"Ao contrário, continuou Tweedledee, se foi assim, poderia ser; e se fosse assim, seria; mas como não é, então não é. Isso é Lógico" (Lewis Carroll, Through the Looking-glass: And what Alice Found There, 1875).

# Variantes de Árvores B

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

# Considere

Considere um Árvore B, onde t=2.

Quantas chaves cada nodo deve armazenar?

Quantos filhos cada nodo pode ter?

# Considere

Considere um Árvore B, onde t=2.

Quantas chaves cada nodo deve armazenar?

1, 2 ou 3 chaves.

Quantos filhos cada nodo pode ter?

2, 3 ou 4 filhos.

# Considere

Considere um Árvore B, onde t=2.

Quantas chaves cada nodo deve armazenar?

1, 2 ou 3 chaves.

Quantos filhos cada nodo pode ter?

2, 3 ou 4 filhos.

Uma Árvore 2-3-4 (também conhecida como 2-4) é o caso especial da Árvore B com t =2, pois cada nodo pode ter 2, 3, ou 4 filhos.

# Uso

Por que árvores B são importantes?

# Uso

Por que árvores B são importantes?

- São árvores balanceadas.
- Reduzem o acesso ao armazenamento secundário e o usam de forma eficiente.
  - Mantém os nodos de forma a ocupar blocos do disco.
  - Têm menor altura quando comparadas a árvores binárias, por exemplo.

# Uso

Por que árvores B são importantes?

- São árvores balanceadas.
- Reduzem o acesso ao armazenamento secundário e o usam de forma eficiente.
  - Mantém os nodos de forma a ocupar blocos do disco.
  - Têm menor altura quando comparadas a árvores binárias, por exemplo.

Uma árvore 2-3-4 mantém quais dessas características?

# Árvore 2-3-4

Podemos manter uma árvore 2-3-4 na memória principal, e temos a garantia de que a árvore é balanceada.

Não precisamos alterar nenhum algoritmo estudado.

Poderíamos também manter árvores com valores mais altos para *t*.

Qual problema em potencial?

# Árvore 2-3-4

Podemos manter uma árvore 2-3-4 na memória principal, e temos a garantia de que a árvore é balanceada.

Não precisamos alterar nenhum algoritmo estudado.

Poderíamos também manter árvores com valores mais altos para *t*.

Qual problema em potencial?

Em uma Árvore B, os nodos podem não estar cheios.

Na média, estão 69% cheios.

Desperdício de memória principal.

# Economizando

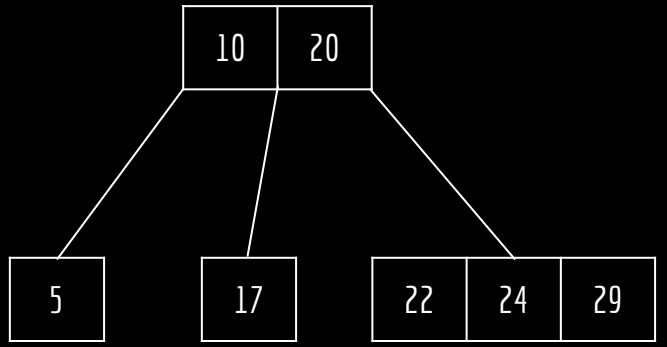
Podemos transformar a Árvore 2-3-4 em uma Árvore de Busca Binária (BST) para economizar memória.

Necessário armazenar a BST de forma que seja possível restaurar a Árvore 2-3-4 posteriormente, se necessário.

# Transformando em uma BST

Necessários dois tipos de links entre nodos:

- Representar chaves que pertencem ao mesmo nodo na 2-3-4.




- Representar conexões entre pai e filho.

# Transformando em uma BST

Necessários dois tipos de links entre nodos:

- Representar chaves que pertencem ao mesmo nodo na 2-3-4.
  - Bayer chamou de ponteiros horizontais.
  - Guibas and Sedgewick chamaram de nodos vermelhos.

- Representar conexões entre pai e filho.
  - Bayer chamou de ponteiros horizontais.
  - Guibas and Sedgewick chamaram de nodos pretos.

# Transformando em uma BST

Necessários dois tipos de links entre nodos:

- Representar chaves que pertencem ao mesmo nodo na 2-3-4.
  - Bayer chamou de ponteiros horizontais.
  - Guibas and Sedgewick chamaram de nodos vermelhos.

- Representar conexões entre pai e filho.
  - Bayer chamou de ponteiros horizontais.
  - Guibas and Sedgewick chamaram de nodos pretos.

Árvores VH e Red-Black.

# Transformando 2-3-4 em Red-Black

Temos 3 possibilidades.

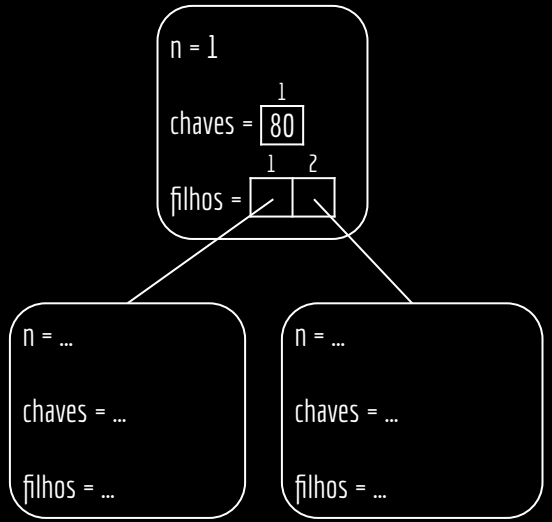
Nodos com 1 chave, 2 chaves, ou 3 chaves

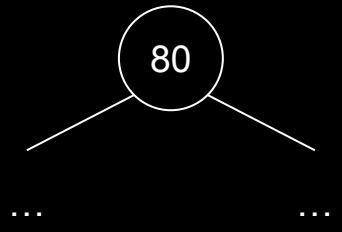
# Nodo com 1 chave

Nodo com 1 chave e 2 filhos.

Se torna uma subárvore de busca binária convencional.

Nodo preto com dois filhos.



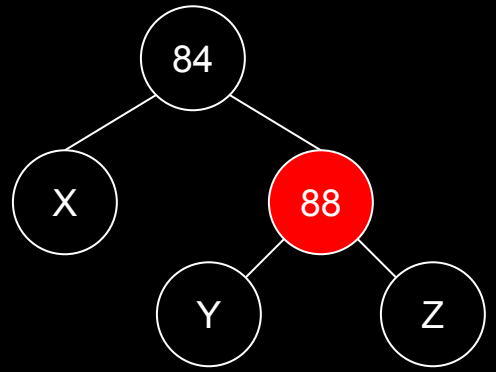
# Nodo com 2 chaves

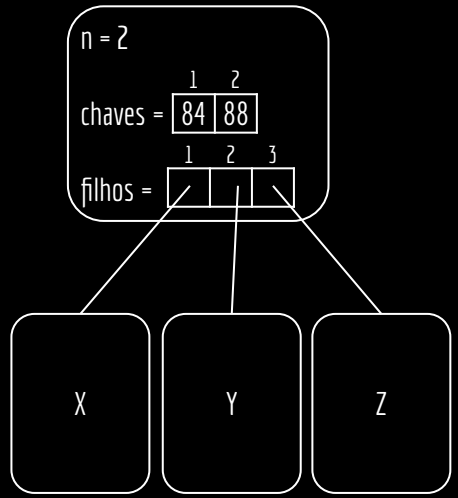
Nodo com 2 chaves e 3 filhos.

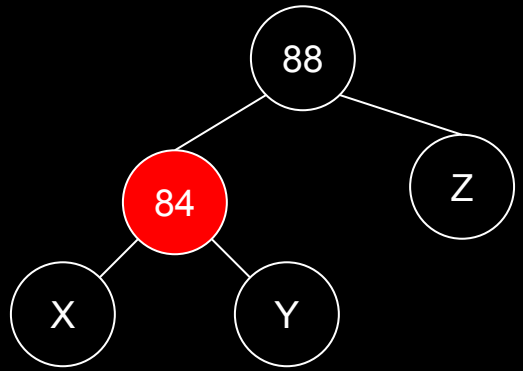
Duas possibilidades.

Primeira chave é nodo preto, e segunda é filho direito vermelho.

Segunda chave é nodo preto, e segunda é filho esquerdo vermelho.



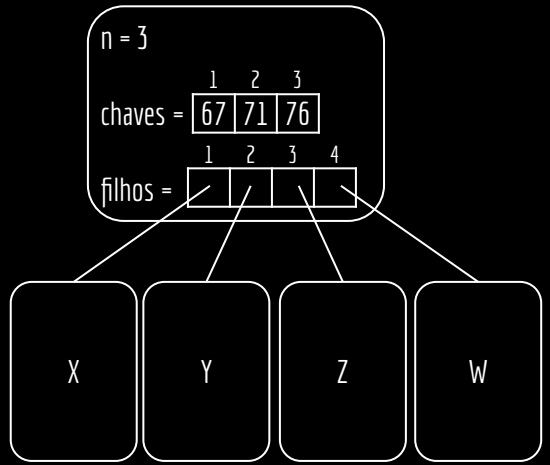



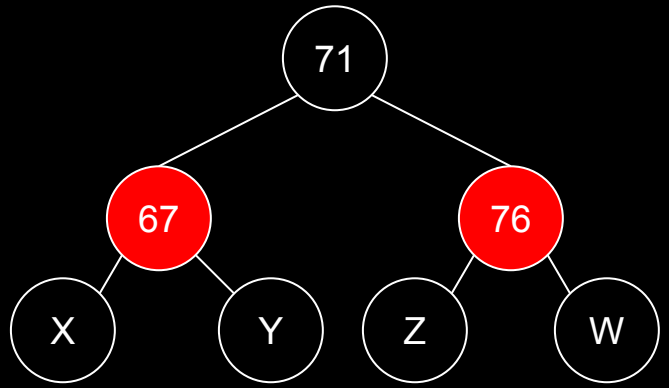
# Nodo com 3 chaves

Nodo com 3 chaves e 4 filhos.

A chave central é um nodo preto.

As chaves esquerda e direita se tornam filhos vermelhos, direito e esquerdo.



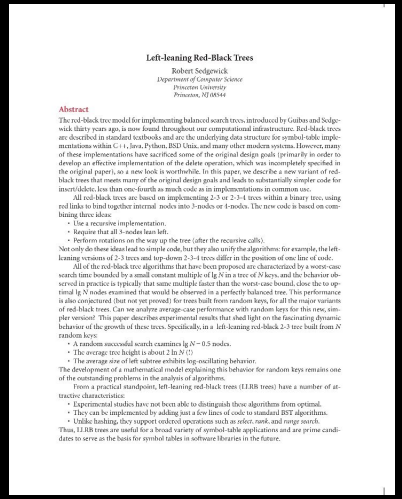
# Rotações

Note que as operações de rotação da Red-Black de aulas passadas são equivalentes as operações de merge com transplante de chaves apresentadas para uma Árvore B.

# Veja esse artigo

sedgewick.io/wp-content/themes/sedgewick/papers/2008LLRB.pdf

Left-leaning Red-Black Trees, Robert Sedgewick (2008).



**Left-leaning Red-Black Trees**

Robert Sedgewick
*Department of Computer Science*
*Princeton University*
*Princeton, NJ 08544*

**Abstract**

The red-black tree model for implementing balanced search trees, introduced by Guibas and Sedgewick thirty years ago, is now found throughout our computational infrastructure. Red-black trees are described in standard textbooks and are the underlying data structure for symbol-table implementations within C++, Java, Python, BSD Unix, and many other modern systems. However, many of these implementations have sacrificed some of the original design goals (primarily in order to develop an effective implementation of the delete operation, which was incompletely specified in the original paper), so a new look is worthwhile. In this paper, we describe a new variant of red-black trees that meets many of the original design goals and leads to substantially simpler code for insert/delete, less than one-fourth as much code as in implementations in common use.

All red-black trees are based on implementing 2-3 or 2-3-4 trees within a binary tree, using red links to bind together internal nodes into 3-nodes or 4-nodes. The new code is based on combining three ideas:

- Use a recursive implementation.
- Require that all 3-nodes lean left.
- Perform rotations on the way up the tree (after the recursive calls).

Not only do these ideas lead to simple code, but they also unify the algorithms: for example, the left-leaning versions of 2-3 trees and top-down 2-3-4 trees differ in the position of one line of code.

All of the red-black tree algorithms that have been proposed are characterized by a worst-case search time bounded by a small constant multiple of $\lg N$ in a tree of $N$ keys, and the behavior observed in practice is typically that same multiple faster than the worst-case bound, close the to optimal $\lg N$ nodes examined that would be observed in a perfectly balanced tree. This performance is also conjectured (but not yet proved) for trees built from random keys, for all the major variants of red-black trees. Can we analyze average-case performance with random keys for this new, simpler version? This paper describes experimental results that shed light on the fascinating dynamic behavior of the growth of these trees. Specifically, in a left-leaning red-black 2-3 tree built from $N$ random keys:

- A random successful search examines $\lg N - 0.5$ nodes.
- The average tree height is about $2 \ln N$ (!)
- The average size of left subtree exhibits log-oscillating behavior.

The development of a mathematical model explaining this behavior for random keys remains one of the outstanding problems in the analysis of algorithms.

From a practical standpoint, left-leaning red-black trees (LLRB trees) have a number of attractive characteristics:

- Experimental studies have not been able to distinguish these algorithms from optimal.
- They can be implemented by adding just a few lines of code to standard BST algorithms.
- Unlike hashing, they support ordered operations such as *select*, *rank*, and *range search*.

Thus, LLRB trees are useful for a broad variety of symbol-table applications and are prime candidates to serve as the basis for symbol tables in software libraries in the future.

# Árvores B+

Existem diversas outras variantes de Árvores B.

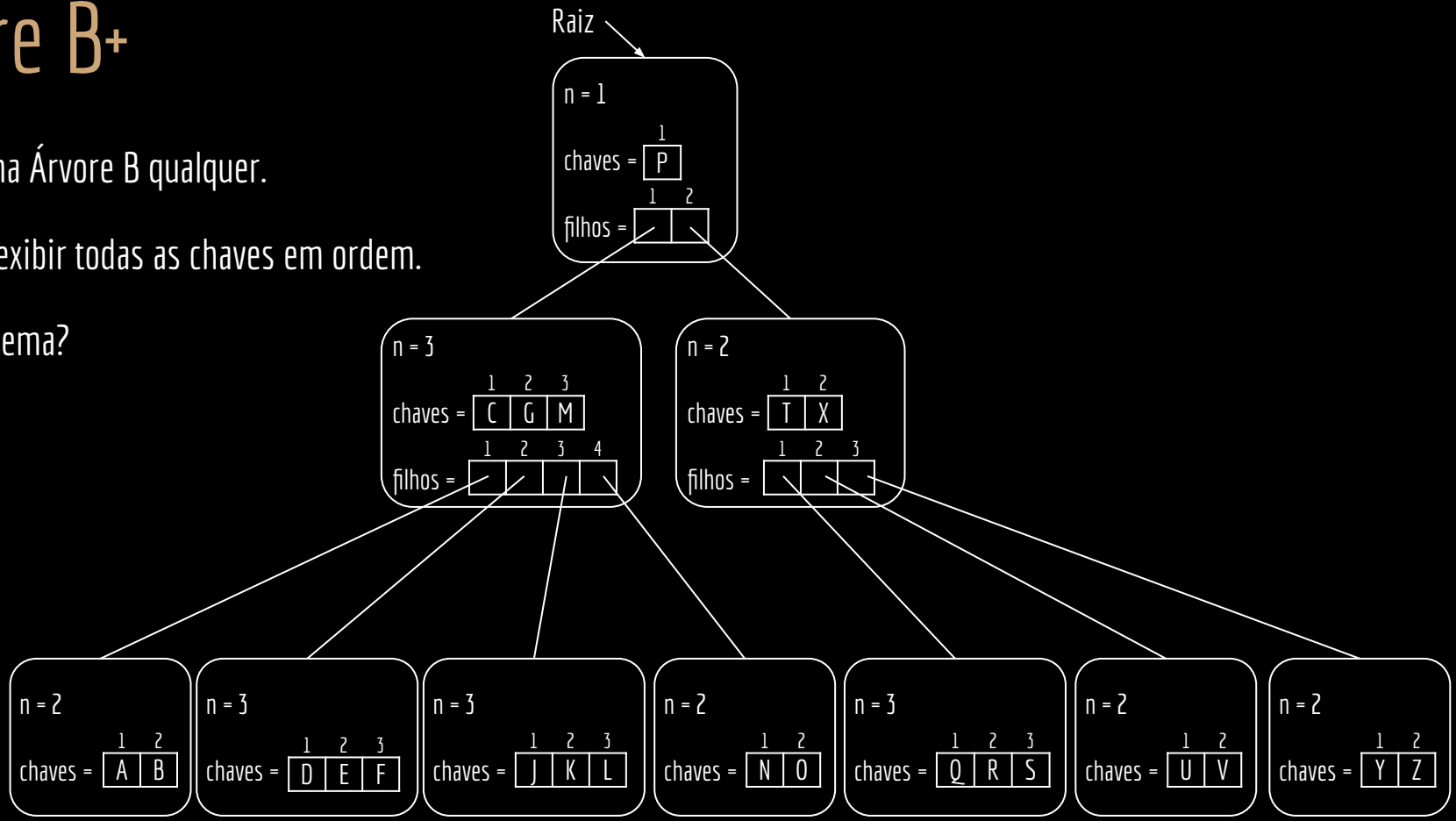
- Árvores B*.
  - Mantém cada nodo pelo menos ⅔ cheio.
- Árvores B+.
  - Mantém as chaves nas folhas.
- Árvores B+ de prefixo.
  - Similar a B+, mas mantém as menores chaves possíveis nos nodos internos.

# Árvore B+

Suponha uma Árvore B qualquer.

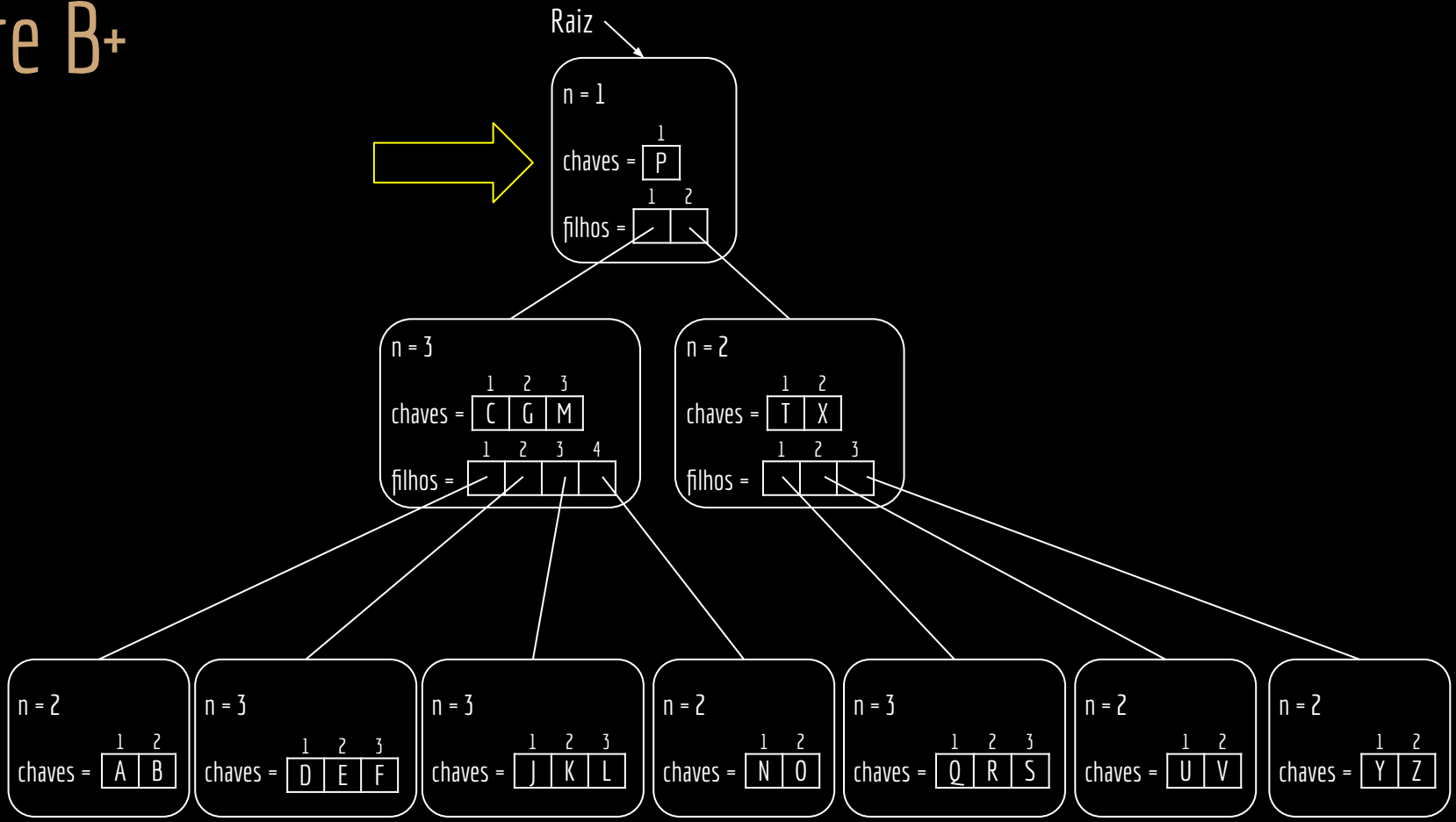
Desejamos exibir todas as chaves em ordem.

Qual o problema?



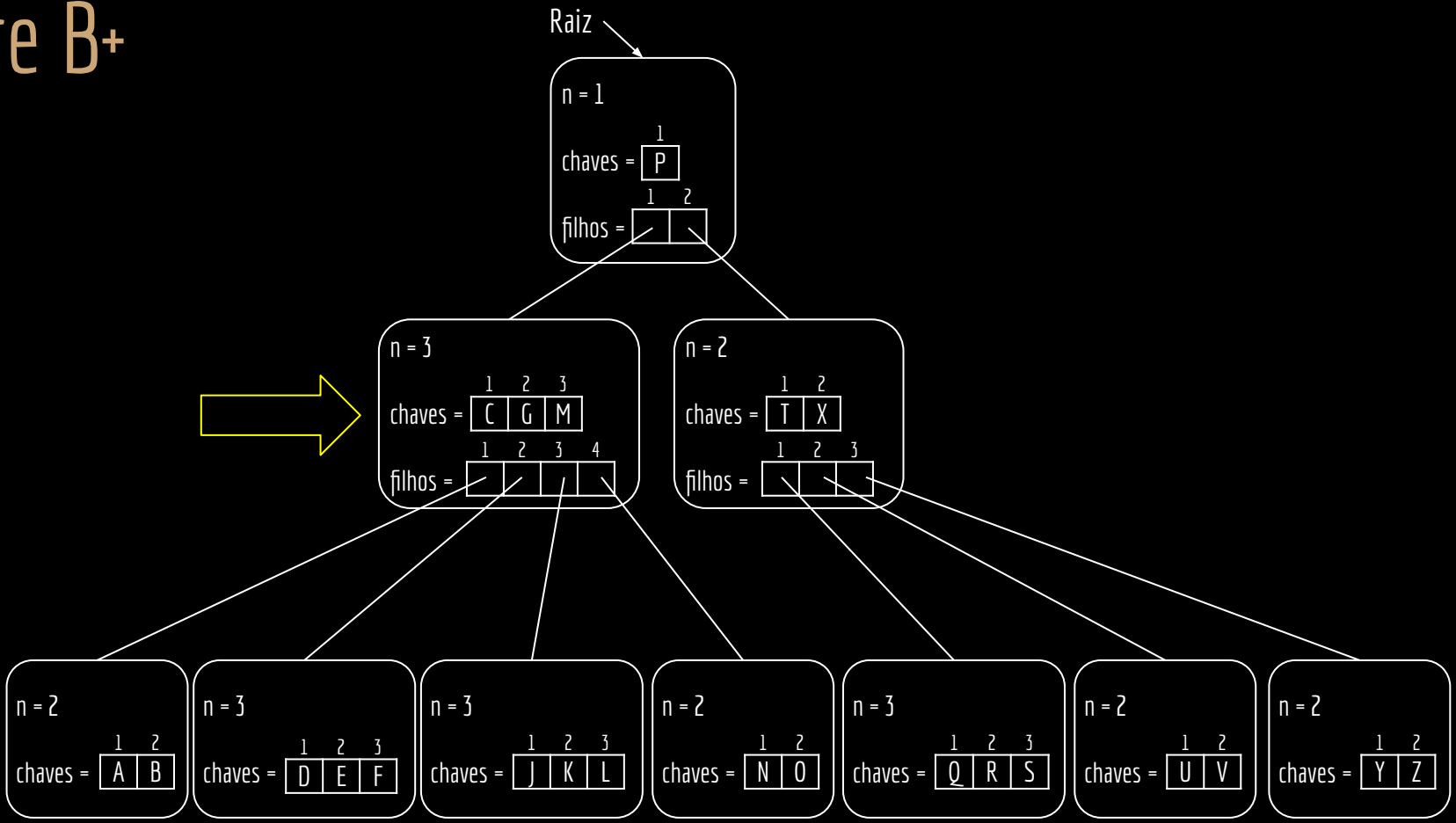
# Árvore B+

Saída:



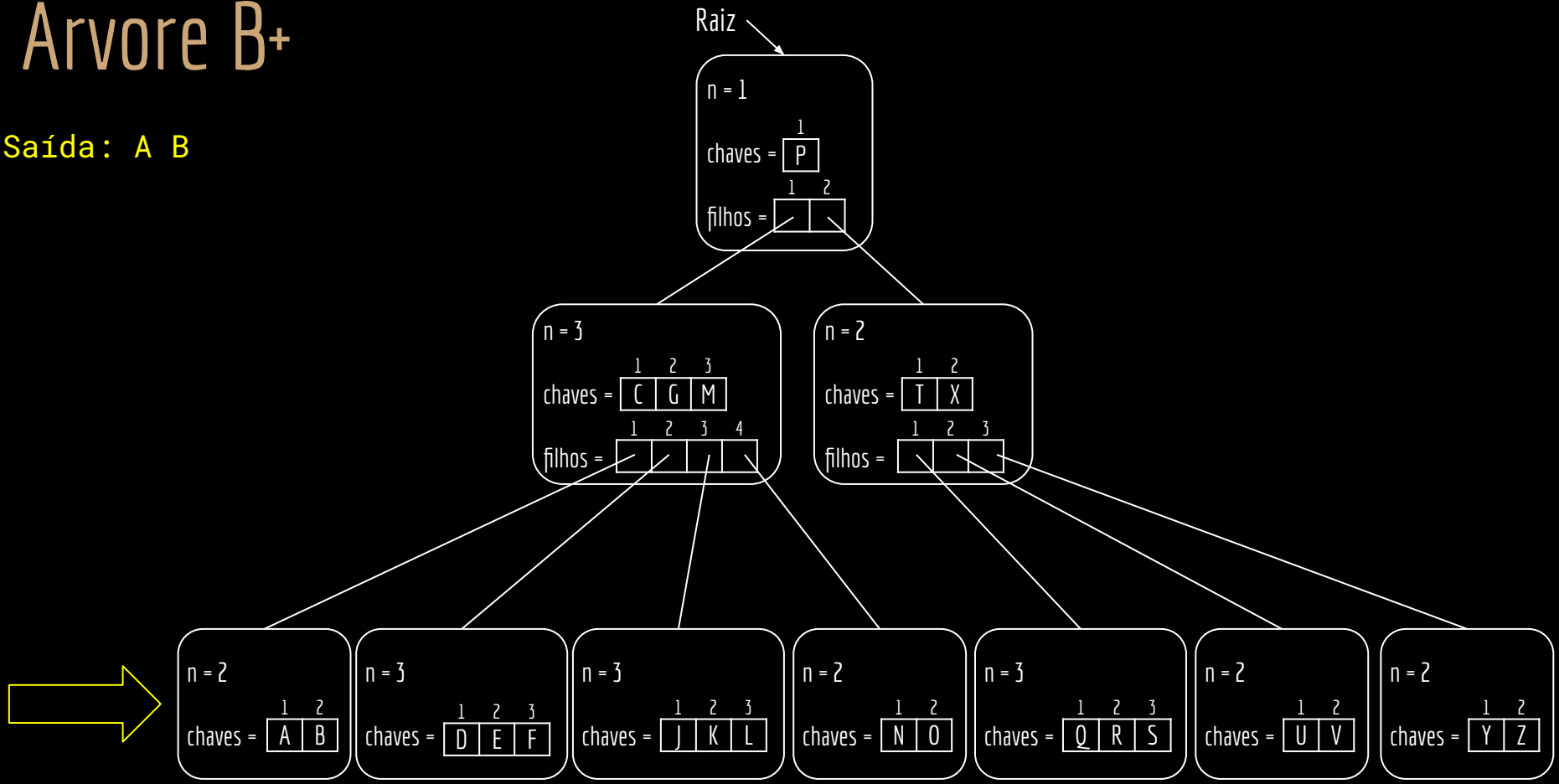
# Árvore B+

Saída:



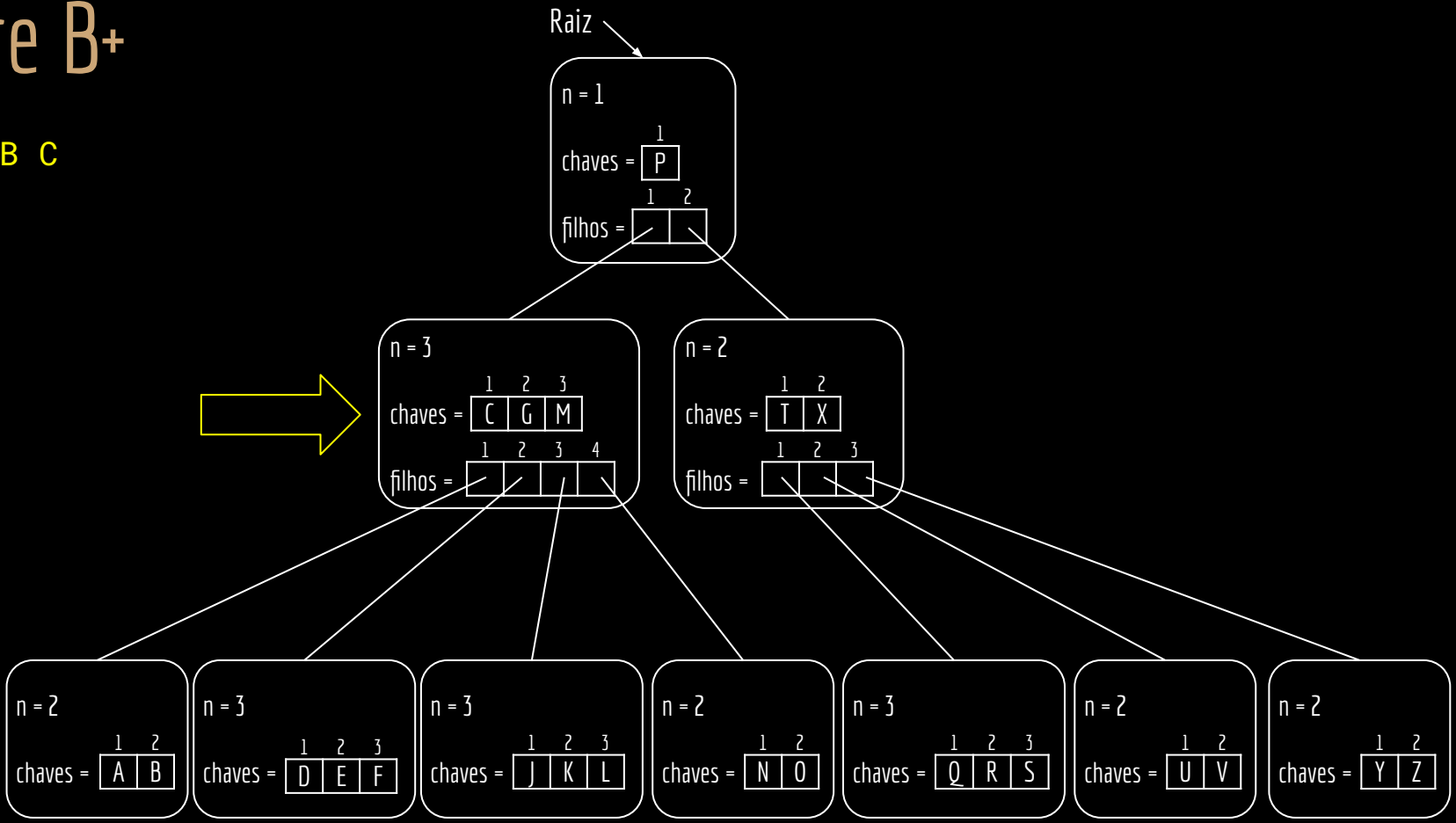
# Árvore B+

Saída: A B



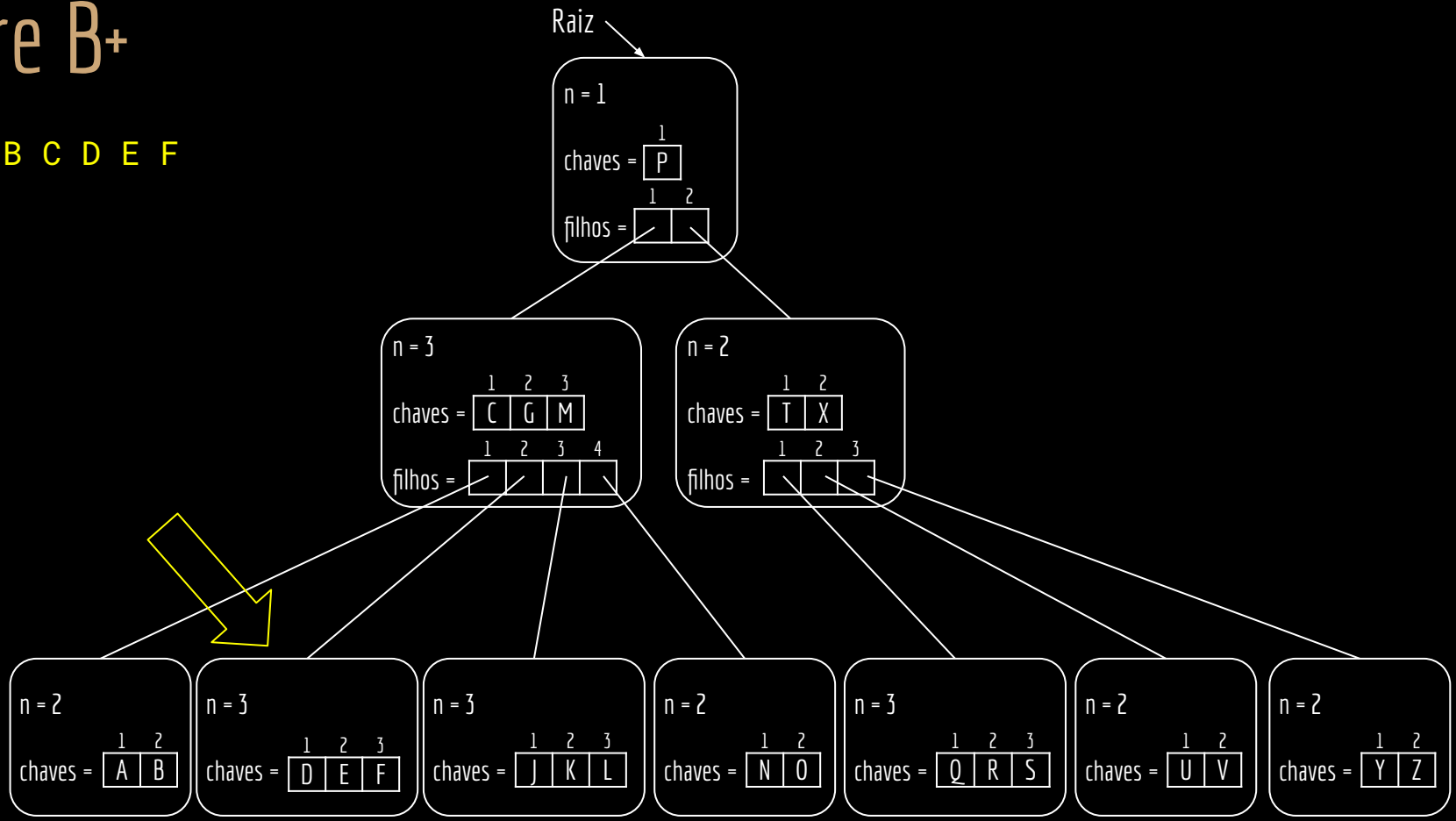
# Árvore B+

Saída: A B C



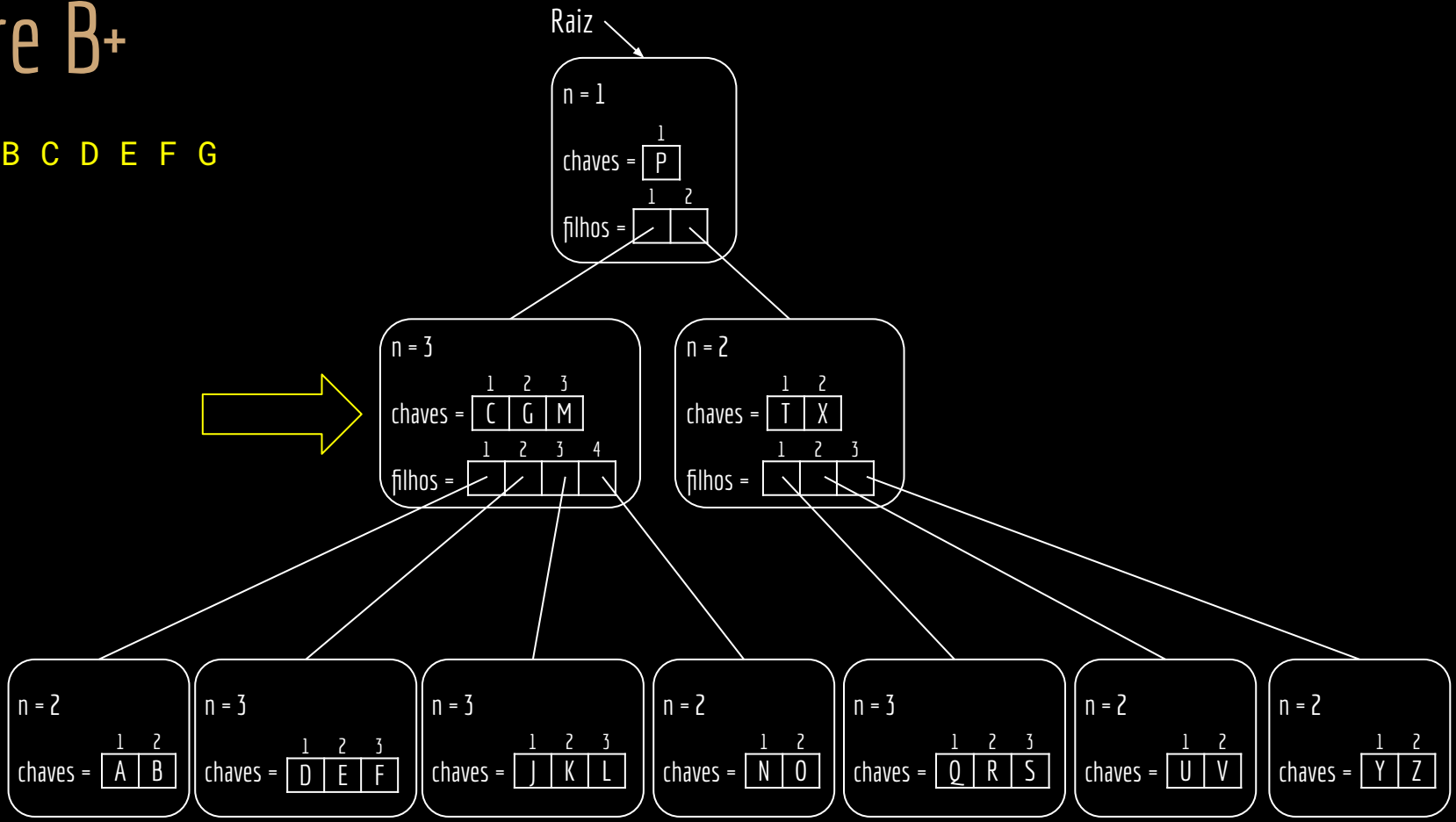
# Árvore B+

Saída: A B C D E F



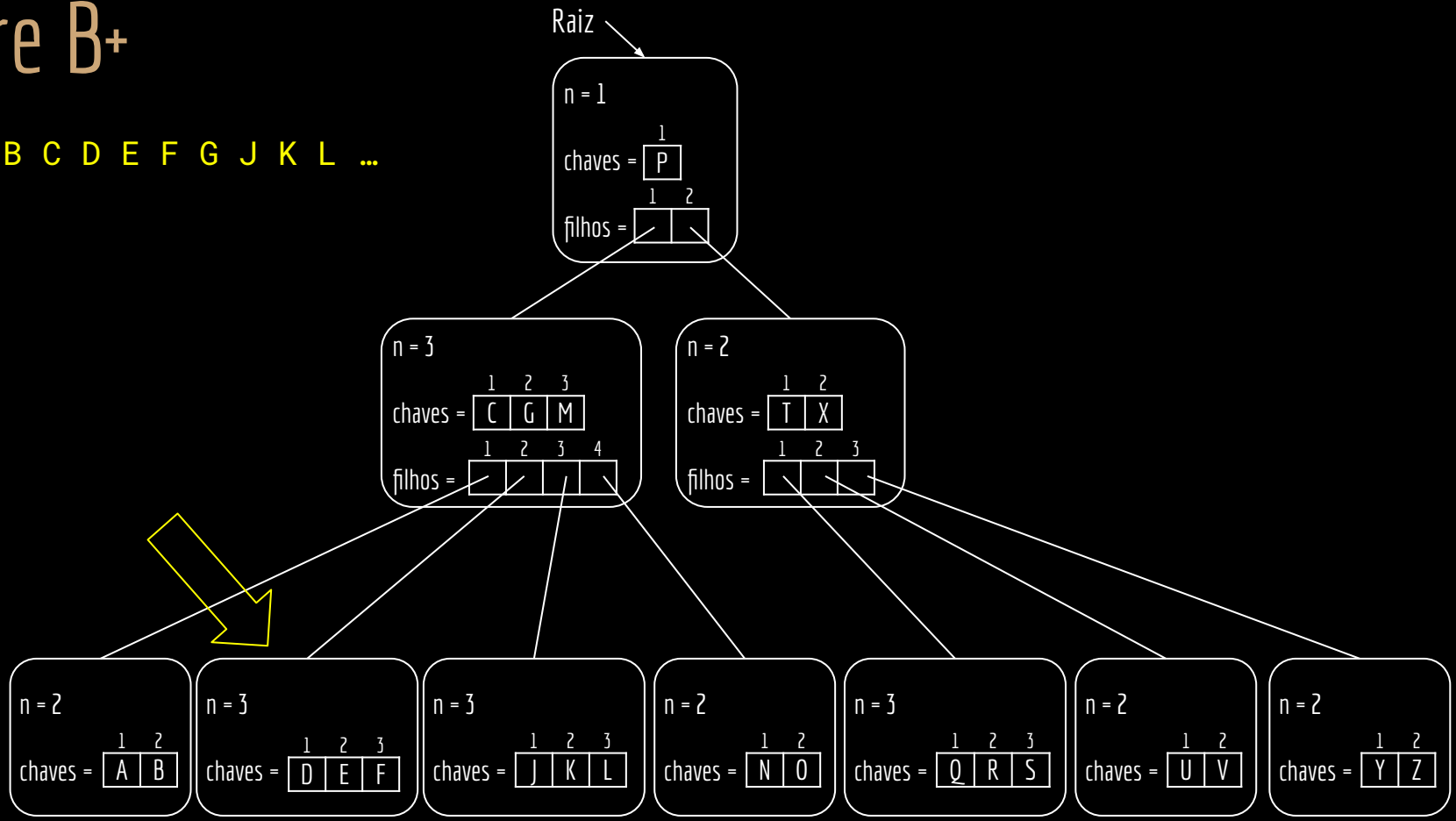
# Árvore B+

Saída: A B C D E F G



# Árvore B+

Saída: A B C D E F G J K L …



# Árvore B+

Em uma Árvore B+ os dados são mantidos somente nas folhas.

Os nodos internos índices para descobrirmos “para que lado ir”.

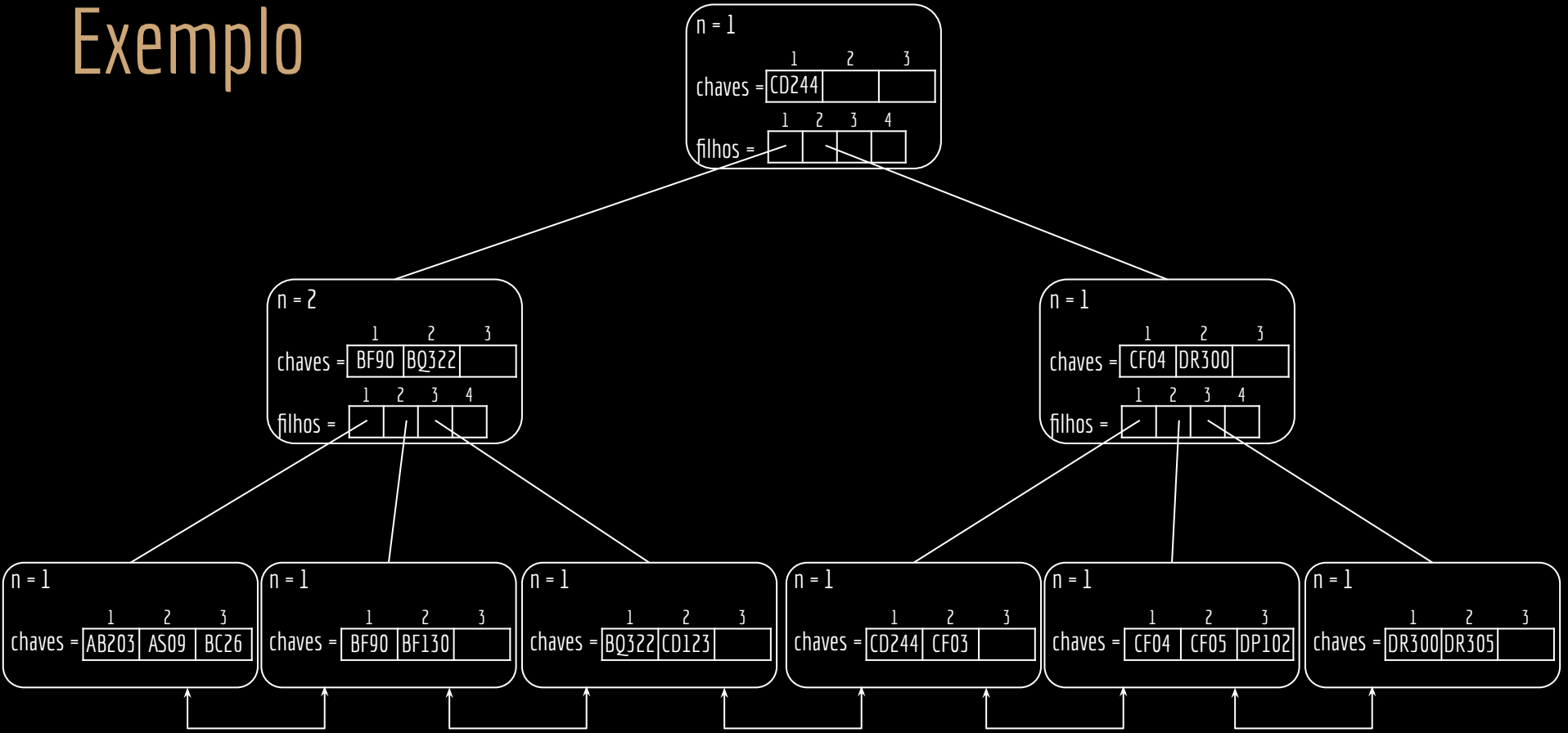
- Conjunto de índices.

As folhas possuem ponteiros para suas irmãs diretas.

- Possibilitar navegar em ordem como em uma lista encadeada.

Exemplo de Drozdek (2016).

# Exemplo

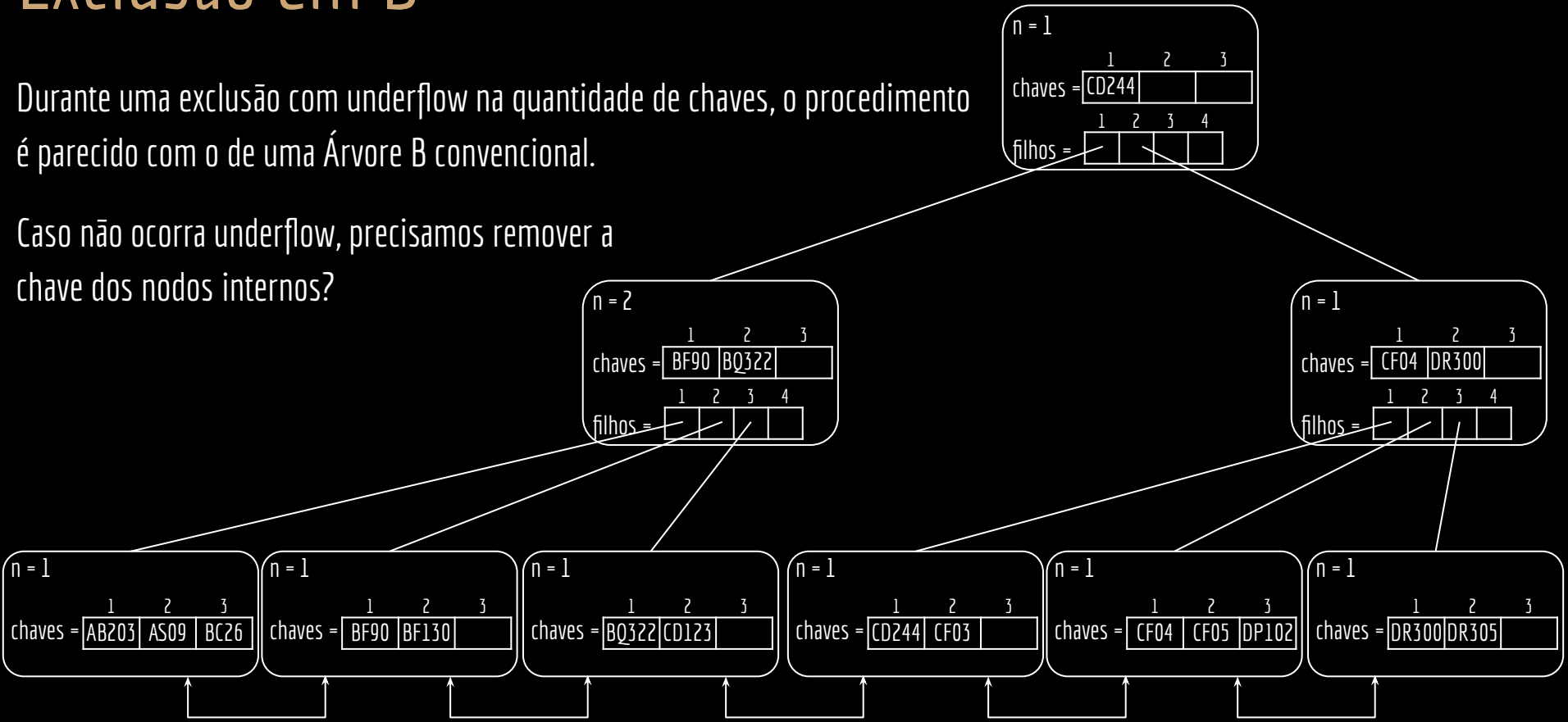



Lista encadeada de folhas.

# Exclusão em B+

Durante uma exclusão com underflow na quantidade de chaves, o procedimento é parecido com o de uma Árvore B convencional.

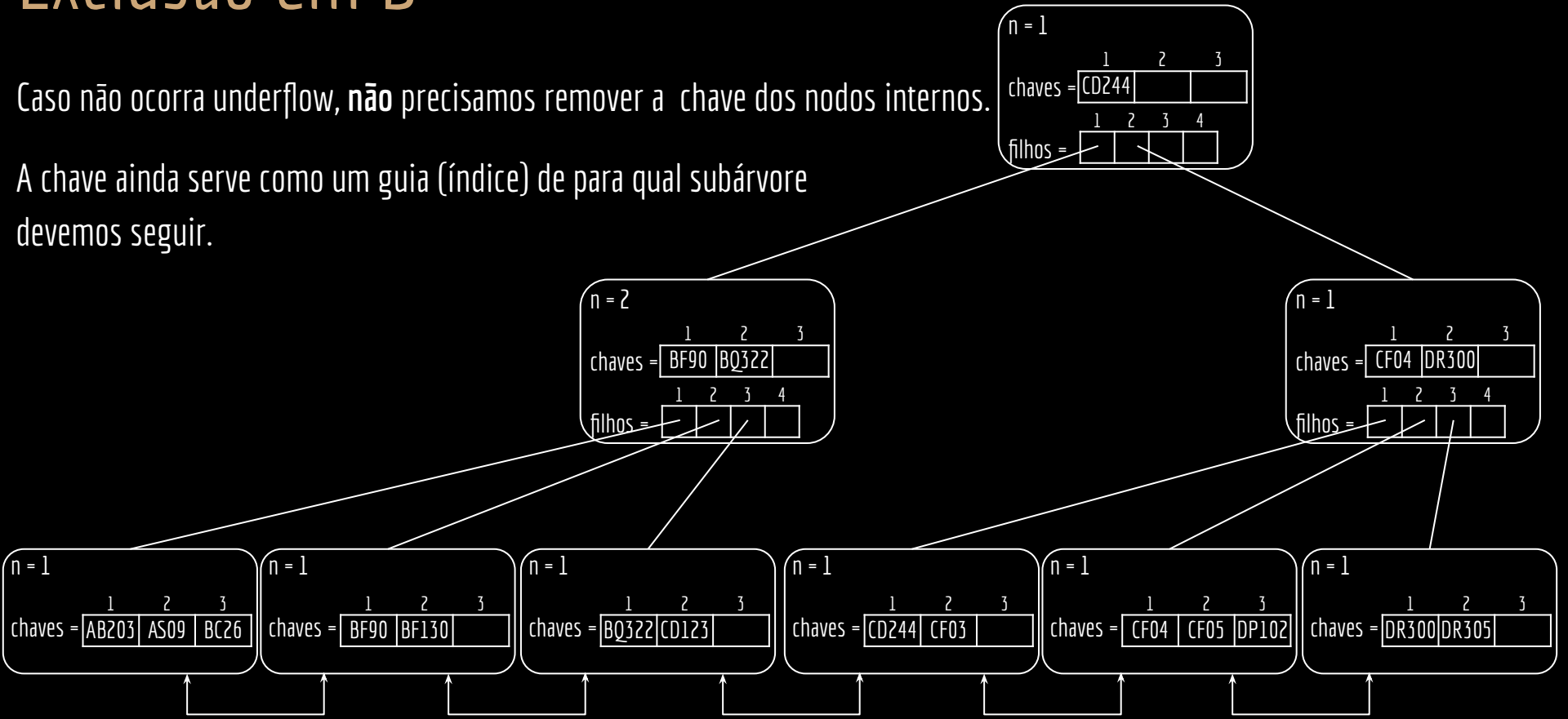
Caso não ocorra underflow, precisamos remover a chave dos nodos internos?



# Exclusão em B+

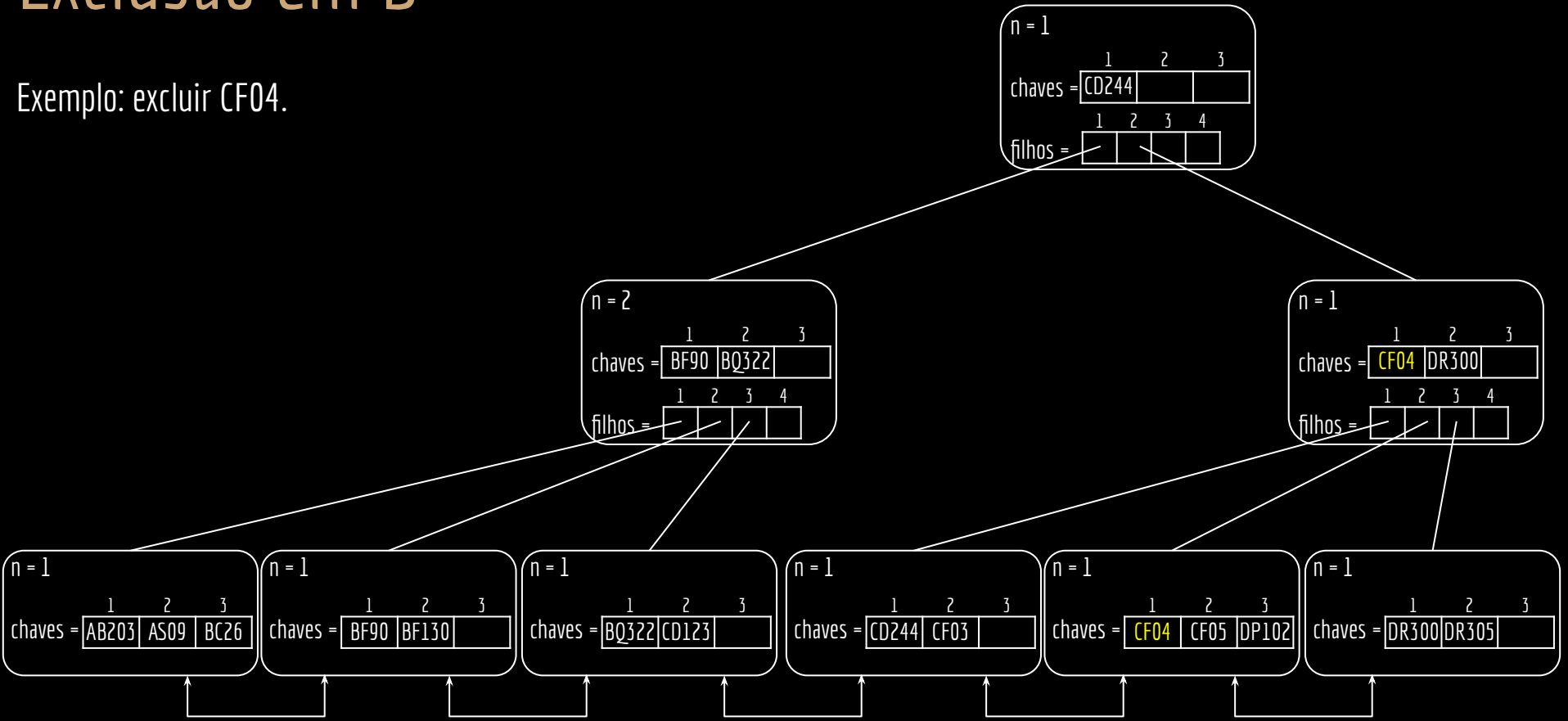
Caso não ocorra underflow, **não** precisamos remover a  chave dos nodos internos.

A chave ainda serve como um guia (índice) de para qual subárvore devemos seguir.



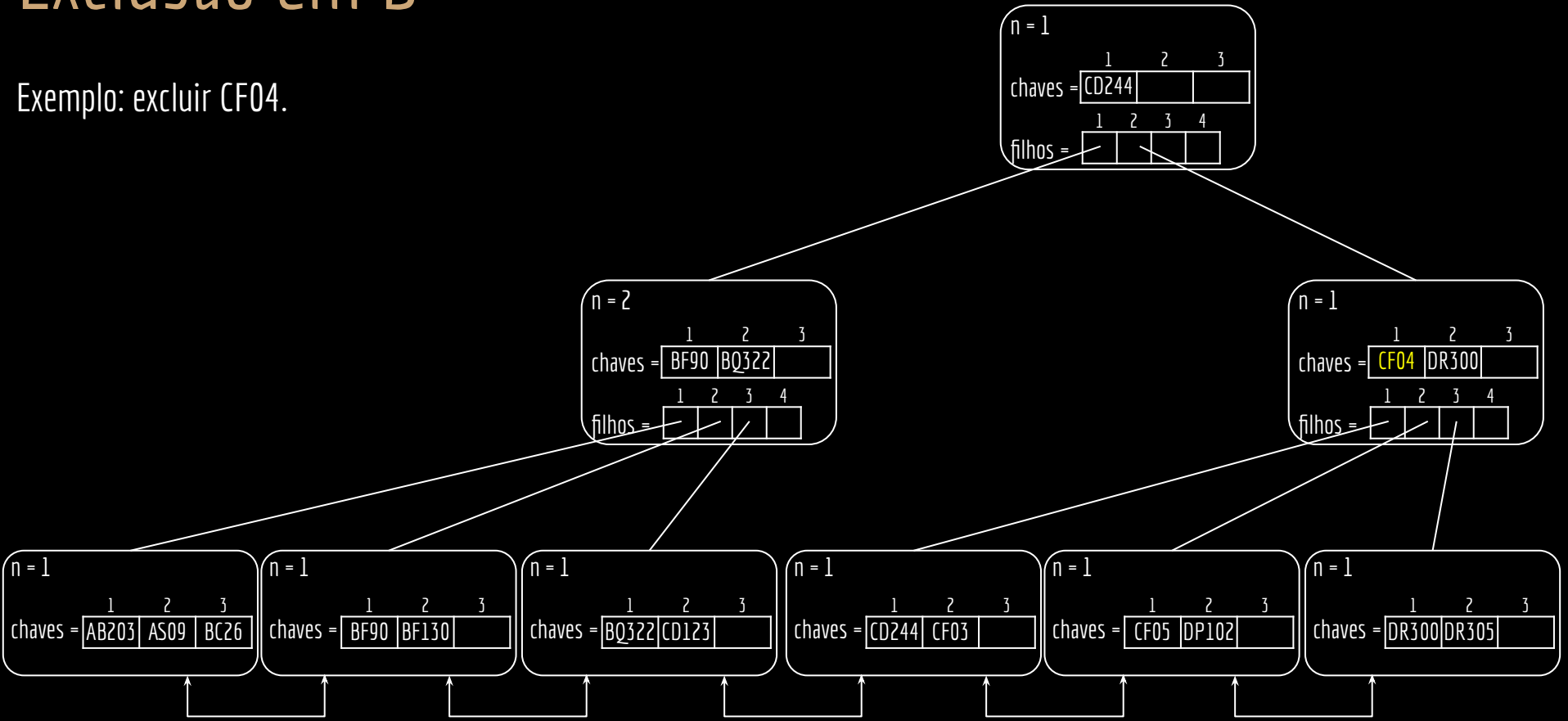
# Exclusão em B+

Exemplo: excluir CF04.



# Exclusão em B+

Exemplo: excluir CF04.

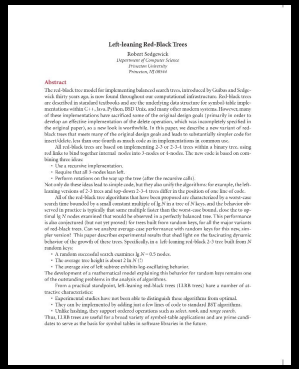


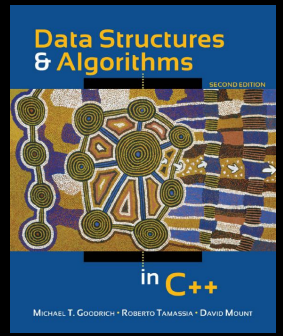
# Exercícios

1. Escreva um algoritmo que, dada uma árvore Red-Black, a transforma em 2-3-4.
2. Escreva um algoritmo que, dada uma árvore 2-3-4, a transforma em Red-Black.
3. Relacione os algoritmos de rotação à esquerda e à direita da Red-Black com os algoritmos de transplante de chave da árvore 2-3-4.
4. Em uma Árvore B+ é especialmente útil que as folhas tenham uma estrutura diferente dos nodos internos (folhas não possuem filhos, então podemos economizar memória com esse vetor). Dê ideias sobre como implementar os nodos da Árvore B+ de forma que as folhas possam ter um estrutura diferente da dos nodos internos.

# Referências

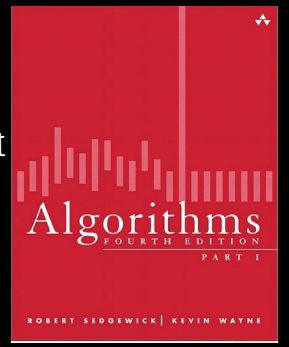

Sedgewick, R. Left-leaning Red-Black Trees, 2008.



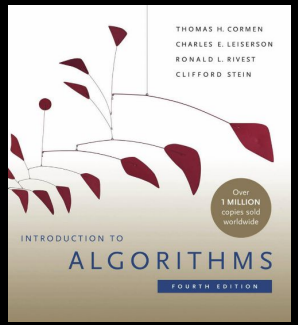

Mount, Goodrich, Tamassia. Data Structures and Algorithms in C++, 2011.

R. Sedgewick, K. Wayne. Algorithms Part I. 4a ed. 2011.

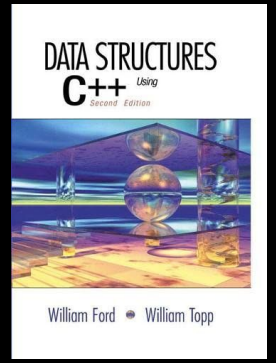



Estrutura de Dados e Algoritmos em C++. A. Drozdek. 4a ed. 2016.


T. Cormen, C. Leiserson, R. Rivest, C. Stein. Algoritmos: Teoria e Prática. 4a ed. 2022.




Topp, Ford. Data Structures with C++ Using STL. 2002.



# Licença