香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

SCHOOL OF
DATA SCIENCE
數據科學學院

CSC3170
Tutorial 6

School of Data Science

The Chinese University of Hong Kong, Shenzhen

The goals of today’s session are:

- Understand the structure and properties of a B+ Tree.
- Learn how to perform insertions and deletions in a B+ Tree.
- Grasp the basics of searching operations in a B+ Tree.

B+Tree

- A **B+Tree** is a self-balancing, ordered tree data structure that allows searches, sequential access, insertions, and deletions in $\boldsymbol{O(\log_m n)}$.
  - Generalization of a binary search tree, since a node can have more than two children.
  - Optimized for systems that read and write large blocks of data.
  - **m** is the fanout of the tree.

B+Tree Properties

- A B+Tree is an ***M***-way search tree with the following properties:
  - It is perfectly balanced (i.e., every leaf node is at the same depth in the tree)
  - Every node other than the root is at least half-full
    $M/2-1 \leq \#keys \leq M-1$
  - Every inner node with k keys has k+1 non-null children

B+Tree: Insert

- Find correct leaf node L.
  Insert data entry into L in sorted order.
- If L has enough space, done!
- Otherwise, split L keys into L and a new node L2
  - Redistribute entries evenly, copy up middle key.
  - Insert index entry pointing to L2 into parent of L.

- To split inner node, redistribute entries evenly, but push up middle key.

B+Tree: Delete

- Start at root, find leaf `L` where entry belongs.
  Remove the entry.
  If `L` is at least half-full, done!
  If `L` has only `M/2-1` entries,
    - Try to re-distribute, borrowing from sibling (adjacent node with same parent as `L`).
    - If re-distribution fails, merge `L` and sibling.

- If merge occurred, must delete entry (pointing to `L` or sibling) from parent of `L`.

Practice

Consider the following B+tree.

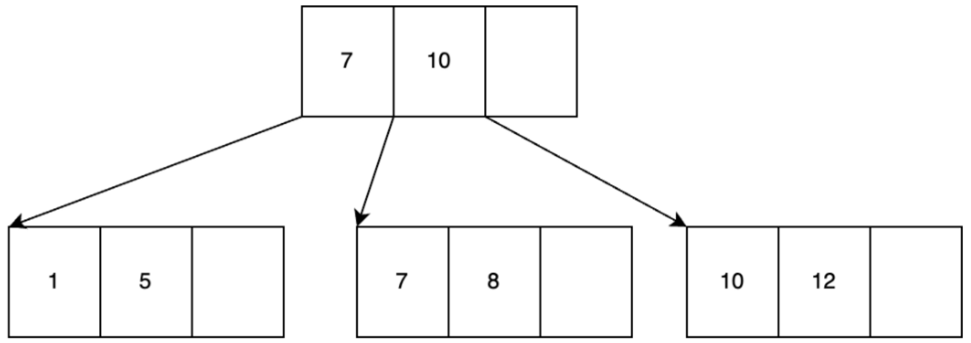


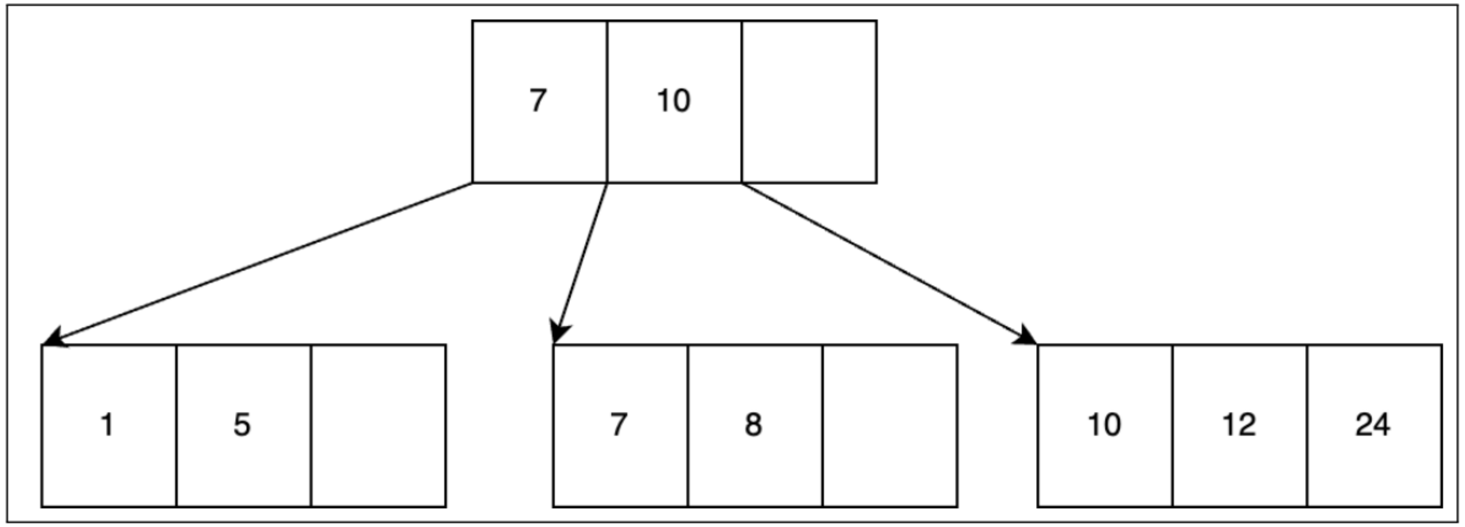
Figure 2: B+ Tree of order $d = 4$ and height $h = 2$.

When answering the following questions, be sure to follow the procedures described in class and in your textbook. You can make the following assumptions:

- A left pointer in an internal node guides towards keys $<$ than its corresponding key, while a right pointer guides towards keys $\geq$.
- A leaf node underflows when the number of **keys** goes below $\lceil \frac{d-1}{2} \rceil$.
- An internal node underflows when the number of **pointers** goes below $\lceil \frac{d}{2} \rceil$.

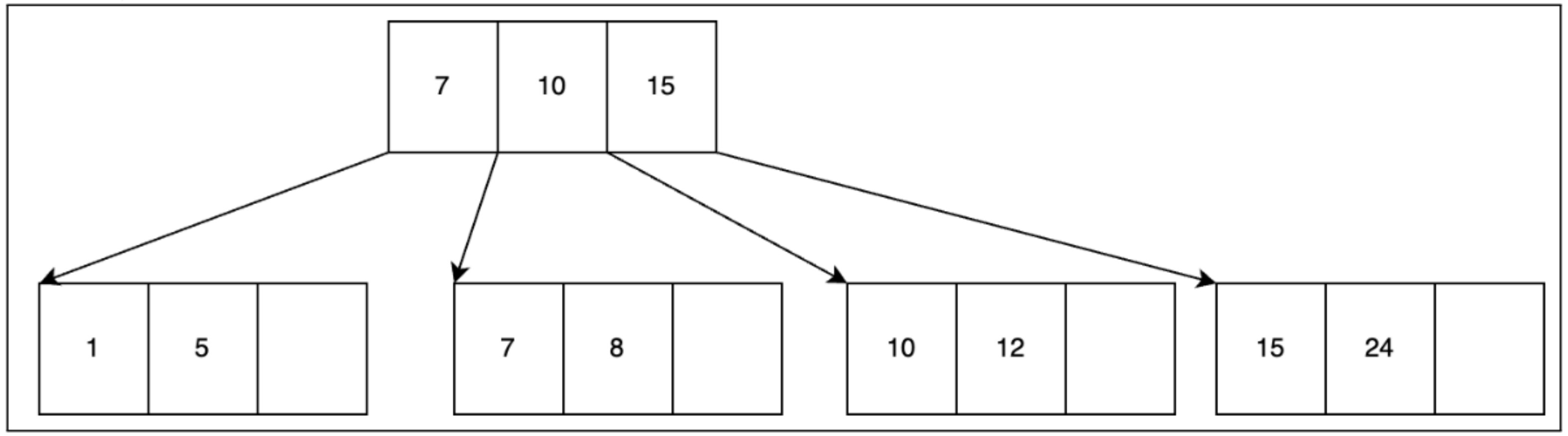
Note that B+ tree diagrams for this problem omit leaf pointers for convenience. The leaves of actual B+ trees are linked together via pointers, forming a singly linked list allowing for quick traversal through all keys.

(1) Insert 24* into the B+tree.



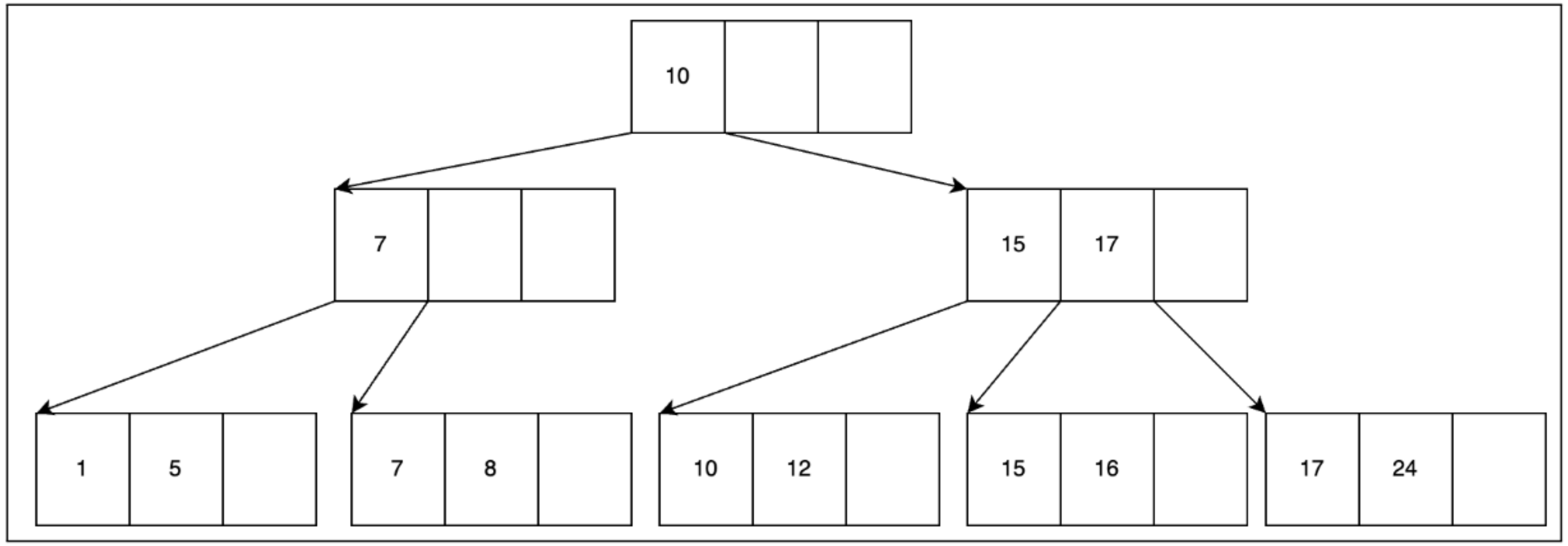
Solution: Inserting 24* adds one element in the right-most leaf, as d = 4, it should not cause any splits or merges.

(2) Starting with the tree that results from (1), insert 15^*.



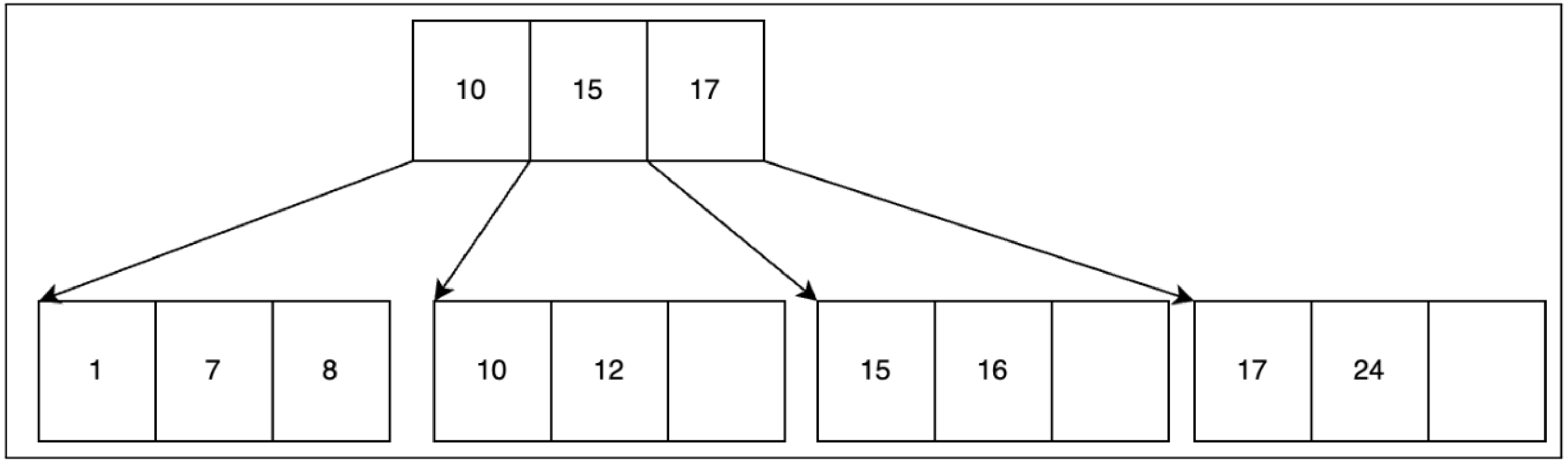
Solution: Inserting 15^* causes the right-most leaf node to split. And as the root level node still have space, it should not cause recursive splits.

(3) Starting with the tree that results from (2), insert 16^* and then insert 17^*. Select the resulting tree (the result after inserting 17^*).



Solution: Inserting 16^* fills in the remaining space of the right-most leaf node. After inserting 16^*, inserting 17^* causes the right-most leaf node to split and as the root-level node is full, it causes recursive splits.

(4) Starting with the tree that results from (3), deletes 5^*.



Solution: Deleting 5^* causes the left-most leaf node underflow and merging of the first two leaf nodes. After merging, the number of pointers in the left internal node is less than 2. It causes recursive merging then.

SCHOOL OF DATA SCIENCE

Q&A