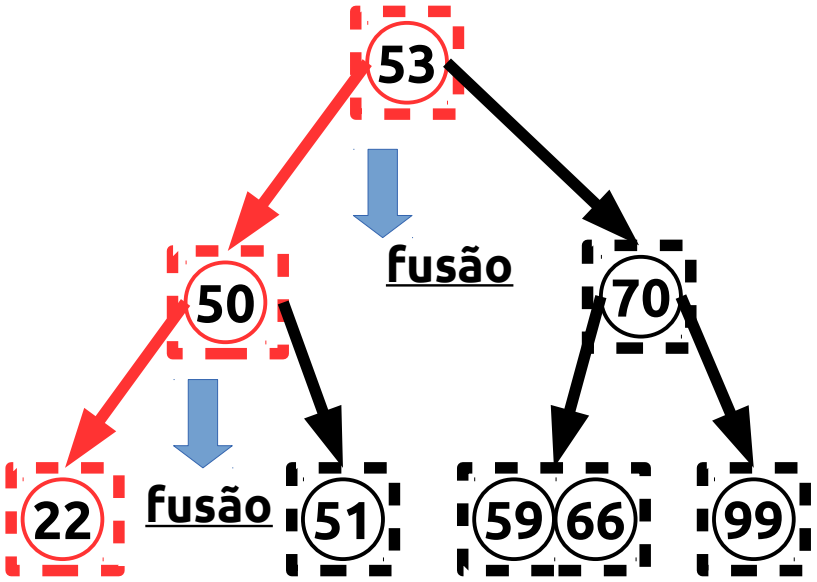
Exemplo:
- remover(33)



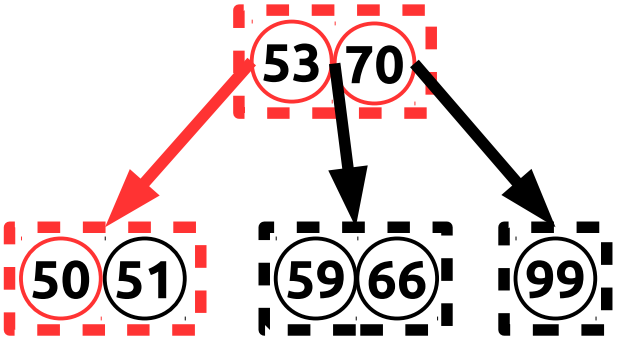
Árvore 2-4 - Remoção

- Quando o pai também é um *2-node*:
  - O processo se repete:
    - se tiver um irmão *3-node* ou *4-node*: transfere!
    - se o irmão for *2-node*: fusão!

Exemplo:
- remover(22)

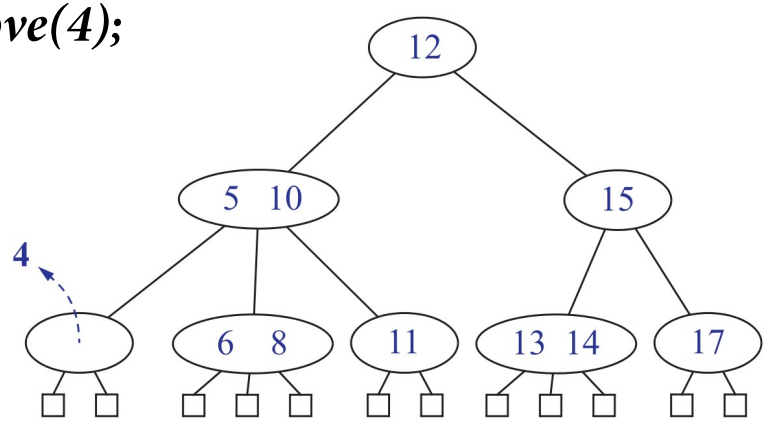


- Quando fazemos a fusão em toda volta até a raiz: a árvore decresce!

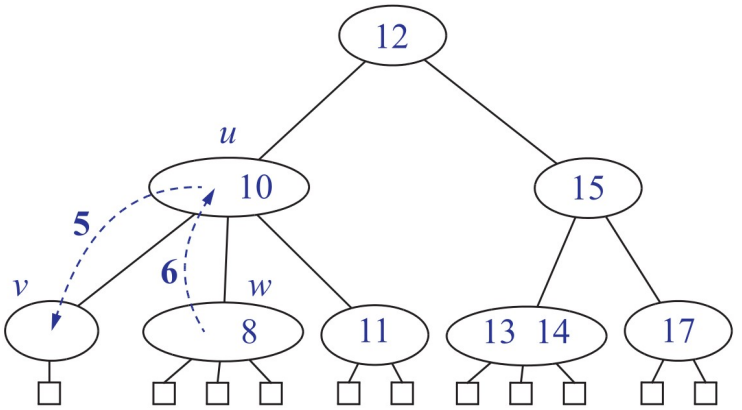


Exemplo: Remoção em árvore 2-4

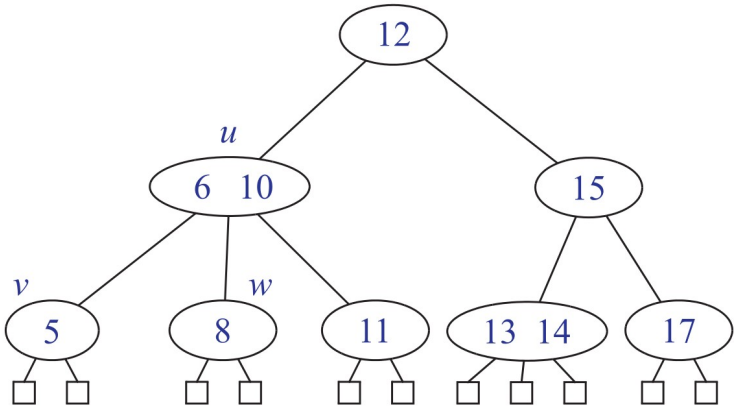
Remove(4);



(a)



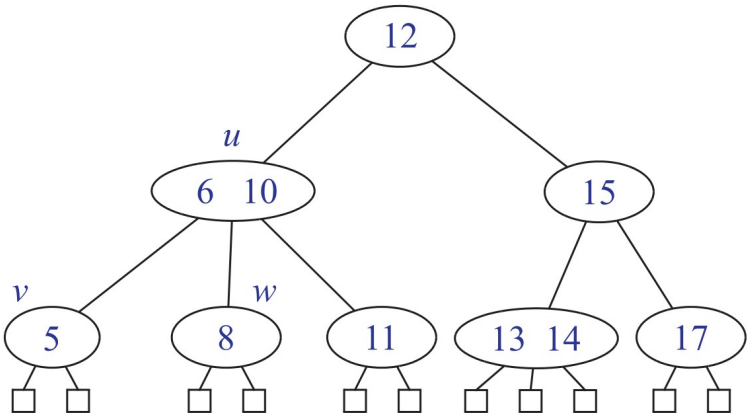
(b)



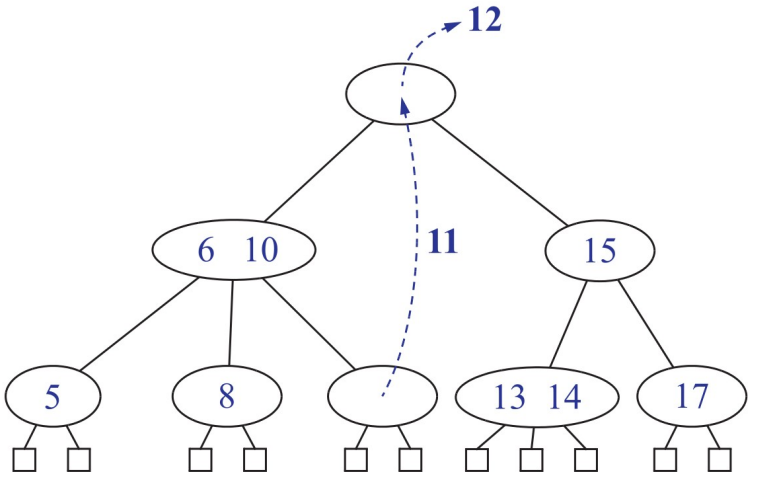
(c)

Exemplo: Remoção em árvore 2-4

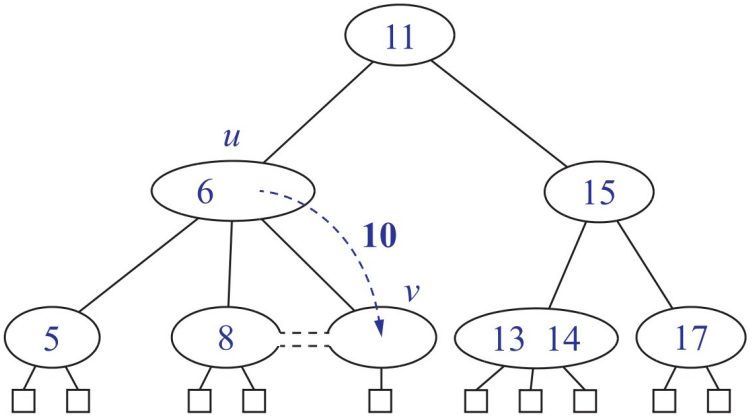
Remove(12);



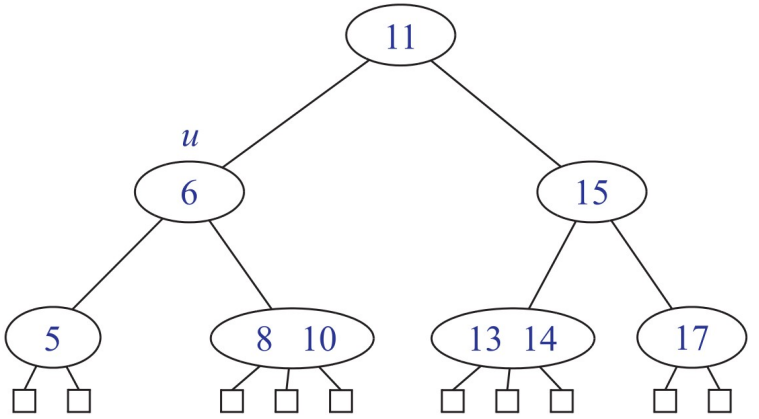
(c)



(d)



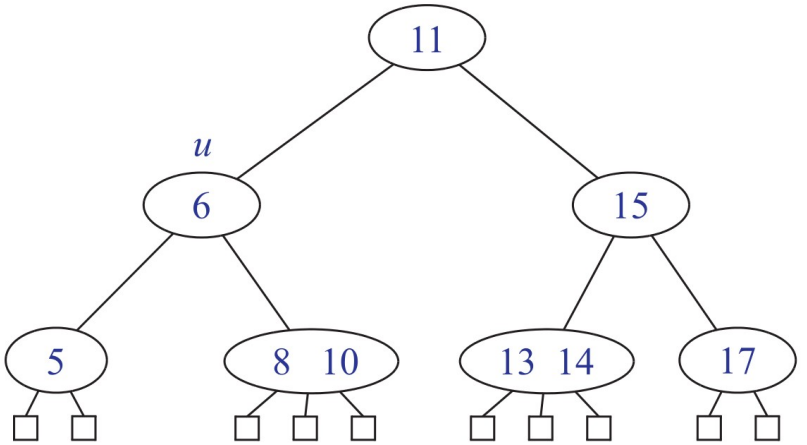
(e)



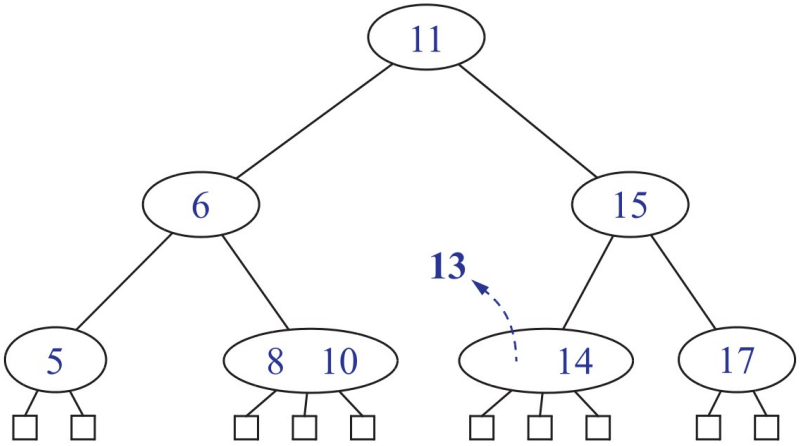
(f)

Exemplo: Remoção em árvore 2-4

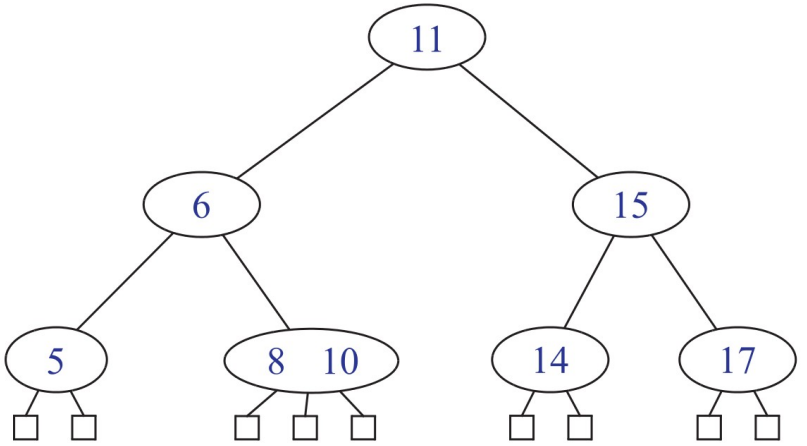
Remove(13);



(f)



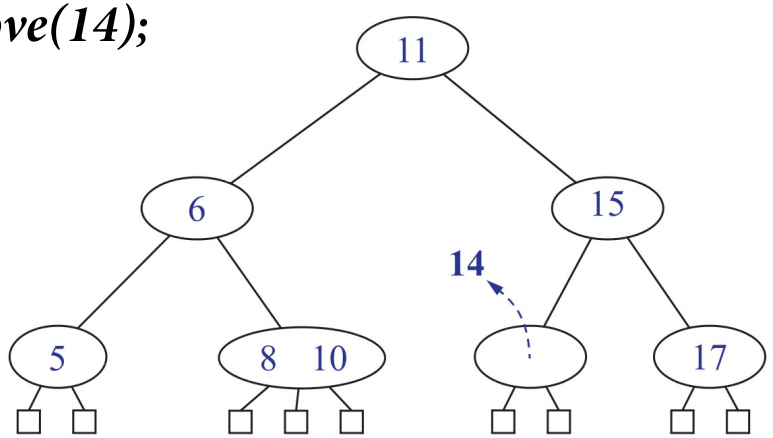
(g)



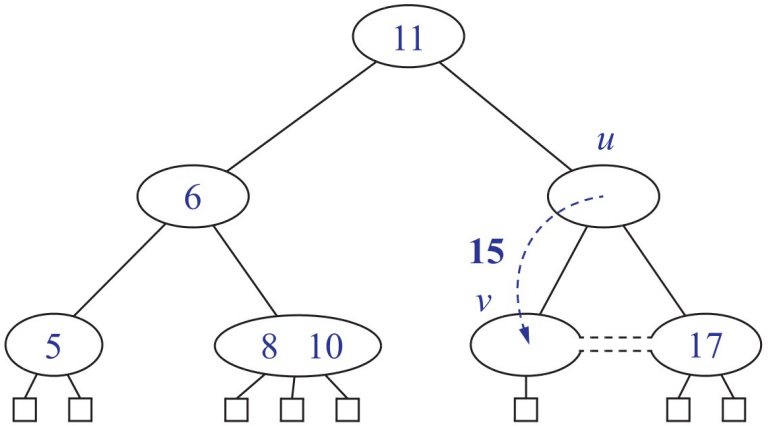
(h)

Exemplo: Remoção em árvore 2-4

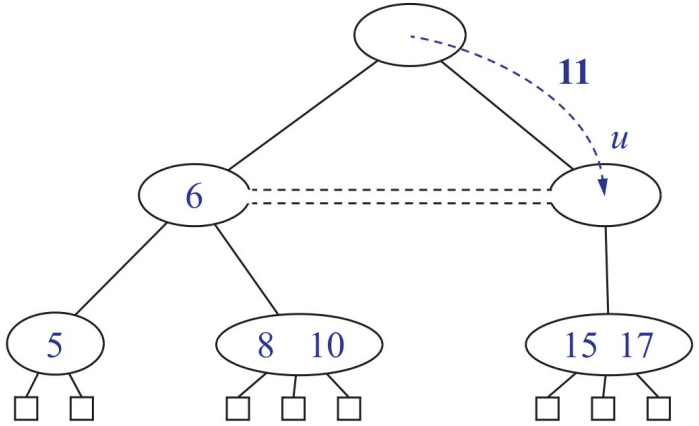
Remove(14);



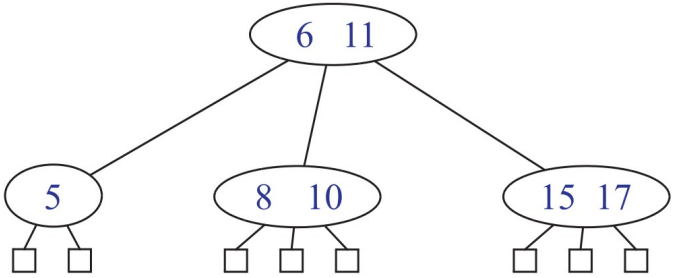
(a)



(b)



(c)



(d)

Árvore 2-4

- Códigos da árvore 2-4:
  - Não vale a pena implementar árvores 2-4
    - Como representaríamos nós de tamanho variável?
      - *Struct* com tamanho máximo? herança?
  - Implementação muito mais eficiente:
    - Versão binária (com todos os nós iguais)
    - Árvore *Red-Black*

Árvores balanceadas

- Árvore 2-4
  - Árvore *n-ária de busca (multi-way)*
  - Implementação ineficiente (comparada a AVL/RB)
  - Base para árvores RB (Red-Black) e B (para Hds)
  - Permite no máximo 3 elementos por nó (*2-nodes*, *3-nodes* e *4-nodes*)
  - Balanceamento perfeito!
  - Crescimento de baixo para cima
  - Algoritmos: Busca, Inserção e Remoção

Referências:

- Livro do Goodrich
- Livro do Robert Sedgewick
  - https://algs4.cs.princeton.edu