Multiway Trees

Instructor: Dr. Sunho Lim (Ph.D., Assistant Professor)

Lecture 12

sunho.lim@ttu.edu

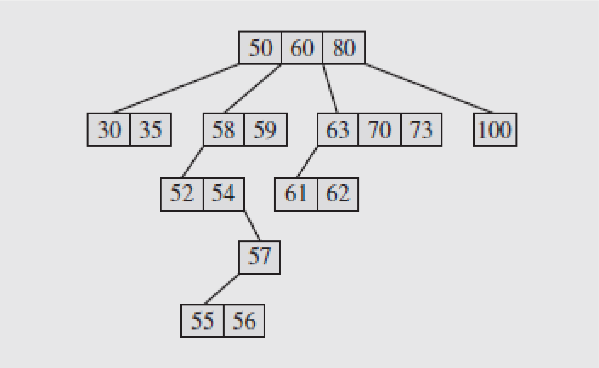
Adapted partially from Data Structures and Algorithms in C++, Adam Drozdek, 4th Edition, Cengage Learning; and Algorithms and Data Structures, Douglas Wilhelm Harder, Mmath

Introduction

- ***Multiway trees of order m*** or ***m-way trees***
  - multiple children
  - can have more than two children
- Four major characteristics:
  - 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

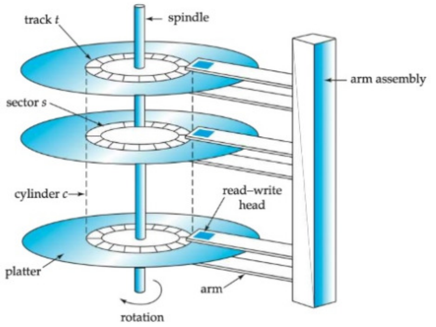
Introduction (cont.)

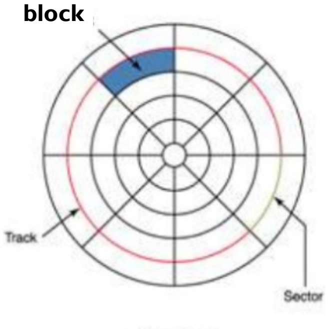
- A 4-way tree,
  - unbalanced



Motivation of B-Trees

- The basic unit of **data transfer** associated with I/O operations on secondary storage devices?
  - **block**
- Blocks, transferred to memory during read operations and written from memory during write operations

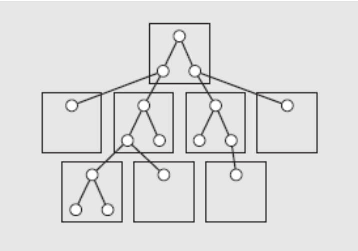




(b) A single disk

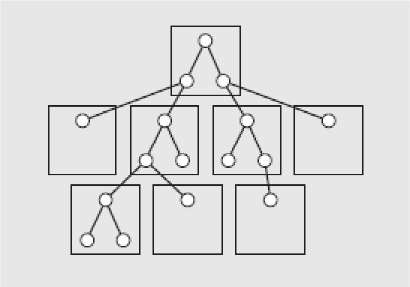
Motivation of B-Trees (cont.)

- A binary tree, for example,
  - may **spread over** a large number of blocks on a disk file
  - in this case two blocks would have to be accessed for each search??
    - performance will suffer
    - a poor choice for **secondary storage access**



Motivation of B-Trees (cont.)

- Better to transfer **a large amount of data** all at once than to have to jump around on the disk to retrieve it
  - due to the high cost of accessing secondary storage
    - access time = seek time + rotational delay (latency) + transfer time
  - If possible, data should be organized to **minimize disk access**

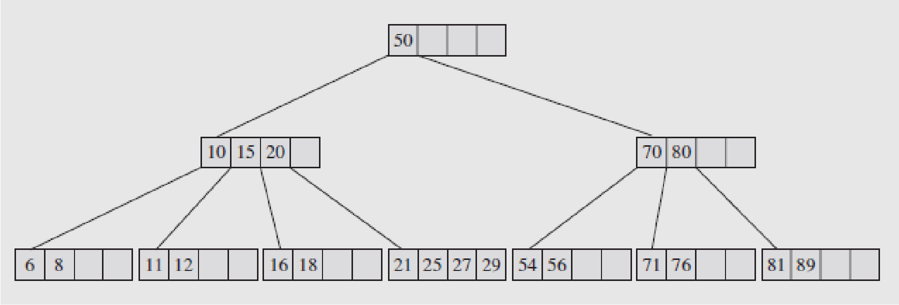


B-Trees

- Can be adjusted to reduce performance issues of secondary storage
  - e.g., the size of a B-tree node can be **as large as a block**
- A B-tree of order m,
  - 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

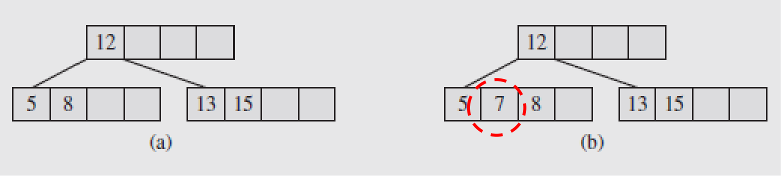
B-Trees (cont.)

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



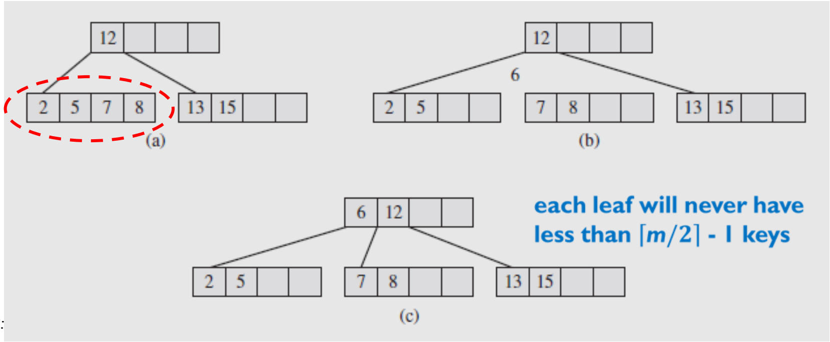
B-Trees (cont.)

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



B-Trees (cont.)

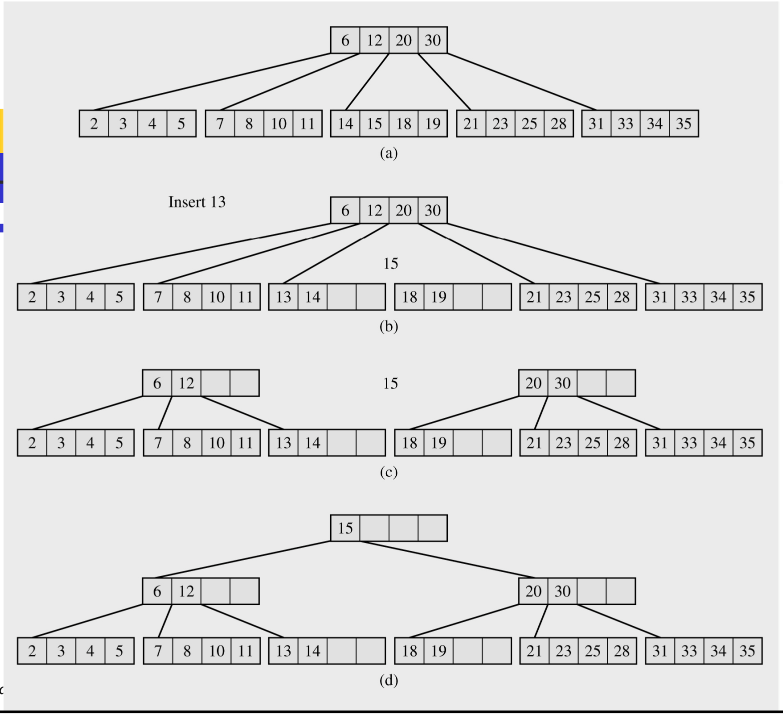
- B-Tree – Inserting a key (cont.)
  - 2nd case, the leaf where the key should be inserted is full
    - insert 6
    - the leaf is ***split***, and half the keys are moved to a new leaf



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



(a)

(b)

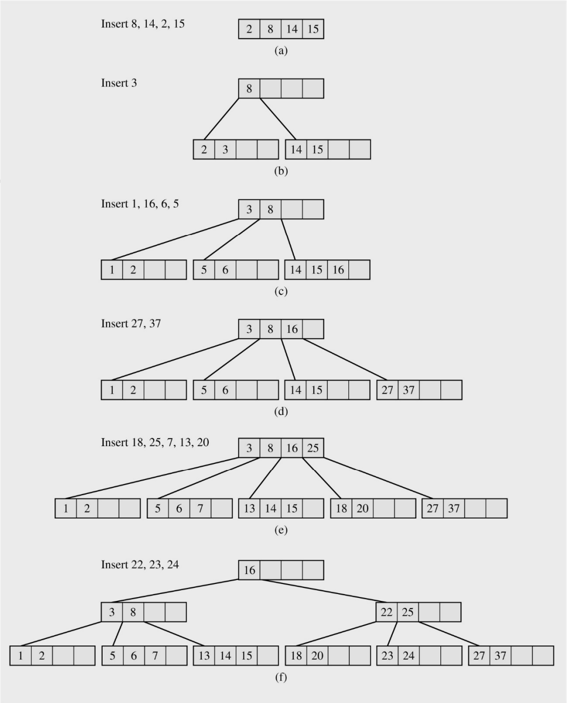
(c)

(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

- Fig. 7.8 (pp. 318)



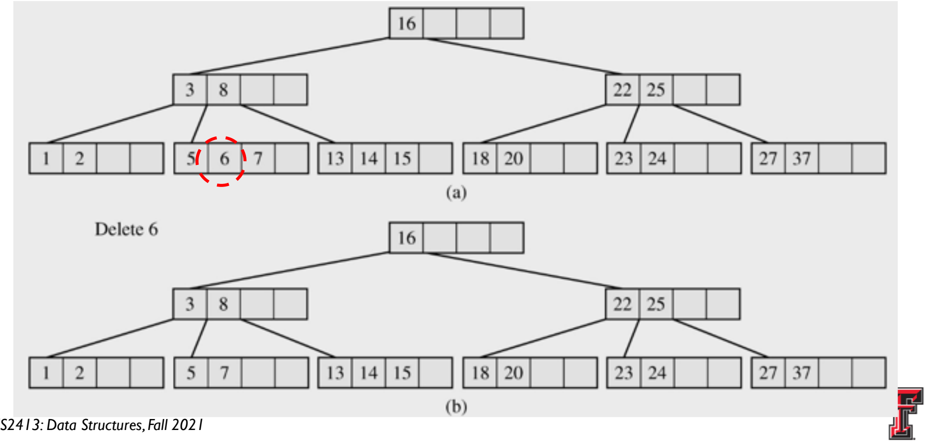
B-Trees (cont.)

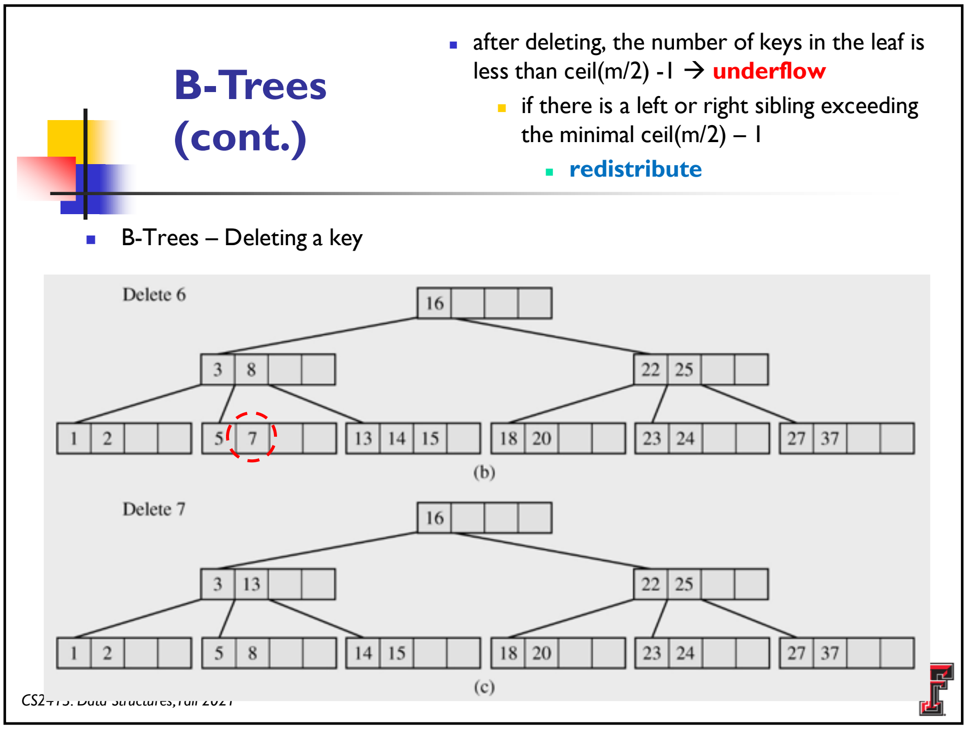
- B-Trees – Deleting a key
  - delete a key from a **leaf**
    - after deleting, if the leaf is at least half full
    - after deleting, the number of keys in the leaf is less than ceil(m/2) - 1 → underflow
      - if there is a left or right sibling exceeding the minimal ceil(m/2) – 1
        - redistribute
      - if the leaf underflows and the number of keys in its siblings is ceil(m/2) – 1
        - merge
  - delete a key from a **non-leaf**
    - the key to be deleted is replaced by its immediate predecessor (or the successor could also be used)

B-Trees (cont.)

- delete a key from a **leaf**
  - after deleting, if the leaf is at least half full

- B-Trees – Deleting a key



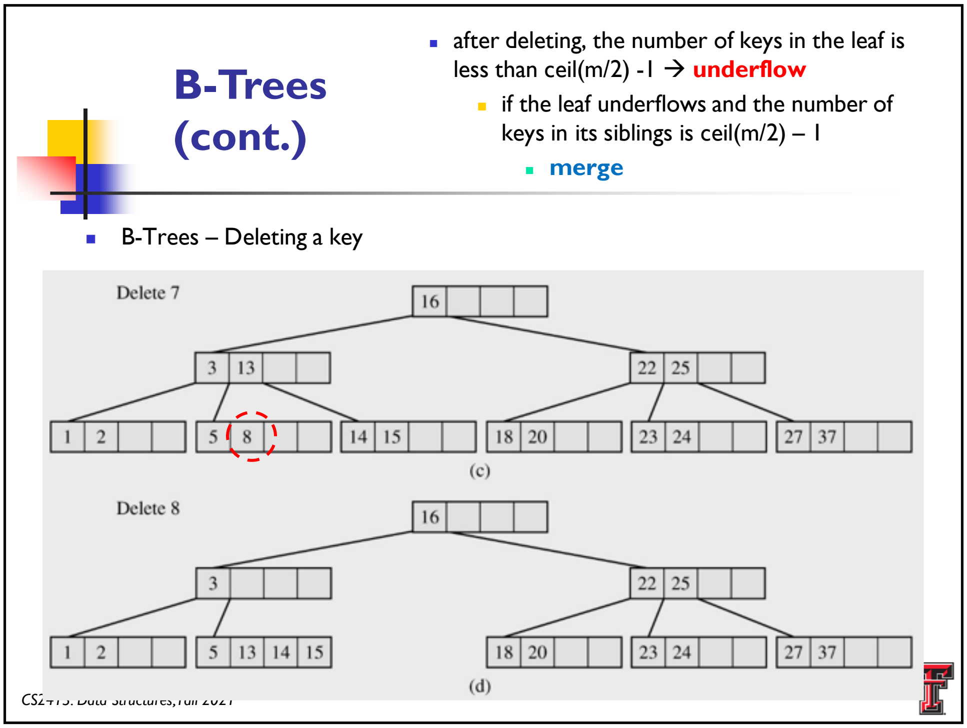


B-Trees (cont.)

- after deleting, the number of keys in the leaf is less than ceil(m/2) -1 → **underflow**
  - if there is a left or right sibling exceeding the minimal ceil(m/2) – 1
    - **redistribute**

- B-Trees – Deleting a key

CS2413: Data Structures, Fall 2021



B-Trees (cont.)

- after deleting, the number of keys in the leaf is less than ceil(m/2) -1 → **underflow**
  - if the leaf underflows and the number of keys in its siblings is ceil(m/2) – 1
    - **merge**

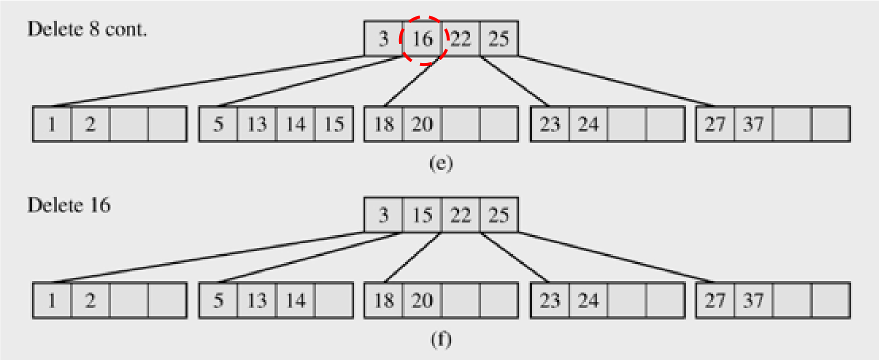
- B-Trees – Deleting a key

CS2413: Data Structures, Fall 2021

B-Trees (cont.)

- delete a key from a **non-leaf**
  - the key to be deleted is **replaced** by its immediate predecessor (or the successor could also be used)

- B-Trees – Deleting a key


Delete 8 cont.
3
16
22
25
1
2
5
13
14
15
18
20
23
24
27
37
(e)
Delete 16
3
15
22
25
1
2
5
13
14
18
20
23
24
27
37
(f)