Multiway Trees

Lecture 15

Instructor: Dr. Cong Pu, Ph.D.

cong.pu@okstate.edu

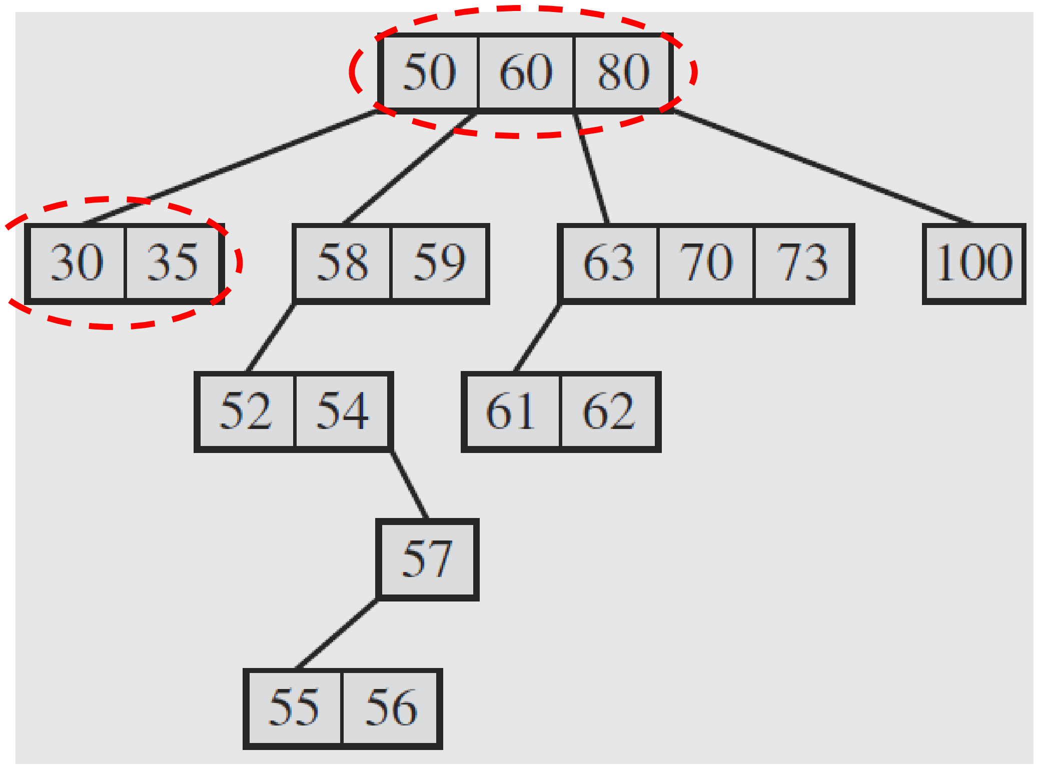
Adapted partially from Data Structures and Algorithms in Java, M.T. Goodrich, R.Tamassia and M. H. Goldwasser, Sixth Edition, Wiley; Data Structures and Algorithms in C++, Adam Drozdek, 4th Edition, Cengage Learning

Introduction

- ***Multiway trees of order m*** or ***m-way trees***
  - multiple children
  - can have *more than* ***two*** children
- Four major characteristics of *m-way search tree:*
  - each node has ***m* children** and ***m* – 1 keys** (values)
  - the keys in each node are in ***ascending*** order
  - the keys in the first *i* children are ***smaller*** than the *i-th* key
  - the keys in the last *m* – *i* children are ***larger*** than the *i-th* key
- Purpose: fast information retrieval and update

Introduction (cont.)

- A 4-way tree, (m is 4)
  - unbalanced



each node has at most 4 children

the keys in each node are in ascending order

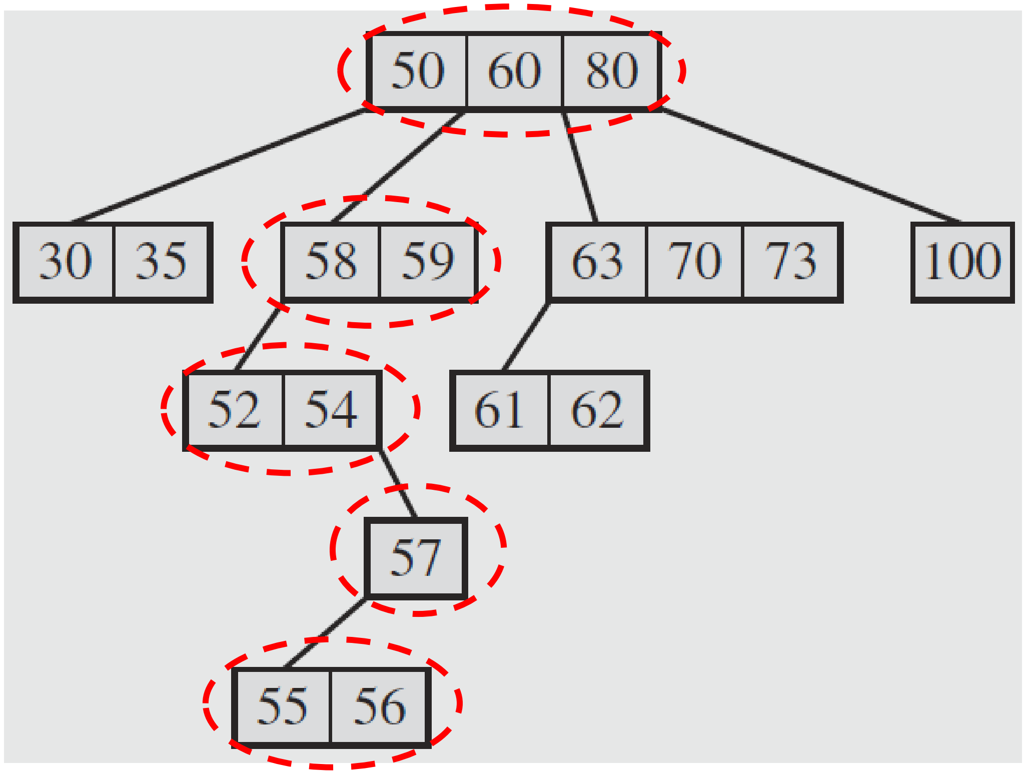
the keys in the first i children are smaller than the i-th key

the keys in the last m – i children are larger than the i-th key

find number 35? two node tests

Introduction (cont.)

- A 4-way tree, (m is 4)
  - unbalanced



each node has at most 4 children

the keys in each node are in ascending order

the keys in the first i children are smaller than the i-th key

the keys in the last m – i children are larger than the i-th key

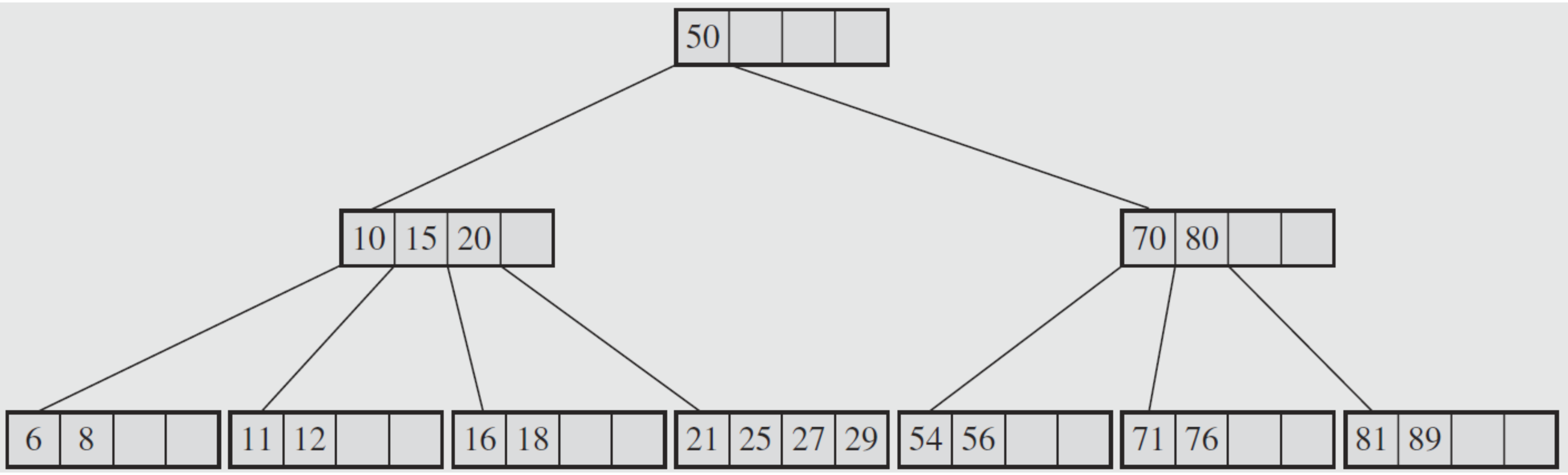
find number 55? five node tests

B-Trees

- A *B-tree of order m* is a multiway search tree with the following properties
  - root node has at least two subtree unless it is a leaf
  - each non-root and non-leaf node
    - store k – 1 keys and k pointers to subtrees, where ceil(m/2) <= k <= m
  - each leaf node
    - store k – 1 keys, where ceil(m/2) <= k <= m
  - all leaves
    - locate at the same level
  - always at least half-full, few levels, and perfectly balanced

B-Trees (cont.)

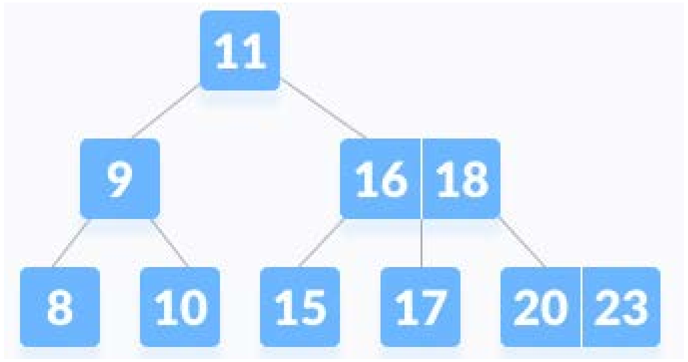
- A B-Tree of order **5**



B-Trees (cont.)

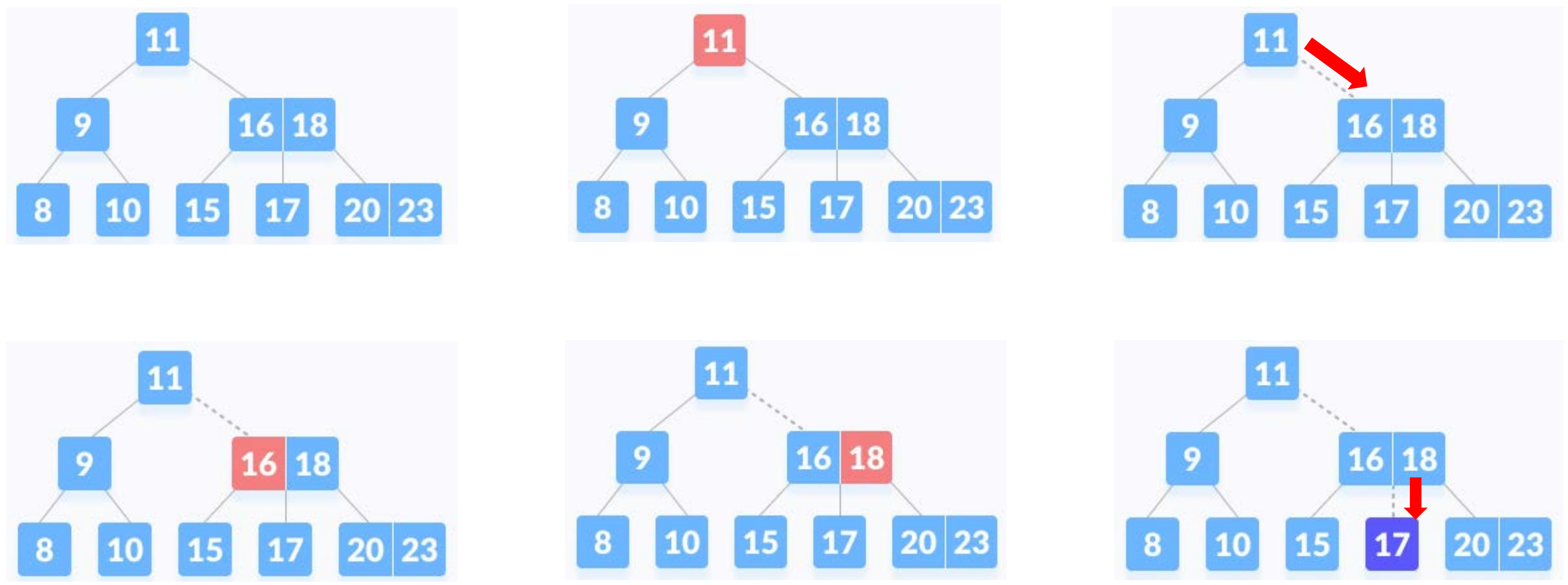
- B-Tree – Searching a key (value)
  - search in B-tree is similar to the search in binary search tree
  - algorithm:
    1. perform a binary search on the records in the current node
    2. if a record with the search key is found, then return that record
    3. if the current node is a leaf node and the key is not found, then report an unsuccessful search
    4. otherwise, follow the proper branch and repeat the process

search 17



B-Trees (cont.)

- B-Tree – Searching a key (value)
  - Search in B-tree is similar to the search in binary search tree
  - E.g., search 17

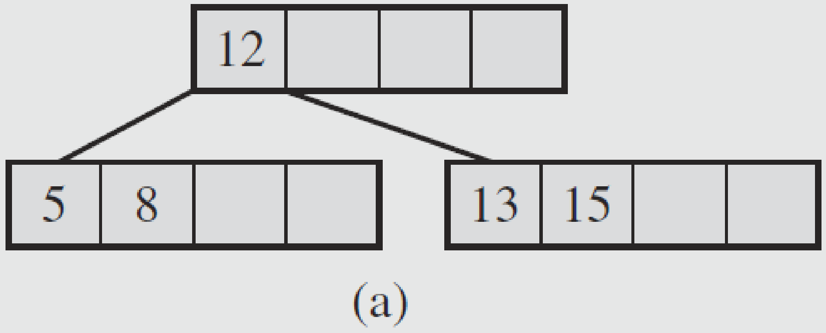


B-Trees (cont.)

- B-Tree – Inserting a key (value)
  - algorithm:
    - go directly to a leaf and place the key there if there is room
    - if the leaf is full
      - another leaf is created
      - the keys are divided between these leaves
      - one key is promoted to the parent
        - if the parent is full, the process is repeated until the root is reached and a new root created

B-Trees (cont.)

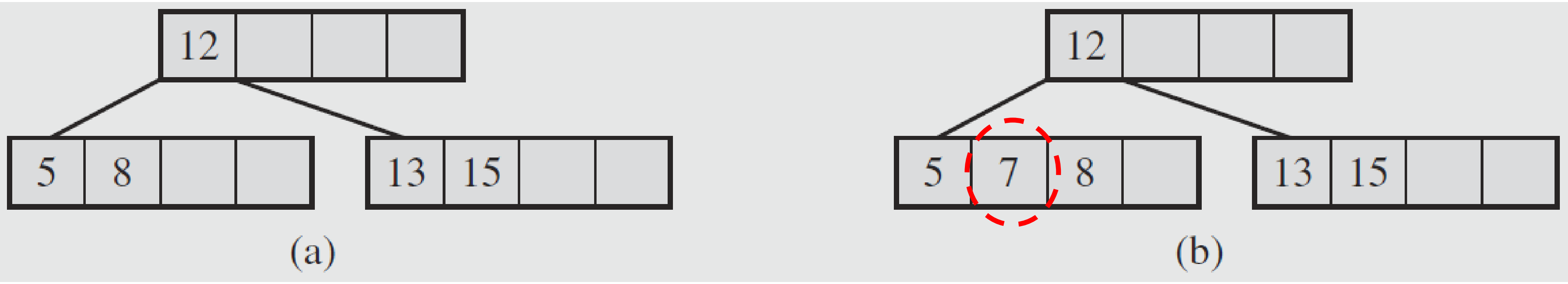
- B-Tree – Inserting a key (value)
  - 1st case, a key is placed in a leaf that still has room
    - insert 7



(a)

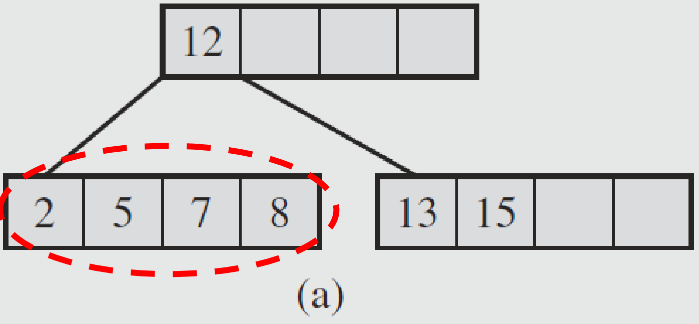
B-Trees (cont.)

- B-Tree – Inserting a key (value)
  - 1st case, a key is placed in a leaf that still has room
    - insert 7



B-Trees (cont.)

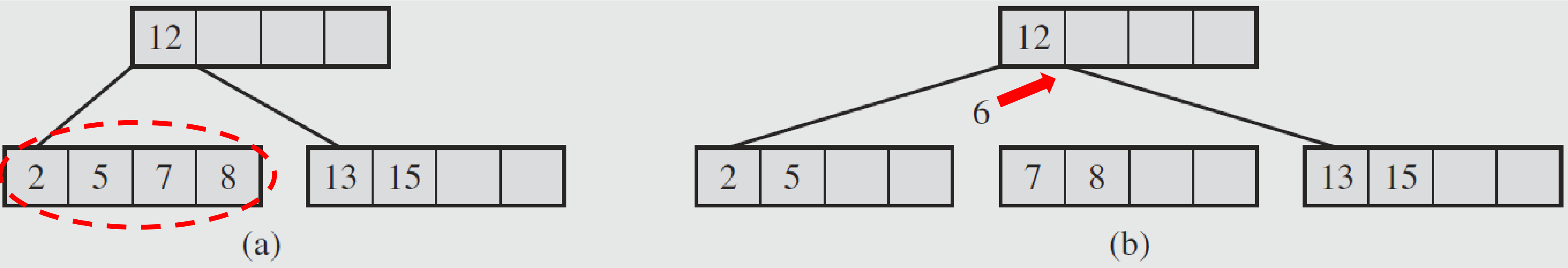
- B-Tree – Inserting a key (cont.)
  - 2nd case, the leaf where the key should be inserted is full
    - insert 6



(a)

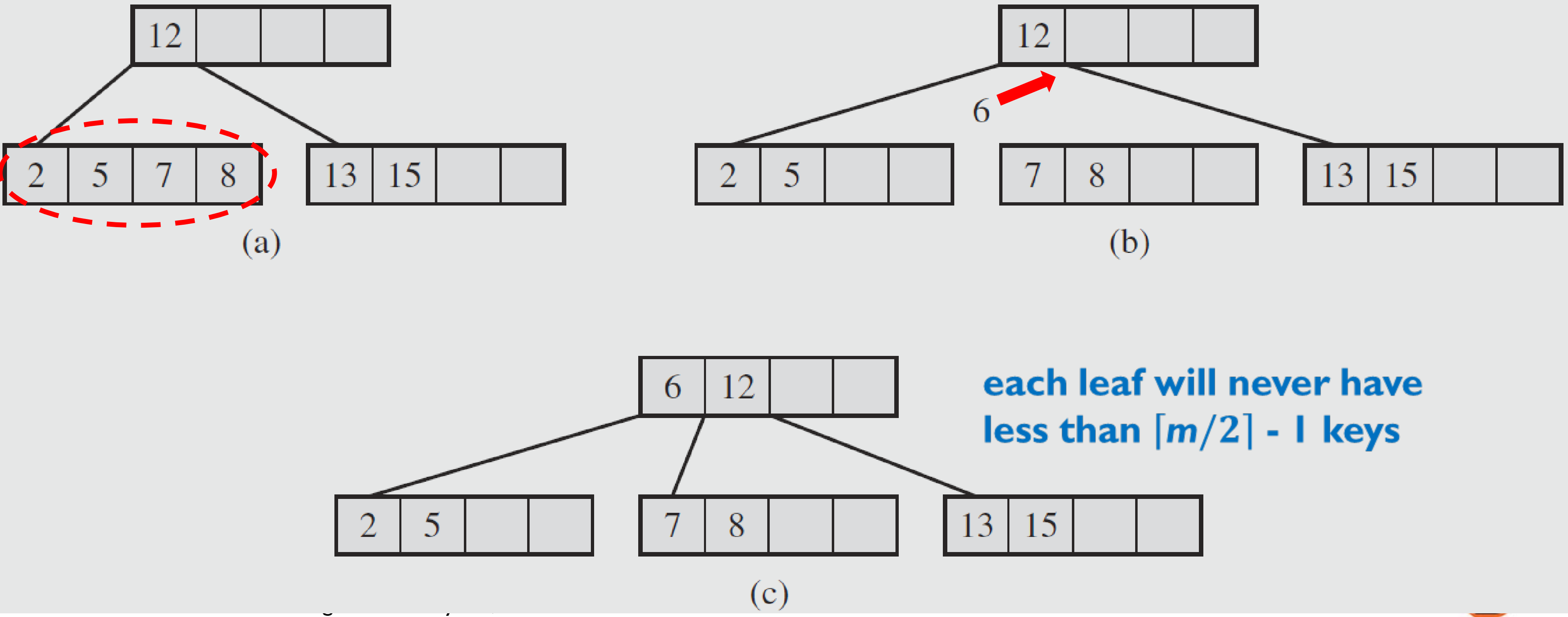
B-Trees (cont.)

- B-Tree – Inserting a key (cont.)
  - 2nd case, the leaf where the key should be inserted is full
    - insert 6



B-Trees (cont.)

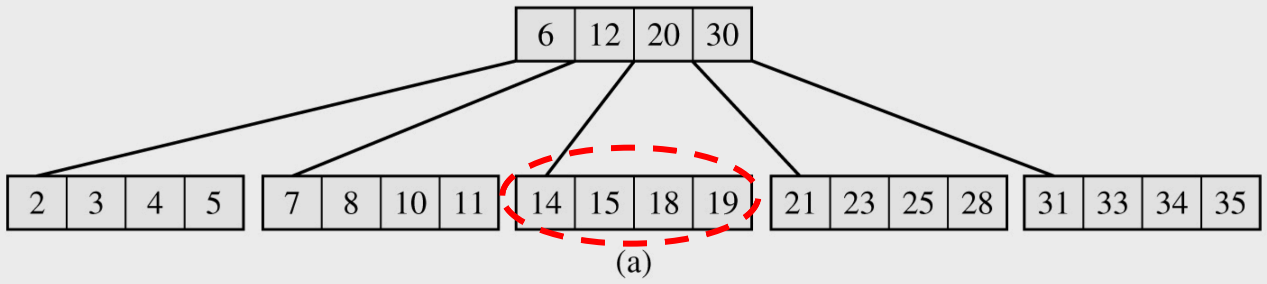
- B-Tree – Inserting a key (cont.)
  - 2nd case, the leaf where the key should be inserted is full
    - insert 6



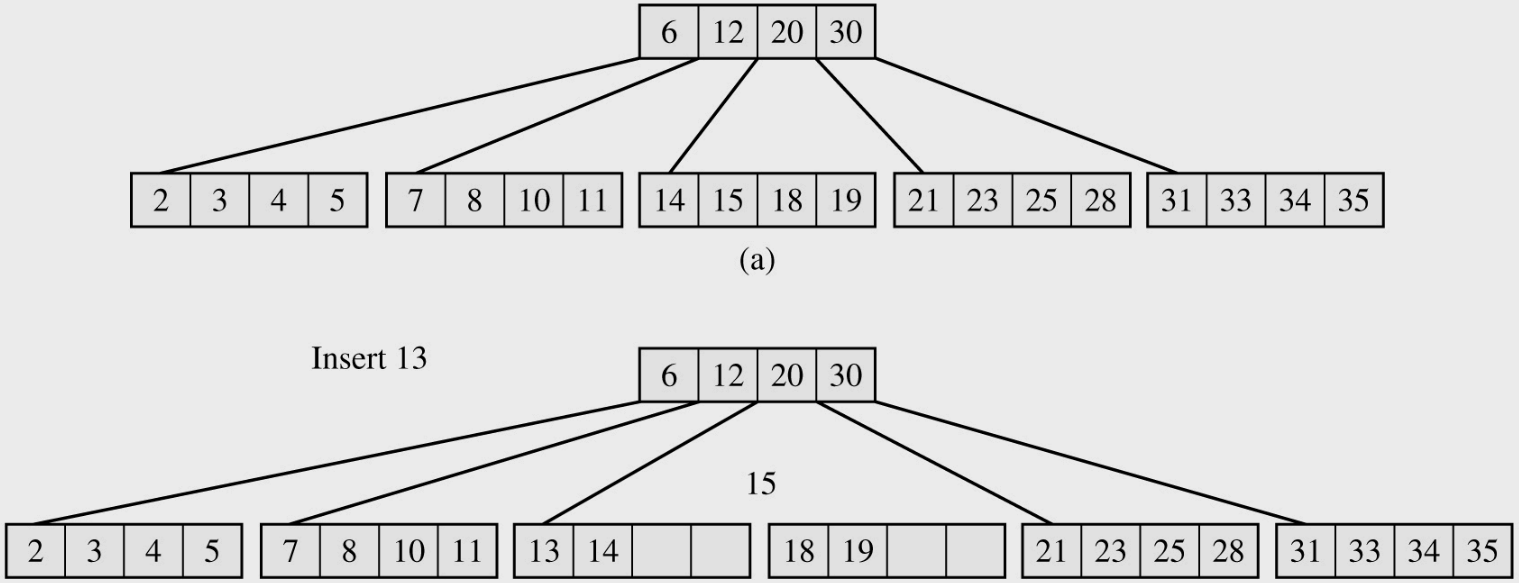
each leaf will never have less than $\lceil m/2 \rceil$ - 1 keys

B-Trees (cont.)

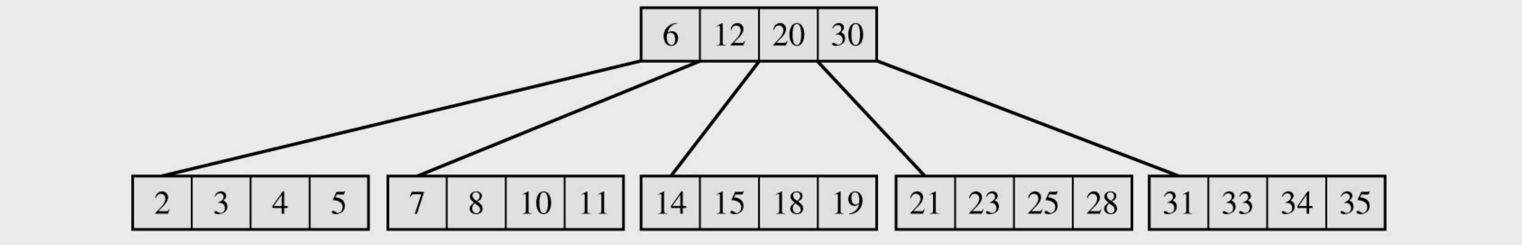
- B-Tree – Inserting a key (cont.)
  - 3rd case, if the root of the B-tree is full
    - a new root and a new sibling of the existing root have to be created

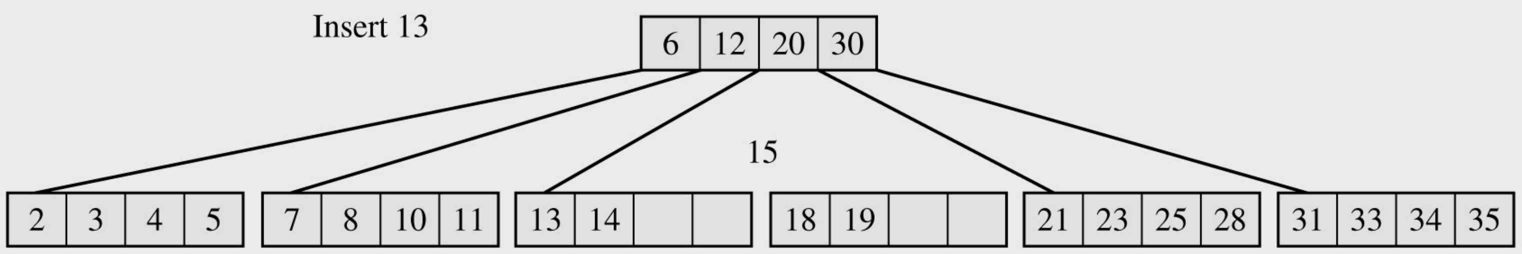


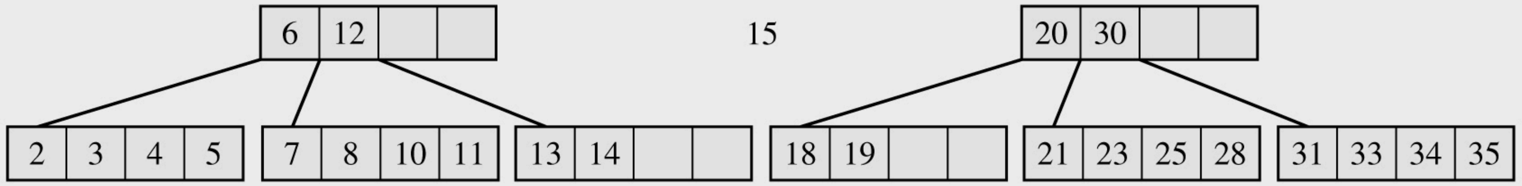
Insert 13

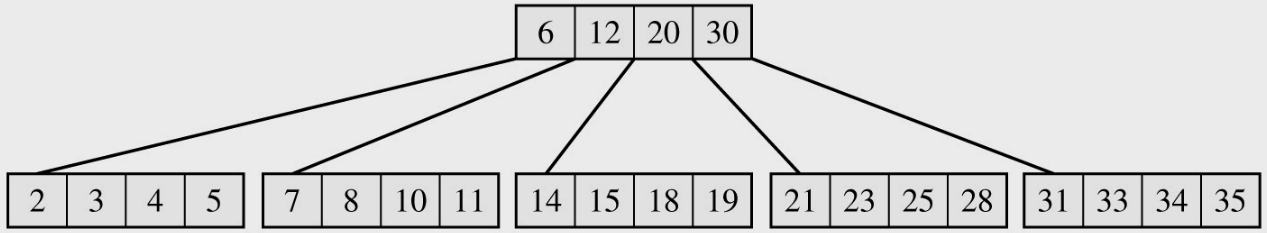


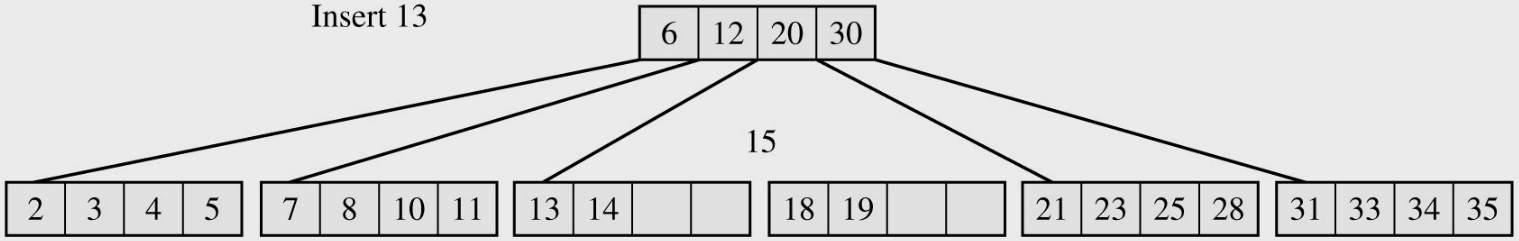
(a)
(b)

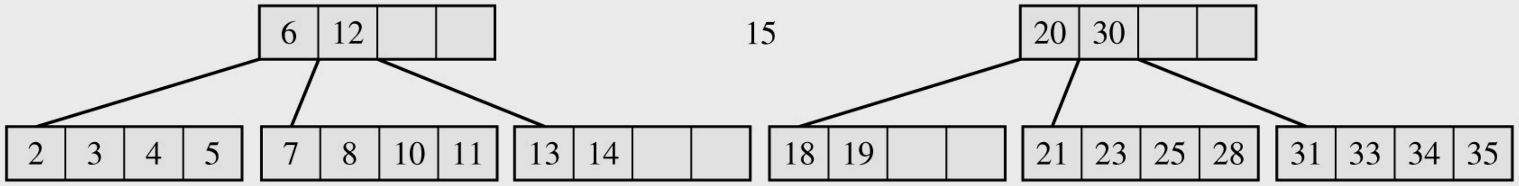

6
12
20
30
2
3
4
5
7
8
10
11
14
15
18
19
21
23
25
28
31
33
34
35
(a)

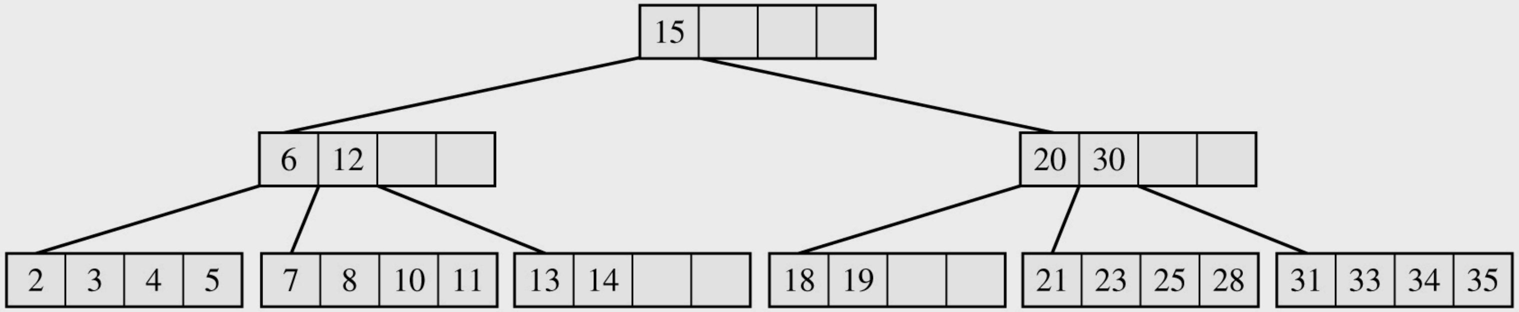

Insert 13
6
12
20
30
15
2
3
4
5
7
8
10
11
13
14
18
19
21
23
25
28
31
33
34
35
(b)


6
12
15
20
30
2
3
4
5
7
8
10
11
13
14
18
19
21
23
25
28
31
33
34
35
(c)


(a)

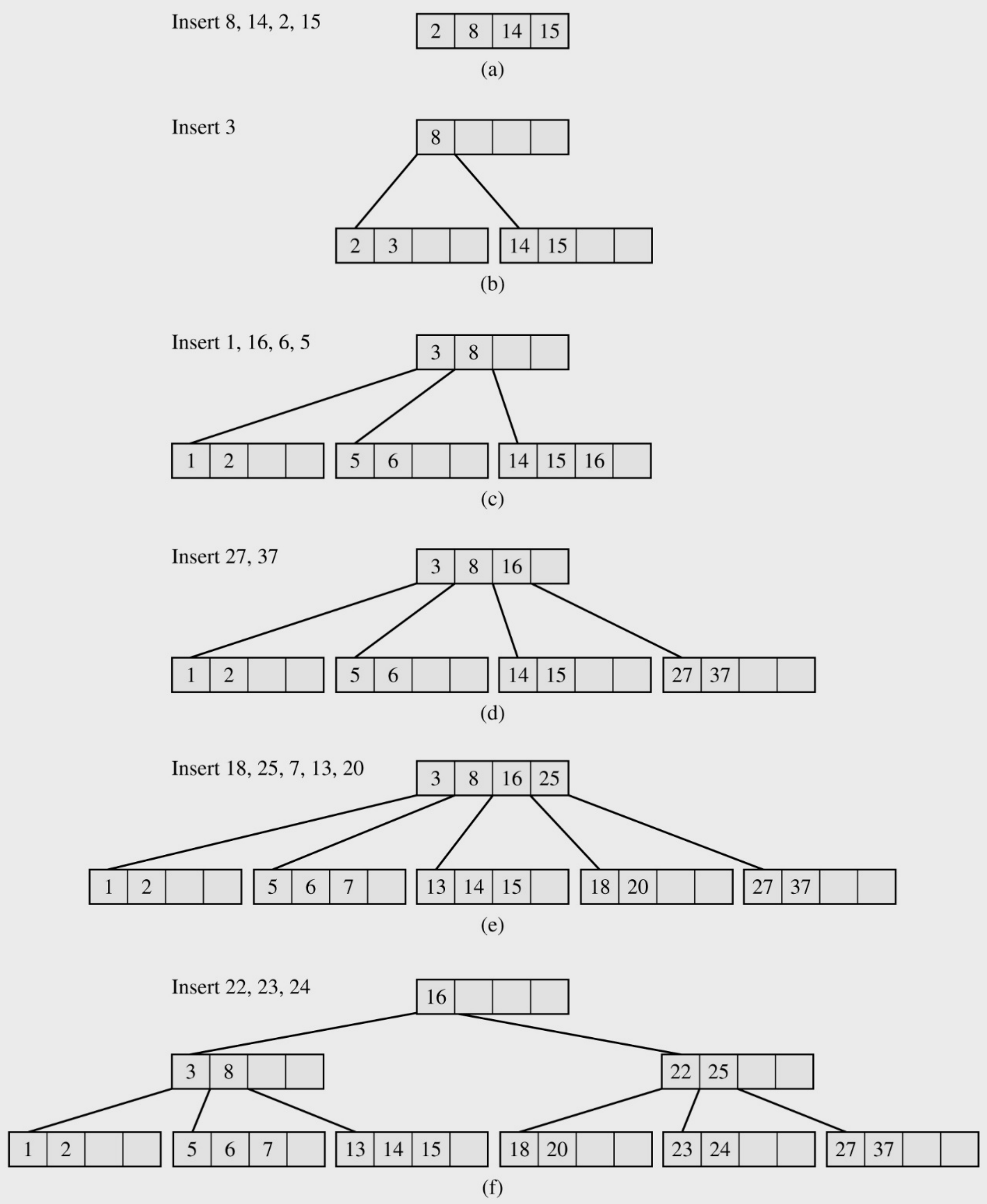

Insert 13
6
12
20
30
15
2
3
4
5
7
8
10
11
13
14
18
19
21
23
25
28
31
33
34
35
(b)


6
12
15
20
30
(c)


15
6
12
20
30
(d)

B-Trees (cont.)

- B-Tree – Inserting a key (cont.)
  - build a B-tree of order 5 with the following sequence of data, 8, 14, 2, 15, 3, 1, 16, 6, 5, 27, 37, 18, 25, 7, 13, 20, 22, 23, 24


Insert 8, 14, 2, 15
2 | 8 | 14 | 15
(a)
Insert 3
8
2 | 3
14 | 15
(b)
Insert 1, 16, 6, 5
3 | 8
1 | 2
5 | 6
14 | 15 | 16
(c)
Insert 27, 37
3 | 8 | 16
1 | 2
5 | 6
14 | 15
27 | 37
(d)
Insert 18, 25, 7, 13, 20
3 | 8 | 16 | 25
1 | 2
5 | 6 | 7
13 | 14 | 15
18 | 20
27 | 37
(e)
Insert 22, 23, 24
16
3 | 8
22 | 25
1 | 2
5 | 6 | 7
13 | 14 | 15
18 | 20
23 | 24
27 | 37
(f)