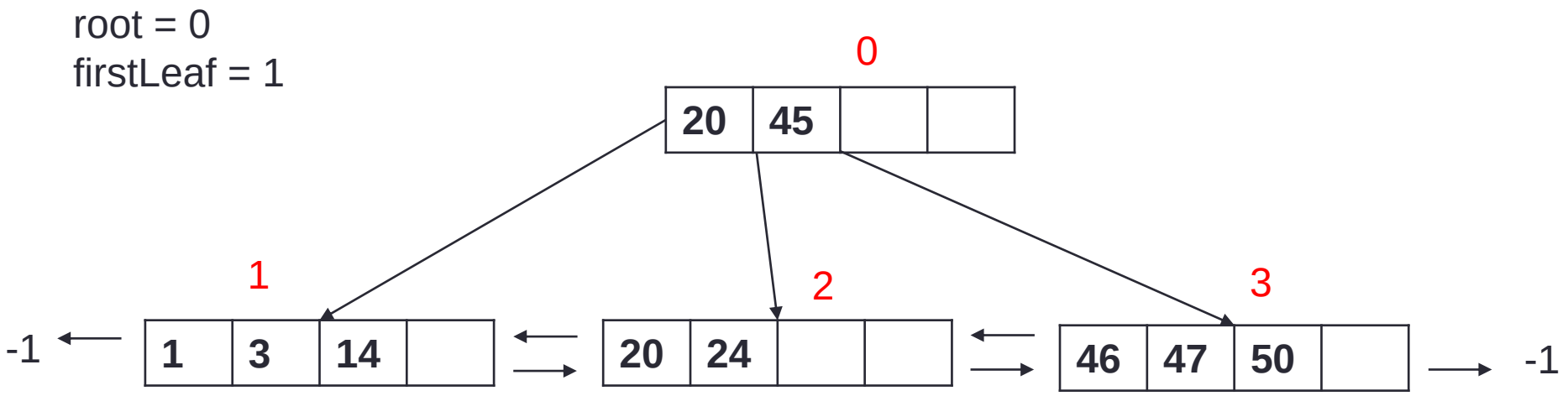
TRABALHO PRÁTICO 1 IBD 2023/2

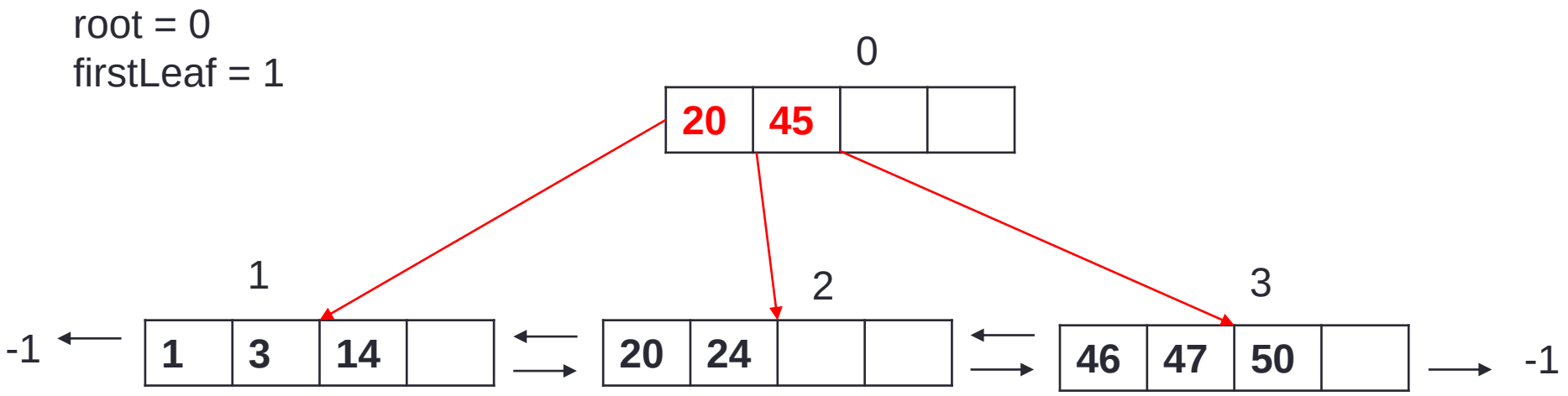
Sergio Mergen

Introdução

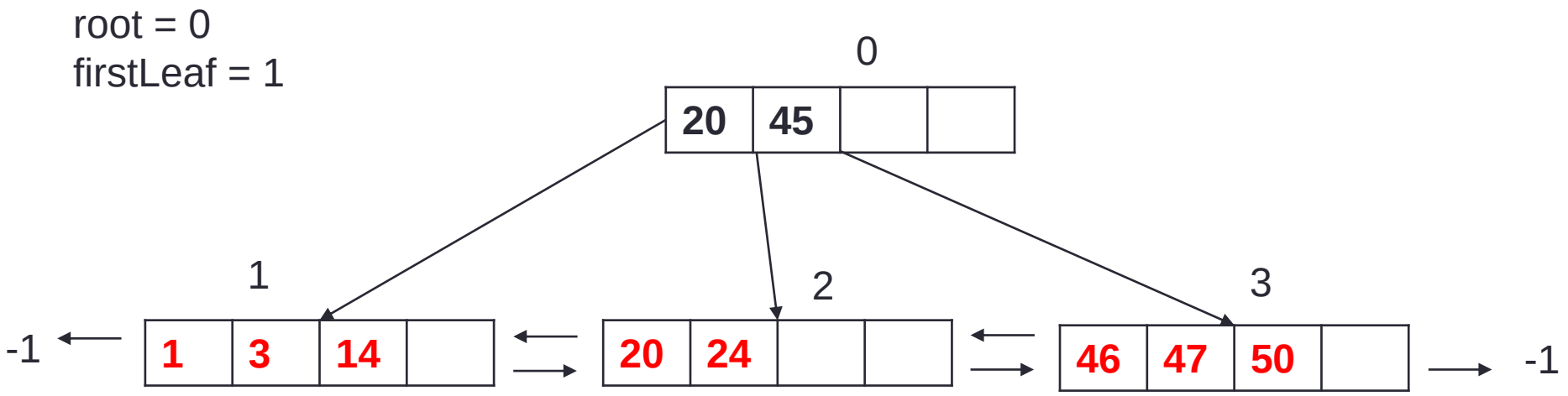
- Os dados de cada tabela são mantidos por uma árvore B+
  - Classe BPlusTreeFile.java
- Dois tipos de nós
  - Nós internos: nós de navegação
  - Nós folha: onde ficam os registros
- Para fins de balanceamento, é importante que cada nó possua pelo menos metade das entradas
  - Nós com menos da metade das entradas são considerados deficientes



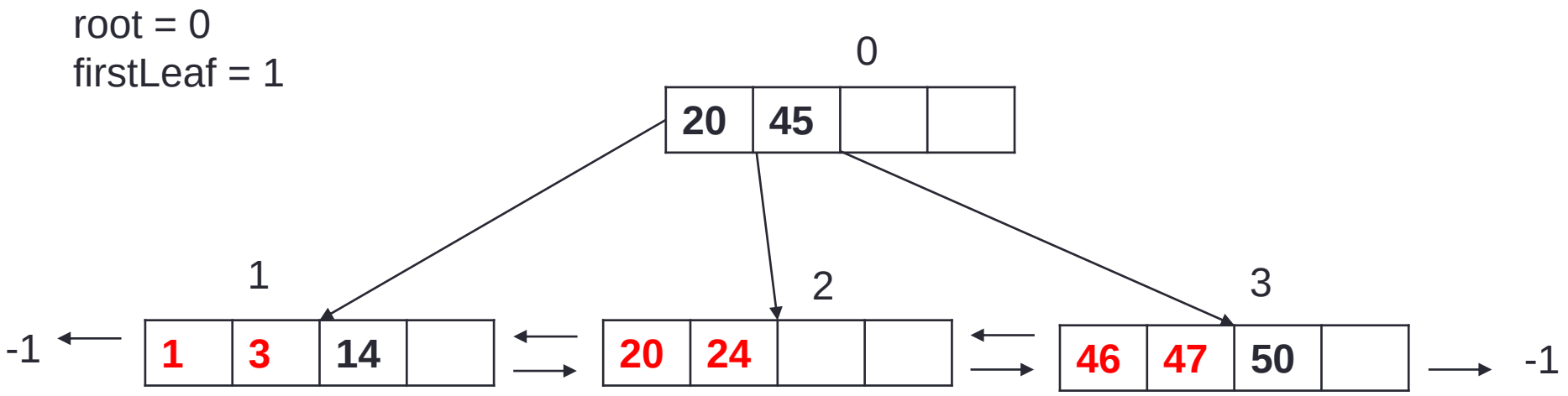
- Todos nós possuem um ID
- Os nós internos possuem chave + ponteiro para o nível abaixo
- Os nós folha possuem chave + registro
  - Os registros não estão sendo exibidos na figura acima
- Os nós folha devem possuir pelo menos metade das chaves
  - Caso contrário, eles são considerados deficientes



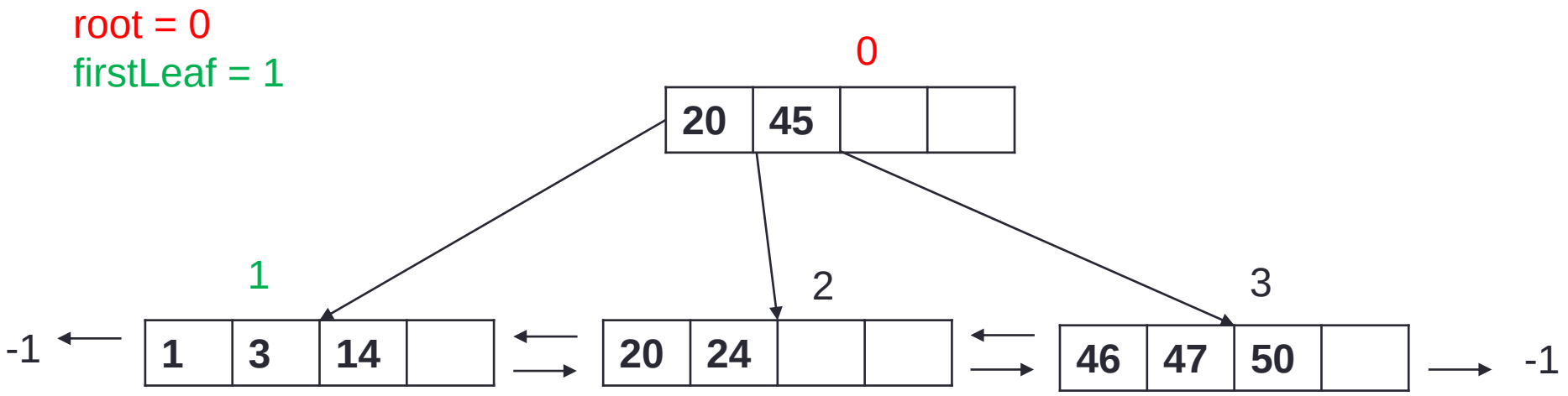
- Todos nós possuem um ID
- Os nós internos possuem chave + ponteiro para o nível abaixo
- Os nós folha possuem chave + registro
  - Os registros não estão sendo exibidos na figura acima
- Os nós folha devem possuir pelo menos metade das chaves
  - Caso contrário, eles são considerados deficientes



- Todos nós possuem um ID
- Os nós internos possuem chave + ponteiro para o nível abaixo
- Os nós folha possuem chave + registro
  - Os registros não estão sendo exibidos na figura acima
- Os nós folha devem possuir pelo menos metade das chaves
  - Caso contrário, eles são considerados deficientes



- Todos nós possuem um ID
- Os nós internos possuem chave + ponteiro para o nível abaixo
- Os nós folha possuem chave + registro
  - Os registros não estão sendo exibidos na figura acima
- Os nós folha devem possuir pelo menos metade das chaves
  - Caso contrário, eles são considerados deficientes

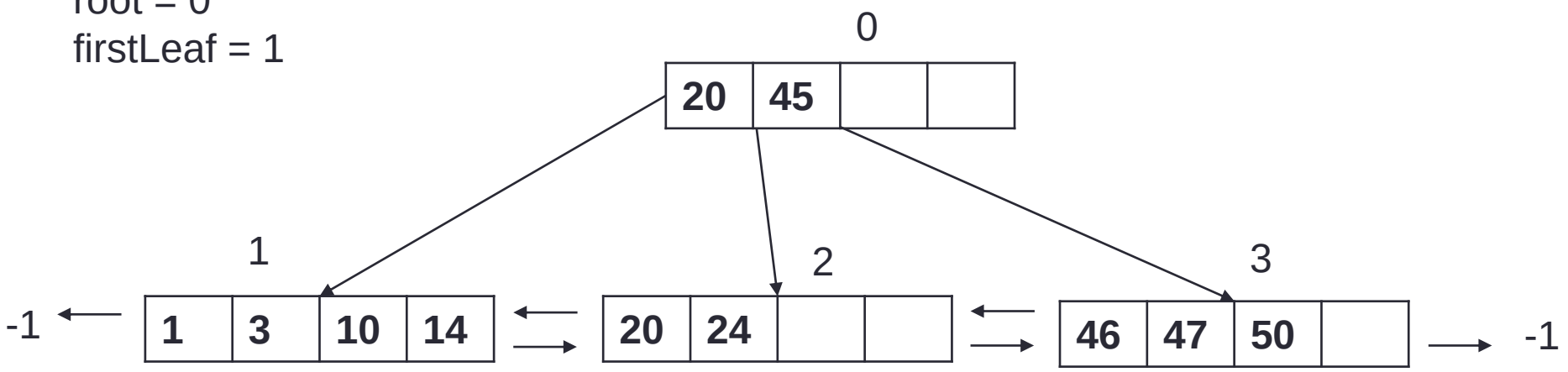


- Duas informações são entradas para a árvore
  - Nó raiz (root): usado para buscas
  - Primeiro nó folha (firstLeaf): usado para acesso sequencial

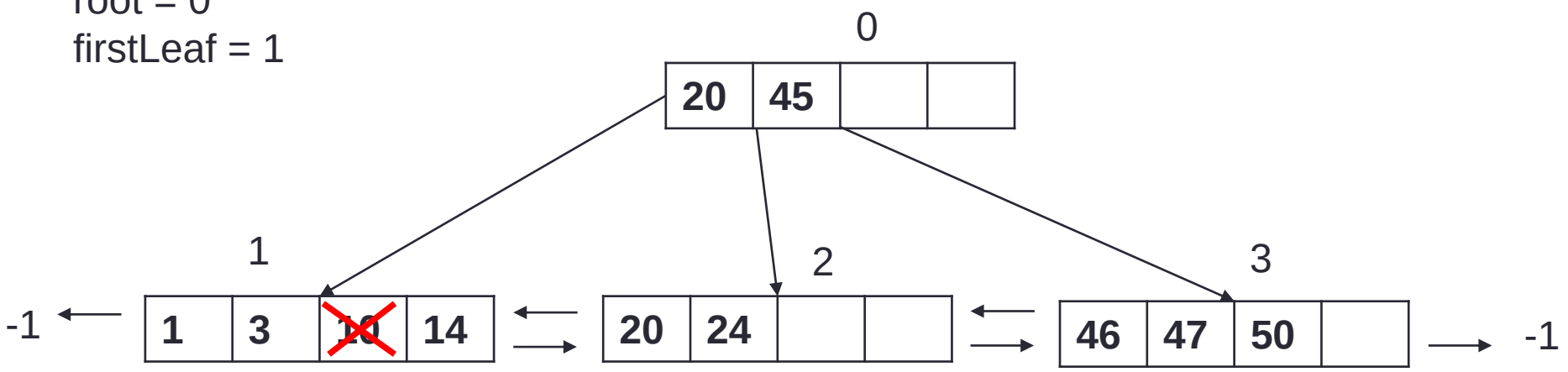
Balanceamento

- Estratégias adotadas
  - Durante a inserção, o nó cheio sofre split
  - Durante a remoção, nenhuma ação de balanceamento é realizada.
    - Ou seja, o nó continua deficiente

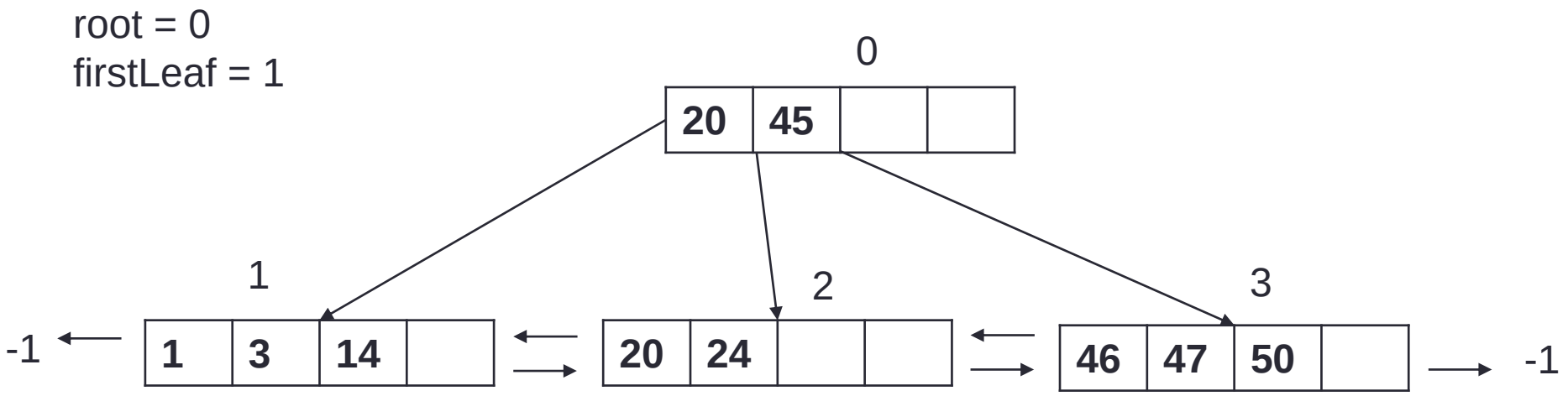
root = 0
firstLeaf = 1



root = 0
firstLeaf = 1



Remoção do 10

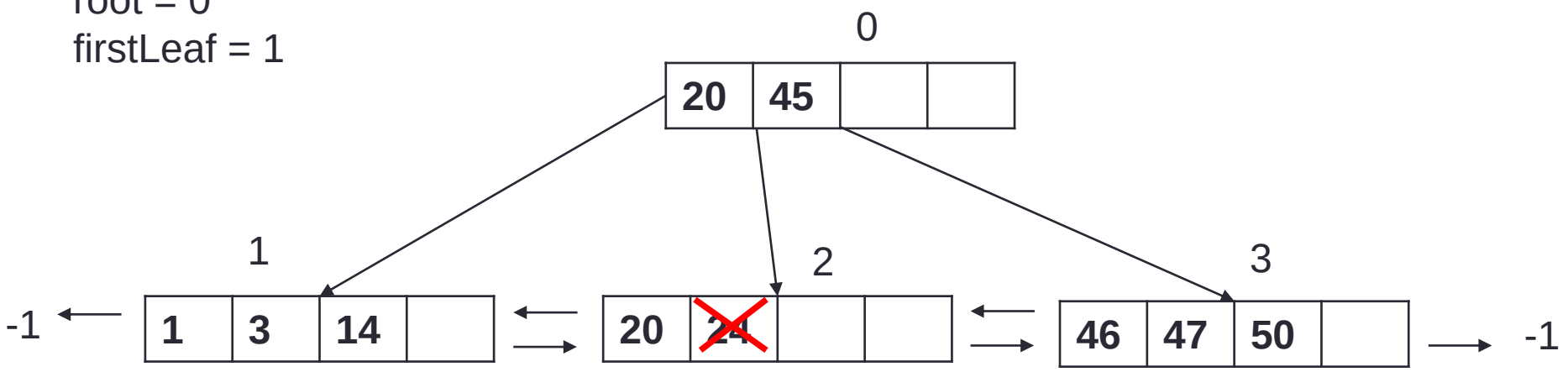


Remoção do 10

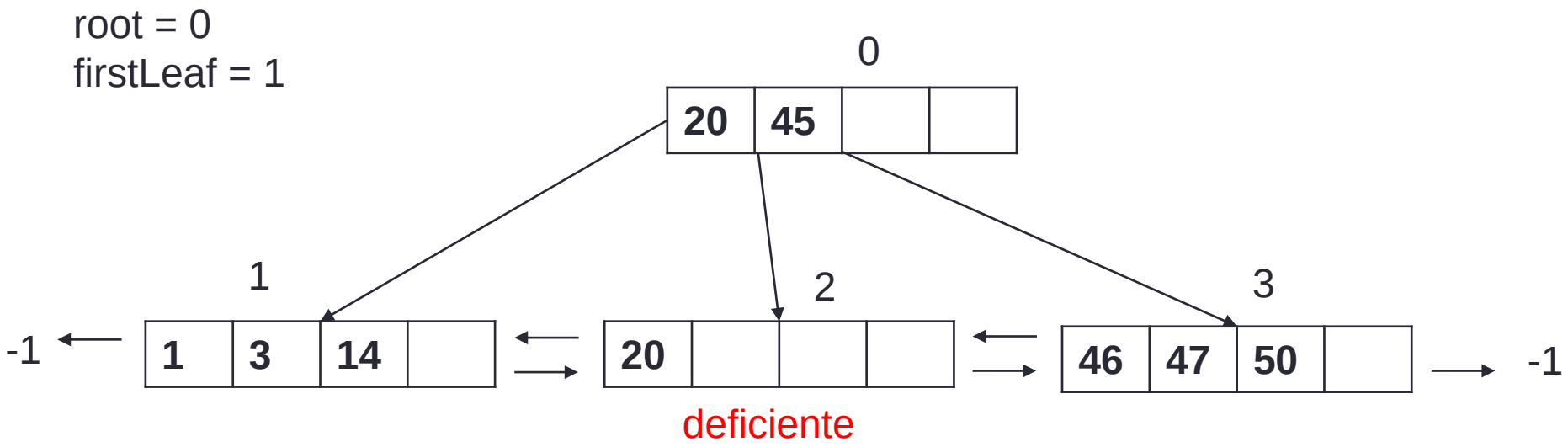
O nó folha continua com o mínimo de valores.

Ou seja, nada precisa ser feito

root = 0
firstLeaf = 1

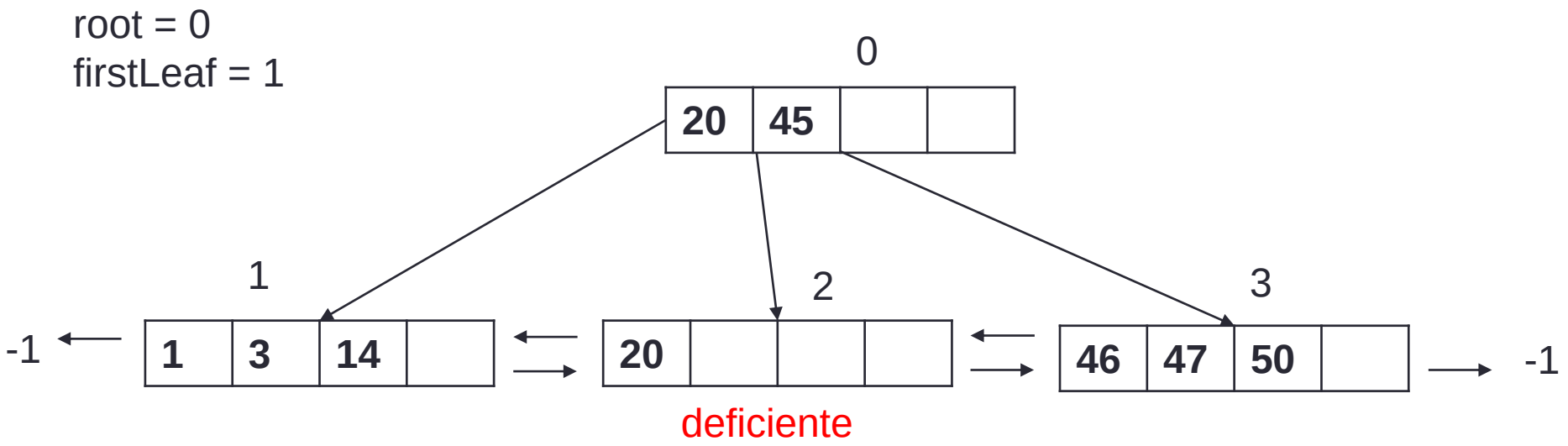


Remoção do 24



Remoção do 24

O nó 2 ficou deficiente.



Remoção do 24

O nó 2 ficou deficiente.

Com o tempo, os nós deficientes levam a um maior número de nós.
Com isso:

- o arquivo ocupa mais espaço
- a árvore fica mais alta (a busca é mais custosa)
- os dados ficam mais espalhados (pior aproveitamento do cache)

Trabalho Prático

- O objetivo é implementar as regras de balanceamento quando o nó folha ficar deficiente após uma remoção
  - **Observação:** em alguns casos, os nós internos também podem ficar deficientes
    - Mas, neste trabalho, iremos nos concentrar apenas no nível folha

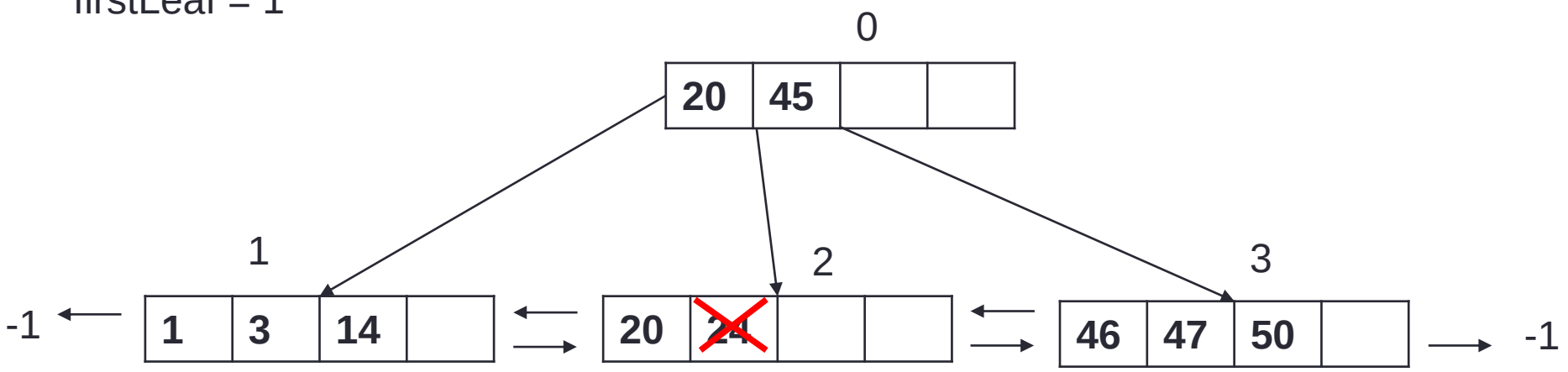
- O nome da classe deve ser BPlusTreeFileXXX,
  - Onde xxx é o nome do aluno
  - O pacote da classe deve ser ibd.index.btree

Situações a serem tratadas

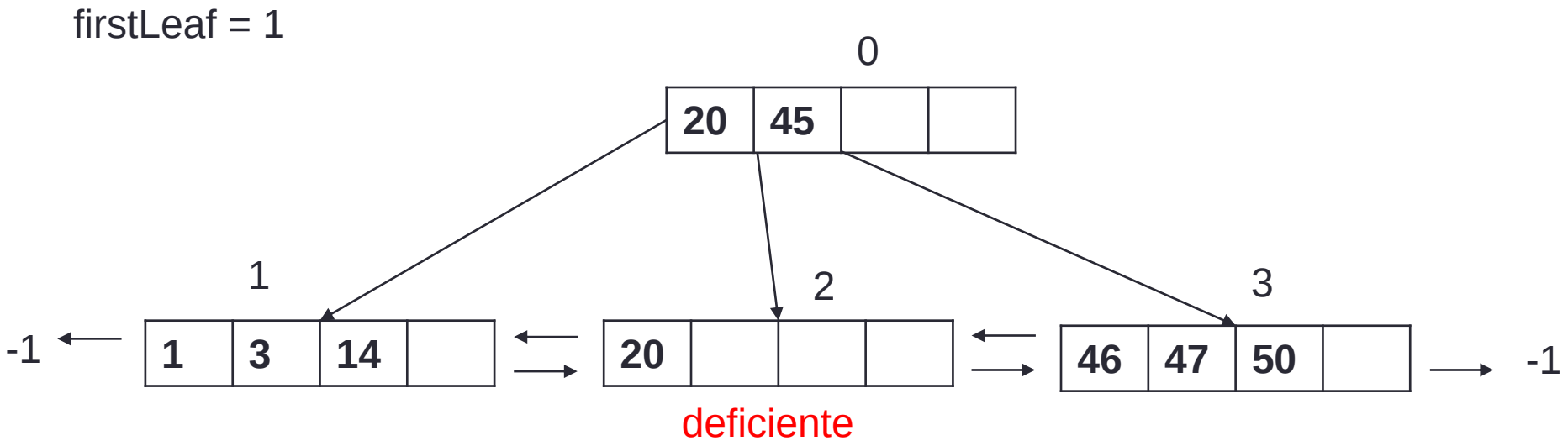
- Quatro situações de resolução devem ser tratadas
  - O nó deficiente empresta um valor do nó da esquerda
  - O nó deficiente empresta um valor do nó da direita
  - O nó deficiente é mesclado com o nó da esquerda
  - O nó deficiente é mesclado com o nó da direita

SITUAÇÃO 1: EMPRESTANDO DO NÓ DA ESQUERDA

firstLeaf = 1



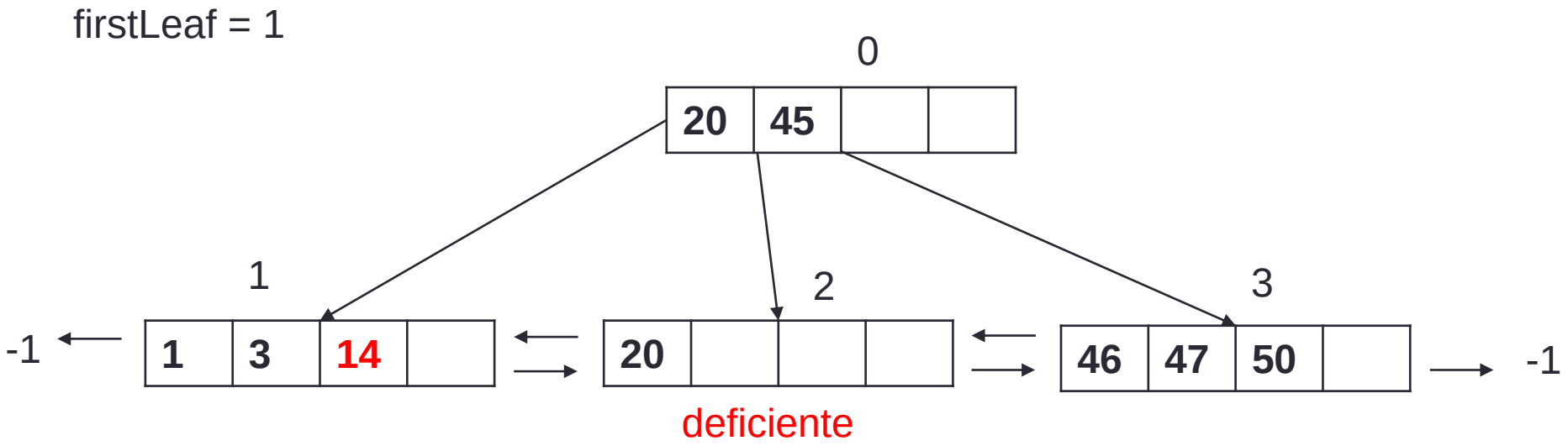
Remoção do 24



Remoção do 24

O nó 2 ficou deficiente.

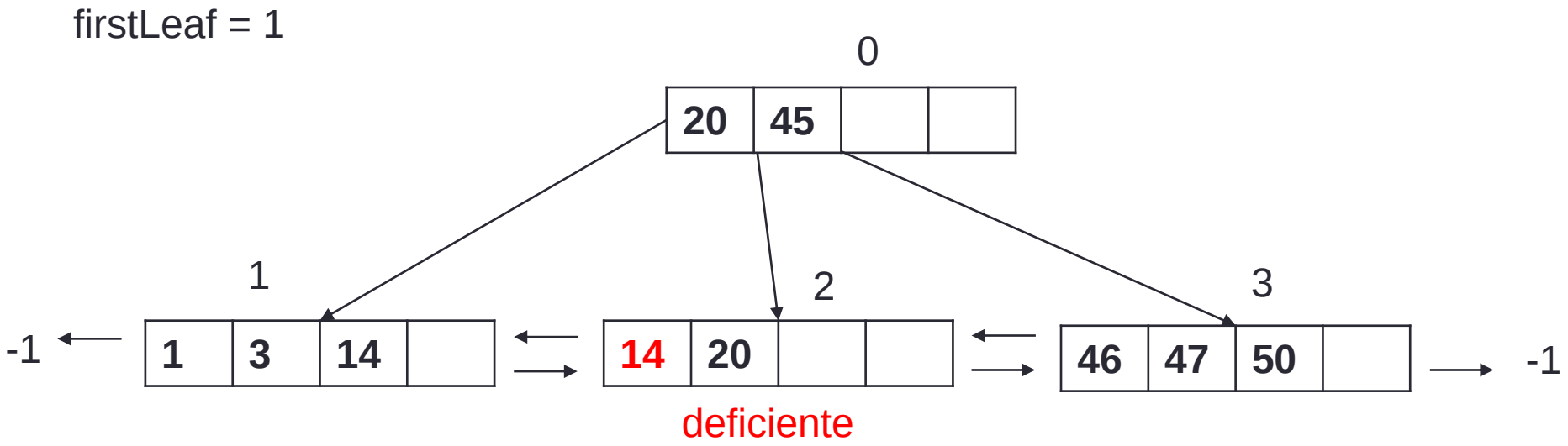
O nó da esquerda (nó 1) pode emprestar



Remoção do 24

O último registro do nó da esquerda (14):

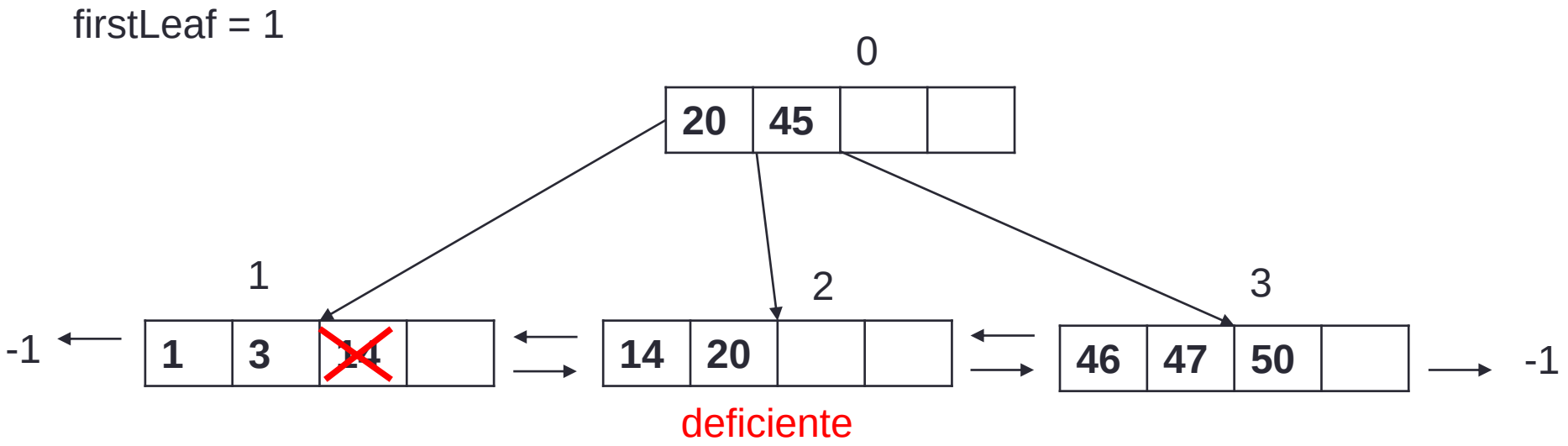
1. é copiado para a primeira posição do nó deficiente
2. é removido do nó da esquerda
3. tem sua chave copiada para a entrada correspondente no nó pai



Remoção do 24

O último registro do nó da esquerda (14):

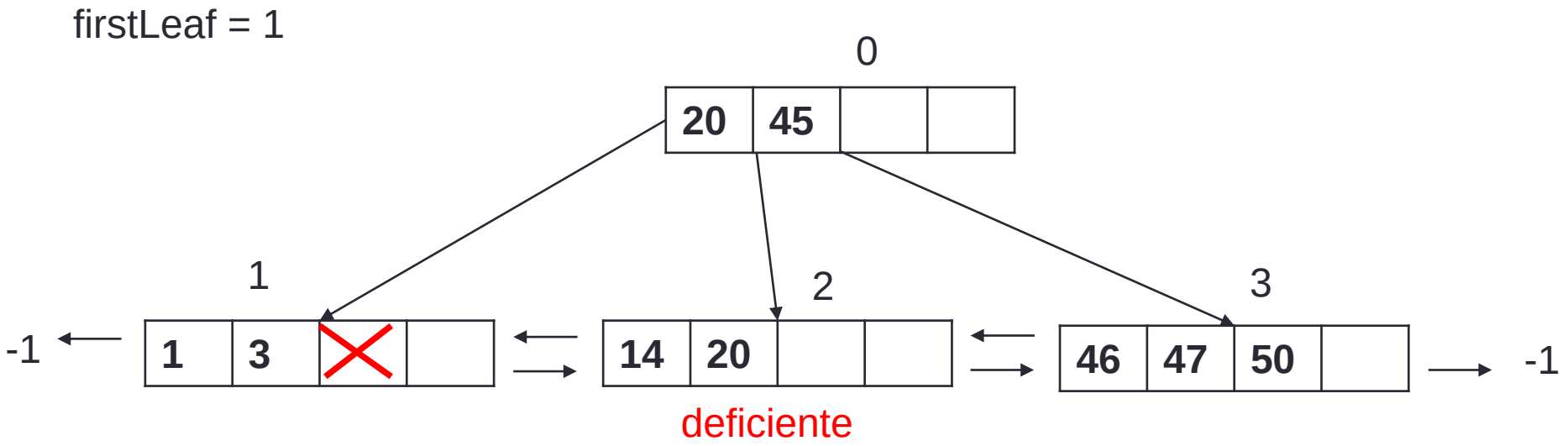
1. é copiado para a primeira posição do nó deficiente
2. é removido do nó da esquerda
3. tem sua chave copiada para a entrada correspondente no nó pai



Remoção do 24

O último registro do nó da esquerda (14):

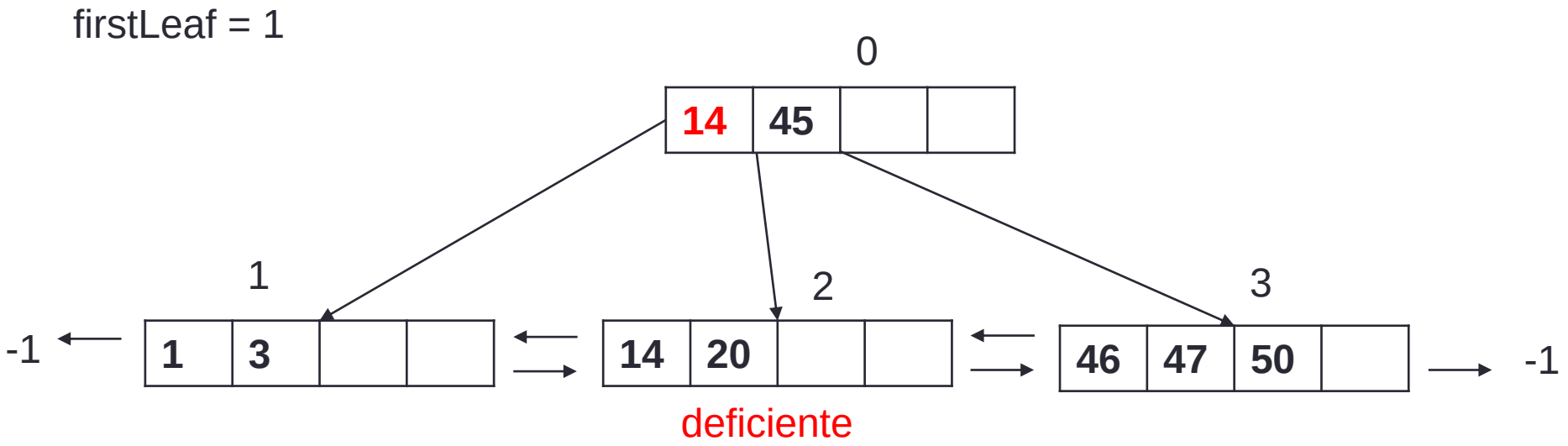
1. é copiado para a primeira posição do nó deficiente
2. é removido do nó da esquerda
3. tem sua chave copiada para a entrada correspondente no nó pai



Remoção do 24

O último registro do nó da esquerda (14):

1. é copiado para a primeira posição do nó deficiente
2. é removido do nó da esquerda
3. tem sua chave copiada para a entrada correspondente no nó pai

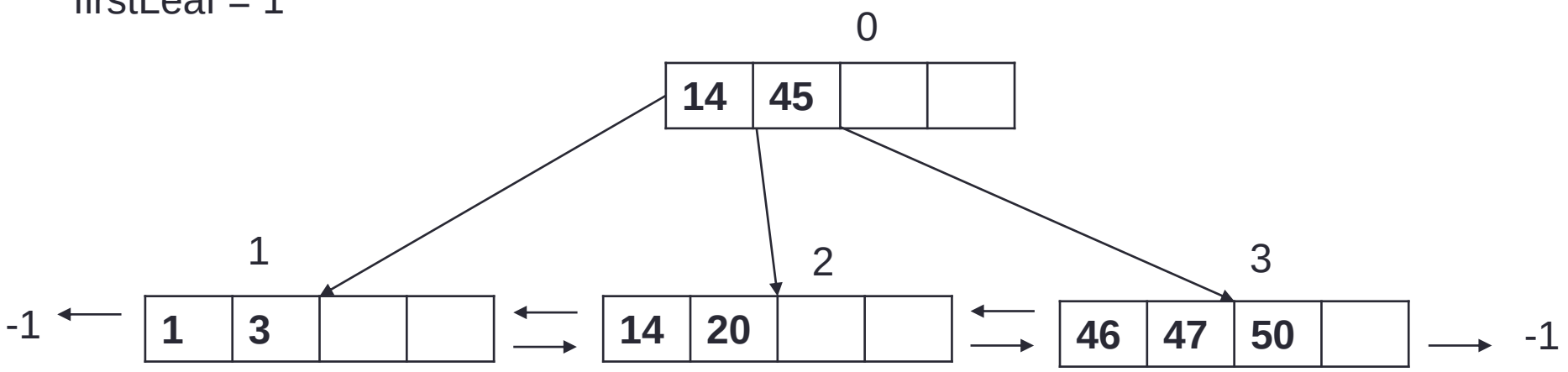


Remoção do 24

O último registro do nó da esquerda (14):

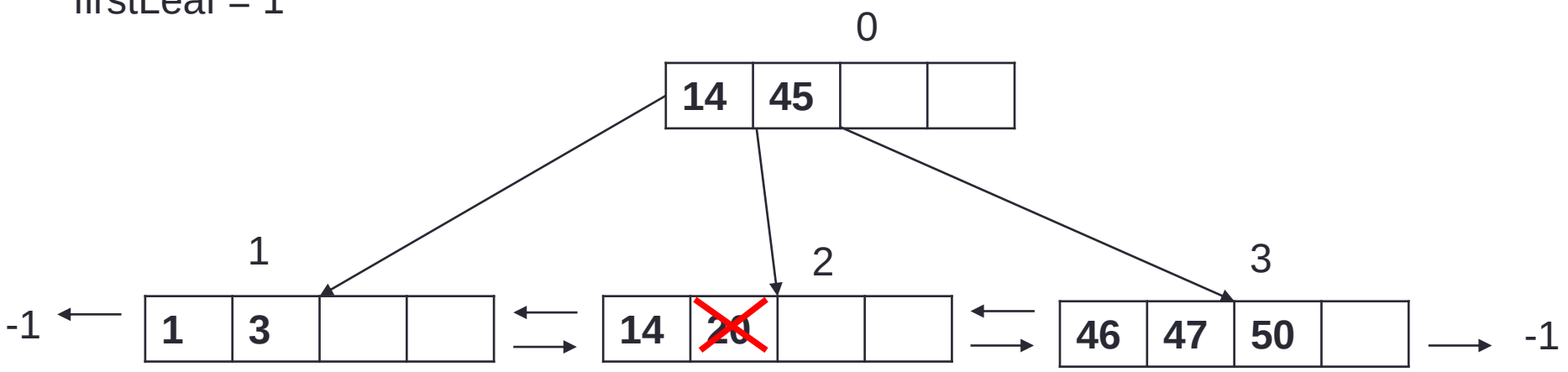
1. é copiado para a primeira posição do nó deficiente
2. é removido do nó da esquerda
3. tem sua chave copiada para a entrada correspondente no nó pai

firstLeaf = 1

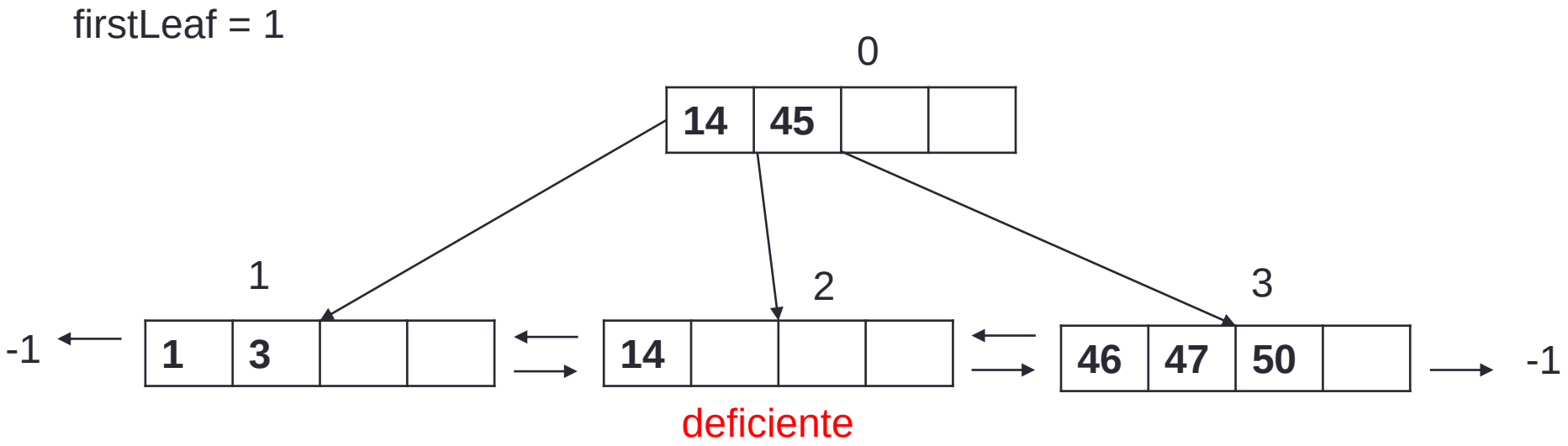


SITUAÇÃO 2: EMPRESTANDO DO NÓ DA DIREITA

firstLeaf = 1



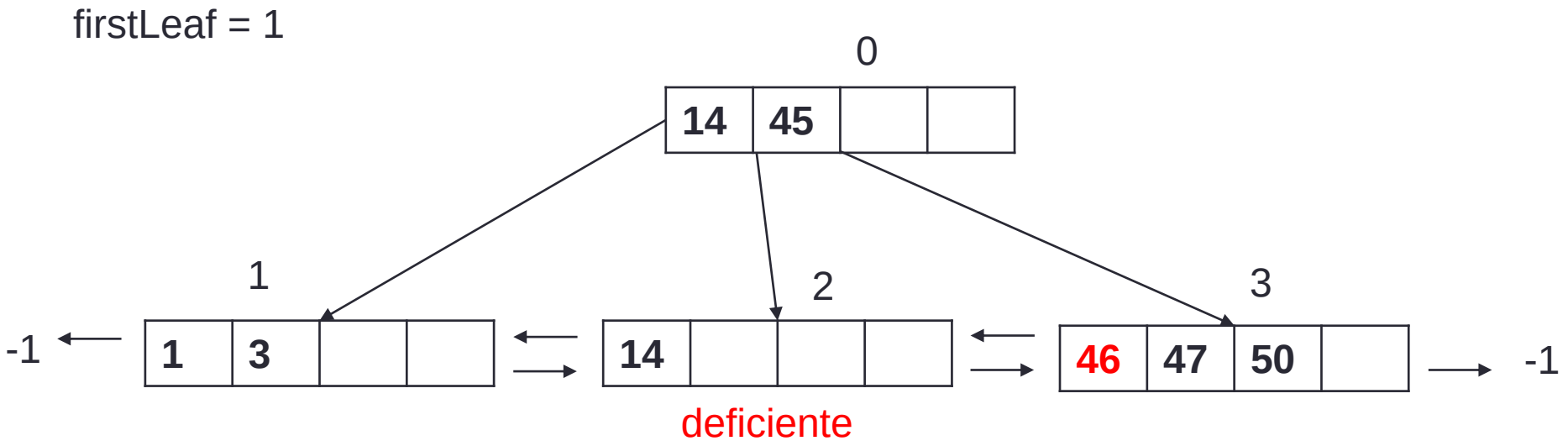
Remoção do 20



Remoção do 20

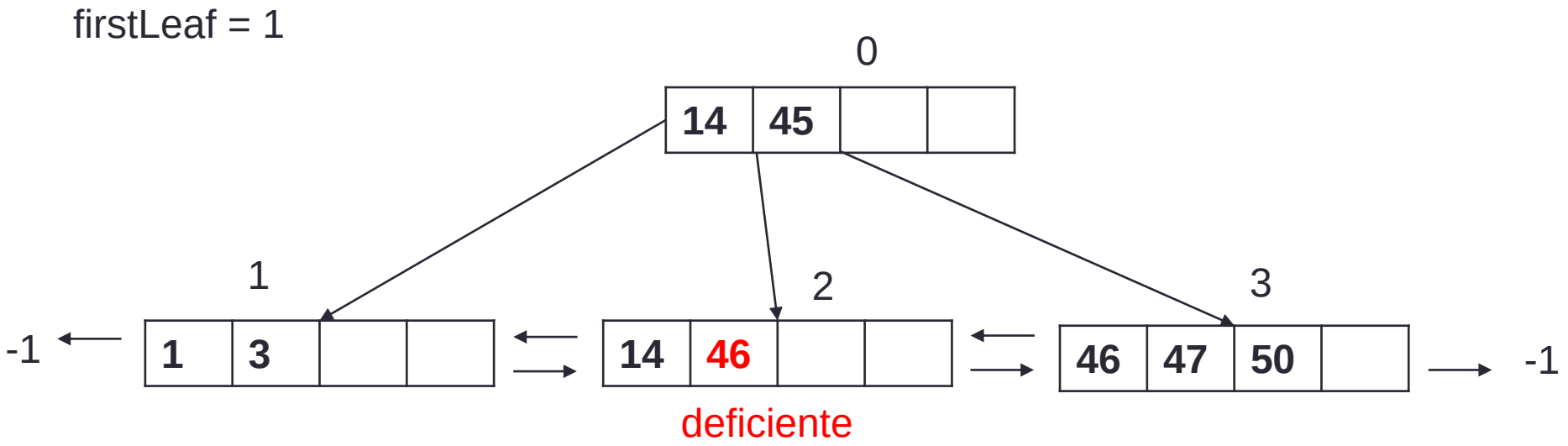
O nó 2 ficou deficiente.

O nó da esquerda (nó 1) não pode emprestar (pois ficaria deficiente)
Mas o nó da direita (nó 3) pode emprestar



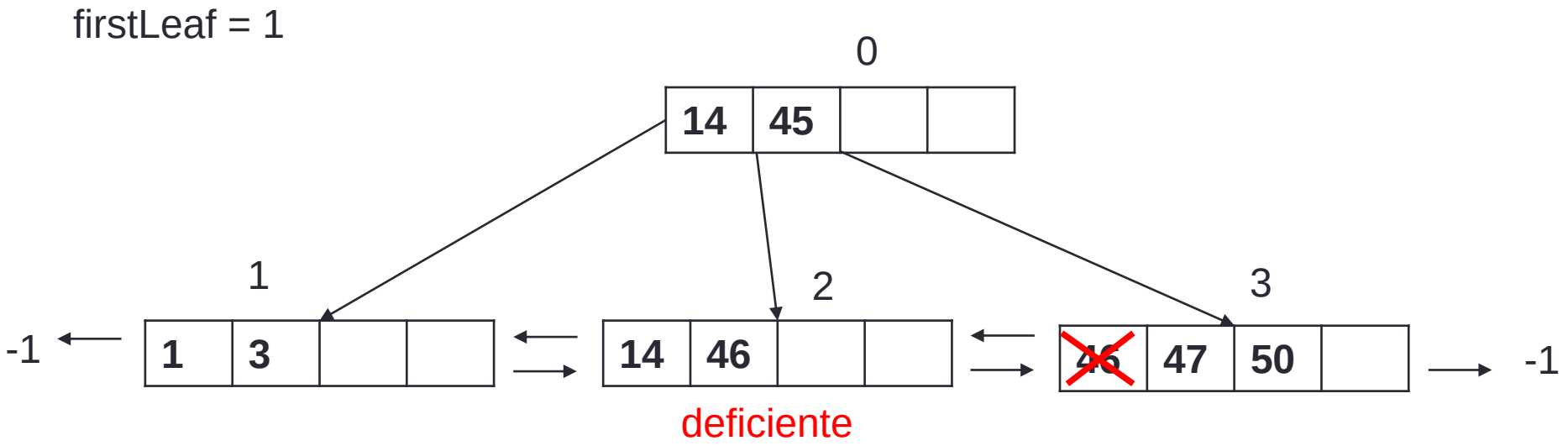
Remoção do 20

1. O primeiro registro do nó da direita (46)
   1. é copiado para a última posição do nó deficiente
   2. é removido do nó da direita
2. A nova menor chave do nó da direita (47) é copiada para a entrada correspondente no nó pai



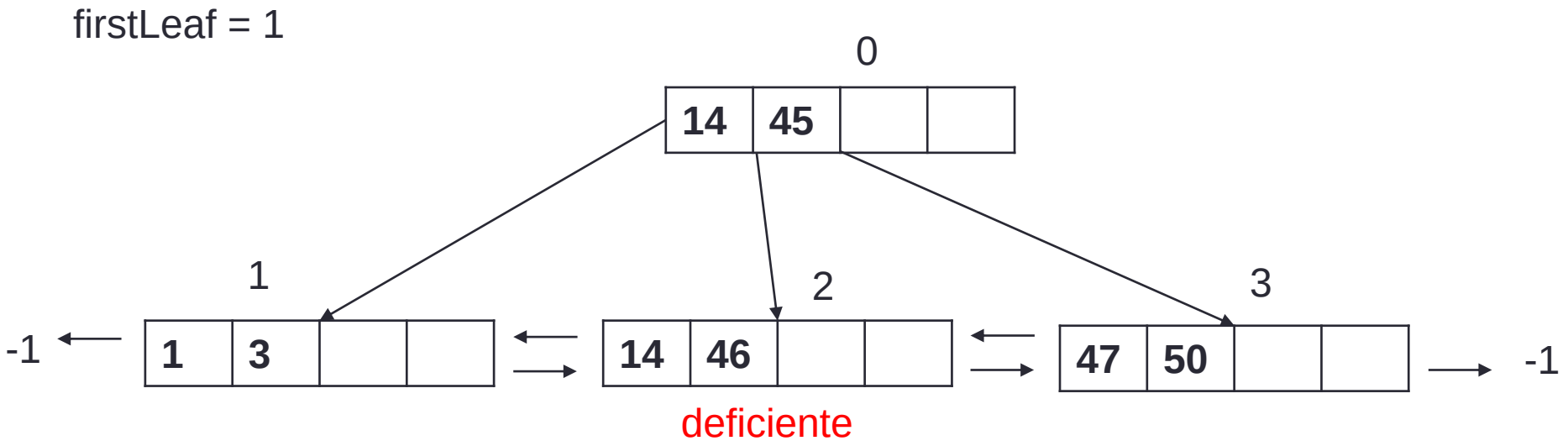
Remoção do 20

1. O primeiro registro do nó da direita (46)
    1. é copiado para a última posição do nó deficiente
    2. é removido do nó da direita
2. A nova menor chave do nó da direita (47) é copiada para a entrada correspondente no nó pai



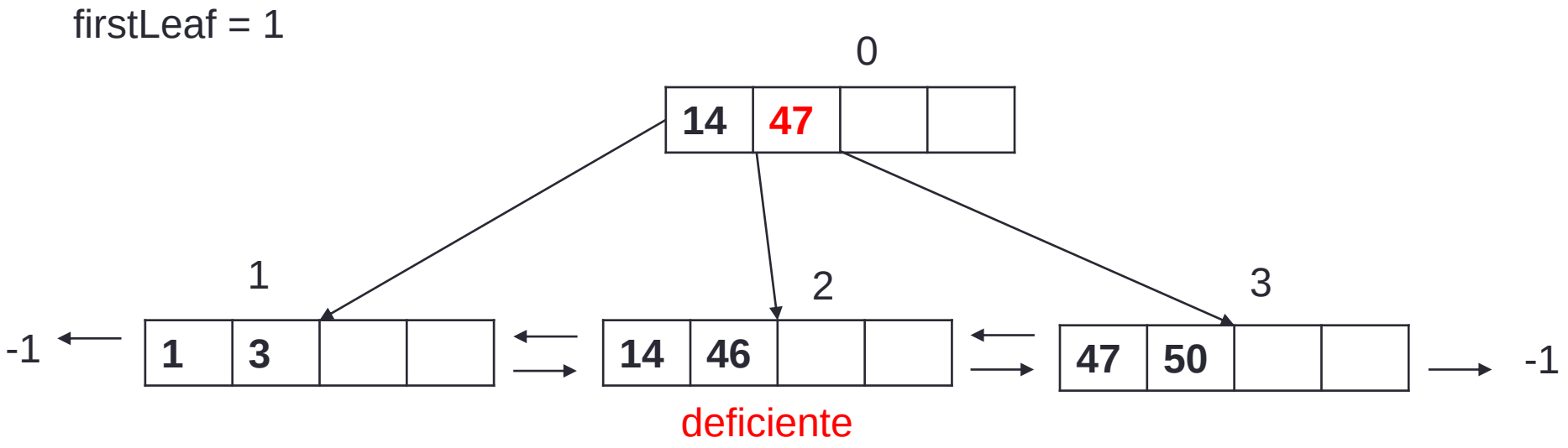
Remoção do 20

1. O primeiro registro do nó da direita (46)
   1. é copiado para a última posição do nó deficiente
   2. é removido do nó da direita
2. A nova menor chave do nó da direita (47) é copiada para a entrada correspondente no nó pai



Remoção do 20

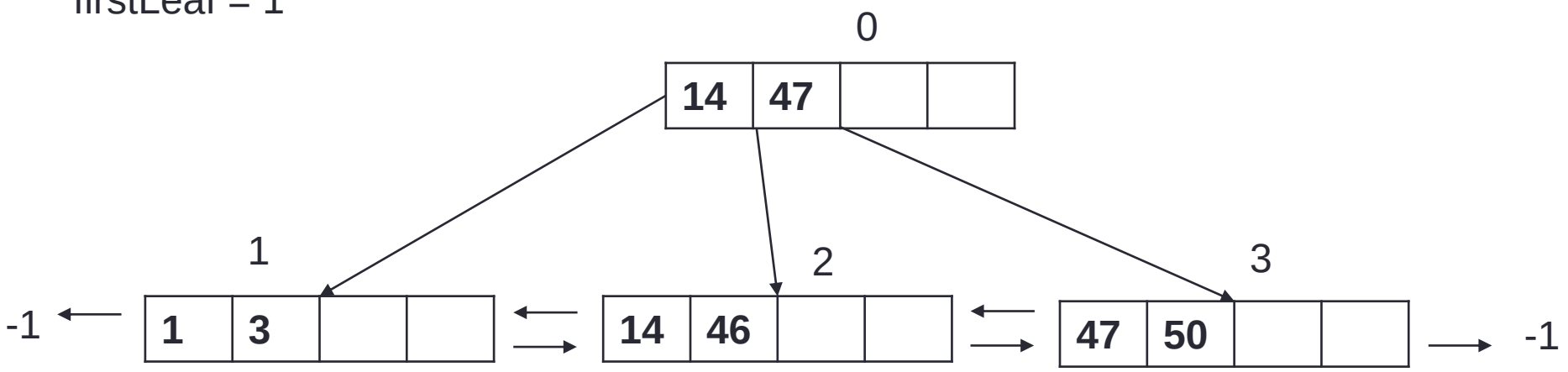
1. O primeiro registro do nó da direita (46)
   1. é copiado para a última posição do nó deficiente
   2. é removido do nó da direita
2. A nova menor chave do nó da direita (47) é copiada para a entrada correspondente no nó pai



Remoção do 20

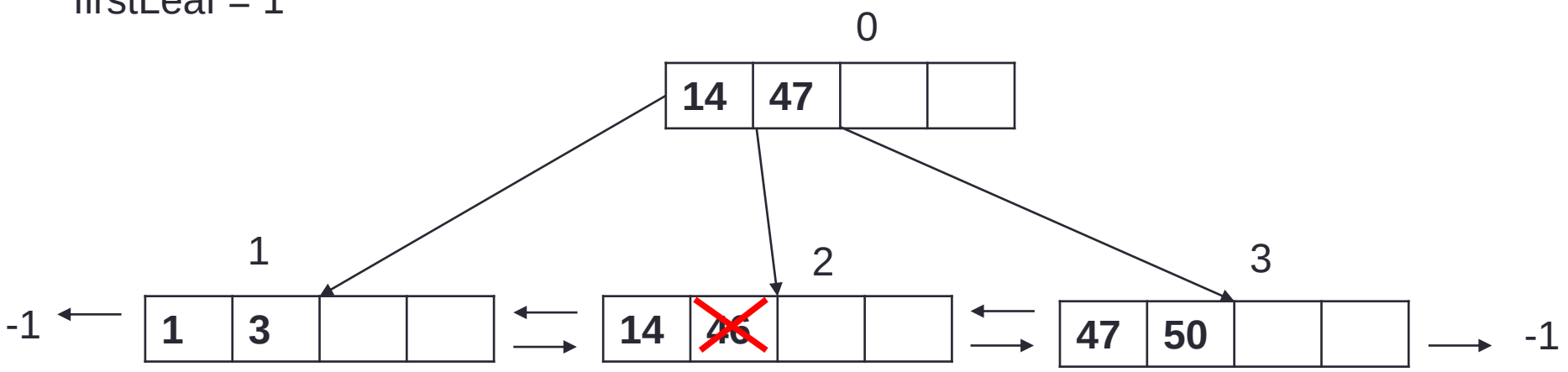
1. O primeiro registro do nó da direita (46)
    1. é copiado para a última posição do nó deficiente
    2. é removido do nó da direita
2. A nova menor chave do nó da direita (47) é copiada para a entrada correspondente no nó pai

firstLeaf = 1

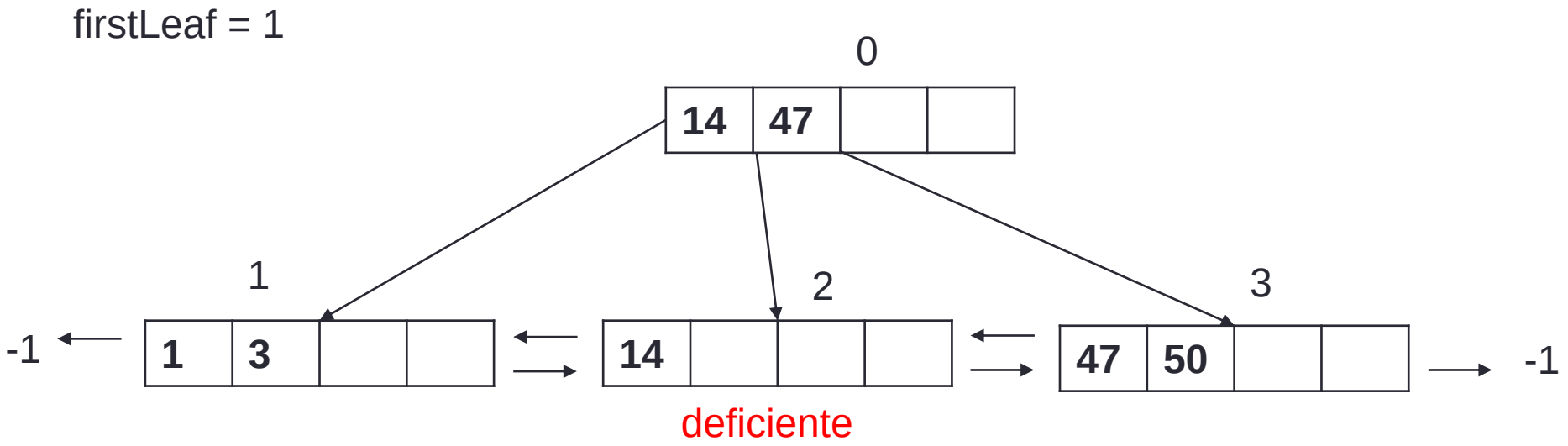


SITUAÇÃO 3: MESCLANDO COM O NÓ DA ESQUERDA

firstLeaf = 1



Remoção do 46

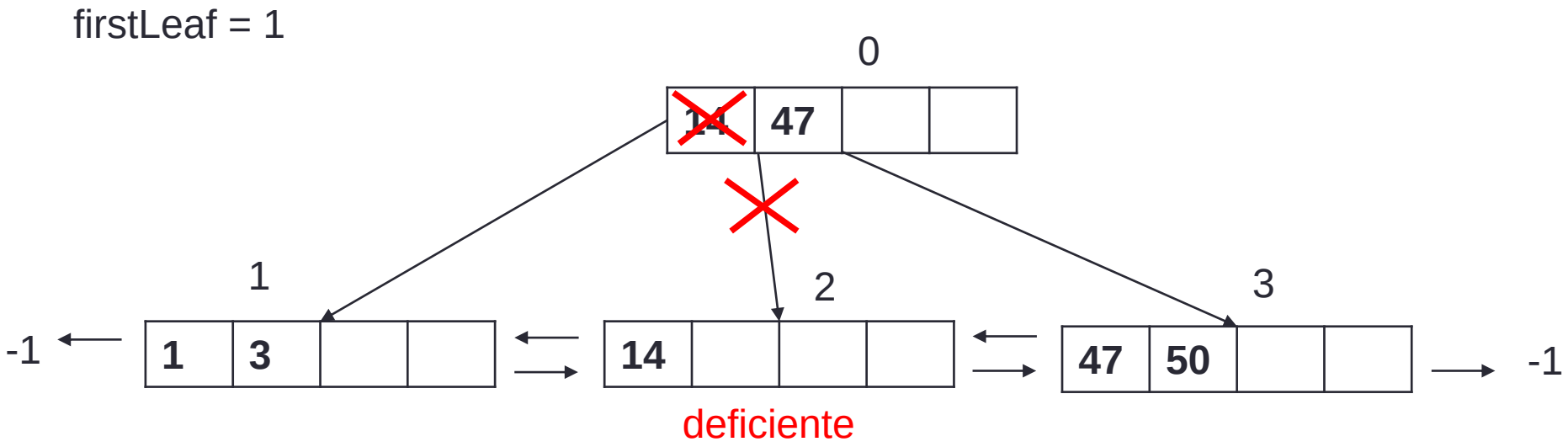


Remoção do 46

O nó 2 ficou deficiente.

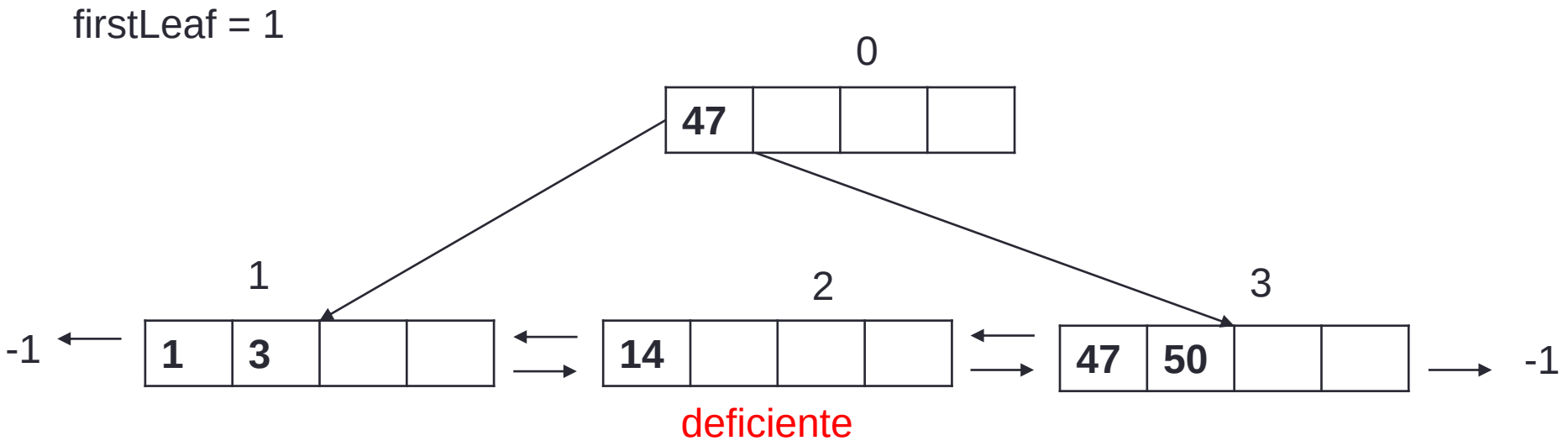
O nó da esquerda (nó 1) não pode emprestar (pois ficaria deficiente)
O nó da direita (nó 3) não pode emprestar (pois ficaria deficiente)

Mas dá para mesclar o nó deficiente com o nó da esquerda



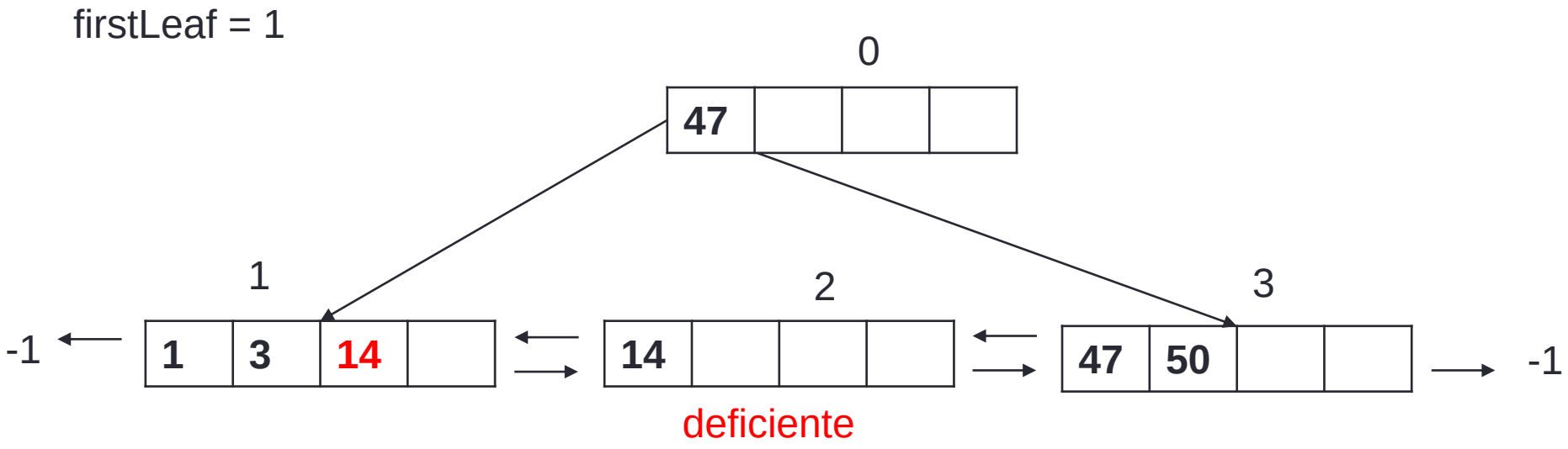
Remoção do 46

1. No pai, remover entrada correspondente ao nó deficiente (ponteiro+chave anterior)
2. Adicionar todas as entradas do nó deficiente no final do nó da esquerda
3. Atualizar ponteiros
4. Remover nó deficiente



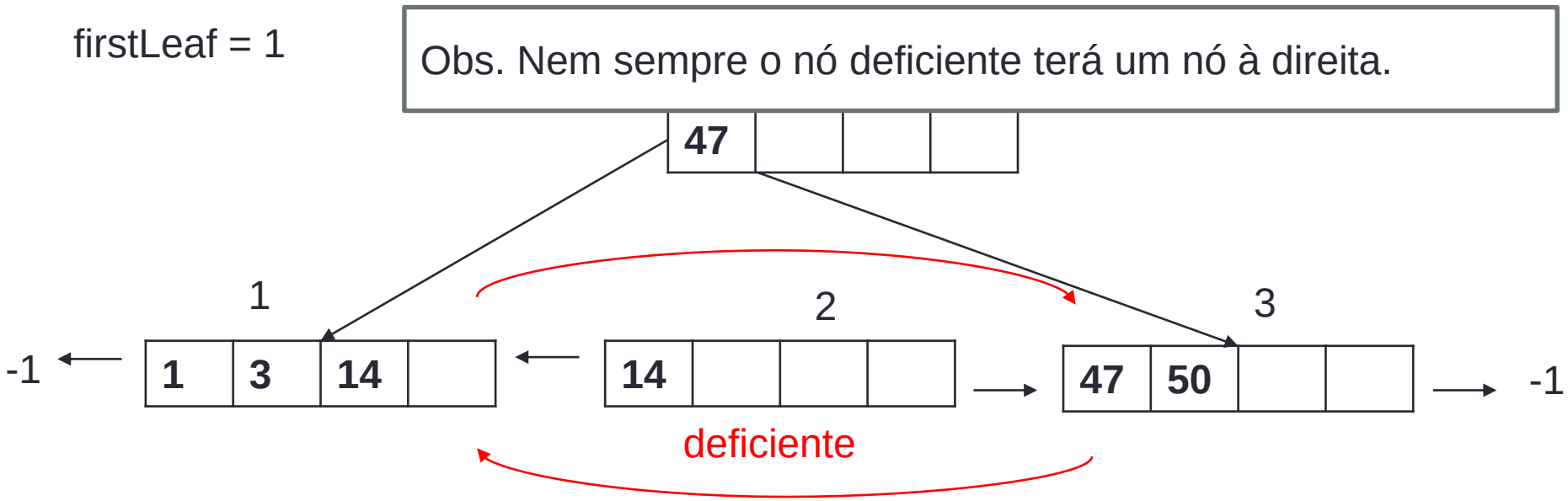
Remoção do 46

1. No pai, remover entrada correspondente ao nó deficiente (ponteiro+chave anterior)
2. Adicionar todas as entradas do nó deficiente no final do nó da esquerda
3. Atualizar ponteiros
4. Remover nó deficiente



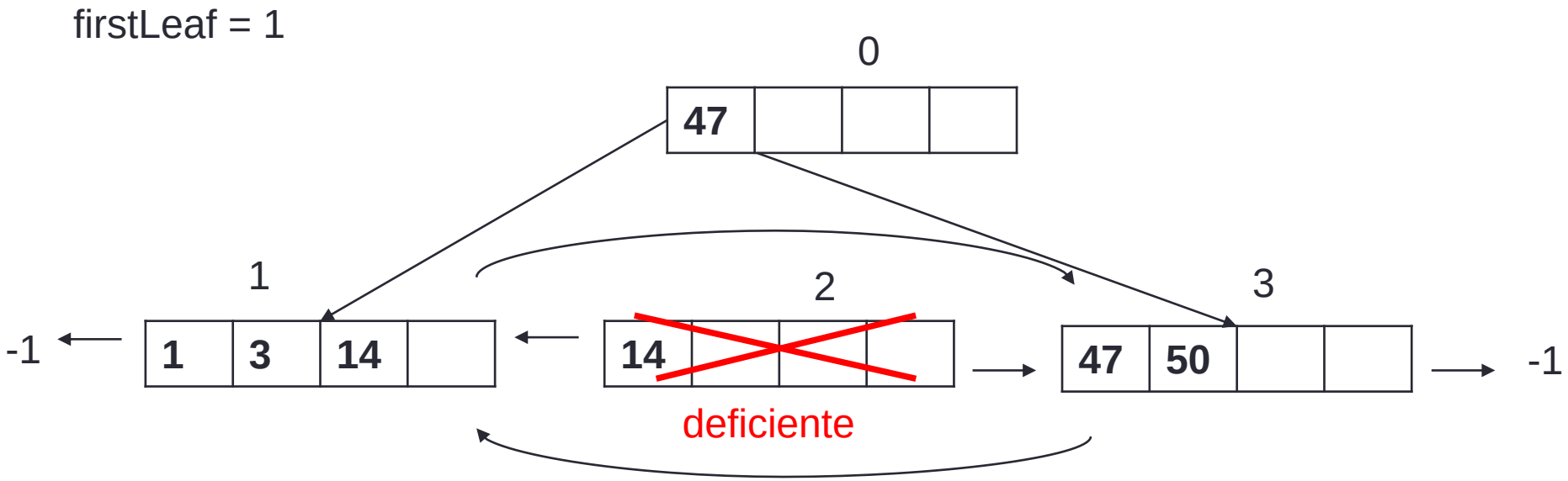
Remoção do 46

1. No pai, remover entrada correspondente ao nó deficiente (ponteiro+chave anterior)
2. Adicionar todas as entradas do nó deficiente no final do nó da esquerda
3. Atualizar ponteiros
4. Remover nó deficiente



Remoção do 46

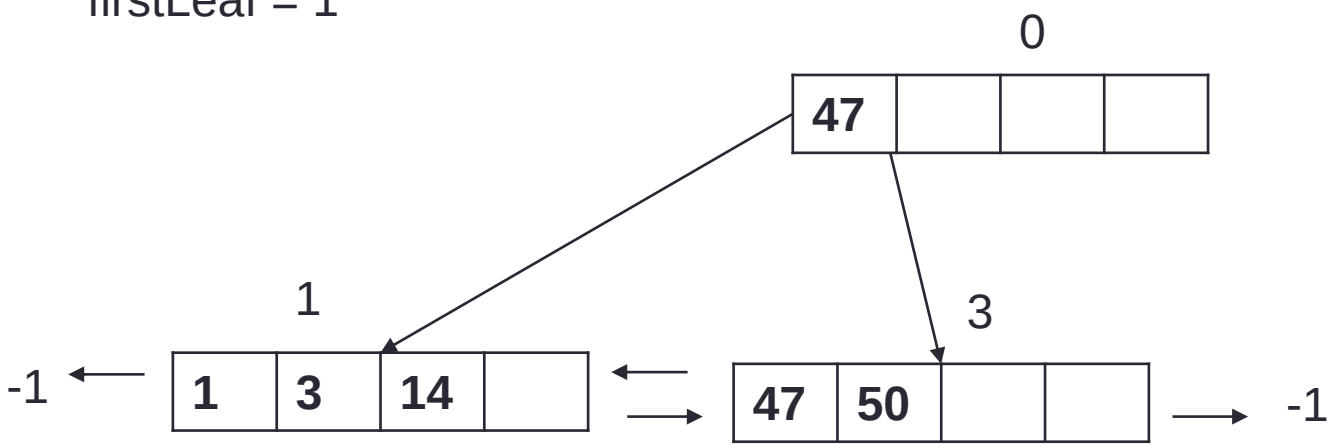
1. No pai, remover entrada correspondente ao nó deficiente (ponteiro+chave anterior)
2. Adicionar todas as entradas do nó deficiente no final do nó da esquerda
3. Atualizar ponteiros
4. Remover nó deficiente



Remoção do 46

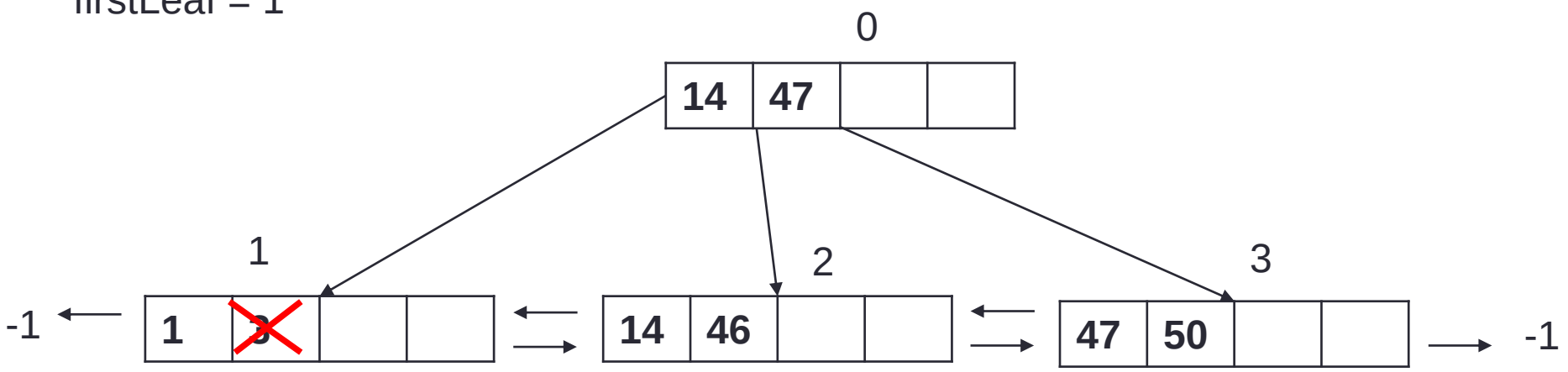
1. No pai, remover entrada correspondente ao nó deficiente (ponteiro+chave anterior)
2. Adicionar todas as entradas do nó deficiente no final do nó da esquerda
3. Atualizar ponteiros
4. Remover nó deficiente

firstLeaf = 1

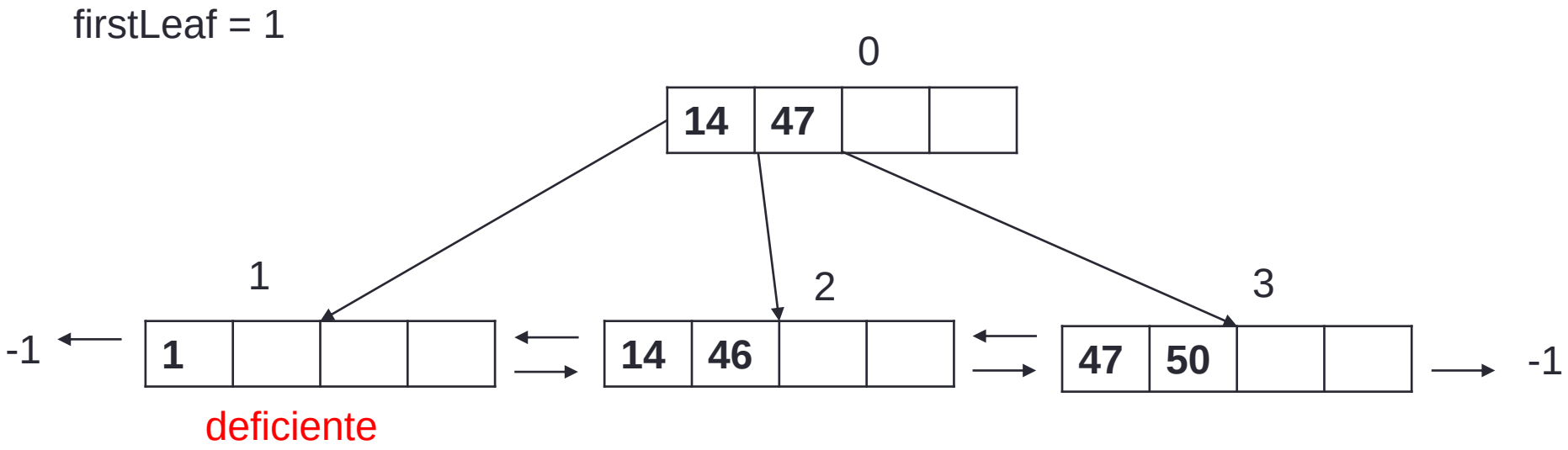


SITUAÇÃO 4: MESCLANDO COM O NÓ DA DIREITA

firstLeaf = 1



Remoção do 3

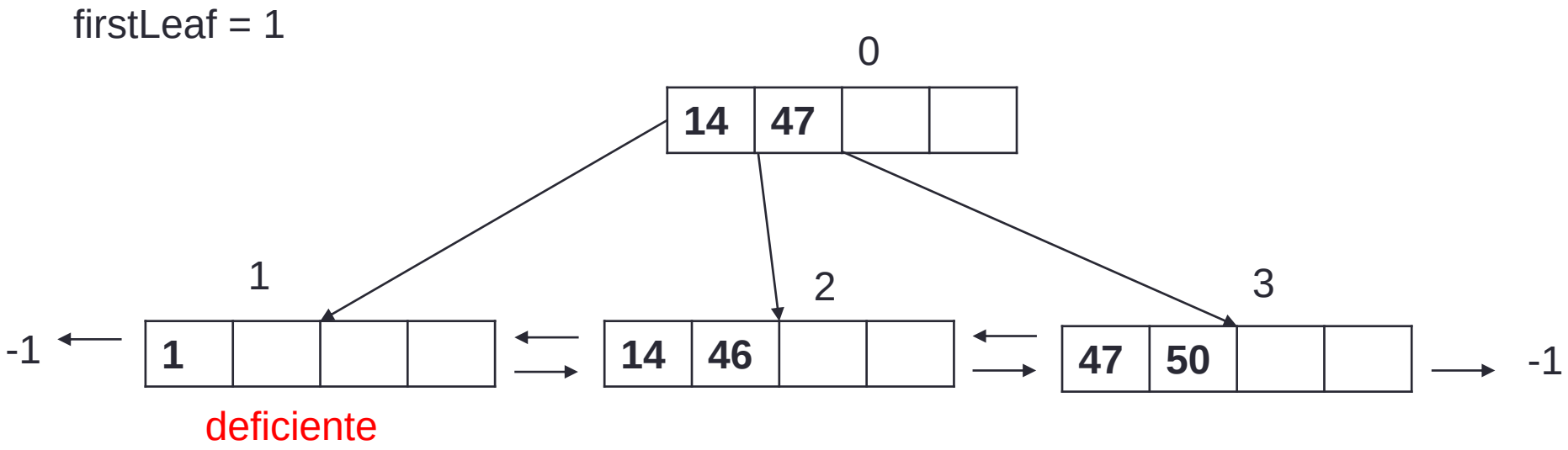


Remoção do 3

O nó 1 ficou deficiente.

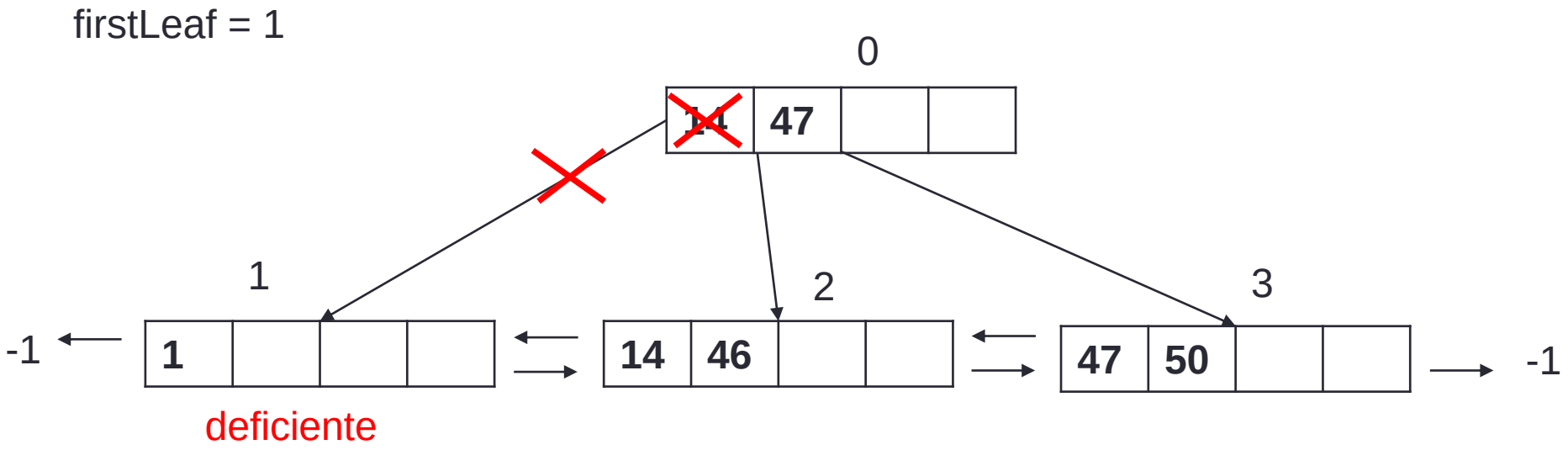
O nó da esquerda não pode emprestar (pois não há)
O nó da direita (nó 3) não pode emprestar (pois ficaria deficiente)

Não dá para mesclar o nó deficiente com o nó da esquerda (pois não há)
Mas dá para mesclar com o nó da direita (nó 2)



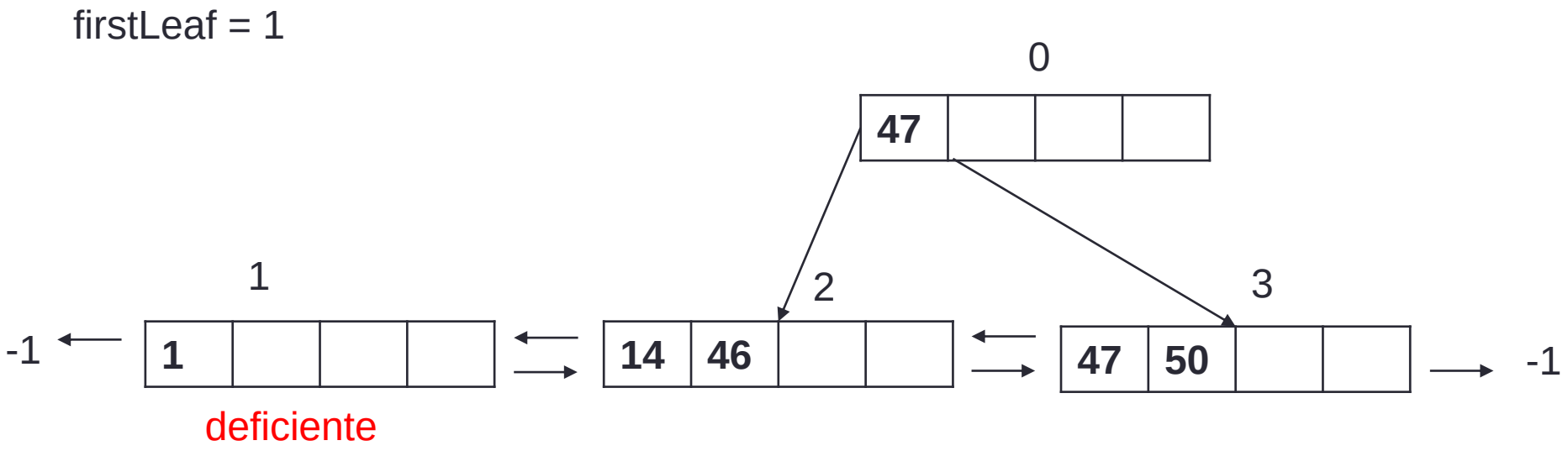
Remoção do 3

1. No pai, remover entrada correspondente ao nó deficiente (ponteiro+chave posterior)
2. Adicionar todas as entradas do nó deficiente no começo do nó da direita
3. Atualizar ponteiros
4. Remover nó deficiente



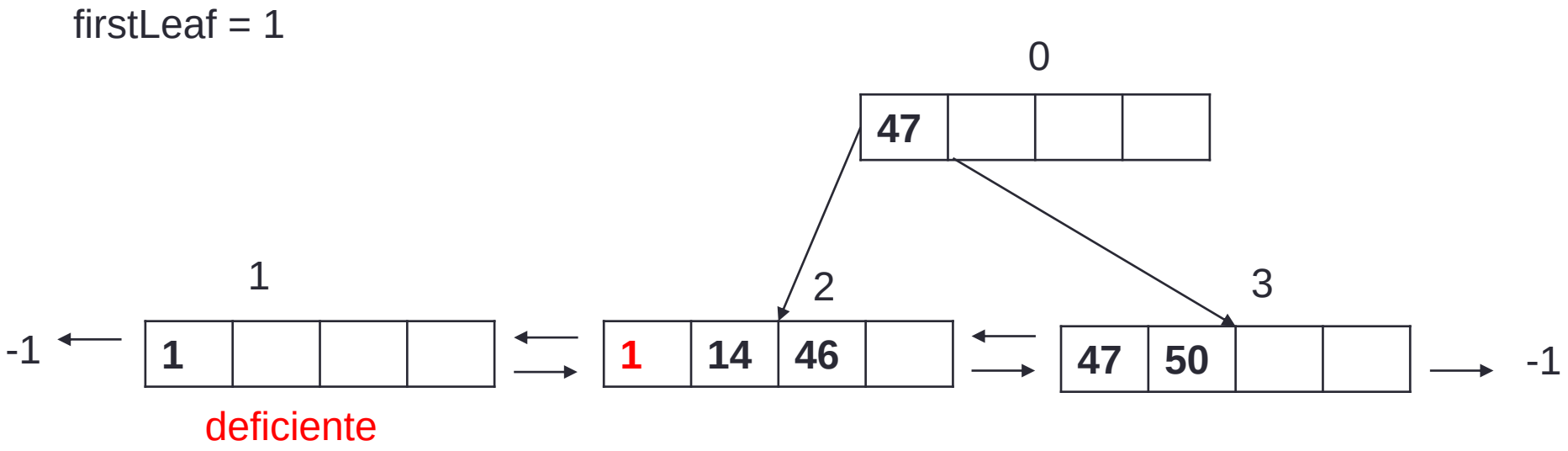
Remoção do 3

1. No pai, remover entrada correspondente ao nó deficiente (ponteiro+chave posterior)
2. Adicionar todas as entradas do nó deficiente no começo do nó da direita
3. Atualizar ponteiros
4. Remover nó deficiente



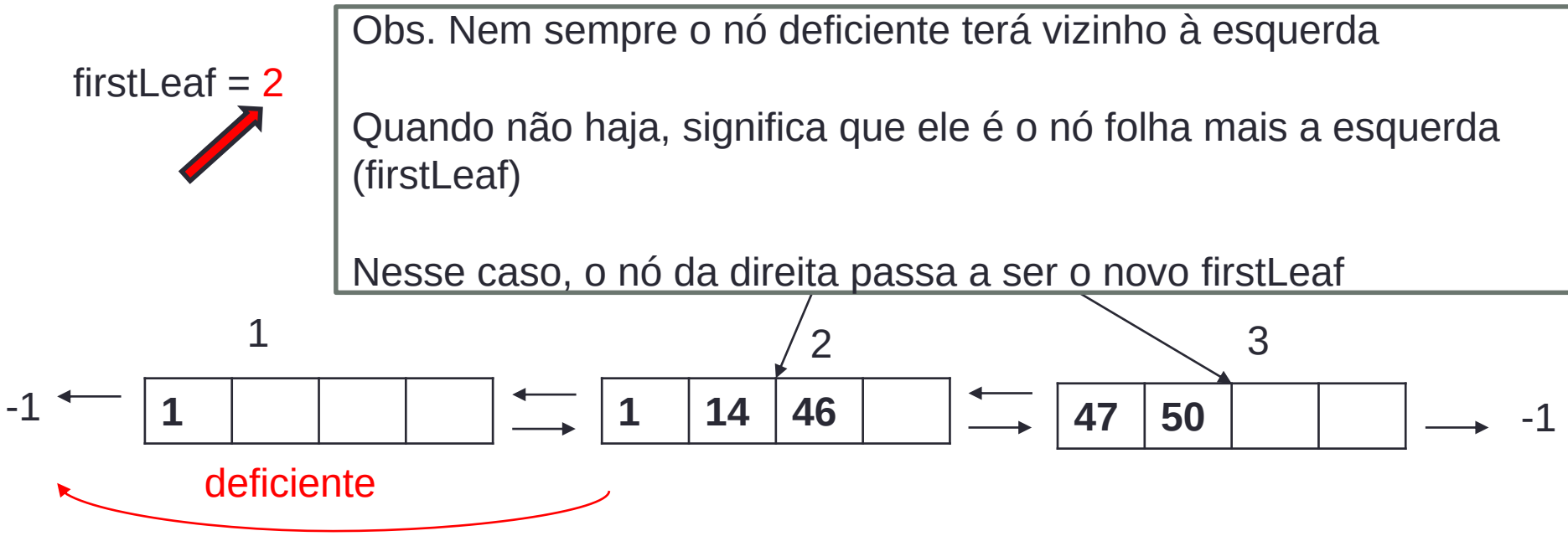
Remoção do 3

1. No pai, remover entrada correspondente ao nó deficiente (ponteiro+chave posterior)
2. Adicionar todas as entradas do nó deficiente no começo do nó da direita
3. Atualizar ponteiros
4. Remover nó deficiente



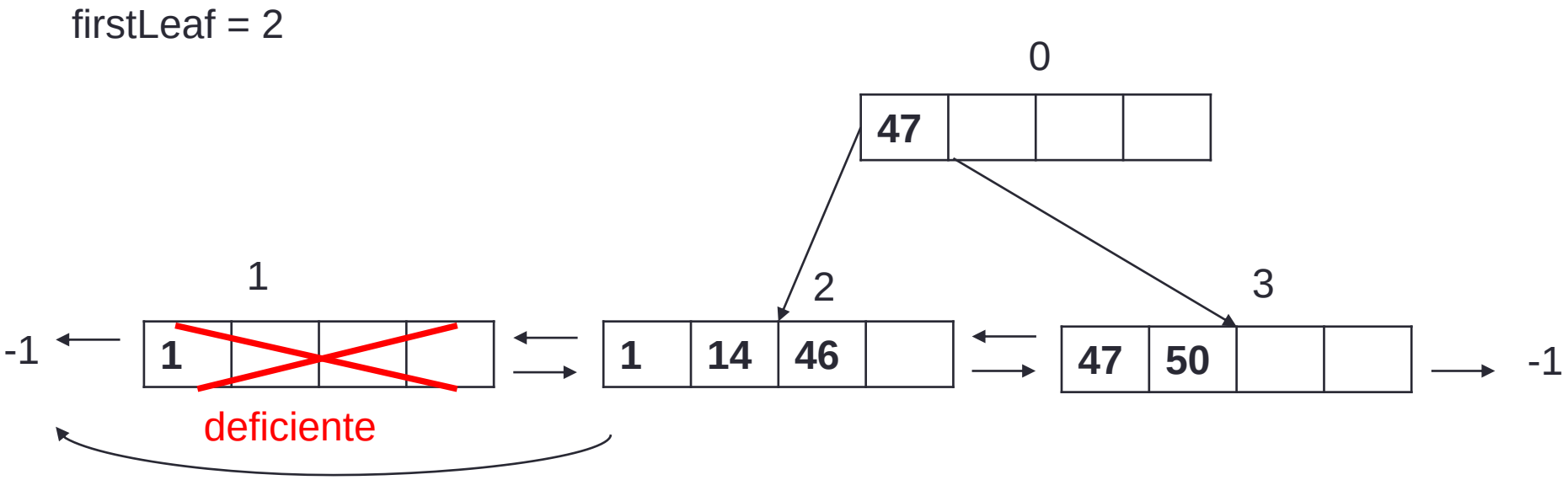
Remoção do 3

1. No pai, remover entrada correspondente ao nó deficiente (ponteiro+chave posterior)
2. Adicionar todas as entradas do nó deficiente no começo do nó da direita
3. Atualizar ponteiros
4. Remover nó deficiente



Remoção do 3

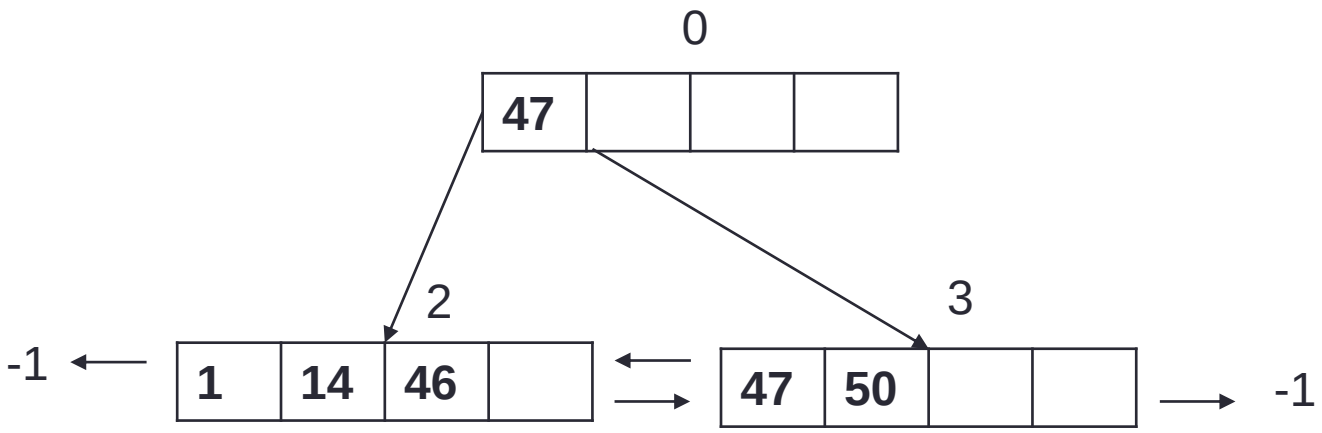
1. No pai, remover entrada correspondente ao nó deficiente (ponteiro+chave posterior)
2. Adicionar todas as entradas do nó deficiente no começo do nó da direita
3. Atualizar ponteiros
4. Remover nó deficiente



Remoção do 3

1. No pai, remover entrada correspondente ao nó deficiente (ponteiro+chave posterior)
2. Adicionar todas as entradas do nó deficiente no começo do nó da direita
3. Atualizar ponteiros
4. Remover nó deficiente

firstLeaf = 2



Remoção do 3

1. No pai, remover entrada correspondente ao nó deficiente (ponteiro+chave posterior)
2. Adicionar todas as entradas do nó deficiente no começo do nó da direita
3. Atualizar ponteiros
4. Remover nó deficiente

Trabalho Prático

- A função a ser modificada é a

```
public Value delete(Key key)
```

- Essa função remove o registro do nó folha.
  - No entanto, a versão atual não realiza o balanceamento caso o nó fique deficiente

Trabalho Prático

- As quatro possibilidades de balanceamento estão bem demarcadas dentro do função delete
  - Basta criar o código referente a cada situação

- Ex.
  - o trecho abaixo demarca o bloco de código que deve implementar a estratégia que empresta um registro do nó da esquerda

```
if (ln.leftSibling != null
                    && ln.leftSibling.getParentID() == ln.getParentID()
                    && ln.leftSibling.isLendable()) {


…
}
```

Estrutura do código

- Para fazer ajustes no nó folha deficiente, é importante saber como os nós folhas estão estruturados
  - Um nó folha armazena os registros em um vetor de DictionaryPair
    - Um DictionaryPair possui duas informações
      - Key: a chave do valor
      - Value: o valor que corresponde ao registro

- Ex. O nó folha abaixo possui quatro DictionaryPair

10,”gg”	23,”bb”	45,”fd”	52,”xx”

Nó folha

Estrutura do código

- As seguintes funções são importantes para fazer migrações de registros entre dois nós folha
  - node.getFirstDictionaryPair();
    - Retorna o primeiro DictionaryPair de node
  - node.getLastDictionaryPair() ;
    - Retorna o último DictionaryPair de node
  - node.deleteFirstDictionaryPair();
    - Remove o primeiro DictionaryPair de node
  - node.deleteLastDictionaryPair() ;
    - Remove o último DictionaryPair de node

Estrutura do código

- As seguintes funções são importantes para fazer migrações de registros entre dois nós folha
  - node1.prependPair(dp);
    - Adiciona o DictionaryPair dp no começo de node1
  - node1.appendPair(dp);
    - Adiciona o DictionaryPair dp no final de node1
  - node1.prependPairs(node2);
    - Adiciona todos os DictionaryPairs do node2 no começo do node1
  - node1.appendPairs(node2);
    - Adiciona todos os DictionaryPairs do node2 no final do node1

Estrutura do código

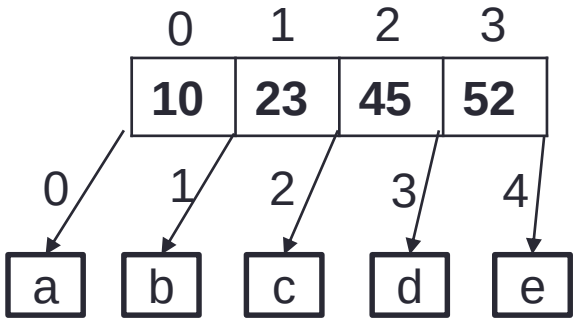
- As seguintes funções são importantes para a atualização de ponteiros no nível folha
  - node.getPageID()
    - Retorna o ID da página referente ao nó
  - node.leftSiblingID
    - Dá acesso direto ao ID dá página referente ao nó vizinho da esquerda
  - node.rightSiblingID
    - Dá acesso direto ao ID dá página referente ao nó vizinho da direita
  - setFirstLeafID(id)
    - Atribui o valor do firstLeaf

- Obs. O valor -1 significa que não há ponteiro
  - Ex. se leftSiblingID = -1, não há vizinho à esquerda

Estrutura do código

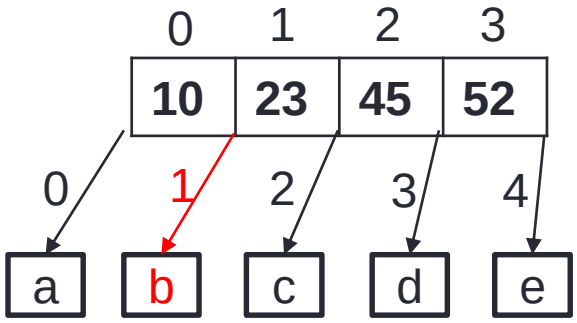
- Para fazer ajustes no nó pai de um folha, é importante saber como o pai (que é um nó interno) está estruturado
- Um nó interno possui dois vetores
  - childPointers: vetor de n posições que guarda os ponteiros para os filhos
  - keys: vetor de n-1 posições que guarda as chaves

  - Ex. O nó abaixo guarda 4 chaves (10,23,45,52) e ponteiros para 5 nós folha, aqui representados por letras (a,b,c,d,e)



Estrutura do código

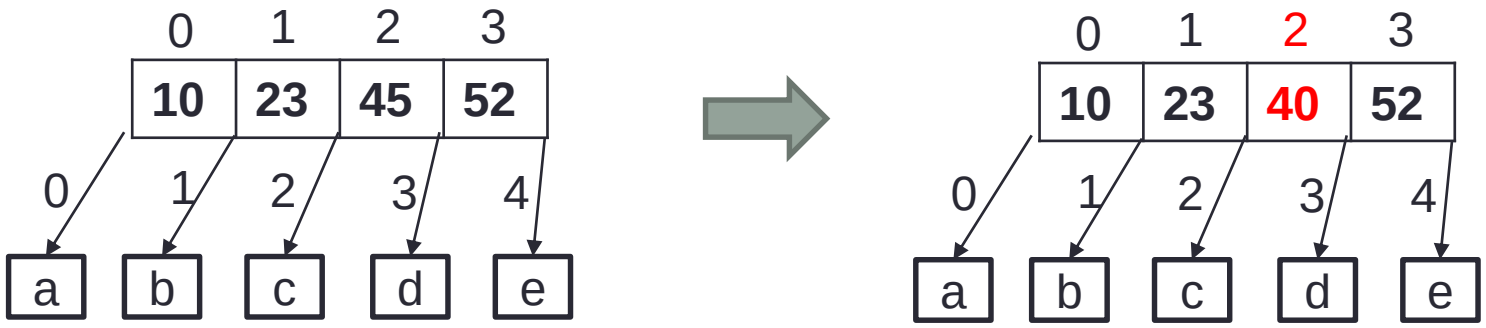
- A função node1.findIndexOfPointer(node2)
  - Encontra dentro do nó interno node1 o índice do ponteiro que leva ao nó node2
  - Ex. a chamada findIndexOfPointer(b) retorna 1



Estrutura do código

- Pode-se mudar o valor de uma chave de um nó interno diretamente usando node.keys

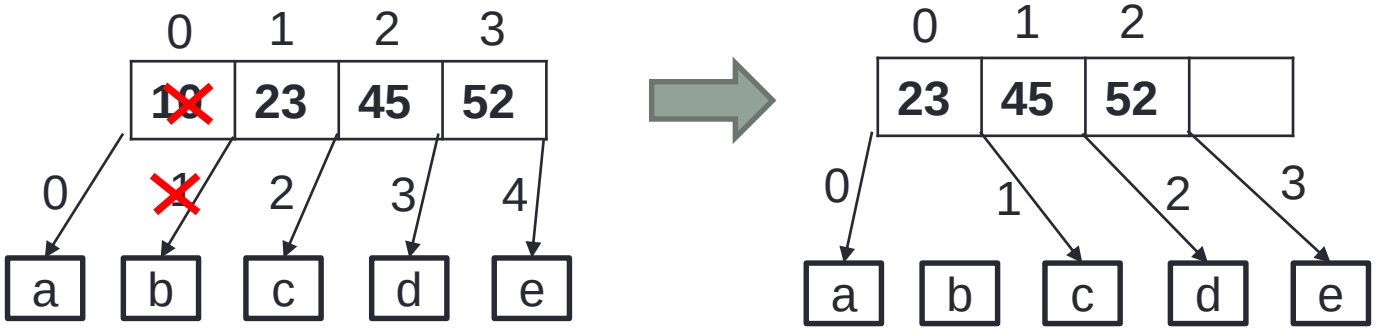
- Ex. keys[2] = 40



Estrutura do código

- A função node1.removeEntry(index);
  - Remove uma entrada (ponteiro + chave anterior) de node1
    - Ponteiro removido: a que possui index
    - Chave anterior: a chave que precede o ponteiro

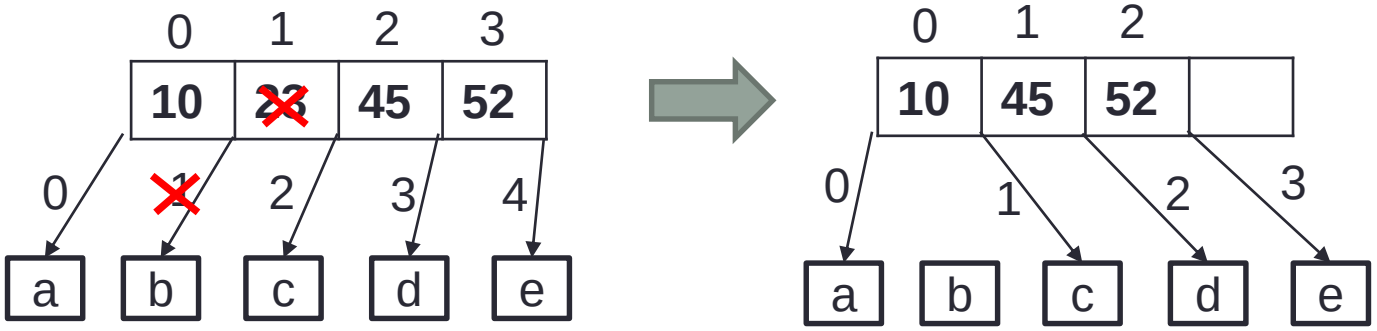
- Ex. removeEntry(1)



Estrutura do código

- A função node1.removeEntry1(index);
  - Remove uma entrada (ponteiro + chave posterior) de node1
    - Ponteiro removido: a que possui index
    - Chave posterior: o que sucede o ponteiro

- Ex. removeEntry1(1)



Estrutura do código

- Acesso aos nós
  - getNode(id)
    - Recupera o nó referente ao id passado por parâmetro
- Ex. Para obter o vizinho de um nó folha
  - LeafNode sibling = (LeafNode) getNode(node.leftSiblingID);
    - Recupera o vizinho da esquerda de node
- Ex. Para obter o pai de um nó
  - InternalNode parent = (InternalNode) getNode(node.getParentID());
    - Recupera o pai de node
- Ao usar getNode(id)
  - cerfique-se que o id seja válido (>0)

Estrutura do código

- Atualização de nós(páginas)
  - deleteNode(node)
    - Remove a página(nó) passado por parâmetro
  - WriteNode(node)
    - Escreve as modificações feitas no nó passado por parâmetro
    - Importante:
      - ao modificar um nó, deve-se necessariamente chamar o writeNode
        - Caso contrário, as modificações não serão salvas no disco
      - Em alguns casos, mais de um nó é modificado
        - Todos os nós modificados precisam ser salvos

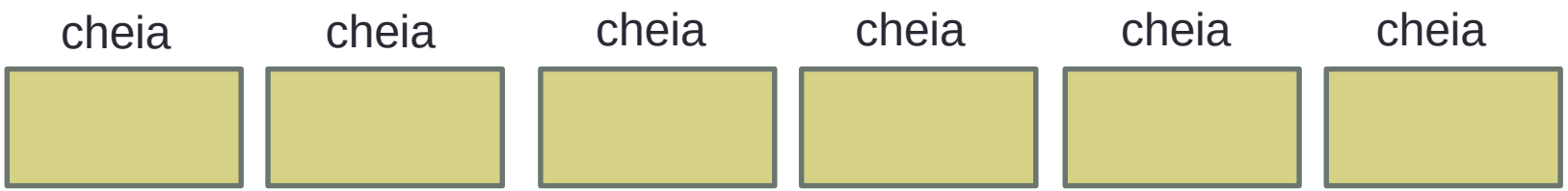
Testes

- Foi disponibilizada uma classe de testes
  - MainTrabalho

- Essa classe contém três funções que devem ser usadas em execuções independentes (uma de cada vez)
  - testBorrowFromSiblingStrategy()
    - Testa se o empréstimo de algum vizinho (da esquerda ou direita) foi bem implementado
  - testLeftSiblingMergeStrategy()
    - Testa se a mescla com o vizinho da direita foi bem implementada
  - testRightSiblingMergeStrategy()
    - Testa se a mescla com o vizinho da esquerda foi bem implementada

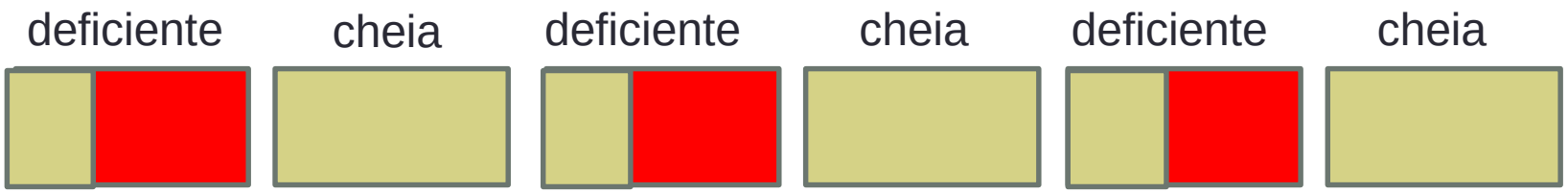
testBorrowFromSiblingStrategy()

- A função testBorrowFromSiblingStrategy()
  - Cria páginas cheias
  - Remove registros de páginas intercaladas, começando da primeira
    - Até deixá-las deficientes
  - Insere registros em páginas intercaladas, começando da segunda



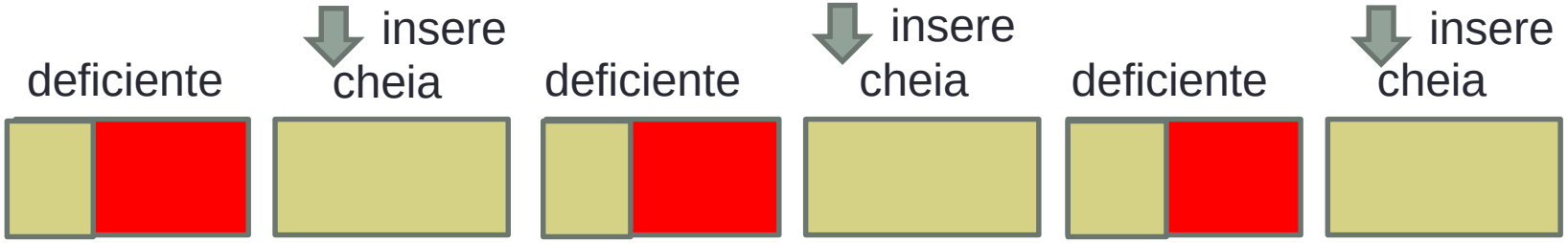
testBorrowFromSiblingStrategy()

- A função de teste testBorrowFromSiblingStrategy()
  - Cria páginas cheias
  - Remove registros de páginas intercaladas, começando da primeira
    - Até deixá-las deficientes
  - Insere registros em páginas intercaladas, começando da segunda



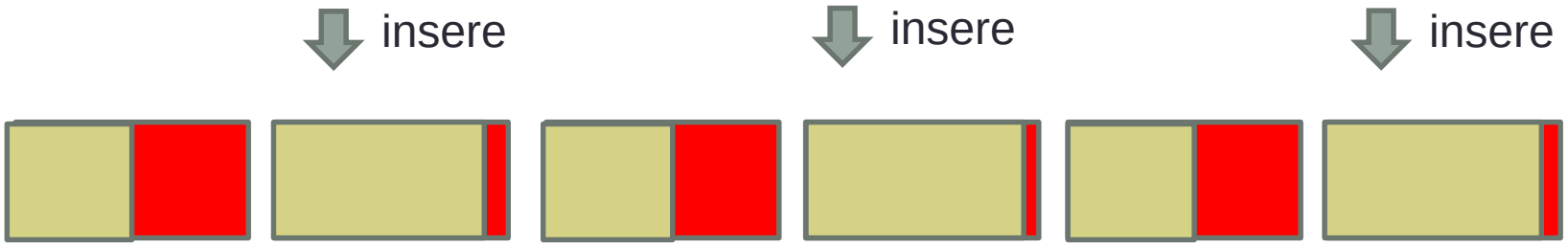
testBorrowFromSiblingStrategy()

- A função de teste testBorrowFromSiblingStrategy()
  - Cria páginas cheias
  - Remove registros de páginas intercaladas, começando da primeira
    - Até deixá-las deficientes
  - Insere registros em páginas intercaladas, começando da segunda



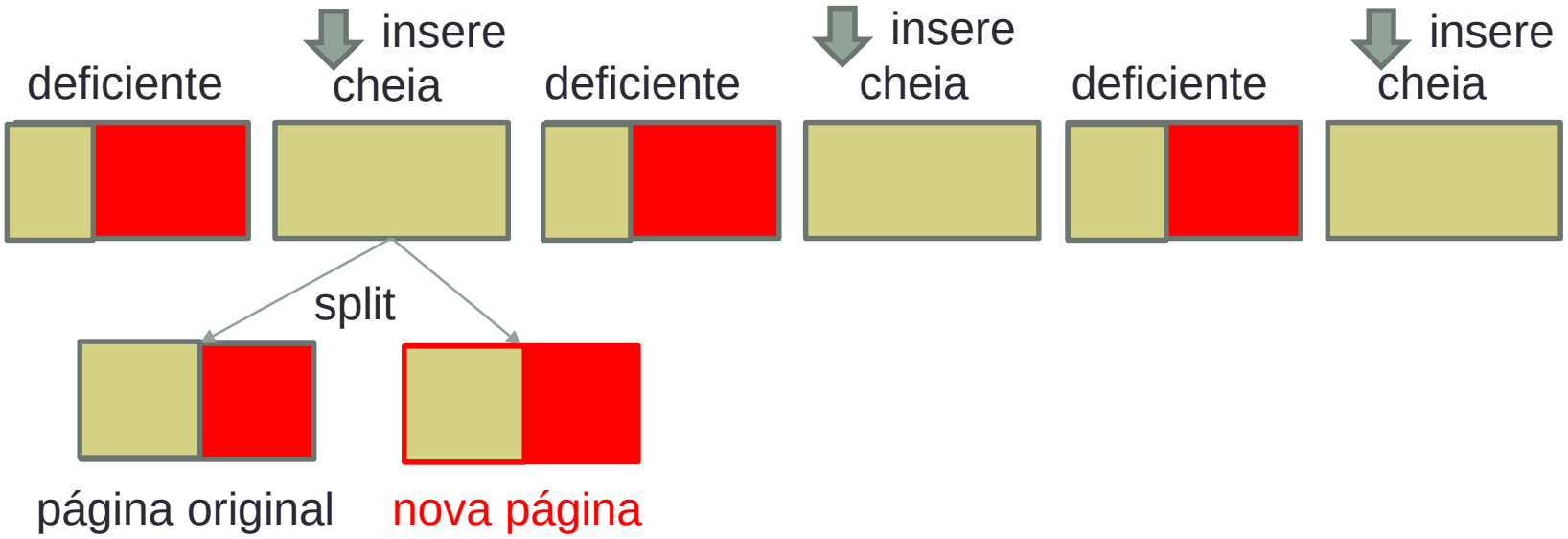
testBorrowFromSiblingStrategy()

- Se for implementada a estratégia de empréstimo
  - O nós cheios emprestarão registros
  - E terão espaço sobrando caso seja necessário inserir novos registros



testBorrowFromSiblingStrategy()

- Se nenhuma estratégia de empréstimo for usada
  - As páginas estarão cheias
  - E com isso, precisarão sofrer split
  - O que gerará mais páginas e fará o arquivo ficar maior

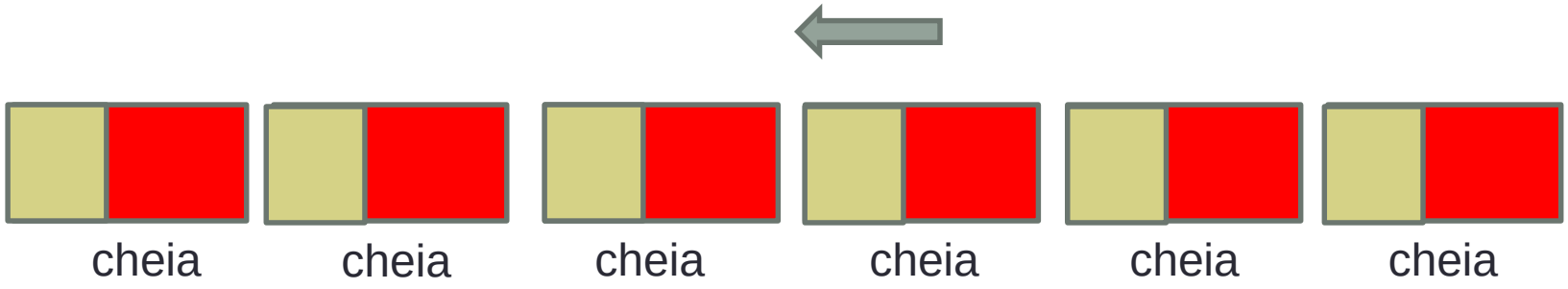


testRightSiblingMergeStrategy()

- A função de teste testRightSiblingMergeStrategy()
  - Cria páginas cheias
  - Remove registros de todas as páginas, processando da direita para a esquerda, deixando-as deficientes
  - Insere registros maiores do que todos os existentes, o que leva a novas páginas

testRightSiblingMergeStrategy()

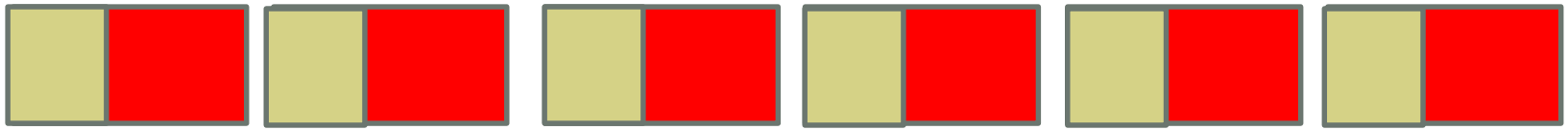
- A função de teste testRightSiblingMergeStrategy()
  - Cria páginas cheias
  - Remove registros de todas as páginas, processando da direita para a esquerda, deixando-as deficientes
  - Insere registros maiores do que todos os existentes, o que leva a novas páginas



testRightSiblingMergeStrategy()

- A função de teste testRightSiblingMergeStrategy()
  - Cria páginas cheias
  - Remove registros de todas as páginas, processando da direita para a esquerda, deixando-as deficientes
  - Insere registros maiores do que todos os existentes, o que leva a novas páginas

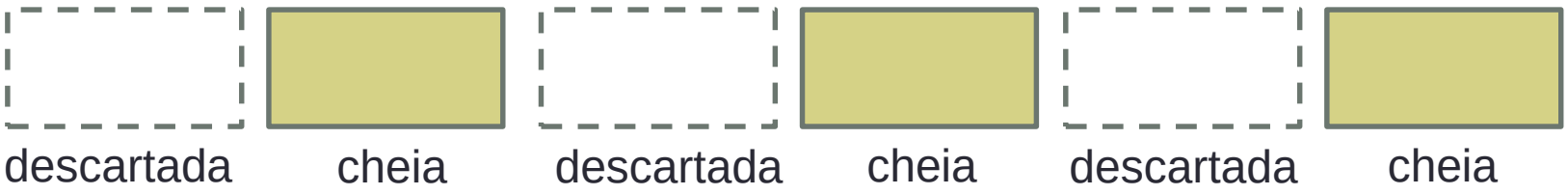
nova página



cheia cheia cheia cheia cheia cheia

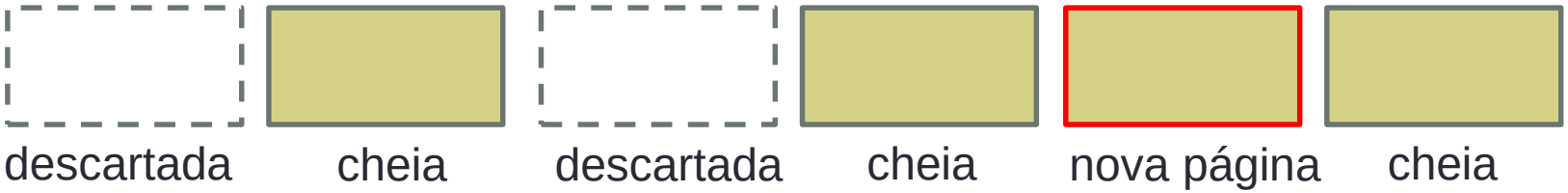
testRightSiblingMergeStrategy()

- Se a estratégia de mescla for implementada, uma página deficiente será mesclada com a sua vizinha da direita
  - E será descartada



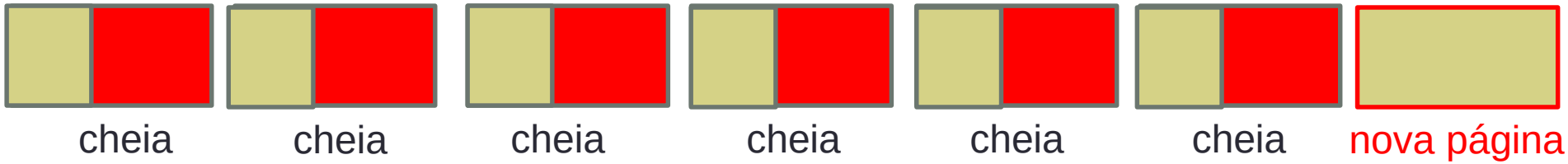
testRightSiblingMergeStrategy()

- Quando for necessário gerar novas páginas, serão usados os espaços descartados em vez de criar novas páginas no fim do arquivo
  - O que evita que o arquivo aumente de tamanho



testRightSiblingMergeStrategy()

- Caso contrário, uma página será criada no fim do arquivo, fazendo com que ele fique maior

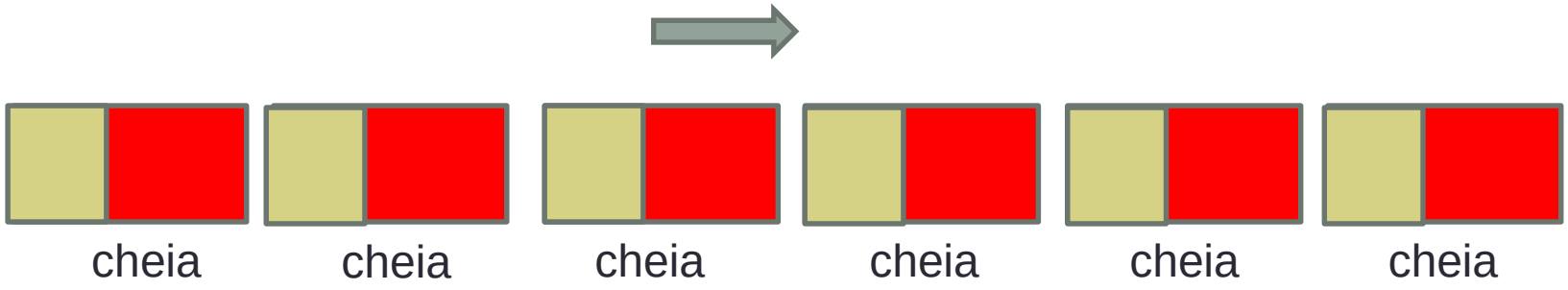


testLeftSiblingMergeStrategy()

- A função de teste testLeftSiblingMergeStrategy()
  - Cria páginas cheias
  - Remove registros de todas as páginas, processando da esquerda para a direita, deixando-as deficientes
  - Insere registros maiores do que todos os existentes, o que leva a novas páginas

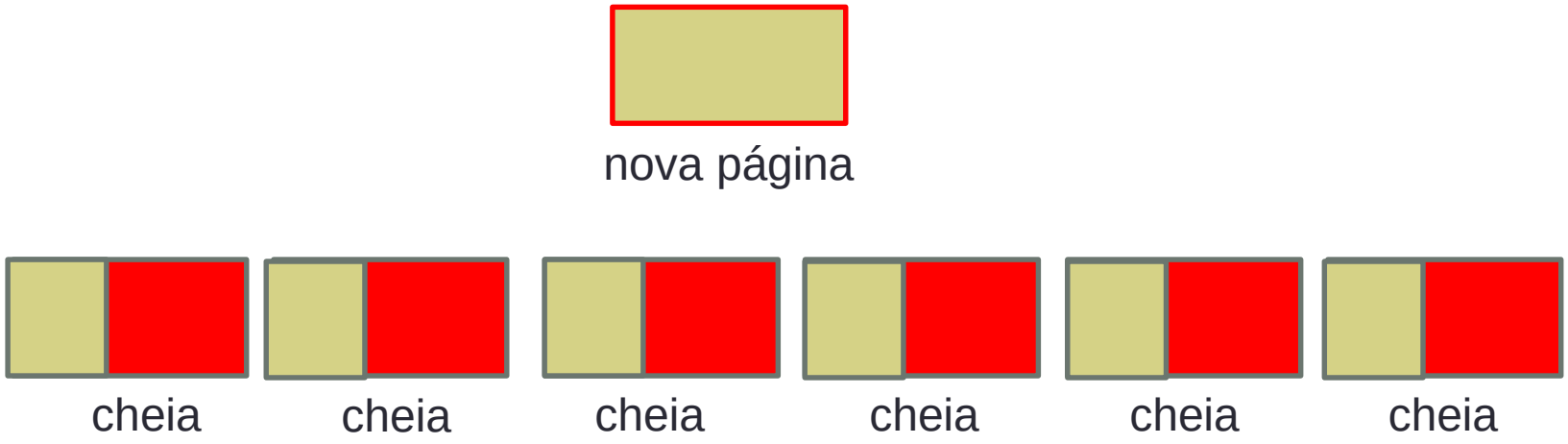
testLeftSiblingMergeStrategy

- A função de teste testLeftSiblingMergeStrategy()
  - Cria páginas cheias
  - Remove registros de todas as páginas, processando da esquerda a direita, deixando-as deficientes
  - Insere registros maiores do que todos os existentes, o que leva a novas páginas



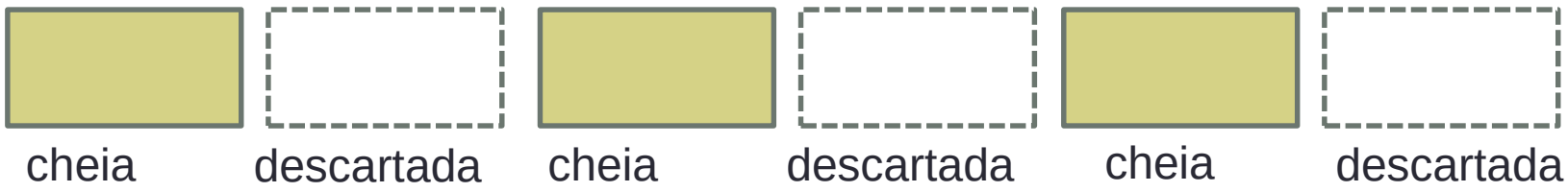
testLeftSiblingMergeStrategy

- A função de teste testLeftSiblingMergeStrategy()
  - Cria páginas cheias
  - Remove registros de todas as páginas, processando da esquerda para a direita, deixando-as deficientes
  - Insere registros maiores do que todos os existentes, o que leva a novas páginas



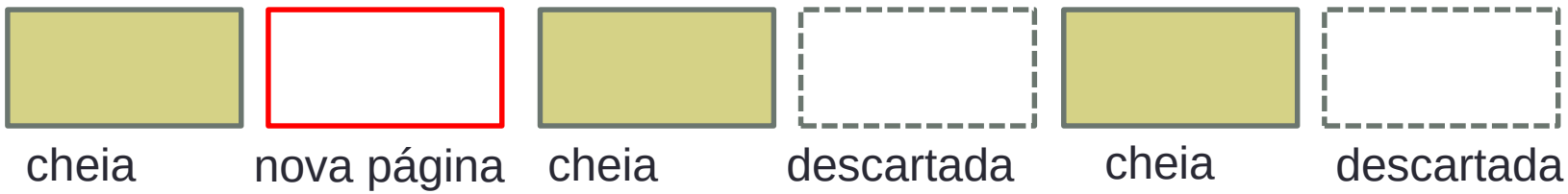
testLeftSiblingMergeStrategy

- Se a estratégia de mescla for implementada, uma página deficiente será mesclada com a sua vizinha da direita
  - E será descartada



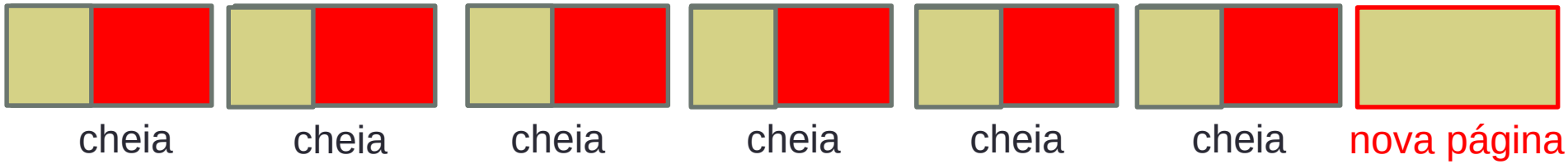
testLeftSiblingMergeStrategy

- Quando for necessário gerar novas páginas
  - serão usados os espaços descartados em vez de criar novas páginas no fim do arquivo
  - O que evita que o arquivo fique maior



testLeftSiblingMergeStrategy

- Caso contrário, uma página será criada no fim do arquivo, fazendo com que ele fique maior



Testes

- As três funções de teste realizam verificações que conferem se os registros manuseados foram devidamente inseridos e removidos da árvore
  - Uma mensagem de erro aparece em caso de falha.
- As funções também exibem quantas páginas foram criadas durante o processamento
  - Quanto mais páginas, maior é o arquivo gerado
- Use esses retornos para verificar se a implementação está correta

Testes

- A tabela abaixo compara o número de páginas geradas dependendo da estratégia de balanceamento usada

Caso de teste	Estrategia	Número de páginas
Borrow from sibling	Nenhuma	754
Borrow from sibling	Borrow from left	505
Borrow from sibling	Borrow from right	504
Left sibling merge	Nenhuma	2013
Left sibling merge	Merge with left sibling	1348
Right sibling merge	Nenhuma	2013
Right sibling merge	Merge with right sibling	1348

Testes

- Os números apresentados no slide anterior são atingidos quando apenas a estratégia testada está habilitada
- Para desabilitar as demais, basta modificar a condição de entrada no bloco de código
- Ex.
  - O trecho incluído em vermelho garante que a condição não seja satisfeita
    - Assim, a estratégia de empréstimo do vizinho da esquerda não é utilizada

```
if (ln.leftSibling != null
                    && ln.leftSibling.getParentID() == ln.getParentID()
                    && ln.leftSibling.isLendable() && 1==0) {


…
}
```

Testes

- Para garantir que cada estratégia foi bem implementada
  - desabilite as outras três no momento do teste

- É o que eu farei durante a correção

```
if (ln.leftSibling != null
                    && ln.leftSibling.getParentID() == ln.getParentID()
                    && ln.leftSibling.isLendable() && 1==0) {


…
}
```

Testes

- A troca da classe que implementa a Btree deve ser feita dentro da classe BTreeTable

```
public BTreeTable(String folder, String name, int pageSize, boolean override)
throws Exception {
        …
        if (lru!=null)
            tree = new bplustreeFileXXX(5, 7, lru, keyPrototype, valuePrototype);
        else tree = new bplustreeFileXXX(5, 7, p, keyPrototype, valuePrototype);
    }
```

Avaliação

- Os principais aspectos que serão analisados para a avaliação do trabalho são
  - O código deve estar funcional
  - As funções complementares(caso hajam) foram criadas como privadas
  - O arquivo correto foi enviado (.java)
  - O pacote da classe foi especificado da forma correta
  - A quantidade de páginas gerada foi reduzido de forma esperada
  - Os dados armazenados são corretos

- Obs. geral: cada caso não implementado ou com erros leva a um desconto máximo de 25% na nota

Entrega

- Entrega pelo moodle
- Não entregue o projeto inteiro
  - Apenas a classe java solicitada
- O trabalho é **individual**
  - O compartilhamento de código entre alunos leva à anulação da nota
- Use a última versão do código disponível no moodle

Entrega

- A nota máxima possível depende do dia em que for feita a entrega

Prazo	Nota máxima
08/09 23h59min (sexta)	100%
09/09 23h59min (sábado)	80%
10/09 23h59min (domingo)	60%
11/09 23h59min (segunda)	40%

- Entregas feitas após o dia 11/09 não serão avaliadas