Trees

Dov Kruger

Department of Electrical and Computer Engineering
Stevens Institute of Technology

September 27, 2022

©2022 Stevens Institute of Technology

Definition

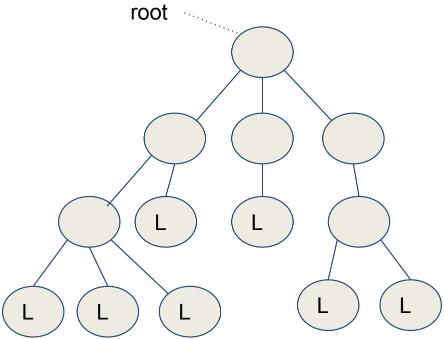
A tree is composed of nodes

Each node can have 0 or more children

A special node called root is the top of the tree

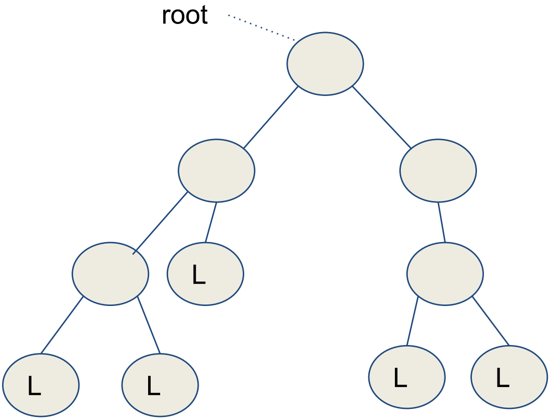
- Computer science trees grow upside-down!

Nodes with 0 children are called leaf nodes



©2022 Stevens Institute of Technology

In a binary tree all nodes have no more than 2 children



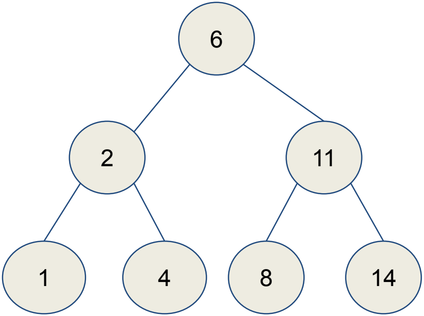
©2022 Stevens Institute of Technology

Ordered Binary Trees/Binary Search Tree

An Ordered Binary Tree is a binary tree where

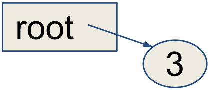
- Each node contains a key
- For each node
  $\text{key}_{\text{left child}} < \text{key}_{\text{this node}}$
  $\text{key}_{\text{right child}} \geq \text{key}_{\text{this node}}$
  or
  $\text{key}_{\text{left child}} \leq \text{key}_{\text{this node}}$
  $\text{key}_{\text{right child}} > \text{key}_{\text{this node}}$
- Furthermore this rule applies to the whole subtree

- Example: anything to the right of the root ≥ 6



STEVENS INSTITUTE of TECHNOLOGY THE INNOVATION UNIVERSITY
©2022 Stevens Institute of Technology

```
class Node {
  public int val;
  public Node left, right;
}
```



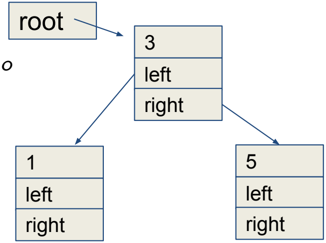
Tree must define a Node

```
OrderedBinaryTree.insert(v)
  if root == null
    root = new Node(v)
    return // special case for empty tree
  end
```


©2022 Stevens Institute of Technology

Insert into OrderedBinaryTree, contd

```
p <- root
   while p != null
      if p.val < v // can be <= also
         if p.right == null
            p.right <- new Node(v)
            return
         end
      else
         if p.left == null
            p.left <- new Node(v)
            return
         end
    end
   p <- p.next
```



©2022 Stevens Institute of Technology

There are three classic traversal algorithms for binary trees

- Inorder
- Preorder
- Postorder

©2022 Stevens Institute of Technology

Inorder

```
inorder(node)
  if node == null
    return // termination condition for recursion!
  end
  inorder(node.left)
  do whatever you want to do to this node (like print it)
  inorder(node.right)
end
```


©2022 Stevens Institute of Technology

Preorder

```
preorder(node)
  if node == null
    return // termination condition for recursion!
  end
  do whatever you want to do to this node (like print it)
  preorder(node.left)
  preorder(node.right)
end
```


©2022 Stevens Institute of Technology

Postorder

```
postorder(node)
  if node == null
    return // termination condition for recursion!
  end
  postorder(node.left)
  postorder(node.right)
  do whatever you want to do to this node (like print it)
end
```


©2022 Stevens Institute of Technology

Another way of Terminating Recursive Traversals

The following code requires testing first that root is not null, but it is more efficient

Complexity is the same

```
postorder(node)
  if node.left != null
    postorder(node.left)
  end
  if node.right != null
    postorder(node.right)
  end
  do whatever you want to do to this node (like print it)
end
```


©2022 Stevens Institute of Technology

Expressions and Relationship to Trees

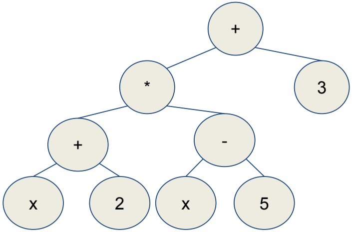
In mathematics we are taught to write expressions inline

$(x+2)(x-5)+3$

The operations the above expression represents have an order

Conventions are defined mathematically

The end result is a tree of operations like this:



©2022 Stevens Institute of Technology

Three Ways of Writing Expressions

Inorder: $(x+2)(x-5)$

Preorder: $* + x2 - x5$

Postorder: $x2 + x5 - *$

All 3 reflect the same tree of operations, traversed using inorder, preorder, postorder

©2022 Stevens Institute of Technology

Test Yourself

©2022 Stevens Institute of Technology

Sorting Using an Ordered Binary Tree

An ordered binary tree is intrinsically sorted

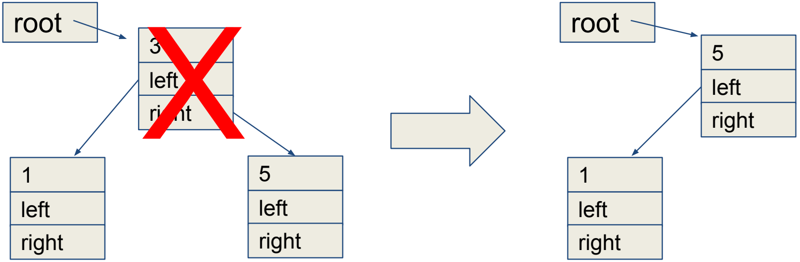
Just add elements to the tree

Print out the elements using inorder traversal

©2022 Stevens Institute of Technology

Deletion from an Ordered Binary Tree

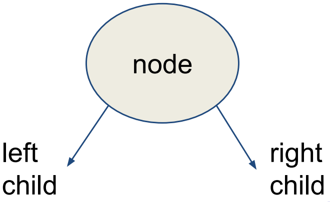
To delete, remove a node, then promote one of the children in its place



©2022 Stevens Institute of Technology

Implementation of an Ordered Binary Tree

```
class OrderedBinaryTree {
private:
  class Node {
  public:
    Node* left;
    Node* right;
    int val;
  };
  Node* root; // an empty tree has root=null
  ...
}
```

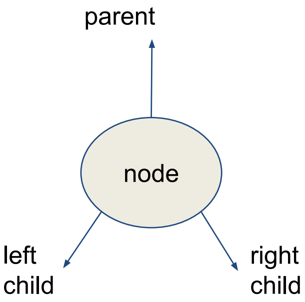


©2022 Stevens Institute of Technology

Adding a Parent Pointer

For some applications, if the algorithm is looking at a node it is useful to also be able to find the parent

```
class Node {
public:
  Node* left;
  Node* right;
  Node* parent;
  int val;
};
```



STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

©2022 Stevens Institute of Technology

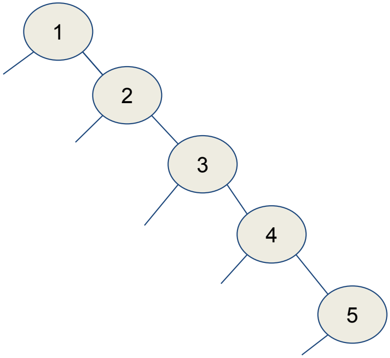
Ordered Binary Trees/Binary Search Trees can get lopsided

- Performance is as bad as lists
- Added overhead of the unused left pointers

Example: Add 1,2,3,4,5 to ordered binary tree

What is the complexity of insertion into a BST?

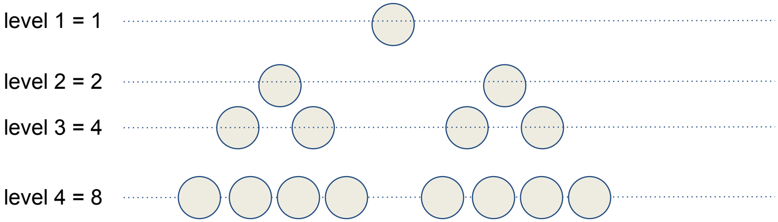
O(?)



©2022 Stevens Institute of Technology

Balanced Case

When a Tree is balanced, n elements fit into $log_2 n$ levels

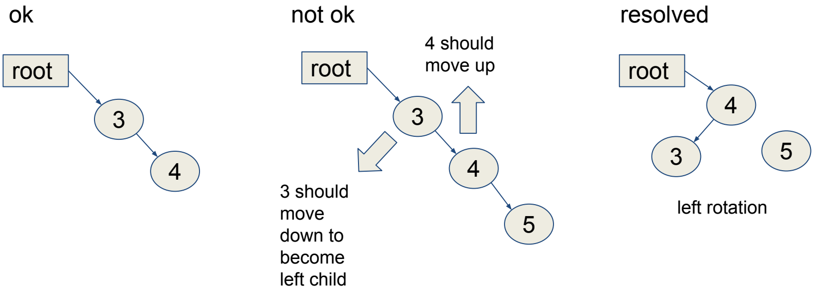


$$n = 15 \text{ elements, levels} = log_2(n+1) = 4$$

©2022 Stevens Institute of Technology

Trees Cannot be Perfectly Balanced

A Tree with one element on a new level is by definition, unbalanced



©2022 Stevens Institute of Technology

Balanced Binary Trees

Ensure trees remain balanced under all possible operation

Insertion and deletion will have to trigger "rebalancing" while not slowing down

©2022 Stevens Institute of Technology

There are a number of implementations

- Red-Black Trees (RB-Trees)
- AVL Trees
- Fibonacci Trees

We will just do one: RB-Trees

©2022 Stevens Institute of Technology

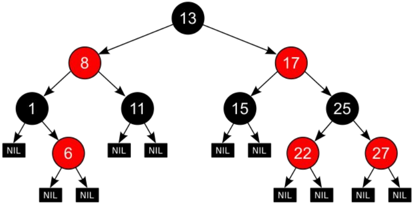
RB Trees

Objective: Maintain a balanced tree no matter the order of insertion

No worse than $2\log_2 n$

Rules

- Each node is assigned a color (red or black, really just binary)
- Red nodes cannot have red children
  - Colors alternate
- Optional: root is black



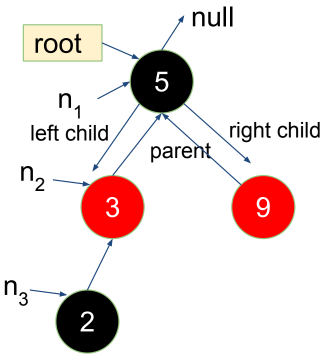
©2022 Stevens Institute of Technology

n_1.parent is node that has n as a child. This is null for root

n.grandparent() is n's parent's parent, null if it does not exist

n.uncle is a node's grandparent's other child (not parent)

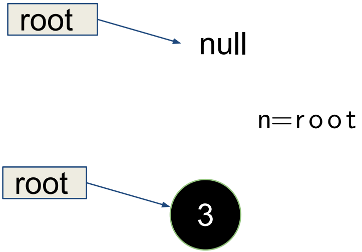
- If a node has no grandparent, it has no uncle
- n_2 has no uncle
- n_3 uncle is 9



©2022 Stevens Institute of Technology

Insertion into RB Tree

```
Start with an empty tree
RBTree.insert(3)
  calls BST.insert(3)
  then RBTree.correct(n)
```

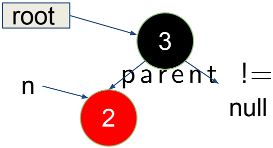



```
case 1:
if n.parent == null
  return // special case, root
```


©2022 Stevens Institute of Technology

Insertion into RB Tree, part 2

```
RBTree.insert(1)
   calls BST.insert(1)
   then RBTree.correct(n)
```



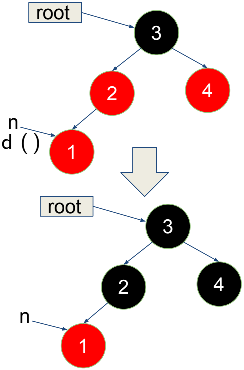

```
parent != null
```

```
case 2:
if parent.color == BLACK
    return
end
```


©2022 Stevens Institute of Technology

Insertion into RB Tree, part 3

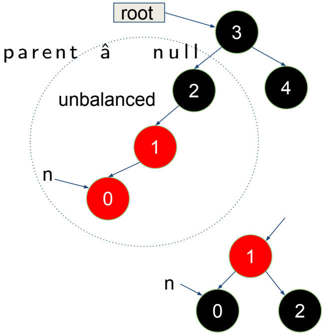
```
RBTree.insert(1)
  calls BST.insert(1)
  then RBTree.correct(n) parent != null
parent is RED
case 3:
  if n.parent.isred() and n.uncle().isred()
    flip color of parents
    flip color of grandparent
    RBTree.correct(grandparent)
recursive
  end
```



©2022 Stevens Institute of Technology

Insertion into RB Tree, part 3

```
RBTree.insert(0)
   calls BST.insert(0)
   then RBTree.correct(n)
parent is RED
[TBD]
```

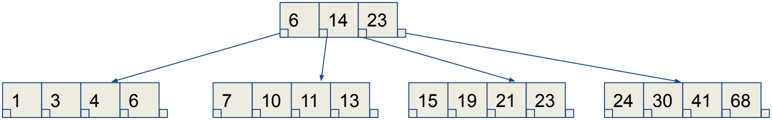


©2022 Stevens Institute of Technology

BTrees are a more complicated generalization of RBTree with a higher degree than 2

Used in database systems to try to keep the depth of the tree to a minimum

Shown: BTree of degree 4



©2022 Stevens Institute of Technology

Properties of Disk Access

Hard Disk Drives, and the more modern Solid State Drives (SSD)

- Are Block-oriented devices
- Have high overhead to access a block (latency, and OS call)
- Therefore, try to keep reads to a minimum
- Each block is large, so degree of the BTree can be kept very high (64)

With $n = 64 \ (2^6)$ we can store

Levels	Power of 2	Elements stored
1	2^6	64
2	2^{12}	4096
3	2^{18}	256k
4	2^{24}	16M
5	2^{30}	1 billion

©2022 Stevens Institute of Technology

How BTrees are used in Databases

SQL was developed in the 1970s

Creates tables of data, typically with lookup via BTree

The following table defines a primary key

- The records are stored in a BTree using the key
- The records therefore are in sorted order
- Searching for a specific city by name is O(log n)

```
CREATE TABLE Cities (
name varchar(10) primary key,
population int
)
```


©2022 Stevens Institute of Technology

BTrees in SQL

```
SELECT * FROM cities                                   results sorted by name
SELECT * FROM cities ORDER BY population    requires sorting
SELECT * FROM cities WHERE name='New York' O(log n)
INSERT INTO cities values ('Tokyo', 15000000)    2x slower
```


STEVENS INSTITUTE of TECHNOLOGY THE INNOVATION UNIVERSITY®
©2022 Stevens Institute of Technology

Adding an Index

```
CREATE INDEX cities_bypopulation              Second sorting order
ON cities (population);

SELECT * FROM cities ORDER BY population      no sorting

SELECT * FROM cities WHERE population=100    O(log n)
```


©2022 Stevens Institute of Technology

Test Yourself

©2022 Stevens Institute of Technology

Tries

A Trie is a complete tree with a high degree

- 26 for lowercase English letters
- Has not been used much because storage requirements are high
- Hash maps have dominated tries for most applications

Today, there are applications

- Autocompletion of words
- DNA searches

https://brilliant.org/wiki/tries/

©2022 Stevens Institute of Technology

Code Structure of a Trie

A Trie node has

- boolean stating whether the path is a word or not
- Pointers to the next node

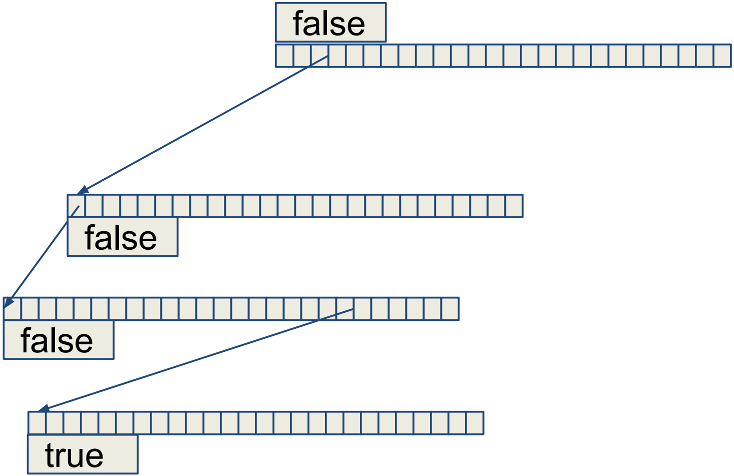
A trie dictionary can contain the root node

- Does not have to be a pointer
- Can never be null

```
class Trie {
  struct Node {
    bool isWord;
    Node* next[26];
  }
  Node root;
public:
  void add(const string& w);
  void remove(const string& w);
  bool contains(const string& w);
  bool startsWith(const string& w)
};
```


©2022 Stevens Institute of Technology

Inserting into a Trie



©2022 Stevens Institute of Technology

Searching for a Word in a Trie

©2022 Stevens Institute of Technology

Searching for a Prefix in a Trie

©2022 Stevens Institute of Technology