CSC 212: Data Structures and Abstractions

Binary search trees (part 2)

Prof. Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

Fall 2025

THINK BIG WE DO℠

Operations: remove

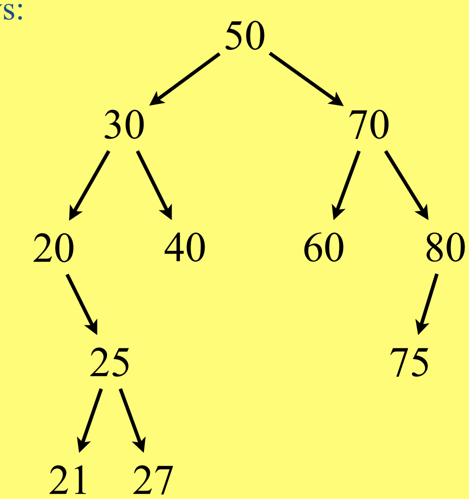
Remove

- Approach
  - ✓ locate the node to be removed, then apply one of the following cases
  - ✓ **case 1**: node is a leaf (no children)
    - delete the node and set the parent's pointer to `nullptr`
  - ✓ **case 2**: node has exactly one child
    - bypass the node by updating the parent's pointer to point to the node's only child
    - delete the node
  - ✓ **case 3**: node has two children
    - find the inorder successor (the smallest node in the right subtree)
      - **alternative**: inorder predecessor (the largest node in the left subtree) may be used instead
    - copy the successor's data to the current node
    - recursively delete the successor node (in the right subtree) or predecessor node (in the left subtree)
- Time complexity
  - ✓ $O(h)$, where h is the height of the tree

`if root is the node to delete, update root pointer after deletion`

Practice

- Remove the following keys:
  - ✓ 27, 40, 80, 20, 30, 50



Analysis

Practice

- Starting from an empty BST, insert the following keys in the order given
  - ✓ 20 10 30 5 15 25 35
  - ✓ 10 20 5 15 30 35 25
  - ✓ 5 10 15 20 25 30 35
  - ✓ How is the order of insertion related to the shape of the tree?
  - ✓ How is the number of nodes related to the height of the tree?

Practice

- Complete the following table with rates of growth
  - ✓ as a function of the number of nodes

Operation	Best case	Average case	Worst case
Insert			
Remove			
Search			

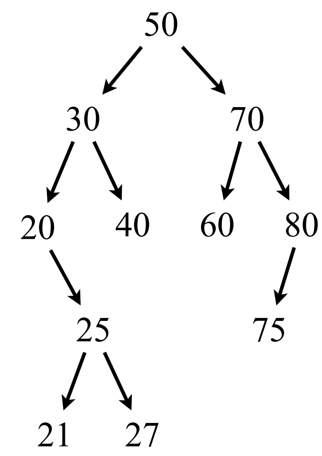
Average case

- Proposition
  - ✓ if n distinct keys are inserted in random order into an initially empty BST, the expected number of comparisons for a search is $\sim c \log n$
    - this results can be formally justified through probabilistic analysis (beyond the scope of this class)
- Implications
  - ✓ even without explicit balancing mechanisms, randomly built BSTs provide logarithmic expected time for search, insert, and delete operations

Traversals

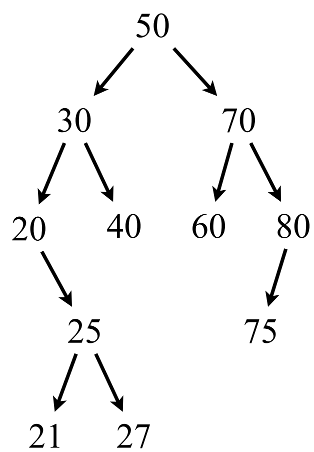
Preorder traversal

- Depth-first traversal that visits the root node first, then recursively traverses the left subtree, followed by the right subtree
  - ✓ visit (process) the root node
  - ✓ recursively traverse the left subtree
  - ✓ recursively traverse the right subtree



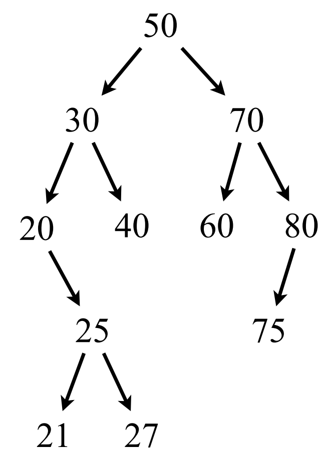
Postorder traversal

- Depth-first traversal that recursively traverses the left subtree, then the right subtree, and finally visits the root node
  - ✓ recursively traverse the left subtree
  - ✓ recursively traverse the right subtree
  - ✓ visit (process) the root node



Inorder traversal

- Depth-first traversal that recursively traverses the left subtree, then visits the root node, and finally traverses the right subtree
  - ✓ recursively traverse the left subtree
  - ✓ visit (process) the root node
  - ✓ recursively traverse the right subtree



Practice

- Which traversal is best for printing all values in sorted order?
- Which traversal is best for deleting all nodes in a tree?
- What is the time complexity of each traversal?

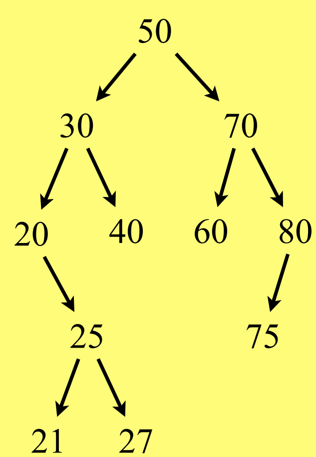
Practice

- Write an algorithm that performs preorder traversal on a ternary tree
  - ✓ assume pointers to children are `left, mid, right`
- Write an algorithm that performs postorder traversal on a k-ary tree with any value of k
  - ✓ assume pointers to children are in an array `children`

Practice

- Trace the following algorithm and explain what it does

```
algorithm mistery(root) {
    queue q
    q.enqueue(root)
    while not q.isEmpty() {
        node n = q.dequeue()
        print(n.value)
        if n.left
          q.enqueue(n.left)
        if n.right
          q.enqueue(n.right)
    }
}
```



Collections

Operation	Description	Sequential (unordered)	Sequential (ordered)	BST
search	search for a key	O(n)	O(log n)	O(h)
insert	insert a key	O(n)	O(n)	O(h)
delete	delete a key	O(n)	O(n)	O(h)
min/max	find smallest/largest key	O(n)	O(1)	O(h)
floor/ceiling	find predecessor/successor	O(n)	O(log n)	O(h)
rank	count number of keys less than key	O(n)	O(log n)	O(h)