Data Structures & Algorithms

Alexandr Klimchik
Head of Intelligent Robotic Systems Laboratory
Innopolis University

Based on the material from Adil M. Khan

Previous Lecture

- Balanced Binary Search Trees
- AVL Trees
- Insertions and Deletions in AVL Trees
- Height of AVL Trees

Objectives

- 2-3-4 Trees
- Insertions and Deletions in 2-3-4 Trees
- B-Trees
- Insertions and Deletions in B-Trees

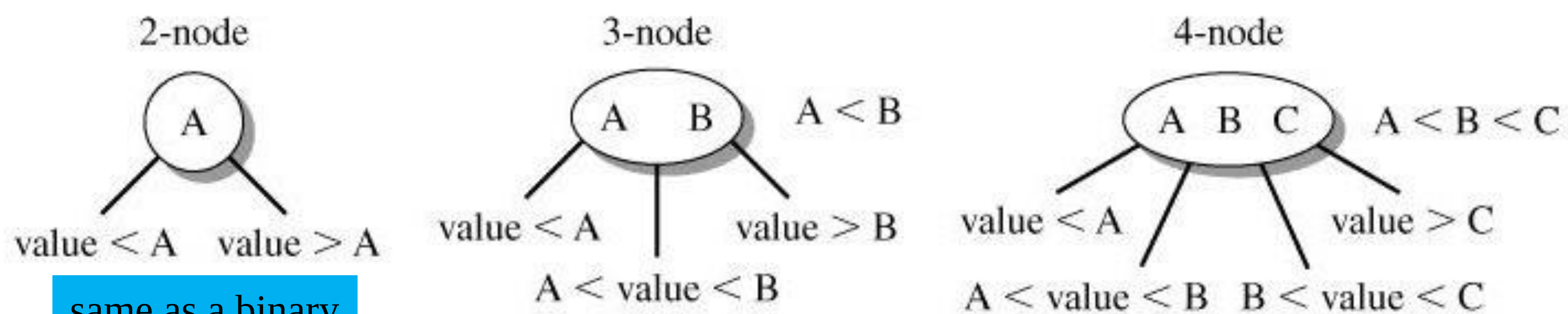
2-3-4 Trees

Chapter 11.5 - Data Structures and Algorithms in Java (6nd Edition) by Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser, 2014

2-3-4 Trees

The numbers refer to the maximum number of branches that can leave the node

- In a 2-3-4 tree:
  - a 2-node has 1 value and a max of 2 children
  - a 3-node has 2 values and a max of 3 children
  - a 4-node has 3 values and a max of 4 children



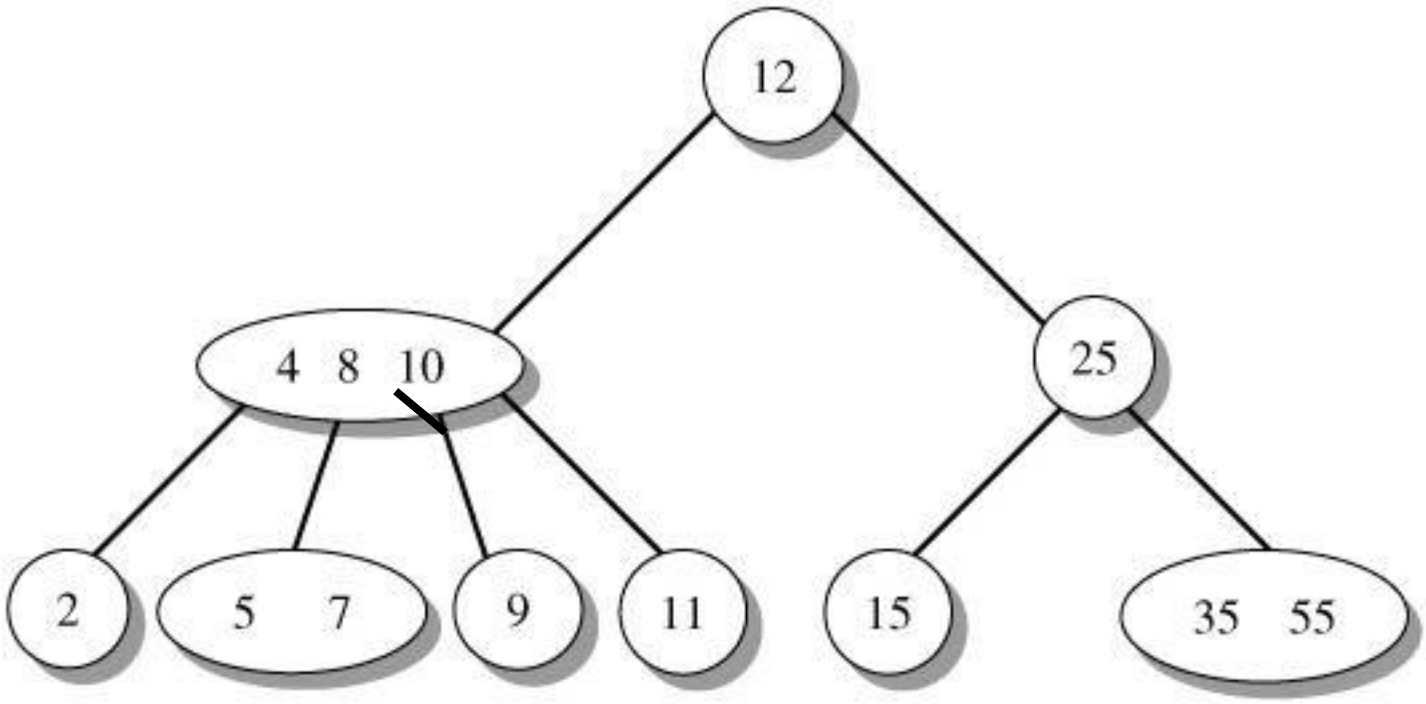
same as a binary tree node

Searching a 2-3-4 Tree

- To find an item:
  - start at the root and compare the item with <u>all</u> the values in the node;
  - if there's no match, move down to the appropriate subtree;
  - repeat until you find a match or reach an empty subtree

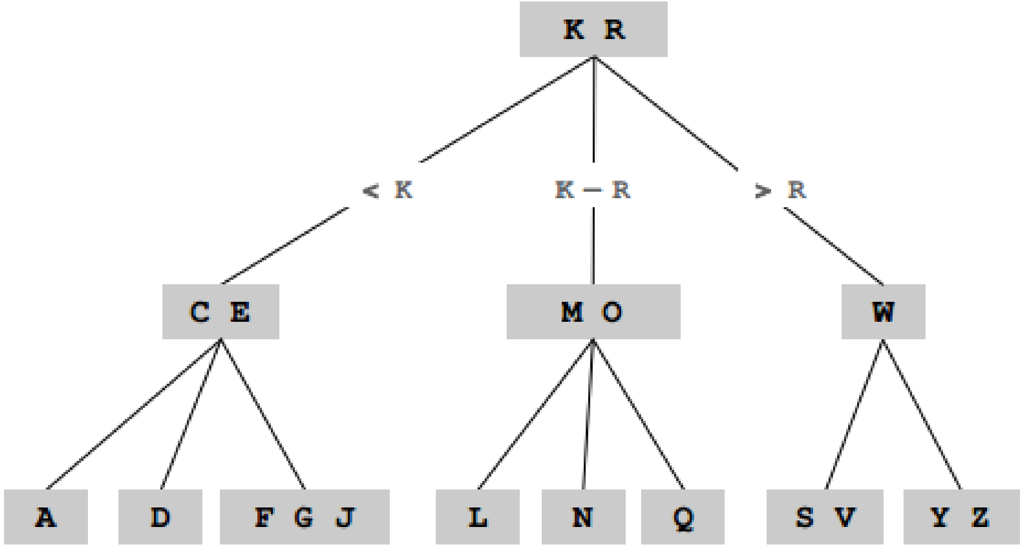
Search Example

Try finding 9 and 30



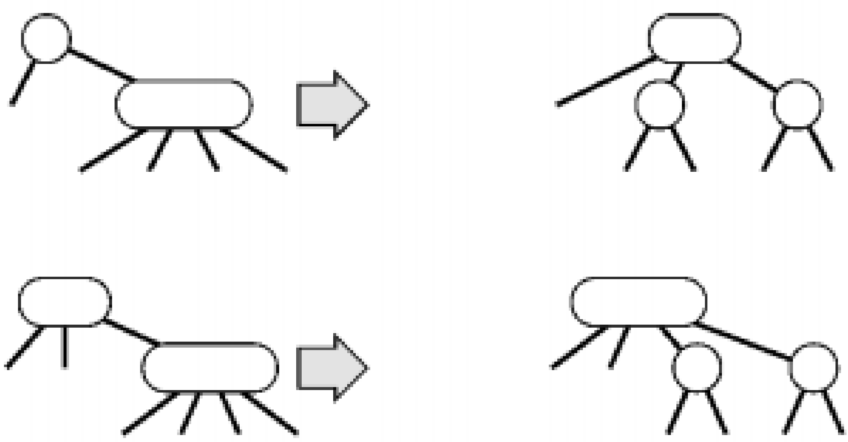
Inserting into a 2-3-4 Tree

- Search to the bottom for an insertion node
  - 2-node at bottom: convert to 3-node
  - 3-node at bottom: convert to 4-node
  - 4-node at bottom: ??



Splitting 4-nodes

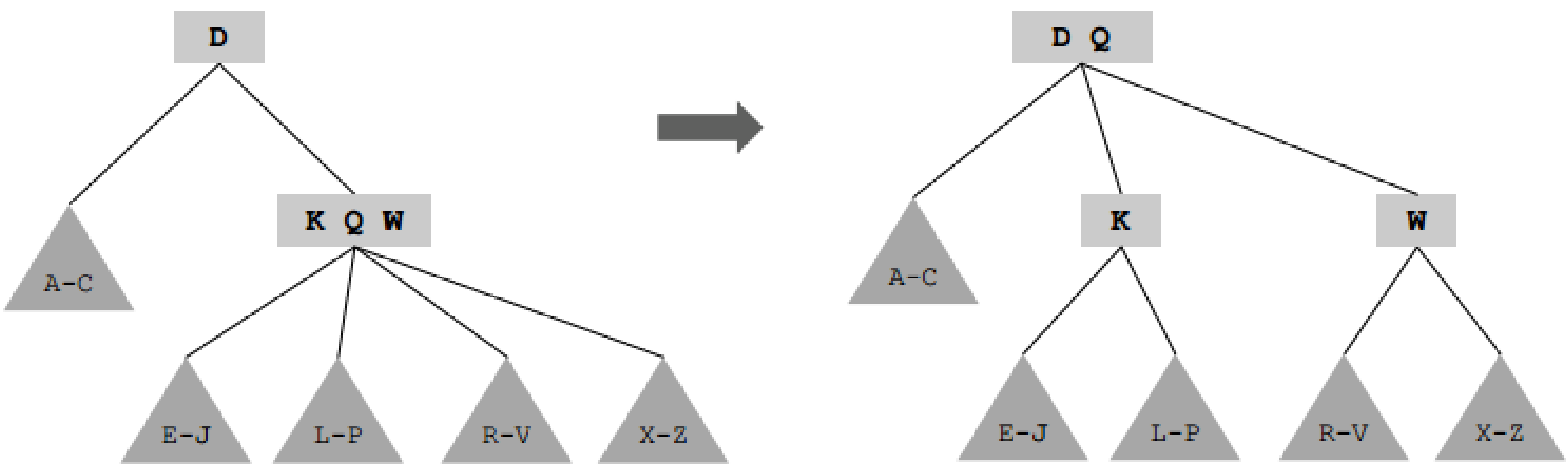
- Transform tree on the way down:
  - ensures last node is not a 4-node
  - local transformation to split a 4-node



Insertion at the bottom is now easy since it's not a 4-node

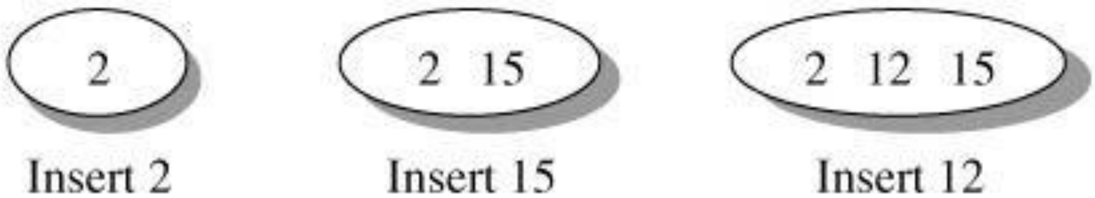
Example

- To split a 4-node. move middle value up.

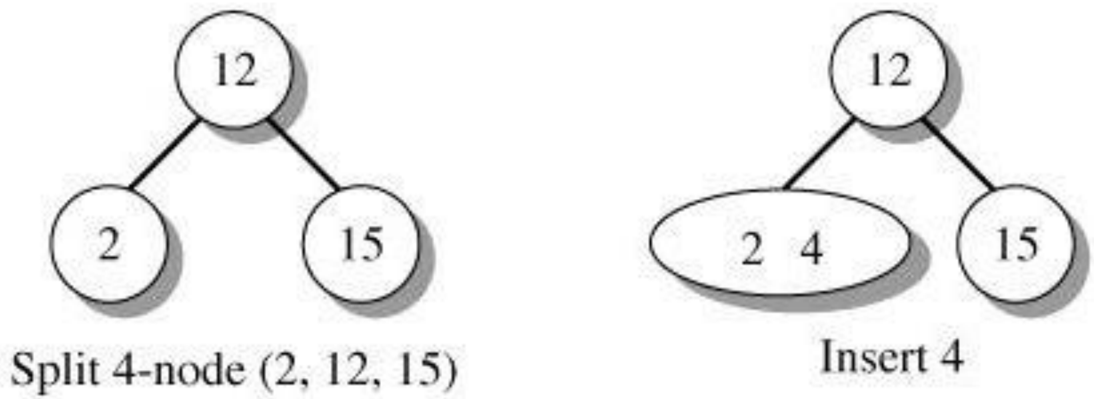


Building

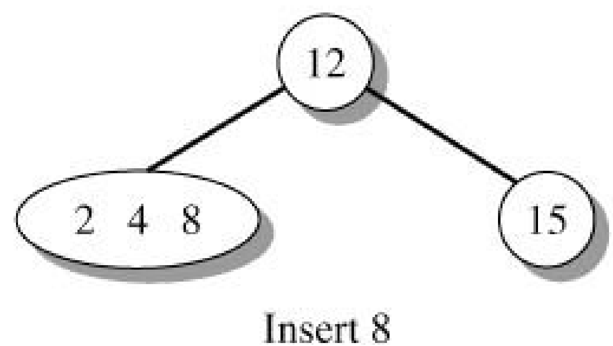
This 4-node will be split during the next insertion.



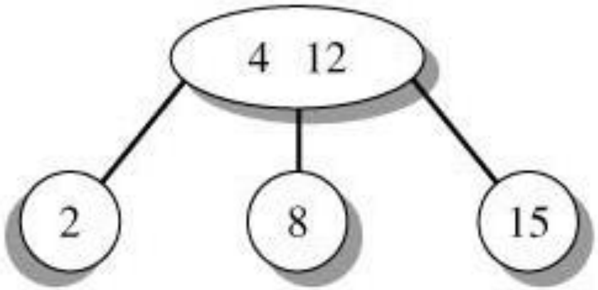
insert 4



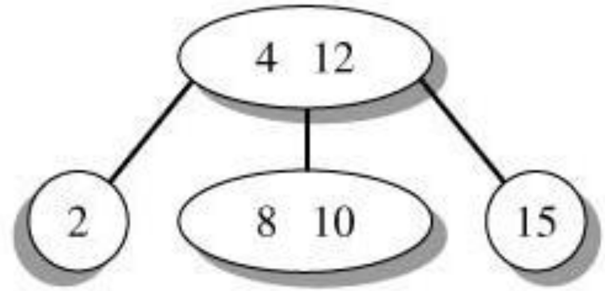
This 4-node will be split during the next insertion.



insert 10

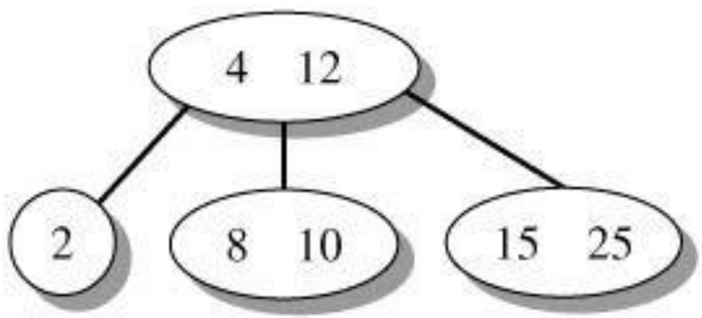


Split 4-node (2, 4, 8)

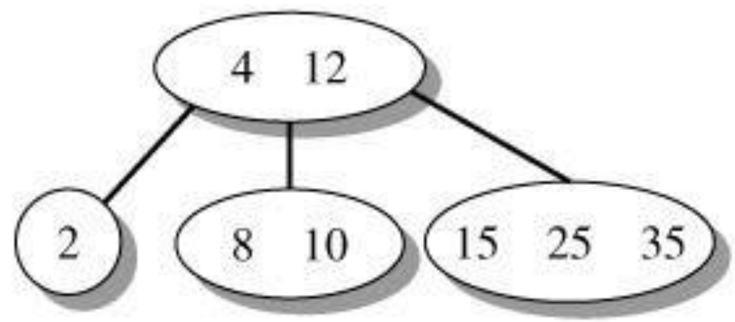


Insert 10

Insertions happen at the bottom.



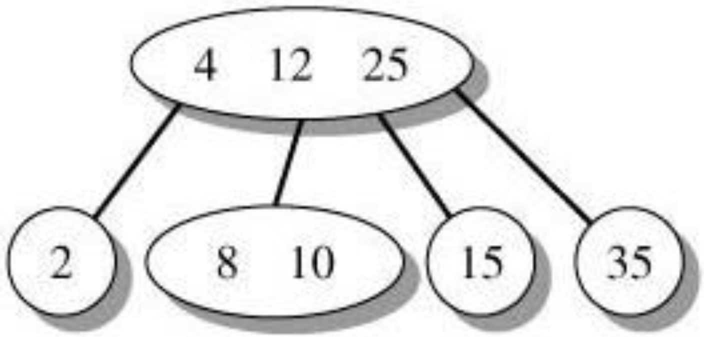
Insert 25



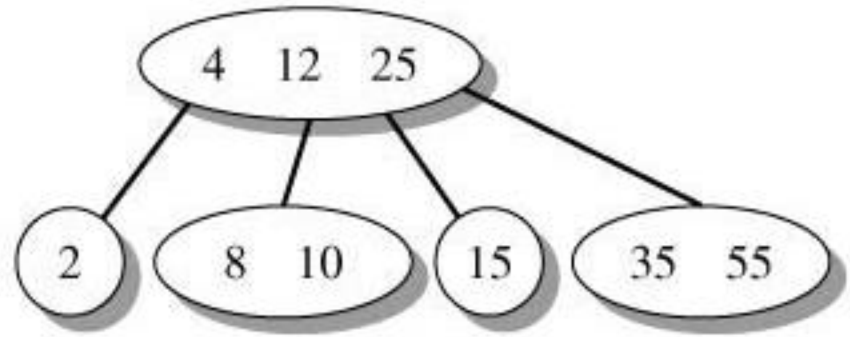
Insert 35

This 4-node will be split during
the next insertion.

insert 55



Split 4-node (15, 25, 35)

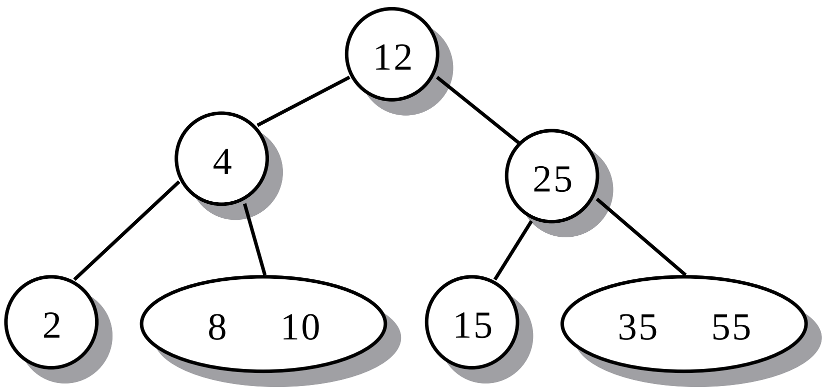


Insert 55

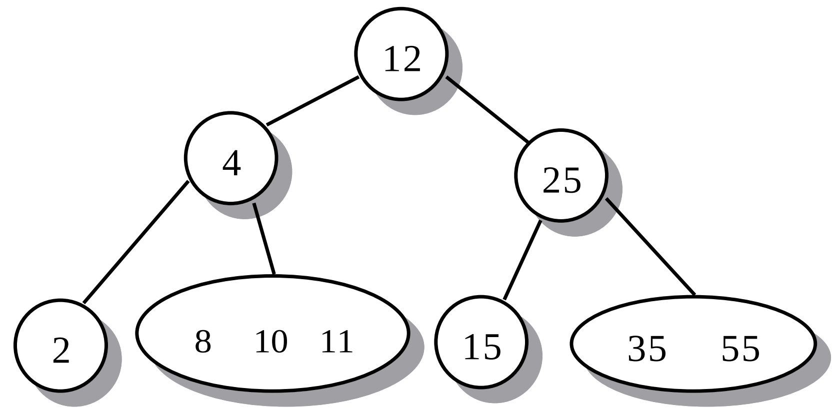
The insertion point is at level 1, so the new 4-node at level 0 is not split during this insertion.

continued

insert 11



Split 4-node (4, 12, 25)



Insert 11

This 4-node will be split during the next insertion.

Efficiency of 2-3-4 Trees

fast!

- Searching for an item in a 2-3-4 tree with n elements:
  - the max number of nodes visited during the search is $\mathrm{int}(\log_2 n) + 1$
- Inserting an element into a 2-3-4 tree:
  - requires splitting no more than $\mathrm{int}(\log_2 n) + 1$ 4-nodes
    - normally requires far fewer splits

Drawbacks of 2-3-4 Trees

- Since any node may become a 4-node, then all nodes must have space for 3 values and 4 links
  - but most nodes are not 4-nodes
  - lots of wasted memory, unless impl. is fancier

- Complex nodes and links
  - slower to process than binary search trees

B Tree

Chapter 18 - Introduction to Algorithms (3nd Edition) by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, 2009

Two Types of Memory

- Main memory (RAM)
- External storage: hard disk
- Different considerations are important when designing algorithms and data structures for main vs. secondary memory

External Storage

- So far, we analyzed data structures assuming that all data is kept in main memory

- If data is kept in secondary storage, we should take into account disk access time

External Storage

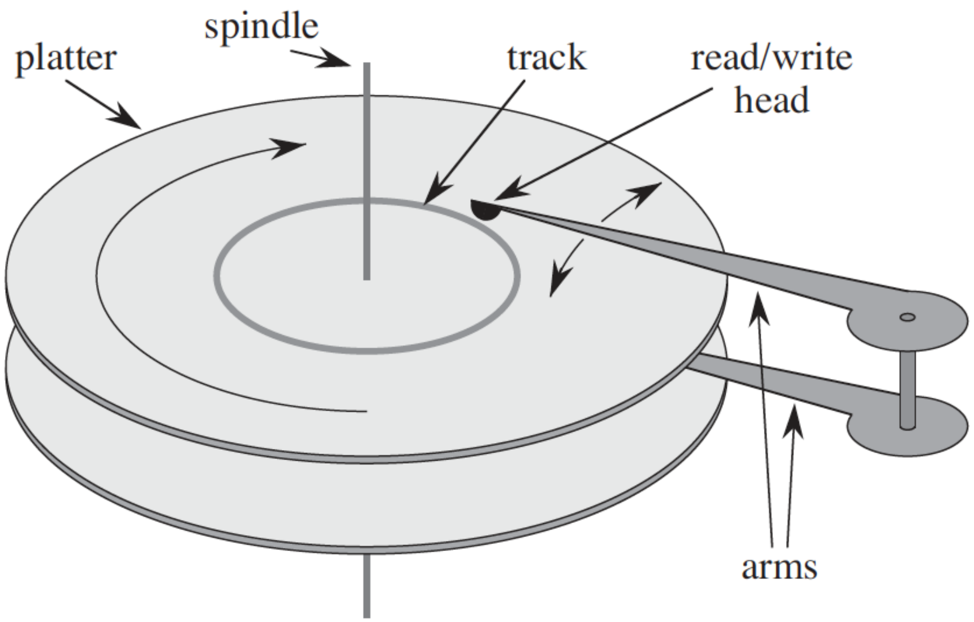


Figure 18.2 A typical disk drive. It comprises one or more platters (two platters are shown here) that rotate around a spindle. Each platter is read and written with a head at the end of an arm. Arms rotate around a common pivot axis. A track is the surface that passes beneath the read/write head when the head is stationary.

"*Introduction to Algorithms*", by Cormen

Main Principles

- Data is stored on disks in chunks (pages, blocks
- Disk drive reads or writes a minimum one page at a time
- Thus, we should
  - Minimize disk accesses
  - Read and write multiple pages

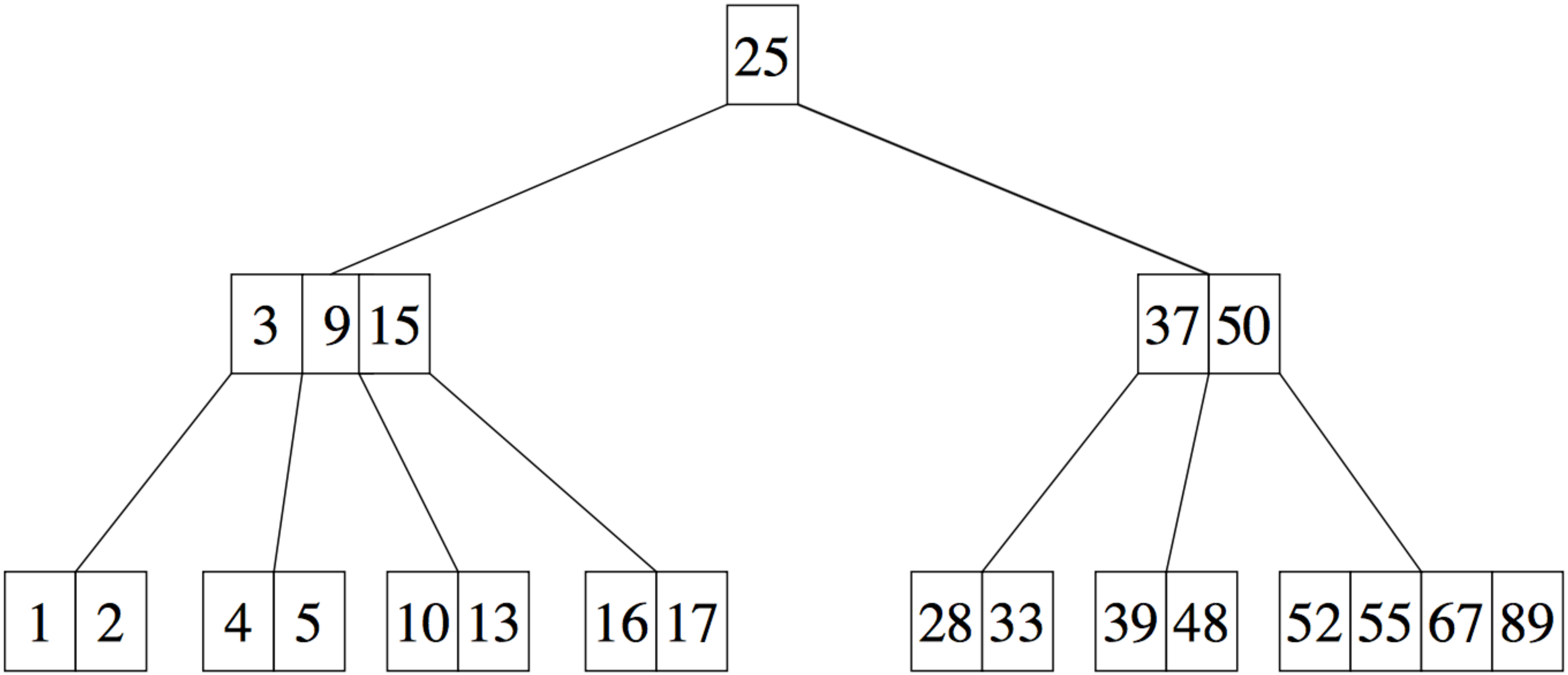
B-Tree

- A good data structure for external storage

- A B-tree is a balanced tree in which balanced is achieved by permitting the nodes to have multiple keys and more than two children.

- If an internal B-tree node x contains $x.n$ keys then x has $x.n + 1$ children.

Example of B-Tree



Example of B-Tree

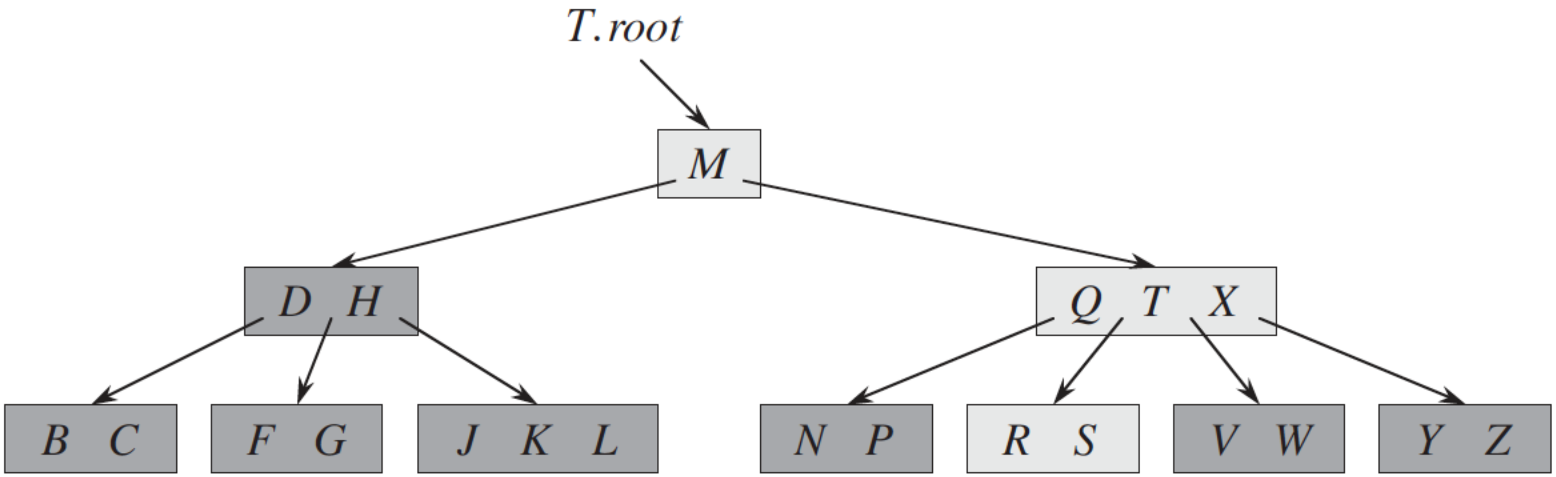


Figure 18.1 A B-tree whose keys are the consonants of English. An internal node x containing $x.n$ keys has $x.n + 1$ children. All leaves are at the same depth in the tree. The lightly shaded nodes are examined in a search for the letter R.

"*Introduction to Algorithms*", by Cormen

Model for Disk Operations

x = a pointer to some object
DISK-READ(x)
operations that access and/or modify the attributes of x
DISK-WRITE(x) // omitted if no attributes of x were changed
other operations that access but do not modify attributes of x

"*Introduction to Algorithms*", by Cormen

Another example of B-Tree

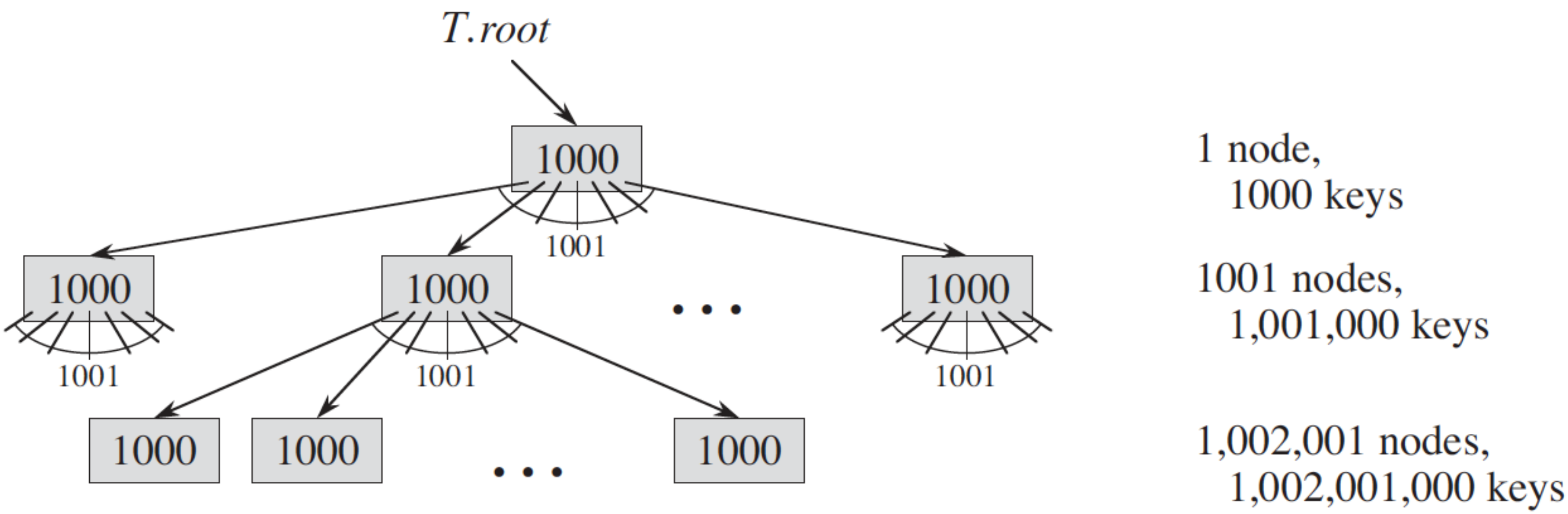


Figure 18.3 A B-tree of height 2 containing over one billion keys. Shown inside each node x is $x.n$, the number of keys in x. Each internal node and leaf contains 1000 keys. This B-tree has 1001 nodes at depth 1 and over one million leaves at depth 2.

"*Introduction to Algorithms*", by Cormen

Properties of B-Tree

- A B-Tree T is a rooted tree having the following properties:

1. Every node x has the following fields:

   a. $x.n$ the number of keys currently stored in node x
   b. The n keys themselves, stored in non-decreasing order, so that $x.key_1 \leq x.key_2 \ldots\ldots \leq x.key_{n-1} \leq x.key_n$.
   c. $x.leaf$, a Boolean value that is TRUE if x is a leaf and FALSE if x is an internal node.

Properties of B-Tree

2. Each internal node x also contains $x.n + 1$ pointers (Children)

3. All leaves have the same depth, which is the tree's height h.

Properties of B-Tree (cont.)

4. Lower and upper bounds on the no. of keys a node can contains: can be expressed in terms of a fixed integer $t \geq 2$ called the minimum degree of B-Tree.

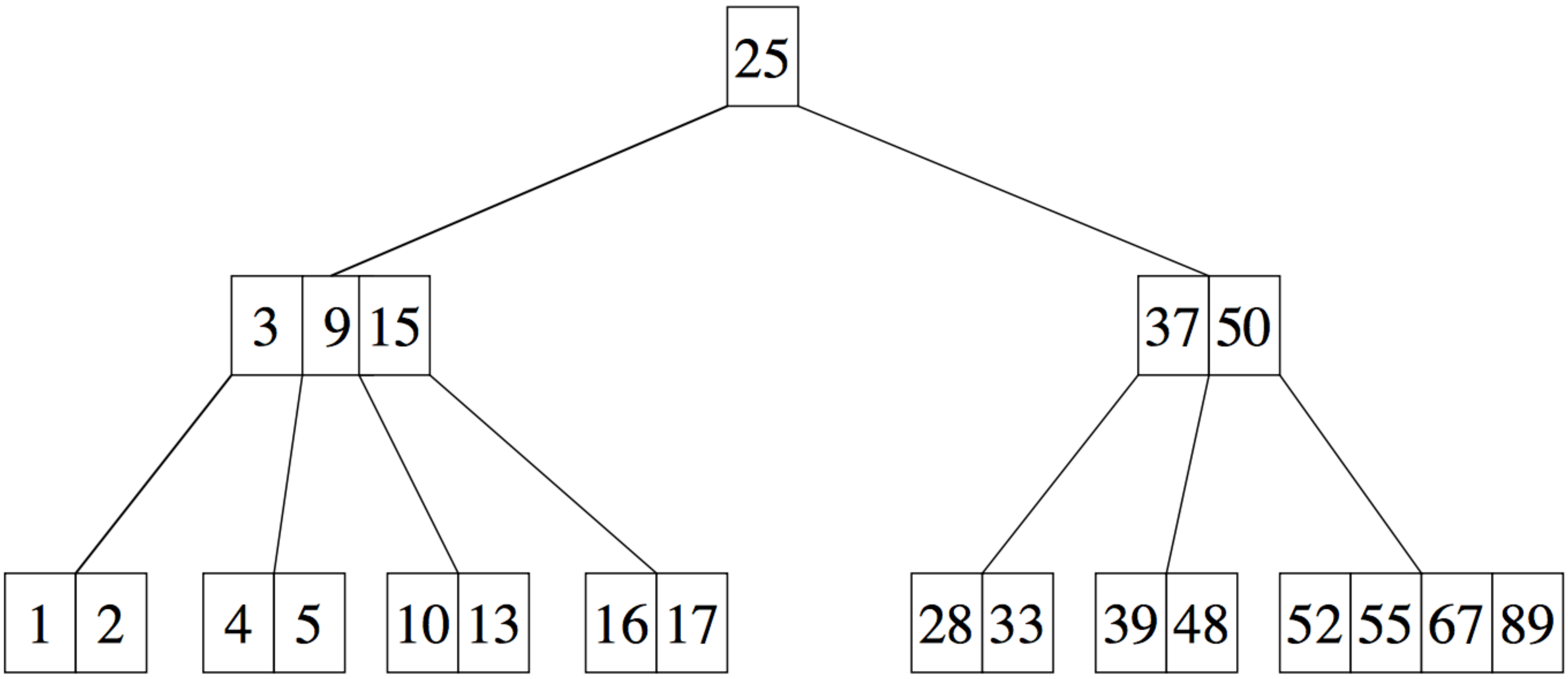
Properties of B-Tree (cont.)

4. Lower and upper bounds on the no. of keys a node can contains: can be expressed in terms of a fixed integer $t \geq 2$ called the minimum degree of B-Tree.
   - Every node other than the root must have at least $t - 1$ keys. Every node other than root has at least t children. if the tree is non empty the root must have at least one key.

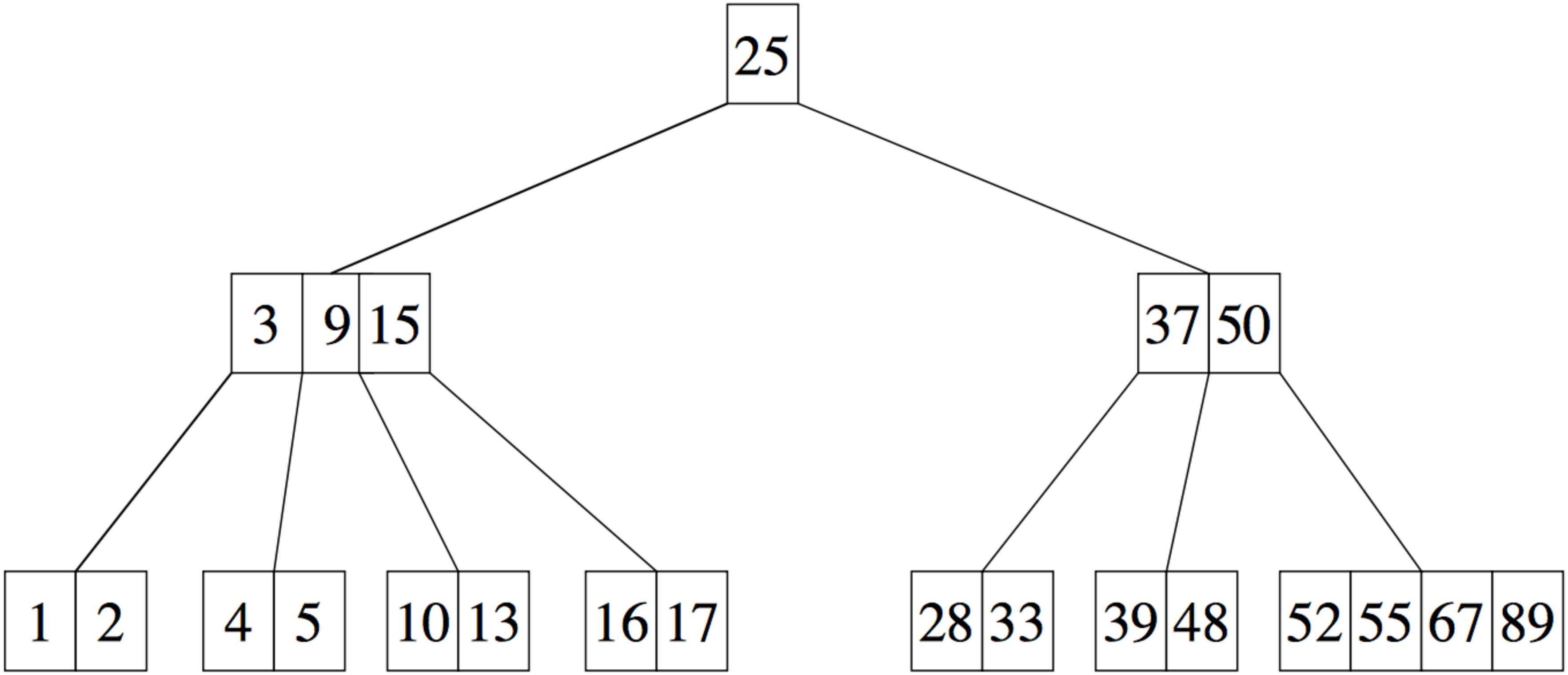
Properties of B-Tree (cont.)

4. Lower and upper bounds on the no. of keys a node can contains: can be expressed in terms of a fixed integer $t \geq 2$ called the minimum degree of B-Tree.
   - Every node other than the root must have at least $t-1$ keys. Every node other than root has at least t children. if the tree is non empty the root must have at least one key
   - Every node can contain at most $2t-1$ keys. Therefore, an internal node can have at most $2t$ children. We say that a node is full if it contains

B-Tree of degree 3



B-Tree of degree 3



We can also say, “It’s a B-Tree of order 5”

Height of B-Tree

If $n \geq 1$, then for any n-key B-tree T of height h and minimum degree $t \geq 2$,

$$h \leq \log_t \frac{n+1}{2} .$$

Proof on whiteboard!

Height of B-Tree

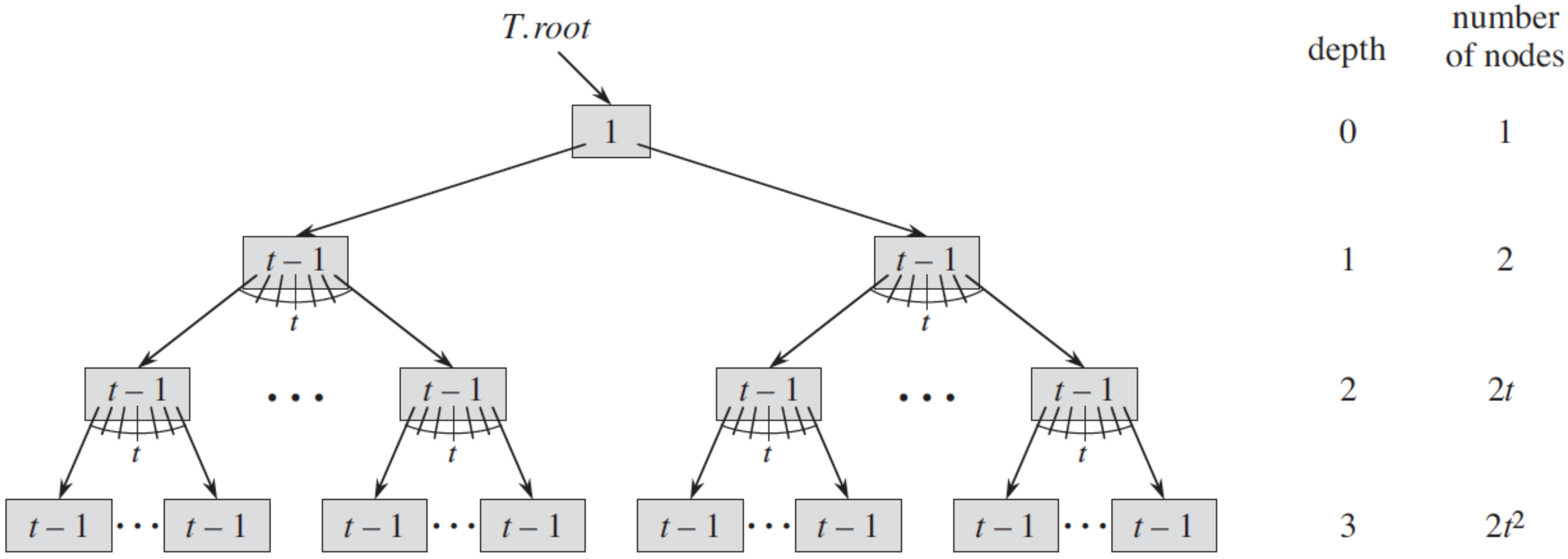


Figure 18.4 A B-tree of height 3 containing a minimum possible number of keys. Shown inside each node x is $x.n$.

"*Introduction to Algorithms*", by Cormen

Basic operation on B-Tree

- B-TREE-SEARCH :-Searching in B Tree
- B-TREE-INSERT :-Inserting key in B Tree
- B-TREE-CREATE :-Creating a B Tree
- B-TREE-DELETE :- Deleting a key from B Tree

Searching a B-Tree

```
B-TREE-SEARCH(x, k)
i = 1
while i ≤ x.n and k > x.key_i
    i = i + 1
if i ≤ x.n and k == x.key_i
    return (x, i)
elseif x.leaf
    return NIL
else DISK-READ(x.c_i)
    return B-TREE-SEARCH(x.c_i, k)
```

“*Introduction to Algorithms*”, by Cormen

Searching a B-Tree

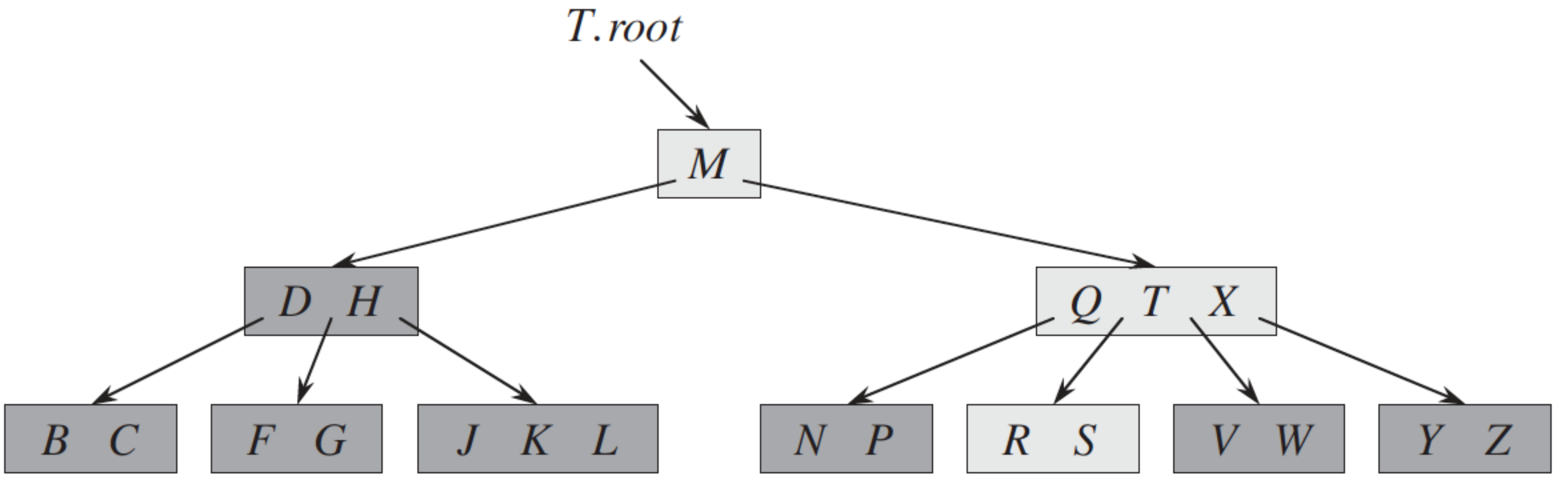
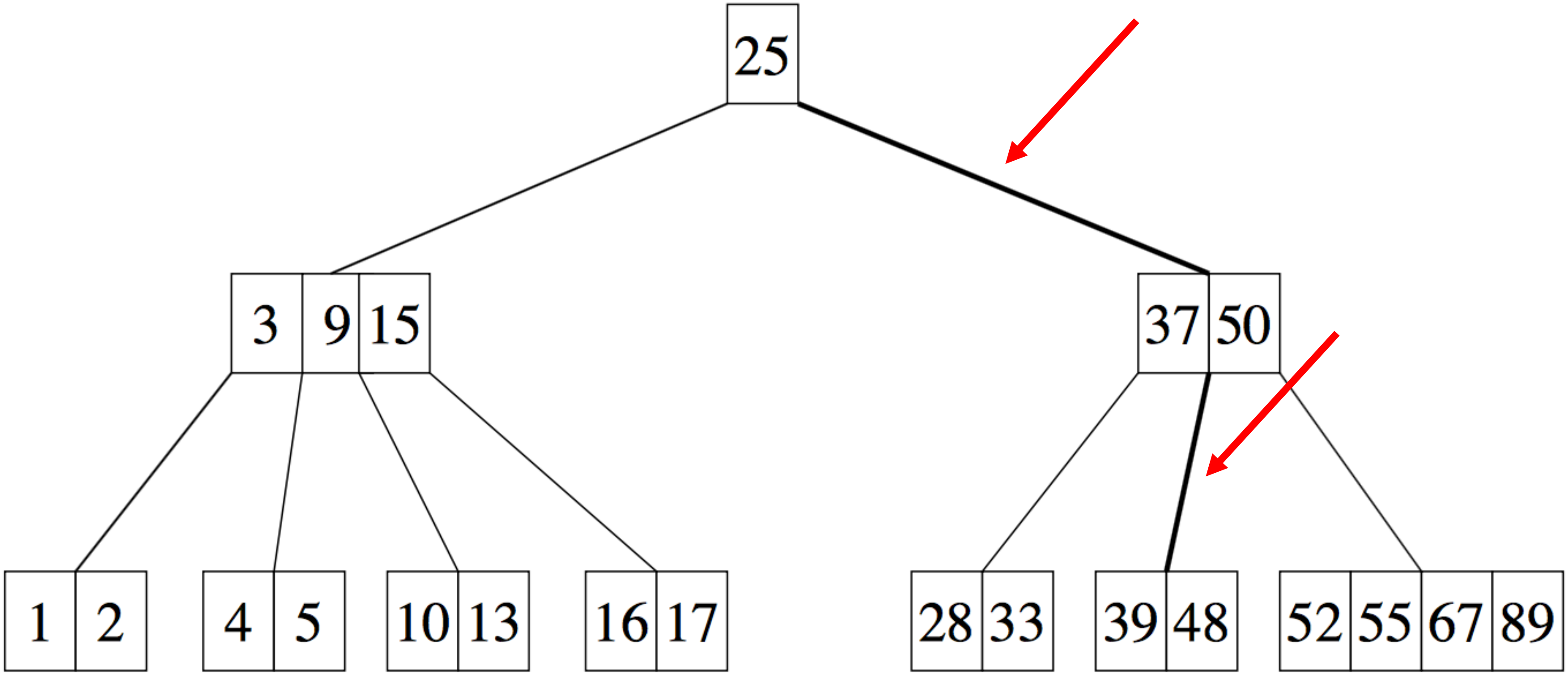


Figure 18.1 A B-tree whose keys are the consonants of English. An internal node x containing $x.n$ keys has $x.n + 1$ children. All leaves are at the same depth in the tree. The lightly shaded nodes are examined in a search for the letter R.

"*Introduction to Algorithms*", by Cormen

Searching a B-Tree



Search for a record with key 40

Creating a B-Tree

B-TREE-CREATE(T)

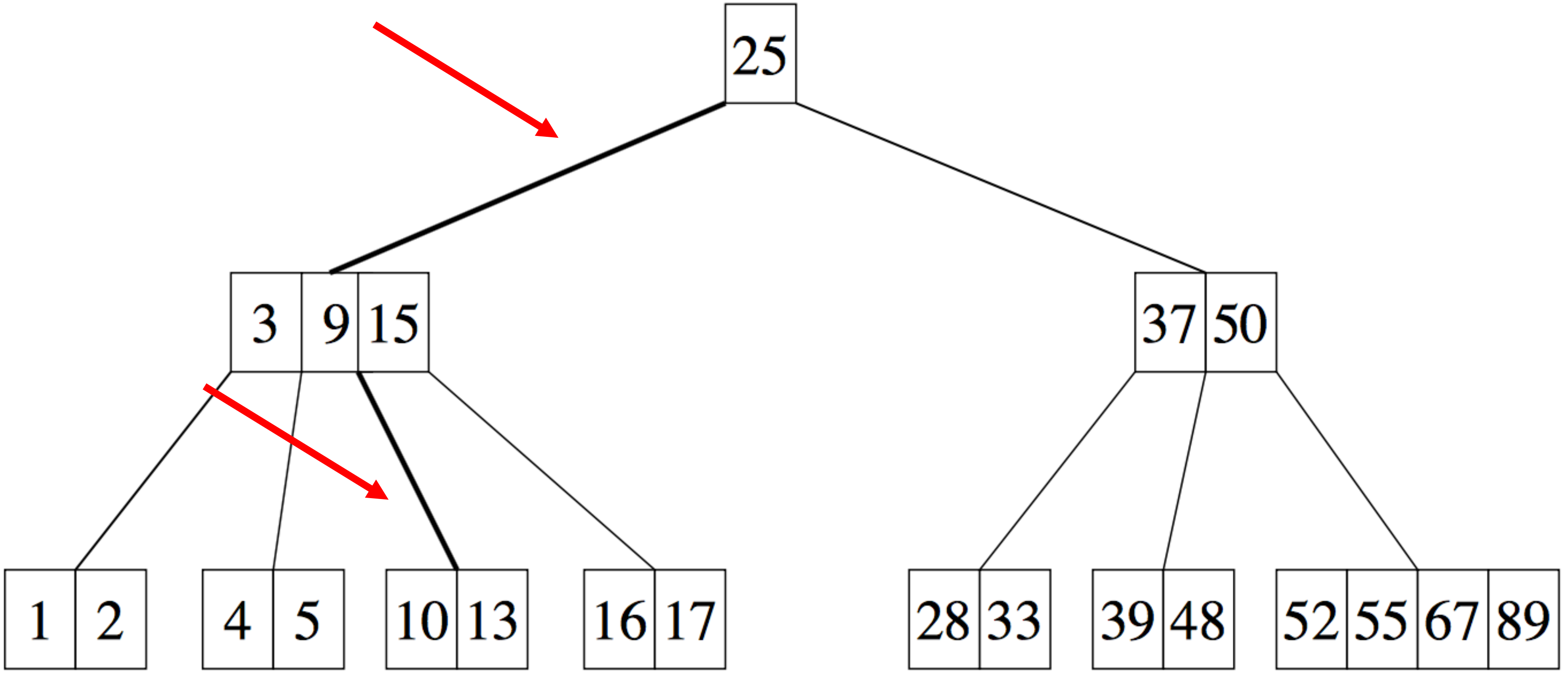
x = ALLOCATE-NODE()
$x.leaf$ = TRUE
$x.n = 0$
DISK-WRITE(x)
$T.root = x$

"*Introduction to Algorithms*", by Cormen

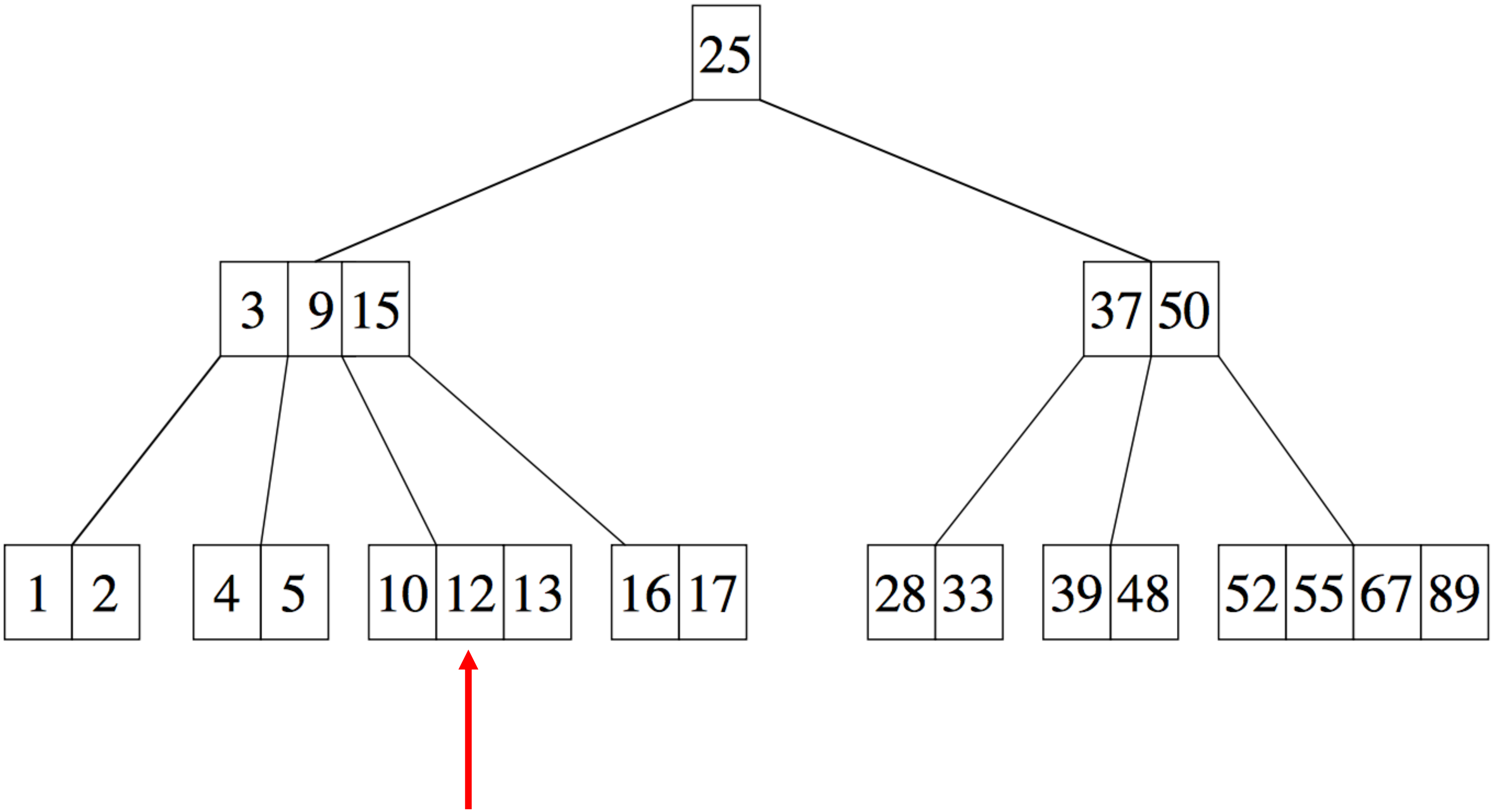
Inserting in a B-Tree

- Just like BST, find where the items has to be inserted
- If the node is not full, insert the item into the node in order
- If the node is full, it has to be split

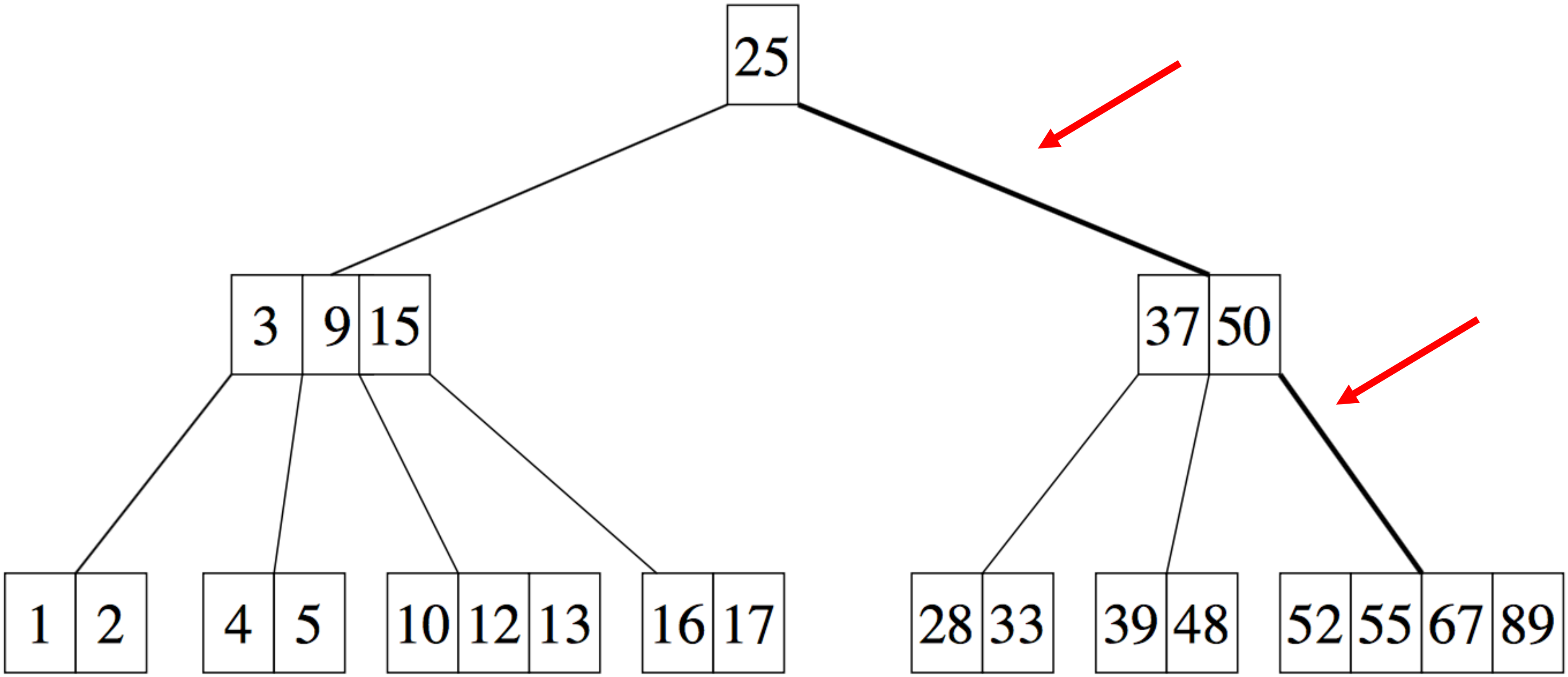
Inserting 12



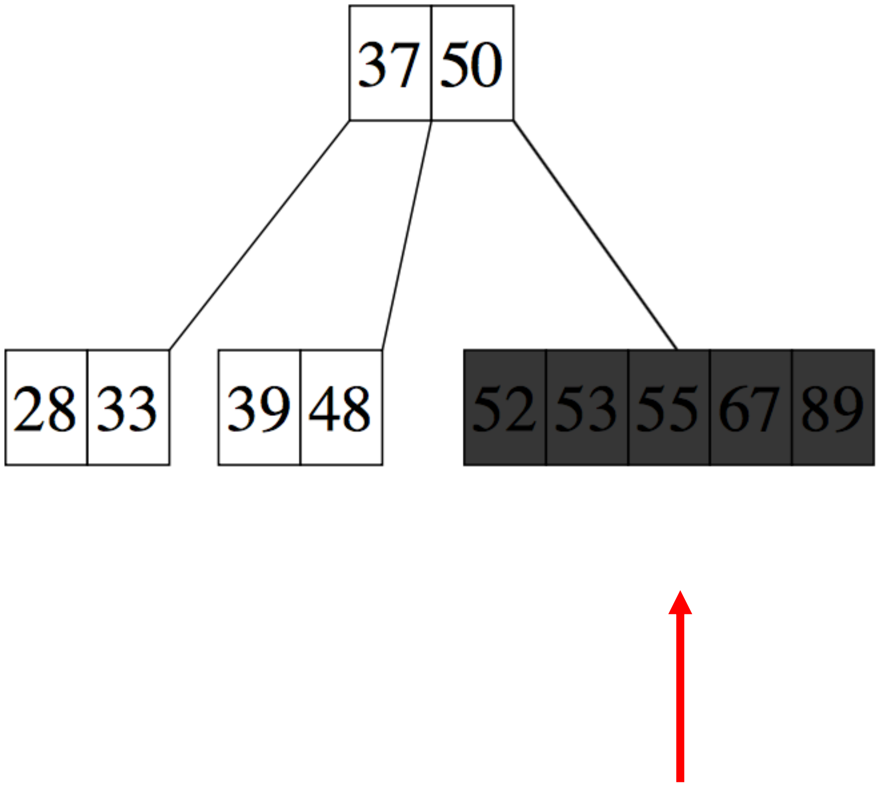
Inserting 12



Inserting 53



Inserting 53

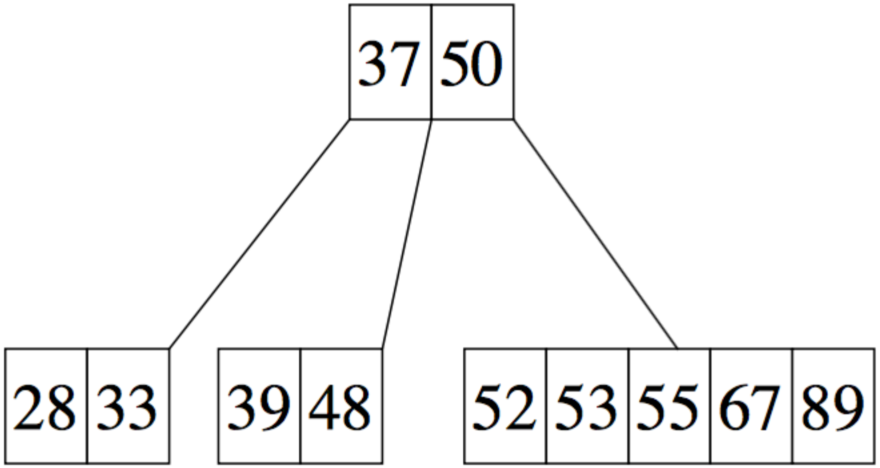


Node gets full!

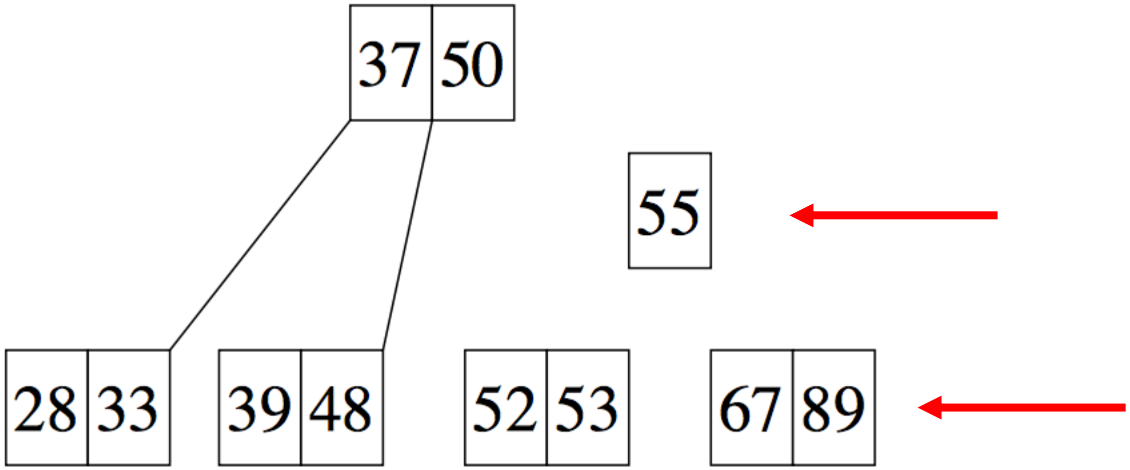
Splitting a Node

1. Find the middle value (old keys + the new key)
2. Create a new node
3. Move the records with greater key than the middle in the new node
4. Keep the records with keys smaller than the middle in the old node
5. Push the middle up into parent node

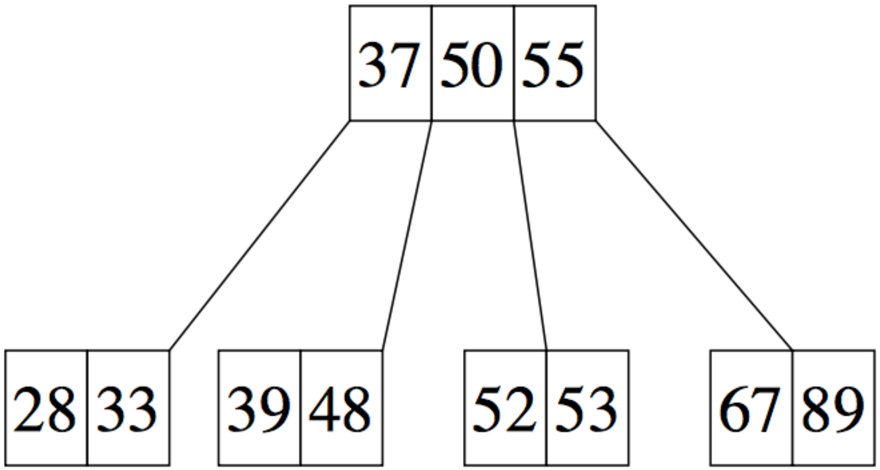
Splitting a Node



Splitting a Node



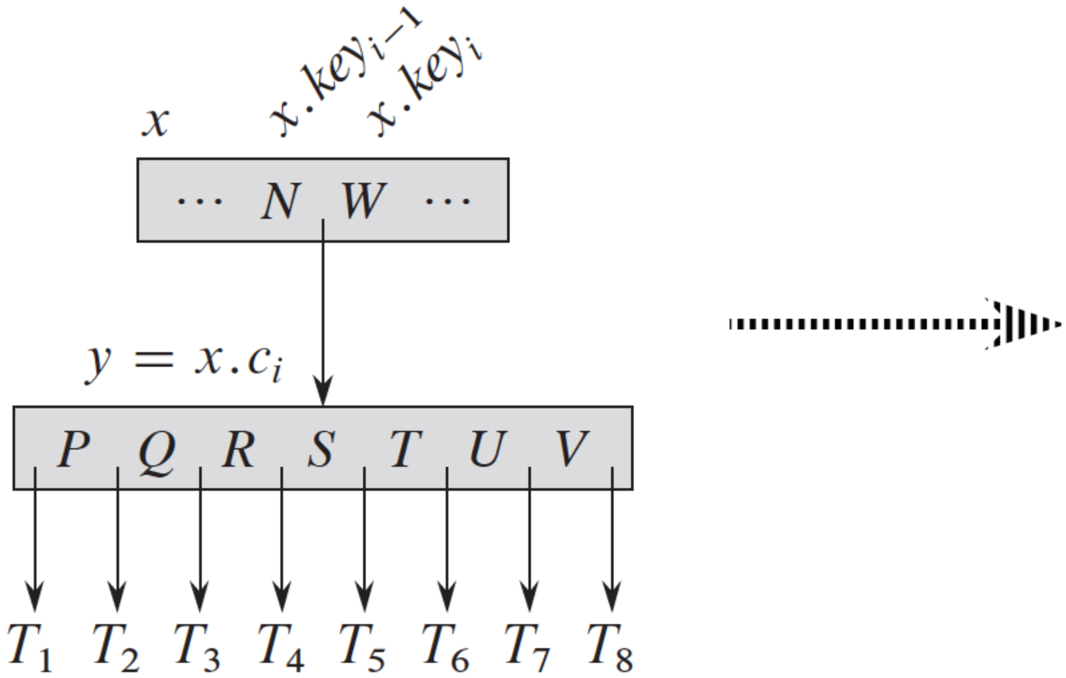
Splitting a Node



Inserting in a B-Tree

- If this makes the parent full, split the parent too and push its middle item upwards
- Continue doing this until either some space is found in an ancestor node, or a new root node is created

Another Example: Split in a B-Tree



Another Example: Split in a B-Tree

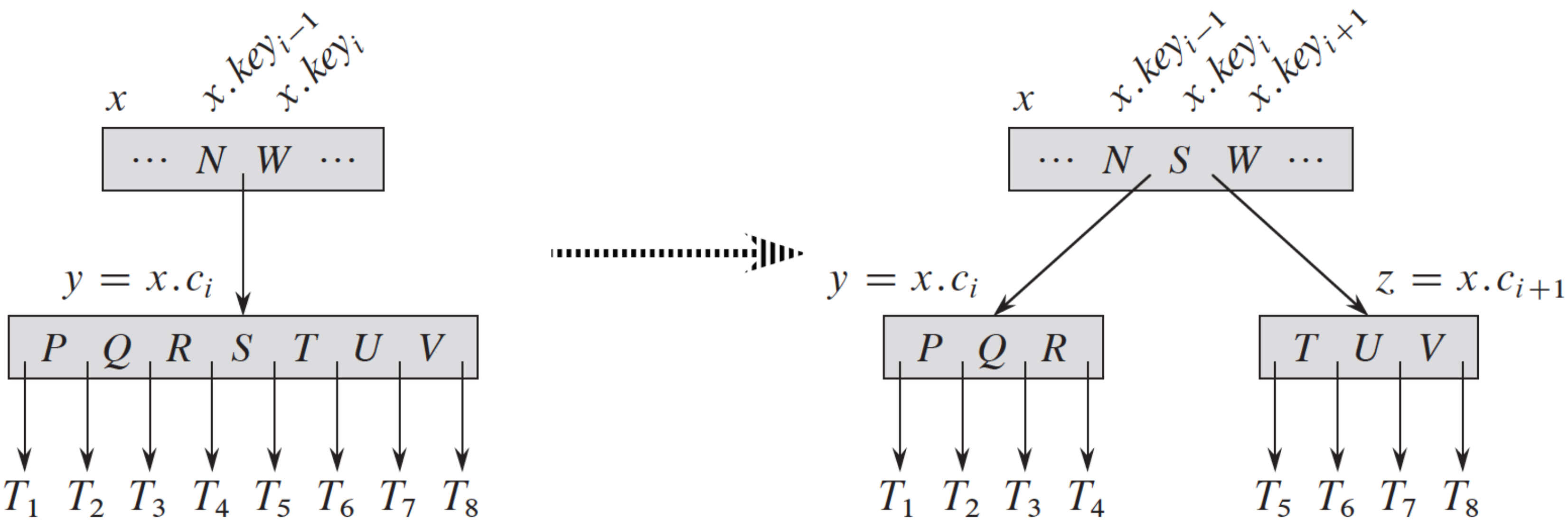


Figure 18.5 Splitting a node with $t = 4$. Node $y = x.c_i$ splits into two nodes, y and z, and the median key S of y moves up into y's parent.

"*Introduction to Algorithms*", by Cormen

Pseudocode: Split in a B-Tree

```
B-TREE-SPLIT-CHILD(x, i)
z = ALLOCATE-NODE()
y = x.c_i
z.leaf = y.leaf
z.n = t - 1
for j = 1 to t - 1
    z.key_j = y.key_{j+t}
if not y.leaf
    for j = 1 to t
        z.c_j = y.c_{j+t}
y.n = t - 1
for j = x.n + 1 downto i + 1
    x.c_{j+1} = x.c_j
x.c_{i+1} = z
for j = x.n downto i
    x.key_{j+1} = x.key_j
x.key_i = y.key_t
x.n = x.n + 1
DISK-WRITE(y)
DISK-WRITE(z)
DISK-WRITE(x)
```

"*Introduction to Algorithms*", by Cormen

Splitting the root

- Splitting the root is the only way to increase the height of a B-tree

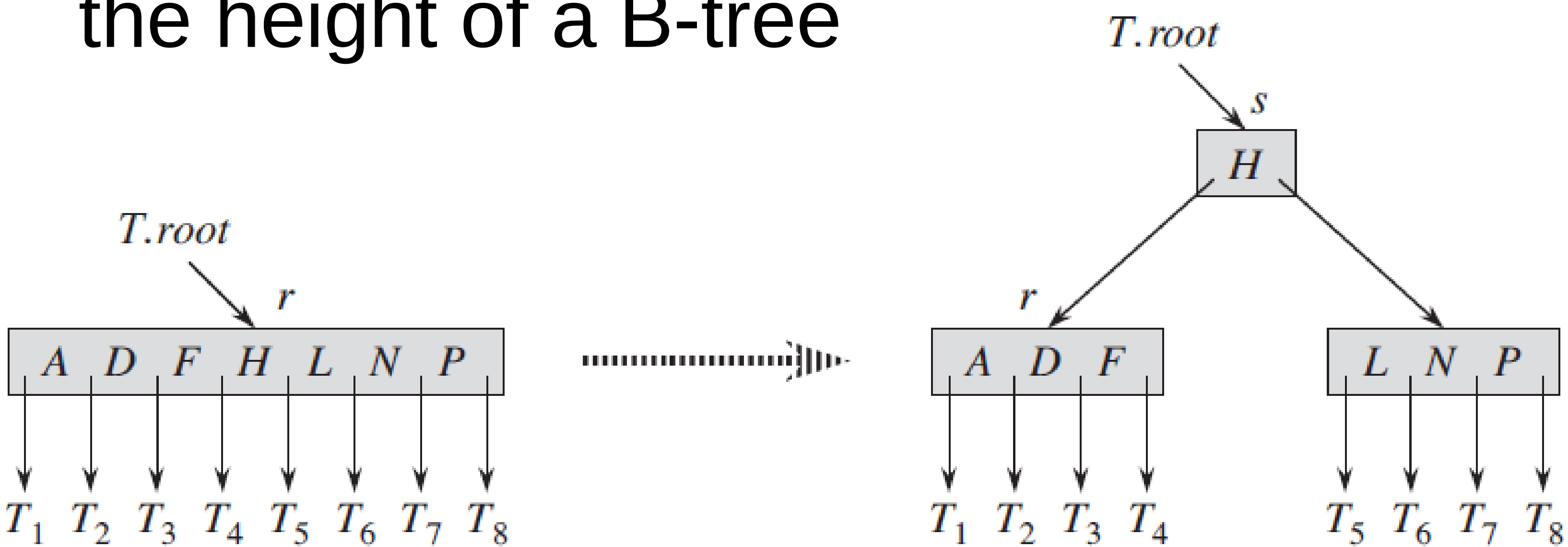
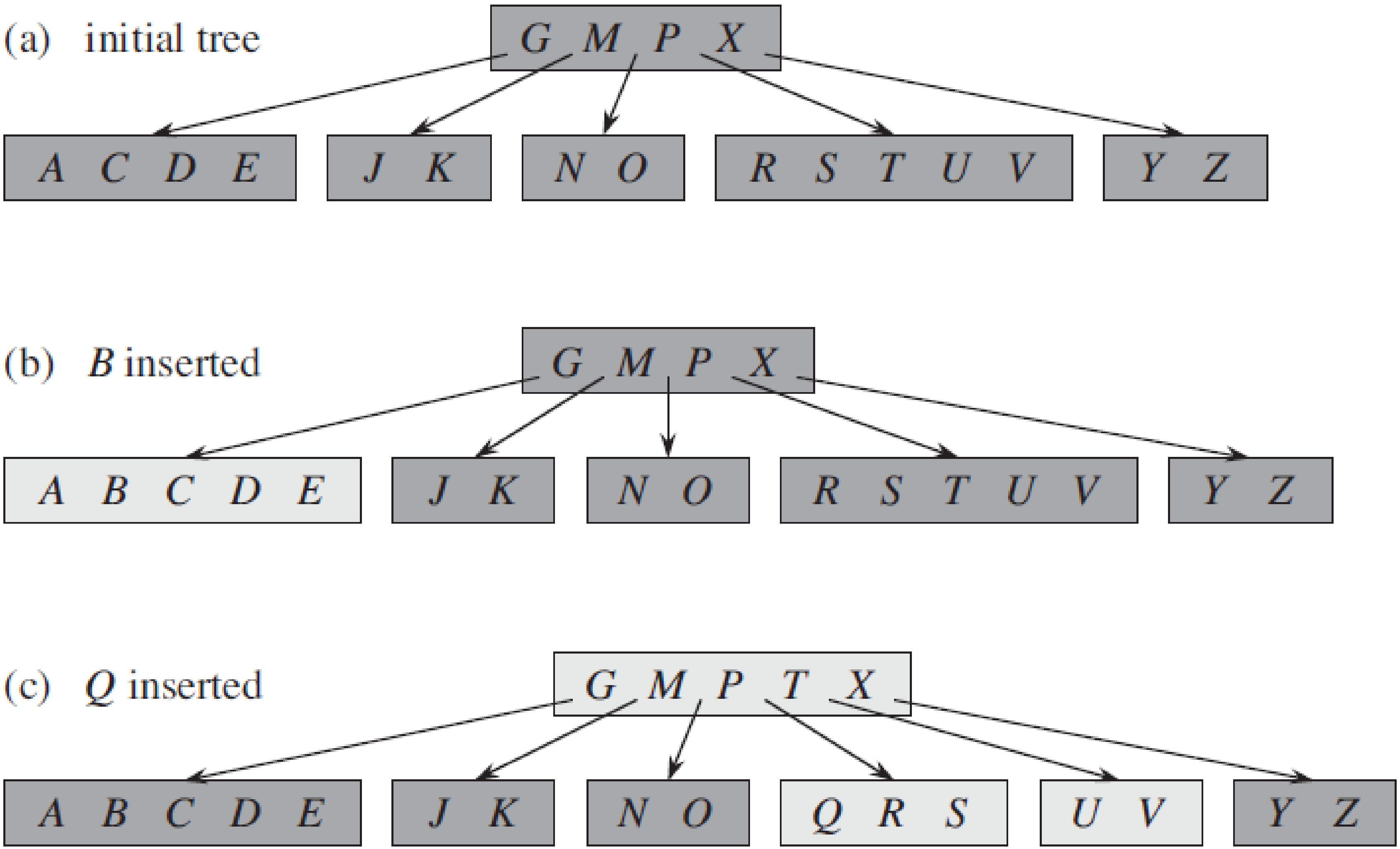


Figure 18.6 Splitting the root with $t = 4$. Root node r splits in two, and a new root node s is created. The new root contains the median key of r and has the two halves of r as children. The B-tree grows in height by one when the root is split.

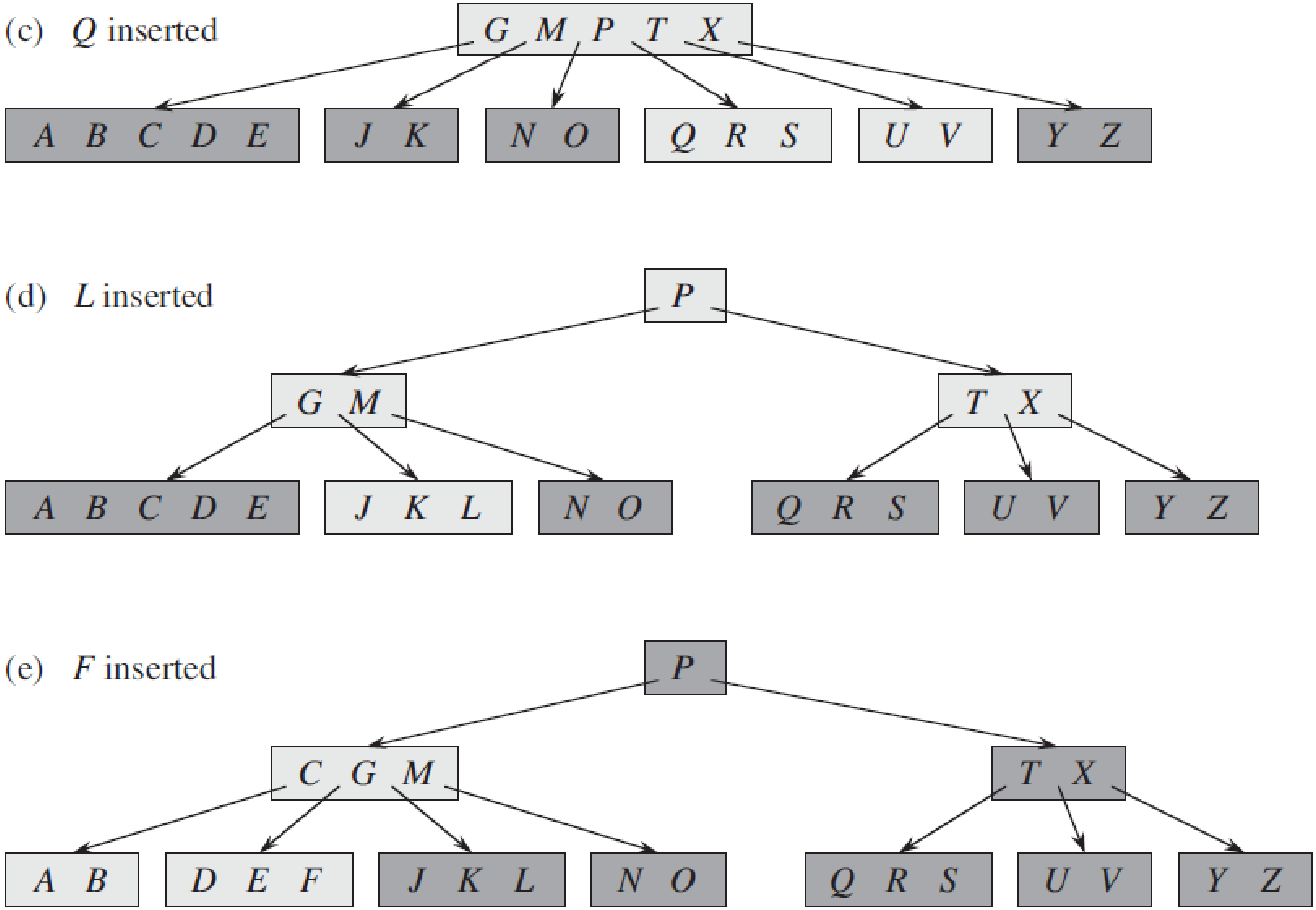
"*Introduction to Algorithms*", by Cormen

Example



"*Introduction to Algorithms*", by Cormen

Example



"*Introduction to Algorithms*", by Cormen

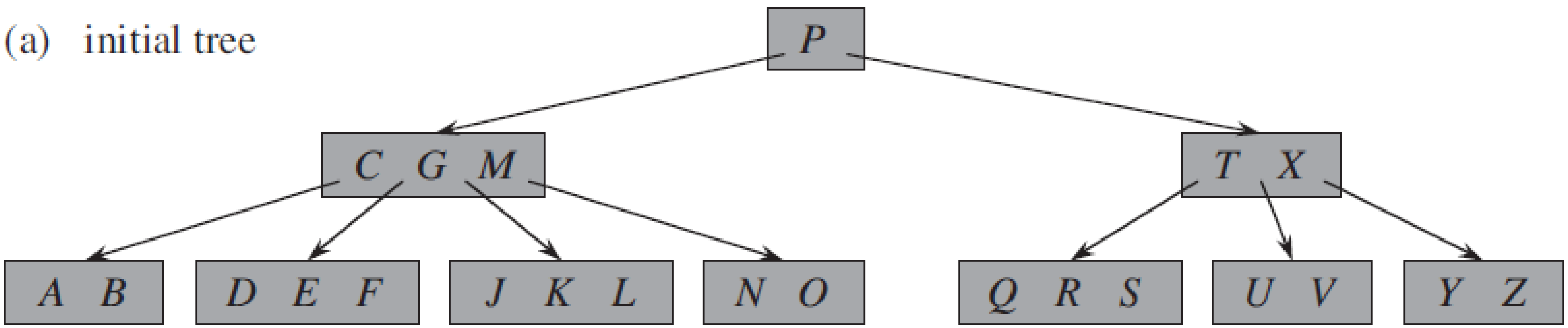
Deletion in a B-Tree

- Just like as in BST
- May cause underflow
- Depending on how many records the sibling of the node has, this can be fixed either by fusion or by transfer

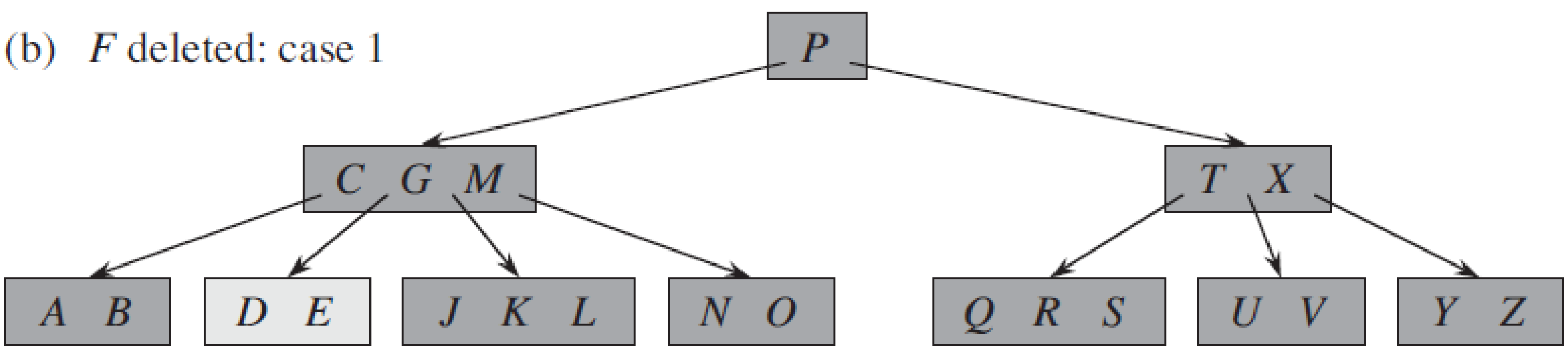
Case 1

1. If the key k is in node x and x is a leaf, delete the key k from x.

(a) initial tree



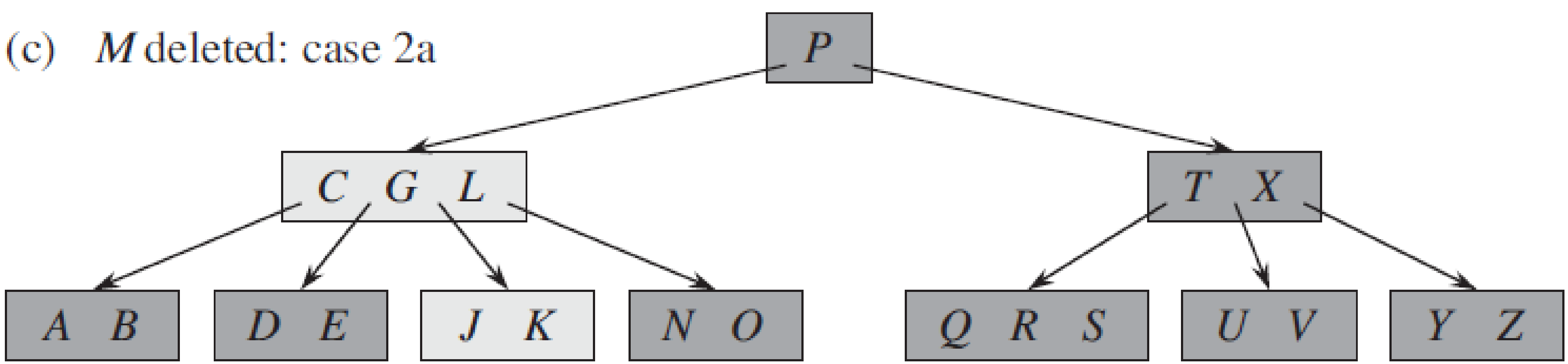
(b) *F* deleted: case 1



"*Introduction to Algorithms*", by Cormen

Case 2

2. If the key k is in node x and x is an internal node:
   - a. If *the child y that precedes k in node x has at least t keys*, then find the predecessor k’ of k in the subtree rooted at y. Recursively delete k, and replace k by k’ in x. (We can find k’ and delete it in a single downward pass.)



“*Introduction to Algorithms*”, by Cormen

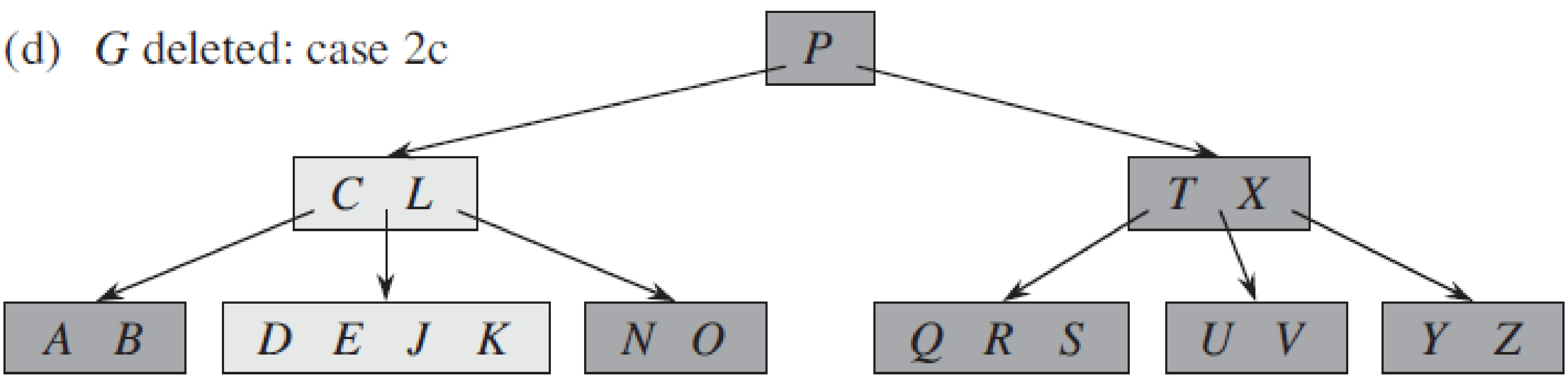
Case 2

2. If the key k is in node x and x is an internal node, do the following:
   - b. If *y has fewer than t keys*, then, symmetrically, examine the child z that follows k in node x. If z has at least t keys, then find the successor k' of k in the subtree rooted at z. Recursively delete k', and replace k by k' in x.
     (We can find k' and delete it in a single downward pass.)

Case 2

2. If the key k is in node x and x is an internal node, do the following:
   - c. Otherwise, if both y and z have only t-1 keys, merge k and all of z into y, so that x loses both k and the pointer to z, and y now contains 2t-1 keys. Then free z and recursively delete k from y.

(d) *G* deleted: case 2c



"*Introduction to Algorithms*", by Cormen

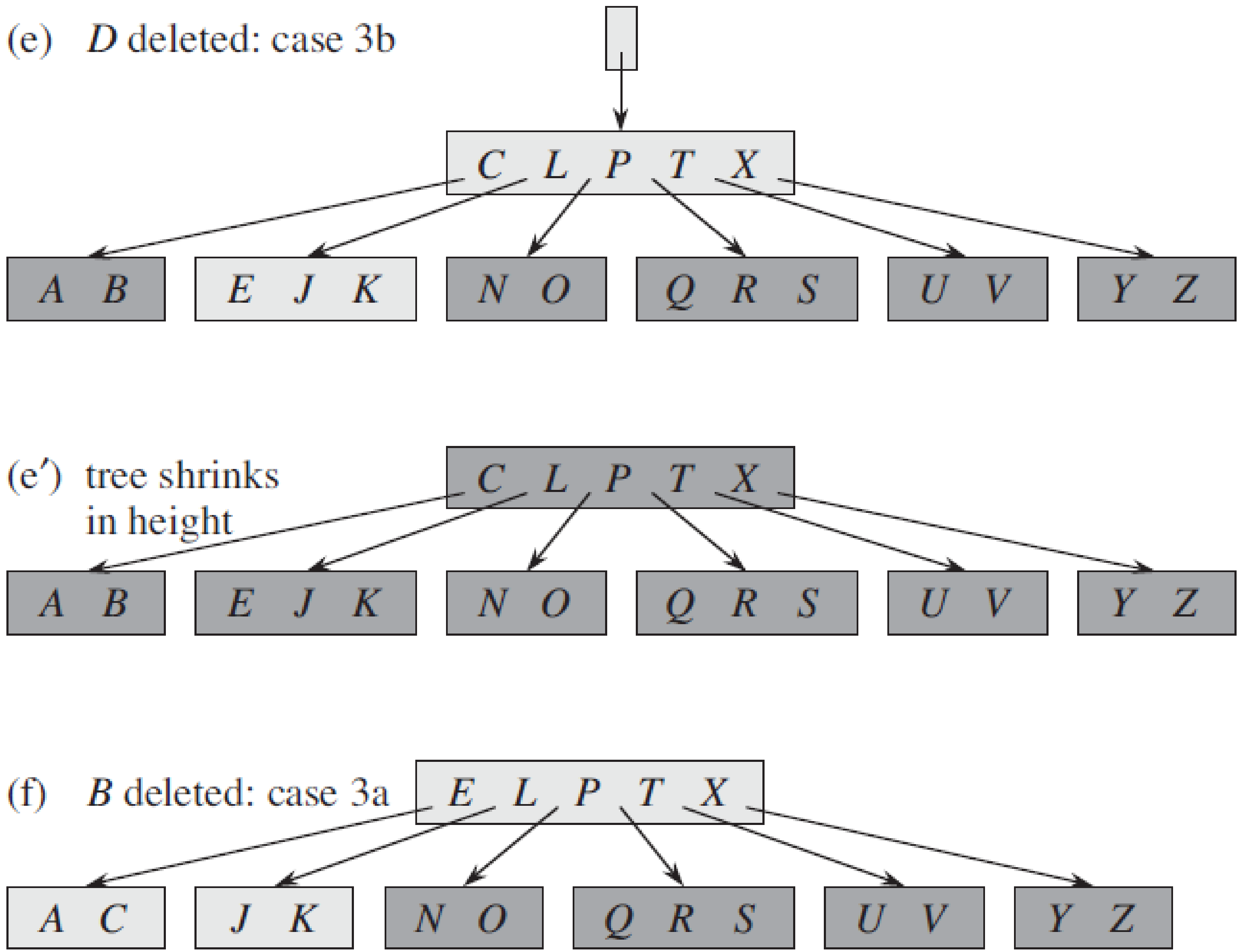
Case 3

3. If the key k is not present in internal node x, determine the root $x.c_i$ of the appropriate subtree that must contain k, if k is in the tree at all. If $x.c_i$ has only t -1 keys, execute step 3a or 3b as necessary to guarantee that we descend to a node containing at least t keys. Then finish by recursing on the appropriate child of x:

Case 3

- a. If $x.c_i$ has only t -1 keys but has an immediate sibling with at least t keys, give $x.c_i$ an extra key by moving a key from x down into $x.c_i$, moving a key from $x.c_i$'s immediate left or right sibling up into x, and moving the appropriate child pointer from the sibling into $x.c_i$.
- b. If $x.c_i$ and both of $x.c_i$'s immediate siblings have t -1 keys, merge $x.c_i$ with one sibling, which involves moving a key from x down into the new merged node to become the median key for that node.

Case 3

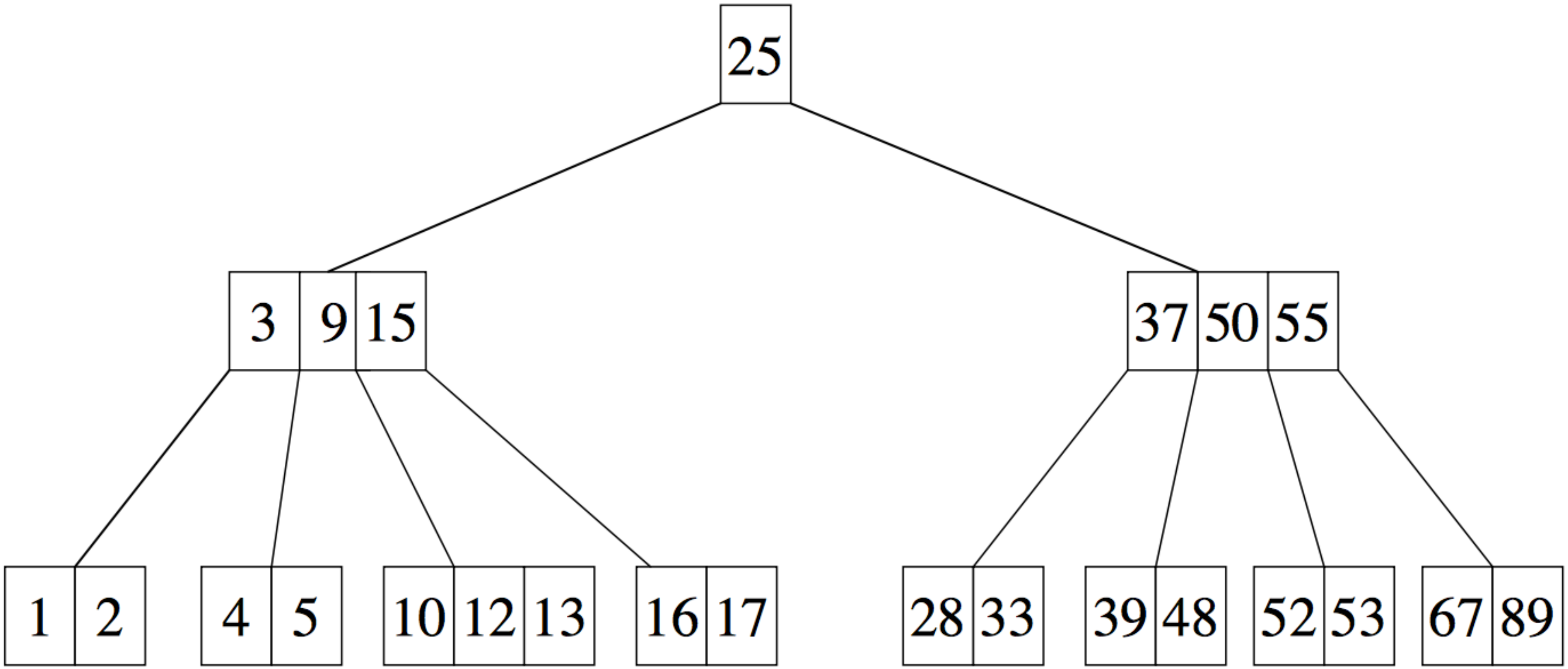


"*Introduction to Algorithms*", by Cormen

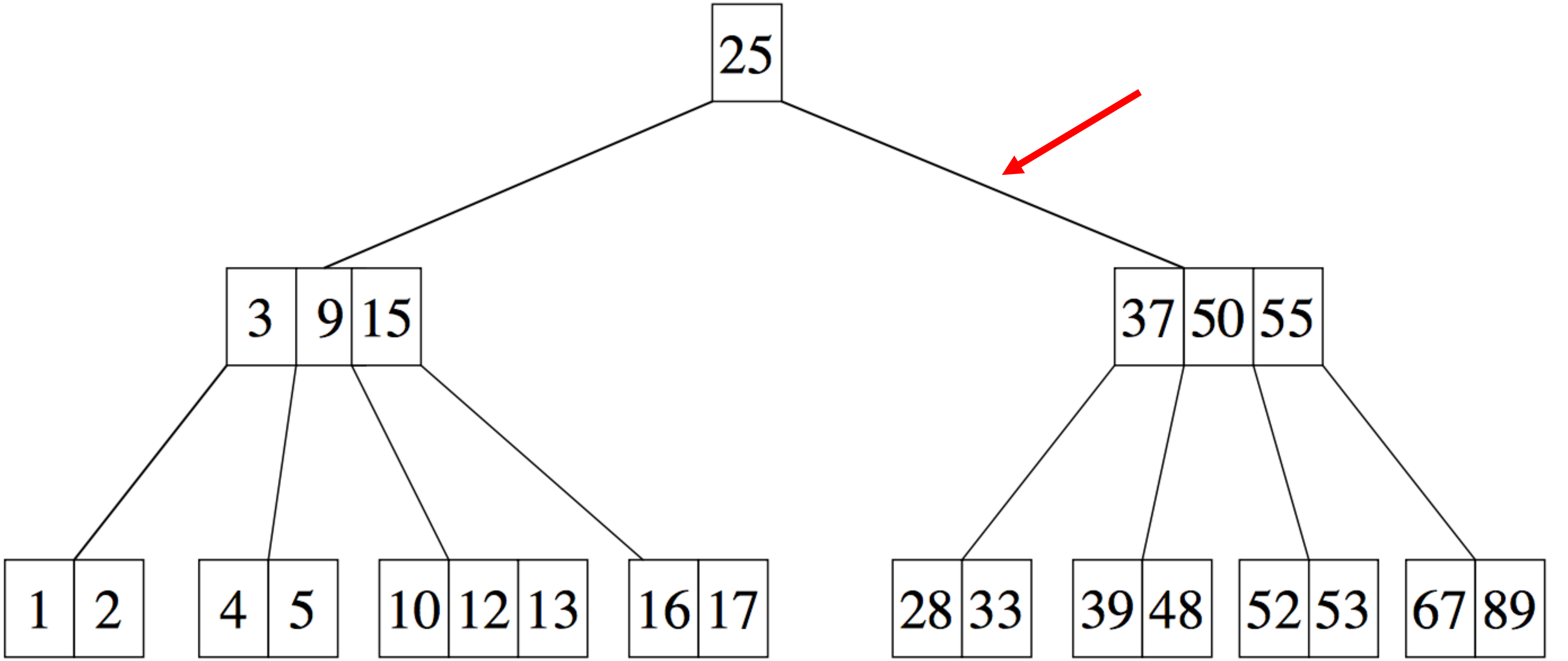
Deletion in a B-Tree

- Just like as in BST
- May cause underflow
- Depending on how many records the sibling of the node has, this can be fixed either by fusion or by transfer

Example: Delete 37

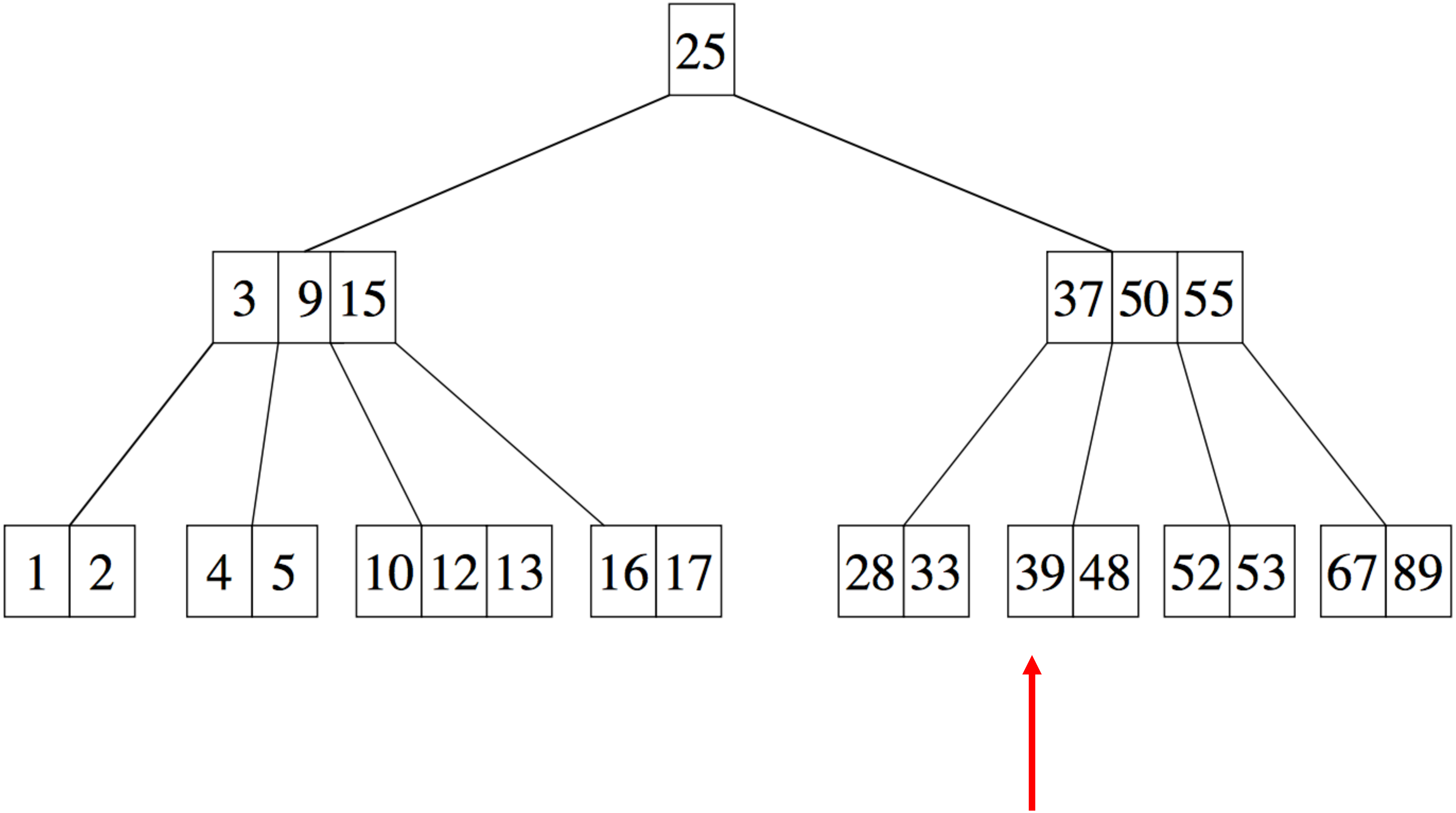


Delete 37



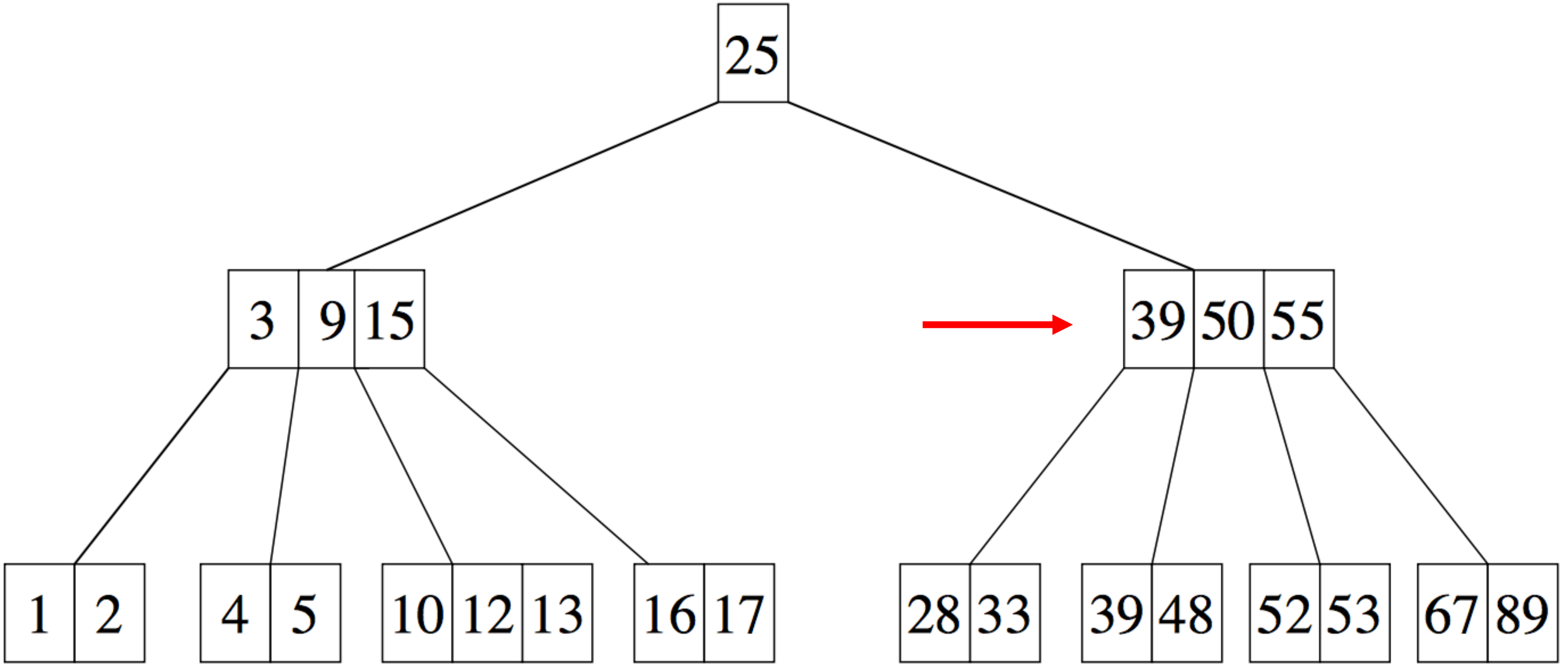
Search for 37

Delete 37



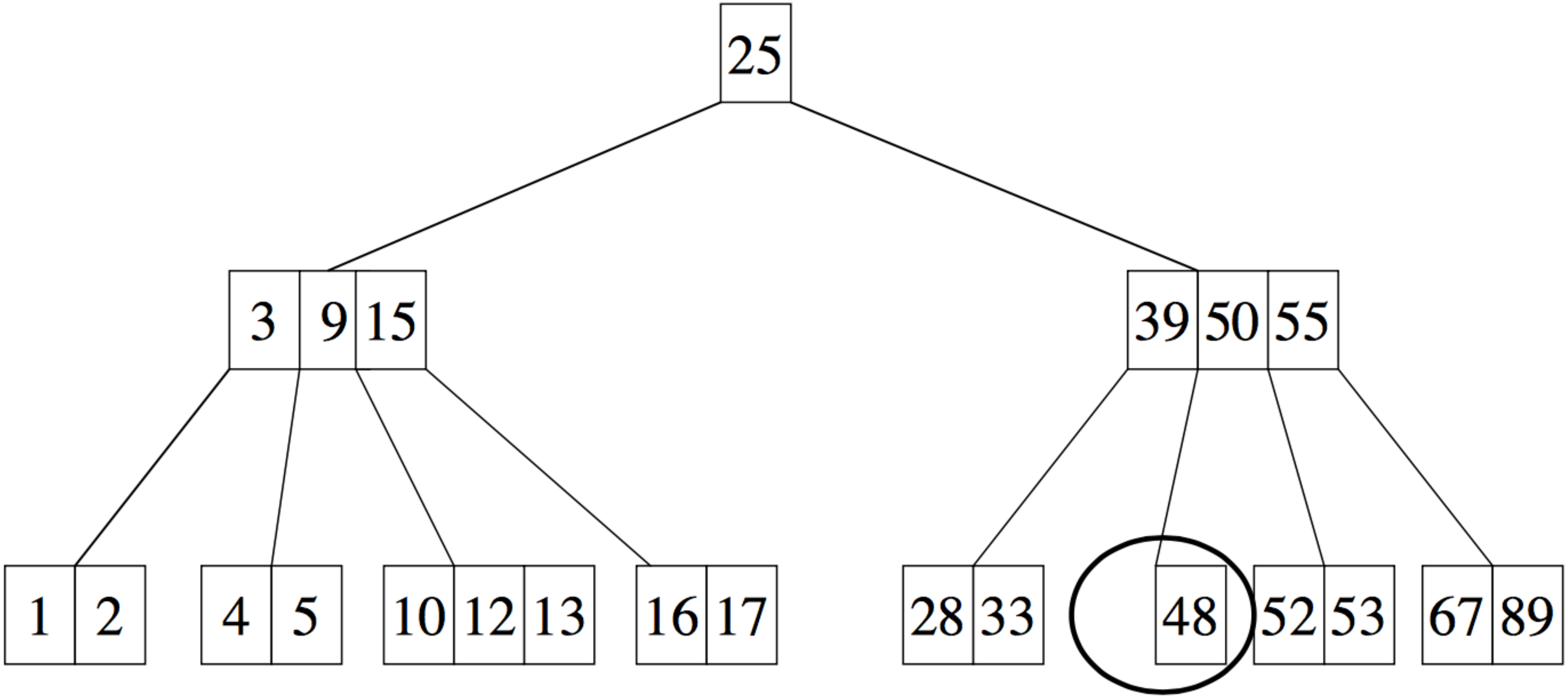
Find its replacement!

Delete 37



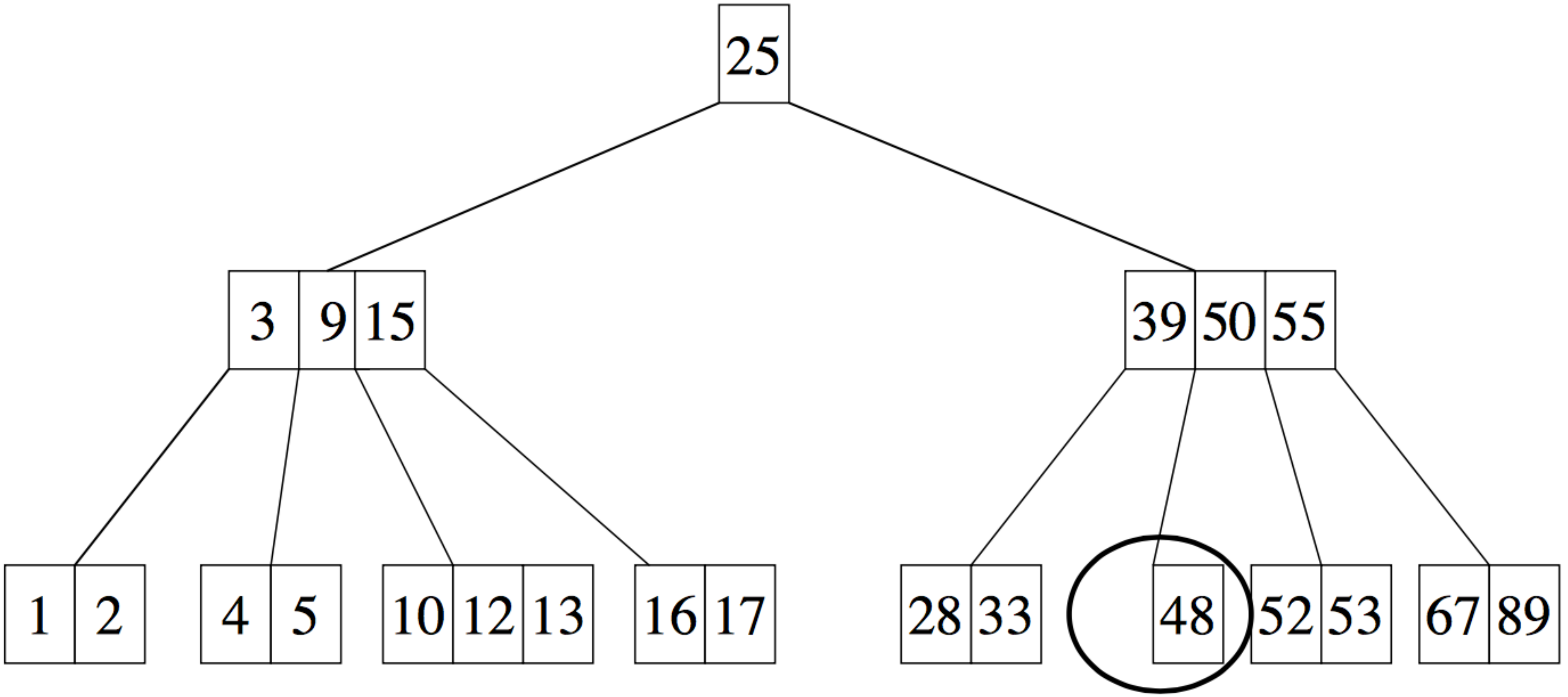
Replace 37 with its replacement!

Delete 37



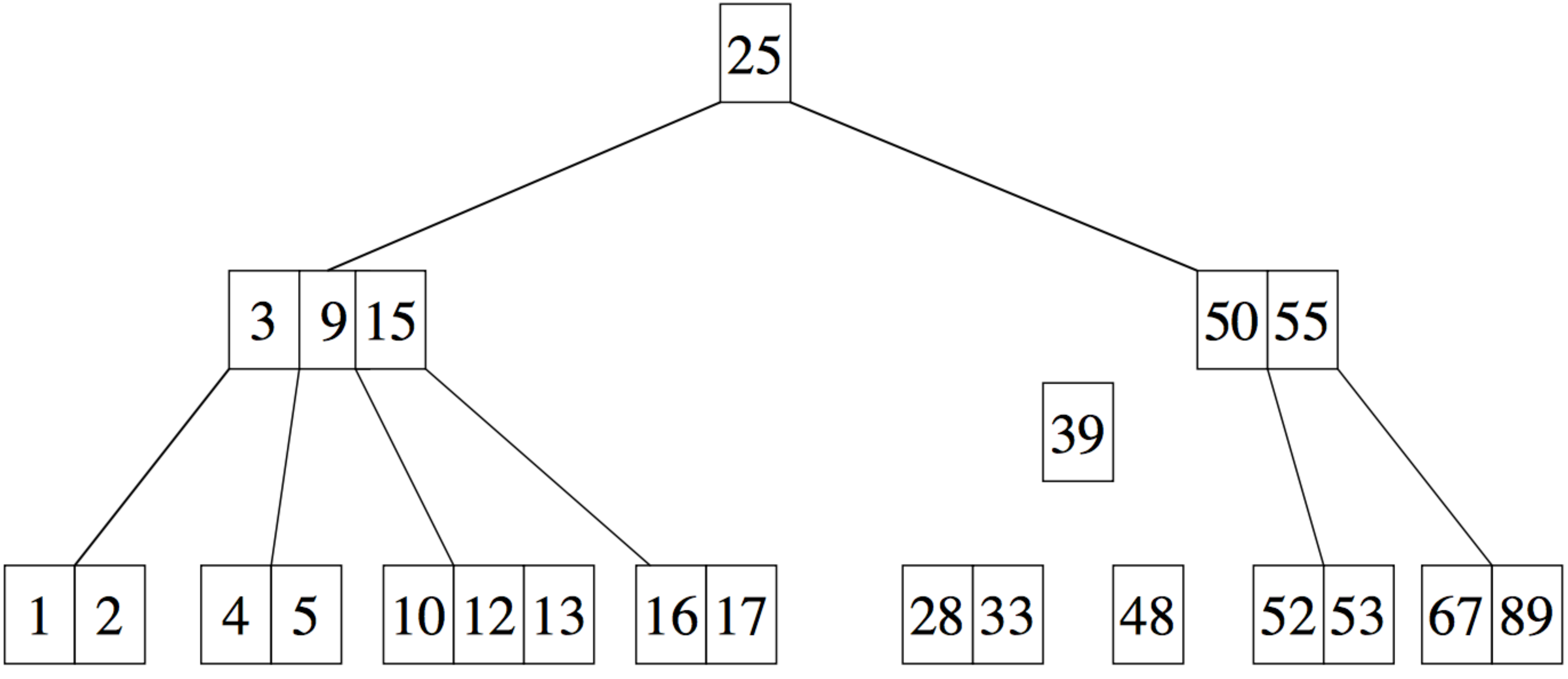
Remove 39 from the leaf node – Results in Underflow

Delete 37

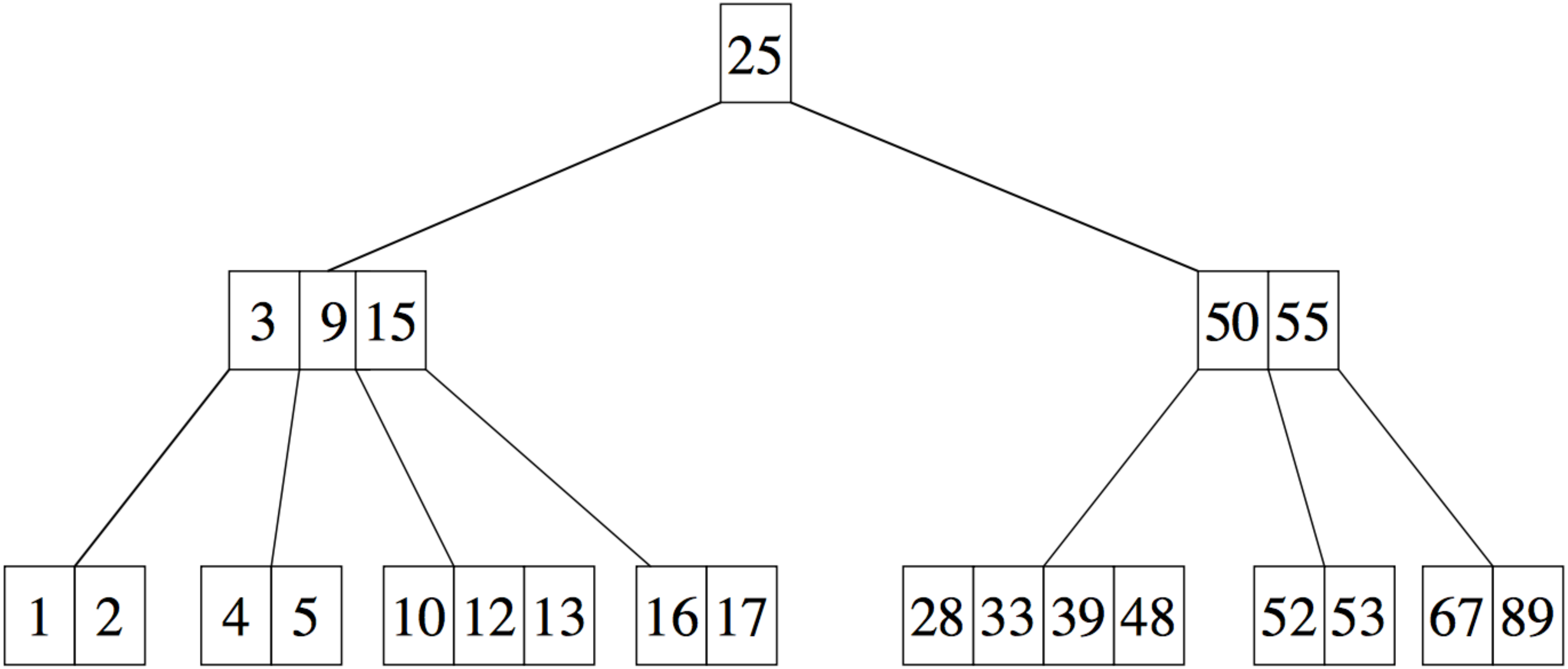


How to fix this underflow?

Delete 37: Fixing Underflow

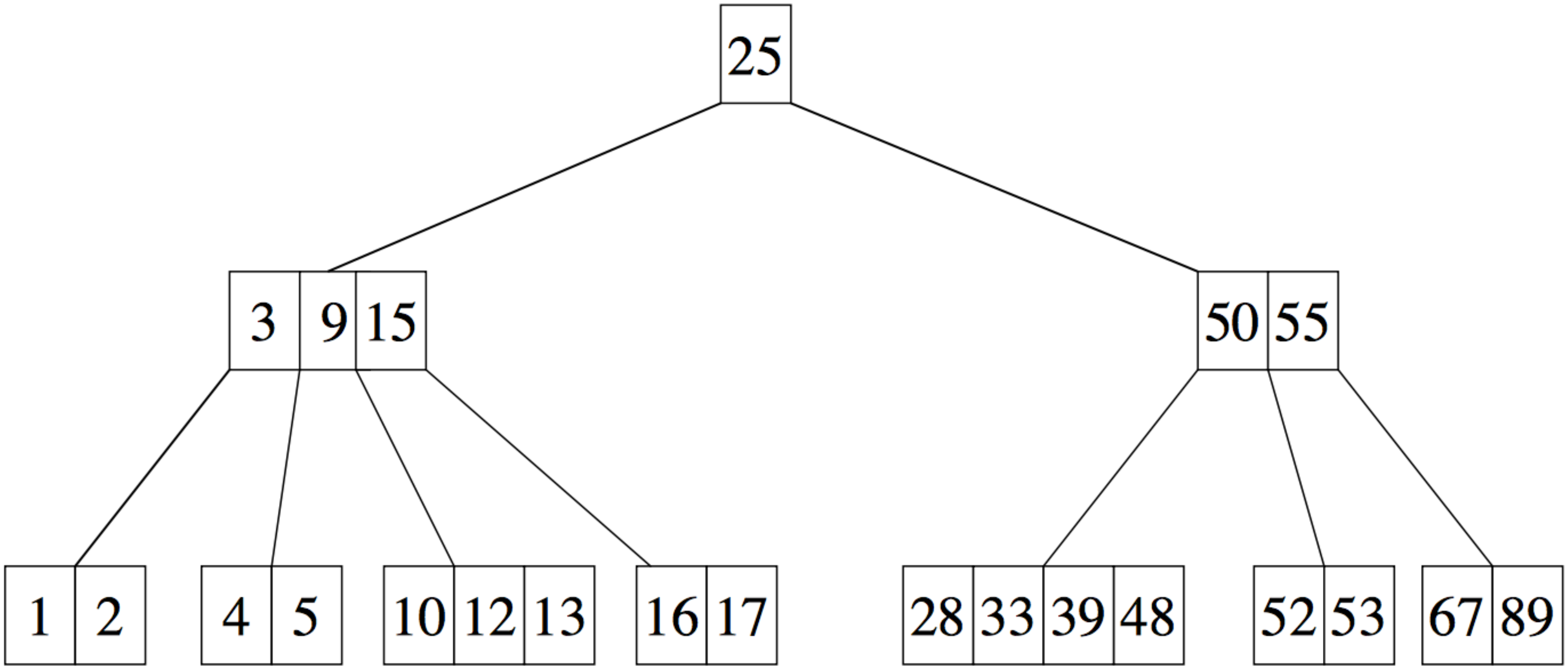


Delete 37: Fixing Underflow

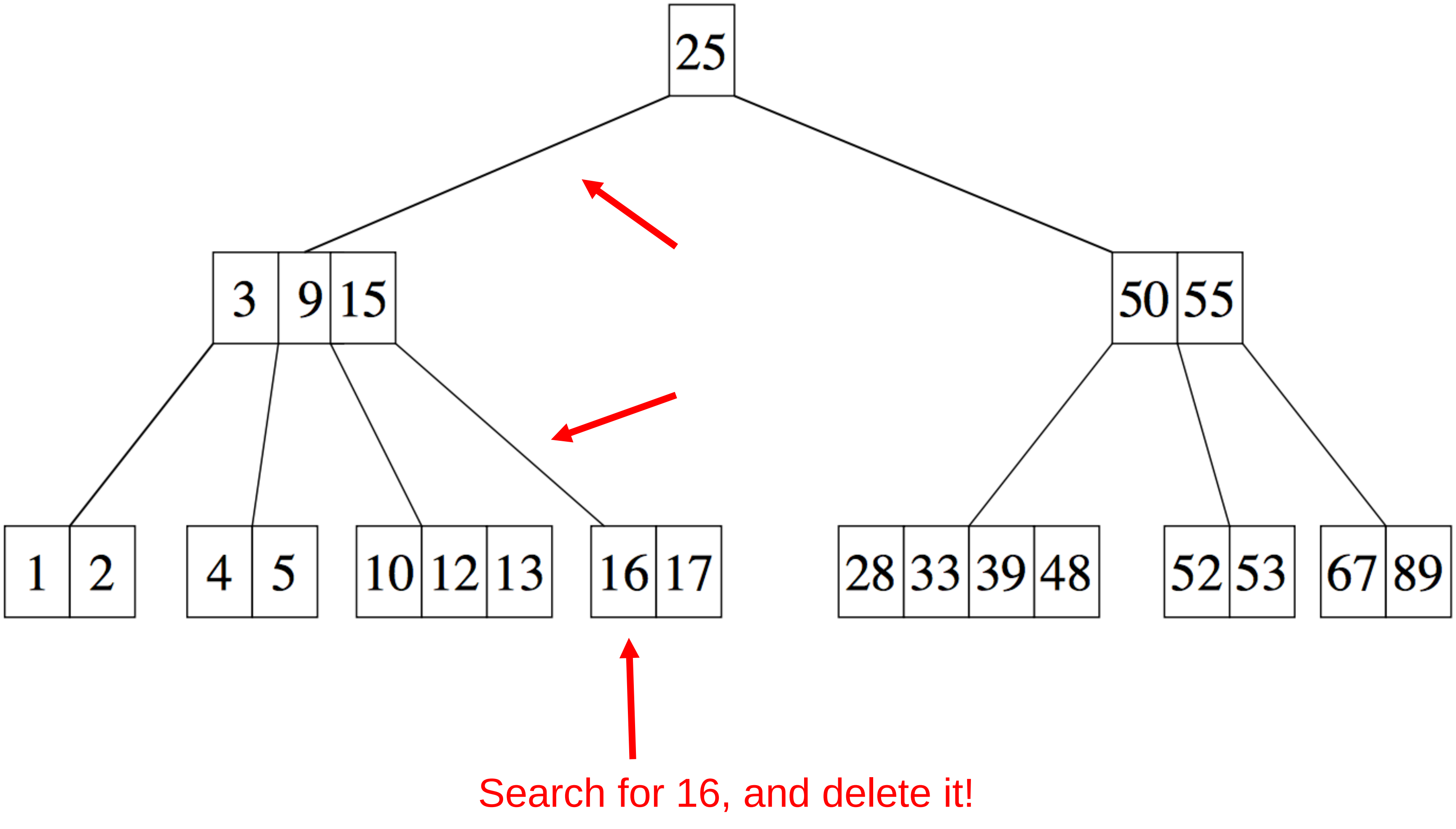


This is called "Fusion".

Delete 16

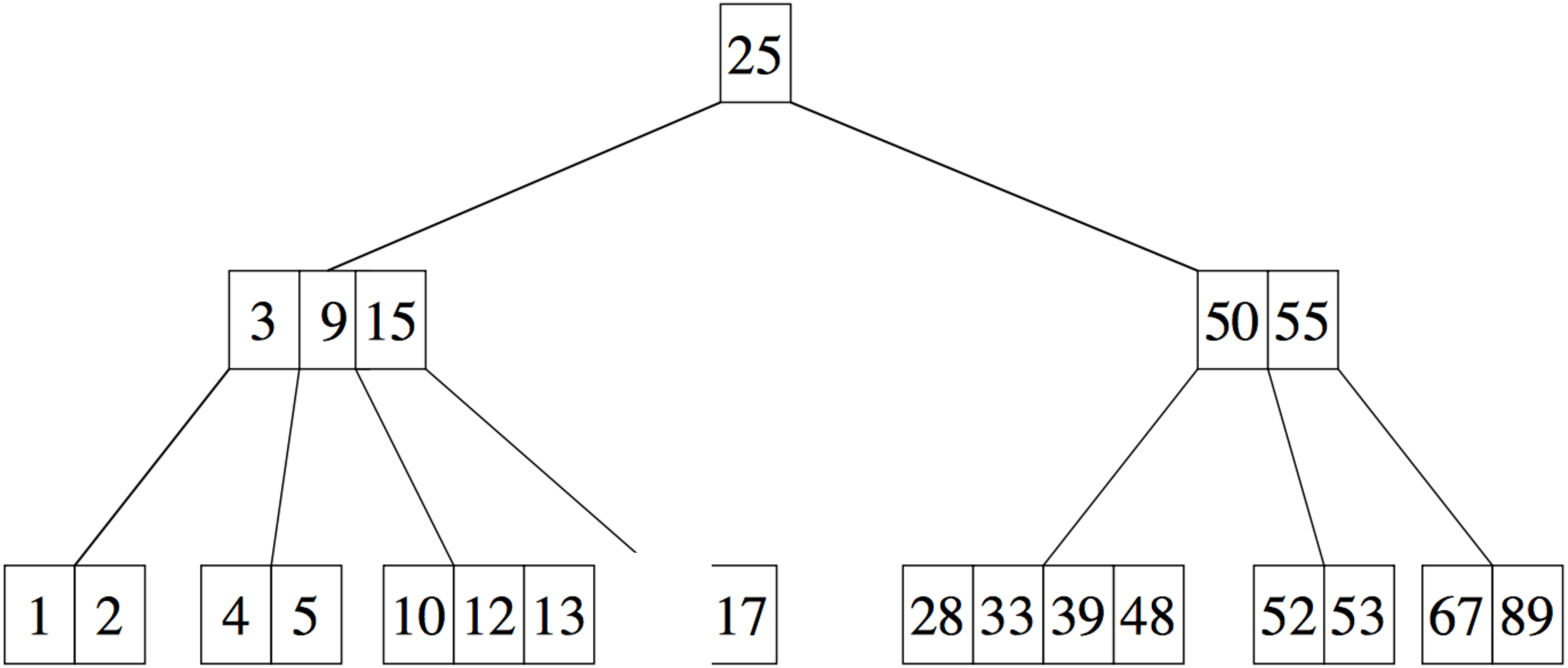


Delete 16



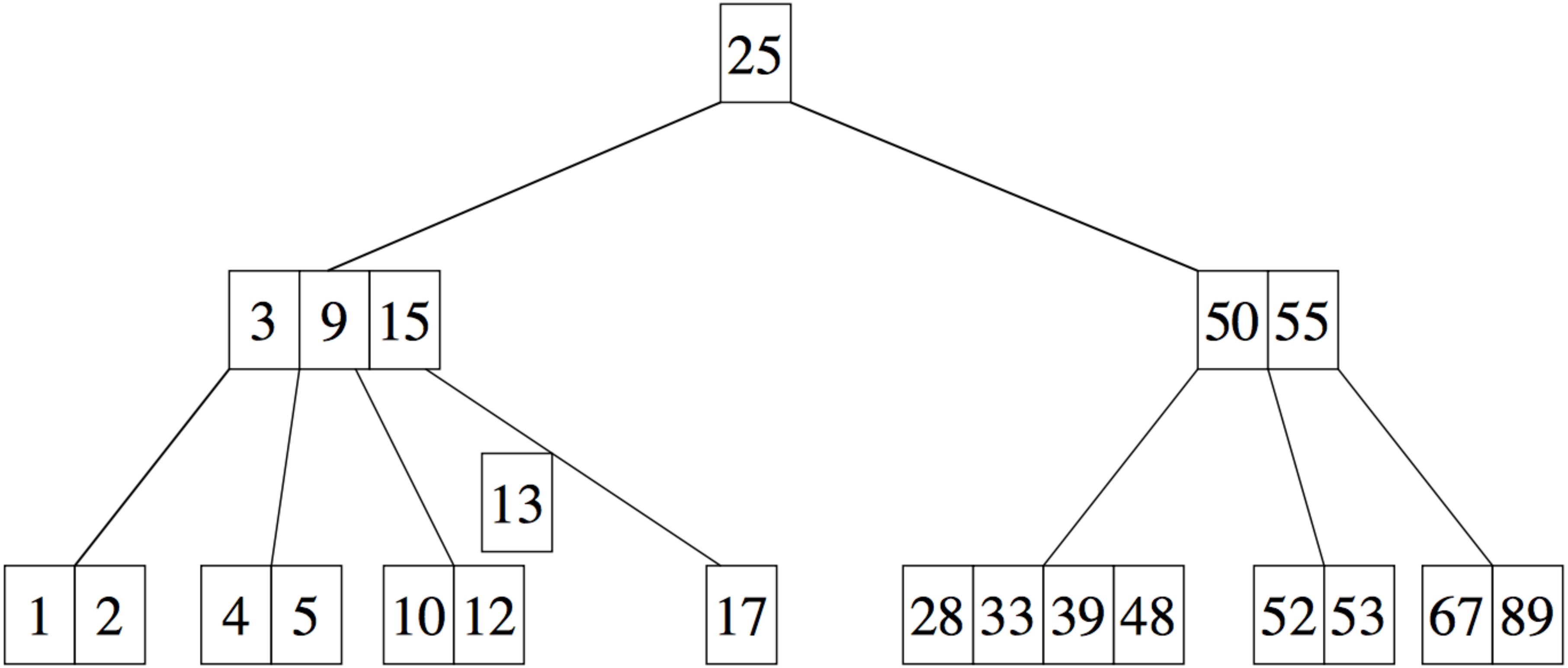
25
3
9
15
50
55
1
2
4
5
10
12
13
16
17
28
33
39
48
52
53
67
89
Search for 16, and delete it!

Delete 16



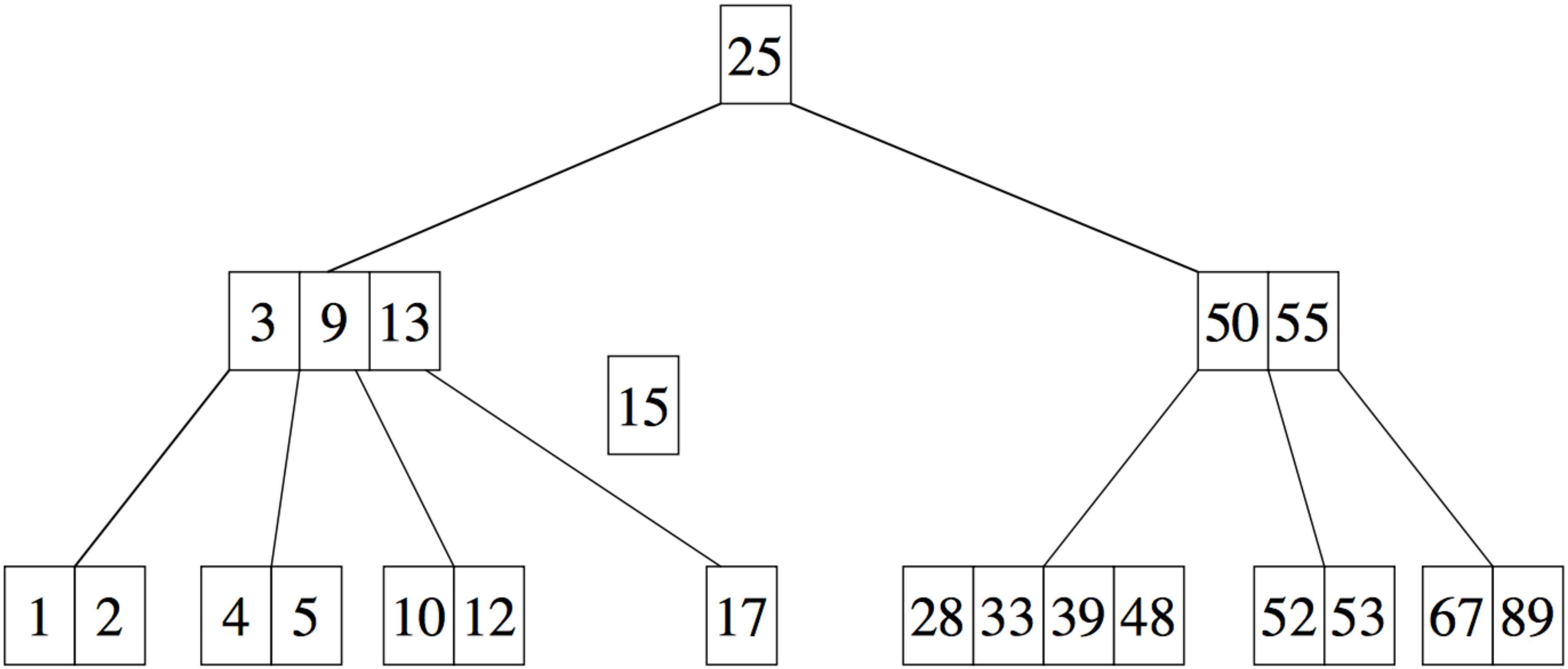
Creates underflow!

Delete 16: Fixing Underflow



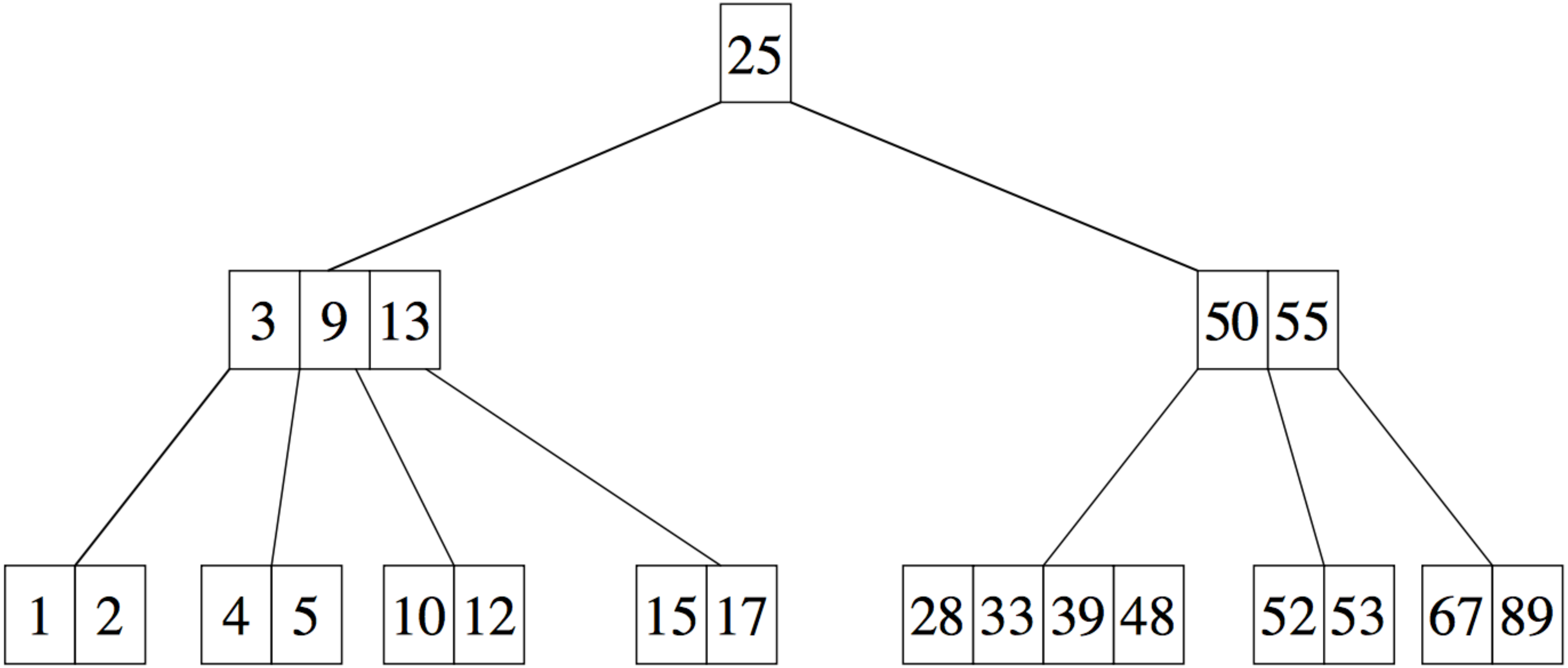
Fixing underflow!

Delete 16: Fixing Underflow



Fixing underflow!

Delete 16: Fixing Underflow



This is called “Transfer”.

Red-Black Trees

Chapter 11.6 - Data Structures and Algorithms in Java (6nd Edition) by Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser, 2014

Did we achieve today's objectives?

- 2-3-4 Trees
- Insertions and Deletions in 2-3-4 Trees
- B-Trees
- Insertions and Deletions in B-Trees