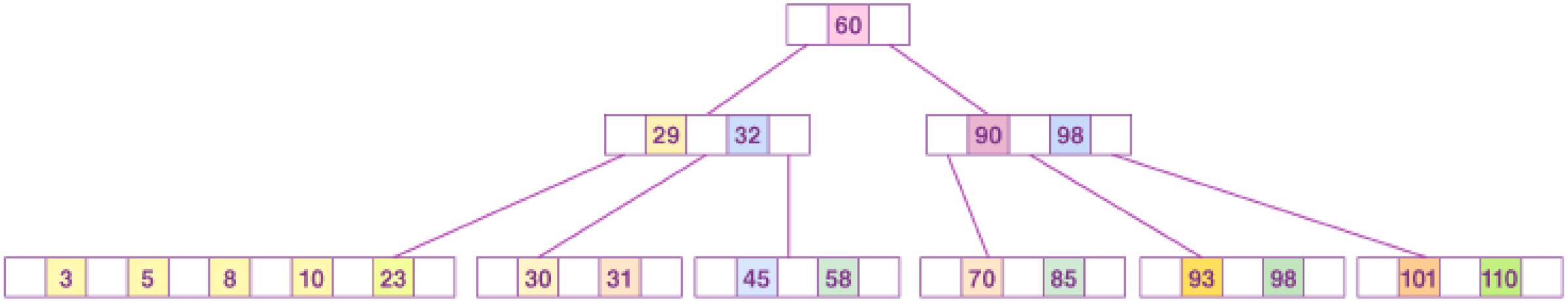
cin.ufpe.br

Monitoria de Algoritmos e Estruturas de Dados

Árvores-B

Árvores-B

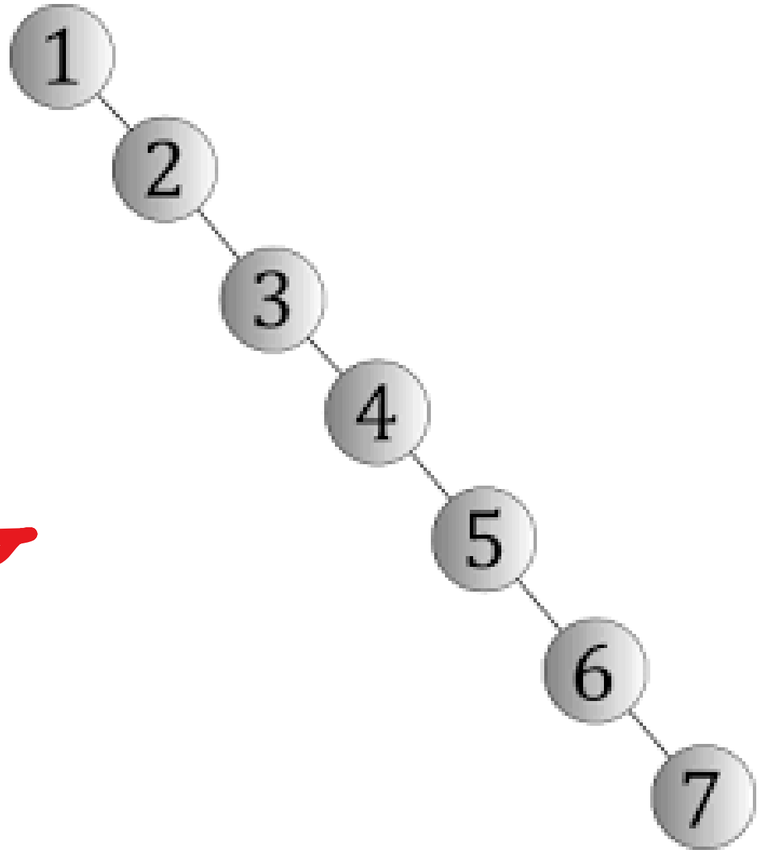


Intuição

- Mais “achatada” do que árvores de busca binária tradicionais
- Muitos valores dentro de um mesmo nó (valor **t** qualquer)
- Totalmente ordenada (Inclusive elementos de um mesmo nó)

Vantagens

- Auto-Balanceada

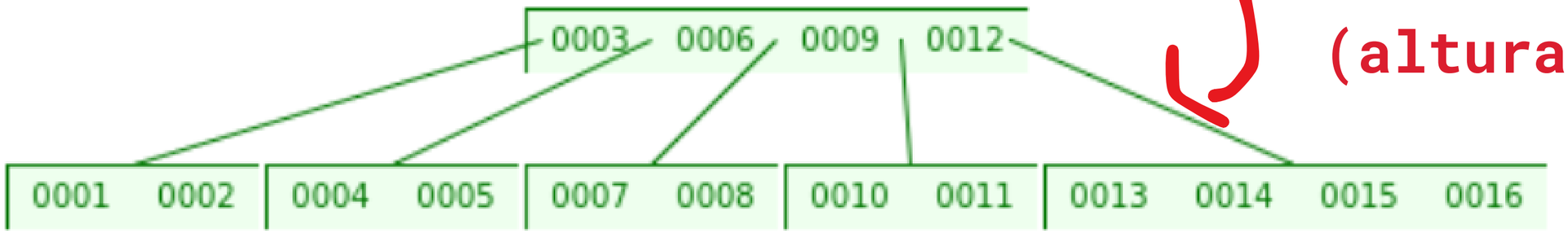


Não ocorre:

Vantagens

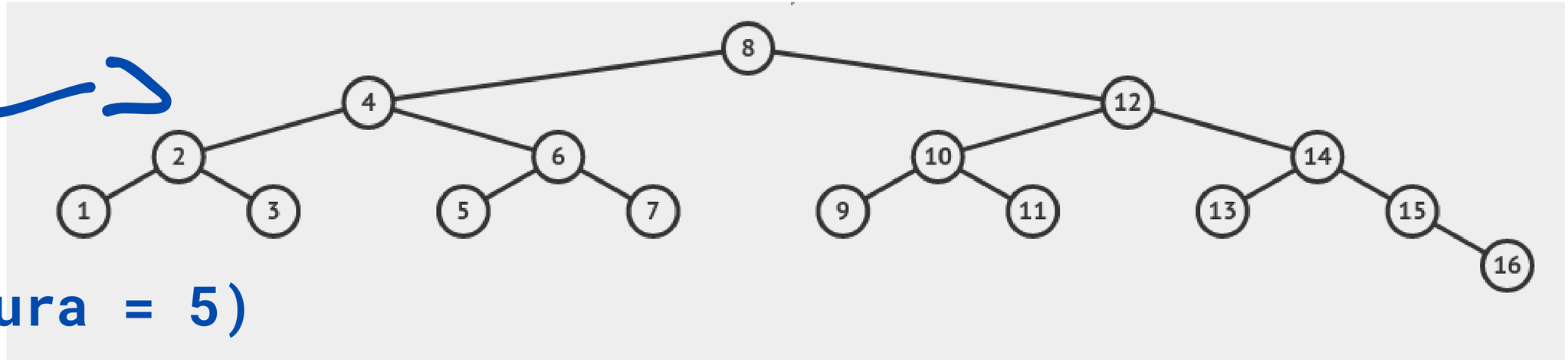
- Menor altura:

Árvore-B



(altura = 2)

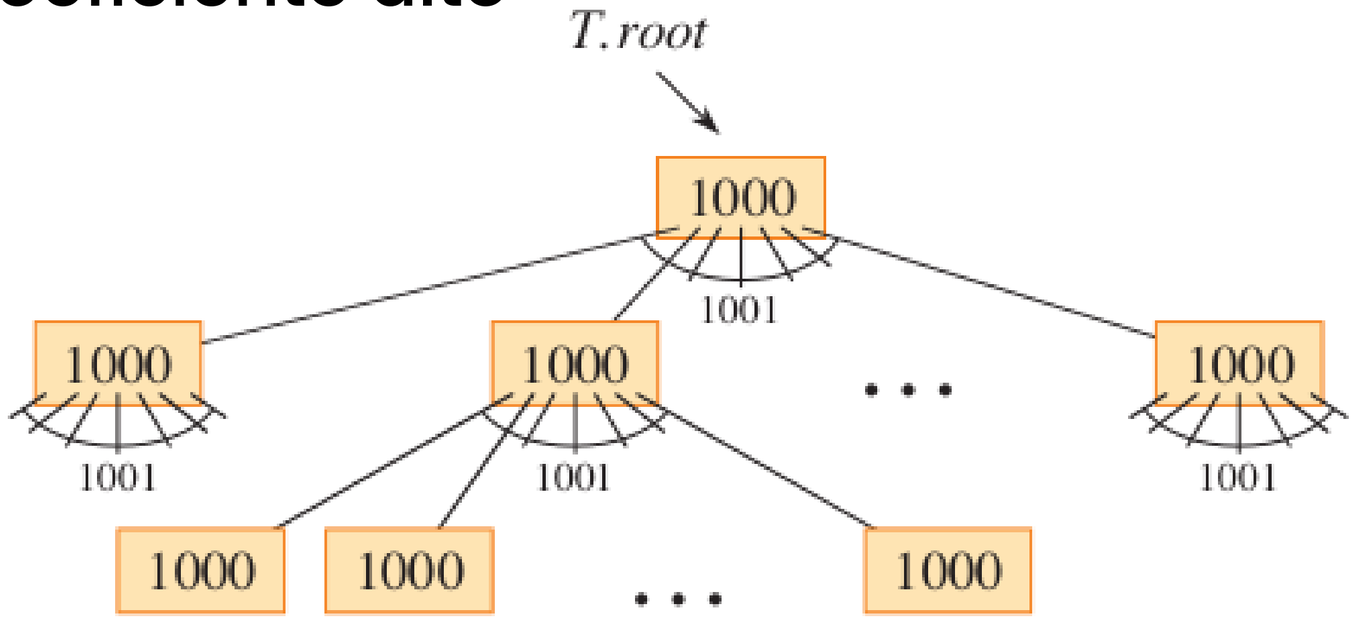
AVL



(altura = 5)

Vantagens

- Menor altura:
  - inserção de 1 bilhão de valores
  - coeficiente alto



1 node,
1000 keys

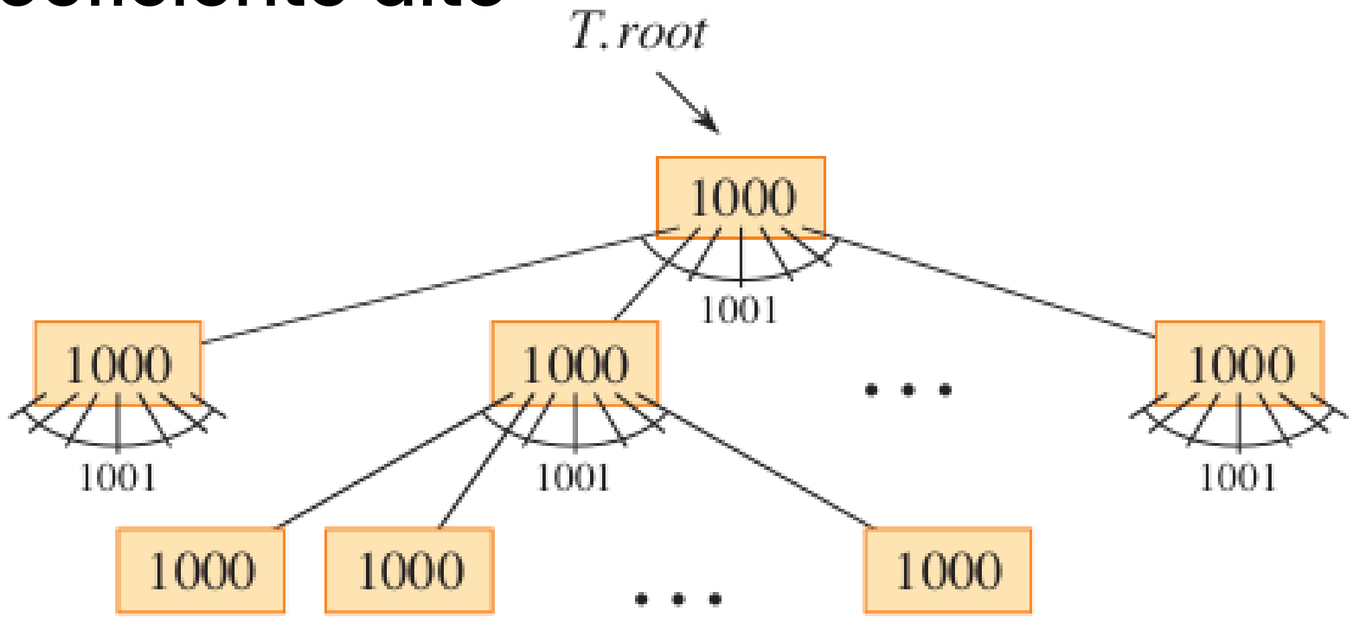
1001 nodes,
1,001,000 keys

1,002,001 nodes,
1,002,001,000 keys

Altura = 3

Vantagens

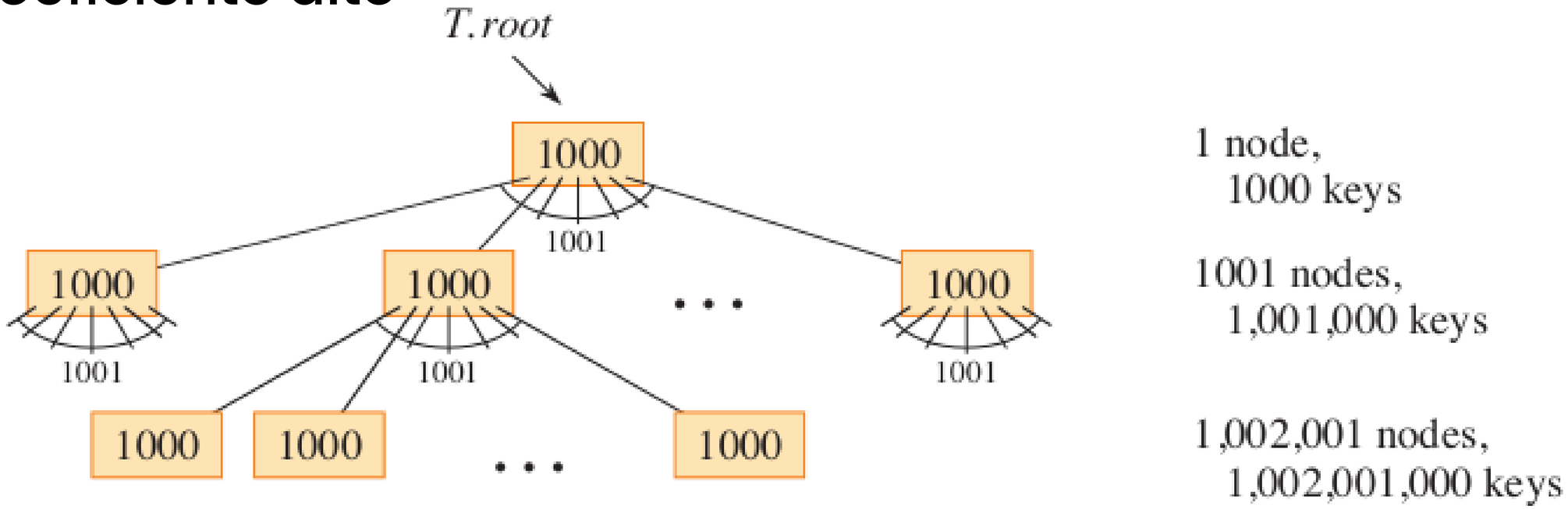
- Menor altura:
  - inserção de 1 bilhão de valores
  - coeficiente alto



Qual seria a altura de uma AVL com 1 bilhão de elementos?

Vantagens

- Menor altura:
  - inserção de 1 bilhão de valores
  - coeficiente alto



Qual seria a altura de uma AVL com 1 bilhão de elementos? **31**

Vantagens

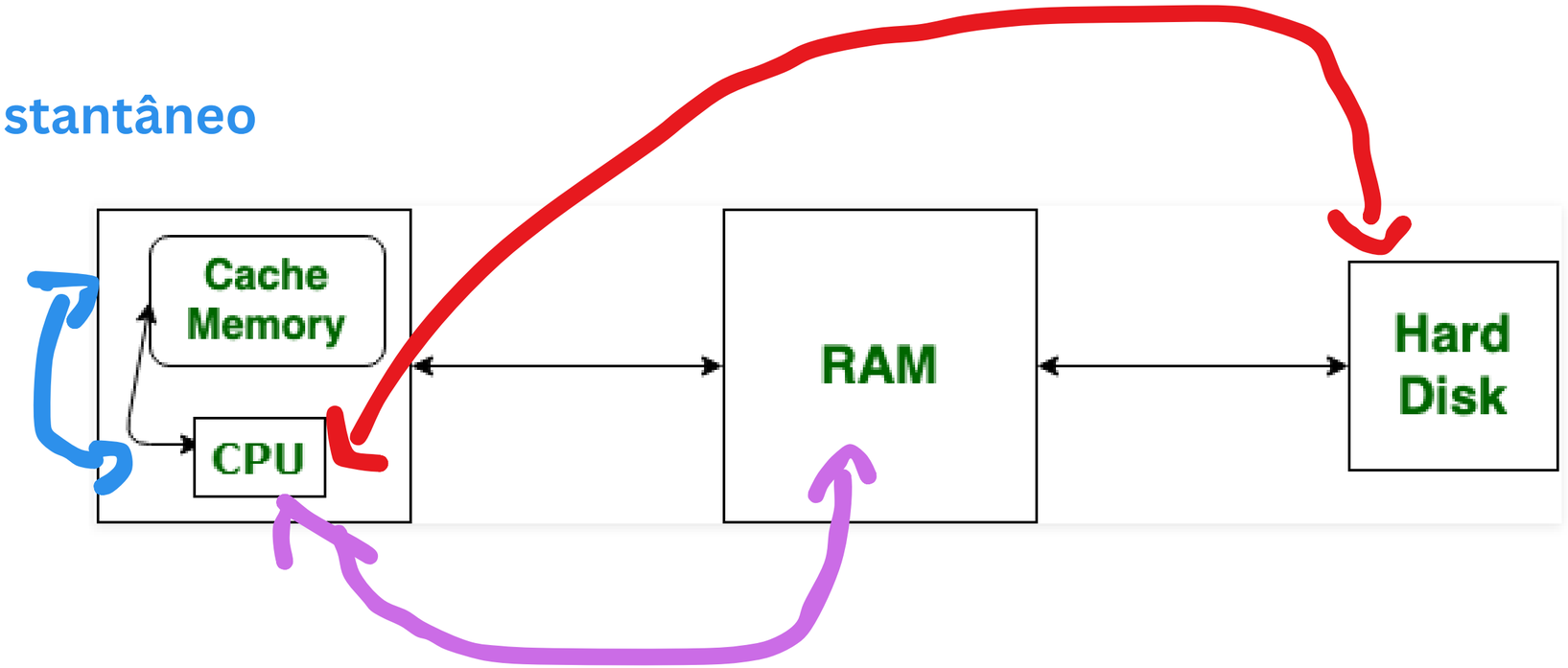
- Mas busca dentro de um nó com 1000 elementos continua ***O(1000)*** se for feita uma busca linear, então porque isso é melhor do que uma AVL?

possivelmente $O(\log_2 1000)$ com busca binária

Vantagens

Devagar, cerca de **80X**

Quase instantâneo



Rápido, algum valor **X**

Aplicações

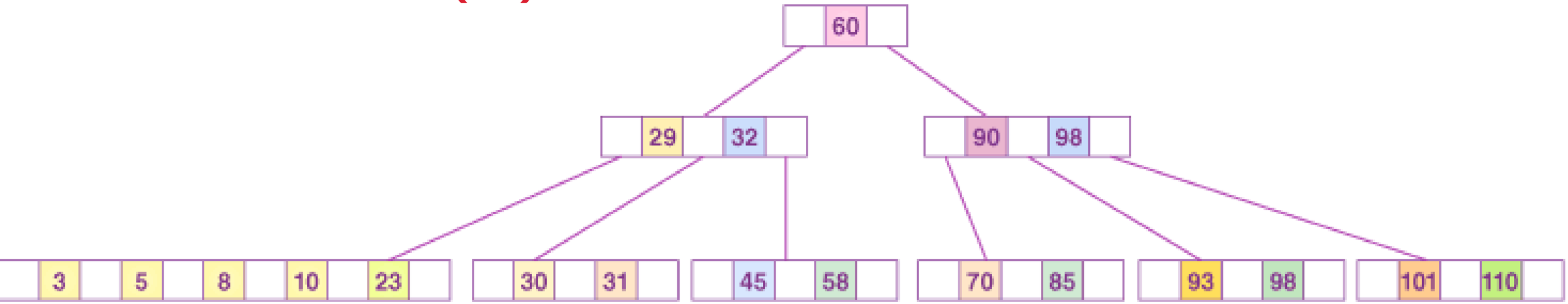
- Bases de dados: MySQL, SQLite
- Sistemas de arquivos:
  - Apple: MacOS HFS/HFS+
  - Windows: NTFS
  - Linux: ext3

Propriedades árvores-B

- Possui uma constante T que estabelece o grau mínimo da árvore
  - Todo nó interno, exceto a raiz, possui C chaves ($T-1 \leq C \leq 2T-1$) e F filhos ($F = C+1$)
- Todas as folhas estão no mesmo nível

Busca em árvore-B

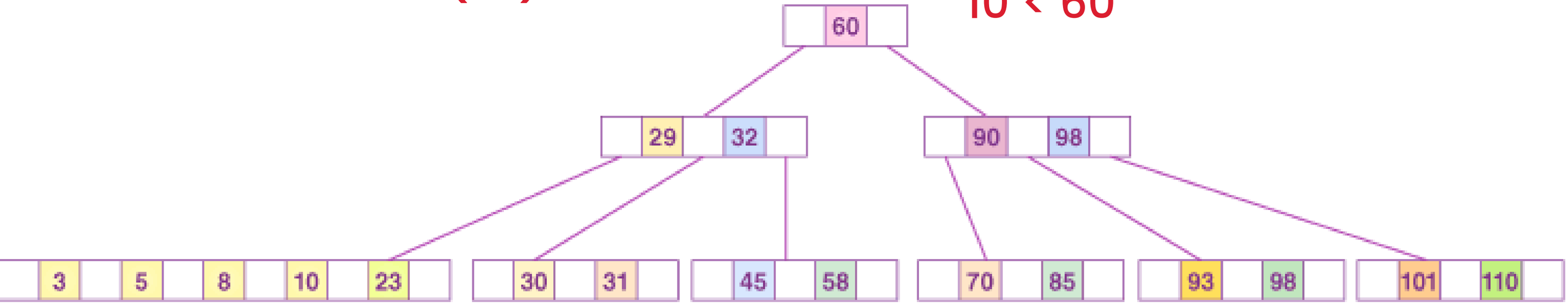
arvore.busca(10)



Busca em árvore-B

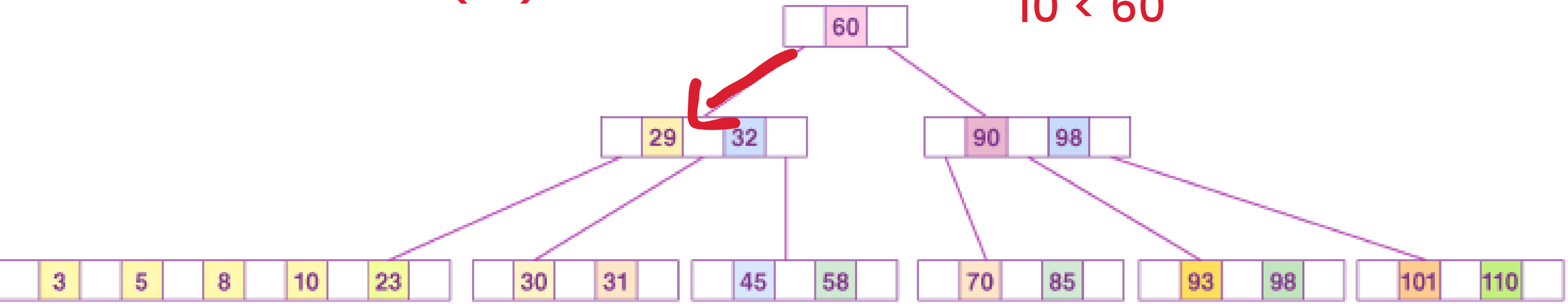
arvore.busca(10)

10 < 60



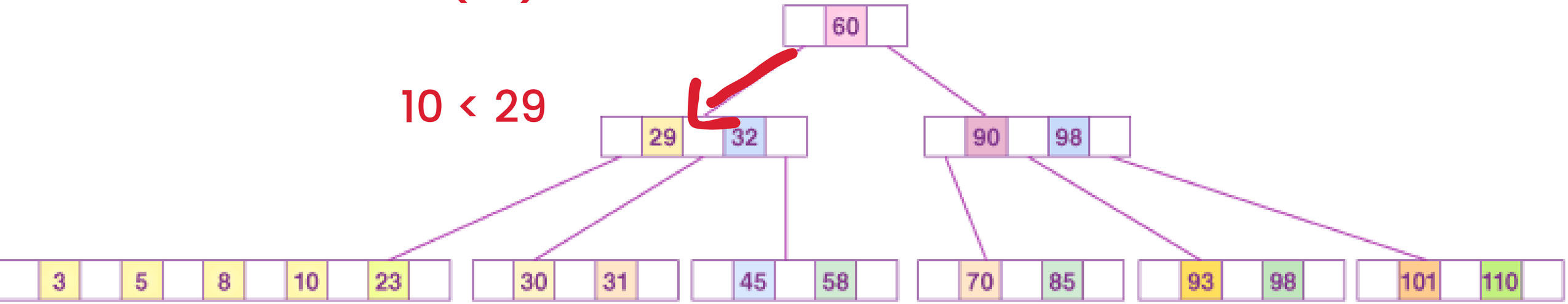
Busca em árvore-B

arvore.busca(10)



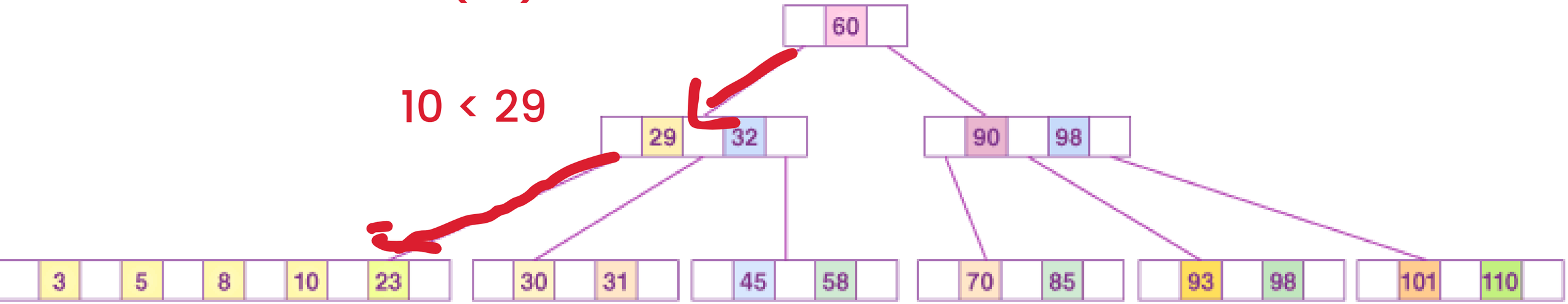
Busca em árvore-B

arvore.busca(10)



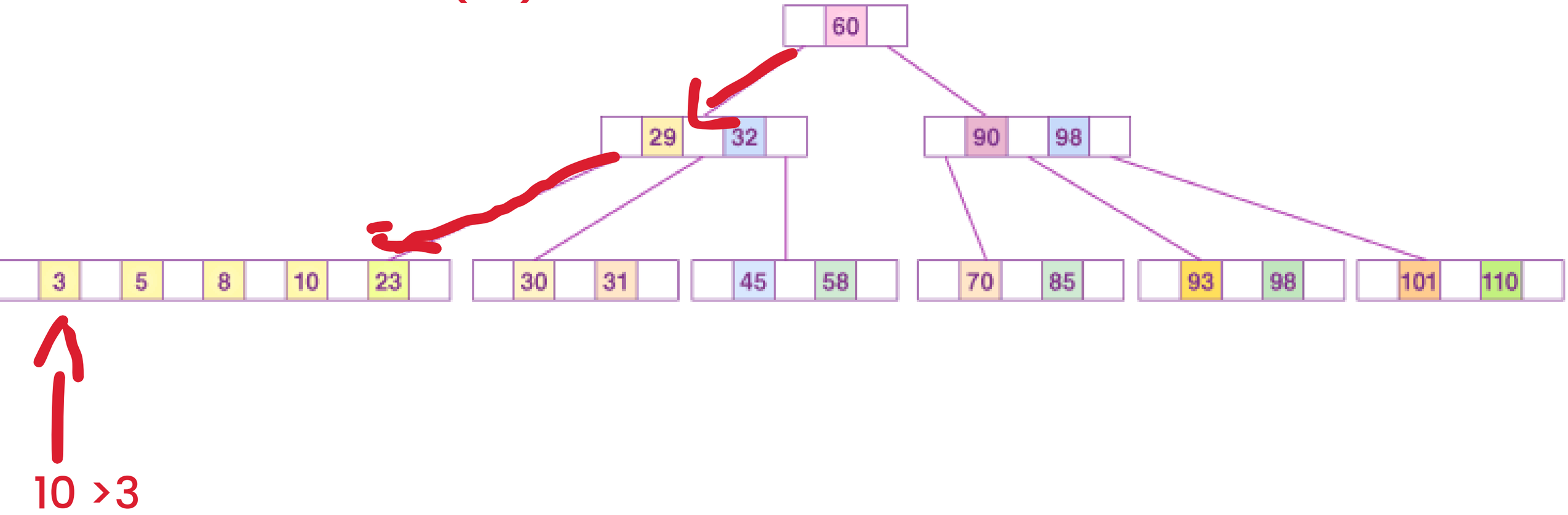
Busca em árvore-B

arvore.busca(10)



Busca em árvore-B

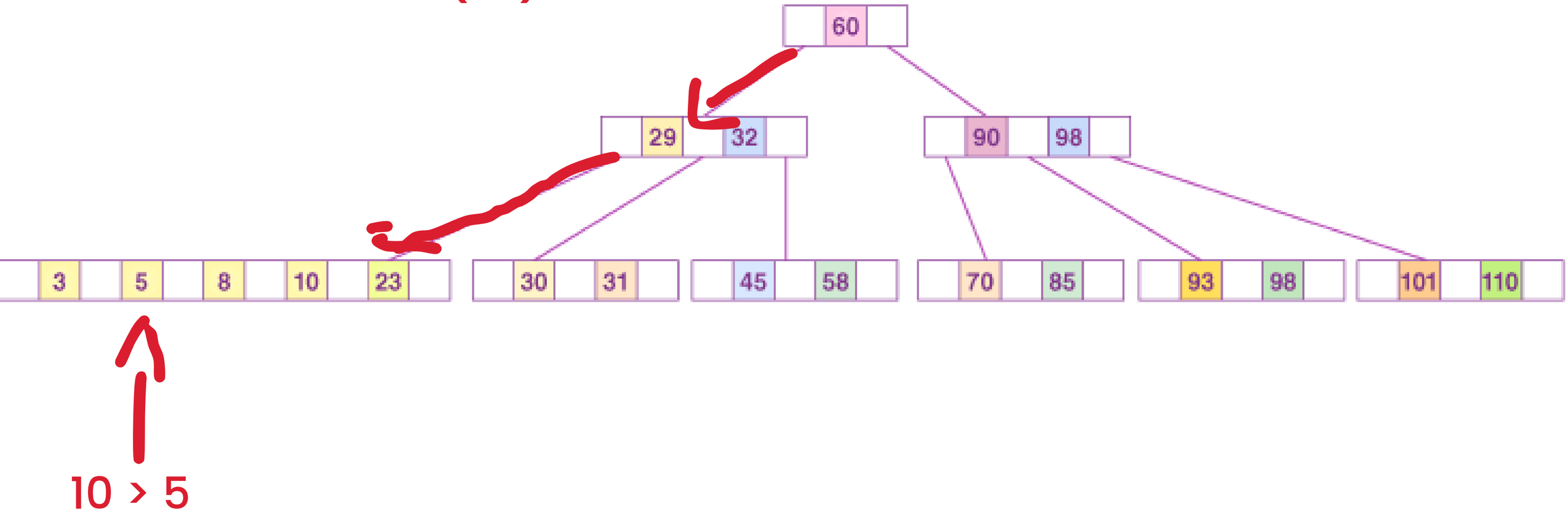
arvore.busca(10)



10 >3

Busca em árvore-B

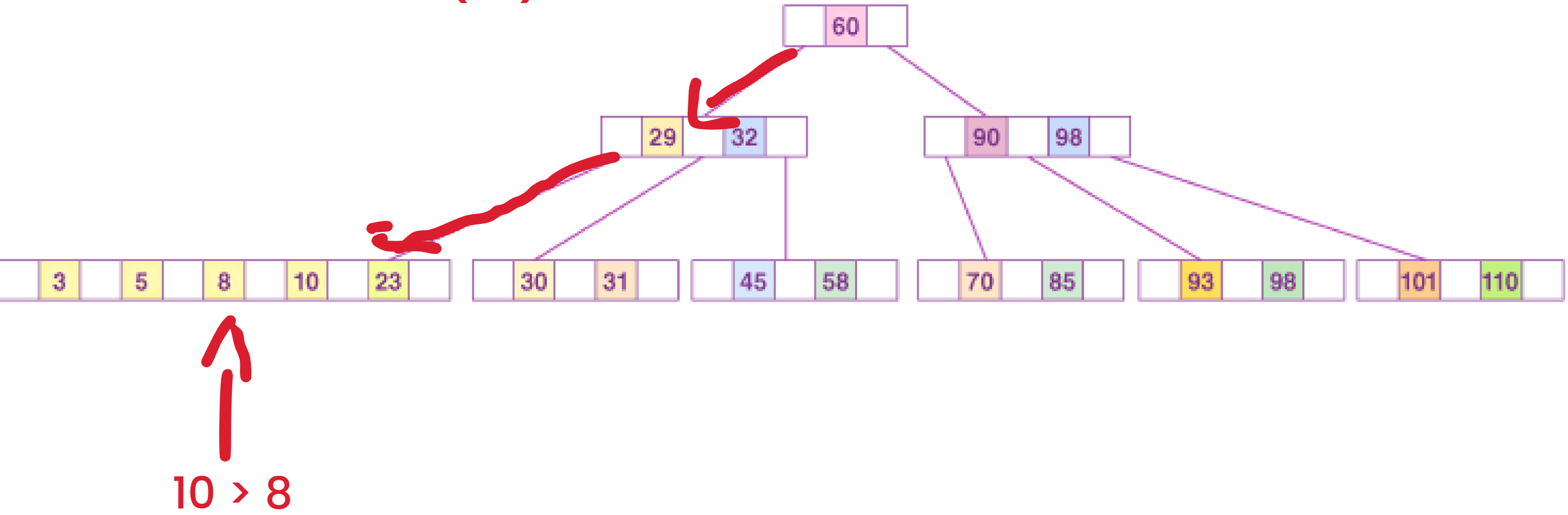
arvore.busca(10)



10 > 5

Busca em árvore-B

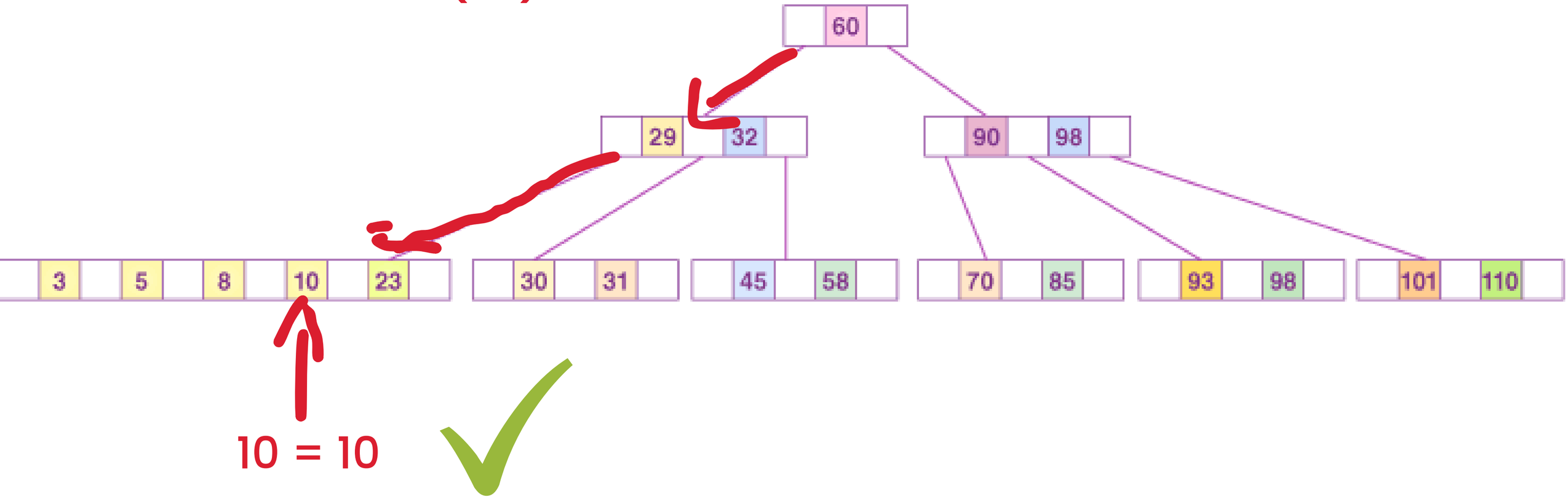
arvore.busca(10)



10 > 8

Busca em árvore-B

arvore.busca(10)



Inserção em árvore-B (T=3)

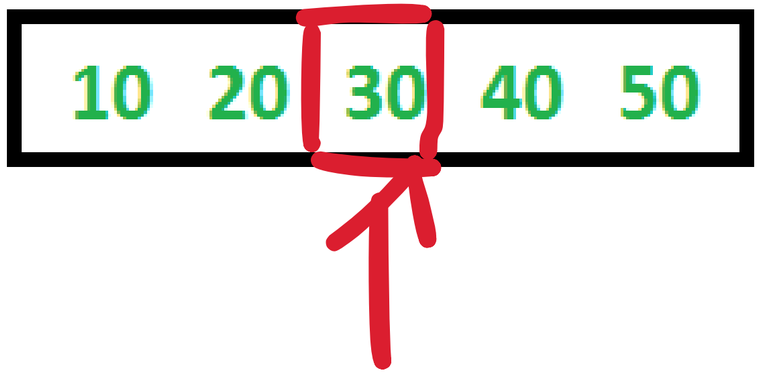
Insert 10

Inserção em árvore-B (T=3)

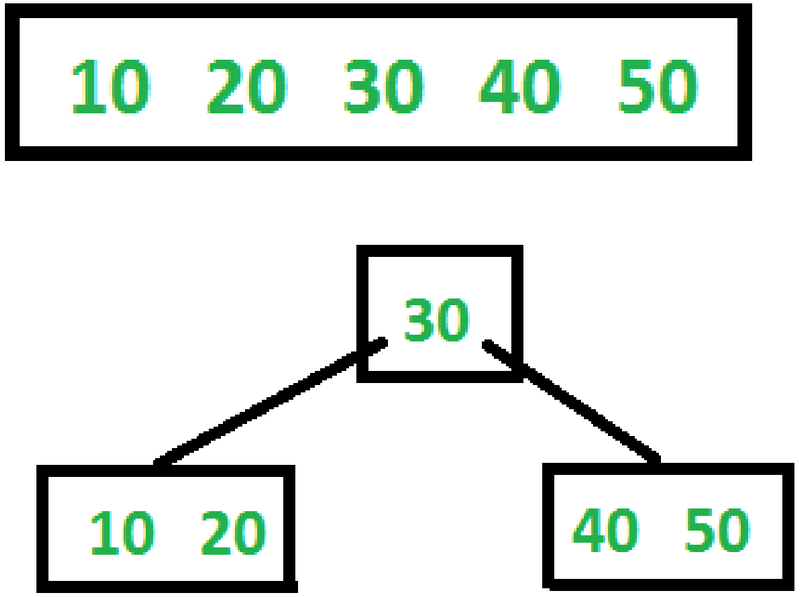
Insert 20, 30, 40 and 50

arvore.inserir(60)

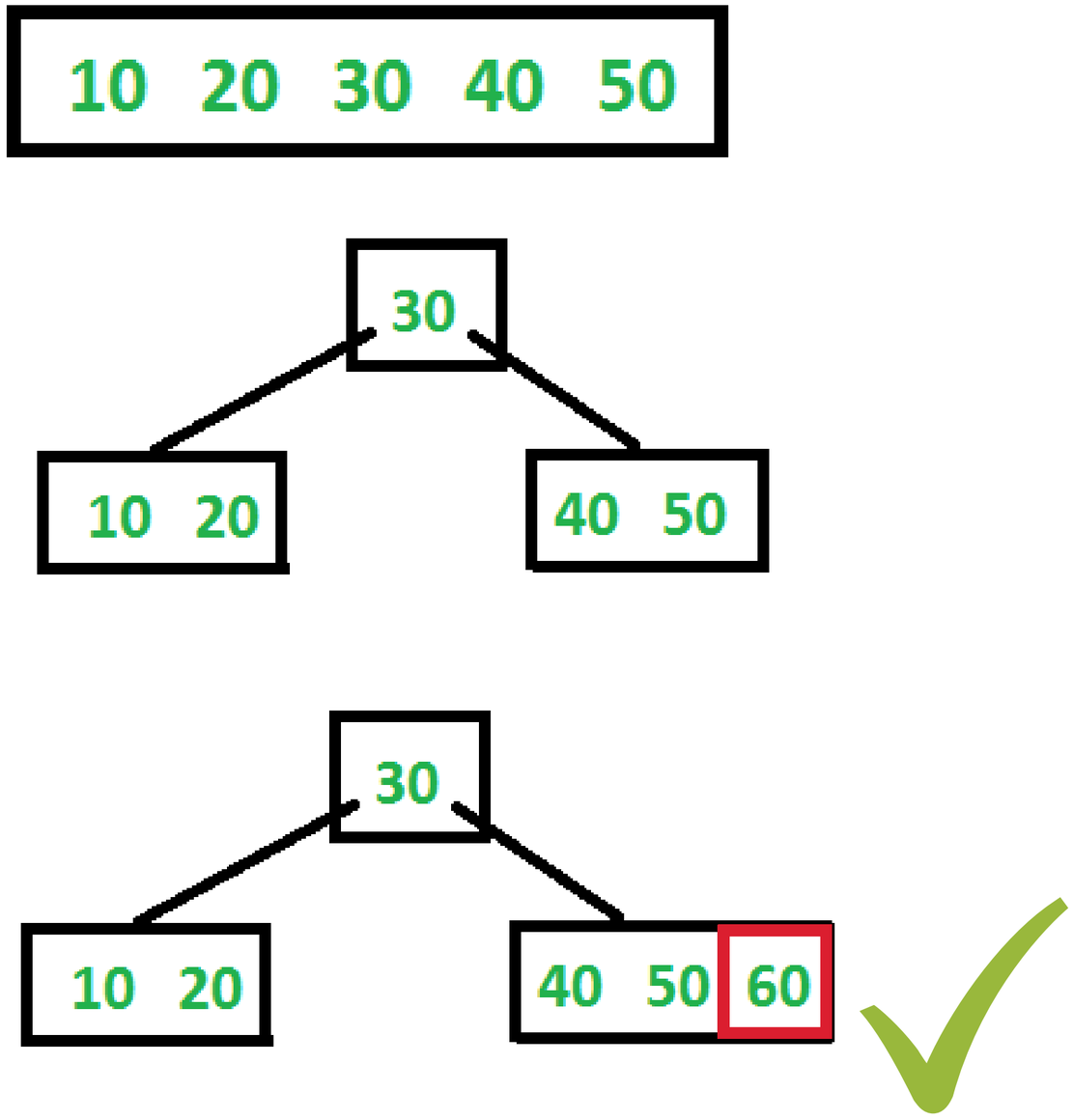
arvore.inserir(60)



arvore.inserir(60)



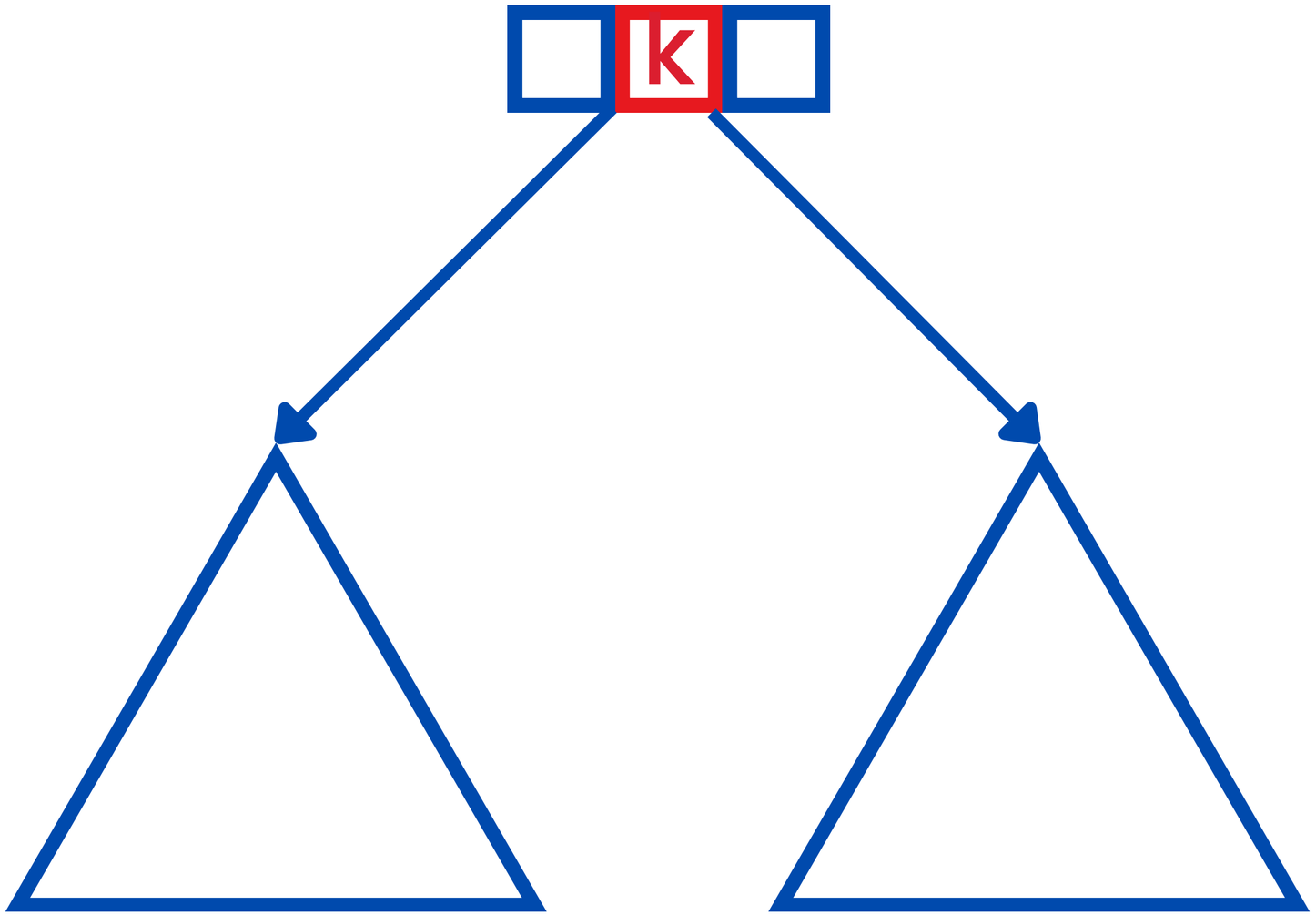
arvore.inserir(60)



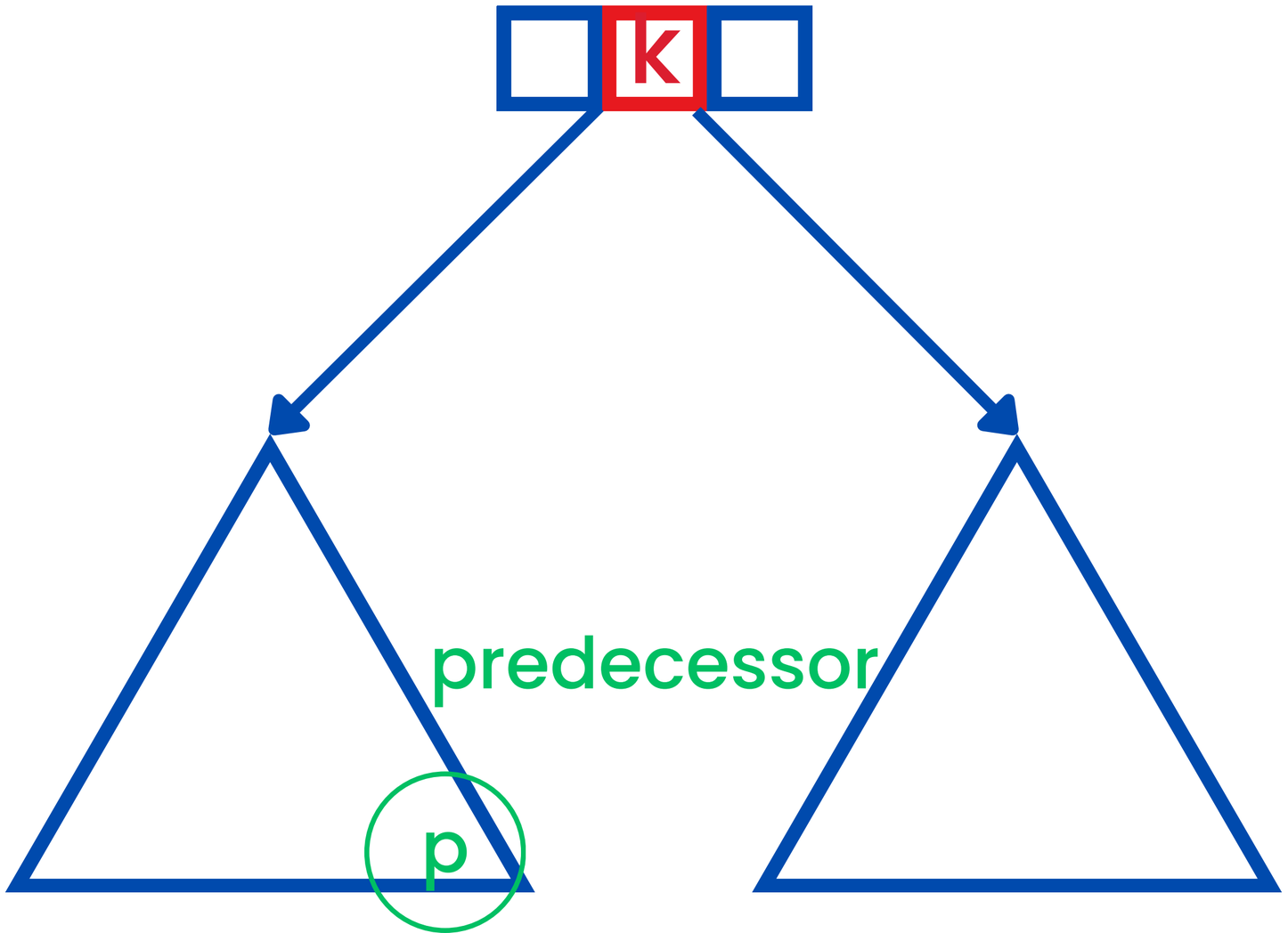
Deleção

- Muitos casos, importante é entender rotações que preservam propriedades da árvore

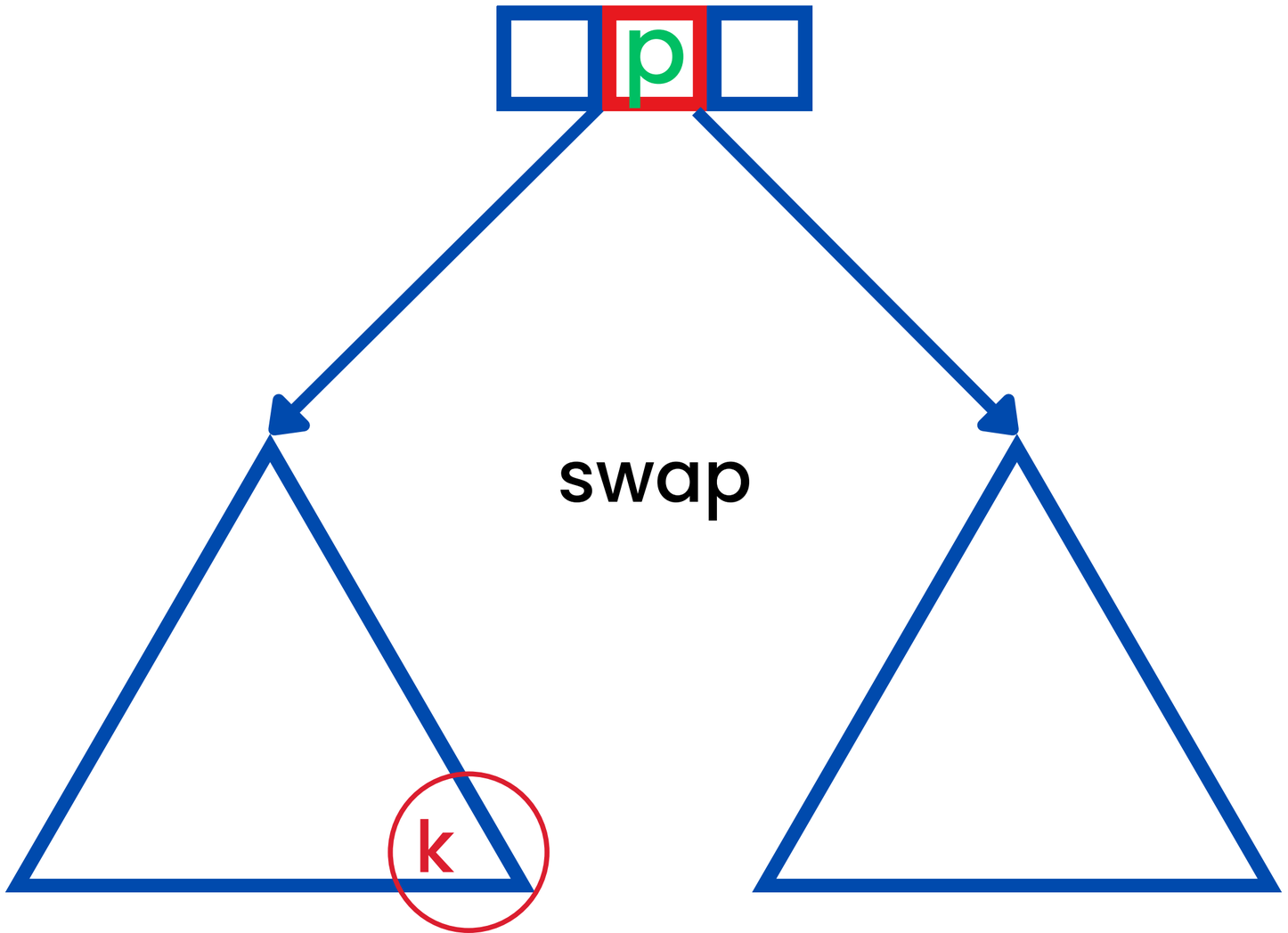
Deleção: deletando k



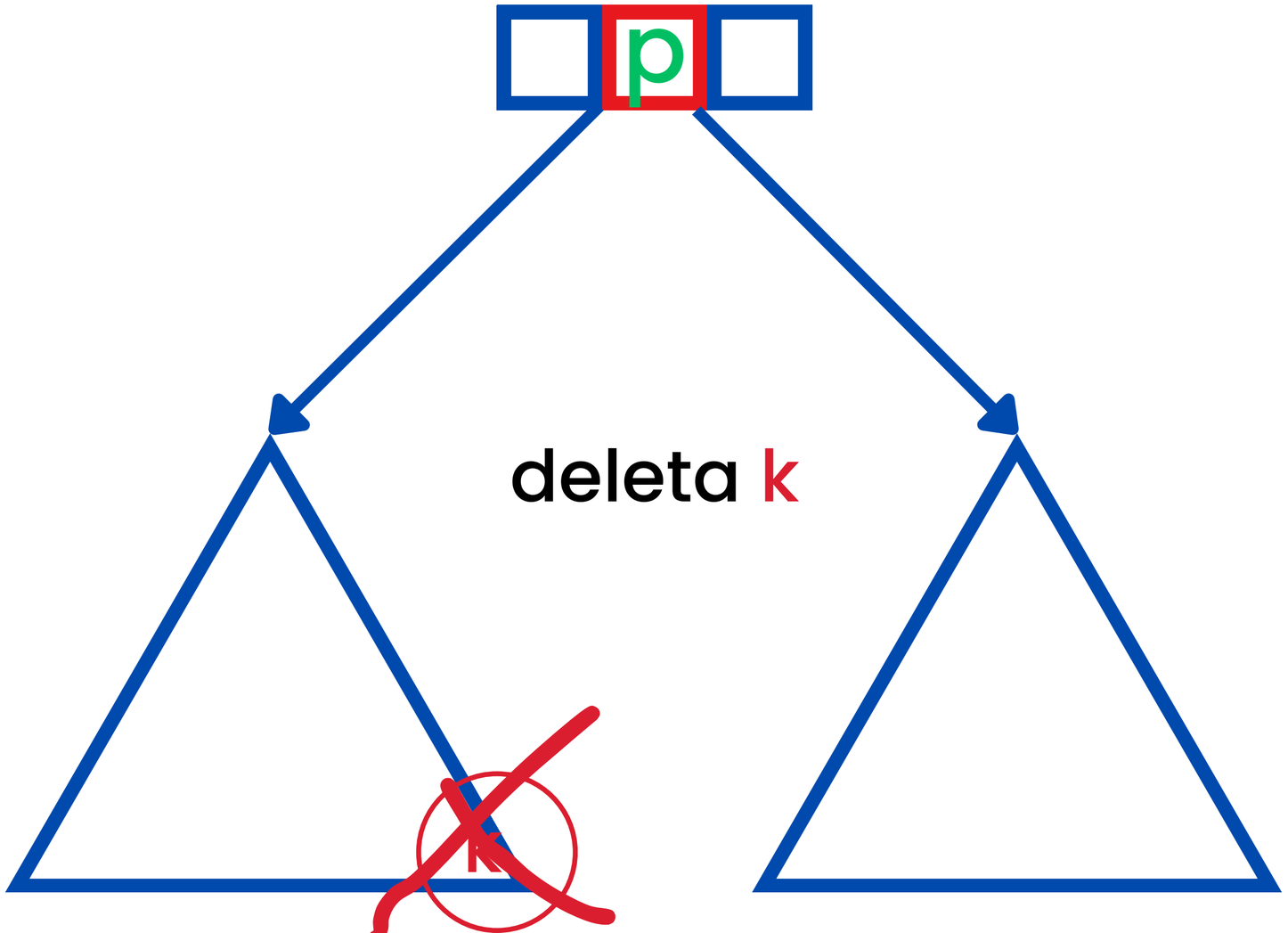
Deleção: deletando k



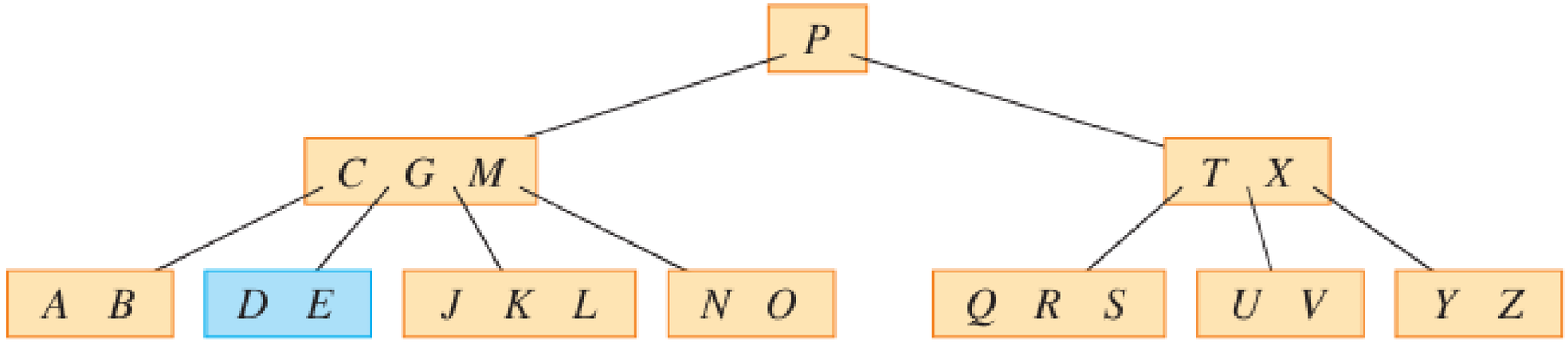
Deleção: deletando k



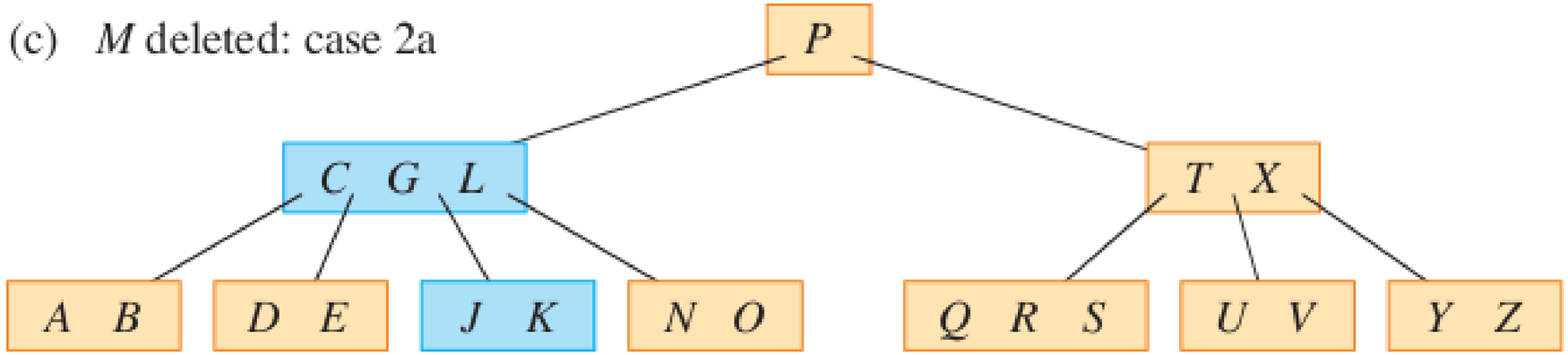
Deleção: deletando k



Deleção: exemplo



(c) *M* deleted: case 2a



Complexidade:

seja B coeficiente da árvore e N elementos:

Busca: $O(B \log_B N)$

Inserção: $O(B \log_B N)$

Deleção: $O(B \log_B N)$

Complexidade:

seja B coeficiente da árvore e N elementos:

Busca: $O(B \log_B N)$

Inserção: $O(B \log_B N)$

Deleção: $O(B \log_B N)$

Em última análise, tempo logarítmico.

Referências:

Livro: Introduction to Algorithms, *Cormen et Al*

Videoaula: MIT Opencourseware

Visualização Árvore-B