Lecture 7

Instructor: Subrahmanyam Kalyanasundaram

29th August 2019

Plan

- Last class, we saw structure of randomly built BSTs
- We saw that the expected average depth is $O(\log n)$
- We also mentioned that expected height is $O(\log n)$ (without proof)

Plan

- Last class, we saw structure of randomly built BSTs
- We saw that the expected average depth is $O(\log n)$
- We also mentioned that expected height is $O(\log n)$ (without proof)

- Today, we see 2-3-4 trees (or (2,4)-trees), another height balanced tree
- This generalizes to (a, b)-trees and B-trees

Course grading scheme

- 60% – Exams (2 or 3)
- 30% – Programming Assignments
- 10% – Attendance and Quizzes

Course grading scheme

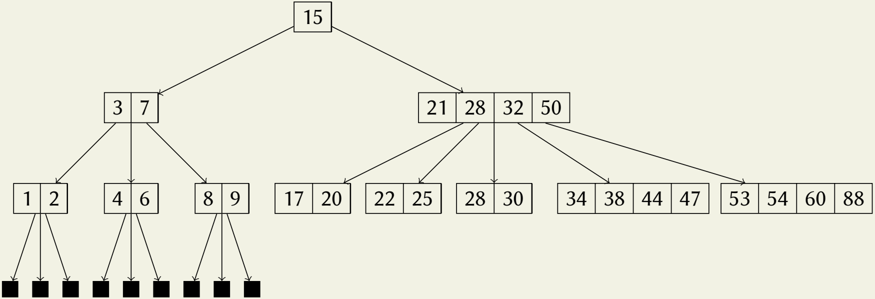
- 60% – Exams (2 or 3)
- 30% – Programming Assignments
- 10% – Attendance and Quizzes

Exam on Thursday, 5 Sep?

Multiway search Trees

- Search trees, but not binary search trees
- Each node has at least 2 children
- Each node can store many keys
- If a node stores d keys, then it has $d + 1$ children
- All leaf nodes are NIL nodes
- All leaf nodes are at the same level

Example



All the NIL nodes are not shown above

2-3-4 Trees

- Multiway search tree where each node has 1, 2 or 3 keys.
- Consequently, each node has 2, 3 or 4 children

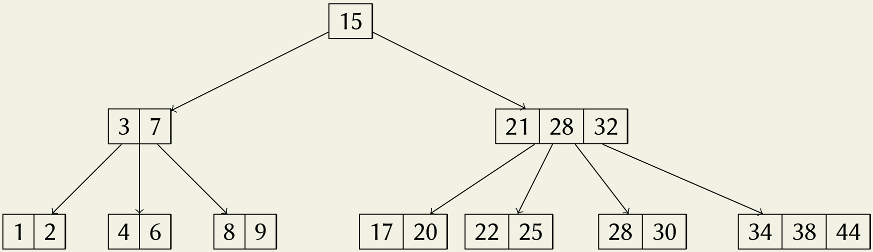
- What can we say about the height of a 2-3-4 tree?

2-3-4 Trees

- Multiway search tree where each node has 1, 2 or 3 keys.
- Consequently, each node has 2, 3 or 4 children

- What can we say about the height of a 2-3-4 tree?
- $1/2 \log(n+1) \leq h \leq \log(n+1)$

Example



No NIL nodes are shown above

Searching in 2-3-4 tree

- Similar to BST search
- Start from the root node
- Find two keys in the node k_{i-1} and k_i such that the searched value is between these two values
- Search the subtree between k_{i-1} and k_i
- Running time?

Searching in 2-3-4 tree

- Similar to BST search
- Start from the root node
- Find two keys in the node k_{i-1} and k_i such that the searched value is between these two values
- Search the subtree between k_{i-1} and k_i
- Running time?

- Takes $O(\log n)$ time

Other query operations

- How do you find successor/predecessor?

Other query operations

- How do you find successor/predecessor?
- How about Max/Min?

Other query operations

- How do you find successor/predecessor?
- How about Max/Min?
- Running time?

Insertion

- Suppose we want to insert the value x
- Search for x in the tree
- If x not found, insert x in the leaf node where it should ideally have been
- Two cases:

Insertion

- Suppose we want to insert the value x
- Search for x in the tree
- If x not found, insert x in the leaf node where it should ideally have been
- Two cases:
  - The node has room for x – it has 1 or 2 values only
  - The node is full – it has already 3 values

Case 1

The node has room for x

INSERT(x)

Case 1

The node has room for x

Resolution:

- We simply add x to the leaf node where it should have been
- Maintain the keys in sorted order

Case 1

$\boxed{17}$

15		

Case 1

15	17	

Case 1

$\boxed{16}$

15	17	

Case 1

15	16	17

Case 2

The node has no room for x

Case 2

The node has no room for x

Resolution:

- Adding x to the node results in 4 keys
- We cannot have 4 keys

INSERT(x)

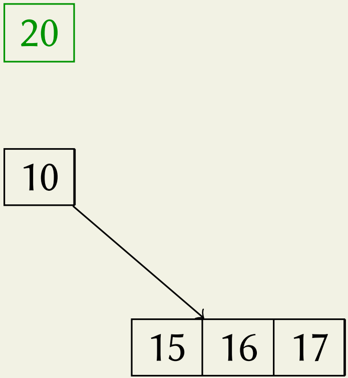
Case 2

The node has no room for x

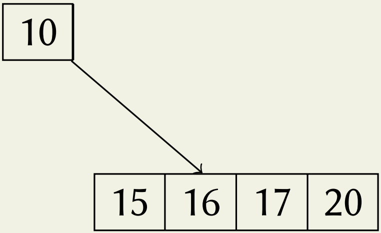
Resolution:

- Adding x to the node results in 4 keys
- We cannot have 4 keys
- We split the node and promote the median

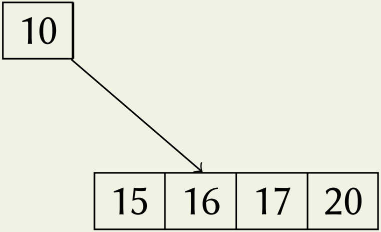
Case 2



Case 2

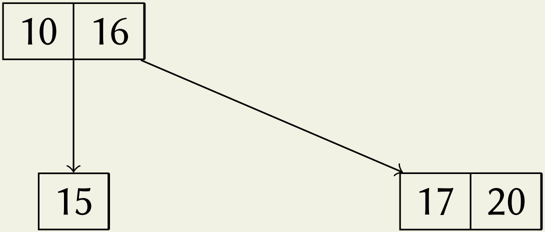


Case 2

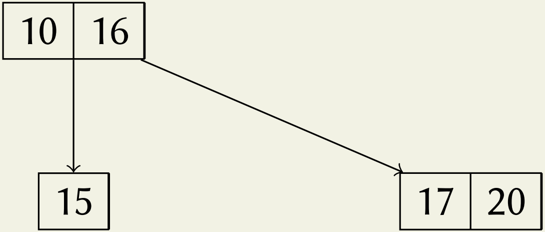


Split and promote!

Case 2

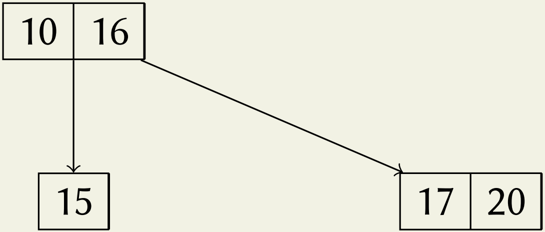


Case 2



- Can we promote any other key?
- What if the parent node doesn't have room?

Case 2



- Can we promote any other key? The other median.
- What if the parent node doesn't have room? Recurse up!

Insert Example

On the board

Deletion

- We want to insert the value x
- If x is in the leaf, we delete x from the leaf
- Else, we swap x with its successor/predecessor and delete the succ/pred

Deletion

- We want to insert the value x
- If x is in the leaf, we delete x from the leaf
- Else, we swap x with its successor/predecessor and delete the succ/pred
- Note: The succ/pred will always be in a leaf node if x is not in a leaf.

Deletion

- We want to insert the value x
- If x is in the leaf, we delete x from the leaf
- Else, we swap x with its successor/predecessor and delete the succ/pred
- Note: The succ/pred will always be in a leaf node if x is not in a leaf.

- From now on, we discuss deletion from leaf node

Deletion

Cases:

Deletion

Cases:

- The node has another key apart from x
- x is the only value in the node, but can “borrow” from sibling
- x is the only value in the node and cannot “borrow” from sibling

Case 1

The node has another key

DELETE(x)

Case 1

The node has another key

Resolution:

- We simply remove the key x

Case 1

15	16	17

- Delete 17

Case 1

15	16	

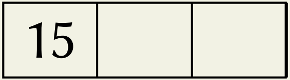
- Delete 17 Done!

Case 1

15	16	

- Delete 17 Done!
- Delete 16

Case 1



- Delete 17 Done!
- Delete 16 Done!

Case 1

15		

- Delete 17 Done!
- Delete 16 Done!
- Delete 15? Next Cases!

Case 2

The node only one key, x

Can “borrow” from sibling node

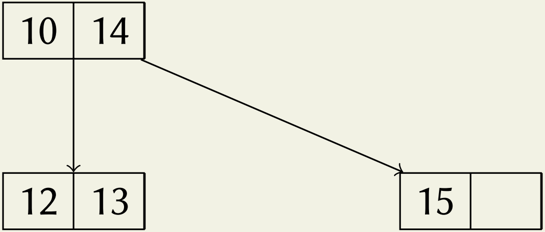
Case 2

The node only one key, x
Can "borrow" from sibling node

Resolution:

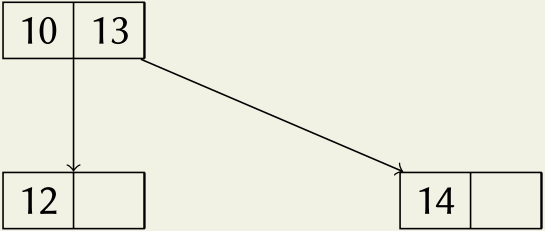
- Adjacent sibling must have ≥ 2 keys
- Can borrow from the adjacent sibling, through the parent

Case 2



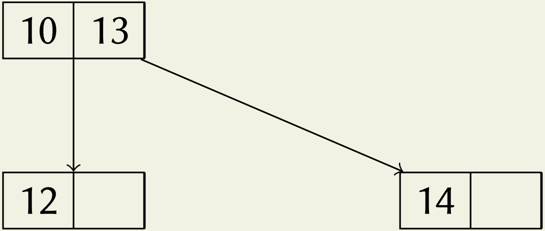
- Delete 15

Case 2



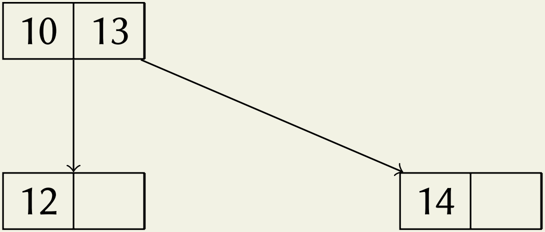
- Delete 15

Case 2



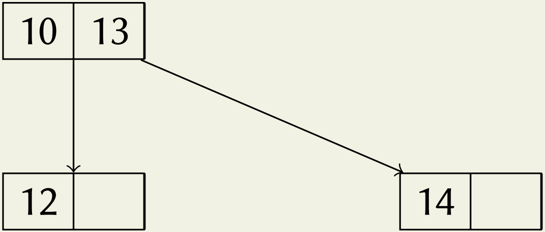
- Delete 15
- 13 is transferred to the parent node, and 14 is brought down
- Similar to

Case 2



- Delete 15
- 13 is transferred to the parent node, and 14 is brought down
- Similar to Rotation!
- Like in rotation, we transfer one child of the sibling node

Case 2



- Delete 15
- 13 is transferred to the parent node, and 14 is brought down
- Similar to Rotation!
- Like in rotation, we transfer one child of the sibling node
- What if we cannot borrow from sibling?

DELETE(x)

Case 3

The node only one key, x

Cannot borrow from sibling

Case 3

The node only one key, x
Cannot borrow from sibling

Resolution:

- Merge with a sibling

DELETE(x)

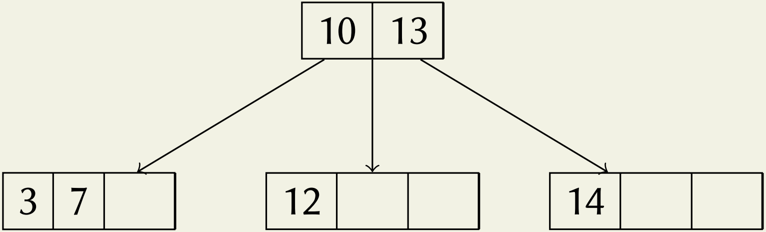
Case 3

The node only one key, x
Cannot borrow from sibling

Resolution:

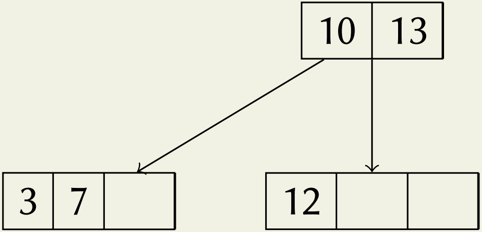
- Merge with a sibling
- Need to bring a key down from parent node

Case 3



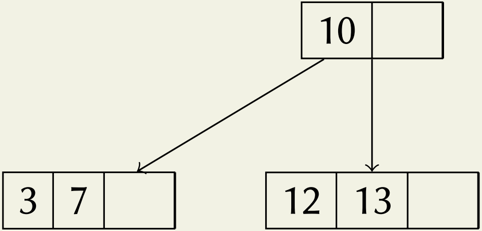
- Delete 14
- Cannot borrow from either sibling

Case 3



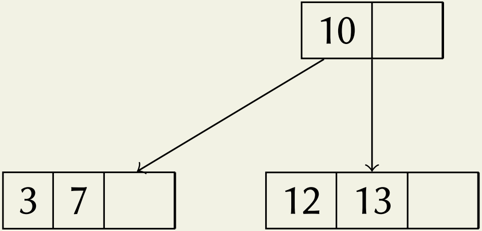
- Delete 14
- Cannot borrow from either sibling
- Once we remove the node, we have an issue

Case 3



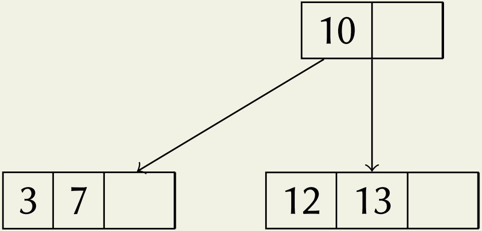
- Delete 14
- Cannot borrow from either sibling
- Once we remove the node, we have an issue
- Bring down a key from parent

Case 3



- Delete 14
- Cannot borrow from either sibling
- Once we remove the node, we have an issue
- Bring down a key from parent
- What if parent has only one key?

Case 3



- Delete 14
- Cannot borrow from either sibling
- Once we remove the node, we have an issue
- Bring down a key from parent
- What if parent has only one key? Recurse!

Delete Example

On the board

Summary of Insert and Delete

- At each node, we do an $O(1)$ time operation
  - Add/remove key
  - Split/Merge
  - Borrow from sibling
  - Promote to/bring down from parent
- We may go up the tree as well, upto height h
- Running time is $O(h) = O(\log n)$

Questions

- Think about how the insert/delete operations compare with the operations in Red-Black Trees.
- Could we extend this notion to an (a, b)-tree? What conditions should be satisfied by a and b?