CS61BL Tutoring Session

Worksheet 7: B-Trees

Hongli (Bob) Zhao

Agenda

- Proj2 due at Saturday midnight!
  - Come to Project Party on Saturday 1-3 pm in Discord
- Quick recap of BST (Binary Search Tree)
  - Question 1: BST runtime
  - Motivation for Balanced Trees
- 2-4 Trees <--> LLRB Trees
  - Equivalent operations
  - Common runtime
- Question 4: conceptual questions
- Question 5: converting 2-4 Trees to LLRB Trees
- Any questions from Quiz 7?
  - Come back between 9-9:30 pm and ask anything

Binary Search Trees

BST Properties

- It is a tree
  - Connected, acyclic, undirected graph
- Nodes on the left have smaller values than root, nodes on the right have larger values than root
  - "smaller/larger" depends on how you define it (ref: `.compareTo()`)
- Common operations:
  - `contains(T key)`, finds the element and returns True
  - `void add(T key)`, inserts element into tree, preserving BST property
  - `T delete(T key)`, deletes element from tree, and returns it, preserving BST property
  - Demo: Credit Fall 2019 Professor Hilfinger's Slides

Question 1: Why do we want Balanced-ness?

1 Runtime Questions

Provide the best case and worst case runtimes in theta notation in terms of N, and a brief justification for the following operations on a binary search tree. Assume N to be the number of nodes in the tree. Additionally, each node correctly maintains the size of the subtree rooted at it. [Taken from Final Summer 2016]

```
boolean contains(T o); // Returns true if the object is in the tree
```

Best: $\Theta($ $)$ Justification:

Worst: $\Theta($ $)$ Justification:

```
void insert(T o); // Inserts the given object.
```

Best: $\Theta($ $)$ Justification:

Worst: $\Theta($ $)$ Justification:

```
T getElement(int i); // Returns the ith smallest object in the tree.
```

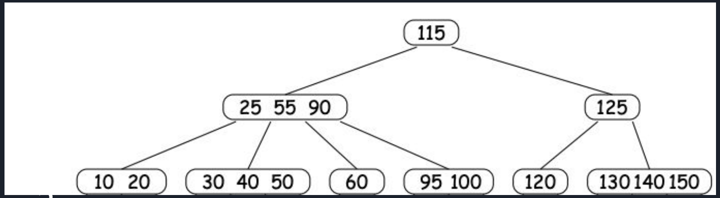
Best: $\Theta($ $)$ Justification:

Worst: $\Theta($ $)$ Justification:

:

If the tree is balanced, we get a log-speed up on average
- But the tree structure is problem dependent

Balanced Search Structures



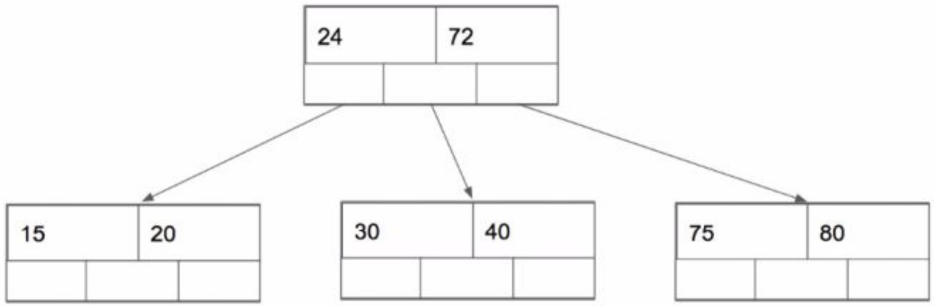
- How do we always achieve a log factor speed-up?
  - Divide the nodes by some constant factor > 1
  - In other words, need to have “bushy” trees
  - Come up with a way such that the height from any leave to the root is constant, or differ by some constant factor
- 2-4 Tree (2-3-4 Tree) Properties:
  - Each node has at least 2 children, at most 4 children
  - Any non-leaf node must have 1 more child than keys
  - Elements in nodes are sorted
- Operations:
  - `find, insert, delete`
  - Guaranteed to have O(logN) runtime
  - Demo

Question 4: Conceptual stuff

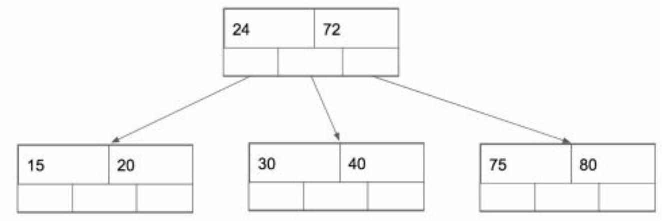
1. Why does a binary search tree have a worst case runtime of $\theta(n)$ for *contains*?

2. Give a sequence of operations, such that if they were inserted in the order they appear, would result in a "poor" binary search tree.

3. Examine this B-tree with order 3. Mark the paths taken when the user calls *contains*(40).



Question 4: Conceptual stuff



4. Now call *insert*(35), and draw the resulting tree.

5. What property of a B-tree rectifies problems of binary search trees, such as the one in 1.1? Why would you not use a B-tree?

LLRB: Left Leaning Red-Black Tree

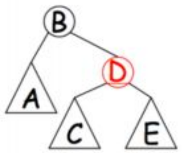
- Make more sense to implement Balanced trees if we need to efficiently store a large amount of data. For our purpose, can try and implementation is tricky (not generalizable).
- 2-4 trees have a one-to-one correspondence with LLRB Trees.
- Properties:
  - Binary Search Tree, with more constraints
  - Root is black
  - Every non leaf node has 2 children
  - Every red root has 2 black children
  - (LLRB) break ties by prioritizing edges on the left
- Operations:
  - `rotateLeft, rotateRight, flipColors`
  - Know how to insert nodes and perform fix ups, and convert LLRB to B-Trees and vice versa
    - Resource: https://inst.eecs.berkeley.edu/~cs61b/fa19/materials/lectures/lect29/

5 The Holy LLRB Invariant

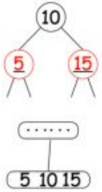
RB Tree Invariants: Node labels are in order from left to right. All paths through the tree contain the same number of black nodes. No red nodes have red parents. As a result, the height of a RB tree with n nodes is $O(logn)$.

LLRB trees must also maintain the following invariant (in addition to the regular red-black invariant):

No right-leaning trees (black parent with right red child):

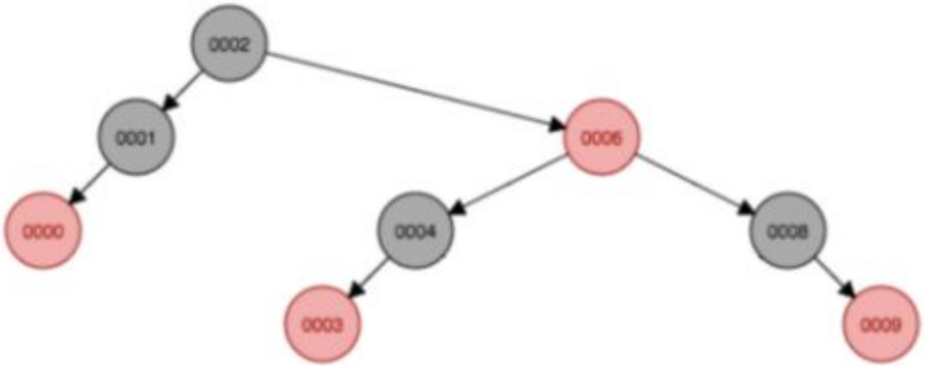


No “4-nodes” (black parent with two red children):



1. What are the ”fixups” for the two cases above in order to preserve the LLRB invariant (i.e. what operations do we perform on each tree to ensure it is a proper LLRB)?

Consider the following RB tree:



2. Draw the tree after applying all necessary fixups to make it a proper LLRB tree.

3. Next, insert 10 into the tree, and apply all fixups to preserve the LLRB invariant.

4. Finally, draw the corresponding 2-3 tree.