Árvores binárias balanceadas

Referências

- Árvores 2-3 (PF),
- BSTs rubro-negras (PF)
- Balanced Search Trees (S&W),
- slides (S&W)

Vídeo

Balanced Search Trees (S&W)

Árvores 2-3

Como implementar uma tabela de símbolos em uma BST de modo que a árvore permaneça aproximadamente balanceada? Ous seja, desejamos que a BST tenha altura próxima de $\lg n$, sendo n o número de nós, qualquer que seja a sequência de buscas e inserções aplicada à árvore.

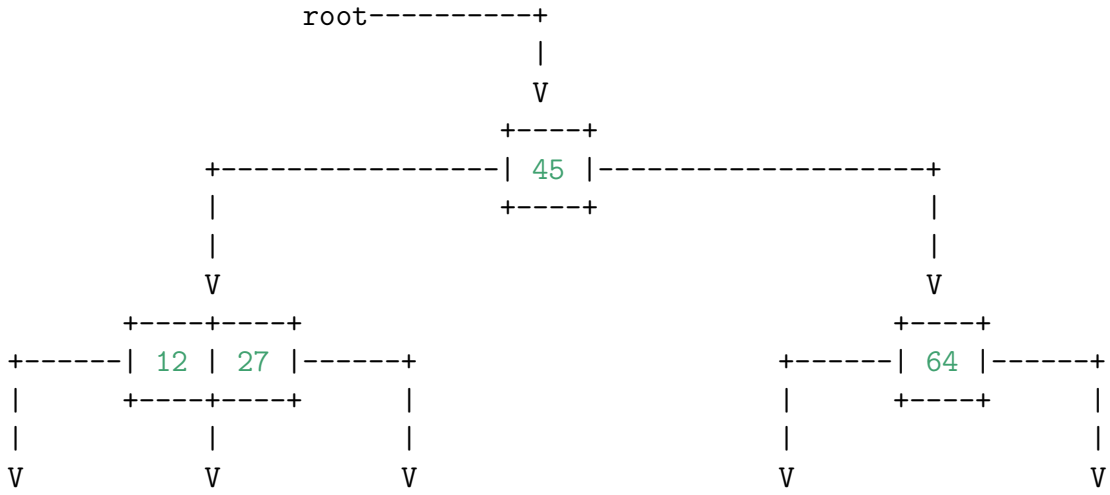
Veremos árvores 2-3 que resolvem o problema em princípio. A implementação da ideia, usando **árvores rubro-negras**, ainda será discutida.

Uma **árvore 2-3** de busca (*2-3 search tree*) é:

- uma árvore vazia;
- ou um nó simples, que contém uma chave e dois links:
  - um link esquerdo para uma árvore 2-3 que tem chaves menores que a chave do nó e
  - um link direito para uma árvore 2-3 que tem chaves maiores;
- ou um nó duplo, que contém duas chave e três links:
  - um link esquerdo para uma árvore 2-3 que tem chaves menores;
  - um link do meio para uma árvore 2-3 que tem chaves entre as duas chaves do nó; e
  - um link direito para uma árvore 2-3 que tem chaves maiores.

Árvores 2-3 têm esse nome porque cada nó tem 2 ou 3 links.

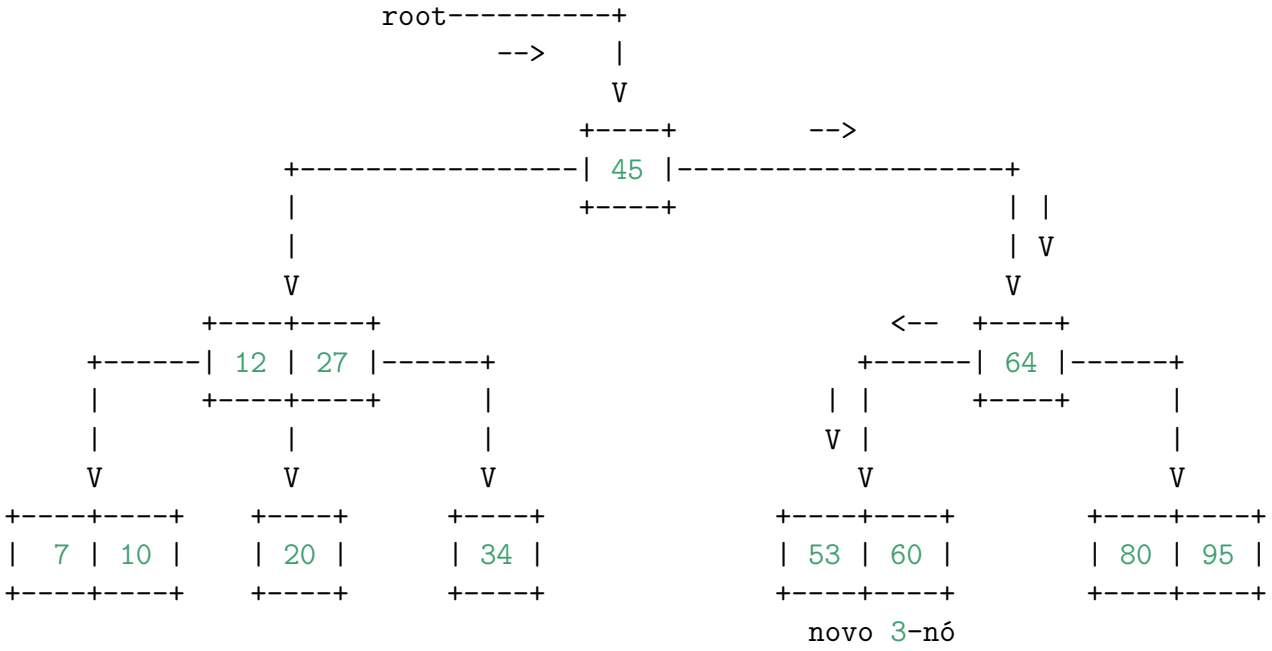
Toda árvore 2-3 é **perfeitamente balanceada**: todos os links `null` estão no mesmo nível.



`put(60)`

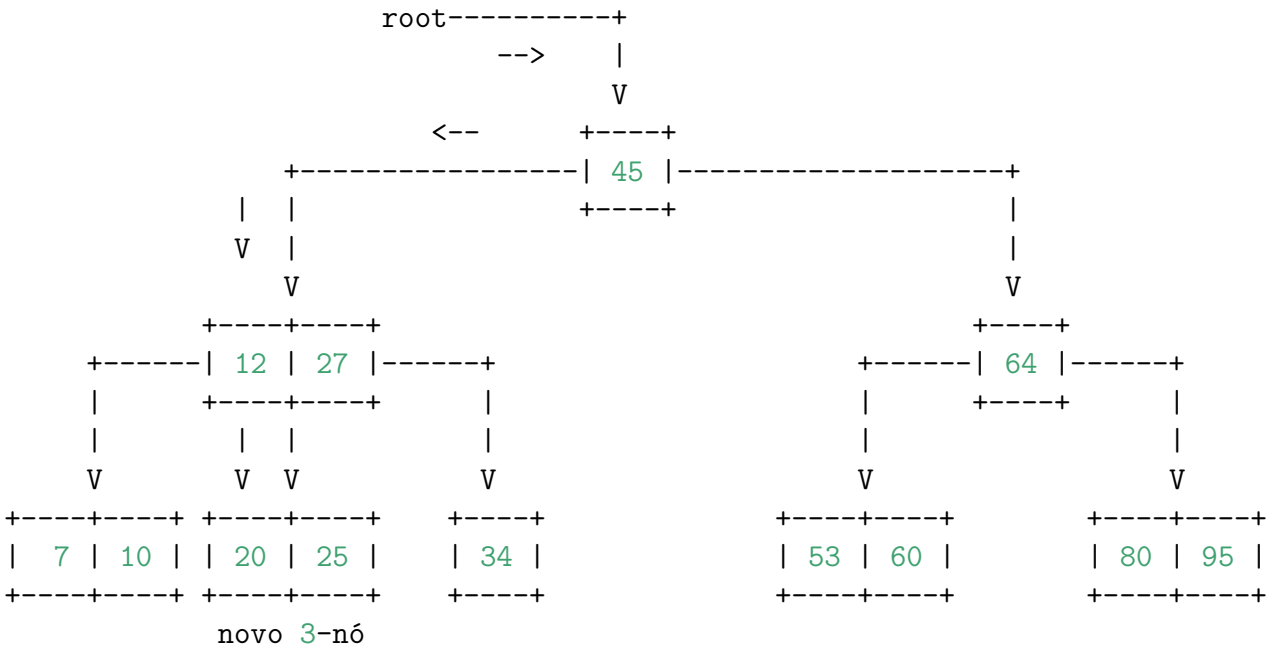
Exemplo: inserção em um nó simples (a operação não estraga o balanceamento):

Procura 60 e transforma um 2-nó em 3-nó.



`put(25)`

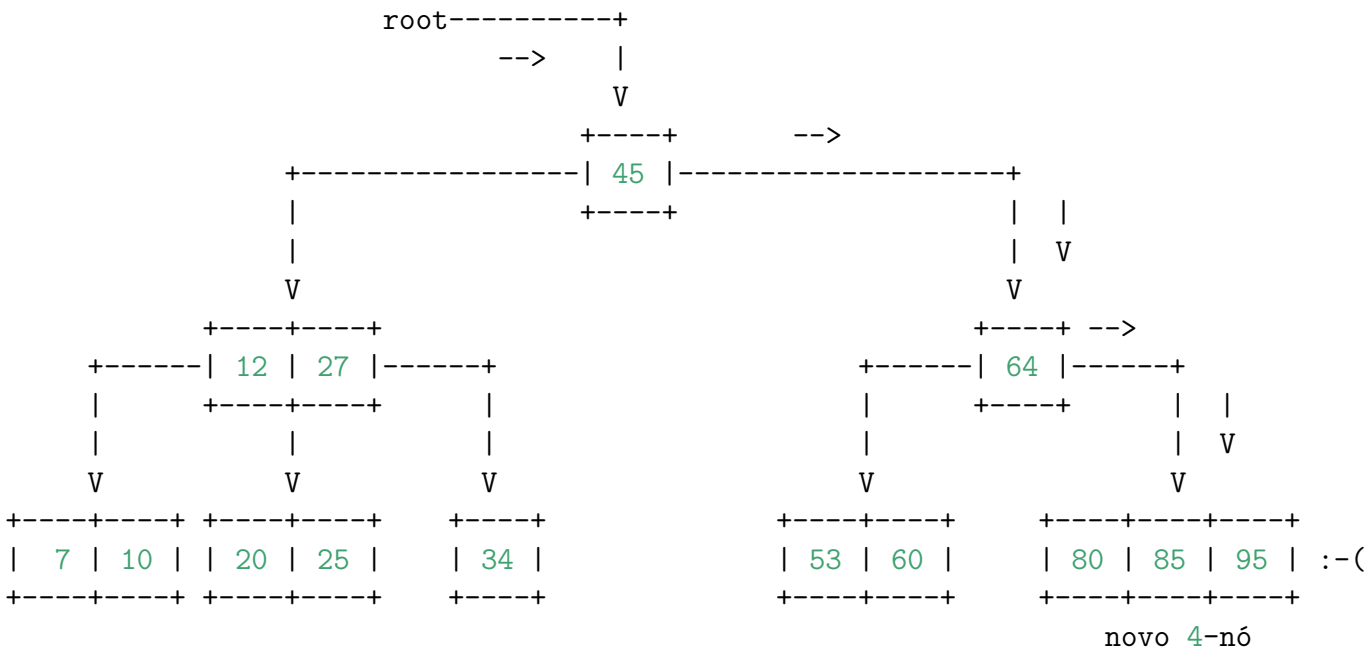
Procura 25 e transforma um 2-nó em 3-nó.



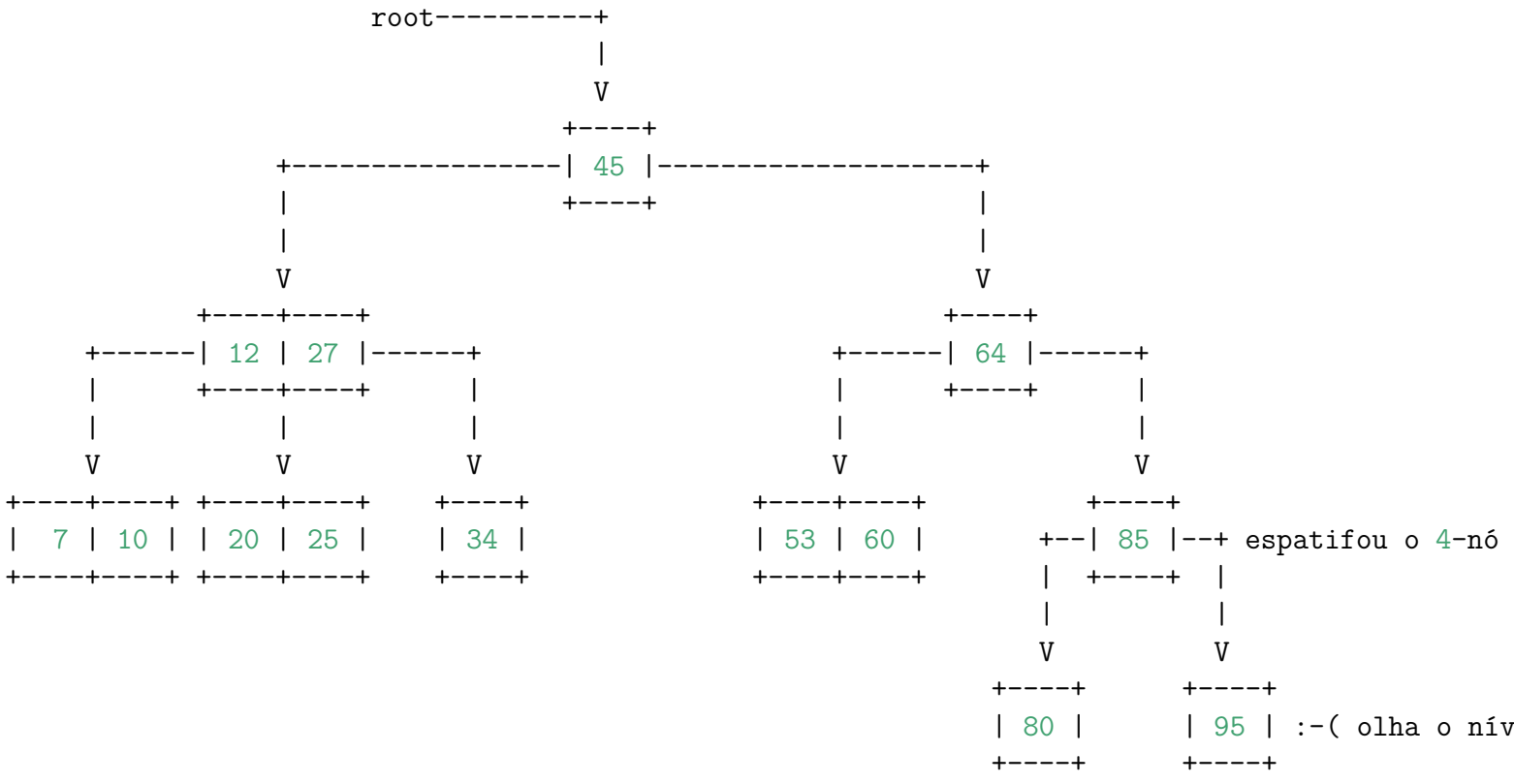
`put(85)`

Exemplo: inserção em um nó duplo cujo pai é um nó simples (a operação não estraga o balanceamento):

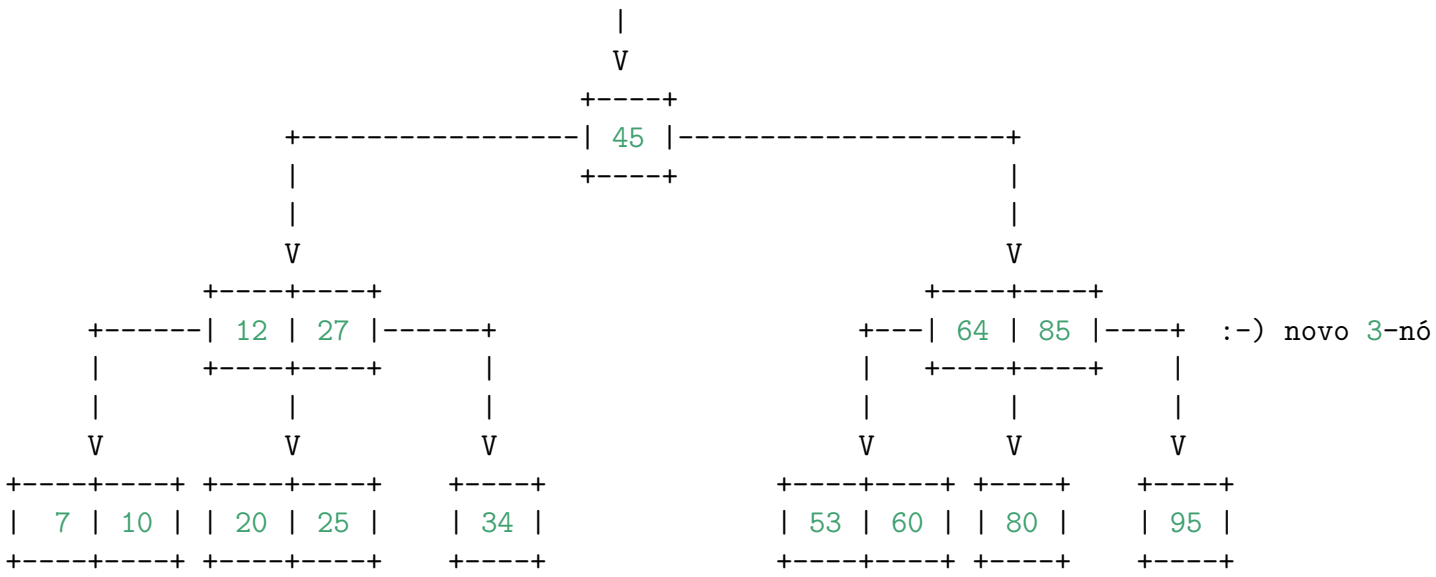
Procura 85 e transforma um 3-nó em 4-nó.



Espatifa o 4-nó.



Transforma o 2-nó pai em 3-nó.

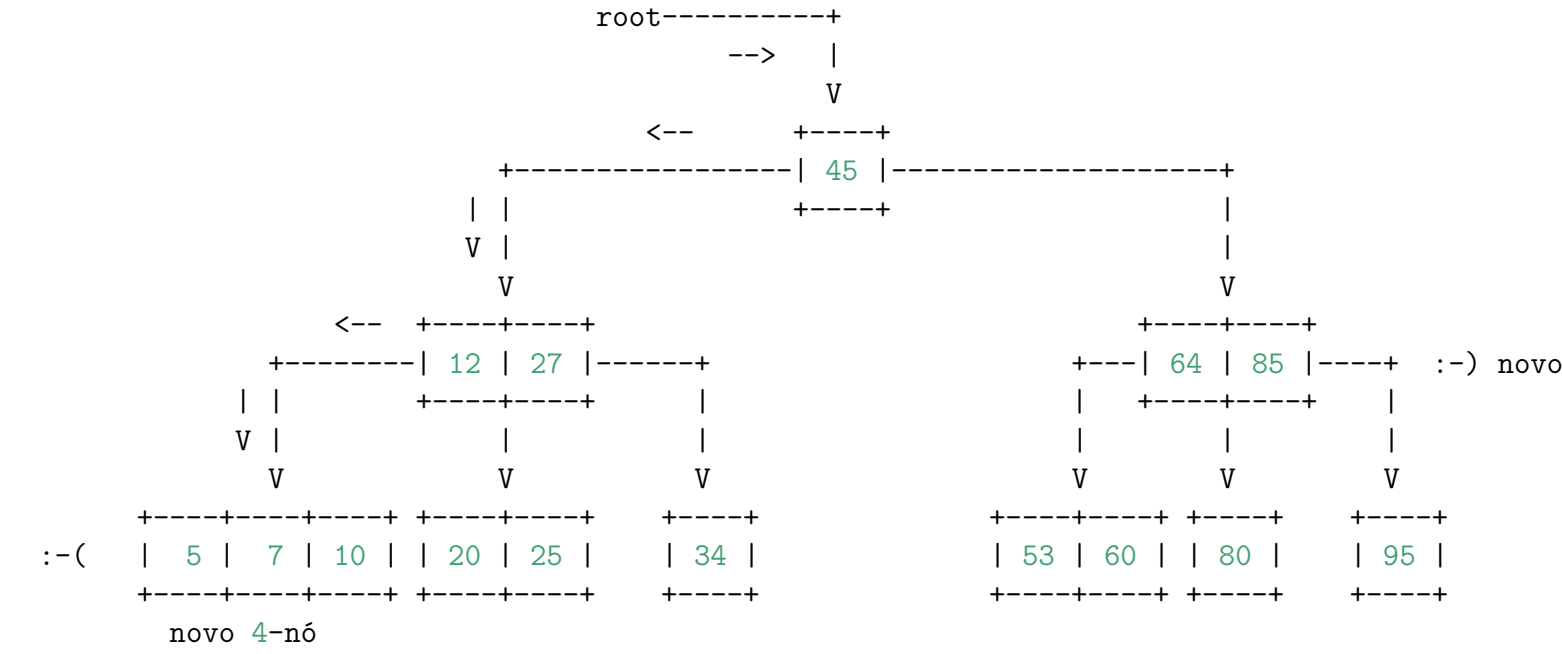


`put(5)`

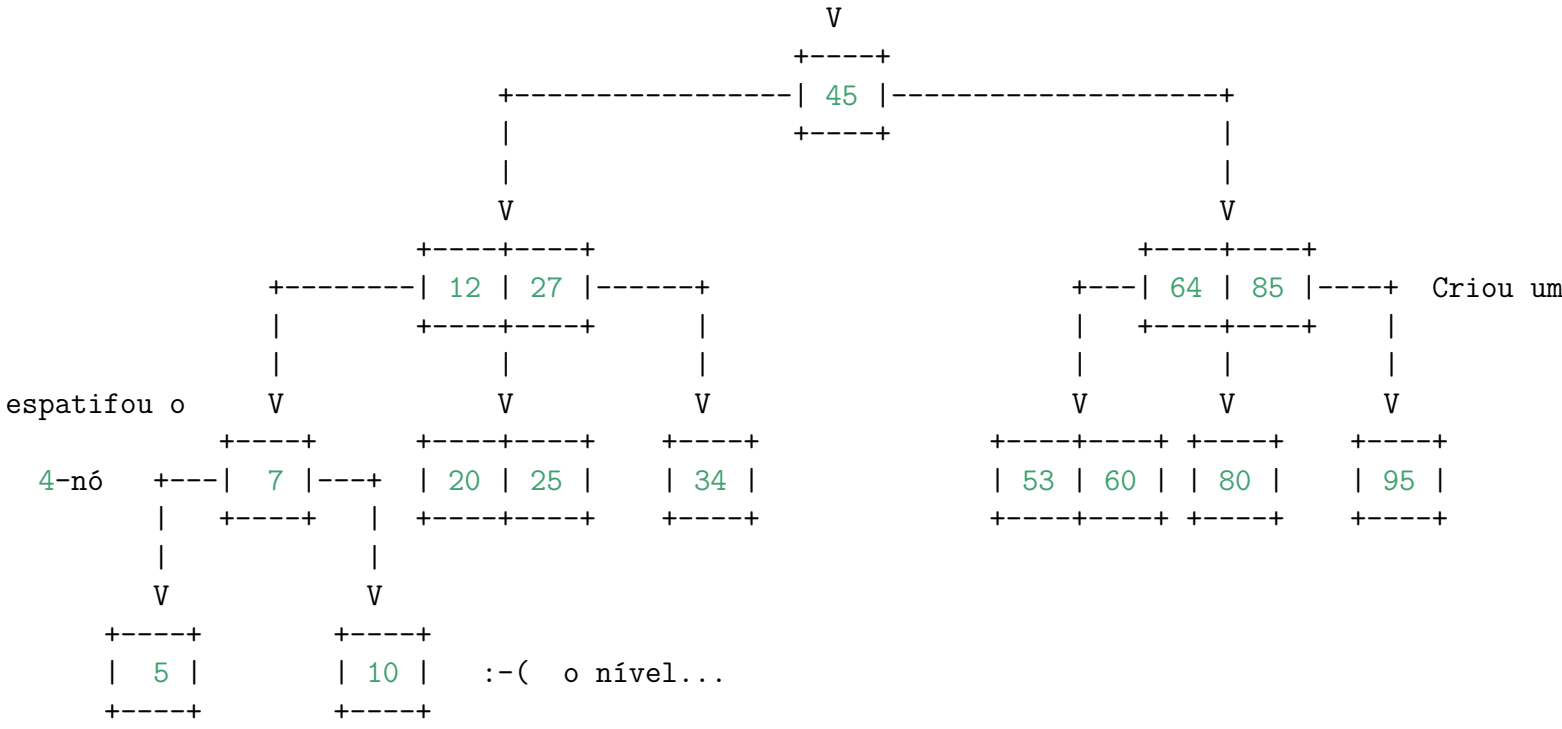
Exemplo: inserção em um nó duplo cujo pai é um nó duplo:

Inserção em um nó duplo cujo pai é um nó duplo: repita a operação subindo em direção à raiz até encontrar um nó simples (nesse caso a altura aumenta). ou até encontrar a raiz (nesse caso a altura aumenta). A operação não estraga o balanceamento.

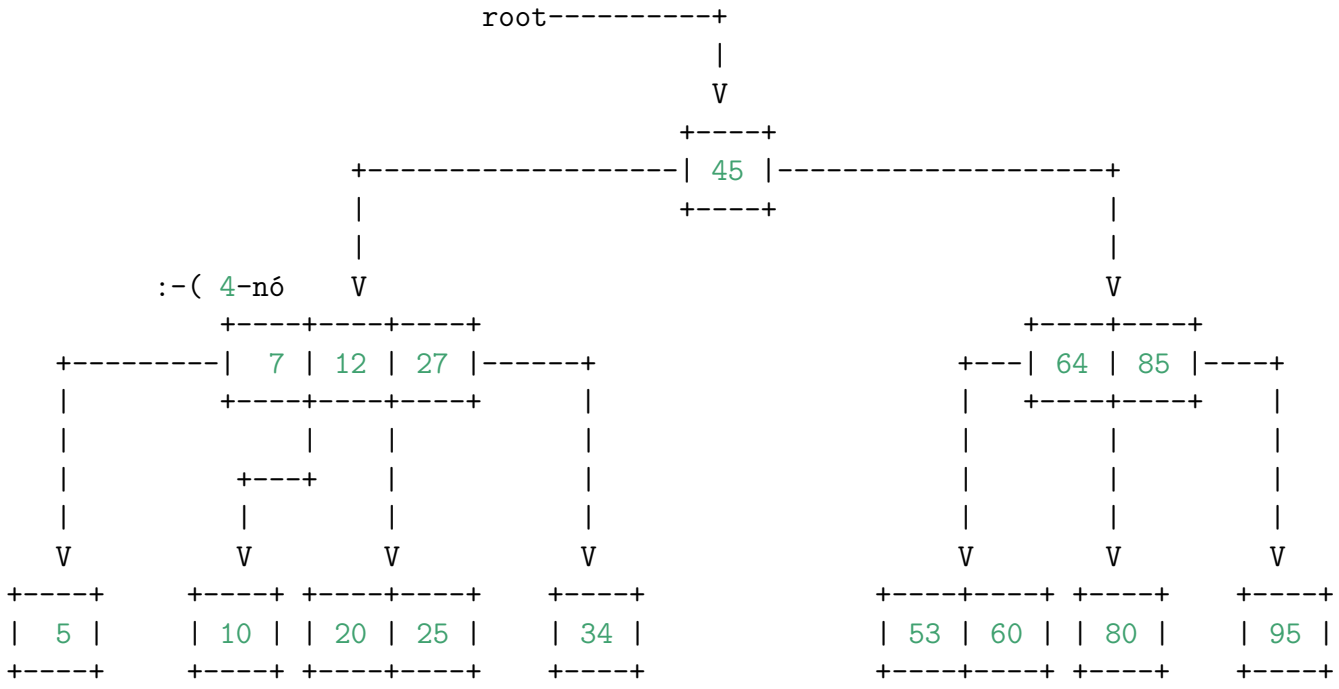
Procura 5 e transforma um 3-nó em 4-nó.



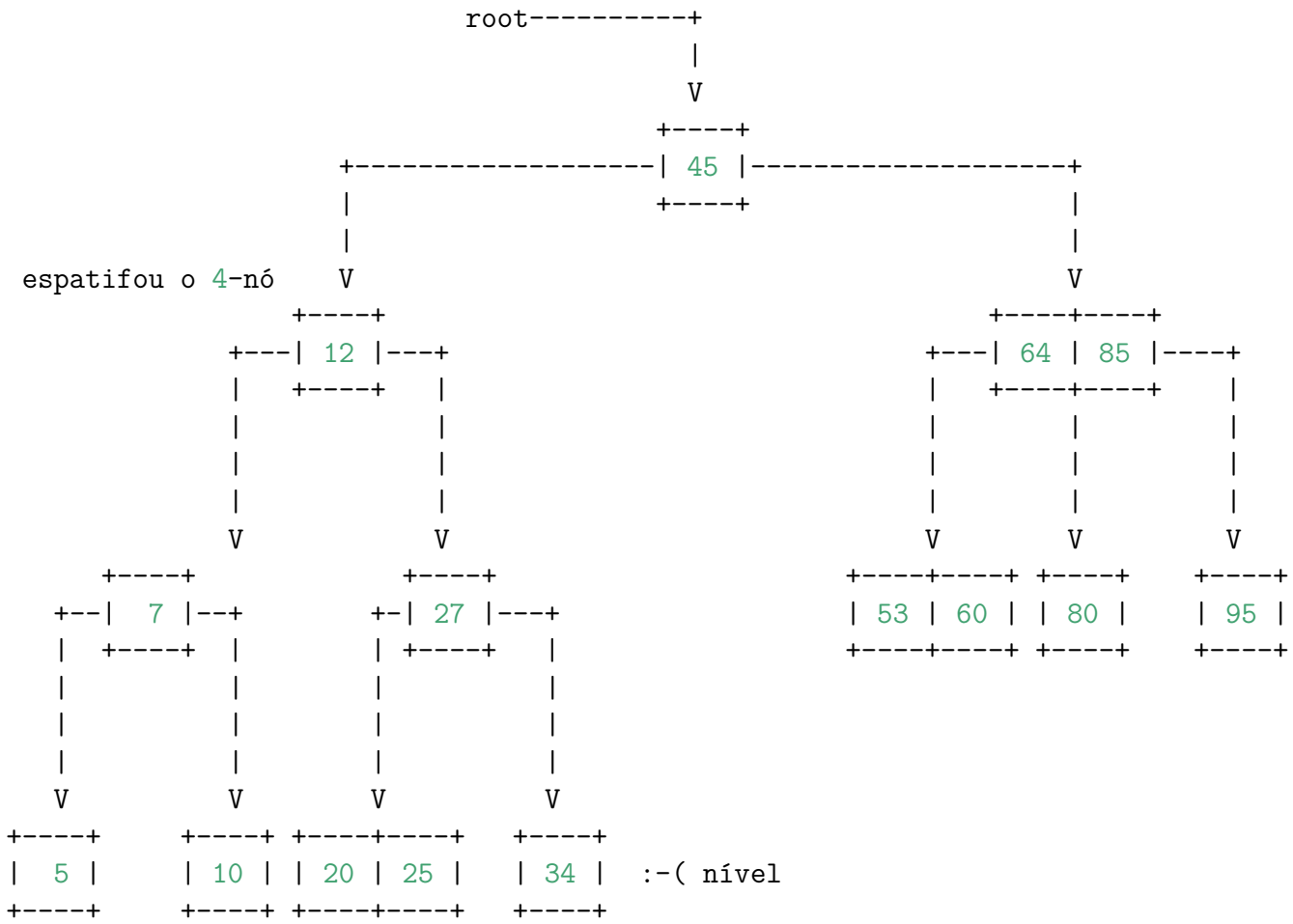
Espatifa o 4-nó.


45
12
27
64
85
Criou um
espatifou o
4-nó
7
20
25
34
53
60
80
95
5
10
:-(o nível...

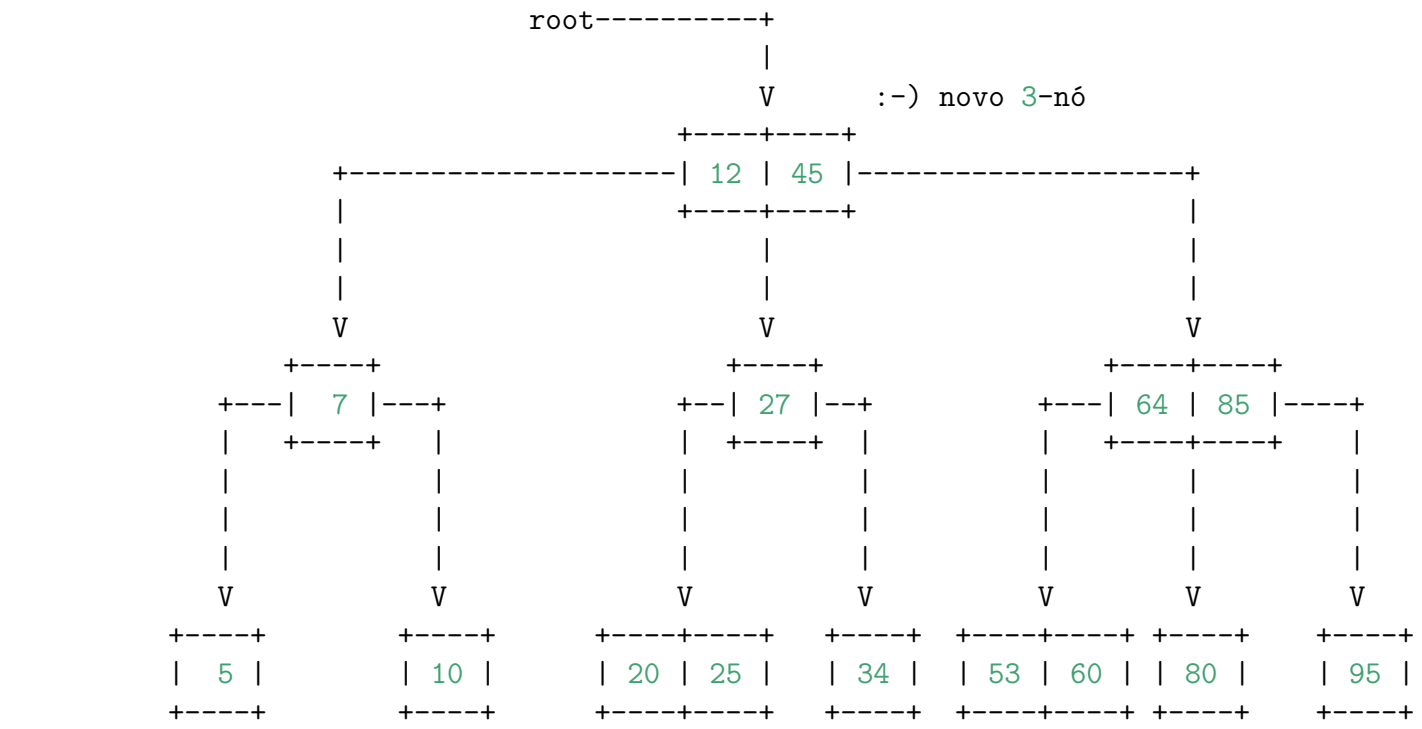
Transforma o 3-nó pai em 4-nó.



Espatifa

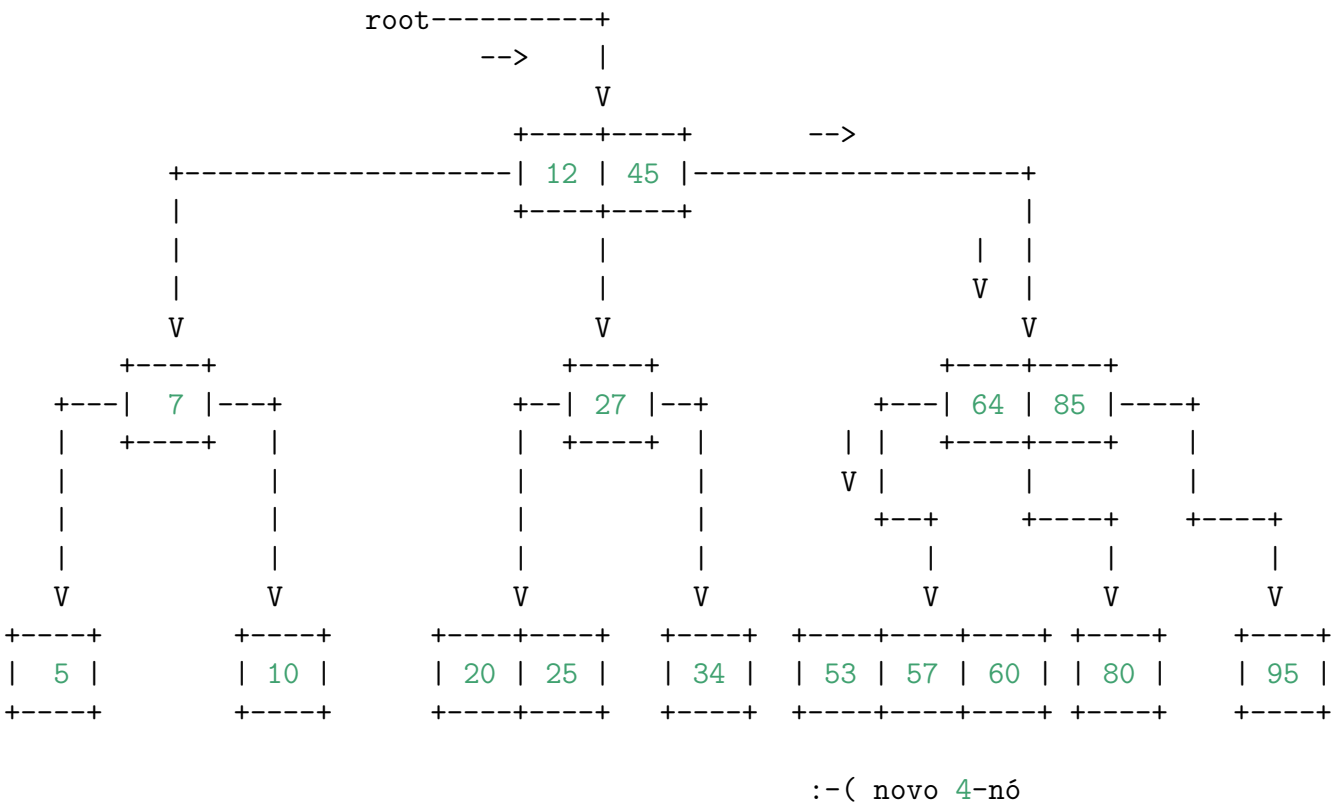


Transforma o 2-nó pai em 3-nó.

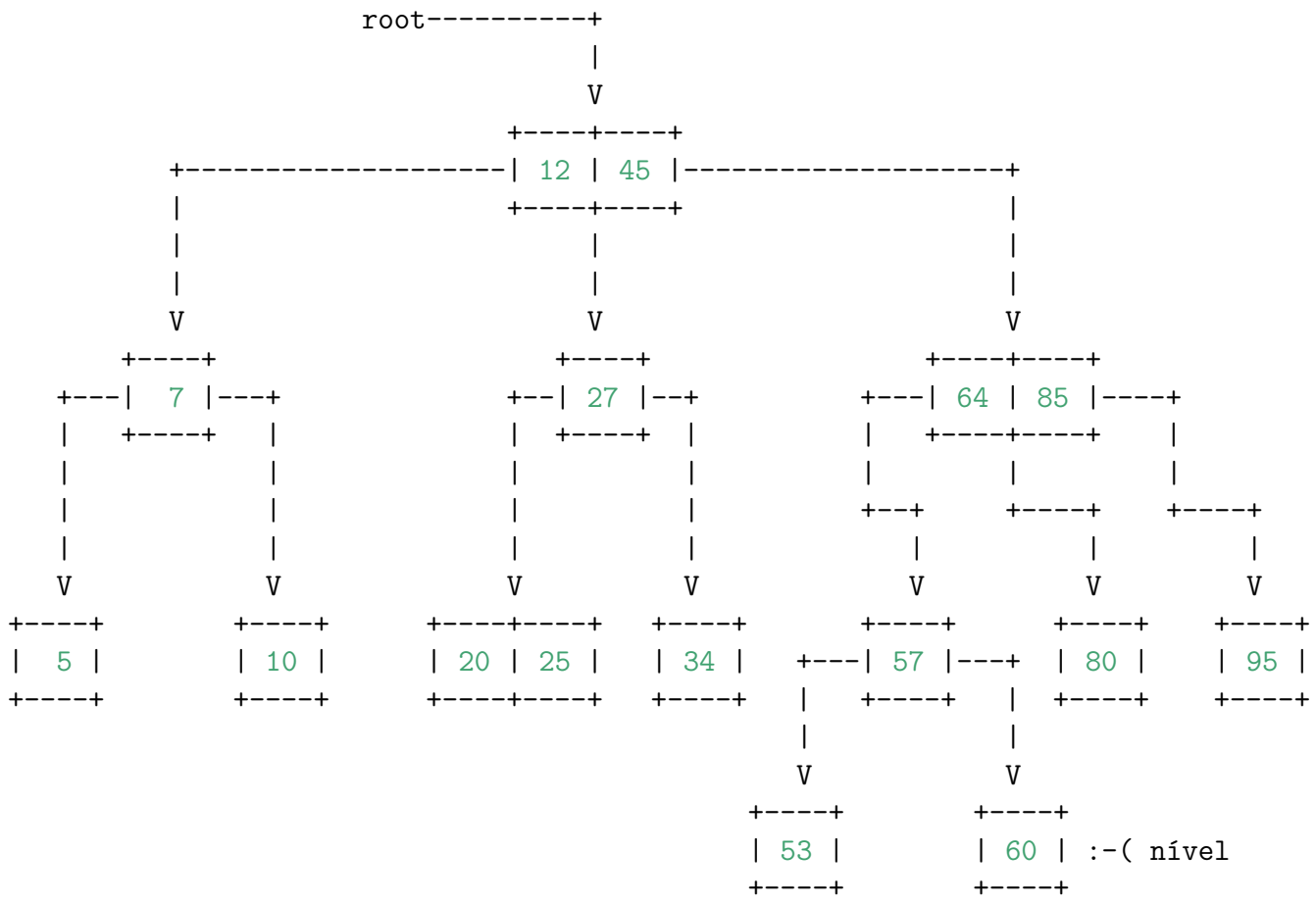


put(57)

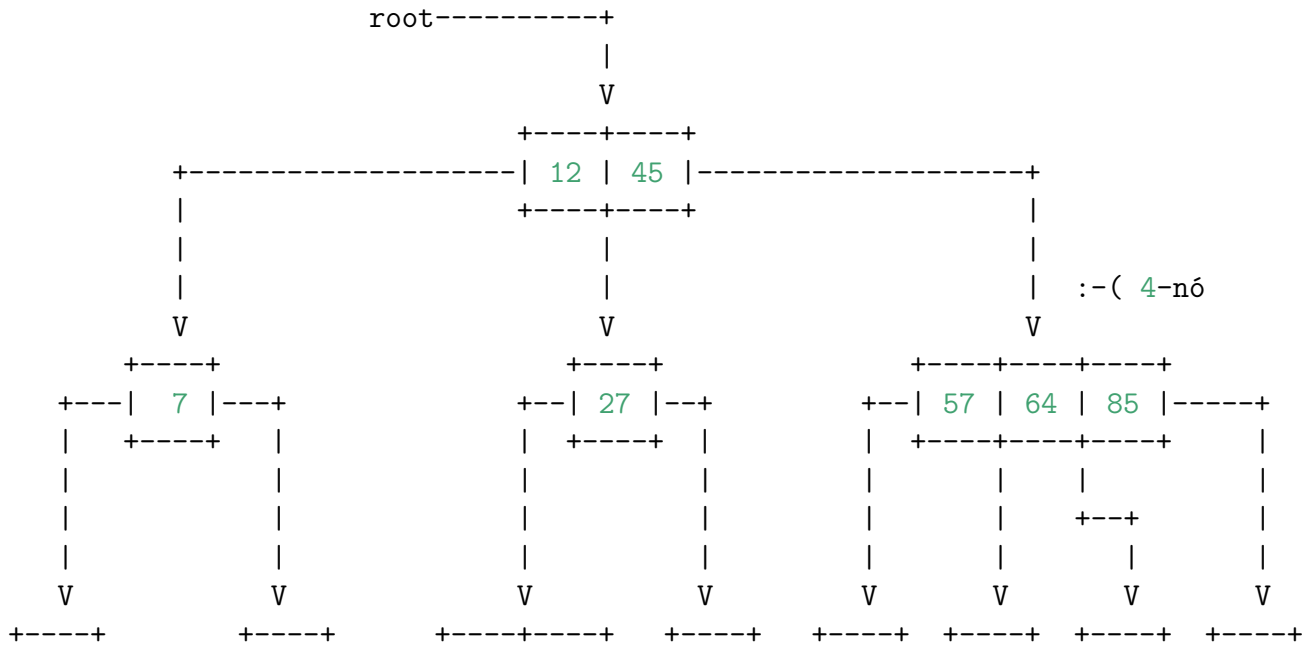
Procura 57 e transforma o 3-nó em 4-nó.

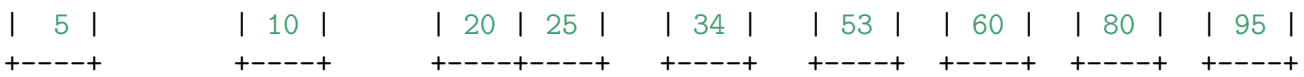


Espatifa o 4-nó.

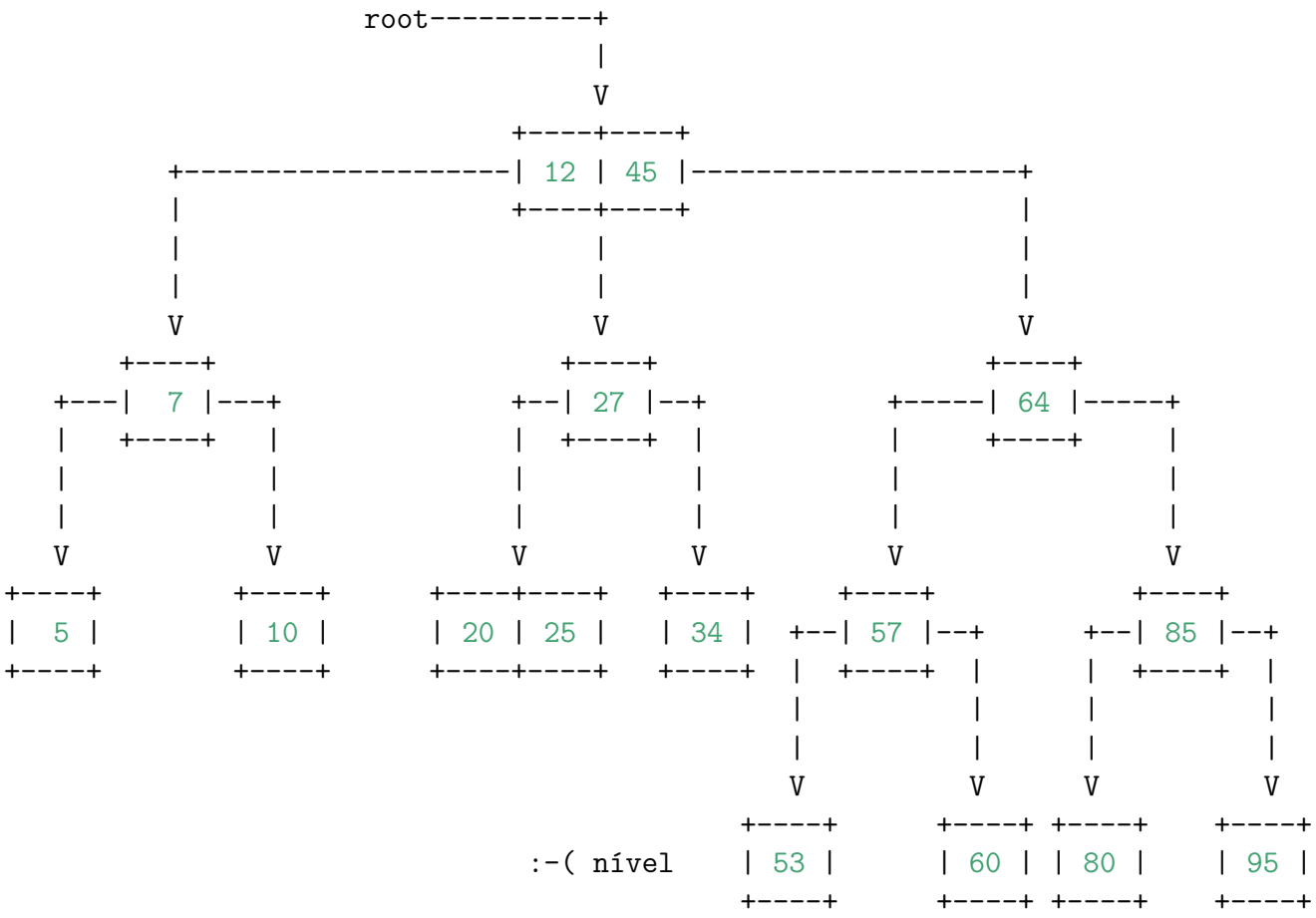


Transforma o 3-nó pai em 4-nó.

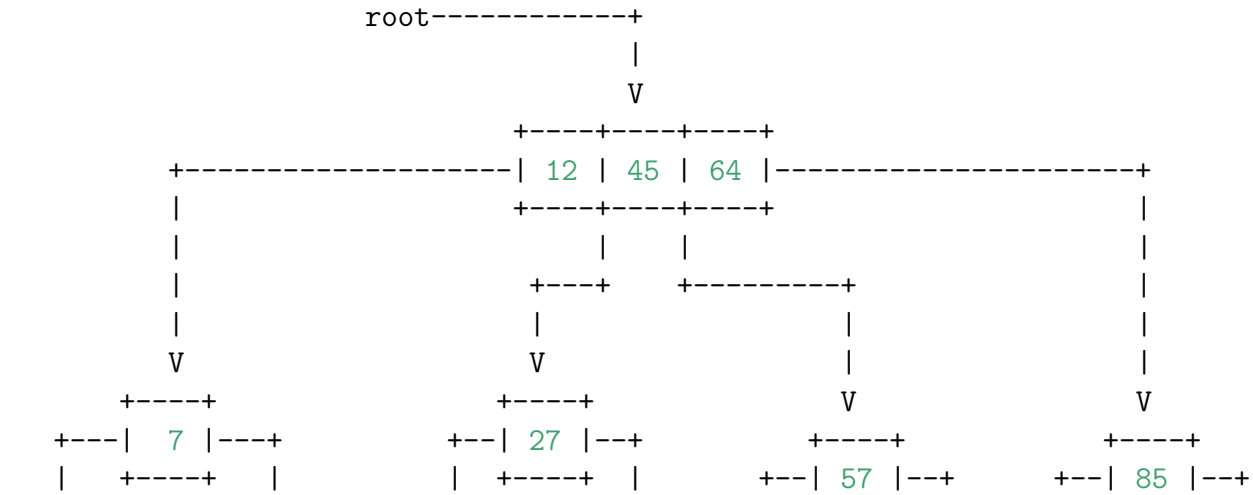


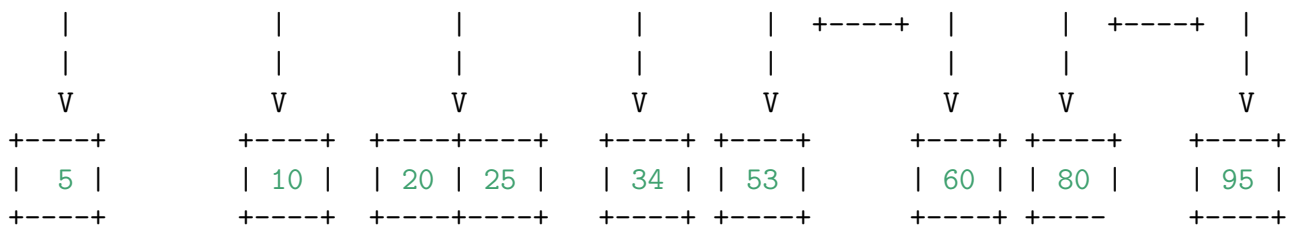


Espatifa o 4-nó.



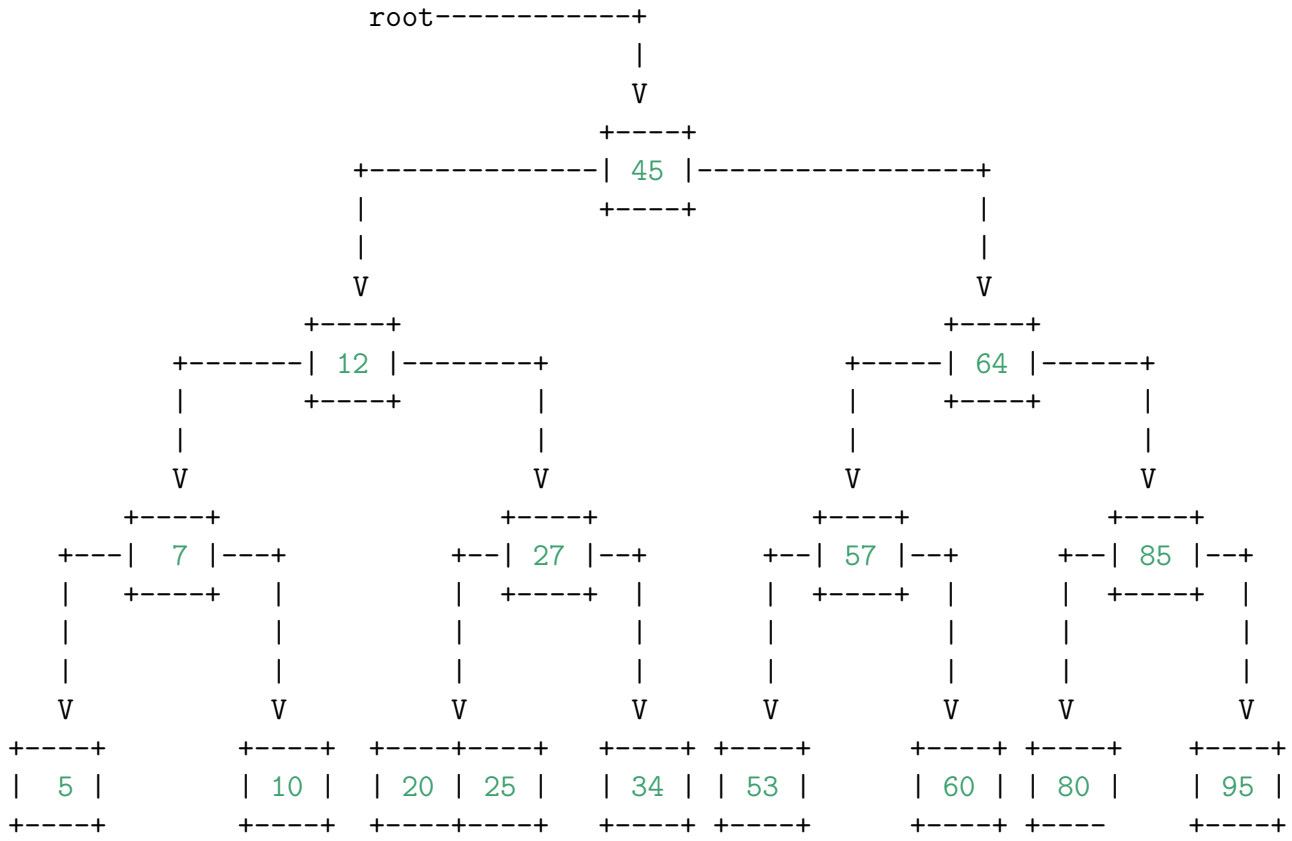
Transforma o 3-nó pai em um 4-nó.





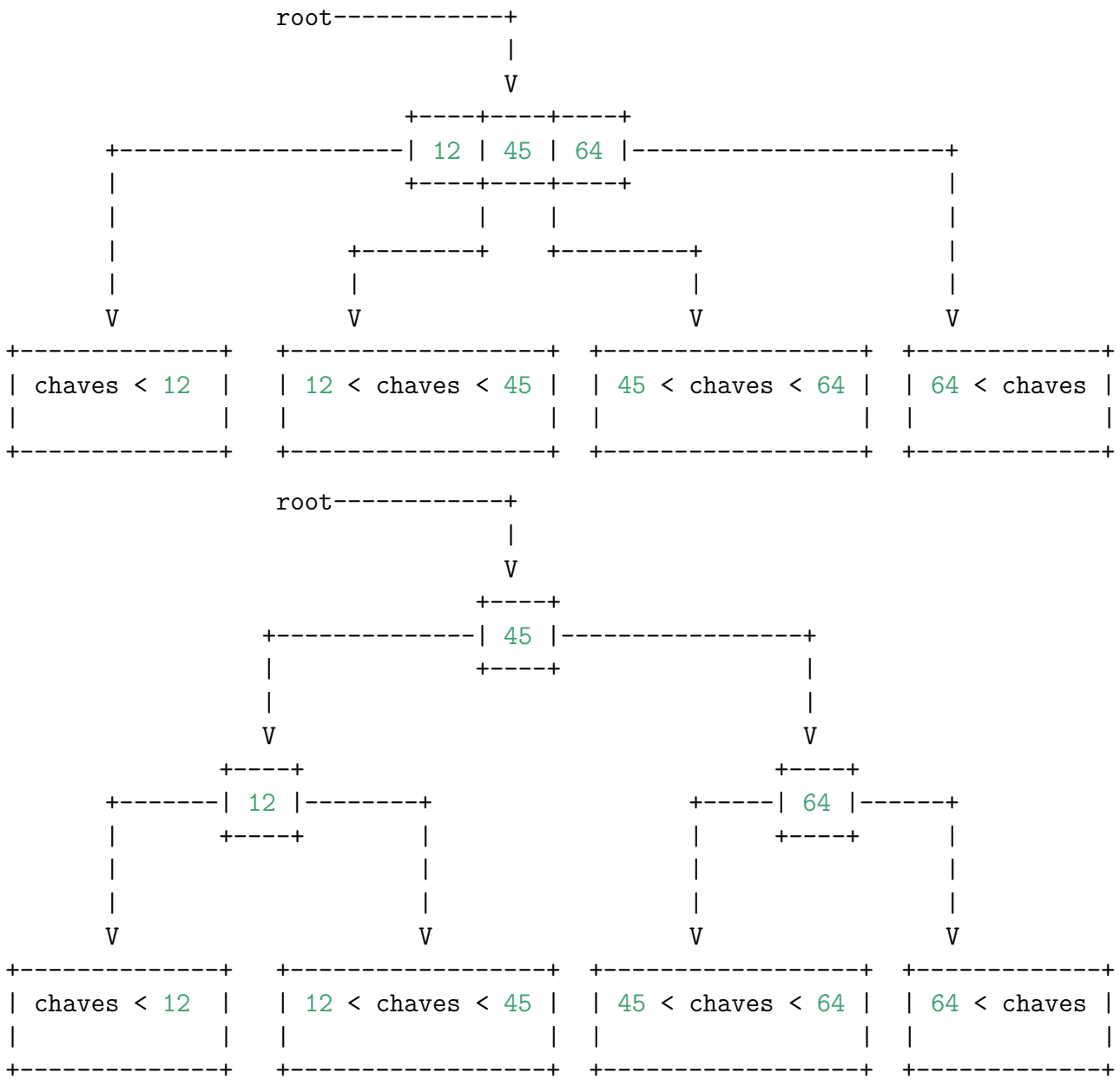
Exemplo: Divisão da raiz (não estraga o balanceamento, mas aumenta a altura):

Espatifa o 4-nó.



Altura foi incrementada!

Transformações preservam as propriedades globais da árvore



Desempenho de pior caso

Considere uma árvore 2-3 com `n` nós.

Se temos apenas nós simples, a altura da árvore será $\lfloor \lg \mathtt{n} \rfloor$.

Se temos apenas nós duplos, a altura da árvore será $\lfloor \log_3 \mathtt{n} \rfloor$.

Conclusão: a altura nunca passa de $\lg \mathtt{n}$.

Proposition F: Numa árvore 2-3 com N nós, busca e inserção nunca visitam mais que $\lg n$ nós, mesmo no pior caso. (Cada visita faz no máximo 2 comparações de chaves.)

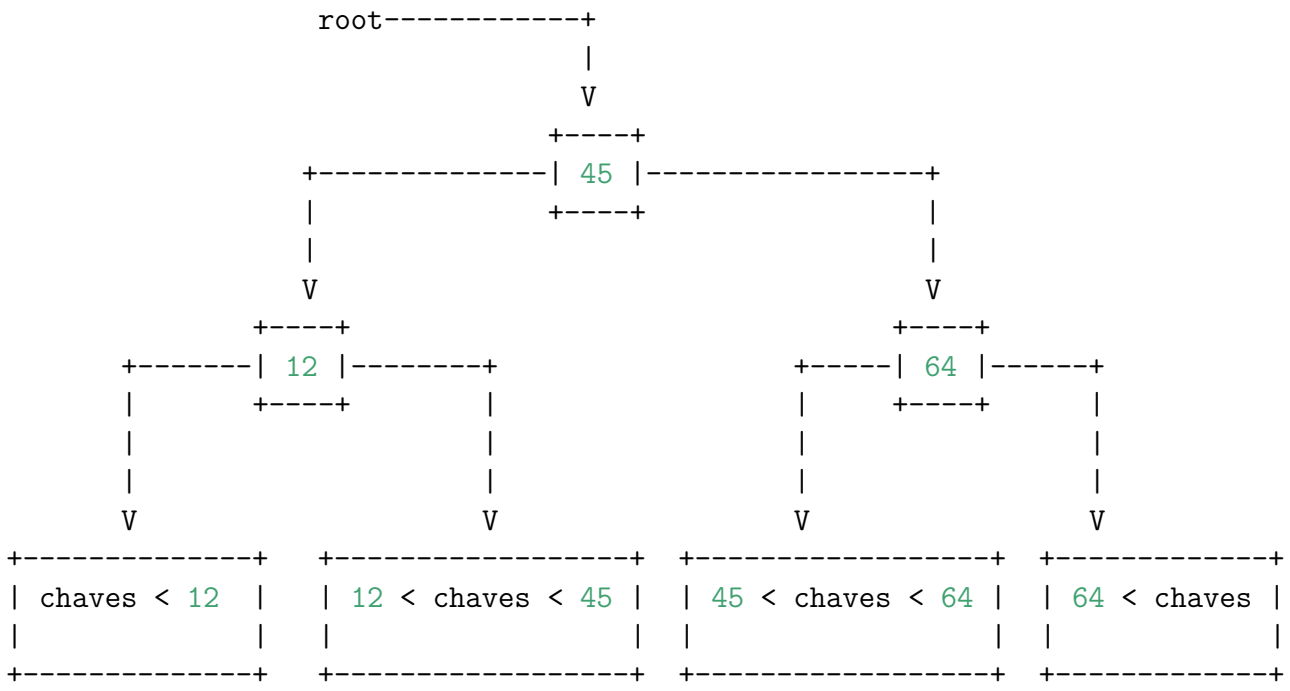
Remoção em árvore 2-3

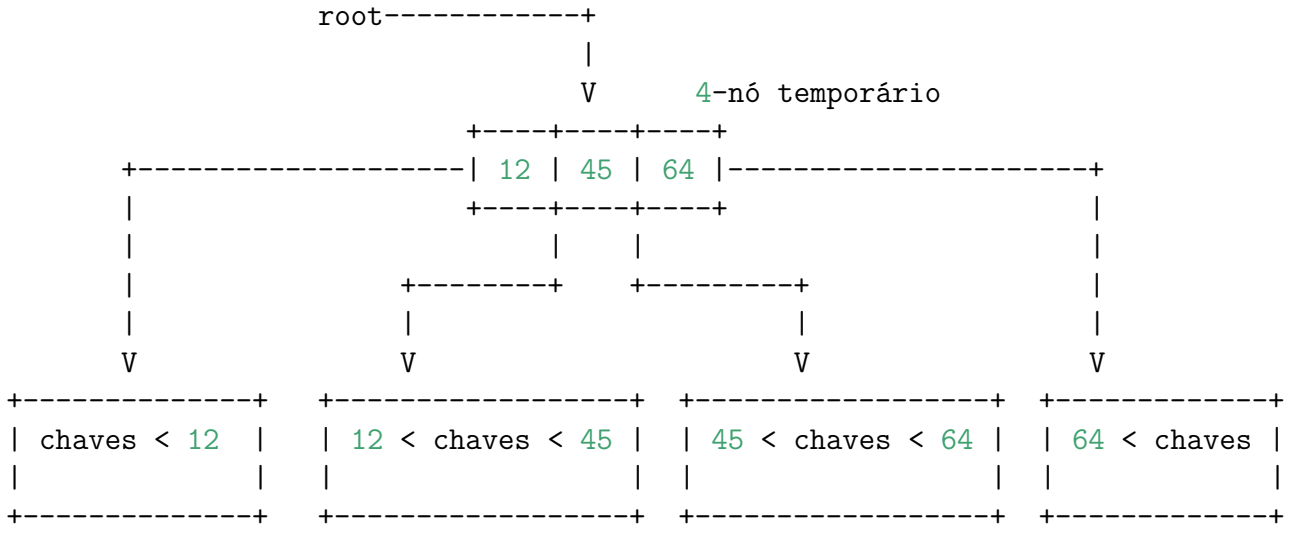
No caminho até a chave a ser removida o algoritmo manté a relação invariante

> o nó sendo examinado é um 3-nó ou um 4-nó (temporário)

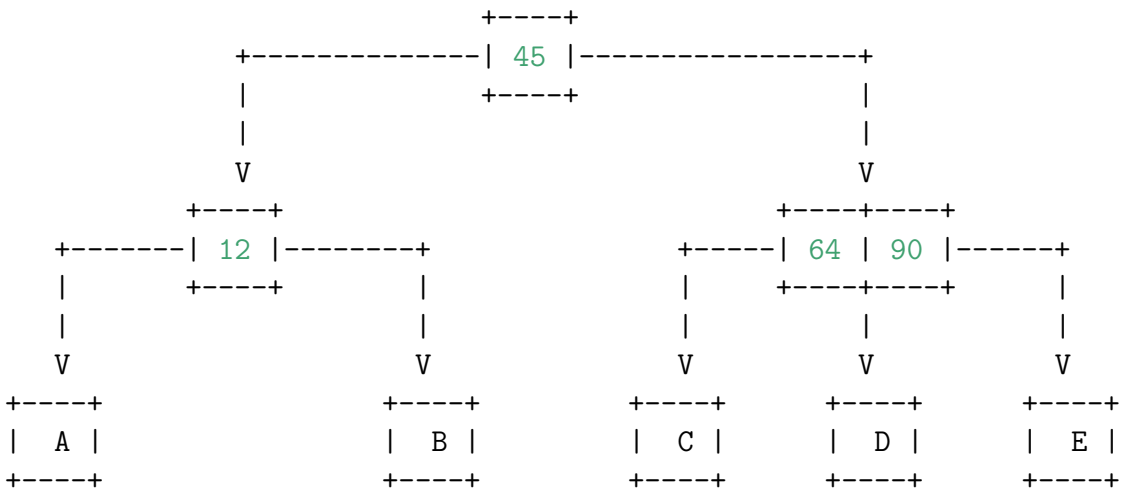
`deleteMin()`

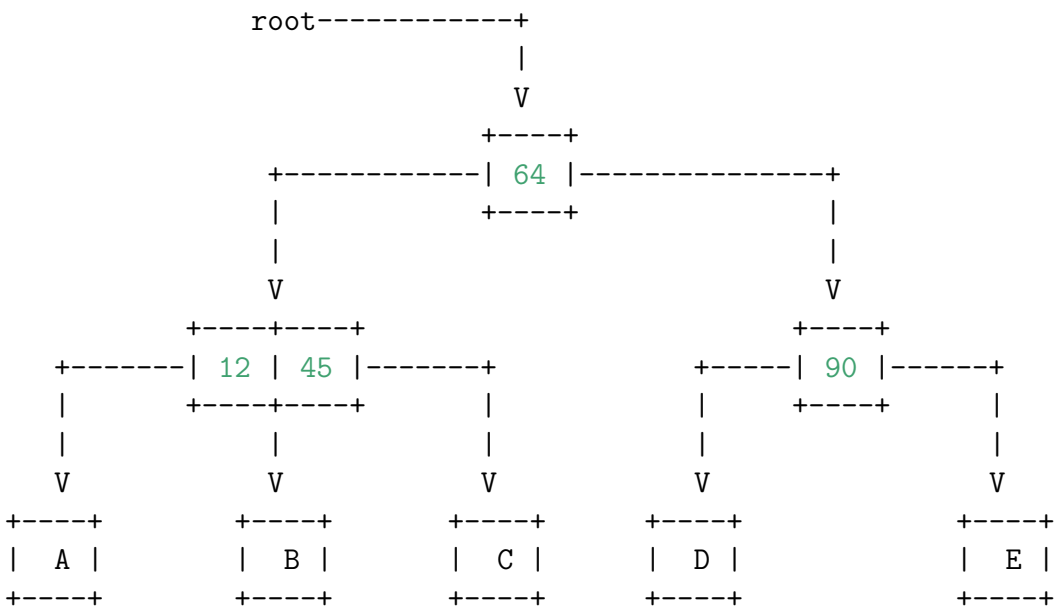
Comecemos com a raiz quando os dois filhos são 2-nós.





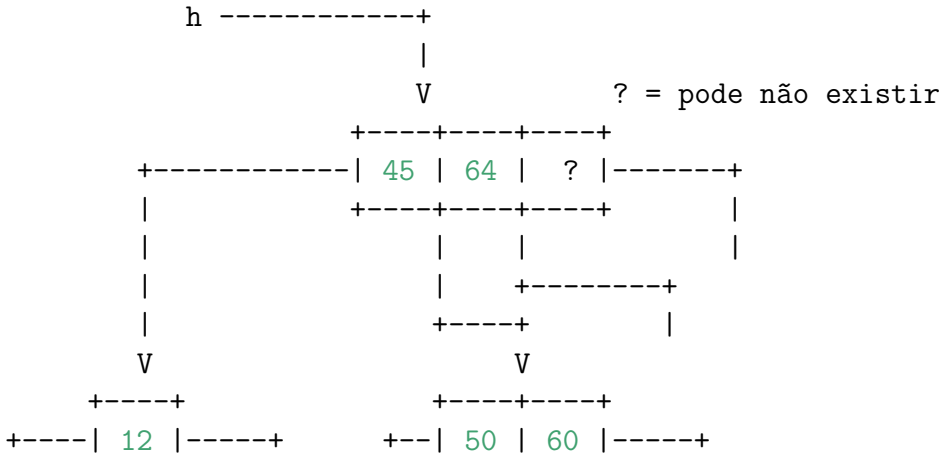
Agora passemos a raiz quando apenas o nó esquerdo é um 2-nó.

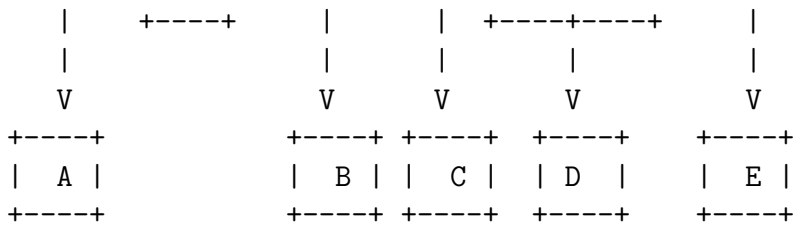


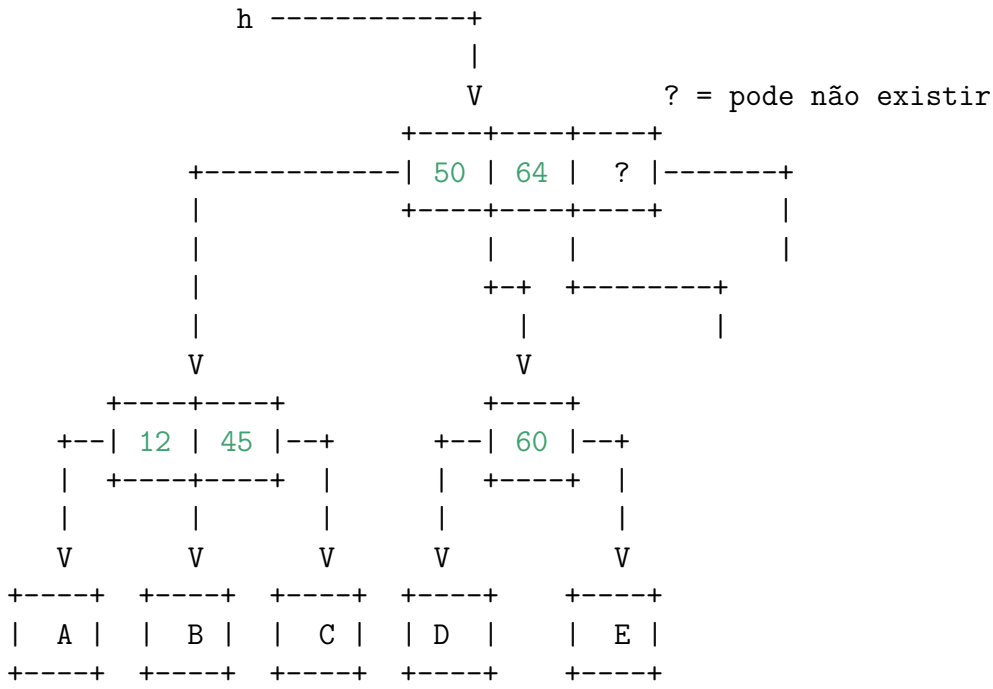


No meio do caminho, sabemos que o nó corrente `h` é um 3-nó ou um 4-nó. Antes de movermos para o nó mais à esquerda precisamos no certificar que esse nó é um 3-nó ou 4-nó. Se ele já é um 3-nó, não precisamos fazer nada.

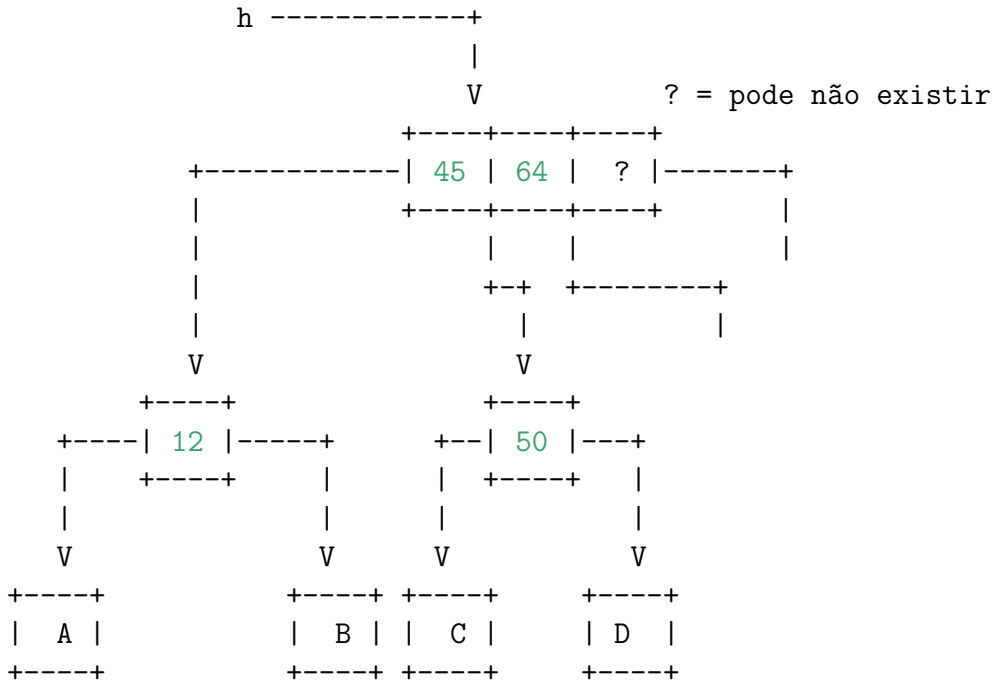
No caso em que o nó mais a equerda de `h` é um 2-nó

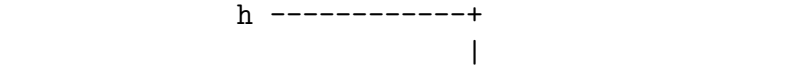


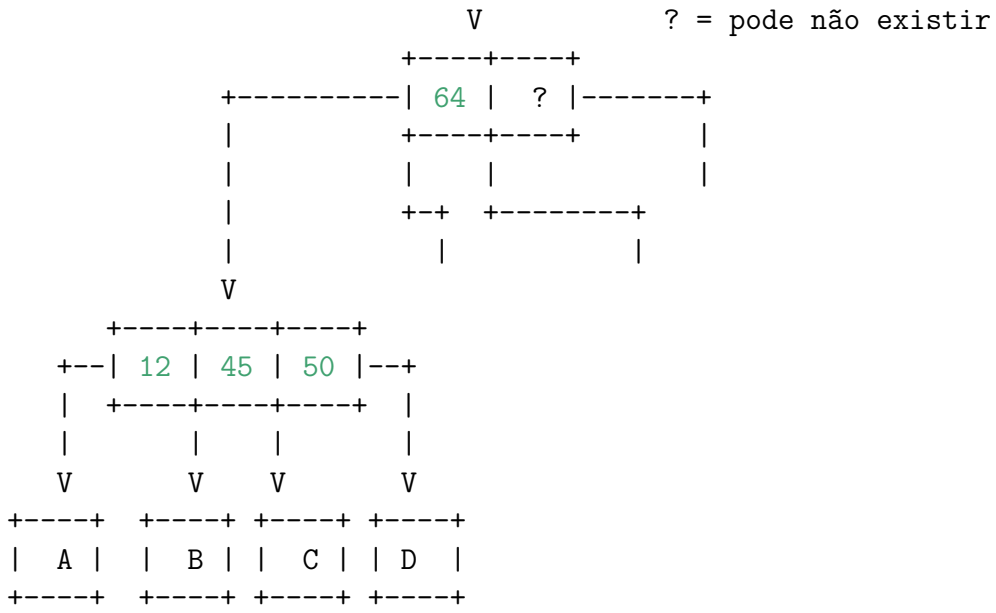




Os dois filhos mais à esquerda de h são 2-nós.







Desta forma quando atingirmos a folha mais a esquerda dessa árvore 2-3-4 teremos chegado a um 3-nó ou 4-no. Removendo o item mais a esquerda teremos um 2-nó ou 3-nó. Depois devemos voltar "espatifando" os 4-nós que por ventura deixamos pelo caminho. Hmm. Talvez seja importante notar que o pai de um 4-nó deixado pelo caminho, que não seja a raiz, é um 2-nó ou 3-nó.